Abschlussbericht für das Abschlussprojekt Deep Learning

Tobias Giesler, Florian Graf, Tim Kleinoth

TH Köln

Cologne, Germany

tobias\_ralf.giesler@smail.th-koeln.de

florian.graf@smail.th-koeln.de

tim.kleinoth@smail.th-koeln.de

*Zusammenfassung*— Dieser Bericht beschreibt das Erstellen einer Deep Learning Architektur, welche die Handzeichen „Stein, Schere und Papier“ erkennen soll. Verwendet werden dabei die Datensätze von „Julien de la Bruère-Terreault“ [1], sowie ein eigens erstellter Datensatz.

Schlagwörter—Deep Learning, Bilderkennung, Künstliche Intelligenz, Klassifizierung

# Einleitung

Im Rahmen dieses Projektberichts wird die Erstellung eines Deep Learning Algorithmus zum Erkennen der Handzeichen „Stein, Schere, Papier, sowie Rest“ beschrieben. Deep Learning ist ein Teilgebiet des maschinellen Lernens, welches im Bereich der Bilderkennung zu sehr genauen Ergebnissen führen kann.

Die Projektarbeit wurde in mehrere Schritte gegliedert. Zunächst wurden andere neuronale Netze im Bereich Klassifizierung betrachtet und aus diesen Netzen wurde eine Architektur für dieses Projekt abgeleitet. Daraufhin wurden unterschiedliche Parameter für die Modellarchitektur festgelegt. Im nächsten Schritt wurde die Leistung des Netzes mit einer Teilmenge des Datensatzes getestet, um zu überprüfen, ob eine korrekte Klassifizierung mit der festgelegten Modellarchitektur erfolgen kann. Nach erfolgreicher Überprüfung der Modellarchitektur wurden Optimierungsalgorithmen festgelegt und mehrere Regularisierungsmethoden implementiert. Zuletzt wurde ein vortrainiertes Netz implementiert.

# Verwandte Arbeiten

Die Architektur des im Projekt erstellten neuronalen Netzes wurde auf Grundlage von einem bereits existierenden Projekt von Kaggle abgeleitet. Dieses neuronale Netz wurde erstellt um die Handzeichen „Stein, Schere, Papier“ zu klassifizieren. Verarbeitet wurde auch der Datensatz von „Julien de la Bruère-Terreault“, wobei eine Genauigkeit von 99% erreicht werden konnte.

**Aufbau von einem bereits existierendem Projekt, wobei eine Genauigkeit von 100% erreicht wurd.** [2]

Bei der Betrachtung des Codes des Projektes von Kaggle konnten wurden viele Eigenschaften und angewendeten Methoden des Netzes erkannt werden. Zunächst erfolgt eine Vorverarbeitung der Bilder des Datensatzes (Data Augmentation). Dann entspricht die Architektur des Netzes dem eines Convolutional Neural Networks, was nur Sinn macht da diese für die Bildverarbeitung am meisten verwendet werden.

Das Convolutional Neural Network besteht aus 32 Filter, mit einer 3\*3 Kernel-Size mit dem Max Pooling betrieben wird. Außerdem wird die Relu-Funktion als Aktivierungsfunktion verwendet. Als Optimizer wird der Adam Optimizer verwendet. Des Weiteren gibt es Callback Funktionen wie Early Stopping und Best Model Checkpoint.

Eine Übernahme von vielen Eigenschaften und Methoden des verwandten Netzes für das eigene neuronale Netz wurde als zielführend betrachtet, da es sich bei dem verwendeten Datensatz um einen Teil des eigenen Datensatzes handelt und die Aufgabe des neuronalen Netzes bis auf die geringfügige Erweiterung der Klassifizierungsmöglichkeit „Rest“ die Gleiche ist.

# Vorstellung Datensatz

Bilder Format 300x200

Um das oben erwähnte Ziel eine Deep Learning Architektur zu entwerfen, die die Handzeichen „Schere, Stein und Papier“ erkennt, wurde ein Datensatz erstellt, der Bilder von den erwähnten Handzeichen enthält. Die Aufnahmen wurden immer so angefertigt das die Handinnenfläche von der Kamera weg zeigt. Zusätzlich wurden zufällige Bilder von Händen erstellt, die weder Stein Schere oder Papier darstellen sollen und mit dem Label „Rest“ gekennzeichnet wurden. Außerdem wurde ein bereits vorhandener Datensatz von „Julien de la Bruère-Terreault“ verwendet (doppelt). Ein Beispiel eines Fotos aus diesem Datensatz ist in Abbildung 2 dargestellt:

Ein Bild, das grün, Spielhaus enthält.

Automatisch generierte Beschreibung

Abbildung 1: Beispielbild aus dem verwendeten Datensatz (Papier)

# Daten vorverarbeitung

Die Bilder des Datensatzes werden nicht direkt verwendet. Stattdessen werden dem Modell augmentierte Bilder zur Verfügung gestellt. Da die Augmentierungen nach dem Zufallsprinzip vorgenommen werden, können sowohl veränderte Bilder als auch detailgetreue Nachbildungen der Originalbilder (z. B. fast ohne Augmentierungen) erzeugt und beim Training verwendet werden.

Die Keras-Bibliothek für Deep Learning bietet die Klasse ImageDataGenerator an, welche zur Daten Augmentierung genutzt werden kann. Mithilfe des Datengenerators konnte ebenfalls der Validierungsdatensatz festgelegt werden <https://keras.io/api/preprocessing/image/>.

Shuffle!

# Model architektur

Nach der Vorverarbeitung der Daten musste eine geeignete Methode zur Klassifizierung der Bilder gefunden werden. Dabei gab es mehrere Aspekte zu berücksichtigen.

