Carl von Ossietzky Universität Oldenburg
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

KI-unterstützte Fabriklayoutoptimierung

von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften der Carl von Ossietzky Universität Oldenburg zur Erlangung des Grades und Titels eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

angenommene Dissertation von Herrn

Patrick Eschemann

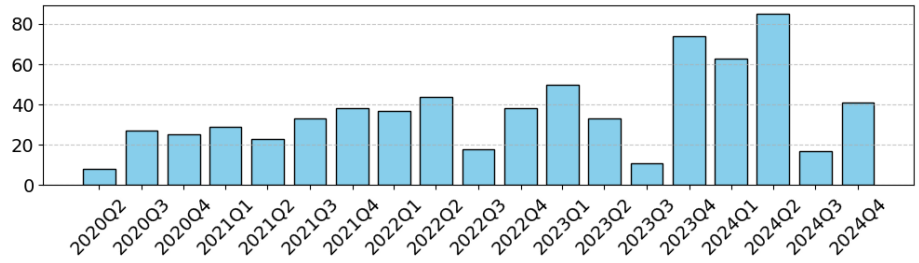
geboren am
22. Januar 1990 in Wilhelmshaven

Gutachter
Prof. Dr.-Ing. habil. Jürgen Sauer
Prof. Dr.-Ing. Astrid Nieße

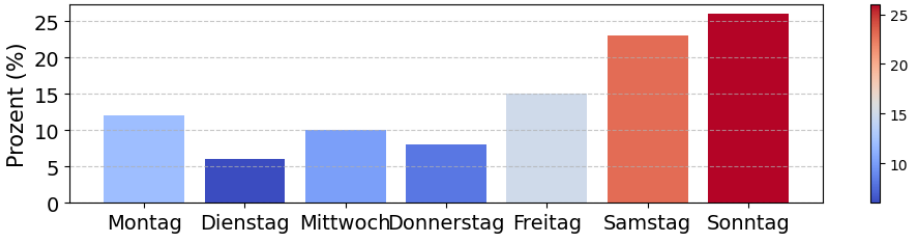
Tag der Disputation
22. November 2024

Vorwort und Danksagung

Die Inspiration zur vorliegenden Dissertation entstand im Frühjahr 2020 während meiner Arbeit am OFFIS – Institut für Informatik in Oldenburg. Im Forschungsprojekt Cyberfactory#1 entwickelten wir Optimierungsmethoden für autonome Transportroboter. Dabei erkannte ich, dass nicht nur die Roboter, sondern vor allem das umgebende Fabriklayout eine entscheidende Rolle für die Effizienz einnimmt. Da die Layoutoptimierung nicht Teil des Projekts war, entstand in mir der Wunsch, den optimierten Robotern eine ebenso optimierte Umgebung bereitzustellen. Diese Überlegung wurde zum zentralen Thema meiner Dissertation, die mich in den folgenden viereinhalb Jahren intensiv beschäftigte. Mich erreichte oft die Frage, wie viel Aufwand mit der Arbeit verbunden war. Kürzlich bin ich auf den Gedanken gestoßen, hierzu die Historie meiner *Git-Commits* zu teilen, mit der man den Verlauf der letzten Jahre schön visualisieren kann. Ein *Commit* entspricht einem jeweiligen gesicherten Zwischenstand der Arbeit, zu dem ein Fortschritt erzielt wurde.



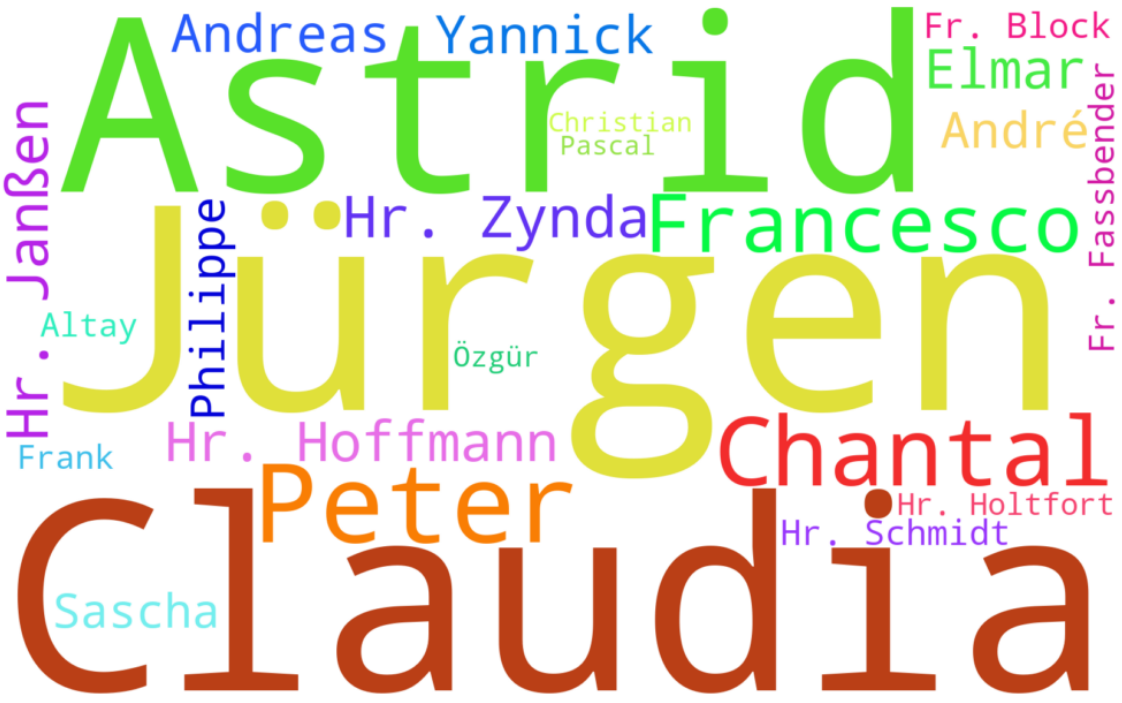
Über den Verlauf der vergangenen vier Jahre sind insgesamt 694 Zwischenstände bis zu diesem fertigen Endprodukt zusammengekommen. Die Verteilung über die Wochentage sieht folgendermaßen aus:



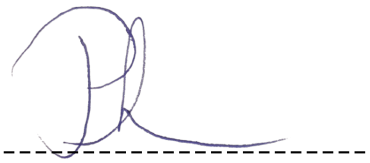
Der Schwerpunkt auf die Wochenenden ist auf meine berufliche Veränderung im Frühjahr 2022 zurückzuführen. Ich wechselte von meiner Anstellung als wissenschaftlicher Mitarbeiter zu einem Industrieunternehmen. All dies wäre nicht ohne große Unterstützung möglich gewesen. Daher spreche ich zunächst meinen Dank der Open-Source-Community aus. Die vielfältigen, frei verfügbaren Programmbibliotheken und Frameworks ermöglichten die Implementierung der Konzepte dieser Arbeit. Neben den technischen Möglichkeiten gab mir der Rat und das offene Ohr meiner Freund*innen und Kolleg*innen oft entscheidende Impulse, ohne die ich nicht bis an diesem Punkt gelangt wäre. Dafür mein besonderer Dank.

Für die freundliche Unterstützung und die Bereitschaft, mir in herausfordernden Zeiten Raum und Flexibilität zu geben, danke ich herzlich meinen Betreuern Prof. Dr.-Ing. habil. Jürgen Sauer und Prof. Dr.-Ing. Astrid Nieße der Universität Oldenburg. Danke, dass ihr stets für einen konstruktiven Austausch offen wart und dabei auch tatkräftige Unterstützung geleistet habt.

Mein herzlichster Dank gilt meiner liebevollen und geduldigen Familie. Ihre fortwährende Ermutigung und ihr Verständnis haben meine Motivation hochgehalten und zur Fertigstellung dieser Arbeit beigetragen.



Ich danke Euch!



Patrick Eschemann
17. Januar 2025

Zusammenfassung

Seit den 1960er-Jahren wird an der Optimierung von Fabriklayouts geforscht. Gängige Praxis in der Produktion ist die iterative Neuanordnung von Fabrikeinheiten, da Algorithmen als ungeeignet zur Ermittlung realistischer Lösungen gelten. Obwohl die Forschung durch erhöhte Rechenkapazitäten und verbesserte Algorithmen signifikante Fortschritte erzielt hat, bleibt diese kostenintensive Vorgehensweise weiterhin verbreitet. Dies ist unter anderem darin begründet, dass aktuelle Technologien aufwendig auf den jeweiligen Anwendungsfall angepasst werden müssen.

Diese Dissertation widmet sich der Untersuchung des Einsatzes von künstlicher Intelligenz zur Optimierung von Fabriklayouts, mit dem Ziel, die Anwendbarkeit bestehender Methoden zu erweitern und den Rechenaufwand zu reduzieren. Beide Aspekte sind entscheidend für die Komplexität der NP-vollständigen Suche nach optimalen Layouts. Ein wesentlicher Ansatzpunkt ist dabei die Bewertung von Layouts, die sowohl einen maßgebenden als auch limitierenden Faktor in Optimierungsalgorithmen darstellt. Maßgebend durch die Definition der Randbedingungen und limitierend aufgrund des Bedarfs an Rechenkapazitäten. Dennoch ist die Simulation der etablierte Standard für die computerunterstützte Bewertung von Layouts. Diese Arbeit trägt zur Forschung durch die Entwicklung und Untersuchung einer KI-Architektur bei, die Layouts mit einer vergleichbaren Güte wie eine Simulation bewerten kann. Dies gelingt durch die Kombination eines Convolutional Neural Networks zur Verarbeitung beliebiger Layouts in Bildform mit einem Multilayer-Perceptron zur Wissensextraktion aus tabellarischen Informationen. Eine Evaluation wurde anhand von fünf unterschiedlichen Layoutkonfigurationen durchgeführt, mit dem Ergebnis, dass die KI den simulierten Durchsatz bei unbekannten Layouts präzise bestimmen kann. Hierdurch können optimierte Layouts erstellt werden, die Fabrikplanern eine qualifizierte Grundlage und Arbeitserleichterung bieten.

Die gewonnenen Erkenntnisse bieten ferner Potenzial für die Übertragung auf weitere Industriebereiche, in denen die optimierte Ressourcen-Anordnung von hoher Bedeutung ist, wie beispielsweise in Verladehäfen, Flughäfen oder Krankenhäusern. Eine optimierte Anordnung kann überdies durch Reduzierung von Materialbewegungen einen signifikanten Beitrag zur Nachhaltigkeit leisten. Künstliche Intelligenz zeigt somit einen positiven Einfluss im Einsatz zur Layoutoptimierung und darüber hinaus auf.

Abstract

Since the 1960s, research has been conducted to optimize factory layouts. In production practice, iterative rearrangement of factory units remains common, as algorithms are often deemed unsuitable for generating realistic solutions. Despite significant advancements in research due to increased computational capacity and improved algorithms, this resource-intensive approach remains widespread. One reason for this persistence is that current technologies often require extensive customization for specific use cases.

This dissertation explores the use of artificial intelligence (AI) in factory layout optimization, aiming to enhance the applicability of existing methods while reducing computational demands. Both aspects are critical given the complexity of the NP-complete problem of finding optimal layouts. A key focus lies on the evaluation of layouts, which is both a determining and limiting factor in optimization algorithms – determining due to the definition of constraints and limiting due to the computational resources required. Nevertheless, simulation remains the standard for computer-assisted layout evaluation. This research contributes to the field by developing and analyzing an AI architecture capable of evaluating layouts with quality comparable to simulation. The architecture combines a convolutional neural network (CNN) for processing arbitrary layouts in image form and a multilayer perceptron (MLP) for extracting knowledge from tabular information. An evaluation conducted using five different layout configurations demonstrated that the AI can accurately predict simulated throughput for unknown layouts. This enables the creation of optimized layouts that provide factory planners with a qualified foundation and facilitate their work.

The insights gained from this study also offer potential for application in other industries where optimized resource arrangement is crucial, such as seaports, airports, or hospitals. Additionally, optimized arrangements can significantly contribute to sustainability by reducing material movements. Artificial intelligence thus demonstrates a positive impact not only in layout optimization but also beyond.

Veröffentlichungen

Während der Dissertation erschienen folgende Veröffentlichungen und Beiträge, die Konzepte und ähnliche Abbildungen dieser Arbeit enthalten:

[1] Patrick Eschemann, Philipp Borchers, Linda Feeken, Ingo Stierand, Jan-Stefan Zernickel und Martin Neumann. „Towards Digital Twins for Optimizing the Factory of the Future“. In: *Modelling and Simulation 2020 - The European Simulation and Modelling Conference, ESM 2020* (2020), S. 208–215.

[2] Patrick Eschemann, Jan Elmar Krauskopf, Jürgen Sauer und Jan Stefan Zernickel. „Optimizing Factory Layouts with Supervised Genetic Algorithm“. In: *Modelling and Simulation 2021 - The European Simulation and Modelling Conference, ESM 2021* (2021), S. 49–56.

[3] Philipp Borchers, Dennis Lisiecki, Patrick Eschemann, Linda Feeken, Mehrnoush Hajnorouzi und Ingo Stierand. „Comparison of Production Dynamics Prediction Methods to Increase Context Awareness for Industrial Transport Systems“. In: *Modelling and Simulation 2021 - The European Simulation and Modelling Conference, ESM 2021* (2021), S. 49–55.

[4] Alexandra Pehlken, Patrick Eschemann, Henriette Garmatter, Fabian Cyris und Astrid Nieße. „Einsatz von Künstlicher Intelligenz in der Digitalisierung von Abfallverbrennungskraftwerken“. In: *INFORMATIK 2021*. Gesellschaft für Informatik, Bonn, 2021, S. 147–156. DOI: 10.18420/informatik2021-011.

[5] Patrick Eschemann, Philipp Borchers, Dennis Lisiecki und Jan Elmar Krauskopf. „Metric Based Dynamic Control Charts for Edge Anomaly Detection in Factory Logistics“. In: *Journal of Physics: Conference Series*. Bd. 2352. 1. IOP Publishing. 2022, S. 012010.

[6] Markku Mikkola, Patrick Eschemann, Linda Feeken und Jarno Salonen. „Supporting Industry 4.0 implementation with modeling and simulation-case study“. In: *XXXIII ISPIM Innovation Conference: Innovating in a Digital World*. Lappeenranta University of Technology. 2022.

[7] Patrick Eschemann und Frank Oppenheimer. *Effiziente Fabriken dank künstlicher Intelligenz*. 5. Mai 2023. DOI: 10.60479/p9yn-s833. URL: https://www.wissenhochn.de/de/themen/auswahl-und-uebersicht/einzelansicht/effiziente-fabriken-dank-kuenstlicher-intelligenz (besucht am 05. 05. 2023).

[8] Patrick Eschemann, Astrid Nieße und Jürgen Sauer. „Determining Throughput of Factory Layouts with Neural Networks“. In: *Industrial Simulation Conference, ISC 2024* (2024).

Towards Digital Twins for optimizing the Factory of the Future [1]

Diese Arbeit stellt einen Ansatz und eine erste Implementierung zur Optimierung einer Flotte von automatisierten Transportfahrzeugen (AGV) vor, die Produkte zwischen Maschinen in der „Fabrik der Zukunft“ transportiert. Der Ansatz nutzt einen Digitalen Zwilling (DT), der aus einem Modell der Fabrik abgeleitet ist und die Artefakte sowie den Informationsfluss repräsentiert, die erforderlich sind, um einen gültigen DT zu erstellen. Er kann schneller als in Echtzeit ausgeführt werden, um verschiedene Konfigurationen zu bewerten, bevor die am besten passende Wahl in die reale Fabrik übernommen wird. Die Arbeit gibt auch einen Ausblick darauf, wie der DT erweitert wird, um ihn für weitere Optimierungsaspekte zu nutzen und die Widerstandsfähigkeit der AGV-Flotte gegen Anomalien zu verbessern.

Optimizing factory layouts with supervised genetic algorithm [2]

In dieser Arbeit wird das Fabriklayoutproblem anhand einer real existierenden Fabrik mit unterschiedlich großen Anlagen und unterschiedlich großem Fabrikboden untersucht. Das Ziel besteht darin, optimierte Layouts zu finden, um die Produktivität des Herstellungsprozesses zu steigern. Hierzu wird ein genetischer Algorithmus in Kombination mit einem digitalen Bild verwendet, das als visuelle Schnittstelle für einen Supervisor dient. In einer 2D- und 3D-Umgebung kann dieser manuelle Mutationen in die aktuelle genetische Algorithmuslösung einbringen. Dies erlaubt die Berücksichtigung einer unbegrenzten Inkludierung von Einschränkungen während der Layoutgenerierung. Die Lösungen werden mit einem real existierenden Fabriklayout verglichen, das über einen Zeitraum von 20 Jahren kontinuierlich verbessert wurde.

Comparison of production dynamics prediction methods to increase context awareness for industrial transport systems [3]

In dieser Arbeit werden Methoden zur Vorhersage zukünftiger Transportaufgaben zwischen Maschinen basierend auf historischen Daten untersucht. Die Leistung von innerbetrieblichen Logistiksystemen spielt eine zentrale Rolle für die Gesamtproduktivität der Fabrik der Zukunft. Ein Schlüsselelement ist die Optimierung von Logistiksystemen basierend auf prädiktiven Analysen von Transportaufgaben, um Änderungen der Produktionsflüsse in der Fabrik vorherzusehen und sich daran anzupassen. Obwohl diese Informationen aus Produktionsplänen und Maschinenkonfigurationen ableitbar sein könnten, sind die Daten aufgrund von Datenschutzbedenken oft nicht für Transportsysteme wie automatisierte

fahrerlose Transportfahrzeuge verfügbar, die von Drittanbietern bereitgestellt werden. Hier muss die Vorhersage auf beobachtbaren Informationen basieren. Dafür werden mehrere Vorhersagemethoden aus der Literatur modifiziert, um Zeitstempel und Ziele zukünftiger Aufgaben vorherzusagen. Die Ergebnisse werden mit Evaluierungsdaten aus einer Fabriksimulation anhand mehrerer Metriken verglichen, die geeignete Prädiktoren basierend auf erreichter Vorhersagegenauigkeit aufzeigen.

Einsatz von künstlicher Intelligenz in der Digitalisierung von Abfallverbrennungskraftwerken [4]

Die thermische Verwertung von Abfällen in Abfallverbrennungskraftwerken steht im Fokus dieser Arbeit. Diese gestaltet sich als komplexe Aufgabe, da das vorrangige Ziel, die Abfallbehandlung, vor dem Ziel der Energiegewinnung steht. Insbesondere die Heterogenität des aus verschiedenen Abfällen bestehenden Brennstoffes stellt vielseitige Anforderungen an den Betrieb. Zur Optimierung der Betriebsführung von Dampferzeugern in Abfallverbrennungskraftwerken auf Basis der Maximierung des Durchsatzes, der Optimierung der Strom- und Wärmeproduktion sowie der Maximierung der Standzeit der Kraftwerkskomponenten unter Berücksichtigung der Emissionsgrenzwerte werden die im Kraftwerk erhobenen Messdaten eingesetzt sowie Bilddaten erhoben. Diese werden aufbereitet und vortrainierten neuronalen Netzen zugeführt, mit dem Ziel verschiedene Materialien zu klassifizieren, um Informationen über die Abfallzusammensetzung und dem Heizwert zu erhalten.

Metric based dynamic control charts for edge anomaly detection in factory logistics [5]

Die Optimierung der Transportlogistik in Produktionsumgebungen unter Verwendung von Methoden des maschinellen Lernens wird in dieser Arbeit untersucht. Autonome fahrerlose Transportfahrzeuge die Transportaufträge in Fabriken ausführen, stehen vor der Herausforderung unvorhergesehene Änderungen und Anomalien im Produktionssystem zu erkennen und darauf zu reagieren. Aufgrund von Datenschutzbedenken sind Details wie Produktionspläne oft nicht für ein externes Transportsystem verfügbar. Daher muss die Anomalieerkennung auf selbst erfassten und beobachteten Daten des Transportsystems wie den aufgetretenen Transportbedarfen oder der Entwicklung interner Metriken basieren. In dieser Arbeit haben wird ein Produktionssystem mit Anomalien im Fertigungsprozess versehen und eine Erkennung basierend auf der Beobachtung von Transportbedarfen demonstriert, um die Lücke durch eingeschränkte Informationen zu überwinden. Für diese Erkennung wurden klassische Regelkarten erweitert, um mit erwarteten Werten zu arbeiten, die auf erlernten dynamischen Produktionscharakteristiken basieren. Das System setzt ein Toleranzfeld so eng wie möglich um dynamisch bestimmte Werte. Dies führt zu einer durchschnittlichen Präzision von 95 % bei der Erkennung variabler Mengen von Transportaufträgen.

Supporting Industry 4.0 implementation with modelling and simulation - two SME cases [6]

Dieser Artikel untersucht das Modellieren und Simulieren, das die Implementierung von Industrie 4.0 durch zwei Fallstudien in kleinen und mittleren Unternehmen (KMU) unterstützt. Die Ergebnisse deuten darauf hin, dass die Bedeutung des Modellierens und Simulierens auch im KMU-Kontext erheblich zunimmt, da der Einsatz von intelligenten Fertigungstechnologien in jeder Branche zunimmt. Anstatt umfassende Modellierungs- und Simulationswerkzeuge anzuwenden, benötigen KMU leichtere, aufgabenorientierte Werkzeuge, die sie flexibel nutzen können, um große Investitionen in Modellierungswerkzeuge sowie in die Kompetenzentwicklung zu vermeiden. Größere Unternehmen führen die Entwicklung von Industrie 4.0 an, aber um die vollen Vorteile auszuschöpfen, müssen sie ihre Lieferketten einbeziehen, die oft KMU umfassen. Weitere Forschung über die durch Industrie 4.0 verursachten Veränderungen ist erforderlich, um die verschiedenen Akteure in den Wertschöpfungsketten besser zu unterstützen. Forschung ist auch notwendig, um die Chancen zu nutzen und auf die Herausforderungen dieser Transformation zu reagieren.

Fabriklayoutoptimierung mit künstlicher Intelligenz [7]

Bei dieser Veröffentlichung handelt es sich um einen Kurztext beim Online-Magazin Wissen^{N}. Im Fokus ist der wenig verbreitete Einsatz von künstlicher Intelligenz zur Lösung des Fabriklayoutproblems. Ursachen sind hauptsächlich ein Mangel an ausreichend großen und qualitativ hochwertigen Trainingsdatensätzen. Das Fabriklayout beeinflusst die logistischen Prozesse innerhalb einer Fabrik, wie die Anordnung von Maschinen, Lagerplätzen und Kränen. Um überwachte KI-Lernmethoden einzusetzen, wurde ein Fabriklayoutgenerator entwickelt, der beliebige Fabriklayouts erstellt und somit einen großen Datensatz für das Training bereitstellen kann.

Determining throughput of factory layouts with neural networks [8]

Diese Arbeit stellt einen überwachten Lernansatz vor, der auf den Ergebnissen dieser Doktorarbeit basiert, um neuronale Netzwerke zur Bewertung von Fabriklayouts zu nutzen. Die Trainingsdaten werden durch einen zufälligen Layout-Algorithmus erstellt, der eine beliebige Anzahl von Layouts generieren kann. Die Layoutdaten werden dann von einer ereignisorientierten Simulation beschriftet. Die Layouts und Simulationsmetriken bilden das Trainingsfundament des neuronalen Netzwerks. Optimierte Parameter für die Experimente werden durch Hyperparameteroptimierung bestimmt. Die KI-Architektur ist eine Kombination aus einem konvolutionellen Netzwerk und einem mehrschichtigen Perzeptron. Es verarbeitet tabellarische und Bilddaten zur Bestimmung des simulierten Durchsatzes.

Inhaltsverzeichnis

1 Einleitung und Motivation **1**
1.1 Fokus der Arbeit 2
1.2 Aufbau der Arbeit 3

2 Forschungsfrage und Ansatz **7**
2.1 Forschungsfragen 8
2.2 Ziele und Risiken der Untersuchung 10

3 Grundlagen **13**
3.1 Theoretische Grundlagen der Layoutplanung 13
3.1.1 Herausforderungen des Fabriklayoutproblems 15
3.1.2 Modellierung des FLP 18
3.2 Ereignisorientierte Simulation 24
3.2.1 Grundlegende Konzepte der ereignisorientierten Simulation 25
3.2.2 Ereignisorientierte Simulationsmodelle 25
3.2.3 Ereignisorientierte Simulationsalgorithmen 25
3.2.4 Anwendung der ereignisorientierten Simulation in der Layoutoptimierung 26
3.2.5 Eingeschwungener Zustand 27
3.2.6 Abgrenzung zur kontinuierlichen Simulation 27
3.2.7 Vorteile und Herausforderungen der ereignisorientierten Simulation 28
3.2.8 Zukünftige Forschungsrichtungen 29
3.3 Grundlagen der künstlichen Intelligenz 30
3.3.1 Heuristische Algorithmen 31
3.3.2 Maschinelles Lernen 35
3.3.3 Deep Learning 55
3.3.4 Transfer Learning und Fine-Tuning 72
3.3.5 Active Learning und Meta-Learning 72
3.3.6 Natural Language Processing 73
3.4 Anwendung von KI in der Layoutoptimierung 74

4 Stand der Wissenschaft und Technik **77**
4.1 Wissenschaftliche Lösungsansätze zum FLP 77
4.2 Analyse wissenschaftlicher Publikationen 78
4.3 KI-basierte Veröffentlichungen zur Lösung des FLP 79

4.3.1 Veröffentlichungen zu metaheuristischen Algorithmen zur Lösung des FLP . . . 79
4.3.2 Machine Learning . . . 82
4.3.3 Deep Learning . . . 83
4.4 Anwendung von KI zur Fabriklayoutoptimierung . . . 84
4.4.1 Fazit zum aktuellen Stand . . . 85

5 Konzeption und Modellierung **87**
5.1 Konzeption zur KI-unterstützten Layoutoptimierung . . . 88
5.2 Phase 1 zur Erzeugung von Trainingsdaten . . . 89
5.2.1 Layoutgenerator . . . 89
5.2.2 Layoutbewertung durch Simulation . . . 90
5.2.3 Datenaufbereitung für das ANN . . . 93
5.3 Phase 2 - Konzeption & Entwicklung der KI-Architektur . . . 97
5.3.1 Design of Experiments für Gestaltung der KI . . . 98
5.3.2 KI-Modellauswahl mit TOPSIS . . . 99
5.3.3 Ergebnisse der optimierten Modellauswahl . . . 99
5.3.4 Architektur des künstlichen neuronalen Netzwerks . . . 101
5.3.5 Konzeption zum Training des neuronalen Netzwerks . . . 101
5.4 Phase 3 zur Integration des ANN in einen GA . . . 102
5.5 Diskussion zur Auswahl des Ansatzes . . . 104

6 Prototypische Realisierung **105**
6.1 Technische Implementierung . . . 105
6.1.1 Vorüberlegungen zur Implementierung . . . 105
6.1.2 Auswahl der Programmiersprache und Entwicklungsumgebung . . . 106
6.1.3 Vergleich zwischen Python, R und C++ . . . 107
6.1.4 Auswahl des KI-Frameworks . . . 107
6.1.5 Technische Informationen zur Ausführung des Quellcodes . . . 111
6.2 Implementierung des Layoutgenerators . . . 111
6.3 Implementierung der Layoutsimulation . . . 119
6.3.1 Modellierung der Maschinenklasse . . . 119
6.3.2 Modellierung der Klasse für das Transportmedium . . . 120
6.3.3 Helferfunktionen für die Simulation . . . 123
6.3.4 Ausführung der Simulation . . . 127
6.4 Implementierung zur Datenaufbereitung . . . 129
6.4.1 Implementierung zur Konsistenzüberprüfung . . . 130
6.4.2 Visualisierung und statistische Analyse der Zielvariablen . . . 131
6.4.3 Statistische Ausreißeranalyse . . . 132
6.5 Implementierung der Datenklasse . . . 134
6.6 Implementierung der KI-Architektur . . . 135
6.7 Implementierung des genetischen Algorithmus . . . 139
6.8 Experimente . . . 140
6.8.1 Datenerzeugung mit Layoutgenerator . . . 141
6.8.2 Dokumentation der Experimente . . . 142
6.8.3 Training des künstlichen neuronalen Netzwerks . . . 149

7 Evaluierung und Validierung **151**
7.1 Evaluierung . . . 151
7.1.1 Ergebnisse des Modells . . . 152
7.1.2 Ergebnisse auf unterschiedliche Simulationszeiten . . . 156
7.1.3 Feature Analyse . . . 157
7.1.4 Isolierte Permutationsanalyse der zusätzlichen Daten . . . 160
7.2 Validierung . . . 162
7.2.1 Überprüfung der verwendeten Daten . . . 162
7.2.2 Validierung der Generalisierbarkeit . . . 167
7.2.3 Performance-Vergleich zwischen Simulation und KI . . . 168
7.3 Vergleich mit weiteren Prädiktionsmethoden . . . 171
7.4 Vergleich mit Baselines . . . 173
7.4.1 Vergleich der Modell-Leistung mit Mittelwert-Schätzer . . . 173
7.4.2 Vergleich der Vorhersagen mit einer Zufalls-Verteilung . . . 175
7.5 Skalierung der Fabrikgröße . . . 176

8 Übertragbarkeit auf weitere Anwendungsszenarien **179**
8.1 Einarbeitung in die Thematik . . . 179
8.1.1 Grundlagen der Layoutoptimierung . . . 179
8.1.2 Spezifika von Krankenhauslayouts . . . 180
8.1.3 Einarbeitung in die Grundlagen der künstlichen Intelligenz . . . 180
8.1.4 Ereignisorientierte Simulation . . . 180
8.1.5 Fazit zur Einarbeitung in die Grundlagen . . . 180
8.2 Auswahl des Systems . . . 181
8.3 Festlegung eines Optimierungskriteriums . . . 181
8.4 Datenerzeugung und Bereinigung . . . 183
8.5 Simulation und Bewertung der Krankenhauslayouts . . . 183
8.6 Modellanpassung und -training . . . 183
8.7 Einbettung des Modells in einen Optimierungsalgorithmus . . . 184
8.8 Einführung des Layouts im Krankenhaus . . . 184

9 Schluss **185**
9.1 Zusammenfassung und Fazit . . . 185
9.2 Ausblick . . . 186

Anhang A **189**

Abbildungsverzeichnis **191**

Tabellenverzeichnis **195**

Literaturverzeichnis **197**

Abkürzungsverzeichnis **215**

1 Einleitung und Motivation

Die Gestaltung von effizienten Fabriklayouts ist ein entscheidender Faktor für die Produktivität und Wettbewerbsfähigkeit von Fertigungsunternehmen. Ein optimales Fabriklayout kann die Produktionsprozesse rationalisieren, die Durchlaufzeiten reduzieren und die Kosten minimieren. Die Herausforderung besteht darin, aus der Vielzahl von Möglichkeiten und unter Berücksichtigung von Randbedingungen das beste Layout auszuwählen. Künstliche Intelligenz (KI) zur Lösung komplexer Entscheidungsprobleme hat in vielen Industriebereichen an Bedeutung gewonnen. Unter zunehmendem Einsatz ist zu erwarten, dass die Güte von Lösungsansätzen steigen wird. Trotzdem gibt es nur eine geringe Anzahl von Ansätzen, die den Einsatz von KI im Zusammenhang zur Optimierung von Fabriklayouts untersuchen.

Im internationalen Wettbewerb sind vor allem die Qualität und Herstellungskosten[1] der Produkte ausschlaggebend für den wirtschaftlichen Erfolg. Mit steigender Qualität steigen in der Regel auch die Herstellungskosten. Diese zu reduzieren, ohne die Produktqualität herabzusetzen, stellt einen erstrebenswerten Wettbewerbsvorteil dar. Ein Beispiel für derartige Kostensenkungen ist die optimale Anordnung der betrieblichen Ressourcen in Form eines Fabriklayouts. Zu den Ressourcen zählen Maschinen, Beschäftigte und sonstige Einrichtungen. Tompkins und White [9] zeigen, dass ein effizientes Layout erstrebenswert ist, da dieses die Herstellungskosten um $10 - 30$ % vermindern kann. Dies liegt an der Minimierung von Transportkosten (engl. Material Handling Costs (MHC)), die Teil der Herstellungskosten sind. Im direkten Gegensatz dazu kann ein schlechtes Layout die Transportkosten um bis zu 36 % erhöhen [10]. Die Auswirkungen der Layoutplanung auf die Transportkosten zeigen dessen Verbundenheit mit dem wirtschaftlichen Erfolg von Unternehmen auf.

Die Relevanz von Fabriklayouts wird bei Betrachtung der Anzahl der jährlich zwischen 2001 bis 2019 in Deutschland fertiggestellten Fabriken deutlich. Im Schnitt wurden nach einer von Breitkopf [11] erstellten Statistik in Deutschland jährlich 3723 Fabrik- und Werkstattgebäude fertiggestellt. Nach einer im Jahr 2010 von Tompkins und White [9] durchgeführten Studie werden in den USA während der zweiten Hälfte des 20. Jahrhunderts jährlich ca. acht Prozent des Bruttoinlandsprodukts für neue Fabriken oder deren Umbauten aufgewendet. Dies ist ein im Kontext der Fabriklayoutplanung oft zitierter Wert, da er deutlich aufzeigt, dass sich Fabrikplaner jedes Jahr viele Gedanken zum optimalen Layout machen. Das Forschungsinstitut *IPH Hannover* bietet explizit Beratungsleistungen zur Layoutplanung

[1] §255 (2) Handelsgesetzbuch (HGB): Herstellungskosten sind Aufwendungen, die durch den Verbrauch von Gütern und die Inanspruchnahme von Diensten für die Herstellung eines Vermögensgegenstands, seine Erweiterung oder für eine über seinen ursprünglichen Zustand hinausgehende wesentliche Verbesserung entstehen. Dazu gehören die Materialkosten, die Fertigungskosten und die Sonderkosten der Fertigung.

und -optimierung an [12]. Das Unternehmen *fabrik-ID* bietet 2D- und 3D-Layoutplanung von Produktionsstätten und die Erschaffung *enormer Wettbewerbsvorteile* an. Zu den Kunden zählen laut Internetseite des Anbieters viele namhafte Unternehmen wie DÜRR und TRUMPF [13]. Beide Anbieter bieten Fachkompetenz und setzen verschiedene Methoden zur Layoutoptimierung ein. Die erzielbaren Lösungen sind abhängig von Expertenwissen und basieren auf einem aufwendigen, manuellen Prozess.

Neben dem industriellen Kontext ist die Suche nach dem optimalen Layout in der Forschung als das *Layoutplanungsproblem* bekannt. Mit diesem ist das der Layoutplanung zugrunde liegende Entscheidungsproblem aufgrund der Vielzahl an Möglichkeiten und zu berücksichtigenden Variablen bezeichnet. Layoutplanungsprobleme gehören Scholz zufolge mathematisch zu den schwierigsten Problemen der kombinatorischen Optimierung [14].

1.1 Fokus der Arbeit

Der Fokus dieser Dissertation liegt auf dem Fabriklayoutproblem und der Verwendung von neuronalen Netzwerken zur Vorhersage des voraussichtlichen Materialdurchsatzes von Fabriklayouts. Zum Einsatz kommen hier zwei Netzwerkarchitekturen. Der erste Typ ist spezialisiert für die Verarbeitung von Bilddaten, die zweite Architektur eignet sich zur Erkennung von Mustern in tabellarischen Daten. Durch das Training des neuronalen Netzwerkes mit Bildern und weiteren Informationen von Fabriklayouts wird das Ziel verfolgt, eine auf KI basierende Methode zur Bewertung von Fabriklayouts zu entwickeln, siehe Abbildung 1.

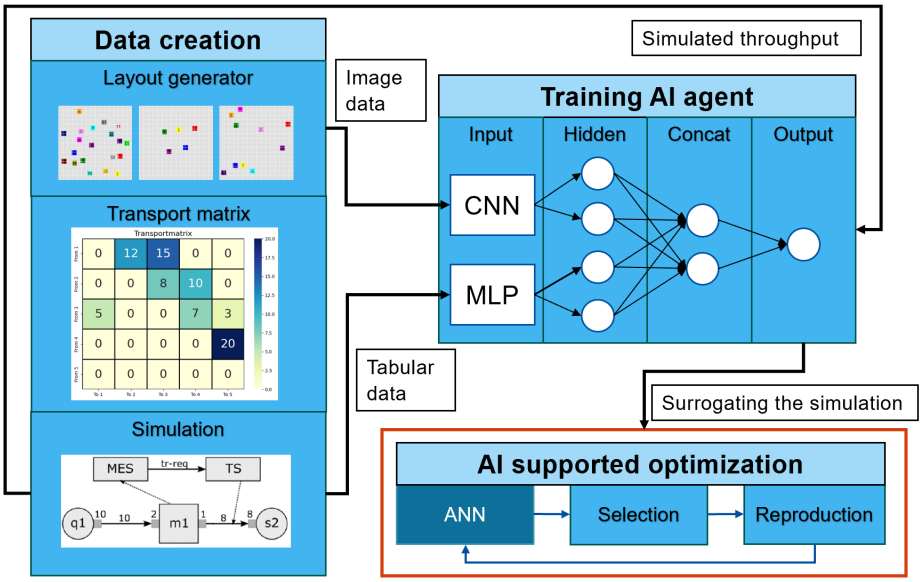


Abbildung 1: Konzept zur KI-unterstützten Layoutevaluation

Zur Erstellung der benötigten Trainingsdaten wurde ein Layoutgenerator konzipiert und entwickelt, der zufällige Fabriklayouts in beliebiger Anzahl generiert. Bei der Erstellung werden weitere Metriken wie die Manhattan-Distanz berechnet. Der durch ereignisorientierte Simulation ermittelte Materialdurchsatz dient als *Ground-Truth* Bewertung der Layouts. Ein zentraler Aspekt dieser Dissertation ist die Untersuchung, inwiefern ein neuronales Netzwerk den simulierten Durchsatz abbilden kann. Dies würde für zukünftige Forschungen neben der Simulation von Layouts eine weitere Möglichkeit zur Bewertung von Layouts eröffnen. Die Konzeption wird im Detail in Kapitel 5 erläutert.

Diese Arbeit verfolgt das Ziel, einen Beitrag zum wissenschaftlichen Forschungsstand im Bereich der Fabriklayoutoptimierung und des Einsatzes von künstlicher Intelligenz zu leisten. Hierzu werden zunächst die theoretischen Grundlagen zur Fabrikplanung, zur ereignisorientierten Simulation, zu metaheuristischen Methoden sowie zur künstlichen Intelligenz dargelegt. Darauf aufbauend wird eine Methodik zur Datenerhebung, Datenverarbeitung sowie zur Entwicklung und zum Training der Modellarchitektur detailliert beschrieben. Im Anschluss werden die Ergebnisse der durchgeführten Experimente analysiert und diskutiert. Abschließend erfolgt eine Zusammenfassung der Arbeit, ergänzt durch einen Ausblick auf potenzielle zukünftige Forschungsrichtungen.

Der Beitrag soll das Verständnis für die Anwendung von künstlicher Intelligenz zur Lösung des Fabriklayoutproblems vertiefen und praktische Anwendungen in der Industrie ermöglichen. Die Ergebnisse dieser Forschung haben das Potenzial, die Entscheidungsfindung in der Fabrikplanung zu verbessern und zur Optimierung von Produktionsprozessen beizutragen.

1.2 Aufbau der Arbeit

Das folgende Kapitel gibt einen Überblick über den Aufbau der vorliegenden Doktorarbeit und Kurzzusammenfassungen zu den Inhalten der Kapitel.

Kapitel 1: Einleitung und Motivation

Die Einleitung und Motivation bilden den ersten Teil dieser Dissertation. In diesem Abschnitt erfolgt eine allgemeine Einführung in das Thema und eine Darstellung der Relevanz des Fabriklayoutproblems. Dazu werden die Auswirkungen optimierter Layouts für die Produktivität und Wettbewerbsfähigkeit von Fertigungsunternehmen dargestellt. Zudem wird die Verwendung von künstlicher Intelligenz als Ansatz zur Lösung des Fabriklayoutproblems vorgestellt.

Kapitel 2: Forschungsfrage und Ansatz der Arbeit

In Kapitel 2 erfolgt die Formulierung der Forschungsfragen, an denen sich die weitere Dissertation orientiert. Die Leitfrage handelt davon, wie und für welche Aspekte künstliche Intelligenz zur Layoutoptimierung eingesetzt werden kann. Die hierzu verfolgten Hypothesen, Ziele und Abgrenzungen werden ebenfalls in diesem Kapitel eingeführt. Dazu zählt die Festlegung auf das *Open Field Layout Problem* auf die Ebene der *Feinlayoutplanung*.

Kapitel 3: Grundlagen

In Kapitel 3 werden die Grundlagen für die in dieser Dissertation enthaltenen Themenbereiche dargelegt. Zu diesen gehören das Fabriklayoutproblem sowie die Grundlagen zur ereignisorientierten Simulation und der künstlichen Intelligenz. Im Sinne der aufgezeigten Forschungsfragen erfolgt jeweils eine kurze Diskussion zum möglichen Einsatz der jeweiligen Techniken zur Optimierung von Fabriklayouts.

Kapitel 4: Stand der Wissenschaft und Technik

In Kapitel 4 wird der aktuelle wissenschaftliche Stand zur Fabriklayoutoptimierung dargestellt. Dazu werden relevante Literaturquellen analysiert und diskutiert. Die verschiedenen Ansätze zur Lösung des Fabriklayoutproblems zeigen Herausforderungen, Forschungslücken und Ansätze zur Konzeption dieser Dissertation auf. Insbesondere wird der wenig verbreitete Einsatz von neuronalen Netzwerken zur Layoutoptimierung deutlich. Dies ist unter anderem auf das Fehlen qualitativ hochwertiger, digitalisierter Layoutdaten in ausreichender Menge sowie auf fehlende Konzepte zur Implementierung von Methoden, die künstliche Intelligenz nutzen, zurückzuführen.

Kapitel 5: Konzeption und Modellierung

In Kapitel 5 wird das übergeordnete Konzept zur Layoutoptimierung unter Verwendung von künstlicher Intelligenz erläutert. In diesem Zusammenhang wird die technische Funktionalität und Implementierung vom Layoutgenerator, der ereignisorientierten Simulation und der Datenverarbeitung erklärt. Weiterhin wird die Architektur des neuronalen Netzwerks, das aus einer Kombination zweier spezifischer Typen besteht, eingeführt. In einem stufenweisen Trainingsprozess wird ein neuronales Netzwerk anhand einer Fabrikkonfiguration mit vier Maschinen vortrainiert. Per Transferlearning wird das gleiche Modell auf weitere Datensätze mit steigender Komplexität trainiert.

Kapitel 6: Prototypische Realisierung

In Kapitel 6 werden die technische Implementierung erläutert und die durchgeführten Experimente zur Bestimmung des Durchsatzes von Fabriklayouts mit dem trainierten

neuronalen Netzwerk beschrieben. Die gesetzten Parameter und Randbedingungen werden zur Nachvollziehbarkeit des Vorgehens aufgezeigt.

Kapitel 7: Validierung und Diskussion der Ergebnisse

In Kapitel 7 werden die Ergebnisse der Experimente anhand einer Hypothese validiert. Nach dieser muss das neuronale Netz den Durchsatz in ähnlicher Qualität wie die Simulation voraussagen bei einer höheren Verarbeitungsgeschwindigkeit. Die hierzu durchgeführten Untersuchungen bestätigen die Hypothese.

Kapitel 8: Übertragbarkeit auf weitere Anwendungsszenarien

In Kapitel 8 wird die Übertragbarkeit des in dieser Dissertation eingeführten Konzepts auf ein weiteres Szenario untersucht, konkret anhand einer Optimierung von Layouts in Krankenhäusern. Die Vorgehensweise wird in acht aufeinanderfolgenden Schritten strukturiert dargestellt. Zusammenfassend zeigt sich, dass eine Übertragung auf dieses und weitere Szenarien möglich ist, sofern aufgrund der Spezifika verwandter Szenarien Anpassungen auf der Ebene des Quellcodes vorgenommen werden.

Kapitel 9: Schluss

In Kapitel 9 wird ein Fazit zu den gewonnenen Erkenntnissen gezogen. Die Ergebnisse der Arbeit werden hervorgehoben und in einen Gesamtkontext zur Beantwortung der Forschungsfragen eingeordnet. Ein Ausblick gibt Hinweise zu praktischen Anwendungen, gefolgt von weiterführenden Fragen und offenem Forschungsbedarf.

2 Forschungsfrage und Ansatz

Dieser Dissertation liegt die These zugrunde, dass der Einsatz von Techniken aus dem Bereich der KI, insbesondere in Kombination mit metaheuristischen Algorithmen, das Potenzial besitzt, verbesserte Ergebnisse bei der Layoutoptimierung zu erzielen. Um diese These im Rahmen der Dissertation zu überprüfen, erfolgt in diesem Abschnitt die Formulierung einer Forschungsleitfrage, sowie weiterführende Teil-Forschungsfragen.

Der Forschungsbereich der Layoutplanung und -optimierung unterteilt sich in verschiedene Disziplinen. Die formulierte Forschungsfrage enthält daher die Beschränkung auf das Open Field Layout Problem (OFLP), siehe Kapitel 3.1. Hier liegt der Fokus auf die Optimierung der Anordnung von Fabrikeinheiten für einen rechteckigen Grundriss auf der Ebene der Feinlayoutplanung.

Als Randbedingungen werden zudem folgende Informationen vorausgesetzt:

- Grundriss und Fläche vom Layout
- Anzahl von Fabrikeinheiten sowie deren räumliche Ausdehnung
- Verweilzeiten einzelner Produkte an den Fabrikeinheiten
- Materialfluss
- Anlagenverfügbarkeit der einzelnen Fabrikeinheiten
- Puffergrößen an den Fabrikeinheiten

Die Voraussetzung der aufgezählten Informationen stellt kein unrealistisches Szenario dar, die ersten vier genannten Punkte stehen in der Regel vor Planung eines konkreten Layouts zur Verfügung. Für die weiteren Informationen stellen die Anlagenhersteller entsprechende Erfahrungswerte zur Verfügung.

Die Optimierung von Layouts ist ein kombinatorisches Problem, das aufgrund eines exponentiell mit der Problemgröße wachsenden Rechenaufwandes als NP-vollständig gilt. In die Problemgröße gehen neben der Anzahl an Fabrikeinheiten eine zu berücksichtigende Anzahl an Randbedingungen ein. Grundsätzlich ist die praktische Anwendbarkeit des Layouts höher bei Inklusion von mehr Randbedingungen. Ein Ziel ist es daher mehr Fabrikeinheiten oder Randbedingungen durch einen auf künstliche Intelligenz basierenden Optimierungsprozess bereitzustellen, ohne den Rechenaufwand exponentiell zu erhöhen.

2.1 Forschungsfragen

Gegenstand dieser Dissertation sind folgende Fragestellungen:

- **Forschungsleitfrage**

  „Wie und für welche Aspekte kann künstliche Intelligenz zur Layoutoptimierung eingesetzt werden?“

- **Forschungsfragen**

  FF1: Welche Schwierigkeiten bestehen bei der Layoutoptimierung und wie können diese überwunden werden?

  FF2: Wie kann eine auf KI-basierende Architektur gestaltet werden, um sie in metaheuristische Lösungsverfahren einzubinden?

  FF3: Inwiefern bietet der Einsatz künstlicher Intelligenz zur Layoutoptimierung gegenüber anderen computerunterstützten Verfahren eine Verbesserung?

Folgende Darstellung zeigt die Aktivitäten und zu erwartenden Artefakte zu den aufgezählten Forschungsfragen, s. Abbildung 2.

Forschungsfragen	Aktivitäten	Artefakte
FF1 Ist- und Solldefinition	Literaturrecherche	Ist-Stand und Hypothesen
	Befragungen	Priorisierungen und Potenziale
	Aggregation und Synthese	Zieldefinition
FF2 Konzept und Umsetzung	Konzept	Architektur
	Implementierung 1	Experimentgrundlage
	Implementierung 2	KI-Prototyp
FF3 Weiterentwicklung	Experimente	Evaluierung und Validation
		Weiterführende Forschungsansätze

Abbildung 2: Aktivitäten und Artefakte zu den Forschungsfragen

Im linken Block sind die Forschungsfragen $1-3$ dargestellt. Der mittlere Block zeigt die durchzuführenden Aktivitäten zu der jeweiligen Forschungsfrage auf. Der rechte Block zeigt die zu erwartenden Artefakte und Ergebnisse, die der Beantwortung der Forschungsfrage dienen. Die erste Forschungsfrage ist durch eine Literaturrecherche, durch Befragungen und eine Synthese der gewonnenen Erkenntnisse geprägt. Im Ergebnis lässt sich der Ist-Stand und Hypothesen ableiten. Ferner sind Priorisierungen und Potenziale gesetzt worden, mit dem Ziel geeignete KI-Techniken und Konzepte zu ermitteln, die in die nachfolgenden Phasen einfließen. Die zweite Forschungsfrage besteht zum einen aus der Erstellung

eines Konzeptes und zum anderen aus zwei Implementierungsblöcken. In diesen wird das Konzept umgesetzt und ein KI-Prototyp, ein Simulator sowie ein Layoutgenerator bereitgestellt. Der Layoutgenerator soll anhand einer definierbaren Anzahl an Fabrik- oder Organisationseinheiten in beliebiger Anzahl unterschiedliche Layouts erzeugen können. Der Simulator soll zu jedem Layout den voraussichtlichen Durchsatz bestimmen können. Der KI-Prototyp soll anhand der Layoutbilder in Kombination mit tabellarischen Daten den simulierten Durchsatz bestimmen können. Die dritte Forschungsfrage ist von der Durchführung und Auswertung von Experimenten geprägt. Nach einer Evaluierung und Validierung der Ergebnisse zeigt sich, inwieweit das eingeführte Verfahren eine Verbesserung gegenüber bestehenden Verfahren bietet. Zusammengefasst besteht die Zielsetzung in der konzeptuellen Ausgestaltung und Erprobung eines KI-Prototypen, mit dem der Durchsatz eines Fabriklayouts vorhergesagt werden kann.

Zum Zwecke der KI-unterstützten Layoutoptimierung wird ein neuronales Netzwerk benötigt, das Layouts in der gleichen Qualität wie eine Simulation bewerten kann. Software, die direkt verwendet werden kann, ist im Zeitraum der Untersuchung (2020 - 2024) nicht bekannt. Daher werden die notwendigen Elemente im Rahmen der Untersuchung implementiert. Das Vorgehen orientiert sich hierbei an den in Abbildung 3 dargestellten Softwareentwicklungslebenszyklus (engl. Software Development Life Cycle (SDLC)).

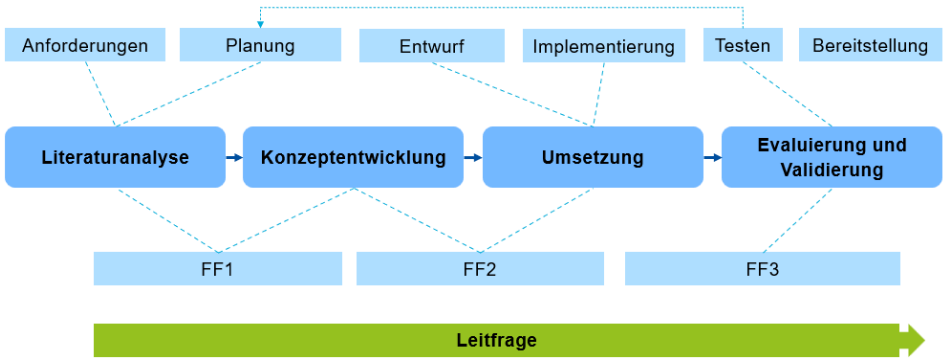


Abbildung 3: Forschungsmethodik

Angesichts der mit den aufgestellten Forschungsfragen einhergehenden, erforderlichen Implementierung skizziert Abbildung 3 die einzelnen Schritte gemäß dem SDLC. Der Beginn ist durch eine Literaturanalyse (s. Kapitel 4) gekennzeichnet, aus der die Anforderungen, Einschränkungen und die weitere Planung abgeleitet werden. Dieser Schritt trägt zur Beantwortung der ersten Teilfrage bei. Die in Kapitel 5 entwickelte Definition des Konzepts sowie dessen Umsetzung in Kapitel 6 resultieren in einem Prototyp, der die zweite Teilfrage dieser Arbeit adressiert. Zur Beantwortung der dritten Teilfrage wird der Prototyp in Kapitel 7 hinsichtlich seiner Eignung im Kontext der Fabriklayoutoptimierung bewer-

tet und validiert. Der Prozess von der Planung bis zur Bewertung wird agil in mehreren Iterationen durchgeführt und adressiert im Ganzen die Leitfrage dieser Arbeit.

2.2 Ziele und Risiken der Untersuchung

Das übergeordnete Ziel der KI-unterstützten Layoutoptimierung besteht neben der Erhöhung des Durchsatzes in verbesserten Layouts auch in der Steigerung der Resilienz gegenüber Störungen. Durch den Einsatz von KI soll der Lösungsraum effizienter exploriert und dadurch optimierte Layouts identifiziert werden. Dieses Ziel wird durch eine auf die Vorhersage des Durchsatzes trainierte KI erreicht, die Wissen aus den Ergebnissen einer ereignisorientierten Simulation extrahiert. Zum Zeitpunkt der Recherche konnte kein vergleichbarer Ansatz identifiziert werden. Zwar wurden im Rahmen einer Literaturrecherche einige wenige verwandte Arbeiten ermittelt, diese weisen jedoch lediglich begrenzte Überschneidungen mit dem hier verfolgten Ansatz auf.

Die folgende Auflistung führt Risiken auf, die der Forschungsthese entgegenstehen können:

- **Datenqualität und -quantität:** Umfangreiche Datensätze sind für das Training künstlicher neuronaler Netzwerke notwendig. Unzureichende Datenmengen können zu einer eingeschränkten Modellleistung führen.
- **Modellkomplexität und Generalisierung:** Komplexe Modelle können zwar feine Muster in den Daten erkennen, neigen jedoch zur Überanpassung (Overfitting), insbesondere bei unzureichenden Datenmengen. Einfachere Modelle sind weniger anfällig für Overfitting, bergen jedoch das Risiko, relevante Muster nicht zu erkennen (Underfitting). Zusätzlich besteht die Gefahr, dass Optimierungsalgorithmen in lokalen Optima steckenbleiben.
- **Allgemeine Gültigkeit des Modells und der Annahmen:** Bei der Entwicklung kann die Gültigkeit der Annahmen auf spezifische Fabriklayouts begrenzt und möglicherweise nicht übertragbar sein.
- **Verarbeitungs- und Speicheranforderungen:** Training und Inferenz von neuronalen Netzen erfordern erhebliche Rechenressourcen und limitieren die untersuchbaren Modell- und Problemgrößen.
- **Risiken bei der Implementierung:** Fehler während der Implementierungsphase können zu Fehlinterpretationen, Verzögerungen und zusätzlichen Kosten führen.
- **Aktualisierung und Wartung des Modells:** Künstliche neuronale Netzwerke erfordern regelmäßige Updates und Anpassungen, um langfristig leistungsfähig zu bleiben. In einer Doktorarbeit, die durch begrenzte personelle, zeitliche und finanzielle Ressourcen geprägt ist, stellen diese Anforderungen ein Risiko dar.

- **Robustheit und Anfälligkeit für Angriffe:** Künstliche neuronale Netze können anfällig für adversarielle Angriffe sein, bei denen geringfügige Eingabeänderungen zu falschen Ausgaben führen können.
- **Interpretierbarkeit und Transparenz:** Künstliche neuronale Netze werden häufig als „Black Boxes“ bezeichnet, da ihre Entscheidungsfindung nur schwer nachvollziehbar ist. Dies stellt ein erhebliches Risiko dar, insbesondere in Anwendungen, bei denen die Nachvollziehbarkeit der Ergebnisse essenziell ist, etwa zur Fehlersuche, Validierung oder zur Überzeugung von Stakeholdern.
- **Datenschutz und ethische Bedenken:** Der Umgang mit sensiblen Daten, insbesondere Betriebsgeheimnissen wie Layouts, erfordert die Einhaltung von Datenschutzrichtlinien und ethischen Standards. Solche Daten können nicht ohne Weiteres für automatisierte Auswertungen verwendet werden, da dies rechtliche und vertraulichkeitsbezogene Risiken birgt. Zudem muss sichergestellt werden, dass die Verarbeitung von Daten keine unzulässigen Verzerrungen oder ungerechtfertigte Entscheidungen fördert.

Die aufgezählten Risiken wurden durch das in Abbildung 3 vorgestellte methodische Vorgehen minimiert. Dieses umfasste eine Literaturrecherche, ein ausgearbeitetes Konzept, die Planung, Gestaltung und Auswertung der Experimente. Die Literaturrecherche half, ein Verständnis für den Umgang mit Risiken in ähnlichen Arbeiten zu entwickeln. Ferner wurden *Lessons Learned* und *Best Practices* in Bezug auf die Datensammlung, Modellauswahl, Modelltraining und Modellvalidierung identifiziert und in das Konzept integriert. Um das Risiko mangelnder Transparenz und Nachvollziehbarkeit („Black Box“) zu reduzieren, wurden zusätzlich Methoden der Explainable AI (XAI) eingesetzt. Insbesondere kamen Mutations- und Perturbationsanalysen zum Einsatz, um die Sensitivität des Modells gegenüber Veränderungen in den Eingabedaten zu untersuchen. Diese Ansätze ermöglichten eine strukturierte Bewertung der Modellentscheidungen und identifizierten Einflussfaktoren, die Stakeholdern eine bessere Nachvollziehbarkeit der Ergebnisse ermöglichten. Um die finanziellen und komplexitätsbedingten Risiken bei begrenzten Kapazitäten zu minimieren, wurde der Fokus der Forschung definiert und eingegrenzt. Entlang eines abgestimmten Meilensteinplans wurde gewährleistet, dass die Ergebnisse der Dissertation den festgelegten Zielen entsprechen. Zur Minderung der Datenschutz- und Vertraulichkeitsrisiken wurden die Layouts anonymisiert und verfremdet, um die Identität der beteiligten Unternehmen zu schützen. Es wurde zudem sichergestellt, dass keine Partei Zugriff auf Layouts anderer Unternehmen hatte. Beispielsweise entstand im Rahmen der Arbeit eine Veröffentlichung, bei der das verwendete Layout anonymisiert und das bereitstellende Unternehmen geheim gehalten wurde, siehe [2].

3 Grundlagen

In diesem Kapitel werden die mit der wissenschaftlichen Fragestellung zusammenhängenden Konzepte und Theorien vorgestellt. Dies umfasst die Grundlagen der Layoutplanung, ereignisorientierten Simulation und der künstlichen Intelligenz.

3.1 Theoretische Grundlagen der Layoutplanung

Die Layoutplanung ist ein wichtiger Aspekt bei der Gestaltung und Organisation von Produktionsstätten. Sie befasst sich mit der Anordnung von Maschinen, Arbeitsplätzen, Materialflüssen, Lagerflächen und weiteren Elementen innerhalb einer Fabrik, um einen optimierten Produktionsablauf zu gewährleisten. Die Hauptziele der Layoutplanung sind:

- Minimierung von Materialflusskosten
- Maximierung der Raumausnutzung
- Verbesserung der Arbeitsbedingungen
- Steigerung der Flexibilität
- Erhöhung der Sicherheit

In der VDI2385 ist ein *Layout* als die grafische Darstellung der räumlichen Anordnung von betrieblichen Funktions- und Struktureinheiten definiert. Zu den Einheiten zählen unter anderem Fertigungs- und Montageplätze, Lager, Produktionsbereiche und Transporteinrichtungen. Ein neues oder verändertes Fabriklayout wird durch eine Erweiterung der Produkte, Einbeziehung neuer Technologien und Anpassung der Fertigungskapazitäten notwendig. Laut Scholz beschäftigt sich die Layoutplanung mit der Frage nach einer *günstigen* Platzierung von Organisationseinheiten (OE) auf einem Standortträger [14]. Zu den OE zählen sämtliche anzuordnende Elemente (Maschinen, Roboter, Kräne, Lager etc.). Der Standortträger wird als räumlich eng umgrenzt angenommen, wie beispielsweise eine Produktionshalle. Somit ist die Layoutplanung abzugrenzen von der betrieblichen Standortplanung. Nach Wirth et al. [15] und Grundig [16] lässt sich die Layoutplanung von Produktionsstätten in die in Abbildung 4 dargestellten drei hierarchischen Ebenen unterteilen.

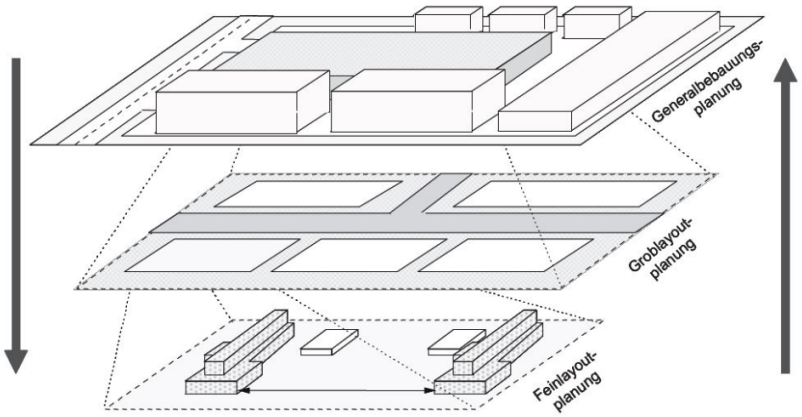


Abbildung 4: Ebenen der Layoutplanung [17]

In der Generalbebauungsplanung wird die Anordnung von Strukturelementen wie Bürogebäuden, Lagerflächen, Parkplätzen und Produktionshallen festgelegt. Die Grobplanung befasst sich mit der Positionierung von Fertigungs- und Lagerflächen innerhalb einer Produktionshalle. In der Feinlayoutplanung werden weitere Details festgelegt. Zu diesen gehören unter anderem die Auslegung und Positionierung von Fertigungsanlagen, Arbeitsplätzen und Logistiksystemen. Eine Anpassung des Werkes an veränderte Produktionsbedingungen wird vorwiegend auf Ebene der Feinlayoutplanung vorgenommen, lediglich in seltenen Fällen wird die Groblayoutplanung angepasst [17]. Bei großen Unternehmen mit bestehendem Erfahrungsschatz zu Werksplanungen wird oftmals auf Standardlayouts für die ersten beiden Stufen der Layoutplanung zurückgegriffen. Die Feinlayoutplanung muss für ein optimales Ergebnis mit dem neuesten technischen Stand zu verfügbaren Maschinen, Produktionstechniken und ggf. gesetzlichen Änderungen neu konzipiert werden. Die Feinlayoutplanung ist demzufolge der Bereich der höchsten Dynamik im Kontext der Fabrikplanung. Grundig führt noch eine vierte Ebene ein, das Arbeitsstationslayout [16]. Hier wird die optimale Platzierung der einzelnen Betriebsmittel einer Arbeitsstation betrachtet.

Nach der VDI2385 hat die Layoutplanung das Ziel, durch die richtige Anordnung von Struktureinheiten und Verbindungselementen den Fertigungsablauf wirtschaftlich und störungssicher zu ermöglichen [18]. Aufgrund der Abhängigkeit von Randbedingungen und unterschiedlichen Zielsetzungen variieren die Angaben zu Optimierungskriterien in unterschiedlichen Quellen. Zu allgemein gültigen Zielen der Layoutplanung und insbesondere der Layoutoptimierung zählen nach Francis et al. folgende Punkte [19]:

- Minimierung von Investitionskosten
- Minimierung der Fertigungszeit
- Minimierung der benötigten Fläche

Die Auslegung eines Fabriklayouts unterliegt einer Vielzahl von zu berücksichtigen

Randbedingungen. Zu diesen gehören unter anderem Gehwegbreiten, Art des Transportmediums, Deckenhöhe, Maschinenabmaße, Energieanbindung etc. Für ein optimales und realistisches Layout sollten möglichst viele Randbedingungen berücksichtigt werden, allerdings vergrößert dies erheblich den Aufwand bei der Modellierung und Bewertung. Die Schwierigkeit, eine optimale Verteilung von Elementen zu ermitteln, ist in die Wissenschaft als das *Facility Layout Problem (FLP)*, zu Deutsch Layoutplanungsproblem, eingegangen. Exakte Lösungen gibt es nur für stark vereinfachte und klein skalierte Sachverhalte. Derartige Ansätze eignen sich zur theoretischen Forschung, allerdings nicht für die praktische Anwendung. Aufgrund des starken praktischen Bezugs des FLP werden daher überwiegend metaheuristische Verfahren angewendet. Zum FLP wird seit den fünfziger Jahren intensiv Forschung betrieben. Erste Arbeiten wurden von Koopmanns und Beckmann (1957) [20], Wilson (1964) [21], El-Rayah (1970) [22] und Moore (1974) [23] veröffentlicht.

3.1.1 Herausforderungen des Fabriklayoutproblems

Untersuchungen zum FLP befassen sich mit der optimalen Anordnung von OE innerhalb der Fabrikfläche unter Berücksichtigung einer Anzahl von zu definierenden Randbedingungen. In Publikationen zum FLP sind die Größe der zur Verfügung stehenden Fläche, Form, Orientierung, vorgegebene Zonen für Annahme bzw. Abgabe von Gütern und Abstände zwischen Einrichtungen die am häufigsten berücksichtigten Randbedingungen [24]. Welche im Einzelfall berücksichtigt werden, hängt von der Betrachtungsweise und Zielsetzung der jeweiligen Untersuchung ab. Aufgrund dessen gibt es mehr als ein Dutzend verschiedene Klassifikationen des FLP. Alle werden als *NP-vollständig* eingestuft, d. h. die benötigte Rechenzeit zur Lösung des Problems steigt exponentiell mit dessen Größe [25]. Neben der Schwierigkeit der geometrischen Layoutplanung im Sinne einer Parkettierung[2], sind die Ermittlung von Kriterien, die ein Layout bewertbar machen und das Aufstellen eines Modells, das die zu erwartende Produktion darstellt, Gegenstände aktueller Forschung. Im Folgenden wird aufgezeigt, wie das FLP bereits bei Berücksichtigung weniger Randbedingungen eine hohe Komplexität erreicht.

3.1.1.1 Form des Layouts

Wird die Form des Layouts betrachtet, werden rechteckige und ungleichförmige Fabrikgrundrisse unterschieden, s. Abbildung 5.

[2] Mathematischer Begriff zur Beschreibung der lückenlosen, überlappungsfreien Überdeckung einer zweidimensionalen Ebene.

Abbildung 5: Rechteckiger und ungleichförmiger Grundriss [26]

Ungleichmäßige Grundrisse zeichnen sich durch Polygone aus, von denen ein Kantenwinkel mindestens 270 Winkelgrad beträgt [26]. Die meisten wissenschaftlichen Veröffentlichungen basieren auf rechteckigen Grundrissen [24].

3.1.1.2 Transport- und Materialflüsse

Über die Form hinaus wird die Komplexität der Layoutplanung durch Betrachtung unterschiedlicher Transportflüsse weiter erhöht, s. Abbildung 6.

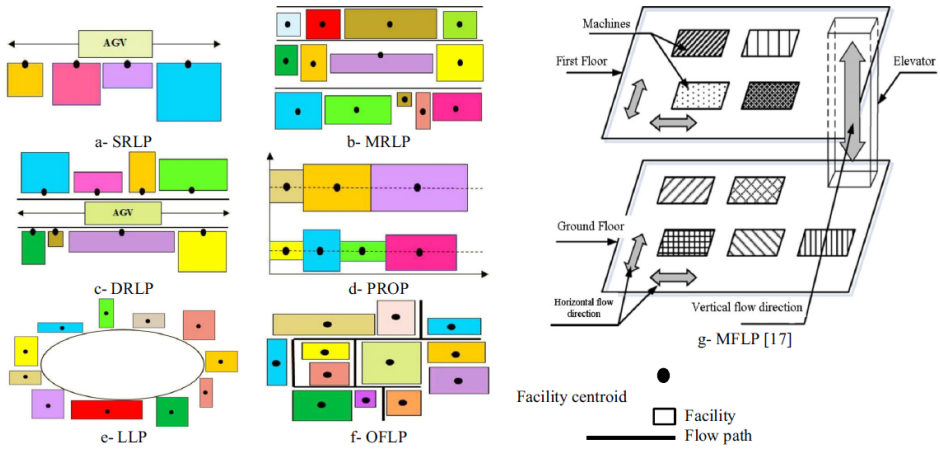


Abbildung 6: Layoutdesign unter Berücksichtigung der Transportflüsse [24]

(a) Das *Single-Row Layout Problem (SRLP)* beschäftigt sich mit der optimalen Platzierung von rechteckigen Einrichtungen entlang einer Linie, um die Transportkosten zu minimieren, die durch die Analyse der Produktströme zwischen den Mittelpunkten (Facility Centroid) der OE bestimmt werden. (b) Das *Multi-Row Layout Problem (MRLP)* fügt eine Anzahl von Reihen in den zweidimensionalen Raum ein, wobei das Ziel darin besteht, die am besten geeignete Reihe für eine Anlage zu ermitteln. (c) Das *Double-Row Layout Problem (DRLP)* bezieht sich auf die optimale Anordnung von rechteckigen Einrichtungen unterschiedlicher Breite, die an gegenüberliegenden Seiten des Bodens platziert werden, um die Handhabungskosten eines *Automated Guided Vehicle (AGV)*-Systems zu minimieren. (d) Im *Parallel-Row Ordering Problem (PROP)* werden Einrichtungen mit gemeinsamen Charakteristiken entlang einer Reihe und die übrigen Einrichtungen auf der parallelen Reihe angeordnet, wobei PROP annimmt, dass zwischen den Reihen ein Raum besteht, im Gegensatz zum DRLP, das keine

Zwischenräume vorsieht. (e) Das *Loop Layout Problem (LLP)* befasst sich mit der Suche nach der optimalen Position von n-Anlagen in einem geschlossenen Ringnetz, zur Reduzierung der Transportkosten [27]. (f) Das *OFLP* gilt für Situationen, in denen kein vordefinierter Materialfluss, wie in (a)-(e) aufgezeigt, anwendbar ist. (g) Bei mehrstöckigen Gebäuden werden zusätzlich vertikale Transportflüsse berücksichtigt. Derartige Sachverhalte sind als *Multi-Floor Layout Problem (MFLP)* definiert [28]. Zusätzliche Randbedingungen wie die Transportart, Transportkapazität, Verbindungsort und -anzahl erhöhen die Komplexität des MFLP. Patsiatzis und Papageorgiou [29] haben das Problem untersucht und ein Framework zur simultanen Bestimmung der Anzahl der Stockwerke, Fläche, Belegung und einzelnen Layouts vorgestellt. Für einen weiterführenden Einstieg zum MFLP wird auf eine aktuelle Studie zu diesem Thema von Ahmadi [30] verwiesen. Die meisten wissenschaftlichen Veröffentlichungen gibt es zum SRLP, MRLP und OFLP, s. Hosseini [24].

Über die generellen Materialflüsse hinaus werden zwei Arten der Flussbewegung unterschieden, das *Backtracking* und *Bypassing*. Zur grafischen Veranschaulichung s. Abbildung 7.

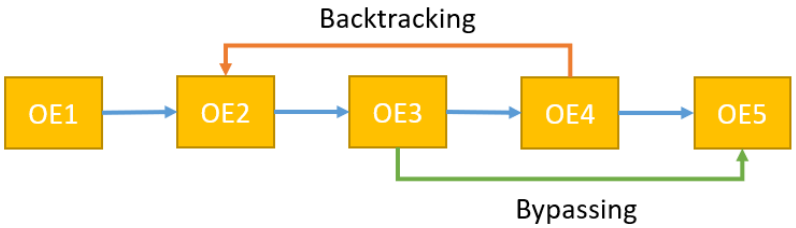


Abbildung 7: *Backtracking* und *Bypassing*, Darstellung nach [26]

In diesem Beispiel wird eine Produktionslinie mit fünf Stufen gezeigt. Die Kästen sind beliebige Fertigungsstufen (Maschinen, Roboter usw.) und die blauen Pfeile zeigen den Materialfluss an. In einem effizienten Layout durchlaufen die Produkte nacheinander die OE. Die meisten Fabriken stellen jedoch unterschiedliche Produkte her und nicht jedes Produkt folgt der gleichen linearen Prozedur. Das *Backtracking* ist das Fließen eines Teils in entgegengesetzter Richtung. Dabei wird außerdem eine Einrichtung übersprungen. Das *Bypassing* tritt auf, wenn ein Teil eine Einrichtung in Flussrichtung überspringt. Derartige Abweichungen vom linearen Materialfluss sind nach Zhou [31] als das *Production Line Formation Problem (PFLP)* benannt. Generell wird es im Rahmen der Layoutoptimierung angestrebt, das *Backtracking* und *Bypassing* zu vermeiden.

