**Universität Passau**

Institut für Softwaresysteme in technischen Anwendungen der Informatik (FORWISS Passau)

Fakultät für Informatik und Mathematik

Innstrasse 43

94032 Passau


Masterarbeit

# Lernen von Aufnahmeparametern für CT-Messungen

*Alexander Brunner*

Erstprüfer: Prof. Dr. Tomas Sauer

Zweitprüfer: Prof. Dr. Michael Granitzer

28. September 2019

# Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift

# Kurzfassung

Zur Vorhersage der optimalen Maschinenparameter bei dreidimensionalen computertomographischen Scans soll im Rahmen dieser Arbeit ein neuronales Netz vorgestellt werden. Das neuronale Netz soll die Parameter anhand von drei Projektionen des gescannten Objekts vorhersagen können. Dazu werden in dieser Arbeit Convolutional Neural Networks eingesetzt. Diese stellen den Standard dar, wenn es um Bildverarbeitung mit Hilfe von Machine Learning Algorithmen geht.

Das Hauptaugenmerk dieser Arbeit liegt auf der experimentellen Erstellung eines Convolutional Neural Networks, welches unter Berücksichtigung von Bilddateien eines Objekts und Kombinationen der Maschinenparameter die zu erwartende, minimale Abweichung zwischen dem Istwert und Sollwert einer Messung voraussagt. Diese Arbeit beschränkt sich auf eine Teilmenge der zur Verfügung stehenden Objekte.

Mit der Python Bibliothek TensorFlow und der darauf aufsetzenden API Keras wurde begonnen ein Multilayer Perceptron zu entwickeln, das die grundlegende Funktionalität zur Verarbeitung der Maschinenparameter ermöglicht. Dieses Modell wurde mit zusätzlichen Parametern erweitert. Es wurden ein serielles und ein paralleles Covolutional Neural Network Modell erstellt, welche die Verarbeitung der Bilddateien übernehmen. Im nächsten Schritt wurden das Multilayer Perceptron und die Convolutional Neural Networks zusammengefügt. Die erstellten Modelle wurden mit Testdaten evaluiert und es konnte festgestellt werden, dass die Modelle in der Lage sind die Messabweichung mit großer Genauigkeit vorherzusagen. Mit einem abschließenden Test wurde versucht die optimalen Maschinenparameter zu bestimmen, bei dem sich die Modelle sehr gut an die tatsächlichen Parameter angenähert haben.

# Abstract

In this assignment a neural network for the prediction of machine parameters, which are related to three dimensional computer tomography scans, is discovered and evlaluated. The neural network should predict the parameters depending on projections of a measured object. For this purpose convolutional neural networks should be used. These networks are state-of-the-art in case of image processing together with machine learning algorithms.

The primary focus of this work is the experimental development of a convolutional neural network, which is able to predict the minimum difference between the acutal value and the setpoint value of a measurement. It should regard the machine parameters of the measurement as well as the corresponding image pojections. This work is restricted to a subset of the available objects.

A multilayer perceptron model was developed with the use of the Python library TensorFlow and the API Keras, which enables the neural net to process the machine parameters. This model was enhanced with additional parameters. A serialized as well as a parallel model of convolutional neural network were desgined, which are responsible for the image processing. In a further step the multilayer perceptron and the convolutional neural networks were concatenated. The resulting models were evaluated with test data. It could be recognized that the models were able to predict the measurement difference with high accuracy. In a final test the models were used to predict the best machine parameters of a measurement. As a result, the models could approximate the real parameters very well.

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| 3D-CT | 3D-Röntgen-Computertomographie |
| ADALINE | Adaptive Linear Neuron |
| Adam | Adaptive Moment Estimation |
| API | Application Programming Interface |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CT | Computertomograph |
| ELU | Exponential Linear Unit |
| GPU | Graphical Processing Unit |
| ILSVRC | ImageNet Large Scale Visual Recognition Challenge |
| IQA | Interquartilsabstand |
| KNN | Künstliches Neuronales Netz |
| MAE | Mean Absolute Error |
| MLP | Multilayer Perceptron |
| MSE | Mean Squared Error |
| MSLE | Mean Squared Logarithmic Error |
| ReLU | Rectified Linear Unit |
| ResNet | Residual Neural Network |
| RMSE | Root Mean Squared Error |
| SGD | Stochastic Gradient Descent |
| SLP | Single Layer Perceptron |
| tanh | Tangens hyperbolicus |
| VGG | Visual Geometry Group |

# Symbolverzeichnis

| | |
|---|---|
| $\eta$ | Lernrate |
| $\mu$ | Erwartungswert |
| $\nabla J_\theta(\theta)$ | Gradient der Kostenfunktion in Anhängigkeit der Modellparameter |
| $\sigma$ | Standardabweichung |
| $\theta$ | Modellparameter |
| $F$ | Größe des Filters |
| $P$ | Zero-Padding Parameter |
| $S$ | Schrittweite |
| $W$ | Breite |
| $w_i$ | Gewicht einer Verbindung zwischen zwei Neuronen |
| $\text{Error}_{abs}$ | Absolutbetrag der Differenz zwischen Ist- und Sollwert |
| IQA | Interquartilsabstand |
| M | Masse |
| p | Grauwert eines Pixels |
| $Q_3$ | Drittes Quartil |
| X | Zufallsvariable |
| Z | Standardisierte Zufallsvariable |

# 1 Einleitung

Die 3D-Röntgen-Computertomographie (3D-CT) ist eines der berührungslosen und zerstörungsfreien Standardprüfverfahren, welches zur Bestimmung der inneren Materialeigenschaften und den geometrischen Merkmalen von Bauteilen oder Baugruppen eingesetzt wird. Dazu rotiert das zu prüfende Objekt in einem Messgerät, während es mit Röntgenstrahlungen durchleuchtet wird. Aus den aufgenommenen Röntgenbildern bzw. Projektionen wird mit Hilfe von spezieller Software ein 3D-Volumenmodell rekonstruiert. In Abbildung 1.1 ist der Aufbau der 3D-CT Messung und das Ergebnis einer 3D-Rekonstruktion beispielhaft dargestellt. Dieses Verfahren dient zum einen zur Qualitätssicherung fertig produzierter Bauteile, wird aber auch zur Überprüfung von Prototypen angewendet. Es können so zum Beispiel Hohlräume oder Lufteinschlüsse in Bauteilen festgestellt werden. Um eine möglichst genaue 3D-Rekonstruktion eines Objekts erstellen zu können, ist es notwendig, dass die Parameter, welche abhängig vom zu vermessenden Bauteil sind, korrekt am Messsystem eingestellt werden. Diese Parameter sind Filtermaterial, Stromstärke, Spannung und Belichtungsdauer. Da es keine festen Richtlinien zum Einstellen der Maschinenparameter gibt und diese vom Maschinenbediener eingestellt werden, hängt das Messergebnis auch von der Erfahrung des Bedieners ab.

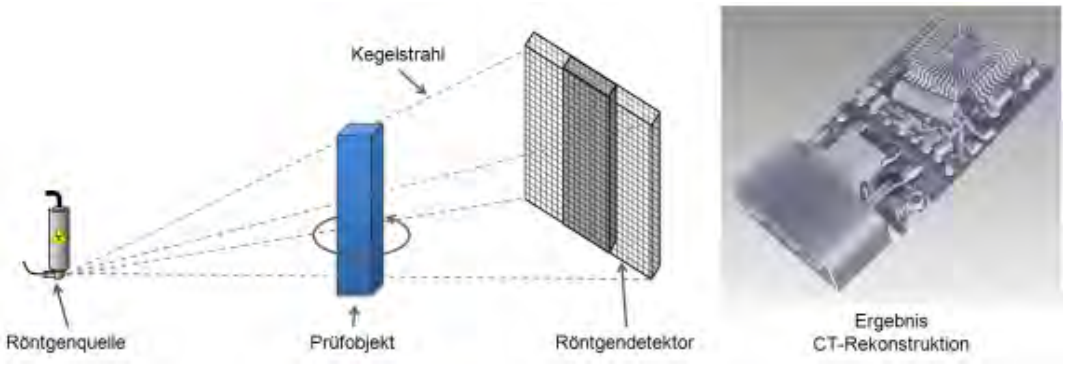



Abbildung 1.1: Beispielhafter Aufbau einer CT Messung und einer CT-Rekonstruktion

Das Fraunhofer Anwendungszentrum CT in der Messtechnik (CTMT) an der technischen Hochschule in Deggendorf beschäftigt sich mit der industriellen Röntgen-CT und hat eine Vielzahl an Messungen zu verschiedenen Objekten durchgeführt. Die Messdaten und die zugehörigen 3D-Rekonstruktionen wurden zur Verfügung gestellt, um im Rahmen dieser Arbeit zu untersuchen, ob es möglich ist anhand der Daten eine Vorhersage zu treffen, welche Maschinenparameter für eine anstehende Messung zu wählen sind. In dieser Arbeit wird versucht

mit Hilfe von Convolutional Neural Networks (CNN) unter Berücksichtigung der Parameter Filtermaterial, Stromstärke, Spannung und Belichtungsdauer, sowie den Volumenmodellen, eine Vorhersage über die zu erwartende Messabweichung durchzuführen.

Im 2. Kapitel sollen die Grundlagen von künstlichen, neuronalen Netzen und Convolutional Neural Networks erläutert werden. Dabei wird auf wichtige Parameter, welche zur Erstellung von Modellen berücksichtigt werden müssen und bereits existierende Modelle eingegangen. Im 3. Kapitel werden die entwickelten Modelle vorgestellt und deren Eigenschaften und Besonderheiten aufgezeigt. Im 4. Kapitel wird die verwendete Hardware und Software vorgestellt. Im 5. Kapitel werden die erstellten Modelle evaluiert. Abschließend wird ein Fazit gezogen und ein Ausblick hinsichtlich Erweiterung und Optimierung der Modelle vorgestellt.

# 2 Grundlagen

Im ersten Kapitel dieser Arbeit werden die Grundlagen der künstlichen neuronalen Netze aufgezeigt. Insbesondere wird hier auf einstellbaren Hyperparameter Aktivierungsfunktion, Kostenfunktion und Lernalgorithmus eines Netzes eingegangen, die beim Design von neuronalen Netzen berücksichtigt werden müssen. Außerdem werden die Besonderheiten von Convolutional Neural Networks vorgestellt. Es sollen auch die Möglichkeit zur Verwendung bestehender Algorithmen mittels Transfer Learning und die bekanntesten CNN Architekturen aufgezeigt werden.

## 2.1 Grundlagen von künstlichen neuronalen Netzen

Die Basis für ein künstliches neuronales Netz (KNN) bildet ein Neuron. Beispielhaft ist ein solches Neuron mit den zugehörigen Komponenten in Abbildung 2.1 dargestellt. Es erzeugt aus anliegenden Eingangssignalen, welche zugleich die Ausgabesignale des Vorgängerneurons sind, seinen Aktivierungszustand und gibt ein Ausgabesignal zurück. Das Ausgabesignal dient nachfolgenden Neuronen als Eingangssignal. Die Propagierungsfunktion ist linear und gibt an, wie stark das Neuron angeregt wird. Die Netzeingabe $net_j$ errechnet sich aus der Summe der Eingangssignale $o_i$ und der Verbindungsgewichte $w_{i,j}$. Mit Hilfe der Aktivierungsfunktion $f_{act,j}(net_j)$ wird ab einem gewissen Schwellwert ein Ausgabesignal $o_j$ geliefert.

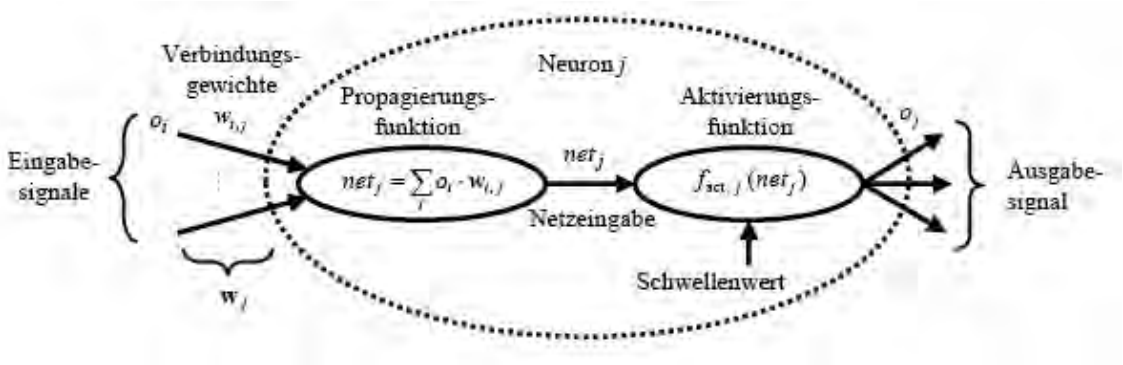



Abbildung 2.1: Datenverarbeitung in einem Neuron[1]

### 2.1.1 Die Aktivierungsfunktion

Beim Design von neuronalen Netzen muss für jedes Neuron eine Aktivierungsfunktion gewählt werden. In der Praxis wird für Neuronen einer Schicht dieselbe Funktion verwendet. Die am häufigsten verwendeten Aktivierungsfunktionen sind in Abbildung 2.2 dargestellt. Die Sigmoid Funktion stellt eine S-Kurve dar und liegt zwischen den Werten Null und Eins. Die Tangens hyperbolicus (tanh) Funktion ist ähnlich zu der Sigmoid Funktion mit dem Unterschied, dass diese Werte zwischen Null und -1 annehmen kann. Der Vorteil dieser beiden Funktionen ist deren stetige Differenzierbarkeit. Beide Funktionen haben die Nachteile, dass eine Sättigung eintritt, wenn ihre Minimal- bzw. Maximalwerte erreicht werden, was zu dem Problem des sog. verschwindenden Gradienten führt. Außerdem verursachen sie einen erhöhten Rechenaufwand. Ein weiterer Nachteil der Sigmoid Funktion ist der, dass es sich hier um keine null-zentrierte Funktion handelt. Das bedeutet, dass die Funktion nie einen negativen Wert annimmt. Die Rectified Linear Unit (ReLU) Funktion ist die standardmäßig verwendete Aktivierungsfunktion in neuronalen Netzen und hat maßgeblich zu deren Erfolg beigetragen[2]. Die ReLU Funktion ist für alle Werte größer Null die Identitätsfunktion und ansonsten Null. Der Nachteil von ReLU liegt auch darin, dass es sich um keine null-zentrierte Funktion handelt. Dafür ist sie aber sehr leicht zu berechnen und wird daher vor allem in CNNs eingesetzt. Die Leaky ReLU und Exponential Linear Unit (ELU) Funktionen sind Anpassungen der ReLU Funktion, so dass auch Werte kleiner als Null angenommen werden können. Die Maxout Funktion wird aus den gelernten Parametern berechnet[3]. In [4] ist die Maxout Funktion beschrieben. Diese stellt eine neue stückweise lineare Aktivierungsfunktion dar, die mittels Gradientenabstieg unabhängig für jedes Neuron angelernt wird. Dabei wurde gezeigt, dass diese Funktion die Performance eines neuronalen Netzes, ohne signifikante Erhöhung der Anzahl der Parameter, verbessert.

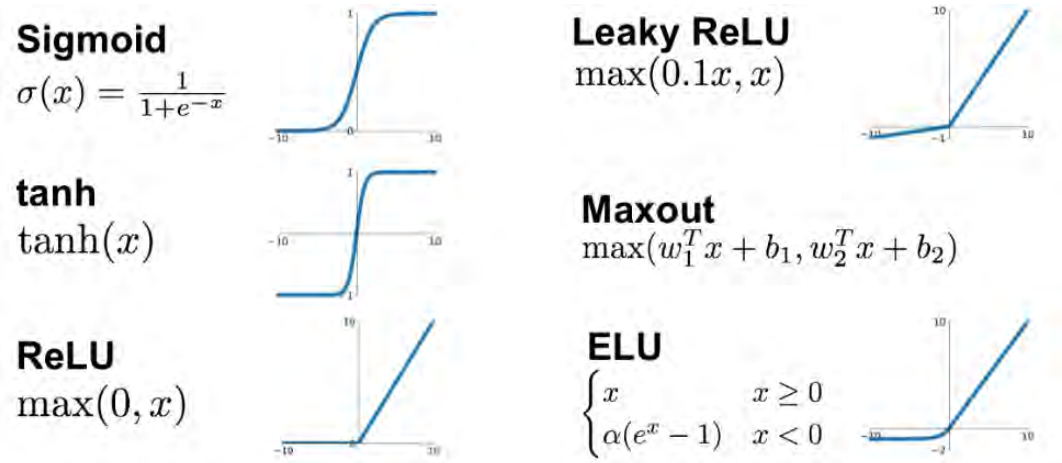



Abbildung 2.2: Aktivierungsfunktionen

### 2.1.2 Klassifizierung von künstlichen neuronalen Netzen

Für künstliche, neuronale Netze gibt es eine Vielzahl von Anwendungsmöglichkeiten. Daher wird nachfolgend die Einteilung der Netze anhand ihrer lokalen Anordnung der Neuronen erklärt. Im nächsten Schritt wird betrachtet, welche Lernverfahren für KNN zur Verfügung stehen.

#### 2.1.2.1 Netzwerktopologie

Künstliche neuronale Netze bestehen aus einer Vielzahl an Neuronen und deren Verbindungen zu einander. Als Topologie wird die Struktur der Neuronenverbindungen in einem Netz bezeichnet. Es lassen sich beim Aufbau von KNN unterschiedliche Grundstrukturen festlegen, wobei die geläufigsten Strukturen in diesem Abschnitt erklärt werden. Die Netzwerktopologie wird auch als Netzwerkarchitektur oder -struktur bezeichnet. Die Zusammenfassung von Neuronen mit gleicher Funktionalität erfolgt in sog. Schichten (engl. layer) und stellt eine grundlegende Architekturkomponente dar. Es können drei unterschiedliche Topologien von KNN unterschieden werden. Eine Übersicht verschiedener Netzwerkmodelle ist in Abbildung 2.3 dargestellt. [5]

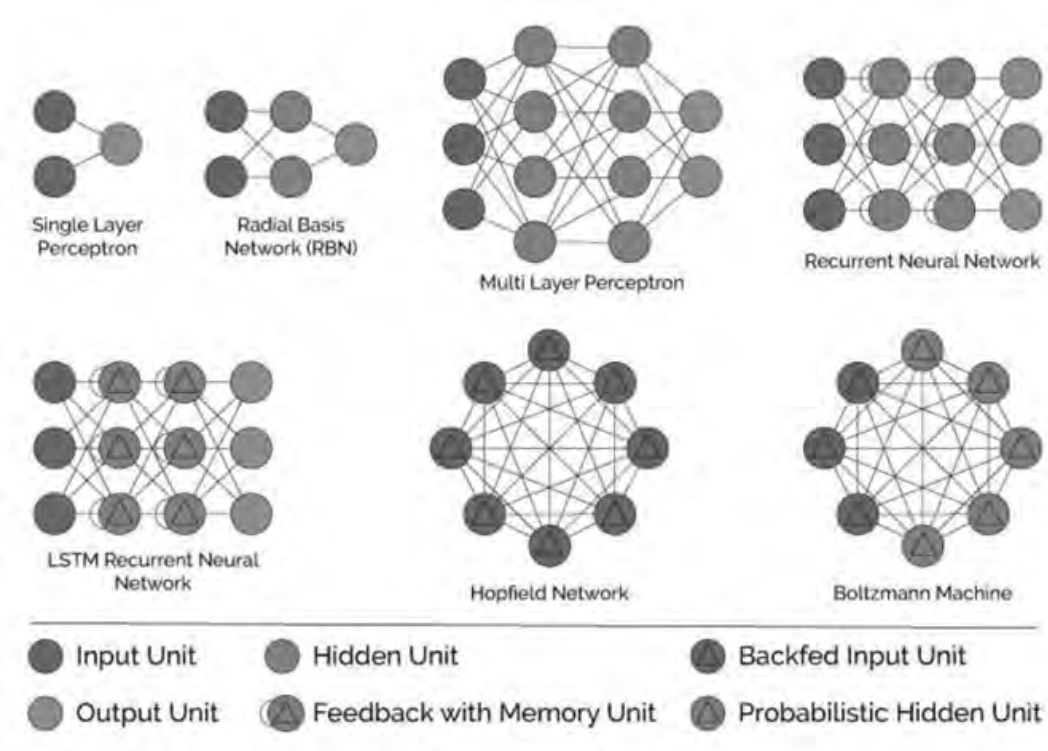



Abbildung 2.3: Verschiedene Architekturen von neuronalen Netzen[6]

**Einschichtiges vorwärts betriebenes Netz** Die einfachste Netzwerkstruktur ist die der einschichtig, vorwärts betriebenen Netze (engl. single layer feed-forward network). Die Eingabeschicht (engl. input layer) ist vollständig mit der Ausgabeschicht (engl. output layer) verbunden, wobei der Datenfluss nur von der Eingabe zur Ausgabe möglich ist. Diese Netzwerkstruktur besitzt in der Regel eine lineare Aktivierungsfunktion. Typische Modelle dieser Architektur sind das Perzeptron (engl. Single Layer Perceptron, SLP) und das Adaptive Lineare Neuron (ADALINE). Diese beiden Modelle sind auf die Lösung linearer Problemstellungen beschränkt. [5]

**Mehrschichtiges vorwärts betriebenes Netz** Das mehrschichtig, vorwärts betriebene Netz (engl. multi layer feed-forward network) ist eine Erweiterung der einschichtigen Netze um eine oder mehrere zusätzliche, verdeckte Schichten (engl. hidden layer). Sind die Neuronen einer Schicht mit allen Neuronen der vorhergehenden Schicht verbunden sind, spricht man von einem vollständig verbundenem Netz (engl. fully-connected). Ein Multilayer Perzeptron (engl. multi layer perceptron, MLP) ist ein Modell dieser Architektur. Ein MLP gilt als universeller Funktionsapproximator, da es jede mathematische Funktion approximieren kann und daher flexibel einsetzbar ist. Die Leistungsfähigkeit von MLPs ist abhängig von der gewählten Anzahl an verdeckten Schichten und Neuronen. In der Praxis ist die Entwicklung eines MLP meist mit einem „Trial & Error“ Prozess verbunden[5]. Durch die Zuordnung von nichtlinearen Aktivierungsfunktionen ist das MLP dazu geeignet nicht linear separierbare Daten zu unterscheiden bzw. Funktionen zu approximieren. Ein MLP stellt die Basis für viele praktische Anwendungen dar. [7]

**Rekurrentes Netz** Diese Netzwerktopologie besitzt im Gengensatz zu den beiden zuvor genannten Architekturen rückgerichtete Kanten und enthält dadurch Rückkopplungen. Hier wird zwischen direkten, indirekten und seitlichen Rückkopplungen unterschieden. Aufgrund der Rückkopplungen erhält das Netz ein dynamisches Verhalten. Diese Netze werden in der Praxis zur Verarbeitung von Sequenzen eingesetzt. [5]

#### 2.1.2.2 Lernverfahren von neuronalen Netzen

Unter Lernen im Zusammenhang mit künstlichen neuronalen Netzen wird ein Prozess verstanden, in dem an ein KNN Daten einer Problemstellung angelegt werden und das Netz seine Verbindungsgewichte so anpasst, dass das Problem gelöst werden kann. Dieser Vorgang wird auch als Training bezeichnet. Daher werden die zur Verfügung stehenden Daten in Trainings- und Testdaten unterteilt. Mit Hilfe der Trainingsdaten wird der Lernprozess des Netzes durchgeführt. Die Testdaten dienen zur Evaluation des KNN. Mit ihnen kann festgestellt werden, wie das neuronale Netz auf neue, bisher unbekannte Daten reagiert. Ein Lernprozess kann in vier Schritte unterteilt werden und ist beendet, wenn ein angegebenes Zielkriterium oder die vorgegebene Anzahl an Lernschritten erreicht ist. [5]

1. Vorbelegung der Verbindungsgewichte
2. Trainingsdaten an das Netz anlegen
3. Berechnung der Ausgabewerte
4. Anpassung der Verbindungsgewichte entsprechend des gewählten Lernalgorithmus

Es gibt 3 verschiedene Arten von Lernverfahren, um ein künstliches neuronales Netz zu trainieren.

