TU WIEN architektur + raumplanung

DIPLOMARBEIT

DIPLOMA THESIS

Parametrische 2D-Architekturzeichnungen durch Formerkennung von Skizzen mittels künstlicher Intelligenz und weiterer algorithmischer Methoden.

ausgeführt zum Zwecke der Erlangung des akademischen Grades
Diplom-Ingenieur/Diplom-Ingenieurin
eingereicht an der TU-Wien, Fakultät für Architektur und Raumplanung

---

submitted in satisfaction of the requirements for the degree of
Diplom-Ingenieur/Diplom-Ingenieurin
at the TU Wien, Faculty of Architecture and Planning

von

Peter Hanna
01325227

unter der Betreuung von Dipl.-Ing. Dr. techn. Wolfgang Lorenz
Institut für Architekturwissenschaften
Forschungsbereich Digitale Architektur und Raumplanung

Technische Universität Wien,
Karlsplatz 13, 1040 Wien, Österreich

Datum _____ Unterschrift _____

=

Kurzfassung/Abstract

Kurzfassung

Das Ziel dieser Diplomarbeit ist die Entwicklung eines Algorithmus, der es ermöglicht, digitale Handskizzen (Skizzen, die in einem Computerprogramm erstellt wurden) in digitale, parametrische, zweidimensionale CAD-Zeichnungen zu konvertieren. Dadurch soll der kreative Arbeitsfluss des Nutzenden unterstützt werden, indem die zeitaufwendige manuelle Übertragung in eine CAD-Software und eine Parametrisierung gekürzt werden, was den Prozess intuitiver und einfacher machen soll.

Im Rahmen der Arbeit wurden Grasshopper und Python eingesetzt. Zudem wurde eine künstliche Intelligenz implementiert und trainiert, um die Umwandlung beliebiger digitaler handgezeichneter Skizzen zu ermöglichen. Die hierfür erforderlichen Trainingsdaten wurden spezifisch für diesen Zweck erstellt, um möglichst nutzungsspezifische und effiziente Ergebnisse zu erzielen.

Abstract

The aim of this thesis is the development of an algorithm that enables the conversion of digital sketches (sketches created in a computer program) into digital, parametric, 2D CAD drawings. This aims to support the creative workflow of users by reducing the time-consuming manual transfer into CAD software and its parameterization, making the process more intuitive and easier.

Grasshopper and Python were used in this work. Additionally, an artificial intelligence was implemented and trained to enable the conversion of any digital hand-drawn sketches. The training data required for this was specially created in order to achieve user-specific and efficient results.

Schlüsselwörter/Keywords

Schlüsselwörter

Künstliche Intelligenz (KI), Maschinelles Lernen (ML), Neuronales Netzwerk (NN), Mehrlagiges Perzeptron, Architektur, Formerkennung, Algorithmus, Parametrisch, Skizze, Zeichnung, CAD, 2D, Rhinoceros, Grasshopper, Python, Computergestütztes Entwerfen, Digitale Werkzeuge, Geometrie

Keywords

Artificial Intelligence (AI), Machine Learning (ML), Neural Network (NN), Multi-Layer Perceptron, Architecture, Shape recognition, Algorithm, Parametric, Sketch, Drawing, CAD, 2D, Rhinoceros, Grasshopper, Python, Computational Design, Digital tools, Geometry

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Datum	Unterschrift

Danksagung

Ich möchte allen danken, die mich in physischer oder geistiger Weise weitergebracht haben.

Mein Dank gilt meinen Eltern, deren Bodenständigkeit und Einfachheit mir eine solide Basis für vieles gegeben haben.

Ebenso gilt mein Dank meinem Bruder, dessen ausgeprägte geduldige Haltung mich beeinflusst hat.

Er gilt überdies meinem Betreuer Dipl.-Ing. Dr. techn. Wolfgang Lorenz für sein Interesse und seine Offenheit gegenüber dem Thema dieser Arbeit, seine umfassende Begleitung und seine kompakte und prägnante Lehrweise.

Mein Dank gilt den Teilnehmenden der Studie, welche das Engagement und die Bereitschaft hatten ihre Zeit zur Verfügung zu stellen.

Auch meinen Kolleginnen und Kollegen möchte ich meinen Dank aussprechen, die durch ihre gute Zusammenarbeit, ihren Teamgeist und Humor die Arbeit und das Arbeitsklima angenehmer gemacht haben.

Weiters gilt mein Dank meinen Freundinnen und Freunden, die auch in Momenten ohne besonderen Anlass an meiner Seite geblieben sind.

Schließlich danke ich den Veranstalterinnen und Veranstalter des GCD-Symposiums 2024, die durch ihre Präsentationen und Diskussionen zusätzliche Motivation geschaffen haben.

$\times$

Parametrische 2D-Architekturzeichnungen durch Formerkennung von Skizzen mittels künstlicher Intelligenz und weiterer algorithmischer Methoden.

Inhaltsverzeichnis

1 Einführung

1.1 Vorwort

Algorithmen (eine Abfolge von Anweisungen [1]) helfen den Menschen, ein bestimmtes Problem zu lösen. Sie automatisieren Prozesse und verarbeiten Informationen. [1] Dabei sind Algorithmen keine Erfindung der Moderne. Ansätze dazu finden sich beispielsweise bereits um 1700 vor Christus [2] in der ägyptischen Mathematik zur Multiplikation. [3]

Ein bedeutender technologischer Fortschritt unserer Zeit ist künstliche Intelligenz [4], eine Form des Algorithmus [5]. Sie wird von Personen sowohl privat als auch in der Arbeitswelt benutzt und besitzt nicht nur gesellschaftliche, sondern auch wirtschaftliche Auswirkungen. Das Potenzial dieser Technologie ist demzufolge groß. [4]

In Zukunft wird laut M.-T. Hütt und C. Schubert nahezu überall eine Form der KI verwendet werden [5], womit hinsichtlich ihres großen Potenzials umfassende Veränderungen und deren Auswirkungen einhergehen werden. [4]

Während meines Architektur-Studiums habe ich mich im Masterstudium auf Algorithmen und Computational Design spezialisiert. Als ich die damals für mich neuartige Software Grasshopper® [6] entdeckte, ließ sie mich nicht mehr los. Die Anziehungskraft, die diese Software auf mich ausübte, war proportional zu den algorithmischen Möglichkeiten, die sie bot. Seitdem habe ich sie in vielen Lehrveranstaltungen und Entwürfen, aber auch privat in verschiedenen Projekten benutzt.

Ich gelangte an einen Punkt, an dem ich das Potenzial eines Algorithmus weiter ausschöpfen und einen Mehrwert durch die Verwendung von KI im täglichen Leben von Architektinnen und Architekten schaffen wollte. Daher kombinierte ich eine grundlegende Tätigkeit des Planenden – das Skizzieren – mit den Möglichkeiten einer KI und erarbeitete mir so den Grundstein des Themas dieser Diplomarbeit.

Das Ziel besteht darin, Architektinnen und Architekten mit einem Werkzeug auszustatten, das ihnen zukunftsweisend dient. Dabei ist im Spezifischen nicht das Ziel, Architektinnen

und Architekten durch KI zu ersetzen, sondern sie oder ihn zu unterstützen. Dieses Konzept bietet eine Antwort auf die Gefahr, dass KI die formgebende und künstlerische Tätigkeit der Architektinnen und Architekten verdrängen könnte [4], wenn sie nicht richtig gelenkt und eingesetzt wird. Ich hoffe, dass die gewonnenen Erkenntnisse inspirieren und für zukünftige wissenschaftliche Zwecke zur Entwicklung dienen können.

Peter Hanna

1.2 Problemstellung

Architektinnen und Architekten haben ein komplexes Arbeitsfeld. Sie müssen sowohl im großen Maßstab als auch im Detail viele Aspekte berücksichtigen. Ihre Arbeit ist multidisziplinär und unterliegt konstanten Änderungen. Mit zunehmender Größe und Komplexität eines Projekts werden die Herausforderungen verstärkt. Architektinnen und Architekten würden demzufolge von einer Vereinfachung der Erstellung und Änderung der Planung profitieren.

Eine grundlegende Tätigkeit von Architektinnen und Architekten ist das Skizzieren, das als erster Schritt zur Konkretisierung einer Idee oder Überlegung dient. [7] Im Anschluss wird diese in eine digitale Version umgewandelt [7], wofür aufgrund deutlicher Vorteile wie einer hohen Genauigkeit oder flexiblen Bearbeitbarkeit typischerweise eine CAD-Software (Computer-Aided Design) benutzt wird. [8] Änderungen, die im Laufe des Projekts notwendig werden, werden anschließend direkt in der digitalisierten Zeichnung vorgenommen.

Dabei stellt sich die Frage, wie Architektinnen und Architekten in dieser grundlegenden Tätigkeit unterstützt werden können. Eine denkbare Lösung besteht darin, Skizzen automatisiert in eine digitale Version umzusetzen, die zugleich leicht änderbar bleibt. Die Umsetzung dieser Lösung birgt einige Fragen und Herausforderungen. Da Skizzen je nach Person unterschiedlich gezeichnet werden und aussehen können, wird ein Mittel zur Erkennung einer Skizze benötigt, um von der Darstellung eines Objekts auf das intendierte Objekt schließen zu können. Weiters sollte das Verfahren die Möglichkeit bieten, nachträgliche Änderungen problemlos umzusetzen. Schließlich sollte das Verfahren in einer bestehenden CAD-Software verwendet werden können, um den Arbeitsablauf bestmöglich zu unterstützen.

1.3 Ziel der Arbeit

Diese Arbeit hat das Ziel, digitale handgezeichnete Skizzen automatisiert in digitale und veränderbare Zeichnungen zu übertragen (siehe Abbildung 1). Ein Fokus liegt dabei auf dem Einsatz von künstlicher Intelligenz (KI), um eine zuverlässige Erkennung der Skizze im Sinne der Erkennung des intendierten Objekts zu ermöglichen. Ein weiterer Fokus liegt auf der anschließenden Umsetzung des Erkannten in eine parametrische Form, um die Änderbarkeit der Zeichnung sicherzustellen. Darüber hinaus geht es darum, ein möglichst nutzungsfreundliches Verfahren zu entwickeln. Dadurch soll eine unterstützende Wirkung erzielt werden, die den kreativen Fluss, also den Prozess der Ideenfindung und Ideenumsetzung, nicht unterbricht.

Hauptsächlich soll eine praktische Lösung entwickelt werden, die den Entwurfs- und Planungsprozess erleichtert. Abschließend ist anzumerken, dass es in dieser Arbeit spezifisch um 2D-Zeichnungen geht. Perspektivisches Zeichnen wird dabei nicht behandelt.

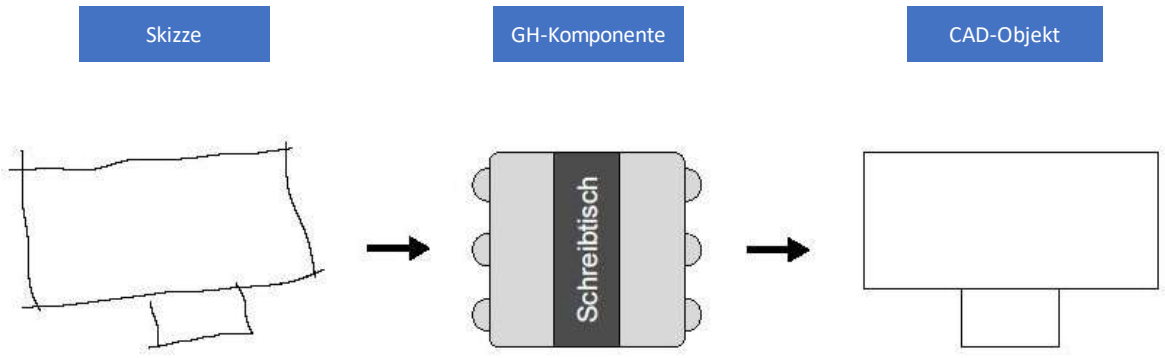


Abbildung 1: Konzeptuelle Darstellung des Ziels der Arbeit (eigene Darstellung)

1.4 Aufbau der Arbeit

Die Arbeit ist folgendermaßen aufgebaut:

Kapitel 1 (Einführung) beinhaltet das Vorwort, welches auch die persönliche Motivation beinhaltet. Die Problembeschreibung, die Zielsetzung und den Aufbau der Arbeit.

Kapitel 2 (Stand der Technik) präsentiert den aktuellen Stand der Forschung. Dafür wurden priorisierte und relevante wissenschaftliche Referenzen gefiltert. Diese wurden einzeln beschrieben und analysiert. Dadurch wurden wichtige und für die vorliegende Arbeit wesentliche Entwicklungen im Bereich des Themas dieser Arbeit beleuchtet.

Kapitel 3 (Methodik) beschreibt die Methode, die anhand der Beantwortung folgender Fragen zur Bearbeitung des Themas verwendet wurde:

- Was ist die grundlegende Tätigkeit von Architektinnen und Architekten und wie kann diese unterstützt werden?
- Wie kann eine Zeichnung automatisch in eine parametrische Zeichnung/ein parametrisches Objekt umgesetzt werden?
- Wie kann das Skizzieren durch KI erkannt werden?
- Wie kann Erkanntes in parametrische Objekte umgesetzt werden?
- Wie wird das parametrische Objekt in Bezug auf das Verfahren mit der KI benutzt?

Kapitel 4 (Umsetzung) beinhaltet die konkrete Umsetzung der Methodik, die in folgende Kapitel unterteilt wurde:

- KI-Modell,
- Skizzendaten und parametrische Objektdaten,
- Integration.

Kapitel 5 (Ergebnisse und Diskussion) stellt die Ergebnisse der Umsetzung dar und reflektiert diese im Zusammenhang mit der Problemstellung und dem aktuellen Stand der Technik. Dabei werden die Grenzen der Arbeit verdeutlicht. Zusätzlich wird eine Studie zur Evaluation des Algorithmus beschrieben und ausgewertet.

Kapitel 6 (Zusammenfassung und Ausblick) stellt eine Zusammenfassung und einen Ausblick auf mögliche Weiterentwicklung bereit.

Kapitel 7 und 8 (Verzeichnisse, Anhang) umfasst die Verzeichnisse und den Anhang, in dem Python-Codes, Protokolle und der Fragebogen für die Studie zu finden sind.

2 Stand der Technik

2.1 Einleitung

In diesem Kapitel wird ein Überblick über den aktuellen Stand der Technik gegeben, der dazu dient, den Kontext sowie die Relevanz dieser Arbeit im bestehenden Wissensfeld zu erfassen. Dabei handelt es sich um eine gezielte Auswahl relevanter Erkenntnisse für diese Arbeit.

Zwei grundlegende Bereiche ergeben sich für die Suche, auf die nachfolgend näher eingegangen wird:

1. **Wissenschaftliche Erkenntnisse**

Dieser Bereich beinhaltet die fundierten theoretischen Auseinandersetzungen und wissenschaftlichen Forschungserkenntnisse, die hauptsächlich aus wissenschaftlichen Artikeln, Buchkapitel und Konferenzbeiträgen entnommen werden. Dies stellt den wissenschaftlichen Stand der Technik dar.

2. **Angewendete Technologien**

Hier liegt der Fokus auf den vorhandenen, praktischen Werkzeugen, die zur Lösung der Probleme in der Praxis eingesetzt werden. Im Zuge dessen werden mögliche Grasshopper-Plugins diskutiert. Insgesamt stellt dies die praktische Anwendung des Standes der Technik dar.

2.2 Wissenschaftliche Erkenntnisse

Für die Recherche wurden verschiedene wissenschaftliche Suchmaschinen und Datenbanken genutzt. Unter anderem wurde BASE (Bielefeld Academic Search Engine) [9] verwendet, die eine der weltweit größten Suchmaschinen für wissenschaftliche Dokumente ist und über 11.000 Datenlieferanten beinhaltet, darunter SpringerLink [10], ScienceDirect [11] und JSTOR (Journal Storage) [12]. Darüber hinaus wurden EOSC (European Open Science Cloud) [13], Scopus [14] und CumInCAD (Cumulative Index of Computer Aided Architectural Design) [15] verwendet.

Bei der Filterung der wissenschaftlichen Publikationen wurden aussagekräftige Begriffe aus den Schlüsselwörtern ausgewählt. Zusätzlich wurde, soweit möglich, eine Filterung nach Themengebiet, Dokumententyp (Artikel, Buchkapitel und Konferenzbeiträge), Veröffentlichungsdatum (innerhalb der letzten 10 Jahre), Relevanz zum Thema und Vertrauenswürdigkeit der Publikation vorgenommen.

Die Relevanz wurde in erster Linie inhaltlich aus dem Titel und der Kurzfassung entnommen. Für die Vertrauenswürdigkeit der Quelle wurde keine Quellenkritik im Sinne einer Bestimmung der Entstehungsumstände der Quelle vorgenommen. [16] Stattdessen wurde darauf geachtet, ausschließlich peer-reviewte Publikationen zu verwenden. In diesem Verfahren werden wissenschaftliche Arbeiten durch andere Forschende aus demselben Fachgebiet überprüft. [17]

Dies ist ein Beispieltext, der in der Suche verwendet wurde:

(sketch OR curve* OR drawing* OR 2d OR shape recognition OR reconstruct* OR shape detection) AND (machine learning OR ai OR neural network OR classification OR parametric* OR algorithm OR grasshopper) NOT (video OR speech OR fashion OR medical) doctype:(1* 7 F) access:(1 2) lang:(eng OR ger) year:[2015 TO *]*

Einige der gefilterten und relevanten wissenschaftlichen Publikationen wurden gelesen, aber drei wesentliche Arbeiten werden im Einzelnen genauer vorgestellt.

Publikation 1

In der Arbeit „A Novel Approach of Deep Convolutional Neural Networks for Sketch Recognition" von Sadouk, Gadi und Essoufi [18] wird ein neuer Ansatz für die Erkennung von Skizzen mithilfe von vielschichtigen faltungsbasierten neuronalen Netzwerken (Convolutional Neural Network, CNN) untersucht.

Hauptziel und Schwerpunkt:

Das Hauptziel dieser Arbeit ist, Merkmale für die Skizzenklassifizierung mithilfe von faltungsbasierten neuronalen Netzwerken zu erlernen. Der Schwerpunkt liegt dabei einerseits auf dem automatischen Extrahieren von Merkmalen unterschiedlicher Abstraktionsgrade einer Skizze und andererseits auf einer hohen Genauigkeit der Klassifizierung. Die Genauigkeit wird in einem Vergleich zu der Genauigkeit von etwa 500 Testpersonen [19] gesetzt, die eine Erkennungsgenauigkeit von 73,1 % aufweisen. [18]

Verwendete Methodik:

Vorerst wurden die Bilder des Datensatzes vorbereitet, indem sie verkleinert und auf eine einheitliche Größe geändert wurden. Da der vorhandene Trainingsdatensatz sehr klein war (56 Bilder pro Kategorie bei 250 Kategorien [18]), musste dieser erweitert werden. Dies wird als Datenaugmentation bezeichnet. Diese wurde durch Replizieren und Transformieren der Bilder bewerkstelligt. Die Transformationen sind eine Spiegelung (über die vertikale Achse), eine zufällige Drehung zwischen -30° und 30° [18], eine Neuskalierung der Breite und Höhe des Bildes und eine horizontale Verschiebung.

Zudem wurde von Grund auf ein eigenes CNN-Modell erstellt. Dieses soll außer dem üblichen Layout des CNN und dessen Parameter ein Verfahren besitzen, um das Criterion (Loss-Funktion) zu optimieren. Dazu mussten einige Hyperparameter angepasst werden,

darunter die Lernrate, das Momentum, die Batchgröße und die Anzahl der Iterationen des Trainings. [18]

Das Layout (bzw. die Architektur) des CNN besitzt acht Schichten, die jeweils unterschiedliche Konfigurationen hinsichtlich der Filtergrößen und Aktivierungsfunktionen aufweisen. [18]

Zu den Hyperparametern, die mit dem Layout und dem Criterion verbunden sind, gehören die versteckten Schichten sowie der Regularisierer. Um Überanpassung zu verhindern, wurde eine stärkere Regularisierung in Betracht gezogen. Zur Anwendung kam schließlich die Dropout-Regularisierung, in der Knoten (Neuronen) zufällig ausfallen. Dadurch wird verhindert, dass einzelne Neuronen zu stark von anderen Neuronen beeinflusst werden. [18]

Um die Genauigkeit der Vorhersagen zu erhöhen, wurde ein Multi-Angle- und Multi-Scale-Voting-Verfahren eingesetzt. Dieses basiert darauf, dass das CNN auf augmentierten Daten trainiert wurde, und umfasst einige Schritte. Acht Bildvarianten werden einzeln durch das Netzwerk geschickt, wodurch acht verschiedene Ausgaben generiert werden. Diese werden gemittelt, um eine einzige zusammengefasste Ausgabe für das Bild zu erhalten. Die Bildvariante mit der höchsten Wahrscheinlichkeit wird als Vorhersage gewählt und mit dem tatsächlichen Etikett (Label) des Bildes verglichen. [18]

Das Verfahren ähnelt dem Ansatz einer Zusammensetzung mehrerer Netzwerke, die unabhängig voneinander trainiert und deren Ausgaben kombiniert werden. Im Unterschied dazu kombiniert das verwendete Verfahren die Ausgaben eines einzelnen Modells, das auf mehreren Varianten eines Bildes getestet wird. Dies führt dazu, dass die Genauigkeit der Vorhersagen deutlich verbessert wird. [18]

Ergebnisse und Erkenntnisse:

Mit höherer Anzahl transformierter Bildvarianten wurde die Genauigkeit der Vorhersagen verbessert, allerdings mit zunehmend geringerer Steigung. Ausgehend von einer

Klassifizierungsgenauigkeit von 71,7 % wurde letztendlich eine Genauigkeit von 75,3 % [18] erreicht. Damit ist das Ergebnis höher als das von Menschen (73,1 % [18]). Es wurde festgestellt, dass größere Filter in den ersten Schichten besser für die Erkennung geeignet sind.

Relevanz für die eigene Arbeit:

Für die eigene Arbeit ist die automatische Anpassung der Lernrate von Relevanz, da diese zu einer Optimierung der Trainingszeit führt. Der vorgestellte Ansatz besteht darin, mit einer Lernrate von 0,01 zu beginnen, um diese dann mit jeder Iteration zu reduzieren, sodass die Steigung geringer wird. Außerdem ist der Ansatz der Datenvermehrung durch Transformationen sinnvoll, da auf diese Weise die Genauigkeit der Vorhersage verbessert wird. Warum so wenige Transformationsarten verwendet wurden, bleibt offen.

Nicht von Relevanz ist, ein CNN-Modell zu benutzen, da dies für Bilddaten geeignet ist; für meine Arbeit sollte ein Modell für den Umgang mit Vektordaten gewählt werden.