3.1.1.3 Zielkriterien des FLP

Wie im vorigen Abschnitt dargestellt leiten sich aus der Form des Layouts und der Art des Transportflusses unterschiedliche Herausforderungen für die Ermittlung eines optimalen Layouts ab. Wann ein Layout optimal ist, hängt von dem gewählten Zielkriterium

ab. Die meisten wissenschaftlichen Ansätze versuchen ein Layout zu ermitteln, in dem alle Maschinen, Logistikelemente, Lagerplätze und sonstige am Wertschöpfungsprozess beteiligten Elemente möglichst nah beieinander angeordnet sind. Das FLP mit nur einem Zielkriterium wird auch *Single-Criterion FLP* genannt [24]. Weitere Zielkriterien können die Höhe der verfügbaren Investitionskosten, das Minimieren von Backtracking/Bypassing, die Energieeffizienz (Green Factory [32]) und weitere sein. Werden mehrere Zielkriterien bei der Lösungssuche vorgegeben, spricht man von einem *Multi-Criterion FLP*.

3.1.2 Modellierung des FLP

Eine erste mathematische Beschreibung des FLP wurde von Koopmans und Beckmann [20] im Jahr 1957 vorgenommen. Diese haben das FLP als allgemeines quadratisches Zuordnungsproblem (engl. *Quadratic Assignment Problem (QAP)*) beschrieben [20]. In der QAP-Modellierung ist das Layout in n-gleich große, rechteckige Standorte unterteilt, s. Abbildung 8.

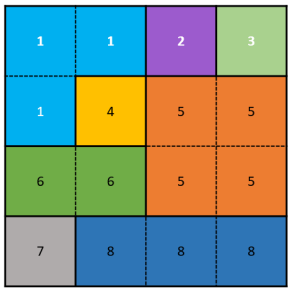


Abbildung 8: Diskrete Repräsentation von Fabriklayouts, Darstellung nach [26]

Bei der diskreten Repräsentation ist jede OE einem Standort zugeordnet und umgekehrt [33]. Größere Anlagen werden mehreren Standorten zugeordnet. In Abbildung 8 ist die Fabrikhalle in 16 Quadrate unterteilt, die von acht beliebigen Einrichtungen belegt sind. Die Einrichtungen 2, 3, 4 und 7 belegen genau einen Standort, die anderen belegen je nach Größe mehrere. Diese Art der Modellierung eignet sich nicht zur Darstellung der exakten Positionierung von Elementen innerhalb der Werkshalle, da sie wie dargestellt zu grob ist. Für feinere Auflösungen müsste die Rasterung erhöht werden, was bei realistischen Fabrikdimensionen erhebliche Rechenleistung benötigt. Als weiteren Nachteil eignet sich die Modellierung nicht, um spezifische Einschränkungen zur Ausrichtung von Elementen, Aufnahme- und Abgabepunkten oder den Abstand zwischen Elementen darzustellen [26]. Eine weitere Art der Modellierung von Fabriklayouts ist die kontinuierliche Repräsentation, s. Abbildung 9.

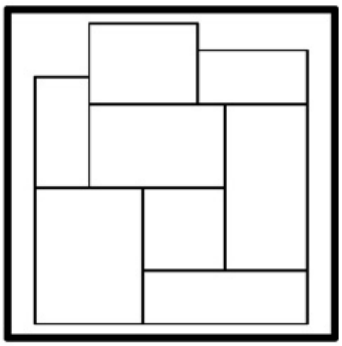


Abbildung 9: Kontinuierliche Repräsentation von Fabriklayouts nach [26]

Bei der kontinuierlichen Repräsentation wird die Fläche in verschieden große Rechtecke unter folgenden Annahmen verteilt [24]:

- Alle Elemente müssen sich innerhalb eines bestimmten, festen Rechtecks befinden.
- Die Anzahl, Fläche und Transportflüsse aller Elemente sind im Voraus bekannt.
- Die Rechtecke überschneiden sich nicht und liegen innerhalb der definierten Grenzen.
- Das Layout muss Beschränkungen zu der maximalen und minimalen Dimension für die Abmessungen der Elemente (Länge und Breite) vorsehen.

3.1.2.1 Übersicht zu Modellierungsklassen

Neben der vorgestellten Art der Modellierung existieren weitere Ansätze aus den mathematischen Teilgebieten der Optimierung und Graphentheorie (GT). Die zum Zeitpunkt dieser Ausarbeitung aktuellste Studie aus 2018 von Hosseini [24] unterscheidet insgesamt sieben verschiedene Modellierungsklassen zum FLP. In einer Studie [34] zum FLP aus dem Jahre 2012 wurden noch vier verschiedene Modellierungsklassen unterschieden. Die nachfolgende Tabelle 1 basiert auf den Ergebnissen von Hosseini [24]. Sie gibt einen Überblick zu den Modellierungsklassen und beschreibt für welche Repräsentation des Layouts sie eingesetzt werden können.

Tabelle 1: Modellierungsklassen zum FLP nach [24]

Modellierungsklasse	*Erläuterung und Einsetzbarkeit*
Quadratic Assignment Problem (QAP)	Wie in Kapitel 3.1.2 erwähnt, eine diskrete Darstellung bei der der Grundriss in gleich große Quadrate eingeteilt wird.
Quadratic Set Covering Problem (QSP)	Eine Modellierung mit ungleich großen Einrichtungen und diskreter Darstellung kann als QSP formuliert werden. Der gesamte von allen Einrichtungen eingenommene Grundriss wird in mehrere kleinere Blöcke aufgeteilt, so dass jede Einrichtung nur einem Standort zugeordnet ist und jeder Block als höchstens eine Einrichtung betrachtet wird [34].
Linear Programming (LP)	LP ist eine Technik zur Optimierung einer linearen Zielfunktion, die linearen Randbedingungen unterworfen ist.
Integer Programming (IP)	IP-Modell ist eine mathematische Optimierung, bei der alle Variablen ganze Zahlen sein müssen.
Mixed Integer Programming (MIP)	MIP ein Modell zur Formulierung eines FLP mit einer kontinuierlichen Repräsentation [35]. Es besteht aus einer Zielfunktion, die ganzzahlige und nicht ganzzahlige Entscheidungsvariablen enthält und einer Reihe von Gleichheits- und Ungleichheitsbeschränkungen unterliegt [34].
Nonlinear Programming Problem (NLPP)	Das Problem wird als NLPP bezeichnet, wenn die Zielfunktion nichtlinear ist und/oder der machbare Bereich durch nichtlineare Zwänge bestimmt wird.
Graphentheorie (GT)	Mithilfe der Graphentheorie kann das Anlagenlayout als ebener Graph mit gewichteten Kanten modelliert werden. Die Knoten stellen Anlagen dar und die Kanten zeigen Beziehungen zwischen den Anlagen auf [36]. Das Kantengewicht stellt entweder die Kosten oder den Nutzen zweier benachbarter Einrichtungen dar. Wenn das Kantengewicht die Kosten (den Nutzen) repräsentiert, besteht das Problem darin, eine Anordnung zu finden, die die Gesamtkosten (den Nutzen) minimiert (maximiert). Bei diesem Ansatz wird davon ausgegangen, dass der erwünschte Standort benachbarter Maschinen bekannt ist [37].

Die nachfolgende Abbildung 10 zeigt die Verbreitung der aufgezählten Modellierungen nach der Anzahl der wissenschaftlichen Veröffentlichungen.

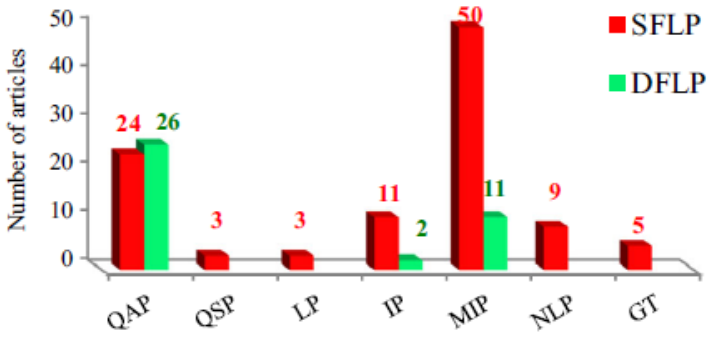


Abbildung 10: Anzahl der FLP-Publikationen nach mathematischem Modell [24]

Neben dem Verbreitungsgrad wird zusätzlich zwischen dem *Static Facility Layout Problem (SFLP)* und *Dynamic Facility Layout Problem (DFLP)* unterschieden. Eine Erläuterung hierzu folgt in Kapitel 3.1.2.3. Von den genannten Modellierungsklassen wird die QAP- und MIP-Modellierung am häufigsten angewendet.

3.1.2.2 QAP-Modellierung des FLP

Im Rahmen der mathematischen Formulierung des FLP als ein QAP besteht das Hauptziel darin, die relativen Standorte der Anlagen so zu bestimmen, dass die MHC minimiert werden. Die entsprechende Zielfunktion wird wie folgt mathematisch beschrieben [26, 38]:

$$min \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} \sum_{l=1}^{N} f_{ik} d_{jl} X_{ij} X_{kl} \tag{1}$$

N	Anzahl der Standorte in einem Layout
f_{ik}	Transportkosten von i zu k
d_{jl}	Distanz von Ort j zu l
X_{ij}	1, wenn Einrichtung i dem Standort j zugeordnet ist
X_{ij}	0, wenn Einrichtung i nicht dem Standort j zugeordnet ist
i, k	Fabrikelemente im Layout
j, l	Standorte im Layout

Diese Gleichung summiert die Produkte der Transportkosten f_{ik} zwischen zwei Fabrikelementen i und k und der Distanzen d_{jl} zwischen den ihnen zugewiesenen Standorten j und l. Die binären Variablen X_{ij} und X_{kl} nehmen den Wert 1 an, wenn das Fabrikelement i bzw. k dem Standort j bzw. l zugeordnet ist, und 0 sonst. Diese Formulierung ermöglicht die Erfassung der Interaktion zwischen den Positionen der Anlagen und den dadurch entstehenden Transportkosten.

Zur Einhaltung der Grundregeln des Layouts werden folgende Nebenbedingungen eingeführt:

$$\sum_{i=1}^{N} X_{ij} = 1, \quad j = 1, \ldots, N \tag{2}$$

$$\sum_{j=1}^{N} X_{ij} = 1, \quad i = 1, \ldots, N \tag{3}$$

Die zweite Gleichung stellt sicher, dass jeder Standort genau einer Anlage zugewiesen wird. Die dritte Gleichung gewährleistet umgekehrt, dass zur Sicherung der Konsistenz und Vollständigkeit des Layouts jede Anlage genau einem Standort zugeordnet ist [36].

3.1.2.3 Statisches und dynamisches FLP

Viele wissenschaftliche Ansätze gehen von anhaltend gleichbleibenden Verhältnissen eines einmal festgelegten Layouts aus. Nach Gupta und Seifoddini [39] werden allerdings in einem Drittel aller Fabriken der USA alle zwei Jahre größere Änderungen vorgenommen.

Auf Grundlage dieses Zusammenhangs wird das FLP weiter in das *Static Facility Layout Problem (SFLP)* und *Dynamic Facility Layout Problem (DFLP)* unterteilt [40]. Die erste bekannte Arbeit und Beschreibung zum Problem eines sich dynamisch ändernden Fabriklayouts wurde von Rosenblatt [41] veröffentlicht. Grundsätzlich basieren sowohl das statische als auch das dynamische FLP auf dem QAP. Im Unterschied zum SFLP bezieht das DFLP nicht nur MHC, sondern auch Unkosten durch dynamische Veränderungen des Materialflusses über einen fest definierten Planungshorizont mit ein [40]. Dieser wird periodisch in Wochen, Monaten oder Jahren unterteilt. Während einer Periode wird der Materialfluss als statisch bzw. vorhersagbar angenommen.

Allgemein beschrieben umfasst das DFLP die Fragestellung, welches (statische) Layout für die derzeitige Periode optimal ist und ob eine Umstellung des Layouts zur Anpassung an die folgende Periode sinnvoll ist [42]. Maßgebend sind unter anderem die Kosten durch eine Anpassung des Layouts, erzielbare Effizienzsteigerung und weitere Kriterien. Balakrishnan et al. [43] haben folgende Formel zum DFLP als Erweiterung des QAP aufgestellt, die sowohl die laufenden Betriebskosten als auch die Umstellungskosten berücksichtigt:

$$min \sum_{t=1}^{P} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} f_{tik} d_{djl} X_{tij} X_{tkl} + \sum_{t=2}^{P} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} A_{tijl} Y_{tijl} \tag{4}$$

Die Definitionen der Variablen und Konstanten lauten wie folgt:

P	Anzahl der Perioden im Planungshorizont
n	Anzahl der Standorte in einem Layout
i, k	Fabrikelemente im Layout
j, l	Standorte im Layout
f_{ik}	Transportkosten von i zu k
d_{jl}	Distanz von j zu l
Y_{tik}	Die 0, 1 Variable i vom Standort j zu l im Zeitraum t zu transportieren
X_{tij}	Die 0, 1 Variable um i am Standort j im Zeitraum t zu lokalisieren
A_{tijl}	Zusätzliche Transportkosten von j zu l im Zeitraum t (wobei $A_{tijj} = 0$ ist)

Die Zielfunktion minimiert die kumulierten Transportkosten über mehrere Perioden, die durch die Entscheidungen für die Platzierung von Einrichtungen und deren Änderungen über die Zeit beeinflusst werden. Unter den Nebenbedingungen:

$$\sum_{i=1}^{n} X_{tij} = 1, \quad j = 1, \ldots, n, \quad t = 1, \ldots, P \tag{5}$$

$$\sum_{j=1}^{n} X_{tij} = 1, \quad i = 1, \ldots, n, \quad t = 1, \ldots, P \tag{6}$$

$$Y_{tijl} = X_{(t-1)ij} X_{til}, \quad i, j, l = 1, \ldots, n, \quad t = 2, \ldots, P \tag{7}$$

Hierbei stellt Y_{tijl} eine Binärvariable dar, die angibt, ob die Einrichtung i vom Standort j zum Standort l zwischen den Perioden $t-1$ und t verschoben wird. Dies reflektiert die Umstellungskosten, die im zweiten Summanden der Zielfunktion berücksichtigt werden.

3.1.2.4 Lösungsmethodik

Nach einer Untersuchung zur Forschungslage unterteilen Balakrishnan und Cheng [44] und Culturel-Konak [45] die Lösungsverfahren zum FLP in metaheuristische und exakte Algorithmen. Wird ein Layout-Problem über t Perioden betrachtet, in denen jeweils jede mögliche Anordnung vorkommen kann, multipliziert sich die Anzahl der möglichen Layouts für jede Periode mit der Anzahl der möglichen Layouts mit denen der folgenden Perioden. Balakrishnan et al. [43] ermitteln für ein Layout mit sechs OE über fünf Perioden $1,93 * 10^{14}$ mögliche Anordnungen:

$$(n!)^t = 6!^5 = 720^5 = 1.93 \times 10^{14} \tag{8}$$

Bei sechs Abteilungen gibt es 720 mögliche Anordnungen pro Periode (6! = 720). Über fünf Perioden betrachtet, ist die theoretische Anzahl der Kombinationsmöglichkeiten 720^5, was etwa $1,93 \times 10^{14}$ entspricht. Dies zeigt, dass für realistisch konzipierte Berechnungen

nur metaheuristische Verfahren mit vertretbarem Rechenaufwand bei der Lösungssuche eingesetzt werden können. Verfahren dieser Art sind u. a. genetische Algorithmen, Tabu-Search, Simulated Annealing und kombinierte Ansätze. In Abbildung 11 sind eine Übersicht metaheuristischer Verfahren und die Häufigkeit ihrer Anwendung in wissenschaftlichen Publikationen dargestellt.

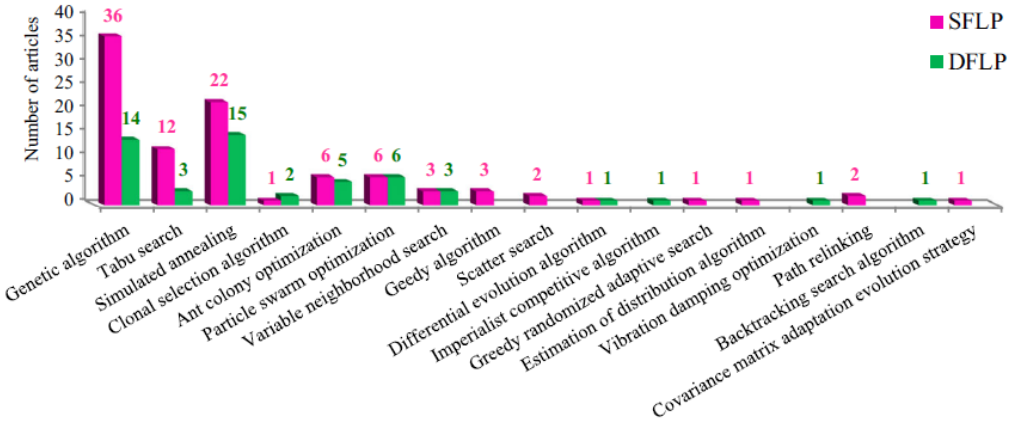


Abbildung 11: Anzahl von Publikationen mit metaheuristischen Lösungsverfahren [24]

Genetische Algorithmen, Tabu-Search und Simulated Annealing sind die meist angewendeten Optimierungsverfahren zur Lösung des FLP. Die aufgeführten Techniken werden im folgenden Kapitel 3.3.1 eingeführt.

3.2 Ereignisorientierte Simulation

In dieser Dissertation nimmt die ereignisorientierte Simulation (engl. Discrete Event Simulation (DES)) eine zentrale Rolle bei der Generierung von Metriken für zufällig erzeugte Layouts ein. Dieses Kapitel widmet sich den Grundlagen der ereignisorientierten sowie der kontinuierlichen Simulation und erläutert deren Einsatzmöglichkeiten zur Untersuchung komplexer Systeme. Besonders hervorgehoben wird die Anwendung dieser Methoden im Bereich des *Operations Research*[3] und der Systemanalyse, mit einem Fokus auf der Optimierung von Fabriklayouts. Ferner erfolgt eine Darstellung und Abgrenzung der ereignisorientierten gegenüber der kontinuierlichen Simulation, um die jeweiligen Stärken und Anwendungsbereiche beider Simulationsarten insbesondere bezogen auf die Fabriklayoutoptimierung zu verdeutlichen.

[3] Operations Research ist ein Bereich der angewandten Mathematik, der sich mit der Entwicklung und Anwendung analytischer Methoden zur Entscheidungsfindung und Problemlösung beschäftigt.

3.2.1 Grundlegende Konzepte der ereignisorientierten Simulation

Die DES beruht auf der Idee, dass Systeme durch eine Sequenz diskreter Ereignisse modelliert und analysiert werden können, die den Zustand des Systems im Laufe der Zeit verändern [46]. Ein Ereignis ist definiert als eine zeitliche Veränderung, die einen Einfluss auf den Zustand des Systems hat. Beispiele für Ereignisse in einer Fertigungsumgebung sind das Eintreffen von Materialien, das Fertigstellen von Produkten und das Wechseln von Arbeitsstationen.

3.2.2 Ereignisorientierte Simulationsmodelle

Ein ereignisorientiertes Simulationsmodell besteht aus vier Hauptkomponenten:

- **Entitäten**: Entitäten sind die grundlegenden Objekte oder Elemente des Systems, die Ereignisse auslösen oder von Ereignissen betroffen sind. In einer Fertigungsumgebung können Entitäten Produkte, Maschinen, Arbeitsstationen oder Materialien sein.
- **Attribute**: Attribute sind Eigenschaften oder Merkmale, die den Zustand der Entitäten beschreiben. Beispiele für Attribute sind die Größe, das Gewicht oder die Bearbeitungszeit eines Produkts.
- **Ereignisse**: Wie bereits erwähnt, sind Ereignisse zeitliche Veränderungen, die den Zustand des Systems beeinflussen. Ereignisse werden in einer Ereignisliste gespeichert und in chronologischer Reihenfolge abgearbeitet.
- **Systemzustand**: Der Systemzustand ist eine Sammlung von Variablen, die den aktuellen Zustand des Systems zu einem bestimmten Zeitpunkt beschreiben. Der Systemzustand ändert sich in Abhängigkeit von den auftretenden Ereignissen.

Die aufgelisteten Komponenten bilden die Grundstruktur eines ereignisorientierten Simulationsmodells und eröffnen die Möglichkeit diverse Systeme zu modellieren.

3.2.3 Ereignisorientierte Simulationsalgorithmen

Ereignisorientierte Simulationsalgorithmen bilden die Logik für die Analyse und Optimierung dynamischer Systeme ab. Diese Algorithmen folgen einem allgemeinen Rahmen, der in mehreren spezifischen Implementierungen variieren kann, jedoch stets die gleichen grundlegenden Schritte beinhaltet:

- **Initialisierung**: Festlegung des initialen Systemzustands und Erstellung der Ausgangs-Ereignisliste.
- **Ereignisverarbeitung**: Identifikation und Bearbeitung des nächsten Ereignisses aus der Ereignisliste zur Anpassung des Systemzustands und der betroffenen Entitäten.

- **Ereignisgenerierung**: Erzeugung neuer Ereignisse als Reaktion auf die Verarbeitungsergebnisse, die zur Ereignisliste hinzugefügt werden.
- **Terminierung**: Abschluss der Simulation basierend auf vordefinierten Kriterien wie der Erreichung eines Endzeitpunkts oder der Verarbeitung einer festgelegten Anzahl von Ereignissen.
- **Analyse und Statistik**: Auswertung der gesammelten Daten zur Bewertung der Systemleistung und zur Ableitung statistischer Kennzahlen.

Diese Schritte bilden die Basis für die Modellierung und Ausführung ereignisorientierter Simulationen, die je nach Anwendungsfall und spezifischer Problemstellung angepasst werden können. Variablen wie die Generierungsfrequenz von Ereignissen und die Kriterien für die Terminierung variieren typischerweise zwischen verschiedenen Algorithmen zur flexiblen Anpassung an die jeweiligen Untersuchungsbedingungen.

3.2.4 Anwendung der ereignisorientierten Simulation in der Layoutoptimierung

Die DES kann in der Fabriklayoutoptimierung eingesetzt werden, um die Leistung verschiedener Layoutvarianten zu bewerten und einen Vergleich zu ermöglichen. Dabei werden die Layouts als Systeme modelliert, und die Simulation wird verwendet, um die Auswirkungen von Änderungen auf die OE-Platzierung, Materialflüsse, Durchlaufzeiten, Ressourcenauslastung und andere relevante Leistungskennzahlen zu untersuchen.

Ein typisches Vorgehen bei der Anwendung der ereignisorientierten Simulation in der Fabriklayoutoptimierung umfasst die folgenden fünf Schritte:

1. **Modellentwicklung**: Zunächst wird ein Simulationsmodell entwickelt, das das Fabriksystem und das Layout in Form von Entitäten, Attributen und Ereignissen repräsentiert. Dabei werden die relevanten Aspekte des Systems, wie zum Beispiel Materialflüsse, Maschinen, Arbeitsstationen und Lagerhaltung, berücksichtigt.
2. **Datenerfassung**: Um das Simulationsmodell mit realistischen Daten zu versorgen, werden historische Daten oder Expertenschätzungen zu Materialflüssen, Bearbeitungszeiten, Ressourcenauslastung und anderen relevanten Faktoren gesammelt und analysiert.
3. **Experimentdesign**: Basierend auf den zu untersuchenden Layoutvarianten wird ein experimentelles Design entwickelt, das die Durchführung von Simulationsexperimenten und die Erhebung von Daten ermöglicht.
4. **Simulationsexperimente**: Die Simulationsexperimente werden durchgeführt, indem das Simulationsmodell für die verschiedenen Layoutvarianten ausgeführt wird. Dabei werden die Systemleistung und andere relevante Kennzahlen erfasst.
5. **Ergebnisanalyse und Entscheidungsfindung**: Die Ergebnisse der Simulationsexperi-

mente werden analysiert, um die Leistung der Layoutvarianten zu bewerten und zu vergleichen. Basierend auf den Ergebnissen können Entscheidungen über die Auswahl des optimalen Layouts getroffen werden.

Die ereignisorientierte Simulation ist somit ein Werkzeug zur Bewertung und Optimierung von Fabriklayouts, da sie die Möglichkeit bietet, die Auswirkungen von Layoutänderungen auf das Systemverhalten und die Leistung zu untersuchen, ohne dass Änderungen im realen Fabrikbetrieb vorgenommen werden müssen.

3.2.5 Eingeschwungener Zustand

Der eingeschwungene Zustand (engl. „Steady State“) bezieht sich auf den stabilen Zustand eines Systems, in dem sich die Schlüsselvariablen des Modells nicht mehr signifikant ändern [46]. In den Anfangsphasen einer Simulation können die Zustände des Modells durch bestimmte Anfangsbedingungen beeinflusst werden, die nicht repräsentativ für das typische Verhalten des Systems sind. Beispielsweise könnten in einem Fertigungsszenario Pufferspeicher leer sein oder es könnten noch keine Aufträge in der Warteschlange sein. Um solche anfänglichen Verzerrungen zu eliminieren, wird in der Simulation eine Aufwärmphase definiert. Während dieser erreicht das System einen stabilen Zustand, der das normale Verhalten repräsentiert. Bei der simulationsbasierten Bewertung eines Systems werden Daten erst nach Erreichen dieses stabilen Zustands gesammelt. Dies stellt sicher, dass die Analyse auf repräsentativen, nicht verrauschten Daten basiert.

3.2.6 Abgrenzung zur kontinuierlichen Simulation

Die kontinuierliche Simulation (engl. Continuous Simulation (CS)) unterscheidet sich von der ereignisorientierten Simulation hinsichtlich des Modellierungsansatzes und der Art, wie die Zeit im Simulationsmodell behandelt wird. In diesem Abschnitt werden die Grundkonzepte der kontinuierlichen Simulation erläutert und ein Vergleich zwischen der ereignisorientierten und kontinuierlichen Simulation bezüglich der Eignung zur Bewertung von Fabriklayouts durchgeführt.

3.2.6.1 Grundkonzepte der kontinuierlichen Simulation

Die CS beruht auf der Modellierung von Systemen durch Differenzialgleichungen, die die zeitlichen Veränderungen von Systemvariablen beschreiben, wie z. B. Temperatur, Druck oder Konzentrationen in chemischen Prozessen. Eine typische allgemeine Form für solche Modelle lautet:

$$\frac{dx(t)}{dt} = f(x(t), u(t), t) \tag{9}$$

wobei $x(t)$ den Zustand, $u(t)$ Eingangsgrößen und t die Zeit darstellt. Die Funktion f beschreibt die Dynamik des Systems. Initialbedingungen wie $x(0) = x_0$ legen den Startpunkt fest.

Im Gegensatz zur DES, bei der Systemzustände diskret und ausschließlich durch Ereignisse verändert werden, wird in der CS die zeitliche Dynamik fortlaufend dargestellt. Anwendungsbereiche sind die Modellierung physikalischer, chemischer oder biologischer Prozesse, bei denen kontinuierliche zeitliche Veränderungen von Relevanz sind.

Zur Lösung werden häufig numerische Methoden wie das Runge-Kutta-Verfahren eingesetzt. Simulationstools wie MATLAB/Simulink oder Python-Bibliotheken wie SciPy.

3.2.6.2 Vergleich zwischen ereignisorientierter und kontinuierlicher Simulation

Die Wahl zwischen DES und CS zur Bewertung von Fabriklayouts richtet sich nach den spezifischen Anforderungen und Eigenschaften des jeweiligen Systems. DES eignen sich besonders für die Modellierung von diskreten Systemen, bei denen der Systemzustand durch eine Sequenz einzelner Ereignisse verändert wird. Dazu zählen insbesondere Fertigungssysteme, bei denen Materialflüsse, Maschinenfunktionen und Arbeitsprozesse typischerweise einzeln folgende Ereignisse sind. Weiterhin zeigen DES prinzipbedingte Vorteile im Umgang mit komplexen Fertigungssystemen, bei denen die Materialflüsse und Ressourcenauslastung von stochastischer Natur ist sowie von einer Vielzahl verschiedener Ereignisse beeinflusst werden. Zuletzt erlauben DES eine Analyse der Systemleistung durch Identifikation von Engpässen, Ineffizienzen und Verbesserungspotenzialen in Fertigungssystemen. CS hingegen sind besser geeignet für die Modellierung von Systemen, bei denen die Systemzustände kontinuierlich und dynamisch über die Zeit variieren, unter der Voraussetzung, dass die Zustandsänderungen durch kontinuierliche Funktionen der Zeit beschrieben werden können. Beispiele sind thermische oder fluide Prozesse.

Insgesamt zeigt die DES eine höhere Eignung für die Bewertung von Fabriklayouts, insbesondere in Bezug auf diskrete Fertigungssysteme, bei denen Materialflüsse, Maschinenfunktionen und Arbeitsprozesse im Vordergrund stehen. Die DES erlaubt eine Analyse der Systemleistung und zeigt Auswirkungen von Layoutänderungen auf Fertigungsprozesse und Ressourcenauslastung. Die CS ist in Szenarien überlegen, bei dem die Simulation des Energieverbrauchs einer Maschine oder des gesamten Layouts erfolgen soll. Sollte diese bei der Layoutoptimierung im Fokus stehen, kann die CS als ergänzendes Werkzeug zur DES verwendet werden. Derartige hybride Simulationsansätze berücksichtigen sowohl diskrete als auch kontinuierliche Aspekte des Systems.

3.2.7 Vorteile und Herausforderungen der ereignisorientierten Simulation

Die Anwendung der DES in der Fabriklayoutoptimierung bietet folgende Vorteile:

- **Risikominimierung**: Virtuelle Experimente erlauben das Testen und Bewerten von Layoutvarianten, ohne reale Änderungen vorzunehmen. Dies reduziert Fehlentscheidungen und minimiert die Kosten von Layoutänderungen.
- **Effizienz**: Die Bewertung von vielen Layoutvarianten erfolgt schnell und ressourcenschonend.
- **Flexibilität und Anpassungsfähigkeit**: Verschiedene Szenarien und Parameter lassen sich flexibel analysieren, um dynamischen Anforderungen gerecht zu werden.
- **Nachvollziehbarkeit**: Ergebnisse sind reproduzierbar und erleichtern durch visuelle Darstellungen eine Kommunikation der Optimierungsvorschläge.
- **Modularität**: DES-Modelle sind erweiterbar und anpassbar.

Trotz dieser Vorteile gibt es folgende Herausforderungen:

- **Modellierungskomplexität**: Die Modellierung von komplexen Fertigungssystemen und Layouts wird nie vollständig erfolgen und ist stets als Abstraktion zu verstehen. Diese ist limitiert durch die verfügbare Zeit und vorhandenen Mittel.
- **Datenerfassung und -qualität**: Die Qualität der Simulationsmodelle und der daraus resultierenden Ergebnisse hängt von den Eingabedaten ab. Diese können unvollständig, ungenau oder veraltet sein.
- **Validierung und Verifikation**: Die Validierung und Verifikation der Simulationsmodelle dienen dem Abgleich mit dem realen System, bedingen jedoch Expertenwissen und Erfahrung.

3.2.8 Zukünftige Forschungsrichtungen

Zukünftige Forschungen im Bereich der DES und Fabriklayoutoptimierung könnten sich auf die Entwicklung von Methoden zur Verbesserung der Modellierungskomplexität, Datenqualität und Validierung konzentrieren. Darüber hinaus könnten neue Techniken und Ansätze erforscht werden, um die Genauigkeit der Simulationsergebnisse zu erhöhen und den Entscheidungsprozess zu unterstützen (sog. Decision Support System (DSS)).

Beispiele sind die Integration von künstlicher Intelligenz in die DES, um die Modellierung, Analyse und Optimierung von Fabriklayouts zu automatisieren und zu verbessern. Des Weiteren könnte die Entwicklung von hybriden Simulationsansätzen, die DES mit anderen Modellierungstechniken wie agentenbasierten Ansätzen oder CS kombinieren, zu einer besseren Abbildung von Fertigungssystemen und deren Dynamik beitragen. Schließlich könnte die Anwendung von fortgeschrittenen Datenanalyse- und Visualisierungstechniken dazu beitragen, die Interpretation und Kommunikation von Simulationsresultaten zu verbessern.

Ein weiterer Forschungsbereich ist die Erkundung von Echtzeit- und adaptiven Simulationsansätzen, bei denen Simulationsmodelle kontinuierlich anhand von Echtzeitdaten aus

der Produktionsumgebung aktualisiert und angepasst werden (Digital Twin). Dies könnte dazu beitragen, die Reaktionsfähigkeit und Flexibilität von Fertigungssystemen in einer zunehmend dynamischen Umwelt zu erhöhen, siehe hierzu auch die im Kontext dieser Dissertation entstandene Veröffentlichung von Eschemann et al. [1].

3.3 Grundlagen der künstlichen Intelligenz

Künstliche Intelligenz (KI) ist ein interdisziplinäres Forschungsgebiet, das sich mit der Entwicklung von Algorithmen, Methoden und Systemen beschäftigt, die es Computern ermöglichen, Aufgaben auszuführen, die normalerweise menschliche Intelligenz erfordern [47]. Die KI umfasst eine Vielzahl von Subdisziplinen und Techniken, einschließlich maschinellem Lernen, Deep Learning, heuristischer Verfahren, Optimierung, Wissensrepräsentation und -verarbeitung, Computer Vision und natürlicher Sprachverarbeitung [48].

Die Erforschung und Anwendung von KI haben das Ziel, Systeme zu entwickeln, die autonom oder in Zusammenarbeit mit Menschen intelligente Entscheidungen treffen und Probleme lösen können. Dabei werden Aspekte wie Lernen, Anpassungsfähigkeit, Abstraktion, Planung und Problemlösung berücksichtigt [49]. KI-Techniken werden in einer Vielzahl von Anwendungen eingesetzt, von der Automatisierung industrieller Prozesse über die medizinische Diagnostik bis hin zur Steuerung autonomer Fahrzeuge [50, 51, 52].

In Kontext dieser Dissertation eröffnet die KI verschiedene Ansätze zur Layoutoptimierung, mit der Einsetzbarkeit zur Lösung komplexer, dynamischer und unvorhersehbarer Problemstellungen. Die Einbindung von maschinellem Lernen, Optimierungsalgorithmen und künstlichen neuronalen Netzen ermöglicht es, den Planungsprozess zu automatisieren, die Leistungsfähigkeit der Lösungen zu verbessern und die Reaktionsfähigkeit auf Veränderungen in der Produktionsumgebung zu erhöhen [26].

Im Zusammenhang zur Layoutoptimierung besonders hervorzuhebende Eigenschaften von KI-basierten Methoden sind:

- Verarbeitung und Analyse großer Datenmengen zur Mustererkennung.
- Automatisiert optimale oder nahezu optimale Layouts erzeugen, die auf den spezifischen Anforderungen und Zielen des Unternehmens basieren.
- Flexibel auf Veränderungen in der Produktionsumgebung reagieren und das Layout kontinuierlich anpassen.

Mit diesem Abschnitt des Grundlagenkapitels zur KI wird die erste Teil-Forschungsfrage zur Einsetzbarkeit der KI zur Layoutoptimierung adressiert. Hierzu wird das Einsatzgebiet von KI beginnend bei den heuristischen Methoden bis hin zu fortschrittlichen Techniken wie maschinellem Lernen, Deep Learning im Hinblick auf ihre Anwendbarkeit auf das FLP hin untersucht. In den folgenden Unterkapiteln werden dementsprechend neben den

allgemeinen Grundlagen der jeweiligen KI-Technik, ihre Anwendung in der Fabriklayoutoptimierung und die spezifischen Vor- und Nachteile diskutiert.

3.3.1 Heuristische Algorithmen

Heuristische Algorithmen sind einfache, praktische Regeln oder Strategien, die verwendet werden, um Probleme zu lösen oder Entscheidungen zu treffen, insbesondere wenn vollständige Informationen fehlen oder die Probleme zu komplex sind, um mit exakten Methoden gelöst zu werden [53]. Durch Vereinfachung oder Reduzierung des Suchraums wird nach einer Lösung der Zielfunktion gesucht. Hier wird zwischen einem globalen Maximum oder Minimum und lokalen Maximum oder Minimum unterschieden. Das globale Maximum (oder Minimum) ist der höchste (oder niedrigste) Wert der Zielfunktion im gesamten Suchraum. Lokale Maxima (oder Minima) sind dagegen Punkte, an denen die Zielfunktion höhere (oder niedrigere) Werte als in der unmittelbaren Umgebung hat, aber nicht unbedingt den höchsten (oder niedrigsten) Wert im gesamten Suchraum.

Heuristische Methoden werden zur KI gezählt, weil sie zur Lösung komplexer Probleme menschenähnliche Entscheidungsfindungs- oder Problemlösungsprozesse wie die Versuchs- und Irrtums-Strategie imitieren [47], [53]. Darüber hinaus ermöglichen diese Methoden in Situationen mit begrenzter Information, Unsicherheit oder unvollständigem Wissen intelligente Entscheidungen zu treffen [47].

Auf heuristischen Prinzipien bauen metaheuristische Algorithmen auf. Diese können aufgrund einer höheren Abstraktionsebene in verschiedenen Problemdomänen eingesetzt zu werden, ohne dass sie an die spezifischen Merkmale des jeweiligen Problems angepasst werden müssen [54]. Metaheuristiken nutzen verschiedene Suchstrategien, um den Lösungsraum zu erkunden und dabei eine gute Balance zwischen Exploration (globale Suche) und Exploitation (lokale Suche) zu finden. In den folgenden Unterkapiteln werden Metaheuristiken vorgestellt und auf ihre Einsetzbarkeit zur Suche nach optimalen Layouts diskutiert.

Genetische Algorithmen

Genetische Algorithmen (GA) zählen zu den Optimierungsverfahren und werden unter anderem aufgrund ihrer Herkunft auch evolutionäre Algorithmen genannt. Der bis heute populärste Ansatz zu GA stammt von John Holland aus dem Jahr 1975 [55]. Primär werden GA bei der Suche nach optimalen Lösungen in einem großen Suchraum angewendet. Die Technik basiert auf den Erkenntnissen der Evolutionstheorie von Charles Darwin. Aufgrund dessen wird zur Beschreibung GA in der Literatur eine biologische Terminologie verwendet. Tabelle 2 zeigt wie die verwendeten Begrifflichkeiten aus Sicht der Informatik und im Kontext des FLP verstanden werden können.

Tabelle 2: Begriffsterminologie zu genetischen Algorithmen

Biologie	**Informatik**
Population	Die Menge der aktuell betrachteten Lösungsmöglichkeiten (Menge an Layouts)
Generation	Aufeinanderfolgende Mengen von Lösungsmöglichkeiten
Individuum	Lösungsmöglichkeit
Genom	Eigenschaften der Lösungsmöglichkeit (x, y Position einer Maschine)
Eltern	Zwei Teilmengen einer zu kombinierenden Lösung (x, y Position zweier Maschinen, die kombiniert werden)
Kinder	Durch Kombination entstandene Lösungsmöglichkeiten (x, y Position einer neu positionierten / mutierten Maschine)

Jedes Individuum zeichnet sich durch sein Genom aus. Durch Selektion, Mutation oder Kombination werden neue Generationen von Individuen geschaffen. Die Individuen mit den besten Ergebnissen überleben, während die anderen aussortiert werden. Die mathematische Darstellung eines Genoms, also der Eigenschaften einer Lösungsmöglichkeit, erfolgt in Form eines Vektors oder Tupels. Darüber hinaus wird die Anwendbarkeit der Lösung durch eine sogenannte *Fitnessfunktion* beschrieben. Diese ermittelt zu einem Individuum eine reelle Zahl, die dessen Fitness beschreibt. Hierdurch wird eine Vergleichbarkeit verschiedener Individuen geschaffen. Nach Weicker wird der simulierte evolutionäre Zyklus genetischer Algorithmen durch folgende fünf Schritte realisiert [56]:

1. *Initialisierung* einer Grundpopulation durch zufällige Bestimmung der Genome.
2. *Evaluation* der Individuen durch Zuordnung einer Fitness über die Fitnessfunktion.
3. *Selektion*, die fittesten Individuen überleben, während die übrigen aussortiert werden (z. B. die besten 50 % überleben).
4. *Kombination* und *Klonen* der Lösungsmöglichkeiten, um wieder die Anzahl der Grundpopulation zu erreichen.
5. *Mutation* einer festgelegten Anzahl von Individuen, z. B. durch Umkehren bestimmter Eigenschaften. Dieser Schritt kann helfen lokale Maxima zu vermeiden.

Nach der Mutation wird wieder im Rahmen der Evaluation die Fitness der einzelnen Lösungsmöglichkeiten ermittelt. Der Ablauf zwischen Schritt $2 - 5$ wird so oft durchgeführt, bis die Fitness der Population auf einem hohen Niveau stagniert oder andere definierte Kriterien erreicht sind. Das Festlegen von Elternpaaren auf der Kombinationsebene kann entweder nach dem Kriterium der Fitness oder zufällig geschehen. Bei ersterem Vorgehen besteht das Risiko der Kombination immer gleicher Lösungen und einer Verlangsamung des Erreichens einer optimalen Lösung. Werden die Elternpaare über eine zufällige Gleichverteilung gefunden, kann dieser Umstand vermieden werden. Bei einer Kreuzung zweier

Individuen werden an einem Kreuzungspunkt die Genome der Eltern ausgetauscht. Bei mehreren Kreuzungspunkten entstehen mehrere unterschiedliche Kinder.

GA sind gut geeignet, um große und komplexe Suchräume zu erkunden und haben in einer Vielzahl von Anwendungen, einschließlich Fabriklayoutoptimierung, gute Ergebnisse erzielt [57]. Allerdings können sie aufgrund der zufälligen Natur ihrer Suchmechanismen auch rechenintensiv sein.

Ein Hauptvorteil von GA besteht darin, dass sie aufgrund ihres populationsbasierten Ansatzes in der Lage sind, den Suchraum breit zu erkunden. Anstatt sich auf eine einzelne Lösung oder einen kleinen Teil des Suchraums zu konzentrieren, arbeiten GA mit einer Population von Lösungen, die sich im Laufe der Generationen durch die Prinzipien der natürlichen Evolution (Selektion, Kreuzung und Mutation) weiterentwickeln. Dies ermöglicht ihnen in manchen Problemstellungen lokale Optima zu vermeiden und globale Optima zu finden [58]. Ein weiterer Vorteil von GA ist ihre Flexibilität in Bezug auf das Problemmodell. Sie können auf eine Vielzahl von Problemstellungen angewendet werden, einschließlich diskreter, kontinuierlicher und gemischter Variablenräume. Dies zeigt die Eignung für das FLP auf, bei dem verschiedene Arten von Entscheidungen und Beschränkungen berücksichtigt werden müssen [59].

Darüber hinaus eignen sich GA gut für die Integration von KI-basierten Techniken und hybriden Ansätzen. Beispielsweise können sie mit maschinellem Lernen kombiniert werden, um heuristische Informationen zu generieren, die den Suchprozess leiten und die Konvergenzgeschwindigkeit verbessern [60]. GA können auch mit anderen Metaheuristiken wie der Tabu-Suche oder der Ameisenkolonieoptimierung in einem hybriden Ansatz kombiniert werden, um die Stärken der verschiedenen Methoden auszunutzen und ihre Schwächen auszugleichen [43].

Tabu-Suche

Die Tabu Suche (TS) ist eine fortschrittliche Metaheuristik, die darauf abzielt, die Begrenzungen lokaler Suchmethoden zu überwinden, indem sie Gedächtnisfunktionen in den Suchprozess integriert. Diese Methode wurde erstmals von Glover im Jahr 1989 eingeführt und hat sich seither in zahlreichen Anwendungen bewährt, insbesondere in der Optimierung von Fabriklayouts [61, 62, 63].

Ein Kernpunkt der TS ist die Gestaltung einer Tabu-Liste, einschließlich ihrer Größe und der Dauer, für die bestimmte Lösungen oder Aktionen als tabu gekennzeichnet werden. Die Definition der Nachbarschaft, also jener Lösungen, die von einer aktuellen Lösung aus erreichbar sind, sowie die Festlegung von Aspirationskriterien, die es erlauben, unter bestimmten Umständen eine Tabu gesetzte Lösung dennoch zu akzeptieren, sind weitere Konfigurationselemente. Die Definition der Parameter der Tabu-Liste ermöglicht eine

Einflussnahme zur Vermeidung von lokalen Optima und fördert die Exploration weiterer Regionen im Lösungsraum. Durch das Vermeiden bereits explorierter Lösungen wird die Wahrscheinlichkeit erhöht, globale Optima zu erreichen.

Simulierte Abkühlung

Die Simulierte Abkühlung (SA) ist eine Metaheuristik, die auf der physikalischen Metapher des Abkühlungsprozesses von Materialien basiert. Sie wurde erstmals von Kirkpatrick et al. im Jahr 1984 als allgemeiner Optimierungsalgorithmus eingeführt und hat seitdem breite Anwendung in vielen Optimierungsbereichen gefunden, darunter auch die Fabriklayoutoptimierung [64].

Der Grundgedanke der SA besteht darin, dass durch das temporäre Akzeptieren von Verschlechterungen im Lösungsprozess das Feststecken in lokalen Optima vermieden wird. In den frühen Phasen des Algorithmus wird eine höhere Wahrscheinlichkeit zugelassen, schlechtere Lösungen zu akzeptieren. Dies diversifiziert die Suche und führt zu einer umfassenderen Erkundung des Suchraums. Mit fortschreitender „Abkühlung“ des Systems wird diese Wahrscheinlichkeit schrittweise reduziert, wodurch der Algorithmus zunehmend eine feinere Exploration in der Nähe einer bereits guten Lösung durchführt. Eine Herausforderung bei der Anwendung der SA ist die Festlegung der Abkühlungsparameter, darunter der Starttemperatur, der Abkühlungsplan und die Endtemperatur. Diese Parameter beeinflussen die Balance zwischen Exploration und Exploitation.

Zusammenfassend lässt sich sagen, dass die SA durch ihre Fähigkeit, sowohl breit zu suchen als auch lokal zu optimieren, ein leistungsstarkes Werkzeug in der Optimierungslandschaft darstellt. Ihre Anwendung bietet die Möglichkeit, über die Grenzen lokaler Suchmethoden hinaus globale Lösungen zu identifizieren.

Ameisenkolonieoptimierung

Die Ameisenkolonieoptimierung (engl. Ant Colony Optimization (ACO)) ist ein metaheuristischer Algorithmus, der das Suchverhalten von Ameisen in der Natur nachbildet. Diese Metaheuristik basiert auf dem Prinzip, dass Ameisen kürzeste Pfade zwischen ihrem Nest und einer Nahrungsquelle finden können, indem sie Pheromone auf ihren Wegen hinterlassen. Diese chemischen Substanzen dienen als markierende Hinweise, die von anderen Ameisen der Kolonie wahrgenommen und verstärkt werden, wenn sie denselben Weg wiederholt benutzen. Dieser Mechanismus der Pheromonverstärkung kann als positive Rückkopplung verstanden werden, die zur Identifikation der kürzesten Pfade führt.

In der Optimierung hat sich die ACO als wirkungsvoll in der Lösung von kombinatorischen Problemen erwiesen, insbesondere bei Aufgaben, die Pfadfindung und Routing betreffen, wie sie etwa in logistischen und Netzwerk-Anwendungen auftreten [65]. Trotz der

primären Ausrichtung auf solche Probleme sind Anpassungen und Modifikationen möglich, die die ACO auch für die speziellen Anforderungen der Fabriklayoutoptimierung geeignet machen [26]. Eine der zentralen Herausforderungen bei der Anwendung von ACO auf das Fabriklayoutproblem ist die Definition geeigneter Pheromon- und Heuristikinformationen, die den Suchprozess leiten.

Zusammenfassend bietet die ACO durch ihre Fähigkeit, effiziente Pfade durch systematische Erkundung zu identifizieren, eine Perspektive für die Optimierung komplexer Systeme, auch außerhalb ihrer ursprünglichen Anwendungsgebiete.

Partikel-Schwarm-Optimierung

Die Partikel-Schwarm-Optimierung (PSO) ist eine weitere metaheuristische Methode, die auf Schwarmintelligenz basiert und zur Lösung komplexer Optimierungsprobleme eingesetzt wird. Entwickelt von Kennedy und Eberhart im Jahr 1995, ist PSO von den Bewegungsmustern und Verhaltensweisen natürlicher Schwärme wie Vogelschwärmen oder Fischschwärmen inspiriert [66]. Die Kernidee von PSO besteht darin, dass eine Gruppe von Partikeln, die Lösungskandidaten repräsentieren, den Suchraum nach optimalen Lösungen durchforstet. Jedes Partikel passt seine Position und Geschwindigkeit im Suchraum an, basierend auf den erfolgreichsten Erfahrungen innerhalb des Schwarms, wodurch individuelles Lernen und Gruppendynamik in den Lösungsprozess integriert werden [67].

Die Anwendung der PSO auf das Gebiet der Fabriklayoutoptimierung wurde unter anderem von Jahantigh und Samarghandi untersucht. In ihrer Forschung haben sie das „fuzzy dynamic facility layout problem“ mithilfe einer PSO-basierten Methode angegangen, die das Problem als ein mehrstufiges stochastisches Programm modelliert, in dem Unsicherheiten in Kosten und Nachfragen durch Fuzzy-Logik integriert werden [68]. Diese Kombination aus PSO und Fuzzy-Logik ermöglicht es, dynamische Änderungen und Unsicherheiten zu berücksichtigen. Die Ergebnisse ihrer Studie zeigen, dass dieser PSO-basierte Ansatz im Vergleich zu anderen bekannten Optimierungsmethoden überlegen ist.

3.3.2 Maschinelles Lernen

Maschinelles Lernen (ML) ist ein Teilgebiet der KI, das sich auf die Entwicklung von Algorithmen konzentriert, die Zusammenhänge aus Daten lernen und ihre Leistung eigenständig verbessern können [69]. In diesem Kapitel werden die grundlegenden Konzepte, Methoden und Algorithmen von ML vorgestellt. Darüber hinaus werden verschiedene Lernansätze und Anwendungsmöglichkeiten im Bereich der Fabriklayoutoptimierung betrachtet. In diesem Zusammenhang ist insbesondere ein Einsatz von ML zur Vorhersage von Materialflüssen, zur Erkennung von Mustern in Layouts und zur Identifizierung von Optimierungsmöglichkeiten denkbar [70].

ML kann in drei Hauptkategorien eingeteilt werden: überwachtes Lernen, unüberwachtes Lernen und bestärkendes Lernen, die sich hauptsächlich in der Art der verfügbaren Daten und der Trainingsmethodik unterscheiden [71]:

- **Überwachtes Lernen (Supervised Learning (SL))**: Eingabe und Ausgabe werden dem Modell vorgegeben.
- **Unüberwachtes Lernen (Unsupervised learning)**: Nur die Eingabe wird dem Modell vorgegeben.
- **Bestärkendes Lernen (Reinforcement Learning (RL))**: Keine explizite Vorgabe von Eingabe-Ausgabe-Paaren.

Eine wichtige Rolle nehmen die verfügbaren Daten ein. Ein Datensatz ist in diesem Zusammenhang eine Sammlung von Datenpunkten, die für ein maschinelles Lernmodell verwendet werden. Datensätze bestehen aus Merkmalen (Features) und Zielvariablen (Labels). Merkmale sind die Eingangsvariablen, die das System verwendet, um Vorhersagen zu treffen, während Zielvariablen die gewünschten Ausgangswerte repräsentieren [72]. Das Bild einer Zahl ist zum Beispiel eine Eingangsvariable, während die Zahl an sich die Zielvariable ist. Datensätze werden in der Regel in Trainings-, Validierungs- und Testdaten unterteilt, um Modelle zu trainieren und ihre Leistung zu bewerten.

Die Grundlagen zu den Lernmethoden sowie weitere Mischformen werden in den folgenden Unterkapiteln erläutert und Anwendungsmöglichkeiten im Bereich der Fabriklayoutoptimierung betrachtet.

3.3.2.1 Überwachtes Lernen

Lässt sich jeder Dateneingabe eine definierte Ausgabe zuordnen, kann SL durchgeführt werden. Der Prozess der Zuordnung von Eingaben (Features) und Ausgaben (Labels) wird als *Daten-Labeling* bezeichnet. Bei diesem Ansatz werden die Modelle auf Basis von fest zugeordneten Eingabe-Ausgabe-Paaren (Features und Labels) trainiert. Ziel ist es, eine Funktion zu erlernen, die die Beziehung zwischen Eingabe- und Ausgabedaten beschreibt. Überwachte Lernalgorithmen können für Klassifikations- und Regressionsaufgaben eingesetzt werden.

Beim klassischen Programmieren gilt es, zu einer Aufgabenstellung eine Applikation zu entwickeln. Die Aufgabenstellung wird in einzelne Funktionen zerlegt, die manuell und unter Zuhilfenahme von Programmbibliotheken implementiert werden. Beim SL wird die Funktionsimplementierung von der künstlichen Intelligenz erfüllt. Einer KI werden die Eingangs- und Ausgangsdaten gezeigt, die Implementierung erfolgt selbstständig durch Anpassung auf die Daten. Am Beispiel der Konvertierung eines Farbbildes in ein schwarzweiß Bild wird dies im Folgenden veranschaulicht. In der klassischen Programmierung würde ein Entwickler folgende Funktionen implementieren:

- Einlesen der Bilddaten und Überführung in Matrix
- Methodik zur Umwandlung von Farben in das schwarz-weiße Spektrum, Normalisierung
- Speichern und Ablage des Bildes auf die Festplatte

Der zweite Punkt wird je nach Herangehensweise und Algorithmus in viele weitere Funktionen aufgeteilt. Dabei benötigt der Entwickler ein Verständnis von der jeweiligen Disziplin, hier der Bildverarbeitung. Die Bewältigung der Aufgabe unter Verwendung einer KI benötigt kein Fachwissen bezüglich der Behandlung von Bilddaten. Die KI erhält als Eingang das farbige Bild und am Ausgang das schwarz-weiße Bild. Den Weg, um aus den Eingangsdaten die trainierten Ausgangsdaten zu erzeugen, findet die KI durch den Trainingsprozess autonom. Nach Abschluss des Trainings kann ein Modell beliebige Bilder in ein schwarz-weißes Format konvertieren.

3.3.2.1.1 Regression im überwachten Lernen

In diesem Unterkapitel werden die im SL angewendeten Regressionstechniken betrachtet. Die Regression wird häufig zur Vorhersage von kontinuierlichen Werten verwendet, wie z. B. zur Schätzung von Produktionszeiten oder Materialflüssen in Fabriken.

Die lineare Regression ist eine grundlegende statistische Methode zur Modellierung der Beziehung zwischen einer abhängigen Variable y und einer oder mehreren unabhängigen Variablen $x_1, x_2, \ldots, x_n$ [73].

Bei der einfachen linearen Regression wird nur eine unabhängige Variable berücksichtigt, und die lineare Beziehung wird durch die Gleichung

$$y = \beta_0 + \beta_1 x + \epsilon \tag{10}$$

modelliert. Dabei bezeichnet β_0 den Achsenabschnitt, β_1 die Steigung, und ϵ den Fehlerterm (Residuum). Ziel ist es, die Parameter β_0 und β_1 so zu schätzen, dass die Summe der quadrierten Abweichungen (Fehler) zwischen den beobachteten Werten y_i und den vorhergesagten Werten $\hat{y}_i$ minimiert wird:

$$\text{Minimiere: } \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{11}$$

wobei $\hat{y}_i = \beta_0 + \beta_1 x_i$.

Die mehrfache lineare Regression erweitert die einfache lineare Regression auf mehrere unabhängige Variablen $x_1, x_2, \ldots, x_n$ und wird durch die Gleichung

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon \tag{12}$$

beschrieben. Die Methode der kleinsten Quadrate wird verwendet, um die Parameter $\beta_0, \beta_1, \ldots, \beta_n$ zu schätzen, indem sie die Summe der Residuenquadrate minimiert. Diese Form der Regression wird häufig in komplexen Szenarien angewendet, z. B. zur Analyse des Einflusses mehrerer Maschinenparameter auf die Produktionszeit [73].

Nichtlineare Regression wird verwendet, wenn die Beziehung zwischen der abhängigen Variablen y und den unabhängigen Variablen $x_1, x_2, \ldots, x_n$ nichtlinear ist. Ein Beispiel hierfür ist die polynomiale Regression, bei der die unabhängigen Variablen auf höhere Potenzen erhöht werden, um eine nichtlineare Beziehung zu modellieren:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d + \epsilon. \tag{13}$$

Die Wahl des Polynomgrads d beeinflusst die Balance zwischen Unteranpassung und Überanpassung. Nichtlineare Regressionstechniken sind nützlich zur Modellierung komplexer Zusammenhänge in der Fabriklayoutoptimierung.

Gauß-Markov-Annahmen

Die Gauß-Markov-Annahmen und -Tests bilden eine Gruppe statistischer Verfahren, die dazu dienen, die Grundannahmen des Gauß-Markov-Theorems in linearen Regressionsmodellen zu überprüfen. Dieses besagt, dass unter bestimmten Voraussetzungen die Methode der kleinsten Quadrate (Ordinary Least Squares (OLS)) die beste Methode zur Schätzung der Regressionskoeffizienten ist. Diese Schätzer werden als Best Linear Unbiased Estimator (BLUE) bezeichnet, wenn sie die geringste Varianz aller linearen und unverzerrten Schätzer aufweisen unter Erfüllung der Modellannahmen [74].

Die Diskussion der Gauß-Markov-Annahmen im Kontext von SL ist sinnvoll, da diese eine systematische Möglichkeit bieten, potenzielle Schwächen der linearen Regression zu identifizieren und geeignete Maßnahmen zu ergreifen, wie den Einsatz robuster Regressionstechniken oder Regularisierungsansätze (z. B. Ridge oder Lasso Regression). Die Annahmen des Gauß-Markov-Theorems und die zugehörigen Tests sind [74]:

1. **Linearität in den Parametern**: Die Beziehung zwischen der abhängigen Variable und den unabhängigen Variablen ist linear.
2. **Unabhängigkeit der Fehlerterme**: Die Beobachtungen (Datenpunkte) sind unabhängig voneinander. Das bedeutet, dass das Vorhandensein oder der Wert einer Beobachtung keinen Einfluss auf die anderen Beobachtungen hat. In der Zeitreihenanalyse kann diese Annahme verletzt sein, wenn die Beobachtungen zeitlich miteinander korreliert sind (Autokorrelation).
3. **Homoskedastizität**: Die Varianz der Fehlerterme (Störgrößen) ist für alle Werte der unabhängigen Variablen konstant. Wenn diese Annahme verletzt ist, spricht man von Heteroskedastizität.

4. **Keine perfekte Multikollinearität**: Keine unabhängige Variable ist eine exakte Linearkombination der anderen unabhängigen Variablen.
5. **Exogenität**: Die unabhängigen Variablen (Regressoren) sind exogen, d. h. sie sind unkorreliert mit den Fehlertermen. Wenn diese Annahme verletzt ist, spricht man von Endogenität, und die OLS-Schätzer sind inkonsistent.

Wenn eine oder mehrere dieser Annahmen verletzt sind, können die Ergebnisse der OLS-Schätzung verzerrt sein. Mit den Ergebnissen der Gauß-Markov-Tests werden diese Verletzungen identifiziert und geeignete Korrekturmaßnahmen zur Verbesserung der Regressionsanalyse eingeleitet [74]. Die nachfolgende Auflistung erläutert die wichtigsten Gauß-Markov-Tests:

- **t-Test und F-Test**: Diese Tests überprüfen die Signifikanz der Regressionskoeffizienten bzw. des gesamten Modells. Der t-Test wird verwendet, um die Nullhypothese zu testen, dass ein einzelner Regressionskoeffizient gleich Null ist, d. h. er hat keinen signifikanten Effekt auf die abhängige Variable. Der F-Test wird verwendet, um die Nullhypothese dahingehend zu testen, dass alle Regressionskoeffizienten gleich Null sind, d. h., das gesamte Regressionsmodell hat keine signifikante Vorhersagekraft für die abhängige Variable [74]. Die t-Tests und F-Tests sind keine direkten Tests für die Gauß-Markov-Annahmen. Sie werden verwendet, um die statistische Signifikanz der Regressionskoeffizienten und des gesamten Modells zu überprüfen.
- **Breusch-Pagan-Test, White-Test oder Goldfeld-Quandt-Test**: Diese Tests werden verwendet, um Homoskedastizität bzw. Heteroskedastizität in den Fehlertermen zu überprüfen [75, 76, 77].
- **Durbin-Watson-Test**: Dieser Test überprüft die Autokorrelation in den Fehlertermen, insbesondere die erste Ordnung der Autokorrelation [78].
- **Variance Inflation Factor (VIF)**: Dieser Test wird verwendet, um Multikollinearität in den unabhängigen Variablen zu überprüfen [79].
- **Hausman-Test**: Dieser Test wird verwendet, um die Exogenität der unabhängigen Variablen zu überprüfen, indem geschätzt wird, ob es einen signifikanten Unterschied zwischen den Koeffizienten der endogenen und exogenen Variablen gibt [80].

Eine weitere Annahme, die häufig verwendet wird, um die Schätzer der linearen Regression zu verbessern, ist die Normalverteilung der Fehler. Die Fehlerterme (Störgrößen) folgen dabei einer Normalverteilung. Diese Annahme ist insbesondere für die inferenzstatistischen Methoden wie Hypothesentests und Konfidenzintervalle von Bedeutung [72]. Wenn die Normalverteilung der Fehler nicht gegeben ist, können die inferenzstatistischen Ergebnisse ungenau ausfallen.

Die Überprüfung dieser Annahmen und das Durchführen der entsprechenden Tests stellen die Gültigkeit der Ergebnisse eines linearen Regressionsmodells sicher. Wenn die

Annahmen des Gauß-Markov-Theorems verletzt werden, können die OLS-Schätzer möglicherweise nicht die optimalen Schätzungen der Regressionskoeffizienten liefern [74]. In solchen Fällen sollten alternative Ansätze wie die Verwendung robuster Standardfehler oder generalisierter kleinster Quadrate (GLS) erwogen werden, um die Genauigkeit der Schätzungen zu verbessern [74].

Anwendung der Regression im Fabriklayout

Regressionstechniken können in verschiedenen Aspekten der Fabriklayoutoptimierung angewendet werden. Beispielsweise können sie zur Vorhersage von Materialflüssen, zur Schätzung von Produktionszeiten oder zur Analyse der Beziehung zwischen Arbeitsabläufen und Layouteffizienz eingesetzt werden. Dabei können sowohl in Abhängigkeit von der Art der Beziehungen zwischen den betrachteten Variablen lineare als auch nichtlineare Regressionstechniken verwendet werden.

3.3.2.1.2 Klassifikation im überwachten Lernen

Im Kontext von ML ist die Klassifikation eine Problemstellung, bei dem ein Modell aus einer Menge von Beispieldaten trainiert wird, die jeweils einer bestimmten Klasse zugeordnet sind [81]. Ziel ist es, eine Funktion zu lernen, die Eingabedaten auf zugehörigen korrekten Klassen abbildet. Diese Funktion wird oft als Hypothese oder Klassifikator bezeichnet. Klassifikationsalgorithmen können in parametrische und nichtparametrische Ansätze unterteilt werden. Parametrische Methoden, wie die logistische Regression, gehen davon aus, dass die Daten einer bestimmten Verteilung folgen, und versuchen, die Parameter dieser Verteilung zu schätzen. Nichtparametrische Methoden, wie Entscheidungsbäume und Support Vector Machines, machen keine Annahmen über die Verteilung der Daten und bieten in vielen Fällen eine größere Flexibilität bei der Modellierung komplexer Zusammenhänge [81].

Klassifikationsprobleme sind Bestandteil vieler Anwendungen in der Fabriklayoutoptimierung, beispielsweise bei der Vorhersage von Maschinenausfällen, der Klassifizierung von Produkten oder der Gruppierung von ähnlichen Arbeitsabläufen [82]. Hierbei spielen sowohl lineare als auch nichtlineare Klassifikationsmethoden eine wichtige Rolle [81]. In folgenden Abschnitten werden die Grundlagen und weitere Eigenschaften von Klassifikationsansätzen vorgestellt, darunter logistische Regression, Entscheidungsbäume und *Support Vector Machines*.

Logistische Regression

Die logistische Regression ist eine statistische Methode, die häufig zur Modellierung von binären Klassifikationsproblemen eingesetzt wird. Im Gegensatz zur linearen Regression, bei der eine kontinuierliche Zielvariable vorhergesagt wird, zielt die logistische Regression

darauf ab, die Wahrscheinlichkeit einer diskreten binären Zielvariable zu schätzen [83]. Die Grundidee der logistischen Regression besteht darin, eine logistische Funktion (auch Sigmoid-Funktion genannt) auf die lineare Kombination von Eingabevariablen anzuwenden, um eine Wahrscheinlichkeit zwischen 0 und 1 zu erhalten. Die Sigmoid-Funktion hat die Form:

$$P(Y = 1|X) = \frac{1}{1 + \exp(-(B_0 + B_1X_1 + B_2X_2 + \cdots + B_nX_n))} \tag{14}$$

In dieser Formel repräsentiert Y die binäre Zielvariable und X den Vektor der Eingabevariablen, wobei $P(Y = 1|X)$ die geschätzte Wahrscheinlichkeit dafür ist, dass die Zielvariable Y den Wert 1 annimmt, gegeben die Eingabevariablen $X_1, X_2, ..., X_n$. Die Regressionskoeffizienten $B_0, B_1, ..., B_n$ werden üblicherweise durch die Maximierung der logistischen Likelihood-Funktion geschätzt, einem statistischen Verfahren, das die „Passung“ des Modells an die beobachteten Daten bewertet, indem es die Wahrscheinlichkeit maximiert, dass das Modell die beobachteten Daten generiert. Sobald die Koeffizienten geschätzt sind, kann das Modell zur Vorhersage der Wahrscheinlichkeiten und zur Klassifizierung neuer Beobachtungen verwendet werden [72].

In der Fabriklayoutoptimierung kann die logistische Regression beispielsweise zur Vorhersage von Maschinenausfällen eingesetzt werden.

k-Nearest Neighbors (k-NN)

Der k-Nearest Neighbors (k-NN) Algorithmus basiert auf einer Distanzberechnung zwischen Datenpunkten [84]. Um die Klasse eines unbekannten Punkts vorherzusagen, identifiziert der k-NN Algorithmus die k nächstgelegenen Nachbarn aus dem Trainingsdatensatz und weist dem unbekannten Punkt die am häufigsten vorkommende Klasse unter diesen Nachbarn zu.

Ein Vorteil von k-NN ist seine einfache Implementierung und dass er keine spezifischen Annahmen über die Verteilung der Daten macht. Allerdings kann er bei großen Datensätzen aufgrund der hohen Rechenlast ineffizient sein und ist anfällig für den sogenannten „Fluch der Dimensionalität“, wenn die Anzahl der Merkmale hoch ist [85]. Dieser Begriff beschreibt das Phänomen, dass mit zunehmender Anzahl von Dimensionen (Merkmalen oder Variablen) in einem Datensatz die benötigte Menge an Daten exponentiell ansteigt [86].

Naive Bayes

Der Naive Bayes Klassifikator ist ein probabilistischer Algorithmus, der auf dem Bayes'schen Theorem beruht. Dieses Theorem verwendet Wahrscheinlichkeiten und Annahmen über die Unabhängigkeit der Merkmale, um Vorhersagen zu treffen. Er wird besonders häufig in textbasierten Anwendungen wie der Spam-Erkennung und der Sentiment-Analyse

eingesetzt [87].

3.3.2.1.3 Regressions- und Klassifikationsalgorithmen

In diesem Abschnitt werden weitere Algorithmen vorgestellt die sowohl für Regressions- als auch Klassifkationsprobleme angewendet werden.

Entscheidungsbäume

Entscheidungsbäume (Decision Trees (DT)) sind überwachte Lernalgorithmen die sich sowohl für Klassifikations- als auch für Regressionsaufgaben eignen. Diese grafischen Modelle bestehen aus Knoten, die Entscheidungen auf Basis der Merkmale in den Eingabedaten treffen, und Kanten, die die Entscheidungsergebnisse darstellen [88]. Die Struktur eines Entscheidungsbaums ist hierarchisch, wobei interne Knoten Merkmalsprüfungen durchführen und Blattknoten die endgültigen Entscheidungen oder Werte repräsentieren. Die Kanten, die die Knoten verbinden, repräsentieren die möglichen Ergebnisse eines Tests. Ein Pfad von der Wurzel des Baums bis zu einem Blatt repräsentiert eine Regel, die eine Entscheidung oder einen Wert auf Basis der gegebenen Merkmale trifft.

Die Konstruktion eines DT besteht im Wesentlichen aus einem rekursiven Prozess, der den Baum von oben nach unten aufbaut. Unterschieden werden im Aufbau verschiedene Algorithmen wie ID3, C4.5 und CART, die sich in der Art und Weise unterscheiden, wie sie den besten Test an jedem Knoten auswählen und wie sie mit kontinuierlichen Merkmalen, fehlenden Werten und Rauschen in den Daten umgehen. Diese Auswahl zielt auf die Maximierung der Homogenität der Knoten bzw. auf die Minimierung der Unreinheit ab [88]. Ein wesentliches Problem bei DT ist die Neigung zur Überanpassung bei tiefen Bäumen oder verrauschten Daten. Abhilfemaßnahmen umfassen das Beschneiden des Baumes („Pruning“) und den Einsatz von *Ensemble*-Methoden wie *Random Forests* oder *Boosting*, um die Vorhersagestabilität zu erhöhen [89].

DT bieten eine interpretierbare Methode zur Modellierung von Entscheidungsprozessen und können daher potenziell für das FLP eingesetzt werden. Obwohl keine spezifischen Quellen gefunden wurden, die eine direkte Anwendung von Entscheidungsbäumen auf Fabriklayoutprobleme diskutieren, sind sie in verwandten Bereichen der Produktions- und Fertigungsoptimierung verbreitet. Beispielsweise könnten Entscheidungsbäume dazu beitragen, die Zusammenhänge zwischen Produktionsparametern und der Effizienz von Fertigungsprozessen zu ermitteln, Maschinenausfälle zu klassifizieren oder Wartungsanforderungen vorherzusagen. In diesem Zusammenhang ist im Kontext dieser Arbeit folgende Untersuchung veröffentlicht worden: [3]. Eine Zusammenfassung hierzu ist in der Einleitung dieser Arbeit zu finden.

Random Forest

Der Random-Forest-Algorithmus ist ein Ensemble-Klassifikator, der auf einer Vielzahl von Entscheidungsbäumen basiert. Diese Bäume fällen unabhängig Entscheidungen, die dann über eine Mehrheitsabstimmung für Klassifikationsaufgaben oder durch Mittelwertbildung für Regressionen zusammengefasst werden. Dieser Ansatz wurde von Leo Breiman entwickelt und ist in seiner Publikation aus dem Jahr 2001 beschrieben [89]. Durch die Aggregation der Ergebnisse vieler Bäume wird die Varianz des Gesamtmodells reduziert, zugunsten einer erhöhten Vorhersagegenauigkeit. Random Forests eignen sich besonders gut für die Modellierung komplexer und nichtlinearer Zusammenhänge in den Daten, weiterhin sind weniger anfällig für Überanpassung (Overfitting[4]) als einzelne Entscheidungsbäume.

Bootstrap Aggregating

„Bootstrap Aggregating“, auch bekannt als „Bagging“, ist eine weitere Ensemble-Lernmethode, die Leo Breiman in seinem Artikel „Bagging Predictors“ von 1996 beschrieb [91]. Bagging verbessert die Stabilität und Genauigkeit von maschinellen Lernmodellen, insbesondere von Entscheidungsbäumen, durch die Erstellung und Kombination mehrerer Modelle aus Bootstrap-Proben des Trainingsdatensatzes.

Bootstrapping, die zugrunde liegende Technik des *Bagging*, involviert das wiederholte Ziehen von Stichproben aus dem ursprünglichen Datensatz mit Zurücklegen. Dabei kann es vorkommen, dass einzelne Datenpunkte mehrmals in einer Stichprobe erscheinen, während andere ausgelassen werden. Diese zufällig generierten Trainingssets, oft „Beutel“ genannt, werden dann verwendet, um unabhängige Prädiktoren zu trainieren.

Die endgültigen Vorhersagen der einzelnen Prädiktoren werden bei Klassifikationsproblemen durch Mehrheitsabstimmung und bei Regressionsproblemen durch Mittelwertbildung aggregiert. Diese Methodik führt zu einer Reduktion der Varianz und erhöht die Robustheit der Modelle im Vergleich zu einem einzigen Prädiktor, wodurch die Modelle weniger anfällig für Überanpassung sind.

XGBoost

Extreme Gradient Boosting (XGBoost) ist ein leistungsstarker Klassifikations- und Regressionsalgorithmus, der auf der Methode der Gradient Boosting Machines (GBM) basiert [92]. Im Vergleich zu herkömmlichen GBM-Modellen führt XGBoost zusätzliche Regularisierungsterme in die Kostenfunktion ein, die *Overfitting* vorbeugen. Diese Regularisierung, zusammen mit einer optimierten Implementierung von parallelen und verteilten Rechenverfahren, trägt dazu bei, die Trainingsgeschwindigkeit zu erhöhen.