**Überwachtes Lernen** Beim überwachten Lernen (engl. supervised learning) werden die berechneten Ausgabemuster mit den gewünschten Ausgabemustern verglichen und die Gewichte des neuronalen Netzes so angepasst, dass die Differenz der Ausgabe des Netzwerks und der tatsächlichen Ausgabe minimal wird. Für dieses Vorgehen muss zu jedem Eingabedatensatz ein entsprechender Ausgabewert zur Verfügung stehen. Ziel dieses Ansatzes ist die Assoziation von Eingabedaten zu Ausgabedaten. Das neuronale Netz soll also eine Funktion lernen, die die Zuordnungsregel von Eingabedaten zu Ausgabedaten approximiert. Beispiele für dieses Verfahren sind die Delta-Regel bei SLPs und der in Kapitel 2.1.4 beschriebene Backpropagation-Algorithmus für MLPs. [8]

**Unüberwachtes Lernen** Beim unüberwachten Lernen (engl.unsupervised learning) stehen im Gegensatz zum überwachten Lernen die korrekten Ausgabewerte nicht zur Verfügung. Das KNN verändert sich entsprechend der Eingabedaten von selbst. Das Ziel dieses Lernverfahrens ist es, ein Muster in den Eingabedaten zu erkennen. Es soll festgestellt werden, warum manche Eingabemuster häufiger auftreten als andere. Anhand einer festgestellten Regelmäßigkeit der Eingabewerte, werden diese entsprechenden Gruppen zugeordnet. Lernalgorithmen des unüberwachten Lernens sind die adaptive Resonanztheorie und die Hebbsche Lernregel. [8]

**Bestärkendes Lernen** Das bestärkende Lernen (engl. reinforcement learning) wird angewandt, wenn die Ausgabe eines Systems eine Aktion sein soll. In diesem Fall ist eine einzelne Aktion weniger von Bedeutung, als die Sequenz von auszuführenden Aktionen (Taktik). Das neuronale Netz soll in der Lage sein, die Güte von Aktionssequenzen einzuschätzen und basierend auf vergangenen Sequenzen eine neue Taktik zu generieren. Die bekanntesten Algorithmen dieses Lernverfahrens sind die Monte-Carlo-Methoden und das Temporal Difference Learning. [8]

### 2.1.3 Die Kostenfunktion

Neuronale Netze, die zur Klassifikation oder Regression eingesetzt werden, haben das Ziel erlernbare Aufgaben zu lösen, wie zum Beispiel die Zuordnung von Eingabedaten und Ausgabedaten. Wie bereits erwähnt müssen beim überwachten Lernen die Gewichte der Verbindungen

eines Netzes so angepasst werden, dass die tatsächliche Ausgabe des Netzes zu der gewünschten Ausgabe einer zugehörigen Eingabe passt. Um das zu erreichen wird eine Kostenfunktion (engl. Loss Function) benötigt, welche die Differenz zwischen tatsächlichem und gewünschten Ausgabewert misst. Die Kostenfunktion soll mit Hilfe von Trainingsdaten minimiert werden. Abhängig davon, ob das Netz zur Klassifikation oder zur Regression eingesetzt wird, werden in der Praxis verschiedene Kostenfunktionen gewählt. Wird das KNN zur Regression eingesetzt, dann wird standardmäßig der Mean Squared Error (MSE) verwendet. Für den Fall, dass es hier zu unzureichenden Ergebnissen kommt, kann auch der Mean Absolute Error (MAE) oder der Mean Squared Logarithmic Error (MSLE) verwendet werden. Bei der Klassifikation wird standardmäßig Cross-Entropy als Kostenfunktion eingesetzt. [8]

### 2.1.4 Der Backpropagation-Algorithmus

Der Backpropgation-Algorithmus wird heute in fast allen neuronalen Netzen eingesetzt. Er ermöglicht es die Gewichte der versteckten Schichten anzupassen. Bei Hidden Layern kann kein direkter Fehler zwischen gewünschtem und tatsächlichem Ausgabewert berechnet werden, weil dieser nur in der Ausgabeschicht bekannt ist. Der Backpropagation-Algorithmus gliedert sich in drei Schritte. Zuerst findet der Forward-Pass statt, bei dem eine Eingabe an das neuronale Netz angelegt wird und der zugehörige Ausgabewert berechnet wird. Anschließend wird der Fehler der Ausgabeneuronen ermittelt. Überschreitet dieser Fehler einen gewissen Schwellwert, dann kommt es zum sog. Backward-Pass. Die Fehlerterme breiten sich in Richtung der Eingabeschicht aus. Anhand dieser Fehlerterme werden die Gewichtsanpassungen durchgeführt, so dass die Fehlerterme minimal werden. [9]

Der Backpropagation-Algorithmus verwendet den Gradientenabstieg (engl. Gradient Descent), um die Kostenfunktion in einem KNN zu minimieren. Es werden die Gewichte $w_i$ der Verbindungen zwischen den Neuronen so bestimmt, dass die Kostenfunktion minimal wird. Dabei werden die Parameter des Netzes in entgegengesetzter Richtung des Gradienten der Kostenfunktion $\nabla J_\theta(\theta)$ angepasst . Die Lernrate $\eta$ bestimmt die Größe der Schritte zum Erreichen eines (lokalen) Minimums. Mit dieser Technik lassen sich mehrstufige Perzeptronen effizient trainieren. [10][11]

#### 2.1.4.1 Varianten des Gradientenabstiegs

Es gibt drei Varianten des Gradient Descent Verfahrens, welche sich durch die Anzahl der verwendeten Daten zur Berechnung des Gradienten unterscheiden. Abhängig davon wird ein Kompromiss zwischen Genauigkeit der Parameteraktualisierung und Dauer der Parameteraktualisierung eingegangen. [12]

**Batch Gradient Descent** Beim Batch Gradient Descent wird eine Anpassung der Parameter $\theta$ nach Verarbeitung aller Daten durchgeführt.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.1}$$

Da bei diesem Ansatz die Gradienten für die vollständigen Trainingsdaten berechnet werden müssen, kann die Ausführung sehr lange dauern. Für Datensätze, die nicht in den Speicher passen, kann Batch Gradient Descent nicht verwendet werden. Es ist garantiert, dass dieses Verfahren für eine konvexe Fehlerfläche zu einem globalen Minimum und für eine nicht konvexe Fehlerfläche zu einem lokalen Minimum konvergiert. [12]

**Stochastic Gradient Descent** Stochastic Gradient Descent (SGD) nimmt eine Anpassung der Parameter $\theta$ nach jedem einzelnen Trainingsdatensatz mit Input $x^{(i)}$ und Output $y^{(i)}$ vor.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.2}$$

Die Schwankungen des SGD ermöglichen zum einen das Erreichen besserer lokaler Minima, können aber auch zur Überschreitung lokaler Minima führen. Es wurde aber bereits gezeigt, dass durch eine langsame Verringerung der Lernrate SGD das selbe Konvergenzverhalten wie Batch Gradient Descent zeigt. [12]

**Mini-Batch Gradient Descent** Mini-Batch Gradient Descent ist eine Mischung der beiden zuvor genannten Verfahren, um deren Vorteile zu vereinigen. Eine Anpassung der Parameter $\theta$ wird für jeden Mini-Batch, welcher aus $n$ Datensätzen besteht, durchgeführt.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.3}$$

Dieser Ansatz hat zum Einen den Vorteil der verringerten Varianz bei der Parameteraktualisierung, was zu einer stabileren Konvergenz führt und zum Anderen bietet er die Möglichkeit der Nutzung effizienter Matrixberechnungen. Für gewöhnlich liegt die Größe der Mini-Batches zwischen 50 und 256. Dieser Algorithmus wird standardmäßig für das Trainieren neuronaler Netze verwendet. [12]

#### 2.1.4.2 Optimierungen des Gradientenabstiegs

Nachfolgend werden die Optimierungsalgorithmen Moment, Adagrad, Adadelta und Adam vorgestellt.

**Moment** Die hohen Schwankungen in der Varianz des SGD führen zu einer langsamen Konvergenz. Mit Hilfe des Moments (engl. Momentum) werden Schritte in Richtung eines Minimas

erhöht und Schwankungen gedämpft. Dazu wird ein Anteil $\gamma$ der letzten Parameteranpassungen zur aktuellen Parameteranpassung hinzugefügt.

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta J(\theta) \tag{2.4}$$

Das führt zu einem Update der Parameter $\theta = \theta - v_t$. Dem Parameter $\gamma$ wird üblicherweise der Wert 0,9 zugewiesen. Der Parameter $\gamma$ erhöht sich für Gradienten, die in die selbe Richtung zeigen und verringert sich bei Richtungswechseln. Diese Technik reduziert unnötige Parameteranpassungen und führt zu einer schnelleren, stabileren Konvergenz und reduziert die auftretenden Schwankungen des SGD. [13]

**Adagrad** Adagard passt die Lernrate $\eta$ abhängig von den Parametern $\theta$ an. Das führt zu stärkeren Anpassungen für seltene Parameter und geringeren Anpassungen für häufige Parameter. Dadurch ist dieser Ansatz besonders für Daten mit fehlenden Attributen geeignet.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{2.5}$$

Der Vorteil dieses Algorithmus ist, dass die Lernrate nicht manuell angepasst werden muss. In derzeitigen Implementierungen wird sie auf den Wert 0,01 gesetzt. Der Nachteil ist eine kontinuierlich abnehmende Lernrate. [14]

**Adadelta** Adadelta ist eine Erweiterung des Adagrad Algorithmus, um das Problem der kontinuierlich abnehmenden Lernrate zu beheben. Anstatt alle zuvor berechneten Gradienten zu speichern, wird der gleitende Mittelwert der Gradienten $E[g^2]_t$ verwendet. Außerdem wird ähnlich wie bei der Moment Technik ein Parameter $\gamma$ eingeführt. Die Parameteränderung berechnet sich dann wie folgt:

$$E[g^2]_t = \gamma \cdot E[g^2]_{t-1} + (1 - \gamma) \cdot g_t^2 \tag{2.6}$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \tag{2.7}$$

Da der Term unter der Wurzel dem Root Mean Squared Error (RMSE) entspricht, kann Adadelta vollständig auf eine Lernrate verzichten, indem einfach der RMSE der zuvor berechneten Gradienten verwendet wird. [15]

**Adam** Adaptive Moment Estimation (Adam) ist eine weitere Methode, die adaptive Lernraten für jeden Parameter berechnet. Zusätzlich werden individuelle Momente verwendet.

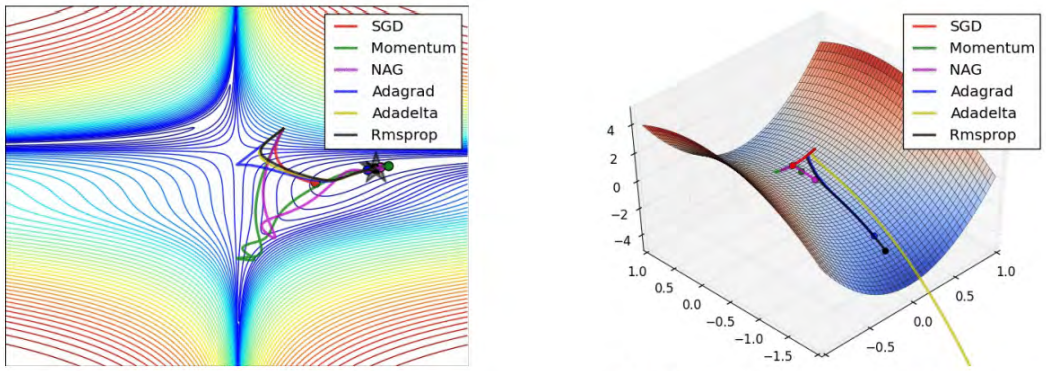
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.8}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.9}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t \tag{2.10}$$

Typischerweise werden für die Parameter $\beta_1$, $\beta_2$ und $\epsilon$ die Werte 0.9, 0.999 und $10e^{-8}$ verwendet. Der Adam Optimierer hat in der Praxis gute Ergebnisse gezeigt und kommt daher in den meisten neuronalen Netzen zum Einsatz. Der Algorithmus konvergiert sehr schnell und die Lerndauer des zu trainierenden Modells ist auch gering. Außerdem behebt er die zuvor genannten Probleme der abnehmenden Lernrate, der langsamen Konvergenz und der hohen Varianz in den Parameteränderungen. [16]

In Abbildung 2.4 ist das Konvergenzverhalten einiger Optimierungsalgorithmen dargestellt.




(a) SGD optimization on loss surface contours

(b) SGD optimization on saddle point

Abbildung 2.4: Konvergenzverhalten der Optmierungsverfahren[12]

## 2.2 Besonderheiten von Convolutional Neural Networks

Convolutional Neural Networks gehören zur Kategorie der tiefen, mehrschichtigen, neuronalen Netze und sind insbesondere für die Verarbeitung von zweidimensionalen Daten wie zum Beispiel Bildern und Videos ausgelegt. CNNs reduzieren die Anzahl der zu lernenden Parameter und verbessern dadurch das Training in vorwärtsgerichteten Netzen mittels Back-Propagation. CNNs wurden ursprünglich als Deep Learning Framework vorgeschlagen, welches nur geringe Anforderungen an die Datenvorverarbeitung stellt. CNNs propagieren die Eingabedaten durch verschiedene Layer des Netzwerks, in denen unterschiedliche digitale Filter angewendet werden, um Features zu detektieren. CNNs können sowohl zur Klassifikation, als auch für Regression eingesetzt werden. [17]

### 2.2.1 Aufbau von Convolutional Neural Networks

Die Besonderheit von Convolutional Neural Networks besteht darin, dass mehrdimensionale Daten, wie zum Beispiel Bilder, Videos oder Sprache, verarbeitet werden können. Diese Netze bestehen hauptsächlich aus Convolution, Pooling und vollständig verbundenen Layern, wobei nachfolgend die ersten beiden Typen vorgestellt werden. In der Praxis folgt ein Pooling Layer immer auf einen Convolution Layer. Diese beiden Layer werden mehrfach hintereinander ausgeführt und dienen zur Feature Extraction. Darauffolgend wird ein MLP verwendet, welche die extrahierten Features benutzt, um eine Klassifikation oder Regression durchzuführen. Für die einfachere Veranschaulichung, wird nachfolgend das CNN am Beispiel der Bildverarbeitung beschrieben.

**Convolution Layer** Die Grundlage für den Convolution Layer bildet der mathematische Faltungs-Operator (engl. convolution). Die Eingabedaten werden mit einer festgelegten Anzahl an digitalen Filtern, die eine feste Pixelgröße haben, analysiert. Dadurch wird nicht jeder Pixel mit einem Neuron verbunden, sondern nur lokale Bereiche des Bildes. Die Größe des lokalen Bereichs entspricht der Größe des Filters und wird als rezeptives Feld (engl. receptive field) bezeichnet. Die Anzahl der Gewichte eines Neurons ergibt sich dann aus dem Produkt des rezeptiven Feldes und der Tiefe des Eingabebildes. Dieses Vorgehen ist in Abbildung 2.5 am Beispiel eines 32×32 Farbbildes und fünf 5×5 Filtern dargestellt. Jedes Neuron im Convolution Layer hat 75 (5×5×3) Verbindungen zu dem Eingabebild. Dabei sind immer 5 Neuronen dem selben Bildausschnitt zugeordnet. Die Größe der Ausgabe ist abhängig von

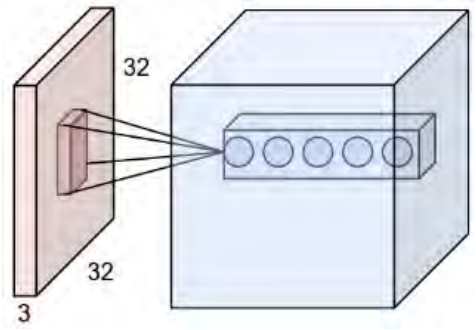



Abbildung 2.5: Funktionsweise des Convolution Layers[18]

den Hyperparametern zero-padding, Schrittweite (engl. stride) und der Anzahl der Filter. Die Anzahl der Filter entspricht der Tiefe der Ausgabe. Zero-padding und stride haben Einfluss auf die Höhe und Breite der Ausgabe. Unter stride versteht man die Schrittweite, mit der ein Filter von links oben nach rechts unten über ein Bild geschoben wird. Zero-padding gibt an wie sich der Filter an den Rändern eines Bildes verhalten soll. Die nachfolgende Formel zeigt,

wie die Ausgabegröße berechnet werden kann. Dabei muss das Ergebnis ein ganzzahliger Wert sein, sonst ist die Konfiguration nicht zulässig.

$$\frac{W - F + 2 \cdot P}{S} + 1 \tag{2.11}$$

In dieser Formel entspricht $W$ der Breite der Eingabe, $F$ der Größe des Filters, $P$ dem zero-padding Parameter und $S$ der Schrittweite. Die Höhe der Ausgabedaten lässt sich analog berechnen. [18]

**Pooling Layer** Der Pooling Layer wird auf das Ergebnis eines Convolution Layers angewendet und dient dazu die Größe der zu verarbeitenden Daten zu reduzieren. Das führt gleichzeitig zu einer Reduzierung der Netzparameter und des Rechenaufwandes. Außerdem wird so das Problem von Overfitting verhindert. Es gibt verschiedene Pooling-Techniken, wobei in der Praxis meist Max-Pooling verwendet wird. Es wird ein Fenster mit festgelegter Größe und Schrittweite von links oben nach rechts unten über die Eingabedaten geschoben und aus jedem Bereich wird der Maximalwert beibehalten. Das Vorgehen ist in Abbildung 2.6 dargestellt. Die Tiefe der Ausgabedaten entspricht der Tiefe der Eingabedaten. Die Größe der Ausgabe lässt sich mit der Formel 2.12 berechnen, wobei $W$ der Breite der Eingabe, $F$ der Größe des Fensters und $S$ der Schrittweite entspricht. Die Höhe der Ausgabedaten lässt sich wieder analog berechnen. [18]

$$\frac{W - F}{S} + 1 \tag{2.12}$$

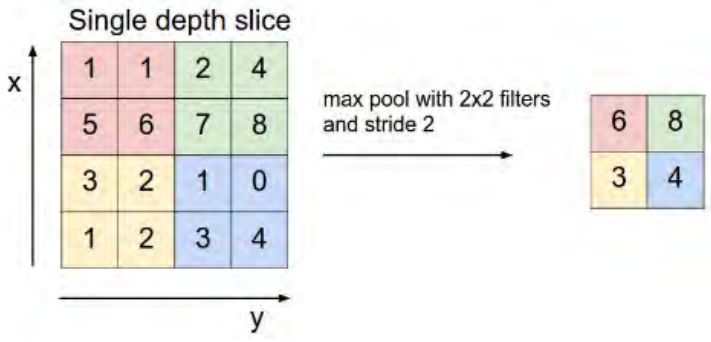



Abbildung 2.6: Funktionsweise von Max-Pooling[18]

### 2.2.2 Architekturen von Convolutional Neural Networks

Nachfolgend werden die gängigsten Architekturen von Convolutional Neural Networks gezeigt. Dazu gehören AlexNet, GoogleLeNet, VGGNet und ResNet.