Eigene kritische Bewertung:

Der Schritt, die Bilder des Datensatzes zunächst einheitlich zu verkleinern, erscheint plausibel. Allerdings stellt sich die Frage, inwiefern dieser Ansatz auf Bilder anderer Datensätze anwendbar ist und welche Größe dabei gewählt werden sollte. Eine stärkere Verkleinerung der Bilder könnte zu schlechteren Ergebnissen bei der Klassifizierung führen, da wichtige Details verloren gehen könnten.

Positiv hervorzuheben ist die Idee, ein CNN mit acht Schichten zu benutzen, bei dem jede Schicht andere Konfigurationen aufweist. Dieses ermöglicht eine diversifizierte Klassifizierung und bietet die Möglichkeit, Merkmale auf unterschiedlichen Abstraktionsebenen zu erfassen, was für Bilddaten sinnvoll erscheint.

Publikation 2

Die Arbeit „SketchGPT: Autoregressive Modeling for Sketch Generation and Recognition“ von Tiwari, Biswas und Lladós [20] befasst sich mit der Anwendung autoregressiver Modelle, um Skizzen zu generieren und zu erkennen.

Hauptziel und Schwerpunkt:

Einige Beiträge wie Sketch-BERT (Sketch Bidirectional Encoder Representation from Transformer), in denen die Sprachmodellierungstechniken von BERT angepasst wurden, um für die Skizzenerkennung geeignet zu sein, gibt es bereits. Das Hauptziel dieses Buchkapitels von Tiwari et al. [20] ist jedoch ein bislang fehlendes ganzheitliches Modell. Dieses Modell umfasst Skizzen von der Generierung bis zur Klassifizierung, ohne dass ein zwei gerichtetes Kontextverständnis (vorwärts und rückwärts) und ein Encoder (Daten Verarbeiter oder Umwandler) notwendig werden. Dem Modell wurde der Name SketchGPT zugewiesen, da ein GPT-Modell (Generative Pretrained Transformer) als Inspirationsquelle genommen wurde. [20]

Verwendete Methodik:

SketchGPT wurde auf dem QuickDraw-Datensatz vortrainiert und soll sequenzielle Abhängigkeiten zwischen den Strichen lernen können. Dadurch soll das Modell für beide Aufgaben – Skizzengenerierung (worunter Generierung und Vervollständigung fallen) und Skizzenklassifizierung – nützlich werden. Der nächste Schritt war eine Abstraktion der Skizzen, indem die Punkte in Strich-3-Format (x, y, p) umgewandelt und als Abfolge von vordefinierten Primitiven ersetzt wurden. So entsteht eine Folge zusammengehöriger Zeichen, die auch als Token bezeichnet werden. Das Strich-zu-Primitiv-Mapping wurde durch die Arbeit von Alaniz et al. [21] inspiriert. Die vordefinierten Primitive sind Linien, die in einem Wörterbuch aufgelistet sind. Abhängig von der Krümmung der Eingabeskizze werden die passenden Linien mit unterschiedlichen Längen gewählt. [20]

Als nächstes wurde das Modell durch einen großen Korpus an Skizzendaten trainiert, um so das nächste Token in der Folge auf Basis der vorherigen Tokens (auch Kontextfenster genannt) vorhersagen zu können. Dazu wurde eine spezielle Zielfunktion (Loss-Funktion) für das Vortraining definiert, die minimiert werden soll. [20]

Nach dem Vortraining wurde das Modell durch einen Feinabstimmungsprozess für weitere Aufgaben verändert. Für die Skizzengenerierung und Skizzenvervollständigung wird das Modell mit unvollständigen Skizzen gefüttert, um die fehlenden Striche vorherzusagen. Diesmal geschieht dies allerdings innerhalb eines bestimmten Klassenkontexts mit einem begrenzten Korpus an Skizzendaten. Dieser Schritt umfasst gleichzeitig die Skizzengenerierung ohne jegliche Eingabe von Skizzendaten. Für die Skizzenklassifizierung wird das trainierte Modell so verändert, dass eine vollständige Skizze als Eingabe eingesetzt und die zugehörige Kategorie vorhergesagt werden. Die Zielfunktion wurde im Feinabstimmungsprozess zugunsten der Klassifikation verändert. [20]

Daraufhin wurde eine experimentelle Überprüfung der Methode durchgeführt, um die Fähigkeiten einzuschätzen. Das Training dazu beinhaltete sowohl das Vortraining mit einem großen Korpus als auch die Feinabstimmung. Sowohl die Skizzengenerierung als auch die Skizzenklassifizierung wurden überprüft. Die Leistung des Modells wurde sowohl quantitativ, durch Vergleich mit dem Modell SketchRNN, als auch qualitativ, durch von Personen bewertete Kriterien, evaluiert. [20]

Ergebnisse und Erkenntnisse:

Die Ergebnisse der Studie zeigen, dass SketchGPT bei der Skizzengenerierung kreativere Ergebnisse und bei der Skizzenklassifizierung genauere Vorhersagen als andere Modelle wie Doodleformer und Pixelor liefern kann. [20]

Durch die Vortrainings- und Feinabstimmungsmethoden ist SketchGPT vielseitig einsetzbar und kann in allen erzielten Aufgabenbereichen eine hohe Leistung erzielen. [20]

Eine Einschränkung des Modells zeigt sich in der Repräsentation der Daten, in der der Abstraktionsprozess zu einem Informationsverlust führt. Daher wurde geraten, dass zukünftige Forschungen einen verbesserten Ansatz für die Repräsentation der Daten entwickeln sollten, um die Modellleistung ausarbeiten zu können. [20]

Relevanz für die eigene Arbeit:

Die Möglichkeit der Datengenerierung ist für meine Arbeit von großer Relevanz. Alternativ könnte ein vorhandener Datensatz wie der QuickDraw-Datensatz benutzt werden, was jedoch nicht beabsichtigt ist. Der Ansatz der Skizzenvervollständigung ist hingegen nicht von Bedeutung. Sehr interessant wäre allerdings, ein autoregressives Model zu erproben, um eine Skizze in Kontext zu vorhandenen Skizzen oder einem Skizzenumfeld setzen zu können. Dadurch eröffnet sich die Möglichkeit, dass die Vorhersage der Klassifikation einer Skizze durch eine Interpretation des Umfeldes der Skizze kontextualisiert wird.

Eigene kritische Bewertung:

Besonders positiv ist der Ansatz der Strich-zu-Primitiv-Umwandlung hervorzuheben, der eine Reduzierung der Komplexität der Skizze darstellt. Zusätzlich wurde eine effiziente Reduzierung durch die Verwendung von vordefinierten Modulen (Linien) erreicht. Dabei geht es um eine mehrfache Vereinfachung einer Skizze, die die Leistung des Modells erhöht.

Allerdings führt dieser Ansatz ebenso zu einem Informationsverlust. Details der Skizze und damit an Genauigkeit gehen verloren, was die Klassifizierung bis zu einem gewissen Grad negativ beeinflussen könnte. Das gilt vor allem, wenn von Menschen gezeichnete Skizzen eine höhere Komplexität oder Genauigkeit aufweisen, die wiederum notwendig für die Unterscheidung oder Klassifizierung ist. Potenziell werden durch den Strich-zu-Primitiv-Prozess die obersten möglichen Prozente einer Vorhersage gekappt, wenn diese auf eine

von Menschen gezeichnete Skizze angewendet wird. Dadurch ergibt sich allerdings die Möglichkeit der Verwendung von Tokens und damit in weiterer Folge die Autoregressivität.

Publikation 3

Der Konferenzbeitrag „SketchOpt: Sketch-based Parametric Model Retrieval for Generative Design“ von Keshavarzi, Hotson, Cheng, Nourbakhsh, Bergin und Asl [22] widmet sich einer neuen Methode, um Skizzen für generatives Design in parametrische Modelle umzuwandeln.

Hauptziel und Schwerpunkt:

Das Hauptziel dieser Arbeit ist ein neues automatisiertes generatives Entwurfssystem mit dem Namen SketchOpt, mit dem Gebäude Designerinnen und Designer einen handgezeichneten Grundriss in ein parametrisches Modell konvertieren. In weiterer Folge sollen damit Gebäudeoptimierungsaufgaben getätigt werden. Der Schwerpunkt liegt dabei auf der Funktion, dass die Nutzenden mithilfe einfacher Anmerkungen in der Skizze verschiedene Variablen in Echtzeit zuweisen kann. Der Prozess soll einsteigenden Designerinnen und Designer ohne Programmierkenntnisse die Erstellung parametrischer Modelle erleichtern. [22]

Verwendete Methodik:

Der erste Schritt besteht darin, handgezeichnete rasterbasierte Grundrisse in vektorisierte Daten umzuwandeln. Dabei werden durch eine spezielle Form einer asymmetrischen Faltung lineare Merkmale erkannt. Dies geschieht über mehrere unterschiedliche Auflösungen, um eine genaue Linienerkennung sicherzustellen. Diese Vektorisierung erfolgt für eine schnelle Verarbeitung in einer Multi-GPU-Umgebung. [22]

Im nächsten Schritt wird das Parametrisierungsmodul eingesetzt. Durch gemeinsame Attribute der vektorisierten geometrischen Segmente werden durch ein Graphensystem topologische und geometrische Beziehungen zwischen den Elementen gefunden. [22]

Gebäudeelemente wie Wände werden automatisch durch vordefinierte architektonische Annahmen generiert und auf Grundlage von Typ, Position und Kontinuität gruppiert. Gruppen haben eine Beziehung zu deren benachbarten Gruppen, sodass gemeinsame Parameter für Elemente entstehen. Jedes Element besteht aus einem Knoten und einem Pfad. Änderungen der Parameter eines Elements wirken sich automatisch auf die anderen Elemente aus. Dadurch bleibt das geometrische Layout erhalten. [22]

Der dritte Schritt ist die Integration von Entwurfsbeziehungen und Einschränkungen. Der Nutzende kann spezifische Variablen und Einschränkungen in die Skizze einfügen, indem einfache Anmerkungen gezeichnet werden. Diese definieren, welche Elemente auf welche Weise parametrisch verändert werden sollen. Die Parameter können aber auch manuell verändert werden. [22]

Abschließend können durch Project Refinery (ein Optimierungswerkzeug) die Genauigkeit und die Berechnungszeit gesteuert werden. Nutzende können Designlösungen analysieren, wobei die Ergebnisse visuell dargestellt werden. [22]

Ergebnisse und Erkenntnisse:

SketchOpt ist eine nützliche Lösung für die Umwandlung handgezeichneter Skizzen, mit deren Hilfe durch Vektorisieren und Parametrisieren optimierbare Modelle entstehen. [22]

Besonders nützlich ist SketchOpt in frühen Designphasen, wo schnelle Anpassungen und Iterationen gefragt sind, wobei dieser generative Arbeitsablauf ebenso für unerfahrene Designerinnen und Designer verwendbar wird. [22]

Relevanz für die eigene Arbeit:

Einige Aspekte sind für die eigene Arbeit theoretisch von Relevanz. Keshavarzi et al. [22] haben das Ziel, eine parametrische graphenbasierte Zeichnung zu erstellen. Im Gegensatz dazu verfolgt die vorliegende Arbeit keinen graphenbasierten Ansatz, sondern einen parametrischen, da graphenbasierte Methoden auf einen spezifischen algorithmischen Weg führen, der für die Zielsetzung dieser Arbeit nicht geeignet erscheint. Die graphenbasierte Methode hat den Vorteil, dass sie bei nachträglicher Änderung das geometrische Layout beibehält. In meiner Arbeit soll die Parametrisierung der Zeichnung jedoch unabhängig von einer festen Topologie erfolgen, da dies eine größere Flexibilität ermöglicht.

Eine Idee aus diesem Beitrag ist jedoch, die Optimierung der parametrischen Zeichnung im Anschluss zu integrieren oder durch Anmerkungen in der Skizze spezifische Einschränkungen oder Funktionen zu definieren.

Eigene kritische Bewertung:

Die vorgestellte Methode ist eine innovative Lösung, die konventionelles Skizzieren und parametrisches Entwerfen verbindet. Besonders positiv hervorzuheben ist die nutzungsfreundliche Eigenschaft, die auch für Designerinnen und Designer ohne algorithmische Kenntnisse zugänglich ist. Allerdings wird in der Arbeit weniger dargestellt, wie das System mit komplexeren oder ungenauen Skizzen umgeht. Dennoch bietet diese generative Entwurfsmethode Potenzial für weitere Entwicklungen und Anwendungen.

2.3 Angewendete Technologien

Wenn es darum geht, sich einen Überblick über die derzeit verwendeten Technologien oder Erweiterungen hinsichtlich der CAD-Software Rhinoceros® [6] (Rhino) und Grasshopper®

[6] zu verschaffen, ist die Webseite Food4Rhino [23] die offizielle und umfangreichste Anlaufstelle. Sie ist eine Plattform, die als zentrale Veröffentlichungsstelle und Marktplatz für Plug-ins und Add-ons für Rhino und Grasshopper dient. Die Plattform wurde von Robert McNeel & Associates® [6], den Entwicklerinnen und Entwickler von Rhino, erstellt.

Erfahrungsgemäß werden die Inhalte auf dieser Seite von unterschiedlichen Gruppen bereitgestellt. Dazu gehören Software Entwicklerinnen und Entwickler, Forschungseinrichtungen und Universitäten sowie Designerinnen und Designer, Architektinnen / Architekten und Ingenieurinnen / Ingenieure. Die zuvor erwähnten Gruppen, aber auch Studentinnen und Studenten, nutzen die Inhalte dieser Plattform für akademische oder private Projekte. Die Plattform Food4Rhino [23] bietet eine Vielzahl von Ressourcen aus verschiedenen Bereichen wie Architektur, Schmuckdesign oder Möbeldesign. Sie ist eine wichtige Quelle für innovative Werkzeuge und Lösungen.

Nachfolgend werden passende Plugins dieser Plattform mit Bezug zu neuronalen Netzwerken und Klassifizierung recherchiert sowie die nützlichen Inhalte ausgewählt und beschrieben.

Plugin Crow

Crow integriert künstliche neuronale Netzwerke und unterstützt sowohl überwachte als auch unüberwachte Lernalgorithmen. Überwachte Algorithmen erhalten Datensätze mit dazugehörigen Bezeichnungen zum Lernen, unüberwachte Algorithmen hingegen Datensätze ohne solche Bezeichnungen. Die früheste Version (Version 0.2), die derzeit verfügbar ist, wurde 2016 veröffentlicht. [24]

Entwicklerinnen und Entwickler:

Der Entwickler von Crow und dem Plugin FlexHopper ist Benjamin Felbrich. [25] Nach seinem Architekturstudium an der Technischen Universität Dresden fokussierte er sich auf

Computational Design und Optimierung. Er arbeitete als Wissenschaftlicher Mitarbeiter an den Themen Computergrafik, Robotik und maschinelles Lernen. [25]

Externe Bibliotheken:

Crow basiert auf der Bibliothek NeuronDotNet.dll [26], die eine neuronale Netzwerk-Engine ist. Für fortgeschrittene KI-Programmierer bietet sie eine Basis, um künstliche neuronale Netzwerke zu entwerfen und zu verwenden. [26]

Funktionsumfang:

Crow integriert neuronale Netzwerke in Grasshopper durch rückwärtspropagierte Netzwerke (Backpropagation Networks) und selbstorganisierende Karten (Self-Organizing Maps). [24] Durch die Rückwärtspropagation werden bereits berechnete Werte korrigiert, was dazu führt, dass das neuronale Netzwerk durch Iterationen optimiert werden kann. Die Funktionspalette ist vergleichsweise einfach und klein gehalten. Daher ermöglicht sie eine schnelle und effektive Umsetzung der Lernalgorithmen.

Nutzungsfreundlichkeit und Einarbeitungszeit:

Nutzende mit grundlegenden Kenntnissen im maschinellen Lernen können mit relativ geringer Einarbeitungszeit die Werkzeuge verwenden, da es sich um eine leicht verständliche Erweiterung mit überschaubarer Anzahl von Komponenten handelt. Diese Gründe machen Crow besonders geeignet für diejenigen, die zwar Anfänger im maschinellen Lernen, aber erfahrene Grasshopper Anwenderinnen und Anwender sind.

Community und Dokumentation:

Eine Unterstützung durch die Community ist kaum gegeben, da Crow eine geringe

Verbreitung hat, die aus der Anzahl der Downloads (8 010) [24] und der Bewertungen (23) [24] auf der Seite Food4Rhino zu erkennen ist. Die Dokumentation ist auf eine grundlegende 2-minütige Videoanleitungen und wenige Beispiele beschränkt.

Weiterentwicklung:

Eine aktive Weiterentwicklung des Plugins wurde eingestellt. Die letzte Version wurde 2020 [27] veröffentlicht Seitdem scheint keine neue Funktionalität hinzugefügt worden zu sein.

Anwendungsfälle:

Crow eignet sich besonders für kleinere Projekte, in denen grundlegende neuronale Netzwerke zur Datenanalyse oder Klassifikation benötigt werden. Es ist nützlich für Lehr- und Lernzwecke sowie für einfache experimentelle Anwendungen.

Plugin LunchBox

LunchBox bietet eine Sammlung von verschiedenen Tools, unter anderem maschinelles Lernen für eine Klassifikation. Die früheste Version (Version 0.253), die derzeit verfügbar ist, wurde 2012 veröffentlicht. [28]

Entwicklerinnen und Entwickler:

Der Entwickler ist Nathan Miller, Gründer des Unternehmens Proving Ground [29], das mithilfe datenbasierter Methoden die digitale Transformation in der Bauindustrie vorantreibt. [30] Miller ist Professor an der Georgetown University McDonough School of Business, mit Forschungsschwerpunkten in den Bereichen Industrieökonomie und

Kartellrecht. Er ist zudem Redakteur und Associate Editor in renommierten Fachzeitschriften. [31]

Externe Bibliotheken:

LunchBox greift auf ML.NET and Accord.NET Frameworks zurück. ML.NET ist ein Machine Learning SDK (Software Development Kit) auf Open-Source-Basis und wird von Microsoft bereitgestellt. [32] Accord.NET ist ein Framework für maschinelles Lernen, kombiniert mit Audio- und Bildverarbeitungsbibliotheken. [33]

Funktionsumfang:

LunchBox bietet eine Sammlung von verschiedenen Tools für parametrische Flächen, Paneelierung, Datenmanagement und Datenworkflow. Außerdem beinhaltet sie eine Reihe von Komponenten für maschinelles Lernen wie Regressions-, Clustering- und Klassifizierungsalgorithmen. [28]

Nutzungsfreundlichkeit und Einarbeitungszeit:

LunchBox ist durch ihre klaren Komponenten leicht zu erlernen. Die umfangreiche Funktionalität erfordert jedoch eine längere Einarbeitungszeit, besonders für Nutzende, die weniger Erfahrung mit Grasshopper haben.

Community und Dokumentation:

LunchBox ist ein bekanntes Grasshopper-Plugin und wird von einer aktiven Community unterstützt. Die Anzahl der Downloads (992 591) [28] und Bewertungen (583) [28] weist ebenso darauf hin. Zudem gibt es viele Tutorials, Beispiele und Forenbeiträge, die den Einstieg erleichtern. [28] Die Dokumentation ist umfassend und gut strukturiert. [34]

Weiterentwicklung:

Updates erfolgen auf regelmäßiger Basis. Die neueste Version ist Ende 2024 erschienen. [28]

Anwendungsfälle:

LunchBox wird häufig von Studierenden und Architektinnen / Architekten für die Generierung komplexer Fassadenmuster sowie für die Erstellung von Gittern und Strukturen genutzt und als Schnittstelle zu Microsoft® Excel® [35] geschätzt.

Plugin Dodo

Dodo stellt eine überschaubare Anzahl von Werkzeugen für maschinelles Lernen, die Optimierung und Geometriemanipulation bereit. Die früheste Version (Version 0.1), die derzeit verfügbar ist, wurde 2015 veröffentlicht. [36]

Entwicklerinnen und Entwickler:

Der Entwickler ist Lorenzo Greco [37], der einen Masterabschluss in Bauingenieurwesen und Architektur besitzt. Er arbeitet als Forschungs- und Entwicklungsingenieur im Bauwesen sowie als Softwareingenieur. Seine Expertise liegt in den Bereichen geometrische und strukturelle Optimierung sowie rechnergestützte Architektur.

Externe Bibliotheken:

Dodo nutzt NLOptDotNet [38] und einige weitere Bibliotheken, die nicht benannt werden oder unzugänglich sind. NLOptDotNet ist eine Open-Source-Bibliothek für nichtlineare Optimierung und verschiedene andere Algorithmen.

Funktionsumfang:

Dodo bietet eine breite Palette an Funktionen. Im Bereich künstlicher Intelligenz bietet es neuronale Netzwerke, Gradientenabstieg und Schwarmoptimierung. Es besitzt Funktionen für die Manipulation von Skalar-, Vektor- und Tensorfelder. Einige Komponenten sind aus wissenschaftlichen Arbeiten implementiert. [36]

Nutzungsfreundlichkeit und Einarbeitungszeit:

Aufgrund der Vielfalt der Werkzeuge und der wissenschaftlichen Ausrichtung ist Dodo anspruchsvoller als die bisher genannten Plugins und kann daher herausfordernd zum Erlernen sein. Nutzende mit guten Vorkenntnissen in Optimierungsalgorithmen oder maschinellem Lernen profitieren jedoch von der Breite des Plugins.

Community und Dokumentation:

Die Community ist klein, was sich aus der Anzahl der Downloads (11 504) [36] und der Bewertungen (30) [36] ableiten lässt. Einige Beispiele liegen vor, die das Erlernen anschaulicher machen. Die Dokumentation ist umfangreich und strukturiert aufgebaut. [39]

Weiterentwicklung:

Obwohl das letzte Update 2019 [36] war, bleibt es durch seine vielseitige Funktionalität relevant.

Anwendungsfälle:

Die Anwendungsfälle lassen sich aus den Komponentengruppen ableiten. Dazu zählen das Arbeiten mit Graphen, die Analyse von polygonalen Netzen, die Generierung von Rauschmustern und die Vorhersage durch neuronale Netzwerke. [40]

3 Methodik

Um eine Antwort auf das Forschungsthema: „Parametrische 2D-Architekturzeichnungen durch Formerkennung von Skizzen mittels künstlicher Intelligenz und weiterer algorithmischer Methoden“ zu erhalten, wurde ein stufenweises Vorgehen gewählt, das folgende Schritte beinhaltet:

1. **KI-Modell**

   Testung vorhandener KI-Modelle

   Vorerst wurden die KI-Modelle von den Grasshopper-Plugins Crow [24] und LunchBox [28] getestet. Ein Hauptaugenmerk lag dabei auf der KI von Crow [24], die intensiver begutachtet wurde. Aufgrund unzureichender Informationen über die praktische Verwendung wurde beschlossen, ein eigenes Modell in der Programmiersprache Python zu schreiben (siehe 4.1.1).