[4] Overfitting wird durch eine kontinuierliche Verbesserung der Modellleistung auf den Trainingsdaten gekennzeichnet, während die Leistung auf den Validierungs- oder Testdaten abnimmt [90].

XGBoost entwickelt ein Ensemble aus schwachen Lernern, typischerweise Entscheidungsbäumen, indem es iterativ Modelle hinzufügt, die gezielt die Fehler des aktuellen Ensembles reduzieren. Dies erfolgt durch Minimierung der Verlustfunktion, wobei der Gradient dieser Funktion genutzt wird, um die effektivste Richtung für Verbesserungen zu bestimmen. XGBoost hat seine Effektivität in zahlreichen Machine-Learning-Wettbewerben und praktischen Anwendungen bewiesen, insbesondere bei der Arbeit mit strukturierten und Tabellendaten.

Support Vector Machines

Support Vector Machines (SVM) sind eine Gruppe von überwachten Lernalgorithmen, die sowohl für Klassifikations- als auch Regressionsprobleme eingesetzt werden können. Sie wurden in den 1990er-Jahren von Vapnik eingeführt und haben sich seitdem als eine wichtige Technik in der Mustererkennung und maschinellem Lernen etabliert [93].

SVM-Modelle streben danach, eine optimale Entscheidungsgrenze (auch Hyperplane genannt) zu finden, die die Klassen in einem Klassifikationsproblem trennt. Diese optimale Hyperplane maximiert den Abstand (Margin) zu den nächstgelegenen Datenpunkten jeder Klasse [94]. Bei linearen SVMs wird eine lineare Trennung der Klassen erreicht. Bei Kernel-SVMs werden die Daten durch den Einsatz von Kernel-Funktionen in einen höherdimensionalen Raum transformiert, um nichtlineare Trennungen zu ermöglichen [95]. SVM werden in verschiedenen Bereichen eingesetzt, wie der Textklassifikation, Handschrifterkennung und Bioinformatik. Aufgrund ihrer Eigenschaft, Overfitting zu widerstehen und komplexe Muster in Daten zu erkennen, zeigen sie hier gegenüber anderen Algorithmen prinzipbedingte Vorteile [96].

Im Kontext von Fabriklayoutproblemen könnten SVM dazu genutzt werden, Merkmale zu identifizieren, die mit höherer Produktivität, geringeren Kosten oder besserer Raumnutzung korrelieren. Durch Analyse historischer Daten verschiedener Fabriklayouts und ihrer Leistungskennzahlen könnte ein SVM-Modell trainiert werden. Dieses könnte dann genutzt werden, um Vorhersagen für neue Layouts zu treffen und unterstützende Entscheidungen für die Optimierung von Fabriklayouts zu bieten.

3.3.2.2 Unüberwachtes Lernen

Unüberwachtes Lernen ist ein Bereich des maschinellen Lernens, bei dem Algorithmen trainiert werden, ohne dass ihnen Zielvariablen oder Klassen im Voraus bekannt sind [72]. Ziel ist es, unbekannte Muster und Strukturen in den Daten zu entdecken, um das Verständnis über die Daten zu verbessern. Im Gegensatz zum überwachten Lernen, bei dem Algorithmen auf Basis bekannter Eingabe-Ausgabe-Paare trainiert werden, extrahieren unüberwachte Lernverfahren selbstständig Zusammenhänge und Strukturen direkt aus den

Eingabedaten [97].

Clustering

Eine der Hauptanwendungen des unüberwachten Lernens ist das Clustering. Zugehörige Algorithmen teilen Datenpunkte anhand ihrer Ähnlichkeit in Gruppen oder Cluster ein. Bekannte Clustering-Algorithmen sind k-Means, der iterativ die Clusterzentroide und die Zuordnungen der Datenpunkte optimiert [98], und Density-Based Spatial Clustering of Applications with Noise (DBSCAN), der Cluster anhand der Dichte von Datenpunkten identifiziert [99].

Dimensionsreduktion

Techniken zur Dimensionsreduktion sind ebenfalls ein wichtiger Aspekt des unüberwachten Lernens. Diese Techniken reduzieren die Anzahl der Variablen in den Daten, was die Visualisierung und Interpretation erleichtert und die Rechenlast verringert. Bekannte Methoden sind die Hauptkomponentenanalyse (Principal Component Analysis (PCA)), die die Varianz maximiert, und t-Distributed Stochastic Neighbor Embedding (t-SNE), die Visualisierungen durch Modellierung der Ähnlichkeiten zwischen Datenpunkten ermöglicht [100].

Anomalieerkennung

Ein weiterer wichtiger Bereich ist die Anomalieerkennung, die darauf abzielt, ungewöhnliche Datenpunkte oder Muster zu identifizieren. Methoden wie der Local Outlier Factor (LOF) und der Isolation Forest können zur Erkennung von Ausreißern in Daten eingesetzt werden [101, 102].

Anwendung in der Fabriklayoutoptimierung

In der Fabriklayoutoptimierung können die unüberwachten Lernalgorithmen Muster in Layouts identifizieren, die mit höherer Produktivität oder geringeren Kosten korrelieren. Clustering kann ähnliche Produktionsbereiche erkennen, während Dimensionsreduktion die Planung vereinfachen kann und Anomalieerkennung frühzeitig potenzielle Probleme in der Fertigung erkennt.

3.3.2.3 Bestärkendes Lernen

Bestärkendes Lernen (Reinforcement Learning (RL)) ist ein Bereich des maschinellen Lernens zur Entwicklung von interaktiven Agenten, die selbstständig durch Interaktion mit ihrer Umgebung optimale Aktionen zur Zielerreichung wählen. Die Basis ist ein Belohnungssystem in dem Agenten Rückmeldungen in Form von Belohnungen oder Strafen

erhalten. Hierdurch korrigieren sie ihre Aktionen [103]. Ein zentraler Ansatz im bestärkenden Lernen ist der Markov-Entscheidungsprozess (MEP), ein mathematisches Modell, das für die sequenzielle Entscheidungsfindung unter Unsicherheit entwickelt wurde. Ein MEP besteht aus einer Menge von Zuständen, Aktionen und Übergangswahrscheinlichkeiten, die beschreiben, wie Aktionen Zustände verändern. Jede Aktion in einem bestimmten Zustand führt zu einer Belohnung und beeinflusst die Wahl zukünftiger Aktionen durch den Einfluss eines Diskontierungsfaktors, der den zukünftigen Belohnungen einen geringeren Wert beimisst. Das Ziel des Agenten ist es, eine Politik zu erlernen, die den erwarteten kumulativen Gewinn über die Zeit maximiert, indem sie angibt, welche Aktionen in jedem Zustand zu wählen sind [104].

Deep Reinforcement Learning (DRL) erweitert die Konzepte des traditionellen RL durch die Integration von Deep Learning-Techniken. Diese ermöglichen es den Agenten aus unstrukturierten Eingabedaten zu lernen. Die Integration tiefer neuronaler Netzwerke erhöht die Eignung komplexe Muster und Beziehungen in Daten zu erkennen. Die hiermit entwickelten Entscheidungsstrategien werden bspw. für autonomes Fahren oder komplexe Spielumgebungen verwendet [50].

Durch die Modellierung des FLP als Markov-Entscheidungsprozess kann ein RL-Agent lernen, die Anordnung von Maschinen und Arbeitsstationen so zu optimieren, dass die Effizienz der Produktionsprozesse maximiert und die internen Transportkosten minimiert werden. Dies geschieht durch schrittweise Anpassungen und Evaluierungen des Layouts basierend auf einem systematischen Belohnungssystem, das erfolgreiche Konfigurationen belohnt und ineffiziente Layouts penalisiert. Solch ein Ansatz kann potenziell eingesetzt werden, um dynamische Anforderungen und Veränderungen in Fertigungsanlagen zu berücksichtigen, indem der Algorithmus kontinuierlich lernt und das Layout entsprechend den sich ändernden Produktionsbedingungen anpasst.

Q-Learning

Q-Learning, ein modellfreies Verfahren des bestärkenden Lernens, zielt darauf ab, durch schrittweise Approximation der optimalen Wertfunktion eine Strategie für Entscheidungsprozesse zu entwickeln. Der Agent lernt eine Funktion, die den Wert (Q-Wert) von Zustands-Aktion-Paaren schätzt, basierend auf den erhaltenen Belohnungen und den geschätzten Werten zukünftiger Zustände, wobei die Q-Werte iterativ aktualisiert werden [105].

Deep Q-Learning erweitert Q-Learning durch den Einsatz von künstlichen neuronalen Netzen zur Approximation der Q-Funktion. Diese Technik nutzt Erfahrungswiedergabe, um aus einem Speicher von Erfahrungen zu lernen, und zielgerichtete Netze, um die Lernziele zu stabilisieren [50].

In der Produktionsoptimierung, wie in der Automobilindustrie, wird bestärkendes Lernen genutzt, um Systeme dynamisch an veränderliche Bedingungen anzupassen, etwa

durch Optimierung von Materialflüssen und Minimierung von Transportkosten [106]. Diese Methoden bieten potenziell transformative Ansätze für die Fabriklayoutoptimierung, besonders in dynamischen und sich schnell ändernden Umgebungen.

3.3.2.4 Mischformen

Zudem gibt es weitere Mischformen zwischen dem überwachten und unüberwachten Lernen, wie das teilüberwachte Lernen und das selbstüberwachte Lernen (Self-Supervised Learning (SSL)).

Teilüberwachtes Lernen

Bei teilüberwachtem Lernen (Semi-supervised learning) wird ein Modell mit einer Kombination aus beschrifteten (labeled) und unbeschrifteten (unlabeled) Daten trainiert. Dabei werden die beschrifteten Daten für das überwachte Lernen verwendet, während die unbeschrifteten Daten für das unüberwachte Lernen eingesetzt werden. Teilüberwachtes Lernen wird eingesetzt, wenn beschriftete Daten teuer oder schwierig zu erhalten sind, während unbeschriftete Daten in großen Mengen verfügbar sind [107].

Selbstüberwachtes Lernen

Das selbstüberwachte Lernen (SSL) ist eine Erweiterung des unüberwachten Lernens. Während beim überwachten Lernen Datenpunkte mit zugehörigen Labels benötigt werden und beim unüberwachten Lernen Daten ohne Labels verwendet werden, generiert das selbstüberwachte Lernen seine eigenen Labels aus den Eingabedaten selbst [108]. Dieser Ansatz hat insbesondere in den letzten Jahren an Popularität gewonnen, da er das Potenzial hat, große Mengen an unbeschrifteten Daten zu nutzen, um Merkmale und Repräsentationen zu lernen. Zudem bietet es die Möglichkeit, Modelle vorzutrainieren, die anschließend mit einer kleineren Menge an gelabelten Daten feinjustiert werden können. Dies wird insbesondere in Szenarien angewandt, in denen gelabelte Daten teuer oder schwierig zu erhalten sind.

Praktische Ansätze von SSL umfassen die Erstellung von Vorhersageaufgaben basierend auf den Eingabedaten selbst. Ein Modell könnte beispielsweise darauf trainiert werden, die nächste Sequenz in einer Zeitreihe vorherzusagen oder einen fehlenden Teil eines Bildes zu rekonstruieren [109]. Ein weiterer Ansatz, der in den letzten Jahren an Popularität gewonnen hat, ist das *kontrastive Lernen*. Bei diesem Ansatz wird das Modell darauf trainiert, ähnliche Datenpunkte voneinander zu unterscheiden und unterschiedliche Datenpunkte voneinander zu trennen. Dies kann erreicht werden, indem das Modell beispielsweise dazu trainiert wird, unterschiedliche Ansichten desselben Datenpunkts voneinander zu unterscheiden [110].

Im Kontext der Fabriklayoutoptimierung könnte das selbstüberwachte Lernen dazu genutzt werden, um komplexe Zusammenhänge und Merkmale in den Daten zu lernen, die zur Verbesserung der Leistungsfähigkeit des Layouts beitragen könnten. Beispielsweise könnte ein selbstüberwachtes Modell dazu genutzt werden, um die Beziehungen zwischen verschiedenen Aspekten des Layouts und der Leistung der Fabrik zu lernen, indem es beispielsweise auf die Vorhersage zukünftiger Leistungswerte oder auf die Identifizierung ähnlicher Layouts trainiert wird.

3.3.2.5 Bewertungsmetriken bei maschinellem Lernen

Die Leistung eines maschinellen Lernalgorithmus wird von Metriken gemessen. Diese bieten ein quantitatives Maß dafür zur Messung der Aufgabenerfüllung des Modells. Im Folgenden werden die wichtigsten Bewertungsmetriken vorgestellt, die bei der Anwendung von ML in der Fabriklayoutoptimierung relevant sind.

Korrelationskoeffizient und Bestimmtheitsmaß

Der Korrelationskoeffizient R misst die Stärke und Richtung der linearen Beziehung zwischen zwei Variablen und kann Werte zwischen -1 und 1 annehmen, wobei Werte nahe ± 1 eine starke lineare Abhängigkeit und Werte nahe 0 eine schwache oder keine lineare Korrelation anzeigen. Das Bestimmtheitsmaß r^2, das den quadrierten Korrelationskoeffizienten darstellt, quantifiziert, wie viel Prozent der Varianz der abhängigen Variablen durch die unabhängigen Variablen im Modell erklärt wird. Ein r^2-Wert nahe 1 bedeutet, dass das Modell die Daten gut erklärt, während ein Wert nahe 0 darauf hinweist, dass das Modell kaum Erklärungskraft besitzt. Die Formel lautet wie folgt:

$$r^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2} = \frac{SQE}{SQT} = \frac{SQT - SQR}{SQT} = 1 - \frac{SQR}{SQT} \tag{15}$$

In dieser Formel ist:

- **Sum of Squares Explained (SQE)** oder auch als *Regression Sum of Squares* bezeichnet, ist die Summe der quadrierten Abweichungen der vorhergesagten Werte $\hat{y}_i$ vom Mittelwert der beobachteten Werte $\bar{y}$.
- **Sum of Squares Total (SQT)** ist die Summe der quadrierten Abweichungen der beobachteten Werte y_i vom ihrem Mittelwert $\bar{y}$.
- **Sum of Squares Residual (SQR)** oder auch als *Error Sum of Squares* bezeichnet, ist die Summe der quadrierten Abweichungen der beobachteten Werte y_i von den vorhergesagten Werten $\hat{y}_i$.

Es ist zu beachten, dass der r^2-Wert nicht die Art der Beziehung zwischen den Variablen angibt. Das heißt, der r^2-Wert allein liefert keine Informationen darüber, ob die Beziehung

zwischen den Variablen positiv oder negativ ist. Ein negativer r^2-Wert kann auftreten, wenn das Regressionsmodell schlechtere Vorhersagen liefert als ein Mittelwert-Schätzer (auch Baseline-Modell genannt). Diese schätzen stets den Mittelwert. Ein Baseline-Modell wird bei fehlenden oder unvollständigen Zielwerten eingesetzt. Ein r^2-Wert von 0 bildet exakt den Mittelwert ab. Ein r^2-Wert von 0.6 bedeutet, dass ein Modell in etwa 60 % der Varianz in der abhängigen Variable erklärt.

Ein wichtiger Aspekt, der beim Umgang mit r^2-Werten zu beachten ist, betrifft das Hinzufügen weiterer unabhängiger Variablen in das Modell. Theoretisch könnte das Hinzufügen von mehr Variablen den r^2-Wert erhöhen, da mehr Informationen zur Vorhersage der abhängigen Variable zur Verfügung stehen. Allerdings besteht das Risiko des Overfitting, wenn zu viele Variablen einbezogen werden. Hier kann der angepasste r^2-Wert angewendet werden. Dieser führt eine Strafe für das Hinzufügen von nicht informativen Variablen ein und passt den r^2-Wert auf der Grundlage der Anzahl der in das Modell einbezogenen Prädiktoren an.

Adjustierter r^2-Wert

Der adjustierte r^2-Wert, auch als angepasstes Bestimmtheitsmaß bekannt, modifiziert den klassischen r^2-Wert, indem er die Anzahl der Prädiktoren im Modell berücksichtigt. Dies dient dazu, eine künstliche Erhöhung des r^2-Wertes durch das einfache Hinzufügen von Variablen zu vermeiden. Der adjustierte r^2-Wert wird mit der folgenden Formel berechnet:

$$\text{adjustiertes } r^2 = 1 - \left(\frac{(1 - r^2) \cdot (n - 1)}{n - p - 1} \right) \tag{16}$$

Hierbei ist n die Anzahl der Beobachtungen, p die Anzahl der unabhängigen Variablen und r^2 das ursprüngliche Bestimmtheitsmaß.

Mittlerer absoluter Fehler

Der mittlere absolute Fehler (Mean Absolute Error (MAE)) misst den durchschnittlichen absoluten Unterschied zwischen den vorhergesagten und den tatsächlichen Werten. Er bietet eine direkte Einschätzung des Fehlers in den gleichen Einheiten wie die Zielvariable und ist robust gegenüber Ausreißern. Die Berechnung des MAE erfolgt nach folgender Formel:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{17}$$

wobei y_i der tatsächliche Wert und $\hat{y}_i$ der vom Modell vorhergesagte Wert ist. n ist die Anzahl der Beobachtungen.

Mittlerer absoluter prozentualer Fehler

Der mittlere absolute prozentuale Fehler (Mean Absolute Percentage Error (MAPE)) erweitert den MAE für Zeitreihenprognosen und andere Kontexte, in denen Fehler relativ zum tatsächlichen Wert bewertet werden. MAPE berechnet den durchschnittlichen absoluten Prozentsatzfehler jeder Vorhersage, aggregiert und normiert auf die Anzahl der Vorhersagen [111]. Diese Metrik ist vorteilhaft, um Leistungsunterschiede über verschiedene Datenmaßstäbe hinweg zu vergleichen.

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{18}$$

In dieser Formel steht y_i für den tatsächlichen Wert und $\hat{y}_i$ für den vom Modell vorhergesagten Wert, während n die Anzahl der Beobachtungen ist.

Mittlerer quadratischer Fehler

Der mittlere quadratische Fehler (Mean Squared Error (MSE)) berechnet den Durchschnitt der quadrierten Differenzen zwischen den vorhergesagten und den tatsächlichen Werten, wobei größere Fehler stärker gewichtet werden. Der MSE ist empfindlicher gegenüber Ausreißern als der MAE.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{19}$$

Wurzel des mittleren quadratischen Fehlers

Die Wurzel des mittleren quadratischen Fehlers (Root Mean Squared Error (RMSE)) berechnet den quadratischen Mittelwert der Fehler und zieht daraus die Quadratwurzel. Ähnlich dem MAE gibt der RMSE den Fehler in denselben Einheiten wie die Zielvariable an.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{20}$$

Im Vergleich zum MAE, der den durchschnittlichen absoluten Fehler quantifiziert, gewichtet der RMSE größere Fehler stärker, mit der Folge, dass Modelle mit gelegentlich sehr großen Fehlern durch den RMSE stärker „bestraft“ werden.

Die Metriken MAE, MSE und RMSE bieten unterschiedliche Einsichten in die Fehlercharakteristik eines Modells. Der MAE misst den durchschnittlichen absoluten Fehler und wird in Szenarien angewandt, bei denen eine gleichmäßige Fehlerverteilung bevorzugt wird. MSE und RMSE betonen größere Fehler durch die Quadrierung der Fehler. Dies ist in Anwendungen mit großen Abweichungen von Nutzen. Der RMSE ist dabei intuitiver, da

er die Fehlermessung in den ursprünglichen Einheiten der Daten zurückführt und somit leichter zu interpretieren ist.

Konfusionsmatrix und davon abgeleitete Metriken

Eine Konfusionsmatrix, auch bekannt als Fehlermatrix, ist ein spezielles Layout zur Visualisierung der Leistung eines Algorithmus für überwachtes Lernen, insbesondere in Klassifikationsaufgaben. Sie enthält Informationen über tatsächliche und prognostizierte Klassifizierungen [112]. Die Konfusionsmatrix ist ein zweidimensionales Array, in dem die Spalten die vorhergesagten Klassen und die Zeilen die tatsächlichen Klassen darstellen.

Die nachfolgende Tabelle 3 zeigt den Aufbau einer Konfusionsmatrix:

Tabelle 3: Konfusionsmatrix für Klassifikationsmodelle

	Vorhergesagt Positiv	**Vorhergesagt Negativ**
Tatsächlich Positiv	Wahr Positiv (TP)	Falsch Negativ (FN)
Tatsächlich Negativ	Falsch Positiv (FP)	Wahr Negativ (TN)

Die Hauptdimensionen der Konfusionsmatrix sind „Positiv“ und „Negativ“, die jeweils in „Wahr“ und „Falsch“ unterteilt sind. Dies führt zu vier Kombinationen: Wahr Positiv (TP), Wahr Negativ (TN), Falsch Positiv (FP), und Falsch Negativ (FN). Aus der Konfusionsmatrix lassen sich u. a. folgende Metriken ableiten:

- **Genauigkeit (Accuracy)**: Dies ist das Verhältnis der korrekt klassifizierten Instanzen zur Gesamtzahl der Instanzen, berechnet als $(TP + TN)/(TP + TN + FP + FN)$.
- **Präzision (Precision)**: Das ist das Verhältnis der wahr positiven Instanzen zu allen als positiv klassifizierten Instanzen, formelhaft ausgedrückt als $Precision = TP/(TP + FP)$.
- **Sensitivität / Trefferrate (Recall)**: Dies ist das Verhältnis der wahr positiven Instanzen zu allen tatsächlich positiven Instanzen, ausgedrückt durch $Recall = TP/(TP + FN)$.
- **F1-Score**: Der F1-Score ist der harmonische Mittelwert von Präzision und Sensitivität, berechnet mit der Formel $F1 = 2 \cdot (Precision \cdot Recall)/(Precision + Recall)$. Die Betrachtung des F1-Scores zeigt die Modellleistung insbesondere in Fällen unterschiedlicher Kosten für Falsch-Positive und Falsch-Negative auf.

Die Konfusionsmatrix bietet zusammenfassend eine Darstellung der Leistung eines Klassifikationsmodells, indem sie nicht nur die Fehlerarten (falsch positive und falsch negative Ergebnisse) aufzeigt, sondern auch die korrekten Vorhersagen (wahr positiv und wahr negativ) berücksichtigt.

Matthews-Korrelationskoeffizient

Der Matthews-Korrelationskoeffizient (MCC) ist speziell für die Bewertung von binären Klassifikationsmodellen geeignet und berücksichtigt unausgewogene Datenklassen. MCC misst die Qualität der Klassifikation, indem alle vier Elemente der Konfusionsmatrix einbezogen werden: wahre Positive (TP), wahre Negative (TN), falsche Positive (FP) und falsche Negative (FN), siehe nachstehende Formel:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{21}$$

Die Werte von MCC reichen von -1 (perfekte Fehlklassifikation) über 0 (zufällige Klassifikation) bis zu +1 (perfekte Klassifikation), wobei Werte nahe 0 auf Zufallsklassifikationen und Werte nahe 1 oder -1 auf hohe bzw. sehr schlechte Modellleistungen hindeuten [113].

Das folgende Beispiel verdeutlicht die Anwendung des Matthews-Korrelationskoeffizienten (MCC) in einem binären Klassifikationsproblem mit unausgewogener Klassenverteilung. Die zugehörige Konfusionsmatrix weist die folgenden Werte auf:

- Wahre Positive (TP): 50
- Wahre Negative (TN): 940
- Falsche Positive (FP): 10
- Falsche Negative (FN): 5

Nach Einsetzen der Werte ergibt sich:

$$\text{MCC} = \frac{(50 \cdot 940) - (10 \cdot 5)}{\sqrt{(50 + 10)(50 + 5)(940 + 10)(940 + 5)}} = \frac{47000 - 50}{\sqrt{60 \cdot 55 \cdot 950 \cdot 945}}$$

Die Berechnung führt zu:

$$\text{MCC} = \frac{46950}{\sqrt{2983275000}} \approx 0.86$$

Ein MCC-Wert von 0.86 deutet auf eine hohe Klassifikationsgüte hin. Dieses Ergebnis zeigt, dass das Klassifikationsmodell für die beispielhaften Werte trotz unausgewogener Verteilung der Klassen robust ist und eine zuverlässige Trennung zwischen positiven und negativen Beispielen ermöglicht.

Fläche unter der ROC-Kurve und Receiver Operating Characteristic

Die Receiver Operating Characteristic (ROC)-Kurve ist ein grafisches Werkzeug zur Bewertung der Klassifikatorleistung. Sie stellt die Sensitivität (True Positive Rate) gegenüber der 1-Spezifität (umgekehrte Spezifität - False Positive Rate) bei verschiedenen Schwellenwerten dar [114]. Die Fläche unter der ROC-Kurve wird Area Under the Receiver Operating Charac-

teristic (AUC-ROC) genannt und dient als zusammenfassendes Maß für die Gesamtgüte des Klassifikators. Ein AUC-ROC-Wert von 1 signalisiert eine perfekte Klassifikation, während ein Wert von 0.5 auf eine Leistung hinweist, die nicht besser als zufälliges Raten ist [115]. Die AUC-ROC ist besonders nützlich, da sie die Modellbewertung unabhängig von einem festen Schwellenwert ermöglicht. Die True Positive Rate (TPR) und die False Positive Rate (FPR) lassen sich wie folgt berechnen:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{22}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{23}$$

Ein praktisches Beispiel verdeutlicht die Anwendung: Angenommen, ein Klassifikationsmodell wird auf einem Datensatz mit einer unausgeglichenen Klassenverteilung getestet. Die Konfusionsmatrix liefert folgende Werte:

- Wahre Positive (TP): 50
- Wahre Negative (TN): 940
- Falsche Positive (FP): 10
- Falsche Negative (FN): 5

Auf Basis dieser Werte wird die Sensitivität (True Positive Rate) und die 1-Spezifität (False Positive Rate) folgendermaßen berechnet:

$$\text{TPR} = \frac{50}{50 + 5} = 0.909, \quad \text{FPR} = \frac{10}{10 + 940} = 0.011$$

Die berechneten Werte zeigen eine nahezu optimale Sensitivität bei gleichzeitig minimaler False Positive Rate. Dies deutet auf eine signifikante Diskriminierungskapazität des Modells hin. Gleichwohl erfordert eine präzise Bestimmung derFläche unter der ROC-Kurve (AUC-ROC) eine weiterführende Analyse, die eine numerische Integration über sämtliche Schwellenwerte und eine detaillierte Betrachtung der Modelldiskriminierung einschließt.

Logarithmischer Verlust

Der logarithmische Verlust, auch bekannt als Log Loss, ist eine Metrik für Klassifikationsmodelle, die Wahrscheinlichkeiten ausgeben. Log Loss misst die Unsicherheit eines Klassifikators durch Berechnung der durchschnittlichen Unsicherheit jeder Klassenvorhersage [116].

Cross-Entropy

Cross-Entropy ist eine Verlustfunktion, die häufig in Klassifikationsproblemen verwendet wird, insbesondere wenn das Modell die Wahrscheinlichkeit für jede Klasse vorhersagt. Sie misst die Divergenz zwischen den tatsächlichen und den vorhergesagten Wahrscheinlichkeitsverteilungen. In der Praxis wird sie häufig bei der Ausbildung von neuronalen Netzwerken und anderen Modellen, die logistische Regression verwenden, eingesetzt [117].

Gini-Koeffizient

Der Gini-Koeffizient ist eine Maßzahl zur Ungleichheit einer Verteilung, benannt nach dem italienischen Statistiker Corrado Gini. Er wird häufig zur Messung der Ungleichheit von Einkommens- oder Vermögensverteilungen verwendet. In der Welt der maschinellen Lernmodelle wird der Gini-Koeffizient jedoch häufig zur Bewertung der Leistung von Entscheidungsbäumen und zugehörigen Algorithmen verwendet, indem er die Qualität von Aufteilungen in den Daten misst [88].

Cohen's Kappa

Cohen's Kappa ist ein statistisches Maß zur Bewertung der Übereinstimmung zwischen den Vorhersagen eines Modells und den tatsächlichen Klassenlabels unter Berücksichtigung zufälliger Übereinstimmungen. Es wird häufig zur Analyse der Modellleistung bei Klassifikationsaufgaben herangezogen, insbesondere wenn unausgeglichene Klassenverteilungen vorliegen. Der Wertebereich von Kappa reicht von -1 (perfekte Nichtübereinstimmung) über 0 (zufällige Übereinstimmung) bis 1 (perfekte Übereinstimmung). Höhere Kappa-Werte signalisieren eine stärkere Übereinstimmung und damit eine bessere Modellleistung [118].

Brier Score

Der Brier Score ist ein Maß für die Genauigkeit von probabilistischen Vorhersagen und eignet sich für Modelle, die mehrere Klassen berücksichtigen. Dieser Score berechnet den quadratischen Unterschied zwischen den vorhergesagten Wahrscheinlichkeiten und den tatsächlichen Ausgängen. Die Werte des Brier Scores liegen zwischen 0 und 1, wobei 0 eine perfekte Vorhersageleistung und 1 eine vollständig inkorrekte Vorhersageleistung darstellt [119].

$$\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}(f_i - o_i)^2 \tag{24}$$

In dieser Formel steht N für die Anzahl der Beobachtungen, f_i für die vorhergesagte Wahrscheinlichkeit der i-ten Beobachtung, dass das Ereignis eintritt, und o_i für den tatsächlichen Ausgang der i-ten Beobachtung (0 oder 1).

3.3.3 Deep Learning

In diesem Kapitel erfolgt eine Vorstellung der Grundlagen des Deep Learning (DL) und dessen Anwendungsmöglichkeiten. Dazu werden die zugrunde liegenden Konzepte, Architekturen und Funktionsweisen von DL gegeben, gefolgt von einer Betrachtung der Architektur und Funktionsweise von künstlichen neuronalen Netzen (Artifical Neural Networks (ANN)) betrachtet. Darüber hinaus werden Techniken und Methoden zum Training und zur Optimierung von tiefen neuronalen Netzen (Deep Neural Networks (DNN)) erläutert. Abschließend werden mögliche Anwendungsbeispiele im Bereich der Fabriklayoutoptimierung diskutiert.

3.3.3.1 Einführung in Deep Learning

Deep Learning (DL) ist ein Teilbereich des maschinellen Lernens und basiert auf ANN mit mehreren Schichten, den sogenannten tiefen neuronalen Netzen (DNN) [120]. Der Forschungsbereich hat in den letzten Jahren aufgrund seiner Fähigkeit, komplexe Muster in großen Datenmengen zu erkennen, an Bedeutung gewonnen. Im Gegensatz zum traditionellen maschinellen Lernen, das in der Regel auf handgefertigten Merkmalen und flachen Lernalgorithmen basiert, ist DL in der Lage, hierarchische Repräsentationen von Daten durch mehrere Ebenen der Transformation und Abstraktion zu lernen [121]. Dies ermöglicht es, hochkomplexe und nichtlineare Beziehungen zwischen Eingabe- und Ausgabevariablen zu modellieren, mit verbesserten Leistungen bei komplexen Anwendungen wie in der Bild- und Spracherkennung [122, 123]. Die Entwicklung von DL geht auf die Anfänge der künstlichen Intelligenz und der künstlichen neuronalen Netze zurück. Die Einführung von Backpropagation durch Rumelhart, Hinton und Williams (1986) ermöglichte das Training von mehrschichtigen neuronalen Netzen und ebnete den Weg für moderne DL-Methoden [124]. Aufgrund von Fortschritten in der Rechenleistung, insbesondere Grafikprozessoren (Graphics Processing Unit (GPU)), und der Verfügbarkeit großer Datenmengen hat sich die Leistungsfähigkeit derartiger Modelle sprunghaft entwickelt [125].

Schmidhuber zeigt in einer Übersicht zum Einsatz von DL in neuronalen Netzwerken eine stetig steigende Anzahl der eingesetzten Methoden [125]. Generell hängen die eingesetzten DL-Strukturen stark vom Anwendungsfall ab [126].

3.3.3.2 Künstliche Neuronen und Aktivierungsfunktionen

Künstliche Neuronen, auch als Knoten oder Einheiten bezeichnet, sind die grundlegenden Bausteine von neuronalen Netzwerken. Sie ahmen die Funktionsweise von biologischen Neuronen in unserem Gehirn nach, indem sie Eingangssignale empfangen, diese verarbeiten und dann ein Ausgangssignal erzeugen. Diese Logik wurde 1943 von McCulloch und Pitts

eingeführt [127]. Ein künstliches Neuron ist eine mathematische Funktion, die eine Menge von reellen Zahlen als Input erhält, verarbeitet und eine reelle Zahl als Output ausgibt. Jeder Input geht mit einer eigenen Gewichtung in das Neuron ein. Die Summe der gewichteten Inputs wird als Netzwert bezeichnet. Zu dem Netzwert wird ein Bias[5] addiert. Diese Zahl wird von der Aktivierungsfunktion φ zum Output-Wert verarbeitet und kann abhängig von der Netztopologie unterschiedliche Typen annehmen (Schwellenwertfunktion, lineare Funktion, Sigmoidfunktion, Rectifierfunktion). Der erzeugte Output wird auch Aktivierung genannt. Neuronale Netze werden durch Anpassung dieser Gewichte und des Bias trainiert. Die Netzbildung mehrerer Neuronen kann auf unterschiedliche Weise erfolgen, z. B. ein *feed forward* neuronales Netz. Die einzelnen Neuronen werden in verschiedenen Schichten verknüpft, wie Input-Schichten, versteckten Schichten und der Output-Schicht. Jede dieser Schichten besteht aus einer definierten Anzahl von Neuronen.

Aktivierungsfunktionen beeinflussen die von einem Neuron zum nächsten weitergegebene Signalmenge und ermöglichen es Nichtlinearitäten in den Daten zu modellieren. Es gibt eine Vielzahl von Aktivierungsfunktionen, von denen jede ihre eigenen Eigenschaften und Anwendungsfälle hat. Die am häufigsten verwendeten Aktivierungsfunktionen sind die Sigmoid-, Tanh-, Rectified Linear Unit (ReLU)- und Softmax-Funktionen [117]. In vielen Fällen wird die Wahl der Aktivierungsfunktion durch empirisches Tuning und Kreuzvalidierung bestimmt. Im Folgenden werden nach Stand der Wissenschaft gängige Aktivierungsfunktionen vorgestellt [117]:

- **Identität oder lineare Funktion**: Diese Funktion ermöglicht die Modellierung eines kontinuierlichen Wertebereichs. Sie wird oft in der Ausgabeschicht von Regressionsproblemen verwendet.
- **Sigmoid- oder logistische Funktion**: Sie komprimiert die Ausgabe auf einen Bereich zwischen 0 und 1 und wird i. d. R. für binäre Klassifikationsprobleme verwendet.
- **Hyperbolischer Tangens (tanh)**: Diese Funktion gibt Werte zwischen -1 und 1 aus und zentriert die Ausgabe um 0, was in vielen Fällen vorteilhaft ist. Sie wird in den versteckten Schichten von neuronalen Netzwerken verwendet.
- **ReLU**: Diese Funktion gibt entweder 0 (für negative Eingänge) oder den Eingang selbst (für positive Eingänge) aus. Es ist die am häufigsten verwendete Aktivierungsfunktion in DL aufgrund seiner Effizienz und Effektivität [128].
- **Leaky ReLU**: Eine Variation der ReLU, die für negative Eingänge einen kleinen, von null abweichenden Ausgabewert liefert, um das Problem „toter Neuronen“ zu vermeiden, dass bei der Verwendung von ReLU auftritt [129]. Das Problem „toter Neuronen“ bezieht sich auf die Situation, in der ein neuronales Netzwerk während

[5] Der Bias in einem neuronalen Netzwerk ist ein zusätzlicher Parameter, der in die Berechnung eines Neurons einfließt, um die Flexibilität und Anpassungsfähigkeit des Modells zu erhöhen.

des Trainings anfängt, stets negative Eingaben zu erhalten und folglich seine Gewichte nicht mehr anpasst, da die Ableitung der ReLU-Funktion für negative Eingaben null ist. Die Leaky ReLU Funktion löst dieses Problem, indem sie für negative Eingaben einen kleinen, nicht-null Ausgabewert liefert, wodurch das Neuron weiterhin lernen und seine Gewichte anpassen kann.

- **Exponential Linear Units (ELU)**: ELU sind eine andere Variation der ReLU, die dazu konzipiert wurde, das Lernen in tiefen Netzwerken zu beschleunigen und die Leistung zu verbessern.
- **Parametric ReLU (PReLU)**: Dies ist eine weitere Variante der ReLU, bei der Leckfaktor für negative Eingänge ein erlernbarer Parameter ist, anstatt festgelegt zu sein. Der Leckfaktor bestimmt, wie stark negative Eingabewerte durch die Funktion weitergegeben werden, um das Problem des toten Neurons zu minimieren.
- **Softmax**: Diese Funktion wandelt einen Vektor von K reellen Zahlen in einen Vektor von K reellen Wahrscheinlichkeiten um. Sie wird oft in den Ausgangsschichten von Klassifikationsnetzwerken verwendet, um Ausgaben so zu skalieren, dass ihre Summe 1 ergibt und sie somit als Wahrscheinlichkeiten interpretiert werden können [81].
- **Swish-Funktion**: Die Swish-Funktion ist eine relativ neue Aktivierungsfunktion, die von Forschern bei Google entwickelt wurde und sich durch ihre nichtlineare, glatte Struktur auszeichnet. Sie wird definiert als $\text{Swish}(x) = x \cdot \sigma(x)$, wobei $\sigma(x)$ die Sigmoid-Funktion ist. Im Gegensatz zu ReLU ist Swish stetig differenzierbar, dies führt zu einer besseren Optimierbarkeit. Ihre besondere Eigenschaft ist, dass sie negative Eingaben nicht vollständig abschneidet, sondern sie je nach Wert der Sigmoid-Funktion abschwächt. Dies ermöglicht eine verbesserte Leistung in der Bildklassifikation oder Sprachverarbeitung [130].

Im Kontext der Fabriklayoutoptimierung können Aktivierungsfunktionen zur Modellierung komplexer Zusammenhänge zwischen den Eingabedaten (z. B. Layouteigenschaften, Produktionsdaten) und den Ausgabedaten (z. B. Leistungskennzahlen) verwendet werden. Durch die Fähigkeit, Nichtlinearitäten zu modellieren, können neuronale Netzwerke Muster in den Daten erkennen, die traditionellen statistischen Modellen möglicherweise entgehen.

3.3.3.3 Lernalgorithmen in Deep Learning

In diesem Kapitel steht die Betrachtung der zentralen Lernalgorithmen im Fokus. Diese Algorithmen kontrollieren, wie neuronale Netzwerke aus Trainingsdaten lernen und sich anpassen, um Vorhersagen zu treffen. Zunächst wird die Backpropagation erörtert, der grundlegende Algorithmus zur Berechnung von Gradienten in ANN. Dieser ermöglicht Vorhersagefehler rückwärts durch das Netzwerk zu propagieren, wodurch die Verantwortung für den Fehler auf die einzelnen Gewichte verteilt wird. Im Anschluss werden die Kosten-

oder Verlustfunktionen thematisiert, die den Gesamtfehler eines Netzwerks quantifizieren. Diese Funktionen messen den Unterschied zwischen den tatsächlichen und den vorhergesagten Ausgaben und liefern ein Maß für die Leistung des Netzwerks. Abschließend werden auf die Optimierungsalgorithmen eingegangen, die auf der Grundlage der durch die Backpropagation berechneten Gradienten und der Kostenfunktion die Gewichte des Netzwerks anpassen. Diese Algorithmen bestimmen die spezifischen Regeln zur Aktualisierung der Modellparameter, um den Gesamtverlust zu minimieren.

Backpropagation

Die Backpropagation ermöglicht die Approximation auf das zu lernende Problem im Rahmen des Trainings von ANN. Der Name „Backpropagation“ bezieht sich auf den Prozess, bei dem der Gradient der Verlustfunktion in Bezug auf die Gewichte des Netzwerks rückwärts durch das Netzwerk propagiert wird [124]. Der Algorithmus besteht im Wesentlichen aus zwei Schritten: der Vorwärtspropagierung und der Rückwärtspropagierung. Während der Vorwärtspropagierung werden die Eingaben durch das Netzwerk geführt und die Ausgabe des Netzwerks berechnet. In der Rückwärtspropagierung wird der Fehler berechnet und der Gradient des Fehlers in Bezug auf die Gewichte des Netzwerks berechnet. Diese Gradienten werden dann verwendet, um die Gewichte des Netzwerks in Richtung der negativen Gradienten zu aktualisieren, um den Fehler zu minimieren.

Kosten- oder Verlustfunktionen

Die Verlust- oder Kostenfunktion quantifiziert, wie gut oder schlecht die Vorhersagen des Modells im Vergleich zu den tatsächlichen Werten sind. Sie bildet eine Brücke zwischen dem Modell und dem Optimierungsalgorithmus, indem sie die Richtung angibt, in die die Modellparameter angepasst werden müssen, um die Vorhersagegenauigkeit zu verbessern [117]. Abhängig von der Aufgabenstellung stehen u. a. folgende Verlustfunktionen nach Stand der Wissenschaft und Technik zur Verfügung:

- **MAE, MSE, RMSE, MAPE, r^2-Wert**, **Cross-Entropy**: Siehe Kapitel 3.3.2.5.
- **Hinge-Verlustfunktion (auch bekannt als Margin-Verlust)**: Diese Verlustfunktion wird häufig in SVM, einigen Arten von neuronalen Netzwerken und „One-vs-All“ Klassifikationsschemata verwendet, insbesondere bei binären Klassifikationsaufgaben. Die Hinge-Verlustfunktion zielt darauf ab, eine große „Marge“ zwischen den Klassen zu erreichen, um eine bessere Klassentrennung zu ermöglichen [81].
- **Huber-Verlustfunktion**: Diese Funktion wird oft als Kompromiss zwischen MSE und MAE verwendet. Sie ist quadratisch für kleine und linear für große Fehler. Dies macht sie weniger anfällig für Ausreißer als die MSE, während sie die mathematischen Eigenschaften der Quadratfunktion für kleine Fehler beibehält [131].

- **Dice-Verlustfunktion**: Diese Funktion wird vor allem in der medizinischen Bildsegmentierung eingesetzt. Sie misst die Ähnlichkeit zwischen zwei Datensätzen und wird z. B. eingesetzt, wenn die Klassen unausgeglichen sind [132].
- **Focal Loss**: Diese Verlustfunktion wurde speziell für die Objekterkennung in Situationen mit unausgeglichenen Klassen entwickelt. Sie verändert die Cross-Entropy-Funktion so, dass das Modell mehr Gewicht auf schwer zu klassifizierende Beispiele legt [133].
- **Triplet Loss**: Diese Funktion wird in Aufgaben verwendet, die mit Ähnlichkeiten arbeiten, wie beispielsweise das Gesichtserkennungstraining. Sie zielt darauf ab, die Distanz zwischen ähnlichen Beispielen zu minimieren und die Distanz zwischen unähnlichen Beispielen zu maximieren [134].
- **Kullback-Leibler Divergenz (KL)**: Auch bekannt als „Relative Entropy“, misst die KL-Divergenz die Differenz zwischen zwei Wahrscheinlichkeitsverteilungen [135]. Sie wird häufig in der Schätzung von Modellparametern und in bestimmten Arten von Deep-Learning-Modellen, wie Autoencodern und Generative Adversarial Networks (GAN), verwendet.
- **Poisson Loss**: Diese Funktion wird verwendet, wenn die Zielvariablen Poisson-verteilt sind. Sie ist geeignet für Probleme, bei denen die Vorhersagen nicht negativ sein sollten, wie z. B. die Vorhersage der Anzahl von Ereignissen.
- **Cosine Proximity**: Diese Verlustfunktion misst den Kosinus des Winkels zwischen den wahren und vorhergesagten Werten. Sie wird u. a. verwendet, wenn die Richtung der Vorhersage wichtiger ist als deren Größe.

Im Kontext der Fabriklayoutoptimierung zeigen sich verschiedene Szenarien. Bei der Vorhersage von kontinuierlichen Variablen wie Durchsatz oder Zykluszeit könnte eine MSE- oder MAE-Verlustfunktion geeignet sein. Bei der Klassifikation von Layoutvarianten oder der Vorhersage von diskreten Ereignissen könnte eine Cross-Entropy- oder Hinge-Verlustfunktion verwendet werden. Die Focal Loss Verlustfunktion ist unter Umständen hilfreich, wenn bestimmte Layout-Typen selten, aber besonders wichtig oder schwierig zu klassifizieren sind. Weiterhin deutet die Betrachtung der Verlustfunktionen an, dass eine spezifische Verlustfunktion entwickelt werden könnte.

3.3.3.3.1 Optimierungsalgorithmen

Bei der Optimierung der Gewichte wird der stochastische Gradientenabstieg (Stochastic Gradient Descent (SGD)) eingesetzt, um die Gewichte inkrementell nach jedem Trainingsbeispiel oder Minibatch zu aktualisieren [136]. Es folgt eine Einführung zu den gängigen Optimierungsalgorithmen:

- **SGD**: Dies ist die am häufigsten verwendete Optimierungsmethode. SGD aktualisiert

die Gewichte des Modells nach jedem Trainingsexemplar, indem es den negativen Gradienten der Kostenfunktion entlang der aktuellen Gewichte verwendet [136].

- **Momentum**: Dies ist eine Variante von SGD, die eine „Trägheit“ hinzufügt, indem sie den vorherigen Schritt im Gewichtsraum in Betracht zieht. Dies hilft, die Oszillationen in der Optimierung zu reduzieren und beschleunigt die Konvergenz [137].
- **Nesterov Accelerated Gradient (NAG)**: Der Nesterov beschleunigte Gradient (NAG) ist eine Variante des klassischen Momentum-Verfahrens, bei dem die Gradientenberechnung nicht an der aktuellen, sondern an einer vorausberechneten Position erfolgt. Diese Vorhersage wird durch die bisherige Geschwindigkeit bestimmt. Dadurch wird das Verfahren reaktionsfähiger gegenüber Änderungen der Zielfunktion und kann schneller konvergieren. NAG wird häufig als Verbesserung zu Momentum angesehen, da es durch die Verwendung von „Lookahead“-Informationen eine Überoptimierung vermeiden kann.
- **Adagrad**: Dieser Algorithmus passt die Lernrate adaptiv für verschiedene Parameter an, indem er die quadrierten Gradienten der bisherigen Schritte speichert. Ein gängiger Anwendungsfall sind kleine Datensätze [138].
- **RMSProp**: ähnlich wie Adagrad, passt RMSProp die Lernrate adaptiv an, verwendet aber einen gleitenden Mittelwert der quadrierten Gradienten statt der kumulativen Summe, um eine zu schnelle Abnahme der Lernrate zu verhindern [139].
- **Adaptive Moment Estimation (Adam)**: Adam kombiniert die Vorteile von Momentum und RMSProp. Es speichert sowohl den gleitenden Mittelwert der vergangenen Gradienten (ähnlich wie Momentum) als auch den gleitenden Mittelwert der vergangenen quadrierten Gradienten [140].
- **Adadelta**: Eine Erweiterung von Adagrad, die versucht, das Problem der abnehmenden Lernrate zu lösen. Im Gegensatz zu Adagrad verwendet Adadelta eine gleitende Fenstertechnik, die die Anzahl der gespeicherten vergangenen Gradientenquadratwerte begrenzt [141].
- **Nadam**: Dies ist eine Kombination aus Adam und NAG. Im Gegensatz zu Adam, der den momentanen Gradienten zur Aktualisierung verwendet, verwendet Nadam den zukünftigen, approximierten Gradienten.
- **AdaMax**: Eine Erweiterung von Adam, die auf der unendlichen Norm basiert und besser für nicht konvexe Funktionen geeignet sein kann.
- **Amsgrad**: Eine Verbesserung von Adam, die zeigt, dass die Adaptivität des Algorithmus zu einem schnellen Abfall der Lernrate führen kann. Amsgrad versucht, dieses Problem durch Hinzufügen einer Maximaloperation[6] in den Aktualisierungsschritt

[6] In der Optimierung bezeichnet eine Maximaloperation die Anwendung des Maximums in den Aktualisierungsschritten eines Algorithmus. Hiermit können zu schnell abfallende Lernraten durch Verwendung des

zu lösen [142].

In der Fabriklayoutoptimierung können je nach konkretem Ziel verschiedene Optimierungsalgorithmen von Vorteil sein. SGD und seine Varianten wie Momentum können schnelle Anpassungen ermöglichen, um auf dynamische Änderungen in der Produktion zu reagieren. Algorithmen wie Adam und RMSProp, die adaptive Lernraten verwenden, sind geeignet, durch komplexe Optimierungslandschaften zu navigieren, die typisch für die räumliche Anordnung in Fabriken sind. Sie beschleunigen die Konvergenz und können Engpässe identifizieren.

Regularisierungsalgorithmen im Deep Learning

Mit der Regularisierung wird Overfitting verhindert und die Generalisierungsfähigkeit eines Modells verbessert. Dies wird durch Begrenzung der Modellkomplexität erreicht. Weiterhin wird das Modell dazu angeregt aussagekräftigere Merkmale zu erlernen. Im Folgenden werden gängige Regularisierungstechniken aufgeführt und erläutert:

- **L1-Regularisierung:** Auch als Lasso-Regularisierung bekannt, fügt diese Technik der Kostenfunktion einen Term hinzu, der proportional zum absoluten Wert der Gewichtungen ist. Eine Eigenschaft der L1-Regularisierung ist die Reduzierung einiger Gewichte auf null zur Merkmalsauswahl.
- **L2-Regularisierung:** Auch als Ridge-Regularisierung bekannt, fügt diese Technik der Kostenfunktion einen Term hinzu, der proportional zum Quadrat der Gewichtungen ist. Im Gegensatz zur L1-Regularisierung führt die L2-Regularisierung dazu, dass die Gewichtungen klein, aber nicht unbedingt null werden.
- **Dropout:** Dropout stellt eine speziell für ANN entwickelte Regularisierungstechnik dar. Während des Trainings schaltet der „Dropout“ zufällig eine bestimmte Anzahl von Neuronen in einer bestimmten Schicht ab, indem ihre Ausgaben auf null gesetzt werden. Durch dieses „Abschalten“ wird eine Abhängigkeit von bestimmten Neuronen und Overfitting verhindert.
- **Early Stopping:** Diese Regularisierungstechnik erstreckt sich auf die Leistung des Modells auf den Trainings- und Validierungsdaten. Sobald die Leistungssteigerung auf den Validierungsdaten stagniert wird das Training gestoppt. Auf diese Weise eine zu starke Anpassung an die Trainingsdaten verhindert werden [143].
- **Batch Normalization:** Eine weitere Technik, die sowohl die Modellleistung verbessern als auch als Regularisierer wirken kann. Sie wurde eingeführt, um das Problem des internen Kovarianzschiebens zu lösen, das auftritt, wenn die Verteilung der Eingaben für jede Schicht während des Trainings verändert wird. Indem die Eingaben normalisiert und die Skalierung und Verschiebung als lernbare Parameter eingeführt werden,

Maximums (berechnet zum Beispiel aus vergangenen Gradienten) vermieden werden.

ermöglicht die Batch-Normalisierung eine beschleunigte Konvergenz während des Trainings [144].

3.3.3.4 Konvergenz und Hyperparameter in neuronalen Netzwerk-Architekturen

In der kontextuellen Optimierung von ANN bezieht sich Konvergenz auf den Prozess, bei dem ein Lernalgorithmus durch iterative Anpassung der Modellparameter eine Lösung findet, die die Kostenfunktion minimiert. Konvergenz ist erreicht, wenn weitere Iterationen die Kostenfunktion nicht signifikant reduzieren. Hyperparameter wie die Wahl der Lernrate beeinflussen die Konvergenz eines Modells. Im Gegensatz zu den durch das Training dynamisch gelernten Parametern werden Hyperparameter in der Regel vor Beginn des Trainings festgelegt. Die folgenden Absätze widmen sich der Erläuterung der wichtigsten Hyperparameter im Kontext dieser Arbeit.

Anzahl der Schichten

Die Anzahl der Schichten (Layer) in einem DNN bestimmt die Tiefe des Modells. Tiefere Modelle können komplexere Funktionen repräsentieren und sind in der Lage, eine Hierarchie von Merkmalen zu erlernen, wobei jedes Layer eine immer abstraktere Darstellung der Eingabedaten liefert [121]. Mit zunehmender Tiefe des Modells sind Probleme wie verschwindende oder explodierende Gradienten zu beachten.[7] Darüber hinaus können tiefere Modelle anfälliger für Overfitting sein, da sie eine größere Anzahl von Parametern haben und daher mehr Kapazität haben, um komplexe Funktionen zu lernen.

Anzahl der Neuronen

Die Anzahl der Neuronen in einer Schicht, oft auch als Breite der Schicht bezeichnet, erlaubt eine größere Anzahl von Merkmalen zu berücksichtigen. Wie bei der Anzahl der Schichten kann eine größere Breite das Overfitting begünstigen und das Training verlangsamen [117]. Sowohl die Tiefe als auch die Breite nehmen Einfluss auf die Modellleistung. Nach dem aktuellen Verständnis geht das Hinzufügen von zusätzlichen Schichten generell mit einem größeren Leistungsgewinn einher als das Hinzufügen von zusätzlichen Neuronen [146]. Es gibt jedoch keine allgemeine Regel zur Anzahl der Neuronen oder Schichten. Die optimale Architektur hängt vom spezifischen Datensatz und der Aufgabe ab. Daher werden üblicherweise verschiedene Architekturen getestet und Parameter gewählt [147].

[7] Explodierende Gradienten beschreiben eine Situation, in der die Gradienten der Gewichte während des Trainings übermäßig groß werden. Dies destabilisiert das Lernverfahren. Verschwindende Gradienten hingegen bezeichnen gegenteilig eine Abnahme der Gradienten bis auf null. Hierdurch verbessert sich das Netzwerk kaum bis gar nicht, insbesondere in den unteren Schichten [145].

Lernrate

Die Lernrate bestimmt die Schrittgröße bei der Aktualisierung der Gewichte und hat somit einen direkten Einfluss auf die Konvergenzgeschwindigkeit und die Stabilität des Trainingsprozesses. Eine zu hohe Lernrate kann zu Instabilitäten und schlechter Konvergenz führen, während eine zu niedrige Lernrate zu langsamem Lernen und Verharren in lokalen Optima führen kann [148]. In der Praxis wird die Lernrate häufig durch „Ausprobieren“ oder durch spezielle Techniken wie Lernratenplanung oder adaptives Lernen bestimmt.[8]

Batch-Größe

Die Batch-Größe ist ein Hyperparameter, der die Anzahl der Trainingsbeispiele bestimmt, die in einer einzigen Iteration des Optimierungsprozesses verwendet werden. Größere Batches ermöglichen eine genauere Schätzung des Gradienten mit der Nebenfolge eines gestiegenen Speicherbedarfs und einer potenziell verminderten Modellqualität [149]. Ein Sonderfall ist die Batch-Größe von $[1]$, die unter dem Begriff „Online-Learning“ bekannt ist. Hierbei werden die Gewichte nach jedem Datenpunkt sofort aktualisiert, anstatt nach einer Gruppe von Datenpunkten, wie es bei Batch-Größen > 1 der Fall ist. Diese Methode ermöglicht eine beschleunigte Anpassung des Modells an neue Daten. Allerdings ist bekannt, dass das Online-Learning zu einer höheren Varianz in der Modellleistung führt. Es wurde experimentell gezeigt, dass Netzwerke bei typischen Batch-Größen im Bereich von 8-64 Datenpunkten in der Lage sind, die Leistung eines Mittelwert-Schätzers schnell zu übertreffen [150].

Anzahl der Trainingsepochen

Die Anzahl der Trainingsepochen bezeichnet vollständige Durchläufe aller Trainingsdaten durch das ANN. Je mehr Epochen durchlaufen werden, desto mehr „Erfahrungen“ sammelt das Netzwerk mit den Daten. Allerdings besteht bei zu vielen Epochen die Gefahr der Überanpassung auf irrelevante Muster, die nicht repräsentativ für die zugrunde liegenden Daten sind (Overfitting) [143]. Im Gegensatz dazu steht die Unteranpassung (Underfitting). Diese tritt auf, wenn ein Modell weder auf den Trainingsdaten noch auf den Testdaten gut abschneidet. Ursachen können die Modellarchitektur (zu wenige Schichten oder Neuronen, falsche Architektur etc.) oder eine nicht ausreichende Anzahl an Trainingsepochen sein.

[8] Diese Lernratenplanung variiert die Lernrate während des Trainingsprozesses systematisch, um anfangs schnell zu konvergieren und später die Genauigkeit zu erhöhen, indem sie die Lernrate schrittweise reduziert. Adaptives Lernen passt die Lernrate automatisch an, basierend auf den beobachteten Gradienten jedes Parameters während des Trainings. Adam und RMSprop nutzen diese Methoden.

Hyperparameter der Optimierungsalgorithmen

Die in Kapitel 3.3.3.3 aufgeführten Algorithmen weisen zum Teil spezifische Hyperparameter auf. Beispielsweise hat der SGD-Algorithmus eine eigene Lernrate, die die Anpassung der Schrittgröße bestimmt. Der Optimierungsalgorithmus Adam beinhaltet zwei Hyperparameter *beta1* und *beta2*, die den Grad der Berücksichtigung des momentanen und des quadratischen Gradients bestimmen [140].

Auswahl und Tuning von Hyperparametern

Die Leistung und Anpassungsfähigkeit eines DNN ist abhängig von der Wahl der Hyperparameter. Zur Identifikation der zu setzenden Werte wird das sogenannte „Hyperparameter-Tuning“ durchgeführt. Hiermit wird die Feinjustierung der Werte beschrieben. Neben einer experimentellen Herangehensweise ist die Grid-Suche, bei der eine Reihe vordefinierter Werte für jeden Hyperparameter getestet wird, gängig [147]. Diese Methoden führen zu guten Ergebnissen, sind aber sehr zeitaufwendig. Jüngere Entwicklungen haben zu automatisierten Methoden der Hyperparameter-Optimierung geführt, die auf komplexen Suchalgorithmen wie der Bayes'schen Optimierung, evolutionären Algorithmen oder Gradienten-basierten Methoden beruhen [151]. Insbesondere die Bayes'sche Optimierung hat sich als effektive Methode zur Hyperparameter-Optimierung etabliert [152]. Diese basiert auf dem Bayes'schen Ansatz zur Inferenz und nutzt diesen, um eine Wahrscheinlichkeitsverteilung über mögliche Zielfunktionen zu erstellen. Sie behandelt das Problem der Hyperparameter-Optimierung als Sequenzproblem, bei dem nach jedem Schritt (d. h. nach jeder Auswertung der Zielfunktion) eine aktualisierte Wahrscheinlichkeitsverteilung erstellt wird. Diese Verteilung wird dann verwendet, um den nächsten Punkt auszuwählen, der ausgewertet werden soll. Dieses Vorgehen erfordert erstens weniger Auswertungen der Zielfunktion, da zur Auswahl der nächsten Punkte Informationen aus früheren Auswertungen nutzt. Zweitens kann sie mit kontinuierlichen und kategorischen Hyperparametern umgehen. Schließlich ermöglicht sie eine natürliche Behandlung von Unsicherheit und Mehrzieloptimierung.

Die Einführung von Optimierungsmethoden wie der Bayes'schen Optimierung bringt allerdings eine neue Ebene von Parametern ein - oft als „Hyperhyperparameter“ bezeichnet. Diese sind Parameter innerhalb der Optimierungsmethode selbst, wie z. B. die Anzahl der Iterationen oder die spezifische Art der verwendeten Wahrscheinlichkeitsverteilung.

3.3.3.5 Neuronale Netze und Architekturen

In diesem Kapitel werden die grundlegenden Konzepte und Architekturen von neuronalen Netzen vorgestellt.

Vorwärtsgerichtete Netzwerke und mehrlagiges Perzeptron

Vorwärtsgerichtete Netzwerke (Feedforward Neural Networks (FNN)), synonym geläufig als mehrlagiges Perzeptron (Multilayer Perceptron (MLP)), sind die Basisform von künstlichen neuronalen Netzwerken. Die Informationen fließen linear von den Neuronen der Eingangsschicht über die Neuronen der versteckten Schicht(en) zu denen der Ausgangsschicht [117]. Die nachfolgende Abbildung 12 zeigt dies für vier Eingangsneuronen, die auf ein Ausgangsneuron reduziert werden:

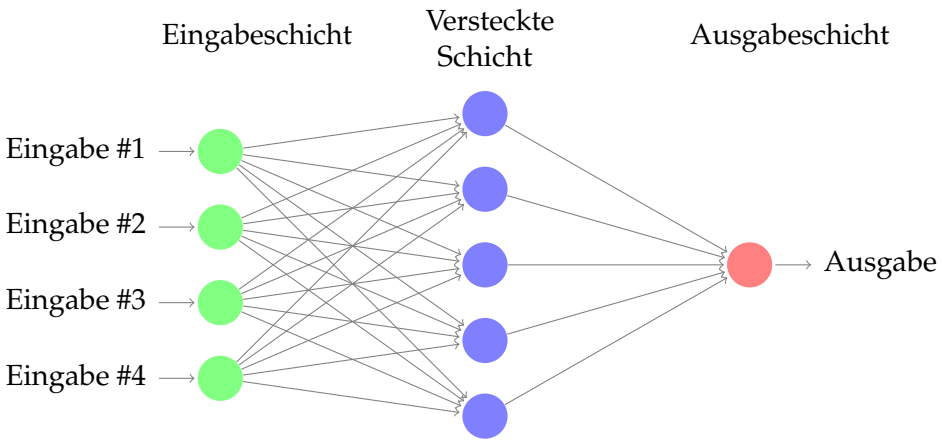


Abbildung 12: Darstellung eines MLP, die Pfeile zeigen den Datenfluss

Das Neuron bildet die grundlegende Einheit eines FNN. Die Verbindungen zwischen den Neuronen haben Gewichte, die im Laufe des Trainingsprozesses angepasst werden. Jedes Neuron empfängt die Ausgabe der vorherigen Schicht, multipliziert sie mit den entsprechenden Gewichten, summiert diese Produkte und wendet dann eine Aktivierungsfunktion auf das Ergebnis an [81].

Im Kontext der Fabriklayoutoptimierung könnten FNN zur Vorhersage des Materialdurchsatzes eingesetzt werden. Die Eingabe für das Netzwerk könnte eine Reihe von Merkmalen sein, die das Layout beschreiben (z. B. Anzahl der Maschinen, durchschnittliche Entfernung zwischen Maschinen etc.), und die Ausgabe könnte die geschätzte Produktionsleistung sein. Durch das Training auf historischen Daten könnte die Beziehung zwischen Layout-Konfigurationen und Produktionsleistung modelliert werden.

Faltende neuronale Netzwerke

Faltende neuronale Netzwerke (Convolutional Neural Network (CNN)) sind speziell für die Verarbeitung von rasterisierten Daten wie Bildern entwickelt. Sie haben ihren Namen von der mathematischen Operation „Convolution“, die in ihren Schichten verwendet wird [153]. Grundsätzlich besteht ein CNN aus drei Strukturen, von denen die ersten beiden in der

Regel wiederholt nacheinander aufeinander aufbauen. In der Faltungsschicht (1) werden kleine, rechteckige Fenster oder „Faltungskerne“ über das gesamte Eingabebild (oder die Ausgabe der vorherigen Schicht) bewegt. An jedem Ort wird eine Reihe von Berechnungen durchgeführt, die auf den Pixelwerten innerhalb des Fensters basieren. Das Ergebnis wird in einer neuen Matrix gespeichert, die als „Feature-Karte“ bezeichnet wird. Eine Pooling-Schicht (2) reduziert die räumliche Größe der Feature-Karten, um die Rechenlast zu minimieren und die Erkennung räumlicher Invarianz zu verbessern. Vollständig verbundene Schichten (3) am Ende des Netzwerks klassifizieren diese Merkmale in höhere Abstraktionsebenen.

Im Kontext der Fabriklayoutoptimierung könnten CNN verwendet werden, um Bilder oder Karten der Fabriklayouts zu analysieren. Sie könnten bspw. dazu trainiert werden, ineffiziente Layouts zu erkennen, Engpässe zu identifizieren oder sogar optimale Layouts aus einer Reihe von Alternativen vorzuschlagen.

Rekurrente neuronale Netzwerke

Rekurrente Neuronale Netzwerke (Recurrent Neural Networks (RNN)) berücksichtigen zeitliche Abhängigkeiten zwischen den Eingabedaten, die dementsprechend sequenziell vorliegen müssen. FNN gehen davon aus, dass alle Eingaben (und Ausgaben) unabhängig voneinander sind. Für Aufgaben, die zeitliche Abfolgen wie Texte oder Zeitreihen beinhalten, ist diese Annahme jedoch unzutreffend [117]. RNN verwenden „verborgene Zustände“, die Informationen aus vorhergehenden Schritten speichern und in die aktuelle Ausgabe einfließen lassen, um Muster in sequenziellen Daten zu erkennen und vorherzusagen [154]. RNN stehen jedoch vor Herausforderungen wie dem „Verschwindenden Gradienten“, der das Lernen langfristiger Abhängigkeiten erschwert, siehe Kapitel 3.3.3.3.

Long Short-Term Memory und Gated Recurrent Units

Long Short-Term Memory (LSTM)-Netzwerke sind als Weiterentwicklung von RNN zur Lösung des Problems verschwindender Gradienten konzipiert worden. Hierzu nutzen sie eine „Gate“-Struktur zur längerfristigen Speicherung von Informationen [155]. LSTM-Einheiten enthalten Eingangs-, Vergessens- und Ausgangsgates, die die Speicherung, das Vergessen und den Abruf von Informationen steuern und so langfristige Datenabhängigkeiten modellieren. Diese Fähigkeit macht LSTM zu einer häufig gewählten Methode für komplexe Sequenzaufgaben [155]. Gated Recurrent Units (GRU), eine vereinfachte Form der LSTM, bieten eine ähnliche Leistung durch eine Architektur mit nur zwei Gates: einem Zurücksetzungs- und einem Aktualisierungsgate. LSTM und GRU können wie RNN in der Fabriklayoutoptimierung zur Analyse zeitabhängiger Produktionsprozesse eingesetzt werden, um langfristige Trends und Muster zu erkennen und Optimierungsstrategien zu unterstützen.

Autoencoder

Autoencoder erlernen Codierungen in Eingabedaten, mit dem Ziel der Dimensionsreduktion oder Rauschunterdrückung [117]. Sie bestehen aus einem Encoder, der die Eingaben in eine verdichtete Repräsentation umwandelt, und einem Decoder, der versucht, aus dieser Repräsentation die ursprünglichen Daten zu rekonstruieren. Während des Trainings wird versucht, den Rekonstruktionsfehler zu minimieren. Ein charakteristisches Merkmal von Autoencodern ist der „Flaschenhals", eine Schicht mit geringerer Dimension als die Ein- und Ausgänge, der das Netzwerk zwingt, eine kompakte Repräsentation der Eingabedaten zu erlernen. Varianten von Autoencodern umfassen:

- **Sparse Autoencoder**: Erzielt die gesuchte Codierung durch Förderung der Sparsamkeit in der versteckten Schicht.
- **Denoising Autoencoder**: In der Lage Eingaben aus ihren korrupten Versionen zu rekonstruieren, insbesondere für den Umgang mit verrauschten Daten.
- **Variational Autoencoder (VAE)**: Nutzt probabilistische Methoden zur Generierung von Daten und findet Einsatz in generativen Modellen.
- **Contractive Autoencoder**: Erhöht die Robustheit der Repräsentation gegenüber Eingabeänderungen durch zusätzliche Regularisierungsterme in der Verlustfunktion.

In der Fabriklayoutoptimierung könnten Autoencoder dazu verwendet werden, wichtige Merkmale aus Layouts zu extrahieren und zu kodieren, um Layouts zu analysieren, zu vergleichen und zu optimieren. Beispielsweise könnten Sparse Autoencoder dazu beitragen, eine komprimierte Darstellung von Layoutmerkmalen zu erstellen, während Denoising Autoencoder helfen könnten, unvollständige oder verrauschte Layouts zu bereinigen. VAE könnten zur Generierung neuer Layoutvarianten genutzt werden und Contractive Autoencoder zur Erzeugung robuster Layouts.

Hopfield-Netzwerke und Boltzmann-Maschinen

Hopfield-Netzwerke dienen als assoziative Speicher zur Lösung von Optimierungsproblemen [156]. Sie bestehen aus einer vollständig vernetzten Schicht von Neuronen. Die Neuronen können zwei Zustände annehmen (-1 oder $+1$) und aktualisieren ihre Zustände basierend auf den gewichteten Eingaben von anderen Neuronen, wodurch das Netzwerk ein Energieminimum erreichen kann [157]. Dieses bezieht sich auf einen stabilen Zustand des Netzes, bei dem die Gesamtenergie minimiert wird. Dieser Zustand repräsentiert eine Lösung des Optimierungsproblems.

Boltzmann-Maschinen, entwickelt von Ackley, Hinton und Sejnowski im Jahr 1985, sind stochastische ANN, die sowohl sichtbare als auch unsichtbare Neuronen enthalten [158]. Im Unterschied zu Hopfield-Netzwerken, die deterministisch sind, aktualisieren Boltzmann-Maschinen die Zustände ihrer Neuronen stochastisch. Diese Netzwerke eignen sich zur

Lösung von Optimierungsproblemen und sind fähig, Minima ihrer Energiefunktion zu identifizieren, auch wenn sie zu lokalen Minima tendieren können [159].

Hopfield-Netzwerke könnten in der Fabriklayoutoptimierung eingesetzt werden, um optimale Anordnungen von OE zu bestimmen. Für einen Einsatz müssten die Daten zu den Layouts in binären oder diskreten Zuständen repräsentiert sein. Diese Zustände könnten etwa die An- oder Abwesenheit einer Maschine an einem bestimmten Ort im Layout repräsentieren. Die Optimierung könnte auf Ziele wie die Minimierung der internen Transportwege, Maximierung der räumlichen Effizienz oder Optimierung des Materialflusses abzielen. Ein Hopfield-Netzwerk würde in diesem Kontext genutzt werden, um durch iterative Anpassung seiner Zustände ein Energieminimum zu erreichen, das einer optimalen Anordnung entspricht. Boltzmann-Maschinen, obwohl ähnlich in der Konstruktionsweise, eignen sich weniger für solch deterministische Optimierungsaufgaben wie die Layoutplanung, da ihre stochastische Natur zu weniger vorhersagbaren Ergebnissen führen könnte.

Generative Adversarial Networks

Generative Adversarial Networks (GAN) generieren neue Daten, die denen aus einem Trainingsdatensatz ähneln [160]. Ein GAN besteht aus einem Generator, der Daten erzeugt, und einem Diskriminator, der zwischen echten und generierten Daten unterscheidet. Beide Modelle werden simultan trainiert, wobei der Generator versucht, den Diskriminator zu täuschen, und dieser wiederum versucht, sich nicht täuschen zu lassen. Dieser kompetitive Prozess verbessert kontinuierlich die Qualität der generierten Daten. GAN haben folgende zwei Herausforderungen:

1. **Modenkollaps**: Dies tritt auf, wenn der Generator beginnt, sehr ähnliche Ausgaben zu erzeugen, was zu einer mangelnden Vielfalt der Daten führt.
2. **Training Instabilität**: GAN können instabil und schwer zu trainieren sein, besonders durch die Nicht-Überlappung der Datenverteilungen von echten und generierten Daten, was die Konvergenz erschwert.

Um die Herausforderungen wie Modenkollaps und Trainingsinstabilität zu überwinden, werden Metriken zur Unterscheidung zwischen realen und generierten Datenverteilungen wie die Minimierung der Jensen-Shannon-Divergenz (JSD) verwendet:

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2} D(P \parallel M) + \frac{1}{2} D(Q \parallel M) \tag{25}$$

Die JSD ist eine Version der Kullback-Leibler-Divergenz, die zwei Wahrscheinlichkeitsverteilungen vergleicht. In der Praxis haben sich verschiedene GAN-Typen etabliert, die abhängig von der spezifischen Optimierungsfunktion angepasst sind, um ein maßgeschneidertes Verständnis für die angestrebte Anwendung zu entwickeln. Einige dieser Typen werden im Folgenden erörtert.

