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

I. 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.

II. Verwandte Arbeiten

Die Architektur des im Projekt erstellten neuronalen Netzes wurde auf Grundlage von einem bereits existierenden Projekt von Kaggle abgeleitet [2]. Dieses neuronale Netz wurde erstellt um die Handzeichen Stein, Schere und Papier zu klassifizieren. Verwendet wurde in dem Projekt von Kaggle auch der Datensatz von „Julien de la Bruère-Terreault“, wobei eine Genauigkeit von 100% erreicht werden konnte.

Bei der Betrachtung des Codes des bereits existierenden Projektes konnten viele Eigenschaften und angewendeten Methoden des Netzes erkannt werden. Zunächst erfolgt eine Vorverarbeitung der Bilder des Datensatzes (Data Augmentation). Die Struktur des neuronalen Netzes entspricht der eines Convolutional Neural Networks. Diese ist im Bereich der Bildverarbeitung am gängigsten und erzielt die besten Erfolge.

Das Convolutional Neural Network besteht aus 64 Filter, mit einer 5*5 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.

III. Vorstellung Datensatz

Um das für diese Projekt verwendete neuronale Netz, zu trainieren, wurde ein eigener Datensatz erstellt. Dieser enthält jeweils ca. 1250 Bilder der Kategorien „Schere“, „Stein“ und „Papier“ und 660 Bilder welche keiner der drei Kategorien zugeordnet sind (diese Daten wurden mit der Bezeichnung „Rest“ gelabelt). Die Grundlage des Datensatzes bildet dabei ein bereits existierender Datensatz von „Julien de la Bruère-Terreault“ [1]. Dieser enthält Bilder der Klassen „Schere“, „Stein“ und „Papier“ und wird durch weitere Bilder aller vier Klassen ergänzt, welche von Studierenden der TH Köln erstellt

Abbildung 1: Zufällig ausgewählte Beispliele des Datensatzes

worden sind. Abbildung 1 zeigt eine zufällige Auswahl des Datensatzes mit jeweiligen Labeln.

Die Aufnahmen haben ein Seitenverhältnis von 3:2 und wurden so angefertigt, dass die Handinnenfläche von der Kamera weg zeigt und das Handzeichen sich mittig im Bild befindet. Es wurde ebenfalls darauf geachtet, dass der Bildhintergrund möglichst einfarbig ist. Die jeweilige Farbe unterscheidet sich jedoch, wie in Abbildung x zu sehen ist, zwischen den Bildern. Dies erhöht zwar die Komplexität der Klassifizierung im Vergleich zu einem Datensatz, wo jedes Bild eine identische Hintergrundfarbe hat, lässt sich jedoch dafür besser auf einen realen Kontext übertragen.

IV. 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 (Formel 1) 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.

$$\sigma(\hat{y})_i = \frac{e^{\hat{y}_i}}{\sum_{k=1}^{K} e^{\hat{y}_k}} \; f\ddot{u}r \; i = 1, \dots, K \qquad (1)$$