   Auswahl eines passenden KI-Modells

   Ein passendes KI-Modell wurde auf Grundlage der benötigten Funktion ausgewählt. Da die Hauptaufgabe eine Klassifikation war, fiel die Wahl auf ein mehrlagiges Perzeptron (MLP, englisch: Multi-Layer Perceptron), eine Grundform neuronaler Netzwerke. Dieses Modell ist vielfältig einsetzbar und erschien daher passend für die benötigte Verwendung (siehe 4.1.2).

   Erstellung des KI-Modells

   Das ausgewählte Modell wurde in Python implementiert. Dabei wurde auf die Möglichkeit einer nutzungsspezifischen Anpassbarkeit Wert gelegt. Input- und Output-Parameter wurden, soweit es sinnvoll erschien, außerhalb der Python-

Komponente konfigurierbar bzw. einsehbar gemacht, um eine Veränderbarkeit zu ermöglichen (siehe 4.1.3).

2. Skizzendaten und parametrische Objektdaten

Erstellung einer Skizze

Die Verfahrensschritte und der Aufwand des Skizzierens wurden optimiert. Anstelle des klassischen handgezeichneten Skizzierens auf einem Blatt Papier wurde ein digitales Skizzieren implementiert, worin eine Computermaus oder ein Eingabestift verwendet wird, um direkt durch Grasshopper vektorisierte Linien und Kurven (typische CAD-Objekttypen) zeichnen zu können. Zwei unterschiedliche Skizziermethoden wurden dafür algorithmisch realisiert (siehe 4.2.1).

Erstellung der parametrischen Objekte

Um eine parametrische Zeichnung zu ermöglichen, werden parametrische Objekte benötigt, deren Parameter in Grasshopper veränderbar bleiben. Dazu wurde eine Reihe von Objekten erstellt, insbesondere 2D-Darstellungen von Einrichtungselementen und Raumaufteilungen für die Verwendung in einem Grundriss (siehe 4.2.2).

Aufbereitung der Skizzen und der parametrischen Objekte

Die Linien und Kurven der Skizze sowie die parametrischen Objekte mussten als Input für die KI aufbereitet werden. Da die KI nur numerische Daten (Ganzzahlen oder Fließkommazahlen) verarbeiten kann, wurde eine Methode gefunden, mit deren Hilfe die gezeichneten Figuren durch Listen von Zahlen beschrieben werden können. Hierfür wurden mehrere Ansätze implementiert (siehe 4.2.3).

Datenaugmentation:

Damit die KI einerseits bestmöglich klassifizieren kann und um andererseits auf unterschiedliche oder ungenaue Skizzen-Qualitäten der verschiedenen Nutzenden Rücksicht zu nehmen, wurde eine künstliche Vervielfältigung der Trainingsdaten durch Transformationen vorgenommen. Dieser Prozess, der als Datenaugmentation bezeichnet wird, soll zu leichten Variationen in den Trainingsdaten führen (siehe 4.2.4).

3. Integration

Training und Klassifizierung der KI:

Die KI wurde mit den erstellten variablen Daten trainiert. Dafür wurden nicht nur die Figur numerisch beschrieben, sondern auch ein Etikett (Label) für jede Figur oder Variante derselben Figur numerisch bezeichnet. So konnte auf Basis vielzähliger Varianten einer Figur eine Klassifizierung vorgenommen werden (siehe 4.3.1).

Auswahl des parametrischen Objektes:

Der Output der Klassifizierung ist eine Vorhersage, die beschreibt, welches Etikett am wahrscheinlichsten der getesteten Figur entspricht. Dieses Etikett wurde verwendet, um das entsprechende parametrische Objekt aus der Liste der verfügbaren Objekte auszuwählen (siehe 4.3.2).

Optimierung der KI:

Für die Optimierung der Hyperparameter wurde ein Fitness-Ziel beschrieben. Aufgrund dessen wurden schrittweise Kombinationen der Hyperparameter

(Parameter des neuronalen Netzwerks) getestet und mit dem Fitness-Wert aufgezeichnet. Diese Daten wurden etappenweise analysiert, um so eine optimale Auswahl der multiplen Hyperparameter zu erreichen (siehe 4.3.3).

4 Parametrische 2D-Zeichnungen aus Skizzen

4.1 KI-Modell

4.1.1 Testung vorhandener KI-Modelle

Was ist künstliche Intelligenz?

Eine vollständige und umfassende Definition existiert derzeit nicht, da der Begriff „Intelligenz“ schwer vollständig zu fassen ist. [41] [42] Allerdings kann erstere wie folgt beschrieben werden:

Künstliche Intelligenz bezeichnet die Fähigkeit einer Technologie oder eines Systems, eigenständig Fähigkeiten auszuüben und Aufgaben zu lösen, die menschlichen kognitiven Fähigkeiten ähneln und auf der Verwendung von Algorithmen basieren.

(Eigene Definition, inspiriert von B. J. Copeland [43])

Warum eine künstliche Intelligenz?

Eine KI wird in dieser Arbeit benötigt, weil ein effizienter und flexibler Umgang mit handgezeichneten (digitalen) Skizzen erforderlich ist. Skizzen können starke Unterschiede aufweisen. Sie variieren nicht nur von Person zu Person, sondern auch zwischen einer Skizze und der anderen Skizze derselben Person. Sie müssen aber einheitlich in eine parametrische Form umgewandelt werden können. Der gewählte Ansatz war daher der Einsatz einer KI, die in der Lage ist, unterschiedliche Skizzen zuverlässig einheitlich zu klassifizieren. Dies ermöglicht eine umfassende Verarbeitung, die mit anderen Algorithmen schwer zu erreichen wäre.

Vorhandene KI-Modelle

Zu Beginn der Erarbeitung des Algorithmus bzw. der Algorithmen für diese Arbeit wurden

die vorhandenen KI-Komponenten der Plugins Crow [24] und LunchBox [28] getestet.

Das Plugin Crow [24] besitzt einige Komponenten für ein Backpropagation-Netzwerk. Als Einstieg in die Nutzung von KI-Komponenten wurde ein einfaches Ziel angestrebt: die Klassifikation handgezeichneter Zahlen von 0 bis 9 mithilfe der vorhandenen Komponenten.

Da der Input ausschließlich numerisch sein musste, war es erforderlich, einen Algorithmus zur Umwandlung geometrischer Kurven und Linien zu implementieren. Dazu wurde jede Figur (eine gezeichnete Zahl) in Punkte mit regelmäßigen Abständen unterteilt und der Abstand dieser Punkte zum Mittelpunkt der Figur berechnet. So entstand für jede Figur eine Liste numerischer Werte, die als Eingabe für das Training geeignet war. Zusätzlich musste jeder Figur ein numerisches Etikett zugewiesen werden, um die Trainingsdaten zu vervollständigen.

Das Setup in Grasshopper wurde so eingerichtet, dass bei der Klassifizierung das entsprechende Etikett gewählt wird. Dadurch wurde die klassifizierte Figur über der getesteten Figur in einer anderen Farbe dargestellt, was eine einfache visuelle Überprüfung der Klassifikation ermöglichte.

Ferner wurden verschiedene Optionen zur Struktur des neuronalen Netzes untersucht, darunter die Anzahl der Schichten, die Anzahl der Neuronen und die Lernrate. Dabei wurde der Mean Square Error (MSE) beobachtet, der vereinfacht als durchschnittlicher Fehler des Netzwerks verstanden werden kann. Das Ziel war, die Netzstruktur so zu verbessern, dass der MSE so gering wie möglich bleibt.

Nach einigen Versuchen wurden geeignete Einstellungen gefunden. Allerdings war unklar, warum genau diese Konfigurationen die besten Ergebnisse lieferten. Erst nach weiterer Recherche zu den Auswirkungen der einzelnen Parameter auf die Berechnungen der KI wurde deutlich, dass ein umfassendes Verständnis dieser Einstellungen nur möglich ist, wenn der gesamte Code der implementierten KI bekannt ist und verstanden wird. Ähnliche

Versuche wurden mit dem Plugin LunchBox [28] durchgeführt. Da jedoch die gleichen Hindernisse wie bei Crow [24] auftraten, wurden diese nicht weiterverfolgt.

Schließlich wurde deutlich, dass für ein besseres Verständnis und einen besseren Umgang mit KI der zugrunde liegende Code des implementierten Modells verstanden werden muss. Da jedoch kein Zugriff auf den Code der verwendeten Komponenten bestand, wurde beschlossen, eine eigene KI zu implementieren.

4.1.2 Auswahl eines passenden KI-Modells

Gliederung der künstlichen Intelligenz

Hinsichtlich der KI-Modelle gibt es verschiedene Möglichkeiten, von denen eine geeignete ausgewählt werden musste. Dazu folgt nun ein spezifischer Überblick (siehe Abbildung 2).

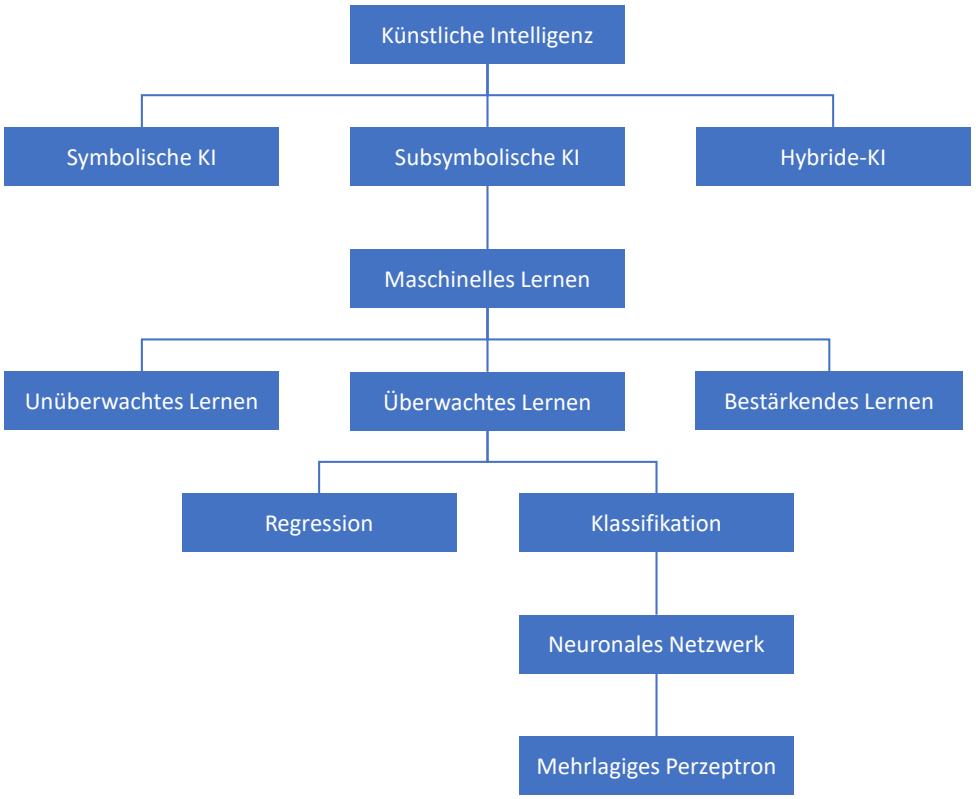


Abbildung 2: Spezifischer Überblick der Gliederung der künstlichen Intelligenz (eigene Darstellung nach [44] [45])

Künstliche Intelligenz kann in drei Kategorien unterteilt werden: 1) Symbolische KI, 2) Subsymbolische KI und 3) Hybride-KI, die eine Kombination aus symbolischer und subsymbolischer KI darstellt. [44]

1) Symbolische KI basiert auf Logik- und Wissenssystemen. Sie verwendet explizite vordefinierte Regeln (Wenn…, dann…) und beschränktes Expertenwissen, um Probleme zu lösen. Die Bezeichnung „symbolisch“ stammt von der symbolischen Logik, die auf Buchstaben und logischen Operatoren (z. B.: ¬ für Negation, ∧ für Konjunktion) basiert. [44]

2) Subsymbolische KI basiert auf Sensorsystemen. Sie verwendet sensorielle numerische Daten, die z. B. visuelle oder akustische Daten umfassen, aus denen statistische Korrelationen und Wahrscheinlichkeiten bestimmt werden. Die Bezeichnung „subsymbolisch“ kommt daher, dass Lernprozesse aus sensoriellen Daten in der Wahrnehmung des Menschen unbewusst im Hintergrund des bewussten logischen Schlussfolgerns ablaufen. [44]

Subsymbolische KI umfasst unter anderem Maschinelles Lernen (ML), das wiederum in verschiedene Arten unterteilt werden kann: 2a) überwachtes Lernen (supervised learning), 2b) unüberwachtes Lernen (unsupervised learning) und 2c) bestärkendes Lernen (reinforcement learning) (siehe Abbildung 3). [45]

2a) Im überwachten Lernen bekommt die KI Datensätze mit dazugehörigen gewünschten Bezeichnungen und kann so den Fehler in der Berechnung „überwachen“/korrigieren, sodass für einen Test-Datensatz die richtige Bezeichnung angenommen werden kann. Die Ausgabe ist eine Zuordnung in Form einer Wahrscheinlichkeit, die auch Vorhersage genannt wird. [45]

2b) Beim unüberwachten Lernen enthalten die Datensätze keine dazugehörigen Bezeichnungen. Im Zuge dessen gibt es keine Korrekturmöglichkeit. Der Datensatz

wird anhand von gemeinsamen Merkmalen segmentiert. Die Ausgabe erfolgt in Klassen (Gruppen), die typische Muster im Datensatz repräsentieren. [45]

2c) Beim bestärkenden Lernen bekommt die KI Datensätze von Zuständen und dazugehörigen Aktionen und soll durch Belohnung und/oder Bestrafung aus wiederholten Zuständen lernen, eine passende Aktion auszuführen. [45]

Überwachtes Lernen kann in zwei Untergruppen eingeteilt werden. Eine ist die Regression, die andere die Klassifikation. Beide haben das Ziel, einen Zusammenhang zwischen Variablen zu finden, allerdings mit dem Unterschied, dass bei einer Regression eine Zahl und bei einer Klassifikation eine Klasse vorhergesagt werden soll. [45]

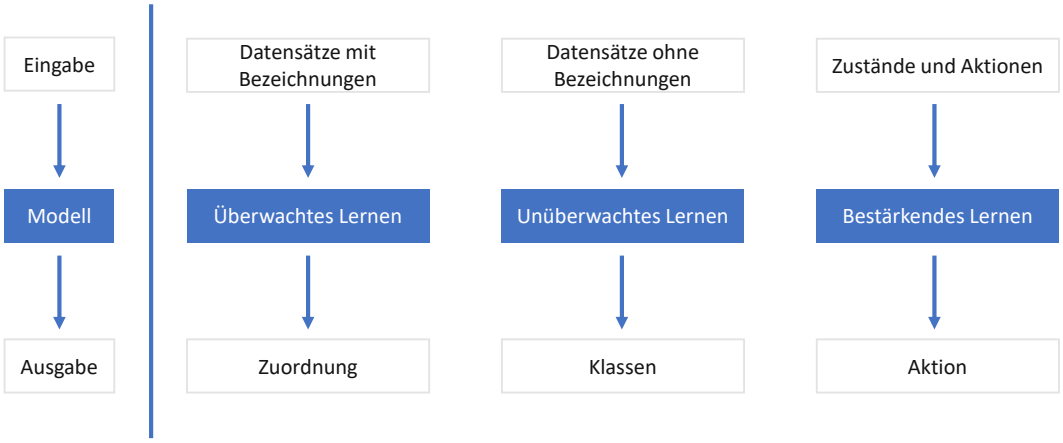


Abbildung 3: Arten des Maschinellen Lernens mit Eingabe und Ausgabe (eigene Darstellung nach [45])

Auswahl der künstlichen Intelligenz

Im Fall dieser Arbeit wird eine Skizze durch CAD-Objekttypen wie Linien und Kurven erstellt, die dann numerisch beschrieben werden. Relevant sind also numerische Daten, die mit dazugehörigen Bezeichnungen versehen werden. Da die Daten bereits bezeichnet sind und eine Klassifikation erforderlich ist, fällt diese Aufgabe unter das überwachte Lernen.

Die Wahl fiel daher auf ein neuronales Netzwerk, das für eine Klassifikation benutzt wird. [46] Konkret wurde die Grundform eines neuronalen Netzwerks, das sogenannte mehrlagige Perzeptron (Multi-Layer Perceptron, MLP), eingesetzt. [47]

Würde die Skizze hingegen als Bild verwendet werden, würde die Wahl für die Klassifikation beispielsweise auf ein faltungsbasiertes neuronales Netzwerk fallen. [48]

4.1.3 Erstellung des KI-Modells

Grundstruktur des neuronalen Netzwerks

Ein mehrlagiges Perzeptron (MLP) ist ein neuronales Netzwerk, das aus mehreren Neuronen und deren Verbindungen besteht. Gemeinsam bilden sie ein Konstrukt, das zur konzeptuellen Darstellung eines neuronalen Netzwerks verwendet wird. Diese Neuronen sind grundsätzlich in drei Schichten angeordnet (siehe Abbildung 4): [47]

- Eingabeschicht (Input Layer),
- Verborgene Schichten (Hidden Layers),
- Ausgabeschicht (Output Layer).

Die Eingabeschicht und die Ausgabeschicht bestehen jeweils immer aus einer einzigen Schicht, während die verborgenen Schichten aus einer oder mehreren Schichten bestehen können. [47]

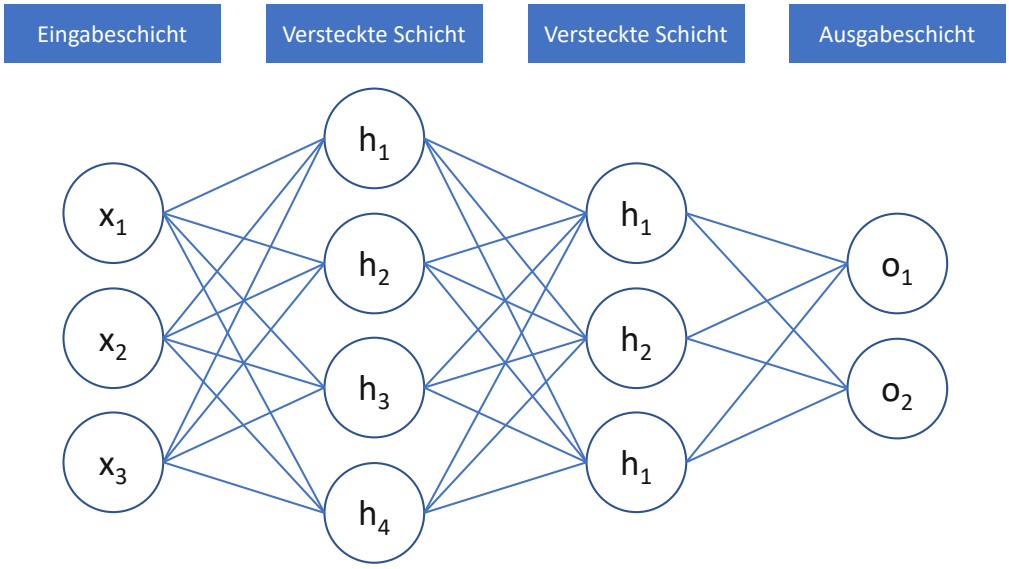


Abbildung 4: Schichten eines neuronalen Netzwerks (eigene Darstellung)

Die Neuronen einer Schicht sind vollständig mit den Neuronen der nächsten Schicht verbunden. Das bedeutet, dass jedes Neuron seine Werte nur an die Neuronen der nächsten Schicht weitergeben kann. Die Verbindungen zwischen den Neuronen werden durch Gewichte (w) repräsentiert (siehe Abbildung 5). Jedes Neuron erhält zusätzlich einen Schwellenwert (Bias-Wert, b), der für eine Verschiebung (Erhöhung oder Erniedrigung) des Wertes des Neurons dient. [47]

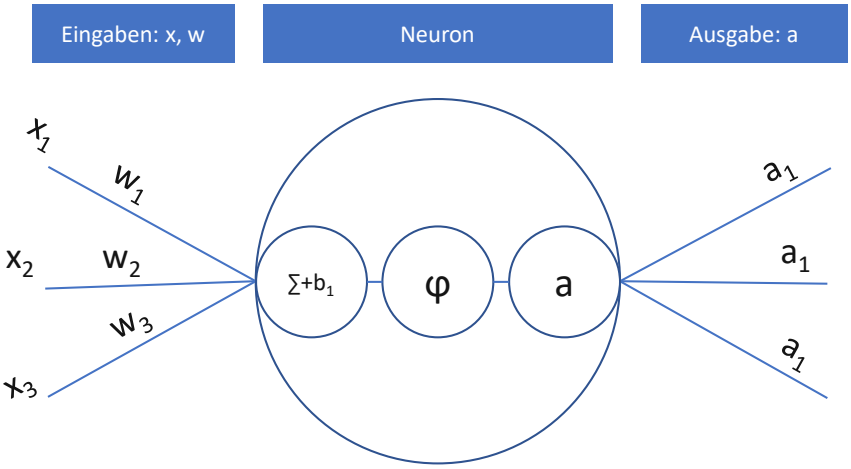


Abbildung 5: Neuron einer Versteckten Schicht mit Eingaben und Ausgabe (eigene Darstellung)

Grundfunktion des neuronalen Netzwerks

Die Grundfunktion eines neuronalen Netzes basiert auf einem iterativen Optimierungsprozess, der sich aus Vorwärtspropagation, Fehlermessung und Rückwärtspropagation zusammensetzt.

Vorerst erfolgt die Vorwärtspropagation, bei der die Eingabe-Neuronen (x_i) die numerischen Werte der Eingabedaten erhalten und an die nächste Schicht weiterleiten. Die versteckten Neuronen (h_i) erhalten diese Eingabewerte, multiplizieren sie mit den entsprechenden Gewichten (w_i), addieren Bias-Werte (b_i) und führen anschließend eine Aktivierungsfunktion (φ) aus, die darüber entscheidet, in welchem Umfang die Werte ausgegeben werden (a_i) (siehe Abbildung 5). Die Aktivierungsfunktion ist notwendig, um eine nichtlineare Transformation zu ermöglichen, da viele Klassifikationsprobleme nicht linear lösbar sind. Dadurch kann das neuronale Netz komplexe (nicht-lineare) Zusammenhänge erstellen.

Dieser Prozess setzt sich bis zu den Ausgabe-Neuronen (o_i) fort, wo die berechneten Werte die aktuelle Ausgabe des Netzwerks darstellen. Für Klassifikationsaufgaben müssen mindestens zwei Neuronen in der Ausgabeschicht vorhanden sein, wobei diese von der Anzahl der unterschiedlichen Klassen abhängen.

