Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Masterarbeit

Objekterkennung mit Hilfe von Convolutional Neural Networks
am Beispiel ägyptischer Hieroglyphen

Leipzig, November 2016

vorgelegt von
J. Nathanael Philipp
Studiengang Informatik

Betreuender Hochschullehrer: Prof. Dr. Gerhard Heyer
Fakultät für Mathematik und Informatik
Abteilung Automatische Sprachverarbeitung

Kurzzusammenfassung

1. Deutsch

Im Rahmen dieser Arbeit wurden unterschiedliche Architekturen von Convolutional Neural Networks (CNN) auf ihre Eignung für die Objekterkennung am Beispiel von ägyptischen Hieroglyphen untersucht.

Zunächst werden Grundlagen zu künstlichen neuronalen Netzen und den Bestandteilen von CNNs, wie Convolutional-Layer, eingeführt und erklärt, dann folgen Erläuterungen zu den verwendeten Datensätzen und die damit verbundenen Schwierigkeiten. Es werden anschließend Bibliotheken für konkrete Implementierung und Verwendung von künstlichen neuronalen Netzen vorgestellt und der Ablauf der Evaluation der Objekterkennung beschrieben.

Die CNNs wurden mit unterschiedlichen Anzahlen an Klassen und der damit verbundenen Anzahl an Bildern trainiert und evaluiert. Die Experimente wurden anhand der drei benutzen Trainingsmethoden unterteilt. Bei der ersten Methode wurden die CNNs mit Hilfe von Autoencodern vortrainiert. Bei der Zweiten wurden die CNNs blockweise trainiert und bei der dritten Methode wurden tiefere Netzarchitekturen untersucht. Es wurden dabei unterschiedliche, in der Literatur beschriebene, Netzarchitekturen wie Residual Networks (ResNet) und Densely Connected Convolutional Networks nach implementiert und evaluiert.

Die Ergebnisse der durchgeführten Experimente zeigen zunächst, dass es möglich ist, CNNs mit bis zu 69 Convolutional-Layer zu trainieren, etwa 6500 unterschiedliche ägyptische Hieroglyphen zu klassifizieren und schlussendlich eine Objekterkennung mit sehr guten Ergebnissen durchzuführen. Die besten Ergebnisse für die Objekterkennung $0,92362$ wurden dabei für ein CNN mit 6465 Klassen und 13 Convolutional-Layer erreicht.

2. Englisch

In this thesis different architectures of Convolutional Neural Networks (CNN) and their suitability for object recognition were investigated by using the example of Egyptian hieroglyphs.

First, basic principles for artificial neural networks and the components of CNNs, such as convolutional layers, are introduced and explained, followed by explanations

of the used data sets and the associated difficulties. We then present libraries for the concrete implementation and use of artificial neural networks and describe the sequence of the evaluation of object recognition.

The CNNs were trained and evaluated with different numbers of classes and the associated number of images. The experiments are divided by the three used training methods. For the first, the CNNs were trained with the help of autoencoders. For the second, the CNNs were trained block-wise, and for the third deeper network architectures were investigated. Various network architectures such as Residual Networks (ResNet) and Densely Connected Convolutional Networks, described in the literature, were implemented and evaluated.

The results of the experiments show that it is possible to train CNNs with up to 69 convolutional layers, to classify about 6500 different Egyptian hieroglyphs and finally to carry out an object recognition with very good results. The best results for object detection 0.92362 were achieved for a CNN with 6465 classes and 13 convolutional layers.

Danksagung

Mein Dank gilt Prof. Dr. Gerhard Heyer für die Möglichkeit, dieses interessante Thema in einer Masterarbeit zu betrachten. Bei der Abteilung Automatische Sprachverarbeitung möchte ich mich für die hilfreiche und angenehme Zusammenarbeit bedanken. Weiterhin gilt mein Dank Maximilian Bryan, der mir immer mit hilfreichen Hinweisen und konstruktiven Diskussionen zur Seite stand. Zum Schluss möchte ich noch meiner Schwägerin, Frau Rita Philipp, danken, die mit viel Geduld meine Schreibfehler korrigiert hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In den letzten Jahren haben künstliche neuronale Netze in vielen Bereichen zu neuen Ansätzen und Verfahren beim maschinellen Lernen geführt und dadurch viele bestehenden Verfahren abgelöst. In einigen Bereichen haben sie sogar bessere Leistungen als Menschen erreicht. Im Bereich der Bilderkennung und -klassifizierung wurden dabei die größten und beeindruckendsten Fortschritte gemacht.

Aber auch in den Medien wird vermehrt über künstliche neuronale Netze berichtet. Besonders Schlagwörtern wie *Deep Learning* oder *künstliche Intelligenz* haben in diesem Zusammenhang an Popularität gewonnen und prägen die öffentliche Meinung, gleichermaßen schüren sie allerdings auch Ängste und befeuern die Euphorie.

Es liegt eine Vielzahl an digitalisierten Büchern, meist Wörterbücher, der frühen Ägyptologie um 1900 vor. Langfristiges Ziel ist es, eine automatische Zeichenerkennung (OCR) dieser Bücher durchzuführen. In dieser Arbeit werden dafür erste Grundlagen gelegt.

Es werden dazu unterschiedliche Architekturen künstlicher neuronaler Netze, genauer Convolutional Neural Networks (CNN), vorgestellt und untersucht. Die unterschiedlichen Architekturen werden dabei hinsichtlich der Erkennung und Klassifizierung von ägyptischen Hieroglyphen untersucht. Es soll vor Allem eine Objekterkennung der Hieroglyphen erreicht werden.

1.1. Motivation

Bisherige Verfahren zur Bild- und Objekterkennung basierten meist auf handgefertigten Merkmalen. Mit der Renaissance von künstlichen neuronalen Netzen in den letzten Jahren sind bestehende Verfahren abgelöst worden. Insbesondere Convolutional Neural Networks (CNN) haben im Bereich der Bild- und Objekterkennung „state of the art“-Ergebnisse geliefert und sind damit praktisch schon zu Standardlösungen geworden. Ein großer Vorteil von künstlichen neuronalen Netzen ist, dass nicht länger Merkmale von Hand angefertigt werden müssen, sondern dass diese sie selbständig lernen. Durch die sich ständig weiterentwickelnde Technik und Forschung werden hier immer wieder neue und tiefere Architekturen vorgestellt.

Ägyptische Hieroglyphen zeichnen sich einerseits mit ca. 7000 unterschiedlichen Zeichen durch ihre Vielfalt, andererseits durch die gleichzeitig bestehende Ähnlichkeit, da sich viele Hieroglyphen nur geringfügig von einander unterscheiden, aus.

Dies macht sie zu einem interessanten, aber auch komplizierten Datensatz, der allein schon einfache Klassifikationsaufgaben extrem schwierig macht. Darüber hinaus ist die ägyptische Zeichensprache eine sehr komplexe Sprache, die sich über die Zeit stark gewandelt hat und außerdem über keine einheitliche Schriftrichtung verfügt.

1.2. Ziel der Arbeit

Das Ziel dieser Arbeit ist es, mittels Convolutional Neural Networks (CNN), eine Objekterkennung von ägyptischen Hieroglyphen zu entwickeln und durchzuführen. Dazu werden unterschiedliche Architekturen von CNNs betrachtet. Zusätzlich werden verschiedene Trainingsmethoden vorgestellt und es wird untersucht, wie sich diese mit einander vereinbaren lassen. Im weiteren Verlauf werden die Ergebnisse der verschiedenen Architekturen miteinander verglichen. Es wird zunächst die grundlegende Möglichkeit zur Klassifizierung der ägyptischen Hieroglyphen betrachtet und anschließend die Qualität der Objekterkennung.

Ziel ist es dabei nicht, ein neuronales Netz zu entwickeln, mit dem die gesamte ägyptische Schriftsprache abgedeckt ist, sondern vielmehr die grundsätzliche Möglichkeit der Erkennung in einem abgegrenzten Bereich aufzuzeigen.

1.3. Aufbau der Arbeit

Diese Arbeit ist in vier Kapitel unterteilt und enthält zusätzlich noch einen Anhang. Im zweiten Kapitel werden die Grundlagen erläutert, die für das Verständnis der folgenden Kapitel die Basis bilden. Im folgenden dritten Kapitel werden die durchgeführten Experimente erläutert und zusätzlich einige wichtige Grundlagen für die unterschiedlichen Netzarchitekturen erläutert. Das vierte und letzte Kapitel enthält neben einer Zusammenfassung einen Ausblick auf zukünftige Arbeiten. Der Anhang enthält zusätzliche Informationen zu den Ergebnissen der einzelnen Experimente.

2. Grundlagen

Im folgenden Kapitel werden die Grundlagen beschrieben, beginnend mit künstlichen neuronalen Netzen, gefolgt von einem Abschnitt über ägyptische Hieroglyphen und zum Abschluss ein Abschnitt über die verwendeten Bibliotheken.

2.1. Künstliche neuronale Netze

Bei künstlichen neuronalen Netzen (engl. Artificial neural networks) handelt es sich um eine Familie von Modellen des maschinellen Lernens, die auf der Struktur des Gehirns basieren. Die meisten künstlichen neuronalen Netze werden zur Gruppe der überwachten Lernverfahren gezählt.

Die Anfänge gehen dabei auf Warren McCulloch und Walter Pitts[25] zurück, die 1943 ein Rechenmodell beschrieben haben, das prinzipiell jede logische oder arithmetische Funktion berechnen kann. Später haben sie darauf hingewiesen, dass ein solches Netz ebenfalls zur räumlichen Mustererkennung eingesetzt werden kann.

Künstliche neuronale Netze bestehen aus Layer (Schichten) von Neuronen. Eine schematische Darstellung eines künstlichen neuronalen Netzes ist in Abbildung 2.1 zu sehen.

Das Rechenmodell, welches McCulloch und Pitts beschrieben, war allerdings praktisch nicht trainierbar. Dies änderte sich, als der Psychologe Donald Hebb im Jahre 1949 seine Lernregel formulierte: *„Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability.… When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.“*[13].

Die Hebbsche Lernregel beschreibt die Veränderung zweier gemeinsam aktiver Neuronen als Gewichtsänderung, definiert als $\Delta\omega_{ij} = \eta a_i o_j$. $\Delta\omega_{ij}$ ist die Veränderung des Gewichtes von Neuron i zu Neuron j, η ist Lernrate, ein konstanter Faktor, a_i ist die Aktivierung von Neuron i und o_j ist die Ausgabe von Neuron j.

Das bedeutet, dass, je häufiger zwei Neuronen zusammen aktiv sind, sie umso stärker aufeinander reagieren werden. Dies hat Hebb auch nachgewiesen. Die Hebbsche Lernregel beschreibt in ihrer allgemeinen Form fast alle Lernverfahren für neuronale

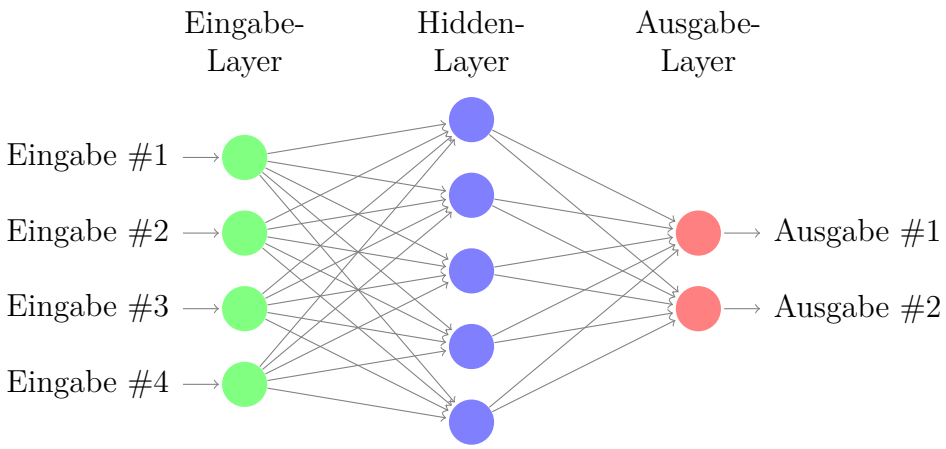


Abbildung 2.1.: Schematische Darstellung eines künstlichen neuronalen Netzes mit einem Eingabe- und Ausgabe-Layer sowie einem Hidden-Layer.

Netze. Für das Training eines künstlichen neuronalen Netzes bedeutet dies vereinfacht dargestellt, dass an der Ausgabe eines Neurones/Layer ein Fehler berechnet wird, anhand dessen die Gewichte geändert werden, wozu man allgemein ein Gradientenverfahren nutzt.

Im Jahr 1974 entwickelte Paul Werbos mit der Backpropagation[34] ein Verfahren, das bis heute die Grundlage für Methoden zum Trainieren von künstlichen neuronalen Netzen ist. Der Backpropagation-Algorithmus arbeitet in drei Phasen. In der ersten Phase werden die Daten durch das Netz propagiert und deren Ausgabe berechnet. In der zweiten Phase wird die Ausgabe mit dem gewünschten Wert verglichen und die Differenz berechnet, der Fehler. In der dritten Phase wird der Fehler zurück durch das Netz propagiert, vom Ausgabe-Layer zum Eingabe-Layer, dabei werden die Gewichte anhand ihres Einflusses auf den Fehler geändert.

Das erste *convolutional neural network* (CNN) wurde von LeCun et al.[22] 1989 eingeführt. CNN sind benannt nach den Convolutional-Layer, die die Hauptbestandteile des Netzes sind. Die CNNs sind inspiriert von der Organisation des tierischen visuellen Kortex. Die genaue Funktionsweise von CNNs wird in Abschnitt 2.1.6 beschrieben.

Künstliche neuronale Netze sind sehr mächtig. Unterschiedliche Netze sind in vielen Gebieten des Maschinellen Lernens die „state of the art“-Methoden und haben bei einigen Wettbewerben sogar besser abgeschlossen als Menschen. So hat zum Beispiel im Jahr 2012 zum ersten Mal bei ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) ein CNN (AlexNet[21]) gewonnen. 2015 wurden dann zum ersten Mal bessere Ergebnisse erzielt als von Menschen[11][17].

Mit diesen Arbeiten wurde auch der Begriff des *Deep Learning*[5] in den Raum

gestellt, um diese neuen Ansätze von künstlichen neuronalen Netzen zu gruppieren. Dabei bezieht sich *Deep Learning* hauptsächlich darauf, dass die Netze immer tiefer werden, d.h. mehr Layer haben, besonders im Vergleich zu den Netzen der 70er und 80er Jahre. Durch diese größere Anzahl an Layern lernen die Netze immer komplexere Features. Auch unterscheiden sich die heutigen Netze insofern, dass Millionen von Trainingsdaten verwendet werden und in den meisten Fällen auf GPUs (Grafikkarten) oder ganzen Clustern (auch GPU-Cluster) berechnet werden.

Des Weiteren gibt es Ansätze mit künstlichen neuronalen Netzen im Bereich des bestärkenden Lernens (engl. reinforcement learning). Hier wurden beispielsweise Netze trainiert, die die alten Atari Spiele, wie zum Beispiel Pong[1], spielen oder Mikromanagementaufgaben in StarCraft vollführen wie das Kontrollieren von Armeen im Kampf[32].

Im nachfolgen Abschnitt wird die genaue Funktionsweise eines künstlichen neuronalen Netzes beschrieben. In den darauffolgenden Abschnitten werden einzelne Aspekte noch genauer erläutert.

2.1.1. Funktionsweise von künstlichen neuronalen Netzen

Der nachfolgenden Abschnitt beschreibt die Funktionsweise von künstlichen neuronalen Netzen. Zum vereinfachten Verständnis wird das kleinst mögliche Netz bestehend aus einem einzigen Neuron betrachtet.[23]

Das Neuron hat die Eingaben $x_i \in X$, mit $i \in 1, \ldots, N$ und N die Anzahl an Eingabewerten. Jede Eingabe wird gewichtet mit w_i. Ein Neuron berechnet entsprechend die Funktion $f_n(x) = \sum_i w_i * x_i$.

Die tatsächliche Ausgabe eines Neurons hängt von der Aktivierungsfunktion f_A (vgl. Abschnitt 2.1.3) ab. Die Aktivierungsfunktion bestimmt, wie ein Neuron auf eine Eingabe reagiert.

Für die weitere Betrachtung hier sei die Aktivierungsfunktion $f_A(x) = x$. Die Ausgabe des Neurones ist somit $y = f_A(f_n(x)) = f_A(\sum_i w_i x_i) = \sum_i w_i x_i$. Das neuronale Netz berechnet somit die Funktion $f_{\mathbf{knn}}(x) = f_A(f_n(x)) = y$.

Am Beispiel eines konkreten Netzes mit einem Neuron, das zwei Eingabe- und einen Ausgabewert hat. Die Gewichte sind zu Beginn $w_0 = 0,5; w_1 = 0,7$.

Das Lernen in einem Netz mit einem Neuron läuft folgendermaßen ab: Angenommen die Eingabewerte sind $x_0 = 2; x_1 = 3$, die Gewichte $w_0 = 0,5; w_1 = 0,7$ und der erwartete Ausgabewert wäre $t = 0$. Daraus ergibt sich die Gleichung $y(x; w) = 0,5 * 2 + 0,7 * 3 = 3,1 \neq 0$.