#### 2.2.2.1 AlexNet

AlexNet[19] wurde von Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton an der Universität von Toronto entwickelt und war der Gewinner der ImageNet Large Scale Visual Recognition Challenge (ILSVRC) im Jahr 2012. ImageNet ist ein jährlich stattfindender Wettbewerb mit Spezialisierung auf Bildklassifikation. Die Architektur ist in Abbildung 2.7 dargestellt. AlexNet verwendet die ReLU Aktivierungsfunktion anstatt der tanh-Funktion. Das führt du zu einem sechs mal schnelleren Trainingsprozess und einer erhöhten Klassifikationsgenauigkeit. Außerdem wird Dropout Regularisierung verwendet, damit es nicht zu Overfitting kommt. Ein weiteres Merkmal von AlexNet ist die Verwendung von überlappendem Pooling, was zu einer schmaleren Netzarchitektur führt und die Top-1 bzw. Top-5 Fehlerraten um 0.4% bzw. 0.3% senkt. Diese Architektur besteht aus fünf Convolution Layern und drei vollständig verbundenen Schichten. Nach jeder dieser Schichten wird die ReLU Aktivierungsfunktion angewandt. AlexNet hat 62.3 Millionen Parameter und 650.000 Neuronen. Der Trainingsprozess dieses Netzes braucht 90 Trainingsepochen und dauert auf zwei GTX 580 Graphical Processing Units (GPU) zwischen fünf und sechs Tagen. Als Lernalgorithmus wird SGD mit einer Lernrate von 0.01, einem Moment von 0.9 und einer Gewichtsabnahme 0.0005 verwendet.

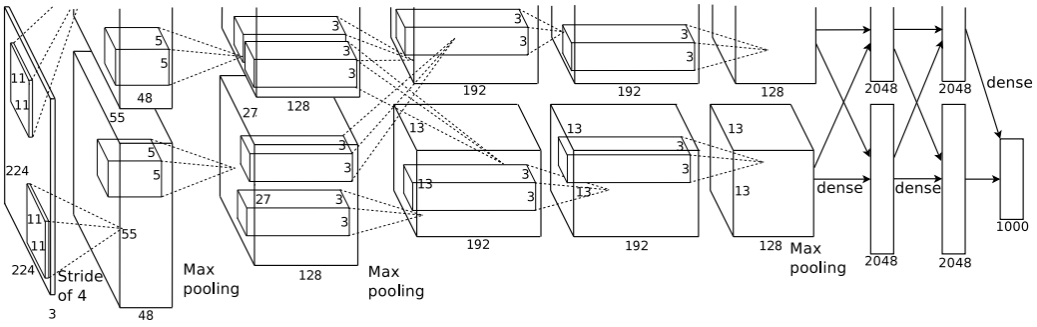



Abbildung 2.7: Architektur von AlexNet[19]

#### 2.2.2.2 GoogleLeNet/Inception

GoogleLeNet[20] auch bekannt unter der Bezeichnung Inception v1 ist der Gewinner der ILSVRC 2014 mit einer Top-5 Fehlerrate von 6.67%. Diese Architektur hebt sich von anderen Architekturen durch die Verwendung von 1×1 Covolution und Global Average Pooling ab. Diese beiden Techniken sind in [21] detailliert beschrieben. GoogleLeNet verwendet ein sog. Inception Modul (Abbildung 2.8), welches erlaubt verschiedene Größen eines Convolution Layers auf die selben Eingabedaten anzuwenden. Dadurch werden verschiedene Features extrahiert und am Ende des Moduls wieder zusammengefügt. Die 1×1 Convolution dient dabei zur Dimensionsreduktion. Durch die Average Pooling Technik werden die, in anderen Architekturen verwendeten vollständig verbundenen Schicht, ersetzt, was zu einer Verringerung von trainierbaren Parametern führt und die Top-1 Klassifikationsgenauigkeit um 0.6%

erhöht. GoogleLeNet hat insgesamt 22 Schichten, wenn man nur die Layer mit Parametern

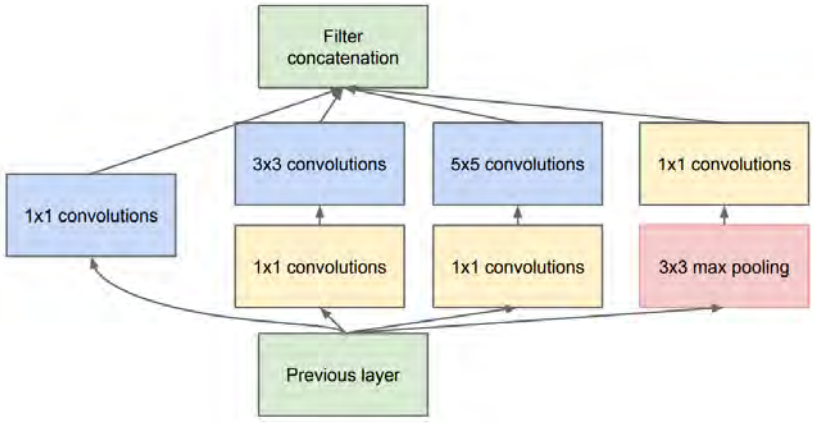



Abbildung 2.8: Aufbau des Inception Moduls[20]

berücksichtigt und stellt im Vergleich zu den hier vorgestellten Architekturen AlexNet und VGGNet ein sehr tiefes Modell dar. Als Lernalgorithmus wurde der asynchrone SGD Algorithmus gewählt, mit einer festen Lernrate, welche alle acht Trainingsepochen um 4% verringert wird. Für das Momentum wird ein Wert von 0.9 gewählt. Für eine bildliche Darstellung der gesamten GoogleLeNet Architektur wird auf [20] verwiesen.

#### 2.2.2.3 VGGNet

VGGNet[22] wurde von der Visual Geometry Group (VGG) an der Universität von Ohio entwickelt und belegte 2014 bei der ILSVRC mit einer Klassifikationsgenauigkeit von 92.3% den zweiten Platz. Diese Netzarchitektur ist vor allem durch den einfachen Aufbau charakterisiert, da sie nur Covolution Layer der Größe 3×3 verwendet, die aneinander gereiht sind, um die Tiefe des Netzes zu erhöhen. Die Dimensionsreduktion wird mit der Max-Pooling Technik durchgeführt. Abschließend folgen noch zwei vollständig verbundene Schichten mit jeweils 4096 Neuronen (Abbildung 2.9). VGGNet gibt es mit unterschiedlichen Tiefen. Am häufigsten werden die Architekturen mit 16 bzw. 19 Schichten verwendet.

#### 2.2.2.4 ResNet

Die Residual Neural Network Architektur[24] (ResNet) wurde von Microsoft entwickelt und hat die ILSVRC im Jahr 2015 mit einer Top-5 Fehlerrate von 3.57% gewonnen. Diese Architektur behandelt das Problem des verschwindenden Gradienten beim Trainieren von tiefen neuronalen Netzen. Die Architektur enthält sog. Skip-Verbindungen mit deren Hilfe, die Gradienten im Netz zurückgereicht werden können. Es gibt verschiedene Ausprägungen dieses Modells, abhängig von der Anzahl an Schichten. Das Siegermodell 2015 hatte 152 Layer. In Abbildung 2.10 ist die ResNet Architektur mit 34 Schichten dargestellt. Nach je einer Co-

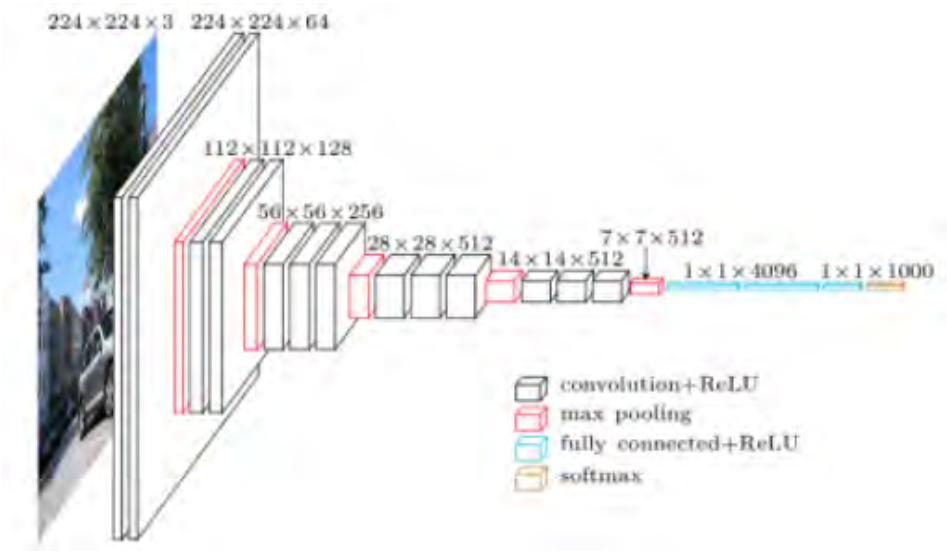



Abbildung 2.9: Architektur von VGGNet-16[23]

volution und Pooling Schicht folgen vier Verarbeitungsblöcke mit der selben Funktionalität. Einziger Unterschied ist die Anzahl der verwendeten Filter (64, 128, 256, 512).

### 2.2.3 Transfer Learning

Transfer Learning bezeichnet das Übertragen von Ergebnissen eines bereits fertig trainierten neuronalen Netzes auf eine neue Aufgabe. Das Trainieren von KNN, insbesondere CNNs zur Objekterkennung, kann sehr rechen- und zeitintensiv sein. Daher kann es sinnvoll sein, wenn auf den bereits gelernten Features eines Netzes aufgesetzt wird. Es können die Schichten eines trainierten Netzes konstant gehalten und nur die Ausgabeschichten angepasst werden (Fine-Tuning). Der Vorteil dieser Technik ist, dass die bereits extrahierten Features eines Netzes verwendet werden können und nur noch die Zuordnung zu den neuen Klassen trainiert werden muss. Es gibt aber auch die Möglichkeit einige oder alle Schichten weiter zu trainieren. Dann werden die bereits angepassten Gewichte als Initialwert für das Trainieren des Netzes verwendet, was zu einer Reduktion der Trainingszeit führt. Welche Technik verwendet wird ist abhängig von der Ähnlichkeit der Quell- und Zieldaten, sowie der Größe des neuen Datensatzes. In Tabelle 2.1 ist dargestellt, welche Technik für welche Ausgangslage verwendet werden sollen. [18]

Haben die Obkjekte in Bildern ähnliche Strukturen und es stehen für das Trainieren eines Netzes nur wenige Daten zur Verfügung, dann kann die Ausgabeschicht ersetzt werden. Die restlichen Schichten des Netzes bleiben gleich und werden nicht in das Training miteinbezogen. Sind deutliche Unterschiede in den zu erkennenden Objekten zu finden und wenig Beispieldaten vorhanden, reicht es nicht aus nur die Ausgabeschicht zu ändern. Hier müssen zusätzlich vordere Schichten des Netzes mit in den Trainingsprozess aufgenommen werden. Für große

| | **Ähnliche Daten** | **Unterschiedliche Daten** |
|---|---|---|
| **Großer Datensatz** | Fine-Tuning | Fine-Tuning oder neu trainieren |
| **Kleiner Datensatz** | nur letzte Schichten trainieren | frühere Schichten auch trainieren |

Tabelle 2.1: Verwendung von Transfer Learning abhägig vom Datensatz

Datensätze mit ähnlichen Strukturen kann ein bereits fertig trainiertes Netz genommen werden. Hier muss nur die Ausgabe entsprechend der gewünschten Ergebnisse angepasst werden. Der Trainingsprozess wird für das gesamte Netz angestoßen, ist aber aufgrund der bereits vorbelegten Gewichte deutlich geringer. Sind keine Ähnlichkeiten eines großen Datensatzes zu einem bestehenden Problem vorhanden, dann kann in einem ersten Versuch probiert werden, das Netz wie im vorherigen Fall mittels Fine-Tuning anzupassen. Die zweite Möglichkeit ist nur die Struktur eines bestehenden Netzes zu verwenden und mit initialen Gewichten einen neuen Trainingsprozess starten.

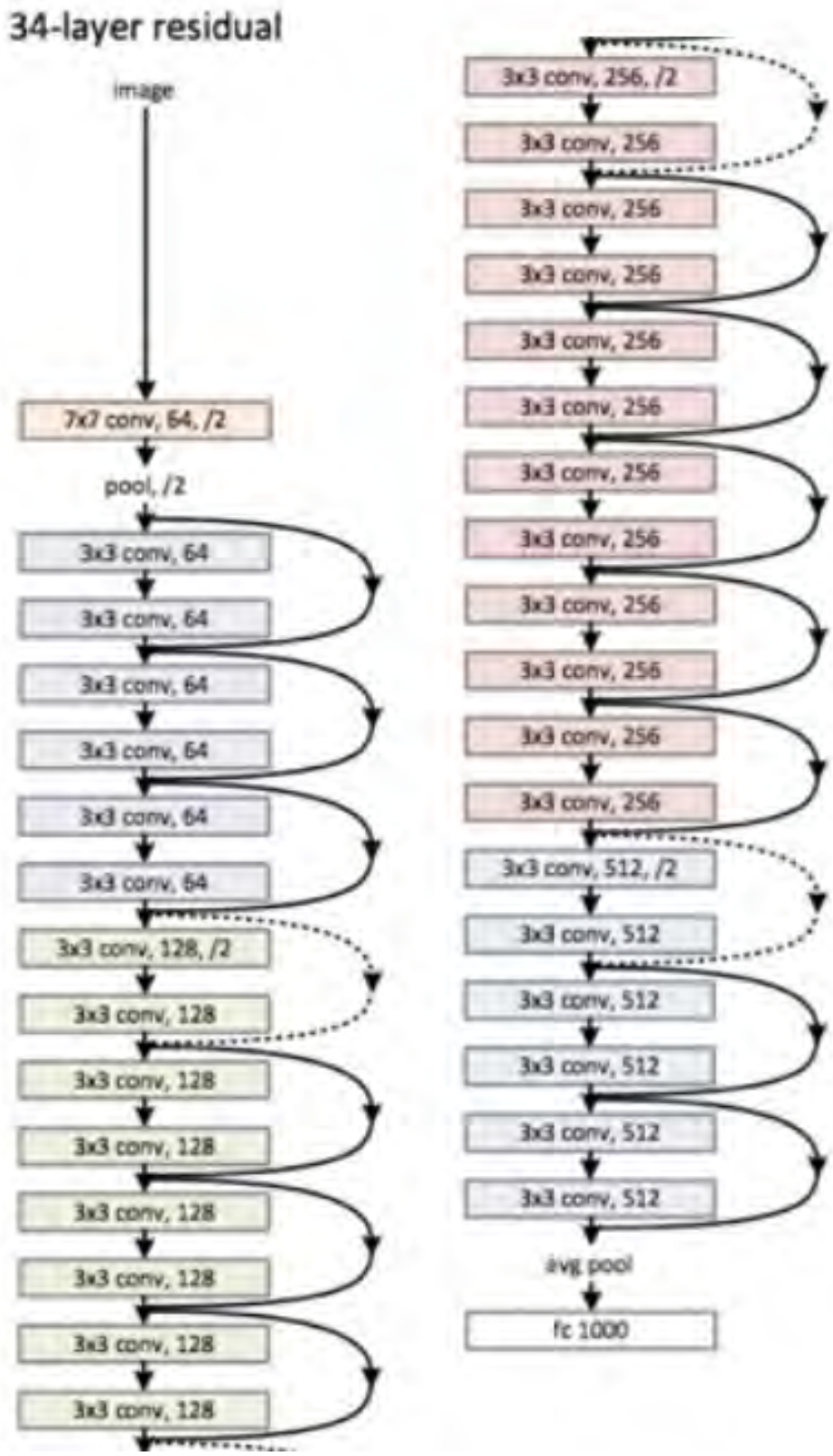



Abbildung 2.10: Architektur von ResNet mit 34 Schichten

# 3 Analyse der zur Verfügung stehenden Daten und Erstellung der neuronalen Netze

In diesem Abschnitt der Masterarbeit sollen die zur Verfügung gestellten Daten analysiert und deren Vorverarbeitung, sowie die erstellten Modelle beschrieben werden. Basierend auf der Datenanalyse wird eine Vorverarbeitung durchgeführt, um abschließend verschiedene Modelle zur Vorhersage der Maschinenparameter vorzustellen. Das Ergebnis eines dreidimensionalen computertomographischen Scans ist abhängig von den eingestellten Parametern am Computertomographen (CT). Es soll versucht werden die einzustellende Stromstärke, Spannung und Belichtungsdauer an der Maschine, sowie das Filtermaterial, welches verwendet werden soll, zu bestimmen, so dass ein möglichst genauer 3D-Scan durchgeführt werden kann. Hierfür wird versucht mit neuronalen Netzen eine Regressionsaufgabe zu lösen. Es soll die Abweichung des gemessenen Istwertes einer Messung von dessen Sollwert vorhergesagt werden. Die Modelle werden dann mit Intervallen der jeweiligen Maschinenparameter versorgt und von den vorhergesagten Abweichungen wird anschließend das Minimum bestimmt. Die zugehörigen Eingabewerte stellen das Ergebnis der vorgeschlagenen Maschinenparameter dar. In diesem Abschnitt wird nur die Struktur der Modelle erklärt. Die Trainings- bzw. Validierungsergebnisse werden im Kapitel 5 gezeigt.

## 3.1 Datenanalyse

Für diese Arbeit wurden vom Fraunhofer Institut in Deggendorf drei Datenbanktabellen der bisher durchgeführten Messungen mit den zugehörigen Messergebnissen, als auch deren 3D Bilddateien im Dateiformat REK zur Verfügung gestellt.

### 3.1.1 Analyse der Messparameter

Die drei Datenbanktabellen liegen im SQL-Format vor und beinhalten Informationen zu den Messungen, den Messobjekten und den verwendeten Maßen. Jede Zeile in der Datenbank Messungen entspricht genau einem durchgeführten 3D CT-Scan. In der Datenbank Maße ist für jedes gemessene Maß eine Zeile vorhanden. Die Datenbank Messobjekt beinhaltet alle verwendeten Objekte. In der Tabelle 3.1 sind die einzelnen Spalten der Datenbanktabellen beschrieben.

Insgesamt wurden 15319 Messungen an 62 verschiedenen Objekten durchgeführt. Das führte zu 212954 gemessenen Maßen. In dieser Arbeit werden nur die Daten zu den gehärteten Stahlzylindern mit Durchmessern zwischen einem und zehn Millimeter betrachtet. Zu den Stahlzylindern sind die Maße Durchmesser und Zylindrizität an verschiedenen Stellen des jeweiligen Objekts gemessen worden. Es kann festgestellt werden, dass für die Stahlzylinder die Maße Durchmesser, D_Z-10, D_Z-5, D_Z0, D_Z5, D_Z10, Zylindrizität, FT_Z-10, FT_Z-5, FT_Z0, FT_Z5 und FT_Z10 existieren.

| **Datenbankname** | **Spaltenname** | **Beschreibung** |
|---|---|---|
| Messungen | Messnummer | Eindeutige ID einer CT-Messung |
| Messungen | Stromstärke, Spannung, Belichtungsdauer, Qualität, Schritte, Filter | Maschinenparameter |
| Messungen | Dunkelwert, Hellwert | Dunkelster bzw. hellster Grauwert mit der gegebenen Parameterkombination |
| Messungen | Auflösung | Vergrößerungsstufe der CT |
| Messungen | Leistung | Stromstärke * Spannung |
| Messungen | Objektnummer | Eindeutige ID des gemessenen Objekts |
| Maße | Typ | Typ des Maßes (z.B. Längenmaß, Durchmesser, etc. ) |
| Maße | Istwert | Gemessener Wert des Maßes |
| Maße | Sollwert | Richtiger Wert des Maßes |
| Maße | uTol, oTol | Toleranzen |
| Maße | Maß | Bezichnung des Maßes |
| Maße | Messnummer | Eindeutige ID einer CT-Messung |
| Messobjekte | Objektnummer | Eindeutige ID des gemessenen Objekts |
| Messobjekte | Material | Material des Objekts |
| Messobjekte | Maximale Durchstrahlungslänge | Die maximale Länge durch das Objekt in der gewählten Positionierung in der CT |
| Messobjekte | Bezeichnung | Beschreibung des Objekts |

Tabelle 3.1: Übersicht der zur Verfügung gestellten Datenbankdumps

Für ein erstes Verständnis der Daten werden die optimalen Maschinenparameter Stromstärke, Spannung, Belichtungsdauer und Filtermaterial beispielhaft für einen Stahlzylinder mit 2mm Durchmesser (Objektnummer 7) ermittelt. Als Bewertungskriterium der Messungen wird die absolute Abweichung zwischen Istwert und Sollwert festgelegt. Umso höher dieser Wert, desto schlechter sind die benutzten Maschinenparameter. Für Messungen, bei denen der Istwert mit dem Sollwert übereinstimmt, werden die Maschinenparameter als optimal angenommen. Es kann festgestellt werden, dass abhängig vom Maß verschiedene Kombinationen der Ein-

gabeparameter zu einem guten Ergebnis führen. In Abbildung 3.1 sind diese Kombinationen dargestellt. Es lässt sich erkennen, dass es keine Kombination der Parameter gibt, welche für jedes Maß zu einem optimalen Ergebnis führt. Daher muss berücksichtigt werden, dass für die Ermittlung der besten Maschinenparameter für eine anstehende Messung spezifiziert werden muss, zu welchem Maß die Parameter ermittelt werden sollen. Außerdem ist in diesem Beispiel aufgefallen, dass es für Maße, die die Zylindrizität betreffen, keine optimale Parameterkombination gibt. Die verwendeten Objekte sind in der Tabelle 3.2 dargestellt.