Nach der Vorwärtspropagation folgt die Fehlermessung, bei der die aktuell berechnete Ausgabe des Netzwerks mit den erwarteten Sollwerten verglichen wird, die durch Etiketten bereitgestellt werden. Die Abweichung zwischen diesen Werten wird durch eine Verlustfunktion (Loss-Function) berechnet, die als Maß für den aktuellen Gesamtfehler des Netzwerks dient.

Anschließend erfolgt die Rückwärtspropagation mit dem Ziel, diesen Fehler zu minimieren. Der Fehler wird rückwärts durch das Netzwerk propagiert, sodass die Gewichte und Bias-Werte schrittweise korrigiert werden. Die Lernrate bestimmt dabei, in welchem Umfang diese Korrekturen vorgenommen werden.

Durch die iterative Wiederholung der drei Schritte passt sich das neuronale Netzwerk immer besser an die gegebenen Daten (Sollwerte) an und optimiert seine Ausgabe. Dieser Trainingsprozess wird so lange wiederholt, bis eine gewünschte Genauigkeit erreicht oder eine vordefinierte maximale Anzahl an Iterationen durchgeführt wurde.

Der Code für das Training des neuronales Netzwerks wurde in der Python-Komponente in Grasshopper geschrieben. Nützliche Parameter wurden außerhalb zugänglich gemacht (siehe Abbildung 6). Der darin enthaltene Code ist im Anhang dargestellt (siehe 8.1), auf dessen Inhalt nun näher eingegangen wird.

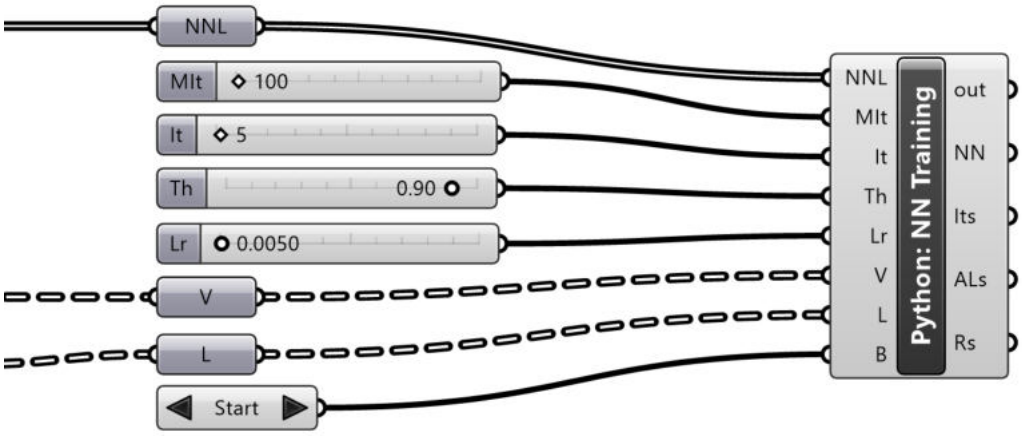


Abbildung 6: Python-Grasshopper-Komponente mit Code für das Training des neuronalen Netzwerks (eigene Darstellung)

Erläuterung des Codes für das Training

Zeilen 001–031

Zu Beginn des Codes wird darauf hingewiesen, dass der Code urheberrechtlich geschützt ist. Danach werden die Eingaben und Ausgaben der Python-Komponente in Grasshopper definiert und kurz beschrieben. Außerdem werden die Rhino- und Grasshopper-Version, in der der Code geschrieben wurde, angegeben und der Name des Autors sowie das Datum der Erstellung der Komponente angeführt.

Zeilen 032–104

Darauf folgt der Import von benötigten Bibliotheken: *math* für mathematische Funktionen, *random* für Zufallszahlen und *Grasshopper.Kernel.Data.GH_Path* für den Zugriff auf Grasshopper-Baumstrukturen. Eine GH-Baumstruktur besteht aus mehreren Listen, die durch Pfadangaben hierarchisch organisiert sind und Daten enthalten. Dabei ist anzumerken, dass bewusst auf den Einsatz gängiger externer Bibliotheken wie *TensorFlow, Keras, PyTorch, Scikit-learn oder NumPy* verzichtet wurde. Stattdessen wurde der Code ausschließlich mit den nativen Bibliotheken der Grasshopper-Python-Komponente realisiert.

Anschließend werden die Eingabedaten der Python-Komponente verarbeitet. Dazu gehören die Anzahl der Neuronen aller Schichten sowie die Anzahl der Neuronen in der Ausgabeschicht, die aus dem Layout des neuronalen Netzwerks extrahiert werden.

Daraufhin erfolgt die Vorbereitung weiterer Eingabedaten, darunter die Werte der Eingabeneuronen, die Etiketten (Labels) sowie die Zielwerte für das Training. Da die Eingaben (V und L) als Grasshopper-Bäume vorliegen, müssen sie in Python-Bäume, als Listen innerhalb von Listen, umgewandelt werden. Diese Listenstruktur ist maßgeblich für den gesamten Code des neuronalen Netzwerks, da anstelle von Matrizen bewusst mit verschachtelten Listen gearbeitet wurde. Das ist damit zu begründen, dass einerseits die gewohnte Handhabung des Listenmanagements angewendet werden kann und andererseits die Umschreibung der Daten angepasst an die Struktur der Grasshopper-Bäume erfolgen kann.

Für jede Klasse wird ein eigener One-Hot-Vektor (One Hot Encoding) erstellt, der aus n-Elementen besteht und dessen Elemente ausschließlich die Werte 0 und 1 annehmen. Bei drei Klassen, beispielsweise Waschbecken, Schreibtisch, und Toilette, würden die Vektoren folgendermaßen aussehen: [1, 0, 0], [0, 1, 0], [0, 0, 1]. [49]

Diese Darstellung sorgt für vereinfachte einheitliche Etiketten, die als Zielwerte für die Optimierung des neuronalen Netzwerks verwendet werden können.

Zeilen 105–149

Hier werden die Gewichte und Bias-Werte für das neuronale Netzwerk erstellt, basierend auf der Anzahl der Neuronen in den einzelnen Schichten.

Die Gewichte werden mithilfe der Xavier-Initialisierung (siehe Formel 1) erstellt. Dies bedeutet, das Zahlen zufällig innerhalb einer gleichmäßigen Domaine [-a, +a] gewählt werden, die von der Anzahl der Eingabeneuronen und der Ausgabeneuronen abhängig ist. [50] [51]

$$\mathrm{D} = \left[-\sqrt{\frac{6}{\mathrm{n_{in}}+\mathrm{n_{out}}}}, \sqrt{\frac{6}{\mathrm{n_{in}}+\mathrm{n_{out}}}}\right]$$

Formel 1: Xavier-Initialisierung

Dabei gilt:

D = Domaine,

n_{in} = Anzahl der Eingabeneuronen,

n_{out} = Anzahl der Ausgabeneuronen.

Die Bias-Werte werden hingegen alle auf 0 gesetzt, um eine einheitliche Ausgangslage zu gewährleisten und sicherzustellen, dass die Aktivierungen ohne Verschiebung der Werte durch die Eingaben und Gewichte bestimmt werden.

Zeilen 150–208

Nun folgen Implementierungen verschiedener gängiger Aktivierungsfunktionen und deren Ableitungen, darunter Sigmoid, ReLU (Rectified Linear Unit), Leaky ReLU (LReLU) und Softmax (siehe Formel 2 bis Formel 5 sowie Abbildung 7 und Abbildung 8). [52] [53]

$$f(x) = \frac{1}{1 + e^{-x}}$$

Formel 2: Sigmoid

$$f(x) = \max(0, x)$$

Formel 3:ReLU

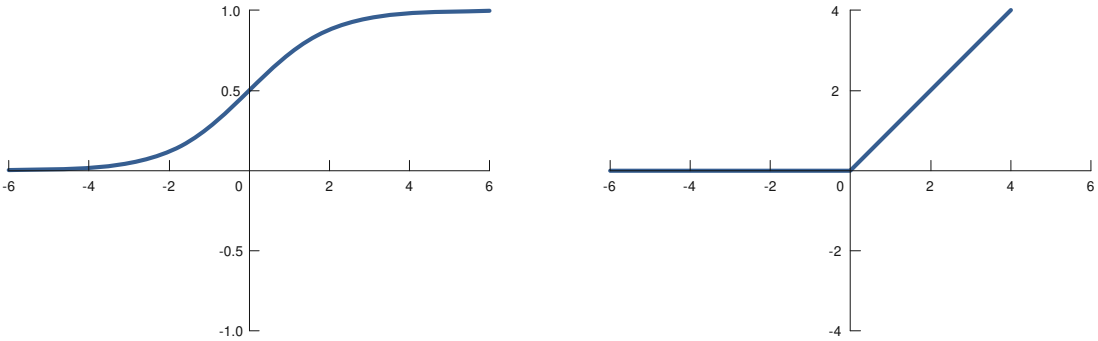


Abbildung 7: Graphische Darstellung der Aktivierungsfunktionen Sigmoid und ReLU (eigene Darstellung)

Dabei gilt:

$f(\cdot)$ = Der aktivierte Wert,

x = Eingabewert,

e = Eulersche Zahl (ca. 2,718),

$\max(0, x)$ = Nimmt den größeren Wert von 0 und x.

$$f(x) = \begin{cases} x, & if\ x > 0 \\ \alpha x, & if\ x \leq 0 \end{cases}$$

Formel 4: LReLU

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Formel 5: Softmax

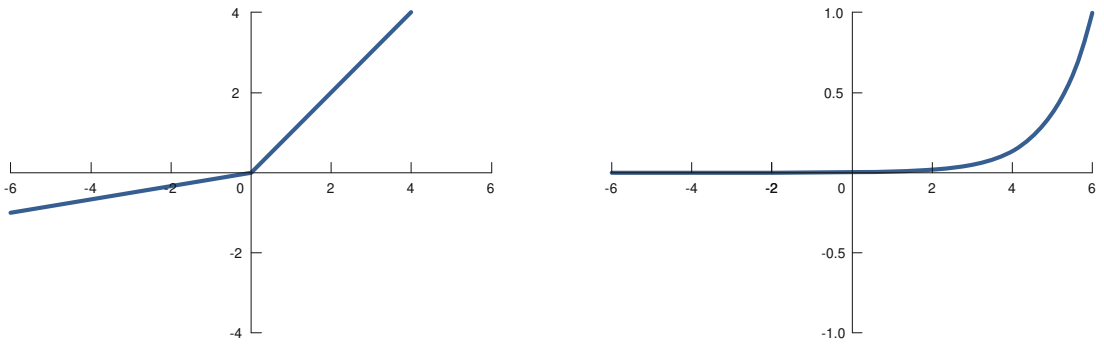


Abbildung 8: Graphische Darstellung der Aktivierungsfunktionen LReLU und Softmax (eigene Darstellung)

Dabei gilt:

f(·) = Der aktivierte Wert,

x = Eingabewert,

e = Eulersche Zahl (ca. 2,718).

α = Faktor (ca. 0,1–0,01),

x_i = Eingabe x der Klasse i,

x_j = Eingabe x aller Klassen j,

j = Alle Klassen,

$\sum_j$! = Summe der Werte aller Klassen (zur Normalisierung).

Für die versteckten Schichten wird eine dieser Aktivierungsfunktionen gewählt. Die Softmax-Aktivierungsfunktion wird speziell für die Ausgabeschicht implementiert, da sie notwendig ist, um die Werte in Wahrscheinlichkeiten umzuwandeln. Dabei beträgt die Summe aller Wahrscheinlichkeiten immer 1, was eine notwendige Voraussetzung für die Wahrscheinlichkeitsverteilung von Klassifikationsaufgaben ist. [53]

Zeilen 209–228

Hier werden die Kreuzentropie-Verlustfunktion (Cross-Entropy Loss; siehe Formel 6) sowie ihre Ableitung implementiert. Diese Funktion berechnet den Fehler des Netzwerks, indem sie die vorhergesagten Wahrscheinlichkeiten mit den tatsächlichen Zielwerten vergleicht. Eine höhere Abweichung führt zu einem höheren Fehlerwert, was während des Trainings als Grundlage für die Korrektur der Gewichte und Bias-Werte dient. Der Verlustwert wird als Summe aller Klassen berechnet: [54]

$$\mathrm{L} = -\sum y \cdot \log(\hat{y})$$

Formel 6: Kreuzentropie-Verlustfunktion

Dabei gilt:

L = Verlustwert,

$\log(\cdot)$ = Logarithmus,

y = Tatsächlicher Zielwert (Zielvektor) einer Klasse,

$\hat{y}$ = Vorhergesagte Wahrscheinlichkeit einer Klasse.

Zeilen 229–275

Die Vorwärtspropagierung (Forward Propagation) wird nachfolgend implementiert. Hierbei werden die Eingaben durch das neuronale Netzwerk vorwärts geleitet. Dabei erhält jede Neuronenschicht ihre Eingaben aus der vorherigen Schicht. Zunächst werden die Eingaben mit den zugehörigen Gewichten multipliziert und anschließend wird ein Bias-Wert addiert. Danach wird eine Aktivierungsfunktion angewendet, um Nichtlinearitäten einzuführen und das Modell in die Lage zu versetzen, komplexe Korrelationen zu lernen. Dieser Vorgang wiederholt sich für jede Schicht, bis zu der Ausgabeschicht. Dort wird die Softmax-Funktion verwendet, um die Ergebnisse in Wahrscheinlichkeiten umzuwandeln, sodass sie als Klassifikationsergebnisse interpretiert werden können. [47]

Die Berechnung der Ausgabe eines Neurons kann mathematisch durch folgende Formel dargestellt werden (siehe Formel 7 und Abbildung 5): [47]

$$a_j = f\left(\left(\sum w_{ji} \cdot a_i\right) + b_j\right)$$

Formel 7: Ausgabe eines Neurons

Dabei gilt:

a_j = Aktivierung des Neurons j der aktuellen Schicht (Ausgabe des Neurons),

w_{ji} = Gewicht zwischen Neuron i der vorherigen Schicht zum Neuron j der aktuellen Schicht,

a_i = Aktivierung des Neurons i der vorherigen Schicht,

b_j = Bias-Wert des Neurons j der aktuellen Schicht,

$\sum w_{ji} \cdot a_i$ = Gewichtete Summe aller Eingaben,

$f(\cdot)$ = Aktivierungsfunktion, die auf das Gesamtergebnis angewendet wird.

Zeilen 276–359

Hier wird die Rückwärtspropagierung (Backpropagation) implementiert. Dieser Schritt dient dazu, die Fehler aus der Vorwärtspropagierung zurück durch das Netzwerk zu leiten. Dieser Korrekturprozess beginnt in der Ausgabeschicht und setzt sich Schicht für Schicht rückwärts bis zur ersten versteckten Schicht fort. [47]

Nun erfolgt die Aktualisierung der Gewichte und Bias-Werte, die durch den Fehler des Netzwerks entstanden ist. Die Berechnung erfolgt nach folgenden Formeln (siehe Formel 8 und Formel 9) : [47]

$$w_{ji} = w_{ji} - \eta \cdot \frac{\partial L}{\partial w_{ji}}$$

Formel 8: Aktualisierung der Gewichte

$$b_j = b_j - \eta \cdot \frac{\partial L}{\partial b_j}$$

Formel 9: Aktualisierung der Bias-Werte

Dabei gilt:

w_{ji} = Gewicht, das aktualisiert wird (Gewicht zwischen Neuron i der vorherigen Schicht zum Neuron j der aktuellen Schicht),

η = Lernrate, die bestimmt, wie stark die Anpassung erfolgt,

$\frac{\partial L}{\partial w_{ji}}$ = Gradient der Verlustfunktion L zu dem Gewicht w_{ji} gibt an, wie stark das Gewicht zum Gesamtfehler beiträgt,

b_j = Bias-Wert, der aktualisiert wird (Bias-Wert j der aktuellen Schicht),

$\frac{\partial L}{\partial b_j}$ = Gradient der Verlustfunktion L zu dem Bias-Wert b_j gibt an, wie stark der Bias-Wert zum Gesamtfehler beiträgt.

Zeilen 360–388

Zu Beginn der Trainingsschleife werden die bereits implementierten Funktionen zur Datenvorbereitung, Gewichtserstellung und Initialisierung der Variablen aufgerufen. Anschließend wird festgelegt, ob das Training entweder auf einer festen Anzahl von Iterationen (It) oder einem Fehlerschwellenwert (Th) basiert. Die Anzahl der gespeicherten Ergebnisse wurde begrenzt, um eine übermäßige RAM-Nutzung zu verringern.

Zeilen 389–455

Die Trainingsschleife startet, indem das neuronale Netzwerk iterativ trainiert wird. Zuerst erfolgt die Vorwärtspropagierung, bei der die Ausgaben für eine Eingabe berechnet

werden. Dann wird die Fehlermessung durchgeführt, indem die Kreuzentropie-Verlustfunktion verwendet wird. Danach folgt die Rückwärtspropagierung, um die Gewichte und Bias-Werte entsprechend anzupassen. Falls eine Fehlerschwelle (Th) definiert wurde und erreicht wird, bricht die Schleife vorzeitig ab, um unnötige Berechnungen zu vermeiden. Sonst laufen die Iterationen bis zu der festgesetzten Maximalanzahl (MIt).

Am Ende des Codes wird das trainierte neuronale Netzwerk (NN) mit seinen Gewichten, Bias-Werten, Layout-Informationen, Fehlerwerten und Ergebnissen ausgegeben, sodass es für Vorhersagen in Grasshopper genutzt werden kann.

Der Code für die Vorhersage des neuronales Netzwerks wurde in der Python-Komponente in Grasshopper geschrieben. Nützliche Parameter wurden außerhalb zugänglich gemacht (siehe Abbildung 9). Der darin enthaltene Code ist im Anhang dargestellt (siehe 8.1), auf dessen Inhalt nun näher eingegangen wird.

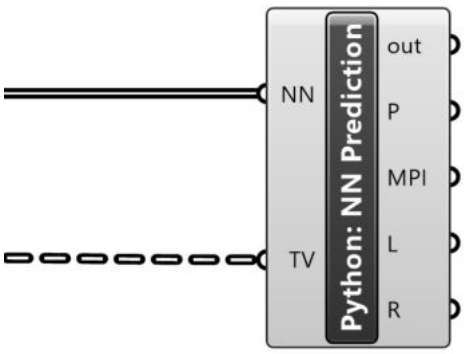


Abbildung 9: Python-Grasshopper-Komponente mit Code für die Vorhersage des neuronalen Netzwerks (eigene Darstellung)

Erläuterung des Codes für die Vorhersage

Zeilen 001–025

Der Beginn entspricht dem Aufbau des Trainingscodes. Angeführt werden der

urheberrechtliche Schutz, die Eingaben und Ausgaben der Python-Komponente in Grasshopper, die verwendeten Rhino- und Grasshopper-Versionen, der Name des Autors sowie das Datum der Erstellung der Komponente.

Zeilen 026–058

Als Nächstes folgt der Import der Bibliothek *math*, die für Berechnungen benötigt wird. Danach werden, wie bereits im Trainingscode beschrieben, verschiedene Aktivierungsfunktionen wie Sigmoid, Leaky ReLU und Softmax definiert (siehe Seite 52).

Zeilen 059–113

Die Vorwärtspropagierung im neuronalen Netz erfolgt nach demselben Aufbau wie im Trainingscode.

Zeilen 114–133

In diesem Abschnitt werden die notwendigen Daten für die Vorhersage importiert.

Zunächst werden die Parameter aus dem trainierten neuronalen Netz (NN) extrahiert, darunter die Gewichte, Bias-Werte und Layer-Struktur. Anschließend wird die Vorwärtspropagierung ausgeführt, indem die Testwerte (TV) als Eingabe in das neuronale Netz eingesetzt werden. Die Berechnungen müssen auf denselben Rechenvorgängen und Strukturen basieren wie im Training, um eine konsistente und sinnvolle Nutzung der trainierten Modelle zu gewährleisten. Die berechnete Ausgabe des Netzwerks wird als Vorhersage (P) gespeichert.

Der Wert mit der höchsten Wahrscheinlichkeit (Maximum Prediction) gibt an, welche Klasse auf Grundlage des aktuellen Trainings am wahrscheinlichsten den Testwerten entspricht. Zudem wird der MPI (Maximum Prediction Index), also der Index der höchsten

Wahrscheinlichkeit, ausgegeben, sodass das entsprechende Etikett ausgewählt und weitere Verarbeitungsschritte ermöglicht werden.

Dieser Prozess führt schließlich dazu, dass auf Basis von Testwerten und dem trainierten Netzwerk eine Klassifizierung durchgeführt werden kann.

4.2 Skizzendaten und parametrische Objektdaten

4.2.1 Erstellung einer Skizze

Der ursprüngliche Weg, eine Skizze zu erstellen, besteht darin, mit einem Stift auf ein Blatt Papier zu zeichnen. Diese Skizze müsste für die Verwendung mit der KI anschließend fotografiert oder eingescannt werden. Danach würde sie in Grasshopper importiert und vektorisiert werden, sodass aus einer Pixelgrafik Linien und Kurven entstehen. Dieser Prozess ist für den Arbeitsablauf jedoch zeitaufwendig und umständlich. Daher entstand die Idee des digitalen Skizzierens, bei dem eine Computermaus oder ein Eingabestift verwendet wird, um direkt in Grasshopper vektorisierte Linien und Kurven – typische CAD-Objekttypen – zu zeichnen. Dafür wurden zwei unterschiedliche Skizziermethoden algorithmisch realisiert (siehe Abbildung 10).

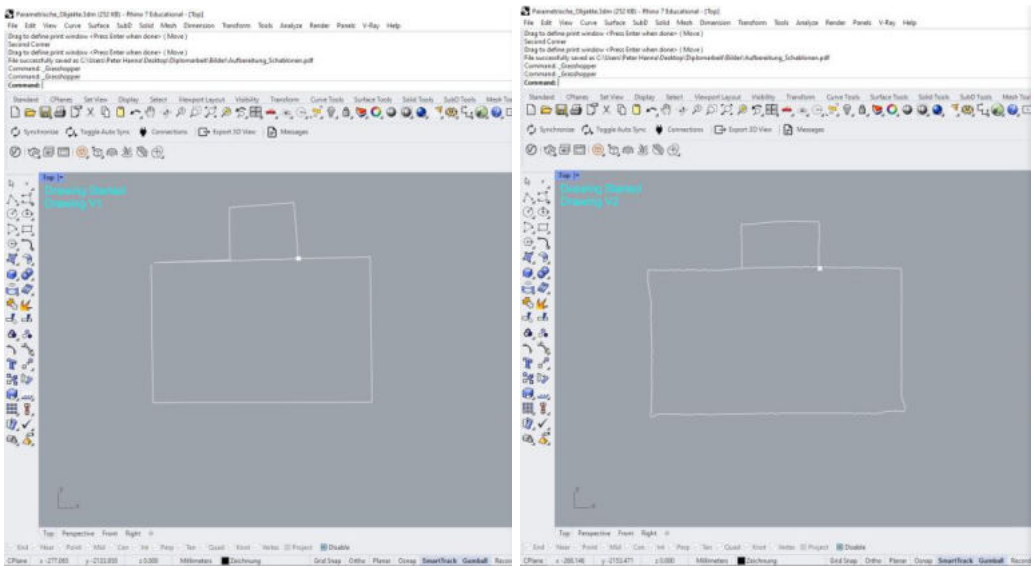


Abbildung 10: Ergebnis der Skizziermethode 1 (links), Ergebnis der Skizziermethode 2 (rechts) (eigene Darstellung)

Zunächst wird die gewollte Skizziermethode durch einen Regler (Slider) ausgewählt. Anschließend wird der Algorithmus mit einem Boolean aktiviert. Da es sich um 2D-Skizzen handelt, erfolgt das Zeichnen in der Draufsicht in Rhino.