Es wird nun der Loss (Fehler) anhand einer Fehlfunktion (vgl. Abschnitt 2.1.4) berechnet. Im folgenden wird MSE (engl. mean squared error) verwendet, damit

[1] `https://karpathy.github.io/2016/05/31/rl/`

ergibt sich ein Loss von $MSE = (t-y)^2 = (3,1-0)^2 = 9,61$. Ziel des Trainingsprozesses ist es, die Fehlfunktion zu minimieren, weswegen die Fehlfunktion oft auch als Zielfunktion bezeichnet wird. Dies wird erreicht durch eine Anpassung der Gewichte.

Mittels des Backpropagation-Algorithmus[34] ergibt sich für die Gewichtsänderung die Formel $\Delta w_{ij} = \eta \delta_j x_i$, mit η einer konstanten Lernrate und $\delta_j = f'_A(x_j)(y_j - t_j)$ für den Fall, dass j ein Ausgabeneuron ist, wie im Beispiel. Es ergibt sich somit $\delta_0 = 1 * (0 - 3,1) = -3,1$ und mit einer Lernrate von $\eta = 0,1$ ist die Gewichtsänderung $\Delta w_{00} = 0,1 * -3,1 * 2 = -0,62$ und $\Delta w_{10} = 0,1 * -3,1 * 3 = -0,93$. Die neuen Gewichte werden berechnet mittels $w_{ij}^{\text{neu}} = w_{ij}^{\text{alt}} + \Delta w_{ij}$ und sind entsprechend $w_{00}^{\text{neu}} = w_{00}^{\text{alt}} + \Delta w_{00} = 0,5 - 0,62 = -0,12$ und $w_{10}^{\text{neu}} = w_{10}^{\text{alt}} + \Delta w_{10} = 0,7 - 0,93 = -0,23$.

Damit liefert das Netz jetzt $y(x; w) = -0,12 * 2 + -0,23 * 3 = -0,93 \neq 0$. Richtig ist das Ergebnis immer noch nicht, der Loss hat sich aber erheblich verringert auf $MSE = (t-y)^2 = (-0,93-0)^2 = 0,8649$. Ein solcher Trainingsschritt vom Propagieren der Daten, dem Zurückpropagieren des Loss, und des Änderns der Gewichte wird als Epoche bezeichnet. Der gesamte Trainingsprozess besteht aus mehreren Epochen. In der Praxis wird entweder eine feste vorgegebene Anzahl an Epochen trainiert, oder es wird solange trainiert, bis sich der Loss nicht mehr signifikant verbessert. Alternativ kann die Genauigkeit (engl. accuracy) als Maßstab zur Evaluation des Netzes genommen werden. Eine weitere Möglichkeit ist die Verwendung eines zweiten Datensatzes (validation) und deren Betrachtung bezüglich Loss oder Genauigkeit.

Bei sehr großen Datenmengen ist es nicht möglich, die gesamte Datenmenge im Speicher zu halten, so werden diese in Batches unterteilt und abgearbeitet. Im Training werden dann pro Epoche die Daten Batch-weise abgearbeitet und die Gewichtsänderungen vorgenommen.

2.1.2. Autoencoder

Eine Spezialform von künstlichen neuronalen Netzen sind Autoencoder. Bei ihnen sind Ein- und Ausgaben in der Regel gleich, es gibt aber auch Fälle, in denen sie sich unterscheiden, aber nicht die selbe Beziehung haben wie beim überwachten Lernen. Autoencoder werden deshalb zu den unüberwachten Lernverfahren gezählt. Ein Autoencoder besteht aus zwei Teilen: einem Encoder und einem Decoder. Die Eingabe wird zunächst im Encoder encodiert, d.h. auf eine Schicht mit einer geringeren Anzahl an Knoten als in der Eingabeschicht, abgebildet. Anschließend decodiert der Decoder die encodierten Daten wieder. Ziel ist es, dass das Netz automatisch lernt, wie die Daten am besten encodiert werden müssen, um die Eingabe wieder herzustellen. In Abbildung 2.2 ist eine schematische Darstellung eines Autoencoders abgebildet.

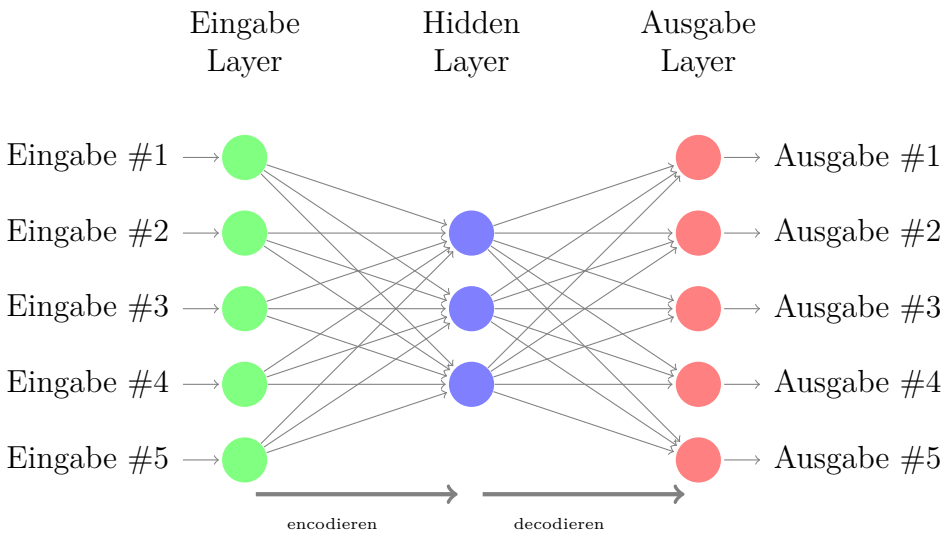


Abbildung 2.2.: Schematische Darstellung eines Autoencoders.

Wird der Encoder als Funktion $f(X) = v$ aufgefasst, so kann der Decoder als die inverse Funktion des Encoder $f^{-1}(v) = X$ gesehen werden. Der Autoencoder berechnet damit die Funktion $f^{-1}(f(X)) = X$. Betrachtet man die Layerstruktur, so ist für jedes Layer im Encoder ein entsprechendes gegensätzliches Layer im Decoder an der reversen Position zu finden.

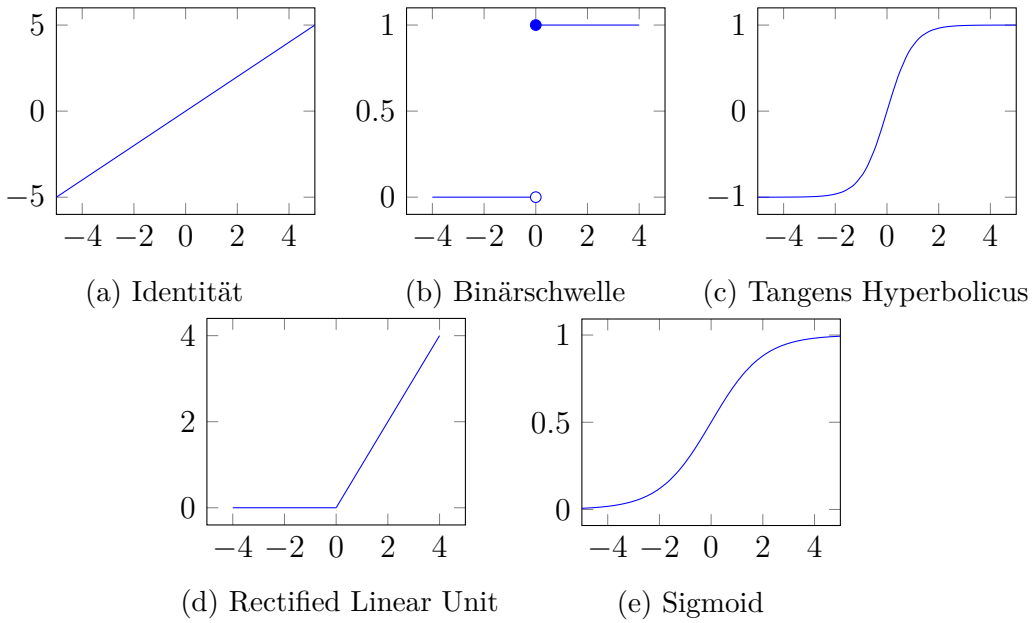
Durch den Aufbau eines Autoencoders und die Dimensionsreduktion hinsichtlich der Ausgabe des Encoders, dient der Autoencoder dazu, selbständig abstrakte Zusammenhänge in den Daten zu finden. Die Daten sollen dabei möglichst klein abgebildet werden, aber noch rekonstruierbar sein. Aus diesem Grund gibt es viele Ansätze, bei denen entweder die Ausgaben des Encoders genommenen werden, um weite Analysen zu machen (Word2Vec), oder die Gewichte des Autoencoders werden als Initialwerte anderer Netze genommen.

2.1.3. Aktivierungsfunktionen

Es gibt verschiedene Aktivierungsfunktionen, die unterschiedliche Auswirkungen auf den Ausgabewert eines Neurons haben. Eine Aktivierungsfunktion sollte vorzugsweise die folgenden Eigenschaften erfüllen:[9]

- **Nichtlinear** Durch die Nichtlinearität ist ein zweischichtiges Netz ein universeller Funktionsapproximator. Ein mehrschichtiges Netz hingegen, das die Identität als Aktivierungsfunktion benutzt, ist äquivalent zu einem einschichtigen Netz.

- **Durchgehend differenzierbar** Diese Eigenschaft ist erforderlich für gradientenbasierende Optimierungsverfahren.
- **Monoton** Bei einer monotonen Aktivierungsfunktion ist es garantiert, dass die Fehlerfläche eines einschichtigen Netzes konvex ist.
- $f(x) \approx 0$ **für** $x \approx 0$ Diese Eigenschaft hilft, das Netz effizient zu trainieren, wenn die Gewichte mit kleinen zufälligen Werten initialisiert worden sind, ansonsten ist bei der Initialisierung der Gewichte besondere Vorsicht geboten.
- **Wertebereich** Bei einem endlichen Wertebereich der Aktivierungsfunktion laufen gradientenbasierende Optimierungsverfahren in den meisten Fällen stabiler. Bei einem unendlichen Wertebereich funktioniert das Training im Allgemeinen besser, allerdings sind kleinere Lernraten in den meisten Fällen erforderlich.



(a) Identität (b) Binärschwelle (c) Tangens Hyperbolicus

(d) Rectified Linear Unit (e) Sigmoid

Abbildung 2.3.: Aktivierungsfunktionen

Identität ist eine einfache Aktivierungsfunktion, siehe Abbilding 2.3a, definiert als $f(x) = x$ mit einem Wertebereich $W = (-\infty, \infty)$. Sie ist durchgehend differenzierbar, monoton und es gilt $f(x) \approx x$ für $x \approx 0$.

Binärschwelle ist definiert als $f(x) = \begin{cases} 0 \text{ für } x < 0 \\ 1 \text{ für } x \geq 0 \end{cases}$ mit einem Wertebereich $W = \{0, 1\}$, siehe Abbilding 2.3b. Die Binärschwelle ist monoton aber für $x = 0$ nicht differenzierbar und alle anderen Werte differenziert sie zu 0, ebenfalls gilt nicht $f(x) \approx x$ für $x \approx 0$.

Tangens Hyperbolicus oder kurz tanh ist definiert als $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ mit einem Wertebereich $W = (-1, 1)$. Der tanh ist durchgehen differenzierbar, monoton und es gilt $f(x) \approx x$ für $x \approx 0$.

Rectified Linear Unit (ReLU) ist definiert als $f(x) = \begin{cases} 0 \text{ für } x < 0 \\ x \text{ für } x \geq 0 \end{cases}$ mit einem Wertebereich $W = [0, \infty)$. ReLU ist monoton, durchgehend differenzierbar aber es gilt nicht $f(x) \approx x$ für $x \approx 0$.

Parametric Rectified Linear Unit (PReLU)[12], ist eine Erweiterung von ReLU um einen Parameter a, der lernbar ist (vgl. Gleichung 2.1). ReLU kann dazuführen, dass Neuronen faktisch *tot* sind, da für $x < 0$ die Werte 0 sind und hier gradientenbasierte Verfahren keine Gewichtsänderungen mehr vornehmen können.

S-shaped Rectified Linear Activation Units (SReLU)[18] ist ebenfalls eine Erweiterung von ReLU, hat aber vier lernbare Parameter (vgl. Gleichung 2.2). Dadurch kann SReLU sowohl konvexe als auch nicht-konvexe Formen annehmen, je nachdem was beim Trainieren zu besseren Ergebnissen führt.

$$f(x) = \begin{cases} x, & \text{für } x > 0 \\ ax, & \text{für } x \geq 0 \end{cases} \tag{2.1}$$

$$f(x) = \begin{cases} t_i^r + a_i^r(x_i - t_i^r), & x_i \leq t_i^r \\ x_i, & t_i^r > x_i > t_i^l \\ t_i^l + a_i^l(x_i - t_i^l), & x_i \leq t_i^l \end{cases} \tag{2.2}$$

Sigmoid (Logistische Funktion) ist definiert als $f(x) = \frac{1}{1+e^{-x}}$ mit einem Wertebereich $W = (0, 1)$. Sie ist durchgehend differenzierbar, monoton und es gilt ebenfalls nicht $f(x) \approx x$ für $x \approx 0$.

Softmax [2] wird in der Regel nur beim letzten Layer eines neuronalen Netzes benutzt, da es die Ausgabe so verändert, dass eine Wahrscheinlichkeitsverteilung dargestellt wird. Alle Werte sind zwischen 0 und 1 und die Summe aller Ausgabewerte

[2] `http://www.thefullwiki.org/Softmax_activation_function`

ist 1 (vgl. Gleichung 2.3). Die Softmax-Aktivierung ist eine Verallgemeinerung der logistischen Funktion.

$$p_i = \frac{e^{q_i}}{\sum_{j=1}^{n} e^{q_j}} \tag{2.3}$$

2.1.4. Fehlfunktionen

Eine Fehlfunktion, auch Kostenfunktion oder Zielfunktion gennant, ist eine Funktion, die die Ausgabewerte eines neuronalen Netzes in Verhältnis zu den erwarteten Werten setzt. Je niedriger der Wert der Fehlfunktion, desto besser hat das Netz gelernt, die Ausgabewerte zu bestimmen. Der Trainingsprozess ist somit eine Aufgabe der Funktionsminimierung.

Es gibt viele verschiedene Fehlfunktionen, die bekannteste ist MSE[24] (engl. mean squared error, vgl. Gleichung 2.4) und die logarithmische Variante MSLE[33] (engl. mean squared logarithmic error, Gleichung 2.5).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2 \tag{2.4}$$

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(\ln t_i - \ln y_i)^2 \tag{2.5}$$

Zusätzlich sei hier noch categorical crossentropy[31] (kategorische Kreuzentropie, vgl. Gleichung 2.6) aufgeführt. Bei dieser Fehlfunktion wird die Kreuzentropie zwischen einer geschätzten Verteilung T und einer echten Verteilung Y berechnet. Handelt es sich um zwei Wahrscheinlichkeitsverteilungen, so misst die Kreuzentropie die durchschnittliche Anzahl an Bits, die es braucht, um ein Ereignis aus einer Menge von Wahrscheinlichkeiten zu identifizieren.

$$H(T, Y) = -\sum_{x} Y(x) \log T(X) \tag{2.6}$$

2.1.5. Lernverfahren

Bei den Lernverfahren oder auch Optimierungsverfahren handelt es sich um die Verfahren, mit deren Hilfe die Gewichtsänderung berechnet und durchgeführt wird. Es handelt sich hier um Verfahren, die den Gradientenabstieg beschleunigen sollen, um schneller zu einem Optimum zu kommen. Das bekannteste und meist genutzte Verfahren ist SGD (engl. stochastic gradient descent)[16]. SGD versucht iterativ das

Minimum bzw. Maximum zu finden: $Q(w) = \sum_{i=1}^{n} Q_i(w)$ mit w, das $Q(w)$ minimiert und bestimmt werden muss. Im iterativen Ablauf muss der wahre Gradient von $Q(w)$ eines einzelnen Wertes approximiert werden $w = w - \eta \bigtriangledown Q_i(w)$, mit der Lernrate η. Das Setzen der Lernrate wird als besonders problematisch angesehen, da ein zu hoher Wert dazu führen kann, dass SGD divergiert, ein zu kleiner Wert hingegen, dass SGD langsamer konvergiert. Darüber hinaus ist SGD relativ langsam, es werden (zu) kleine Änderungen gemacht, weswegen viele Iterationen gebraucht werden. Hinzu kommt, dass in Umgebungen von Sattelpunkten[6] dies besonders deutlich hervortritt. Aus diesen und weiteren Gründen gibt es viele Erweiterungen und Varianten von SGD.