Zunächst musste eine grundsätzliche Variante eines künstlichen neuronalen Netzes bestimmt werden. Grundsätzlich ist es möglich, Feedforward-Netze für Bilder zu verwenden, bei denen jeder Pixel ein Merkmal darstellt. Dabei stoßen wir jedoch auf mehrere Probleme. Denn Feedforward-Netze berücksichtigen nicht die räumliche Struktur der Pixel, da diese Netze beispielsweise die Beziehungen zwischen dem ersten und dem zweiten Pixel genauso berücksichtigen wie die Beziehung zwischen dem ersten und dem zehnten Pixel. Feedforward-Netze lernen also globale Beziehungen in den Merkmalen anstelle von lokalen Mustern. Es ist daher ohne weiteres nicht möglich, Objekte innerhalb eines Bildes zu erkennen. Aus diesem Grund wurde ein Convolutional Neural Network (CNN) als geeignete Variante festgelegt. Diese haben die Fähigkeit, die oben beschriebenen räumlichen Strukturen der Pixel zu berücksichtigen und bringen viele weitere wichtige Eigenschaften für den Bereich der Bildverarbeitung mit. Eine Erklärung von CNNs ginge jedoch über den Rahmen dieses Berichtes hinaus.

Um einen weiteren Aspekt zu bestimmen, ist ein Blick auf den Umfang der Eingabe- und Ausgabedaten hilfreich.

Wie bereits beschrieben, dienen die Bilder der Handzeichen mit einer Originalgröße von 300x200 Pixeln als Eingabedaten. Im Prinzip können Bilder beliebiger Größe in ein CNN eingespeist werden. Um jedoch auf Standardmethoden wie die Verwendung eines vortrainierten Netzes zurückgreifen zu können, ist es hilfreich, die Bilder auf eine quadratische Größe zu ändern. Daher wurden diese auf die Größe 224x224 Pixel angepasst. Genauer gesagt kann ein Eingabebild besser als dreidimensionalen Tensor beschrieben werden, da sie drei Dimensionen enthalten: Breite × Höhe × Tiefe. Die Tiefe beschreibt dabei die Farbe eines Pixels, welche in den Farben Rot Grün Blau ausgedrückt wird. Die finale Eingabegröße des CNNs ist daher 224x224x3.

Als Ausgabedaten soll ein Bild einer der Kategorien Stein, Schere, Papier oder Rest zugeordnet werden. Dies ist eine bekannte Aufgabe, die als Multiklassen-Klassifikationsproblem für neuronale Netze bezeichnet wird. Abgeleitet von diesem Hintergrund kann bereits die Ausgangsschicht des neuronalen Netzes mit 4 Neuronen bestimmt werden. Des Weiteren dient die sogenannte Softmax-Funktion als Aktivierungsfunktion der Ausgabeschicht. Auf diese Weise kann eine Kombination mehrerer binärer Klassifikatoren verhindert werden. Die Funktion liefert ein Array von vier Werten mit einer Gesamtsumme von 1. Jedes Element des Arrays spiegelt die Wahrscheinlichkeit einer zu bestimmenden Kategorie wider.

softmax(ˆy)i = exp(ˆyi )

Summe j exp(ˆyj )

* Vorlesung 4 S. 152 => als richtige Formel darstellen

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Abbildung 2: Architektur des ersten verwendeten CNNs

Abbildung 2zeigt den Aufbau des ersten verwendeten CNNs. Neben der bereits beschriebenen Größe der Eingabebilder sowie Anzahl der Neuronen der Ausgabeschicht, kann ebenfalls der restliche Aufbau des Netzes betrachtet werden. Das CNN besteht aus drei Faltungsschichten, denen jeweils eine Pooling-Schicht angegliedert ist. Durch diesen typischen hierarchischen Aufbau wird erzwungen, dass immer höherwertige und abstrakte Repräsentationen gelernt

werden können [3, pp. Vorlesung 2, S.64]. Die in den Faltungsschichten verwendeten Filter besitzen jeweils eine Kernelgröße von 3x3, da kleinere Filter weniger Parameter und weniger Berechnungen benötigen. Außerdem liefern diese im Allgemeinen eine bessere Leistung als größere Filter [4, p. 461]. Bei der Anzahl der Filter wurde sich an den bereits vorgestellten verwandten Projekten orientiert. Die Ausgabe einer Faltungsschicht wird als Feature-Map (Merkmalskarte) bezeichnet. Dessen Größe lässt sich wie folgt berechnen:

Dabei beschreibt die Eingangsgröße und F die Filtergröße sowie s den Stride. Wie in Abbildung 2 zu sehen, ergibt sich daher beispielsweise als Ausgabe der ersten Faltungsschicht 16 Feature-Maps jeweils in einer Größe von 222x222x3.

Als letzte Schicht vor der Ausgabeschicht wird eine vollständig verbundene Feed-Forward-Schicht verwendet, um die gesammelten Informationen zusammenzuführen, bevor die Ausgabeschicht eine Klassifizierung vornimmt.

Aus Abbildung 2 kann jedoch nicht jede Information über das Netz abgelesen werden. Denn die Faltungsschichten verwenden jeweils die Rectified Linear Unit-Funktion (ReLU) als Aktivierungsfunktion, da sie eine unübertroffene Rechengeschwindigkeit hat und sich in der Praxis sehr bewährt hat [4, p. 292]. Um den Fehler des Netzes zu bestimmen, wird die kategorische Kreuzentropie aus Tensorflow Keras als Loss-Funktion verwendet. Als Gradientenabstiegs-Optimierungsalgorithmus muss das Verfahren Adam sein Potenzial zeigen. Es ist einer der beliebtesten Algorithmen, da er schneller als die regulären Gradientenabstiegsalgorithmen ist und das Modelltraining beschleunigen kann.