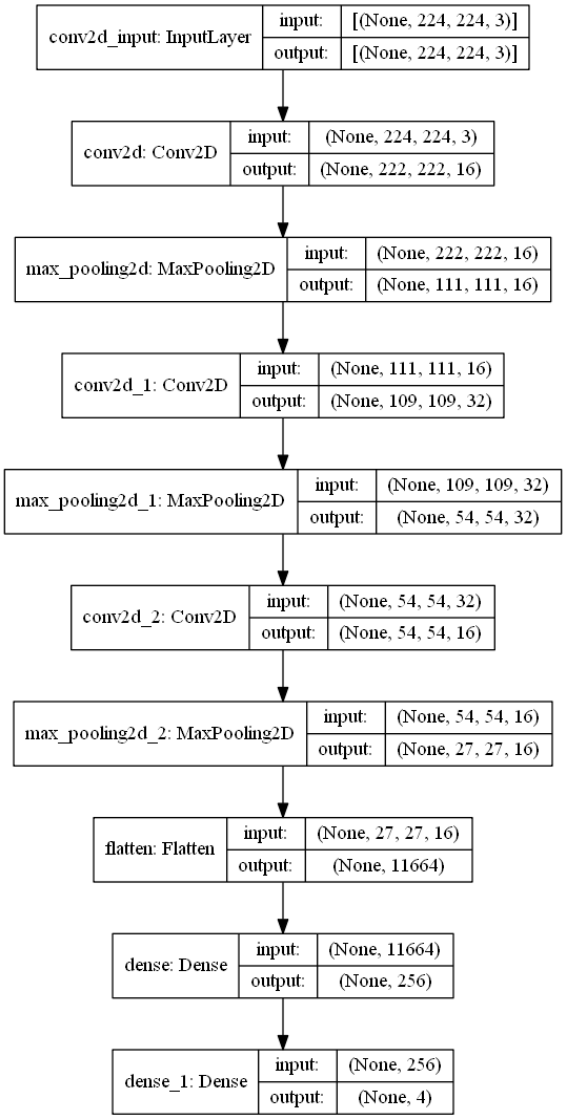

Abbildung 2: Architektur des ersten verwendeten CNNs

Abbildung 2 zeigt 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:

$$N_{out} = \frac{(N_{in} - F)}{s} + 1 \qquad (2)$$

Dabei beschreibt N_{in} 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-Optimierungs-algorithmus 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.

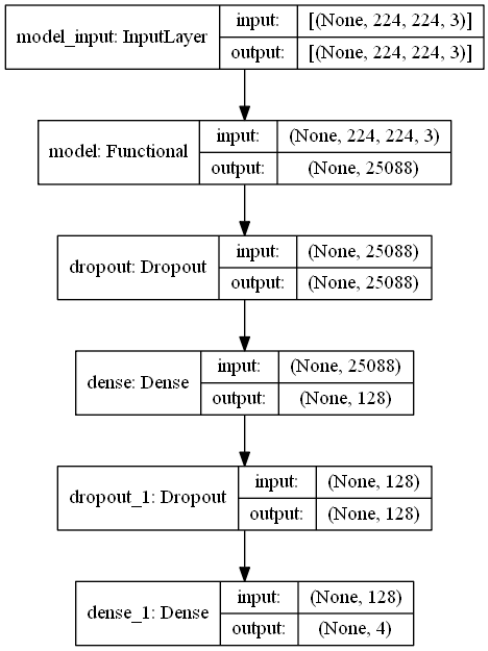

Abbildung 3: Architektur des Modells mit dem Basismodell VGG-16

Abbildung 3 zeigt die Architektur des Modells, welches VGG-16 als Basismodell nutzt. Die Architektur des VGG-Modells wurde im Anhang 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.

V. Experimente

A. *Leistungstest*

Um festzustellen, ob die, im vorherigen Kapitel erläuterte, gewählte Basistopologie des eingesetzten Netzes, überhaupt in der Lage ist, die gegebenen Daten richtig zu klassifizieren, wurde ein sogenannter „sanity check“ durchgeführt.

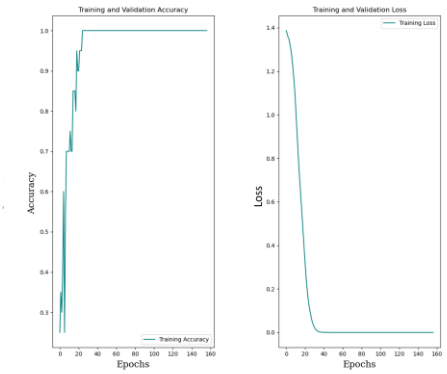

Abbildung 4: Klassifizierungsergebnis mit 20 Trainingsbildern

Hierzu wurden die Trainingsdaten auf jeweils 5 zufällige Instanzen jeder Klasse beschränkt. Mit diesem stark verkleinerten Datensatz wurde das Netz anschließend trainiert. Für das Training wurde die Methode des Early *Stoppings (val_loss, patience=20)* angewendet. Hierdurch muss keine feste Anzahl an Trainingsepochen festgelegt werden, sondern das Modell beendet das Training automatisch, wenn sich der Validierungsfehler über die letzten 20 Epochen nicht verringert hat. Außerdem wurde *Batchnormalization* verwendet, um das Training zu beschleunigen. Das Ergebnis ist in Abbildung 4 zu sehen.