Eine Erweiterung beispielsweise ist die Momentum-Methode[26]. Diese merkt sich die Änderungen Δw von jeder Iteration und berechnet in der nächsten Iteration die Änderung als konvexe Kombination des Gradienten und des Vorherigen, dass heißt $\Delta w = \eta \bigtriangledown Q_i(w) + \alpha \Delta w_{t-1}$ und $w = w - \Delta w_t$, dass zu $w = w - \eta \bigtriangledown Q_i(w) + \alpha \Delta w_{t-1}$ für den Iterationsschritt bzw. der Aktualisierungsregel führt.

Adaptive Gradient Algorithm (AdaGrad)[7] ist ein modifizierter SGD mit einer Lernrate für jeden Parameter. Dies verbessert besonders die Konvergenz auch bei selten benutzen (engl. sparse) Parametern, da die Lernrate für diese größer ist und für häufig genutzte Parameter kleiner. Formell definiert ist die Aktualisierung als $w_j = w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j$ mit dem dyadischen Produkt $G = \sum_{\tau=1}^{t} g_\tau g_\tau^T$, mit dem Gradienten $g_\tau = \bigtriangledown Q_i(w)$ bei der Iteration τ und der Diagonalen $G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2$.

Eine Variante, die im Rahmen dieser Arbeit benutzt wird, ist Adaptive Moment Estimation (Adam)[19]. Adam ist ein Algorithmus, der auf den gleitenden Durchschnitten der Gradienten und ihrer Beträge basiert. Adam ist definiert durch die Gleichungen 2.7, mit m_t als der gleitende Durchschnitt der Gradienten, v_t als der quadrierte Gradient, $\hat{m}_t$ und $\hat{v}_t$ als Bias korrigierten Werte für m_t und v_t, γ_1 und γ_2 als „Vergessens“-Faktoren und abschließend die Gleichungen 2.7e als Aktualisierungsregel für die Gradienten.

Die Aktualisierungsregel für einzelne Gewichte besagt, dass ihre Gradienten invers proportional zu ihrer (skalierten) L^2-Norm ihrer aktuellen und vergangenen individuellen Gradienten zu skalieren ist. Die L^2-Norm basierende Aktualisierungsregel kann zur L^p-Norm Basierenden generalisiert werden. Diese Varianten werden numerisch instabil für große p, aber für den Fall, dass $p \to \infty$ ergibt sich ein einfacher und stabiler Algorithmus: AdaMax.

$$m(w,t) = \gamma_1 m(w,t-1) + (1-\gamma_1) \bigtriangledown Q_i(w) \tag{2.7a}$$

$$v(w,t) = \gamma_2 v(w,t-1) + (1-\gamma_2) \left(\bigtriangledown Q_i(w)\right)^2 \tag{2.7b}$$

$$\hat{m}(w,t) = \frac{m(w,t)}{1-\gamma_1^t} \tag{2.7c}$$

$$\hat{v}(w,t) = \frac{v(w,t)}{1-\gamma_2^t} \tag{2.7d}$$

$$w = w - \frac{\eta}{\sqrt{\hat{v}(w,t)} + \epsilon} \hat{m}(w,t) \tag{2.7e}$$

Explodierender oder Verschwindender Gradienten-Problem

Als explodierendes oder verschwindendes Gradienten-Problem (engl. exploding/vanishing gradient)[14] wird das Problem bezeichnet, das zum Beispiel auftritt, wenn die Lernrate unpassend gewählt wurde und der Gradient dadurch entweder viel zu groß oder viel zu klein wird, so dass ab diesen Punkt ein Trainieren des Netzes praktisch nicht mehr statt findet.

Das Verschwinden des Gradienten kann aber auch bei sehr tiefen Netzen auftreten. Hier werden die Gewichte der letzten Layer zuerst und ordentlich geändert. Je weiter man zum Eingabe-Layer kommt, desto weniger bleibt vom Gradienten übrig, da beim Backpropagation-Algorithmus[34] der Gradient über die Kettenregel durch das Netz propagiert und dadurch mit jeder zusätzlichen Schicht minimiert wird. Schlussendlich werden die Gewichte einiger Layer nicht mehr geändert, da der Gradient gegen 0 geht.

2.1.6. Layer

Im Folgenden werden einige Layertypen genauer beschrieben, die im weiteren Verlauf der Arbeit wichtig sind.

Convolutional-Layer

In diesem Abschnitt wird die Grundlage und die Funktionsweise des Convolutional-Layers erläutert, das auch der Namensgeber für das CNN ist.

In Abbildung 2.4 ist ein CNN mit zwei Convolutional-Layer und einem Pooling-Layer (vgl. Abschnitt 2.1.6) schematisch dargestellt. In rot sind dabei die Filter des Convolutional-Layers dargestellt.

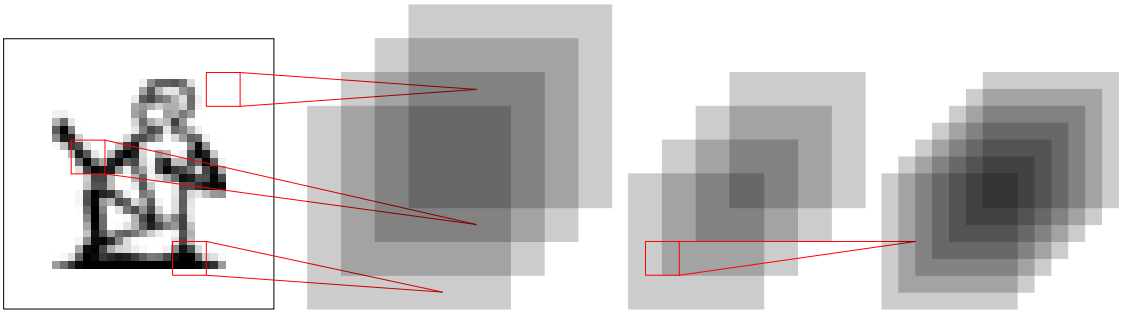


Abbildung 2.4.: Schematische Darstellung eines CNN, bestehend aus zwei Convolutional-Layer und einem Pooling-Layer.

Für die Convolution bildet die mathematische Funktion der Faltung die Grundlage. Die Faltung, wie in Gleichung 2.8 definiert, ist ein mathematischer Operator, der für zwei Funktionen f und g eine dritte Funktion $f * g$ berechnet. Veranschaulicht dargestellt ist die Faltung $f * g$ der gewichtete Mittelwert der Funktion f, mit der Gewichtung von g, der Funktionswert $f(x)$ wird dabei mit dem Wert $g(x - y)$ gewichtet, dadurch erhält der Wert x aus f einen anderen gewichteten Mittelwert.[35]

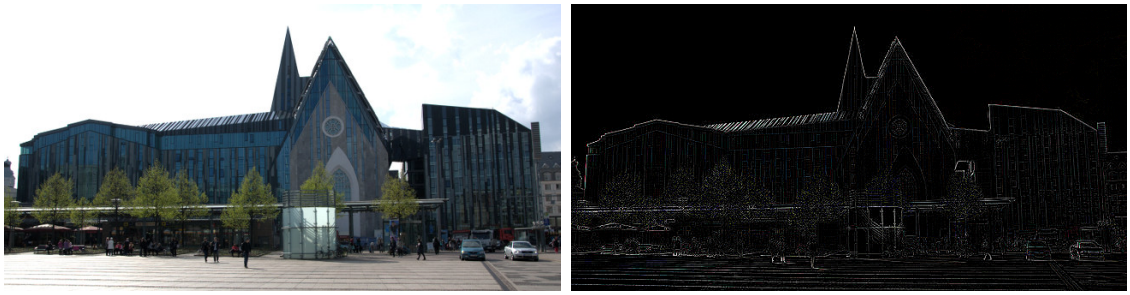
$$(f * g)(x) = \int_{\mathbb{R}^n} f(x - y)g(y)dy = \int_{\mathbb{R}^n} f(y)g(x - y)dy = (g * f)(x) \tag{2.8}$$

Ein Beispiel für die Faltung kommt aus der Stochastik. Wenn X und Y zwei stochastisch unabhängige Zufallsvariablen sind, mit den Dichtfunktionen f und g, dann ist die Dichte der Summe $X + Y$ gleich der Faltung $f * g$.

Ein weiter Anwendungsfall kommt aus der Optik bzw. der digitalen Bildbearbeitung. So können unter Anderem die Kanten in einem Bild durch die Faltung bestimmt werden. Hierbei handelt es sich um eine Faltung im zweidimensionalen Raum. Das Bild entspricht dabei der ersten Funktion f. Die zweite Funktion der Faltung g ist eine Matrix der Form Gleichung 2.9, diese wird in der Regel als Kernel bzw. Filter bezeichnet.

$$g = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{2.9}$$

In Abbildung 2.5 ist diese Faltung beispielhaft dargestellt. Das Originalbild der Universität Leipzig, in Abbildung 2.5a wurde mit dem in Gleichung 2.9 beschriebenen Filter gefaltet. Das Ergebnis der Faltung ist in Abbildung 2.5b zu sehen. Es ist dabei zu erkennen, dass durch diesen Filter nur noch die Kanten sichtbar sind.



(a) Universität Leipzig[1] (b) Ergebnis der Faltung

Abbildung 2.5.: Beispiel der Faltung für Kanten

Das Convolutional-Layer arbeitet auf der Grundlage der im vorherigen Abschnitts 2.1.6 beschriebenen Methode der Convolution. Die Eingabe eines Convolutional-Layers ist entweder ein Bild, oder die Ausgabe eines anderen Convolutional-Layers.

Anhand der Abbildung 2.6 soll dies an einem Zahlenbeispiel verdeutlicht werden. Dabei ist links das Bild als 4×4-Matrix dargestellt, in der Mitte zwei 3×3-Filter und rechts das resultierende Ergebnis.

Jeder Filter eines Convolutional-Layers wird dabei pixelweise über das ganze Bild geschoben. Für die jeweils rot markierten Zellen rechts ergeben sich dabei folgende Rechnungen, jeweils aus gehend von der gelben Zelle:

- für den oberen Filter: $3*1+2*0+1*0+3*0+2*-1+8*1+6*-1+7*0+2*0 = 3$
- für den unteren Filter: $3*0+2*1+1*1+3*-1+2*0+8*1+6*0+7*-1+2*1 = 3$

Wie in Abbildung 2.6 bereits angedeutet, kann bei einem Convolutional-Layer die Anzahl an Eingabe- und Ausgabematrizen unterschiedlich sein. Das erste Convolutional-Layer, das direkt auf dem Bild arbeitet, hat als Eingabe entweder drei Matrizen entsprechend den RGB-Werten, oder eine Matrix mit Graustufenwerten. Spätere Convolutional-Layer haben mehrdimensionale Matrizen als Eingabe. Die Filter arbeiten dabei dreidimensional auf allen Eingabematrizen.

Durch das Training werden die Werte der Filter bestimmt.

Die Randbehandlung bedarf einer genaueren Betrachtung, weil nicht zwangsläufig für alle Werte des Filters ein Wert in der Eingabe vorhanden ist. Beispielsweise bei einem 3×3 Filter der auf die linken oberen Ecke der Eingabe angewendet wird, sind vier Werte vorhanden und Fünf nicht.

Es gibt zwei Arten, wie mit den Rand umgegangen werden kann. Zum Einen kann entweder der Filter nur so über die Eingabe gelegt werden, dass er komplett in der Eingabe liegt. Bei einem 3×3 Filter verringert sich die Eingabedimension zur

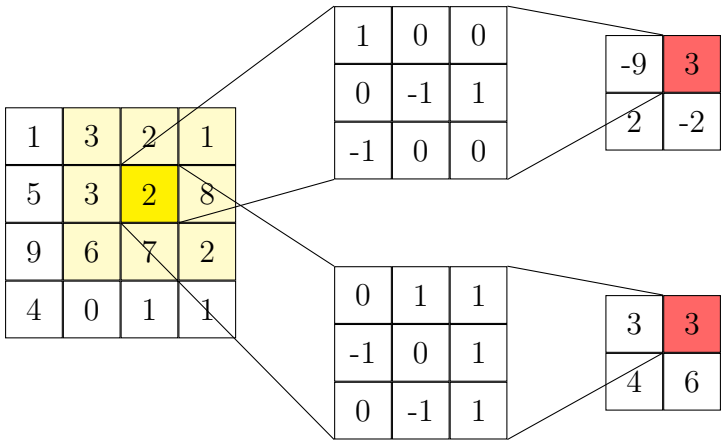


Abbildung 2.6.: Beispielhafte Darstellung der Funktionsweise eines Convolutional-Layer anhand zweier Filter.

Ausgabedimension um 2×2. Für den allgemeinen Fall lässt sich die Ausgabegröße einer Dimension wie folgt berechnen: $o = (i-k) + 1$, mit i Größe der Eingabe und k Größe des Filters.

Bei der anderen Art der Randbehandlung wird die Eingabe auf allen vier Seiten mit Nullen gefüllt. Es kann dabei zwischen implizierten und explizierten Auffüllen unterschieden werden. Beim implizierten Auffüllen werden nur entsprechend so viele Nullen hinzugefügt, bis Ein- und Ausgabe gleich groß sind.

Beim expliziten Auffüllen wird exakt angeben, wie viel aufgefüllt werden soll. Hier können natürlich die Werte so gewählt werden, dass Ein- und Ausgaben ebenfalls gleich groß sind. Möglich ist auch, dass die Ausgabe größer als die Eingabe ist.

Pooling-Layer

Im Allgemeinen arbeiten Pooling-Layer ähnlich wie die zuvor beschriebenen Convolutional-Layer. Sie haben ebenfalls einen Filter, der über die Daten geschoben wird. Der Filter hat in den meisten Fällen eine Größe von 2×2, seltener auch von 3×3. Größere Filtergrößen kommen bei den Pooling-Layer nicht vor, da sich gezeigt hat, dass diese zu zerstörerisch sind.

Im Unterschied zu den Convolutional-Layer wird bei einem Pooling-Layer zusätzlich noch eine Schrittgröße (stride) definiert. Dies bedeutet, es werden nicht alle Eingabewerte betrachtet, sondern in der Regel nur jeder Zweite (2×2).

Durch die Pooling-Layer wird eine Invarianz in den Eingaben geschaffen. Durch die Pooling-Layer soll zum Einen die räumliche Dimension und damit verbunden die Anzahl an Parametern reduziert werden. Des Weiteren dient es auch der Kontrolle von Overfitting (deu. Überanpassung).

Pooling-Layer haben keine Aktivierungsfunktionen.

Das Max-Pooling-Layer ist eine Form des Pooling-Layers, bei dem der Max-Operator angewendet wird, das heißt es wird nur der maximale Wert im Filter ausgegeben. In Abbildung 2.7 ist dies beispielhaft dargestellt. Als Beispiel anhand des gelben Bereichs ergibt sich $\max(1;3;5;3) = 5$.

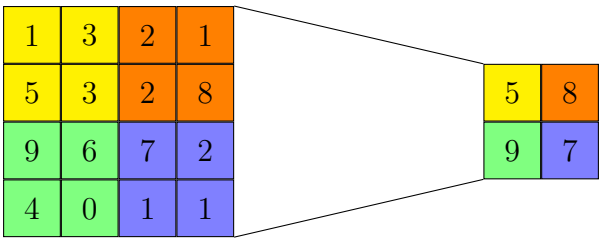


Abbildung 2.7.: Beispielhafte Darstellung der Funktionsweise eines Max-Pooling-Layer mit einer Filtergröße von 2x2 und einer Schrittgröße von 2x2.

Das Average-Pooling-Layer funktioniert so ähnlich wie das Max-Pooling-Layer, aber anstelle der Berechnung des Maximums wird hier der Mittelwert berechnet. Dies ist beispielhaft in Abbildung 2.8 dargestellt. Hier ergibt sich für den gelben Bereich folgende Rechnung: $(1+3+5+3)/4 = 3$.

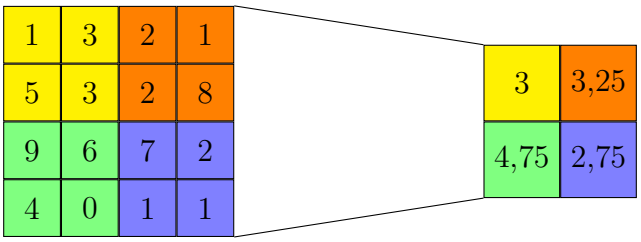


Abbildung 2.8.: Beispielhafte Darstellung der Funktionsweise eines Average-Pooling-Layers mit einer Filtergröße von 2x2 und einer Schrittgröße von 2x2.

2.1.7. Dropout

Bei Dropout[29] handelt es sich um eine Methode, mit der Overfitting vermieden werden soll. Dabei werden zufällig mit einer vorher angegebenen Wahrscheinlickeit

p pro Epoche während des Trainings Neuronen ausgeschaltet. Das bedeutet, dass mit z.B. einer Wahrscheinlichkeit von $p = 0,5$, während des Trainings nur 50% der Neuronen eines Layers aktiv sind und trainiert werden. Dabei wird pro Epoche immer wieder neu bestimmt, welche Neuronen dies genau sind. Dadurch können Neuronen nicht zu spezifische Feature lernen, da ein Neuron sich nicht darauf verlassen kann, das es immer mit einem anderen Neuron gemeinsam aktiv ist, wodurch ein mögliches Overfitting verhindert werden kann.