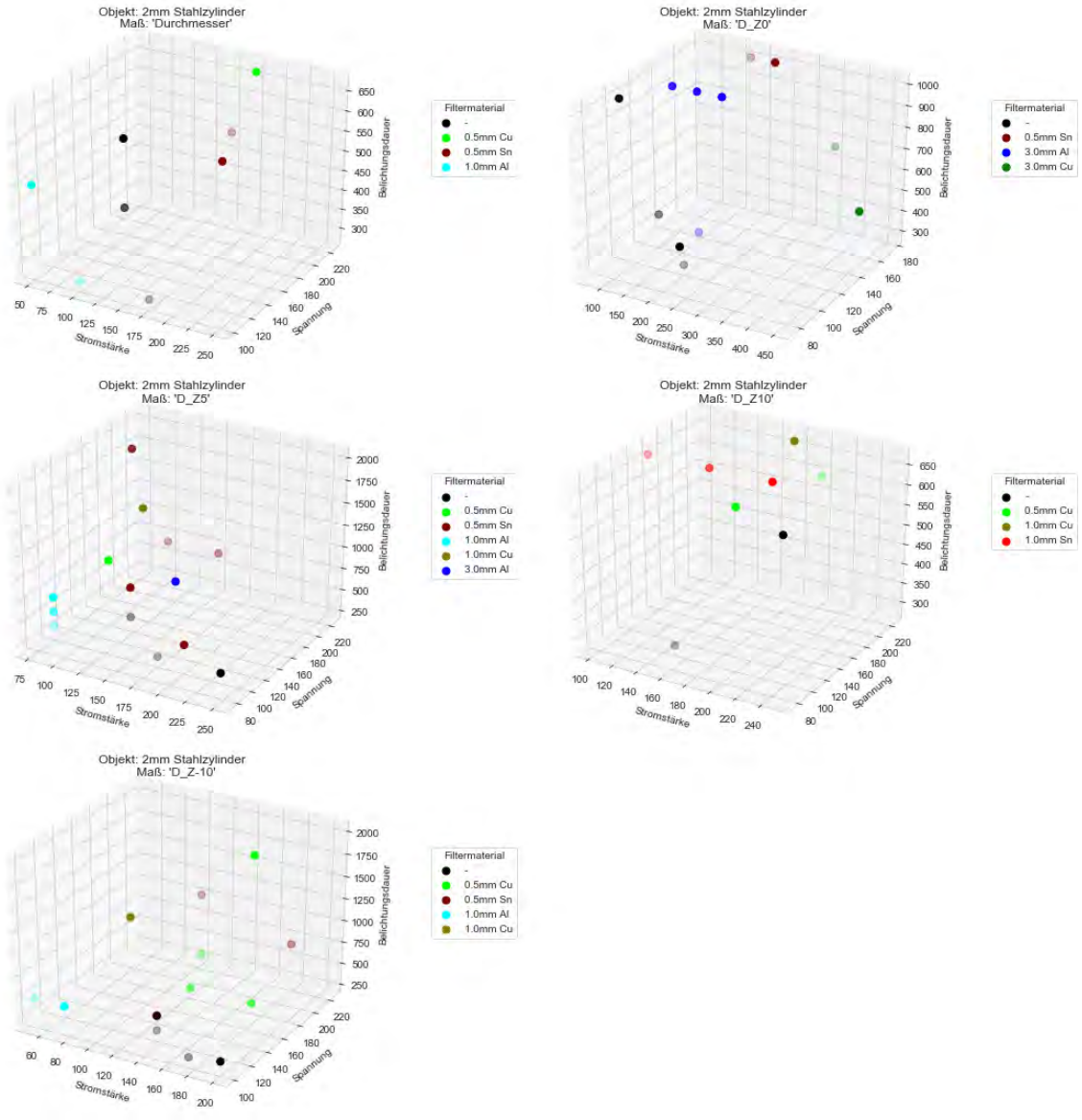



Abbildung 3.1: Optimale Kombinationen der Maschinenparameter abhängig vom gemessenen Maß für einen Stahlzylinder mit 2mm Durchmesser

### 3.1.2 Analyse der 3D CT-Scans

Zusätzlich zu den Datenbanktabellen wurden auch 3D CT-Scans in Form von REK Dateien zur Verfügung gestellt. Da eine REK Datei die Größe von ca. 1,5 Gigabyte hat, müssen diese zuerst vorverarbeitet werden. Zu jeder 3D Bilddatei werden mittels eines zur Verfügung gestellten Programms drei Projektionen (x-Achse, y-Achse, z-Achse) erstellt, welche im CSV Format abgespeichert werden. Um das Einlesen dieser Projektionsdateien zu beschleunigen werden diese in das PNG Format konvertiert. Die Projektionen entlang der x- und y-Achse haben eine Abmessung von 850×991 Pixel und die Projektion entlang der z-Achse von 991×991 Pixel. Eine Projektion im CSV Dateiformat hat nur noch die Größe von ca. 6 Megabyte und eine Projektion im PNG Format von 300 Kilobyte. Insgesamt wurden 37190 Projektionsdateien erstellt. Da bei auftretenden Fehlern während der Konvertierung von den 3D CT Aufnahmen in die 3 Projektionsebenen kein Abbruch erfolgen durfte, wurde die Konvertierung mit der nächsten Datei weitergeführt. Daher gibt es nicht zu jedem 3D CT-Scan drei zugehörige Projektionen, was bei der weiteren Verarbeitung berücksichtigt werden muss. In Abbildung 3.2 sind beispielhaft die drei Projektionen zu Messnummer 22241 dargestellt. Nachfolgend ist die Aufteilung der Projektionsdateien in P-Projektionen, S-Projektionen und T-Projektionen aufgelistet, wobei die Buchstaben P, S und T für primär, sekundär und tertiär stehen:

- 12401 P-Projektionen
- 12397 S-Projektionen
- 12392 T-Projektionen

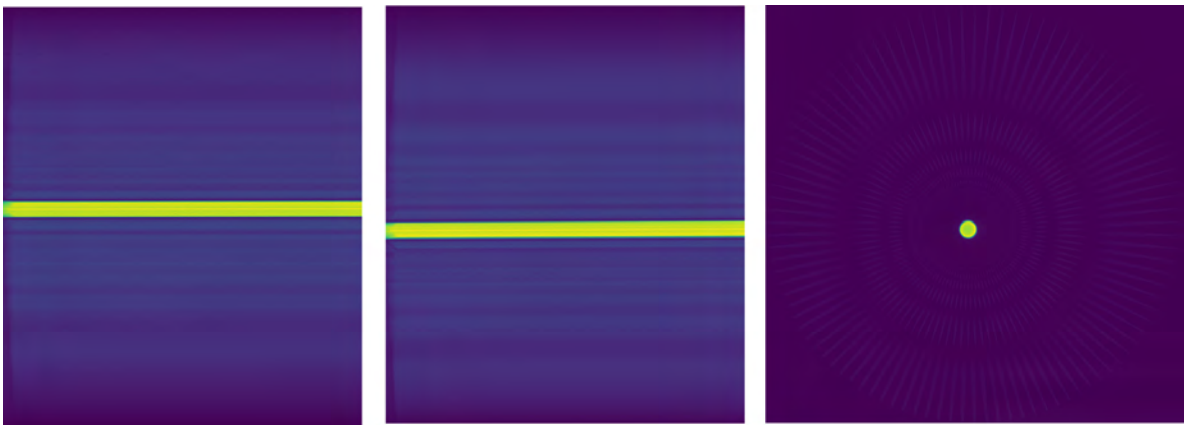


Abbildung 3.2: Projektionen zur Messung 22241

## 3.2 Datenvorverarbeitung

Um die Abweichung zwischen gemessenen Istwert und zugehörigem Sollwert ermitteln zu können, wird deren Differenz gebildet und anschließend die Absolutbeträge mit dem Faktor

1000 multipliziert. Dadurch erhält man den absoluten Fehler zwischen Istwert und Sollwert in Mikrometer ($\mu$m).

Nach Einlesen der Daten zu den Stahlzylindern, stehen insgesamt 62748 gemessene Maße zur Verfügung. Es werden alle Messungen mit einer vierstelligen Messnummer verworfen, weil es keine zugehörigen 3D CT-Scans gibt. Mit Hilfe des Interquartilabstands (IQA) werden Ausreißer bestimmt, welche dann ebenfalls aussortiert werden. Dazu wird das erste und dritte Quartil der Abweichungen bestimmt und daraus der IQA. Alle Abweichungen, welche die Bedingung 3.1 erfüllen werden auch aussortiert.

$$Error_{abs} > Q_3 + 1.5 * IQA \tag{3.1}$$

Nachdem Entfernen der Ausreißer und der Messungen, zu denen keine REK-Dateien verfügbar sind, bleiben noch 22.108 Datensätze übrig. Die Maschinenparameter Schritte und Qualität werden nicht betrachtet, da sie für jede Messung den selben Wert haben und daher keine weiterhin notwendige Information zur Verfügung stellen. In der Tabelle 3.2 ist die Verteilung der Datensätze auf die jeweiligen Messobjekte dargestellt.

| **Durchmesser [mm]** | **Objektnummer** | **Anzahl der verwendeten Datensätze** |
|---|---|---|
| 2.00 | 7 | 4570 |
| 2.50 | 33 | 1888 |
| 4.00 | 15 | 3497 |
| 4.50 | 4 | 467 |
| 5.00 | 16 | 4529 |
| 6.00 | 17 | 2442 |
| 6.50 | 24 | 384 |
| 7.00 | 18 | 2170 |
| 8.00 | 19 | 336 |
| 10.00 | 8 | 99 |

Tabelle 3.2: Anzahl der gemessenen Maße bzw. durchgeführten Messungen pro Objekt

In Abbildung 3.3 ist ein Pairplot dargestellt. Es wird der Zusammenhang zwischen den Parametern Stromstärke, Spannung, Belichtungsdauer, Filtermaterial und der absoluten Differenz von Ist- und Sollwert betrachtet. Die Diagramme auf der Hauptdiagonalen stellen das Histogramm der jeweiligen Größe dar. Histogramme geben die Häufigkeitsverteilung einer Variablen an. Es lässt sich erkennen, dass der größte Teil der Messungen mit einer Stromstärke bis maximal 250 Milliampere durchgeführt wurde. Das selbe gilt für die Belichtungsdauer mit einem Wert von 1000 Sekunden. Die Spannung wurde in größeren Intervallsprüngen verändert, da für einige Spannungsbereiche keinerlei Werte vorhanden sind. Für die Abweichungen von Ist- und Sollwert lässt sich erkennen, dass die Werte zwischen 0 und 20 Mikrometer gleich verteilt

sind, mit einem Maximum zwischen zwei Mikrometer und fünf Mikrometer. Die Nebendiagonalen stellen den Zusammenhang zwischen den jeweiligen Parametern zueinander dar. Hier lässt sich kein linearer Zusammenhang erkennen. Das belegt auch die in Abbildung 3.4 dargestellte Korrelationsmatrix. Die Matrix enthält keinerlei Werte, welche sich dem Wert 1 oder -1 nähern. Ein Wert nahe 1 würde einen positiven linearen Zusammenhang und der Wert -1 einen negativen linearen Zusammenhang bedeuten. Die Werte in dieser Matrix liegen im Intervall [-0.22, 0,32]. Die Korrelationsmatrix belegt somit die Vermutung, dass es keinen linearen Zusammenhang zwischen den Parametern gibt.

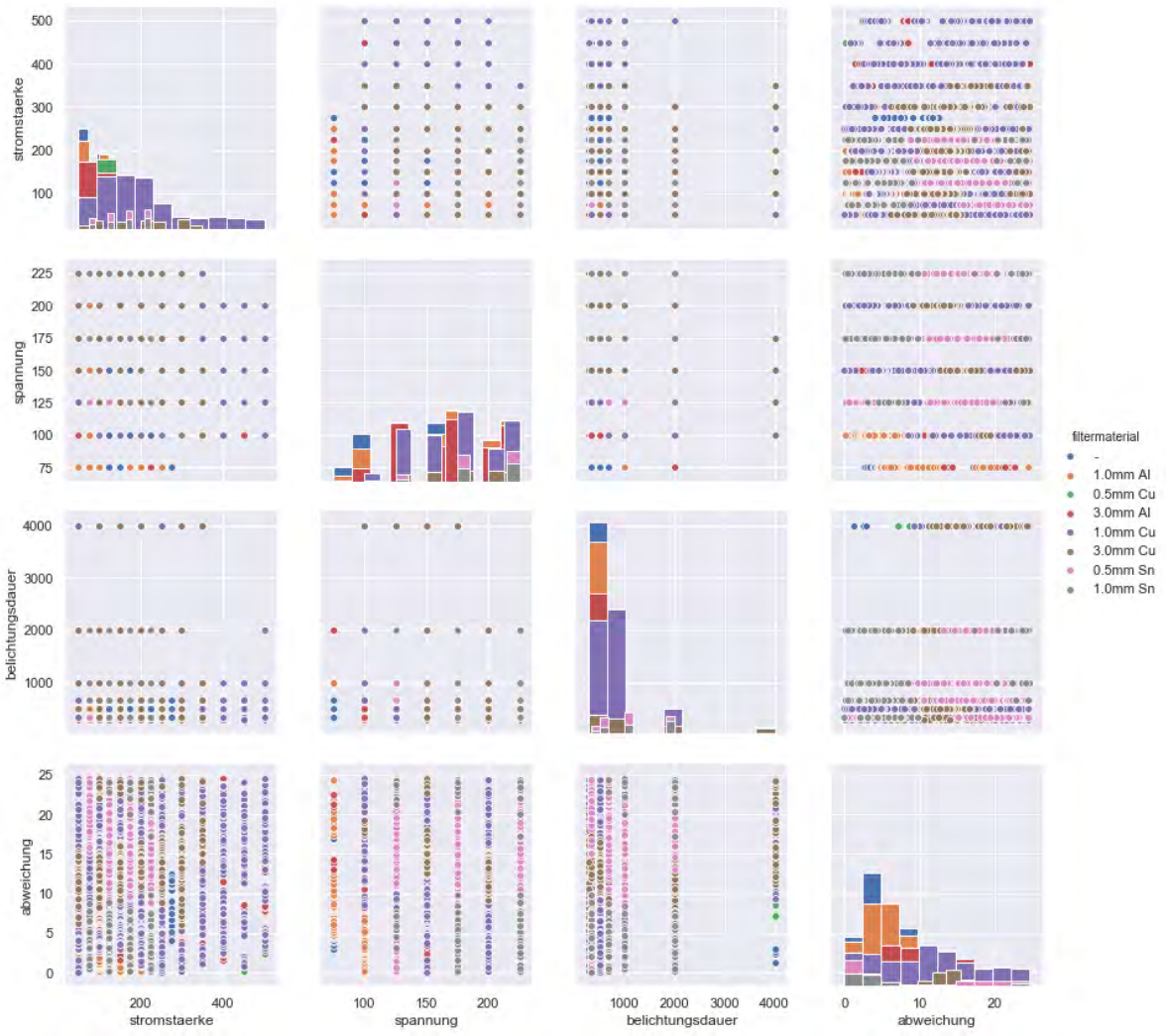



Abbildung 3.3: Zusammenhang zwischen Maschinenparamater Stromstärke, Spannung, Belichtungsdauer und der absoluten Differenz zwischen Ist- und Sollwert

| | stromstaerke | spannung | belichtungsdauer | filtermaterial | abweichung |
|---|---|---|---|---|---|
| stromstaerke | 1 | -0.22 | -0.16 | 0.32 | 0.25 |
| spannung | -0.22 | 1 | -0.15 | 0.24 | -0.14 |
| belichtungsdauer | -0.16 | -0.15 | 1 | 0.24 | 0.0062 |
| filtermaterial | 0.32 | 0.24 | 0.24 | 1 | 0.15 |
| abweichung | 0.25 | -0.14 | 0.0062 | 0.15 | 1 |

Abbildung 3.4: Korrelationsmatrix von Maschinenparameter und Abweichung zwischen Ist- und Sollwert

### 3.2.1 Verarbeitung der Messparamter

Bevor die Messdaten und deren Ergebnis aus den Datenbanktabellen verwendet werden können um ein Neuronales Netz zu trainieren, müssen noch einige Schritte zur Aufbereitung der Daten durchgeführt werden. Als Erstes findet eine Aufteilung der Daten in Trainings-, Validierungs- und Testdaten im Verhältnis von 60:20:20 statt. Mit den Trainingsdaten wird das Multilayer Perceptron bzw. das Convolutional Neural Network antrainiert. Die Testdaten werden zur Evaluation der erstellten Modelle verwendet. Die Validierungsdaten werden verwendet, um das selbe Modell mit unterschiedlichen Konfigurationen vergleichen zu können. Bei der Vorverarbeitung muss zwischen den numerischen Attributen Stromstärke, Spannung, Belichtungsdauer und den kategorischen Attributen Filtermaterial, Maß unterschieden werden.

**Numerische Attribute** Für jedes der numerischen Attribute Stromstärke, Spannung und Belichtungsdauer wird die Z-Transformation (Standardisierung) wie in Formel 3.2 durchgeführt. Das bedeutet, dass nach dieser Transformation jedes Attribut den Erwartungswert Null und die Varianz Eins besitzt.

$$Z = \frac{X - \mu}{\sigma} \tag{3.2}$$

**Kategorische Attribute** Die numerischen Attribute werden in Integerwerte transformiert und anschließend mittels One-Hot-Encoding verarbeitet. Beim One-Hot-Encoding werden für alle unterschiedlichen Klassen eines Attributs neue Spalten hinzugefügt. Jede neue Spalte entspricht genau einem Wert des kategorischen Attributs. Diese neuen Spalten sind binär, d.h. es ist eine Eins enthalten, wenn ein Datensatz zu der Klasse gehört oder eine Null, wenn er zu einer anderen Klasse gehört.

Die beiden unterschiedlichen Verarbeitungsschritte für numerische und kategorische Attribute werden zu einer Preprocessing-Pipeline zusammengefasst. Das hat den Vorteil, dass die

beiden Transformationen nur einmal anhand der Trainingsdaten durchgeführt werden müssen. Die Pipeline wird dann im PKL-Format gespeichert und kann für das Vorverarbeiten der Testdaten oder auch zur Vorverarbeitung der Trainingsdaten in einem anderen Modell wiederverwendet werden.

### 3.2.2 Verarbeitung der 3D CT-Scans

Zur Anreicherung des neuronalen Netzes mit Informationen aus den 3D-Rekonstruktionen, werden für alle Projektionsdateien geometrische Kennwerte berechnet. Diese Kennzahlen werden dem neuronalen Netz als zusätzliche Inputparameter zur Verfügung gestellt, um eine genauere Vorhersage der Messabweichung zu bekommen. Durch die Aufnahme der geometrischen Features, soll das Modell Informationen über das Messobjekt selbst erhalten. Damit die Kennzahlen ermittelt werden können, muss für jede Projektionsdatei eine Vorverarbeitung stattfinden. Zu dieser zählt das Binarisieren mit einem geeigneten Schwellwert und das Ausfiltern von Störsignalen mittels eines Median Filters.

In einem ersten Schritt werden die einzelnen Pixelwerte der CSV-Projektionsdateien auf das Intervall 0 - 255 skaliert. Die CSV-Dateien entsprechen dann einem Graustufenbild. Um die Projektionen anschließend binarisieren zu können, muss ein geeigneter Schwellwert ermittelt werden. Dieser Wert legt die Grenze fest, ab welchem Wert einem Pixel der Wert 0 und ab wann der Wert 1 zugewiesen wird. Für die Bestimmung des Schwellwerts wird der Otsu Algorithmus[25] verwendet. Abschließend erfolgt die Anwendung des Median Filters. Nach dieser Vorverarbeitung der Projektionsdateien werden die nachfolgenden Kennzahlen berechnet. Diese Kennzahlen sollen es dem neuronalen Netz ermöglichen, die zu verarbeitenden Objekte zu unterscheiden. Die geometrischen Features werden zur Erweiterung des Basis-Modells als zusätzliche Eingabeparameter zur Verfügung gestellt. Es werden die nachfolgenden Features berechnet.

**Masse** Die Masse des Objekts soll Aufschluss darüber geben, wie gut das Objekt durchstrahlt wurde. Zur Berechnung der Masse werden alle Pixelwerte addiert.

$$M = \sum_{i=1}^{n*m} p_i \tag{3.3}$$

**Höhe und Breite** Um die Höhe und Breite des in der Projektion dargestellten Objekts zu ermitteln, wird das minimal umschließende Rechteck zu dem Objekt ermittelt. Die Höhe und Breite des Objekts entsprechen den Maßen des Rechtecks.