Skizziermethode 1

In der ersten Methode wird gezeichnet, indem die Computermaus per Linksklick im grafischen Bereich von Rhino betätigt wird. Dadurch werden einzeln Punkte gesetzt, die automatisch als Polylinie miteinander verbunden werden. Dabei besteht immer eine Verbindung vom letzten Punkt zur aktuellen Mausposition, was bedeutet, dass nicht von der Skizze abgesetzt werden kann.

Alternativ kann die Maus gedrückt gehalten werden, wodurch in einem einstellbaren Zeitintervall (z. B. alle 100 Millisekunden) automatisch Punkte an der Mausposition gesetzt werden. Beide Möglichkeiten können kombiniert genutzt werden.

Skizziermethode 2

In der zweiten Methode muss die Maustaste durchgehend gedrückt gehalten werden, um zu zeichnen. Dabei werden ebenfalls in einem einstellbaren regelmäßigen Zeitintervall Punkte an der Mausposition gesetzt. Im Gegensatz zur ersten Methode wird keine permanente Verbindung zur Mausposition hergestellt, was bedeutet, dass von der Skizze abgesetzt werden kann.

In beiden Skizziermethoden ist es möglich, durch das Drücken der Taste „D“ auf der Tastatur die Skizzen zu löschen. Mit der Taste „F“ werden Skizzen fertiggestellt, wodurch die skizzierten Figuren automatisch in einem neuen Kurven-Kontainer (Curve Container) gespeichert werden. Diese Kurven-Komponente wird ebenfalls automatisch mit einer Entwine-Komponente verbunden und an den ersten freien Input angeschlossen. Dadurch wird jede abgefertigte Skizze in einer eigenständigen Liste separiert, da eine Entwine-Komponente die Funktion besitzt, getrennte Eingaben (Inputs) in separate Listen zu setzen (siehe Abbildung 12).

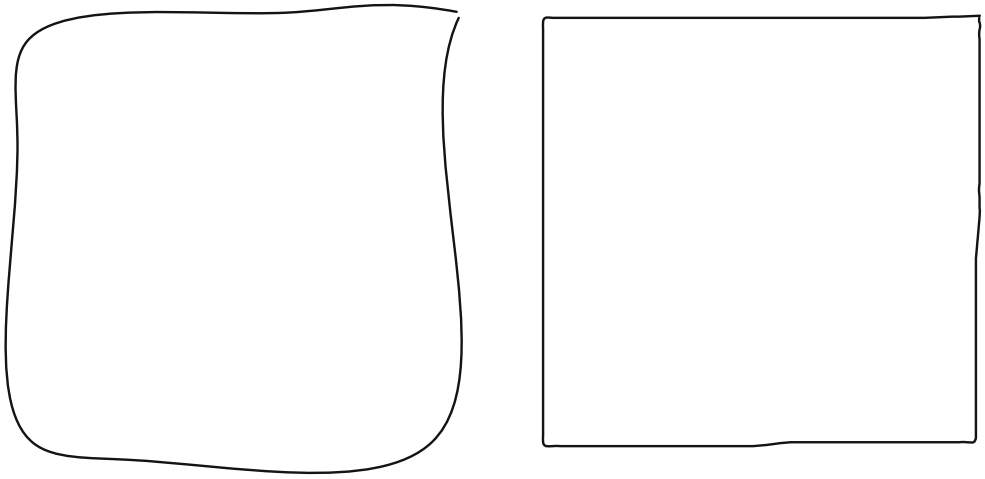


Abbildung 11: Links: integrierter Befehl „Sketch". Rechts: eigene Skizziermethode 2. (eigene Darstellung)

Rhino bietet bereits einen Befehl zum Skizzieren, der über die Python-Komponente in Grasshopper genutzt werden könnte. Allerdings ist dieser ungenau und es gibt keine Möglichkeit, die Genauigkeit oder die Anzahl der Punkte zu steuern. Ein Beispiel für diesen Nachteil ist das Zeichnen eines Quadrats. Wenn versucht wird, ein Quadrat zu zeichnen, dann entsteht ein Quadrat mit stark abgerundeten Ecken (siehe Abbildung 11). Diese Ungenauigkeit ist für die Klassifikation von Nachteil, weswegen die eigenen Skizzieralgorithmen erstellt wurden.

Der Code für den Transfer wurde in der Python-Komponente in Grasshopper geschrieben (siehe Abbildung 12). Der darin enthaltene Code ist im Anhang dargestellt (siehe 8.1), auf dessen Inhalt nun näher eingegangen wird.

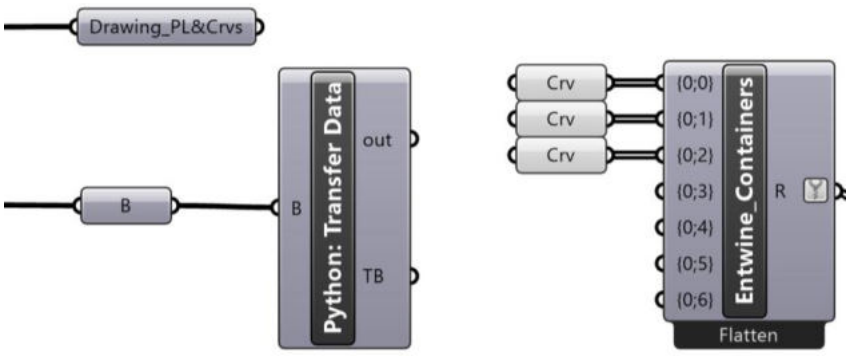


Abbildung 12: Data Transfer von „Drawing_PL&Crvs“ zu einem neuen Kurven-Kontainer (eigene Darstellung)

Erläuterung des Codes für den Transfer

Zeilen 001–018

Der Beginn entspricht dem Aufbau des Trainingscodes. Angeführt werden der urheberrechtliche Schutz, die Eingaben und Ausgaben der Python-Komponente in Grasshopper, die verwendeten Rhino- und Grasshopper-Versionen, der Name des Autors sowie das Datum der Erstellung der Komponente.

Zeilen 019–038

Nun folgt der Import der benötigten Bibliotheken, die für die Interaktion mit Grasshopper und die grafische Positionierung verwendet werden.

Anschließend wird eine Funktion zur Suche einer Komponente anhand ihres Namens definiert. Diese Funktion durchläuft alle Objekte im Grasshopper-Dokument (ghDoc) und gibt die gefundene Komponente mit dem entsprechenden Namen zurück. Dafür wurden spezifische Namen vergeben, um sie identifizieren zu können.

Danach wird die Funktion ausgeführt, um zwei Komponenten zu finden: die Komponente „Drawing_PL&Crvs“, von der die Daten entnommen werden sollen, und die „Entwine_Containers“, zu der ein neuer Container erstellt wird, um die Daten zu transferieren und mit der Entwine Komponente zu verbinden.

Zeilen 039–076

In diesem Abschnitt wird nach dem ersten freien Eingang der Entwine-Komponente gesucht. Falls bereits ein freier Eingang vorhanden ist, wird dessen Index gespeichert. Falls jedoch alle Eingänge belegt sind, wird ein neuer Eingangsparameter hinzugefügt. Dieser neue Parameter erhält eigene Eigenschaften wie eine Bezeichnung (NickName), eine Beschreibung (Description) und einen Zugangstyp als Baumstruktur (Access), da dies die nativen Eigenschaften eines neuen Parameters sind, wenn dieser manuell hinzufügt wird. Sobald der neue Parameter angelegt ist, muss der Parameter registriert und aktualisiert werden.

Zeilen 077–149

Ein neuer Kontainer wird relativ zum freien Eingang der Entwine-Komponente positioniert und eingefügt. Der Kontainer wird mit der Drawing_PL&Crvs-Komponente, die die Daten enthält, und der Entwine-Komponente verbunden. Daraufhin werden die Daten von der Drawing_PL&Crvs-Komponente in den neuen Kontainer anhaltend (persistent) übertragen. Abschließend wird die direkte Verbindung zwischen dem neuen Kontainer und Drawing_PL&Crvs entfernt. Die persistente Übertragung entspricht der Grasshopper-Funktion der Initialisierung, im Sinne einer Einsetzung und Speicherung der Daten.

4.2.2 Erstellung der parametrischen Objekte

Da das Ziel eine parametrische Zeichnung aus einer Skizze ist, wurden parametrische Objekte benötigt. Dazu wurden 2D-Darstellungen von Einrichtungselementen für die Verwendung in einem Grundriss parametrisch definiert (siehe Abbildung 13). Insgesamt wurden dreizehn Einrichtungselemente parametrisch erstellt: Badewanne, Dusche,

Waschbecken, Toilette, Bett, Garderobe, Sofa, Schreibtisch, Esstisch, Kasten, Tür und Fenster. Zudem wurde auch ein parametrisches Objekt für Raumaufteilungen definiert.

Die Eingaben der Komponenten umfasst einen Punkt für die Position (Pt), eine oder mehrere Grenzkurven (B), an denen das Objekt ausgerichtet werden soll, sowie Länge (L), Breite (W) und Rotationswinkel (D). Abhängig vom Einrichtungselement oder Raumaufteilung gibt es weitere oder andere Parameter.

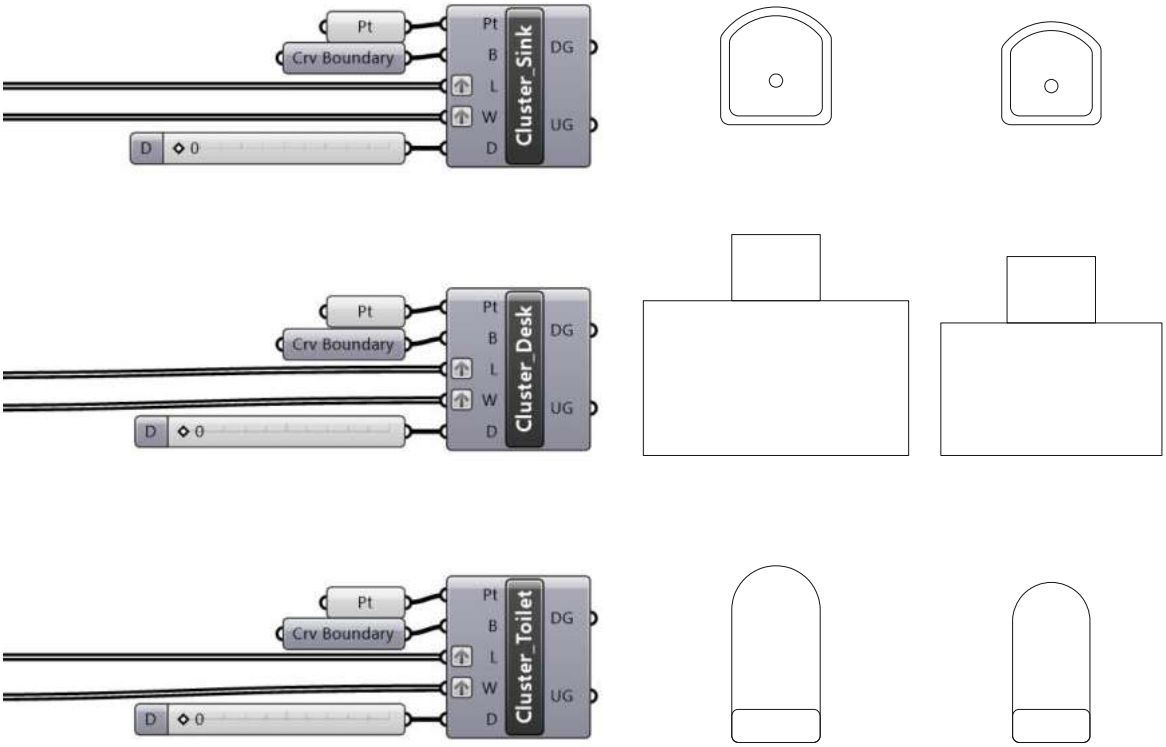


Abbildung 13: Einige Parametrische Einrichtungselemente. Komponenten und 2D-Darstellungen (eigene Darstellung)

Da die meisten Einrichtungselemente an Wänden positioniert sind, nimmt der Algorithmus den eingegebenen Punkt und sucht den nächstgelegenen Punkt an der Raumgrenzkurve. Falls mehrere Grenzkurven vorhanden sind, kann der Algorithmus automatisch diejenige auswählen, in der sich der Punkt befindet. Hierbei sollten geschlossene Polylinien als Raumgrenzen verwendet werden.

Die Ausgabe ist das 2D-Objekt, das sowohl am angegebenen Punkt (UG, Undocked Geometry) liegt als auch an der entsprechenden Grenzkurve angedockt wird (DG, Docked Geometry). Die Andockung erfolgt so, dass das 2D-Objekt an der Kurvennormalen der Grenzkurve ausgerichtet wird.

Zudem wurde besonderes Augenmerk auf das Listenmanagement gelegt, sodass mehrere unterschiedliche Eingabemaße und Einstellungen gleichzeitig verarbeitet werden können. Dadurch können mehrere eigenständige 2D-Objekte entstehen, ohne dass mehrere Komponenten desselben Einrichtungselements benötigt werden.

4.2.3 Aufbereitung der Skizzen und der parametrischen Objekte

Die Linien und Kurven der Skizze sowie die parametrischen Objekte mussten als Eingabe für die KI aufbereitet werden. Da die KI nur numerische Daten (Ganzzahlen oder Fließkommazahlen) verarbeiten kann, wurde ein Algorithmus erstellt, um die Form der gezeichneten Figuren durch eine Liste von Zahlen (Merkmalen) zu beschreiben. Hierfür wurden mehrere Ansätze implementiert (siehe Abbildung 14).

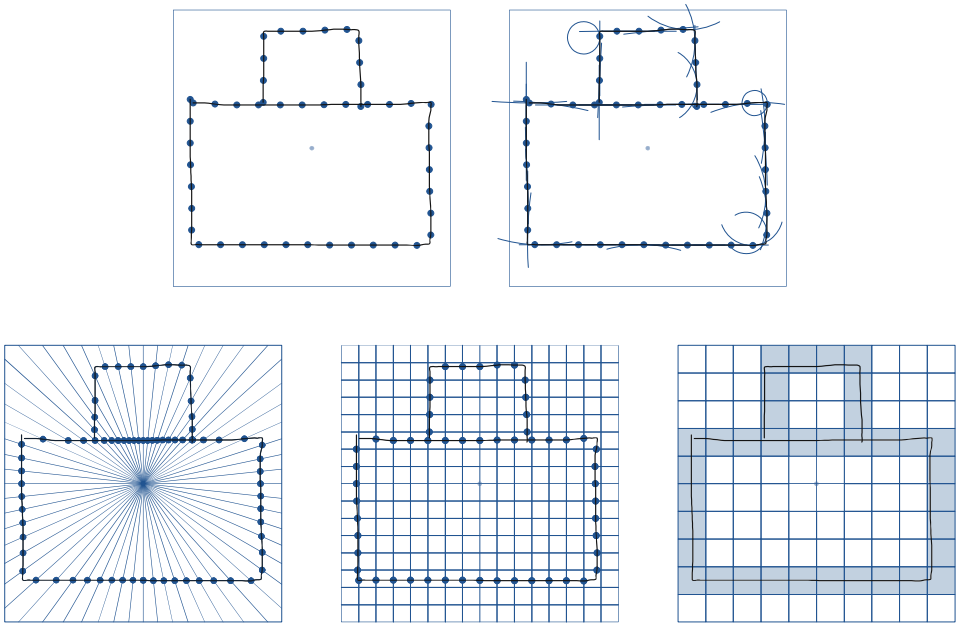


Abbildung 14: Aufbereitung der Skizzen und der parametrischen Objekte. Ansätze 1–5. (eigene Darstellung)

Ansatz 1

Jede Figur wird durch eine festgelegte Anzahl von Punkten regelmäßig unterteilt (siehe Abbildung 14, oben links). Für jeden dieser Punkte wird die Distanz zum Mittelpunkt des Umgrenzungsrechtecks der Figur berechnet und anschließend auf eine Domäne zwischen 0 und 1 normalisiert. Das Umgrenzungsrechteck passt sich parametrisch der Größe der Figur an. Dadurch werden die Grenzwerte aller Figuren vereinheitlicht. Das Ergebnis ist eine Liste numerischer Werte, die zwischen 0 und 1 liegen und sich als Eingabedaten für das neuronale Netzwerk eignen.

Ansatz 2

In diesem Ansatz erfolgt ebenfalls eine regelmäßige Unterteilung der Figur in eine bestimmte Anzahl von Punkten (siehe Abbildung 14, oben rechts). An jedem dieser Punkte wird die Krümmung berechnet, die dem Kehrwert des Kurvenradius an dieser Stelle entspricht. Auch diese Krümmungswerte werden anschließend auf einen Bereich von 0 bis 1 normalisiert.

Ansatz 3

Hier wurde eine parametrische Extraktionsmaske erstellt, die aus radialen Linien besteht (siehe Abbildung 14, unten links). Durch eine Extraktionsmaske werden Merkmale aus einer Figur extrahiert. Diese Extraktionsmaske wird auf jede Figur gelegt und es werden die Schnittpunkte zwischen den Linien und der Figur ermittelt. Die Distanzen dieser Schnittpunkte zum Mittelpunkt des Umgrenzungsrechtecks der Figur werden berechnet und ebenfalls auf einen Bereich zwischen 0 und 1 normalisiert.

Ansatz 4

Für diesen Ansatz wurde eine parametrische Extraktionsmaske erstellt, die aus einem rechteckigen Raster besteht (siehe Abbildung 14, unten mittig). Ähnlich wie zuvor wird diese auf die Figuren gelegt, woraufhin die Schnittpunkte mit der Figur ersichtlich werden. Die Entfernungen der Schnittpunkte zum Mittelpunkt des Umgrenzungsrechtecks der Figur werden berechnet und zwischen 0 und 1 normalisiert.

Ansatz 5

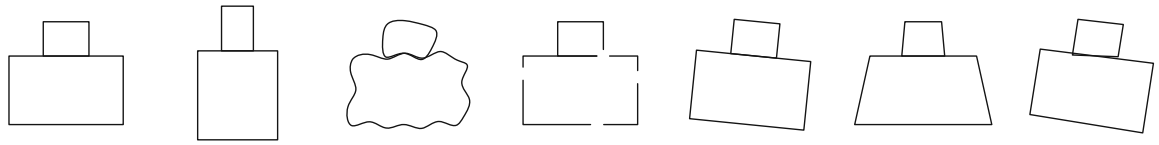
In diesem Ansatz kommt eine Extraktionsmaske aus rechteckigen Feldern zum Einsatz (siehe Abbildung 14, unten rechts). Diese wird innerhalb des Umgrenzungsrechtecks der Figur gesetzt. Felder, die die Figur schneiden, werden als „Wahr“ gekennzeichnet, während alle anderen Felder als „Falsch“ gekennzeichnet werden. Die Wahrheitswerte werden anschließend in 0 und 1 übersetzt.

4.2.4 Datenaugmentation

Damit die KI einerseits bestmöglich klassifizieren und andererseits auf unterschiedliche oder ungenaue Skizzen-Qualitäten der verschiedenen Nutzenden Rücksicht nehmen kann, wurde eine künstliche Vervielfältigung der Trainingsdaten durch Transformationen vorgenommen. Dieser Prozess, der als Datenaugmentation bezeichnet wird, soll zu leichten Variationen in den Trainingsdaten führen.

Dabei wurde darauf geachtet, eine möglichst große Anzahl unterschiedlicher Transformationen zu benutzen. Diese sind: Skalieren, Verzerren, Zerteilen, Scheren, Verjüngen und Winden (siehe Abbildung 15). Für jede dieser Transformationen wurden maximale Grenzzustände durch Grenzwerte definiert, die in den meisten Fällen Faktoren

von 0,9 bzw. 1,1 entsprechen. Ein Parameter steuert die Anzahl der verschiedenen Variationen innerhalb dieser Grenzwerte für jede Transformation.



Normal Skalieren Verzerren Zerteilen Scheren Verjüngen Winden

Abbildung 15: Transformationen am Beispiel Schreibtisch (eigene Darstellung)

Zusätzlich wurden alle Varianten der Transformationen durch eine Rotationstransformation (0° und 180°) um den Mittelpunkt des Umgrenzungsrechtecks jeder Figur erweitert. Dies ist notwendig, damit das neuronale Netzwerk auch gedreht gezeichnete Formen klassifizieren kann. Der Rotationswinkel kann nutzungsspezifisch über einen Parameter verkleinert, wodurch die Anzahl der Varianten erhöht werden kann.

Durch all diese Transformationen erhöht sich die Gesamtzahl der Variationen pro Figur, wodurch es der KI ermöglicht wird, aus einer größeren Anzahl unterschiedlicher Varianten zu lernen.

Schließlich wurde die Listenstruktur der Varianten so angepasst, dass alle Varianten derselben Figur in einer eigenen Liste in der Hierarchie „A“ organisiert sind. Das bedeutet, dass alle Varianten von Waschbecken in A = {0}, die Varianten von Schreibtischen in A = {1} und die Varianten von Toiletten in A = {2} abgelegt werden. Diese Struktur wurde parametrisch so erstellt, dass bei der Nutzung mehrerer Figuren die Organisation automatisch erfolgt.

Die Parameter der erstellten Varianten wurden separat gesammelt und ebenfalls parallel dazu sortiert. Für jedes Set identischer Parameter einer Figur wurde ein eigenes Etikett als numerische Ganzzahl vergeben, um die Zuordnung der Varianten zur jeweiligen Klasse zu erleichtern.

4.3 Integration

4.3.1 Training und Klassifizierung der KI

Für das Training wurden drei parametrische Einrichtungselemente ausgewählt: Waschbecken, Schreibtisch und Toilette. Für jede dieser Figuren wurden durch die Transformationen jeweils 108 Varianten erstellt. Die Eingabeparameter des neuronalen Netzwerks (Hyperparameter) wurden auf Basis des eigenen Verständnisses der KI schätzungsweise gewählt (siehe Tabelle 1) und sodann das Training gestartet.