2.1.8. Batch Normalization

Batch Normalization[17] ist ein Normalisierungsverfahren, bei dem die Ausgabe eines Layers vor der Aktivierungsfunktion normalisiert wird, so dass der Mittelwert nahe an 0 liegt und die Standartabweichung nahe bei 1. Die Normalisierung wird während des Trainings Batch-Weise berechnet und später werden laufende Mittelwerte, die während des Trainings bestimmt worden sind, verwendet. Formal ist die Batch Normalization definiert nach den Gleichungen 2.10.

$$\text{Durchschnitt: } \mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=0}^{m} x_i \tag{2.10a}$$

$$\text{Varianz: } \sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=0}^{m}(x_i - \mu_{\mathcal{B}})^2 \tag{2.10b}$$

$$\text{Normalisierung: } \hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \tag{2.10c}$$

$$\text{Skalierung und Verschiebung: } BN_{\gamma,\beta}(x_i) \equiv y_i = \gamma\hat{x_i} + \beta \tag{2.10d}$$

Insbesondere im Zusammenhang mit Sigmoid als Aktivierungsfunktion ist Batch Normalization besonders gut geeignet, da die Ableitung der Sigmoid-Funktion beim Gradientenabstieg bei $x = 0$ den höchsten Wert mit $y = 0,25$ hat und mit größerer werdender Abweichung von 0 immer kleinere Werte annimmt.

Durch Batch Normalization wird einerseits das Problem des verschwindenden Gradienten bis zu einem gewissen Grad gelöst. Zum Anderen macht es die Verwendung von Dropout überflüssig, weil durch die Batch Normalization eine Regulation des Netzes erreicht wird. Durch zufällig zusammengesetzte Batches werden unterschiedliche Trainingsbeispiele gezeigt und somit wird das Netz keinen deterministischen Wert für ein einzelnes Trainingsbeispiel liefern. Zusätzlich wird eine Beschleunigung des Trainings erreicht, weil durch Batch Normalization die Layer, während des Trainings, eine geringere interne kovariate Verschiebung aufweisen.

2.2. Ägyptische Hieroglyphen

Bei ägyptischen Hieroglyphen handelt es sich um die ältesten Schriftzeichen des ägyptischen Schriftsystems. Sie wurden von etwa 3200 v. Chr. bis 300 n. Chr. im Alten Ägypten und Nubien für die früh-, alt-, mittel- und neuägyptische Sprache, sowie für das sogenannte ptolemäische Ägyptisch benutzt. Ursprünglich handelte es sich bei den Hieroglyphen um eine reine Bilderschrift, im weiteren Verlauf kamen weitere Zeichen mit unterschiedlichen Bedeutungen hinzu. So wurden aus ursprünglich etwa 700 Zeichen etwa 7000 Zeichen.

Die Schreibweise war ursprünglich meist in Spalten (Kolumnen) von oben nach unten und von rechts nach links geschrieben, aus graphischen Gründen konnte die Schreibrichtung jedoch auch variieren. In selteneren Fällen wurden die Hieroglyphen auch in Zeilen geschrieben. Die Schriftrichtung ist sehr leicht festzustellen, da die Zeichen immer in Richtung des Textanfanges gewandt sind, dem Leser also entgegenblicken. Dies ist am deutlichsten bei den Darstellungen von Tierformen oder Menschen erkennbar. In einzelnen Fällen, wie beispielsweise auf der Innenseite von Särgen, ist die Schrift verkehrt herum, die Hieroglyphen sind dem Textende zugewandt.

Für die Hieroglyphen existieren mehrere Nummerierungssysteme, das wichtigste ist das von A. H. Gardiner[3], auch bekannt als Gardiner-Liste. Dabei werden die Hieroglyphen in 26 Untergruppen kategorisiert. Jede Untergruppe wird durch einen Buchstaben identifiziert, so ist z. B. Gruppe A *Der Mann und seine Tätigkeiten* und Grupppe B *Die Frau und ihre Tätigkeiten*. In dieser Liste sind insgesamt 763 Zeichen.

2.2.1. Datensätze

Neben den im Folgenden beschriebenen hieroglyphischen Zeichensätzen liegt eine Vielzahl von digitalisierten Büchern der frühen Ägyptologie vor, die um 1900 entstanden sind und zusätzlich einige Bücher mit Computer-generierte Hieroglyphen. Bei einem Großteil der Bücher handelt es sich um die frühen Wörterbücher der Ägyptologie. Langfristiges Ziel ist es, die Hieroglyphen in diesen Büchern zu finden und zu erkennen.

Der erste Datensatz besteht aus drei unterschiedlichen Zeichensätzen. Für 1070 Hieroglyphen sind Bilder aus allen drei Zeichensätzen vorhanden, insgesamt 3210 Bilder. Für 6239 Hieroglyphen sind Bilder aus mindestens zwei Zeichensätzen vorhanden, insgesamt 13548 Bilder. Insgesamt sind 13980 Bilder vorhanden mit 6671 unterschiedlichen Hieroglyphen.

[3] `http://www.khemet.de/altaegypten/schrift/gardiner.html`

Der zweite Datensatz hat zusätzlich zu den Zeichensätzen aus dem ersten Datensatz noch 666 Bilder von Hieroglyphen aus Wikipedia[4] und zwei Zeichensätze, die aus Schriftarten für Hieroglyphen extrahiert wurden und jeweils die 1071 Hieroglyphen umfassen, die im Unicode-Standard definiert sind. Für 665 Hieroglyphen sind aus allen sechs Zeichensätzen 3990 Bilder vorhanden. Für 694 sind aus mindestens fünf Zeichensätzen Hieroglyphen vorhanden, 4135 Bilder. Für mindestens vier Zeichensätze sind es 846 Hieroglyphen, 4743 Bilder. Für 1286 sind aus drei Zeichensätzen Hieroglyphen vorhanden, 6063 Bilder. Für 6465 Hieroglyphen sind aus mindestens zwei Zeichensätzen Hieroglyphen vorhanden, 16421 Bilder. Insgesamt sind 6832 unterschiedliche Hieroglyphen in diesem Datensatz mit 16788 Bildern.

2.2.2. Besonderheiten

Die Datensätze sind aufgrund ihrer Zusammensetzung sehr speziell, besonders im Vergleich zu den sonst üblichen Datensätzen. So hat zum Beispiel der Datensatz des oben erwähnten ILSVRC-2012 Wettbewerbs 1,2 Million Trainingsbilder (50.000 Validierungsbilder, 100.000 Testbilder) mit 1000 Klassen. Ein weiterer beliebter Datensatz ist der CIFAR[20] Datensatz mit 50.000 Trainingsbildern und 10.000 Testbildern, unterteilt in CIFAR-10 mit 10 Klassen und CIFAR-100 mit 100 Klassen.

Der große Unterschied zwischen diesen Datensätzen ist das Verhältnis von Klassen zur Anzahl an Trainingsbildern, von hunderten/tausenden pro Klasse zu maximal sechs bei dem zweiten Hieroglyphen Datensatz. Dies macht das Training für die Erkennung der Hieroglyphen äußerst schwierig und erfordert einiges an Regulierung, so dass kein Overfitting passiert.

Ein zweiter Unterschied ist die Aufteilung der Hieroglyphen in Trainings- und Testbilder. Damit ist gemeint, dass einige Bilder nicht für das Training benutzt werden und mit ihnen später ein Test durch geführt wird, um sagen zu können, wie gut das Netz Bilder erkennt, die es zuvor nicht gesehen hat, die, ebenfalls aufgrund der geringen Anzahl an Bildern pro Klasse, nicht stattfindet.

Hinzu kommt, dass die Bilder je nach Quelle unterschiedliche Auflösungen und Formate haben. Damit sie gleichmäßig vom neuronalen Netz verarbeitet werden können, müssen sie angeglichen werden. Daher wurden alle Bilder auf die gleiche Größe skaliert und in PNGs umgewandelt.

Für das Training wurden noch weitere Vorverarbeitungsschritte durchgeführt, die sich je nach Experiment unterscheiden. Die genauen Schritte sind jeweils bei den Experimenten aufgelistet. Hier wurden auch Methoden angewandt um die Trainingsbilder künstlich „zu vermehren“, soll heißen, dass dasselbe Bild in jeder Epochen verändert wurde, um so eine größere Vielfalt an Trainingsbildern zu generieren.

[4] `https://de.wikipedia.org/wiki/Gardiner-Liste`

2.3. Bibliotheken

2.3.1. Theano

Theano[3][2][5] ist ein Compiler für mathematische Ausdrücke in Python. Theano optimiert die in Python geschrieben Ausdrücke automatisch und setzt sie in maschinennahen Quellcode entweder für die CPU oder GPU um und ermöglicht so ein schnelles und effizientes ausführen von komplexen Operationen.

Die in den beiden nächsten Abschnitten beschrieben Bibliotheken Pylearn2 (vgl. Abschnitt 2.3.2) und Keras(vgl. Abschnitt 2.3.3) bauen jeweils auf den Funktionalitäten von Theano auf.

2.3.2. Pylearn 2

Die erste Bibliothek, die für die Experimente angeschaut wurde, ist Pylearn2[6][8]. Dabei handelt es sich um eine Forschungsbibliothek für maschinelles Lernen, dass heißt Pylearn2 ist nicht nur eine Sammlung von Algorithmen des maschinellen Lernens, sondern es ist für Flexibilität und Erweiterbarkeit ausgelegt, um Forschungsprojekte zu erleichtern, die entweder neue oder ungewöhnliche Anwendungsfälle haben.

Pylearn2 baut großteils auf Theano (vgl. Abschnitt 2.3.1) auf, das es unter anderem ermöglicht Berechnungen auf der Grafikkarte (GPU) auszuführen, die dadurch um Einiges schneller sind.

Die Konfiguration eines Experiments erfolgt dabei über ein YAML-File, in dem die Datenquelle, die Layer des Neuronalen Netzes, sowie der Lernalgorithmus definiert werden. Eine Beispiel Konfiguration eines CNNs ist Anhang A zu finden.

Für etwas umfangreichere Netze, bei denen Änderungen am Code notwendig sind, ist Pylearn2 sehr umständlich, da es aufgrund der umfangreichen Funktionalität und Abstraktion sehr schwer ist, sich in die Bibliothek einzuarbeiten. Zur Zeit wird Pylearn2 zudem nicht weiter entwickelt.

2.3.3. Keras

Keras[4][7] ist eine minimalistische, hoch modulare Bibliothek für neuronale Netze. Es ist in Python geschrieben und benutzt im Hintergrund entweder Theano (vgl. Abschnitt 2.3.1) oder TensorFlow[8].

Während der Entstehung dieser Arbeit wurde die Bibliothek konstant weiterentwickelt. Die größte Änderung kam dabei mit Version 1.0.0. Ab Version 1.0.0 ver-

[5] `http://deeplearning.net/software/theano/`
[6] `http://deeplearning.net/software/pylearn2/`
[7] `http://keras.io/`
[8] `https://www.tensorflow.org/`

fügt Keras zusätzlich zur auch schon vorher bestehenden API, bei der ein Netz mit `model = Sequential()` erstellt wird und dann mittels `model.add(Dense(32))` die einzelnen Layer hinzugefügt werden, über eine funktionale API. Funktionale API heißt in dem Fall, dass jedes Layer als Funktion aufgerufen werden kann, Bsp. `y = Dense(32)(x)`. Dies ermöglicht es, dass jedes Layer als eine Funktion deklariert werden kann und so das gesamte Netz als Verkettung von Funktionen definiert werden kann, was der mathematischen Definition näher kommt.

Neben der Einführung der funktional API mit Version 1.0.0 wurde das `Autoencoder`-Layer, mit dem vorher Autoencoder erzeugt werden mussten, entfernt. So müssen mit Version 1.0.0 Autoencoder ebenfalls über die funktionale API erzeugt werden.

Zusätzlich verfügt Keras über einen `ImageDataGenerator` mit dem Bilder automatisch augmentiert werden können. So können zum Beispiel die Bilder pro Epoche zufällig rotiert, geschert oder normalisiert werden. Dies dient dazu, dass selbst bei wenig Trainingsdaten genug Varianz im Training vorhanden ist und kein Overfitting statt findet.

Für die ersten Experimente wurde der `ImageDataGenerator` erweitert, so dass ein Bild zufällig in einem größeren Bild positioniert werden kann, was in jeder Epoche erneut geschieht. Genau heißt das, dass zum Beispiel ein Bild mit einer Hieroglyphe mit einer Größe von 24×24 Pixel zufällig in einem Bild mit der Größe 48×48 Pixel positioniert wird, in Abbildung 3.2 ist dies beispielhaft für eine Hieroglyphe dargestellt. Hierbei ist der größere Wert die Eingabegröße des Netzes. So kann es sein, dass während des Trainings in der ersten Epoche die Hieroglyphe rechts oben ist, in der nächsten links unten und in der Dritten zentriert. Ziel ist, dass das Netz Positionsunabhängigkeit lernt. Mit Keras Version 1.0.4 hat sich der `ImageDataGenerator` soweit geändert, dass die zufällige Positionierung nicht mehr möglich ist. Entsprechend wurden die Experimente im Abschnitt 3.6 ohne die zufällige Positionierung trainiert.

Die Experimente, die im Laufe dieser Arbeit vorgestellt werden, wurden alle mit Keras erstellt. Aufgrund der sich ändernden Version sind die Netze der unterschiedlichen Experimente mit verschiedenen Version erstellt worden. Die Experimente des Abschnitts 3.4 wurden mit Version 0.3.3 erstellt. Die Experimente des folgenden Abschnitts 3.5 wurden mit Version 1.0.3 erstellt und die Experimente des letzten Abschnitts 3.6 wurden mit den Versionen 1.0.7, 1.0.8 und 1.1.0 erstellt.

Im Anhang D ist beispielhaft eine JSON-Konfiguration des besten Modells (21. Experiment, CNN 6465) zu finden.

3. Experimente

Das folgende Kapitel behandelt die durchgeführten Experimente. Begonnen wird mit einer detaillierten Beschreibung zum Aufbau der neuronalen Netze, gefolgt von einer Beschreibung der erfolgten Experimente, unterteilt anhand der verschiedenen Trainingsmethoden.

3.1. Aufbau der neuronalen Netze

In der Literatur sind einige unterschiedliche Ansätze bezüglich des Aufbaues, der Layerstruktur von CNNs zu finden. Die meisten folgen dabei einem gemeinsamen grundlegenden Ansatz, der in Gleichung 3.1 aufgeführt ist.

$$\text{EINGABE} \rightarrow [[\text{CONV}]^*N \rightarrow \text{POOL?}]^*M \rightarrow [\text{FC}]^*K- > \text{FC} \tag{3.1}$$

Auf die Eingabe eines Bildes folgt ein oder mehrere Convolutional-Layer, gegebenenfalls gefolgt von einem Pooling-Layer. Ein solcher Block kann mehrmals vorkommen. Zur Klassifizierung folgen mehrere Dense-Layer (auch Fully-Connected genannt) (ein Layer bei dem alle Eingänge mit allen Ausgängen verbunden sind). Dabei sind N, M und K in der Regel $0 \leq N \leq 3$, $M \geq 0$ und $0 \leq K < 3$.[9]

3.1.1. Convolutional-Layer

Die Größe der Filter der Convolutional-Layer ist in der Regel 3×3. Dies hat mehrere Gründe. Der offensichtliche Vorteil von größeren Filtern ist, dass mehr Werte betrachtet werden und ein Convolutional-Layer so einen größeren Kontext betrachten kann. Dadurch können bessere Ergebnisse erzielt werden. Der Nachteil von großen Filtern ist allerdings, dass diese meist schwerer zu trainieren sind. Das liegt vor Allem daran, dass die Anzahl der Parameter wesentlich größer ist. Ein Convolutional-Layer mit einem 7×7 Filter, hat bei gleicher Anzahl von Ein- und Ausgabefiltern c, $c(7 * 7 * c) = 49c^2$ Parameter, ein Layer mit einem 5×5 Filter $c(5 * 5 * c) = 25c^2$ Parameter und ein Layer mit einem 3×3 Filter $c(3 * 3 * c) = 9c^2$ Parameter.

Werden zwei Convolutional-Layer hintereinander geschaltet mit jeweils einem 3×3 Filter, so betrachtet das zweite Layer hinsichtlich der Eingabe des ersten Layer den

[9] `http://cs231n.github.io/convolutional-networks/`

gleichen Kontext wie ein Convolutional-Layer mit einem 5×5 Filter. Dabei haben die zwei Convolutional-Layer nur $2(c(3 * 3 * c)) = 18c^2$ Parameter statt $25c^2$. Drei Convolutional-Layer mit jeweils 3×3 Filter haben $3(c(3 * 3 * c)) = 27c^2$ Parameter, dies sind knapp halb so viele wie ein Layer mit einem 7×7 Filter ($49c^2$).