Die Struktur des CNN wurde experimentell um Schichten erweitert und es wurde eine höhere Anzahl von Filtern verwendet. Da auf diese Weise die Rechenlast enorm zunimmt wurde jedoch nach weiteren Möglichkeiten gesucht.

In den vergangenen Jahren wurden etliche Varianten von CNN-Architekturen veröffentlicht, die sich als hoch effizient erwiesen haben [4, pp. 463-482]. Als Maß für den Fortschritt der Architekturen kann die Fehlerquote des bekannten Bilderklassifikationswettbewerbs „ImageNet“ dienen [5]. Unter den Top fünf Modellen der vergangenen Jahre zählten VGG und ResNet. Beide Modelle dienten als Basis, um die Technik des Transfer Learning zu untersuchen. Beim Transfer Learning werden bereits erworbene Kenntnisse zur Lösung eines neuen Problems verwendet. Die vortrainierten Modelle VGG und ResNet konnten daher bereits zwischen 1000 Klassen des Datensatzes ImageNet unterscheiden.

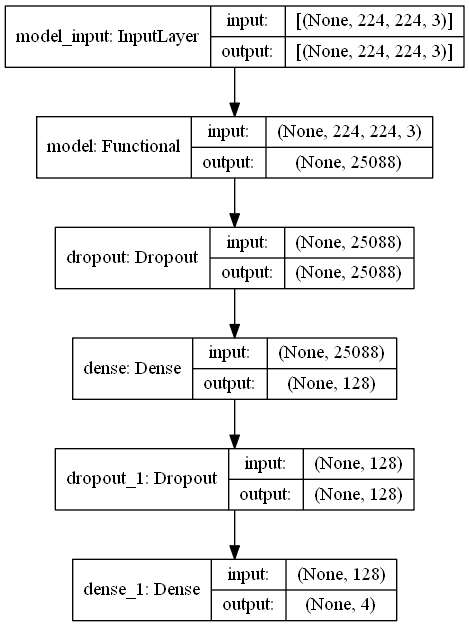


Abbildung 3: Architektur des Modell mit dem Basismodell VGG-16

Abbildung 3 zeigt die Architektur des Modells, welches VGG-16 als Basismodell nutzt. Die Architektur des VGG-Modells wurde im Angang aufgeführt. Die Gewichte des vortrainierten Netzes wurden eingefroren, sodass diese nicht mehr trainiert werden können. Damit das Netz weniger empfindlich auf spezifische Gewichte des vortrainierten Modells ist, wurde als Regularisierungstechnik eine Dropout-Schicht hinzugefügt. Die Regularisierungstechnik wird in dem folgenden Kapitel erläutert.

# Experimente

### Baseline tests => Tests mit wenigen bildern, um zu zeigen das das Modell das Problem lernen kann

### Regularisierungen

…

…

### Ergebisse des CNNs aus Abbildung 2

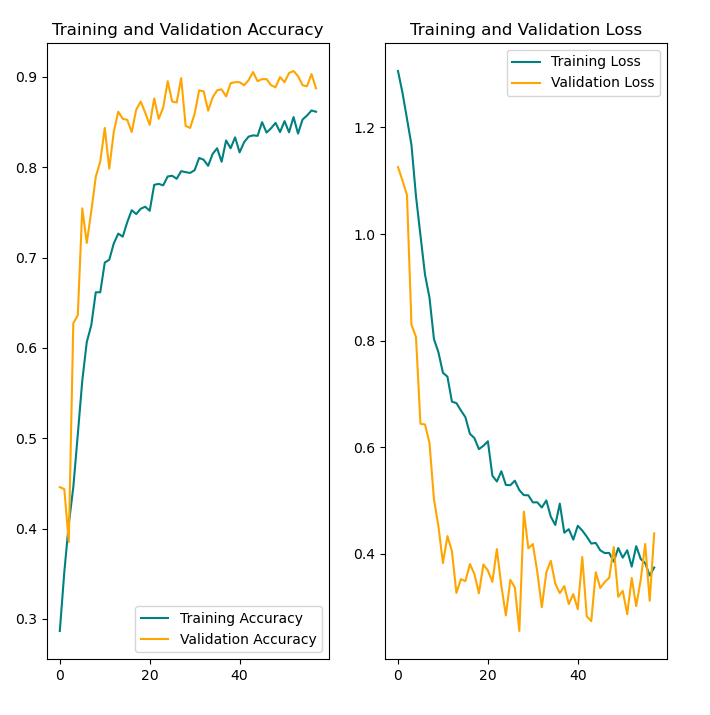
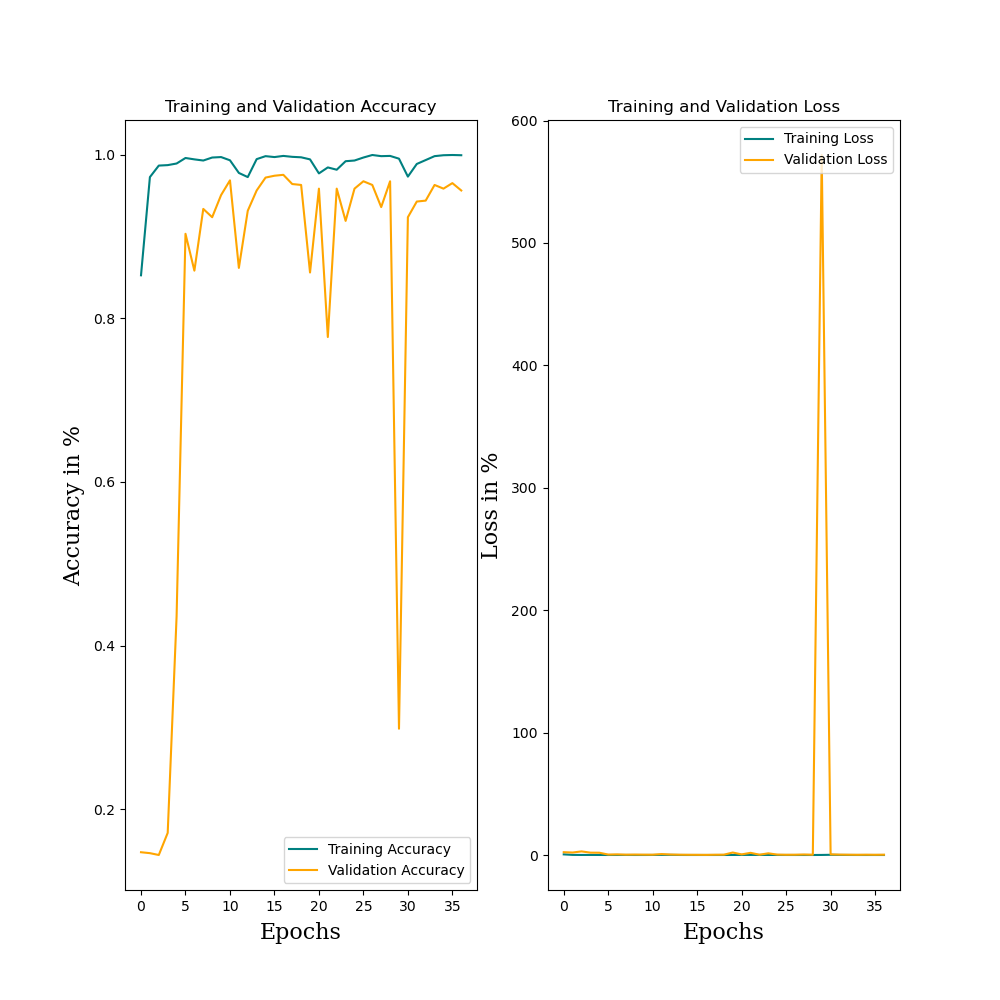