Man kann erkennen, dass sich nach 40 Epochen die Genauigkeit bei 100% befindet und die Loss-Funktion den Wert 0 erreicht hat. Das ausgewählte Modell ist also in der Lage, die Klassifizierung aller gegebenen Klassen aus den Trainingsdaten zu erlernen. Somit kann im nächsten Schritt ein erster Test auf den gesamten Trainingsdaten durchgeführt werden. Abbildung 5 zeigt die Leistung des Netzes auf dem gesamten Trainingsset.

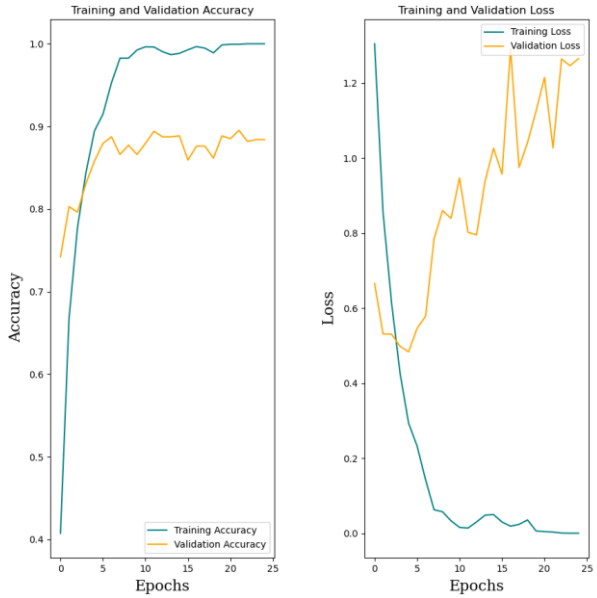

Abbildung 5: Klassifizierungsergebnis des Leistungstestes auf vollständigem Trainingsdatensatz

Die in Abbildung 5 dargestellten Ergebnisse zeigen, dass auch auf den gesamten Trainingsdaten sehr gute Trainingsergebnisse erzielt werden. Jedoch sind die Ergebnisse auf dem Validierungsset deutlich schlechter. Es ist schnell ersichtlich, dass das Modell *overfittet*, also die Trainingsdaten auswendig lernt und somit schlecht generalisiert.

B. Regularisierung

Um Overfitting zu verhindern, wurden die im folgenden aufgelisteten Regularisierungsmethoden auf das Basismodell angewandt:

1. **Data Augmentation** – Die Testdaten werden gedreht, verschoben oder die Bildhelligkeit wird verändert. Hierfür wurde die Klasse *ImageDataGenerator* der Deep Learning Bibliothek *Keras* verwendet, welche solche Augmentierungen per Zufall durchführt. Dies hat zur Folge, dass der Trainingsdatensatz um solche zufällig veränderten Bilder erweitert wird. Durch diese künstlich erhöhte Menge an Trainingsbeispielen ist das Netz in der Lage besser zu generalisieren und somit die Gefahr von *Overfitting* zu reduzieren.

2. **Parameter Norm Panalties** – Es wurden L1 und L2 Strafterme verwendet. Diese verhindern das einzelne Neuronen die Entscheidung des Netzes zu stark beeinflussen, indem sie den Fehlerwert, abhängig von den Gewichten erhöhen. Durch Gradienten abstieg werden so die Gewichtungswerte automatisch klein gehalten, da der Fehler minimiert wird und somit auch die Gewichte. Hierdurch wird verhindert, dass sich das neuronale Netz zu stark an Trainingsdaten anpasst (*Overfitting*) und es kann besser generalisiert werden.