Ein weiterer Vorteil von mehreren Convolutional-Layer ist, dass jedes eine Aktivierungsfunktion besitzt und in Abhängigkeit von der Aktivierungsfunktion hier nichtlineare Features gelernt werden können. Allerdings haben mehrere Convolutional-Layer einen höheren Speicherverbrauch als ein einzelnes Layer mit einem großen Filter.

Auch haben Simonyan und Zisserman[27] mit ihrem Beitrag zu ImageNet 2014 gezeigt, dass man durch die kleineren Filter tiefere Netze erstellen kann und damit sehr gute Ergebnisse erzielt. Auf ihre Netze wird im Allgemeinen als VGG Net verwiesen.

Diesem Ansatz folgend haben in allen folgenden Experimente die Convolutional-Layer einen 3×3 Filter und es sind immer mehrere Convolutional-Layer hintereinander geschaltet.

3.1.2. Pooling-Layer

Das am häufigsten verwendete Pooling-Layer ist das Max-Pooling (vgl. Abschnitt 2.1.6), mit einen 2×2 Filter und einer *stride* von 2×2.

Werden die Arbeitsweise von Convolutional-Layer und Pooling-Layer näher betrachtet und miteinander verglichen, so sind die Unterschiede zwischen beiden sehr gering. Beide Layer haben einen Filter, über den ein Teil der Eingaben betrachtet und die Ausgaben berechnet werden.

Hauptsächlich unterscheiden sie sich in den Funktionsweisen der Filter, der Aktivierungsfunktion des Convolutional-Layers und der größeren *stride* des Pooling-Layers. Die *stride* ist lediglich ein theoretischer Unterschied, sie kann genauso für Convolutional-Layer wie Pooling-Layer definiert werden, was in der Praxis selten gemacht wird. Zum Vergleich der Funktionsweise von Pooling-Layer und Convolutional-Layer sind die Formeln gegeben, für das Pooling-Layer Gleichung 3.2 und für das Convolutional-Layer Gleichung 3.3.

$$s_{i,j,u}(f) = \left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} |f_{g(h,w,i,j,u)}|^p \right)^{1/p} \tag{3.2}$$

$$c_{i,j,o}(f) = \sigma \left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{u=1}^{N} \theta_{h,w,u,o} \cdot f_{g(h,w,i,j,u)} \right) \tag{3.3}$$

Dabei ist $f = W \times H \times N$ die Eingabe mit Breite W, Höhe H und Filter N, k die Filter- oder Poolgröße, r die Schrittgröße, $g(h, w, i, j, u) = (ri + h, rj + w, u)$ als Abbildungsfunktion für die Indizes in Abhängigkeit von der Schrittgröße, p die Ordnung der p-norm mit $p \to \infty$ = Max-Pooling, σ die Aktivierungsfunktion, θ die Convolutional Filter und $o \in [1, M]$ die Anzahl der Ausgabefilter.

Ein Pooling-Layer kann beschrieben werden, als würde es eine feature-weise Convolution durchführen, mit $\theta_{h,w,u,o} = 1$ wenn $u = 0$, sonst $\theta_{h,w,u,o} = 0$, und statt der Aktivierungsfunktion wird die p-norm verwendet um die letztendliche Ausgabe zu berechnen.

Die daraus resultierende Frage, warum es eines Pooling-Layers überhaupt bedarf und dies nicht von einem Convolutional-Layer geleistet werden kann, kann durch folgende drei Punkte ansatzweise beantwortet werden:

- durch die p-norm werden die Repräsentationen in einem CNN invarianter
- durch die räumliche Dimensionsreduktion des Poolings ermöglicht es in tieferen Layer größere Teile der Eingabe abzudecken
- die feature-weise Art der Pooling-Operation, im Gegensatz zu einem Convolutional-Layer, bei dem die Features gemischt werden, könnte die Optimierung erleichtern

Ausgehend davon, dass nur der zweite Punkt wichtig ist für eine gute Performanz eines CNNs, können die Pooling-Layer durch Convolutional-Layer mit einer *stride* ≥ 2 ersetzt werden.[28] Darauf aufbauend wurden die Netze der Experimente im Abschnitt 3.4 entweder mit einem Max-Pooling-Layer oder mit einem Convolutional-Layer mit einer *stride* von 2×2 trainiert.

3.2. Objekterkennung

Die Objekterkennung ist ein Teilgebiet des maschinellen Sehens und beschäftigt sich mit dem Auffinden eines bekannten Objektes in einem Objektraum, z.B. dem Auffinden eines Objektes in einem Bild. Bei dieser Arbeit sind die zu findenden/erkennenden Objekte ägyptische Hieroglyphen. Dabei sind in den Bildern mehrere Hieroglyphen vorhanden. Die Bilder wurden dabei aus mehreren zufällig ausgewählten Hieroglyphen erzeugt.

Die Bilder wurden wie folgt erzeugt: Die Höhe eines Bildes ist fest vorgegeben und entspricht der Eingabehöhe des CNNs. Die Breite des Bildes wird nach folgender Formel berechnet: $w = i_w * N - \frac{i_w - w_h}{2} * N$, mit i_w der Breite der Eingabe des CNNs, w_h der maximalen Breite einer Hieroglyphe und N die Anzahl an Hieroglyphen im Bild. Für ein CNN mit 48×48 als Eingabegröße, einer maximalen Hieroglyphen

Breite von 24 Pixel und zehn Hieroglyphen, berechnet sich die Breite wie folgt $48 * 10 - \frac{48-24}{2} * 10 = 360$.

Anschließend wird ein `offset` berechnet mit $\phi = \frac{w - w_h * N}{N}$. Danach wird anhand einer Normalverteilung mit $\mathcal{N}(\phi, (\frac{\phi}{2})^2)$ N Werte gezogen, sodass $\sum_N \mathcal{N}(\phi, (\frac{\phi}{2})^2) \leq w - w_h * N$. Anhand dieser Werte wird der Abstand zwischen zwei Hieroglyphen gewählt, wobei der erste Werte halbiert wird, da er den Abstand zwischen Rand und erster Hieroglyphe und letzter Hieroglyphe und Rand bestimmt. In Abbildung 3.1 sind beispielhaft fünf der generierten Sequenzen dargestellt.

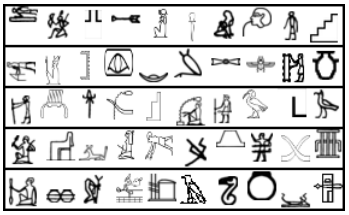


Abbildung 3.1.: Beispiel von fünf Sequenzen für die Objekterkennung, mit jeweils zehn Hieroglyphen.

Da die CNNs eine feste Eingabegröße haben, werden die Bilder für die eigentliche Erkennung wieder in einzelne Bilder unterteilt, mit der Größe, die die Eingabe des CNNs hat. Dabei wird für die Breite der Abstand vom linken Rand wie folgt berechnet: $a_l = \frac{w}{N}$, das i-te Bild beginnt damit bei $i * a_l$ und geht bis $i * a_l + i_w$. Die Bilder überschneiden sich damit um $u = \frac{i_w - a_l}{i_w}$. Für das Beispiel oben ergibt sich damit $a_l = \frac{360}{10} = 36$ und eine Überschneidung von $u = \frac{48-36}{48} = \frac{1}{4}$ bzw. 25%.

Insgesamt wurden für die Objekterkennung 10000 Bilder mit jeweils zehn Hieroglyphen erstellt. Für die Experimente, bei denen die CNNs eine Eingabegröße von 48×48 haben, haben die Bilder eine Größe von 360×48 Pixel, für die CNNs mit einer Eingabegröße von 32×32 haben die Bilder eine Größe von 280×32 Pixel. Wie gut die CNNs die Sequenz erkennen wird mit dem Wert der Objekterkennungsgenauigkeit beschrieben, d.h. wenn alle Hieroglyphen richtig erkannt wurde ist die Objekterkennungsgenauigkeit 1.

3.3. System

Das Training erfolgte auf dem High Performance Computing (HPC) Cluster des Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH)[10] der Technischen Universität Dresden. Dabei wurde jeweils auf einer nVIDIA K80 Grafikkarte gerechnet. Das Rechnen auf der Grafikkarte war notwendig, da das Trainieren auf der CPUs um ein vielfaches langsamer wäre.

[10] `https://tu-dresden.de/zih/`

Es ist zu beachten, dass die Jobs auf dem Cluster auf eine maximal Zeit von sieben Tagen ($168h$) beschränkt sind.

3.4. Vortrainieren mittels Autoencoder

Im folgenden Abschnitt werden die Experimente und Ergebnisse beschrieben, bei denen ein Autoencoder vortrainiert wurde.

Insgesamt sind hier acht Experimente erfolgt, die alle den ersten Datensatz (vgl. Abschnitt 2.2.1) benutzt haben. In jedem Experiment wurde zunächst ein Autoencoder auf allen 13980 Bilder trainiert. Daran anschließend wurden jeweils vier CNNs trainiert. Zwei CNNs mit 3210 Bildern (1070 Klassen) und zwei CNNs mit 13980 Bildern (6671 Klassen).

In einigen ersten Tests, die hier nicht weiter aufgeführt sind, hatte sich bereits abgezeichnet, dass die CNNs nicht trainierbar sind, d.h. nach einigen wenigen Epochen in denen sich der Loss Wert verbessert hat, ist er auf einen meist doppelt so großen Wert gesprungen und hat sich nicht mehr verändert. Aufgrund dieses Verhaltens wurde zunächst ein Autoencoder trainiert. Der Autoencoder wurde dabei blockweise trainiert, d.h. das zunächst nur ein Block, bestehend aus mehreren Convolutional-Layer gefolgt von einem Layer das eine Dimensionsreduktion (Pooling-Layer oder Convolutional-Layer mit *stride*) durchführt, trainiert. Im weitern Verlauf wurde jeweils ein weiterer Block hinzugefügt, wobei die Gewichte des ersten behalten wurden. Die CNNs wurden anschließend trainiert, wobei als Initialgewichte die Gewichte des Encoders des Autoencoders genommen wurden.

Dieses Trainingsschema basiert unter Anderem auf den Arbeiten von Zeiler et al.[36] und von Szegedy et al.[30]. Zeiler et al. haben eine Visualisierungsstrategie entwickelt, mit denen beliebige Layer visualisiert werden können. Dabei werden die Ausgaben beliebiger Layer wieder zurück in die Eingabeschicht projiziert, dieser Ansatz arbeitet ähnlich wie ein Autoencoder. Szegedy et al.[30] stellen eine neue Netzarchitektur vor, die sie GoogLeNet nennen. GoogLeNet besteht aus sogenannten Inception-Blöcken, diese Blöcke haben unterschiedliche Convolutional-Layer neben einander und fassen deren Ausgaben am Ende des Blockes wieder zusammen. Damit GoogLeNet trotz der großen Tiefe, insgesamt 22 Layer, trainierbar ist, sind hier drei Softmax-Klassifikatoren an unterschiedlichen Tiefen.

Die Eingabe in den folgen Experimenten waren Graustufenbilder mit einer Größe von 48×48 Pixel. Für das Training wurden die Bilder mit den Hieroglyphen (24×24 Pixel) zufällig in dem größeren Eingabebild platziert, dies ist in Abbildung 3.2 beispielhaft für eine Hieroglyphe dargestellt. Für die anschließenden Tests wurden die Hieroglyphen zentriert. Die Positionierung wurde über eine Erweiterung des `ImageDataGenerator` von Keras realisiert. Beim dritten und vierten Experiment wurden die Bilder zusätzlich, über den `ImageDataGenerator` normalisiert, dabei

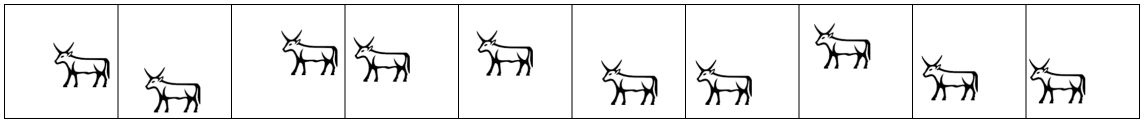


Abbildung 3.2.: Beispielhafte Darstellung der zufälligen Positionierung einer Hieroglyphe für die Eingabe in die neuronalen Netze.

wurde der Mittelwert auf 0 verschoben (featurewise_center) und anschließend alle Werte durch die Standardabweichung geteilt (featurewise_std_normalization).

3.4.1. Autoencoder

Im folgenden Abschnitt werden die Experimente mit dem Autoencoder näher beschrieben, beginnend mit den Grundlagen und dem Aufbau gefolgt von den Ergebnissen und Bewertungen.

Wie bereits erwähnt wurde der Autoencoder nicht im Ganzen trainiert, sondern blockweise. Insgesamt sind es drei Blöcke. Dabei besteht ein Block aus drei Convolutional-Layer mit einer Filtergröße von 3×3 gefolgt von einem Layer zur Dimensionsreduktion, dies ist entweder ein Max-Pooling-Layer mit einer Filtergröße von 2×2 und einer *stride* von 2×2 oder ein Convolutional-Layer mit einer Filtergröße von 3×3 und einer *stride* von 2×2.

Die Größen und Anzahl an Layer wurden dabei ausgehend von den Eingabebildern (48×48 Pixel) und der Umkehrbarkeit der Dimensionsreduktion gewählt, dafür ist wichtig zu beachten, dass Matrizen nur positive ganzzahlige Größen haben können. Für den Encoder mit allen drei Blöcken heißt das $48/2 = 24/2 = 12/2 = 6$ und für den Decoder entsprechend $6 * 2 = 12 * 2 = 24 * 2 = 48$.

Alle hier im Autoencoder verwendeten Convolutional-Layer, abgesehen von den Layer, die eine *stride* von 2×2 haben, sind Ein- und Ausgabe-Dimensionen gleich, d. h. auch die Anzahl der Filter wurde nicht geändert. Die im Encoder verwendeten Max-Pooling-Layer haben als Umkehrung im Decoder ein `UpSampling`-Layer, das die Zeilen und Spalten wiederholt, hier verdoppelt. Für die Umkehrung der Convolutional-Layer mit einer *stride* wird ein `ZeroPadding`-Layer gefolgt von einem Convolutional-Layer benutzt. Das `ZeroPadding`-Layer dient dabei der Umkehrung der Dimensionsreduktion im Encoder und das anschließende Convolutional-Layer der zur Umkehrung der Convolution.

Die Netzstruktur für die Autoencoder der Experimente eins bis vier ist in Tabelle 3.1 aufgeführt. Die Netzstruktur für die Experimente fünf bis acht unterscheidet sich insoweit, dass jeweils vor das erste Layer ein `GaussianNoise`-Layer gesetzt wurde. Dabei handelt es sich um ein Layer, das Rauschen über die Eingabedaten legt. Das

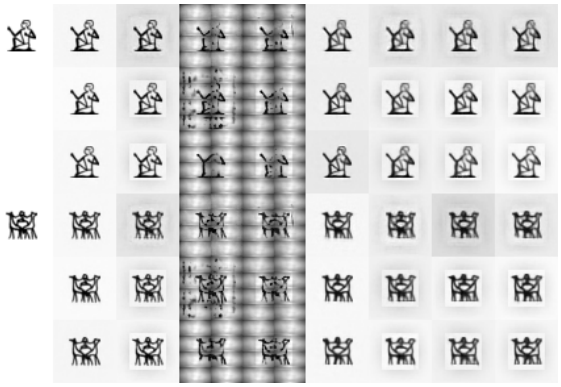


Abbildung 3.3.: Rekonstruktionen einer Hieroglyphe mittels eines Autoencoders in den verschiedenen Experiment und Blöcken.

Rauschen wird anhand einer Gauß-Verteilung ($\mu = 0, \sigma = 0,3$) ermittelt. Das fünfte Experiment hat die selbe Struktur wie das erste Experiment und die Experimente sechs bis acht die gleiche Struktur wie das zweite Experiment.

Das `Flatten`- und `Reshape`-Layer transformiert jeweils die Dimensionalität der Matrizen, es werden aber alle Werte beibehalten. Dies ist aufgrund der verwendeten `Autoencoder`-Klasse nötig. Durch das `Flatten`-Layer wird die Dimensionalität von $(256, 6, 6)$ auf $(9216,)$ transformiert. Anschließend wird sie durch das `Reshape`-Layer wieder zurück auf $(256, 6, 6)$ transformiert.

Die Convolutional-Layer haben bei allen Experimenten im ersten Block 64 Filter, im zweiten Block 128 und im dritten Block 256. Als Optimierungsalgorithmus wurde jeweils AdaMax (vgl. Abschnitt 2.1.5) benutzt. Die Aktivierungsfunktionen war jeweils SReLU (vgl. Abschnitt 2.1.3) und als Kostenfunktion msle (vgl. Abschnitt 2.1.4).

In Tabelle 3.2 sind die Losswerte für Training und Test aufgeführt zusammen mit den Trainingszeiten und der Anzahl der Trainingsepochen.

In Abbildung 3.3 ist beispielhaft für eine Hieroglyphe die Rekonstruktion der Autoencoder dargestellt, dabei entsprechen die Zeilen den Blöcken und die Spalten den Experimenten.