Abbildung 4: Trainingsverlauf des ersten CNN-Modell

* Early Stopping eingesetzt (val\_loss, patient=20)
* Accuracy: 0.91, Loss: 1.13
* Ohne Data Augmentation

### Ergenisse des Modells mit dem Basismodell ResNet



* Acc: 0.98, Loss: 571.85
* Ohne Data Augmentation

### Ergebnisse des Modells mit dem Basismodell VGG-16

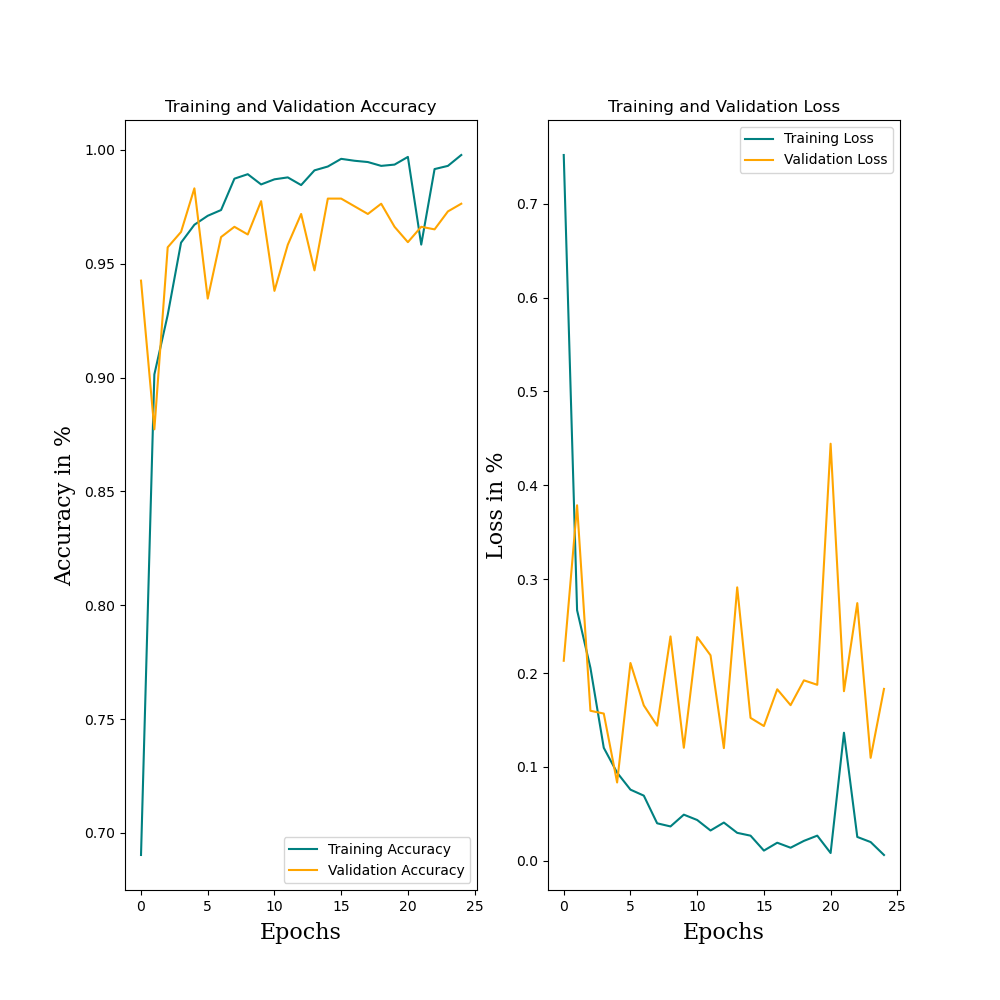


Abbildung 5: Ergebnisse des Modells mit dem Basismodell VGG-16

* Overfitting
* Accuracy: 0.98, Loss: 1.12
* Ohne Data Augmentation
* => Bestes Ergebniss