L	D	M	HL1	HL2	Lr	Th	MIt
27	324	20	40	0	0,02	0,9	200

Tabelle 1: Geschätzte Parameter des neuronalen Netzwerks (eigene Daten)

Dabei gilt:

L = Anzahl der variablen Etiketten (Labels),

D = Gesamtanzahl aller Figuren einschließlich Variationen,

M = Merkmale: Attribute pro Figur,

HL1 = Anzahl der Neuronen in der versteckten Schicht 1,

HL2 = Anzahl der Neuronen in der versteckten Schicht 2,

Lr = Lernrate,

Th = Schwellenwert in %,

MIt = Maximale Iterationsanzahl.

4.3.2 Auswahl des parametrischen Objektes

Durch jede Klassifizierung einer Figur wurde durch das Neuronale Netzwerk eine Liste von Wahrscheinlichkeiten für jedes Etikett generiert. Der Index des Etiketts mit der höchsten Wahrscheinlichkeit wurde verwendet, um das dazugehörige Set von Parametern auszuwählen. Dadurch konnte automatisch die richtige Komponente mit den korrekten Parametern auf dem Grasshopper-Canvas platziert werden.

Um diesen Prozess des Platzierens zu automatisieren, wurde ein Algorithmus implementiert. Zunächst werden die Cluster für die Einrichtungselemente bzw. der Raumaufteilungen als Dateien in einem Ordner gespeichert, sodass sich mit der Zeit eine Bibliothek parametrischer Objekte aufbauen lässt, die für die Erstellung parametrischer Zeichnungen genutzt werden kann. Anschließend wird der passende Cluster basierend auf der Klassifizierung aus dem Ordner ausgewählt und auf dem Canvas platziert. Die Grenzkurve und der Punkt für die Position werden automatisch mit der neu eingefügten Cluster-Komponente verbunden. Zudem werden das durch die Klassifizierung ausgewählte Set von Parametern als Slider erstellt und automatisch an den entsprechenden Eingängen der Komponente angeschlossen.

Das Endergebnis dieses gesamten Prozesses und Algorithmus ist die Umwandlung einer Skizze in eine parametrische Zeichnung.

4.3.3 Optimierung der KI

Da Skizzen manchmal nicht richtig klassifiziert wurden, wurde versucht, die Parameter des neuronalen Netzwerks anzupassen und zu optimieren. Für die Optimierung wurde ein Fitness-Ziel definiert, anhand dessen die Leistung des neuronalen Netzwerks bewertet werden konnte.

Vorerst wurde die Optimierung mithilfe von Galapagos, einem evolutionären Optimierer, getestet. Aufgrund der langen Berechnungszeit reagierte das Interface von Galapagos jedoch nicht. Eine Mehrzieloptimierung (Multi-objective Optimization) mithilfe des Plugins Octopus [55], ebenfalls einem evolutionären Optimierer, wurde in Betracht gezogen, jedoch verworfen, da anzunehmen war, dass dasselbe Problem auftreten würde. Stattdessen wurde eine manuelle Methode angewendet.

Manuelle Optimierung

Ein Parameter wurde mehrmals verändert, wobei die übrigen gleich belassen wurden. Alle Parameter inklusive des Fitness-Werts (Z; siehe Formel 10) wurden dazu aufgezeichnet. Der Wert mit der höchsten Fitness und/oder der Wert mit der höchsten Anzahl korrekter Vorhersagen (in %) wurde/n für den nächsten Schritt, die Optimierung der nächsten Variable, genommen. So wurden schrittweise verschiedene Parameterkombinationen getestet. Diese Parameter und die entsprechenden Fitness-Werte wurden aufgezeichnet und analysiert, um eine optimale Auswahl der multiplen Parameter zu erreichen. Hierfür wurden dieselben Beispiele, die im Training verwendet wurden, für die Klassifizierung genutzt, um zu überprüfen, wie viele Figuren des Trainings korrekt klassifiziert werden.

$$\mathrm{Z} = \frac{C \cdot F}{T}$$

Formel 10: Fitness-Zielfunktion (eigene Formel)

Dabei gilt:

Z = Fitness-Ziel,

C = Anzahl der richtigen Klassifizierungen,

F = Faktor als Gewichtung,

T = benötigte Rechenzeit.

In der Formel geht es darum, C zu maximieren, während gleichzeitig T minimiert werden soll. Mithilfe des Gewichtungsfaktors F soll C gegenüber T priorisiert werden.

Im Anhang befindet sich die Protokollierung der manuellen Optimierung (siehe 8.2).

Das Ergebnis der Optimierung ist wie folgt:

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	30	2160	120	30	0	0,001	1,0	400	1542	46,2	0,714	334

Tabelle 2: Ergebnis der manuellen Optimierung der Parameter des neuronalen Netzwerks (eigene Daten)

Dabei gilt:

L = Anzahl der variablen Etiketten (Labels),

V = Varianten pro Figur,

D = Gesamtanzahl aller Figuren einschließlich Variationen,

M = Merkmale: Attribute pro Figur,

HL1 = Anzahl der Neuronen in der versteckten Schicht 1,

HL2 = Anzahl der Neuronen in der versteckten Schicht 2,

Lr = Lernrate,

Th = Schwellenwert in % (Threshold),

MIt = Maximale Iterationsanzahl,

C = Anzahl der korrekten Vorhersagen,

T = Berechnungszeit in Minuten,

A = Korrekte Vorhersagen in %,

Z = Fitness-Ziel: $Z = \frac{C \cdot F}{T}$,

F = Gewichtungsfaktor (10).

5 Ergebnisse und Diskussion

Die Folgenden dargestellten Ergebnisse und Diskussionen beruhen auf eigener Testung, Beobachtung, kritischer Reflexion sowie der Interpretation der Algorithmen und ihrer Funktionen bzw. Ausgaben. Im Anschluss daran folgt eine Beschreibung und Auswertung einer Studie, in der Testpersonen herangezogen wurden.

Digitales Skizzieren

Das digitale Skizzieren bietet im Vergleich zum analogen Skizzieren auf Papier, das anschließend digitalisiert werden muss, mehrere Vorteile. Zum einen entfällt die aufwändige Übertragung der Skizze auf den Computer, zum anderen kann direkt in einem vorhandenen digitalen Plan skizziert werden.

Dabei ist es möglich, mit einer Computermaus oder einem Eingabestift zu skizzieren. Der Eingabestift funktioniert allerdings nur, wenn dieser durch eine „Touchpad“-Funktion benutzt wird, da sonst die Interpretation in der Rhino-Umgebung nicht möglich ist.

Durch die algorithmischen Ansätze des Skizzierens kann genauer gezeichnet werden als mit dem integrierten Rhino-Befehl „Sketch“, da die Anzahl der Punkte pro Intervall für Kurven selbst mithilfe eines Reglers bestimmt werden kann. Dies wurde untersucht, indem ein einfaches Quadrat, das sich im Hintergrund befindet, (ohne Objektfangoptionen) nachgezeichnet wurde (siehe Abbildung 16). Anschließend wurden die Distanzen zwischen regelmäßig gesetzten Punkten auf dem Quadrat und der gezeichneten Figur gemessen, summiert und durch die Anzahl der Punkte geteilt, um eine durchschnittliche Abweichung zu ermitteln. Wird die durchschnittliche Abweichung durch die Gesamtlänge des Quadrats (Umfang) dividiert, ergibt dies eine relative Abweichung.

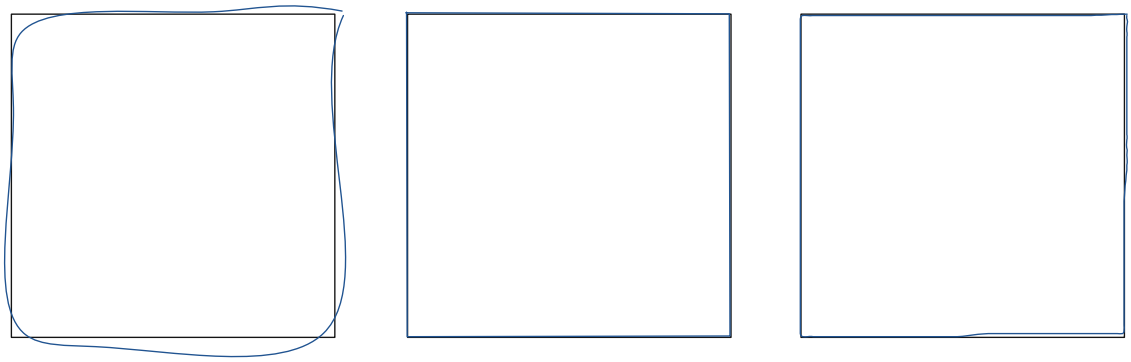


Abbildung 16: Genauigkeit des digitalen Skizzierens. Von links nach rechts: Rhino-Befehl „Sketch“, Skizziermethode 1, Skizziermethode 2. (eigene Darstellung)

Die Skizze, die mit dem Befehl „Sketch“ erstellt wurde, wies eine relative Abweichung von 0,49 % auf, während die Skizziermethode 2 eine Abweichung von 0,11 % und Methode 1 0,05 % Einheiten ergaben. Skizziermethode 1 ist in dieser Testung die genaueste, da hier lediglich die Eckpunkte gesetzt werden müssen, die anschließend zu einer Polylinie verbunden werden.

Zu empfehlen ist, das Intervall für den Algorithmus nicht unter 10 Millisekunden einzustellen, da jede Berechnung eines Durchlaufs ca. 10 Millisekunden in Anspruch nimmt. Besser wäre ein Intervall von ca. 30 bis 50 Millisekunden, damit keine Stapelung der Berechnungen entsteht. Das Intervall regelt ausschließlich die Anzahl der Punkte, die in einer wiederkehrenden Zeitspanne gesetzt werden, und muss deshalb nicht in Abhängigkeit von der Geometrie oder der Größe der Figur gewählt werden. Bei einem Intervall von 30 bis 50 Millisekunden ergibt dies eine gute Punktdichte und damit eine hohe Zeichengenauigkeit (siehe Abbildung 16).

Aufbereitung der Daten

Nach der Erstellung und Testung von Ansatz 1 und Ansatz 2 (siehe 4.2.3) wurde deutlich, dass beide ein Problem aufweisen: Die Unterteilung der Figur durch Punkte beginnt immer dort, wo jede Kurve ihren Anfang besitzt, demnach wo begonnen wurde, zu zeichnen. Das

führt dazu, dass die Punktsetzung bei jeder Skizze potenziell an einer anderen Stelle startet, wodurch unterschiedliche Ergebnisse bei der Aufbereitung entstehen können.

Jede Figur sollte jedoch einheitlich beschrieben werden – unabhängig davon, wie sie gezeichnet wurde oder wie sie aussieht. Daher wurden in den darauffolgenden Ansätzen Extraktionsmasken benutzt, die auf die Figur gelegt werden.

Ansatz 3 (siehe 4.2.3) hat die Problematik, dass Skizzenstriche, die parallel zu den radialen Linien der Extraktionsmaske verlaufen, entweder gar nicht oder nur mit geringer Genauigkeit geschnitten werden.

Ansatz 4 (siehe 4.2.3) weist diese Problematik nicht auf, da die Linien der Extraktionsmaske aus mehreren Richtungen auf die Figur treffen und so eine genauere Extraktion ermöglichen.

Ansatz 5, der auf rechteckigen Feldern basiert (siehe 4.2.3), kann als Abstraktion einer Pixelgrafik gesehen werden, die ausschließlich aus schwarzen oder weißen Feldern (ohne Graustufen) besteht. Um jedoch eine möglichst genaue Beschreibung der Figur zu erhalten, müsste die Anzahl der Felder sehr hoch gewählt werden, was wiederum zur Folge hätte, dass auch die Anzahl der Merkmale für die KI steigt. Dieser Ansatz hat jedoch den Vorteil, dass keine Fließkommazahlen entstehen, wie es bei den vorherigen Ansätzen der Fall ist, sondern ausschließlich Werte von Null oder Eins. Dadurch muss die KI nicht mit Zahlen mit mehreren Nachkommastellen und entsprechend hoher Genauigkeit arbeiten.

Ansatz 4 (siehe 4.2.3) scheint von allen der geeignetste zu sein, da dieser mit vergleichsweise wenigen Messpunkten eine sehr genaue numerische Beschreibung der Figur liefert und daher gegenüber Ansatz 5 (und den anderen Ansätzen) bevorzugt wurde.

Datenaugmentation

Für die künstliche Vervielfältigung der Trainingsdaten wurden verschiedene Transformationen auf die Idealfigur (parametrisch erstellte Figur) angewendet. Allerdings ist anzumerken, dass keine Kombinationen von Transformationen eingesetzt wurden.

Eine mögliche Verbesserung wäre, alle Kombinationen von Transformationen hinzuzufügen, beispielsweise eine Scherung in Kombination mit einer Verzerrung. Eine weitere Verbesserung wäre, zusätzlich Beispiele hinzuzufügen, die von verschiedenen Personen gezeichnet wurden. Durch diese beiden Maßnahmen könnten die Trainingsdaten sinnvoll erweitert und die Klassifikationsleistung der KI verbessert werden.

Die beste Erweiterung wäre, sämtliche Transformationen sowie deren Kombinationen sowohl auf die Idealfigur als auch auf die von Personen gezeichneten Figuren anzuwenden. Dabei ist jedoch zu beachten, dass für jede manuell gezeichnete Figur für die Eingabe der entsprechenden parametrischen Objekte zusätzlich manuell eine Liste von Parametern gesammelt werden müsste, was den Aufwand erhöht.

Auch die Grenzwerte der Transformationen könnten optimiert werden, da sich sowohl zu große als auch zu kleine Transformationen negativ auf die Klassifikation auswirken können.

Für die Rotationstransformation wurden bislang zwei Winkel (0° und 180°) verwendet. Für die praktische Anwendung sollte auf einen anderen brauchbaren Winkelabstand gesetzt werden, beispielsweise alle 45° oder 10°. Dadurch wird es der KI möglich, gedreht skizzierte Figuren zu erkennen.

Optimierung der Hyperparameter

Durch die manuelle Optimierung der Hyperparameter des neuronalen Netzwerks konnte eine Klassifizierungsgenauigkeit von 71,4 % erreicht werden. Nach der Studie von Eitz et al. [19] erzielten Testpersonen eine Erkennungsgenauigkeit von 73,1 %. Das ergibt eine Differenz von 1,7 %.

Wird die Verlustkurve betrachtet, indem für jede Iteration der durchschnittliche Gesamtverlust des Netzwerks ermittelt und dieser als Graphen darstellt wird (x-Achse: Iterationszahl, y-Achse: Höhe des Verlusts), ergibt sich eine Kurve mit exponentiell fallendem Verlauf (siehe Abbildung 17). Das bedeutet, dass die Reduktion des Verlusts zu Beginn hoch ausfällt, mit zunehmender Iterationszahl jedoch abnimmt, bis sie auf einem Plateau asymptotisch verläuft.

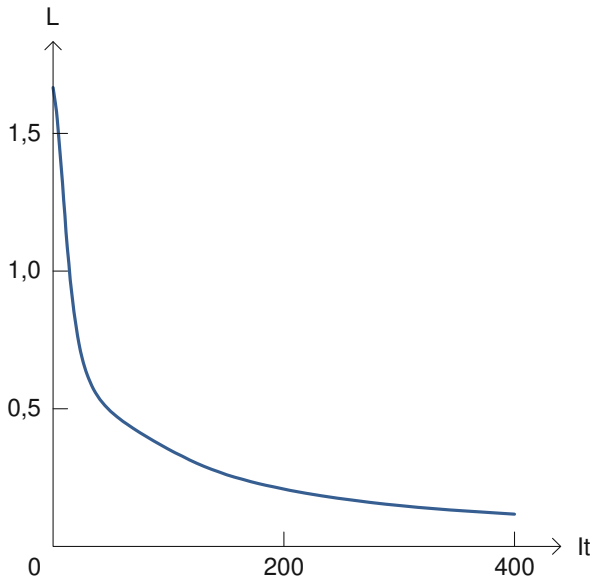


Abbildung 17: Verlustkurve des Ergebnisses der manuellen Optimierung. (eigene Darstellung)

Dies führt zu der Idee, die Lernrate im Trainingsverlauf nicht konstant zu halten, sondern adaptiv an den Verlustwert anzupassen. Je höher der Verlustwert einer Iteration ist, desto höher könnte die Lernrate (in Prozent) verwendet werden. Dennoch bleibt die Frage nach den Grenzen (Domaine) dieser Adaption und damit dem Adaptionswert selbst, die je nach Netzwerkeinstellung und Trainingsfortschritt unterschiedlich ausfallen können. Die Stärke und Richtung der Adaption müssten von der Steigung des Kurvenverlaufs an der jeweiligen Iterationsstelle abhängen. Um diesen zu erhalten, müssten zumindest die Verlustwerte von zwei vorherigen Iterationen gespeichert werden. Zusammengefasst handelt es sich dabei um eine adaptive Lernrate, die auf dem Moment des Verlaufs basiert.

Nach einiger Recherche wurde erkenntlich, dass ein ähnlicher Algorithmus für neuronale Netzwerke von Kingma und Ba [56] mit der Bezeichnung „Adam“ vorhanden ist. Dieser richtet sich nicht direkt nach dem Verlust, der durch die Verlustfunktion berechnet wird, sondern nach dessen erste Ableitung, welche auch die Steigung angibt.

Benutzung

Wenn weitere parametrische Objekte benötigt werden, müssen diese entsprechend den vorhandenen Komponenten als Cluster angelegt werden. Dabei ist darauf zu achten, dass die Reihenfolge der Eingänge gleich bleibt, da diese später nach der Klassifizierung automatisch verbunden werden. Die Cluster sollten als ghcluster-Dateityp (per Rechtsklick auf dem Cluster) in einem spezifischen Ordner gespeichert und gesammelt werden, da die Auswahl der richtigen Komponente nach der Klassifizierung aus diesem Ordner erfolgt.

Anschließend sollten die Varianten der parametrischen Objekte über deren Parameter (etwa Länge oder Breite) sowie die Anzahl der Varianten über die Datenaugmentation bzw. die Rotationstransformation mithilfe von Reglern eingestellt werden. Danach kann das neuronale Netzwerk trainiert werden.

Sobald das Netzwerk trainiert ist, kann durch das digitale Skizzieren eine Figur gezeichnet werden, die anschließend automatisch klassifiziert wird. Die richtige Komponente des parametrischen Objekts wird daraufhin automatisch auf den Canvas gelegt. Auch die passenden Eingaben für diese Komponente werden automatisch erzeugt und verbunden.

Empfohlen wird, dass bei der Erweiterung um weitere algorithmische Objekte kein zweites neuronales Netzwerk (z. B. durch eine Kopie der Komponente) trainiert wird, sondern ein einzelnes Netzwerk für alle Objekte verwendet wird.

Arbeitsablauf und mögliches Einsatzszenario

In einer Konzeptphase möchte eine Architektin oder ein Architekt, der gerne freihändig zeichnet, einige Grundrissideen für den Wohnbau ausprobieren. Gewöhnlich verwendet sie oder er einen Stift und skizziert auf ein Blatt Papier. Dabei muss sie oder er auf Maßstab und Maße achten. Anschließend werden als gut befundene Skizzen manuell in ein CAD-Programm übertragen.

Durch die Verwendung des erstellten Algorithmus kann stattdessen direkt digital mit der Maus oder einem Eingabestift in Rhino skizziert werden. Die Raumaufteilung und Einrichtungselemente können dabei ohne Maßstab oder genaue Maße skizziert werden. Dazu wird die Datei mit dem Algorithmus in Grasshopper gestartet und mit einem Klick auf den Start-Knopf kann direkt mit dem Skizzieren begonnen werden.

Wenn eine Skizze gelöscht oder wiederholt werden soll kann die „d“ Taste benutzt werden. Wenn sie passt, kann durch die die Taste „f“ zum Klassifizieren übergeben werden. Dadurch wird das passende parametrische Objekt mitsamt den entsprechenden Parametern auf dem Canvas platziert. Im Nachhinein können über die Parameter, falls gewollt oder erforderlich, die Position oder die Maße des Objekts verändert werden.

Durch den Einsatz von Algorithmen wurden, soweit möglich, alle Schritte des Ablaufs automatisiert, wodurch der manuelle Aufwand reduziert und die Benutzung vereinfacht wurden.

Da es in dieser Arbeit darum geht, 2D-parametrische Architekturzeichnungen zu erstellen und zu verwenden, erfordert der vorgeschlagene Arbeitsablauf ein Maß an Kompetenz im Umgang mit Grasshopper, wenn eigene oder andere parametrische Komponenten hinzugefügt werden sollen.

Dieser Arbeitsablauf dient Architektinnen und Architekten als ein Werkzeug. Auf diese Weise wird die Tätigkeit nicht von einer KI ersetzt, sondern von dieser gezielt unterstützt.

Beschreibung und Auswertung der Studie

Eine qualitative Studie wurde mit zehn Personen durchgeführt, die einen Ausbildungshintergrund in der Architektur besitzen. Die Teilnehmenden wurden etwa zwei Minuten lang in die Handhabung des Algorithmus eingeführt und erhielten anschließend einige Minuten Zeit, um diesen auszuprobieren. Danach wurden sie gebeten, den vorgestellten Algorithmus anhand von drei Kriterien in einem Fragebogen zu bewerten. Der Fragebogen ist im Anhang dargestellt (siehe 8.3).

Der Fragebogen umfasste persönliche Angaben wie Name, aktuelles Studium (Bachelor oder Master) bzw. vorhandenen Abschluss falls kein Studium absolviert wird, sondern ausschließlich gearbeitet wird. Außerdem wurde abgefragt, ob Erfahrung mit Grasshopper vorhanden ist. Der Fragebogen beinhaltete folgende Kategorien und Fragen:

- (K1) Verständlichkeit:
  (F1) Wie beurteilen Sie die Dauer, die Sie für den Einstieg in die Nutzung des Algorithmus benötigt haben?
  (F2) Wie nachvollziehbar war der Zweck des Algorithmus für Sie?

- (K2) Nutzungsfreundlichkeit:
  (F3) Wie intuitiv und nutzungsfreundlich empfanden Sie die Nutzung des Algorithmus?
  (F4) Wie überzeugend fanden Sie die Funktion und Ergebnisse des Algorithmus?

- (K3) Praxistauglichkeit:
  (F5) Würden Sie den Algorithmus in Ihrem Arbeitskontext (Studium oder Beruf) einsetzen?
  (F6) Wie gut lässt sich der Algorithmus aus Ihrer Sicht in einen Arbeitsablauf integrieren?

Die Fragen waren auf einer Skala von 1 bis 10 zu beantworten, wobei 1 für „sehr schlecht“ und 10 für „sehr gut“ stand. Eine Auflistung der Bewertungen der Testpersonen ist in der Tabelle dargestellt (siehe Tabelle 3).