Sowohl anhand der Losswerte, als auch an den rekonstruierten Bildern, lässt sich erkennen, dass die Autoencoder in allen Experimenten gut gelernt haben. Dabei ist auch zu erkennen, dass die Normalisierung der Daten beim dritten und vierten Experiment einen negativen Einfluss haben, die Losswerte sind hier um mehr als das Hundertfache höher. Die rekonstruierten Bilder zeigen auch, dass der Autoencoder hier die Normalisierung mit gelernt hat (dritte und vierte Spalte in Abbildung 3.3). Der Einfluss des `GaussianNoise`-Layer hingegen ist wesentlich geringer und trotz

		1./3. Experiment	2./4. Experiment
Encoder	1. Block	conv 3×3 conv 3×3 conv 3×3 pool 2×2 (*stride* 2×2) dropout $p = 0,25$	conv 3×3 conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$
	2. Block	conv 3×3 conv 3×3 conv 3×3 pool 2×2 (*stride* 2×2) dropout $p = 0,25$	conv 3×3 conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$
	3. Block	conv 3×3 conv 3×3 conv 3×3 pool 2×2 (*stride* 2×2) dropout $p = 0,25$	conv 3×3 conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$
		flatten	flatten
Decoder		reshape	reshape
	3. Block	dropout $p = 0,25$ upsample 2×2 conv 3×3 conv 3×3 conv 3×3	dropout $p = 0,25$ zero padding 3×3 conv 3×3 conv 3×3 conv 3×3 conv 3×3
	2. Block	dropout $p = 0,25$ upsample 2×2 conv 3×3 conv 3×3 conv 3×3	dropout $p = 0,25$ zero padding 6×6 conv 3×3 conv 3×3 conv 3×3 conv 3×3
	1. Block	dropout $p = 0,25$ upsample 2×2 conv 3×3 conv 3×3 conv 3×3	dropout $p = 0,25$ zero padding 12×12 conv 3×3 conv 3×3 conv 3×3 conv 3×3

Tabelle 3.1.: Netzstrukturen der Autoencoder der ersten vier Experimente.

Experiment	Block	Epochen	Trainings-Loss	Test-Loss	Trainingszeit
1.	1.	40	0,00015	0,00005	1,48h
	2.	80	0,00019	0,00011	4,53h
	3.	120	0,00042	0,00023	8,73h
2.	1.	40	0,00395	0,00044	1,21h
	2.	80	0,00364	0,00035	4,02h
	3.	120	0,00370	0,00046	8,44h
3.	1.	40	0,00009	0,03462	1,48h
	2.	80	0,00011	0,06994	4,73h
	3.	120	0,00037	0,04882	8,74h
4.	1.	40	0,00615	0,05247	1,20h
	2.	80	0,00613	0,05357	3,95h
	3.	120	0,00614	0,05398	8,31h
5.	1.	40	0,00156	0,00066	1,50h
	2.	80	0,00134	0,00058	4,72h
	3.	120	0,00148	0,00078	9,16h
6.	1.	40	0,00513	0,00115	1,21h
	2.	80	0,00464	0,00109	4,06h
	3.	120	0,00459	0,00119	8,44h
7.	1.	50	0,00505	0,00108	1,51h
	2.	100	0,00453	0,00105	5,07h
	3.	150	0,00456	0,00116	10,54h
8.	1.	60	0,00503	0,00102	1,78h
	2.	120	0,00452	0,00093	5,99h
	3.	180	0,00445	0,00097	12,45h

Tabelle 3.2.: Ergebnisse des Trainings der Autoencoder der Experiment eins bis acht.

des Rauschens an der Eingabe des Autoencoders sind die rekonstruierten Bilder nahezu identisch mit dem Eingabebild.

Das Steigern der Trainingsepochen bei den Experimenten sechs bis acht verbessert die Losswerte jeweils, allerdings steigt hier auch die Trainingszeit erheblich, von insgesamt $13,71h$ $(1,21h + 4,06h + 8,44h)$ beim sechsten Experiment über $17,12h$ $(1,51h+5,07h+10,54h)$ beim Siebten bis zu $20,22h$ $(1,78h+5,99h+12,45h)$ beim Achten.

Die Experimente, bei denen ein Max-Pooling-Layer zur Dimensionsreduktion verwendet wurde, haben wesentlich längerer Trainingszeiten als die Experimente, die hier ein Convolutional-Layer benutzt haben. Dies widerspricht der Erwartung, da die Convolutional-Layer wesentlich mehr Parameter haben als die Max-Pooling-Layer. Es wird vermutet, dass dies mit einer Unterschiedlichen Implementierung der Funktionalität im Backend zusammenhängt. Im Gegensatz dazu haben die Autoencoder mit den Max-Pooling-Layer geringere Losswerte bei gleicher Epochenanzahl, dies war aber zu erwarten, da die Autoencoder wesentlich weniger Parameter haben.

Insgesamt wurden die niedrigsten Trainingslosswerte im ersten und zweiten Experiment erreicht. Im ersten Experiment, der Autoencoder mit Max-Pooling-Layer, ist der Loss bei $0,00023$ und im zweiten Experiment, für den Autoencoder mit nur Convolutional-Layer, ist der Loss doppelt so groß mit $0,00046$. Dafür ist die Trainingszeit um etwa $1h$ geringer, $13,67h$ $(1,21h + 4,02h + 8,44h)$ im Vergleich zu $14,74h$ $(1,48h + 4,53h + 8,73h)$.

Das `GaussianNoise`-Layer, das ab dem fünften Experiment hinzugefügt wurde, hat im Vergleich zum ersten und zweiten Experiment die Trainingslosswerte etwa verdreifacht. So ist im fünften Experiment der Loss bei $0,00078$ und im sechsten bei $0,00119$. Vor allem fürs fünfte Experiment hat sich hier auch die Trainingszeit um etwa $1h$ verschlechtert auf $15,38h$ $(1,5h + 4,72h + 9,16h)$, im sechsten Experiment ist sie nahezu unverändert bei $13,71h$ $(1,21h + 4,06h + 8,44h)$.

3.4.2. CNN

In jedem der acht Experimente wurden jeweils vier unterschiedliche CNNs trainiert. Diese rekonstruieren die Bilder nicht mehr, sondern klassifizieren sie. Zwei der CNNs wurden mit dem kleineren Datensatz mit 3210 Bildern (1070 Klassen) und zwei mit allen 13980 Bildern (6671 Klassen) trainiert. Bei jedem Datensatz wurde ein CNN mit zwei Dense-Layer nach dem letzten Convolutional-Layer und ein CNN ohne die zwei Dense-Layer trainiert. Die CNNs eines Experimentes werden im Folgenden anhand der Anzahl an Klassen unterschieden, mit dem Zusatz *Dense* wenn das CNN über die zwei zusätzlichen Dense-Layer verfügt. Die Netzstruktur ist in Tabelle 3.3 einmal schematisch dargestellt.

Die CNNs sind gleich aufgebaut wie der Encoder des Autoencoders im jeweiligen

<table>
<tr><td colspan="4">conv 3×3
conv 3×3
conv 3×3
pool 2×2 / conv 3×3 (*stride* 2×2)
dropout $p = 0,25$</td></tr>
<tr><td colspan="4">conv 3×3
conv 3×3
conv 3×3
pool 2×2 / conv 3×3 (*stride* 2×2)
dropout $p = 0,25$</td></tr>
<tr><td colspan="4">conv 3×3
conv 3×3
conv 3×3
pool 2×2 / conv 3×3 (*stride* 2×2)
dropout $p = 0,25$</td></tr>
<tr><td colspan="4">flatten</td></tr>
<tr><td></td><td>dense 4096
dense 4096</td><td></td><td>dense 4096
dense 4096</td></tr>
<tr><td>dense 1070</td><td>dense 1070</td><td>dense 6671</td><td>dense 6671</td></tr>
<tr><td colspan="4">softmax</td></tr>
</table>

Tabelle 3.3.: Schematische Darstellung der Netzstruktur der CNNs der ersten acht Experimente.

Experiment mit nachfolgenden Dense-Layer für die Klassifikation und haben dementsprechend die Gewichte des Encoders als Initialgewichte bekommen (jeweils für die Layer, die in Beiden vorkommen). Dementsprechend haben die CNNs in den Experimenten vier bis acht als Erstes ein `GaussianNoise`-Layer. Insgesamt sind für die ersten acht Experimente 32 CNNs trainiert worden.

Vor dem eigentlichen Training der CNNs wurden die Gewichte aus den Encodern kopiert und als Initialgewichte für die CNNs genutzt. Dies ist natürlich nur für die Layer möglich, die auch im Encoder vorkommen, also nur die Convolutional- und Max-Pooling-Layer, das heißt also dass die Features, die im Autoencoder gelernt wurden, im CNN schon vorhanden sind. Hier muss nur noch der Klassifikator trainiert werden, wodurch das Training insgesamt schneller ist.

Die Convolutional-Layer haben wie beim Autoencoder im ersten Block 64 Filter, im zweiten Block 128 und im dritten Block 256. Als Optimierungsalgorithmus wurde jeweils AdaMax (vgl. Abschnitt 2.1.5) benutzt. Die Aktivierungsfunktionen war jeweils SReLU (vgl. Abschnitt 2.1.3) und als Kostenfunktion categorical crossentropy (vgl. Abschnitt 2.1.4).

In Tabelle 3.4 sind die Anzahl der Epochen des Trainings sowie Trainingswerte für Loss, Genauigkeit und Laufzeit für die CNNs der acht Experimente aufgeführt. In Tabelle 3.5 sind entsprechend die Testwerte für Loss und Genauigkeit, sowie die Genauigkeit der Objekterkennung aufgeführt.

Wie schon bei den Autoencodern, sind auch bei den CNNs die Netze mit den Max-Pooling-Layer im Vergleich zu denen mit Convolutional-Layer langsamer im Training, haben aber dafür niedrigere Loss- und höhere Genauigkeitswerte. Das fünfte Experiment, verglichen mit den Experimenten sechs bis acht veranschaulicht diesen Umstand besonders deutlich, dass die Netze mit den Convolutional-Layer, Experiment sechs bis acht, mehr Trainingsepochen brauchen, als die CNNs im fünften Experiment mit Max-Pooling-Layer. Die größere Anzahl an Trainingsepochen heißt dabei nicht unbedingt mehr Zeit, die CNNs 1070 und Dense 1070 im fünften Experiment brauchten $1,30h$ und $1,36h$ bei 120 Epochen, im achten Experiment mit 60 Epochen mehr waren es $1,34h$ und $1,39h$. Dies ist ein Unterschied von $2,4min$ und $1,8min$ bei 60 Epochen mehr. Für die CNNs mit 6671 Klassen sieht dies ähnlich aus, $5,63h$ und $5,80h$ beim fünften Experiment, $4,18h$ und $4,20h$ beim Sechsten, $5,22h$ und $5,24h$ im Siebten und schließlich $6,18h$ und $6,20h$ im achten Experiment, macht einen Unterschied von $33min$ und $24min$.

Wie auch schon bei den Autoencoder haben die CNNs der Experimente drei und vier, bei denen die Bilder normalisiert wurden, sehr schlechte Test- und Erkennungswerte. Die Werte im Training sind nahezu gleich wie die in den vorherigen zwei Experimenten, was bedeutet, dass die Veränderungen der Normalisierung keinen Einfluss auf das Training haben, aber wenn die Normalisierung nicht mehr stattfindet, wie in den Tests und bei der Erkennung, zeigt sich, das die CNNs nicht gelernt haben, die Hieroglyphen von den durch die Normalisierung eingebrachten Änderungen zu unterscheiden.

Die beste Trainingsgenauigkeit $0,97432$ und die beste Testgenauigkeit $0,98991$ wurde im ersten Experiment beim CNN mit 6671 Klassen ohne zusätzliche Dense-Layer erreicht. Dieses CNN hat allerdings nur eine Objekterkennungsgenauigkeit von $0,60250$, schon im gleichen Experiment hat das CNN mit 6671 Klassen und den zusätzlichen Dense-Layer eine höhere Objekterkennungsgenauigkeit $0,66560$. Die beste Objekterkennungsgenauigkeit $0,72870$ wurde im achten Experiment beim CNN mit 6671 Klassen ohne zusätzliche Dense-Layer erreicht.

Die besten Werte für das CNN mit 1070 Klassen ohne zusätzliche Dense-Layer wurden im ersten Experiment erreicht mit einer Trainingsgenauigkeit $0,96854$ und einer Testgenauigkeit $0,98567$. Die beste Objekterkennungsgenauigkeit mit $0,65070$ (Trainingsgenauigkeit $0,90000$, Testgenauigkeit $0,96573$) wurde ebenfalls erst im achten Experiment erreicht.

Für das CNN mit 1070 Klassen und den zwei zusätzlichen Dense-Layer wurde im dritten Experiment die beste Trainingsgenauigkeit $0,94112$ erreicht, aber durch

Experiment	CNN	Epochen	Loss	Genauigkeit	Zeit
1.	1070	120	0,12137	0,96854	1,29h
	Dense + 1070	120	0,21509	0,93271	1,35h
	6671	120	0,09700	0,97432	5,76h
	Dense + 6671	120	0,12687	0,96245	5,97h
2.	1070	120	0,22681	0,92991	0,90h
	Dense + 1070	120	0,31321	0,90592	0,94h
	6671	120	0,15745	0,95665	4,18h
	Dense + 6671	120	0,21554	0,93505	4,21h
3.	1070	120	0,10624	0,96604	1,29h
	Dense + 1070	120	0,18799	0,94112	1,37h
	6671	120	0,09832	0,97332	5,82h
	Dense + 6671	120	0,11995	0,96245	6,02h
4.	1070	120	0,25435	0,92368	0,89h
	Dense + 1070	120	0,26898	0,91620	0,93h
	6671	120	0,15522	0,95751	4,14h
	Dense + 6671	120	0,16394	0,95021	4,15h
5.	1070	120	0,27984	0,91464	1,30h
	Dense + 1070	120	0,33167	0,89408	1,36h
	6671	120	0,38903	0,88412	5,63h
	Dense + 6671	120	0,40210	0,87668	5,80h
6.	1070	120	0,44852	0,86573	0,94h
	Dense + 1070	120	0,46529	0,85265	0,94h
	6671	120	0,57033	0,83369	4,18h
	Dense + 6671	120	0,47617	0,85601	4,20h
7.	1070	150	0,35208	0,89564	1,13h
	Dense + 1070	150	0,41445	0,87040	1,17h
	6671	150	0,40869	0,88247	5,22h
	Dense + 6671	150	0,43195	0,86645	5,24h
8.	1070	180	0,33056	0,90000	1,34h
	Dense + 1070	180	0,36013	0,88847	1,39h
	6671	180	0,41698	0,87568	6,18h
	Dense + 6671	180	0,39294	0,87775	6,20h

Tabelle 3.4.: Ergebnisse des Trainings der CNNs der Experiment eins bis acht.

Experiment	CNN	Loss	Genauigkeit	Objekterkennung
1.	1070	0,04523	0,98567	0,56860
	Dense + 1070	0,08520	0,97352	0,45600
	6671	0,03151	0,98991	0,60250
	Dense + 6671	0,04049	0,98569	0,66560
2.	1070	0,08718	0,97383	0,60060
	Dense + 1070	0,11499	0,96137	0,49430
	6671	0,06416	0,98104	0,61150
	Dense + 6671	0,10230	0,96652	0,46280
3.	1070	5,25139	0,19782	0,20360
	Dense + 1070	9,74315	0,09128	0,11250
	6671	9,50991	0,05815	0,09650
	Dense + 6671	11,37735	0,04399	0,07570
4.	1070	14,51582	0,00872	0,00890
	Dense + 1070	15,25460	0,00249	0,00410
	6671	11,11006	0,03705	0,07660
	Dense + 6671	13,87078	0,00973	0,03470
5.	1070	0,06733	0,97788	0,61010
	Dense + 1070	0,11758	0,95981	0,77120
	6671	0.09000	0,96924	0,72530
	Dense + 6671	0,11625	0,95858	0,78200
6.	1070	0,10584	0,96636	0,59590
	Dense + 1070	0,16804	0,94455	0,49270
	6671	0,15331	0,94900	0,61660
	Dense + 6671	0,12240	0,95894	0,56540
7.	1070	0,09743	0,96947	0,60480
	Dense + 1070	0,09618	0,96542	0,53530
	6671	0,12581	0,95730	0,71460
	Dense + 6671	0,11545	0,96009	0,63990
8.	1070	0,09765	0,96573	0,65070
	Dense + 1070	0,07818	0,97414	0,56570
	6671	0,08527	0,97088	0,72870
	Dense + 6671	0,09447	0,96745	0,64300

Tabelle 3.5.: Ergebnisse der Tests und der Objekterkennung der CNNs der Experiment eins bis acht.

die hier verwendete Normalisierung sind sowohl Test- als auch Objekterkennungsgenauigkeit die schlechtesten. Die zweitbeste Trainingsgenauigkeit $0,93271$ wurde auch wieder im ersten Experiment erreicht. Die Testgenauigkeit $0,97352$ ist hier die zweitbeste, die kaum bessere $0,97414$ wurde im achten Experiment erreicht. Die beste Objekterkennungsgenauigkeit $0,77120$ wurde im fünften Experiment erreicht.