3. **Dropout** – Wird auf eine Schicht eines neuronalen Netzes ein *Dropout* angewendet, wird ein fester Prozentsatz zufälliger Neuronen dieser Schicht, deaktiviert. Dies führt zu einer besseren Vernetzung der Neuronen untereinander und erhöht somit die Robustheit und Generalisierungsfähigkeit des Netzes. Es wurde ein *Dropout* von 20% angewandt.

Abbildung 6 zeigt die Ergebnisse des Netzes mit Regularisierung. Auf den ersten Blick scheinen diese sich verbessert zu haben, jedoch stellt man bei genauerer Betrachtung fest, dass sich nur die Skalierung der Lossfunktion verändert hat (die hohen Losswerte im Training entstehen durch die Regularisierung).

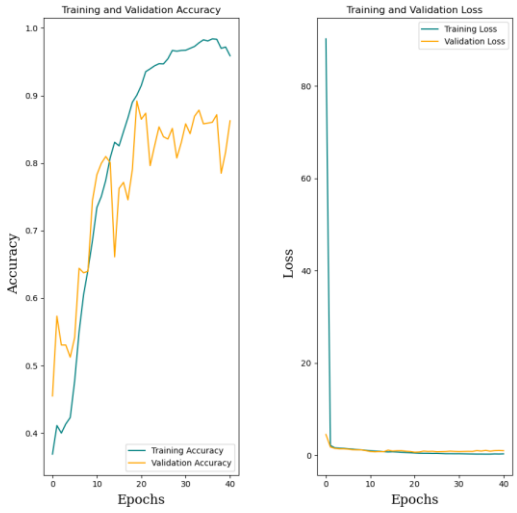

Abbildung 6: Klassifizierungsergebnis mit Regularisierung

Da durch die Anwendung der vorgestellten Regularisierungsmethoden kein besseres Klassifizierungsergebnis erreicht werden konnte, wurden im nächsten Schritt vortrainierte Netze angewandt, um das Ergebnis zu verbessern.

C. *Vortrainierte Netze*

Wie in Kapitel V erwähnt, wurden als letztes versucht, das Klassifizierungsergebnis durch vortrainierte Netze zu verbessern. Abbildung 7 zeigen die Ergebnisse mit „VGG-16“ als Basismodelle (Eine Grafik der Ergebnisse mit „ResNet“ als Basismodell befindet sich im Anhang [Abb. 11]).

Beide Modelle wurden mit *early stopping (val_loss, patience=20)* und ohne *data augmentation* trainier und zeigen eine maximale Validierungsgenauigkeit von 98%.

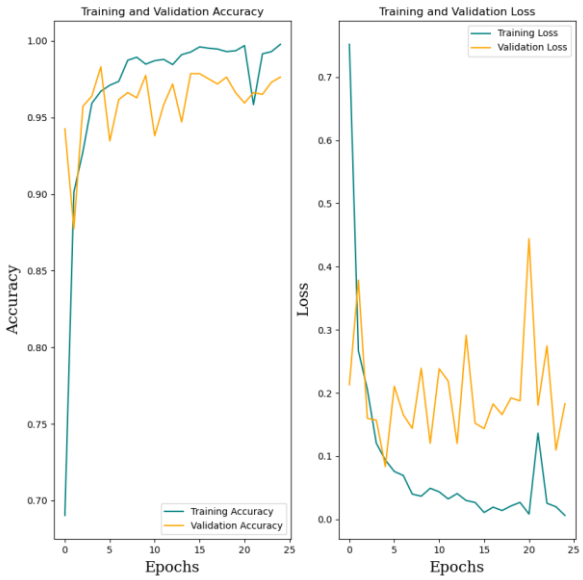

Abbildung 7: Ergebnisse des Modells mit dem Basismodell VGG-16

VI. Ergebnisanalyse

Um herauszufinden, bei welchen Datensatzinstanzen das Netz Klassifizierungsfehler macht, wurde die in Abbildung 8 dargestellte Konfusionsmatrix erstellt. Diese zeigt für welche Klassenlabel, welche Klassifizierungen gemacht worden sind.