- **Deep Convolutional GAN (DCGAN)** Kombinieren CNN mit GAN zur Erzeugung hochwertiger generierter Bilder. Ihre Architekturprinzipien wie Batch-Normalisierung und der Verzicht auf vollständig verbundene Schichten haben die Grundlagen für spätere GAN-Forschungen gelegt [161].
- **Wasserstein GAN (WGAN)** Verbessern die Stabilität des Trainings und fördern die Erzeugung einer vielfältigeren Ausgabereihe. Sie verwenden eine alternative Verlustfunktion, die auf der Wasserstein-Distanz basiert, auch bekannt als Earth Mover's Distance [162].
- **Improved Wasserstein GAN** Eine Weiterentwicklung des Wasserstein GAN (WGAN), das eine Gradientennormbedingung einführt, um das Problem von verschwindenden Gradienten zu lösen und die Trainingsstabilität zu verbessern [163].
- **Relaxed Wasserstein GAN (RWGAN)** Verbessern die Stabilität des WGAN-Trainings durch eine alternative Lösung des Regularisierungsproblems, indem sie eine modifizierte Form der „Kantorovich-Rubinstein-Dualität" für die Wasserstein-Distanz verwenden [164].
- **Least Squares GAN (LSGAN)** Verwenden die Methode der kleinsten Quadrate mit dem Vorteil, dass dies weniger anfällig für das Problem der verschwindenden Gradienten ist und eine verbesserte Bildqualität bei der Generierung ermöglicht [165].
- **Generative Moment Matching Networks (GMMM)** Nutzen einen anderen Ansatz als traditionelle GAN, indem sie die statistischen Momente der generierten und echten Verteilungen vergleichen und anpassen [166].
- **Maximum Mean Discrepancy GAN (MMD-GAN)** Verwenden die Maximum Mean Discrepancy (MMD) Metrik zur Messung der Distanz zwischen zwei Wahrscheinlichkeitsverteilungen, zur Förderung einer stabileren Konvergenz und erhöhten Vielfalt der generierten Daten [167].
- **Cramer GAN** Verwenden die Cramer-Distanz anstelle der Wasserstein-Distanz, um die Diskrepanz zwischen den generierten und echten Datenverteilungen zu messen. Die Cramer-Distanz ist skalierbar und differenzierbar. Dies führt zu einer verbesserten Qualität der generierten Bilder [168].
- **Fisher GAN** Nutzen die sogenannte Fisher-Information. Dies ist ein statistisches Maß für die Menge an Informationen, die eine Zufallsvariable über einen unbekannten Parameter ihrer Verteilung enthält. Dies verbessert die Stabilität des Trainingsprozesses [169].
- **Energy Based GAN (EBG)** Modellieren die GAN-Struktur über ein energiebasiertes Modell, ähnlich einem Autoencoder [170].
- **Boundary Equilibrium GAN (BEGAN)** Zielen darauf ab, das Gleichgewicht zwischen dem Generator und dem Diskriminator zu erhalten, indem sie das Verhältnis ihrer

Verluste steuern [171].

- **Margin Adaption GAN (MAGAN)** Verwenden eine adaptive Margenstrategie, um die Diversität der generierten Beispiele zu erhöhen und das Problem des Modenkollaps zu bekämpfen [172].
- **Self-Attention GAN (SAGAN)** Integrieren eine „Self-Attention"-Komponente in GAN-Architekturen, um globale Abhängigkeiten in den Daten zu berücksichtigen [173].
- **BigGAN** Verwenden größere Modelle und Batch-Größen, um hochauflösende, qualitativ hochwertige Bilder zu erzeugen [174].
- **StyleGAN** Verbessern die GAN-Architektur durch Hinzufügen einer Style-Transfer-Ebene, die es ermöglicht, Stile auf unterschiedlichen Detailstufen zu lernen und anzuwenden [175].
- **Progressive Growing GAN (PROGAN)** Trainieren das Modell schrittweise, beginnend mit der Erzeugung kleiner Bilder und sukzessiver Steigerung der Auflösung [176].
- **CycleGAN** Ermöglichen eine unüberwachte Bild-zu-Bild-Übersetzung zwischen verschiedenen Domänen ohne Paare von korrespondierenden Bildern [177].

GAN und Fabriklayoutproblem

Während der Bearbeitung dieser Dissertation gab es keine Forschungsergebnisse zum Einsatz von GAN speziell zur Lösung des FLP. GAN wurden hauptsächlich in der Bild- und Datenanalyse eingesetzt, beispielsweise zur Erzeugung von realistisch wirkenden Bildern, zur Datenvermehrung oder zur Anonymisierung von Daten. Jedoch könnte das Konzept der GAN potenziell auf das Fabriklayoutproblem angewendet werden. Ein GAN-Modell könnte trainiert werden, um realistische, praktikable Fabriklayouts zu generieren, die optimierte Produktionskosten oder Effizienz aufweisen. Der Diskriminator könnte mit realen Fabriklayouts und den dazugehörigen Kosteninformationen trainiert werden. Eine weitere Anwendungsmöglichkeit ist die Nutzung zur Erzeugung von Layouts für Tests und Evaluierungen, um die Effektivität und Robustheit von Optimierungsansätzen zu verbessern.

Kohonen-Netzwerke

Kohonen-Netzwerke, auch als „Selbstorganisierende Karten" (Self Organized Maps (SOM)) bekannt, sind eine weitere Form künstlicher neuronaler Netzwerke und wurden von Teuvo Kohonen in den 1980er Jahren entwickelt [178]. Sie ermöglichen hochdimensionale Daten durch nicht überwachtes Lernen zu visualisieren und in geringere Dimensionen zu transformieren. Die Architektur basiert auf der Wettbewerbsregel, bei der Neuronen um

die Repräsentation eines Eingangsvektors konkurrieren, wobei das „Gewinnerneuron", die sogenannte Best Matching Unit (BMU), die größte Ähnlichkeit zum Eingangsvektor aufweist, gemessen durch eine Distanzfunktion wie die euklidische Distanz. SOM bewahren die topologischen Eigenschaften des Eingabedatensatzes zum effektiven Einsatz für Zwecke der Mustererkennung und explorativer Datenanalyse. Ein charakteristisches Merkmal ist die Anpassung der Gewichte des Gewinnerneurons und die der unmittelbar benachbarten Neuronen, mit dem Ergebnis einer topologischen Abbildung der Eingangsvektoren. Trotz vielfältiger Anwendungsmöglichkeiten haben Kohonen-Netzwerke auch Einschränkungen, wie Sensitivität gegenüber Anfangszuständen und die Herausforderung, komplexe resultierende Karten zu interpretieren [179].

SOM könnten in der Fabriklayoutoptimierung zur Visualisierung und Strukturierung komplexer Layouts eingesetzt werden. Im Gegensatz zu anderen DL-Konzepten bieten SOM eine intuitive Darstellung der räumlichen und funktionalen Beziehungen innerhalb des Fabriklayouts. Sie könnten zur Erkennung von Mustern und Beziehungen in Layouts, zur Optimierung von Prozessflüssen und der Minimierung von Transportwegen eingesetzt werden. Ein weiterer denkbarer Ansatz ist die Verwendung zur Gruppierung ähnlicher OE, um über verschiedene Layoutkonfigurationen eine Optimierung zu erreichen.

Adaptive Resonance Theory

Adaptive Resonance Theory (ART), entwickelt von Stephen Grossberg in den 1980er Jahren, stellt eine Familie von neuronalen Netzwerkarchitekturen dar, die auf die Überwindung des Stabilität-Plastizität-Dilemmas in Mustererkennungs- und Vorhersageaufgaben ausgerichtet sind [180]. Diese Netzwerke sind dafür bekannt, stabile kognitive Kategorien aus einer Abfolge von Eingabemustern zu lernen, wobei sie gleichzeitig die bereits erworbenen Kenntnisse bewahren und sich flexibel an neue Muster anpassen können. Diese Dynamik wird durch eine Interaktion zwischen den vergleichenden und erkennenden Schichten sowie durch einen Aufmerksamkeitsmechanismus ermöglicht, der die Reaktionsstärke des Netzwerks reguliert.

Ein typisches ART-Netzwerk umfasst eine Vergleichsschicht, die Eingangsvektoren empfängt und verarbeitet, sowie eine Erkennungsschicht, die für Klassifizierung und Lernen zuständig ist. Die Verbindung zwischen diesen Schichten wird durch adaptive Filter, die sogenannten „top-down" und „bottom-up" Gewichte, realisiert, die eine rückkopplungsbasierte Dynamik im Netzwerk fördern. Die Fähigkeit von ART, zwischen Eingangsmustern zu unterscheiden, wird durch eine „Vigilanzvariable" gesteuert, die bestimmt, wie feinstufig das Netzwerk Unterschiede erkennen soll. Diese Funktion ermöglicht es ART, zwischen zu allgemeinen und zu spezifischen Kategorisierungen zu navigieren.

ART-Modelle finden Anwendung in der Robotik, Bildverarbeitung und Anomalieerkennung. Sie sind besonders vorteilhaft in Anwendungen, in denen Echtzeitlernen und schnelle

Reaktionen ohne Vergessen älterer Informationen erforderlich sind. Eine solche Geschwindigkeit wird für das FLP nicht benötigt. Denkbar ist jedoch ein Einsatz zur Optimierung eines AGV-Systems.

3.3.4 Transfer Learning und Fine-Tuning

Transfer Learning und Fine-Tuning sind Methoden zur Steigerung der Modellleistung eines ANN. Transfer Learning nutzt ein auf einer umfangreichen Datenmenge trainiertes Modell als Grundlage für die Entwicklung eines neuen Modells. Dieses wird in der Regel auf einer kleineren und Datenmenge trainiert, um eine verwandte Aufgabe zu bewältigen [181]. Durch dieses Vorgehen wird sowohl der Bedarf an Trainingsdaten als auch die Trainingsdauer reduziert, da das Modell bereits über grundlegendes, übertragbares Wissen verfügt. Fine-Tuning passt ebenfalls ein vortrainiertes Modell weiter an eine spezifische Aufgabe durch geringfügige Modifikationen der Modellgewichte an. Dies kann die Fähigkeit zur Generalisierung erhöhen. Beide Techniken setzen eine gewisse Ähnlichkeit der Aufgabe voraus.

Bezogen auf das FLP ergibt sich mit diesen Techniken die Möglichkeit, ein ursprünglich für das SRLP trainiertes Modell durch Transfer Learning und Fine-Tuning für das DRLP und weitere Layoutklassen anwendbar zu machen.

3.3.5 Active Learning und Meta-Learning

Active Learning und *Meta-Learning* sind zwei fortgeschrittene Lernstrategien, die dazu beitragen, die Adaptivität von Modellen zu steigern, insbesondere in dynamischen und datenlimitierten Szenarien.

Active Learning ist eine Technik, die das Modell aktiv in den Prozess der Datenauswahl einbezieht, um die Lerneffizienz zu maximieren. Es wird verwendet, wenn annotierte Daten knapp oder teuer zu beschaffen sind. Das Modell wählt aktiv die informativsten Beispiele aus, die es trainieren soll, wodurch die Anzahl der benötigten Trainingsdaten reduziert wird, ohne die Modellleistung zu beeinträchtigen [182].

Meta-Learning, oft beschrieben als „lernen zu lernen“, ermöglicht es Modellen, neue Aufgaben zu erlernen, indem sie aus einer Vielzahl vorheriger Lernaufgaben generalisieren. Dieser Ansatz wird in Umgebungen eingesetzt, bei denen Modelle häufig neuen, aber ähnlichen Aufgaben gegenüberstehen und sich schnell anpassen müssen [183].

In der Fabriklayoutoptimierung könnten beide Techniken verwendet werden, um die Generalisierbarkeit von Modellen zu verbessern. Active Learning könnte zum Beispiel eingesetzt werden, um gezielt die kritischsten Aspekte eines Layouts zu identifizieren und zu optimieren. Meta-Learning könnte es Modellen ermöglichen, Erfahrungen aus

einer Reihe von Layoutoptimierungs-Projekten zu übertragen und Anpassungen für neue Fabriklayouts vorzunehmen.

3.3.6 Natural Language Processing

Natural Language Processing (NLP) ist ein interdisziplinäres Forschungsgebiet, das sich mit der Interaktion zwischen Computern und menschlicher Sprache befasst. Es umfasst Techniken, die darauf abzielen, Computern das Verstehen und Generieren menschlicher Sprache zu ermöglichen, um sinnvolle und kontextbezogene Antworten zu liefern [184]. Die Herausforderungen von NLP entstehen durch die strukturelle und semantische Komplexität der menschlichen Sprache, einschließlich ihrer Nuancen, Mehrdeutigkeiten und kulturellen Kontexte. Traditionelle Ansätze verwendeten regelbasierte Systeme und statistische Methoden, die umfangreiche Vorverarbeitung und *Feature-Engineering* erforderten. Diese Methoden waren meist nur innerhalb streng definierter Domänen erfolgreich. Die Einführung von DL-Techniken hat jedoch transformative Möglichkeiten in NLP eröffnet. DL-Netzwerke können aus unverarbeiteten Textdaten lernen und dabei sowohl strukturelle als auch semantische Muster erfassen. Sie haben sich unter anderem bei Aufgaben wie der Sentiment-Analyse, dem maschinellen Übersetzen und der Eigennamenerkennung[9] (NER) als wirksam erwiesen [117].

3.3.6.1 Transformer Architekturen

Im Bereich des NLP repräsentieren Transformer-Modelle eine neuartige Klasse von Modellen, die durch die Publikation „Attention is All You Need“ von Vaswani et al. (2017) eingeführt wurden [185]. Im Vergleich zu traditionellen rekurrenten (z. B. RNN, LSTM) und faltenden neuronalen Netzwerke (CNN) zeigen sie eine verbesserte Leistung bei der maschinellen Übersetzung und Textgenerierung. Dies ist auf die Fähigkeit zurückzuführen, langfristige Abhängigkeiten in Daten besser zu organisieren, ohne durch verschwindende oder explodierende Gradienten beeinträchtigt zu werden. Die Schlüsselkomponente der Transformer-Modelle bildet der Mechanismus der „scaled dot-product attention“, auch bekannt als „Self-Attention“ oder „Intra-Attention“. Dieser ermöglicht die Bedeutung unterschiedlicher Teile eines Textes entsprechend der aktuellen Aufgabenstellung zu gewichten. Transformer können Daten zudem parallel verarbeiten, im Gegensatz zu den sequenziellen Verarbeitungsweisen von RNN und LSTM. Zuletzt beherrschen Transformer durch die Einführung von „Positional Encoding“ die Verarbeitung der Positionsinformation von Wörtern in der Eingabesequenz.

[9] **Eigennamenerkennung (Named Entity Recognition (NER)** bezeichnet im ML die Erkennung und Klassifizierung von Namen, Personen, Organisationen, Orten und weiteren spezifischen Entitäten in unstrukturiertem Text. NER erkennt zum Beispiel in dem Satz „Angela Merkel traf Barack Obama in Berlin“, „Angela Merkel“ und „Barack Obama“ als Personen und „Berlin“ als Ort.

In den letzten Jahren wurde die Transformer-Architektur als Grundlage für eine Reihe fortschrittlicher Modelle wie *Bidirectional Encoder Representations from Transformers (BERT)* [186] und *Generative Pretrained Transformer (GPT)* [187] weiterentwickelt, die die Möglichkeiten im Bereich des maschinellen Lernens und der Sprachverarbeitung maßgeblich erweitert haben.

3.3.6.2 Large Language Models

Large Language Models (LLM) sind fortgeschrittene NLP-Modelle, die auf umfangreichen Textdaten trainiert werden, um Texte zu generieren und zu interpretieren, die menschlichem Schreiben ähneln. Diese Modelle, oft basierend auf Transformer-Architekturen, nutzen Techniken wie Transfer Learning und selbstüberwachte Lernmethoden (SSL - Beispiele sind die Vorhersage fehlende Wörter oder Sätze ohne explizite Vorgabe oder die Neuanordnung von Textteilen oder die Identifikation ähnlicher Sätze.). Ein aktuelles Beispiel für LLM ist GPT-3 von OpenAI, das mit 175 Milliarden Parametern zu den größten seiner Art gehört [188]. GPT-3 kann komplexe und kohärente Texte produzieren und ist in der Lage, Aufgaben im „Zero-Shot“ oder „Few-Shot“ Modus zu erledigen, wobei es lediglich durch minimale Anweisungen in natürlicher Sprache geführt wird. Aktuelle Forschungsfelder sind bedingt durch das „Black-Box-Problem“, die Interpretierbarkeit interner Entscheidungsprozesse, potenzielle Voreingenommenheiten in den Trainingsdaten und hohe Rechenanforderungen für das Training.

3.3.6.3 Bezug zur Fabriklayoutoptimierung

Obwohl NLP und Fabriklayoutoptimierung bisher nicht in einem gemeinsamen Zusammenhang erforscht wurden, könnten diese Modelle zur Generierung oder Interpretation natürlichsprachlicher Beschreibungen von Fabriklayouts verwendet werden. Dies könnte die Kommunikation zwischen Menschen und maschinellen Systemen im Sinne eines Human Machine Interface (HMI) verbessern. Darüber hinaus könnten die Fähigkeiten zur Mustererkennung und Sequenzmodellierung genutzt werden, um Muster in Daten zu identifizieren, die für die Optimierung von Fabriklayouts relevant sind. Zum Beispiel könnte man diese Technologien einsetzen, um ähnliche Layouts zu erkennen oder die Auswirkungen von Änderungen im Layout zu prognostizieren. Für eine genauere Analyse bedarf es weiterer Forschung, um eine potenzielle Anwendung dieser Technologien in der Fabriklayoutoptimierung zu bewerten.

3.4 Anwendung von KI in der Layoutoptimierung

Die theoretischen Grundlagen zur künstlichen Intelligenz offenbaren vielfältige Möglichkeiten zur Verbesserung von Fabriklayouts. Im Nachfolgenden erfolgt eine Darstellung

denkbarer Anwendungsbereiche:

- **Vorhersage und Modellierung von Materialflüssen**: KI-Algorithmen könnten Materialflüsse aus historischen Daten ableiten und als Prädiktor für zukünftige Materialflüsse eingesetzt werden. Dies könnte zur Identifizierung von Engpässen, Ineffizienzen und Optimierungspotenzialen beitragen. Nach dem Stand der Grundlagen eignen sich insbesondere Zeitreihenanalyseverfahren wie LSTM-Netzwerke, um komplexe Muster in historischen Daten zu erkennen und zukünftige Materialflüsse vorherzusagen.
- **Verbesserung des Prozessverständnisses**: KI-Algorithmen könnten komplexe Muster und Abhängigkeiten in Produktionsprozessen erkennen, die für menschliche Analysten schwer erkennbar sind. Die durch Clustering-Verfahren wie *k-Means* gewonnenen Erkenntnisse könnten zur Optimierung von Layouts eingesetzt werden.
- **Automatische Generierung von Layoutvarianten**: Optimierungsalgorithmen wie GA und PSO könnten eine Vielzahl verschiedener Layouts generieren und die besten Varianten basierend auf Kriterien wie Kosten, Durchlaufzeit oder Ressourcenauslastung identifizieren.
- **Bewertung und Optimierung von Layouts**: KI-Algorithmen könnten unterschiedliche Layouts bewerten und Optimierungsvorschläge basierend auf Kriterien wie Kosten und Durchlaufzeiten machen. RL könnte hierbei zur kontinuierlichen Verbesserung eingesetzt werden.
- **Anpassungsfähige Layoutplanung**: KI-Algorithmen ermöglichen die Entwicklung adaptiver Planungssysteme, die auf Veränderungen reagieren könnten (Produktionsbedingungen, Marktbedürfnisse oder Umweltauflagen). ART und SOM könnten zur Entwicklung dieser adaptiven Systeme beitragen.
- **Vorhersage von Wartungsbedarf und Ausfallrisiken**: Durch Zustandsüberwachung könnten KI-Algorithmen den Wartungsbedarf vorhersagen und zur Maximierung der Anlagenverfügbarkeit beitragen. Methoden wie SVM und *Random Forests* könnten auf Basis von Zustandsüberwachungsdaten zudem Ausfallrisiken vorhersagen.
- **Integration von menschlicher Expertise und KI**: Die Kombination aus menschlicher Expertise und KI könnte zu besseren Lösungen führen. Expertensysteme und wissensbasierte Ansätze könnten das Wissen von Experten integrieren und die Entscheidungsfindung unterstützen. Hierzu gibt es eine Reihe veröffentlichter Ansätze, in denen u. a. GA in Zusammenarbeit mit einem *Human Supervisor* interagieren, vgl. die im Zusammenhang zu dieser Dissertation entstandene Veröffentlichung von Eschemann et al. [2].

Folgende Herausforderungen bestehen bezüglich des Einsatzes von KI und seiner Anwendung in der Fabriklayoutoptimierung:

1. **Datenqualität und -verfügbarkeit**: Die Leistung von KI-Algorithmen hängt von der Qualität und Verfügbarkeit der Daten ab. Oft sind die verfügbaren Daten unvollständig, ungenau oder veraltet.
2. **Hohe Rechenanforderungen**: Insbesondere Modelle auf Basis von Deep Learning benötigen erhebliche Hardware- und Energieressourcen. Unternehmen müssen adäquate Ressourcen bereitstellen oder Zugang zu diesen sichern.
3. **Abhängigkeit von Cloud-Diensten**: Der vermehrte Einsatz von Cloud-basierten Plattformen bringt Herausforderungen bezüglich Datensicherheit, Netzwerkverbindungen und Abhängigkeit von Dienstanbietern mit sich. Darüber hinaus müssen regulatorische Fragen hinsichtlich sensibler oder geschützter Daten berücksichtigt werden.
4. **Interpretierbarkeit und Transparenz**: Die Entscheidungsfindung in vielen KI-Algorithmen, vor allem bei DL, ist oft nicht nachvollziehbar („Black-Box"-Systeme). Für die Fabriklayoutoptimierung ist es jedoch wichtig, dass Modellentscheidungen transparent und erklärbar sind.
5. **Integration in bestehende Prozesse**: Die Einführung von KI-Algorithmen in bestehende Systeme erfordert Anpassungen in Arbeitsabläufen und die Entwicklung von Schnittstellen. Dies setzt die Akzeptanz der Stakeholder voraus.
6. **Skalierbarkeit und Anpassungsfähigkeit**: KI-Algorithmen sollten flexibel genug sein, um sich in dynamisch ändernden Produktionsumgebungen und Anforderungen anzupassen. Skalierbarkeit und Flexibilität stellen die Grundlage für einen längerfristigen Einsatz in einer produktiven Umgebung dar.
7. **Ethik und Datenschutz**: Der Einsatz von KI erfordert umfangreichen Zugriff auf Daten. Dies macht Überlegungen zum Datenschutz und ethischen Aspekten notwendig. Unternehmen müssen sicherstellen, dass sie Datenschutzgesetze einhalten und sie ethische Richtlinien für den Einsatz dieser Technologien haben.
8. **Qualifikation und Weiterbildung**: Effektive Nutzung von KI-Algorithmen setzt spezialisierte Kenntnisse voraus. Unternehmen müssen in die Ausbildung und Weiterbildung ihrer Mitarbeiter investieren, um die Technologien erfolgreich zu nutzen.

Vielversprechend ist die Entwicklung von hybriden Ansätzen, die heuristischen Algorithmen, Optimierungsmethoden und menschliche Expertise kombinieren. Darüber hinaus könnten fortgeschrittene Techniken wie Transfer Learning, Active Learning und Meta-Learning dazu beitragen, die Leistungsfähigkeit, Adaptivität und Skalierbarkeit von KI-Algorithmen in der Fabriklayoutoptimierung zu verbessern. Insgesamt zeigt der Einsatz von KI das Potenzial, die Fabriklayoutoptimierung positiv zu befördern, indem es Unternehmen ermöglicht, intelligente Layoutplanungsprozesse zu verwenden, die sich an dynamisch ändernde Produktionsumgebungen anpassen können.

4 Stand der Wissenschaft und Technik

Dieses Kapitel bietet einen Überblick über den Stand der Technik in Bezug auf das FLP und den Einsatz von KI. Es werden relevante Ansätze aus der Literatur vorgestellt, beginnend mit einer Darstellung verschiedener Optimierungsansätze für Fabriklayouts, einschließlich klassischer mathematischer Modelle und heuristischer Methoden. Anschließend wird der Einsatz von KI-Technologien in der Fabriklayoutoptimierung behandelt. Es werden Studien präsentiert, die zeigen wie derartige Netzwerke zur Optimierung von Fabriklayouts genutzt werden. Das Kapitel schließt mit einer Diskussion zum aktuellen Forschungsstand. Offene Forschungsfragen für den Einsatz von KI-Algorithmen in der Fabriklayoutoptimierung werden beleuchtet, um den Forschungsansatz dieser Arbeit für die nachfolgenden Kapitel herauszuarbeiten.

4.1 Wissenschaftliche Lösungsansätze zum FLP

Für die Darstellung des Forschungsstands im Bereich des FLP mit Bezug zur KI wurden die Datenbanken *ResearchGate* und *Google Scholar* durchsucht. Dabei wurden sowohl spezifische als auch kombinierte Suchbegriffe in verschiedenen Schreibweisen in überwiegend englischer Sprache verwendet:

- FLP und KI
- Facility layout problem
- Layout planning
- Facility layout problem and machine learning
- Facility layout problem and deep learning

Es sei darauf hingewiesen, dass Variationen der Begriffe wie „Fabrik-Layout“ ebenfalls berücksichtigt wurden. Die Recherche ergab eine Dominanz von Publikationen aus den Ingenieurwissenschaften und der Informatik. Arbeiten aus den Wirtschafts- und Sozialwissenschaften wurden für diese Analyse nicht berücksichtigt. Das durchgeführte Vorgehen basiert auf den Empfehlungen von Ball und Tunger und Brocke et al. zur wissenschaftlichen Literaturrecherche und deren Transparenz durch Dokumentation [189, 190]. Weiterhin wurden themenrelevante Arbeiten berücksichtigt, auf die in den gesichteten Artikeln verwiesen wurde, die aber nicht über die initiale Recherche gefunden wurden. Der durch das Vorgehen erarbeitete wissenschaftliche Forschungsstand wird in den folgenden Unterkapiteln zusammengefasst.

4.2 Analyse wissenschaftlicher Publikationen

Die durch die initiale Recherche identifizierten Publikationen wurden ergänzt und verglichen mit den Erkenntnissen aus der aktuellen Studie von Burggraef et al., die den Einsatz von KI im Kontext des FLP untersuchten [191]. In dieser Studie wurden 1.290 Artikel aus einer ursprünglichen Menge von 11.851 Artikeln gesichtet, die aus neun verschiedenen Datenbanken stammten, ergänzt durch 134 Artikel aus einer Schneeballstichprobe. Von diesen wurden 22 Artikel identifiziert, die KI-Techniken zur Lösung von FLP diskutierten und somit die Einschlusskriterien für Relevanz im Bereich FLP und den Einsatz von KI-Lösungen erfüllten. Von den 22 relevanten Publikationen setzten neun auf *Supervised Learning*, elf auf *Unsupervised Learning* und zwei auf *Reinforcement Learning* als Haupttechniken.

Des Weiteren zeigt die historische Betrachtung der KI-Forschung die drei „Winter"-Phasen, in denen die Finanzierung und das Forschungsinteresse zu KI eingebrochen war [47]. Der erste begann in den späten 1960er und frühen 1970er-Jahren, als die ersten überhöhten Erwartungen an die KI nicht erfüllt wurden und die Finanzierung durch das US-Verteidigungsministerium zurückgefahren wurde. Ein weiterer KI-Winter folgte in den späten 1980er-Jahren mit einem Rückgang des Interesses an Expertensystemen. Ein dritter ereignete sich in den 1990er-Jahren mit einem nachlassenden Interesse an ANN. Diese Phasen endeten jeweils mit der Einführung neuer ML-Techniken und dem Beginn der „Big Data"-Ära Ende der 1990er und Anfang der 2000er-Jahre [47]. Nachstehend eine zeitliche Darstellung zur Verteilung der Veröffentlichungen mit Beginn von 1985 bis 2020, s. Abbildung 13.

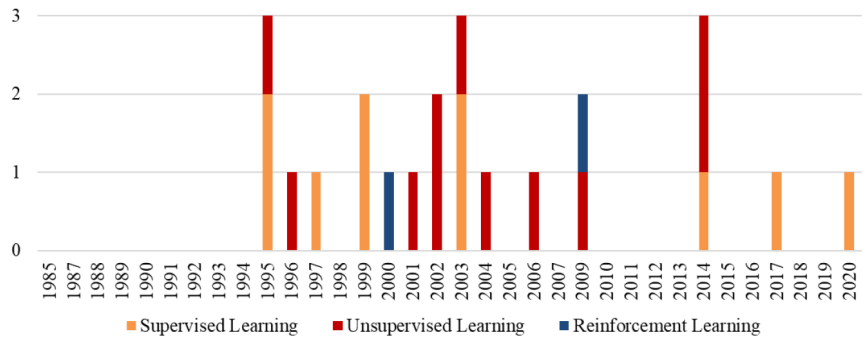


Abbildung 13: Zeitreihenanalyse zu KI-Veröffentlichungen mit Bezug zum FLP [191]

Mit dem Ende des letzten Forschungswinters erscheinen die ersten Veröffentlichungen von KI-Ansätzen mit Bezug zum Fabriklayoutproblem.

4.3 KI-basierte Veröffentlichungen zur Lösung des FLP

In diesem Abschnitt werden die Veröffentlichungen vorgestellt, die in der zuvor durchgeführten systematischen Literaturrecherche als relevant identifiziert wurden, da sie einen ausgeprägten Fokus auf die Anwendung von KI haben. Die Publikationen sind entsprechend ihrem Hauptfokus nach in drei Kategorien - metaheuristisch, ML-basiert und DL-basiert geordnet.

4.3.1 Veröffentlichungen zu metaheuristischen Algorithmen zur Lösung des FLP

Abbildung 14 zeigt das Ergebnis einer Untersuchung zur Anwendung metaheuristischen Algorithmen mit Bezug zum FLP.

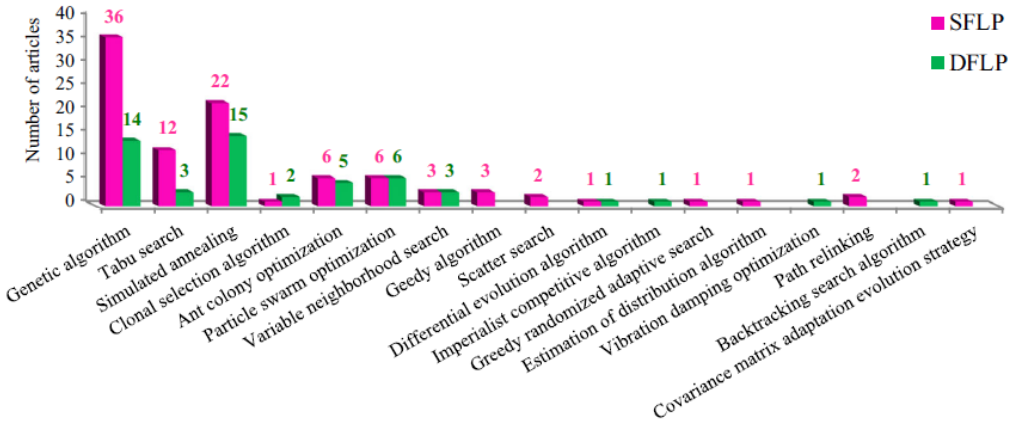


Abbildung 14: Anzahl Veröffentlichungen zu metaheuristischen Algorithmen [24]

Die meisten Ansätze zur Fabriklayoutoptimierung konzentrieren sich auf den Einsatz von GA, gefolgt von TS und SA. Dies gilt sowohl für statische (SFLP), als auch für dynamische Anwendungsfälle (DFLP). Grundsätzlich zeigt sich allerdings, dass in diesem Forschungsfeld die gesamte Bandbreite metaheuristischer Algorithmen zum Einsatz kommt.

Conway et al. stellten 1994 einen ersten Ansatz zur Lösung des DFLP auf der Grundlage genetischer Algorithmen vor [192]. Das Layout wird als eine Zeichenfolge (oder ein „Chromosom“) modelliert, wobei jedes Element der Zeichenfolge die Anordnung einer OE in einem bestimmten Zeitraum darstellt. Ein zufälliger „Splicing“-Punkt wird gewählt, um Teile von zwei Zeichenfolgen miteinander zur Erzeugung neuer Varianten zu kreuzen. Hiermit gelang die Berücksichtigung von mehr Randbedingungen als in vorherigen Arbeiten. Die Validität vom GA wird an zwei Problemstellungen experimentell veranschaulicht.

Dorigo und Di Caro (1999) verwendeten einen kombinierten GA- und ACO-Ansatz zur Lösung des FLP [193]. Ihr Ansatz konnte verbesserte Lösungen generieren, erforderte

jedoch eine erhebliche Anzahl von Iterationen und könnte daher in realistischen Szenarien mit begrenzten Rechenressourcen weniger effektiv sein.

Balakrishnan und Cheng (2000) entwickelten einen verschachtelten Schleifen-GA mit einem speziellen Punkt-zu-Punkt-Kreuzungsoperator, der zur Erweiterung des Suchraums eingesetzt wird [194]. Im Gegensatz zu Conway et al., wo Kreuzungen zufällig an einem Punkt der Zeichenfolge durchgeführt werden, verwendet der Punkt-zu-Punkt-Kreuzungsoperator feste Punkte für die Kombination der elterlichen Gene, zur Ermöglichung einer gezielteren Suche. Die innere Schleife des GA nutzt diesen festen Crossover-Operator, während die äußere Schleife regelmäßig eine Anzahl von „schlechten" Elternteilen in der Population ersetzt, um durch Mutationen und den Generationswechsel die Diversität zu fördern.

Ein weiterer Ansatz mit Fokus auf das DFLP wurde von Azadivar und Wang (2000) vorgestellt. Sie integrieren qualitative und strukturelle Entscheidungsvariablen in die Fabriklayoutoptimierung unter Verwendung von GA [62]. Der Ansatz nutzt eine Zerlegung des Layouts in rechteckige Bereiche, bekannt als „Slicing". Diese Methode kombiniert GA, Computersimulation und einen automatisierten Generator für Simulationsmodelle. Jedes Layout wird für den GA als Zeichenkette modelliert und dann von einem Simulationsmodellgenerator in ein Simulationsmodell übersetzt. Genetische Algorithmen optimieren das Layout, während die Simulation zur Bewertung der Lösungen dient. Laut den Autoren überwindet dieser Ansatz die Grenzen traditioneller Methoden der Layoutoptimierung und ermöglicht es, nahezu optimale Lösungen zu erzielen.

Ficko et al. lösten im Jahr 2004 einen mehrreihiges, kontinuierliches MRLP in einem „flexiblen Fertigungssystem" (DFLP) durch Verwendung von GA [195]. Die Fabrik bestand aus 14 Einheiten mit 8.7×10^{10} möglichen Anordnungen. Im Ergebnis wurde ein scheinbar optimales Layout nach zehn Generationen und der Bewertung von insgesamt 30.000 Konfigurationen gefunden. Die optimale Lösung weichte nach mehreren Durchläufen voneinander ab, sodass die Autoren schlussfolgern, dass bestimmte Bereiche eines Layouts flexibel gestaltet werden können.

Hu et al. (2007) entwickelten einen GA zur Optimierung von Fabriklayouts, die durch vorgegebene rechteckige Zellen mit festen Formen und Platzierungen charakterisiert sind [196]. Diese Zellen repräsentieren verschiedene Produktionsbereiche mit definierten Liefer- und Abholpunkten, die flexibel an den Zellgrenzen positioniert werden können. Zur Bewertung der Layouts nutzten die Autoren das „Perimeter-Distanz-Maß", das sich von den in verwandten Studien häufig verwendeten „Zentroiden-zu-Zentroiden-Maß" unterscheidet.[10] Dies verbessert theoretisch die praktische Relevanz ihres Ansatzes. Der Algorithmus wurde

[10] Das „Perimeter-Distanz-Maß" bewertet die Layouteffizienz basierend auf der Gesamtlänge der Außenlinien der Zellen, was die Zugänglichkeit und externe Verbindung zwischen den Zellen hervorhebt. Das „Zentroiden-zu-Zentroiden-Maß" hingegen berechnet die mittleren Distanzen zwischen den Zellmittelpunkten, wodurch die interne Nähe und direkte Wege zwischen den Zellen betont werden.

mit Testproblemen aus der Literatur überprüft, die zwischen $7 - 20$ OE umfassten. Nach dem Resümee der Autoren konnte unter erhöhtem Rechenaufwand bessere Ergebnisse erzielt werden als in zeitgemäßen vergleichbaren Studien.

Wu und Ji (2007) entwickelten einen modifizierten GA für das QAP und erweiterten die Selektionsstrategie um zwei Richtlinien - „replace-parent“ und „replace-worst“ [197]. Die „replace-parent“-Strategie ersetzt die Elternindividuen durch ihre Nachkommen, sofern diese eine bessere Fitness aufweisen. Dies geschieht in jeder neuen Generation. Die „replace-worst“-Strategie hingegen wird periodisch angewendet und ersetzt die schlechtesten Individuen der Population durch bessere zur Überwindung lokaler Optima und Erhaltung der Populationsvielfalt. Durch die Frequenzanpassung der „replace-worst“-Strategie, konnten die Autoren den Suchraum effektiver erkunden und schneller optimale Lösungen finden.

Krishnan et al. (2008) entwickelten drei mathematische Modelle zur Optimierung des Anlagenlayouts unter Berücksichtigung variabler Produktnachfrage in einer nicht deterministischen Umgebung. Diese Modelle zielen auf die (1) Minimierung der maximalen Verluste durch MHC, (2) Minimierung der gesamten erwarteten Verluste durch Anpassung an verschiedene Szenarien und deren Wahrscheinlichkeiten, und (3) Optimierung von Layoutübergängen über mehrere Perioden hinweg, einschließlich der Berücksichtigung von Übergangskosten. Die Lösung dieser Modelle erfolgte durch den Einsatz von GA, um flexible und effiziente Layouts zu generieren, die auf Veränderungen in der Nachfrage reagieren können [198].

Garcia-Hernandez et al. (2018) haben einen Ansatz für Layouts mit ungleich großen Einrichtungen (vgl. kontinuierliche Repräsentation in Kapitel 3.1.2) veröffentlicht [199]. In ihrem Ansatz lassen sie eine repräsentative Teilmenge einer Generation durch einen Experten bewerten. Das quantitative Ziel der Reduzierung von Transportdistanzen und die Expertenbewertungen werden als getrennte Ziele behandelt. Mithilfe der Pareto-Analyse identifizieren sie optimale Lösungen, die in der nächsten Iteration durch den GA weiterentwickelt werden.

Liang und Fang (2019) entwickelten eine Methode zur Parallelisierung der Berechnung komplexer Systeme durch dezentralisierte GA [200]. Sie teilten das Layout in verschiedene Subsysteme auf, die gleichzeitig von identischen GA bearbeitet werden. Nach jeder Iteration findet ein Datenaustausch zwischen benachbarten Subsystemen statt. In sechs Experimenten verglichen sie die Leistung des parallelisierten GA mit der eines herkömmlichen GA und stellten fest, dass der parallelisierte GA in allen Fällen schneller konvergiert. Mit zunehmender Systemkomplexität wurde der Vorteil der Parallelisierung deutlicher, obwohl die Ergebnisse eine höhere Volatilität aufwiesen als die des herkömmlichen GA.

Besbes et al. (2020) haben einen GA zur Ermittlung des effizientesten Layouts in einem begrenzten Raum publiziert [201]. Durch Integration eines A*-Algorithmus in ihren GA berechneten sie die kürzesten euklidischen Distanzen zwischen zwei Maschinen und erstell-

ten über mehrere Iterationen verschiedene Layouts. In über 100 Simulationsdurchläufen fanden sie mittels eines Monte-Carlo-Algorithmus optimale GA-Parameter, die verbesserte Layouts generieren konnten.

4.3.2 Machine Learning

Jaber et al. (2007) diskutierten die Tendenz von GA, sich evolutionär in Sackgassen zu entwickeln. Die sei ein inhärentes Problem dieser Algorithmen [202]. Sie argumentierten, dass die evolutionäre Methode, die GA nutzen, zwar effektiv ist, um Lösungen zu ermitteln, jedoch auch dazu führen kann, dass sich die Lösungen von ungünstigen Bedingungen weg, aber nicht notwendigerweise hin zu optimalen Lösungen entwickeln. Dies kann zu evolutionären Sackgassen (lokale Optima) führen:

> „While the great advantage of GA is the fact that they find a solution through evolution, this is also the biggest disadvantage. Evolution is inductive; in nature life does not evolve towards a good solution but it evolves away from bad circumstances. This can cause a species to evolve into an evolutionary dead end [202].“

Um dieses Problem zu überwinden, erweiterten Jaber et al. ihren GA um eine Lerneinheit, die sie als „Keeping Efficient Population (KEP)“ bezeichnen. Dieses Tool vergleicht vergangene und zukünftige Generationen um zu entscheiden, mit welcher Generation die nächste Iteration fortgesetzt wird. Dadurch kann der GA den Suchraum effektiver und gleichmäßiger erkunden.

Rummukainen et al. (2018) präsentierten einen Ansatz zur Lösung des FLP, der auf traditionelle mathematische Modellierungen verzichtet und stattdessen ML-basierte Algorithmen verwendet [203]. Ihr Algorithmus trainiert auf Layouts, die von Experten für ähnliche Fabriken entworfen wurden. Die Bestimmung der Ähnlichkeit erfolgt über ein von den Autoren entwickeltes „Ähnlichkeitsmodell“. Das Ziel-Layout wird als *Multi-Floor Layout Problem (MFLP)* betrachtet, wobei die Ähnlichkeit zu den Expertenlayouts herangezogen wird. Rummukainen et al. zeigten, dass ihr Ansatz für kleinskalige Testdaten valide ist. Für präzisere Modelle wäre eine größere Menge an Trainingsdaten erforderlich.

Im Jahr 2022 veröffentlichten Hikaru Ikeda et al. das Paper „Towards Automatic Facility Layout Design Using Reinforcement Learning“, in dem sie einen Mechanismus zur optimierten Anordnung von OE vorstellen [204]. Dieser ermöglicht die Modellierung der OE entsprechend ihrer tatsächlichen physischen Repräsentation. In Experimenten zeigt der RL-Algorithmus eine Neigung zur bevorzugten Platzierung größerer vor kleineren Einheiten. Des Weiteren wurde beobachtet, dass der Agent effizienter mit einem kontinuierlich Zufluss neuer Informationen lernt, im Gegensatz zur wiederholten Bereitstellung bekannter Daten.

4.3.3 Deep Learning

Tsuchiya et al. (1996) untersuchten ein ANN, zur Lösung eines QAP, bei dem es darum geht, N Einrichtungen auf einem Gitter mit N^2-Positionen anzuordnen [205]. Jeder Knoten im Netzwerk repräsentiert eine Position und besitzt eine Aktivierungsfunktion. Diese kann Werte zwischen 0 und 1 annehmen und wird durch das höchste Gewicht zu einem bestimmten Zeitpunkt aktiviert. Die Gewichte werden mittels der Manhattan-Distanz bestimmt, wobei Knoten mit geringeren Transportkosten höhere Gewichte erhalten.

Die Analyse von Burggräf et al., [191] führt zwei verwandte Arbeiten von Kobayashi et al. [206] und Ueda et al. [207] zur Selbstorganisation von Potenzialfeldern in der Halbleiterherstellung auf. Maschinen werden durch die Knoten des ANN repräsentiert und erhalten Eigenschaften wie „im Leerlauf", „in Bearbeitung", „Fehler", „Wartung" und „Puffer mit Produkt". Anschließend werden Anziehung- und Abstoßungsfelder zur Vermeidung von Überlappungen erzeugt. Die Attribute der Anziehungsfelder für Transporter und Puffer werden zur Simulation der Fabrik-Dynamik anhand der Anforderungen des Produkts angepasst. Die Knoten (Maschinen) bewegen sich in Abhängigkeit von den generierten Feldern. Dadurch entsteht eine optimierte Maschinenanordnung, die die Gesamtreisezeit des Produkts minimiert. Die Simulationsergebnisse zeigten, dass die am häufigsten besuchten Maschinen im Herstellungsprozess von ihren jeweiligen Unterstützungsfunktionen umkreist wurden.

Tam und Tong veröffentlichten 2003 einen Ansatz zur Optimierung der Positionen von Turmkränen durch Minimierung der Transportzeiten [208]. Dazu kombinierten sie ein ANN mit einem genetischen Algorithmus. Das Netzwerk bestand aus fünf Eingangsknoten, einer vollständig verbundenen versteckten Schicht und einem Ausgangsknoten zur Vorhersage der Hebezeiten (Hin- und Rückweg). Der GA erzeugt weitere Layouts unter Berücksichtigung der vorhersagten Hebezeiten. Insgesamt wurden in dieser Studie die Daten aus sieben Projekten verwendet.

Garcia et al. (2014) entwickelten einen Ansatz zur Layoutoptimierung mithilfe eines GA, der durch Expertenbewertungen ergänzt wird [209]. Diese bezogen sich auf Materialfluss, Nachbarschaftsanforderungen und das Verhältnis von Raumabmessungen anhand einer fünfstufigen Likert-Skala[11]. Die Bewertungen dienen als Grundlage für das Training eines DNN, das die Rolle des Experten übernimmt. Insgesamt wurden 546 Layouts erstellt und in zwei Sets aufgeteilt: 365 für das Training und 181 für die Validierung des Modells. Das DNN prognostiziert mittels Softmax-Funktion die Bewertungen für neue Layouts (Softmax s. Kapitel 3.3.3.2). Die Genauigkeit dieser Vorhersagen variierte zwischen 80 % und 91 %, wobei das DNN als digitaler Experte agiert, der auf realen Expertenbewertungen basiert.

[11] Die Likert-Skala ist eine psychometrische Skala, die häufig in Fragebögen verwendet wird und es den Befragten ermöglicht, ihre Einstellungen zu einem Thema durch Ankreuzen auf einer mehrstufigen Antwortskala auszudrücken, die typischerweise von „stimme voll zu" bis „stimme gar nicht zu" reicht.

Azimi und Soofi entwickelten eine Methode, die DNN mit einem „Hybrid Non-dominated Sorting Genetic Algorithm II“ (H-NSGA-II[12]) kombiniert, um dynamische Fabriklayouts zu optimieren [210]. Das System erzeugt 24 verschiedene Layouts und bestimmt die Produktionsdauer durch eine ereignisorientierte Simulation. Diese Daten trainieren dann ein DNN, das in der Lage ist, Produktionszeiten effizient vorherzusagen, wobei reale Produktionsbedingungen wie zufällige Ausfälle und Bearbeitungszeiten simuliert werden.

In einer weiteren Studie von Hou et al. (2022) wird ein Ansatz mittels CNN zur Vorhersage des zu erwartenden Durchsatzes von Fabriklayouts vorgestellt [211]. Sie führen eine Technik zur Reduzierung einer ungleichen Verteilung der Vorhersagen ein (genannt „Zielvektorisierung“) mit dem Ziel die Genauigkeit der Vorhersagen zu verbessern. In Experimenten mit verschiedenen CNN-Konfigurationen konnte die Studie zeigen, das mit der Technik die Genauigkeit verbessert werden konnte.

4.4 Anwendung von KI zur Fabriklayoutoptimierung

Die im vorigen Abschnitt aufgeführten Ansätze unterstützen die Ergebnisse von Burggräf et al., dass der Einsatz von KI in der Fabriklayoutoptimierung unterrepräsentiert ist [191]. In dieser Studie wurde anhand von folgenden zwei Einschlusskriterien der bestehende Forschungsstand gefiltert:

- Relevanz der Veröffentlichung für das Forschungsfeld zum FLP.
- Die Veröffentlichung setzt ein ML-Framework (exakte, heuristische oder intelligente Methoden) zur Lösung des FLP ein.

Zum Einsatz von **Supervised Learning** wurden unter Berücksichtigung dieser Kriterien neun Arbeiten identifiziert. Diese setzen SL jedoch im Rahmen eines kombinierten Ansatzes ein und nicht direkt zur Lösung des FLP. Hauptursachen sind der fehlende Zugriff auf geeignete gelabelte Daten und die NP-Vollständigkeit bei Problemgrößen mit mehr als 15 OE [191].[13] Insgesamt sind zwei Ansätze erkennbar, um diesen Umstand zu mitigieren. Zum einen werden Layouts künstlich erzeugt, zum anderen werden bestehende Layouts durch Experten beurteilt. Der erste Ansatz weist den Vorteil auf, Daten in beliebigen Mengen erstellen zu können. Ein Nachteil ist allerdings ein hoher Implementierungsaufwand und eine Begrenzung zur praktischen Anwendbarkeit. Die durch Experten bewerteten Layouts weisen eine höhere praktische Anwendbarkeit auf, allerdings unterliegen die Bewertungen

[12] Der NSGA-II ist ein genetischer Algorithmus, der zur Lösung von mehrkriteriellen Optimierungsproblemen entwickelt wurde. In der „Hybrid“-Version (also dem H-NSGA-II) wird der NSGA-II-Algorithmus in Kombination mit anderen Techniken oder Algorithmen genutzt, um die Effizienz und die Lösungsqualität zu verbessern.

[13] Dieser Wert entspricht der Angabe der durchgeführten Literaturanalyse mit Stand 2021 und wird sich mit verbesserten Rechenkapazitäten in der Zukunft erhöhen.

einer subjektiven Qualität und sind daher ungeeignet zum Training eines generalisierenden ANN.

Ungefähr die Hälfte der identifizierten Beiträge zu **Unsupervised Learning** wird direkt zur Lösung von FLP eingesetzt, hauptsächlich unter Verwendung von Kohonen- oder Hopfield-Netzwerken. was wahrscheinlich auf ihre geringe Rechenlast im Vergleich zu Metaheuristiken zurückzuführen ist. Die restlichen Arbeiten konzentrierten sich auf „Group Technology“, einen Forschungsbereich, der eher der Produktionsplanung als dem Layout-Design zuzuordnen ist.

86% (19 von 22) der untersuchten Veröffentlichungen verwenden sowohl überwachte als auch unüberwachte Ansätze. Die spezifischen Gründe und Vorteile für die Wahl von KI-basierten Algorithmen gegenüber anderen Algorithmen bleiben jedoch oft unklar. Nur zwei der gesichteten Studien setzen **Reinforcement Learning** ein, ohne jedoch beide Einschlusskriterien zu erfüllen. Das Potenzial von RL in der FLP-Forschung, insbesondere in der Neugestaltung klassischer Layout-Formulierungen als Markov-Entscheidungsprozesse, bleibt bestehen, vgl. Kapitel 3.3.2.3.

Die Ergebnisse verdeutlichen, dass intelligente Ansätze, die eines der drei Machine Learning Paradigmen - Supervised, Unsupervised und Reinforcement Learning, sowie das querschnittliche Paradigma Deep Learning nutzen, im Vergleich zu reinen Metaheuristiken in der FLP-Literatur unterrepräsentiert sind, vgl. Abbildung 14. Die zeitliche Verteilung zeigt, dass vor 1994 keine Forschungsaktivitäten zum Thema ML in FLP stattfanden, mit einem Anstieg der Aktivität bis 2004. Beiträge nach diesem Zeitpunkt sind marginal. Dies wird dem Ende des „KI-Winters“ im Jahr 1997 zugeschrieben, der möglicherweise zu frischer Finanzierung für ML-Forschung in verschiedenen praktischen Bereichen geführt hat. Eine weitere Auffälligkeit ist die signifikant geringere Zitierung von Veröffentlichungen mit strengerem KI-Bezug als zu denen mit metaheuristischen Ansätzen. Dies ist gegenläufig zu Trends in anderen Forschungsrichtungen, bei denen KI-basierte Ansätze eine zunehmend höhere Aufmerksamkeit erfahren.

4.4.1 Fazit zum aktuellen Stand

Metaheuristische Methoden haben sich als wirkungsvoll zur Fabriklayoutoptimierung und für das FLP etabliert. Insbesondere GA, SA und ACO liefern qualitativ hochwertige Lösungen für FLP-Fragestellungen, siehe Kapitel 4.3.1. Praktische Anwendungen dieser Techniken sind in kommerziellen Softwarelösungen wie *Plant Sim, OptQuest, und Gurobi* prominent vertreten. Allerdings bieten Metaheuristiken in der Regel approximative Lösungen und garantieren nicht das Erreichen des globalen Optimums. Zusätzlich hängt die Effektivität dieser Algorithmen von Parametereinstellungen ab, die oftmals schwer zu kalibrieren sind. In diesem Sinne ist ihre breite Verbreitung sowohl ein Indikator für ihre Vielseitigkeit und

Leistungsfähigkeit bei der Behandlung von FLP, als auch ein Zeichen für die Komplexität ihrer optimalen Anwendung.

Bezogen auf die Anwendung von **maschinellem und tiefem Lernen** zeigt der dargestellte Stand der Technik eine begrenzte Einsetzbarkeit zur Lösung von FLP. Die meisten Forschungsarbeiten setzen Supervised, Unsupervised oder Reinforcement Learning nicht direkt zur Lösung von FLP ein. Zum Zeitpunkt der Ausarbeitung sind keine kommerziellen Produkte bekannt, die KI-basierte Solver zur Lösung des FLP einsetzen.

Insgesamt wird deutlich, dass die Zukunft weniger in der alleinigen Anwendung einzelner metaheuristischer oder maschineller Lernalgorithmen zu finden ist, sondern vielmehr in der **Kombination verschiedener Techniken**, um die jeweiligen Stärken zu bündeln und Nachteile zu kompensieren. Diese These wird durch den dargestellten Forschungsstand unterstützt, der vermehrt hybride Ansätze aufweist. Dies scheint insbesondere für das FLP durch seine inhärente Komplexität und Variabilität von Bedeutung zu sein. Eine monolithische Lösung mit einem einzelnen KI-basierenden Algorithmus erscheint unwahrscheinlich. Der Einsatz von Metaheuristiken, um das Suchverhalten zu lenken und maschinelles Lernen zur Anpassung und Optimierung an spezifische Problemstellungen, ist der meist verwendete hybride Ansatz.

Eine Herausforderung liegt in der Datenbeschriftung. Dies kann durch den Einsatz ereignisorientierter Simulation zum Erstellen von Labels mitigiert werden. Obwohl Fabriklayouts grafische Darstellungen sind, wurden CNN in den untersuchten Arbeiten bisher nicht angewendet. CNN, die in der Bild- und Mustererkennung erfolgreich sind, könnten eine neue Methode zur Analyse von Layoutdaten darstellen, indem sie beispielsweise Materialflussmuster oder Produktionsengpässe erkennen. Die Anwendung von CNN auf das Fabriklayoutproblem wird durch ihre Fähigkeit unterstützt, lokale und räumliche Muster in den Daten zu erkennen. Von Vorteil ist, dass die relative Position von Maschinen und Transportwegen eine wichtigere Rolle als die absolute Position einer Maschine im Raum einnimmt. In diesem Zusammenhang könnte auch die Translationsinvarianz von CNN vorteilhaft sein, da sie identische Muster unabhängig von ihrer Position im Layout erkennen können. CNN können Layouts mit unterschiedlichen Anzahlen von Maschinen und Transportwegen zu verarbeiten, da sie nicht wie MLP-Architekturen auf einen festen Eingangsvektor begrenzt und trainiert sind. Dies eröffnet die Möglichkeit zur Generalisierung auf verschiedene Layoutkonfigurationen.

Konkatenierte ANN, die CNN und MLP kombinieren, könnten eine weitere Methode zur Analyse von Fabriklayouts darstellen. Ein MLP könnte genutzt werden, um Analysen der vom CNN erfassten Merkmale durchzuführen und subtilere Abhängigkeiten zu identifizieren, die nicht direkt aus den räumlichen Beziehungen abgeleitet werden können. Es könnte auch zur Integration von nicht räumlichen Informationen wie Produktionsraten oder Maschinenkapazitäten beitragen.

5 Konzeption und Modellierung

In diesem Kapitel wird das der Arbeit zugrunde liegende Konzept erläutert. Dieses dient als Leitfaden für die Entwicklung, Durchführung der Experimente und Analyse der Ergebnisse im Zusammenhang mit der Forschungsfrage zum Einsatz von KI in der Layoutoptimierung. Das Konzeptverständnis dieser Arbeit folgt der nachstehenden Definition [212]:

> „Unter dem Begriff ‚Konzept' versteht man eine Planung, welche aus den Ideen einer Absicht oder eines Entschlusses und deren Ziele die Mittel und Wege (Maßnahmen) definiert, um die fixierten Ziele zu erreichen. Ein Konzept beschreibt alle Details der Vorgehensweise. Ein Konzept kann sowohl skizzenhaft, aber auch detailliert und verbindlich gestaltet werden."

Wesentliche Teile des Konzeptes sind die Modellierung der Fabrikprozesse sowie die Entwicklung einer KI-Architektur. Eine wissenschaftlich anerkannte Modelldefinition gibt es nicht. Für diese Arbeit bezieht sich auf das Verständnis für die Modellierung nach Kossiakoff et al. [213]. Er beschreibt diese folgendermaßen:

> „A model of a system can be thought of as a simplified representation or abstraction of reality used to mimic the appearance or behavior of a system or system element."

Blanchard und Fabrycky zufolge sind folgende Arten der Modellierung nach anerkannt [214]. Die Auflistung orientiert sich an der allgemeinen Reihenfolge von zunehmender Realität und abnehmender Abstraktion:

- **Schematische Modelle:** Dies sind Diagramme oder Grafiken, die ein Systemelement oder einen Prozess repräsentieren und gehören zur Kategorie *deskriptiver Modelle*. Beispiele sind Cartoons, Architekturen, Blockdiagramme, Kontextdiagramme, Funktionsflussdiagramm, Datenflussdiagramm, und 3D-Modelle. In dieser Arbeit wird ein Ablaufdiagramm zur Darstellung des Trainingsprozesses vom ANN verwendet.
- **Mathematische Modelle:** Diese Modelle verwenden eine mathematische Notation zur Beschreibung einer Beziehung oder Funktion. Beispiele hierfür sind Approximationen, elementare Beziehungen[14], statistische Verteilungen und Graphen. In dieser Arbeit wird ein Graph für die Darstellung der Abhängigkeiten der Simulation verwendet.
- **Physische Modelle:** Diese Modelle reflektieren einige oder die meisten physischen Eigenschaften eines Systems oder Systemelements. Das physische Modell kann ein

[14] Zum Beispiel die Newtonsche Gesetze.

tatsächlicher Teil eines realen Systems sein, oder ein repräsentativer Dummy. Beispiele sind Miniaturen (Erd-Globus), ein Kugel-Stab-Modell, Mock-ups und Prototypen. Im Konzept dieser Arbeit wird kein physisches Modell verwendet. In einer im Kontext veröffentlichten Untersuchung zum Einsatz eines Supervisors in einem GA wurde ein Fabrik-Zwilling implementiert, siehe [2].

In der Darstellung zum Stand der Wissenschaft wurde herausgearbeitet, dass überwachte Lernansätze in der Fabriklayoutoptimierung nicht verwendet werden. Dies ist vor allem auf den fehlenden Zugang zu beschrifteten Layouts zurückzuführen. Das im Folgenden vorgestellte Konzept baut auf dem aus der Literatur erlerntem Wissen auf und erweitert es um weitere Annahmen und Erkenntnisse. Hierbei wird ein methodischer Ansatz durchgeführt, der die Erzeugung von Trainingsdaten, die Auswahl und Training eines geeigneten KI-Frameworks sowie dessen Evaluierung und Validierung umfasst.

5.1 Konzeption zur KI-unterstützten Layoutoptimierung

Abbildung 15 zeigt die Konzeption für einen KI-unterstützten Optimierungsansatz.

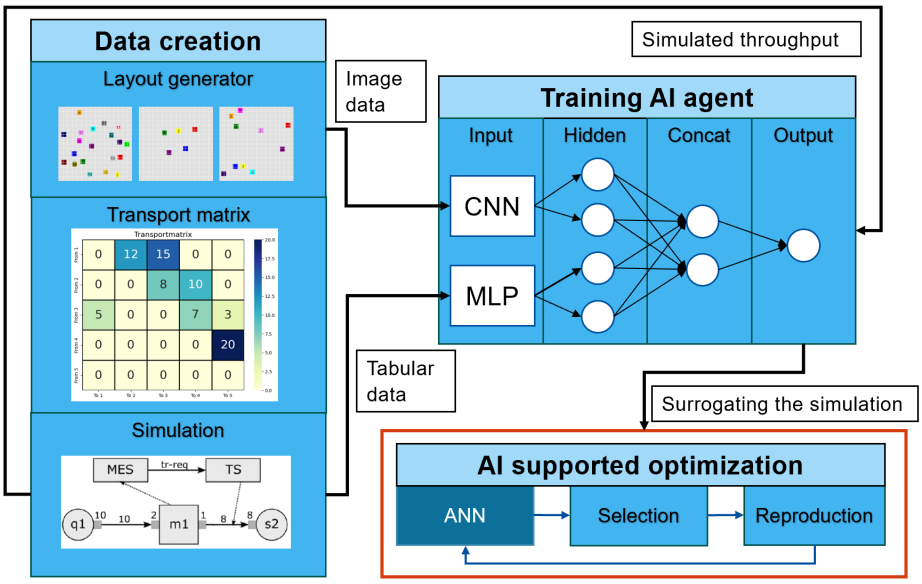


Abbildung 15: Konzeption für die KI-unterstützte Layoutoptimierung

Die Konzeption besteht aus drei Hauptphasen, die in den folgenden Unterabschnitten erläutert werden. Die erste Phase umfasst einen Mechanismus zur Generierung verschiedener Layoutvarianten. Dieser ermöglicht die Erstellung einer Datenbasis, die unterschiedliche Konfigurationen enthält. Weitere Komponenten dieser Phase sind eine Transportmatrix, die die Verbindungen zwischen den Fabrikeinheiten abbildet und eine Simulationsumgebung.

Diese ermöglicht es, zu jedem Layout unter definierten Randbedingungen Metriken zu erstellen. Zu diesen zählen unter anderem der vom jeweiligen Layout erzielte Materialdurchsatz oder die Manhattan-Distanz. Aus den Metriken werden Labels definiert, die die zu lernende Zielfunktion des ANN repräsentieren. In der zweiten Phase werden die gelabelten Daten verwendet, um ein ANN zu trainieren. Ziel ist es, eine Netzwerkarchitektur zu entwickeln, dass die Fähigkeit besitzt, basierend auf Positionen der OE den voraussichtlichen Durchsatz zu bestimmen. Die letzte Phase integriert das Netzwerk in einen GA mit dem Zweck, das ANN zur Bewertung unbekannter Layouts anstelle der Simulation einzusetzen, s. Kapitel 5.4.

5.2 Phase 1 zur Erzeugung von Trainingsdaten

In [209] wurde ein GA zur Erzeugung von Layoutdaten eingesetzt. Dieser Ansatz generiert eine große Menge an Trainingsdaten, deckt jedoch den Suchraum nur eingeschränkt ab, da sie sich auf Bereiche konzentrieren, in denen bereits Lösungen gefunden wurden. Dies führt dazu, dass schlechte Layoutvarianten unterrepräsentiert sind. Gerade diese sind zum Training des ANN wichtig, da sie zur vollständigen Abbildung des Problems beitragen und steigern dessen Fähigkeit zur Generalisierung. Die in den verwandten Studien verwendeten Datenmengen waren gering: [209] nutzte 365 Samples und [210] 24 Samples. Diese geringen Datenmengen erhöhen das Risiko des Overfitting, wobei das Modell zu stark auf die Trainingsdaten angepasst wird. Für diese Arbeit ist ein unbeschränkter Zugang zu Layoutdaten angestrebt. Eine Studie von Sun et al. zeigte, dass die Genauigkeit von ANN mit der Größe des Trainingsdatensatzes skaliert und dass eine Erhöhung der Datenmenge die Modellleistung signifikant verbessern kann, selbst wenn weitere Faktoren wie die Modellarchitektur oder die Optimierungsmethode konstant gehalten werden [215].

5.2.1 Layoutgenerator

Basierend auf eingangs dargelegter Überlegung wurde ein Layoutgenerator entwickelt, der auf einer Monte-Carlo-Simulation basiert. Diese Methode ist geeignet für die Erzeugung von repräsentativen Daten durch Ziehen zufälliger Stichproben aus einer Wahrscheinlichkeitsverteilung. Im Vergleich zu heuristischen Methoden wie GA bietet die Monte-Carlo-Simulation eine umfassendere Abbildung des Lösungsraums, s. Abbildung 16.

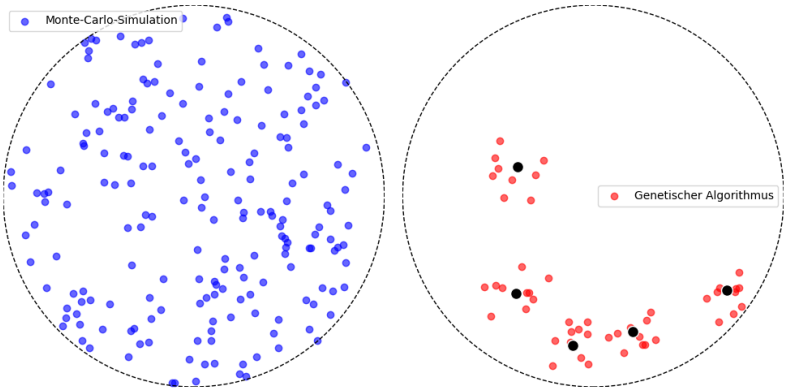


Abbildung 16: Suchraumabdeckung von Monte-Carlo-Algorithmus und GA

Die vom Monte-Carlo Algorithmus (links) generierten Layouts sind unabhängig voneinander und werden ohne Berücksichtigung vorheriger Generationen erstellt. Hierdurch können auch unerwartete und unkonventionelle Layouts generiert werden, die in einer heuristischen Suche möglicherweise übersehen würden. Die rechte Hälfte der Abbildung stellt einen GA dar, der sich auf spezifische Bereiche konzentriert, in denen der Startpunkt gesetzt wurde oder bereits Lösungen gefunden wurden. Dies wird durch die rot markierten Punkte und die schwarzen Zentren der Cluster verdeutlicht. Die Clusterbildung zeigt, dass der GA seine Suche um erfolgreiche Lösungsbereiche fokussiert und dadurch andere potenziell wichtige Bereiche des Suchraums weniger berücksichtigt. Daher ist die Implementierung einer Monte-Carlo-basierten Methode zur Generierung von Fabriklayouts eine geeignete Strategie, um eine ausführliche und diversifizierte Datengrundlage für das Training von ANN zu erstellen. Details zu Randbedingungen und Datenstrukturen des Layoutgenerators werden in den Erläuterungen zur Implementierung in Kapitel 6.2 behandelt.

5.2.2 Layoutbewertung durch Simulation

Der Layoutgenerator generiert eine beliebige Anzahl an Layouts, einschließlich grafischer Darstellungen und Koordinatenpositionen der OE sowie Distanzmetriken. Eine qualitative Bewertung der Eignung dieser Layouts für den Einsatz in produktiven Umgebungen erfolgt durch Simulation, analog zu den Methoden in [62] und [210].

Hinsichtlich der infrage kommenden Simulationstechnik erfüllt die ereignisorientierte Simulation die Anforderungen an die Erstellung von Leistungsmetriken zu den Layouts. Durch die zugrunde liegende Systematik lassen sich individuelle Ereignisse in einem Layout, wie die Ankunft von Materialien, Bearbeitungszeiten oder Ausfälle, unabhängig voneinander modellieren. Jedes Ereignis führt zur Änderung des Systemzustands und zur Planung neuer Ereignisse. Auch die kontinuierliche Simulation kann zur Modellierung und Abbil-

dung von Prozessen innerhalb einer Fabrik verwendet werden. Ihr Fokus liegt allerdings auf Prozesse, die sich über die Zeit kontinuierlich verändern, wie der Energieverbrauch oder Temperaturveränderungen innerhalb einer Fabrik, vgl. Kapitel 3.2. Diese Prozesse haben abhängig von dem Optimierungskriterium ebenfalls Auswirkungen auf die optimale Platzierung von OE, sind aber nicht Bestandteil dieser Untersuchung. Die Anwendung der DES zur Bewertung von Fabriklayouts ist aus folgenden Gründen zur Behandlung der Forschungsfrage zielführend:

- **Geschwindigkeit:** Ereignisorientierte Simulationen sind schneller als kontinuierliche Simulationen, da sie diskrete Ereignisse isoliert betrachten.
- **Detailgrad:** DES erlauben es, komplexe Systeme in definierbarer Detailtiefe zu modellieren.
- **Flexibilität:** DES sind in der Lage, beliebige Layoutvarianten und Randbedingungen zu berücksichtigen.
- **Effizienz:** Da Ereignisse nur dann behandelt werden, wenn sie eintreten, werden unnötige Berechnungen vermieden.
- **Realitätsnähe:** Durch das Modellieren diskreter Ereignisse, die in realen Produktionsumgebungen auftreten, können realistischere Ergebnisse erzielt werden.

Abbildung 17 zeigt den Simulationsgraphen zur ereignisorientierten Simulation, siehe hierzu ebenfalls folgende im Kontext dieser Arbeit entstandenen Veröffentlichungen [1, 2, 3, 8], vgl. Zusammenfassungen in der Einleitung ab Seite ix bis xii.

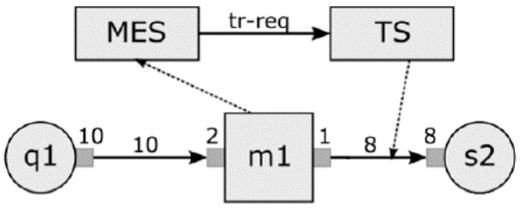


Abbildung 17: Simulationsgraph der ereignisorientierten Simulation

Der Simulationsgraph modelliert ein Fabriklayout, das sich auf den Produktionsfluss von den Quellen zu den Senken konzentriert, wobei die Maschineneinheiten als zentrale Verarbeitungseinheiten fungieren. Die minimale Konfiguration umfasst eine Quelle, die unendliches Material liefert, und eine Senke, die die verarbeiteten Güter empfängt. Die Materialflüsse werden durch unidirektionale Linien dargestellt, die die transportierte Materialmenge anzeigen und ein System hervorheben, in dem Materialien gemäß vordefinierten Rezepten von den Ausgangs- zu den Eingangspuffern bewegt werden. Jeder Produktionszyklus wird initiiert, wenn die Eingangspuffer die erforderlichen Materialmengen erreicht haben und ausreichend Platz in den Ausgangspuffern für die Fertigprodukte vorhanden

ist. Das Manufacturing Execution System (MES) gleicht volle Ausgangspuffer mit leeren Eingangspuffern ab, um Transportanfragen (engl. transport requests - tr-req) zu generieren, die an das Transportsystem (TS) weitergeleitet werden. Dieses Framework ermöglicht eine vielseitige Konfiguration von Fabriklayouts durch die Anpassung der Anzahl von Maschinen, Lagern, Puffern und Rezepten. Die Implementierung des Frameworks im Rahmen der DES bestimmt die folgenden Metriken für die Layouts:

- Durchlaufzeit von Materialien
- Tatsächlich zurückgelegte Strecke des Transportmediums (MHC)
- Durchsatz von Materialien

Die durch DES ermittelten Metriken ermöglichen das Erstellen von Labels bzw. Zielwerten für ANN. Die OE und das Transportsystem wurden in eigenen Klassen implementiert, siehe später folgendes Kapitel 6.3. Das Diagramm in Abbildung 18 stellt den Ablauf der Simulation dar.

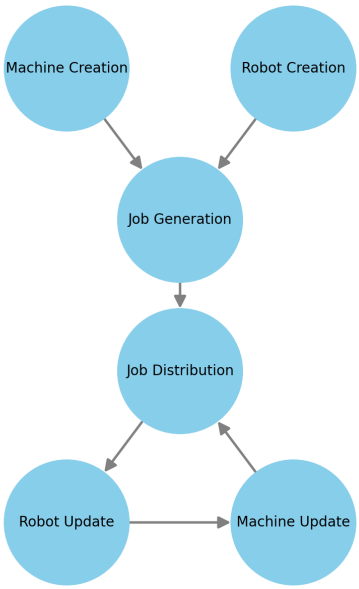


Abbildung 18: Simulationslogik

In der Simulation werden zunächst n-Maschinen und m-Roboter erstellt, s. Anfangsknoten. Je nach Anzahl der Transporteinheiten und Maschinen werden alle möglichen Transportaufträge erzeugt. Im nachfolgend dargestellten Kreislauf werden die Schritte der Aufgabenverteilung sowie Aktualisierung der Transporteinheiten und Maschinen sukzessive durchlaufen. Beim Transportroboter wird in jeder Iteration die aktuelle Position, der Ladezustand und Zustand (Laden / Entladen) aktualisiert. Bei den Maschinen werden der Eingangs- und Ausgangspuffer sowie der Zustand (Warten, Produzieren) aktualisiert. Schließlich fließt der Maschinen-Update-Knoten zurück in Aufgabenverteilung, um zu

prüfen ob neue Liefer- oder Abholaufträge generiert werden. Transportgeschwindigkeiten, Fertigungsdauern und Ladezeiten sind konfigurierbar und wurden experimentell im Rahmen der Implementierung festgelegt, s. Kapitel 6.3.4.

5.2.3 Datenaufbereitung für das ANN

Die Datenaufbereitung bereitet die erzeugten Layouts und Metriken für den Einsatz in ANN auf. Diese Phase beeinflusst die Leistung des Modells, indem sie die Qualität und Quantität der Trainingsdaten optimiert. Für die Vorhersage des simulierten Durchsatzes wurden die im Folgenden erläuterten Vorverarbeitungsschritte durchgeführt.