Betrachtung der falsch klassifizierten Ergebnisse

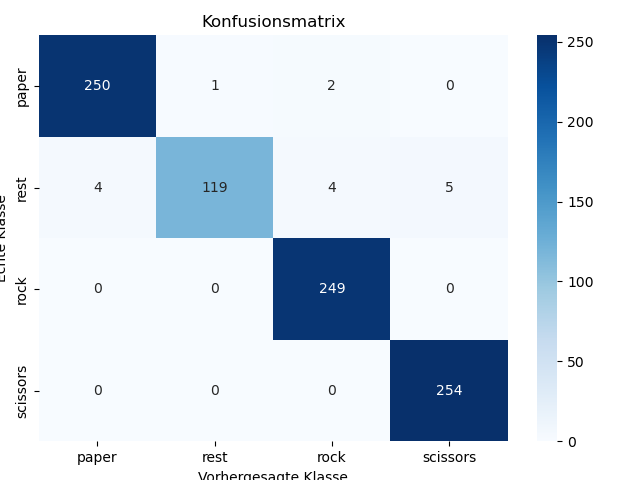


Abbildung 6: Konfusionmatrix der Vorhersageergebnisse des Modells auf Basis des VGG-Netzes

* X-Achse= Vorhergesagte Klasse, Y-Achse=Echte Klasse
* 16 von 888 Bilder wurden falsch klassifiziert
* Als Beispiel als Rock vorhergesagt

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 7: Beispiel eines falsch klassifizierten Bildes

Ein Bild, das Text, drinnen, Gericht enthält.

Automatisch generierte Beschreibung

Abbildung 8: Weiteres Beispiel eines falsch klassifizierten Bildes

Probiert Data Augmentation => versucht Brightness zu variieren => ohne Erfolg Beispielbild wurde immer noch falsch erkannt sowie mehr falsch klassifiziert.

# Aufbau und Optimierung des deep learning algorithmus

In diesem Kapitel wird der Grundaufbau des Algorithmus sowie Regularisierungsmethoden zur Verbesserung der „Generalisierungsfähigkeit“ thematisiert. Die Generalisierungsfähigkeit bezeichnet dabei die Fähigkeit eines neuronalen Netzes das durch den Trainingsdatensatz „Erlernte“ auf einen fremden Datensatz (zum Beispiel dem Validierungsdatensatz) anzuwenden. Eine gute Generalisierungsfähigkeit ist damit ein Indikator für ein gut trainiertes neuronales Netz.

Regularisierungsmethoden helfen dem neuronalen Netz eine höhere Generalisierungsfähigkeit zu erreichen. Regularisierungsmethoden zielen darauf ab „overfitting“, also die Überanpassung an ein neuronales Netz an einen Trainingsdatensatz zu vermeiden.

Des Weiteren wird auf den Trainingsdatensatz eingegangen.

## Ergebnisse der Lernrate einer nicht optimierten KI

Der beschriebene Algorithmus wurde mithilfe des oben beschriebenen Datensatzes erwähnt. Dieser wurde unterteilt in einen Trainingsdatensatz, also ein Datensatz, mit dem das neuronale Netz trainiert wird und einem Validierungsdatensatz, der einen „fremden“ Datensatz darstellen soll, mit dem das trainierte neuronale Netz getestet wird. Anhand der erzielten Ergebnisse der entworfenen Deep Learning Architektur im Trainingsdatensatz sowie im Validierungsdatensatz kann man gut ein gewisses Lernverhalten der entworfenen Architektur begutachten und beurteilen. Die Ergebnisse, die mit dem nicht optimierten Datensatz erzielt wurden, werden in Abbildung 3 dargestellt:

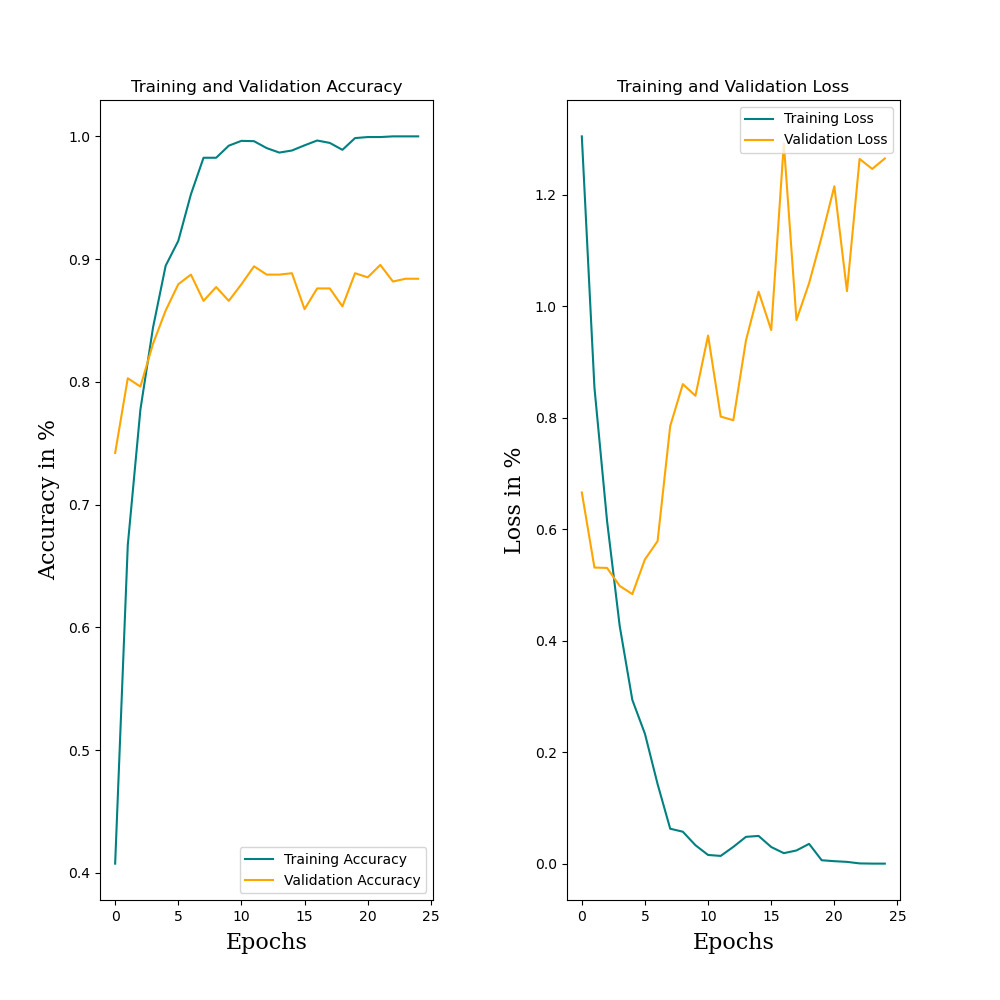


Abbildung 9: Ergebnisse des nicht optimierten neuronalen Netzes mit schlechter Generalisierungsfähigkeit

Betrachtet man die Ergebnisse so erkennt man hier ein Beispiel für das sogenannte „overfitting“. Overfitting bezeichnet die Überanpassung von einem neuronalen Netz an einen Trainingsdatensatz. In Abbildung 3 kann man erkennen das das neuronale Netz in der Genauigkeit der Bestimmung der Handzeichen im Trainingsdatensatz 100% erreicht, während im Validierungsdatensatz weniger Prozent erreicht werden (<90%). Betrachtet man die Loss-Funktion so erkennt man eine starke Abweichung der Werte des Trainingsdatensatzes zu den Werten des Validierungsdatensatzes. Während die Loss-Funktion im Trainingsdatensatz sich gegen 0 bewegt, so steigt die Fehlermenge des neuronalen Netzes im Validierungsdatensatz mit mehreren Epochen sogar an. Das neuronale Netz wird zu sehr an den Trainingsdatensatz angepasst und verliert somit an Generalisierungsfähigkeit. Da die Generalisierungsfähigkeit ein Maß für die Qualität eines neuronalen Netzes darstellt, ist dies ein unerwünschtes Phänomen. Um den Effekt des Overfittings zu verhindern, gibt es unterschiedliche Ansätze, die im Folgenden erläutert werden.

## Dataset Augmentation

„Dataset Augmentation“ beschreibt eine Regularisierungsmethode, die auf eine bessere Generalisierungsfähigkeit abzielt, indem sie mehr Daten aus einem Datensatz erzeugt. Mehr Daten im Datensatz sind grundsätzlich der erfolgversprechendste Weg, um eine bessere Generalisierungsfähigkeit zu erreichen. Im Falle der Bildbearbeitung werden hierbei mehr Daten erzeugt indem gewisse Filter verwendet werden. Die verwendeten Bilder aus dem Datensatz werden nach einem zufälligen Prinzip verschoben, gedreht, bzw. vergrößert oder ein Blur-Effekt wird eingesetzt, die dann leicht veränderten Bilder werden erneut für das Trainieren des neuronalen Netzes verwendet. Somit erzeugt man sich aus dem vorhandenen Datensatz eine wesentlich größere Menge an Daten, wodurch wiederum eine bessere Generalisierungsfähigkeit erzielt werden kann.

Bei der Anwendung von Dataset Augmentation in diesem Projekt wurden die Bilder nach einem Zufallsprinzip gedreht und die Helligkeit wurde verändert. Beim Einfügen der Verschiebung der Bilder, in der Dataset Augmentation Funktion, ist in diesem Projekt ein Fehler aufgetreten, welcher starke Schwankungen in den Lernkurven verursacht hat. Bei Betrachtung der Inhalte der Datensätze ist dann aufgefallen das durch die Verschiebung einige Bilder keine, bzw eine kaum noch zu erkennende Hand enthielten. Dies wurde in der folgenden Abbildung festgehalten:

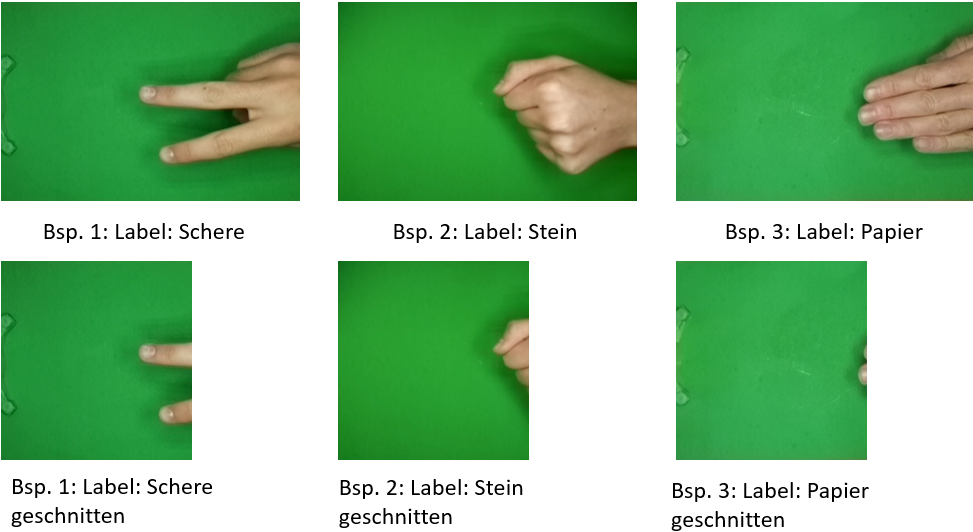


Abbildung 10: Abbildung bearbeitete Bilder. Durch die Verschiebung des Bildes ist eine Hand im Bild kaum noch enthalten