Insgesamt sind 16 der 888 Bilder im Validierungsdatensatz falsch klassifiziert worden. Es lässt sich jedoch erkennen, dass alle Instanzen der Klassen „Schere“ und „Stein“ korrekt klassifiziert worden sind, während die Klasse „Rest“ die meisten Fehler aufweist. Abbildung 9 zeigt beispielhaft, drei der falsch klassifizierten Bilder.

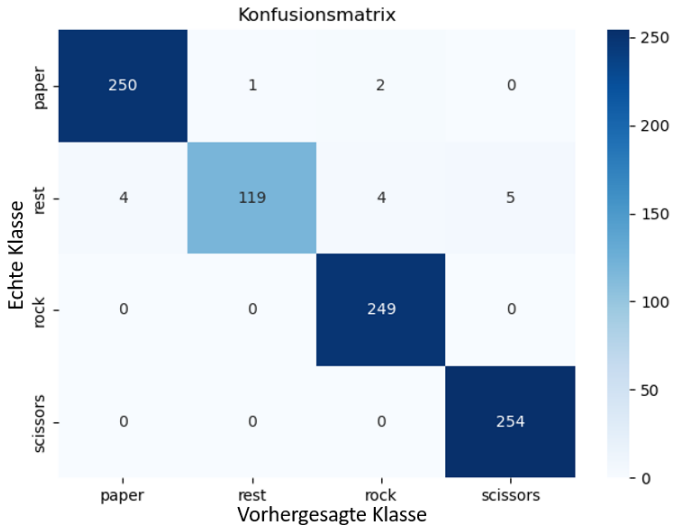

Abbildung 8: Konfusionmatrix der Vorhersageergebnisse des Modells auf Basis des VGG-Netzes

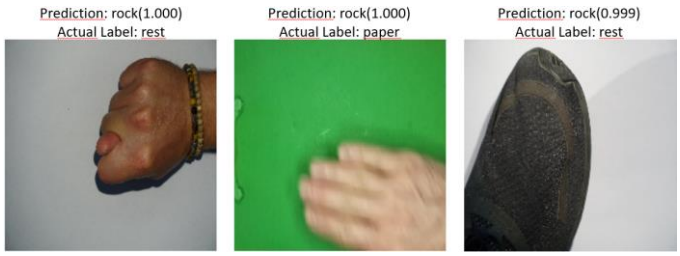

Abbildung 9: Beispiele für falsch klassifizierte Bilder

VII. Fazit

Abschließend lässt sich festhalten, dass eine Validierungsgenauigkeit von 98% für die gegebene Problemstellung ein gutes Ergebnis darstellt. Betrachtet man die Bilder, welche nicht korrekt klassifiziert worden sind, kann man gut nachvollziehen, warum das neuronale Netz hier gescheitert ist.

Würde man dennoch eine accuracy von 100% erreichen wollen, könnte man in einem nächsten Schritt die Kapazität des Modells erhöhen, um solche komplexeren Klassifizierungen, wie die in Abbildung 9 gezeigten, ebenfalls zu bewältigen. Hierbei müsste jedoch ein erhöhtes Risiko von Overfitting berücksichtigt werden (gegeben durch das komplexere Modell). Ein deutlich größerer Datensatz sollte jedoch in der Lage sein, dieses Risiko zu minimieren und dem neuronalen Netz helfen, besser zu generalisieren. Außerdem könnten durch mehr Daten weitere spezielle Fälle (wie die in Abbildung 9 gezeigten) abgedeckt werden.

Eine weitere Möglichkeit die Klassifizierungsgenauigkeit zu steigern könnte sein, alle Bilder der Klasse „Rest“ durch *Clustering* oder *Anomaly Detection* bereits vor der eigentlichen Klassifizierung herauszufiltern (hierfür würden sich *unsupervised learning* Methoden eignen). Im Anschluss müssten dann nur noch die drei Klassen „Schere“, „Stein“ und „Papier“ klassifiziert werden, wodurch die Genauigkeit gesteigert werden würde.