5.2.3.1 Normalisierung der Daten

Die Normalisierung der Ein- und Ausgangsdaten auf einen einheitlichen Wertebereich fördert eine schnellere Konvergenz des Netzwerks durch Stabilisierung der Gradientenberechnung während des Backpropagation-Prozesses. Folgende Methoden zur Normalisierung der Daten stehen hier zur Verfügung:

1. **Z-Score-Normalisierung**: Die Zielvariable wird transformiert, sodass sie einen Mittelwert von null und eine Standardabweichung von eins aufweist.

$$x_{\text{z-score}} = \frac{x - \mu}{\sigma} \tag{26}$$

   Hierbei ist μ der Mittelwert und σ die Standardabweichung der ursprünglichen Zielvariable.

2. **Min-Max-Normalisierung**: Diese Methode skaliert die Zielvariable in einen festgelegten Bereich, oft [0, 1].

$$x_{\text{min-max}} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{27}$$

   Hier sind *min* und *max* die minimalen und maximalen Werte der ursprünglichen Zielvariablen.

Die Wahl der Methode und des Normalisierungsintervalls hängt von der Datenstruktur, der Aktivierungsfunktion und dem Typ des ANN ab. Das Intervall [0, 1] wird häufig verwendet, wenn die Aktivierungsfunktionen des neuronalen Netzwerks im positiven Bereich operieren, wie bei der ReLU-Aktivierungsfunktion. Das Intervall [-1, 1] ist vorteilhaft für Aktivierungsfunktionen wie Tangens Hyperbolicus (tanh), die sowohl positive als auch negative Werte effektiv nutzen. Die Z-Score-Normalisierung, die weniger anfällig für Ausreißer ist als die Min-Max-Normalisierung, verwendet den Mittelwert und die Standardabweichung, um extreme Datenpunkte weniger einflussreich zu machen.

Der durch Layoutgenerator und Simulation erzeugte Datensatz enthält Bilder, tabellarische Daten und Zielmetriken. Die Bilder sind in RGB-codiert. Die Intensitätswerte I jeder Farbkomponente, die ursprünglich im Bereich von 0 bis 255 liegen, werden durch 255 dividiert, um sie zu normalisieren. Dies entspricht der Min-Max-Normalisierung. Dadurch liegen die neuen Intensitätswerte I im Intervall von 0 bis 1.

$$I_{\text{normalisiert}} = \frac{I_{\text{original}}}{255} \tag{28}$$

Die Normalisierung der tabellarischen Daten und der Zielvariablen haben ebenfalls stabilisierende Auswirkungen auf den Trainingsprozess und werden gleichsam mit der Min-Max-Normalisierung auf das Intervall von 0 bis 1 normalisiert. Der Durchsatz von Fabriklayouts weist einen natürlich begrenzten Bereich auf, dies begünstigt die Min-Max-Normalisierung und die Rücktransformation der normalisierten Vorhersagen in die ursprüngliche Skala. Die Rücktransformation erfolgt nach folgender Formel:

$$y_{\text{original}} = y_{\text{min-max}} \times (\max - \min) + \min \tag{29}$$

Die Rücktransformation in das ursprüngliche Maß hilft bei der weiteren Interpretation und Auswertung der Ergebnisse.

5.2.3.2 Festlegung des Layouthintergrunds

Die Layoutfläche wurde auf eine einheitliche Eingabegröße von 128×128 Pixel skaliert. Diese Anpassung erfolgt mittels Bilinear-Interpolation, die einen guten Kompromiss zwischen Rechenzeit und Bildqualität bietet.

$$\text{Einheitliche Bildgröße der Layouts} = 128 \times 128$$

Dieser Wert wurde experimentell und auf Basis der folgenden Abschätzung bestimmt. Die Anzahl der Pixel limitiert direkt die Platzierungsmöglichkeiten des Layoutgenerators. Formate wie 32×32 Pixel bieten weniger Vielfalt bei der Generierung von Layouts. Der Zieldatensatz sollte mindestens 100.000 Layouts enthalten, die sich deutlich voneinander unterscheiden. Außerdem sollte die Generierung dieser Anzahl von Layouts einen Algorithmus nicht zu viel Zeit kosten. Die theoretischen Platzierungsmöglichkeiten für fünf Fabrikeinheiten, jede mit einem effektiven Maß von 10×10 Pixeln in einem 128×128 Pixel Layout, bei Annahme sukzessiv verringerter Platzierungsoptionen durch Überlappungsvermeidung und Abstand zwischen den Einheiten, lassen sich wie folgt annähern:

$$N = (\max\{0, (128 - 10 + 1)\})^2 \times \left(\prod_{i=1}^{4} \left(\max\{0, (128 - 10 + 1)\}^2 \times 0.6^i \right) \right) \tag{30}$$

Der Ausdruck $128 - 10 + 1$ berechnet die Anzahl der möglichen Startpositionen für eine Einheit, die unter Berücksichtigung eines 5-Pixel-Abstands effektiv eine Fläche von 10×10 Pixeln einnimmt. Dies stellt sicher, dass die Einheit vollständig innerhalb der 128×128 Pixel Grenze bleibt. Die Formel stellt eine Näherung dar, die davon ausgeht, dass jede weitere Einheit 60 % der Platzierungsmöglichkeiten der vorherigen verbraucht, was die kombinatorische Komplexität der exakten Berechnung deutlich reduziert. Mit dem Ergebnis von etwa 3.44×10^{18} möglichen Layouts wird die Zielvorgabe von über 100.000 Layouts weit übertroffen. Diese große Anzahl möglicher Layouts ermöglicht dem Algorithmus schnell eine Vielzahl unterscheidbarer Varianten zu generieren.

5.2.3.3 Datenvergrößerung

Durch *Datenvergrößerung* (engl. Data Augmentation) werden durch kleine Modifikationen der vorhandenen Daten die Größe des Datensatzes künstlich erhöht [216]. Gängige Techniken sind die Rotation oder Vergrößerung von Bilddaten. Diese Methode wurde nicht angewendet, da die Layoutbilder und zugehörigen Metriken spezifische Informationen enthalten, die durch Datenvergrößerung möglicherweise verfälscht werden. Zudem ermöglicht der Layoutgenerator die Erzeugung einer Vielzahl von Layouts unter kontrollierten Bedingungen, wodurch eine künstliche Vergrößerung des Datensatzes überflüssig ist.

5.2.3.4 Behandlung von Ausreißern

Ausreißer können den Trainingsprozess stören, da sie die Modelle dazu verleiten können, sich auf atypische, extreme Werte zu konzentrieren. Dies verringert die Generalisierungsfähigkeit [217]. Um die Auswirkungen von Ausreißern zu minimieren, wurden folgende Methoden zur Identifikation und Analyse von Ausreißern angewendet:

1. **Visualisierung**: Durch den Einsatz von Scatterplots und Boxplots wurde die Verteilung der Manhattan-Distanzen visuell analysiert, um Anomalien zu erkennen.
2. **Statistische Tests**: Der Z-Score, ein statistisches Maß, das angibt, wie viele Standardabweichungen ein Datenpunkt vom Mittelwert entfernt ist, wurde für jeden Datenpunkt berechnet. Datenpunkte mit einem Z-Score größer als 3 oder kleiner als -3 wurden als Ausreißer klassifiziert. Diese Methode ist effektiv, um Extremwerte zu identifizieren, die das Lernverhalten des Modells negativ beeinflussen könnten.
3. **Robuste Statistiken**: Zur weiteren Überprüfung wurden robuste Statistiken wie der Median und der Interquartilsabstand (IQR) verwendet. Dieser misst die statistische Streuung, um die Auswirkungen von Ausreißern zu verringern, da er weniger empfindlich auf extrem abweichende Werte reagiert als der Mittelwert.

Diese Methoden stellen sicher, dass die Datenqualität und -zuverlässigkeit für den Trainingsprozess erhalten bleibt und verbessern die Robustheit der Ergebnisse.

5.2.3.5 Datenbereinigung

Die Datenbereinigung ist ein weiterer Schritt der Datenaufbereitung, der die Qualität und somit die Zuverlässigkeit der Modellleistung beeinflusst [217]. In diesem Rahmen wurden folgende Maßnahmen in das Konzept integriert, um die Datenqualität für das Training der Modelle zu gewährleisten:

1. **Konsistenzprüfung**: Es wurde sichergestellt, dass alle Datenpunkte den erwarteten Wertebereichen und Datentypen entsprechen. Zum Beispiel sollten alle simulierten Durchsätze positive Ganzzahlen und von Null verschieden sein.
2. **Fehlende Werte**: Der Datensatz wurde auf fehlende oder unvollständige Einträge untersucht. Es wurden keine fehlenden Werte festgestellt.
3. **Duplikateprüfung**: Duplikate wurden bereits bei der Erstellung der Layouts verhindert. Eine weitere Überprüfung auf Dubletten folgte in Bezug auf die simulierten Durchsätze, vgl. Kapitel 7.2.1.4.
4. **Anomalieerkennung**: Es wurden Methoden zur Anomalieerkennung angewendet, um ungewöhnliche Muster oder Werte zu identifizieren, die auf fehlerhafte Daten hindeuten könnten.

Mit diesen Schritten wurde der Datensatz auf Konsistenz und Fehlerfreiheit überprüft. Diese Qualitätssicherung soll das Vertrauen in die späteren Ergebnisse erhöhen.

5.2.3.6 Feature-Auswahl

Die Feature-Auswahl ist eine Technik, die zur Reduzierung der Anzahl der verwendeten Merkmale und zur Verbesserung der Modellleistung eingesetzt wird [218]. Für den vorliegenden Zusammenhang wurde beschlossen, alle verfügbaren Features zu nutzen, da jedes Feature wichtige Informationen zur Vorhersage des Durchsatzes enthält. Daher wurde keine Feature-Auswahl oder -Reduzierung durchgeführt.

5.2.3.7 Datenteilung

Die Aufteilung der Daten in Trainings-, Validierungs- und Testsets ermöglicht eine Beurteilung der Leistung und Generalisierbarkeit des trainierten Modells [117]. Die vorliegenden Daten wurden im folgenden Verhältnis aufgeteilt:

$$\text{Trainingsdaten} : \text{Validierungsdaten} : \text{Testdaten} = 70\% : 15\% : 15\%$$

Dieses Verhältnis gewährleistet, dass das Modell mit einer ausreichenden Menge an Daten trainiert wird und zugleich genügend Daten für Validierung und Tests verfügbar sind. Diese Verteilung wird häufig in verwandter Literatur empfohlen [219]. Die Implementierung und Umsetzung zur Datenaufbereitung ist in Kapitel 6.4 dokumentiert.

5.3 Phase 2 - Konzeption & Entwicklung der KI-Architektur

Die Entwicklung einer KI-Architektur ist ein iterativer und komplexer Prozess, der sorgfältige Planung erfordert. Dies umfasst die Auswahl geeigneter Algorithmen, Optimierung von Hyperparametern, Datenbereinigung und abschließende Modellvalidierung. Das entwickelte Modell muss auf Trainings- und Testdaten gleich gute Ergebnisse erzielen. In diesem Rahmen werden zwei Methoden zur Entwicklung einer für die Problemstellung geeigneten KI-Architektur angewandt: das Design of Experiments (DoE) in Kombination mit einer auf *Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS)* basierenden Analyse. Das DoE ermöglicht einen systematischen Ansatz zur Durchführung von Experimenten, um die Effekte verschiedener Faktoren und deren Interaktionen auf die Modellleistung zu verstehen, siehe Kapitel 5.3.1. Nach der Entwicklung einer Reihe von Modellen wird die TOPSIS-Analyse eingesetzt, um aus den erstellten Modellen das für die Problemstellung am besten geeignete zu identifizieren. Das Konzept hierzu wird in Abbildung 19 visuell dargestellt.

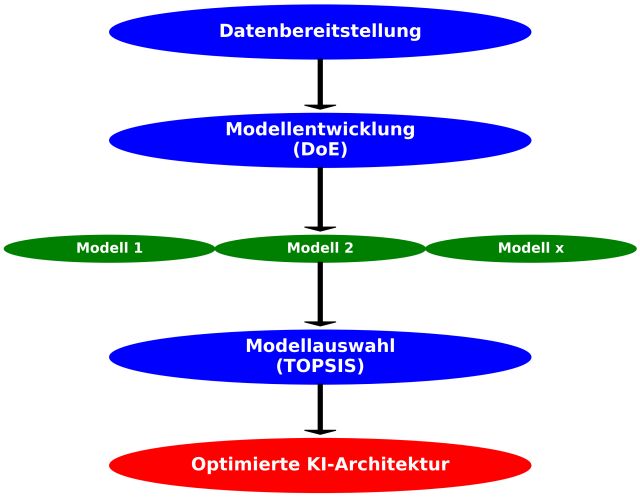


Abbildung 19: Visualisierung der Modellentwicklung

Die Modellentwicklung wird in folgende Phasen unterteilt:

- **Datenbereitstellung**: Diese Phase befasst sich mit der Sammlung und Aufbereitung der notwendigen Daten für die Modellentwicklung.

- **Modellentwicklung (DoE)**: Hier werden verschiedene Architekturen entwickelt, um potenzielle Kandidaten für die optimale Lösung zu generieren.
- **Modellvarianten**: Diese Phase stellt die verschiedenen erstellten Architekturen dar.
- **Modellauswahl**: Die TOPSIS-Analyse wird genutzt, um aus den entwickelten Modellen das optimale bezogen auf die Vorhersage des Durchsatzes zu bestimmen.
- **Optimierte KI-Architektur**: Das Ergebnis des Prozesses ist die Auswahl der optimierten Modellarchitektur, das sich in der finalen Phase herauskristallisiert.

Die Pfeile zwischen den Ebenen veranschaulichen den Informationsfluss und die Prozessabfolge von der Datenbereitstellung bis zur Auswahl der optimierten Modellarchitektur.

5.3.1 Design of Experiments für Gestaltung der KI

Das Design of Experiments (DoE) ist eine etablierte statistische Methode, die in der Planung, Durchführung, Analyse und Interpretation von kontrollierten Experimenten verwendet wird. Ziel ist es, die Einflüsse verschiedener Faktoren auf eine Zielvariable zu evaluieren. Im Kontext der Gestaltung einer KI-Architektur umfassen diese Faktoren typischerweise diverse Hyperparameter des Modells, unterschiedliche Datenverarbeitungstechniken, Regularierungstechniken, Aktivierungsfunktionen und verschiedene ANN-Typen. Dabei werden die Eingabevariablen (wie Hyperparameter und Modellkonfigurationen) systematisch variiert, um deren Einfluss auf die Modellleistung zu verstehen. Die Umsetzung des DoE folgt mit diesen Schritten:

1. **Definition der Faktoren**: Festlegung der zu variierenden Faktoren, ANN-Typ, Anzahl Schichten, Neuronen, der Architektur, Aktivierungsfunktion etc.
2. **Auswahl des Designs**: Bestimmung des experimentellen Designs, also der Art und Weise, wie die Faktoren systematisch verändert werden.
3. **Durchführung der Experimente**: Ausführung der Experimente entsprechend des gewählten Designs.
4. **Datenanalyse**: Analyse der gesammelten Daten, um die Effekte der einzelnen Faktoren und ihrer Interaktionen zu bestimmen.
5. **Optimierung**: Anwendung der Analyseergebnisse zur Bestimmung der optimalen Modellkonfigurationen.

Durch Einsatz von DoE wird schrittweise anhand von Zwischenergebnissen optimiert. Dies trägt zur Verbesserung der Modellleistung bei und vermeidet überflüssige Experimente bei denen alle möglichen Kombinationen von Faktoren getestet werden.

5.3.2 KI-Modellauswahl mit TOPSIS

Die TOPSIS-Analyse ist ein Multi-Kriterien-Entscheidungsfindungsverfahren, das darauf abzielt, das optimale Produkt oder System aus einer Reihe von Alternativen auszuwählen, die basierend auf verschiedenen Kriterien bewertet werden [220]. Bei der Entwicklung neuer ANN für die Optimierung von Fabriklayouts resultiert eine Reihe von Modellen mit unterschiedlichen Leistungsmerkmalen. Die Auswahl des besten Modells erfolgt systematisch mittels TOPSIS. Das Verfahren basiert darauf, dass das optimale Modell die kürzeste Distanz zur idealen und die längste Distanz zur schlechtesten Lösung aufweist. Dies ermöglicht ganzheitliche Bewertung der Modellleistung über verschiedene Metriken hinweg. Innerhalb der Fabriklayoutoptimierung werden Metriken wie Genauigkeit der Vorhersagen, Rechenzeit und Modellkomplexität als Bewertungskriterien herangezogen. Diese Kriterien werden jeweils gewichtet, um ihre Bedeutung in der Gesamtbewertung widerzuspiegeln, wobei besonderes Gewicht auf die Genauigkeit der Vorhersagen und die Fähigkeit zur Generalisierung gelegt wird. Die ideale Lösung repräsentiert den Datenpunkt mit der besten Leistung in allen Kriterien, während die negative ideale Lösung die schlechteste Leistung zeigt. Nach Identifikation dieser Lösungen wird die geometrische Distanz jedes Modells zu diesen Punkten berechnet. Das Modell, das die kürzeste Distanz zur idealen Lösung und die längste zur negativen idealen Lösung aufweist, wird als das Beste ausgewählt.

5.3.3 Ergebnisse der optimierten Modellauswahl

Die in Abbildung 20 dargestellte Auswertung zeigt die Ergebnisse nach einer Reihe von Experimenten.

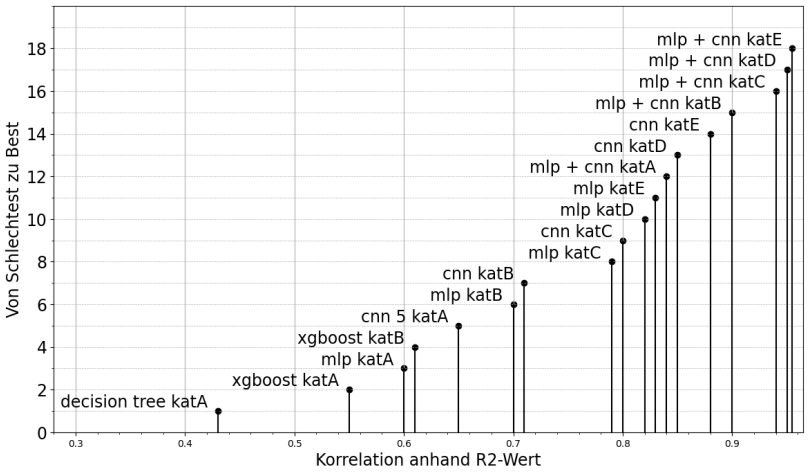


Abbildung 20: Performance der Modelle nach r^2-Wert

In der Abbildung sind die erzielten r^2-Werte von insgesamt 18 Modellen bezogen auf den

simulierten Durchsatz von Fabriklayouts mit sechs OE dargestellt. Die Modelle sind nach dem r^2-Wert vom schlechtesten bis zum besten Modell aufsteigend sortiert. Zusätzlich wurden Parameter der Architektur kategorisiert und die Effekte verschiedener Konfigurationen auf die Modelle erfasst. Nachfolgende Tabelle 3 zeigt die vorgenommenen Kategorisierungen im DoE:

Tabelle 4: Kategorien und Aufschlüsselung zum DoE

Kategorie	**Aufschlüsselung**
katA	5 Schichten oder 50 Bäume und 50.000 Layouts
katB	10 Schichten oder 100 Bäume und 50.000 Layouts
katC	10 Schichten und 100.000 Layouts
katD	15 Schichten und 100.000 Layouts
katE	15 Schichten und 150.000 Layouts

In die Untersuchung sind fünf ML-Architekturen eingeflossen; Decision Trees (DT), Random Forest, XGBoost, MLP, CNN und eine Kombination aus den beiden letztgenannten Architekturen. Es wurden fünf Kategorisierungen (katA bis katE) definiert, die sich bezüglich der Anzahl der Schichten bei neuronalen Netzwerken, Anzahl der Bäume bei Ensemble-Algorithmen und Anzahl der verwendeten Trainingsdaten unterscheiden. Im Ergebnis der ersten Runde (katA) zeigen DT und XGBoost einen kleinen positiven Effekt bei Erhöhung der Datenmenge. Größere Effekte erzielen das MLP und CNN. Da die DT die niedrigsten Werte in der ersten Kategorie erzielt haben, wurden sie in den weiteren DoE-Versuchsreihen nicht betrachtet. In der zweiten Runde wurde die Anzahl der Schichten auf 10 bzw. die Anzahl der Bäume auf 100 erhöht (katB). XGBoost kann auch durch Erhöhung der Baumanzahl nicht in die Wertebereiche von MLP und CNN vordringen. Ab der dritten Runde werden daher nur noch MLP und CNN sowie die Kombination aus beiden betrachtet.

Über alle Runden zeigte eine Kombination aus MLP und CNN die konstant höchsten Werte somit ist dieses ANN mit der besten Eignung für die Abbildung des durch den Datensatz dargestellten Problems identifiziert. Die durchgeführten Versuche zeigen die Relevanz der Auswahl einer angepassten ANN-Architektur auf. Darüber hinaus ist ein Potenzial zu weiteren Leistungssteigerungen bei Verwendung größerer Datenmengen erkennbar. Bei entsprechend großen Rechenressourcen ist die Entwicklung eines Modells denkbar, das über die in Kapitel 3.1.1 vorgestellten Typen von FLP generalisieren kann. Aufgrund verfügbarer Rechenressourcen und Kostenerwägungen sind die Modelle der katC im Hinblick auf die Performance geeignet für den Einsatz im FLP, siehe gelb markierte Zeile in Tabelle 3.

In weiterführenden Experimenten zum DoE sind zudem Hyperparameter, Aktivierungsfunktionen, Lernrate und die Batch-Größe eingeflossen. Diese Untersuchungen dienen allerdings nicht mehr der optimierten Modellfindung, sondern der Modelloptimierung und

werden in Kapitel 6.8.2 beschrieben.

5.3.4 Architektur des künstlichen neuronalen Netzwerks

Abbildung 21 zeigt die Architektur des verwendeten ANN, das über iterative Entwicklungs- und Evaluationsphasen optimiert wurde, vgl. Kapitel 5.3.3.

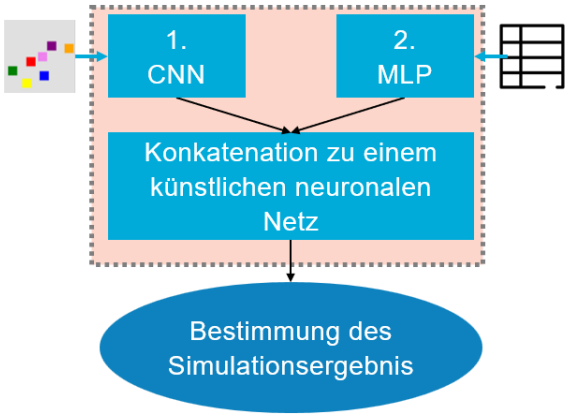


Abbildung 21: Architektur des ANN

Das ANN besteht aus zwei Typen: einem auf CNN basierenden Netzwerk, das Bildinformationen empfangen und verarbeiten kann, und einem zusätzlichen auf MLP basierenden Netzwerk zur Integration von Informationen aus tabellarischen Daten. Diese Konfiguration bietet Vorteile in der Fähigkeit des neuronalen Netzwerks, sich auf verschiedene Layoutkonfigurationen zu generalisieren. Die Eingabeschicht ist nicht auf eine vorher festgelegte Anzahl von Neuronen beschränkt; vielmehr passt sie sich inhärent an das Bildformat der Eingaben an. Diese Flexibilität ermöglicht die Umwandlung aller Bilder in dieses spezifizierte Format und unterstützt Konfigurationen mit variierender Anzahl von OE, die im jeweiligen Bild dargestellt werden. Solange das Bild das korrekte Format hat, ist es ein gültiger Eingabewert für das CNN. Andererseits hat der MLP-Teil des Netzwerks konstruktionsbedingt eine Eingabeschicht mit einer festgelegten, fixen Dimension. Um die Generalisierungsfähigkeit des CNN zu erhalten, enthalten die zusätzlichen tabellarischen Daten n-Platzhalter, die eine Skalierung der Anzahl der OE ermöglichen. Beispielsweise sind in einem Layout mit der maximalen Fabrikkonfiguration von acht OE alle Platzhalter ausgefüllt; in einem Layout mit fünf OE bleiben drei Platzhalter unbesetzt. Die Implementierung der KI-Architektur ist in Kapitel 6.6 erläutert.

5.3.5 Konzeption zum Training des neuronalen Netzwerks

Das grundsätzliche Trainingskonzept des ANN entspricht Abbildung 22.

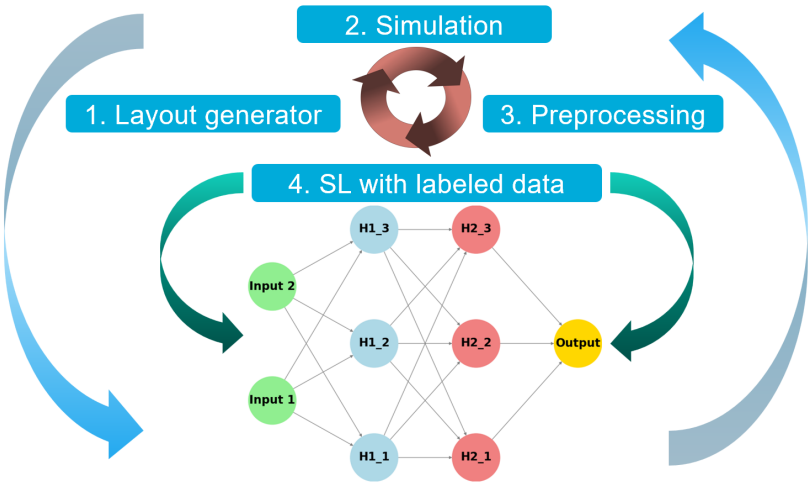


Abbildung 22: Grundkonzept zur KI-unterstützten Bestimmung des Durchsatzes

Die erste Phase beinhaltet den Layoutgenerator zur Generierung einer beliebigen Anzahl von Layoutvarianten. In der zweiten Phase wird der Durchsatz der Layouts durch Simulation bestimmt. In der dritten Phase werden die erstellten Daten bereinigt, siehe Kapitel 5.2.3. In der vierten Phase werden diese in einem überwachten Lernsatz dem ANN zugeführt, s. voriges Kapitel. Das Zusammenspiel dieser drei Komponenten ermöglicht das Training des ANN mit dem Ziel, die Simulation von Layouts abzubilden.

Der blau markierte innere Kreislauf skizziert hierbei den Prozess zur Erstellung von Trainingsdaten, beginnend mit der Generierung von Layouts und gefolgt von dem *Labeling* durch die Simulation. Der resultierende Datensatz wird vorverarbeitet und dient sowohl als Ein- und Ausgabe für das ANN in einem überwachten Lernansatz. Die äußeren Pfeile deuten einen zusätzlichen Zyklus an, der das Transferlernen darstellt. Anfangs wird das ANN an einem Datensatz mit niedrigerer Komplexität trainiert, der Layouts mit vier OE umfasst. In nachfolgenden Iterationen wird das vortrainierte Netzwerk schrittweise mit komplexeren Datensätzen trainiert. Dieses Verfahren wird im Rahmen der durchgeführten Experimente fünfmal wiederholt und mit einem Datensatz, der Layouts mit acht OE enthält abgeschlossen.

5.4 Phase 3 zur Integration des ANN in einen GA

Ein ANN mit der erläuterten Struktur kann in metaheuristische Algorithmen integriert werden, um die Layoutbewertung anstelle einer Simulation zu erfüllen. Die folgende Abbildung 23 zeigt die Konzeption zur Integration des ANN in einen genetischen Algorithmus.

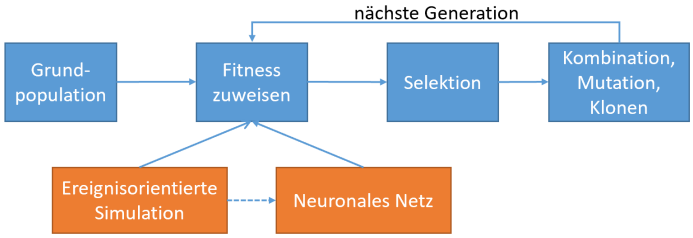


Abbildung 23: Konzeption zur KI-Integration in einen GA

In der dargestellten Konzeption wird ein ANN in einen GA integriert, um die Bewertung von Populationen effizienter im Vergleich zur traditionellen Simulation durchzuführen. Ein verwandter Ansatz wurde in [62] erforscht und in Kapitel 4.3 diskutiert. Die KI-basierte Bewertung bietet im Vergleich zur Simulation mehrere Vorteile: Sie ist schneller und ermöglicht die Einbeziehung zusätzlicher Randbedingungen bei einer gleichzeitigen Berücksichtigung einer größeren Anzahl von OE. Darüber hinaus kann ein trainiertes ANN für verschiedene oder unvollständige Fabrikkonfigurationen eingesetzt werden.

Die Integration in einen GA kann zudem auf zweierlei Weise für eine fortlaufende Validierung der KI genutzt werden. Erstens ermöglicht der Vergleich der von der KI gelieferten Bewertungen mit den durch Simulation erzeugten Bewertungen eine kontinuierliche Beurteilung der Präzision. Indem beide Bewertungsmethoden während einer GA-Iteration gegenübergestellt werden, können Abweichungen der KI von den simulierten Ergebnissen erkannt und ggf. für RL zur weiteren Verbesserung verwendet werden. Zweitens bietet der Vergleich kompletter Durchläufe des GA, die entweder ausschließlich auf Simulation oder KI-Bewertungen basieren, eine Einschätzung zur KI als Bewertungsfunktion. In einem Versuchsaufbau wird das jeweils beste Layout zum Zeitpunkt der Konvergenz gegenübergestellt. Diese Analyse prüft neben der Fähigkeit zu Einzelbewertungen, ob im Kontext des gesamten GA konsistente Ergebnisse geliefert werden.

Ablauf der Integration

Zu Beginn des Optimierungslaufs werden 100 Prozent aller Layouts durch die Simulation bewertet. In den folgenden Generationen wird ein initial geringer Anteil der Layouts durch das neuronale Netz bewertet. Bei einer Übereinstimmung der Bewertung von Simulation und neuronalem Netz wird der Simulationsanteil sukzessive verkleinert und der ANN-Anteil vergrößert, bis die Fitness nahezu ausschließlich durch das ANN bestimmt werden kann. Hierauf aufbauend erfolgt eine Erweiterung des in Kapitel 3.3.1 vorgestellten Ablaufs eines GA:

- **Erste Iteration:**
  1. *Initialisierung* einer Grundpopulation (Generiere 1000 Layouts).

2. Bestimme Durchsatz jedes Layouts (Individuum) mittels Simulation (Sollwert für KI).
3. Nutze die generierten Layouts und die Simulationsergebnisse, um das ANN zu validieren und trainieren.
4. *Selektion*, die besten 50 % Individuen gemäß Simulation überleben.
5. *Kombination* und *Mutation* der Lösungsmöglichkeiten, um wieder die Anzahl der Grundpopulation zu erreichen.

- **Fortlaufende Iterationen bis zur Konvergenz:**
  1. Bestimme Simulationsanteil anhand des vorherigen Validierungsergebnisses.
  2. Bestimme Durchsatz nach aktualisiertem Teilungsverhältnis von DES und ANN.
  3. Sortiere die Individuen (Layouts) anhand des Durchsatzes.
  4. Nutze die Individuen (Layouts) und die Simulationsergebnisse um das ANN weiter zu trainieren + Soll-Ist-Vergleich.
  5. Selektiere die Individuen nach dem simulierten Durchsatz (wenn vorhanden) und ansonsten mit dem vom ANN bestimmten Durchsatz.
  6. Nutze die Überlebenden, besten y-Prozent (z.B. 20 %) der Individuen, um durch Kombination und Mutation die Größe wiederherzustellen.

Ein Vorteil dieses Vorgehens ist das kein Zugriff auf einen bestehenden Datenbestand notwendig ist, da die Simulation als Supervisor eingesetzt wird und sukzessive durch das ANN (Estimator) ersetzt wird. Die Implementierung des GA ist in Kapitel 6.7 dokumentiert.

5.5 Diskussion zur Auswahl des Ansatzes

Die Entscheidung zur Anwendung einer Monte-Carlo-Simulation bei der Generierung von Daten basiert auf der Fähigkeit, die Komplexität und Variabilität realer Fabrikumgebungen zu berücksichtigen. Anschließend wird eine ereignisorientierte Simulation zur Ermittlung objektiver Leistungsmetriken durchgeführt. Diese dienen als Labels für die Trainingsdaten, die verwendet werden um die Vorteile des überwachten Lernens zu explorieren. Das ANN kombiniert Bild- und tabellarische Datenverarbeitung zur Bestimmung der Simulationsergebnisse. Dies ermöglicht Suchräume weitläufig zu erkunden, die durch zunehmende Anzahl an OE oder zusätzliche Randbedingungen exponentiell wachsen. Die rechenintensive DES wird durch die effizientere Evaluationsfähigkeit des ANN kompensiert, hierdurch sinkt mit jeder Iteration des GA der Rechenaufwand. Dieser Vorteil könnte KI letztlich für die Layoutplanung nutzbar machen, indem durch Reduzierung der Rechenauslastung mehr Randbedingungen als bei bisherigen Verfahren inkludierbar werden.

6 Prototypische Realisierung

In diesem Kapitel werden die Implementierung der einzelnen Komponenten (siehe Kapitel 6.1 bis Kapitel 6.7) sowie der experimentelle Einsatz (siehe Kapitel 6.8) erläutert.

6.1 Technische Implementierung

Neben dem primären Ziel, die Forschungsfrage dieser Arbeit zu beantworten, fungiert die technische Implementierung als experimentelle Plattform, die es ermöglicht, Thesen und Erkenntnisse zur weitergehenden Erforschung des FLP abzuleiten. Der Fokus der Beobachtung liegt hierbei auf Stärken und Schwächen verschiedener Algorithmen unter kontrollierten Bedingungen. Ferner bietet die Implementierung eine Grundlage für die Realisierung von Tools, die zur Lösung von realen Fabriklayoutproblemen eingesetzt werden können. Die Behandlung der Forschungsfrage wird nicht als abschließendes Ziel betrachtet, sondern als kontinuierlicher Prozess mit dem Ziel das Wissen in diesem Bereich zu erweitern und weiterführende Forschungsfragen zu generieren. Es folgen die grundsätzlichen Vorüberlegungen und Entscheidungen zur technischen Implementierung.

6.1.1 Vorüberlegungen zur Implementierung

Die Entscheidung, die notwendigen Algorithmen selbst zu entwickeln, statt kommerzielle Software zu nutzen, wurde aufgrund folgender Kriterien getroffen:

- **Anpassungsfähigkeit**: Eigene Entwicklungen ermöglichen maßgeschneiderte Lösungen, die spezifisch auf die Forschungsfragen abgestimmt sind. Sie bieten die nötige Flexibilität, um auf Herausforderungen, die während der Experimente auftreten, zu reagieren. Im Gegensatz dazu sind kommerzielle Softwarelösungen in ihrem Funktionsumfang festgelegt und weniger flexibel in der Anpassung an spezifische Forschungsbedürfnisse.
- **Schnittstellen**: Bei Entwicklung von hybriden Architekturen müssen diese in der Lage sein miteinander Daten auszutauschen. In der vorgestellten Architektur erfolgt ein Datenaustausch zwischen dem Layoutgenerator, der Simulation und dem ANN. Die Eigenentwicklung ermöglicht die Kontrolle über den Funktionsumfang, den verwendeten Formaten und Schnittstellen.
- **Transparenz**: Die eigene Implementierung erhöht die Transparenz der verwendeten Methoden und Algorithmen. Bei kommerzieller Software ist dies nur eingeschränkt

gegeben. Der entwickelte Quellcode kann auf Plattformen wie GitHub[15] veröffentlicht werden, zur Erhöhung der Zugänglichkeit und Überprüfbarkeit.

- **Forschungsfokus**: Durch die Verwendung selbst entwickelter Methoden wird ein tiefgreifendes Verständnis der Problemstellungen und der angewendeten Lösungsstrategien ermöglicht. Kommerzielle Lösungen sind häufig nicht auf die spezifischen Anforderungen wissenschaftlicher Fragestellungen ausgerichtet.
- **Innovationspotenzial**: Die Entwicklung eigener Lösungsansätze bietet die Möglichkeit, innovative Techniken zu erforschen, die über den aktuellen Stand der Technik hinausgehen können. Hier wird mehr der Fokus auf die Untersuchung anstelle der Performance entwickelter Algorithmen gelegt.
- **Kostenaspekte**: Die Verwendung eigener Entwicklungen vermeidet hohe Kosten für Softwarelizenzen und Schulungen, die mit kommerziellen Produkten verbunden sein können.

6.1.2 Auswahl der Programmiersprache und Entwicklungsumgebung

Die Entscheidung für eine geeignete Programmiersprache und eine passende Entwicklungsumgebung basiert auf Kriterien wie Skalierbarkeit, Einsetzbarkeit in verschiedenen Umgebungen, Unterstützung spezifischer Algorithmen und Funktionen sowie Benutzerfreundlichkeit. Aufgrund dieser Anforderungen erwies sich Python als geeignete Wahl, besonders hervorgehoben durch seine umfassende Unterstützung für maschinelles Lernen. Im Folgenden sind die relevanten Überlegungen dargestellt:

- **Open-Source-Verfügbarkeit**: Python, als Open-Source-Programmiersprache, gewährleistet Zugänglichkeit und Transparenz. Dies wird durch eine aktive Entwicklergemeinschaft unterstützt, die kontinuierlich zur Verbesserung der Sprache beiträgt.
- **Gemeinschaft und Ressourcen**: Python genießt eine robuste und aktive Gemeinschaft, präsent auf Plattformen wie Stack Overflow und GitHub. Diese Gemeinschaft stellt eine umfangreiche Wissensbasis zur Verfügung, die sowohl Neulingen als auch erfahrenen Entwicklern zugutekommt.
- **Effiziente Entwicklung**: Python erleichtert die Implementierung komplexer Algorithmen. Die Bibliothek *Scikit-learn* bietet zum Beispiel eine umfangreiche Palette maschineller Lernmethoden (u. a. Random Forests, Grid Search), die direkt genutzt werden können.
- **Nachhaltigkeit und Benutzerfreundlichkeit**: Die klare und einfache Syntax von Python steigert die Produktivität durch minimierten Schreibaufwand und erleichterte Fehlersuche zur Erhöhung der Wartbarkeit und Weiterentwicklung des Codes.

[15] GitHub ist eine Plattform für die Versionsverwaltung und Kollaboration in der Softwareentwicklung, s. https://github.com.

- **Plattformübergreifende Verfügbarkeit**: Python ist eine plattformunabhängige Sprache, die auf einer Vielzahl von Betriebssystemen und Entwicklungsumgebungen lauffähig ist. Dies bietet Flexibilität bei der Wahl der Arbeitsumgebung.
- **Spezialisierte ML-Unterstützung**: Python wird in der Community des maschinellen Lernens breit unterstützt, unter anderem durch Bibliotheken wie TensorFlow und PyTorch, die spezialisierte Funktionen für das Training von neuronalen Netzwerken bereitstellen.

6.1.3 Vergleich zwischen Python, R und C++

Die Entscheidung für eine Programmiersprache berücksichtigte Alternativen wie R und C++. Erstere ist für seine statistischen Funktionen und grafischen Darstellungsmöglichkeiten bekannt, die es für Datenanalyse und Visualisierung prädestinieren. Dennoch bietet Python eine größere Flexibilität und eine breitere Unterstützung verschiedener Programmierparadigmen sowie eine umfassende Bibliotheksvielfalt, insbesondere für maschinelles Lernen.

C++ zeichnet sich durch seine Leistungsfähigkeit aus, die es geeignet für ressourcenintensive oder zeitkritische Anwendungen macht. Der Entwicklungsprozess in C++ ist jedoch komplexer und zeitaufwendiger im Vergleich zu Python. Dies ist auf das Fehlen einer automatischen Speicherverwaltung und eine weniger intuitive Syntax zurückzuführen.

Python wurde letztendlich aufgrund seiner Vielseitigkeit, der starken Unterstützung durch eine aktive Entwicklergemeinschaft, dem Fokus auf Forschung statt Performance und seiner umfangreichen Bibliotheken für maschinelles Lernen als die geeignetste Sprache für diese Arbeit ausgewählt.

6.1.4 Auswahl des KI-Frameworks

Bei der Auswahl des passenden KI-Frameworks wurde eine Bewertung verschiedener Bibliotheken durchgeführt. Abbildung 24 illustriert eine Analyse von Andrej Karpathy zu den Erwähnungen von ML-Frameworks in arXiv[16]-Artikeln, basierend auf 43.000 Publikationen zwischen 2014 und 2018.

[16] arXiv ist ein Online-Archiv und Vertriebsdienst für wissenschaftliche Artikel in den Bereichen Physik, Mathematik, Informatik, quantitativer Biologie, quantitative Finanzwissenschaften und Statistik, die vor ihrer offiziellen Veröffentlichung in wissenschaftlichen Zeitschriften eingereicht und zugänglich gemacht werden, s. `https://arxiv.org`.

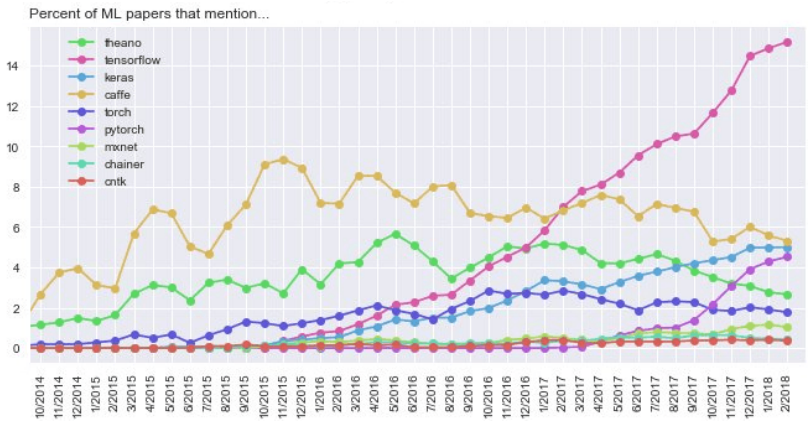


Abbildung 24: Trend zum Einsatz von ML-Frameworks [221]

Die Analyse hebt *Tensorflow, Caffe, Keras* und *PyTorch* als die am häufigsten verwendeten Frameworks hervor. Um ein geeignetes Framework für die wissenschaftliche Fragestellung dieser Dissertation zu identifizieren, wurde die folgende Bewertung durchgeführt.

6.1.4.1 Gegenüberstellung der Machine-Learning Frameworks

Im Folgenden werden Tensorflow, Keras und Pytorch gegenübergestellt.

TensorFlow

TensorFlow wurde 2015 vom Google Brain Team veröffentlicht und ist eine Open-Source-Softwarebibliothek für numerische Berechnungen, die vorrangig für ML und DL konzipiert ist [222]. Diese Bibliothek dient als umfangreiche und flexible Plattform für die Entwicklung und Umsetzung von Algorithmen sowie Modellen. Der Name „TensorFlow“ leitet sich von den Tensoren ab, den zentralen Datenstrukturen der Bibliothek. Tensoren sind Verallgemeinerungen von Vektoren und Matrizen auf höhere Dimensionen und repräsentieren mehrdimensionale Datenarrays. Diese Struktur ermöglicht die effiziente Verarbeitung komplexer Datensätze wie beispielsweise Bildern. Ein Bild kann als Tensor dargestellt werden, dessen Dimensionen typischerweise Höhe, Breite und Farbkanäle umfassen. Tensoren sind von grundlegender Bedeutung für das Design und die Implementierung komplexer ML-Modelle, da sie die Beziehungen zwischen verschiedenen Datenelementen in mehrdimensionalen Räumen darstellen können [117].

Eine Schlüsselfunktion von TensorFlow ist die Nutzung eines statischen Berechnungsgraphen, der die Abhängigkeiten zwischen Operationen abbildet und eine Verteilung und Parallelisierung von Berechnungen über mehrere Central Processing Unit (CPU) oder GPU ermöglicht. Diese Architektur führt zu optimierten Berechnungen und effizienter Ressourcennutzung. Mit der Einführung von TensorFlow 2.0 wurde zudem der „eager execution“

Modus implementiert, der die Unterstützung für dynamische Berechnungsgraphen bietet und somit eine flexiblere Entwicklungsumgebung schafft.

Keras

Keras ist eine im Jahr 2015 von François Chollet veröffentlichte Open-Source-Deep-Learning-Bibliothek [223]. Ziel von Keras ist es, das Erstellen und Trainieren von ANN so zugänglich und benutzerfreundlich wie möglich zu gestalten. Keras dient als Schnittstelle zu leistungsfähigeren Deep-Learning-Frameworks wie TensorFlow, wobei TensorFlow das bevorzugte Backend ist. Keras kann jedoch auch mit Theano zusammenarbeiten, einem Framework, das sich durch seine Effizienz bei der Ausführung von Berechnungen auf mehrdimensionalen Arrays auszeichnet. Diese Modularität ermöglicht es Keras, ein breites Spektrum von Anwendungen zu unterstützen, von einfachen Modellierungen bis hin zu komplexen Forschungsprojekten.

PyTorch

PyTorch, eine weitere Open-Source-Machine-Learning-Bibliothek, wurde im Oktober 2016 von *Facebooks AI Research Lab* veröffentlicht, um den wachsenden Bedarf an einem flexibleren Framework für wissenschaftliche Berechnungen zu erfüllen [224]. PyTorch ist in Python und C++ geschrieben und stellt eine Weiterentwicklung der vorher etablierten Bibliothek Torch dar, die in Lua geschrieben ist. PyTorch unterstützt dynamische Berechnungsgraphen, die während der Laufzeit konstruiert werden. Dieser Ansatz erleichtert iterative Entwicklungsprozesse und die Implementierung komplexer Architekturen.

Caffe

Caffe, entwickelt von Yangqing Jia und veröffentlicht durch das Berkeley Vision and Learning Center im Jahr 2014, ist eine Open-Source-Deep-Learning-Bibliothek, die sich durch ihre Geschwindigkeit bei der Modellierung und Skalierbarkeit auszeichnet [225]. Caffe ist besonders bekannt für seine Effizienz in der Bildklassifikation und anderen bildbasierten Verarbeitungsaufgaben. Die Bibliothek bietet eine modulare Architektur, die es Forschern und Entwicklern ermöglicht, schnell und einfach neuronale Netze zu entwerfen, zu trainieren und zu implementieren. Caffe unterstützt viele Arten von DL-Architekturen, insbesondere CNN, und ist aufgrund seiner Leistung und Genauigkeit bei der Bildverarbeitung besonders beliebt. Des Weiteren ermöglicht das Framework ein Training der Netzwerke sowohl auf der CPU als auch auf der GPU.

6.1.4.2 Bewertung der Machine-Learning-Frameworks

Bei der Bewertung der verschiedenen ML-Frameworks wurden Berechnungsgraphen, Schnittstellen (Application Programming Interface (API)), Flexibilität, Skalierbarkeit, Unterstützung komplexer Netzwerkarchitekturen sowie der Community-Support berücksichtigt. Tabelle 5 zeigt einen Vergleich dieser Kriterien für die betrachteten Bibliotheken PyTorch, TensorFlow, Keras und Caffe.

Tabelle 5: Vergleich der Machine-Learning-Bibliotheken

Kriterium	**PyTorch**	**TensorFlow**	**Keras**	**Caffe**
Hersteller	Facebook's AI Research lab	Google Brain Team	Francois Chollet (Google)	Berkeley AI Research
Sprache	Python, C++	Python, C++	Python	Python, C++
Design-Philosophie	Komplex, flexibel	High- und Low-Level APIs, flexibel	Einfach, auf TensorFlow aufbauend	Schnell, für CNNs optimiert
Berechnungs-graph	Dynamisch	Statisch	Statisch	Statisch
API-Klarheit	Hoch	Mittel	Hoch	Mittel
Architektur	Komplex	Anspruchsvoll	Einfach	Komplex
Flexibilität	Hoch	Mittel	Mittel	Hoch
Backend	ATen (Tensor Bibliothek)	Eigen, NVIDIA cuDNN	TensorFlow	cuDNN, Intel MKL, OpenBLAS
Skalierbarkeit	Hoch	Hoch	Hoch	Hoch
Geschwindig-keit	Hoch	Hoch	Variiert (abhängig von TensorFlow)	Hoch (insbesondere für CNNs)
Datasets	Hoch	Hoch	Hoch	Klein
Community-Support	Hoch	Hoch	Hoch	Mittel

Nach diesem Vergleich wird für die weitere Implementierung PyTorch aufgrund der nachfolgenden Überlegungen festgelegt.

Erstens bietet PyTorch dynamische Berechnungsgraphen, die während der Laufzeit erstellt werden. Diese Eigenschaft fördert die Flexibilität in einer Forschungsumgebung mit häufigen Änderungen am Modell und experimentellen Design. Das FLP und dessen Anwendung für KI erfordern ein iteratives experimentelles Vorgehen. **Zweitens** ist die PyTorch-API intuitiv und umfassend dokumentiert. Dies erleichtert den Umgang mit der Bibliothek und den Aufbau von Prototypen. **Drittens** unterstützt PyTorch eine Vielzahl von Modellen, einschließlich faltender und rekurrenter neuronaler Netze. Diese umfangreiche

Modellunterstützung erweitert die Möglichkeiten zur Lösung des gegebenen Problems. **Viertens** zeichnet sich PyTorch durch eine starke Community-Unterstützung aus, die sich in einer Vielzahl an Tutorials, Dokumentationen und Forenbeiträgen zeigt. Die aktive PyTorch-Community kann dazu beitragen, auftretende Schwierigkeiten zu überwinden und „Best Practices“ anzuwenden.

Es sei erwähnt, dass auch Keras und Tensorflow für dieses Forschungsprojekt in Betracht gezogen wurden. Keras ermöglicht schnelle Prototypenentwicklung und ist ebenfalls gut dokumentiert. Jedoch bietet Keras weniger Flexibilität bei der Erstellung nicht standardmäßiger Modelle. Tensorflow bietet trotz der Verbesserungen in seiner aktuellen Version 2.0 keine spezifischen Vorteile für dieses Forschungsvorhaben. Der zentrale Vorteil von Tensorflow bleibt die Verwendung statischer Berechnungsgraphen, die vor der Ausführung eines Programms vollständig definiert werden. Während dieser Ansatz Vorteile in Bezug auf Leistung und Effizienz bietet, kann er sich als ein Nachteil in Forschungs- und Entwicklungsumgebungen erweisen, in denen Flexibilität und Iteration wichtig sind.

Zusammenfassend erwies sich PyTorch nach einer Abwägung der spezifischen Anforderungen und Prioritäten dieser Dissertation als geeignete Wahl. PyTorch kombiniert Flexibilität und Unterstützung mit einer intuitiven API und einer umfangreichen, frei zugänglichen Wissensbasis.

6.1.5 Technische Informationen zur Ausführung des Quellcodes

Der in dieser Arbeit vorgestellte Quellcode erfordert spezifische systemtechnische Voraussetzungen und Abhängigkeiten. Er ist in Python, Version 3.11.1, geschrieben und getestet. Die verwendeten Bibliotheken sind Open-Source und die genauen Versionen werden in den nachfolgenden Kapiteln und im Anhang beschrieben. Python bietet grundsätzlich Plattformunabhängigkeit, jedoch können spezifische Funktionen die Ausführung auf bestimmte Systeme beschränken. Der Code wurde auf einem Windows 11 System in einer JupyterLab Umgebung (Version 4.0.2) entwickelt und getestet. Die Systemanforderungen umfassen:

- Mindestens 16 GB Arbeitsspeicher,
- Eine dedizierte Grafikkarte mit mindestens 8 GB VRAM,
- Ein Prozessor vom Typ Intel Core i5-11400 alternativ AMD Ryzen 5 3600 oder neuer.

Diese Konfiguration gewährleistet eine optimale Leistung für die Erzeugung von Layouts, die Durchführung von Simulationen und das Training der KI.

6.2 Implementierung des Layoutgenerators

Der im Folgenden vorgestellte Algorithmus erstellt eine spezifizierte Anzahl von quadratischen Bildlayouts auf einem weißen Hintergrund. Die Quadrate repräsentieren OE, wie

Maschinen sowie Ein- und Ausgangslager. Der Hintergrund ist als Matrix organisiert, wobei jeder Pixel einen RGB-Wert zur Farbdarstellung besitzt. Die Platzierung der Fabrikeinheiten basiert auf einer Monte-Carlo-Simulation. Ziel ist es, eine Anordnung zu erstellen, in der die Quadrate sich weder überlappen noch einen festgelegten Mindestabstand unterschreiten. Jede OE wird durch die Koordinaten ihrer oberen linken Ecke definiert. Eine Kollision wird verhindert, indem jede neue OE-Position mit den Positionen bereits platzierter OE verglichen wird. Überlappt sich eine Position oder liegt sie zu nahe an einer anderen, wird sie verworfen und eine neue Position zufällig generiert. Dieser Prozess wiederholt sich, bis eine gültige Position für alle OE gefunden ist. Der Begriff „Monte Carlo“ bezieht in diesem Zusammenhang sich auf das stochastische Verfahren, das zufällige Positionen generiert und auf Kollisionen überprüft, um ein einzigartiges Layout zu erreichen. Für die Implementierung werden folgende Open-Source Python-Bibliotheken eingesetzt:

```
import random
from PIL import Image, ImageDraw
from tqdm import tqdm
import math
```

- **import random**: Das `random`-Modul ist eine in Python eingebaute Bibliothek, die Funktionen zur Erzeugung von Zufallszahlen bereitstellt. Sie wird hier verwendet, um zufällige Positionen für die Quadrate innerhalb der Layouts zu generieren.
- **from PIL import Image, ImageDraw**: Die Python Imaging Library (PIL) ist ein Open-Source-Werkzeug, das umfangreiche Unterstützung für Bildformate, effiziente interne Darstellungen und Funktionen zur Bildbearbeitung bietet. Im vorliegenden Code wird `PIL` eingesetzt, um Bilder zu erzeugen und zu manipulieren.
- **from tqdm import tqdm** (optional): Die `tqdm`-Bibliothek wird zur Visualisierung des Fortschritts in Schleifen verwendet. Hier wird sie zur Überwachung des Fortschritts bei der Erstellung der Layouts genutzt.
- **import math**: Die `math`-Bibliothek stellt eine Sammlung mathematischer Funktionen zur Verfügung, die hier zur Berechnung der euklidischen Distanz zwischen den Layout-Quadraten genutzt wird.

Folgende globale Variablen definieren die Randbedingungen des Layoutgenerators. Über diese können die Anzahl der Maschinen, die Layoutgröße, die Größe der Maschinen, die Anzahl verschiedener Layouts, die Distanzen zwischen den Layouts und der Abstand zum Rand festgelegt werden.

```
# Definition der Hintergrundgroessee
BG_SIZE = (400, 400)

# Definition der Quadratgroesse
SQUARE_SIZE = 40
```

```
# Max Anzahl von Versuchen, ein einzigartiges Layout zu erstellen
MAX_ATTEMPTS = 20
NUM_LAYOUTS = 100000

# Festlegen einer festen Farbe fuer jedes Quadrat
COLORS = ["green", "yellow", "red", "blue", "purple"]
NUM_SQUARES = 5

# Festlegen eines Mindestabstands zwischen den Quadraten
MIN_DIST = 10
GRID_STEP = 20

# Definition des Randabstands fuer das Platzieren der Quadrate
MARGIN = 20

# Erstellen der Listen
layouts = []
centers_list = []
```

Die Anzahl der erstellbaren Layouts ist theoretisch unbegrenzt, abhängig von der verfügbaren Rechenzeit des Skripts. In diesem Beispiel wurde die Anzahl der platzierbaren Maschinen exemplarisch auf fünf begrenzt. Dies beeinflusst direkt die Komplexität des Optimierungsproblems: Je mehr Maschinen berücksichtigt werden müssen, desto mehr mögliche Layouts gibt es, die zu einer exponentiellen Zunahme der benötigten Rechenzeit führen. Zudem beeinflusst ein höherer Grad an Komplexität bei der Datenerzeugung die spätere Verarbeitung der Daten durch das ANN, da dieses mehr Daten zum Training benötigt um gleichwertige Ergebnisse zu liefern.

Die nachfolgende Funktion `create_layout` erstellt ein Layout mit einer bestimmten Anzahl von zufällig platzierten Quadraten und stellt sicher, dass keine zwei Quadrate zu nahe beieinander liegen oder sich überlappen und das Layout innerhalb einer bestimmten Anzahl von Versuchen erstellt wird.

```
def create_layout(num_squares, min_dist, max_attempts, margin, fixed_seed):
  # Create an empty image for the background
  bg = Image.new('RGB', BG_SIZE, (224, 224, 224))

  # Create a list to store the coordinates of the squares
  squares = []

  # Keep track of how many squares we've created
  squares_created = 0
  attempts = 0
  fixed_start_seed = fixed_seed
  while squares_created < num_squares and attempts < max_attempts:
```

```
    # Generate a random position for the square with a margin
    random.seed(fixed_start_seed)
    x = random.randint(margin, BG_SIZE[0] - SQUARE_SIZE - margin)
    y = random.randint(margin, BG_SIZE[1] - SQUARE_SIZE - margin)
    fixed_start_seed += 1

    # Check if the square collides with any existing squares
    if not check_collision(x, y, squares, min_dist):
      # Add the square to the list of squares
      squares.append((x, y))
      squares_created += 1
      attempts = 0
    else:
      attempts += 1

  if squares_created < num_squares:
    return None, None

  # Set the font size
  font = ImageFont.truetype("arial", 8)

  # Draw the squares onto the background image
  draw = ImageDraw.Draw(bg)
  for color, square in enumerate(squares):
    draw.rectangle((square[0], square[1], square[0] + SQUARE_SIZE,
    square[1] + SQUARE_SIZE), fill=COLORS[color])

  # Calculate the centers of the squares
  centers = []
  for square in squares:
    center_x = square[0] + SQUARE_SIZE // 2
    center_y = square[1] + SQUARE_SIZE // 2
    centers.append((center_x, center_y))

  return bg, centers
```

Die Funktion `create_layout` nimmt fünf Parameter entgegen: die Anzahl der im Layout zu platzierenden Quadrate (Maschinen) (`num_squares`), den minimalen Abstand zwischen den Quadraten (`min_dist`), die maximale Anzahl von Versuchen zur Erstellung eines einzelnen Quadrats (`max_attempts`) und den Randabstand innerhalb dessen keine Quadrate platziert werden sollen (`margin`). (`fixed_seed`) ist eine Variable zur Kontrolle der Zufälligkeit und Reproduzierbarkeit von den Experimenten. Zu Beginn der Funktion wird ein neues Bild mit der Größe `BG_SIZE` und der RGB-codierten Farbe `(200, 255, 255)` erstellt und in der Variable `bg` gespeichert. Weiterhin werden zwei leere Listen `squares` und `centers` für die Speicherung der Quadrate und deren Zentren erstellt. Es gibt auch zwei Zähler

`squares_created` und `attempts` zur Verfolgung der Anzahl der erfolgreich erstellten Quadrate und der Anzahl der Versuche zur Erstellung eines einzelnen Quadrats. Anschließend startet die Funktion eine `while`-Schleife, die so lange läuft, bis entweder die gewünschte Anzahl von Quadraten erstellt wurde oder die maximale Anzahl von Versuchen erreicht ist. Diese Routine wurde programmiert, da die Anzahl der Fabrikeinheiten und deren Größe so konfiguriert werden können, dass sie nicht mehr in die Abmaße vom Layout passen. Im Schleifendurchlauf werden zunächst zufällige x- und y-Koordinaten innerhalb der durch `margin` definierten Grenzen generiert. Danach wird überprüft, ob das neu erstellte Quadrat mit einem bereits existierenden Quadrat über die Funktion `check_collision` kollidiert. Die Funktion zur Kollisionsüberprüfung wird im weiteren Kapitel erklärt. Wenn keine Kollision vorliegt, wird das neue Quadrat zur Liste squares hinzugefügt und der Zähler `squares_created` wird erhöht. Andernfalls wird der Versuchszähler `attempts` erhöht. Falls nach der maximalen Anzahl von Versuchen nicht genug Quadrate erstellt wurden, gibt die Funktion `(None, None)` zurück und bricht ab. Andernfalls wird das Zeichnen der Quadrate auf dem Bild `bg` fortgesetzt. Jedes Quadrat in der Liste `squares` wird mit der entsprechenden Farbe auf das Bild gezeichnet. Falls der Parameter `grid_step` größer als 0 ist, werden zusätzlich Gitterlinien auf das Bild gezeichnet. Schließlich berechnet die Funktion die Zentren der Quadrate und fügt diese zur Liste `centers` hinzu. Als letzter Schritt gibt die Funktion das Bild `bg` und die Liste `centers` zurück.

Die Funktion `check_collision()` wird aufgerufen, um eine Kollision zwischen einem neu generierten Quadrat und allen vorher erstellten Quadraten zu überprüfen. Sie nimmt die x- und y-Koordinaten des neu zu generierenden Quadrats sowie eine Liste von Koordinatenpaaren für alle bereits erstellten Quadrate und einen minimalen Abstand als Eingabe. Der Algorithmus basiert auf der Überlegung, dass sich zwei Quadrate überlappen, wenn sich ihre Grenzen innerhalb eines bestimmten minimalen Abstands voneinander befinden. Dieser stellt sicher, dass die Quadrate nicht zu eng beieinander liegen und genügend Platz zwischen ihnen bleibt.

```
def check_collision(x, y, squares, min_dist):
  for square in squares:
    if abs(square[0] - x) < SQUARE_SIZE + min_dist and abs(square[1] - y) <
                                        SQUARE_SIZE + min_dist:
      return True
  return False
```

Die Funktion iteriert über alle vorher erstellten Quadrate in der übergebenen Liste und prüft für jedes, ob es eine Kollision mit dem neu zu platzierendem Quadrat gibt. Die Überprüfung erfolgt durch den Vergleich der x- und y-Koordinaten der bereits erstellten Quadrate mit denen des neu zu platzierenden Quadrats unter Berücksichtigung des minimalen Abstands. Wenn die Differenz der x-Koordinaten kleiner als die Summe aus der

Quadratgröße und dem minimalen Abstand und die Differenz der x-Koordinaten kleiner als die Summe aus der Quadratgröße und dem minimalen Abstand ist, wird eine Kollision festgestellt. In diesem Fall gibt die Funktion `True` zurück, andernfalls gibt sie `False` zurück.

Um ausschließlich einzigartige Layouts zu erzeugen, werden die erstellen Layouts gespeichert und neu erstellte mit diesen abgeglichen, siehe `is_unique_layout()`:

```
def is_unique_layout(new_centers, existing_centers):
  return not any(set(new_centers) == set(centers) for centers in
                                        existing_centers)
```

Die Ausführung der bis hierhin erläuterten Funktionen erfolgt im Rahmen einer Schleife:

```
layout_seed = 82
for i in tqdm(range(NUM_LAYOUTS)):
  unique_layout_created = False
  while not unique_layout_created:
    layout, centers = create_layout(NUM_SQUARES, MIN_DIST,  MAX_ATTEMPTS,
                                      MARGIN, GRID_STEP, layout_seed)

    if layout is None:
      print("Failed to create unique layout. Skipping...")
      break  # skip this iteration and move on to the next one

    if is_unique_layout(centers, centers_list):
      unique_layout_created = True
      layouts.append(layout)
      centers_list.append(centers)
    else:
      pass
    layout_seed = layout_seed + 1
```

Dieser Codeblock führt die oben erläuterte `create_layout`-Funktion für eine bestimmte Anzahl von Layouts aus. Die Anzahl der zu generierenden Layouts wird durch die Variable `NUM_LAYOUTS` festgelegt. Die Funktion `tqdm` wird hier verwendet, um einen Fortschrittsbalken für die Schleife anzuzeigen. Für jedes `i` in der Anzahl der zu erstellenden Layouts wird die `create_layout`-Funktion aufgerufen. Die Parameter für diese Funktion sind durch die entsprechenden Variablen `NUM_SQUARES`, `MIN_DIST`, `MAX_ATTEMPTS`, `MARGIN` und `GRID_STEP` definiert. Die Funktion gibt ein Layout und die Zentren der Quadrate dieses Layouts zurück. Sollte die `create_layout`-Funktion (`None`, `None`) zurückgeben (dies passiert, wenn sie nach der maximalen Anzahl an Versuchen keine ausreichende Anzahl an Quadraten erzeugen konnte), wird eine entsprechende Fehlermeldung ausgegeben und die aktuelle Iteration der Schleife mit `continue` übersprungen. Falls ein gültiges Layout erstellt werden konnte, wird dieses Layout der Liste `layouts` hinzugefügt und die entsprechenden Zentren der Quadrate der Liste `centers_list` hinzugefügt.

Der nachfolgende Codeblock berechnet im Anschluss die Distanzen zwischen den Zentren der Quadrate (Maschinen) sowohl in Form der Manhattan-Distanz als auch der euklidischen Distanz.

```
# Calculate the distances
manhattan_distances = []
euclidean_distances = []

for centers in centers_list:
  # Calculate the Manhattan and Euclidean distance between squares
  manhattan_dists = []
  euclidean_dists = []
  for i in range(len(centers) - 1):
    dist_x = abs(centers[i][0] - centers[i+1][0])
    dist_y = abs(centers[i][1] - centers[i+1][1])
    manhattan_dists.append(dist_x + dist_y)
    euclidean_dists.append(math.sqrt(dist_x**2 + dist_y**2))

    # Calculate the total distance for the layout
    manhattan_distance = sum(manhattan_dists)
    euclidean_distance = sum(euclidean_dists)

    manhattan_distances.append(manhattan_distance)
    euclidean_distances.append(euclidean_distance)

# Show the number of successfully created layouts
print("Anzahl erfolgreich erzeugter Layouts: " + str(len(layouts)))
print("Image size: ", layouts[0].size)
```

Zu Beginn werden zwei leere Listen `manhattan_distances` und `euclidean_distances` initialisiert, die später dazu dienen, die berechneten Gesamtdistanzen der einzelnen Layouts zu speichern. Die äußere Schleife läuft dann über alle in der `centers_list` gespeicherten Quadratzentren der einzelnen Layouts. Für jedes Layout werden zunächst wiederum zwei leere Listen `manhattan_dists und euclidean_dists` erzeugt, die die berechneten Distanzen zwischen jeweils zwei benachbarten Quadraten speichern sollen. Die innere Schleife läuft dann über alle benachbarten Paare von Quadratzentren innerhalb eines Layouts. Hierbei wird die Manhattan-Distanz (die Summe der absoluten Distanzen der x- und y-Koordinaten) und die euklidische Distanz (der Quadratwurzel der Summe der quadrierten Distanzen der x- und y-Koordinaten) berechnet. Die Ergebnisse werden dann den jeweiligen Listen `manhattan_dists und euclidean_dists` hinzugefügt. Nachdem alle Distanzen zwischen benachbarten Quadraten eines Layouts berechnet wurden, wird die Gesamtdistanz des Layouts berechnet, indem alle Distanzen innerhalb der Listen `manhattan_dists und euclidean_dists` aufsummiert werden. Diese layoutbezogenen Gesamtdistanzen (MHC) werden zuletzt den Listen `manhattan_distances und euclidean_distances` hinzugefügt.

Am Ende des Codeblocks werden dann zu Überprüfungszwecken die Anzahl der erfolgreich erstellten Layouts, die Größe der Bilder der Layouts sowie die Anzahl der berechneten Manhattan-Distanzen ausgegeben. Die Anzahl der Manhattan-Distanzen und der euklidischen Distanzen sollte gleich der Anzahl der erfolgreich erstellten Layouts sein.

Abbildung 25 zeigt ein exemplarisches Layout aus einer Codeausführung mit Koordinatengitter und aufsteigend nummerierten OE. Die erste OE (grün) stellt hierbei ein Eingangslager und die letzte OE (pink) ein Ausgangslager dar.

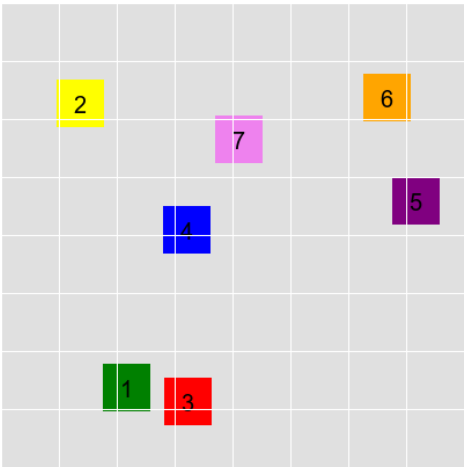


Abbildung 25: Beispiellayout aus einer Codeausführung des Layoutgenerators

Die dargestellten Gitterlinien sind per Boolean aktivierbar und helfen bei der Entwicklung und Überprüfung der internen Skriptlogik.

Das Verhältnis zwischen den Maschinen, der Produktionsstrom, wird durch eine Transportmatrix beschrieben, die von der nachfolgenden Funktion erzeugt wird.

```
def create_transport_matrix(num_inputs, num_outputs):
  # Initialize the transport matrix with zeros
  T = [[0 for _ in range(num_inputs)] for _ in range(num_outputs)]

  # Set the values for the matrix
  for i in range(min(num_inputs, num_outputs)):
    T[i][i] = 1

  return T

NUM_INPUTS = NUM_SQUARES - 1
NUM_OUTPUTS = NUM_SQUARES - 1

transport_matrix = create_transport_matrix(NUM_INPUTS, NUM_OUTPUTS)
```

Die in Abbildung 26 dargestellte Matrix ist eine quadratische Identitätsmatrix der Dimension $n \times n$, wobei n die Anzahl der Maschinen in der betrachteten Produktionskette darstellt. Diese Matrix repräsentiert eine idealisierte Transportmatrix, die den Übergang eines Arbeitsschrittes von einer Maschine zur nächsten in der Prozesskette modelliert. Die Einträge der Matrix sind so definiert, dass auf der Hauptdiagonalen Einsen stehen, während alle anderen Einträge null sind.

```
[1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1]
```

Abbildung 26: Transportmatrix

In der Funktion `create_transport_matrix` wird die Variable `T` als eine Liste von Listen initialisiert, wobei jede innere Liste eine Zeile der Matrix repräsentiert und alle Einträge auf null gesetzt sind. Die Schleife läuft dann über die kleineren der beiden Werte `num_inputs` oder `num_outputs`. Für jeden Durchlauf der Schleife wird der Eintrag an der Position $[i][i]$ auf Eins gesetzt. Dies hat zur Folge, dass in der resultierenden Matrix jede OE seine Arbeit an die nächste OE weitergibt. Schließlich wird die erstellte Matrix zurückgegeben. Durch diese Funktion wird eine Situation modelliert, in der die Maschinen gleichmäßig verteilt sind und OE ihre Arbeit in der Reihenfolge ihres Auftretens an die nächste OE weitergeben.

6.3 Implementierung der Layoutsimulation

Für die OE und das Transportsystem wurden jeweils eigene Klassen implementiert. So können von jeder Klasse beliebig viele Instanzen erstellt werden.

6.3.1 Modellierung der Maschinenklasse

Die Modellierung der Maschinen bildet die spezifischen Eigenschaften und Verhaltensweisen der OE in der Produktion ab. Die nachfolgende Python-Klasse namens `class Machine` zeigt die Implementierung.

```
class Machine:
  def __init__(self, buffer_cap, robot_load_space, position):
    self.input_buffer = 0
    self.output_buffer = 0
    self.producing = False
    self.position = position
    self.buffer_cap = buffer_cap
    self.robot_load_space = robot_load_space
    self.produced_goods = 0
    self.pickup_request = False
    self.delivery_request = True
    self.loading = False
```

```
  def produce(self):
    if (self.producing == True and self.loading == False):
      self.input_buffer -= 1
      self.output_buffer += 1

  def check_buffers(self):
    if self.input_buffer <= 0 or self.output_buffer >= self.buffer_cap:
      self.producing = False
    else:
      self.producing = True
    if ((self.buffer_cap - self.input_buffer) > self.robot_load_space * 0.9)
                                        :
      self.delivery_request = True
    else:
      self.delivery_request = False
    if (self.output_buffer > self.robot_load_space * 0.8):
      self.pickup_request = True
    else:
      self.pickup_request = False

    # Stop to produce when the output buffer cannot hold more items
    if (self.output_buffer >= self.buffer_cap):
      self.producing = False
```

Die Maschinen-Klasse definiert die Attribute `input_buffer`, `output_buffer`, `producing` und `position` um den Zustand der Maschine zu modellieren. Diese Attribute werden beim Aufruf der `__init__`-Methode initialisiert. Die Methode `produce` modifiziert die Zustände der Eingabe- und Ausgabepuffer der Maschine, wenn sie sich im Produktionsmodus befindet und nicht gerade beladen wird. Die Methode `check_buffers` überprüft den Zustand der Materialpuffer und aktualisiert die Zustände `producing`, `delivery_request` und `pickup_request` entsprechend. Diese Methode sorgt dafür, dass die Maschine die Produktion stoppt, wenn der Eingabepuffer leer oder der Ausgabepuffer voll ist. Sie setzt außerdem `delivery_request` auf `True`, wenn der Eingabepuffer fast leer ist, und `pickup_request` auf `True`, wenn der Ausgabepuffer fast voll ist.