Die beste Trainingsgenauigkeit $0,97432$ für das CNN mit 6671 Klassen ohne zusätzliche Dense-Layer wurde im ersten Experiment erreicht, ebenso die beste Testgenauigkeit $0,98991$. Die beste Objekterkennungsgenauigkeit wurde ebenfalls erst im achten Experiment erreicht mit $0,72870$ (Trainingsgenauigkeit $0,87568$, Testgenauigkeit $0,97088$).

Und schließlich für das CNN mit 6671 Klassen und zusätzlichen Dense-Layer wurde die beste Trainingsgenauigkeit $0,96245$ sowohl im ersten wie auch im dritten Experiment erreicht, der Loss war dabei im dritten Experiment niedriger $0,11995$ zu $0,12687$. Die beste Testgenauigkeit $0,98569$ wurde im ersten Experiment erreicht. Die beste Objekterkennungsgenauigkeit $0,78200$ im fünften Experiment (Trainingsgenauigkeit $0,87668$, Testgenauigkeit $0,95858$).

Dass die besten Test- und Trainingsgenauigkeiten im ersten Experiment erreicht wurden, ist damit zu begründen, dass hier weder eine Normalisierung der Daten erfolgte (drittes und viertes Experiment) und kein `GaussianNoise`-Layer verwendet wurde (fünftes bis achtes Experiment). Da beide Methoden dazu dienen, Rauschen ins Training zu bringen, sodass robustere Features gelernt werden, ist es ersichtlich, dass hier das Training schlechtere Werte liefert. Zum Anderen zeigt sich auch, dass die Ersetzung der Max-Pooling-Layer durch Convolutional-Layer im Vergleich zum zweiten Experiment und damit verbunden eine größere Anzahl an Parametern, zwar schlechte Werte liefert, für Trainings- und Testgenauigkeit die aber meistens die Zweitbesten sind.

Die Auswirkung der Anzahl an Epochen zeigt sich auch an den Experimenten sechs bis acht, der einzige Unterschied hier ist die höhere Anzahl an Trainingsepochen, die Trainings- und Testgenauigkeit verbessert sich jeweils.

Für die besten Objekterkennungsgenauigkeiten zeichnet sich ein etwas anderes Bild. Für die CNNs ohne zusätzliche Dense-Layer wurden hier die besten Ergebnisse im achten Experiment erreicht, für die CNNs mit zusätzlichen Dense-Layer im Fünften. Daran zeigt sich zum Einen, dass durch das zusätzliche `GaussianNoise`-Layer die CNNs bessere/robustere Features gelernt haben und so diese bei der Objekterkennung die Hieroglyphen besser unterscheiden konnten. Auch waren die zusätzlichen Epochen im Training nötig, sodass die Convolutional-Layer, statt der Max-Pooling-Layer, ausreichende Features lernen konnten. Die Ersetzung von Max-Pooling-Layer und Convolutional-Layer ist somit grundsätzlich möglich. Zum andren haben die CNNs mit zusätzlichen Dense-Layer nur im fünften Experiment, abgesehen vom CNN mit 6671 Klassen im ersten Experiment, bessere Ergebnisse als die

Netze ohne zusätzliche Dense-Layer geliefert. Hauptsächlich liegt das daran, dass durch die zusätzlichen Layer die Parameteranzahl erhöht wurde und dadurch bei gleich bleibender Trainingsepochen Anzahl nicht gleich schnell konvergieren.

Dass die zwei CNNs mit 6671 Klassen jeweils bessere Objekterkennungsgenauigkeiten haben als die zwei CNNs mit 1070 Klassen, hängt damit zusammen, dass trotz der 5601 zusätzlichen Klassen auf einer größeren Datenmenge trainiert wurde und so stärkere/robustere Features gelernt worden sind und so die Erkennung der Hieroglyphen insgesamt weniger fehleranfällig geworden ist.

3.5. Blockweises Trainieren

Im Folgenden werden die Experimente beschrieben, bei denen die CNNs blockweise trainiert wurden.

In ersten vorläufigen Tests, die nach Anpassung auf die neue Keras Version durchgeführt wurden, hat sich bereits abgezeichnet, dass das Vorgehen wie im vorherigen Abschnitt mit dem Vortrainieren eines Autoencoders, dem anschließenden Kopieren der Gewichte ins CNN und dem erst anschließenden Trainieren des CNNs keine vergleichbaren Ergebnisse liefert und dass der Autoencoder im Allgemeinen schlechtere Ergebnisse liefert (vgl. Abschnitt 3.5.1). Es folgt ein Experiment, bei dem ein Autoencoder trainiert wurde, anschließen folgen zehn Experimente, bei denen ausschließlich ein CNN trainiert wurde.

Die insgesamt elf Experimente wurden jeweils mit dem zweiten Datensatz erstellt, wobei der Autoencoder mit allen 16788 Bildern trainiert wurde, die CNNs hingegen nur mit 6063 Bildern (1286 Klassen), das heißt es sind jeweils mindestens drei Bilder pro Klasse vorhanden.

Die Eingabe in den folgen Experimenten waren RGB-Bilder mit einer Größe von 48×48 Pixel. Für das Training wurden die Bilder mit den Hieroglyphen (24×24 Pixel) ebenfalls wieder zufällig in dem größeren Eingabebild platziert und für die Tests zentriert. Darüber hinaus wurden noch weitere Transformationen mit Hilfe des `ImageDataGenerator` durch geführt, die genauen Transformationen werden im weiteren Verlauf aufgeführt.

Der Autoencoder wurde bei diesen Experimenten nicht blockweise trainiert, sondern vollständig in Einem. Die CNNs dahingegen wurden blockweise trainiert. Es wurden dabei jedes mal vier Blöcke trainiert.

3.5.1. Autoencoder

Der Autoencoder wurde mittels der funktionalen API erstellt. Die generelle Struktur folgt dabei der aus den vorhergegangen Experimenten und ist in Tabelle 3.6 noch

Encoder	Decoder
conv 3×3	conv 3×3
conv 3×3	conv 3×3
conv 3×3	upsample 2×2
conv 3×3 (*stride* 2×2)	dropout $p = 0,25$
dropout $p = 0,25$	conv 3×3
conv 3×3	conv 3×3
conv 3×3	upsample 2×2
conv 3×3 (*stride* 2×2)	dropout $p = 0,25$
dropout $p = 0,25$	conv 3×3
conv 3×3	conv 3×3
conv 3×3	conv 3×3
conv 3×3 (*stride* 2×2)	upsample 2×2
dropout $p = 0,25$	dropout $p = 0,25$
	conv 3×3

Tabelle 3.6.: Netzstruktur des Autoencoder des neunten Experiments.

Anzahl an Parameter	Epochen	Trainings Loss	Test Loss	Trainingszeit
3.838.659	10	0,07408	0,06727	0,58h

Tabelle 3.7.: Ergebnisse des Autoencoders des neunten Experiments.

einmal aufgeführt. Trainiert wurde mit allen 16788 Bildern aus dem zweiten Datensatz und zusätzlich zur zufälligen Positionierung wurden die Bilder noch zufällig um bis zu 20° Grad rotiert und zufällig geschert um bis zu $0,3rad$.

Die Convolutional-Layer haben alle 64 Filter. Als Optimierungsalgorithmus wurde Adam (vgl. Abschnitt 2.1.5) benutzt. Die Aktivierungsfunktionen war jeweils SReLU (vgl. Abschnitt 2.1.3) und als Kostenfunktion wurde binary crossentropy verwendet, einer Abwandlung der categorical crossentropy (vgl. Abschnitt 2.1.4), die mit folgender Formel berechnet $f(t, o) = -(t \log(o) + (1 - t) \log(1 - o))$ wird.

In Tabelle 3.7 sind die Ergebnisse des Trainings und der Tests aufgeführt zusammen mit Parameteranzahl, Trainingszeit und der Anzahl der Trainingsepochen und in Abbildung 3.4 sind Rekonstruktionen von 26 Hieroglyphen dargestellt.

Der Autoencoder braucht wesentlich weniger Trainingsepochen als in den vorangegangen Experimenten, 10 Epochen mit 0,58h Trainingszeit. Die Losswerte sind hier allerdings auch wesentlich größer. Für das Training ist der Unterschied $0,07408$ zu $0,00370$ und für den Test $0,06727$ zu $0,00046$ mit den Werten aus dem zweiten Experiment, dem die Netzstruktur am ähnlichsten ist.

Abbildung 3.4.: Ausgabebilder des Autoencoders des neunten Experimentes.

Schaut man sich die rekonstruierten Hieroglyphen in Abbildung 3.4 und vergleicht sie mit den Rekonstruktionen aus Abbildung 3.3, so lässt sich erkennen, dass die Rekonstruktionen aus diesem Experiment wesentlich unschärfer sind, besonders wenn es sich um Details oder feine Linien handelt. Im Gegensatz dazu haben die Rekonstruktionen hier kein Problem mit einem gräulichen Hintergrund. Allgemein sind die Hieroglyphen trotz allem recht gut zu erkennen.

3.5.2. CNN

Die folgenden CNNs wurden ebenfalls mittels der funktionalen API erstellt, hätten aber auch mit der bisherigen API erstellt werden können, und haben grob die selbe Netzstruktur wie in den vorherigen Experimenten.

Zum Trainieren wurde ebenfalls der zweite Datensatz benutzt. Es wurden allerdings nicht alle Bilder benutzt, sondern nur diejenigen, bei denen mindestens drei Bilder pro Klasse vorhanden sind, dass heißt es wurde mit 6063 Bildern mit 1286 Klassen trainiert.

In Tabelle 3.8 sind für die Experimente zehn bis neunzehn die Netzstrukturen genau aufgeführt.

Die CNNs, die hier verwendet wurden, bestehen ausschließlich aus Convolutional-Layer mit ein paar eingefügten Dropout-Layer. In den Experimenten zehn bis 14., 16. und 18. bestehen die Blöcke eins bis drei aus drei Convolutional-Layer mit einem 3 Filter, gefolgt von einem Convolutional-Layer mit 3×3 Filter und einer *stride* von 2×2 und anschließend ein Dropout-Layer mit $p = 0,25$. Für die Experiment 15., 17., und 19. haben die letzten beiden Blöcke nur zwei Convolutional-Layer, ansonsten sind die Blöcke gleich.

Der vierte Block ist bei allen Experimenten gleich und besteht aus zwei Convolutional-Layer, das erste mit einem Filter von 6×6 und das zweite mit einem Filter von 1×1. Dieses CNN ist vergleichbar mit dem CNN mit zwei Dense-Layer in den vorherigen Experimenten. Die zwei Convolutional-Layer sind dabei praktisch gleichzusetzen mit den Dense-Layer durch die gewählten Filtergrößen.

Das Convolutional-Layer im fünften Block übernimmt dabei die Aufgabe der Dense-Layer der vorherigen CNNs, d. h. die Ausgabematrix hat eine Größe von $(\#Klassen, 1, 1)$ mit $\#Klassen = 1286$, der Anzahl an Klassen. Das anschließende Flatten-Layer transformiert die Dimensionalität von $(1286, 1, 1)$ auf $(1286,)$ runter sodass die Softmax-Aktivierung durchgeführt werden kann. Sowohl die Softmax-

Block	Experiment	
	10., 11., 12., 13., 14., 16., 18.	15., 17., 19.
1.	conv 3×3 conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$	conv 3×3 conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$
2.	conv 3×3 conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$	conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$
3.	conv 3×3 conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$	conv 3×3 conv 3×3 conv 3×3 (*stride* 2×2) dropout $p = 0,25$
4.	conv 6×6 conv 1×1	
5.	conv flatten softmax	

Tabelle 3.8.: Netzstruktur der CNNs der Experimente zehn bis 19.

Experiment	Block			
	1.	2.	3.	4.
10.	64	64	64	5144
11.	64	64	64	5144
12.	64	64	64	5144
13.	64	64	64	5144
14.	64	64	64	5144
15.	64	64	64	5144
16.	128	128	128	5144
17.	128	128	128	5144
18.	32	32	32	5144
18.	32	32	32	5144

Tabelle 3.9.: Anzahl der Filter der Convolutional-Layer in den Experimenten zehn bis 19.

Aktivierung als auch das Flatten-Layer haben keine eigenen Gewichte.

Abgesehen vom Convolutional-Layer im fünften Block, das eine Softmax-Aktivierung hat, haben alle anderen SReLU (vgl. Abschnitt 2.1.3).

Die CNNs wurden, wie bereits erwähnt, blockweise trainiert. Dabei wurde mit dem ersten Block angefangen, dann wurde der Zweite hinzugefügt, danach der Dritte und Vierte dabei wurden die Gewichte aus den vorherigen Blöcken beibehalten. Der fünfte Block bildet dabei eine Ausnahme, da es sich hier um den Klassifikator handelt, wurde dieser immer als letzter Block verwendet, egal welcher Block davor kam. Die Gewichte wurden ebenfalls nicht beibehalten, da sich die Dimensionalität jedes mal ändert, da das Convolutional-Layer unterschiedliche Filtergrößen hat. Die Filtergröße ist abhängig vom Block, der davor verwendet wird, dass bedeutet, wenn das Convolutional-Layer nach dem ersten Block folgte, hat es einen Filter von 24×24, nach dem zweiten von 12×12, nach dem dritten von 6×6 und nach dem vierten von 1×1. Dies hat den Grund, da die Ausgabegröße immer gleich bei $(1286, 1, 1)$ bleiben sollte.

In Tabelle 3.9 sind die Filteranzahlen der einzelnen Convolutional-Layer aufgeführt, dabei haben die Convolutional-Layer in einem Block immer die gleiche Anzahl an Filtern. Die Experimente zehn bis 15. haben dabei immer 64 Filter in den Blöcken eins bis drei, danach folgen jeweils zwei Experimente mit 128 und 32 Filtern. Der vierte Block hat in allen Experimenten 5144 Filter.

Die Bilder wurden während des Trainings, bevor sie ins Netz gegeben wurden, wieder zusätzlich mit Hilfe des `ImageDataGenerator` augmentiert. Wie auch schon bei den vorherigen Experimenten wurden die Hieroglyphen (24×24 Pixel) zufällig in dem Eingabebild (48×48 Pixel) positioniert. Darüber hinaus wurden die Bilder

Experiment	Rotation	Scherung	Breitenverschiebung	Höhenverschiebung
10.				
11.				
12.	20°	0,3		
13.	20°	0,3		
14.	20°	0,3	0,2	0,2
15.	20°	0,3	0,2	0,2
16.	20°	0,3	0,2	0,2
17.	20°	0,3	0,2	0,2
18.	20°	0,3	0,2	0,2
19.	20°	0,3	0,2	0,2

Tabelle 3.10.: Transformationen mittels des `ImageDataGenerator` in den Experimenten zehn bis 19.

noch zufällig um bis zu 20° rotiert, um bis zu $0,3rad$ geschert oder in der Höhe oder Breite um bis zu einen Anteil von 0,2 der Höhe oder Breite verschoben. Dabei wurden in unterschiedlichen Experimenten unterschiedliche Methoden gewählt, eine genau Auflistung ist in Tabelle 3.10 zu finden. Grundlage dafür bildet unter anderem der Blogpost *Building powerful image classification models using very little data*[11] von François Chollet, dem Autoren von Keras.

In den Tabellen 3.11 und 3.12 sind die Ergebnisse der zehn CNN Experimente des blockweisen Trainierens aufgeführt. In Tabelle 3.11 sind die Ergebnisse des Trainings aufgeführt, beginnend mit der Anzahl an Parameter und Trainingsepochen und folgend mit Loss und Genauigkeit und Trainingszeit. In Tabelle 3.12 sind die Ergebnisse der Tests und der Objekterkennung aufgeführt, mit Loss und Genauigkeit und Objekterkennungsgenauigkeit.