VIII. ANHANG

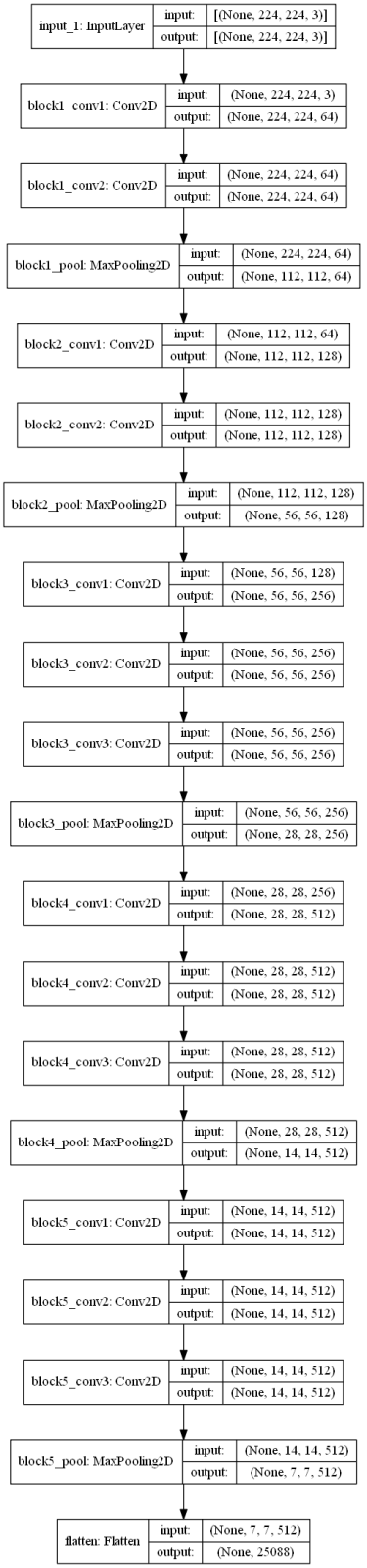

Abbildung 10: Architektur des vortrainierten Netzes VGG

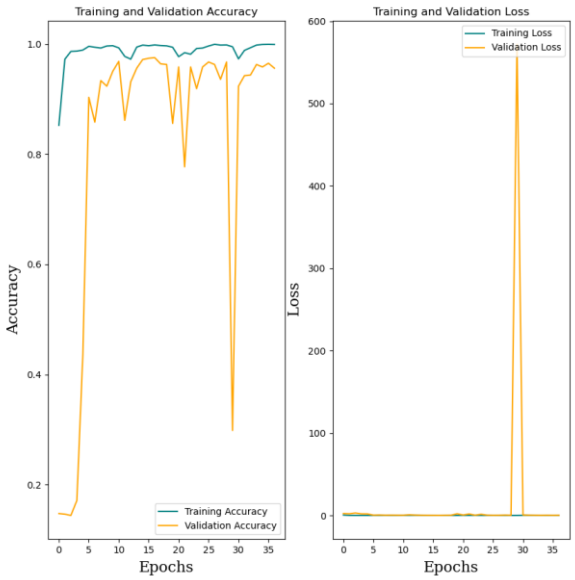

Abbildung 11: Ergebnisse des Modell mit dem Basismodell *ResNet*

IX. LITERATURVERZEICHNIS

[1] J. d. l. Bruère-Terreault, „Kaggle, Rock-Paper-Scissors Images,“ [Online].Available: https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors.

[2] J. Salmen, „Vorlesungsfolien „Deep Learning“,“ [Online].

[3] A. Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, O’Reilly Media, 2019.

[4] S. V. Lab, „ImageNet,“ 11 03 2021. [Online]. Available: https://www.image-net.org/.

[5] Keras, „Keras API reference,“ [Online]. Available: https://keras.io/api/preprocessing/image/.

[6] A. Naresh, „Kaggle, RockPaperScissors 100% Accuracy,“ [Online]. Available: https://www.kaggle.com/code/recursion17/rockpaperscissors-100-accuracy.

QUELLENANGABEN

[1] Jan Salmen, Vorlesungsfolien „Deep Learning” S. 31

[2] S. XV