**Momente 1. Ordnung** Momente entsprechen in der Bildverarbeitung gewichteten Pixelwerten und dienen dazu Objekte in einem Bild zu beschreiben. Mit Hilfe von Momenten lassen

sich zum Beispiel die Fläche, der Schwerpunkt und die Ausrichtung eines Objekts berechnen. Daraus ergibt sich für Binärbilder, dass das Moment $M_{00}$ die Fläche bzw. für Grauwertbilder, die Summe der Grauwerte angibt. Die zweidimensionalen Momente (p+q)-ten Grades sind wie folgt definiert:

$$m_{pq} = \int_{\infty}\int_{\infty} x^p * y^q f(x,y)dxdy \qquad \text{mit p,q} = 0,1,2,... \tag{3.4}$$

**Radius** Zur Ermittlung des Radius eines Objekts wird ähnlich wie bei der zuvor erwähnten Berechnung von Höhe und Breite vorgegangen. Es wird der minimal umschließende Kreis um das Objekt gebildet. Der Radius entspricht dem Radius des Kreises.

**Ausdehnung** Unter Ausdehnung versteht man das Verhältnis zwischen der tatsächlichen Fläche des Objekts und der Fläche des umschließenden Rechtecks.

$$Ausdehnung = \frac{A_{Obj}}{A_R} \tag{3.5}$$

**Solidität** Unter Solidität versteht man das Verhältnis zwischen der tatsächlichen Fläche des Objekts und der Fläche der konvexen Hülle. Die konvexe Hülle ist die minimale Kurve, die das Objekt umschließt.

$$Solidität = \frac{A_{Obj}}{A_{konvex}} \tag{3.6}$$

**Flächendurchmesser** Der Flächendurchmesser ist der Durchmesser eines Kreises, der die selbe Fläche hat, wie das Objekt selbst

$$d_A = \sqrt{\frac{4 * A_{Obj}}{\pi}} \tag{3.7}$$

**Intensität** Intensität bezeichnet den durchschnittliche Pixelwert eines Bildes. Das ist bei Farbbildern der durchschnittliche Farbwert bzw. bei Graubildern der durchschnittliche Grauwert.

## 3.3 Erstellung eines Multilayer Perceptrons zur Vorhersage der Messabweichung

In diesem Abschnitt wird ein Multilayer Perceptron (MLP) beschrieben, welches anhand der Parameter Stromstärke, Spannung, Belichtungsdauer, Filtermaterial und Maß die absolute Abweichung in Mikrometer zwischen Ist- und Sollwert einer Messung vorhersagt. In Abbildung 3.5 ist der Aufbau des MLP dargestellt. Es besteht aus einem Input Layer, zwei

Hidden Layern und einem Output Layer. Der Input Layer besteht es 23 Neuronen, was genau der Anzahl der zu verarbeitenden Attribute entspricht. Diese sind die numerischen Features Stromstärke, Spannung, Belichtungsdauer und die one-hot-encodierten kategorischen Attribute Filtermaterial und Maß. Es gibt 8 verschiedene Filtermaterialien und 12 unterschiedliche Maße. Bei den beiden Hidden Layern handelt es sich um Fully-Connected Layer (Dense Layer) mit jeweils 64 Neuronen. Als Aktivierungsfunktion wird die RELU-Funktion verwendet. Jede der 23 Eingangsgrößen des Input Layers ist mit jedem der 64 Neuronen des ersten Hidden Layer verbunden. Auch die Neuronen der beiden Hidden Layer sind alle miteinander verbunden. Der Output Layer besteht aus einem Neuron, welches mit den 64 Neuronen des zweiten Hidden Layers verbunden ist. Der Output Layer verwendet die lineare Identitätsfunktion als Aktivierungsfunktion. Das MLP besitzt insgesamt 5761 trainierbare Parameter. Die maximale Anzahl an Trainingsepochen für das MLP ist auf 1000 beschränkt. Es ist aber zusätzlich ein Stoppmechanismus vorhanden, der dafür sorgt, dass das Training des MLPs abgebrochen wird, wenn sich der Validierungsfehler über zehn Epochen nicht mehr verbessert. Dieser Mechanismus soll das Overfitting des neuronalen Netzes beschränken.

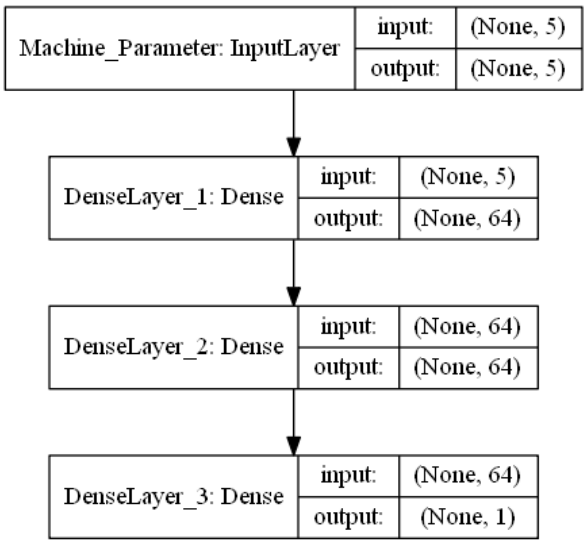



Abbildung 3.5: Aufbau des MLP zur Vorhersage der absoluten Abweichung

### 3.3.1 Erweiterung des Modells mit zusätzlichen Parametern

Im vorher beschriebenen MLP wurde die Leistung nicht berücksichtigt, da diese abhängig von den beiden Maschinenparametern Stromstärke und Spannung ist. In einem erneuten Trainingsversuch des MLP wird die Leistung dann als zusätzlicher Inputparameter mitaufgenommen. Es wird angenommen, dass der zusätzliche Parameter weitere Informationen für das Netzwerk bereitstellen kann, obwohl es sich um eine redundante Information handelt. In der Abbildung 3.6 ist der Zusammenhang zwischen dem neuen Parameter Leistung und den zuvor bereits verwendeten Maschinenparametern zu sehen. Es lässt sich der bekannte lineare

Zusammenhang mit den beiden Größen Stromstärke und Spannung erkennen. Außerdem fällt ebenfalls auf, dass es eine negative Korrelation zwischen Leistung und Belichtungsdauer gibt. Betrachtet man hingegen die Korrelationskoeffizienten von Leistung und den restlichen Maschinenparametern in Abbildung 3.7, stellt man fest, dass die zweitgrößte Korrelation zwischen Leistung und Filtermaterial besteht. Dagegen ist die vermutete Korrelation zur Belichtungsdauer als nicht relevant einzustufen. Die Anzahl der trainierbaren Parameter des Netzes hat sich durch hinzufügen des neuen Inputs Leistung auf 5825 erhöht.


Abbildung 3.6: Korrelation des zusätzlichen Parameters Leistung zu den bereits verwendeten Größen

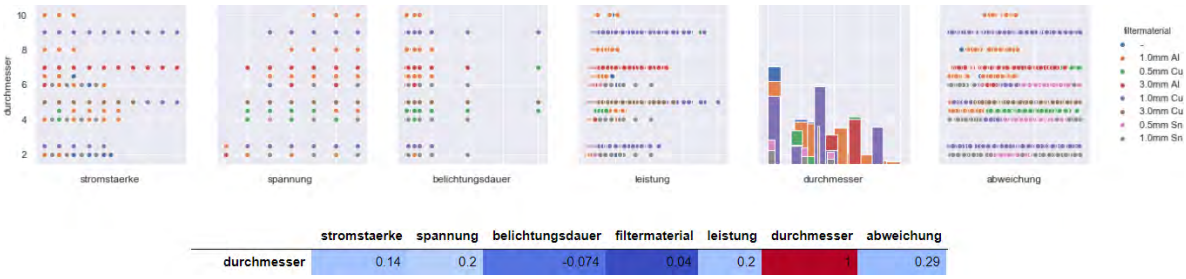
| | stromstaerke | spannung | belichtungsdauer | filtermaterial | leistung | abweichung |
|---|---|---|---|---|---|---|
| leistung | 0.88 | 0.15 | -0.16 | 0.43 | 1 | 0.15 |

Abbildung 3.7: Korrelationskoeffizienten des zusätzlichen Parameters Leistung zu den bereits verwendeten Größen

In einem nächsten Schritt wird dem MLP ein weiterer zusätzlicher Inputparameter mitgegeben. Dieser Parameter ist der Durchmesser des gescannten Stahlzylinders. Durch Hinzufügen des Durchmessers soll das MLP berücksichtigen, dass es sich bei den Objekten um Stahlzylinder unterschiedlicher Größe handelt und eine optimale Maschinenparameterkombination davon abhängig ist. Der zusätzliche Inputparameter erhöht die Anzahl der zu trainierenden Netzwerkparameter auf 5889. Abbildung 3.8 stellt den Zusammenhang zwischen dem Parameter Durchmesser und den bereits im Modell enthaltenen Parametern dar. Außerdem sind auch die Einträge der Korrelationsmatrix zu dem Parameter Durchmesser dargestellt. Es lässt sich feststellen, dass kein linearer Zusammenhang zwischen einem der Attribute und dem Durchmesser besteht.

### 3.3.2 Erweiterung des Modells mit geometrischen Eigenschaften

Das neuronale Netz soll unabhängig von der Eingabe des zu messenden Objekts sein, daher wird der Parameter Durchmesser nicht weiter berücksichtigt und stattdessen die in Kapitel 3.2.2 berechneten geometrischen Features als zusätzliche Eingabeparameter hinzugefügt. Diese Features werden aus den Projektionen berechnet und sollen es dem Netz ermöglichen, unabhängig vom tatsächlichen Objekt die Vorhersage der Messabweichung zu verbessern. Diese Parameter sind Masse, Höhe, Breite, 1. Moment, 2. Moment, Radius, Ausdehnung, Solidität,




| | stromstaerke | spannung | belichtungsdauer | filtermaterial | leistung | durchmesser | abweichung |
|---|---|---|---|---|---|---|---|
| durchmesser | 0.14 | 0.2 | -0.074 | 0.04 | 0.2 | 1 | 0.29 |

Abbildung 3.8: Korrelation des zusätzlichen Parameters Durchmesser zu den bereits verwendeten Größen

Flächendurchmesser und Intensität. Diese 10 geometrischen Features werden für jede Projektion berechnet. Das führt dazu, dass das Netz insgesamt 30 neue Eingabeparameter erhält. Der Input Layer besteht aus 54 Eingabeparametern. Die Anzahl der trainierbaren Parameter des MLPs steigt auf 7745.

## 3.4 Erstellung eines Convolutional Neural Networks zur Vorhersage der Messabweichung

Nachdem ein Basis MLP erstellt wurde, welches in 3 Schritten mit zusätzlichen Eingabeparametern erweitert wurde, werden nun zwei Convolutional Neural Network Modelle erstellt. Das erste Modell wird als Single-Input Modell und das Zweite als Multiple-Input Modell bezeichnet. Da es zu einer Messung nur eine 3D-Rekonstruktion, aber mehrere gemessene Maße und dadurch auch mehrere Messabweichungen gibt, verwenden diese Modelle nur Messungen mit dem Maß D_Z5. Ansonsten würden den beiden CNNs verschiedene Messabweichungen für die selben Projektionen zur Verfügung stehen, was das Training unmöglich macht. Für die Verarbeitung der Projektionen in CNNs werden diese durch Downsampling auf eine Größe von 64×64 Pixel gebracht.

### 3.4.1 Das Single-Input-CNN Modell

Beim Single-Input-CNN Modell werden die drei Projektionen zu einem Bild zusammengesetzt, welches dann die Größe von 192×64 hat. Für das Zusammenfügen der Projektionen gibt es sechs verschiedene Möglichkeiten. Es werden alle sechs möglichen Permutationen zum Trainieren des Netzes verwendet. Das erhöht zum Einen die Datenmenge für das Training und zum Anderen die Robustheit des Netzes. Für das CNN soll es nicht von Bedeutung sein, wie das Eingabebild zusammengefügt wurde, sondern welches Objekt abgebildet ist. In Abbildung 3.9 ist die Struktur des Single-Input-CNNs dargestellt. Das Netz besteht aus vier Convolution Layern, vier Pooling, Layern einem Flatten Layer und drei fully-connected Layern. Neben den Schichten ist die Dimension der Eingabe und Ausgabe einer Rechenoperation angegeben. Auf

jeden Convolution Layer folgt immer ein Pooling Layer. Diese Schema wird viermal wiederholt und dient der Feature Extraction. Der Flatten Layer transformiert die dreidimensionalen Daten in einen Vektor, der von den fully-connected Layern verarbeitet werden kann. Diese Layer berechnen aus den extrahierten Features die Messabweichung. Der zweite Convolution Layer verwendet 64 Filter, die anderen drei jeweils 32, wobei alle Filter die Größe 3×3 und die feste Schrittweite eins haben. Die Padding Funktionalität der Convolution Layer sorgt dafür, dass die Größe nach der Verarbeitung beibehalten wird. Als Aktivierungsfunktion wird im letzten Layer die Identitätsfunktion verwendet. Alle anderen Layer benutzen die ReLU Funktion. Der Pooling Layer verwendet ein Fenster der Größe 2×2 und eine feste Schrittweite von zwei. Die drei fully-connected Layer bestehen aus 16 Neuronen, 4 Neuronen bzw. 1 Neuron. Das Single-Input-CNN Modell besitzt insgesamt 71193 trainierbare Parameter.

### 3.4.2 Das Multiple-Input-CNN Modell

Im Gegensatz zum Single-Input-CNN Modell wird beim Multiple-Input-CNN Modell für jede der Projektionsdateien zu einer Messung ein Inputparameter verwendet. Da aber auch hier die Projektionen an den Eingängen vertauscht werden können, werden auch bei diesem Modell die sechs Möglichkeiten zur Zuordnung von Projektionsdatei und Inputparameter berücksichtigt. Wie bereits erwähnt, hilft dieses Vorgehen zur Erhöhung der Datenmenge und der Robustheit des Convolutional Neural Networks. Die Struktur dieses Modells und die Dimension vor und nach einer Verarbeitungsschicht sind in Abbildung 3.10 dargestellt. In diesem Modell findet die Feature Extraction für jede Projektionsdatei parallel statt. Es werden die gleichen Layer mit den selben Einstellungen wie im Single-Input Modell verwendet, um die beiden CNNs besser miteinander vergleichen zu können. Jeder Zweig des Netzes besteht aus jeweils 4 abwechselnden Covolution und Pooling Layern. Nach der Feature Extraction werden die Zweige des Netzes mit dem Concatenate Layer zusammengefügt. Aus den drei zu verarbeitenden Datensätzen mit der jeweiligen Dimension 4×4×32 wird ein Datensatz mit der Dimension 4×4×96. Anschließend wird die gleiche Verarbeitungslogik durchlaufen wie im Single-Input Modell. Das Modell besteht aus 3 Input Layern, 12 Convolution Layern, 12 Pooling Layern, einem Concatenate Layer, einem Flatten Layer und 3 Dense Layern, wobei der letzte Dense Layer der Output Layer ist. Das Multiple-Input-CNN Modell besitzt insgesamt 164249 trainierbare Parameter.

## 3.5 Zusammenfügen des Multilayer Perceptrons und des Convolutional Neural Networks

In einem letzten Schritt werden die beiden Convolutional Neural Networks Modelle jeweils mit dem zu Beginn erstellten Multilayer Perceptron Modell verknüpft. Es werden die Ein-

gabeparameter Stromstärke, Spannung, Belichtungsdauer, Filtermaterial, Maß und Leistung verwendet.

### 3.5.1 Das Single-Input-CNN-MLP Modell

Das Single-Input-CNN-MLP Modell ist in Abbildung 3.11 dargestellt. Die Projektionsdateien werden für dieser Modell wieder zusammengefügt, so dass sich eine Bildgröße von 192×64 Pixel ergibt. Es werden auch wieder alle Permutationen der möglichen Zusammensetzungen gebildet. Das Modell besteht aus 2 Input Layern, 4 Convolution Layern, 4 Pooling Layern, einem Flatten Layer, einem Concatenate Layer und 6 Dense Layern. Es werden die gleichen Einstellungen für die jeweiligen Schichten benutzt, wie in den vorherigen CNN Modellen bzw. dem MLP Modell. Nach der Feature Extraction werden die Ergebnisse mit denen aus dem MLP, welches die Maschinenparameter verarbeitet, zusammengeführt. Danach folgen 4 fully-connected Layer mit 128 Neuronen, 64 Neuronen, 4 Neuronen und einem Neuron. Die Convolution Layer und alle Dense Layer bis auf den Letzten verwenden die ReLU Funktion als Aktivierungsfuktion. Der letzte Dense Layer, der zugleich die Ausgabeschicht darstellt, verwendet die Identitätsfunktion als Aktivierungsfunktion. Das Single-Input-CNN-MLP Modell hat insgesamt 175689 trainierbare Parameter.

### 3.5.2 Das Multiple-Input-CNN-MLP Modell

Das Multiple-Input-CNN-MLP Modell ist in Abbildung 3.12 dargestellt. Dieses Modell benötigt als Eingabe die drei Prjektionsdateien einer Messung in der Größe 64×64 Pixel und die Parameter Stromstärke, Spannung, Belichtungsdauer, Filtermaterial, Maß und Leistung. Zur Erhöhung der Robustheit werden auch für dieses Modell die sechs Kombinationen der Zuordnung von Bilddatei zu Inputparameter gebildet. Das Modell besteht aus 4 Input Layern, 12 Convolution Layern, 12 Pooling Layern, 2 Concatenate Layern, einem Flatten Layer und 6 Dense Layern. Dieses Modell verwendet die selbe Konfiguration der Layer wie das Single-Input-CNN-MLP Modell. Die Ergebnisse aus dem Teil der Feature Extraction werden in einen Vektor transformiert und mit den Ergebnissen des MLP verknüpft. Anschließend wird die selbe Verarbeitungs- und Ausgabelogik durchlaufen, wie in dem vorher genannten Modell. Ingesamt hat das Modell 268745 trainierbare Parameter.