6.3.2 Modellierung der Klasse für das Transportmedium

Um die Interaktion der OE mit den Transportsystemen zu modellieren, wird eine `Transport`-Klasse implementiert. Diese repräsentiert ein Transportmedium, das Güter zwischen den OE transportieren kann.

```
class Transport:
  def __init__(self, trans_capacity, trans_speed, current_position):
```

```
    self.transport_capacity = transport_capacity
    self.transport_speed = transport_speed
    self.current_position = current_position
    self.target_position = None
    self.remaining_distance = 0
    self.driven_distance = 0
    self.cargo = 0
    self.loading = False
    self.unloading = False
    self.available = True
    self.transport_job = None

  def set_transport_job(self, trans_job):
    self.transport_job = trans_job
    self.available = False
    self.set_target_position_manhattan(trans_job)

  def set_target_position_manhattan(self, destination):
    if self.current_position != destination[0]:
      # If curr pos not equal to the pickup machine, drive to it first
      self.target_position = destination[0]
    else:
      # If already at pickup pos, drive to delivery machine
      self.target_position = destination[1]

    dx = self.target_position[0] - self.current_position[0]
    dy = self.target_position[1] - self.current_position[1]
    self.remaining_distance = abs(dx) + abs(dy)

  def load(self, cargo):
    if self.cargo < self.transport_capacity:
    self.cargo += cargo
    self.loading = True
    elif self.cargo >= self.transport_capacity:
    self.loading = False

  def unload(self, cargo):
    if (self.cargo - cargo) > 0:
      self.unloading = True
      self.cargo -= cargo
    else:
      self.cargo = 0
      self.available = True
      self.unloading = False

  def reset_position(self):
    self.remaining_distance = 0 # this stays at zero
```

```
    new_x = self.current_position[0]
    new_y = self.current_position[1] - 1

    self.current_position = (new_x, new_y)

  def update_position_manhattan(self, time_step):
    if self.target_position == None:
      return

    if self.loading == True: # No update while loading
      return True
    elif self.unloading == True:
      return True

    dx = self.target_position[0] - self.current_position[0]
    dy = self.target_position[1] - self.current_position[1]
    distance = abs(dx) + abs(dy) # Manhattan distance

    if distance <= 0:
      self.remaining_distance = 0

    time_to_travel = distance / self.transport_speed
    proportion_of_distance = min(1, time_step / time_to_travel)
    new_x = self.current_position[0]
    new_y = self.current_position[1]

    if dx > 0:
      new_x += min(self.transport_speed * time_step, abs(dx))
    elif dx < 0:
      new_x -= min(self.transport_speed * time_step, abs(dx))

    if dy > 0 and dx == 0:  only update y is x is finished.
      new_y += min(self.transport_speed * time_step, abs(dy))
    elif dy < 0 and dx == 0:
      new_y -= min(self.transport_speed * time_step, abs(dy))

    self.current_position = (new_x, new_y)
    self.remaining_distance -= self.transport_speed * time_step
    self.driven_distance += self.transport_speed * time_step
    self.remaining_distance = int(self.remaining_distance)

    if self.remaining_distance <= 0: # Goal reached?
      self.current_position = self.target_position
      self.target_position = None
      self.remaining_distance = 0

      if self.current_position ==  self.transport_job[0]:
```

```
        self.loading = True
        self.set_target_position_manhattan(self.transport_job)
      elif self.current_position ==  self.transport_job[1]:
        self.unloading = True
        self.transport_job = None
```

Die Transport-Klasse modelliert einen Transportroboter mit verschiedenen Attributen wie `transport_capacity`, `transport_speed`, `current_position` und `target_position`. Darüber hinaus verfügt der Roboter über eine Methode zum Aktualisieren seiner Position (`update_position_manhattan`), Methoden zum Laden und Entladen von Gütern (`load`, `unload`) und zum Festlegen eines Transportauftrags (`set_transport_job`).

Die `set_transport_job`-Methode nimmt ein Transportjob-Objekt entgegen, setzt es als aktuellen Transportjob und macht den Roboter für weitere Transportjobs nicht verfügbar. Die `load`- und `unload`-Methoden erhöhen bzw. verringern die Menge an Gütern, die der Roboter trägt. Die Methode `update_position_manhattan` aktualisiert die Position des Roboters entsprechend seiner Geschwindigkeit und des aktuellen Zeitintervalls unter Verwendung der Manhattan-Distanz als Metrik zur Berechnung der Entfernung zwischen der aktuellen und der Zielposition des Transportmediums. Diese Entfernungsmetrik reflektiert realistische Bewegungsbedingungen in Produktions- und Lagerumgebungen, die oft einem Gitternetz von Pfaden ähneln - ähnlich wie Autos, die sich auf einem Stadtnetz bewegen. In solchen Umgebungen ist die Bewegung entlang der kürzesten euklidischen Distanz, also eine direkte „Luftlinie“, zwischen zwei Punkten in der Regel nicht möglich. Dies wäre nur der Fall, wenn das Transportmittel in der Lage wäre, direkt durch Maschinen, Geräte oder andere Hindernisse zu navigieren, was in den meisten realen Szenarien nicht der Fall ist.

Die Klasse `Transport` steht hierbei für ein nicht näher definierte Transportmedium. Das Modell gilt nicht nur für Roboter, sondern auch für eine Vielzahl anderer Transportmittel in einem Produktions- oder Lagerumfeld, einschließlich Deckenkräne, Förderbänder und menschliche Akteure. Obwohl diese unterschiedliche Grade an Bewegungsfreiheit haben, sind sie gezwungen sich entlang vorgegebener Wege zu bewegen.

6.3.3 Helferfunktionen für die Simulation

Die `generate_jobs()`-Funktion erstellt eine Liste aller generell möglichen Transportjobs unter Berücksichtigung der Anzahl an OE und der Transportmatrix. Ein Transportjob wird durch Koordinatenpaare von Absender und Empfänger definiert.

```
# Create list of transport jobs, ordered pairs of source and target
def generate_jobs(matrix, machine_pos):
  jobs = []
  for i in range(len(matrix)):
    for j in range(len(matrix[i])):
```

```
      if matrix[i][j] == 1:
        jobs.append((machine_pos[i], machine_pos[j + 1]))
  return jobs
```

Die Eingabeparameter sind:

1. **matrix**: Dies ist mit der Funktion `create_transport_matrix` erstellte Transportmatrix, die die Abhängigkeiten zwischen den OE modelliert, s. Abbildung 26.
2. **machine_pos**: Dies ist eine Liste von OE-Koordinaten, die vom Layoutgenerator gespeichert wurden. Der Index in der Liste entspricht der OE-Nummer, so dass `machine_pos[i]` die Position der i-ten OE ist.

Die Funktion iteriert über jede Zelle in der Matrix und erzeugt zur einer Transportabhängigkeit einen Auftrag der als Tupel von Start- und Zielkoordinaten dargestellt wird. Der Startpunkt ist die Position der aktuellen Maschine, und der Endpunkt ist die Position der nächsten Maschine. Diese Aufträge werden in der Liste `jobs` gesammelt. Am Ende gibt die Funktion die Liste `jobs` zurück, die alle generierten Transportaufträge enthält.

Die Funktion `reset_simulation()` setzt die Simulation zurück. Diese Funktionalität wird benötigt, da die Gesamtheit der Layouts in einer Schleife simuliert wird, und jeweils die Instanzen zurückgesetzt werden.

```
def reset_simulation(m_instances, t_robot):
  # configure the first machine, which is the input warehouse
  for index, machine in enumerate(m_instances):
    machine.input_buffer = 0
    machine.output_buffer = 0
    machine.pickup_request = False # false in default
    machine.delivery_request = True # true in default
    machine.loading = False
  m_instances[0].pickup_request = True
  m_instances[0].delivery_request = False
  m_instances[0].buffer_cap = 1000000
  m_instances[0].output_buffer = 1000000
  m_instances[len(m_instances) - 1].buffer_cap = 1000000
  m_instances[len(m_instances) - 1].delivery_request = True

  t_robot = Robot(transport_capacity = robot_cap, transport_speed = 10,
                                       current_position = (SQUARE_SIZE / 2,
                                       SQUARE_SIZE / 2))
  t_robot.loading = False
  t_robot.unloading = False
  t_robot.cargo = 0
  t_robot.driven_distance = 0
  t_robot.remaining_distance = 0
  SIMULATION_TIME = 60 * 60 * 24 # 1 day
  SIMULATION_TIME = 100 # 10 seconds
```

```
  sim_time = 0
  sim_step = 1
  t_jobs_pending = []
  num_t_job_list = []

  return m_instances, t_robot, sim_time, SIMULATION_TIME, t_jobs_pending,
                                        num_t_job_list
```

Die Funktion `create_transport_jobs()` erstellt eine Liste von anstehenden Transportaufträgen, basierend auf dem aktuellen Zustand der Maschinen und des Roboters.

```
def create_transport_jobs(machines, robot, sim_layout, t_jobs,
                                        t_job_numbered):
  trans_job = []

  if robot.available == False:
    return trans_job, t_job_numbered

  # create a list of pending transport jobs.
  for index, machine in enumerate(machines[:-1]):
    # If the job is already currently handled by the robot, skip it
    if (t_jobs[index] == robot.transport_job):
      continue

    # first machines can deliver and second machine able to receive
    if (machines[index].pickup_request == True and machines[index+1].
                                        delivery_request == True):
      trans_job.append(t_jobs[index]) # insert at index the transport job

  # Sort transport jobs according to lowest input buffer and keep lowest
  dict_of_input_buffer = {}
  for machine in machines[1:]:
    dict_of_input_buffer[machine.position] = machine.input_buffer
  dict_of_input_buffer = sorted(dict_of_input_buffer.items(), key=lambda x:
                                        x[1])[0]

  # loop through transport jobs Kanban principle
  for job in trans_job:
    if job[1] in dict_of_input_buffer:
      trans_job = []
      trans_job.append(job)
      # Convert the transport jobs to machine numbers
      conv_job_coords = "Von " + str(sim_layout.index(job[0]) + 1) + " zu "
                                        + str(sim_layout.index(job[1]) + 1)

      t_job_numbered.append(conv_job_coords + " - " + str(job))
```

```
return trans_job, t_job_numbered
```

Die Funktion nimmt folgende Parameter entgegen:

1. `machines`: Eine Liste von Maschinenobjekten (OE), die den Zustand und die Position jeder Maschine in der Anlage repräsentieren.
2. `robot`: Ein Roboterobjekt, das den aktuellen Zustand und die Position des Roboters repräsentiert.
3. `sim_layout`: Eine Liste, die das aktuelle Layout der Anlage repräsentiert.
4. `t_jobs`: Eine Liste von bereits definierten Transportaufträgen.
5. `t_job_numbered`: Eine Liste von bisher zugewiesenen Transportaufträgen, dargestellt als Strings mit den Maschinennummern und Positionen.

Zunächst wird überprüft, ob der Roboter verfügbar ist. Wenn nicht, werden keine neuen Transportaufträge erstellt und die Funktion gibt die Liste `t_job_numbered` unverändert zurück. Anschließend wird über die Liste der OE iteriert (bis auf das Ausgangslager). Wenn ein Transportauftrag zwischen zwei OE besteht (die erste Maschine hat eine Anforderung zur Abholung und die zweite OE hat eine Anforderung zur Lieferung), wird dieser Transportauftrag der Liste `trans_job` hinzugefügt. Danach wird eine Liste von OE erstellt, sortiert nach ihrem Eingangspuffer. Die Maschine mit dem kleinsten Eingangspuffer wird ausgewählt.

Schließlich wird die Liste `trans_job` durchlaufen und nur der Transportauftrag mit der geringsten Eingangspuffermenge beibehalten. Die Logik basiert auf dem Kanban- oder Pull-Prinzip, ein zentraler Bestandteil von „Just-in-Time“-Produktionsstrategien und Lean-Management-Praktiken. Das Prinzip beruht auf der Idee, dass Produktion und Transport von Materialien dann stattfinden, wenn tatsächlich ein Bedarf besteht. Die Bedürfnisse des „Verbrauchers“ (in diesem Fall die Maschine mit dem geringsten Eingangspuffer) leiten die Produktion und den Transport. Dies steht im Gegensatz zu Push-basierten Systemen, in denen die Produktion und der Transport von Materialien auf Basis von Prognosen oder Plänen unabhängig vom unmittelbaren Bedarf stattfinden. Die Entscheidung für ein Kanban-basiertes System in diesem Kontext bietet mehrere Vorteile. Erstens fördert es die Effizienz durch die Reduzierung von Überproduktion und die Minimierung von Lagerbeständen. Zweitens verbessert es die Flexibilität, da es in der Lage ist, sich an wechselnde Bedarfe anzupassen. Drittens fördert es die Kontrolle über den Produktionsprozess, da es die Transportaufträge auf der Grundlage der aktuellen Bedürfnisse der Maschinen erstellt.

Die Funktion gibt zwei Listen zurück: `trans_job`, eine Liste der neu erstellten Transportaufträge und eine aktualisierte Liste der bisher zugewiesenen Transportaufträge `t_job_numbered`.

Die Funktion `distribute_transport_jobs` ist verantwortlich für die Zuweisung von

Transportaufträgen an den Transportroboter.

```
def distribute_transport_jobs(robot, t_job_list):
  # Pruefe das Vorhandensein offene Transportjobs
  if len(t_job_list) > 0:
    print("{} offene t-jobs in Warteschlange.".format(len(t_jobs_pending)))

    # Pruefe ob das Transportmedium frei ist
    if t_robot.available == False:
      print("Transport nicht moeglich.")
      return
    else:
      robot.set_transport_job(t_job_list[0])
```

Die Funktion `distribute_transport_jobs` beginnt zunächst mit der Überprüfung nach offenen Transportaufträgen in der Liste `t_job_list`. Falls keine vorhanden sind, endet die Funktion mit dem `return`-Statement. Falls offene Aufträge vorhanden sind, wird die Verfügbarkeit des Roboters überprüft. Ist der Roboter nicht verfügbar, da er beispielsweise gerade am Laden oder Abladen ist, endet die Funktion ohne einen neuen Auftrag zuzuweisen. Ist der Roboter hingegen verfügbar, wird der Transportauftrag mit der höchsten Priorität (der erste in der Liste `t_job_list`) zugewiesen. Dies geschieht durch den Aufruf der Methode `set_transport_job` mit dem entsprechenden Auftrag als Parameter.

6.3.4 Ausführung der Simulation

Die Umsetzung der Simulation erfolgt durch die nachfolgende `simulation`-Funktion. Diese initiiert den Prozess mit der Generierung der Transportaufgaben, basierend auf den Zuständen und Positionen der Maschinen. Dieser Schritt entspricht dem „Job Generation“-Knoten im Diagramm von Abbildung 18. Nach der Generierung der Aufgaben erfolgt die Zuweisung der Aufgaben an die Roboter. Dies korrespondiert mit dem „Job Assignment“-Knoten. Im weiteren Verlauf werden die Zustände und Positionen der Roboter und Maschinen aktualisiert. Diese Aktualisierung spiegelt die „Robot Update“ und „Machine Update“-Knoten im Diagramm wider.

```
def simulation(t_jobs_pending, numb_tjob, m_instances, t_robot, s_layout,
                                        s_time):
  t_jobs = generate_jobs(transport_matrix, s_layout)

  t_jobs_pending, numb_t_job_list = create_transport_jobs(m_instances,
                                        t_robot, s_layout, t_jobs, numb_tjob)
  distribute_transport_jobs(t_robot, t_jobs_pending)

  if t_robot.loading == False and t_robot.unloading == False:
    goal_reached = t_robot.update_position_manhattan(time_step = sim_step)
```

```
    for i, machine in enumerate(m_instances):
      machine.loading = False
  else:
    if t_robot.loading == True:
      for i, machine in enumerate(m_instances):
        if machine.position == t_robot.current_position and t_robot.loading
                                    == True and machine.output_buffer > 0
                                    :
          machine.loading = True
          if (machine.output_buffer - loading_speed) < 1:
            t_robot.load(cargo = machine.output_buffer)
            machine.output_buffer = 0
            t_robot.loading = False
        else:
          machine.output_buffer -= loading_speed
          t_robot.load(cargo = loading_speed)

    elif t_robot.unloading == True:
      t_robot.unload(cargo = loading_speed)
      for i, machine in enumerate(m_instances):
        if machine.position == t_robot.current_position and t_robot.
                                    unloading == True:
          machine.input_buffer += loading_speed
          machine.loading = True
        if t_robot.cargo == 0:
        t_robot.reset_position()

  for i, machine in enumerate(m_instances):
    machine.check_buffers()
    if machine.producing == True:
      machine.produce()

  s_time += sim_step

  return t_jobs_pending, numb_tjob, m_instances, t_robot, s_time
```

Der nachfolgende Code simuliert unterschiedliche Layouts und erfasst Leistungskennzahlen.

```
sim_end_time = 25000 # Sekunden
metrics = []
print("Simulating " + str(len(centers_list)) + " layouts...\n")
for index in tqdm(range(len(centers_list))):
  # Neues Layout fuer Simulation einrichten
  t_robot = Robot(transport_capacity = robot_cap, transport_speed = 10,
                                    current_position = (SQUARE_SIZE / 2,
                                    SQUARE_SIZE / 2))
```

```
m_instances = []
for machine in range(len(centers_list[index])):
  m_instances.append(Machine(buffer_cap = buffer_cap, robot_load_space =
                                    robot_cap, position = centers_list[
                                    index][machine]))

m_instances, t_robot, sim_time, SIMULATION_TIME, t_jobs_pending,
                                    num_t_job_list = reset_simulation(
                                    m_instances, t_robot)

while sim_time < sim_end_time:
  t_jobs_pending, num_t_job_list, m_instances, t_robot, sim_time =
                                    simulation(t_jobs_pending,
  num_t_job_list, m_instances, t_robot, centers_list[index], sim_time)

# Ausgangspuffer nach eingeschwundenen Zustand zuruecksetzen
if sim_time == 5000:
  m_instances[-1].output_buffer = 0
  t_robot.driven_distance = 0

metrics.append({
  "machine_number": len(centers_list[index]),
  "machine_coordinates": centers_list[index],
  "transport_robots": 1,
  "throughput": m_instances[-1].output_buffer,
  "driven_distance": t_robot.driven_distance,
  "manhattan_distance": manhattan_distances[index],
})
```

Im dargestellten Code befindet sich die Hauptschleife, die durch die generierten Layouts iteriert. Zu jedem Layout werden die Maschinen und Transporteinheiten erstellt und die Simulationsvariablen zurückgesetzt. Die darauffolgende „for-Schleife“ simuliert das jeweilige Layout über die definierte Höchstdauer von 25000 Sekunden. Um anfängliche Verzerrungen zu eliminieren, wurden 5000 Sekunden als Aufwärmphase definiert, vgl. Kapitel 3.2.5. Nach dieser Zeit werden die gesammelten Metriken zurückgesetzt. Diese umfassen den Durchsatz und die gefahrene Distanz der Transporteinheit.

6.4 Implementierung zur Datenaufbereitung

Der nachfolgende Code zeigt die Implementierung der in Kapitel 5.2.3 konzeptionierten Datenaufbereitung.

```
# Normalisierung der Bilder (Teilen durch 255)
images = np.array(images) / 255.0
X = images
```

```
# Normalisierung des sim. Durchsatzes mit MinMaxScaler (Array, Spalte)
scaler = MinMaxScaler()
metric = np.array(sim_df["throughput"]).reshape(-1, 1)
metric_scaled = scaler.fit_transform(metric)
y = metric_scaled

# Aufteilung inTrainings-, Validierungs- und Testdatensatz
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.15,
                                    random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=
                                    0.1765, random_state=42)
```

Der Code führt folgende Schritte durch:

1. **Normalisierung der Bilder**: Zuerst werden die Rohbilder `images` in ein NumPy-Array umgewandelt und normalisiert, indem die Farbintensität jedes Pixels durch 255 geteilt wird. Dadurch liegen die Pixelwerte im Intervall [0, 1]. Das Array wird in der Variable X gespeichert.
2. **Normalisierung der Zielvariablen**: Der simulierte Durchsatz wird aus dem DataFrame `sim_df` extrahiert und in ein 2D-NumPy-Array umgewandelt (durch `reshape(-1, 1)`). Die Min-Max-Normalisierung in das Intervall [0, 1] wird dann mit dem `MinMaxScaler` von *scikit-learn* angewendet. Das Array wird in der Variable y gespeichert.
3. **Aufteilung in Trainings-, Validierungs- und Testsets**: Die Daten werden in Trainings-, Validierungs- und Testsets aufgeteilt. Dabei wird ein Verhältnis von 70%:15%:15% verwendet, wie zuvor beschrieben. Die Aufteilung der Daten erfolgt in einem zweistufigen Prozess:
    a) Zuerst wird der gesamte Datensatz in einen temporären Datensatz (`X_temp`, `y_temp`) und einen Testdatensatz (`X_test`, `y_test`) aufgeteilt. Dabei werden 15% der Daten für den Testdatensatz verwendet.
    b) Danach wird der temporäre Datensatz in Trainings- und Validierungsdaten aufgeteilt. Der Wert `test_size=0.1765` sorgt dafür, dass 17.65% der verbleibenden 85% des Datensatzes als Validierungsdaten verwendet werden. Dies führt zu einer Aufteilung, die nah an 70% für das Training, 15% für die Validierung und 15% für die Tests liegt.

6.4.1 Implementierung zur Konsistenzüberprüfung

Der nachfolgende Code überprüft mithilfe der `assert`-Anweisung, ob in den Zielmetriken negative Werte enthalten sind. Außerdem wird der Datensatz hinsichtlich Null-Werten

überprüft.

```
# Ueberpruefung auf positive Ganzzahlen
assert all(sim_df["throughput"] >= 0)

# Ueberpruefung auf fehlende Werte im Datensatz
assert sim_df["throughput"].isnull().sum() == 0
```

Falls die jeweils aufgestellte Bedingung nicht erfüllt ist, wird die `assert`-Anweisung eine Ausnahme auslösen und das Programm abbrechen.

6.4.2 Visualisierung und statistische Analyse der Zielvariablen

Um die Qualität des Datensatzes zu evaluieren, wurden verschiedene Visualisierungs- und statistische Methoden eingesetzt. Diese umfassen Scatterplots, Boxplots und Ausreißeranalysen. Nachfolgend werden diese Methoden beschrieben und die Ergebnisse der zugehörigen Grafiken interpretiert. Abbildung 27 präsentiert einen Scatterplot der simulierten Durchsätze, wobei die x-Achse den Index der Datenpunkte und die y-Achse die Bandbreite der simulierten Durchsätze darstellen.

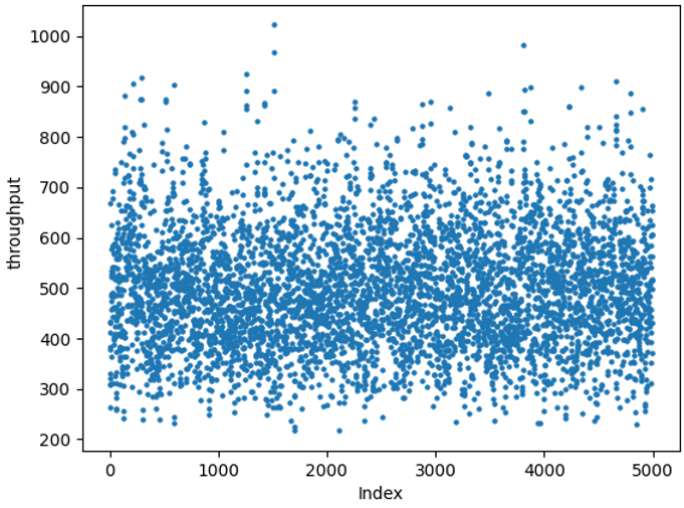


Abbildung 27: Scatterplot der simulierten Durchsätze

Die gleichmäßige Verteilung der Punkte im Scatterplot weist keine signifikante Anhäufung an einem bestimmten Wertebereich auf. Dies ist ein positiver Indikator für die Qualität des Datensatzes, da eine hohe Variabilität in den Daten typischerweise das Training von ANN begünstigt. Abbildung 28 zeigt einen „Box-and-Whisker-Plot“ (kurz Box-Plot) zu dem simulierten Durchsatz.

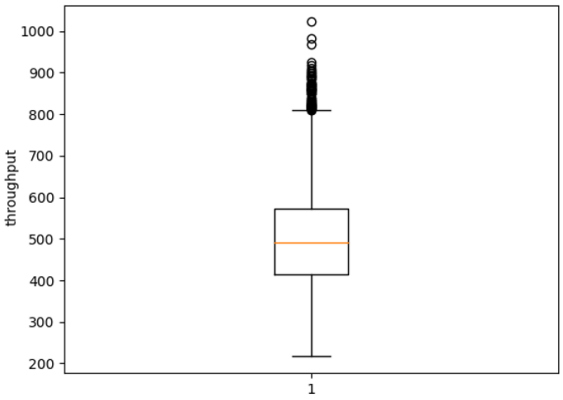


Abbildung 28: Boxplot zum simulierten Durchsatz

- **Minimum**: Der kleinste Wert im Datensatz liegt bei 217 und wird durch das untere Ende des „Whiskers“ dargestellt.
- **Unteres Quartil (Q1, 25. Perzentil)**: 25% der Datenpunkte sind kleiner als dieser Wert. Dies ist das untere Ende der Box.
- **Median (Q2, 50. Perzentil)**: Der Wert, der die Daten in zwei Hälften teilt. 50% der Datenpunkte sind kleiner als dieser Wert. Der Median wird durch die rote Linie innerhalb der Box dargestellt.
- **Oberes Quartil (Q3, 75. Perzentil)**: 75% der Datenpunkte sind kleiner als dieser Wert. Dies ist das obere Ende der Box.
- **Maximum**: Der größte Wert im Datensatz, der nicht als Ausreißer identifiziert wurde ist 817 und wird durch das obere Ende des „Whiskers“ dargestellt. Der maximale Ausreißer hat einen Wert von 1022.
- **Ausreißer**: Sämtliche Punkte außerhalb der Whisker-Linie. In diesem Fall gibt es lediglich obere Ausreißer.

6.4.3 Statistische Ausreißeranalyse

Neben der visuellen Untersuchung der Daten durch Scatterplots und Boxplots wurden die Trainingsdaten im nächsten Schritt hinsichtlich ihrer statistischen Beschaffenheit analysiert. Dies dient der Identifikation von Ausreißern, die die Leistung des Modells während des Trainingsprozesses beeinträchtigen könnten. Hierfür wurde der Z-Score als statistisches Maß verwendet. Zusätzlich wird die Bandbreite der Daten durch die Bestimmung des Minimums und Maximums gezeigt. Dies sind die Ergebnisse der Analyse eines Durchlaufs:

- Negative Ausreißer-Werte: []
- Positive Ausreißer-Werte: [881, 905, 875, 918, 875, 870, 874, 904, 862, 855, 925, 891, 868, 863, 969, 1022, 891, 870, 857, 864, 869, 858, 886, 983, 851, 893, 899, 859, 861, 898, 911, 887,

855]

- Bandbreite: Minimum: 217, Maximum: 1022
- Median: 490.0, IQR[17]: 158.0

Der zugehörige Code für diese Analyse ist im Folgenden dargestellt:

```
# Berechnung des z-Scores aller Datenpunkte
z_scores = zscore(sim_df["throughput"])
# Identifizierung von negativen Outliern (Z-Score < -3)
negative_outliers = np.where(z_scores < -3)[0]
negative_outlier_values = sim_df["throughput"].iloc[negative_outliers]
# Positive Outliers (Z-Score > 3)
positive_outliers = np.where(z_scores > 3)[0]
positive_outlier_values = sim_df["throughput"].iloc[positive_outliers]

# Zeige die Bandbreite (Minimum und Maximum) der Daten
min_value = sim_df["throughput"].min()
max_value = sim_df["throughput"].max()
print(f"Minimum: {min_value}, Maximum: {max_value}")

# Entfernen von den erkannten Outlier
all_outliers = np.concatenate([negative_outliers, positive_outliers])
all_outliers = np.sort(all_outliers)

sim_df = sim_df.drop(index=all_outliers)

# Nach dem Entfernen der Outlier Index zuruecksetzen
sim_df.reset_index(drop=True, inplace=True)

# Analog die Bilder mit dem gleichen Index entfernen
images = np.delete(images, all_outliers, axis=0)
```

Nach dem Entfernen der ersten Runde von Ausreißern ändern sich der Mittelwert und die Standardabweichung des Datensatzes. Durch iterative statistische Untersuchung können weitere Ausreißer gefunden werden. In den vorliegenden Daten handelt es sich hier um vernachlässigbare Größen < 5. Daher wird die Ausreißer-Analyse nicht iterativ durchgeführt.

Mit der statistischen Ausreißeranalyse wurde insbesondere der simulierte Durchsatz berücksichtigt. Da die Layouts nur eine begrenzte Zeit simuliert wurden, kann es vorkommen, dass bestimmte Layouts durch *Bottlenecks* oder *Dead Ends* keinen Durchsatz generieren können und bieten als Zielmetrik keinen Mehrwert für das ANN. Der nachfolgende Code erkennt und entfernt Ausreißer:

```
# Minimum der Spalte finden
```

[17] Der Interquartilsabstand (IQR) ist ein Maß für die statistische Streuung und wird als Differenz zwischen dem 75. und 25. Perzentil berechnet. Er gibt den Bereich an, in dem die mittleren 50% der Datenpunkte liegen.

```
min_wert = metric_df['throughput'].min()
mask = metric_df['throughput'] == min_wert

# Anzahl der Vorkommen des minimalen Wertes
anzahl_vorkommen = mask.sum()

# Indizes des minimalen Wertes
indizes_min_wert = metric_df.index[mask].tolist()

# Entfernen von Durchsatz == 0
metric_df_filtered = metric_df[~mask]

# Anhand des Indexes die jeweiligen Bilder entfernen
mask_to_keep = ~mask

if isinstance(images, torch.Tensor) and len(images) == len(metric_df):
  filtered_images_tensor = images[mask_to_keep]
```

Zuerst wird das Minimum der *throughput*-Spalte in *metric_df* ermittelt. Anschließend wird eine Maske erstellt, die angibt, wo in der *throughput*-Spalte dieses Minimum auftritt. Die Anzahl der Vorkommen dieses minimalen Wertes wird gezählt, und die Indizes dieser Vorkommen werden in einer Liste gespeichert. Danach wird der Datensatz *metric_df* gefiltert, um alle Einträge zu entfernen, bei denen der *throughput* gleich dem minimalen Wert ist. Schließlich wird überprüft, ob das Objekt *images* ein Torch-Tensor ist und die gleiche Länge wie *metric_df* hat; wenn ja, werden die entsprechenden Bilder, die nicht dem minimalen Wert entsprechen, in einem neuen Tensor gespeichert.

6.5 Implementierung der Datenklasse

Um den in Kapitel 5.3.4 Ansatz zu ermöglichen, wurde eine benutzerdefinierte Datenklasse innerhalb des PyTorch-Frameworks implementiert. Diese bündelt Eingabebilder, zugehörige zusätzliche Daten und Labels in einem einzigen Datensatz.

```
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, X, add_data, y):
    self.X = X  # Layout-Daten
    self.add_data = add_data
    self.y = y  # Zielwerte (Labels)

  def __len__(self):
    return len(self.X)

  def __getitem__(self, idx):
    return self.X[idx], self.add_data[idx], self.y[idx]
```

Die Implementierung der Klasse erbt von *torch.utils.data.Dataset*. Die Methoden *__len__* und *__getitem__* gewährleisten die Kompatibilität mit dem DataLoader von PyTorch. Dies ermöglicht eine effiziente Batching- und Sampling-Strategie während des Trainings, bei der die jeweiligen Anteile gezielt in das ANN eingespeist werden, siehe Kapitel 6.6.

6.6 Implementierung der KI-Architektur

In diesem Abschnitt wird die Implementierung des konkatenierten ANN, bestehend aus MLP und CNN erörtert, vgl. Konzeption in Kapitel 5.3.

```
class Layout_Evaluator(nn.Module):
  def __init__(self, pool_size=(4,4)):
    super(Layout_Evaluator, self).__init__()

    # Konvolutions- und Pooling-Schichten
    self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
    self.bn1 = nn.BatchNorm2d(64)
    self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
    self.bn2 = nn.BatchNorm2d(128)
    self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
    self.bn3 = nn.BatchNorm2d(256)
    self.conv4 = nn.Conv2d(256, 512, 3, padding=1)
    self.bn4 = nn.BatchNorm2d(512)
    self.conv5 = nn.Conv2d(512, 1024, 3, padding=1)
    self.bn5 = nn.BatchNorm2d(1024)
    self.pool = nn.MaxPool2d(2, 2)
    self.adapool = nn.AdaptiveAvgPool2d(pool_size)

    # Groesse basierend auf der Ausgabe der Pooling-Schicht
    self.flat_size = 512 * pool_size[0] * pool_size[1]
    self.combined_size = self.flat_size + 512

    # Zusaetzliche Daten
    self.add_data_layer = nn.Linear(2 * max_machines + 4, 512)

    # Voll verbundene Schichten
    self.fc1 = nn.Linear(self.combined_size, 1024)
    self.fc2 = nn.Linear(1024, 512)
    self.fc3 = nn.Linear(512, 256)
    self.fc4 = nn.Linear(256, 128)
    self.fc5 = nn.Linear(128, 1)

    # Dropout und Aktivierung
    self.dropout = nn.Dropout(0.1)
    self.relu = nn.ReLU()
```

Das entwickelte ANN besteht aus fünf konvolutionellen Schichten, ergänzt durch Batch-Normalisierung zur Reduktion der internen Kovarianzverschiebung[18], und weiteren fünf voll verbundenen Schichten. Die konvolutionellen Schichten erhöhen die Anzahl der Merkmalskarten schrittweise von 64 auf 1024, mit einer Kernelgröße von $3x3$ und einer gleichbleibenden *Padding*[19]-Einstellung. Pooling-Schichten reduzieren die räumlichen Dimensionen der Merkmalskarten, ergänzt durch eine adaptive *Average-Pooling*-Schicht, die die Ausgabe auf eine spezifizierte Größe standardisiert. Die anschließenden voll verbundenen Schichten verarbeiten die geflatteten[20] Bildmerkmale zusammen mit weiteren, in einer dedizierten linearen Schicht vorverarbeiteten Daten. Dies ermöglicht eine erweiterte Integration von Merkmalen, die in die Vorhersage einfließt. Um Overfitting zu vermeiden, wird ein Dropout von 10% angewendet, während die ReLU-Aktivierungsfunktion nach jeder voll verbundenen Schicht eingesetzt wird, um die Nichtlinearität innerhalb des Netzwerkes zu erhöhen.

Der nachfolgende Vorwärtspass innerhalb der Klasse ist für die Durchführung der Datenverarbeitung durch das ANN verantwortlich:

```
def forward(self, x, add_data):
  # Vorwaertspass durch das Netzwerk
  x = self.pool(self.relu(self.bn1(self.conv1(x))))
  x = self.pool(self.relu(self.bn2(self.conv2(x))))
  x = self.pool(self.relu(self.bn3(self.conv3(x))))
  x = self.adapool(self.relu(self.bn4(self.conv4(x))))
  # Adapool direkt nach der vierten Schicht

  x = x.view(x.size(0), -1)  # Flattening

  # Daten aus dem zusaetzlichen Pfad
  additional_out = self.relu(self.add_data(add_data))

  # Kombinieren von Daten aus beiden Pfaden
  combined_out = torch.cat((x, additional_out), dim=1)

  # Voll verbundene Schichten
  x = self.relu(self.fc1(combined_out))
  x = self.dropout(x)
  x = self.relu(self.fc2(x))
  x = self.dropout(x)
  x = self.relu(self.fc3(x))
  x = self.relu(self.fc4(x))  # ReLU nach fc4
  x = self.fc5(x)             # Verwendung von fc5
```

[18] Interne Kovarianzverschiebung bezieht sich auf die Veränderung der Verteilung der Aktivierungen innerhalb des Netzwerks während des Trainings.

[19] Padding in CNN bezieht sich auf das Hinzufügen zusätzlicher Zeilen und Spalten an den Rändern eines Eingabebildes oder einer Merkmalskarte, z. B. nullen, um die Größe der Ausgabe zu steuern.

[20] *Flattening* ist ein Prozess in der Datenverarbeitung, bei dem mehrdimensionale Arrays, wie 2D-Bilder oder 3D-Merkmalskarten, in eindimensionale Vektoren konvertiert werden.

```
    # Entfernen der Singleton-Dimension
    return x.squeeze(1)
```

Im Vorwärtspass werden die Eingabedaten x sequenziell durch mehrere konvolutionelle Schichten geführt, jede gefolgt von einer Batch-Normalisierung und einer ReLU-Aktivierungsfunktion. Die Pooling-Schichten, insbesondere eine adaptive Durchschnittspooling-Schicht nach der vierten Konvolution, reduzieren die räumlichen Dimensionen der Merkmalskarten und standardisieren die Ausgabegröße. Anschließend werden die Daten durch die Operation *view(x.size(0), -1) geflattet*. Dies ermöglicht die Umwandlung der mehrdimensionalen Feature-Maps in einen eindimensionalen Vektor.

Parallel zur Verarbeitung der bildbasierten Daten werden zusätzliche Daten in einer separaten linearen Schicht bearbeitet, die durch eine weitere ReLU-Aktivierung ergänzt wird. Diese Verarbeitung integriert weitere, nicht bildbasierte Informationen, die dann mit den geflatteten Bildmerkmalen kombiniert werden (*torch.cat*). Diese integrierten Daten durchlaufen daraufhin fünf voll verbundene Schichten, die durch zwei Dropout-Schichten ergänzt werden. Die finale Ausgabe wird durch die letzte voll verbundene Schicht erzeugt, wobei *squeeze(1)* verwendet wird, um Dimensionen ohne Informationsgehalt zu entfernen.

Die nachfolgende Trainingsfunktion organisiert den Lernprozess, wobei die Daten in *Mini-Batches* durch das Modell verarbeitet und die Modellgewichte entsprechend aktualisiert werden:

```
def train(model, train_loader, criterion, optimizer, device):
  model.train()
  running_loss = 0.0
  all_preds = []
  all_labels = []
  for inputs, add_data, labels in train_loader:
    inputs = inputs.to(device)
    add_data = add_data.to(device)
    labels = labels.to(device)
    outputs = model(inputs, add_data)

    optimizer.zero_grad()

    loss = criterion(outputs, labels)
    loss.backward()

    # Gradient Clipping vor dem Optimierungsschritt
    clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()
    running_loss += loss.item() * inputs.size(0)
    all_preds.append(outputs.detach().cpu())
    all_labels.append(labels.detach().cpu())
```

```
    epoch_loss = running_loss / len(train_loader.dataset)
    all_preds = torch.cat(all_preds)
    all_labels = torch.cat(all_labels)
    train_mae = mean_absolute_error(all_labels.numpy(), all_preds.numpy())
    train_r2 = r2_score(all_labels.numpy(), all_preds.numpy())

    return epoch_loss, train_mae, train_r2
```

Zu Beginn jeder Iteration wird das Modell in den Trainingsmodus versetzt, dies aktiviert die Dropout- und Batch-Normalisierungsschichten. Die Variable *running_loss* berechnet die Summe der Verluste über alle Batches. Aus dieser wird am Ende jeder Epoche der durchschnittliche Verlust ermittelt. Im Kern der Funktion durchläuft ein Loop alle Batches der Trainingsdaten, wobei jeder Batch aus Eingabedaten, zusätzlichen Daten und den zugehörigen Labels besteht. Diese Daten werden auf die GPU geladen. Das Modell berechnet dann die Vorhersagen (*outputs*) für die gegebenen Eingaben und zusätzlichen Daten. Anschließend wird der Optimierer zurückgesetzt (*optimizer.zero_grad()*), um sicherzustellen, dass keine Gradienten von vorherigen Trainingsdurchläufen übrig bleiben. Die Verlustfunktion (*criterion*) berechnet den Fehler zwischen den Vorhersagen und den tatsächlichen Labels. Der Rückwärtsdurchlauf (*loss.backward()*) leitet die Gradienten zurück durch das Netzwerk, um die Gewichte anzupassen. Um die Stabilität des Trainings zu gewährleisten, wird der Gradient Clipping-Mechanismus angewendet, der die Größe der Gradienten zur Verhinderung der *Gradientenexplosion* beschränkt. Nachdem die Gradienten berechnet wurden, aktualisiert der Optimierungsschritt (*optimizer.step()*) die Gewichte des Modells. Die Verlustwerte werden akkumuliert, und die Vorhersagen sowie die Labels werden gespeichert, um die Trainingsmetriken nach der Verarbeitung aller Batches zu berechnen. Der mittlere absolute Fehler (*train_mae*) und das Bestimmtheitsmaß (*train_r2*) werden berechnet, um die Leistung des Modells zu bewerten. Die Validierungsfunktion unterscheidet sich nur darin, dass hier die Unterschiede nicht durch das Netzwerk propagiert werden. Auf eine Darstellung wird daher verzichtet.

Die Gewichtsinitialisierung der Netzwerkschichten erfolgt durch die He-Initialisierung (auch bekannt als Kaiming-Initialisierung), die speziell für Netzwerke mit ReLU-Aktivierungsfunktionen entwickelt wurde. Diese Methode initialisiert die Gewichte einer Schicht aus einer Normalverteilung mit einem Mittelwert von 0 und einer Standardabweichung von $\sqrt{\frac{2}{n}}$, wobei n die Anzahl der Eingaben der betrachteten Schicht darstellt:

```
def init_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        torch.nn.init.kaiming_normal_(m.weight, nonlinearity='relu')
    if m.bias is not None:
        torch.nn.init.zeros_(m.bias)
```

Diese Initialisierungstechnik trägt dazu bei, die Probleme des verschwindenden und explodierenden Gradienten während des Trainingsprozesses zu reduzieren. Die optionale Zeile zur Bias-Initialisierung stellt sicher, dass ein vorhandener Bias-Tensor auf null gesetzt wird, um konsistente Startbedingungen zu gewährleisten.

6.7 Implementierung des genetischen Algorithmus

In diesem Kapitel wird dargelegt, wie der GA implementiert wurde. Der Code baut auf die bisher eingeführten Funktionen in Kapitel 6.2 und insbesondere die Funktionen `create_layouts()`, `check_collision()` und `is_unique_layout()` auf. Diese erstgenannte Funktion wurde erweitert, sodass diese ein bestehendes Layout klonen oder eine Mutation auf Korrektheit überprüfen kann.

```
def select_best_layouts(layouts, fitness, num_best):
  sorted_layouts = sorted(zip(layouts, fitness), key=lambda x: x[1])
  return [layout for layout, _ in sorted_layouts[:num_best]]
```

Die Funktion `select_best_layouts()` dient zur Auswahl der besten Layouts aus einer gegebenen Liste von Layouts basierend auf ihren jeweiligen Fitnesswerten. Sie nimmt drei Parameter: `layouts`, `fitness` und `num_best`. `layouts` ist eine Liste von Layoutvarianten, `fitness` ist eine Liste von Fitnesswerten und `num_best` gibt die Anzahl der Layouts an, die ausgewählt werden sollen. Die Funktion verbindet zuerst jedes Layout mit seinem Fitnesswert, sortiert diese Paare dann nach dem Fitnesswert und extrahiert die `num_best` besten Layouts aus dieser sortierten Liste.

Die Funktion `mutate_layout()` führt eine Mutation auf einem gegebenen Layout durch.

```
def mutate_layout(layout, num_points, mutation_span):
  mutated_layout = layout[:]
  for _ in range(num_points):
    mutate_index = random.randint(0, len(layout) - 1)
    new_x = max(0, min(BG_SIZE[0], mutated_layout[mutate_index][0] + random.
                                        randint(-mutation_span, mutation_span)
                                        ))
    new_y = max(0, min(BG_SIZE[1], mutated_layout[mutate_index][1] + random.
                                        randint(-mutation_span, mutation_span)
                                        ))
    mutated_layout[mutate_index] = (new_x, new_y)

  is_valid = is_valid_layout(mutated_layout, MIN_DIST)
  return mutated_layout if is_valid else layout

def is_valid_layout(centers, min_dist):
  # Check, ob das Layout gueltig
  for i in range(len(centers)):
```

```
    for j in range(i + 1, len(centers)):
      if abs(centers[i][0] - centers[j][0]) < SQUARE_SIZE + min_dist and abs
                                        (centers[i][1] - centers[j][1]) <
                                        SQUARE_SIZE + min_dist:
        return False
  return True
```

Die Funktion `mutate_layout()` hat drei Eingabeparameter:`layout`, `num_points` und `mutation_span`. `layout` ist die Layoutvariante, die mutiert wird, `num_points` gibt die Anzahl der Mutationspunkte im Layout an, und `mutation_span` definiert die Größe der Mutation. Betroffen von der Mutation ist die Position der OE innerhalb des angegebenen Mutationsbereichs. Nach der Durchführung der Mutationen wird das mutierte Layout zurückgegeben, sofern es die Validitätskriterien erfüllt (durch `is_valid_layout` überprüft); andernfalls wird das ursprüngliche Layout zurückgegeben. Die Validitätsprüfung untersucht, ob durch die Mutation sich die Position der OE überlappen oder sich außerhalb der definierten Grenzen befinden. Es wurde zudem die nachfolgende Kreuzungsfunktion implementiert:

```
# Erzeugt neue Layouts durch Kombination von Elternlayouts
def crossover(parent1, parent2):
  child = parent1[:len(parent1)//2] + parent2[len(parent2)//2:]
  return child
```

Die dargestellte Kreuzungsfunktion *crossover* wurde entwickelt, um neue Layouts durch die Kombination von zwei Elternlayouts zu erzeugen. Diese Funktion teilt die Layouts der Eltern in der Mitte und kombiniert die erste Hälfte des ersten Elternteils mit der zweiten Hälfte des zweiten Elternteils, um ein neues „Kindlayout“ zu erstellen. Obwohl diese Methode theoretisch dazu beitragen könnte, die Vielfalt der Lösungen zu erhöhen, zeigte sie in den durchgeführten Tests keinen messbaren Effekt auf die Ergebnisse und wurde daher in weiteren Iterationen nicht eingesetzt.

6.8 Experimente

In diesem Abschnitt sind die Vorgehensweise zum Training der in Kapitel 5.3.4 vorgestellten ANN-Architektur und durchgeführte Experimente dokumentiert. Zunächst wird mit dem in Kapitel 5.2.1 eingeführten Layoutgenerator eine Vielzahl verschiedener Layoutvarianten mit zugehörigen Metriken erzeugt. Anschließend wird die in Kapitel 5.2.2 dargelegte Simulation verwendet, um zu jedem Layout den erwartbaren Durchsatz zu bestimmen. Der resultierende Datensatz wird überprüft und dazu verwendet, das ANN auf die Bestimmung des simulierten Durchsatzes zu trainieren, vgl. Abbildung 22.

6.8.1 Datenerzeugung mit Layoutgenerator

Vor Start des Algorithmus werden diesem eine Reihe an Parametern mitgegeben. Zu diesen Parametern gehören:

- Größe der Layoutfläche
- Anzahl an Fabrikeinheiten OE
- Anzahl zulässiger Typen verschiedener Fabrikeinheiten (Eingangslager, Maschine A, Maschine B etc.)
- Größe und Form der OE
- Mindestabstand zwischen den OE
- Mindestabstand der OE vom Rand des Layouts
- Gesamtanzahl zu erzeugender Layouts
- Startwert für den Zufallsgenerator

Der Layoutgenerator wird insgesamt fünfmal verwendet, um Datensätze zu erzeugen die $4 - 8$ OE enthalten. Die übrigen Parameter werden bis auf die Gesamtanzahl der Layouts konstant gehalten. Das heißt alle Layoutvarianten enthalten jeweils ein Eingangs- und Ausgangslager, gleiche Abstände etc. Die Gesamtanzahl der erstellten Layouts richtet sich nach der Komplexität des Layouts, die abhängig von der Anzahl der OE ist. Eine Übersicht über die gesamten Parameter stellt die folgende Tabelle 6 dar:

Tabelle 6: Skalierung der erstellten Layouts nach Komplexität

OE Größe	Randabstand	OE Abstand	Anzahl OE	Anzahl generierter Layouts
4	1	2	4 OE	60.000
4	1	2	5 OE	70.000
4	1	2	6 OE	80.000
4	1	2	7 OE	90.000
4	1	2	8 OE	100.000

Die OE sind über alle Layoutkonfigurationen rechteckig und vier Pixel groß. Der Mindestabstand zwischen OE muss mindestens zwei Pixel enthalten, zum Rand des Layouts ist ein Abstand von einem Pixel gesetzt. Diese Festlegungen erfolgen, um einem Transportmedium den notwendigen Platz freizuhalten.

Alle Layouts platzieren die OE auf einem Layout mit einheitlicher Größe. Diese wurde nach der Layoutkonfiguration mit der höchsten Komplexität ermittelt und liegt bei 128×128 Pixel, vgl. Kapitel 5.2.3.2. Aufgrund der mit der Anzahl der OE zunehmenden ANN-Leistung wurde die Anzahl der erstellten Layouts von 60.000 bei vier OE auf 100.000 Layoutvarianten bei acht OE skaliert. Für jedes der erzeugten Layouts wird die folgende Bewertungsmatrix durch die DES erstellt.

- Transportstrecke, eines Materials von Eintritt in das Layout bis zum Austritt.
- Verweildauer, die ein Material innerhalb eines Layouts hat.
- Durchsatz an Materialien pro simulierte Anzahl an Tagen.

Bei der Simulation werden die Einschwingphase und stationäre Phase berücksichtigt. Die Statistik wird nach dem Erreichen eines eingeschwungenen Zustands zurückgesetzt [46]. Der Übergang zur stationären Phase wird nach der Mittelwertmethode ermittelt. Eine weiterführende Erläuterung hierzu folgt in Kapitel 7.2.3.

6.8.2 Dokumentation der Experimente

Bei ersten Trainingsdurchläufen wurden die Modelle mit Datensätzen trainiert, die jeweils eine einzige Fabrikkonfiguration z. B. aus fünf Maschinen enthalten. Aufgrund geringer Lernfortschritte wurde der Lernprozess analysiert. Folgende Parameter haben einen Einfluss auf die Lernfähigkeit gezeigt:

- Lernrate des Modells
- Größe der Batches
- Poolgröße des CNN

In den nachfolgenden Abschnitten wird dargestellt, wie für die aufgelisteten Parameter optimierte Werte ermittelt wurden. Da für die Parameter von ANN im Zusammenhang mit dem FLP keine Tabellen- oder Erfahrungswerte existieren, wurden diese experimentell ermittelt.

6.8.2.1 Ermittlung einer optimierten Lernrate

Um die optimale Lernrate zu finden wird nach einem in Leslie N. Smith in seinem Artikel „Cyclical Learning Rates for Training Neural Networks“ vorgestellten Konzept vorgegangen [226]. Ziel ist die Identifizierung derjenigen Lernrate, bei der das ANN am Effizientesten lernt und die Verlustfunktion am Schnellsten abnimmt. Zur Ermittlung einer initialen Lernrate wird eine Bandbreite von 10^{-10} bis 10^{0} untersucht, siehe:

```
# Lernratenfinder
def find_learning_rate(net, criterion, optimizer, train_loader, device):
  lr_start=1e-7
  lr_end=1
  lr_multiplier = (lr_end / lr_start) ** (1 / len(train_loader))

  losses = []
  lrs = []

  net.train()
```

```
pbar = tqdm(enumerate(train_loader, start=1), total=len(train_loader),
                                        desc="LR Finder")
for batch_idx, (inputs, labels) in pbar:
  inputs, labels = inputs.to(device), labels.to(device)

  optimizer.zero_grad()
  outputs = net(inputs)
  loss = criterion(outputs, labels)
  loss.backward()
  optimizer.step()

  losses.append(loss.item())
  lrs.append(optimizer.param_groups[0]['lr'])

  for pg in optimizer.param_groups:
    pg['lr'] *= lr_multiplier

  if optimizer.param_groups[0]['lr'] > lr_end:
    break

return lrs, losses
```

Die Lernrate wird zunächst mit einem sehr kleinen Wert (`lr_start`) initialisiert und schrittweise bis zum Endwert (`lr_end`) erhöht. Der Multiplikator für die Lernrate pro Schritt wird basierend auf der Anzahl der Batches berechnet. Das ANN wird in den Trainingsmodus versetzt. Für jeden Batch werden die Eingaben und Labels auf die GPU verschoben und der Optimierer wird zurückgesetzt. Ein Vorwärtsdurchlauf wird durchgeführt, um die Ausgaben des Netzwerks zu berechnen. Die Verlustfunktion wird angewendet und die Werte werden durch das ANN zurückpropagiert während der Optimierer die Gewichte aktualisiert. Der aktuelle Verlustwert und die Lernrate werden aufgezeichnet. Anschließend wird die Lernrate mit dem zuvor berechneten Multiplikator multipliziert. Der Prozess wird abgebrochen, wenn die Lernrate den Endwert überschreitet. Die Funktion gibt eine Liste der verwendeten Lernraten und die entsprechenden Verluste zurück, die zur Analyse und Auswahl einer geeigneten Lernrate verwendet werden können. Abbildung 29 zeigt den ermittelten Lernratenverlauf.

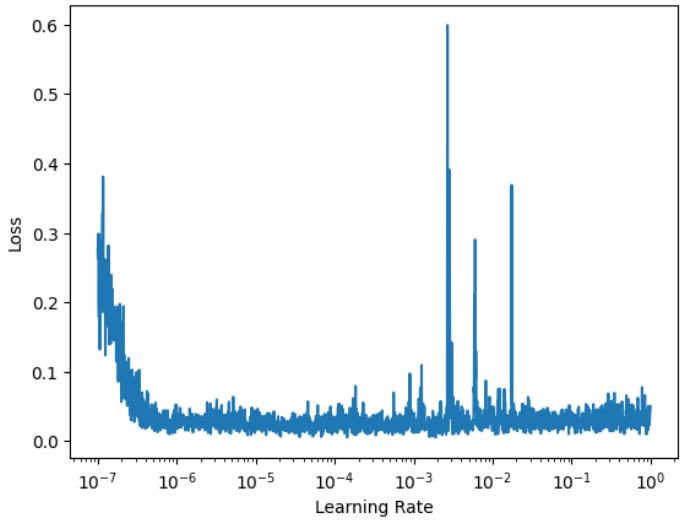


Abbildung 29: Verlauf der Lernraten im Versuchsaufbau

Eine grafische Bestimmung einer optimierten Lernrate ist aufgrund des Verlaufs allein nicht möglich. Mithilfe der nachfolgenden Funktion lässt sich aus den Ergebnissen die Lernrate mit der besten Konvergenz bestimmen.

```
def find_best_lr(lrs, losses):
  # Berechne die Ableitung der Verluste
  gradient = np.gradient(losses)

  # Finde den Index des maximalen Gradienten
  idx_max_gradient = np.argmax(gradient)

  # Ermittlung der Lernrate kurz vor diesem Punkt
  best_lr = lrs[idx_max_gradient - 1]

  return best_lr
```

Durch die Berechnung des Gradienten (der Ableitung) der Verluste wird die Veränderungsrate des Verlusts in Bezug auf die Lernrate quantifiziert. Im nächsten Schritt wird der Punkt mit dem maximalen Gradienten ermittelt, der die Lernrate mit der höchsten Steigung des Verlusts darstellt. Anschließend wird eine Lernrate kurz vor diesem Punkt ausgewählt. Diese Methode basiert auf der Annahme, dass ein steiler Abfall des Verlusts darauf hinweist, dass der Verlust wahrscheinlich bald wieder ansteigen wird. Dies bedeutet, dass das Netzwerk anfangen könnte zu oszillieren oder zu divergieren. Im Ergebnis wurde eine nah optimale Lernrate von 0.002633 identifiziert, die als Startwert festgelegt wird. Bei einer Stagnierung der Verbesserungen wird die Lernrate von diesem Wert ausgehend dynamisch angepasst. Dazu wird eine Methode namens *ReduceLROnPlateau* eingesetzt. Analog

zur wörtlichen Übersetzung wird die Lernrate reduziert, wenn der Validierungsverlust auf einem Plateau ist und sich über eine bestimmte Anzahl von Epochen hinweg nicht verbessert hat.

6.8.2.2 Ermittlung einer optimierten Batch-Größe

Mit dem nachfolgenden Code wird eine optimierte Batch-Größe ermittelt.

```
batch_sizes = [1, 2, 4, 8, 16, 32, 64, 128]  # Inklusive Online-Lernen
pool_size = (4, 4)
avg_val_losses, epochs_per_batch, final_val_losses, loss_histories = [], [],
                                                    [], []
patience, best_val_loss, early_stopping_counter = 4, float('inf'), 0

for batch_size in batch_sizes:
  num_epochs = 10
  epochs_per_batch.append(num_epochs)
  train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=
                                          True, pin_memory=use_cuda)
  val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False,
                                           pin_memory=use_cuda)
  model = Layout_Evaluator(pool_size=pool_size).to(device)
  model.apply(init_weights)
  model.train(True)
  optimizer = optim.Adam(model.parameters(), lr=0.001)
  criterion = nn.MSELoss()
  avg_val_loss = 0.0
  batch_val_loss_history = []
  best_val_loss = float('inf')

  for epoch in range(num_epochs):
    train_loss, _, _ = train(model, train_loader, criterion, optimizer,
                                          device)
    val_loss, _, _ = validate(model, val_loader, criterion, device)
    avg_val_loss += val_loss
    batch_val_loss_history.append(val_loss)
    if val_loss < best_val_loss:
      best_val_loss = val_loss
      early_stopping_counter = 0
    else:
      early_stopping_counter += 1
      if early_stopping_counter == patience:
        break

  loss_histories.append(batch_val_loss_history)
  avg_val_loss /= (epoch + 1)
  avg_val_losses.append(avg_val_loss)
```

```
    final_val_losses.append(val_loss)

best_batch_size = batch_sizes[avg_val_losses.index(min(avg_val_losses))]
```

Der Code initialisiert eine Liste von möglichen Batch-Größen, die von 1 (Online-Lernen[21]) bis 128 reichen. Für jede Batch-Größe werden die Anzahl der Epochen festgelegt, Datenloader erstellt, sowie das Modell und die Optimierer initialisiert. Der Trainings- und Validierungsprozess wird für die festgelegte Anzahl von Epochen durchlaufen, wobei der Validierungsverlust nach jeder Epoche aufgezeichnet wird. Eine *Early Stopping*-Bedingung bricht das Training ab, wenn sich der Validierungsverlust über vier Epochen nicht verbessert. Nach dem Training wird der durchschnittliche Validierungsverlust für die aktuelle Batch-Größe berechnet und gespeichert. Abschließend wird die Batch-Größe mit dem geringsten durchschnittlichen Validierungsverlust als die beste gespeichert. Abbildung 30 zeigt den Verlauf der Auswirkungen unterschiedlicher Batch-Größen auf den Validierungsverlust (linke Achse) und die Trainingsdauer (rechte Achse) auf.

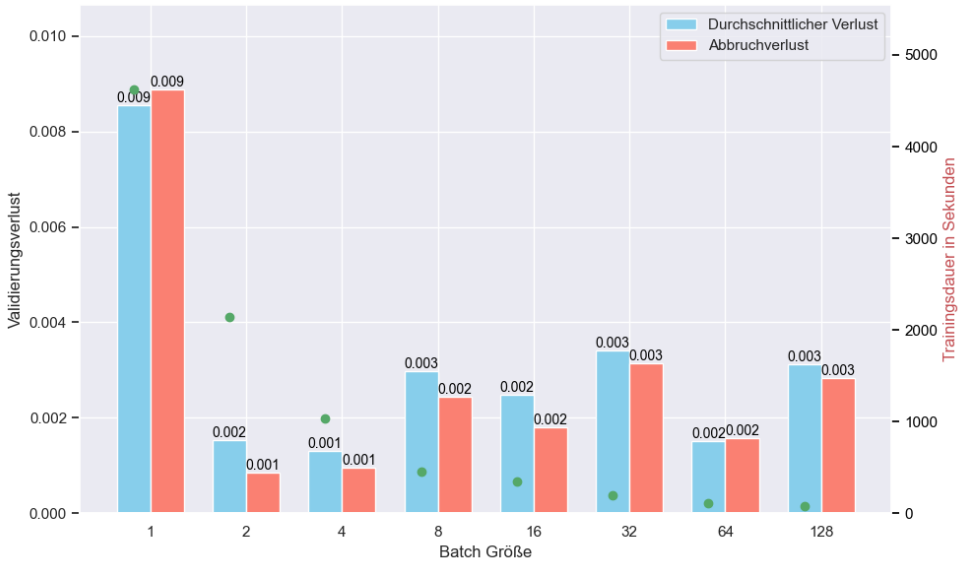


Abbildung 30: Auswirkungen unterschiedlicher Batch-Größen

Das Online-Learning scheidet aufgrund des schlechtesten Validierungsverlusts und der höchsten Trainingsdauer als Batch-Größe aus. Die weiteren Batch-Größen unterscheiden sich marginal bezüglich des Validierungsverlusts, im Hinblick auf die Zeitdauer (grüner Punkt - rechte Achse) ist eine höhere Dauer bei kleineren Batch-Größen erkennbar. In Abbildung 31 ist der Verlustverlauf zu den Batch-Größen über die trainierten Epochen dargestellt.

[21] Definiert in Kapitel 3.3.3.4

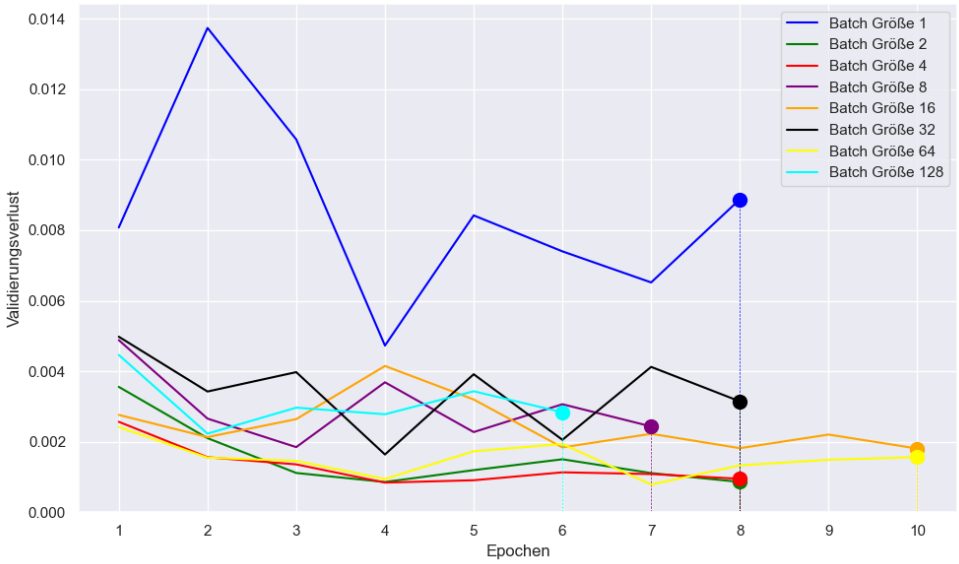


Abbildung 31: Verlustverlauf unterschiedlicher Batch-Größen

Auf Basis dieser Untersuchungen wird für das Modelltraining eine Batch-Größe von 64 ausgewählt. Mit dieser wird ein niedriger Validierungsverlust bei einer kurzen Trainingsdauer erreicht.

6.8.2.3 Ermittlung einer optimierten Pool-Größe

Neben der Batch-Größe wurde auch die Optimierung der Pool-Größe behandelt.

```
pool_sizes = [(4, 4), (5, 5), (6, 6), (7, 7)]
val_loss_history = {size: [] for size in pool_sizes}
best_pool_size, overall_best_val_loss = None, float('inf')
patience, num_epochs, learning_rate = 5, 12, 0.001
avg_val_losses, final_val_losses, pool_best_val_losses = [], [], []

for size in pool_sizes:
  model = Layout_Evaluator(pool_size=size).to(device)
  model.apply(init_weights)
  optimizer = optim.Adam(model.parameters(), lr=learning_rate)
  criterion = nn.MSELoss()
  current_best_val_loss, early_stopping_counter, avg_val_loss = float('inf')
                                        , 0, 0.0
  epoch_loss_history = []

  for epoch in range(num_epochs):
    train_loss, _, _ = train(model, train_loader, criterion, optimizer,
                                        device)
```

```
    val_loss, _, _ = validate(model, val_loader, criterion, device)
    epoch_loss_history.append(val_loss)
    avg_val_loss += val_loss
    if val_loss < current_best_val_loss:
      current_best_val_loss = val_loss
      early_stopping_counter = 0
      if val_loss < overall_best_val_loss:
        overall_best_val_loss = val_loss
        best_pool_size = size
    else:
      early_stopping_counter += 1
      if early_stopping_counter == patience:
        break

  epoch_loss_history += [None] * (num_epochs - len(epoch_loss_history))
  val_loss_history[size] = epoch_loss_history
  pool_best_val_losses.append(current_best_val_loss)
  avg_val_losses.append(avg_val_loss / (epoch + 1))
  final_val_losses.append(val_loss)
```

Pool-Größen in der Bandbreite von $(4,4) - (7,7)$ werden zur Ermittlung derjenigen mit dem geringsten Validierungsverlust getestet. Für jede Pool-Größe wird ein Modell initialisiert, trainiert und validiert. Der Validierungsverlust wird nach jeder Epoche aufgezeichnet. Eine (*Early Stopping*)-Bedingung beendet das Training vorzeitig, wenn sich der Validierungsverlust nicht verbessert. Der durchschnittliche Validierungsverlust und der beste Validierungsverlust für jede Pool-Größe werden gespeichert. Die Pool-Größe mit dem geringsten durchschnittlichen Validierungsverlust wird als die beste Pool-Größe ausgewählt. In Abbildung 32 sind die jeweiligen Verluste zu den Pool-Größen dargestellt und in Abbildung 33 die Verläufe über die Epochen.

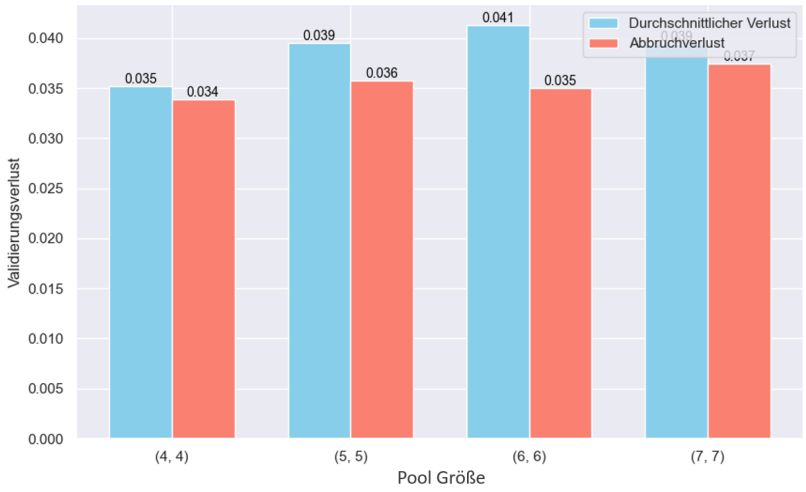


Abbildung 32: Verluste unterschiedlicher Pool-Größen

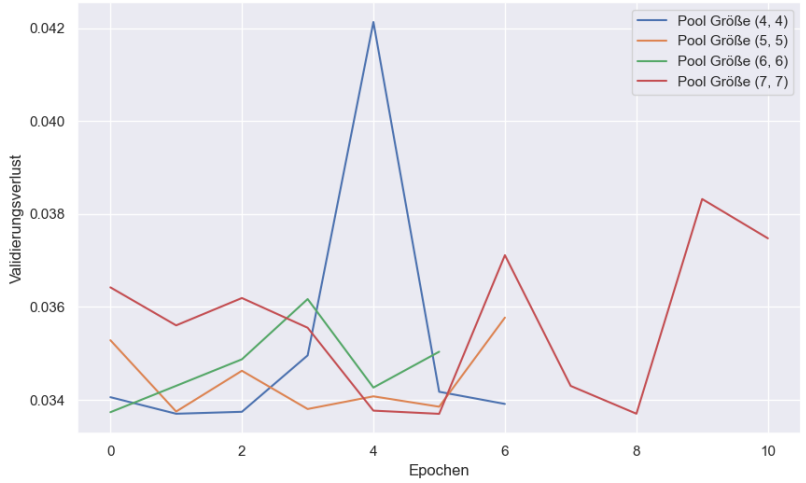


Abbildung 33: Verlustverlauf unterschiedlicher Pool-Größen

Die grafische Analyse zeigt außer bei der Pool-Größe $(4, 4)$ stabile Trainingsverläufe auf.

6.8.3 Training des künstlichen neuronalen Netzwerks

Das Modell wird mit den optimierten Parametern bis zum Early Stopping trainiert. Es wird folgende Nullhypothese formuliert:

> „Das neuronale Netz schätzt den durchschnittlichen Durchsatz gemessen am r^2-Wert schlechter als ein Baseline-Modell ein.“

Die Nullhypothese wird zurückgewiesen, wenn das Modell zuverlässig bessere Ergebnisse als das Baseline-Modell bestimmt. Die folgenden Abbildungen 34 - 35 zeigen einen Trainingsverlauf mit konstanten Verbesserungen.

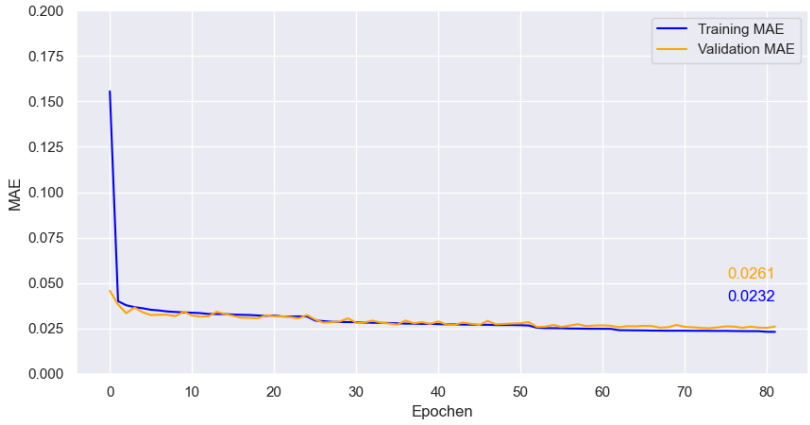


Abbildung 34: MAE-Verlauf über die Trainingsepochen

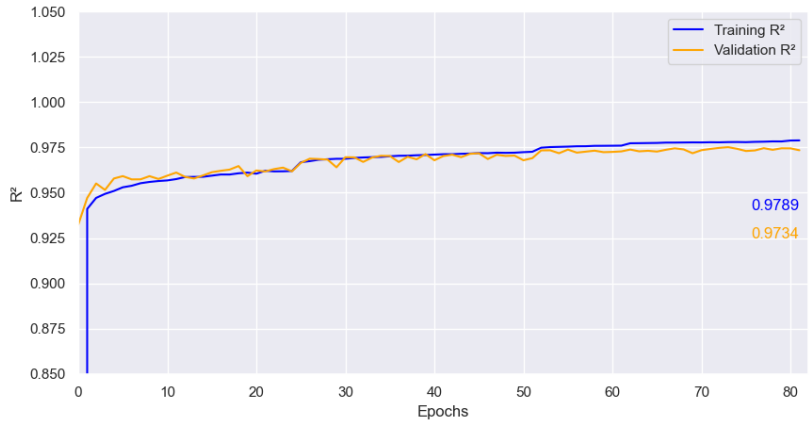


Abbildung 35: r^2-Verlauf über die Epochen

Das Modell hat nach einer Trainingsdauer von 82 Epochen und einem r^2-Wert von > 0.95 die Nullhypothese zurückgewiesen.

7 Evaluierung und Validierung

In Kapitel 5.3 wurde die entwickelte KI-Architektur vorgestellt und in Kapitel 6 die zugehörigen Experimente erläutert. Im nun folgenden Kapitel wird überprüft, ob das Modell geeignet ist, die mit den Forschungsfragen einhergehenden Anforderungen zu erfüllen. Die Evaluierung und Validierung der Forschungsergebnisse orientieren sich an der im Folgenden aufgestellten Hypothese:

> „Das entwickelte künstliche neuronale Netzwerk ist **geeignet**, die Simulation abzubilden und ist dabei **performanter**."

Zur Feststellung der Eignung werden folgende Untersuchungen durchgeführt:

- Ist das neuronale Netz im Hinblick auf die Forschungsfragen zum Einsatz in der Layoutoptimierung geeignet?
- Permutationsanalyse, Vergleich mit weiteren KI-Methoden, Baseline-Vergleich und Feature-Analyse.
- Skalierung auf realistische Fabrikgrößen.