Eventuell noch Bild der Lernrate mit Verschiebung zeigen?

Nach Anwendung von Dataset Augmentation (ohne Verschiebung der Bilder) konnte ein Lernverhalten festgestellt werden welches in Abbildung 5 gezeigt wird. Vergleicht man nun Abbildung 4 mit Abbildung 5 so kann man erkennen, dass der Unterschied der Loss-Funktion vom Trainingsdatensatz zum Validierungsdatensatz geringer geworden ist. Auch die Genauigkeit in der Validierungsfunktion ist in ihrem Maximum ein wenig gewachsen. Darauf lässt sich zurückschließen dass die Generalisierungsfähigkeit des neuronalen Netzes ein wenig verbessert werden konnte.

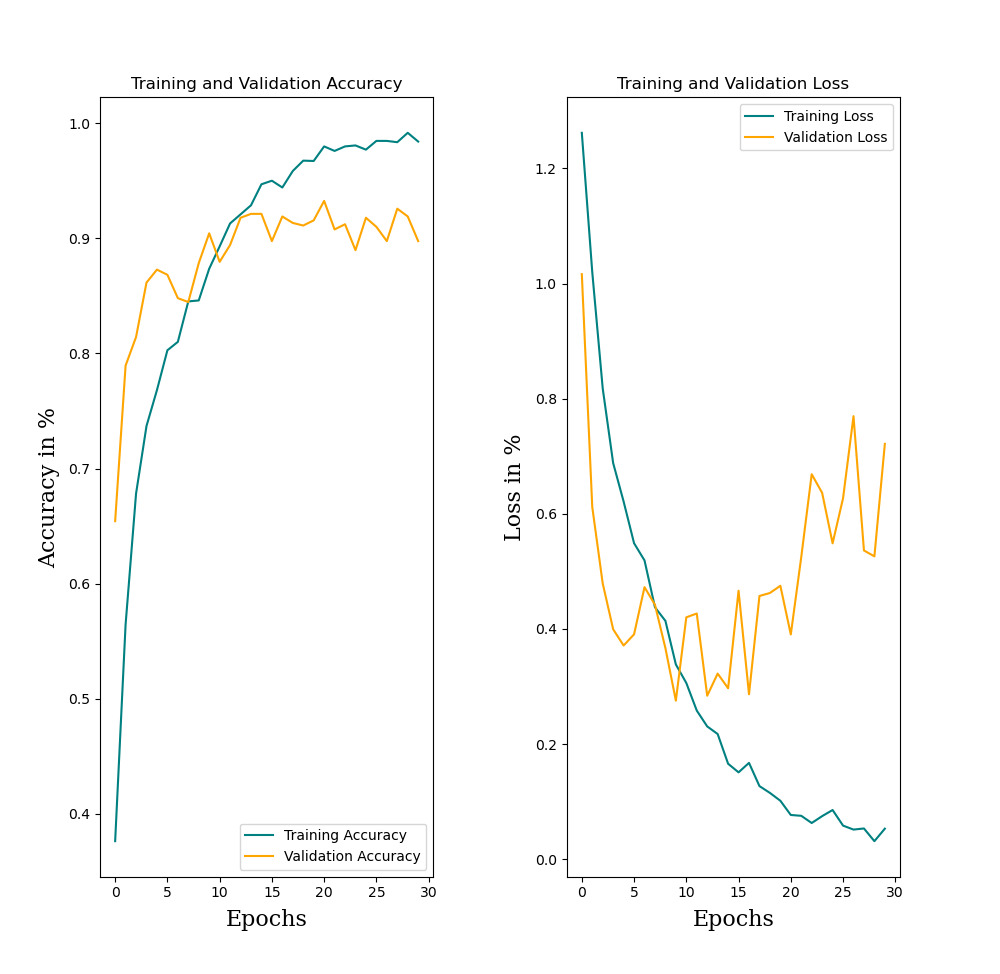


Abbildung 11: Lernverhalten des neuronalen Netzes mit Dataset Augmentation

## Early Stopping

Eine der Verbesserungsmöglichkeiten der Generalisierungsfähigkeit ist das sogenannte „Early Stopping“. Early Stopping wird verwendet, um zu vermeiden, dass sich das entworfene neuronale Netz zu sehr an einen Trainingsdatensatz anpasst und damit übertrainiert wird (overfitting). Ein übertrainiertes Netz erzielt zwar im Trainingsdatensatz, durch die Überanpassung, bessere Ergebnisse, erzielt aber im bisher fremden Validierungsdatensatz schlechtere Ergebnisse.

Early Stopping kann hierbei gut ansetzen. Beim Early Stopping wird die Fehlerfunktion des Validierungsdatensatzes überwacht. TensorFlow erlaubt es, die Early Stopping Methode anzupassen, indem Parameter an den Algorithmus übergeben werden, z.B. der Validierungsverlust zur Beobachtung und die Geduld zur Bestimmung des Beobachtungsintervalls. Außerdem wird ein Checkpoint implementiert, um das Modell zu sichern, das die besten Ergebnisse erzielt hat.