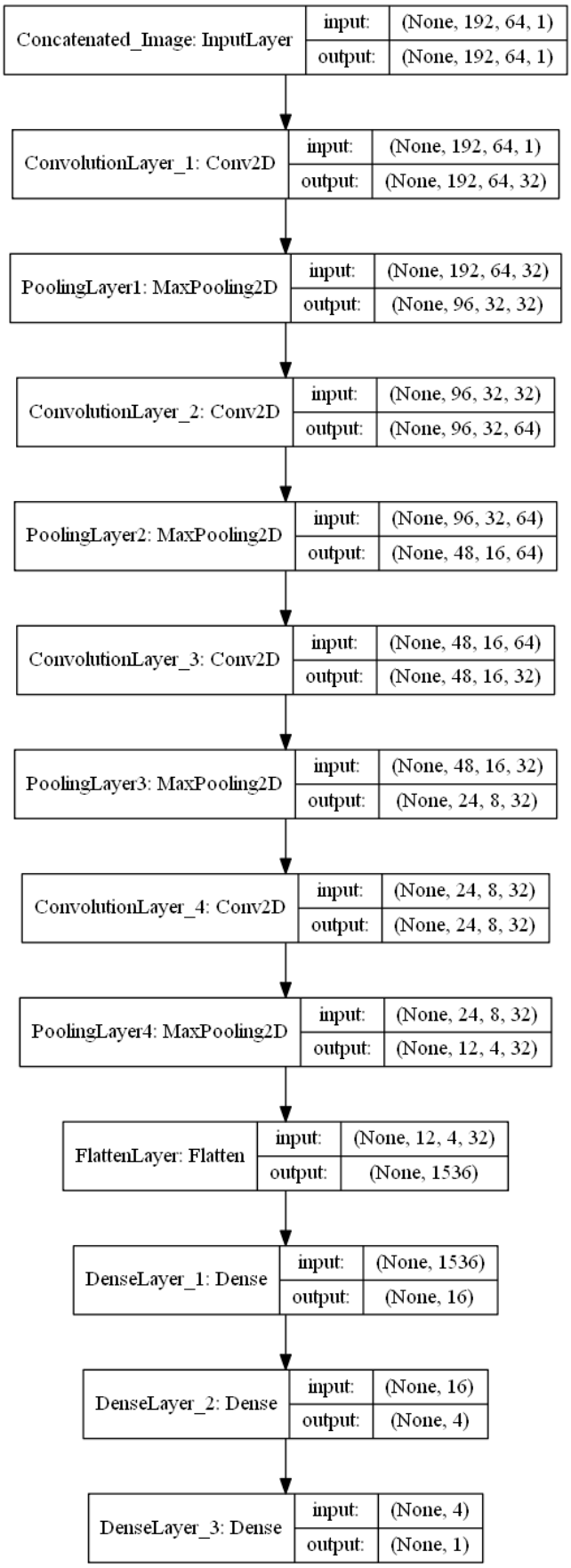



Abbildung 3.9: Aufbau des Single-Input CNN

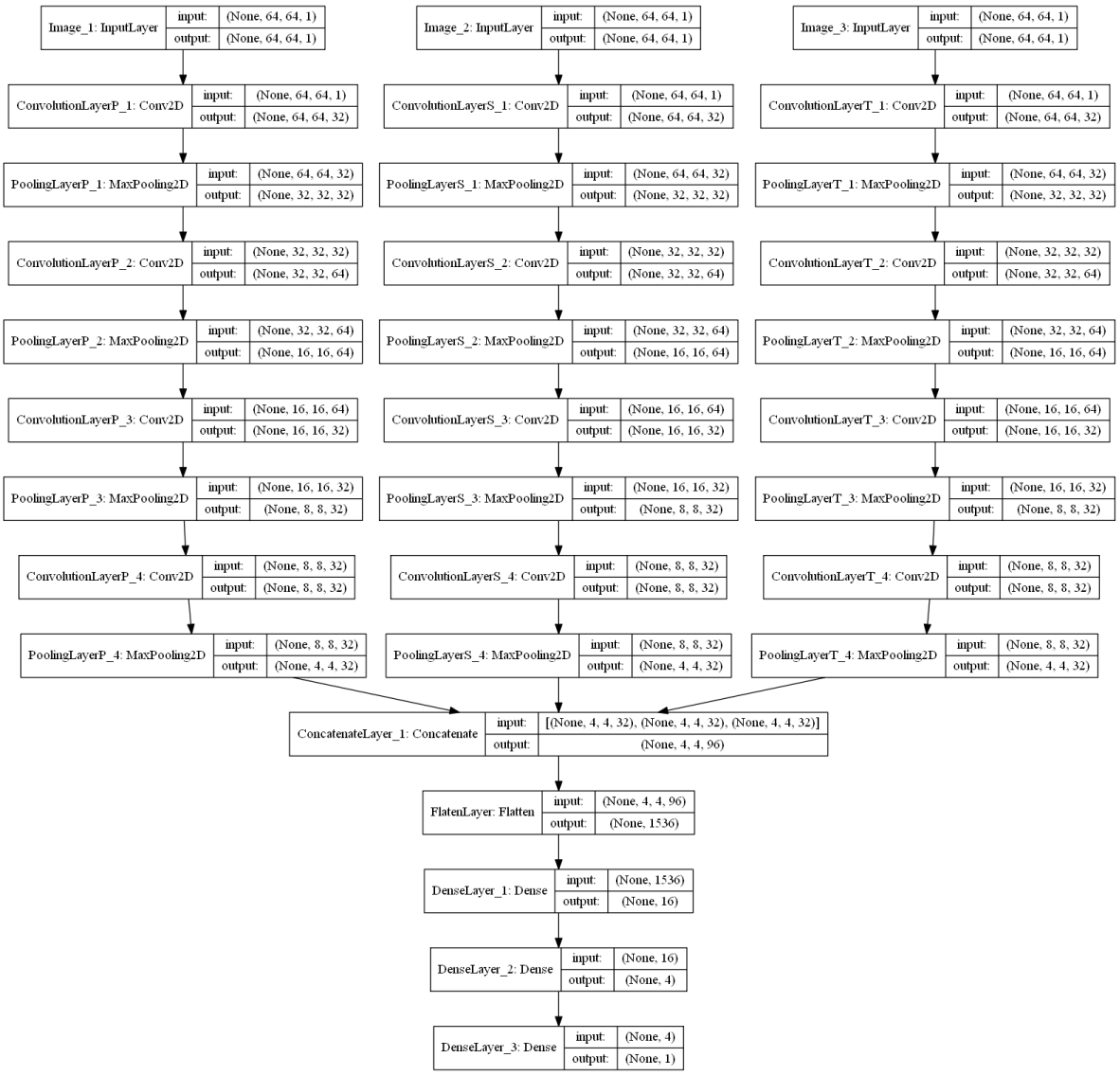



Abbildung 3.10: Aufbau des Multiple-Input CNN Modells

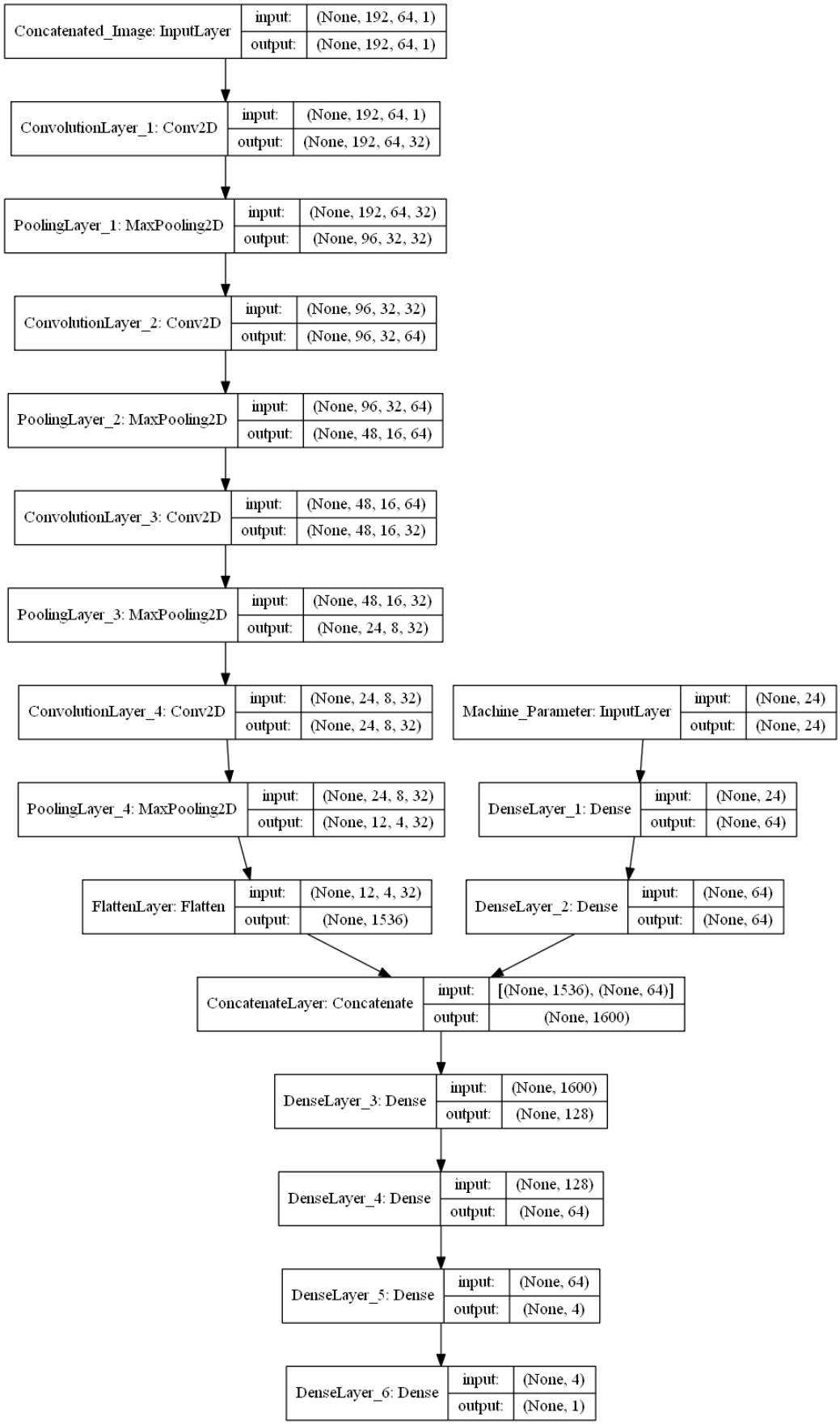



Abbildung 3.11: Aufbau des Single-Input-CNN-MLP Modells

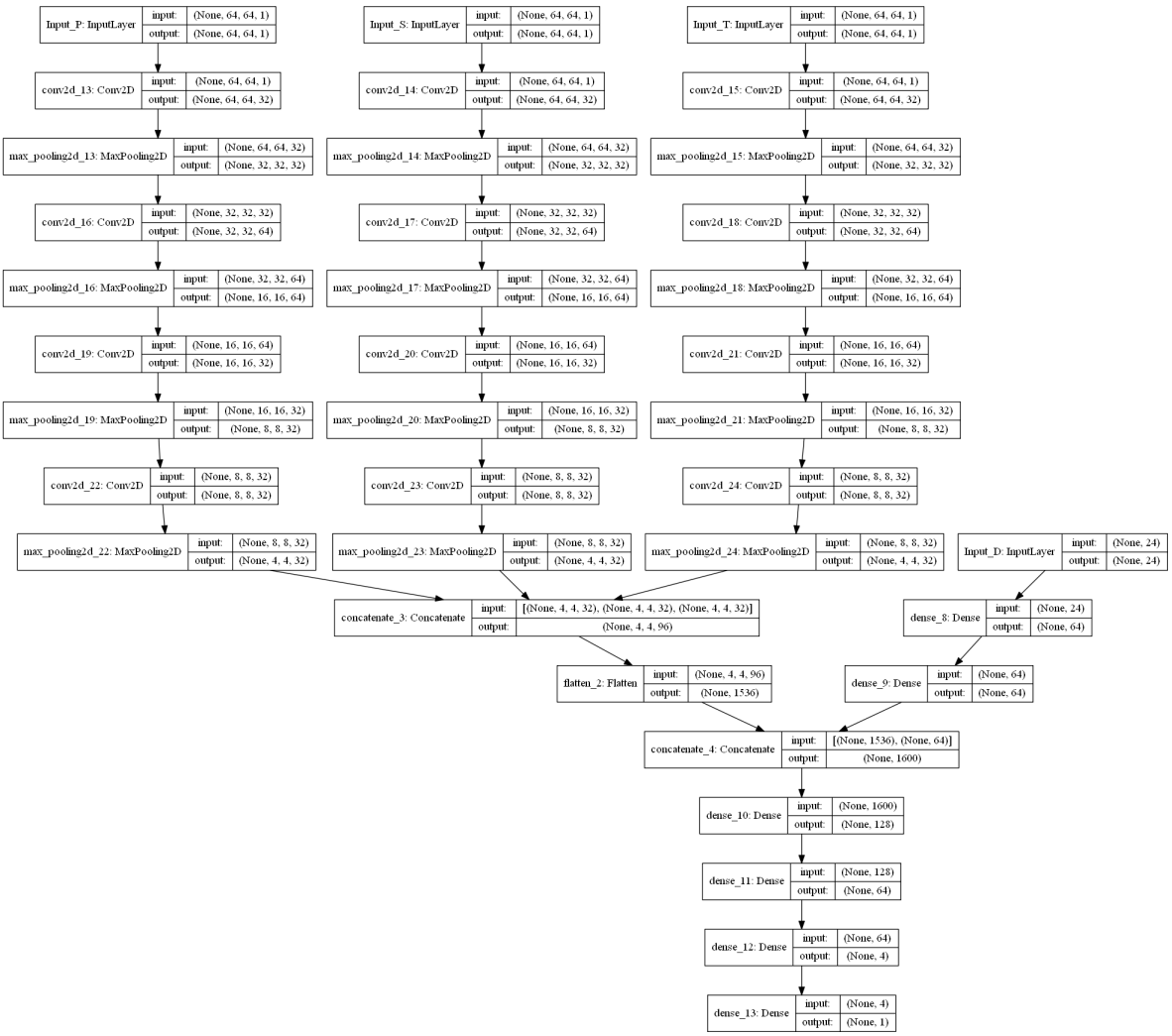



Abbildung 3.12: Aufbau des Multiple-Input-CNN-MLP Modells

# 4 Implementierung

Nachdem zuvor die Modelle beschrieben wurden, soll in diesem Kapitel darauf eingegangen werden, welche Hardware- und Softwarekomponenten zur Erstellung der Modelle benutzt werden. Dafür wird zuerst die Systemkonfiguration des benutzten Rechners genannt und anschließend die eingesetzten Python-Bibliotheken erläutert. Es werden nur die verwendeten Komponenten beschrieben. Für detaillierte Informationen ist das Coding zu betrachten. Im Anhang ist eine Übersicht der erstellten Programme enthalten.

## 4.1 Systemkonfiguration

Für die Implementierung und das Trainieren der neuronalen Netze wurde ein Laptop mit folgenden Hardware- bzw. Softwarekomponenten verwendet:

- Prozessor: Intel(R) Core(TM) i7-6600 CPU @ 2.60 GHz
- Arbeitsspeicher: 16 GB RAM
- Festplatte: 256 GB SSD
- Grafikkarte: Intel(R) HD Graphics 520
- Betriebssystem: Windows 7 64-bit

Die Implementierung zum Trainieren der verschiedenen Netze wird mit der Programmiersprache Python in Version 3.7 umgesetzt. Zum Organisieren der verschiedenen, benötigten Pakete wird Anaconda benutzt. Diese Software bietet einen leichten Einstieg in die Programmiersprache Python und stellt den industriellen Standard dar, wenn es um die Entwicklung, das Testen oder Trainieren auf einem einzelnen Computersystem geht. Anaconda stellt mit Conda ein eigenes Paketverwaltungssystem, die Entwicklungsumgebung Spyder, den Kommandozeilenintepreter IPython und ein webbasiertes Frontend Jupyter Notebook zur Verfügung. Alle erstellten Programme im Rahmen dieser Arbeit werden mit Jupyter Notebook entwickelt.

## 4.2 Verwendete Python-Bibliotheken für Machine Learning

Zur Umsetzung in Python ist es notwendig Bibliotheken einzubinden. In dieser Arbeit werden die Bibliotheken TensorFlow 1.13.1, Keras 2.2.4 und Scikit-Learn 0.20.1 zur Implementie-

rung und Evaluation von neuronalen Netzen, insbesondere Convolutional Neural Networks, eingesetzt.

### 4.2.1 TensorFlow

TensorFlow[26] ist ein open-source Framework von Google, das zur Programmierung von vielen Machine Learning Algorithmen dient. Dieses Framework bildet die Basis für Anwendungen von neuronalen Netzen im Rahmen der Bild- und Sprachverarbeitung. In dieser Arbeit wird es als Backend für das Training der erstellen Modelle verwendet. TensorFlow kann sowohl auf Central Processing Units (CPU), als auch auf GPUs ausgeführt werden. Diese Bibliothek stellt neuronale Netze mit Hilfe von gerichteten, zyklenfreien Graphen dar. Die Kanten eines solchen Graphen stellen die Eingabe bzw. Ausgabe dar. Die Knoten bilden die Verarbeitungseinheiten und entsprechen somit den Neuronen in einem künstlichen neuronalen Netz. TensorFlow benutzt für die Rechenoperationen mehrdimensionale Datenarrays (sog. Tensoren). Ein Vektor entspricht einem Tensor 1. Ordnung, eine Matrix einem Tensor 2. Ordnung und eine dreidimensionale Matrix einem Tensor 3. Ordnung. TensorFlow bietet vielzählige Anwendungsmöglichkeiten in der Praxis und wird zum Beispiel von PayPal zur Betrugserkennung eingesetzt.

### 4.2.2 Keras

Keras[27] ist ein Application Programming Interface (API) für neuronale Netze, das auf TensorFlow und anderen Machine Learning Frameworks wie CNTK oder Theano aufsetzt. Keras unterstützt die Implementierung von Convolutional Neural Networks sowie die Verarbeitung auf GPU und CPU Level. Ein weiterer Vorteil dieser API ist die Möglichkeit zur Erstellung von Multi-Input und Multi-Output Netzen. In dieser Arbeit wird Keras zur Erstellung der neuronalen Netze benutzt. Die API stellt alle benötigten Layer als Klassen zur Verfügung, so dass nur die Hyperparameter eingestellt werden müssen. Außerdem ist möglich mit sog. Callbacks in das Training des neuronalen Netzes einzugreifen. Es können individuelle Funktionen erstellt werden, die während des Trainingsprozesses ausgeführt werden können. Die Klasse EarlyStopping wird verwendet, um den Trainingsprozess zu beenden, falls für eine gewisse Anzahl an Trainingsepochen keine Verbesserung des Validierungsfehlers festzustellen ist. Keras stellt die verschiedenen Optimierungsstrategien des SGD bereits fertig implementiert zur Verfügung und bietet zusätzliche Hilfsfunktionen, wie zum Beispiel die graphische Darstellung eines erstellten Netzes.

### 4.2.3 Scikit-Learn

Scikit-Learn[28] ist eine Bibliothek für die Programmiersprache Python und wird zum maschinellen Lernen eingesetzt. Sie enthält eine Vielzahl an Machine Learning Algorithmen, die

in verschiedene Teilbibliotheken, je nach Aufgabenstellung, unterteilt sind. Zu diesen Aufgabegebieten gehören Klassifikation, Regression, Clustering, Dimensionsreduktion und Preprocessing. In dieser Arbeit wird Scikit-Learn zum Preprocesinng und zur Evaluierung eingesetzt. Die Bibliothek stellt zum Beispiel den verwendeten StandardScaler und die Möglichkeit zur Erstellung von Pipelines für die Datenvorverarbeitung bereit. Zur Evaluation werden die Metriken MSE, MAE und R2-Score der Scikit-Learn Bibliothek verwendet. Außerdem bietet Scikit-Learn eine besonders einfache Funktionalität zur Aufteilung von Datensätzen in Trainings-, Validierungs- und Testdaten.

# 5 Evaluation

In diesem Kapitel werden die in Kapitel 3 vorgestellten Modelle ausgewertet. Dabei wird Auf den Verlauf des Trainingsfehler und des Validierungsfehlers eingegangen. Außerdem wird ein Vergleich zwischen vorhergesagten Werten und tatsächlichen Werten gezeigt. Abschließend wird am Beispiel einer Messung die Vorhersage der Maschinenparameter durchgeführt.

Als Kostenfunktion zum Trainieren der Modelle wird der Mean Squared Error benutzt, welcher mit dem in Abschnitt 2.1.4.2 vorgestellten Lernalgorithmus Adam Optimizer minimiert wird. Jedes Netz soll für maximal 1000 Epochen trainiert werden. Zur Reduzierung des Overfitting Problems wird das Training abgebrochen, wenn sich der MSE der Validierungsdaten über die Dauer von 10 Epochen nicht mehr verbessert. Die jeweils vom Modell verwendeten Daten werden im Verhältnis 60:20:20 in Trainingsdaten, Validierungsdaten und Testdaten aufgeteilt. Zusätzlich zum MSE wird auch der Mean Absolute Error für die Trainings- und Validierungsdaten ausgewertet. Mit Hilfe der Testdaten wird untersucht, wie sich die Modelle verhalten, wenn sie unbekannte Daten verarbeiten sollen. Hierfür wird zusätzlich zu MSE und MAE auch die R2-Score betrachtet.

## 5.1 Trainingsdauer

In Tabelle 5.1 ist die Gesamtanzahl der zur verwendeten Daten je Modell sowie deren Aufteilung dargestellt. Die Werte in Klammern entsprechen der Anzahl an Daten, die für das Training zur Verfügung stehen, nachdem alle Permutationen der Projektionen gebildet wurden. In der Abbildung 5.1 ist die benötige Gesamttrainingsdauer in Minuten und die benötigte Anzahl an Trainingsepochen abhängig vom jeweiligen Modell aufgezeichnet. Für die MLP Modelle dauert das Training ungefähr eine Minute, obwohl die benötigten Trainingsepochen zwischen 50 und 90 liegen. Die Schwankung der Trainingsepochen lässt sich durch die zufällige Initialisierung der Verbindungsgewichte erklären. Auch die beiden CNN Modelle liegen mit ungefähr 2 Stunden Trainingsdauer gleich auf, wobei das Single-CNN nach 81 Epochen das Training beendet und das Multiple-CNN Modell nach 117, was den Maximalwert aller Modelle darstellt. Die CNN-MLP Modelle haben ein ähnliches Verhalten. Sie benötigen beide ungefähr 5 Stunden bis das Training nach 20 bzw. 25 Epochen abgeschlossen ist. Es ist festzustellen, dass mit einer sprunghaften Erhöhung der trainierbaren Netzwerkparameter auch die Trainingsdauer entsprechend zunimmt. Kleine Parameteränderungen sind in der Trainingsdauer nicht zu bemerken. Die minimalen Werte der Trainingsepochen bei den tiefen Netzen lässt

sich durch die erhöhte Anzahl an Trainingsdaten erklären. Mehr Trainingsdaten führen dazu, dass weniger Epochen benötigt werden.

| | **Trainingsdaten** | **Validerungs-daten** | **Testdaten** | **Gesamtdaten** |
|---|---|---|---|---|
| **MLP Modelle** | 13264 | 4422 | 4422 | 22108 |
| **CNN Modelle** | 1172 (7033) | 391 (2345) | 391 | 1954 |
| **CNN-MLP Modelle** | 12660 (75960) | 4220 (25320) | 4220 | 21100 |

Tabelle 5.1: Aufteilung und Gesamtanzahl der verwendeten Daten

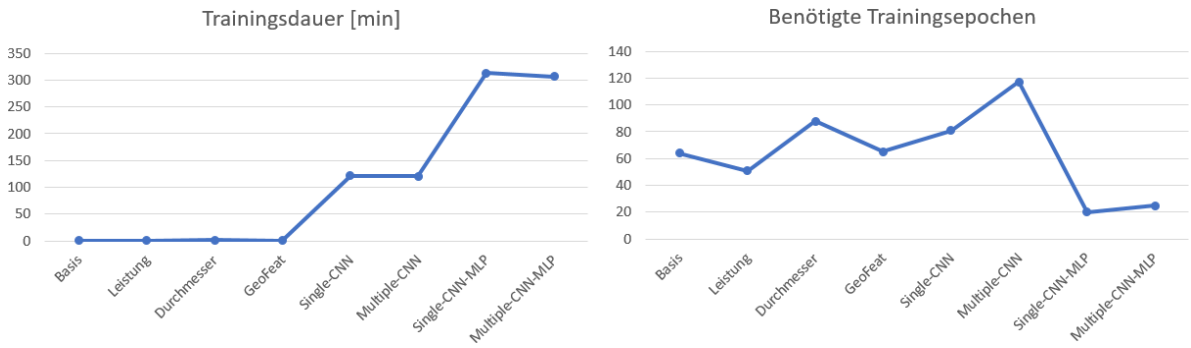



Abbildung 5.1: Trainingsdauer und Trainingsepochen je Modell

## 5.2 Auswertung mit Trainingsdaten und Validierungsdaten

Die Verläufe der Training- und Validierungsfehler werden nachfolgend für jedes Modell betrachtet. Dazu werden auch die finalen Werte von MSE und MAE angegeben.