Neben der Eignung wurde überprüft, ob das ANN architekturbedingte Vorteile gegenüber der Simulation aufweist. Ein Hauptvorteil der in einen direkten Zusammenhang mit limitierten Ressourcen der NP-Vollständigkeit einhergeht ist die Performance. Ein Vorteil in diesem Bereich würde die Forschung zum FLP durch die Möglichkeit der Integration von mehr Randbedingungen bei verkürzten Rechenzeiten voranbringen. Um dies zu untersuchen wurden folgende Fragestellungen aufgestellt:

- Welche ist die kleinste, notwendige Dauer bis zum eingeschwungenen Bereich der ereignisorientierten Simulation?
- Ist das neuronale Netz performanter als die ereignisorientierte Simulation?
- Welchen Einfluss hat die Skalierung der Fabrikkomplexität auf die Leistung?

Im Weiteren erfolgt eine Darstellung der vom Modell erzielten Leistungsmetriken auf Trainings-, Validierungs- und Testdaten in Kapitel 7.1. Darauf aufbauend werden die Ergebnisse im Rahmen der Validierung in Kapitel 7.2 auf ihre Integrität und Verlässlichkeit hin überprüft.

7.1 Evaluierung

Dieses Kapitel beschreibt die durchgeführten Schritte und Ergebnisse zur Evaluierung des Modells. Dies umfasst die Messung der Modell-Performance, eine Auswertung zur Genera-

lisierbarkeit und Ergebnisse zu Wirkungseinflüssen der einzelnen Features. Die Ergebnisse der Experimente in diesem Abschnitt wurden mit verschiedenen *Seed*[22]-Parametern fünfmal wiederholt. Dabei zeigte sich, dass die Ergebnisse sowohl über verschiedene Fabrikkonfigurationen als auch über die verschiedenen Durchführungen reproduzierbar sind.

7.1.1 Ergebnisse des Modells

Abbildung 36 zeigt die Ergebnisse des trainierten Modells auf den Testdatensatz. Insgesamt wurden r^2-Werte von > 0.9 und MSE von < 0.0003 erreicht.

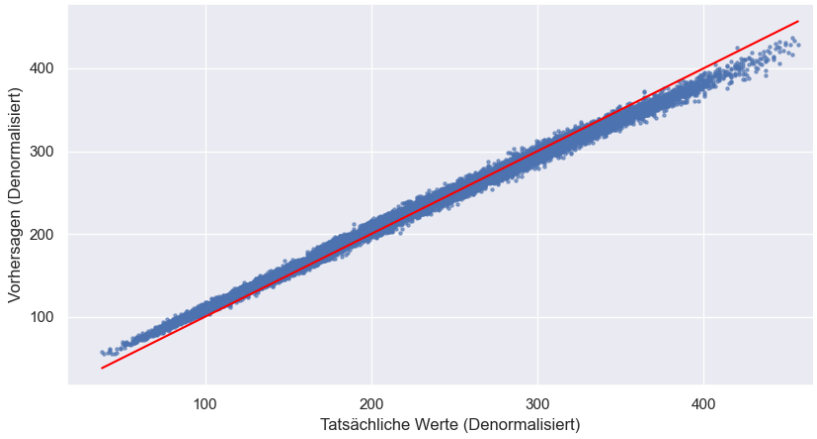


Abbildung 36: Wahre vs vorhergesagte Werte anhand Testdaten (Original Scale)

Die Achsenskalierung wurde in den Originalmaßstab zurückgesetzt. Das globale Minimum über alle Fabrikkonfigurationen beträgt in dem hier dargestellten Setup 38, das globale Maximum 458.

7.1.1.1 Analyse der Residuen

Das in Abbildung 37 dargestellte Histogramm zeigt die Verteilung der Residuen, die als Differenzen zwischen den tatsächlichen Werten und den vorhergesagten Werten definiert sind.

[22] Variable zur Steuerung der Zufälligkeit in den Experimenten.

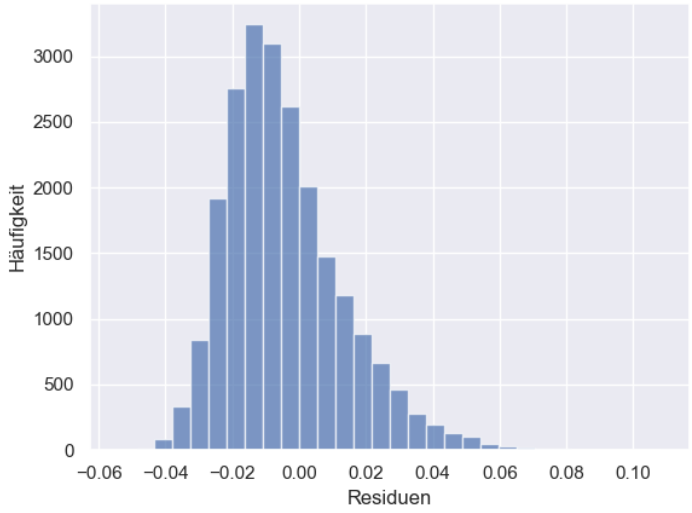


Abbildung 37: Histogramm der Residuen

Das Histogramm zeigt eine leichte Linksverschiebung, beziehungsweise eine leicht negative Verteilung der Residuen (berechneter Mittelwert -0.0044, Standardabweichung 0.0177). Dies deutet darauf hin, dass die tatsächlichen Werte vom Modell tendenziell höher geschätzt werden. Die folgenden statistischen Tests dienen der weiterführenden Analyse zur Verteilung.

Kolmogorov-Smirnov-Test

Der Kolmogorov-Smirnov-Test (KS-Test) ist ein nichtparametrischer Test, der die Verteilung einer Stichprobe mit einer Referenzverteilung vergleicht. In diesem Fall wurde die Verteilung der Residuen mit einer Normalverteilung verglichen. Ein niedriger KS-Statistikwert und ein hoher p-Wert deuten darauf hin, dass die Residuen einer Normalverteilung folgen. Die Ergebnisse ergaben eine Statistik von 0.0723 und einen p-Wert von annähernd 0, was darauf hindeutet, dass die Residuen nicht normalverteilt sind.

Shapiro-Wilk-Test

Der Shapiro-Wilk-Test ist ein weiterer Test zur Überprüfung der Normalverteilung der Residuen. ähnlich wie beim KS-Test impliziert ein hoher p-Wert eine Übereinstimmung mit der Normalverteilung. In dieser Analyse ergab der Shapiro-Wilk-Test einen Statistikwert von 0.9535[23] und einen p-Wert von 0, was ebenfalls die Schlussfolgerung unterstützt, dass die Residuen nicht normalverteilt sind.

[23] Der Shapiro-Wilk-Test ist laut Literatur bei Stichproben > 5000 als Maß nicht zuverlässig.

Schiefe und Kurtosis

Zusätzlich zu diesen Tests wurden die Schiefe und die Kurtosis der Residuen berechnet. Die Schiefe, ein Maß für die Asymmetrie einer Verteilung, ergab einen Wert von 0.9402, was auf eine leichte Linksverteilung hinweist. Die Kurtosis, ein Maß für die „Spitzigkeit“ der Verteilung, wies einen Wert von 1.3052 auf, was auf eine mäßig steilere Verteilung als die Normalverteilung hinweist.

Weitere Tests

Um die Ergebnisse zu bekräftigen, wurden auch der Anderson-Darling-Test (255.5228), der Jarque-Bera-Test (4892.4380 und p-Wert 0.0) sowie der D'Agostino's K^2-Test durchgeführt, mit einem Ergebnis von (3036.7759 und p-Wert 0.0). Diese Tests zeigen ebenfalls an, dass die Residuen nicht einer Normalverteilung folgen.

Schlussfolgerungen der Residuenanalyse

Die durchgeführten Tests und Berechnungen zeigen, dass die Residuen des Modells nicht einer strengen Normalverteilung folgen. Obwohl eine visuelle Analyse des Histogramms eine annähernde Normalverteilung andeutet, widerlegen die statistischen Tests diese Annahme. Dies weist auf nichtlineare Beziehungen oder das Vorhandensein von Ausreißern in den Daten hin. Dem entgegen steht die Berechnung, dass 95.26[24] Prozent der Residuen innerhalb des 95-Konfidenzintervalls liegen. Dieses Ergebnis ist konsistent mit der Erwartung für normalverteilte Daten, wonach etwa 95 Prozent der Werte innerhalb von 1.96 Standardabweichungen vom Mittelwert liegen sollten.

Die Diskrepanz zwischen der visuellen Darstellung im Histogramm und den Ergebnissen der statistischen Tests deutet auf komplexe Strukturen in den Daten, die nicht durch lineare Modelle erfasst werden können. Dies unterstreicht die Notwendigkeit eines ANN zur Abbildung des Problems, da diese nichtlinearen Zusammenhänge modellieren können.

7.1.1.2 Modellleistung über verschiedene Layoutkonfigurationen

In diesem Abschnitt wird das Vorhersageergebnis des ANN von zwei aufeinanderfolgenden Trainingsläufen dargestellt. Die Ergebnisse wurden bei der *Industrial Simulation Conference* im Juni 2024 publiziert [8]. Im ersten Trainingslauf wurde das ANN auf die Vorhersage der Manhattan-Distanz zwischen den einzelnen OE trainiert. Im zweiten Lauf wurde das ANN auf die Vorhersage des simulierten Durchsatzes trainiert. Abbildung 38 zeigt das Vorhersageergebnis bezogen auf die Manhattan-Distanz der Maschinenverteilung über fünf verschiedene Layoutkonfigurationen.

[24] Anzahl der Residuen im 95% Konfidenzintervall: 21347 von 22409

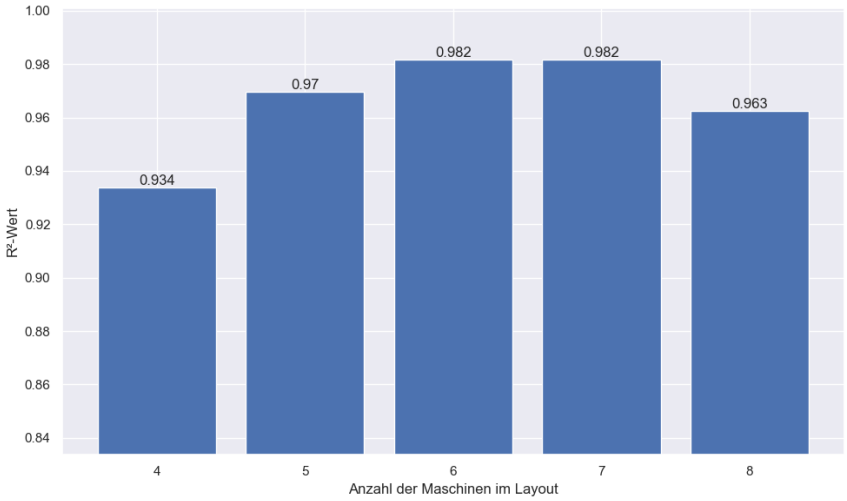


Abbildung 38: ANN-Performance bei Vorhersage der MHC

Jeder Balken stellt das Ergebnis des ANN auf eine Layoutkonfiguration dar. Es wurden Layouts verwendet, die zwischen vier bis maximal acht OE enthalten. Die Untersuchung zeigt über alle Layoutkonfigurationen r^2-Werte über 90 Prozent. Es sind keine erkennbaren Abweichungen zwischen den verschiedenen Konfigurationen aufgetreten. Abbildung 39 zeigt hingegen ein absinkendes Vorhersageergebnis bezogen auf den simulierten Durchsatz.

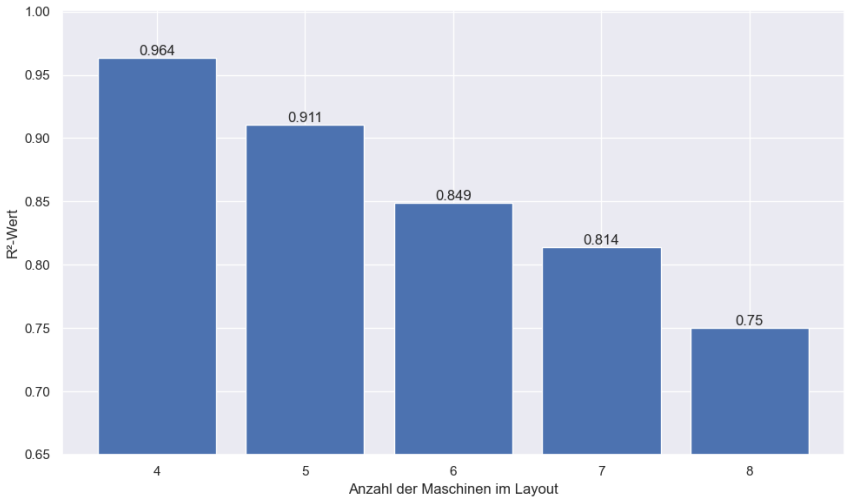


Abbildung 39: Modellleistung nach Layoutkonfigurationen

Die Vorhersage des simulierten Durchsatzes ist von mehr Randbedingungen und Variablen abhängig als die Berechnung der Manhattan-Distanz. Dies wird durch die Abnahme der r^2-Werte bei steigender Komplexität der Layouts deutlich und steht im Gegensatz zur

Manhattan-Distanz, bei der das ANN mit den gleichen Daten konstant akkurate Vorhersagen getroffen hat. Dies zeigt die Notwendigkeit von größeren Datensätzen bei steigender Komplexität der Layouts auf, wenngleich das Modell sich Vorhersageleistungen über alle Konfigurationen erhalten konnte.

7.1.2 Ergebnisse auf unterschiedliche Simulationszeiten

In den Ergebnissen von Abbildung 39 wurden die Layouts mit einer über alle Konfigurationen konstanten Simulationsdauer simuliert. Diese Festlegung basiert auf den Überlegungen von Kapitel 7.2.3. Um die Fähigkeit des ANN das Simulationsergebnis vorherzusagen weiterführend zu untersuchen, wurde ein Versuchsaufbau mit variierenden Simulationsdauern durchgeführt. Abbildung 40 zeigt die Vorhersageergebnisse in der Bandbreite von 5.000s bis 15.000s.

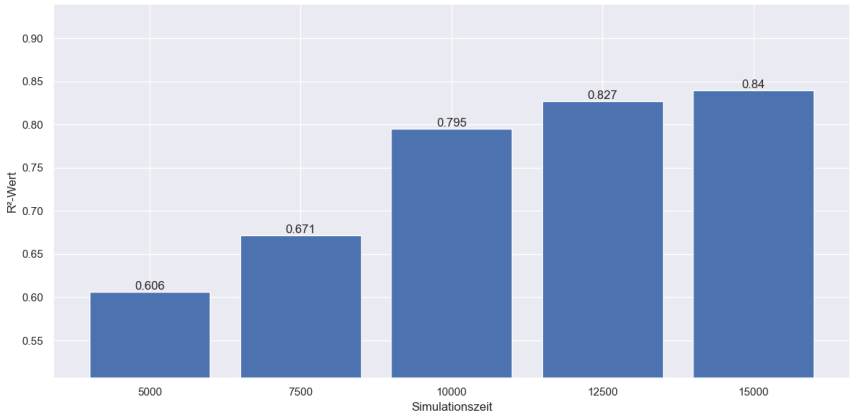


Abbildung 40: Modellleistung auf variierende Simulationsdauern

Auf der x-Achse sind die in den Datensatz aufgenommenen Simulationsdauern, auf der y-Achse die vom ANN erreichten r^2-Werte dargestellt. Dieses kann in diesem Versuchsaufbau nicht ausschließlich anhand des Bildinputs den Durchsatz vorhersagen, sondern muss zusätzlich die tabellarischen zusätzlichen Daten mit den Bildinformationen zusammenbringen. Die Untersuchung zeigt, dass das Modell bessere Ergebnisse für Layouts liefert, die länger simuliert wurden. Dies ist möglicherweise darauf zurückzuführen, dass Simulationen mit einer höheren Dauer sich länger in der stationären Phase befinden. Im Gegensatz dazu zeigen Simulationen mit kürzerer Dauer eine anfängliche Variabilität auf. Dies gilt insbesondere für Layouts mit komplexen Konfigurationen.

7.1.3 Feature Analyse

Die Architektur des ANN ist so konzipiert, dass Layoutbilder und zusätzliche Daten verarbeiten werden. Die nachfolgende Analyse zeigt den spezifischen Einfluss auf. Hierfür wurden die Daten abwechselnd permutiert.

7.1.3.1 Permutationsanalyse - Bilder vs. zusätzliche Daten

Um die Wichtigkeit des jeweiligen Inputs für das Modell zu ermitteln, wurden die Datensätze abwechselnd verfremdet bzw. permutiert. Permutierte Daten enthalten keine Informationen, die einen Rückschluss auf das tatsächliche Simulationsergebnis zulassen. In einem Lauf wurden lediglich die Bilddaten permutiert, in einem weiteren Lauf die zusätzlichen Daten. Abbildung 41 zeigt, wie sich eine isolierte Permutation der Bilddaten im Vergleich zu einer isolierten Permutation der zusätzlichen Daten auf das Bestimmtheitsmaß auswirkten.

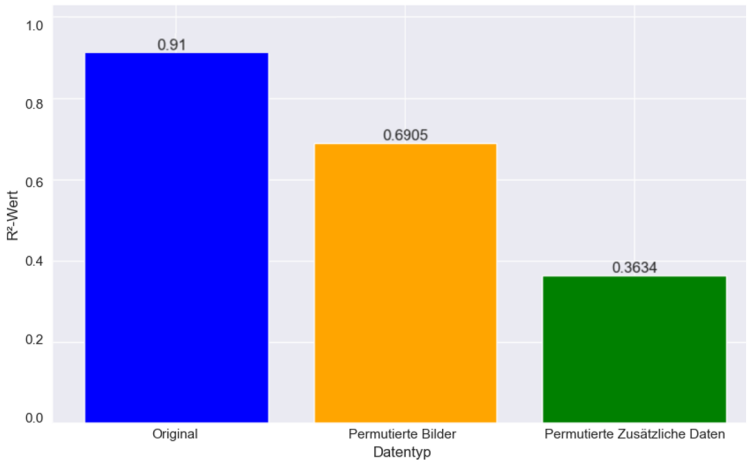


Abbildung 41: Einfluss der Permutation auf die Modellvorhersagen

Ein stärkerer Rückgang im r^2-Wert nach der Permutation eines bestimmten Datentyps deutet darauf hin, dass dieser wichtiger für die Performance ist. Das mit nicht permutierten Daten trainierte ANN erreicht einen r^2-Wert von 0.91 (blauer Balken). Dies zeigt die Fähigkeit des Modells auf, einen großen Teil der Variabilität in den Testdaten zu erklären. Wenn die Bilder permutiert und die tabellarischen Daten unverändert gelassen werden sinkt der r^2-Wert auf 0.6905 (oranger Balken). Durch diesen Rückgang wird deutlich, dass die Bilder eine wichtige Rolle für die Vorhersagegenauigkeit einnehmen. Die Tatsache, dass der Wert immer noch relativ hoch ist, deutet darauf hin, dass das Modell immer noch einige Vorhersagefähigkeiten durch die nicht permutierten zusätzlichen Daten behält. Nach der Permutation der zusätzlichen Daten fällt der r^2-Wert stärker ab auf 0.3634 (grüner Balken). Dieser deutlich niedrigere Wert im Vergleich zum Versuch mit permutierten Bildern

demonstriert eine größere Bedeutung der zusätzlichen Daten für das Modell.

Grundsätzlich wird geschlussfolgert, dass beide Datentypen für die Performance des ANN wichtig sind. Die zusätzlichen Daten zeigen hierbei einen größeren Einfluss, da ihre Permutation zu einem stärkeren Rückgang der Performance führt. Abbildung 42 zeigt ein nicht permutiertes Bild im Vergleich mit einem permutierten Bild. Die Permutation ist derart stark, dass die Platzierung der OE nicht mehr ersichtlich ist. Lediglich die Anzahl bleibt indirekt als Farbcodierung erhalten.

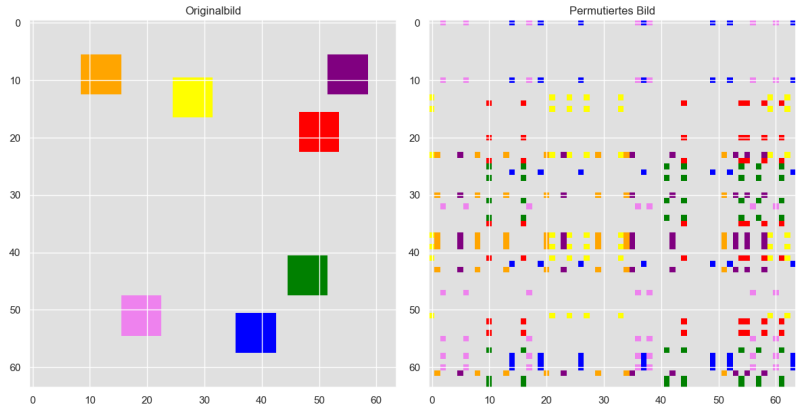


Abbildung 42: Originalbild im Vergleich zum permutierten Bild

Bei den Bildern wurde die Permutation auf Kanalebene durchgeführt. Für jedes Bild im Batch wird zufällig ein Kanal ausgewählt und innerhalb dieses Kanals werden die Pixelpositionen sowohl in der Höhe als auch in der Breite zufällig permutiert. Dies bedeutet, dass die räumliche Struktur des Bildes innerhalb des ausgewählten Kanals stark verändert wird, während die anderen Kanäle unverändert bleiben. Tabelle 7 zeigt ein nicht verändertes Sample der zusätzlichen Daten im direkten Vergleich zu einem permutierten Sample.

Tabelle 7: Originale und permutierte zusätzliche Daten

Feature	Information	Original	Permutiert	Unterschied
1	m1-X	0.82	0.40	-51.22 %
2	m1-Y	0.00	0.82	inf
3	m2-X	0.28	0.34	21.43 %
4	m2-Y	0.40	0.28	-30.00 %
5	m3-X	0.20	0.98	390.00 %
6	m3-Y	0.84	0.70	-16.67 %
7	m4-X	0.70	0.88	25.71 %
8	m4-Y	0.30	0.08	-73.33 %
9	m5-X	0.12	0.98	716.67 %
10	m5-Y	0.20	0.98	390.00 %
11	m6-X	1.00	0.60	-40.00 %
12	m6-Y	0.68	0.62	-8.82 %
13	m7-X	0.30	0.98	226.67 %
14	m7-Y	0.06	0.42	600.00 %
15	m8-X	-1.0	0.00	-100.00 %
16	m8-Y	-1.0	0.28	-128.00 %
17	Count	0.75	1.00	33.33 %
18	Driven Distance	0.556	0.666	19.86 %

Bei den zusätzlichen Daten wird die Permutation auf der Ebene der Datensätze durchgeführt. Jedes einzelne darin enthaltene Feature wird so zufällig permutiert, dass der Informationsgehalt zu den jeweiligen Bildern und Labels verloren geht. In der Folge kann das ANN nur noch aus den in diesem Versuchsaufbau unveränderten Bilddaten genaue Vorhersagen treffen. Durch die Permutation wurden die Daten durchschnittlich zu 71 Prozent von ihrem bisherigen Wert verändert. Insgesamt zeigt dies einen größeren Effekt auf die Genauigkeit der Vorhersagen. Darüber hinaus wird deutlich, dass die Bilder einen verbessernden Faktor darstellen, da die höchsten r^2-Werte durch Zuführung beider Inputs in nicht permutierter Form erzielt wurden. Insgesamt sind die Ergebnisse trotz starker Permutation in beiden Fällen positiv und zeigen in dem trainierten Modell eine hohe Resilienz gegenüber Veränderungen auf. Dies stellt einen systemischen Vorteil für eine mögliche industrielle Anwendung dar.

7.1.4 Isolierte Permutationsanalyse der zusätzlichen Daten

Die im vorherigen Abschnitt durchgeführte Analyse zeigte einen größeren Einfluss der zusätzlichen Daten auf die Prädiktionsgenauigkeit auf. Darauf aufbauend wird der Einfluss einzelner Bestandteile untersucht. Insgesamt enthalten die zusätzlichen Daten 18 Features, die in Tabelle 7 namentlich vorgestellt wurden. Die ersten 16 Features stellen jeweils die x- und y-Position der OE dar, während die zwei weiteren Features die OE-Anzahl und die zurückgelegte Strecke der Transporteinheit beschreiben. Abbildung 43 zeigt das Ergebnis einer Untersuchung, bei der die zusätzlichen Daten einzeln permutiert und ihr Einfluss auf die Modellvorhersagen ermittelt wurde. Die Bildinputs wurden in diesem Versuchsaufbau unverändert gelassen. Um zu vermeiden, dass die ermittelten Wichtigkeiten der Features nicht von einer zufälligen Permutation abhängig sind, wurde der Test mit fünf verschiedenen *Seeds* wiederholt und die Ergebnisse gemittelt. Dies ist rechenintensiv und zeitaufwendig, da das Modell über den gesamten Testdatensatz für jedes permutierte Feature fünfmal ausgewertet wird.

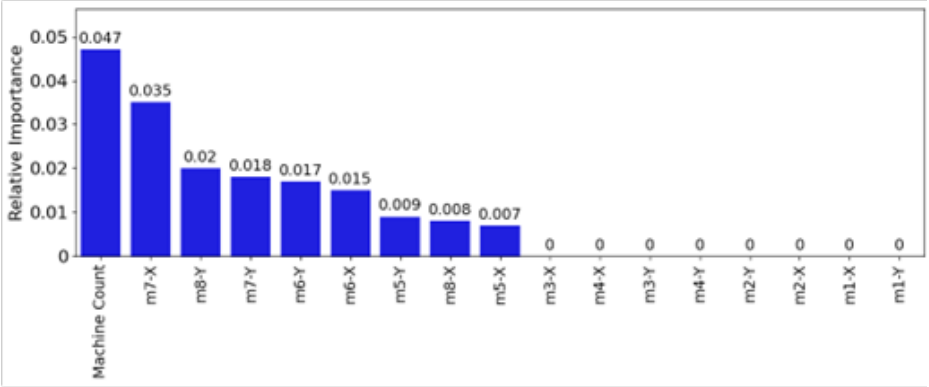


Abbildung 43: Gemittelte relative Wichtigkeiten der zusätzlichen Daten

Auf der x-Achse ist pro Feature ein Balken dargestellt, auf der y-Achse wird zum Aufzeigen des spezifischen Einflusses die Differenz im r^2-Wert (relative Wichtigkeit) dargestellt. Dies ist in diesem Kontext ein Maß dafür, wie viel schlechter die Vorhersagen werden, wenn das bestimmte Feature permutiert wird. Es ist der Unterschied im r^2-Wert zwischen dem Originalmodell und dem Modell mit einem permutierten Feature. Ein größerer Unterschied bedeutet, dass das Feature wichtiger ist.

Die höchste Wichtigkeit nimmt die Anzahl der OE im Layout ein. Dies scheint für die Logik des Modells ein wichtiger Parameter für die korrekte Berechnung des Durchsatzes zu sein. Auffällig ist weiterhin, dass vor allem die Koordinaten der OE in den späteren Stadien des Produktionsablaufs (Maschinen 5-8) eine höhere Wichtigkeit aufweisen als die der früheren Maschinen (Maschinen 1-4). Dies ist auch keine zufällige Beobachtung, da

die Maschinen paarig als entweder wichtig oder unwichtig für in Erscheinung treten. Dies könnte auf folgende Faktoren zurückzuführen sein:

1. **Komplexität des Produktionsablaufs**: In den späteren Stadien des Produktionsablaufs könnten komplexere oder kritischere Prozesse stattfinden, die einen stärkeren Einfluss auf die Zielvariable haben. Das Modell könnte daher sensibler auf Veränderungen in diesen Bereichen reagieren.
2. **Datenstruktur**: Es könnte eine Besonderheit in den Daten vorliegen, die bewirkt, dass Variationen in den Koordinaten der späteren OE stärker mit der Zielvariable korrelieren. Dies könnte auf spezifische Muster oder Zusammenhänge in den Trainingsdaten zurückzuführen sein.
3. **Modellcharakteristika**: Die Architektur oder das Training könnten zu einer bevorzugten Behandlung bestimmter Features führen. Beispielsweise könnten die späteren Koordinatenfeatures aufgrund ihrer Position im Datensatz oder ihrer Beziehung zu anderen Variablen vom Modell als relevanter angesehen werden.

Da die einzelnen OE-Positionen in ihrer Gesamtheit auch als ein Feature verstanden werden können, wurden diese im Weiteren zusammengefasst, siehe Abbildung 44.

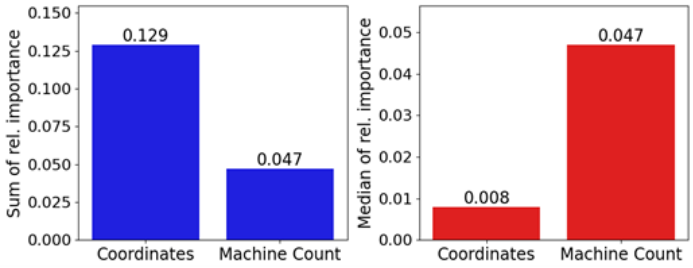


Abbildung 44: Zusammengefasste gemittelte relative Wichtigkeiten der Features

Zwei verschiedene Ansätze wurden verfolgt, um die Wichtigkeit der Koordinaten zu bestimmen: die Summierung (links in blau) und die Durchschnittsbildung (rechts in rot) der Wichtigkeiten aller Koordinatenfeatures. Bei der Summierung werden die Wichtigkeiten aller Koordinatenfeatures zur Bildung der Gesamtwichtigkeit addiert. Im Gegensatz zur Auswertung in Abbildung 43 zeigte sich hier der größte Einfluss der Koordinaten auf die Vorhersagegenauigkeit. Bei der in der rechts dargestellten Auswertung wird der durchschnittliche Einfluss aller Koordinatenfeatures berechnet. Diese Betrachtung zeigte auf, dass einzelne Koordinatenfeatures im Durchschnitt weniger Einfluss haben als die Anzahl der OE.

Die Ergebnisse dieser Analysen offenbaren Einblicke in die Funktionsweise des Modells. Während die Summierung der Wichtigkeiten verdeutlicht, dass die räumliche Anordnung der OE insgesamt von großer Bedeutung ist, zeigt die Durchschnittsbildung, dass einzelne

Koordinaten im Vergleich zu anderen Features wie der Maschinenanzahl weniger Einfluss haben. Dies legt nahe, dass das Modell besonders sensibel auf Veränderungen in der Gesamtanordnung der OE reagiert, während die spezifische Position einzelner OE eine geringere Rolle spielt.

7.2 Validierung

Um zu überprüfen, ob das beschriebene Vorgehen dieser Untersuchung verlässlich ist und den Zweck erfüllt, werden gängige als auch auf den Sachverhalt zugeschnittene Validierungsmethoden durchgeführt. Dazu zählen Vergleiche mit weiteren Prädiktionsmethoden, der Mittelwert-Baseline und einer Zufallsverteilung. Bei der Entwicklung, dem Modelltraining und den Auswertungen wurden Validierungsmethoden zur Datenaufteilung, Early Stopping etc. eingehalten. Weiterführend wurden die verwendeten Daten, der entwickelte Layoutgenerator und die Simulation logischen Tests unterzogen. Die Modellarchitektur und ihre Eignung wurden unter anderem einer *K-Fold Kreuzvalidierung* unterzogen, sowie mit Veränderungen in den Simulationszeiten konfrontiert. Zuletzt erfolgt ein Geschwindigkeitsvergleich zwischen der Simulation und den Vorhersagen des Modells.

7.2.1 Überprüfung der verwendeten Daten

ANN unterliegen im Allgemeinen dem Paradigma das sie durch die Qualität der verwendeten Trainingsdaten limitiert sind und niemals besser werden können. Da in weiten Teilen dieser Untersuchung zur Datenerzeugung eigene Implementierungen verwendet wurden, müssen diese validiert werden.

7.2.1.1 Verifizierung und Debugging des Layoutgenerators

Die Eigenentwicklung des Layoutgenerators bringt den Vorteil einer maßgeschneiderten Lösung mit sich, führt jedoch auch zu Unsicherheiten hinsichtlich der Verlässlichkeit und Qualität der Daten. Die entwickelte Implementierung ist nicht durch eine breite Masse getestet und kann nicht erkannte Bugs / Fehler aufweisen. Um dies zu mitigieren, wurden Funktionen implementiert, um die ordnungsgemäße Funktion zu überprüfen.

Die nachfolgende Funktion ermöglicht eine optische Überprüfung zur Anordnung der Fabrikeinheiten innerhalb eines Layouts. Aus den erzeugten Daten wird jeweils das Layout mit der höchsten und niedrigsten Manhattan-Distanz ausgegeben.

```
def show_min_max_layouts(layouts, manh_dist):
  # Get the index of the layout with the min / max distance
  min_index = manh_dist.index(min(manh_dist))
  max_index = manhattan_distances.index(max(manh_dist))
```

```
    # Get the min / max layout and distances
    min_layout = layouts[min_index]
    min_distance = manh_dist[min_index]
    max_layout = layouts[max_index]
    max_distance = manh_dist[max_index]

    # Show the min /  max layouts and their distances
    display(min_layout)
    display(max_layout)

show_min_max_layouts(layouts, manh_dist)
```

Zunächst identifiziert die Funktion die Indizes der Layouts mit den minimalen und maximalen Manhattan-Distanzen, indem sie die `index`-Methode zusammen mit den Funktionen `min` und `max` verwendet. Sie extrahiert dann die entsprechenden Layouts und ihre Distanzen aus den Eingabelisten. Schließlich gibt die Funktion die Distanzen aus und verwendet die `display`-Funktion, um die Layouts anzuzeigen, s. Abbildung 45.

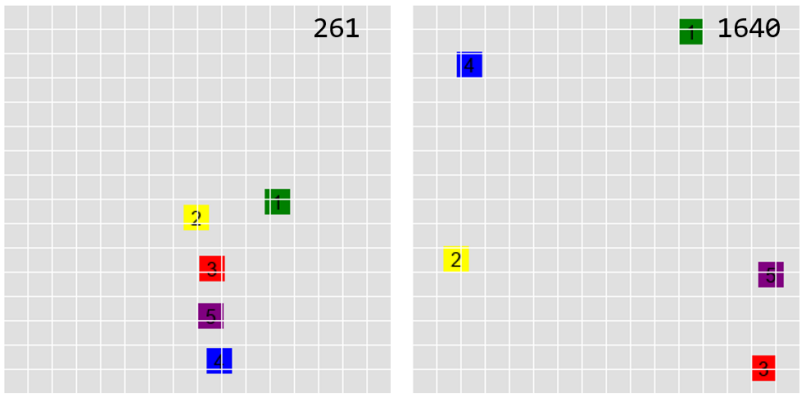


Abbildung 45: Niedrige und hohe Manhattan-Distanz im Vergleich

Anhand der Nummerierung der OE und Gitterlinien im Raster von 20 Pixel wurden stichprobenartig die berechneten Werte visuell überprüft.

7.2.1.2 Verifizierung und Debugging der Layoutsimulation

Die Layoutsimulation wurde ebenfalls visualisiert und optisch auf den korrekten Ablauf untersucht. Im Ergebnis zeigte die Simulation nach mehreren Iterationen zur Behebung von Bugs das erwartete Verhalten und war in der Lage in einer nachvollziehbaren Weise den definierten Produktionsablauf für einen vorgegebenen Zeitraum zu simulieren und entsprechende Metriken zu erfassen. Die vom Layoutgenerator erzeugten Layouts und die

durch die Simulation erstellten Metriken wurden über fünf verschiedene Layoutkonfigurationen zu einem Gesamtdatensatz zusammengefasst. Der folgende Abschnitt beschreibt den weiteren Umgang mit diesen Daten.

7.2.1.3 Datenerzeugung und -aufteilung

Die Datenaufteilung wurde in Kapitel 5.2.3 vorgestellt. Die generierten Daten werden in Trainings-, Test- und Validierungsdaten in folgendem Verhältnis geteilt:

$$\text{Trainingsdaten} : \text{Validierungsdaten} : \text{Testdaten} = 70\% : 15\% : 15\%$$

Es ist sichergestellt, dass das ANN während des Trainings keinen Zugriff auf die separierten Testdaten hat. Darüber hinaus wird das ANN nach jeder Trainingsepoche anhand der Validierungsdaten überwacht. Erst nach Abschluss des Trainings wird die Evaluierung anhand der Testdaten durchgeführt. Der Datensatz enthält fünf verschiedene Fabrikkonfigurationen, von Layouts mit vier OE, bis Layouts mit acht OE. Eine gleichmäßige Verteilung der Konfiguration wurde mit dem nachfolgenden Code überprüft:

```
def check_dataset_distribution(additional_data, label):
  source_column = additional_data[:, -3]
  unique_sources = np.unique(source_column)
  print(f"Verteilung im {label}-Set:")
  for source in unique_sources:
  count = np.sum(source_column == source)
  print(f"Datensatz {source}: {count} Beispiele")

check_dataset_distribution(additional_data_train, 'Trainings')
check_dataset_distribution(additional_data_val, 'Validierungs')
check_dataset_distribution(additional_data_test, 'Test')
```

Tabelle 7 zeigt, dass die verschiedenen Layoutkonfigurationen bei der Aufteilung gleichmäßig in den Trainings-, Validierungs- und Testdatensatz mit nur leichten Varianzen verteilt worden sind.

Tabelle 8: Verteilung der Daten in Trainings-, Validierungs- und Testset

Datensatz	Training	Validierung	Test
4.0	21087	4441	4319
5.0	20856	4522	4493
6.0	20940	4492	4455
7.0	20786	4504	4613
8.0	20897	4453	4529

Analyse von Beziehungen und Abhängigkeiten im Datensatz

Nach erfolgter Sicherstellung der Datenqualität erfolgte eine Untersuchung der Daten hinsichtlich ihrer inneren Struktur, Beziehungen und Abhängigkeiten über eine Korrelationsmatrix, die den Pearson-Korrelationskoeffizienten für jedes Variablenpaar anzeigt, s. Abbildung 46.

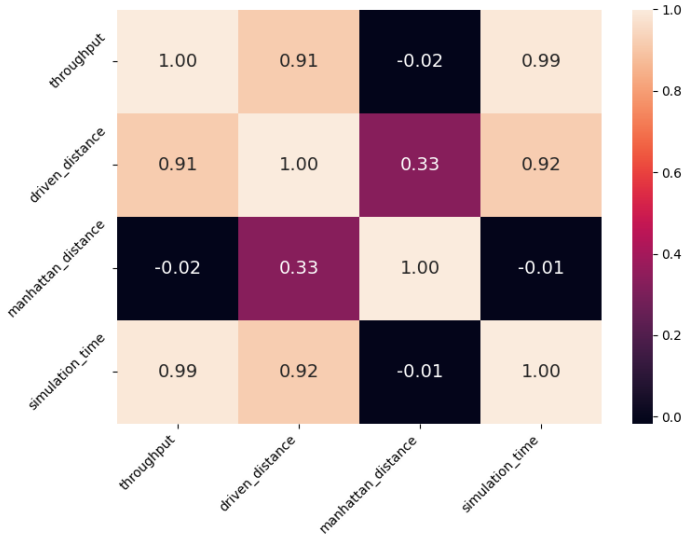


Abbildung 46: Korrelationsmatrix zu den Simulationsdaten

Die Korrelationskoeffizienten liegen im Bereich von -1 bis 1. Ein Wert nahe 1 weist auf eine starke positive Korrelation hin, ein Wert nahe -1 auf eine starke negative Korrelation und ein Wert um 0 deutet auf keine signifikante Korrelation hin. Die ermittelten Korrelationswerte zeigen, dass insbesondere die zurückgelegte Distanz und der Durchsatz korrelieren. Dies ist nachvollziehbar, da ein hoher Durchsatz mit einer erhöhten zurückgelegten Strecke des Transportmediums einhergeht. Darüber hinaus korreliert der Durchsatz stark mit der Simulationsdauer. Auch dies ist schlüssig, da eine längere Simulationsdauer unabhängig von der Fabrikkonfiguration zu einem erhöhten Durchsatz und einer längeren zurückgelegten Strecke führt. Die Manhattan-Distanz und die durchschnittliche Fahrtdistanz weisen eine mäßig starke Korrelation auf. Dies ist darauf zurückzuführen, dass der Datensatz absichtlich auch suboptimale Layouts enthält, wodurch die Manhattan-Distanz nur schwache Rückschlüsse auf die tatsächlich notwendigen Materialbewegungen gemäß dem Produktionsfluss zulässt. Beispielsweise könnte eine große Manhattan-Distanz durch einen großen Abstand zwischen zwei OE verursacht werden. Wenn diese beiden OE jedoch seltener angefahren werden als die übrigen OE, die günstiger positioniert sind, verringert sich die Korrelation zwischen diesen beiden Größen. Dies zeigt auch der nachfolgende Scatterplot zwischen der Manhattan-Distanz und der gefahrenen Strecke in Abbildung 47

auf:

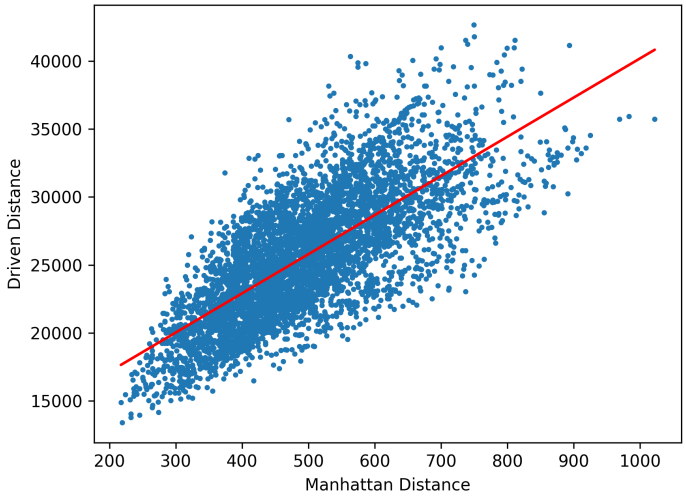


Abbildung 47: Scatterplot zur Manhattan- und gefahrenen Distanz

Die aufgezeigte Abhängigkeit ist schwach korreliert, zeigt aber das ordnungsgemäße Erfassen der Daten während der Simulation.

7.2.1.4 Duplikateprüfung

Duplikate in den Trainingsdaten können zu einer Überanpassung des Modells führen und dessen Fähigkeit zur Generalisierung auf andere Fabrikkonfigurationen beeinträchtigen, indem das ANN dazu verleitet wird, nur für duplizierte Layouts zutreffende Muster zu lernen, anstelle allgemeiner Zusammenhänge zum Produktionsablauf. Zur Sicherstellung der Einzigartigkeit erzeugter Layouts wurde die Funktion `check_for_duplicates` implementiert.

```
def check_for_duplicates(images):
  unique_images = set()
  dup_indices = []
  for i, img in enumerate(images):
    img_hash = hash(img.tobytes())
    if img_hash not in unique_images:
      unique_images.add(img_hash)
    else:
      dup_indices.append(i)
    if len(dup_indices) == 0:
      print("No duplicates found!")
    else:
      print(f"{len(dup_indices)} duplicates at  {dup_indices}.")
```

```
check_for_duplicates(layouts)
```

Die Funktion nutzt die Eigenschaften von Python-Sets und Hashing, um Duplikate zu identifizieren. Zunächst werden ein leeres Set `unique_images` und eine leere Liste `duplicate_indices` initialisiert. Dann erfolgt ein Durchlauf der Eingabebilder, bei dem für jedes Bild ein eindeutiger Hash-Wert anhand der Byte-Darstellung berechnet wird. Ein Duplikat ist erkannt, wenn ein Hash-Wert bereits im Set vorhanden ist. Der Index vom gefundenen Duplikat wird zur Liste `duplicate_indices` hinzugefügt.

Hinsichtlich der zusätzlichen Daten wurde untersucht, ob es Häufungen bei der Manhattan-Distanz oder dem Durchsatz gibt. Trotz einzigartiger Layouts kann es vorkommen, dass Manhattan-Distanzen und simulierte Durchsätze häufiger vorkommen. Die Analyse ergab, dass bei der Manhattan-Distanz ein Duplikate-Verhältnis von 0.15 Prozent besteht, während es beim Durchsatz 4.3 Prozent beträgt. In der Kombination von Manhattan-Distanz und Durchsatz liegt das Duplikate-Verhältnis bei 0.02 Prozent. Es ist nicht zu erwarten, dass diese geringen Werte einen Einfluss auf den Trainingsprozess haben, daher wurden sie nicht aus dem Datensatz entfernt.

7.2.2 Validierung der Generalisierbarkeit

Um die Generalisierungsfähigkeit des ANN unabhängig von der gewählten Datenaufteilung zu untersuchen, wurde eine K-Fold-Kreuzvalidierung durchgeführt. Bei diesem Verfahren wird der Datensatz nicht statisch in einen Trainings- und Testdatensatz getrennt. Dies könnte die Frage aufwerfen, ob die gewählte Aufteilung optimal war. Bei der Kreuzvalidierung wird der Datensatz in K-Teilmengen aufgeteilt. Das ANN wird dann K-mal trainiert und validiert, wobei jedes Mal eine andere Teilmenge als Validierungsdatensatz verwendet wird und die restlichen K-1 Teilmengen zum Training dienen. Die endgültige Leistung wird anhand des durchschnittlichen Ergebnisses der K-Durchläufe berechnet. Ein Vorteil dieser Methode besteht darin, dass der gesamte Datensatz sowohl zum Training als auch zum Testen eingesetzt wird. Abbildung 48 zeigt einen Boxplot über den Trainingsverlust als Ergebnis dieser Untersuchung.

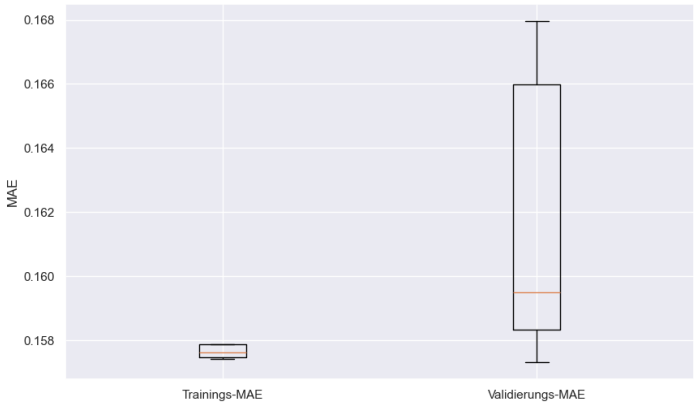


Abbildung 48: Verteilung der Trainings- und Validierungs-MAE über die Folds

Die enge Spanne beim Trainingsverlust deutet auf eine konstante Leistung über die fünf Teilmengen (Folds) hin. Unabhängig vom Fold, wurde eine vergleichbare Genauigkeit erreicht. Die breitere Spanne im Boxplot für den Validierungsverlust deutet eine größere Variabilität auf unbekannten Daten an. Dies könnte bedeuten, dass das ANN nicht generalisiert oder dass die Menge der Trainingsdaten durch die Aufteilung zu klein geworden ist. Der durchschnittliche r^2-Wert für Training liegt nach dieser Auswertung bei 0.962 und der durchschnittliche r^2-Wert für Validierung bei 0.957. Dies ist im Vergleich zu den Ergebnissen ohne Kreuzvalidierung (r^2-Wert bei 0.97, siehe Kapitel 6.8.3) eine Abweichung von weniger als zwei Prozent, sodass das Modell als validiert nach dieser Methode bezeichnet werden kann.

7.2.3 Performance-Vergleich zwischen Simulation und KI

Ein Teil der Forschungsfrage ist, inwiefern der KI-unterstützte Ansatz eine Verbesserung gegenüber herkömmlichen Optimierungsverfahren darstellt. Dies ist erfüllt, wenn das ANN bei der Bestimmung des Durchsatzes einen Geschwindigkeitsvorteil im Vergleich zur Simulation aufweist. Die Genauigkeit der Ansätze lässt sich anhand der Ergebnisse der Experimente und der Evaluierung bewerten. Grundsätzlich weist die Simulation die höchste Genauigkeit auf. Das ANN konnte einen r^2-Wert von 0.97 erzielen. Dieser Wert ist jedoch ausreichend, um bei einem vorhandenen Tempovorteil von einem Fortschritt bei der KI-unterstützten Layoutoptimierung auszugehen. Abbildung 49 zeigt die durchschnittliche Simulationsdauer des Layoutgenerators.

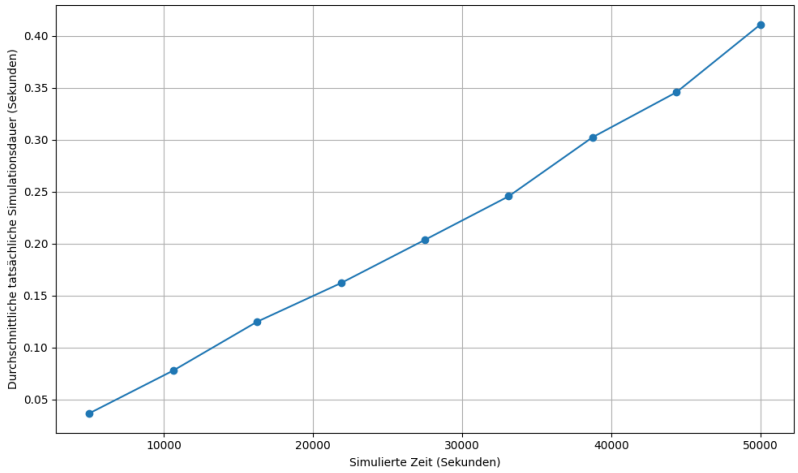


Abbildung 49: Simulationsdauer vs. Simulierte Zeit

Für dieses Experiment wurden neun verschiedene Simulationsdauern im Bereich von 5000*s* bis 50.000*s* anhand einer Fabrikkonfiguration mit acht OE getestet. Jede Simulationsdauer wurde für jeweils 3888 Layouts beibehalten. Auf der y-Achse ist die durchschnittliche Simulationsdauer in Sekunden dargestellt. Die Ergebnisse zeigen einen linearen Verlauf und einen proportional ansteigenden Zusammenhang der Simulationsdauer mit der simulierten Zeit. Dies ermöglicht eine Extrapolation und bedeutet für einen Performance-Vergleich zwischen ANN und DES, das diejenige Simulationsdauer als Vergleichswert herangezogen werden muss, ab der eine linearer Anstieg beginnt. Dies wäre die kleinst notwendige Simulationsdauer. Um diesen Zeitpunkt zu finden, wurden in einem Folgeexperiment wieder neun aufsteigende Simulationsdauern in der Spannweite 500 bis 5000 Sekunden simuliert, s. Abbildung 50.

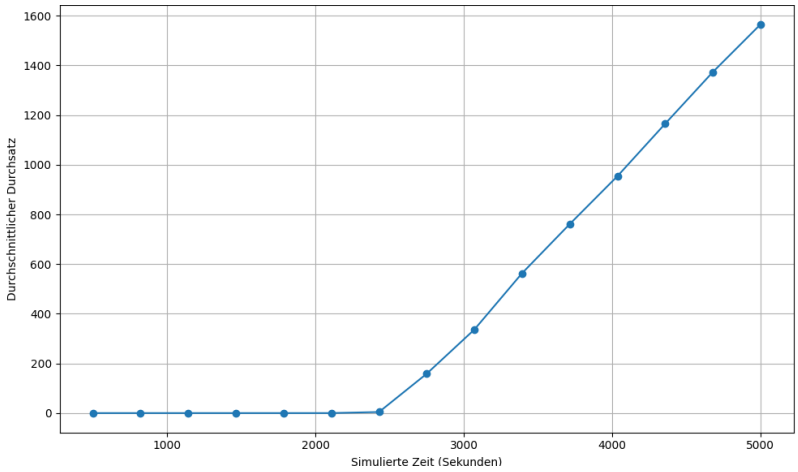


Abbildung 50: Entwicklung des Durchsatzes über die Simulationsdauer

Die Auswertung zeigt, dass zum ersten Mal nach ca. 2500s ein Durchsatz an dem Ausgangslager gemessen wird. Zwischen 2500s und 3500s beginnt der lineare Bereich. Um Schwankungen zu kompensieren, wird nach diesem Versuch die Simulationsdauer von $\sim$ 4000s als minimal notwendige Zeiteinheit bestimmt, siehe auch Tabelle 9.

Tabelle 9: Daten der Performance vom Simulator (Acht OE)

Simulationszeit (s)	Mittlere Dauer (s)	Mittlerer Durchsatz (Stk)	Layouts
500	0.003581	0.000000	3888
821	0.005792	0.000000	3888
1143	0.008061	0.000000	3888
1464	0.010362	0.000000	3888
1786	0.012694	0.000000	3888
2107	0.015094	0.000000	3888
2429	0.017546	4.302186	3888
2750	0.019991	158.264895	3888
3071	0.022076	336.024432	3888
3393	0.024377	563.067295	3888
3714	0.026649	760.861123	3888

Aus den Daten wird deutlich, dass die Fabrikkonfigurationen eine Hochlauf-Phase benötigen, da die Puffer der OE zum Start der Simulation leer sind. Nach dem Hochlauf erreichen die Fabrikkonfigurationen einen eingeschwungenen Zustand, der in einer Proportionalität zwischen dem durchschnittlichen Durchsatz und der Simulationsdauer mündet. Vergleicht man die Simulationsdauern bei unterschiedlichen Fabrikkonfigurationen, zeigt sich der in Abbildung 51 dargestellte Zusammenhang.

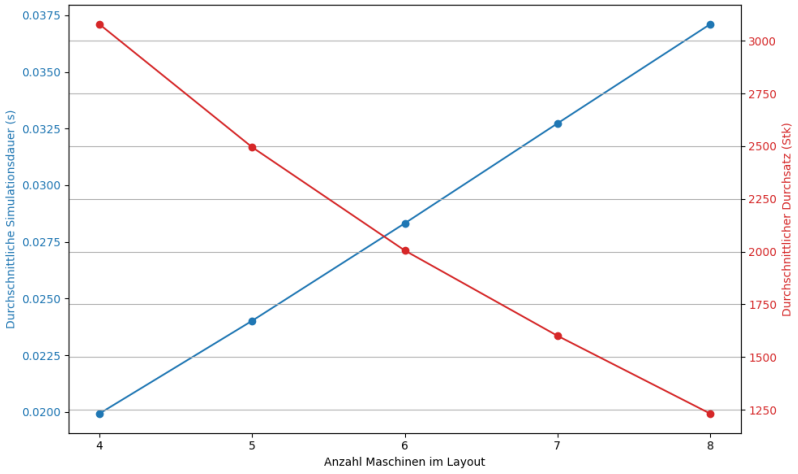


Abbildung 51: Auswertung der Simulation über die Fabrikkonfigurationen

Die Simulationsdauer zeigt ebenfalls einen proportionalen Zusammenhang mit der An-

zahl der OE im Layout auf (siehe blaue Linie). Der in rot dargestellte Durchsatz zeigt eine leicht exponentielle Abnahme auf.

Für den Performance-Vergleich wird die Fabrikkonfiguration mit der niedrigsten und höchsten Komplexität im Folgenden herangezogen, s. Abbildung 52.

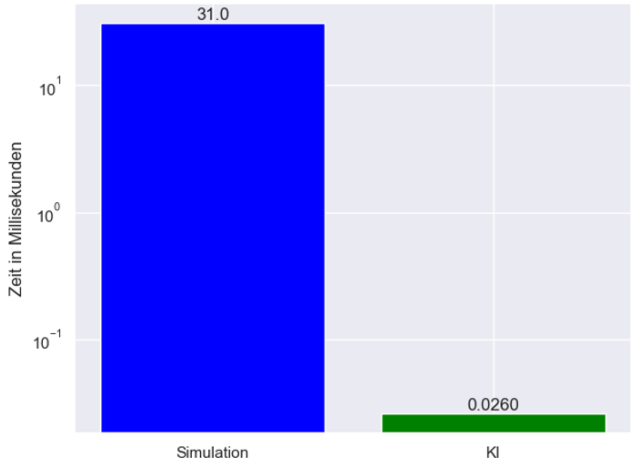


Abbildung 52: Performance-Vergleich zwischen Simulation und KI

Auf identischer Hardware erreicht das trainierte ANN eine durchschnittliche Vorhersagegeschwindigkeit von 0.026 Millisekunden pro Layout. Bei der niedrigst möglichen Simulationsdauer unter günstigen Bedingungen benötigt die Fabriksimulation nach den im Vorfeld vorgenommenen Untersuchungen etwa 31 Millisekunden pro Layout. Somit ist das ANN 1154-mal schneller als die DES.

Es wäre theoretisch möglich, die Simulation für parallele Verarbeitung und effiziente Hardware-Nutzung zu optimieren. Dies wäre jedoch mit weiteren Kosten, Entwicklungsaufwänden und unsicheren Erfolgsaussichten verbunden. Ferner erscheint es unwahrscheinlich, dass hierdurch der Performance-Unterschied ausgeglichen werden kann. Darüber hinaus wurden die Ergebnisse vom ANN auf einem Datensatz mit fünf verschiedenen Fabrikkonfigurationen erzielt, während das Simulationsergebnis den günstigsten Wert einer einzigen Fabrikkonfiguration darstellt. Daher lässt sich aus dieser Auswertung schließen, dass der KI-basierte Ansatz einen systematischen Vorteil gegenüber der DES hinsichtlich der Geschwindigkeit und Anwendbarkeit über mehrere Fabrikkonfigurationen hinweg aufweist.

7.3 Vergleich mit weiteren Prädiktionsmethoden

Die Ergebnisse des ANN wurden mit den Ergebnissen eines XGBoost-Prädiktors verglichen, der separat mit den zusätzlichen Daten und den Bilddaten trainiert wurde. Die Ergebnisse werden in Tabelle 10 zusammengefasst.

Tabelle 10: Vergleich der Modellleistungen

Modell	**r^2-Wert**
Originalmodell	0.91
XGBoost mit zusätzlichen Daten	0.6743
XGBoost mit Bilddaten	−0.0512

Tabelle 11 listet die für diesen Versuch verwendeten Parameter auf:

Tabelle 11: Konfiguration des XGBoost-Prädiktors

Parameter	**Wert**
n_estimators	100
max_depth	6
learning_rate	0.02
subsample	0.8
colsample_bytree	0.8
min_child_weight	1
early_stopping_rounds	10

- **n_estimators (100):** Bestimmt die Anzahl der zu trainierenden Boosting-Runden bzw. Bäume.
- **max_depth (6):** Gibt die maximale Tiefe eines Baumes an.
- **learning_rate (0.02):** Dies ist die Schrittgröße, die bestimmt, wie schnell das Modell lernt. Eine niedrigere Lernrate kann zu präziseren Modellen führen, benötigt aber mehr Boosting-Runden.
- **subsample (0.8):** Anteil der Stichproben, der für das Training jedes Baumes verwendet wird.
- **colsample_bytree (0.8):** Anteil der Features, der für das Training jedes Baumes verwendet wird.
- **min_child_weight (1):** Minimales Gewicht (oder Anzahl von Beispielen), das benötigt wird, um einen weiteren Knoten zu bilden.
- **early_stopping_rounds (10):** Anzahl der aufeinanderfolgenden Runden ohne Verbesserung auf den Validierungsdaten, nach denen das Training angehalten wird.

Die Datenstruktur der Eingabedaten wurde speziell für die Anwendung mit dem ANN entwickelt. XGBoost kann nicht direkt mit einem Datensatz, der sowohl aus Bildern als auch aus zusätzlichen Daten besteht, trainiert werden. Daher wurden für diesen Versuch zwei getrennte Ansätze verfolgt.

Zunächst wurde XGBoost isoliert auf die zusätzlichen Daten trainiert mit einem Ergebnis von 0.6743 bezogen auf den r^2-Wert. Dieser Wert zeigt, dass das Modell einen großen Teil der

Varianz in den Zielvariablen über zufällige Schätzungen hinaus anhand der zusätzlichen Daten erklären kann. Die zusätzlichen Daten konnten dem Modell direkt übergeben werden, da XGBoost auf tabellarische Daten spezialisiert ist. Um dem Modell die Bilddaten zu übergeben, wurde zunächst ein vortrainiertes ResNet50-Modell zur Feature-Extraktion aus den Bildern verwendet. Diese extrahierten Features wurden dann als Eingabe für das XGBoost-Modell genutzt. Dieser Ansatz führte jedoch zu einem negativen r^2-Wert von -0.0512 auf den Testdaten. Dies zeigt eine Ungeeignetheit von XGBoost für diese Art von Daten auf.

Die Ergebnisse des ANN mit integrierten CNN im Vergleich zum XGBoost-Prädiktor bestätigen die Bedeutung der konzipierten Architektur. In den durchgeführten Untersuchungen erreichte das konkatenierte Modell einen r^2-Wert von > 0.9, vgl. Kapitel 7.1.1. Insgesamt zeigt die Untersuchung, dass tiefere neuronale Netzwerke mehr Gestaltungsspielraum bei der Architektur aufweisen. Dies führt dazu, dass eine Kombination aus Bilddaten und zusätzlichen Merkmalen parallel verarbeitet werden können. Im Gegensatz dazu erfordert der Einsatz von XGBoost eine Transformation der Daten.

7.4 Vergleich mit Baselines

Um die Modellleistung zu verifizieren, wird ein Vergleich zwischen einem Mittelwert-Schätzer, einer Zufallsverteilung und einer Lift-Analyse durchgeführt.

7.4.1 Vergleich der Modell-Leistung mit Mittelwert-Schätzer

Die nachfolgende Tabelle 12 zeigt die erzielten Metriken des ANN im Vergleich zu einem Mittelwert-Schätzer:

Tabelle 12: Modellleistung im Vergleich zu einem Mittelwert-Schätzer

Metrik	**Baseline**	**Test**
Loss	0.0340	0.0003
MAE	0.1527	0.0148
r^2-Wert	-0.0002	0.91

Im Ergebnis erzielt das ANN akkuratere Vorhersagen auf Testdaten als ein Mittelwert-Schätzer. Abbildung 53 zeigt das visuelle Ergebnis zu dieser Untersuchung:

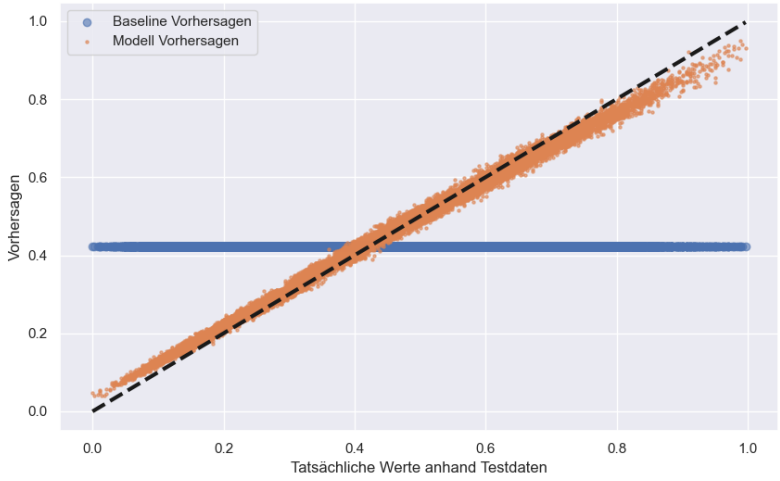


Abbildung 53: Baseline Scatter-Plot der tatsächlichen Werte gegen Vorhersagen

Die blauen Punkte sind die Vorhersagen des Mittelwert-Schätzers. Vor allem im Vergleich zu den rot gekennzeichneten ANN-Prädiktion wird deutlich, dass aus den Eingabedaten Wissen extrahiert wird, das über bloßes Schätzen hinausgeht, vergleiche außerdem Abbildung 54.

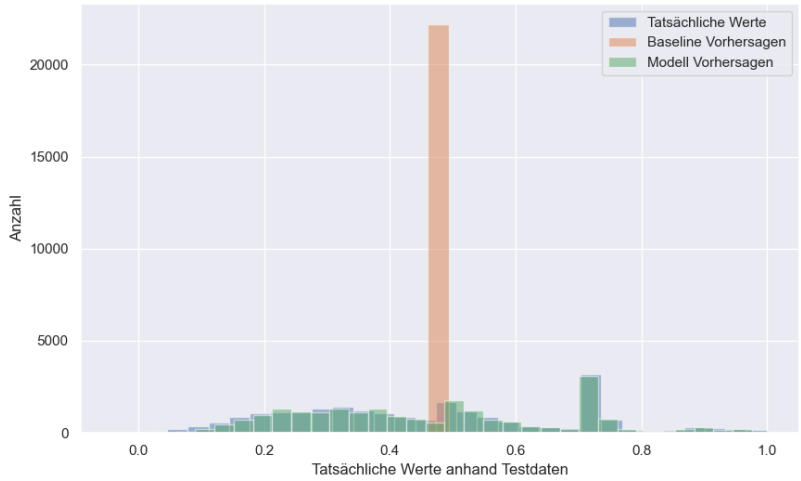


Abbildung 54: Baseline Histogramm der tatsächlichen Werte und Vorhersagen

Das Histogramm zeigt eine konstante Leistung des ANN über die gesamte Bandbreite (in grün). Insbesondere der Peak zwischen 0.6 un d 0.8 wird zuverlässig vorhergesagt. Dies

spricht gegen zufällige, willkürliche Vorhersagen. Ob die Vorhersagen auf einer vom ANN gelernten passenden Zufallsverteilung basieren, wird im folgenden Abschnitt untersucht.

7.4.2 Vergleich der Vorhersagen mit einer Zufalls-Verteilung

In diesem Abschnitt werden die Erkenntnisse zur Untersuchung der Leistung des ANN im Vergleich mit einer Zufalls-Verteilung untersucht. Ziel ist die Beantwortung der Frage, ob das ANN über das Erkennen zufälliger Muster hinausgeht oder lediglich eine passende Zufallsverteilung nachahmt.

7.4.2.1 Methodik zum Vergleich der Modellleistung mit einer zufälligen Verteilung

Um eine objektive Vergleichsbasis zu schaffen, wurden zufällige Vorhersagen generiert, die auf der Verteilung der Trainingslabels basieren. Diese dienten als Benchmark für die Bewertung der Modellleistung. Als Hauptmetriken wurden der MSE und der r^2-Wert verwendet. Zur weiteren Einschätzung der Ergebnisse wurden zusätzlich ein t-Test und ein p-Test durchgeführt. Der t-Wert ist ein statistisches Maß für den Unterschied der beobachteten Mittelwerte zweier Gruppen, in diesem Fall der Vorhersagen des Modells und der zufälligen Baseline, gemessen in Einheiten des Standardfehlers. Eine größere t-Statistik deutet auf einen größeren Unterschied zwischen den Gruppen hin. Der p-Wert gibt die Wahrscheinlichkeit an, unter der Nullhypothese (kein Unterschied zwischen den Gruppen) ein Ergebnis zu beobachten, das mindestens so extrem ist wie das tatsächlich beobachtete. Üblicherweise wird ein p-Wert unter 0.05 als Schwellenwert für statistische Signifikanz verwendet. Beide Tests wurden 500-Mal durchgeführt, um die Zuverlässigkeit der Ergebnisse zu bestätigen. Durch die Durchschnittsbildung über mehrere Durchläufe werden zudem Zufallsschwankungen minimiert und ein repräsentativeres Ergebnis erzielt.

7.4.2.2 Ergebnisse der Modellleistung im Vergleich mit einer Zufallsverteilung

Im Ergebnis lag der MSE der Zufallsverteilung bei 0.06758 und der r^2-Wert bei einem negativen Wert von -0.990695. Im direkten Vergleich dazu erzielte das ANN signifikant bessere Ergebnisse, siehe Tabelle 13:

Tabelle 13: Leistung im Vergleich mit einer Zufallsverteilung

Metrik	**Zufallsverteilung**	**ANN**
MSE	0.06758	0.0003
r^2-Wert	-0.990695	0.91

Die wiederholte Anwendung des t-Tests ergab eine durchschnittliche t-Statistik von 4.353742 und einen p-Wert von 0.000556. Dies zeigt eine statistisch signifikante Überlegen-

heit gegenüber der Zufallsverteilung auf. Die Diskrepanz zwischen den Ergebnissen des ANN und denen der Zufalls-Baseline belegt die Fähigkeit des ANN relevante Muster in den Daten zu erfassen, die über zufällige Schätzungen hinausgehen.

7.5 Skalierung der Fabrikgröße

Zur Überprüfung der Eignung des ANN unter realistischen Bedingungen wurde es auf eine Fabrikkonfiguration mit 20 OE trainiert. Dazu wurde ein Datensatz bestehend aus 100.000 Layouts mit je 20 OE erstellt und nach identischer Vorgehensweise zum Training eingesetzt. Im ersten Schritt wurde jeder OE eine einzigartige Farbe zugeordnet. Dieser dient dem ANN zur eindeutigen Einordnung der OE im Produktionsfluss. Die nachfolgende Abbildung 55 zeigt exemplarisch ein Layout aus diesem Datensatz.

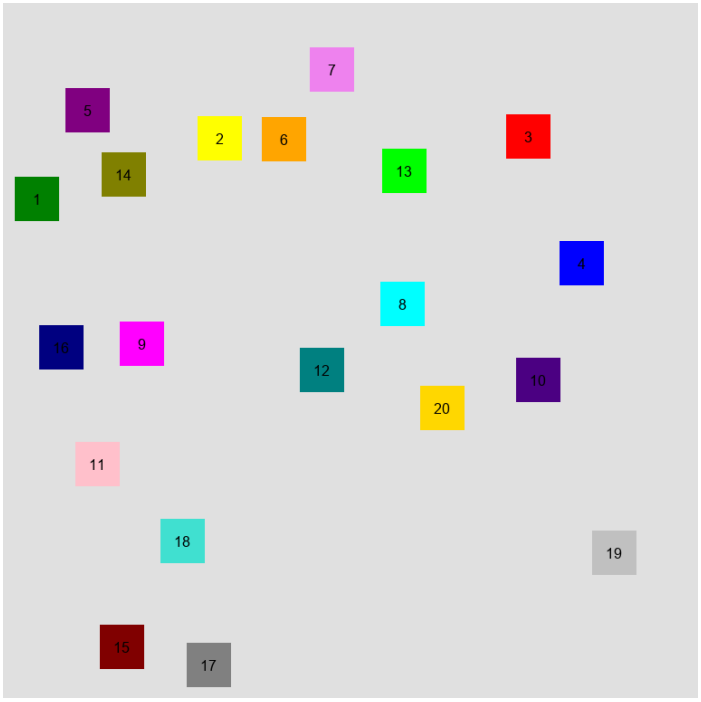


Abbildung 55: Exemplarisches Layout aus dem Datensatz mit 20 OE

Nach Erstellung der Layouts wurde untersucht, wie viel Zeit die aktuelle Fabrikkonfiguration benötigt, um in den eingeschwungenen Bereich zu kommen, s. Abbildung 56.

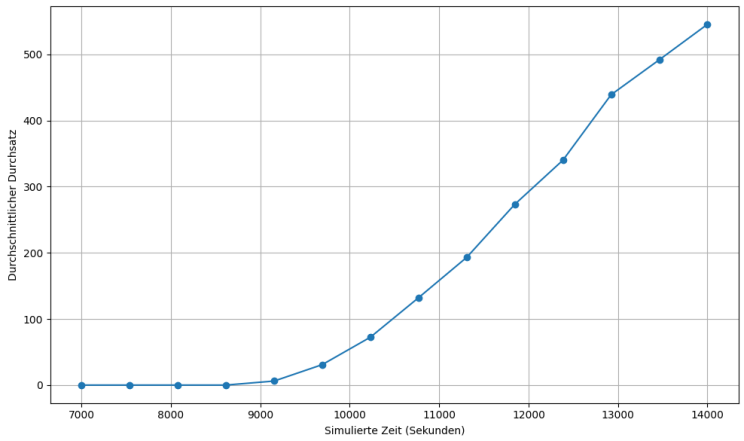


Abbildung 56: Entwicklung des Durchsatzes über die Simulationsdauer bei 20 OE

Es ist zu sehen, dass ab $\sim 9000s$ zum ersten Mal Produkte am Ausgangslager der modellierten Fabrik eintreffen. Anschließend ist der Beginn des eingeschwungenen Bereichs anhand des proportionalen Zusammenhangs zwischen der simulierten Zeit und dem Durchsatz zu beobachten. Zur Erfassung des Durchsatzes wird die Simulation basierend auf dieser Analyse nach $10.000s$ zurückgesetzt. Die insgesamt notwendige Simulationsdauer beträgt für eine Fabrikkonfiguration mit 20 OE demzufolge $12.000s$. Mit diesem ermittelten Kennwert wurde die Simulation aller 100.000 Layouts bezüglich des Durchsatzes durchgeführt. Die nachfolgende Abbildung 57 zeigt den Trainingsverlauf über 29 Epochen.