		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
K1	F1	10	10	10	10	10	10	10	10	10	10
	F2	10	10	9	10	10	10	10	8	10	10
K2	F3	8	9	10	10	10	10	10	10	10	10
	F4	10	10	10	10	10	10	10	9	9	8
K3	F5	10	10	8	10	10	9	10	10	9	8
	F6	10	10	8	10	10	10	10	10	10	8

Tabelle 3: Bewertungen der Testpersonen (eigene Daten)

Dabei gilt:

K1, K2, K3 = Kategorie (Verständlichkeit, Nutzungsfreundlichkeit, Praxistauglichkeit)

F1, F2, F3, …, Fn = Frage

P1, P2, P3, …, Pn = Person

Zusammenfassend lässt sich sagen, dass die Bewertungen der Fragen unabhängig von der persönlichen Situation der Teilnehmenden (Studium oder Abschluss) sowie ihrer Erfahrung mit Grasshopper waren.

Im Durchschnitt ergab sich über alle Testpersonen hinweg folgende Auswertung der einzelnen Kategorien, wobei 100 % die maximal erreichbare Punktzahl entspricht:

- Verständlichkeit: 98,5 %
- Nutzungsfreundlichkeit: 96,5 %
- Praxistauglichkeit: 95%

Der Durchschnitt der drei Kategorien ergab 96,6 %.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Arbeitsablauf entwickelt, der es Architektinnen und Architekten ermöglicht, anhand digitaler Skizzen eine parametrische Zeichnung durch vordefinierte parametrische Objekte zu erzeugen. Im Zentrum stand dabei der Einsatz eines neuronalen Netzwerks, das anhand von eigenen Trainingsdaten in der Lage ist, skizzierte Figuren zu klassifizieren und die passenden parametrischen Komponenten in Grasshopper zu platzieren. Unterstützt wurde dieser Prozess durch verschiedene Datenaufbereitungsmethoden, Transformationen zur Datenaugmentation sowie eine manuelle Optimierung der Hyperparameter.

Die Ergebnisse zeigen, dass der Ansatz auf Basis eines Mehrlagigen Perzeptrons in der Lage ist, eine zufriedenstellende Klassifikationsgenauigkeit zu erreichen. Durch die Verwendung der digitalen Skizzen sowie Extraktionsmasken zur Vereinheitlichung der Eingabedaten konnte ein zusammenhängendes digitales System geschaffen werden. Der Arbeitsablauf wurde dabei so gestaltet, dass möglichst viele Schritte durch Algorithmen automatisiert ablaufen, um den manuellen Aufwand zu minimieren und eine einfache Nutzung im architektonischen Entwerfen zu ermöglichen.

Neben der erarbeiteten Funktionalität besteht dennoch Potenzial zur Weiterentwicklung. Eine mögliche Verbesserung bestünde darin, durch Anmerkungen in der Skizze spezifische Einschränkungen oder Funktionen bestimmen zu können, ähnlich wie bei Keshavarzi et al. [22]. Auch die Benutzung eines autoregressiven Modells, das eine Skizze im Kontext bereits vorhandener Skizzen oder Planzeichnungen interpretiert, wäre eine Verbesserung. Dadurch eröffnet sich die Möglichkeit, dass die Klassifikation einer Skizze kontextabhängig erfolgt, wie es bei Tiwari et al. [20] der Fall ist. Der dort verwendete Ansatz der Strich-zu-Primitiv-Umwandlung erscheint allerdings für diesen Zweck wenig geeignet. Vermutlich wäre der Einsatz eines völlig anderen KI-Modells zielführender, das alle Objekte eines gegebenen Ausschnitts gleichzeitig partitionieren und klassifizieren kann. Denkbar wäre hier ein Ansatz, wie er bei Bilddaten durch das Faster Region-Based Convolutional Neural Network (Faster R-CNN) verwendet wird, jedoch angepasst auf numerische Daten.

Die Verbindung von intuitiver Skizzeneingabe und parametrischer Zeichnungserstellung könnte ein zukunftsweisendes Werkzeug für den architektonischen Entwurfsprozess darstellen. Die vorgestellte Methode verdeutlicht, dass KI nicht als Ersatz, sondern als unterstützendes Werkzeug der Entwurfstätigkeit verstanden werden kann, ohne dabei die gestalterische Rolle der Architektinnen und Architekten zu unterbinden.

7 Verzeichnisse

7.1 Tabellenverzeichnis

7.2 Abbildungsverzeichnis

7.3 Literaturverzeichnis

[1] T. A. Heilmann, „Algorithmus," in *Mensch-Maschine-Interaktion*, Stuttgart, J.B. Metzler, 2019, p. 229–231.

[2] v. M. Emden, „Egyptian multiplication and some of its ramifications," arXiv, Ithaca, New York, 2019.

[3] A. A. Stepanov und D. E. Rose, From mathematics to generic programming, Upper Saddle River, NJ: Addison-Wesley, 2015.

[4] M. H. Akhtar und J. Ramkumar, „AI in Architecture," in *AI for Designers*, Singapur, Springer, 2024, pp. 67-84.

[5] M.-T. Hütt und C. Schubert, „Fairness von KI-Algorithmen," in *Philosophisches Handbuch Künstliche Intelligenz*, Wiesbaden, Springer VS, 2020, pp. 1-22.

[6] „Rhino3d," [Online]. Available: https://www.rhino3d.com/mcneel/trademarks/. [Zugriff am 08 02 2025].

[7] F. Lyn und R. Dulaney Jr., „A Case for Drawing," *Enquiry: The ARCC Journal for Architectural Research,* Bd. 6, Nr. 1, pp. 23-30, 01 12 2009.

[8] H. Nejadriahi und K. Arab, „A Study on the Impacts of Computer Aided Design on the Architectural Design Process," *World Academy of Science, Engineering and Technology - International Journal of Architectural and Environmental Engineering,* Bd. 11, Nr. 8, p. 1054–1058, 03 06 2017.

[9] „Bielefeld Academic Search Engine (BASE)," [Online]. Available: https://www.base-search.net/. [Zugriff am 15 01 2025].

[10] „Springer Link," [Online]. Available: https://link.springer.com/. [Zugriff am 15 01 2025].

[11] „Science Direct," [Online]. Available: https://www.sciencedirect.com/. [Zugriff am 15 01 2025].

[12] „JSTOR," [Online]. Available: https://www.jstor.org. [Zugriff am 15 01 2025].

[13] „European Open Science Cloud," [Online]. Available: https://open-science-cloud.ec.europa.eu/resources/publications. [Zugriff am 15 01 2025].

[14] „Scopus,“ [Online]. Available: https://www.scopus.com/search/form.uri?zone=TopNavBar&origin=searchbasic&display=basic. [Zugriff am 15 01 2025].

[15] „CumInCAD,“ [Online]. Available: https://papers.cumincad.org/. [Zugriff am 15 01 2025].

[16] „Universität Wien,“ [Online]. Available: https://gonline.univie.ac.at/methoden-quellen/quellenkritik/. [Zugriff am 13 01 2025].

[17] „Universitätsbibliothek Humboldt,“ [Online]. Available: https://www.ub.hu-berlin.de/de/bibliotheksglossar/peer-review-verfahren. [Zugriff am 13 01 2025].

[18] L. Sadouk, T. Gadi und E. H. Essoufi, „A Novel Approach of Deep Convolutional Neural Networks for Sketch Recognition,“ in *Proceedings of the 16th International Conference on Hybrid Intelligent Systems (HIS 2016)*, Cham, Springer, 2017, pp. 99-112.

[19] M. Eitz, J. Hays und M. Alexa, „How do humans sketch objects?,“ *ACM Transactions on Graphics (TOG),* Bd. 31, Nr. 4, pp. 1-10, 01 07 2012.

[20] A. Tiwari, S. Biswas und J. Lladós, „SketchGPT: Autoregressive Modeling for Sketch Generation and Recognition,“ in *Document Analysis and Recognition - ICDAR 2024*, Cham, Springer, 2024, p. 421–438.

[21] S. Alaniz, M. Mancini, A. Dutta und F. Galasso, „Abstracting Sketches Through Simple Primitives,“ in *Computer Vision – ECCV 2022*, Cham, 2022.

[22] M. Keshavarzi, C. Hotson, C.-Y. Cheng, M. Nourbakhsh, M. Bergin und M. Rahmani Asl, „SketchOpt: Sketch-based Parametric Model Retrieval for Generative Design,“ in *CHI EA '21: Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, Yokohama, 2021.

[23] „Food4Rhino,“ [Online]. Available: https://www.food4rhino.com/en. [Zugriff am 15 01 2025].

[24] „Food4Rhino,“ [Online]. Available: https://www.food4rhino.com/en/app/crow-artificial-neural-networks. [Zugriff am 15 01 2025].

[25] „University of Stuttgart,“ [Online]. Available: https://www.icd.uni-stuttgart.de/team/Felbrich/. [Zugriff am 15 01 2025].

[26] „NuGet,“ [Online]. Available: https://www.nuget.org/packages/NeuronDotNet. [Zugriff am 15 01 2025].

[27] „GitHub,“ [Online]. Available: https://github.com/HeinzBenjamin/Crow. [Zugriff am 15 01 2025].

[28] „Food4Rhino,“ [Online]. Available: https://www.food4rhino.com/en/app/lunchbox. [Zugriff am 15 01 2025].

[29] „Proving Ground,“ [Online]. Available: https://apps.provingground.io/lunchbox/. [Zugriff am 15 01 2025].

[30] „Proving Ground,“ [Online]. Available: https://provingground.io/about/. [Zugriff am 15 01 2025].

[31] „Nathan Miller,“ [Online]. Available: https://www.nathanhmiller.org/. [Zugriff am 15 01 2025].

[32] „Microsoft,“ [Online]. Available: https://learn.microsoft.com/de-de/azure/architecture/ai-ml/guide/data-science-and-machine-learning?utm_source=chatgpt.com. [Zugriff am 15 01 2025].

[33] „Accord-Framework,“ [Online]. Available: https://accord-framework.net/. [Zugriff am 15 01 2025].

[34] „Proving Ground,“ [Online]. Available: https://apps.provingground.io/docs/lunchbox-documentation/. [Zugriff am 15 01 2025].

[35] „Microsoft Corporation,“ [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/excel. [Zugriff am 19 04 2025].

[36] „Food4Rhino,“ [Online]. Available: https://www.food4rhino.com/en/app/dodo. [Zugriff am 15 01 2025].

[37] „LinkedIn,“ [Online]. Available: https://www.linkedin.com/in/loregreco/?originalSubdomain=uk. [Zugriff am 15 01 2025].

[38] „NLopt Doc,“ [Online]. Available: https://nlopt.readthedocs.io/en/latest/. [Zugriff am 15 01 2025].

[39] „Raphos,“ [Online]. Available: https://raphos.com/geometry/dodo-scientific-tools-and-ai-for-grasshopper/. [Zugriff am 15 01 2025].

[40] „GrasshopperDocs,“ [Online]. Available: https://grasshopperdocs.com/addons/dodo.html. [Zugriff am 15 01 2025].

[41] A. Mockenhaupt, „Grundlagen der Künstlichen Intelligenz (KI),“ in *Digitalisierung und Künstliche Intelligenz in der Produktion*, Wiesbaden, Springer, 2024, pp. 53-104.

[42] H. Helbig, „Wunder der Technik und das Problem der Künstlichen Intelligenz,“ in *Welträtsel aus Sicht der modernen Wissenschaften*, Berlin, Heidelberg, Springer, 2018, p. 473–56.

[43] B. Copeland, „Britannica,“ [Online]. Available: https://www.britannica.com/technology/artificial-intelligence?utm_source=chatgpt.com. [Zugriff am 25 03 2025].

[44] K. Mainzer und R. Kahle, „Zum Begriff der Künstlichen Intelligenz,“ in *Grenzen der KI – theoretisch, praktisch, ethisch*, Berlin, Heidelberg, Springer, 2022, pp. 1-7.

[45] F. Weber, „Künstliche Intelligenz,“ in *Künstliche Intelligenz für Business Analytics*, Wiesbaden, Springer , 2020, p. 37–72.

[46] M. Trabs, M. Jirak, K. Krenz und M. Reiß, „Klassifikation,“ in *Statistik und maschinelles Lernen*, Berlin, Heidelberg, Springer Spektrum, 2021, p. 185–234.

[47] S. Stefan, „Neuronale Netze,“ in *Data Science Training - Supervised Learning*, Berlin, Heidelberg, Springer, 2024, p. 237–263.

[48] N. Handa, Y. Kaushik, N. Sharma, M. Dixit und M. Garg, „Image Classification Using Convolutional Neural Networks (ICAICR 2020),“ in *Advanced Informatics for Computing Research*, Singapore, 2021.

[49] S. Selle, „Datenvorbereitung,“ in *Data Science Training - Supervised Learning*, Berlin, Heidelberg, Springer, 2024, p. 129–158.

[50] G. A. Gomes de Sá, C. H. Fontes und M. Embiruçu, „A new method for building single feedforward neural network models for multivariate static regression problems: a combined

weight initialization and constructive algorithm," *Evolutionary Intelligence,* Bd. 17, Nr. 4, p. 1221–1233, 26 12 2024.

[51] K. Wong, R. Dornberger und T. Hanne, „An analysis of weight initialization methods in connection with different activation functions for feedforward neural networks," *Evolutionary Intelligence,* Bd. 17, Nr. 4, p. 2081–2089, 24 11 2022.

[52] E. C. Seyrek und M. Uysal, „A comparative analysis of various activation functions and optimizers in a convolutional neural network for hyperspectral image classification," *Multimedia Tools and Applications,* Bd. 83, p. 53785–53816, 24 11 2024.

[53] B. H. Nayef, S. N. H. Sheikh Abdullah, R. Sulaiman und Z. A. A. Alyasseri, „Optimized leaky ReLU for handwritten Arabic character recognition using convolution neural networks," *Multimedia Tools and Applications,* Bd. 81, Nr. 2, p. 2065–2094, 19 10 2022.

[54] S. Selle, „Mehrklassen-Klassifikation," in *Data Science Training - Supervised Learning*, Berlin, Heidelberg, Springer Vieweg, 2024, p. 373–393.

[55] „Food4Rhino," [Online]. Available: https://www.food4rhino.com/en/app/octopus. [Zugriff am 01 02 2025].

[56] D. P. Kingma und J. Ba, „Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR 2015)*, San Diego, 2014.

[57] M. Eitz, J. Hays und M. Alexa, „How do humans sketch objects?," *ACM Transactions on Graphics (TOG),* Bd. 31, Nr. 4, pp. 1-10, 01 07 2012.

8 Anhang

8.1 Code

Code für das Training des Neuronales Netzwerks

```
# Copyright (c) 2025 Peter Hanna.
# All rights reserved.
# This code was created as part of a diploma thesis and
may not be copied, modified, or distributed. Scientific
use is exempt, provided that proper citation is given.

"""
Neural Network Training.
    Inputs:
        NNL: Neural Network Layout.
        MIt: Maximum Iterations (used with Threshold).
        It: Iterations; Number of training passes.
        Th: Success threshold (between 0 and 1); If
        used, iterations will be omitted.
        Lr: Learning rate (between 0 and 1); Determines
        the step size for correction in each iteration.
        V: Values for training.
        L: Labels for training.
        B: Boolean flag for start.
    Outputs:
        NN: Trained neural network.
        Its: Number of iterations performed.
        ALs: Average Loss from each Iteration.
        Rs: Results of each Iteration.
"""

# Rhino-Version = 7 SR37
# Grasshopper-Version = 1.0.0007

__author__ = "Peter Hanna"
__version__ = "2024.05.01"

import math
import random
import Grasshopper.Kernel.Data.GH_Path as ghpath


def NNLayout(NNL):
    layer_sizes = NNL # Number of neurons in all layers
    (Neural Network Layout)
    #input_size = NNL[0] # Number of neurons in the input
    layer
```

```
    #hidden_sizes = NNL[1:-1] # Number of neurons in the
    hidden layers
    output_size = layer_sizes[-1] # Number of neurons in
    the output layer

    return layer_sizes, output_size



def data_preparation(V, L, output_size):
    # Convert Grasshopper tree to a Python tree (lists of
    lists)
    # Tree of values:
    alphas = []
    for i in range(V.BranchCount-1):
        if str(V.Path(i)[1]) == str(0):
            alphas.append(i)
    alphas.append(V.BranchCount)

    V_tree = [] # Tree of inputs
    for i in range(len(alphas) - 1):
        V_brach = []
        for j in range(alphas[i+1] - alphas[i]):
            V_list = []
            for k in range(V.Branch(i).Count):
                V_list.append(V[ghpath(i, j), k])
            V_brach.append(V_list)
        V_tree.append(V_brach)


    # Tree of labels:
    alphas = []
    for i in range(L.BranchCount-1):
        if str(L.Path(i)[1]) == str(0):
            alphas.append(i)
    alphas.append(L.BranchCount)

    L_tree = [] # Tree of labels
    for i in range(len(alphas) - 1):
        L_brach = []
        for j in range(alphas[i+1] - alphas[i]):
            L_list = []
            for k in range(L.Branch(i).Count):
                L_list.append(L[ghpath(i, j), k])
            L_brach.append(L_list)
        L_tree.append(L_brach)



    # Create one-hot vectors:
    vectors = []
```

```
094         o_z = output_size
095         for i in range(o_z):
096               vector = [0] * o_z
097               vector[i] = 1
098               vectors.append(vector)
099         targets = vectors
100
101         return V_tree, L_tree, targets
102
103
104
105
106   def create_parameters(layer_sizes):
107         # Set limits for Xavier initialization:
108         n_in = layer_sizes[0] # Number of neurons in input
109         layer
110         n_out = layer_sizes[-1] # Number of neurons in output
111         layer
112         limit = math.sqrt(6 / (n_in + n_out)) # Limits for
113         domain; # Xavier initialization
114         #limit = math.sqrt(2 / n_in)  # He initialization
115         #limit = math.sqrt(4 / (n_in + n_out))  # Hybrid
116         initialization
117
118         # Create Weights
119         weights = []
120         for ls in range(len(layer_sizes)-1):
121               weights_a = []
122               for i in range(int(layer_sizes[ls + 1])):
123                     weights_b = []
124                     for j in range(int(layer_sizes[ls])):
125                           #weights_b.append(random.random()) #
126                           Create random biases between 0 - 1
127                           weights_b.append(random.uniform(-
128                           limit, limit))
129                     weights_a.append(weights_b)
130               weights.append(weights_a)
131
132
133         # Create Biases
134         biases = []
135         for ls in range(len(layer_sizes)-1):
136               biases_a = []
137               for i in range(int(layer_sizes[ls + 1])):
138                     #biases_a.append(random.random()) # Create
139                     random biases between 0 - 1
140                     biases_a.append(0) # Set bias to 0 or
141                     0.01; biases shift neuron activation levels
142                     - which is not necessary
143               biases.append(biases_a)
144
```

```
145         return weights, biases
146
147
148
149
150
151     ## Activationfunction Sigmoid
152     #def sigmoid(x):
153     #    return 1 / (1 + math.exp(-x))
154     #
155     #
156     ## Derivative of Activationfunction Sigmoid
157     #def sigmoid_derivative(x):
158     #    return x * (1 - x) # Since only outputs get in x, the
159         derivation is correct, otherwise it would be: sig(z) *
160     (1 - sig(z))
161
162
163     ## Activationfunction ReLU
164     #def relu(x):
165     #    return max(0, x)
166     #
167     ## Derivative of Activationfunction ReLU
168     #def relu_derivative(x):
169     #    if x > 0:
170     #        return 1
171     #    else:
172     #        return 0
173
174
175
176     def leaky_relu(x, alpha=0.01):
177         # Activationfunction Leaky ReLU
178         if x > 0:
179             return x
180         else:
181             return alpha * x
182
183
184     def leaky_relu_derivative(x, alpha=0.01):
185         # Derivative of Activationfunction Leaky ReLU
186         return 1 if x > 0 else alpha
187         if x > 0:
188             return 1
189         else:
190             return alpha
191
192
193
194
195     def softmax(x):
```

```
196       # Activationfunction Softmax
197       exps = []
198       for i in x:
199             exps.append(math.exp(i))
200             sum_exps = sum(exps)
201
202       probabilities = []
203       for j in exps:
204             probabilities.append(j / sum_exps)
205
206       return probabilities
207
208
209
210 def cross_entropy_loss(predictions, targets):
211       # Cross Entropy Loss Funktion
212       loss = 0
213       for i in range(len(predictions)):
214             loss = loss + targets[i] *
215             math.log(predictions[i])
216       return -loss
217
218
219
220 def cross_entropy_loss_derivative(predictions, targets):
221       # Derivative of Cross Entropy Loss Funktion
222       dloss = []
223       for i in range(len(predictions)):
224             dloss.append(predictions[i] - targets[i])
225       return dloss
226
227
228
229
230 def forward_propagation(inputs, weights, biases):
231       # Calculate output of neurons
232       outputs = []
233       neuron_inputs = []
234       neuron_inputs.append(inputs)
235
236       for i in range(len(layer_sizes)-1):
237             outputs_a = []
238             z = []
239
240             if i == 0:
241                   inp = inputs
242             else: # if i > 0
243                   inp = outputs[i - 1]
244
245             for j in range(int(layer_sizes[i + 1])):
246                   outputs_b = []
```

```
                for k in range(int(layer_sizes[i])):
                    o = inp[k] * weights[i][j][k]
                    outputs_b.append(o)

                s = sum(outputs_b) + biases[i][j]
                z.append(s)

                if i < len(layer_sizes)-2:
                    #for s in z:
                    #outputs_a.append(sigmoid(s)) # Use
                    sigmoid activation for all neurons
                    before the output neuron
                    #outputs_a.append(relu(s)) # Use ReLU
                    activation for all neurons before the
                    output neuron
                    outputs_a.append(leaky_relu(s)) # Use
                    Leaky ReLU activation for all neurons
                    before the output neuron
            if i == len(layer_sizes)-2:
                outputs_a = softmax(z) # Use Softmax
                activation for output neuron

            outputs.append(outputs_a)
            neuron_inputs.append(outputs_a)


    return outputs



def backward_propagation(outputs, inputs, targets):
    # Calculate errors for neurons in each layer
    n_errors = []

    output_pred = outputs[-1]
    dloss = cross_entropy_loss_derivative(output_pred,
    targets)

    for i in reversed(range(len(outputs))):

            if i == len(outputs) - 1: # Error for the first
            layer of neurons (output layer)
                neuron_layer_errors = []
                for j in range(len(outputs[i])):
                    neuron_layer_errors.append(dloss[j])
                n_errors.append(neuron_layer_errors)

            else: # Error for neurons in hidden layers
                neuron_layer_errors = []
                for j in range(len(outputs[i])):
                    neuron_errors = []  # List for delta
```

```
                    values of the next layer
                    for k in range(len(outputs[i + 1])):
                    # Delta values for each neuron in the
                    next layer
                        neuron_errors.append(n_errors[-
                        1][k] * weights[i + 1][k][j])
                    s = sum(neuron_errors)
                    #neuron_layer_errors.append(s *
                    sigmoid_derivative(outputs[i][j]))
                    #neuron_layer_errors.append(s *
                    relu_derivative(outputs[i][j]))
                    neuron_layer_errors.append(s *
                    leaky_relu_derivative(outputs[i][j]))
                n_errors.append(neuron_layer_errors)

    n_errors.reverse()



    # Calculate errors for weights and biases in each
    layer
    w_errors = []
    b_errors = []
    for i in range(len(weights)):
        hidden_w_error = []
        hidden_b_error = []

        if i > 0:
            inp = outputs[i - 1]

        else: # if i == 0
            inp = inputs

        for j in range(len(weights[i])):
            hidden_w_error_a =[]
            for k in range(len(weights[i][j])):

                hidden_w_error_a.append(n_errors[j][i]
                * inp[k])

            hidden_w_error.append(hidden_w_error_a)

        w_errors.append(hidden_w_error)

    b_errors = n_errors
    return weights, biases, w_errors, b_errors



def update(weights, biases, w_errors, b_errors):
    # Update weights and biases
```

```
    for i in range(len(weights)):
        for j in range(len(weights[i])):
            for k in range(len(weights[i][j])):
                weights[i][j][k] = weights[i][j][k] -
                Lr * w_errors[i][j][k]
            biases[i][j] = biases[i][j] - Lr *
            b_errors[i][j]
    return weights, biases




# Prepare NNL: Neural Network Layout
layer_sizes, output_size = NNLayout(NNL)

# Prepare data
V_tree, L_tree, targets = data_preparation(V, L,
output_size)

# Create parameters
weights, biases = create_parameters(layer_sizes)

# Lists for each iteration from tree
results = []
all_loss = []
all_av_loss = []
cancel = False


# MIt, Threshold and max_results parameters
max_it = MIt
if Th: #Th = threshold
    count = max_it # If threshold is defined, use max_it
    as iteration count
else:
    count = It # Otherwise, use the predefined iteration
    count

max_results = 3 # Limit stored results to avoid very high
RAM usage

# Train each list from tree
for iteration in range(count):
    result_a = []
    loss = []

    for i in range(len(V_tree)):
        result_b = []

        for j in range(len(V_tree[i])):
            # Forward Propagation
```

```
                outputs =
                forward_propagation(V_tree[i][j], weights,
                biases)

                # Check for early stopp
                if Th:
                      Its = iteration
                      if iteration > 5:
                            if max(all_loss[-1]) < 1 - Th:
                            # or av_loss < 1 - Th:
                                  cancel = True
                                  break


                # Backward Propagation
                weights, biases, w_errors, b_errors =
                backward_propagation(outputs, V_tree[i][j],
                targets[i])


                # Updates
                weights, biases = update(weights, biases,
                w_errors, b_errors)

                if cancel:
                      break

                # Results
                result_b.append(outputs[-1]) # Result from
                Outputneurons

          if cancel:
                break

          loss.append(cross_entropy_loss(outputs[-1],
          targets[i])) # append
          av_loss = sum(loss)/len(loss)
          result_a.append(result_b)

    if cancel:
          break

    # To save memory
    if len(results) > max_results:
          results.pop(0) # Remove the first (oldest)
          entry

    results.append(result_a) # Store current entry
    all_loss.append(loss)
    all_av_loss.append(av_loss)
    ALs = all_av_loss
```

```
451         Rs = results
452
453
454     # Export NN data for Prediction:
455     NN = [weights, biases, layer_sizes, all_loss, results]
```