### 5.2.1 Das Multilayer Perceptron

In Abbildung 5.2 ist der Verlauf von MSE und MAE der Trainings- und Validierungsdaten für das MLP-Basis Modell mit den verwendeten Eingabeparametern Stromstärke, Spannung, Belichtungsdauer, Filtermaterial und Maß veranschaulicht. Es kann festgestellt werden, dass Trainingsfehler und Validierungsfehler den gleichen Verlauf haben, wobei der Trainingsfehler am Ende des Trainings leicht geringer ist, als der Validierungsfehler. Es lässt sich somit erkennen, dass das MLP die trainierbaren Parameter anpasst, so dass eine kontinuierlich Verringerung des Trainingsfehlers erfolgt. Der Validierungsfehler nimmt ebenfalls mit einem ähnlichen Verlauf wie der Trainingsfehler ab. Nach Beendigung des Trainings liegt der Wert des MSE (MAE) für das MLP-Basis Modell für die Trainingsdaten bei 8.91 $\mu m^2$ (2.27 $\mu m$) und für die Validierungsdaten bei 9.91 $\mu m^2$ (2.39 $\mu m$). Da zwischen den Fehlern von Trai-

ningsdaten und Validierungsdaten nur ein sehr geringer Unterschied besteht, ist nicht davon auszugehen, dass bei dem erstellten Modell ein Overfitting stattfindet.

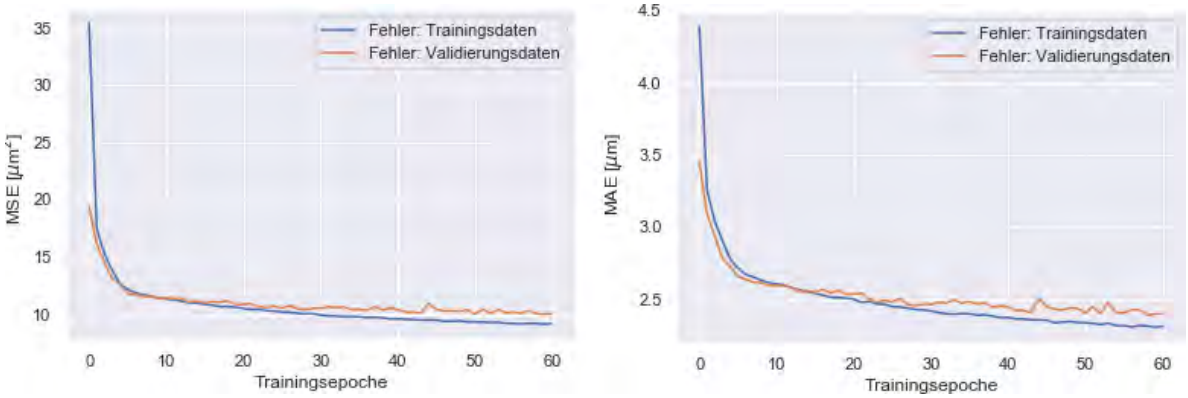



Abbildung 5.2: Verlauf des MSE und MAE der Trainings- bzw. Validierungsdaten für das MLP-Basis Modell

Betrachtet man im Vergleich dazu den Trainings- und Validierungsfehler des MLP-Modell mit dem zusätzlichen Parameter Leistung in Abbildung 5.3, lässt sich feststellen, dass durch Hinzufügen des Eingabeparameters Leistung keine Verbesserung des MLP erreicht wird. Die Werte von MSE und MAE haben sich nur minimal verringert. Der MSE und MAE betragen 9.01 $\mu m^2$ und 2.28 $\mu m$ für den Trainingsfehler und 10.05 $\mu m^2$ und 2.40 $\mu m$ für den Validierungsfehler. Der zusätzliche Parameter Leistung bietet also kaum Mehrwert für das neuronale Netz. Das bestätigt die Annahme, dass ein redundanter Parameter, der sich aus anderen Eingabewerten berechnen lässt, keine zusätzliche Information für das Netz bereitstellt. Da aber der Trainingsaufwand durch diesen Parameter auch nicht erhöht wird, wird er weiterhin in den anderen Modellen als Eingabeparameter benutzt.

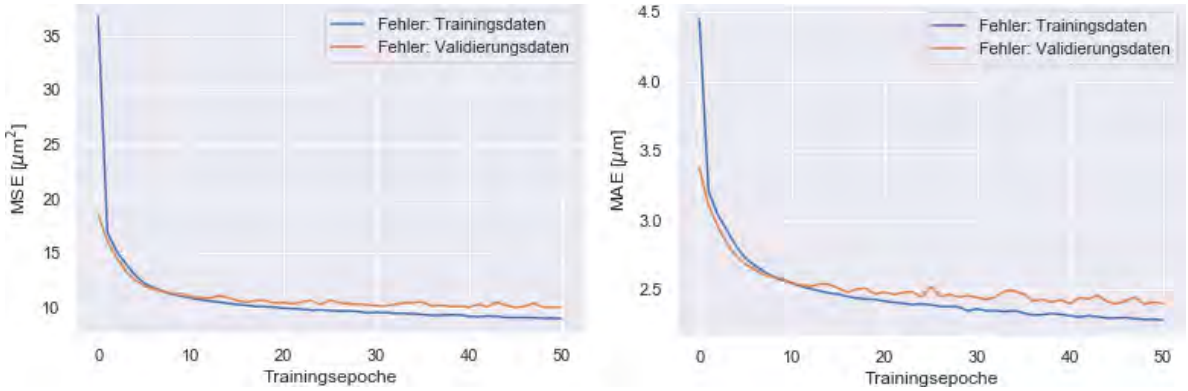



Abbildung 5.3: Verlauf des MSE und MAE der Trainings- bzw. Validierungsdaten für das MLP-Leistung Modell

In der Abbildung 5.4 sind der Trainingsfehler und der Validierungsfehler für das MLP Modell unter Berücksichtigung des Objektdurchmessers dargestellt. Der MSE und MAE der Trainingsdaten betragen 1.62 $\mu m^2$ bzw. 0.93 $\mu m$. Die Validierungsfehler erreichen Werte von 2.36

$\mu m^2$ und 1.13 $\mu m$. Es lässt sich erkennen, dass der Parameter die Genauigkeit der Vorhersage des Modells deutlich erhöht. Daraus lässt sich schließen, dass es für das neuronale Netz wichtig ist, Informationen über das Objekt zu haben, zu dem die Maschinenparameter ermittelt werden.

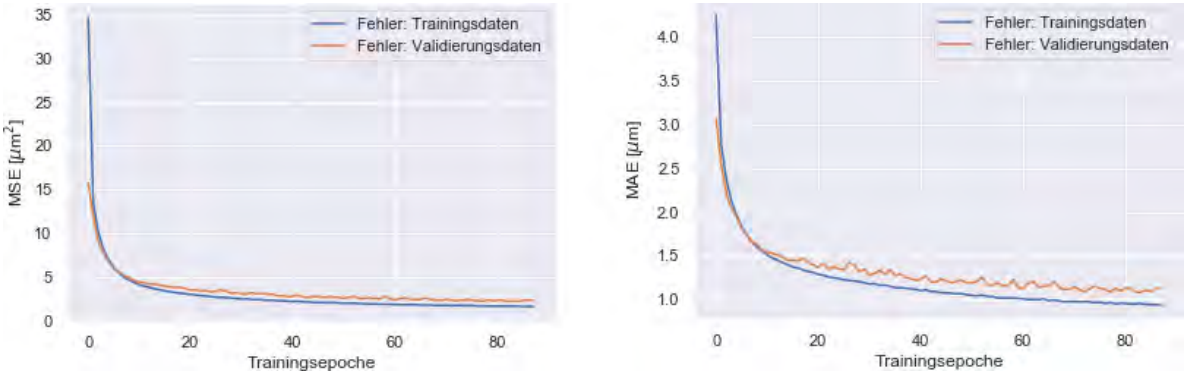



Abbildung 5.4: Verlauf des MSE und MAE der Trainings- bzw. Validierungsdaten für das MLP-Durchmesser Modell

Das MLP-GeoFeat Modell benutzt statt dem Objektdurchmesser die berechneten geometrischen Kennzahlen als zusätzliche Eingabeparameter für das neuronale Netz. Betrachtet man die Fehlerverläufe in Abbildung 5.5, so kann festgestellt werden, dass die berechneten Kennzahlen den Objektdurchmesser nicht nur ersetzen können, sondern das gesamte Modell signifikant verbessern. Der Trainingsfehler liegt bei 0.85 $\mu m^2$ und 0.67 $\mu m$. Der Validierungsfehler erreicht die Werte 1.15 $\mu m^2$ und 0.77 $\mu m$.

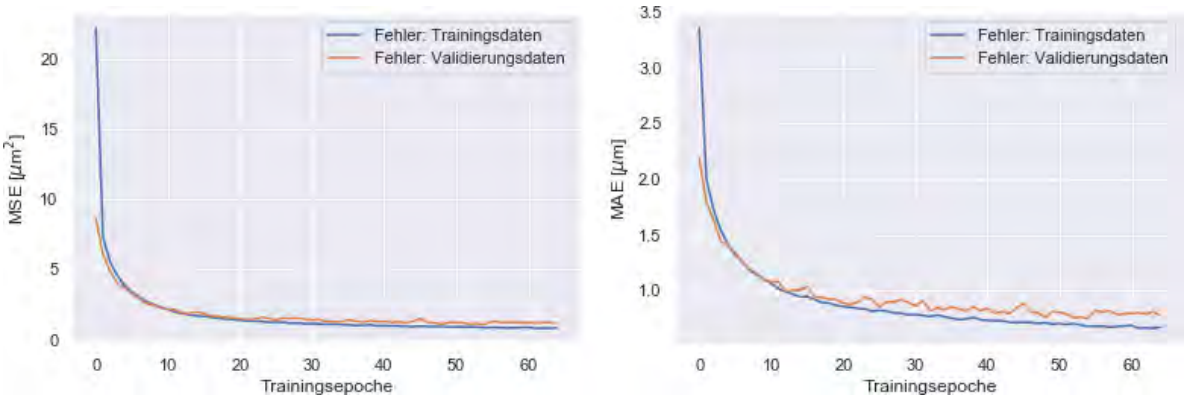



Abbildung 5.5: Verlauf des MSE und MAE der Trainings- bzw. Validierungsdaten für das MLP-GeoFeat Modell

### 5.2.2 Die CNN Modelle

In Abbildung 5.6 sind die MSE und MAE Fehlerverläufe der beiden CNN Modelle dargestellt. Es lässt sich erkennen, dass die Fehlerkurven deutlich mehr Schwankungen aufweisen,

als die der MLP Modelle. Das lässt sich auf den deutlich geringeren Umfang an Daten, die für das Training zur Verfügung stehen, zurückführen. Es werden aber trotzdem Werte erreicht, die zwischen dem MLP-Leistung Modell und dem MLP-Durchmesser Modell liegen. Für das Single-Input CNN Modell beträgt der MSE (MAE) 1.42 $\mu m^2$ (0.86 $\mu m$) für die Trainingsdaten und 2.51 $\mu m^2$ (1.15 $\mu m$) für die Validierungsdaten. Das Multiple-Input CNN Modell erreicht für die Trainingsdaten einen MSE von 1.94 $\mu m^2$ und einen MAE von 1.01 $\mu m$. Für die Validierungsdaten kann ein MSE von 2.98 $\mu m^2$ und ein MAE von 1.32 $\mu m$ erreicht werden. Die Differenz von Trainings- und Validierungsfehler ist höher als in den MLP Modellen. Das ist ein Anzeichen, dass die Modelle zu Overfitting neigen. Das kann aber auch auf die geringe Menge an Trainingsdatensätze zurückgeführt werden.

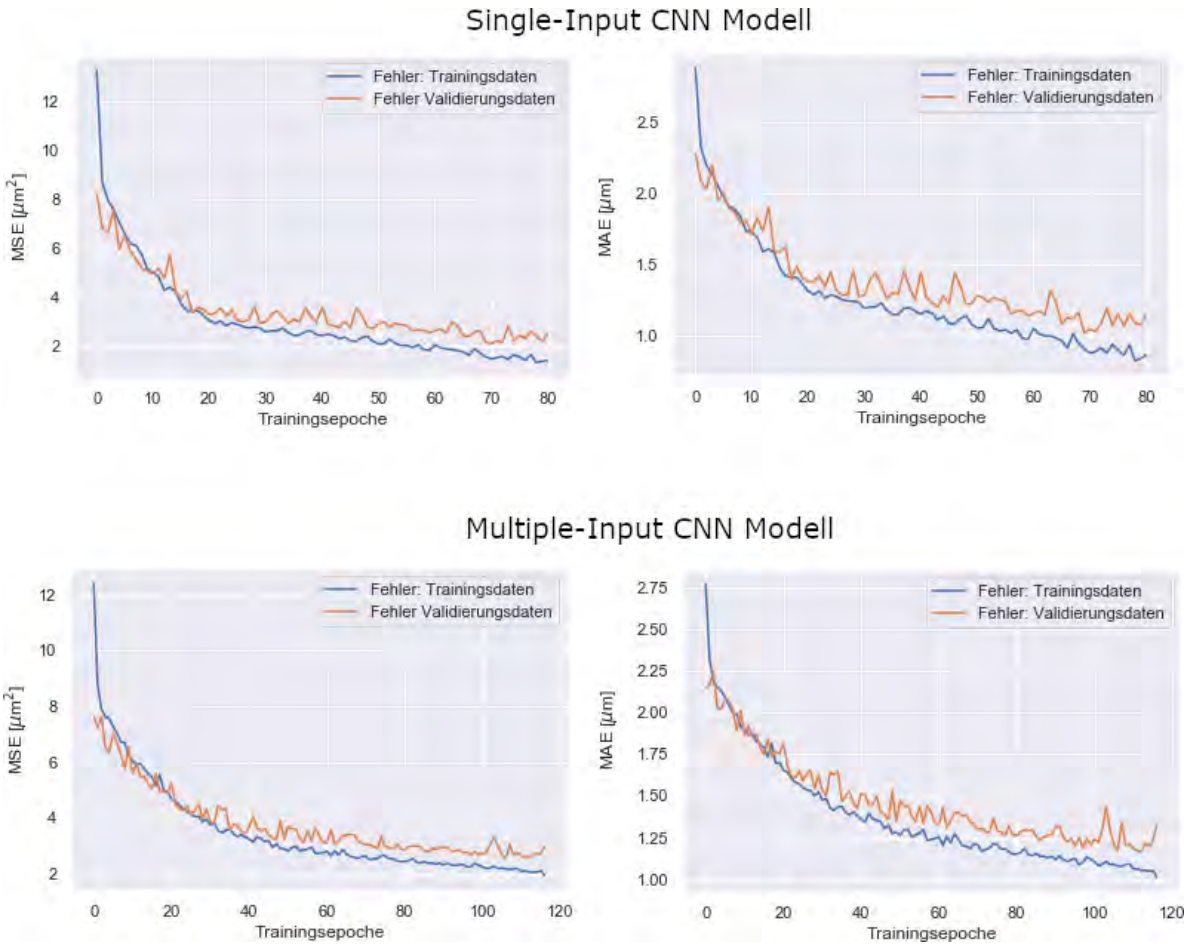



Abbildung 5.6: Verlauf des MSE und MAE der Trainings- bzw. Validierungsdaten für die CNN Modelle

### 5.2.3 Die CNN-MLP Modelle

Die CNN-MLP Modelle erzielen die besten Werte für Trainingsfehler und Validierungsfehler, wie sich in Abbildung 5.7 erkennen lässt und das obwohl sie für deutlich weniger Epochen

trainiert werden als die anderen Netze. Das liegt an der großen Menge an zur Verfügung stehenden Trainingsdaten. Das Single-Input-CNN-MLP Modell hat einen MSE von 0.38 $\mu m^2$ und einen MAE von 0.45 $\mu m$ bei den Trainingsdaten sowie einen MSE von 1.07 $\mu m^2$ und einen MAE von 0.70 $\mu m$ bei den Validierungsdaten. Das Multiple-Input-CNN-Modell erreicht mit einem MSE (MAE) von 0.40 $\mu m^2$ (0.46 $\mu m$) der Trainingsdaten und einem MSE (MAE) von 1.1 $\mu m^2$ (0.70 $\mu m$) ähnliche Werte. Die beiden CNN-MLP Modelle sind von den Fehlerraten vergleichbar mit dem MLP-GeoFeat Modell.

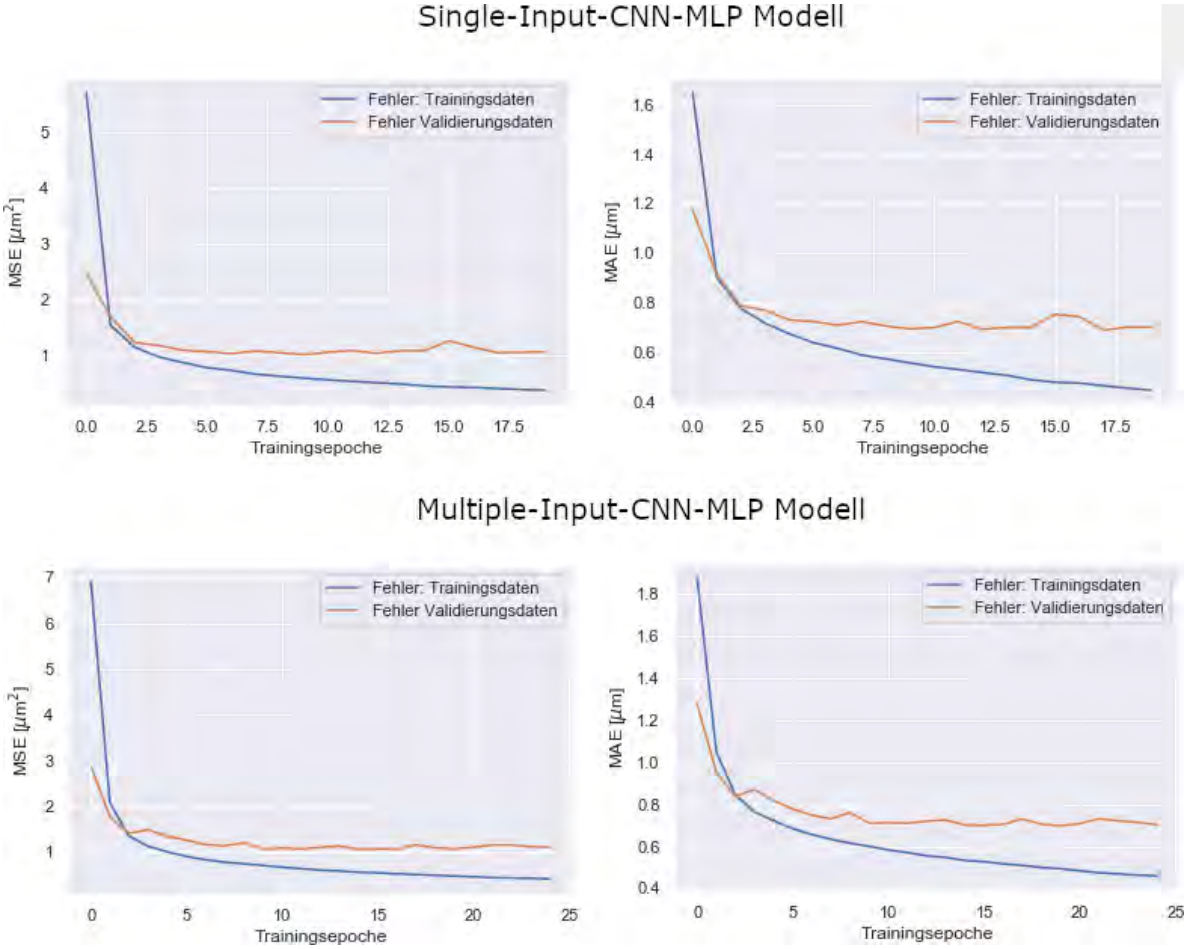



Abbildung 5.7: Verlauf des MSE und MAE der Trainings- bzw. Validierungsdaten für die CNN-MLP Modelle

## 5.3 Auswertung mit Testdaten

Um zu überprüfen, wie die entwickelten Modelle auf unbekannte Daten reagieren, werden die Validierungsmetriken MSE, MAE und R2-Score mit den Testdaten ermittelt. Mit Hilfe der Testdaten und den genannten Metriken kann festgestellt werden, wie genau die Modelle tatsächlich sind. Dazu sind in Abbildung 5.8 die Werte des MSE und MAE der Testda-

ten für jedes Modell dargestellt. Die besten Ergebnisse liefern die Modelle MLP-GeoFeat, Single-Input-CNN-MLP und Multiple-Input-CNN-MLP mit MSE (MAE) Werten von 1.13 $\mu m^2$ (0.77 $\mu m$), 1.04 $\mu m^2$ (0.69 $\mu m$) und 1.00 $\mu m^2$ (0.68 $\mu m$). Die Fehlerraten der Modelle MLP-Durchmesser, Single-Input-CNN und Multiple-Input-CNN liegen im selben Bereich. Ihre MSE Fehlerraten betragen 2.31 $\mu m^2$, 2.47 $\mu m^2$ und 2.88 $\mu m^2$. Die MAE Fehlerraten liegen bei 1.13 $\mu m$, 1.16 $\mu m$ und 1.32 $\mu m$. Die beiden Modelle MLP-Basis und MLP-Leistung weisen mit einem MSE (MAE) von 9.79 $\mu m^2$ (2.39 $\mu m$) und 9.66 $\mu m^2$ (2.39 $\mu m$) deutlich höhere Werte aus.