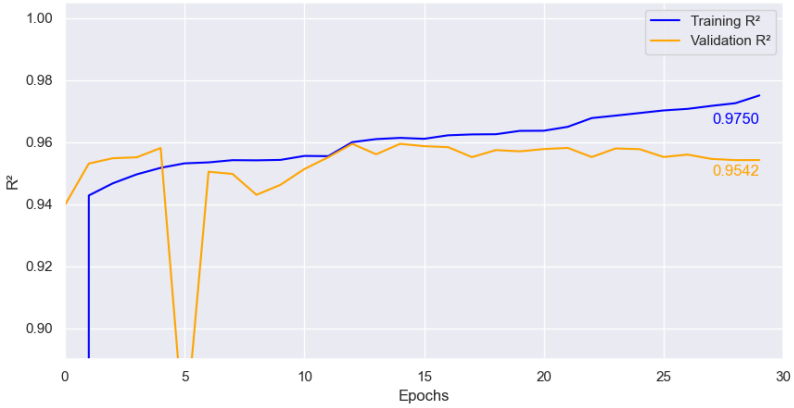


Abbildung 57: Trainingsverlauf des ANN mit der skalierten Fabrikkonfiguration

Auf den Trainingsdaten erreicht das ANN einen r^2-Wert von ~ 0.97, auf den Validierungsdaten einen r^2-Wert von ~ 0.95. Eine Untersuchung auf Testdaten hat einen r^2-Wert von ~ 0.92 ergeben. Diese Werte zeigen, dass das ANN den simulierten Durchsatz bei

einer komplexen Fabrikkonfiguration abbilden kann. Die Bandbreite der vorherzusagenden denormalisierten Durchsätze lag zwischen einem globalen Minimum von 1 bis zum globalen Maximum von 11454, s. Abbildung 58 zur Verteilung der ANN-Schätzungen über die Testdaten.

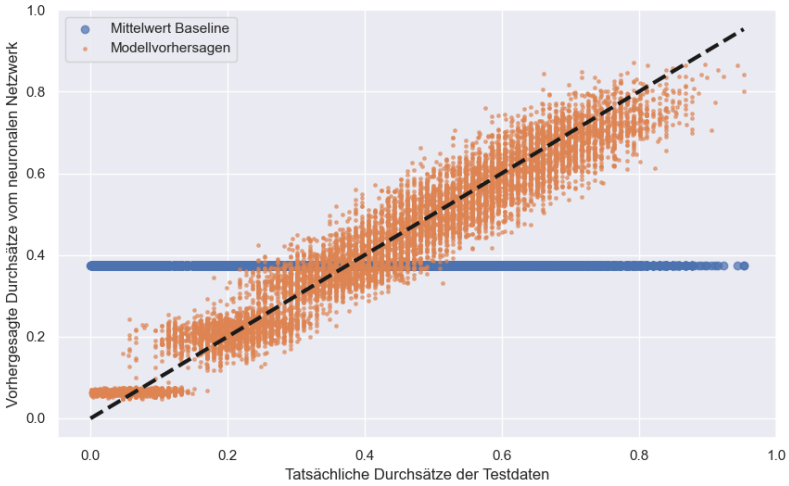


Abbildung 58: Bewertungen des ANN gegen die tatsächlichen simulierten Werte

Im Ergebnis ist die Leistung über die gesamte Bandbreite hinweg konstant.

8 Übertragbarkeit auf weitere Anwendungsszenarien

In diesem Kapitel wird dargelegt, wie die eingeführte Optimierungsmethodik auf einen anderen Kontext übertragen werden kann und welche Limitierungen bestehen. Konkret erfolgt dies an der Layoutgestaltung in einem Krankenhaus. Für eine Übertragung der KI-unterstützten Layoutoptimierung nach dem Konzept dieser Arbeit werden nachfolgend acht Schritte definiert:

1. Einarbeitung in Thematik
2. Auswahl des Systems
3. Festlegung eines Optimierungskriteriums
4. Datensammlung oder -erzeugung und Bereinigung
5. Simulation und Bewertung der Layouts nach dem Optimierungskriterium
6. Modellanpassung und -training
7. Einbettung des Modells in einen Optimierungsalgorithmus
8. Implementierung des Layouts

Die einzelnen Schritte werden in den folgenden Unterkapiteln erläutert.

8.1 Einarbeitung in die Thematik

Die Übertragung auf den Krankenhauskontext setzt die Kenntnis einer Vielzahl von Disziplinen voraus. Diese Arbeit adressiert die Grundlagen der Layoutgestaltung, metaheuristischer Lösungsmethoden, ereignisorientierter Simulation und KI im Zusammenhang mit der Optimierung von Fabriklayouts. Darüber hinaus ist eine Einarbeitung in die spezifischen Anforderungen, die sich aus dem Krankenhauskontext ergeben notwendig. Die Literaturempfehlungen in den nachfolgenden Abschnitten dienen als Einstieg und sollen einen allgemeinen Zugang zur Thematik ermöglichen.

8.1.1 Grundlagen der Layoutoptimierung

Für einen Einstieg in die Themen der Layoutoptimierung empfehlen sich die Werke von Tompkins et al. [9] und Grundig [16], die eine Einführung in die Planung und Optimierung von Layouts bereitstellen. Diese Grundlagen bilden die Basis, die durch Spezifika von Krankenhauslayouts erweitert werden.

8.1.2 Spezifika von Krankenhauslayouts

Die Komplexität von Krankenhauslayouts erfordert Wissen, das über die allgemeine Layoutplanung hinausgeht. Einschlägige Werke behandeln Themen zur Optimierung von Patientenflüssen, Planung von Notaufnahmen, die Gestaltung von Patientenzimmern und Operationssälen. Das Lehrbuch von Miller et al. zur Planung und Gestaltung von Krankenhäusern und Gesundheitseinrichtungen könnte hier einen Einstieg für diesen Forschungsbereich darstellen [227]. Bei der Übertragung auf einen anderes Anwendungsszenario wie bspw. Verladehäfen muss an dieser Stelle eine Einarbeitung in die dortigen Spezifika erfolgen.

8.1.3 Einarbeitung in die Grundlagen der künstlichen Intelligenz

Für die Einarbeitung in Grundlagen der künstlichen Intelligenz können Standardwerke oder auch Online-Kurse genutzt werden. Als Literaturquelle dienten insbesondere die Ausführungen von Russel und Norvig [47] sowie die auf YouTube veröffentlichten Lehrvideos von Dr. Jeff Heaton, der als Lehrbeauftragter (Adjunct Instructor) am Sever Institute der Washington University in St. Louis tätig ist [228]. Das erstgenannte Werk deckt eine breite Palette von Themen ab, einschließlich Suchalgorithmen, Wissensrepräsentation, maschinelles Lernen, natürliche Sprachverarbeitung und Robotik. Die zum Zeitpunkt dieser Arbeit aktuellste Ausgabe enthält auch Abschnitte zu aktuellen Entwicklungen um Deep Learning und ethischen Überlegungen.

8.1.4 Ereignisorientierte Simulation

Zur Einarbeitung in die ereignisorientierte Simulation wurde das Grundlagenwerk von Liebl verwendet [46]. Dieses behandelt ausführlich das Thema der DES und enthält darüber hinaus Hinweise zur Implementierung.

8.1.5 Fazit zur Einarbeitung in die Grundlagen

Das vorliegende Kapitel skizziert einen multidisziplinären Ansatz zur Einarbeitung in die Thematik der Layoutoptimierung im Krankenhauskontext. Die Herausforderung besteht darin, ein tiefes Verständnis für die theoretischen Grundlagen zu entwickeln, während gleichzeitig die spezifischen Anforderungen und Besonderheiten des Anwendungsbereichs berücksichtigt werden. Da für die Übertragung unter Umständen Änderungen in der Implementierung vorgenommen werden müssen, sind zudem Kenntnisse in Programmiersprachen, insbesondere Python, notwendig.

8.2 Auswahl des Systems

Die Übertragung der KI-unterstützten Optimierung auf Krankenhauslayouts erfordert eine Festlegung des anvisierten Systems und dessen Randbedingungen. Die vorliegende Untersuchung konzentriert sich auf die Lösung des Open Field Layout Problem (OFLP), bei dem die optimale Anordnung von OE innerhalb eines begrenzten Layouts gesucht wird. Wie in den theoretischen Grundlagen in Kapitel 3 erläutert wird, existieren weitere Layoutklassifizierungen, von denen in einem Krankenhausumfeld mehrere parallel auftreten. Die eingeführte Methodik ist nicht auf das OFLP beschränkt, jedoch lassen sich nicht mehrere Layoutklassen parallel behandeln. Daher ist für die Übertragung der Methodik eine Einschränkung auf eine Layoutklasse notwendig. Die Beschränkung auf das OFLP ist von Vorteil, da die eingeführte Implementierung dieses Problem behandelt und es somit die geringsten Aufwände bei der Anpassung nach sich führt. Im Gegensatz dazu beinhalten andere Layoutklassen, wie das Single-Row Layout Problem (SRLP), spezifische Anforderungen, die Anpassungen in der Implementierung erforderlich machen. Sollte im Einzelfall eine Abweichung vom OFLP notwendig sein, müssten sowohl der Layoutgenerator als auch die Simulation entsprechend angepasst werden.

Im Beispiel eines SRLP ist der Layoutgenerator so anzupassen, dass er die Einheiten ausschließlich entlang von Reihen platziert. Die Simulation muss so angepasst werden, dass nur Bewegungen entlang dieser definierten Reihen gesteuert werden. Zusammenfassend ist eine Übertragung des Konzeptes auf weitere Layoutklassen möglich, sofern das Konzept neu implementiert oder die bestehende Implementierung angepasst wird.

8.3 Festlegung eines Optimierungskriteriums

Im Rahmen der vorliegenden Untersuchung wurden Fabriklayouts hinsichtlich des Materialdurchsatzes optimiert. Dies gelingt durch eine DES des Fertigungsablaufs zur Ermittlung des erzielten Durchsatzes nach einer definierten Zeitdauer. Neben diesem Kriterium ist es auch denkbar, eine Optimierung nach Aspekten der Sicherheit, Nachhaltigkeit oder Mischformen durchzuführen. Grundsätzlich beeinflusst das Optimierungskriteriums die Ergebnisse, da das sicherste Layout und das mit dem höchsten Durchsatz voneinander abweichen werden. Für Mischformen, also die Berücksichtigung mehrerer Optimierungskriterien, muss ein Faktor zur Gewichtung definiert werden, der in die Layoutbewertung einfließt. Im Umfeld von Krankenhäusern sind u. a. folgende Optimierungskriterien denkbar:

- **Patientenfluss**: Das Hauptziel ist es, Wartezeiten zu minimieren und die zurückzulegenden Distanzen innerhalb der Einrichtung zu reduzieren.

- **Barrierefreiheit**: Gewährleistung des uneingeschränkten Zugangs für alle Patienten und Besucher, insbesondere für Menschen mit Behinderungen und Verletzungen.
- **Interaktion zwischen Abteilungen**: Optimierung der Zusammenarbeit zwischen verschiedenen Abteilungen wie Notaufnahme, Radiologie und Chirurgie.
- **Mitarbeiterwege**: Optimierte Gestaltung der Mitarbeiterbereiche und -wege, um die Arbeitsabläufe zu beschleunigen.
- **Infektionskontrolle**: Schaffung von abgegrenzten Bereichen, um die Ausbreitung von Infektionen zu verhindern.

Neben den aufgeführten Kriterien gibt es eine Vielzahl weiterer, weniger spezifischer Aspekte wie die Energieeffizienz oder Nachhaltigkeit. Es können nicht alle Optimierungskriterien gleichsam in eine Optimierung einfließen. Eine Priorisierung der Infektionskontrolle erhöht bspw. die Wartezeiten oder Wegstrecken. Ferner sind die Kriterien in verschiedenen Layoutplanungsebenen einzuordnen. Wohingegen die Zusammenarbeit verschiedener Abteilungen auf der Ebene der Groblayoutplanung festgelegt wird, ist die stationsbezogene Optimierung von Patientenflüssen, z. B. in der Notaufnahme, auf der Ebene der Feinlayoutplanung eingeordnet, vgl. Kapitel 3.1. Das eingeführte Konzept ist auf die letztgenannte Ebene beschränkt. Grundsätzlich ist eine Übertragung auf die anderen Ebenen mit Implementierungsaufwand möglich. Die Optimierung des Patientenflusses ist ähnlich zur Maximierung des Materialdurchsatzes bei Fabriklayouts. Der Patientenfluss kann sich durch eine Verringerung von Wegstrecken erhöhen, analog zur Erhöhung des Durchsatzes bei Fabriklayouts. Abhängig für einen optimierten Patientenfluss ist die Platzierung der verschiedenen Stationen, die ein Patient von der Aufnahme bis zur Entlassung durchlaufen muss. Tabelle 14 führt diese Stationen auf und zeigt eine exemplarische Gegenüberstellung zu Fabrik-OE:

Tabelle 14: Gegenüberstellung von OE in Fabriken und Krankenhäusern

Fabriklayout	**Krankenhauslayout**
Eingangslager	Rezeption / Empfang
Materialpuffer	Wartezimmer
Produktionslinie	Untersuchungs- und Behandlungsräume
Kran	Aufzug
Spezialmaschinen	Spezialabteilungen (z. B. Radiologie)
Qualitätskontrolle	Diagnostiklabore
Montagebereich	Operationsräume
Verpackungsstation	Apotheke
Ausgangslager	Entlassungsbereich
Zusatzlager	Notwageneinlass / Notaufnahme

Die Gegenüberstellung zeigt auf, dass Ähnlichkeiten in verschiedenen Layout-Kontexten

bestehen. Dies führt zu Vorteilen in einer auf Klassen basierten Implementierung. Die Eingangslager in Fabriken entsprechen den Rezeptionen im Krankenhaus, da beide als primäre Anlaufpunkte zur Aufnahme von Patienten bzw. Materialien dienen. Materialpuffer entsprechen Wartezimmern, Produktionslinien und Untersuchungs- sowie Behandlungsräume sind jeweils Orte der Hauptaktivität (fertigen - behandeln). Spezialmaschinen und Spezialabteilungen wie die Radiologie nutzen spezifische Ausrüstungen, die nur in begrenzter Menge vorhanden sind. Weitere Parallelen finden sich in der Qualitätskontrolle und den Laboren, die beide der Untersuchung dienen. Montagebereiche und Operationsräume sind des Weiteren gegenübergestellt - bei Ersterem werden Komponenten zu einem Endprodukt zusammengefügt, bei Zweiterem operative Eingriffe durchgeführt. Die Funktion der Verpackungsstationen wird durch Apotheken im Krankenhaus abgebildet, bei denen Medikamente für die Patienten vorbereitet werden. Schließlich entspricht das Ausgangslager dem Entlassungsbereich eines Krankenhauses. Diese Analogien unterstreichen die Möglichkeit der Übertragung des Konzepts der Fabriklayoutoptimierung auf das Krankenhaussystem.

8.4 Datenerzeugung und Bereinigung

Die Datenlage zu Krankenhauslayouts unterliegt der identischen Herausforderung wie Fabriklayouts, da beide nicht in ausreichender Menge und Qualität vorhanden sind. Zur Übertragung des Konzeptes ist es im ersten Schritt daher notwendig ein Skript zur Erzeugung beliebiger Mengen von krankenhausähnlichen Layouts zu implementieren. Dazu ist es wie in den vorigen Abschnitten beschrieben notwendig, ein System mit Kriterien festzulegen und die bestehende Implementierung daraufhin anzupassen. Die weiteren Funktionen zur Bereinigung und Aufbereitung der Daten, wie die Duplikateprüfung, können unverändert übernommen werden.

8.5 Simulation und Bewertung der Krankenhauslayouts

Nach Erstellung des Datensatzes erfolgt die ereignisorientierte Simulation der Abläufe in den Layouts mit dem zuvor festgelegtem Optimierungskriterium als Zielmetrik (z. B. Durchsatz an Patienten). Darüber hinaus können weitere Metriken gesammelt werden. Nach Abschluss der Simulation steht ein Datensatz bestehend aus Layoutvarianten und zugehörigen Metriken zur Verfügung.

8.6 Modellanpassung und -training

Das entwickelte ANN zur Bestimmung des simulierten Durchsatzes bei Fabriklayouts ist nicht beschränkt auf eine Layoutebene oder ein bestimmtes Problem. Es ist davon auszuge-

hen, dass es auf die Bestimmung weiterer Zielmetriken wie die von Krankenhauslayouts trainiert werden kann.

8.7 Einbettung des Modells in einen Optimierungsalgorithmus

Das trainierte ANN kann gespeichert und als Modul in einen metaheuristischen Optimierungsalgorithmus integriert werden, um die Bewertung der Layouts anstelle einer Simulation zu übernehmen. Dies nutzt verfügbare Rechenressourcen effizienter aus.

8.8 Einführung des Layouts im Krankenhaus

Insgesamt versteht sich die entwickelte Systematik als ein *Decision Support System (DSS)* zur Bereitstellung einer qualifizierten Layout-Lösung. Es ist aber nicht in der Lage das endgültig beste Layout zu ermitteln, da dies von weiteren nicht simulierbaren Faktoren abhängig ist. Die gefundenen optimierten Layouts werden von Experten bewertet und in kleinen Schritten im Krankenhaus mit kontinuierlichen Anpassungen, basierend auf den realen Abläufen und Rückmeldungen von Personal und Patienten, eingeführt.

9 Schluss

In dieser Arbeit wurde die Anwendung von künstlicher Intelligenz zur Lösung des Fabriklayoutproblems untersucht. Zu diesem existieren seit den 60er Jahren umfangreiche Forschungen, bislang allerdings nur wenige die KI einsetzen. Es wurden drei Forschungsfragen im Rahmen dieser Arbeit behandelt, mit dem Ziel KI für das FLP vorteilhaft zum Einsatz zu bringen. Im Ergebnis konnte ein Mehrwert zur Optimierung von Fabriklayouts dargelegt werden.

9.1 Zusammenfassung und Fazit

Zu Beginn dieser Arbeit wurde in Kapitel 1 das Thema vorgestellt und ein Verständnis für den Nutzen der Fabriklayoutoptimierung geschaffen. In einer Welt, die von Ressourcenknappheit bedroht ist und sich verstärkt in Richtung Nachhaltigkeit orientiert, kann die optimale Anordnung von Organisationseinheiten einen Beitrag zu dieser Entwicklung leisten.

In Kapitel 2 wurden die wissenschaftlichen Fragestellungen dieser Arbeit formuliert, die eine Leitfrage und drei abgeleitete Teilfragen umfassen. Die Leitfrage zielt auf die Untersuchung und Identifikation von Ansatzpunkten künstlicher Intelligenz für die Layoutoptimierung ab. Die erste Teil-Forschungsfrage dient der Identifizierung von Risiken und Potenzialen. Untersuchungen zur Gestaltung und Einbindung von KI-Architekturen werden im Zuge der zweiten Teil-Forschungsfrage behandelt. Die dritte Teil-Forschungsfrage ist auf Untersuchungen zur Validität, den Vorteilen und den Limitierungen fokussiert.

Die weiter gefassten Grundlagen in Kapitel 3 widmeten sich KI-Techniken und diskutierten den potenziellen Einsatz zur Layoutoptimierung. Der Forschungsstand in Kapitel 4 führt verwandte Untersuchungen auf. Insgesamt diente die Bearbeitung von Kapitel 3 und Kapitel 4 der Behandlung der ersten Teil-Forschungsfrage. Die Ergebnisse zeigten, dass die meisten KI-Techniken unterschiedliche Ansatzpunkte bieten, um zur Layoutoptimierung beizutragen. Ein ganzheitliches oder direkt einsetzbares System existiert jedoch nicht. Schwierigkeiten ergeben sich aufgrund einer fehlenden Verfügbarkeit von Daten, Modellen und Konzepten für den sinnvollen Einsatz zur Layoutoptimierung. Vor allem integrative Ansätze wie der kombinierte Einsatz von metaheuristischen Suchalgorithmen und ANN als Bewertungsfunktion führen zu verbesserten Layouts. Zudem offenbart sich ein wachsendes Interesse am Einsatz von Digital Twins, Virtual Reality (VR) und Augmented Reality (AR) zur Unterstützung von Layoutentscheidungen.

Die gewonnenen Erkenntnisse sind in das Konzept von Kapitel 5 zur Behandlung der

zweiten Teil-Forschungsfrage eingeflossen. Dieses integriert die ereignisorientierte Simulation mit einer aus zwei Netzwerktypen bestehenden konkatenierten ANN-Architektur in einem überwachten Lernansatz. Durch einen Layoutgenerator wird eine beliebige Anzahl Layouts generiert, die durch eine Simulation beschriftet werden. Das aus MLP und CNN bestehende künstliche neuronale Netzwerk verarbeitet visuelle Layoutrepräsentationen sowie tabellarische Daten zur Bestimmung des voraussichtlichen Durchsatzes. Die Implementierung des Konzepts behandelt die zweite und dritte Teil-Forschungsfrage. Unter Verwendung von ML-Bibliotheken wurde eine auf die Architektur zugeschnittene Datenklasse erstellt, die Layouts mit tabellarischen Daten zu einem Set vereint. Die Implementierung ermöglichte es zudem, verschiedene Layoutvarianten, ANN-Architekturen und die Auswirkungen zu untersuchen.

Das Modelltraining und die Experimente sind in Kapitel 6 dokumentiert. Mit fünf verschiedenen Layoutkonfigurationen wurde das ANN wiederholt zwischen drei bis fünf Stunden trainiert. Faktoren wie die Größe der OE oder visuell codierte Informationen hatten keinen Einfluss auf die Genauigkeit des ANN. Wesentliche Einflussfaktoren waren die Größe der Datensätze und die durchgeführte Trainingsdauer.

In Kapitel 7 wurden der Ansatz und die Ergebnisse evaluiert und zur Beantwortung der dritten Teil-Forschungsfrage validiert. Die Evaluierung zeigte, dass das ANN mit hoher Genauigkeit die simulierten Durchsätze bestimmen konnte. Insbesondere der Vergleich zu weiteren KI-Techniken wie XGBoost zeigte systemische Vorteile der konzipierten Architektur auf. Die im Rahmen der Validierung durchgeführte Feature-Analyse offenbarte, dass das ANN vor allem von den zusätzlichen tabellarischen Daten profitierte, die allein etwa 60 Prozent der Genauigkeit ausmachten. Von den zusätzlichen Daten wertete das ANN insbesondere die Koordinatenangaben aus. Ein durchgeführter Performance-Vergleich zwischen Simulation und ANN ergab einen etwa 1154-fachen Geschwindigkeitsvorteil des neuronalen Netzwerks. Während die Simulation mit steigender Komplexität der Layouts linear mehr Zeit in Anspruch nahm, konnte das ANN seinen Geschwindigkeitsvorteil beibehalten. Eine Grenze des Ansatzes besteht in der Abhängigkeit von einem initialen Datensatz, der eine Simulation bedingt.

In Kapitel 8 wird dargestellt, dass sich das Konzept auch auf andere, verwandte Anwendungsszenarien übertragen lässt. Konkret werden am Beispiel von Krankenhauslayouts die notwendigen Schritte dargelegt.

9.2 Ausblick

Diese Arbeit wurde im Jahr 2020 begonnen. Als im März 2023 große Sprachmodelle (Large Language Models (LLM)), wie GPT-4 in der Breite das klassische Programmieren und die Texterstellung revolutionierten, waren weite Teile dieser Arbeit bereits abgeschlossen. Den-

noch zeigt sich bei der Codegenerierung ein großes Potenzial dieser Technik insbesondere bei der Fehlersuche. Es zeigt sich, dass durch das Zusammenspiel von LLM und Mensch die Entwicklungszeiten verkürzt werden können, wodurch mehr Zeit für weiterführende Denkansätze frei wird.

Die Untersuchung zeigt ferner, dass ein ANN mit selbst generierten Daten auf die Lösung spezifischer Problemstellungen optimiert werden kann. Dies eröffnet einem LLM mit Programmierkapazitäten theoretisch die Möglichkeit, sich mit selbstständig erzeugten Daten auf die Lösung spezifischer Problemstellungen trainieren könnte. Neben der verstärkten LLM-Integration könnten sich zukünftige Forschungen auf GAN-basierte Ansätze konzentrieren, die andersherum als in diesem Konzept nach Vorgabe eines einzuhaltenden Durchsatzes verschiedene Layoutvarianten generieren, die diesem entsprechen. Dies würde die Integration eines metaheuristischen Algorithmus überflüssig machen und könnte ein erster ganzheitlich auf KI-basierender Ansatz zur Lösung des FLP darstellen.

Zusammenfassend lässt sich sagen, dass mit der vorliegenden Arbeit viele Disziplinen zur Integration von ANN in die Layoutoptimierung exploriert wurden. Die Erkenntnisse bieten Ansatzpunkte für weiterführende Forschungen und Entwicklungen, insbesondere in Verbindung mit rapide entwickelnden LLM. Es bleibt abzuwarten, in welchem Umfang diese Technologien in den kommenden Jahren die industrielle Praxis beeinflussen werden.

Anhang A Implementierung

Zur Implementierung des Fabriklayoutgenerators wurden die Hochsprachen C++ und Python verwendet. Folgende Programmbibliotheken kamen zum Einsatz:

- jsonLohmann [229]
- spdlog [230]
- imgui [231]
- glfw [232]
- glad [233]
- stb [234]

Zur Implementierung der Architekturen folgende Programmierbibliotheken und Frameworks eingesetzt:

- Tensorflow [235]
- PyTorch [236]
- Seaborn [237]
- Pillow [238]
- SciPy [239]
  - Matplotlib [240]
  - Pandas [241]
  - Scikit-Learn [242]

Sämtliche der aufgeführten Bibliotheken sind zum Zeitpunkt der Abgabe Open-Source lizenziert.

Abbildungsverzeichnis

1 – Einleitung und Motivation

1 Konzept zur KI-unterstützten Layoutevaluation 2

2 – Forschungsfrage und Ansatz

2 Aktivitäten und Artefakte zu den Forschungsfragen 8
3 Forschungsmethodik 9

3 – Grundlagen

4 Ebenen der Layoutplanung [17] 14
5 Rechteckiger und ungleichförmiger Grundriss [26] 16
6 Layoutdesign unter Berücksichtigung der Transportflüsse [24] 16
7 *Backtracking* und *Bypassing*, Darstellung nach [26] 17
8 Diskrete Repräsentation von Fabriklayouts, Darstellung nach [26] 18
9 Kontinuierliche Repräsentation von Fabriklayouts nach [26] 19
10 Anzahl der FLP-Publikationen nach mathematischem Modell [24] 21
11 Anzahl von Publikationen mit metaheuristischen Lösungsverfahren [24] . . 24
12 Darstellung eines MLP, die Pfeile zeigen den Datenfluss 65

4 – Stand der Wissenschaft und Technik

13 Zeitreihenanalyse zu KI-Veröffentlichungen mit Bezug zum FLP [191] . . . 78
14 Anzahl Veröffentlichungen zu metaheuristischen Algorithmen [24] 79

5 – Konzeption und Modellierung

15 Konzeption für die KI-unterstützte Layoutoptimierung 88
16 Suchraumabdeckung von Monte-Carlo-Algorithmus und GA 90
17 Simulationsgraph der ereignisorientierten Simulation 91
18 Simulationslogik 92
19 Visualisierung der Modellentwicklung 97
20 Performance der Modelle nach r^2-Wert 99

21 Architektur des ANN 101
22 Grundkonzept zur KI-unterstützten Bestimmung des Durchsatzes 102
23 Konzeption zur KI-Integration in einen GA 103

6 – Prototypische Realisierung

24 Trend zum Einsatz von ML-Frameworks [221] 108
25 Beispiellayout aus einer Codeausführung des Layoutgenerators 118
26 Transportmatrix 119
27 Scatterplot der simulierten Durchsätze 131
28 Boxplot zum simulierten Durchsatz 132
29 Verlauf der Lernraten im Versuchsaufbau 144
30 Auswirkungen unterschiedlicher Batch-Größen 146
31 Verlustverlauf unterschiedlicher Batch-Größen 147
32 Verluste unterschiedlicher Pool-Größen 149
33 Verlustverlauf unterschiedlicher Pool-Größen 149
34 MAE-Verlauf über die Trainingsepochen 150
35 r^2-Verlauf über die Epochen 150

7 – Evaluierung und Validierung

36 Wahre vs vorhergesagte Werte anhand Testdaten (Original Scale) 152
37 Histogramm der Residuen 153
38 ANN-Performance bei Vorhersage der MHC 155
39 Modellleistung nach Layoutkonfigurationen 155
40 Modellleistung auf variierende Simulationsdauern 156
41 Einfluss der Permutation auf die Modellvorhersagen 157
42 Originalbild im Vergleich zum permutierten Bild 158
43 Gemittelte relative Wichtigkeiten der zusätzlichen Daten 160
44 Zusammengefasste gemittelte relative Wichtigkeiten der Features 161
45 Niedrige und hohe Manhattan-Distanz im Vergleich 163
46 Korrelationsmatrix zu den Simulationsdaten 165
47 Scatterplot zur Manhattan- und gefahrenen Distanz 166
48 Verteilung der Trainings- und Validierungs-MAE über die Folds 168
49 Simulationsdauer vs. Simulierte Zeit 169
50 Entwicklung des Durchsatzes über die Simulationsdauer 169
51 Auswertung der Simulation über die Fabrikkonfigurationen 170
52 Performance-Vergleich zwischen Simulation und KI 171
53 Baseline Scatter-Plot der tatsächlichen Werte gegen Vorhersagen 174

54 Baseline Histogramm der tatsächlichen Werte und Vorhersagen 174
55 Exemplarisches Layout aus dem Datensatz mit 20 OE 176
56 Entwicklung des Durchsatzes über die Simulationsdauer bei 20 OE 177
57 Trainingsverlauf des ANN mit der skalierten Fabrikkonfiguration 177
58 Bewertungen des ANN gegen die tatsächlichen simulierten Werte 178

Tabellenverzeichnis

3 – Grundlagen

1 Modellierungsklassen zum FLP nach [24] 20
2 Begriffsterminologie zu genetischen Algorithmen 32
3 Konfusionsmatrix für Klassifikationsmodelle 51

5 – Konzeption und Modellierung

4 Kategorien und Aufschlüsselung zum DoE 100

6 – Prototypische Realisierung

5 Vergleich der Machine-Learning-Bibliotheken 110
6 Skalierung der erstellten Layouts nach Komplexität 141

7 – Evaluierung und Validierung

7 Originale und permutierte zusätzliche Daten 159
8 Verteilung der Daten in Trainings-, Validierungs- und Testset 164
9 Daten der Performance vom Simulator (Acht OE) 170
10 Vergleich der Modellleistungen 172
11 Konfiguration des XGBoost-Prädiktors 172
12 Modellleistung im Vergleich zu einem Mittelwert-Schätzer 173
13 Leistung im Vergleich mit einer Zufallsverteilung 175

8 – Übertragbarkeit auf weitere Anwendungsszenarien

14 Gegenüberstellung von OE in Fabriken und Krankenhäusern 182

Literaturverzeichnis

[9] James A Tompkins, John A White, Yavuz A Bozer und Jose Mario Azaña Tanchoco. *Facilities planning*. John Wiley & Sons, 2010, S. 854 ff.

[10] Jaydeep Balakrishnan und Chun Hung Cheng. „Multi-period planning and uncertainty issues in cellular manufacturing: A review and future directions“. In: *European journal of operational research* 177.1 (2007), S. 281–309.

[11] A. Breitkopf. *Anzahl der Baufertigstellungen von Fabrik- und Werkstattgebäuden in Deutschland in den Jahren 2001 bis 2019*. Juni 2020. URL: https://de.statista.com/statistik/daten/studie/257718/umfrage/baufertigstellungen-von-fabrik-und-werkstattgebaeuden-in-deutschland/ (besucht am 06. 07. 2020).

[12] iph hannover. *Layoutplanung: So gelingt ein optimales Fabriklayout*. 14. Aug. 2020. URL: https://www.iph-hannover.de/de/information/fabrikplanung/layoutplanung/ (besucht am 14. 08. 2020).

[13] fabrik id. *fabrik-ID Ideen für Fabriken*. 14. Aug. 2020. URL: https://fabrik-id.de/ (besucht am 14. 08. 2020).

[14] Daniel Scholz. *Innerbetriebliche Standortplanung: Das Konzept der Slicing Trees bei der Optimierung von Layoutstrukturen*. Springer-Verlag, 2010.

[15] Siegfried Wirth, Hubert Mann und Robert Otto. „Layoutplanung betrieblicher Funktionseinheiten“. In: *TU Chemnitz, Institut für Betriebswissenschaften und Fabriksysteme, Wissenschaftliche Schriftenreihe* 25 (2000).

[16] Claus-Gerold Grundig. *Fabrikplanung: Planungssystematik-Methoden-Anwendungen*. Carl Hanser Verlag GmbH Co KG, 2014.

[17] Lennart Sören Bochmann. „Entwicklung und Bewertung eines flexiblen und dezentral gesteuerten Fertigungssystems für variantenreiche Produkte“. Diss. ETH Zurich, 2018.

[18] Verein Deutscher Ingenieure. *VDI 2385 - Leitfaden für die materialflüssgerechte Planung von Industrieanlagen*. 1989.

[19] John A White, Richard L Francis, RL Francis und Leon F McGinnis. *Facility layout and location: An analytical approach*. Prentice-Hall, 1992.

[20] Tjalling C Koopmans und Martin Beckmann. „Assignment problems and the location of economic activities“. In: *Econometrica: journal of the Econometric Society* (1957), S. 53–76.

[21] Richard C Wilson. „A review of facility design models“. In: *The Journal of Industrial Engineering* 15 (1964), S. 115–121.

[22] TE El-Rayah und RH Hollier. „A review of plant design techniques“. In: *The International Journal of Production Research* 8.3 (1970), S. 263–279.

[23] James M Moore. „Computer aided facilities design: an international survey“. In: *International Journal of Production Research* 12.1 (1974), S. 21–44.

[24] Hasan Hosseini-Nasab, Sepideh Fereidouni, Seyyed Mohammad Taghi Fatemi Ghomi und Mohammad Bagher Fakhrzad. „Classification of facility layout problems: a review study“. In: *The International Journal of Advanced Manufacturing Technology* 94.1-4 (2018), S. 957–977.

[25] Srisatja Vitayasak, Pupong Pongcharoen und Chris Hicks. „A tool for solving stochastic dynamic facility layout problems with stochastic demand using either a genetic algorithm or modified backtracking search algorithm“. In: *International Journal of Production Economics* 190 (2017), S. 146–157.

[26] Amine Drira, Henri Pierreval und Sonia Hajri-Gabouj. „Facility layout problems: A survey“. In: *Annual reviews in control* 31.2 (2007), S. 255–267.

[27] Philipp Hungerlaender. „Single-row equidistant facility layout as a special case of single-row facility layout“. In: *International Journal of Production Research* 52.5 (2014), S. 1257–1268.

[28] JS Kochhar. „MULTI-HOPE: a tool for multiple floor layout problems“. In: *International Journal of Production Research* 36.12 (1998), S. 3421–3435.

[29] Dimitrios I Patsiatzis und Lazaros G Papageorgiou. „Optimal multi-floor process plant layout“. In: *Computers & Chemical Engineering* 26.4-5 (2002), S. 575–583.

[30] Abbas Ahmadi, Mir Saman Pishvaee und Mohammad Reza Akbari Jokar. „A survey on multi-floor facility layout problems“. In: *Computers & Industrial Engineering* 107 (2017), S. 158–170.

[31] Jiande Zhou. „Algorithmes et outils pour l'analyse des flux de production a l'aide du concept d'ordre“. Diss. Université Louis Pasteur (Strasbourg), 1998.

[32] Florian Mueller, Alessandro Cannata, Bojan Stahl, Marco Taisch, Sebastian Thiede und Christoph Herrmann. „Green Factory Planning“. In: *IFIP International Conference on Advances in Production Management Systems*. Springer. 2013, S. 167–174.

[33] Abdullah Konak, Sadan Kulturel-Konak, Bryan A Norman und Alice E Smith. „A new mixed integer programming formulation for facility layout design using flexible bays“. In: *Operations Research Letters* 34.6 (2006), S. 660–672.

[34] Ghorbanali Moslemipour, Tian Soon Lee und Dirk Rilling. „A review of intelligent approaches for designing dynamic and robust layouts in flexible manufacturing systems“. In: *The International Journal of Advanced Manufacturing Technology* 60.1-4 (2012), S. 11–27.

[35] Sadan Kulturel-Konak und Abdullah Konak. „Linear programming based genetic algorithm for the unequal area facility layout problem“. In: *International Journal of Production Research* 51.14 (2013), S. 4302–4324.

[36] Jing Tao, Peng Wang, Hong Qiao und Zhipeng Tang. „Facility layouts based on intelligent optimization approaches“. In: *2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE. 2012, S. 502–508.

[37] J-Y Kim und Y-D Kim. „Graph theoretic heuristics for unequal-sized facility layout problems“. In: *Omega* 23.4 (1995), S. 391–401.

[38] Jaydeep Balakrishnan, Chun-Hung Cheng und Kam-Fai Wong. „FACOPT: A user friendly FACility layout OPTimization system“. In: *Computers & Operations Research* 30.11 (2003), S. 1625–1641.

[39] Tarun Gupta und Ham Id Seifoddini. „Production data based similarity coefficient for machine-component grouping decisions in the design of a cellular manufacturing system“. In: *The international journal of production research* 28.7 (1990), S. 1247–1269.

[40] Saeed Emami und Ali S Nookabadi. „Managing a new multi-objective model for the dynamic facility layout problem“. In: *The International Journal of Advanced Manufacturing Technology* 68.9-12 (2013), S. 2215–2228.

[41] Meir J Rosenblatt. „The dynamics of plant layout“. In: *Management Science* 32.1 (1986), S. 76–86.

[42] Mostafa Abedzadeh, Mostafa Mazinani, Nazanin Moradinasab und Emad Roghanian. „Parallel variable neighborhood search for solving fuzzy multi-objective dynamic facility layout problem“. In: *The International Journal of Advanced Manufacturing Technology* 65.1-4 (2013), S. 197–211.

[43] Jaydeep Balakrishnan, Chun Hung Cheng, Daniel G Conway und Chun Ming Lau. „A hybrid genetic algorithm for the dynamic plant layout problem“. In: *International Journal of Production Economics* 86.2 (2003), S. 107–120.

[44] Jaydeep Balakrishnan und Chun Hung Cheng. „Dynamic layout algorithms: a state-of-the-art survey“. In: *Omega* 26.4 (1998), S. 507–521.

[45] Sadan Kulturel-Konak. „Approaches to uncertainties in facility layout problems: Perspectives at the beginning of the 21 st Century“. In: *Journal of Intelligent Manufacturing* 18.2 (2007), S. 273–284.

[46] Franz Liebl. *Simulation*. Pearson, 2018.

[47] Stuart Russel und Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2021.

[48] David L Poole und Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[49] Nils J Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[50] Volodymyr Mnih u. a. „Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), S. 529–533.

[51] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau und Sebastian Thrun. „Dermatologist-level classification of skin cancer with deep neural networks". In: *nature* 542.7639 (2017), S. 115–118.

[52] Mariusz Bojarski u. a. „End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[53] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[54] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[55] John H Holland. „Adaptation in natural and artificial systems". In: *Ann Arbor* (1975).

[56] Karsten Weicker. *Evolutionäre algorithmen*. Springer-Verlag, 2015.

[57] G Suresh, VV Vinod und S Sahu. „A genetic algorithm for facility layout". In: *International Journal of Production Research* 33.12 (1995), S. 3411–3423.

[58] David E Golberg. „Genetic algorithms in search, optimization, and machine learning". In: *Addion wesley* 1989.102 (1989), S. 36.

[59] Maheswaran Rajasekharan, Brett A Peters und Taho Yang. „A genetic algorithm for facility layout design in flexible manufacturing systems". In: *International journal of Production research* 36.1 (1998), S. 95–110.

[60] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[61] Fred Glover. „Tabu searchâĂŤpart I". In: *ORSA Journal on computing* 1.3 (1989), S. 190–206.

[62] Farhad Azadivar und John Wang. „Facility layout optimization using simulation and genetic algorithms". In: *International Journal of Production Research* 38.17 (2000), S. 4369–4383.

[63] l D Meller und Kai-Yin Gau. „The facility layout problem: recent and emerging trends and perspectives“. In: *Journal of manufacturing systems* 15.5 (1996), S. 351–366.

[64] Scott Kirkpatrick. „Optimization by simulated annealing: Quantitative studies“. In: *Journal of statistical physics* 34 (1984), S. 975–986.

[65] Marco Dorigo, Mauro Birattari und Thomas Stützle. „Ant Colony Optimization-Artificial Ants as a Computational Intelligence Technique. 2006“. In: *IEEE Computational Intelligence Magazine* ().

[66] James Kennedy und Russell Eberhart. „Particle swarm optimization“. In: *Proceedings of ICNN'95-international conference on neural networks*. Bd. 4. IEEE. 1995, S. 1942–1948.

[67] Maurice Clerc und James Kennedy. „The particle swarm-explosion, stability, and convergence in a multidimensional complex space“. In: *IEEE transactions on Evolutionary Computation* 6.1 (2002), S. 58–73.

[68] Hamed Samarghandi und Farzad Firouzi Jahantigh. „A particle swarm optimisation for fuzzy dynamic facility layout problem“. In: *International Journal of Metaheuristics* 1.3 (2011), S. 257–278.

[69] Tom Michael Mitchell u. a. *Machine learning*. Bd. 1. McGraw-hill New York, 2007.

[70] Guilherme E Vieira, Jeffrey W Herrmann und Edward Lin. „Rescheduling manufacturing systems: a framework of strategies, policies, and methods“. In: *Journal of scheduling* 6 (2003), S. 39–62.

[71] John D Kelleher, Brian Mac Namee und Aoife D'arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, 2020.

[72] Trevor Hastie, Robert Tibshirani, Jerome H Friedman und Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Bd. 2. Springer, 2009.

[73] Gareth James, Daniela Witten, Trevor Hastie und Robert Tibshirani. *An introduction to statistical learning*. Bd. 112. Springer, 2013.

[74] Jeffrey M Wooldridge. *Introductory econometrics: A modern approach*. Cengage learning, 2015.

[75] Trevor S Breusch und Adrian R Pagan. „A simple test for heteroscedasticity and random coefficient variation“. In: *Econometrica: Journal of the econometric society* (1979), S. 1287–1294.

[76] Halbert White. „A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity“. In: *Econometrica: journal of the Econometric Society* (1980), S. 817–838.

[77] Stephen M Goldfeld und Richard E Quandt. „Some tests for homoscedasticity“. In: *Journal of the American statistical Association* 60.310 (1965), S. 539–547.

[78] James Durbin und Geoffrey S Watson. *Testing for serial correlation in least squares regression. I*. Springer, 1992.

[79] Robert M OâĂŹbrien. „A caution regarding rules of thumb for variance inflation factors“. In: *Quality & quantity* 41 (2007), S. 673–690.

[80] Jerry A Hausman. „Specification tests in econometrics“. In: *Econometrica: Journal of the econometric society* (1978), S. 1251–1271.

[81] Christopher M Bishop und Nasser M Nasrabadi. *Pattern recognition and machine learning*. Bd. 4. 4. Springer, 2006.

[82] Dimitris Mourtzis, Ekaterini Vlachou und NJPC Milas. „Industrial big data as a result of IoT adoption in manufacturing“. In: *Procedia cirp* 55 (2016), S. 290–295.

[83] David W Hosmer Jr, Stanley Lemeshow und Rodney X Sturdivant. *Applied logistic regression*. Bd. 398. John Wiley & Sons, 2013.

[84] Thomas Cover und Peter Hart. „Nearest neighbor pattern classification“. In: *IEEE transactions on information theory* 13.1 (1967), S. 21–27.

[85] Kilian Q Weinberger und Lawrence K Saul. „Distance metric learning for large margin nearest neighbor classification.“ In: *Journal of machine learning research* 10.2 (2009).

[86] R Bellman. *Mar. 1961. Adaptive control processes: a guided tour*.

[87] Irina Rish u. a. „An empirical study of the naive Bayes classifier“. In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Bd. 3. 22. 2001, S. 41–46.

[88] Leo Breiman, Jerome Friedman, Richard Olshen und Charles Stone. „Cart“. In: *Classification and regression trees* (1984).

[89] Leo Breiman. „Random forests“. In: *Machine learning* 45 (2001), S. 5–32.

[90] Douglas M Hawkins. „The problem of overfitting“. In: *Journal of chemical information and computer sciences* 44.1 (2004), S. 1–12.

[91] Leo Breiman. „Bagging predictors“. In: *Machine learning* 24 (1996), S. 123–140.

[92] Tianqi Chen und Carlos Guestrin. „Xgboost: A scalable tree boosting system“. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, S. 785–794.

[93] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[94] Corinna Cortes und Vladimir Vapnik. „Support-vector networks“. In: *Machine learning* 20 (1995), S. 273–297.

[95] Bernhard Schölkopf, Alexander J Smola, Francis Bach u. a. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[96] Nello Cristianini, John Shawe-Taylor u. a. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[97] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[98] J MacQueen. „Some methods for classification and analysis of multivariate observations“. In: *Proc. 5th Berkeley Symposium on Math., Stat., and Prob*. 1965, S. 281.

[99] Anil K Jain. „Data clustering: 50 years beyond K-means“. In: *Pattern recognition letters* 31.8 (2010), S. 651–666.

[100] Laurens Van der Maaten und Geoffrey Hinton. „Visualizing data using t-SNE.“ In: *Journal of machine learning research* 9.11 (2008).

[101] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng und Jörg Sander. „LOF: identifying density-based local outliers“. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, S. 93–104.

[102] Fei Tony Liu, Kai Ming Ting und Zhi-Hua Zhou. „Isolation forest“. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, S. 413–422.

[103] Richard S Sutton und Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[104] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[105] Christopher JCH Watkins und Peter Dayan. „Q-learning“. In: *Machine learning* 8 (1992), S. 279–292.

[106] Ammar Haydari und Yasin Yılmaz. „Deep reinforcement learning for intelligent transportation systems: A survey“. In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2020), S. 11–32.

[107] Olivier Chapelle, Bernhard Scholkopf und Alexander Zien. „Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]“. In: *IEEE Transactions on Neural Networks* 20.3 (2009), S. 542–542.

[108] Longlong Jing und Yingli Tian. „Self-supervised visual feature learning with deep neural networks: A survey“. In: *IEEE transactions on pattern analysis and machine intelligence* 43.11 (2020), S. 4037–4058.

[109] Spyros Gidaris, Praveer Singh und Nikos Komodakis. „Unsupervised representation learning by predicting image rotations“. In: *arXiv preprint arXiv:1803.07728* (2018).

[110] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie und Ross Girshick. „Momentum contrast for unsupervised visual representation learning“. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, S. 9729–9738.

[111] Rob J Hyndman und Anne B Koehler. „Another look at measures of forecast accuracy“. In: *International journal of forecasting* 22.4 (2006), S. 679–688.

[112] James P Stevens. *Applied multivariate statistics for the social sciences*. Routledge, 2012.

[113] Davide Chicco, Niklas Tötsch und Giuseppe Jurman. „The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation“. In: *BioData mining* 14.1 (2021), S. 1–22.

[114] Tom Fawcett. „An introduction to ROC analysis“. In: *Pattern recognition letters* 27.8 (2006), S. 861–874.

[115] James A Hanley und Barbara J McNeil. „The meaning and use of the area under a receiver operating characteristic (ROC) curve.“ In: *Radiology* 143.1 (1982), S. 29–36.

[116] Robert E Schapire und Yoram Singer. „BoosTexter: A boosting-based system for text categorization“. In: *Machine learning* 39 (2000), S. 135–168.

[117] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep learning*. MIT press, 2016.

[118] Mary L McHugh. „Interrater reliability: the kappa statistic“. In: *Biochemia medica* 22.3 (2012), S. 276–282.

[119] Glenn W Brier u. a. „Verification of forecasts expressed in terms of probability“. In: *Monthly weather review* 78.1 (1950), S. 1–3.

[120] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. „Deep learning“. In: *nature* 521.7553 (2015), S. 436–444.

[121] Yoshua Bengio u. a. „Learning deep architectures for AI“. In: *Foundations and trends® in Machine Learning* 2.1 (2009), S. 1–127.

[122] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Communications of the ACM* 60.6 (2017), S. 84–90.

[123] Geoffrey Hinton u. a. „Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups“. In: *IEEE Signal processing magazine* 29.6 (2012), S. 82–97.

[124] David E Rumelhart, Geoffrey E Hinton und Ronald J Williams. „Learning representations by back-propagating errors“. In: *nature* 323.6088 (1986), S. 533–536.

[125] Jürgen Schmidhuber. „Deep learning in neural networks: An overview“. In: *Neural networks* 61 (2015), S. 85–117.

[126] Andrew Tchircoff. „The mostly complete chart of Neural Networks, explained“. In: *Towards Data Science* (2017), S. 1–29.

[127] Warren S McCulloch und Walter Pitts. „A logical calculus of the ideas immanent in nervous activity“. In: *The bulletin of mathematical biophysics* 5 (1943), S. 115–133.

[128] Vinod Nair und Geoffrey E Hinton. „Rectified linear units improve restricted boltzmann machines“. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, S. 807–814.

[129] Andrew L Maas, Awni Y Hannun, Andrew Y Ng u. a. „Rectifier nonlinearities improve neural network acoustic models“. In: *Proc. icml*. Bd. 30. 1. Atlanta, Georgia, USA. 2013, S. 3.

[130] Prajit Ramachandran, Barret Zoph und Quoc V Le. „Searching for activation functions“. In: *arXiv preprint arXiv:1710.05941* (2017).

[131] Peter J Huber. „Robust estimation of a location parameter“. In: *Breakthroughs in statistics: Methodology and distribution* (1992), S. 492–518.

[132] Fausto Milletari, Nassir Navab und Seyed-Ahmad Ahmadi. „V-net: Fully convolutional neural networks for volumetric medical image segmentation“. In: *2016 fourth international conference on 3D vision (3DV)*. Ieee. 2016, S. 565–571.

[133] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He und Piotr Dollár. „Focal loss for dense object detection“. In: *Proceedings of the IEEE international conference on computer vision*. 2017, S. 2980–2988.

[134] Florian Schroff, Dmitry Kalenichenko und James Philbin. „Facenet: A unified embedding for face recognition and clustering“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, S. 815–823.

[135] Solomon Kullback und Richard A Leibler. „On information and sufficiency“. In: *The annals of mathematical statistics* 22.1 (1951), S. 79–86.

[136] Léon Bottou. „Large-scale machine learning with stochastic gradient descent“. In: *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer. 2010, S. 177–186.

[137] Ilya Sutskever, James Martens, George Dahl und Geoffrey Hinton. „On the importance of initialization and momentum in deep learning“. In: *International conference on machine learning*. PMLR. 2013, S. 1139–1147.

[138] John Duchi, Elad Hazan und Yoram Singer. „Adaptive subgradient methods for online learning and stochastic optimization.“ In: *Journal of machine learning research* 12.7 (2011).

[139] Tijmen Tieleman und G Hinton. „Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning“. In: *Technical report* (2017).

[140] Diederik P Kingma und Jimmy Ba. „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980* (2014).

[141] Matthew D Zeiler. „Adadelta: an adaptive learning rate method“. In: *arXiv preprint arXiv:1212.5701* (2012).

[142] Sashank J Reddi, Satyen Kale und Sanjiv Kumar. „On the convergence of adam and beyond“. In: *arXiv preprint arXiv:1904.09237* (2019).

[143] Lutz Prechelt. „Early stopping-but when?“ In: *Neural Networks: Tricks of the trade*. Springer, 2002, S. 55–69.

[144] Sergey Ioffe und Christian Szegedy. „Batch normalization: Accelerating deep network training by reducing internal covariate shift“. In: *International conference on machine learning*. pmlr. 2015, S. 448–456.

[145] Yoshua Bengio, Patrice Simard und Paolo Frasconi. „Learning long-term dependencies with gradient descent is difficult“. In: *IEEE transactions on neural networks* 5.2 (1994), S. 157–166.

[146] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht und Oriol Vinyals. „Understanding deep learning (still) requires rethinking generalization“. In: *Communications of the ACM* 64.3 (2021), S. 107–115.

[147] James Bergstra und Yoshua Bengio. „Random search for hyper-parameter optimization.“ In: *Journal of machine learning research* 13.2 (2012).

[148] Leslie N Smith und Nicholay Topin. „Super-convergence: Very fast training of neural networks using large learning rates“. In: *Artificial intelligence and machine learning for multi-domain operations applications*. Bd. 11006. SPIE. 2019, S. 369–386.

[149] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy und Ping Tak Peter Tang. „On large-batch training for deep learning: Generalization gap and sharp minima“. In: *arXiv preprint arXiv:1609.04836* (2016).

[150] Dominic Masters und Carlo Luschi. „Revisiting small batch training for deep neural networks“. In: *arXiv preprint arXiv:1804.07612* (2018).

[151] Frank Hutter, Lars Kotthoff und Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[152] Jasper Snoek, Hugo Larochelle und Ryan P Adams. „Practical bayesian optimization of machine learning algorithms“. In: *Advances in neural information processing systems* 25 (2012).

[153] Yann LeCun, Léon Bottou, Yoshua Bengio und Patrick Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), S. 2278–2324.

[154] Jeffrey L Elman. „Finding structure in time“. In: *Cognitive science* 14.2 (1990), S. 179–211.

[155] Sepp Hochreiter und Jürgen Schmidhuber. „Long short-term memory“. In: *Neural computation* 9.8 (1997), S. 1735–1780.

[156] John J Hopfield. „Neural networks and physical systems with emergent collective computational abilities.“ In: *Proceedings of the national academy of sciences* 79.8 (1982), S. 2554–2558.

[157] John A Hertz. *Introduction to the theory of neural computation*. CRC Press, 2018.

[158] David H Ackley, Geoffrey E Hinton und Terrence J Sejnowski. „A learning algorithm for Boltzmann machines“. In: *Cognitive science* 9.1 (1985), S. 147–169.

[159] Kate A Smith. „Neural networks for combinatorial optimization: a review of more than a decade of research“. In: *Informs journal on Computing* 11.1 (1999), S. 15–34.

[160] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville und Yoshua Bengio. „Generative adversarial networks“. In: *Communications of the ACM* 63.11 (2020), S. 139–144.

[161] Alec Radford, Luke Metz und Soumith Chintala. „Unsupervised representation learning with deep convolutional generative adversarial networks“. In: *arXiv preprint arXiv:1511.06434* (2015).

[162] Martin Arjovsky, Soumith Chintala und Léon Bottou. „Wasserstein generative adversarial networks“. In: *International conference on machine learning*. PMLR. 2017, S. 214–223.

[163] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin und Aaron C Courville. „Improved training of wasserstein gans“. In: *Advances in neural information processing systems* 30 (2017).

[164] Xin Guo, Johnny Hong, Tianyi Lin und Nan Yang. „Relaxed Wasserstein with applications to GANs“. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, S. 3325–3329.

[165] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang und Stephen Paul Smolley. „Least squares generative adversarial networks“. In: *Proceedings of the IEEE international conference on computer vision*. 2017, S. 2794–2802.

[166] Yujia Li, Kevin Swersky und Rich Zemel. „Generative moment matching networks“. In: *International conference on machine learning*. PMLR. 2015, S. 1718–1727.

[167] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang und Barnabás Póczos. „Mmd gan: Towards deeper understanding of moment matching network“. In: *Advances in neural information processing systems* 30 (2017).

[168] Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer und Rémi Munos. „The cramer distance as a solution to biased wasserstein gradients“. In: *arXiv preprint arXiv:1705.10743* (2017).

[169] Youssef Mroueh und Tom Sercu. „Fisher gan“. In: *Advances in neural information processing systems* 30 (2017).

[170] Junbo Zhao, Michael Mathieu und Yann LeCun. „Energy-based generative adversarial network“. In: *arXiv preprint arXiv:1609.03126* (2016).

[171] David Berthelot, Thomas Schumm und Luke Metz. „Began: Boundary equilibrium generative adversarial networks“. In: *arXiv preprint arXiv:1703.10717* (2017).

[172] Ruohan Wang, Antoine Cully, Hyung Jin Chang und Yiannis Demiris. „Magan: Margin adaptation for generative adversarial networks“. In: *arXiv preprint arXiv:1704.03817* (2017).

[173] Han Zhang, Ian Goodfellow, Dimitris Metaxas und Augustus Odena. „Self-attention generative adversarial networks“. In: *International conference on machine learning*. PMLR. 2019, S. 7354–7363.

[174] Andrew Brock, Jeff Donahue und Karen Simonyan. „Large scale GAN training for high fidelity natural image synthesis“. In: *arXiv preprint arXiv:1809.11096* (2018).

[175] Tero Karras, Samuli Laine und Timo Aila. „A style-based generator architecture for generative adversarial networks“. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, S. 4401–4410.

[176] Tero Karras, Timo Aila, Samuli Laine und Jaakko Lehtinen. „Progressive growing of gans for improved quality, stability, and variation“. In: *arXiv preprint arXiv:1710.10196* (2017).

[177] Jun-Yan Zhu, Taesung Park, Phillip Isola und Alexei A Efros. „Unpaired image-to-image translation using cycle-consistent adversarial networks“. In: *Proceedings of the IEEE international conference on computer vision*. 2017, S. 2223–2232.

[178] Teuvo Kohonen. „Self-organized formation of topologically correct feature maps“. In: *Biological cybernetics* 43.1 (1982), S. 59–69.

[179] Juha Vesanto, Johan Himberg, Esa Alhoniemi, Juha Parhankangas u. a. „Self-organizing map in Matlab: the SOM Toolbox“. In: *Proceedings of the Matlab DSP conference*. Bd. 99. Espoo. 1999, S. 16–17.

[180] Stephen Grossberg und Stephen Grossberg. „How does a brain build a cognitive code?“ In: *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control* (1982), S. 1–52.

[181] Sinno Jialin Pan und Qiang Yang. „A survey on transfer learning“. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), S. 1345–1359.

[182] Burr Settles. „Active learning literature survey“. In: (2009).

[183] Timothy Hospedales, Antreas Antoniou, Paul Micaelli und Amos Storkey. „Meta-learning in neural networks: A survey“. In: *IEEE transactions on pattern analysis and machine intelligence* 44.9 (2021), S. 5149–5169.

[184] Daniel Jurafsky und James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Bd. 2. Feb. 2008.

[185] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser und Illia Polosukhin. „Attention is all you need“. In: *Advances in neural information processing systems* 30 (2017).

[186] Jacob Devlin, Ming-Wei Chang, Kenton Lee und Kristina Toutanova. „Bert: Pre-training of deep bidirectional transformers for language understanding“. In: *arXiv preprint arXiv:1810.04805* (2018).

[187] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever u. a. „Language models are unsupervised multitask learners“. In: *OpenAI blog* 1.8 (2019), S. 9.

[188] Tom Brown u. a. „Language models are few-shot learners“. In: *Advances in neural information processing systems* 33 (2020), S. 1877–1901.

[189] Rafael Ball und Dirk Tunger. *Bibliometrische Analysen-Daten, Fakten und Methoden-Grundwissen Bibliometrie für Wissenschaftler, Wissenschaftsmanager, Forschungseinrichtungen und Hochschulen*. Bd. 12. Forschungszentrum Jülich, 2005.

[190] Jan vom Brocke, Alexander Simons, Bjoern Niehaves, Bjorn Niehaves, Kai Reimer, Ralf Plattfaut und Anne Cleven. „Reconstructing the giant: On the importance of rigour in documenting the literature search process“. In: (2009).

[191] Peter Burggraef, Johannes Wagner und Benjamin Heinbach. „Bibliometric study on the use of machine learning as resolution technique for facility layout problems“. In: *IEEE Access* 9 (2021), S. 22569–22586.

[192] Daniel G Conway und Munirpallam A Venkataramanan. „Genetic search and the dynamic facility layout problem“. In: *Computers & Operations Research* 21.8 (1994), S. 955–960.

[193] Marco Dorigo und Gianni Di Caro. „Ant colony optimization: a new meta-heuristic“. In: *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. Bd. 2. IEEE. 1999, S. 1470–1477.

[194] Jaydeep Balakrishnan und Chun Hung Cheng. „Genetic search and the dynamic layout problem“. In: *Computers & Operations Research* 27.6 (2000), S. 587–593.

[195] Mirko Ficko, M Brezocnik und J Balic. „Designing the layout of single-and multiple-rows flexible manufacturing system by genetic algorithms“. In: *Journal of materials processing technology* 157 (2004), S. 150–158.

[196] GH Hu, YP Chen, ZD Zhou und HC Fang. „A genetic algorithm for the inter-cell layout and material handling system design“. In: *The International Journal of Advanced Manufacturing Technology* 34.11-12 (2007), S. 1153–1163.

[197] Yongzhong Wu und Ping Ji. „Solving the quadratic assignment problems by a genetic algorithm with a new replacement strategy“. In: *International Journal of Computational Intelligence* 4.3 (2007).

[198] Krishna K Krishnan, S Hossein Cheraghi und Chandan N Nayak. „Facility layout design for multiple production scenarios in a dynamic environment“. In: *International Journal of Industrial and Systems Engineering* 3.2 (2008), S. 105–133.

[199] Laura Garcia-Hernandez, Lorenzo Salas-Morera, Henri Pierreval und Antonio Arauzo-Azofra. „A Novel Hybrid Multi-criteria Decision-Making Model to Solve UA-FLP“. In: *International Symposium on Distributed Computing and Artificial Intelligence*. Springer. 2018, S. 292–299.

[200] Weidong Liang und Dianjun Fang. „Decentralized Genetic Algorithm for Dynamic Plant Layout Problem“. In: *IOP Conference Series: Materials Science and Engineering*. Bd. 677. 5. IOP Publishing. 2019, S. 52–80.

[201] Mariem Besbes, Marc Zolghadri, Roberta Costa Affonso, Faouzi Masmoudi und Mohamed Haddar. „A methodology for solving facility layout problem considering barriers: genetic algorithm coupled with A* search“. In: *Journal of Intelligent Manufacturing* 31.3 (2020), S. 615–640.

[202] Abu Qudeiri Jaber, Yamamoto Hidehiko und Rizauddin Ramli. „Machine Learning in Production Systems Design Using Genetic Algorithms“. In: *International Journal of Computational Intelligence* 4.1 (2007).

[203] Hannu Rummukainen, Jukka K. Nurminen, Timo Syrjänen und J.-P. Numminen. „Machine Learning from Prior Designs for Facility Layout Optimization“. In: 2018.

[204] Hikaru Ikeda, Hiroyuki Nakagawa und Tatsuhiro Tsuchiya. „Towards Automatic Facility Layout Design Using Reinforcement Learning“. In: *Annals of Computer Science and Information Systems* 32 (2022), S. 11–20.

[205] Kazuhiro Tsuchiya, Sunil Bharitkar und Yoshiyasu Takefuji. „A neural network approach to facility layout problems“. In: *European Journal of Operational Research* 89.3 (1996), S. 556–563.

[206] Motohiro Kobayashi, Toshiyuki Makita, S Matsui, Masahiro Koyama, Nobutada Fujii, Itsuo Hatono und Kanji Ueda. „Floor layout planning method based on self-organization“. In: *2001 IEEE International Symposium on Semiconductor Manufacturing. ISSM 2001. Conference Proceedings (Cat. No. 01CH37203)*. IEEE. 2001, S. 381–384.

[207] Kanji Ueda, N Fujii, I Hatono und M Kobayashi. „Facility layout planning using self-organization method“. In: *CIRP Annals* 51.1 (2002), S. 399–402.

[208] CM Tam und Thomas KL Tong. „GA-ANN model for optimizing the locations of tower crane and supply points for high-rise public housing construction“. In: *Construction Management and Economics* 21.3 (2003), S. 257–266.

[209] Laura García-Hernández, M Pérez-Ortiz, Antonio Arauzo-Azofra, Lorenzo Salas-Morera und César Hervás-Martínez. „An evolutionary neural system for incorporating expert knowledge into the UA-FLP“. In: *Neurocomputing* 135 (2014), S. 69–78.

[210] Parham Azimi und Parham Soofi. „An ANN-based optimization model for facility layout problem using simulation technique“. In: *Scientia Iranica* 24.1 (2017), S. 364–377.

[211] Yafei Hou, Kazuto Yano, Norisato Suga, Julian Webber, Eiji Nii, Toshihide Higashimori, Satoshi Denno und Yoshinori Suzuki. „A study of throughput prediction using convolutional neural network over factory environment“. In: *2022 24th International Conference on Advanced Communication Technology (ICACT)*. IEEE. 2022, S. 429–434.

[212] adigiconsult. *Konzept*. 7. Juli 2023. URL: https://www.adigiconsult.ch/glossar/konzept/ (besucht am 07. 07. 2023).

[213] Alexander Kossiakoff, Steven M Biemer, Samuel J Seymour und David A Flanigan. *Systems engineering principles and practice*. John Wiley & Sons, 2020.

[214] BS Blanchard und WJ Fabrycky. „Systems engineering and analysis“. In: (1998).

[215] Chen Sun, Abhinav Shrivastava, Saurabh Singh und Abhinav Gupta. „Revisiting unreasonable effectiveness of data in deep learning era“. In: *Proceedings of the IEEE international conference on computer vision*. 2017, S. 843–852.

[216] Luis Perez und Jason Wang. „The effectiveness of data augmentation in image classification using deep learning“. In: *arXiv preprint arXiv:1712.04621* (2017).

[217] Erhard Rahm, Hong Hai Do u. a. „Data cleaning: Problems and current approaches“. In: *IEEE Data Eng. Bull.* 23.4 (2000), S. 3–13.

[218] Isabelle Guyon und André Elisseeff. „An introduction to variable and feature selection“. In: *Journal of machine learning research* 3.Mar (2003), S. 1157–1182.

[219] Sebastian Raschka. „Model evaluation, model selection, and algorithm selection in machine learning“. In: *arXiv preprint arXiv:1811.12808* (2018).

[220] Majid Behzadian, S Khanmohammadi Otaghsara, Morteza Yazdani und Joshua Ignatius. „A state-of the-art survey of TOPSIS applications“. In: *Expert Systems with applications* 39.17 (2012), S. 13051–13069.

[221] Andrej Karpathy. *Unique mentions of deep learning frameworks in arxiv papers*. 12. Juni 2023. URL: https://twitter.com/karpathy/status/972295865187512320 (besucht am 12. 06. 2023).

[222] Martın Abadi u. a. „TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow. org (2015)“. In: *URL https://www. tensorflow. org* (2015).

[223] François Chollet u. a. *Keras*. https://keras.io. 2015.

[224] Adam Paszke u. a. „Pytorch: An imperative style, high-performance deep learning library“. In: *Advances in neural information processing systems* 32 (2019).

[225] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama und Trevor Darrell. „Caffe: Convolutional architecture for fast feature embedding“. In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, S. 675–678.

[226] Leslie N Smith. „Cyclical learning rates for training neural networks“. In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, S. 464–472.

[227] Richard Lyle Miller und Earl S Swensson. *Hospital and healthcare facility design*. WW Norton & Company, 2002.

[228] Jeff Heaton. *Heaton Research - YouTube Channel*. Zugriff am 29. Dezember 2024. n.d. URL: https://www.youtube.com/@HeatonResearch/videos.

[229] Niels Lohmann. *JSON for Modern C++*. 2022. URL: https://json.nlohmann.me.

[230] Gabi Melman. *SPDLOG Fast C++ Logging Library*. https://github.com/gabime/spdlog. 2022.

[231] Omar Cornut. *Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies*. https://github.com/ocornut/imgui. 2022.

[232] Camilla Löwy. *GLFW - A multi-platform library for OpenGL, OpenGL ES, Vulkan, window and input*. https://github.com/glfw/glfw. 2022.

[233] David Herberth. *Glad - Multi-Language Vulkan/GL/GLES/EGL/GLX/WGL Loader-Generator based on the official specs*. https://github.com/Dav1dde/glad. 2021.

[234] Sean Barrett. *STB - stb single-file public domain libraries for C/C++*. https://github.com/nothings/stb. 2021.

[235] Martín Abadi u. a. „Tensorflow: A system for large-scale machine learning“. In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, S. 265–283.

[236] Adam Paszke u. a. „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems 32*. Hrsg. von H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox und R. Garnett. Curran Associates, Inc., 2019, S. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[237] Michael L. Waskom. „seaborn: statistical data visualization“. In: *Journal of Open Source Software* 6.60 (2021), S. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[238] Alex Clark. *Pillow (PIL Fork) Documentation*. 2015. URL: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

[239] Pauli Virtanen u. a. „SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python“. In: *Nature Methods* 17 (2020), S. 261–272. DOI: 10.1038/s41592-019-0686-2.

[240] J. D. Hunter. „Matplotlib: A 2D graphics environment“. In: *Computing in Science & Engineering* 9.3 (2007), S. 90–95. DOI: 10.1109/MCSE.2007.55.

[241] Wes McKinney. „Data Structures for Statistical Computing in Python“. In: *Proceedings of the 9th Python in Science Conference*. Hrsg. von Stéfan van der Walt und Jarrod Millman. 2010, S. 56 –61. DOI: 10.25080/Majora-92bf1922-00a.

[242] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

Abkürzungsverzeichnis

ACO engl. Ant Colony Optimization 34
Adam Adaptive Moment Estimation 60
AGV Automated Guided Vehicle 16
ANN Artifical Neural Networks 55
API Application Programming Interface 110
AR Augmented Reality 185
ART Adaptive Resonance Theory 71
AUC-ROC Area Under the Receiver Operating Characteristic 52
BERT Bidirectional Encoder Representations from Transformers 74
BLUE Best Linear Unbiased Estimator 38
BMU Best Matching Unit 71
CNN Convolutional Neural Network 65
CPU Central Processing Unit 108
CS Continuous Simulation 27
DBSCAN Density-Based Spatial Clustering of Applications with Noise 45
DES Discrete Event Simulation 24
DFLP Dynamic Facility Layout Problem 21
DL Deep Learning 55
DNN Deep Neural Networks 55
DoE Design of Experiments 97
DRL Deep Reinforcement Learning 46
DRLP Double-Row Layout Problem 16
DSS Decision Support System 29
DT Decision Trees 42
ELU Exponential Linear Units 57
FNN Feedforward Neural Networks 65

FLP Facility Layout Problem 15
GA Genetische Algorithmen 31
GAN Generative Adversarial Networks 59
GBM Gradient Boosting Machines 43
GPU Graphics Processing Unit 55
GPT Generative Pretrained Transformer 74
GRU Gated Recurrent Units 66
GT Graphentheorie 19
HMI Human Machine Interface 74
IP Integer Programming 20
IQR Interquartilsabstand 95
JSD Jensen-Shannon-Divergenz 68
KEP Keeping Efficient Population 82
KI Künstliche Intelligenz 1
k-NN k-Nearest Neighbors 41
KL Kullback-Leibler Divergenz 59
KS-Test Kolmogorov-Smirnov-Test 153
LLM Large Language Models 74
LLP Loop Layout Problem 17
LOF Local Outlier Factor 45
LSTM Long Short-Term Memory 66
LP Linear Programming 20
MAE Mean Absolute Error 49
MAPE Mean Absolute Percentage Error 50
MCC Matthews-Korrelationskoeffizient 52
MEP Markov-Entscheidungsprozess 46
MES Manufacturing Execution System 92
MFLP Multi-Floor Layout Problem 17

MHC Material Handling Costs 1
MIP Mixed Integer Programming 20
ML Maschinelles Lernen 35
MLP Multilayer Perceptron 65
MMD Maximum Mean Discrepancy 69
MRLP Multi-Row Layout Problem 16
MSE Mean Squared Error 50
NAG Nesterov Accelerated Gradient 60
NER Named Entity Recognition 73
NLP Natural Language Processing 73
NLPP Nonlinear Programming Problem 20
OFLP Open Field Layout Problem 7
OE Organisationseinheiten 13
OLS Ordinary Least Squares 38
PCA Principal Component Analysis 45
PFLP Production Line Formation Problem 17
PReLU Parametric ReLU 57
PROP Parallel-Row Ordering Problem 16
PSO Partikel-Schwarm-Optimierung 35
QAP Quadratic Assignment Problem 18
QSP Quadratic Set Covering Problem 20
ReLU Rectified Linear Unit 56
RMSE Root Mean Squared Error 50
RNN Recurrent Neural Networks 66
ROC Receiver Operating Characteristic 52
SA Simulierte Abkühlung 34
SDLC Software Development Life Cycle 9
SFLP Static Facility Layout Problem 21

SGD Stochastic Gradient Descent 59
SL Supervised Learning 36
SOM Self Organized Maps 70
SQE Sum of Squares Explained 48
SQR Sum of Squares Residual 48
SQT Sum of Squares Total 48
SRLP Single-Row Layout Problem 16
SSL Self-Supervised Learning 47
SVM Support Vector Machines 44
TOPSIS Technique for Order of Preference by Similarity to Ideal Solution 97
t-SNE t-Distributed Stochastic Neighbor Embedding 45
RL Reinforcement Learning 36
TS Tabu Suche 33
VAE Variational Autoencoder 67
VR Virtual Reality 185
WGAN Wasserstein GAN 69
XAI Explainable AI 11
XGBoost Extreme Gradient Boosting 43

Abschließende Erklärung

Hiermit erkläre ich, dass diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Oldenburg, den 17. Januar 2025 Patrick Eschemann