Anwendung des Early Stoppings zeigen + Erklärung

## Parameter Norm Penalties

Um die Generalisierungsfähigkeit der Deep Learning weiter zu verbessern kann man die Regularisierungsmethode „Parameter Norm Penalties“ anwenden. In dieser Methode wird ein Strafterm (Ω) verwendet, welcher, in Abhängigkeit von den Gewichten w (siehe Abbildung 1) eine gewisse Zielfunktion (J) verändert. Genauer wird dies in der folgenden Rechnung dargestellt.

+ α Ω(w) (1)

Das Ziel bei dieser Regularisierungsmethode ist es wieder overfitting zu vermeiden und somit eine bessere Generalisierungsfähigkeit zu erzielen. Thema nochmal ausführlicher beschreiben.

## Dropout

Dropout bezeichnet das zufällige Aussetzen von bestimmten Neuronen im Neuronalen Netz (siehe Abbildung 5). Durch das zufällige Aussetzen von Neuronen wird ein Effekt erzeugt, der umliegende Neuronen zwingt, sich stärker zu vernetzen, um die „Fehler“ von umliegenden Neuronen zu korrigieren. Das Model wird somit insgesamt robuster und erreicht wieder eine höhere Generalisierungsfähigkeit.

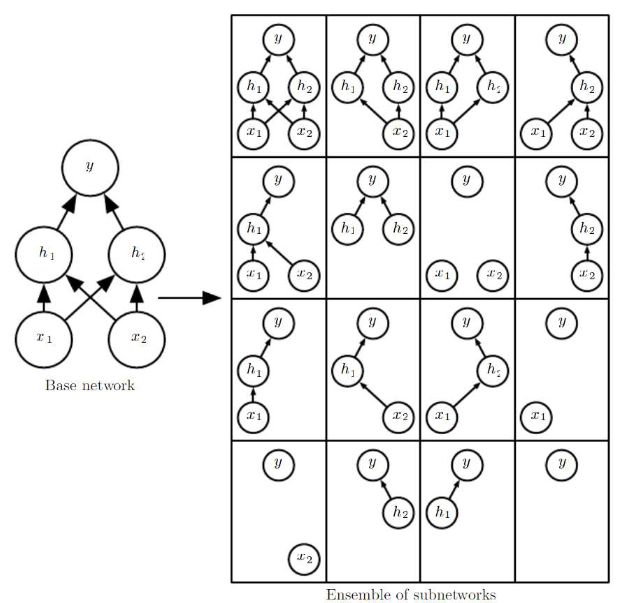


Abbildung 12: Aussetzen von Neuronen (Dropout)

# anhang

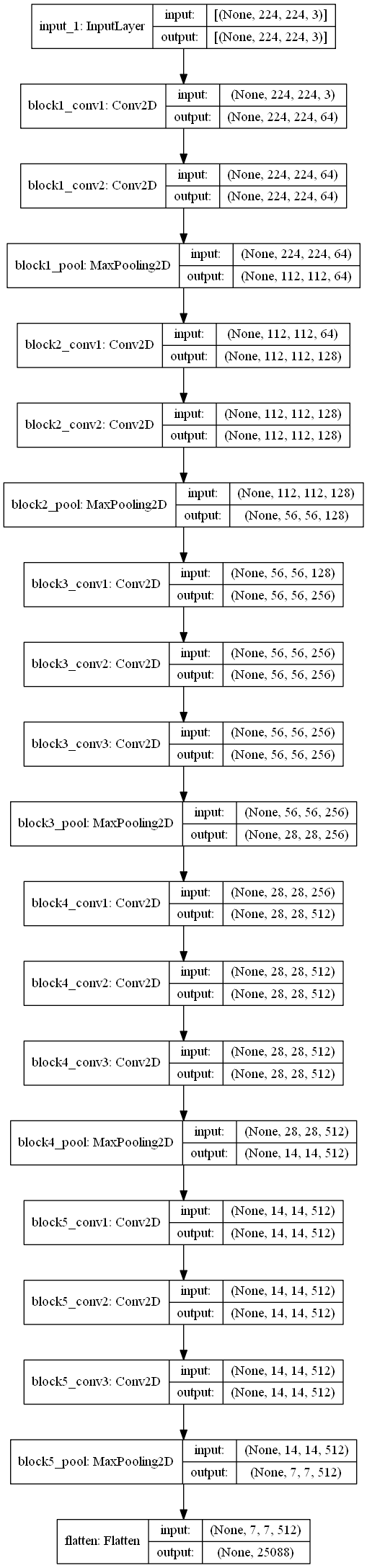


Abbildung 13: Architektur des vortrainierten Netzes VGG

# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | J. d. l. Bruère-Terreault, „Kaggle, Rock-Paper-Scissors Images,“ [Online]. Available: https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors. |
| [2] | A. Naresh, „Kaggle, RockPaperScissors 100% Accuracy,“ [Online]. Available: https://www.kaggle.com/code/recursion17/rockpaperscissors-100-accuracy. |
| [3] | J. Salmen, „Vorlesungsfolien „Deep Learning”,“ [Online]. |
| [4] | A. Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, O’Reilly Media, 2019. |
| [5] | S. V. Lab, „ImageNet,“ 11 03 2021. [Online]. Available: https://www.image-net.org/. |
| [6] | Keras, „Keras API reference,“ [Online]. Available: https://keras.io/api/preprocessing/image/. |

##### Quellenangaben

1. Jan Salmen, Vorlesungsfolien „Deep Learning” S. 31
2. S. XV