In Abbildung 5.9 ist der Zusammenhang zwischen der tatsächlichen Messabweichung und der vorhergesagten Messabweichung des jeweiligen Modells, sowie die zugehörige R2-Score dargestellt. Die R2-Score gibt an, wie viel Streuung der Daten mit Hilfe eines linearen Regressionsmodells erklärt werden kann. Die drei Modelle MLP-GeoFeat, Single-Input-CNN-MLP und Multiple-Input-CNN-MLP erreichen mit den Testdaten eine R2-Score von 97%. Das MLP-Durchmesser Modell hat eine R2-Score von 95%. Die R2-Scores der restlichen Modelle liegen zwischen 80% und 86%.

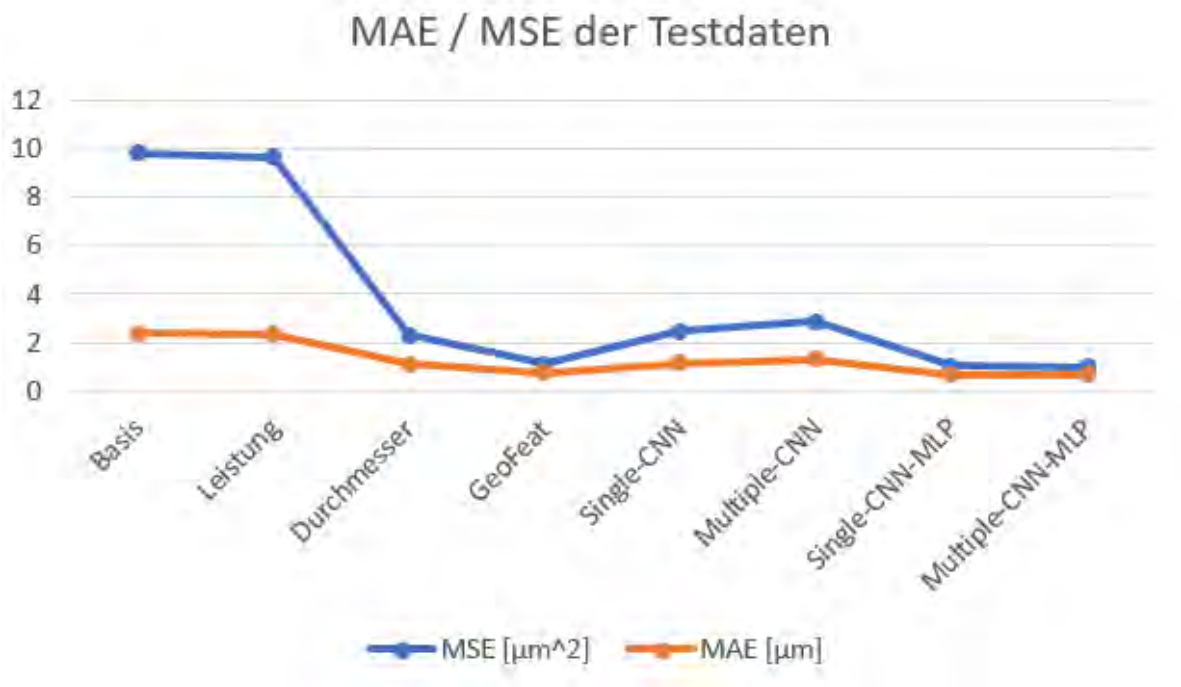



Abbildung 5.8: Verlauf des MSE und MAE der Testdaten für alle Modelle

## 5.4 Vorhersage der Messabweichung an einem Beispiel

Um die entwickelten Modelle für die Praxis testen zu können, wurde ein Programm geschrieben. Dieses Programm benötigt als Eingabe drei Projektionsdateien, das Modell im Dateifor-

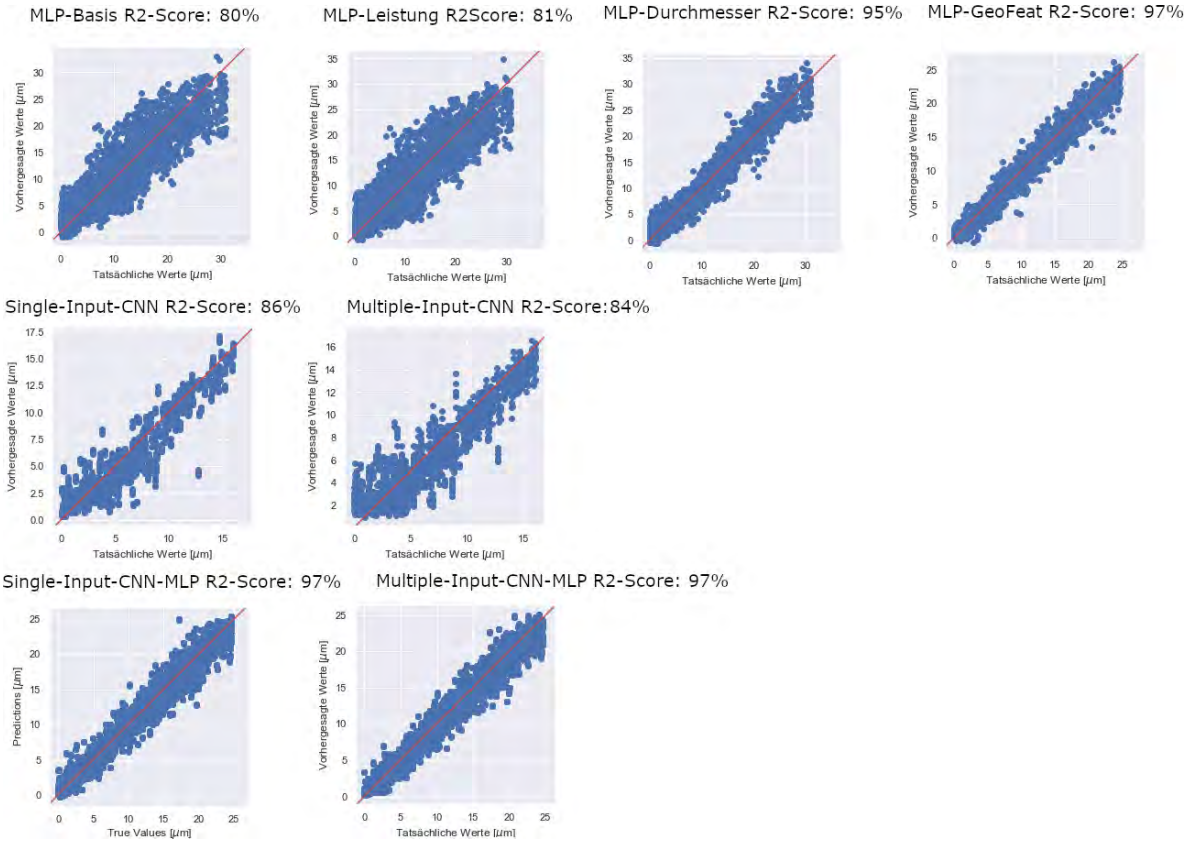



Abbildung 5.9: Vergleich von tatsächlichen und vorhergesagten Messabweichung

mat H5 und eine Preprocessing Pipeline im PKL Format. Der Benutzer kann mit Flags steuern, welches Modell zur Vorhersage verwendet werden soll. Außerdem kann er die Intervalle für Stromstärke, Spannung und Belichtungsdauer einstellen und die Filtermaterialien auswählen. Es soll immer nur ein Maß für die Vorhersage gewählt werden. Das Programm berechnet mit Hilfe des zu verwendenden Modells die Messabweichung zu jeder Parameterkombination der eingestellten Eingabedaten. Aus allen Messabweichungen wird der Minimalwert bestimmt und die zugehörigen Eingabeparameter ermittelt. Mit dieser Parameterkombination soll die beste 3D-Rekonstruktion erstellt werden können.

In einem abschließenden Test wird das Programm jeweils für die Modelle MLP-GeoFeat, Single-Input-CNN-MLP und Multiple-Input-CNN-MLP ausgeführt. Als Eingabeparameter werden jedes Mal Stromstärken im Intervall [50; 250] mit einer Schrittweite von 25, Spannungen im Intervall [60; 240] mit einer Schrittweite von 20, Belichtungsdauer im Intervall [250; 2000] mit einer Schrittweite von 250 und die Filtermaterialien '-', '0.5mm Sn', '1.0mm Al', '0.5mm Cu', '3.0mm Al' und '1.0mm Cu' verwendet. Außerdem wird das Maß D_Z10 gewählt. Das führt zu 3024 Parameterkombinationen. In der Tabelle 5.2 sind die vorgeschlagenen Maschinenparameter der jeweiligen Modelle aufgeführt. Als Projektionsdateien werden die 3 erstellten Projektionen der Messung mit der Messnummer 22241 verwendet. Diese Messung hat für das Maß D_Z10 und die Eingabeparameter Stromstärke 225, Spannung 175, Belich-

tungsdauer 666 und Filtermaterial 0.5mm Cu eine optimale Parameterkombination (Abweichung = 0).

| | **MLP-GeoFeat** | **Single-Input-CNN-MLP** | **Multiple-Input-CNN-MLP** |
|---|---|---|---|
| **Stromstärke** | 225 | 225 | 50 |
| **Spannung** | 60 | 60 | 60 |
| **Belichtungsdauer** | 250 | 250 | 250 |
| **Filtermaterial** | 1.0mm Cu | 1.0mm Cu | - |
| **Abweichung** | 0.32 $\mu m$ | 0.37 $\mu m$ | 5.69 $\mu m$ |

Tabelle 5.2: Vorhergesagte Parameter der Modelle

Wie man sehen kann, geben die beiden Modelle MLP-GeoFeat und Single-Input-CNN-MLP die selbe Parameterkombination mit fast identischer minimaler Abweichung aus. Das Multiple-Input-CNN-MLP weicht hier deutlich stärker ab. Wie gut die vorgeschlagenen Parameterkombinationen tatsächlich sind, müsste in der Praxis an einem Computertomographen getestet werden.

# 6 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Multilayer Perceptron Modell und mehrere Convolutional Neural Networks Modelle entwickelt, welche dazu verwendet werden die Abweichung für Stahlzylinder mit Durchmessern zwischen 1 mm und 10 mm bei dreidimensionalen computertomographischen Rekonstruktionen vorherzusagen.

In einem ersten Schritt wurde ein Multilayer Perceptron entwickelt, das schrittweise mit neuen Parametern erweitert wurde. Dabei konnte festgestellt werden, dass die Eigenschaften eines abgebildeten Objekts von großer Bedeutung für neuronale Netze sind. Sie enthalten wichtige Informationen, die die Genauigkeit der Vorhersage deutlich verbessern konnten. Im Gegensatz dazu stellen redundante Parameter keinen Mehrwert für das Netz dar. Im nächsten Schritt wurde die manuelle Berechnung der geometrischen Eigenschaften eines Objekts durch ein Convolutional Neural Network ersetzt, da diese Netze auf die Feature Extraction von Bildern ausgelegt sind. Es wurde ein serielles CNN und paralleles CNN entwickelt. Anschließend wurden die beiden CNNs mit dem Mulitlayer Perceptron verknüpft.

Die Auswertung der Modelle ergab, dass die Anzahl der Trainingsepochen nicht relevant für die Genauigkeit eines neuronalen Netzes sind. Es ist vielmehr der Umfang an Trainingsdaten, der den Netzen zur Verfügung gestellt wird. Es konnte festgestellt werden, dass drei Modelle sehr gut zur Vorhersage der Messabweichung geeignet sind. Das wurde auch in einem Test bestätigt, bei denen Abweichungen kleiner als 0.5 $\mu m$ vorhergesagt wurden und sich einer optimalen Maschinenparameterkombination genähert wurde.

In einem nächsten Schritt ließen sich die Modelle erweitern, so dass auch andere Objekte als die bisherigen Stahlzylinder verarbeitet werden können. Dazu müssten nur mehr Daten für das Training verwendet werden, was aber dazu führen kann, dass die Hardwareressourcen nicht mehr ausreichen. Die in dieser Arbeit verwendete Hardware ist mit den entwickelten Modellen bis an die Grenzen ausgereizt worden. Es wäre auch möglich die bestehenden Modelle mit Dropout und Batch Normalization Layern zu erweitern, um den Trainingsprozess zu beschleunigen und die Modelle zu optimieren.

Eine weitere Möglichkeit wäre auch das in dieser Arbeit angegangene Regressionsproblem in ein Klassifikationsproblem zu transformieren, um anschließend einen Vergleich zu ziehen, welcher Ansatz das Problem besser löst. Eine andere Möglichkeit wäre, den Fokus auf die Nullabweichungen zu legen und zu versuchen diese mit Anomaly Detection zu identifizieren.

# Literaturverzeichnis


[1] Kroll, A.: *Computational Intelligence*, Gruyter, de Oldenbourg, 2016

[2] He, K.; Zhang, X.; Ren, S.; Sun, J.: *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* In: *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, dec 2015

[3] Glorot, X.; Bordes, A.; Bengio, Y.: *Deep sparse rectifier neural networks.* In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, S. 315–323

[4] Agostinelli, F.; Hoffman, M.; Sadowski, P.; Baldi, P.: *Learning Activation Functions to Improve Deep Neural Networks*,

[5] Strecker, S.: *Künstliche Neuronale Netze – Aufbau und Funktionsweise.* Online, URL: `http://geb.uni-giessen.de/geb/volltexte/2004/1697/pdf/Apap_WI_1997_10.pdf`. – Abrufdatum: 25.09.2019

[6] El-Shahat, A. (Hrsg.): *Advanced Applications for Artificial Neural Networks*, IntechOpen, 2018

[7] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*, MIT Press, 2016. – `http://www.deeplearningbook.org` Abrufdatum: 24.09.2019

[8] Alpaydin, E.: *Maschinelles Lernen (De Gruyter Studium) (German Edition)*, De Gruyter, 2019, URL: `https://www.amazon.com/Maschinelles-Lernen-Gruyter-Studium-German-ebook/dp/B07X6QGQWZ?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B07X6QGQWZ`. – ISBN 978–3–11–061794–8

[9] Günter Daniel Rey, K. F. W.: *Neuronale Netze*, Hogrefe AG, 2018, URL: `https://www.ebook.de/de/product/31309550/guenter_daniel_rey_karl_f_wender_neuronale_netze.html`. – ISBN 3456857969

[10] Le, Q. V.; Brain, G.; Inc, G.: *A Tutorial on Deep Learning Part 1: Nonlinear Classifiers and The Backpropagation Algorithm.* Online, URL: `https://cs.stanford.edu/~quoc1e/tutorial1.pdf`. – Abrufdatum: 24.09.2019

[11] Fischer, P.: *Convolutional Networks to Relate Images*, Albert-Ludwigs-Universität Freiburg, phdthesis, Mai 2016. URL: https://freidok.uni-freiburg.de/fedora/objects/freidok:11418/datastreams/FILE1/content. – Abrufdatum 24.09.2019

[12] Ruder, S.: *An overview of gradient descent optimization algorithms*,

[13] Qian, N.: *On the momentum term in gradient descent learning algorithms*, In: *Neural Networks 12 (1)*, S. 145–151, 1999. http://dx.doi.org/10.1016/s0893-6080(98)00116-6. – DOI 10.1016/s0893–6080(98)00116–6

[14] Duchi, J.; Hazan, E.; Singer, Y.: *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*, In: *J. Mach. Learn. Res. 12*, 2121–2159, 2011. URL: http://dl.acm.org/citation.cfm?id=1953048.2021068. – ISSN 1532–4435

[15] Zeiler, M. D.: *ADADELTA: An Adaptive Learning Rate Method*,

[16] Kingma, D. P.; Ba, J.: *Adam: A Method for Stochastic Optimization*,

[17] Arel, I.; Rose, D. C.; Karnowski, T. P.: *Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]*, In: *IEEE Computational Intelligence Magazine 5 (4)*, S. 13–18, 2010. http://dx.doi.org/10.1109/mci.2010.938364. – DOI 10.1109/mci.2010.938364

[18] Fei-Fei Li, S. Y. Justin Johnson J. Justin Johnson: *CS231n: Convolutional Neural Networks for Visual Recognition - Spring 2019*. Online, URL: http://cs231n.github.io/convolutional-networks/. – Abrufdatum: 25.09.2019

[19] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: *ImageNet Classification with Deep Convolutional Neural Networks*, URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf#note#, In: Pereira, F. (Hrsg.); Burges, C. J. C. (Hrsg.); Bottou, L. (Hrsg.); Weinberger, K. Q. (Hrsg.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., 2012. – Abrufdatum: 26.09.2019

[20] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A.: *Going Deeper with Convolutions*. In: *Computer Vision and Pattern Recognition (CVPR)*, 2015. – Abrufdatum: 26.09.2019

[21] Lin, M.; Chen, Q.; Yan, S.: *Network In Network*, . – Abrufdatum: 26.09.2019

[22] Simonyan, K.; Zisserman, A.: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, . – Abrufdatum: 26.09.2019

[23] Acharya, D.; Yan, W.; Khoshelham, K.: *Real-time image-based parking occupancy detection using deep learning*, 2018

[24] He, K.; Zhang, X.; Ren, S.; Sun, J.: *Deep Residual Learning for Image Recognition*, . – Abrufdatum: 26.09.2019

[25] Otsu, N.: *A Threshold Selection Method from Gray-Level Histograms*, In: *IEEE Transactions on Systems, Man, and Cybernetics 9 (1)*, S. 62–66, 1979. http://dx.doi.org/10.1109/tsmc.1979.4310076. – DOI 10.1109/tsmc.1979.4310076

[26] Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X.: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, URL: https://www.tensorflow.org/. – Software available from tensorflow.org, Abrufdatum 26.09.2019

[27] Chollet, F.; others: *Keras*. https://keras.io, 2015. – Abrufdatum: 26.09.2019

[28] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: *Scikit-learn: Machine Learning in Python*, In: *Journal of Machine Learning Research 12*, S. 2825–2830, 2011

[29] Bradski, G.: *The OpenCV Library*, In: *Dr. Dobb's Journal of Software Tools* , 2000

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Anhang

## A.1 Übersicht der entwickelten Programme und Modelle

In der nachfolgenden Tabelle sind alle entwickelten Programme und Modelle, sowie sonstige in dieser Arbeit erstellten Dateien dargestellt.

| **Name** | **Beschreibung** |
|---|---|
| Multilayer_Perceptron.ipynb | Erstellung des MLP-Basis Modell |
| Multilayer_Perceptron_Leistung.ipynb | Erstellung des MLP-Leistung Modell |
| Multilayer_Perceptron_Durchmesser.ipynb | Erstellung des MLP-Durchmesser Modell |
| Multilayer_Perceptron_GeoFeat.ipynb | Erstellung des MLP-GeoFeat Modell |
| CNN_Concat_Images_Model.ipynb | Erstellung des Single-Input-CNN Modell |
| CNN_Multiple_Images_Model.ipynb | Erstellung des Multiple-Input-CNN Modell |
| CNN_ML_Concat_Images_Model.ipynb | Erstellung des Single-Input-CNN-MLP Modell |
| CNN_MLP_Multiple_Model.ipynb | Erstellung des Multiple-Input-CNN-MLP Modell |
| Predict_Machine_Parameter.ipynb | Testprogramm für erstellte Modelle |
| Convert CSV to PNG.ipynb | Konvertierung von CSV in PNG |
| Datenanalyse.ipynb | Datenanalyse |
| RekToCsv_Multiple | Konvertierung von REK in CSV |
| Pipeline_Data_Preprocessing.pkl | Preprocessing für MLP-Bais Modell |
| Pipeline_Data_Preprocessing_LEISTUNG.pkl | Preprocessing für MLP-Leistung Modell |
| Pipeline_Data_Preprocessing_DURCHMESSER.pkl | Preprocessing für MLP-Durchmesser Modell |
| Pipeline_Data_Preprocessing_GEO_FEAT.pkl | Preprocessing für MLP-GeoFeat Modell |
| MLP_Basic_Model.h5 | MLP-Basis Modell |
| MLP_Leistung_Model.h5 | MLP-Leistung Modell |
| MLP_Durchmesser_Modell.h5 | MLP-Durchmesser Modell |
| MLP_Geo_Feat_Model.h5 | MLP-GeoFeat Modell |
| CNN_SINGLE_MODEL.h5 | Single-Input-CNN Modell |
| CNN_MULTIPLE_MODEL.h5 | Multiple-Input-CNN Modell |
| CNN_ML_Concat_Images_Model.h5 | Single-Input-CNN-MLP Modell |
| CNN_MLP_Multiple_Model.h5 | Multiple-Input-CNN-MLP Modell |
| Messungen_Stahlzylinder | zu verarbeitende Daten |
| Messungen_Stahlzylinder_GeoFeat | zu verarbeitende Daten mit geometrischen Features |

Tabelle A.1: Erstellte Programme und Dateien