Code für die Vorhersage des neuronalen Netzwerks

```
001     # Copyright (c) 2025 Peter Hanna.
002     # All rights reserved.
003     # This code was created as part of a diploma thesis and
004     may not be copied, modified, or distributed. Scientific
005     use is exempt, provided that proper citation is given.
006
007     """
008     Neural Network Prediction.
009         Inputs:
010               NN: Trained Neural Network
011               TV: Test values
012         Output:
013               P: Prediction of Test
014               MPI: Maximum prediction index
015               L: Loss
016               R: Results
017     """
018
019     # Rhino-Version = 7 SR37
020     # Grasshopper-Version = 1.0.0007
021
022     __author__ = "Peter Hanna"
023     __version__ = "2024.05.30"
024
025
026     import math
027
028
029
030     def sigmoid(x):
031         # Activationfunction Sigmoid
032         return 1 / (1 + math.exp(-x))
033
034
035
036     def leaky_relu(x, alpha=0.01):
037         # Activationfunction Leaky ReLU
```

```
038         if x > 0:
039               return x
040         else:
041               return alpha * x
042
043
044   def softmax(x):
045         # Activationfunction Softmax
046         exps = []
047         for i in x:
048               exps.append(math.exp(i))
049         sum_exps = sum(exps)
050
051         probabilities = []
052         for j in exps:
053               probabilities.append(j / sum_exps)
054
055         return probabilities
056
057
058
059   def forward_propagation(inputs, weights, biases):
060         # Calculate output of neurons
061         outputs = []
062         neuron_inputs = []
063         neuron_inputs.append(inputs)
064
065         for i in range(len(layer_sizes)-1):
066               outputs_a = []
067               z = []
068
069               if i == 0:
070                     inp = inputs
071               else: #if i > 0:
072                     inp = outputs[i - 1]
073
074               for j in range(int(layer_sizes[i + 1])):
075                     outputs_b = []
076                     for k in range(int(layer_sizes[i])):
077                           o = inp[k] * weights[i][j][k]
078                           outputs_b.append(o)
079
080                     s = sum(outputs_b) + biases[i][j]
081                     z.append(s)
082
083                     if i < len(layer_sizes)-2:
084                           #for s in z:
085                           #outputs_a.append(sigmoid(s)) # Use
086
087                           sigmoid activation for all neurons
088
```

```
                        before the output neuron
                        #outputs_a.append(relu(s))  # Use ReLU

                        activation for all neurons before the

                        output neuron
                        outputs_a.append(leaky_relu(s))  # Use

                        Leaky ReLU activation for all neurons

                        before the output neuron
            if i == len(layer_sizes)-2:
                outputs_a = softmax(z) # Use Softmax

                activation for output neuron

            outputs.append(outputs_a)
            neuron_inputs.append(outputs_a)


    return outputs



# Import data:
weights = NN[0]
biases = NN[1]
layer_sizes = NN[2]
#all_loss = NN[3]
#results = NN[4]

#L = all_loss
#R = results


# Test: prediction for test values
test_input = TV # Test values
output = forward_propagation(test_input, weights, biases)
P = output[-1] # Prediction


# Index of highest prediction probability
MP = max(P) # Maximum prediction
MPI = P.index(MP) # Maximum prediction index
```

Code für den Transfer

```
# Copyright (c) 2025 Peter Hanna.
# All rights reserved.
# This code was created as part of a diploma thesis and
may not be copied, modified, or distributed. Scientific
use is exempt, provided that proper citation is given.

"""
Transfer Data.
Provides a scripting component.
    Inputs:
        B: Start Boolean
    Output:
        TB: Boolean for transfer completion
        """

__author__ = "Peter Hanna"
__version__ = "2024.10.09"

import Rhino
import Grasshopper.Kernel as ghk
import System.Drawing


ghDoc = ghenv.Component.OnPingDocument()  # Reference to
Grasshopper document

def find_component(component_name):
    for obj in ghDoc.Objects:
        if obj.NickName == component_name:
            return obj
    return None


# Find Komponents
entwine_component = find_component("Entwine_Containers")
drawing_component = find_component("Drawing_PL&Crvs")


if B == True:
    # Find the first available input, else create one
    new_input_required = False
    for i in range(len(entwine_component.Params.Input)):
        if len(entwine_component.Params.

        Input[i].Sources) == 0: # Number of input

        connections
            free_input_index = i
            break
```

```
050         else:   # Add an input parameter
051             new_input_required = True
052             new_param = ghk.Parameters.Param_GenericObject()
053
054             # Create a new input parameter
055             last_index = len(entwine_component.Params.Input)
056             new_param.NickName = "{0;" + str(
057
058             last_index) + "}"
059             new_param.Description  = "Data to entwine"
060             new_param.Access = ghk.GH_ParamAccess.tree  #
061
062             Set access to "Tree"
063
064
065
066             entwine_component.Params.RegisterInputParam(
067
068             new_param)  #
069
070             Register the new parameter
071             entwine_component.Params.OnParametersChanged() #
072
073             Update the parameters
074             free_input_index = last_index # Set
075
076             free_input_index to the new last parameter
077
078
079             # Use the position of the available input to
080
081             place the Point-Container
082         if free_input_index != None:
083             # Create the new Curve-Container
084             curve_param = ghk.Parameters.Param_Curve()
085             ghDoc.AddObject(curve_param, False) # Add the
086
087             new Point-Container
088             curve_param.Hidden = True # Disable preview
089
090             # Define offset for positioning the new
091
092             container
093             if new_input_required == False:
094                 offset_x = -100 # Horizontal offset to the
095
096                 left
097                 offset_y = 0  # Vertical offset
098             elif new_input_required == True:
099                 offset_x = -100  # Horizontal offset to the
100
101                 left
102                 offset_y = 15 *
```

```
            len(entwine_component.Params.Input)  #
            Incremental vertical offset

        # Position the new container relative to the
        input pivot
        entwine_component.Attributes.PerformLayout() #
        Execute layout
        input_position = entwine_component.Params.Input
        [free_input_index].Attributes.Pivot
        new_position = System.Drawing.PointF(
        input_position.X + offset_x, input_position.Y +
        offset_y)
        curve_param.Attributes.Pivot = new_position #
        Set the position of the containers

        # Connect free input/output
        free_input_param =
        entwine_component.Params.Input[free_input_index]
        free_input_param.AddSource(curve_param)  #
        Connect the new container to the free input
        curve_param.AddSource(drawing_component)  #
        Connect the input of the new container to the
        output of Drawing_PL&Crvs

        # Transfer data
        drawing_component_data =
        drawing_component.VolatileData.AllData(True) #
        PersistentData / VolatileData



        curve_param.PersistentData.AppendRange(
        drawing_component_data)

        # Remove input connection
        curve_param.RemoveSource(drawing_component)
        curve_param.ExpireSolution(False)

        TB = True
```

8.2 Protokoll

Protokoll der manuellen Optimierung

L	V	D	M	HL1	HL2	Lr	Th	Mlt	C	T	A	Z
27	2	324	20	40	0	0,02	0,9	200	76	2	0,235	380
27	2	324	40	40	0	0,02	0,9	200	95	2,6	0,293	365
27	2	324	60	40	0	0,02	0,9	200	126	3,1	0,389	406
27	2	324	80	40	0	0,02	0,9	200	99	3,6	0,306	275
27	2	324	100	40	0	0,02	0,9	200	122	4,1	0,377	298
27	2	324	120	40	0	0,02	0,9	200	133	4,6	0,410	289
27	2	324	140	40	0	0,02	0,9	200	124	5,2	0,383	238

L	V	D	M	HL1	HL2	Lr	Th	Mlt	C	T	A	Z
27	2	324	60	40	0	0,02	0,9	200	126	3,1	0,389	406
27	2	324	60	20	20	0,02	0,9	200	97	2,2	0,299	441
27	2	324	60	40	40	0,02	0,9	200	104	5	0,321	208

L	V	D	M	HL1	HL2	Lr	Th	Mlt	C	T	A	Z
27	2	324	60	20	0	0,02	0,9	200	126	1,6	0,389	788
27	2	324	60	30	0	0,02	0,9	200	122	2,3	0,377	530
27	2	324	60	40	0	0,02	0,9	200	125	3	0,386	417
27	2	324	60	50	0	0,02	0,9	200	125	3,7	0,386	338
27	2	324	60	60	0	0,02	0,9	200	124	4,4	0,383	282

L	V	D	M	HL1	HL2	Lr	Th	Mlt	C	T	A	Z
27	2	324	60	20	0	0,02	0,9	200	126	1,6	0,389	788
27	2	324	60	20	0	0,02	0,9	300	134	2,3	0,414	583
27	2	324	60	20	0	0,02	0,9	400	142	3,2	0,438	444
27	2	324	60	20	0	0,02	0,9	500	142	4	0,438	355
27	2	324	60	20	0	0,02	0,9	1000	156	7,9	0,481	197

L	V	D	M	HL1	HL2	Lr	Th	Mlt	C	T	A	Z
27	2	324	60	20	0	0,01	0,9	400	129	3,1	0,398	416

27	2	324	60	20	0	0,02	0,9	400	142	3,1	0,438	458
27	2	324	60	20	0	0,05	0,9	400	137	3,1	0,423	442
27	2	324	60	20	0	0,1	0,9	400	139	3,1	0,429	448
27	2	324	60	20	0	0,2	0,9	400	133	3,1	0,410	429

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	2	324	60	20	0	0,02	0,8	400	135	3	0,417	450
27	2	324	60	20	0	0,02	0,85	400	135	3	0,417	450
27	2	324	60	20	0	0,02	0,9	400	142	3,1	0,438	458
27	2	324	60	20	0	0,02	0,95	400	135	3,1	0,417	435
27	2	324	60	20	0	0,02	0,98	400	136	3,2	0,420	425

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	2	324	60	20	0	0,02	0,9	400	142	3,1	0,438	458
27	3	486	60	20	0	0,02	0,9	400	188	4,6	0,387	409
27	4	648	60	20	0	0,02	0,9	400	245	6	0,378	408

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	4	648	60	1	0	0,02	0,9	400	94	1	0,145	940
27	4	648	60	2	0	0,02	0,9	400	169	1,2	0,261	1408
27	4	648	60	3	0	0,02	0,9	400	222	1,5	0,343	1480
27	4	648	60	5	0	0,02	0,9	400	238	2	0,367	1190
27	4	648	60	10	0	0,02	0,9	400	246	3,4	0,380	724
27	4	648	60	15	0	0,02	0,9	400	242	4,8	0,373	504
27	4	648	60	20	0	0,02	0,9	400	245	6	0,378	408
27	4	648	60	25	0	0,02	0,9	400	247	7,4	0,381	334
27	4	648	60	10	0	0,02	0,9	400	153	3,5	0,236	437
27	4	648	60	10	0	0,02	0,9	100	131	1	0,202	1310
27	4	648	60	20	0	0,02	0,9	100	188	1,6	0,290	1175
27	4	648	60	30	0	0,02	0,9	100	135	2,3	0,208	587
27	4	648	60	20	0	0,005	0,9	100	133	1,6	0,205	831
27	4	648	60	20	0	0,005	0,9	100	168	1,7	0,259	988
27	4	648	60	20	0	0,005	0,9	200	181	3,2	0,279	566
27	4	648	60	20	0	0,005	0,9	200	117	3,3	0,181	355

27	4	648	60	20	0	0,005	0,9	200	216	3,1	0,333	697
27	4	648	60	20	0	0,005	0,9	200	157	3,1	0,242	506

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	4	648	60	3	0	0,02	0,9	800	227	2,9	0,350	783
27	4	648	60	5	0	0,02	0,9	800	251	4	0,387	628
27	4	648	60	10	0	0,02	0,9	800	267	6,6	0,412	405
27	4	648	60	15	0	0,02	0,9	800	270	9,6	0,417	281
27	4	648	60	20	0	0,02	0,9	800	268	12,1	0,414	221
27	4	648	60	25	0	0,02	0,9	800	262	14,7	0,404	178

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	4	648	60	5	0	0,02	0,9	1500	256	7,4	0,395	346
27	4	648	60	10	0	0,02	0,9	1500	292	12,5	0,451	234
27	4	648	60	15	0	0,02	0,9	1500	293	17,7	0,452	166
27	4	648	60	20	0	0,02	0,9	1500	300	23,2	0,463	129

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	2	324	60	20	0	0,002	0,9	500	110	4	0,340	275
27	2	324	60	20	0	0,002	0,9	1000	113	7,7	0,349	147

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	10	1620	60	15	0	0,02	0,9	400	600	11,7	0,370	513
27	10	1620	60	20	0	0,02	0,9	400	578	15,3	0,357	378
27	10	1620	60	15	0	0,02	0,9	800	654	24	0,404	273
27	10	1620	60	20	0	0,02	0,9	800	676	30	0,417	225
27	10	1620	60	15	0	0,02	0,9	1200	741	35,6	0,457	208
27	10	1620	60	20	0	0,02	0,9	1200	761	40	0,470	190
27	10	1620	60	15	0	0,02	0,9	3000	771	101,3	0,476	76

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	10	1620	60	15	0	0,02	0,9	400	600	11,7	0,370	513

27	10	1620	60	15	0	0,01	0,9	400	605	11,7	0,373	517
27	10	1620	60	15	0	0,005	0,9	400	631	11,7	0,390	539

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	10	1620	60	15	0	0,02	0,9	3000	771	101,3	0,476	76
27	10	1620	60	15	0	0,005	0,9	3000	764	101,3	0,472	75

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
27	10	1620	60	15	15	0,005	0,9	3000	751	101,3	0,464	74
18	10	1080	60	15	0	0,005	0,9	800	545	13,8	0,505	395

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
18	30	3240	60	15	0	0,005	0,9	400	1459	20,9	0,450	698
18	30	3240	60	15	0	0,005	0,9	800	1639	40,9	0,506	401
18	30	3240	60	15	0	0,005	0,9	1200	1659	60	0,512	277
18	30	3240	60	20	0	0,005	0,9	400	1478	25,9	0,456	571
18	30	3240	60	50	0	0,005	0,9	400	1273	40	0,393	318
18	30	3240	60	15	0	0,002	0,9	400	1500	19,9	0,463	754
18	60	6480	60	15	0	0,002	0,9	400	2995	39	0,462	768

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
18	30	3240	60	15	0	0,005	0,9	400	1236	20,6	0,381	600
18	30	3240	60	15	0	0,005	0,9	400	1124	20,8	0,347	540
18	30	3240	60	15	0	0,005	0,9	400	1152	20,5	0,356	562
18	30	3240	60	15	0	0,005	0,9	400	1317	20,3	0,406	649
12	30	2160	60	15	0	0,005	0,9	100	1309	3	0,606	4363
12	30	2160	60	15	0	0,005	0,9	200	1282	3,7	0,594	3465
12	30	2160	60	15	0	0,005	1	200	1297	5,9	0,600	2198
12	30	2160	60	15	0	0,005	1	400	1351	12,1	0,625	1117

12	30	2160	60	15	0	0,01	1	200	1344	5,9	0,622	2278
12	40	2880	60	15	0	0,01	1	200	1645	8	0,571	2056
12	60	4320	60	15	0	0,01	1	800	2239	50	0,518	448

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	60	4320	60	15	0	0,01	1	200	2381	12,1	0,551	1968
12	60	4320	60	15	0	0,002	1	200	2740	12,2	0,634	2246
12	60	4320	60	15	0	0,001	1	200	2935	12,1	0,679	2426
12	60	4320	60	15	0	0,0005	1	200	2362	12,2	0,547	1936

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	60	4320	60	15	0	0,01	1	400	2450	24,5	0,567	1000
12	60	4320	60	15	0	0,001	1	400	2714	24,7	0,628	1099
12	60	4320	60	15	0	0,005	1	400	2632	24,7	0,609	1066
12	60	4320	60	15	0	0,0005	1	400	2626	24,3	0,608	1081

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	60	4320	60	15	0	0,001	1	400	2714	24,7	0,628	1099
12	100	7200	60	15	0	0,001	1	400	4824	40,6	0,670	1188

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	30	2160	60	15	0	0,001	1	200	1415	6,4	0,655	2211
12	30	2160	60	10	0	0,001	1	200	1327	4,4	0,614	3016
12	30	2160	60	20	0	0,001	1	200	1305	8,2	0,604	1591

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	30	2160	60	15	0	0,001	1	200	1415	6,4	0,655	2211
12	30	2160	60	15	0	0,001	1	400	1442	12,6	0,668	1144

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	15	2160	60	30	0	0,001	1	800	1485	24,2	0,688	614
12	30	2160	60	30	0	0,001	1	800	1542	46,2	0,714	334
12	50	2160	60	30	0	0,001	1	800	1507	69	0,698	218

L	V	D	M	HL1	HL2	Lr	Th	MIt	C	T	A	Z
12	15	2160	80	15	0	0,001	1	200	1329	7,3	0,615	1821
12	15	2160	80	15	0	0,001	1	400	1440	14,8	0,667	973
12	15	2160	120	15	0	0,001	1	400	1395	20,2	0,646	691
12	30	2160	120	30	0	0,001	1	400	1542	46,2	0,714	334

Dabei gilt:

L = Anzahl der variablen Labels

V = Varianten pro Figur

D = Gesamtanzahl aller Figuren einschließlich Variationen

M = Merkmale: Attribute pro Figur

HL1 = Anzahl der Neuronen in der versteckten Schicht 1

HL2 = Anzahl der Neuronen in der versteckten Schicht 2

Lr = Lernrate

Th = Schwellenwert in %

MIt = Maximale Iterationsanzahl

C = Anzahl der korrekten Vorhersagen

T = Berechnungszeit in Minuten

A = Genauigkeit der korrekten Vorhersagen in %

Z = Fitness-Ziel: $Z = \frac{C \cdot F}{T}$

F = Gewichtungsfaktor (10)

8.3 Fragebogen

Fragebogen

Bitte beantworten Sie die untenstehenden Fragen, die sich auf den vorgestellten Algorithmus beziehen.

Datum:

Persönliche Daten

Anonym ☐ oder Name:

Studium (nur eine Auswahl): Derzeit Bachelorstudium Architektur ☐
Derzeit Masterstudium Architektur ☐
Abgeschlossenes Bachelorstudium Architektur ☐
Abgeschlossenes Masterstudium Architektur ☐

Grasshopper Erfahrung: Ja ☐
Nein ☐

1. Verständlichkeit

Wie beurteilen Sie die Dauer, die Sie für den Einstieg in die Nutzung des Algorithmus benöt gt haben?

Sehr schlecht | ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 | Sehr gut

Wie nachvollziehbar war der Zweck des Algorithmus für Sie?

Sehr schlecht | ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 | Sehr gut

2. Benutzerfreundlichkeit

Wie intuitiv und benutzerfreundlich empfanden Sie die Nutzung des Algorithmus?

Sehr schlecht | ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 | Sehr gut

Wie überzeugend fanden Sie die Funktion und Ergebnisse des Algorithmus?

Sehr schlecht | ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 | Sehr gut

3. Praxistauglichkeit

Würden Sie den Algorithmus in Ihrem Arbeitskontext (Studium oder Beruf) einsetzen?

Sehr schlecht | ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 | Sehr gut

Wie gut lässt sich der Algorithmus aus Ihrer Sicht in einen Arbeitsablauf integrieren?

Sehr schlecht | ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 | Sehr gut