In den Experimenten zehn und elf wurde verglichen, welche Auswirkung eine Verdoppelung der Anzahl der Trainingsepochen hat. Im zehnten Experiment wurde jeder Block 100 Epochen trainiert, im elften 200. Die gesamte Trainingszeit hat sich damit von $5,04h$ $(7,66h)$ auf $10,21h$ $(15,5h)$ mehr als verdoppelt. Dabei ist deutlich zu sehen, dass das Training des ersten Blockes wesentlich länger braucht, als das des zweiten und dritten Blockes. Bevor es wieder für den vierten Block länger braucht. Der Grund dafür hängt mit der Netzstruktur zusammen. Nachdem ersten Block hat die Eingabe in den fünften Block eine Dimension von $(64, 24, 24)$ voraus ein 1286-elementig Vektor errechnet wird, nach dem Zweiten sind es nur noch ein viertel davon $(64, 12, 12)$ und nach dem Dritten nur noch ein viertel davon $(64, 6, 6)$. Der vierten Block halbiert die Eingabe nicht, sondern etwas mehr als verdoppelt sie

[11] `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`

Experiment	Block	#Parameter	Epochen	Loss	Genauigkeit	Zeit
10.	1.	49.437.894	100	0,84426	0,79400	2,51h
	2.	14.509.510	100	0,43837	0,86954	1,35h
	3.	5.888.198	100	0,22347	0,92891	1,18h
	4.	47.904.390	100	0,30069	0,91390	2,62h
11.	1.	49.437.894	200	0,53476	0,86871	5,06h
	2.	14.509.510	200	0,35404	0,90104	2,75h
	3.	5.888.198	200	0,16458	0,94854	2,40h
	4.	47.904.390	200	0,23654	0,93963	5,29h
12.	1.	49.437.894	100	2,01971	0,51460	2,54h
	2.	14.509.510	100	1,06094	0,71136	1,37h
	3.	5.888.198	100	0,55806	0,83078	1,20h
	4.	47.904.390	100	0,54436	0,83127	2,65h
13.	1.	49.437.894	200	1,25569	0,68333	4,96h
	2.	14.509.510	200	0,71800	0,79762	2,82h
	3.	5.888.198	200	0,39892	0,87828	2,38h
	4.	47.904.390	200	0,38210	0,88323	5,20h
14.	1.	49.437.894	200	2,08344	0,50272	5,05h
	2.	14.509.510	200	1,32305	0,64341	2,72h
	3.	5.888.198	200	0,84015	0,76117	2,37h
	4.	47.904.390	200	0,64951	0,80224	5,26h
15.	1.	49.437.894	200	2,15911	0,49431	5,03h
	2.	14.325.126	200	1,32824	0,64291	2,59h
	3.	5.630.022	200	0,88751	0,74650	2,16h
	4.	47.646.214	200	0,71729	0,78806	5,01h
16.	1.	99.095.686	200	1,71044	0,58981	10,24h
	2.	29.533.830	200	1,14264	0,69009	5,69h
	3.	12.586.118	200	0,64094	0,81065	4,80h
	4.	63.491.142	200	0,63707	0,81527	8,54h
17.	1.	99.095.686	200	2,00501	0,53620	9,42h
	2.	29.091.334	200	1,38917	0,63549	4,79h
	3.	11.922.310	200	0,93469	0,73973	3,81h
	4.	62.827.334	200	0,75036	0,76992	7,27h
18.	1.	24.691.942	200	2,40605	0,43196	3,00h
	2.	7.190.886	200	1,24348	0,65727	1,71h
	3.	2.843.366	200	0,69821	0,78558	1,64h
	4.	40.415.142	200	0,61383	0,81593	4,40h
19.	1.	24.691.942	200	2,72803	0,38166	2,83h
	2.	7.107.910	200	1,67804	0,56968	1,51h
	3.	2.732.710	200	1,28173	0,64457	1,40h
	4.	40.304.486	200	0,82658	0,75177	3,99h

Tabelle 3.11.: Ergebnisse des Trainings der CNNs der Experimente zehn bis 19.

Experiment	Block	Loss	Genauigkeit	Objekterkennung
10.	1.	0,67479	0,82006	
	2.	0,14926	0,95794	
	3.	0,07115	0,97427	0,49990
	4.	0,09666	0,96899	0,44610
11.	1.	0,35663	0,90285	
	2.	0,12472	0,96454	
	3.	0,03517	0,98730	0,60060
	4.	0,09613	0,97509	0,47400
12.	1.	1,41235	0,64358	
	2.	0,28831	0,91209	
	3.	0,11225	0,96338	0,67450
	4.	0,13065	0,95217	0,58640
13.	1.	0,66962	0,81824	
	2.	0,16065	0,94442	
	3.	0,08230	0,97229	0,68170
	4.	0,08306	0,96850	0,58650
14.	1.	1,06775	0,71862	
	2.	0,36626	0,89098	
	3.	0,15095	0,94969	0,67320
	4.	0,15989	0,94541	0,60120
15.	1.	1,09006	0,71780	
	2.	0,46027	0,86294	
	3.	0,16658	0,94475	0,64690
	4.	0,15994	0,94079	0,63440
16.	1.	0,79734	0,78377	
	2.	0,34265	0,88949	
	3.	0,16802	0,94343	0,66250
	4.	0,21958	0,92726	0,55760
17.	1.	1,00405	0,73544	
	2.	0,31818	0,89906	
	3.	0,17404	0,94293	0,64690
	4.	0,15181	0,94656	0,61600
18.	1.	1,52238	0,60746	
	2.	0,47037	0,85915	
	3.	0,15008	0,94953	0,64130
	4.	0,21166	0,93023	0,56090
19.	1.	1,60661	0,60795	
	2.	0,58883	0,84348	
	3.	0,27463	0,91918	0,59800
	4.	0,14652	0,95118	0,58650

Tabelle 3.12.: Ergebnisse der Tests und der Objekterkennung der CNNs der Experimente zehn bis 19.

auf $(5144, 1, 1)$, weswegen der vierte Block wieder länger braucht. Dies ist auch an der Anzahl der Parameter zu erkennen, im ersten Block hat das Netz 49.437.894 Parameter, im Zweiten 14.509.510 Parameter, im Dritten 5.888.198 Parameter und im Vierten 47.904.390 Parameter fast wieder so viele wie im Ersten.

Im zehnten Experiment ist der Trainingsloss bei $0,22347$ im dritten Block und bei $0,30069$ im Vierten, die Trainingsgenauigkeit verhält sich ähnlich, mit $0,92891$ im dritten Block und $0,91390$ im Vierten. Die Testwerte zeigen das gleiche Verhalten, sind aber besser. Der Loss liegt bei $0,07115$ im Dritten steigt auf $0,09666$ im Vierten und die Testgenauigkeit ist $0,97427$ im dritten Block und fällt auf $0,96899$ im Vierten. Der Trend der sich hier abzeichnet, das der vierte Block die Ergebnisse nicht verbessert zeigt sich auch bei den Objekterkennungsgenauigkeiten. Für den dritten Block ist sie $0,49990$ und für den Vierten $0,44610$.

Die Steigerung auf 200 Trainingsepochen wirkt sich generell positiv aus. Der Trainingsloss ist $0,16458$ im dritten Block und $0,23654$ im Vierten. Die Trainingsgenauigkeit $0,94854$ im dritten Block und fällt auf $0,93963$ im Vierten. Die Werte für die Test sind ebenfalls wieder niedriger als die des Trainings. Der Loss ist $0,03517$ im dritten Block und $0,09613$ im Vierten und die Genauigkeit $0,98730$ im dritten Block und fällt auf $0,97509$ im Vierten. Die Objekterkennung ist für den dritten Block $0,60060$ und für den vierten Block $0,47400$. Bei Werte sind besser als im vorherigen Experiment, die Steigerung für den dritten Block ist aber deutlich größer $0,10070$ als beim Vierten $0,02790$.

Im zwölften und 13. Experiment ändert sich nur, dass die Trainingsbilder rotiert und geschert werden (vgl. Tabelle 3.10). Das Netz an sich ist das gleiche wie in den zwei vorherigen Experimenten, die Parameteranzahl ebenfalls. Die Trainingszeiten ändern sich geringfügig auf $5,11h$ $(7,76h)$ im zwölften Experiment und auf $10,16h$ $(15,36h)$ im 13. Experiment. Die Loss- und Genauigkeitswerte verschlechterten sich bei beiden Experimenten, sowohl im Training als auch im Test. Im zwölften Experiment ist der Trainingsloss $0,55806$ im dritten Block und $0,54436$ im Vierten, bei den Tests $0,15095$ im dritten Block und steigt auf $0,15989$ im Vierten. Die Trainingsgenauigkeit im zwölften Experiment ist $0,83078$ im dritten Block und $0,83127$ im Vierten. Die Testgenauigkeiten sind $0,97229$ im dritten Block und $0,96850$ im Vierten. Für das 13. Experiment sind sie Genauigkeitswerte des Trainings $0,87828$ im dritten Block und $0,88323$ im Vierten und $0,97229$ und $0,96850$ im Test. Dahingegen sind alle vier Objekterkennungsgenauigkeitswerte besser. Im zwölften Experiment verbessert sich die Genauigkeit auf $0,67450$ für den dritten Block und $0,58640$ für den Vierten und im 13. Experiment weiter auf $0,68170$ für den dritten Block und $0,58650$ für den Vierten. Für den dritten Block ist die Objekterkennungsgenauigkeit des 13. Experimentes die Beste dieser zehn Experimente.

An diesen Ergebnissen zeichnet sich deutlich ab, dass das durch die Rotation und Scherung eingebrachte Rauschen in den Trainingsbildern trotz schlechterer Werte,

sowohl im Training als auch in den Tests, deutlich bessere Ergebnisse bei der Objekterkennung liefert.

Darauf aufbauend wurde im 14. Experiment zusätzlich noch eine Breiten- und Höhenverschiebung vorgenommen (vgl. Tabelle 3.10). Das Netz bleibt dasselbe wie in den bisherigen Experimenten. Im Vergleich zum 13. Experiment verschlechtern sich die Trainings- und Testwerte ebenfalls. Der Trainingsloss liegt bei $0,84015$ im dritten Block und $0,64951$ im vierten, die Genauigkeit zeigt einen ähnlichen Verlauf mit $0,76117$ im dritten Block und $0,80224$ im vierten Block. Die deutliche Verbesserung vom dritten auf den vierten Block zeigt sich bei den Testwerten nicht, der Loss ist $0,15095$ und $0,15989$, die Genauigkeit $0,94969$ und $0,94541$. Beide Werte sind nahezu gleich. Die Objekterkennungsgenauigkeit im dritten Block liegt bei $0,67320$ und damit um $0,0085$ geringer als im vorherigen Experiment, für den vierten Block verbessert sie sich auf um $0,01470$ auf $0,60120$. Daran zeigt sich, dass das durch die Transformationen eingefügte Rauschen, bei gleicher Epochenanzahl, zunächst Verbesserungen bringt, dass aber ab einem bestimmten Punkt das Rauschen auch längere Trainingszeiten erfordert.

Im 15. Experiment wurde eine andere Netzstruktur untersucht. Es wurden jeweils im zweiten und dritten Block ein Convolutional-Layer entfernt (vgl. Tabelle 3.8). Durch geringere Anzahl an Convolutional-Layer und damit verbunden die Anzahl an Parametern, im zweiten Block 14.325.126 statt 14.509.510 (184.384 weniger), im Dritten 5.630.022 statt 5.888.198 (258.176 weniger) und im Vierten 47.646.214 statt 47.904.390 (ebenfalls 258.176 weniger), haben sich die Trainingszeiten insgesamt um $21,6min$ bzw. $36,6min$ zum vorherigen Experiment verringert. Die Trainingswerte sind um Einiges schlechter als im vorherigen Experiment, der Loss ist bei $0,88751$ im Dritten und bei $0,71729$ im vierten Block und die Genauigkeit liegt bei $0,74650$ und $0,78806$. Für die Testwerte ergibt sich ein etwas besseres Bild, sie sind nahezu ähnlich mit denen des 14. Experimentes. Der Loss ist bei $0,16658$ und $0,15994$ und die Genauigkeit bei $0,94475$ und $0,94079$. Auf die Objekterkennungsgenauigkeit wirkt sich das sehr unterschiedlich aus. Im Vergleich zum 14. Experiment sinkt sie um $0,0263$ auf $0,64690$ für den dritten Block, steigt aber um $0,0332$ auf $0,63440$ für den vierten Block. Dies ist der beste Wert für die Objekterkennung nach dem vierten Block bei diesen zehn Experimenten, sie ist trotz allem um $0,0473$ niedriger als die beste Genauigkeit für den dritten Block im 13. Experiment.

Die letzten vier Experimente untersuchen die Auswirkung der Filteranzahl auf die Netzstrukturen. Die bisherigen Experimente haben in allen Convolutional-Layer 64 Filter gehabt, die Experimente 16. und 17. haben jeweils 128 Filter und die Experimente 18. und 19. haben 32 Filter (vgl. Tabelle 3.9). Die Experimente 17. und 19. sind die Beiden bei denen die Netze jeweils zwei Convolutional-Layer weniger haben (vgl. Tabelle 3.8). Damit die Vergleichbarkeit mit den letzten beiden Experimenten sicher gestellt ist, wurden die selben Transformationen der Trainingsbilder durch

geführt und jeder Block 200 Epochen trainiert.

Wie zu erwarten hat eine Vordopplung der Filteranzahl im 16. und 17. Experiment große Auswirkungen auf die Parameteranzahl und Trainingszeit. Das Netz im 16. Experiment hat 99.095.686 Parameter im ersten Block, 29.533.830 im Zweiten, 12.586.118 im Dritten und 63.491.142 im Vierten. Im 17. Experiment sind es 442.496 Parameter weniger im zweiten Block, 663.808 im Dritten und Vierten. Die Trainingszeit im 16. Experiment beträgt $20,73h$ $(29,27h)$ und $18,02h$ $(25,29h)$ im 17. eine Reduzierung von $2,71h$ $(3,98h)$, die nicht unerheblich ist. Sie brauchen trotzdem am längsten zum Trainieren. Wie bereits in den beiden vorherigen Experimenten zu beobachten war, sind die Werte von Training und Test des kleineren Netzes (17. Experiment) schlechter. Der Trainingsloss des 16. Experimentes liegt bei $0,64094$ im dritten Block und $0,63707$ im Vierten und im 17. bei $0,93469$ und $0,75036$. Die Genauigkeiten zeichnen einen ähnlichen Trend $0,81065$ und $0,81527$ im 16. Experiment und $0,73973$ und $0,76992$ im 17. Der Testloss ist im 16. Experiment $0,16802$ im dritten Block und $0,21958$ im Vierten und im 17. Experiment entsprechend $0,17404$ und $0,15181$. Die Genauigkeit liegt bei $0,94343$ und $0,92726$ im 16. Experiment und $0,94293$ und $0,94656$ im 17.

Die Objekterkennungsgenauigkeit ist im 16. Experiment bei $0,66250$ für den dritten Block und $0,55760$ für den Vierten, beide schlechter als im 14. Experiment. Im 17. Experiment sind sie $0,64690$ und $0,61600$, der dritte Block ist gleich wie im 15. Experiment, der Vierte ebenfalls wieder schlechter.

Es zeichnet sich hier schon eine interessante Verhaltensweise ab. Es scheint, dass bei den kleineren Netzen die vierten Blöcke bessere Ergebnisse liefern als die dritten Blöcke, was im Gegensatz zu dem Verhalten bei den größeren Netzen steht. Hier ist sogar die Genauigkeit (und entsprechend auch der Loss) des vierten Blockes besser, als jene im 15. Experiment. In den Objekterkennungsgenauigkeiten zeichnet sich das insofern ab, als dass die Differenzen zwischen dem dritten und vierten Block wesentlich geringer sind.

Die Netze der letzten beiden Experimente 18. und 19. haben nur 32 Filter, halb so viele wie im 14. und 15. Experiment. Daraus ergibt sich im 18. Experiment 24.691.942 Parameter für den ersten Block, 7.190.886 für den Zweiten, 2.843.366 für den Dritten und 40.415.142 für den Vierten. Im 19. Experiment sind es im zweiten Block 82.976 Parameter weniger und 110.656 für den Dritten und Vierten. Diese beiden Experimente haben zum ersten Mal mehr Parameter im vierten Block als im Ersten, es ist somit zu erwarten, dass der vierte Block erheblich schlechtere Ergebnisse liefert. Für die Trainingszeit ergibt sich zunächst einmal $6,35h$ $(10,75h)$ für das 18. Experiment und $5,74h$ $(9,73h)$ für das 19. Experiment und sind für die Experimente mit 200 Epochen die Schnellsten.

Der Trainingsloss ist im 18. Experiment $0,69821$ für den dritten Block mit einer Genauigkeit von $0,78558$ und $0,61383$ für den Vierten mit einer Genauigkeit

von 0, 81593. Für die Tests ergibt sich entsprechend ein Loss von 0, 15008 mit einer Genauigkeit von 0, 94953 im dritten Block und 0, 21166 im Vierten mit einer Genauigkeit von 0, 93023. Zum einen sind die Testwerte des vierten Blockes nicht so schlecht wie erwartet, sie sind sogar noch besser als die im 16. Experimentes und nur gering schlechter als im 14. Experiment. Für den dritten Block sind sie fast gleich mit denen des 14. Experimentes. Die Objekterkennungsgenauigkeit hingegen ist erheblich schlechter, für den dritten Block liegt sie bei 0, 64690 und für den Vierten bei 0, 56090.

Im 19. Experiment ist der Trainingsloss 1, 28173 und die Trainingsgenauigkeit 0, 64457 für den dritten Block und einem Trainingsloss von 0, 82658 für den vierten Block im einer Trainingsgenauigkeit von 0, 75177. Für die Tests ist der Loss bei 0, 27463 im dritten Block und 0, 14652 im Vierten. Die Genauigkeit liegt bei 0, 91918 im dritten Block und 0, 95118 im vierten Block. Die Objekterkennungsgenauigkeit liegt bei 0, 59800 für den dritten Block und 0, 58650 für den Vierten. Die Testgenauigkeit für den vierten Block ist im Vergleich zum 15. und 17. Experiment hier am höchsten, was sich aber nicht in der Objekterkennungsgenauigkeit wieder spiegelt. Insgesamt sind hier die Objekterkennungsgenauigkeiten sowohl für den dritten und vierten Block im Vergleich zum 15. und 17. Experiment am schlechtesten.

Im Vergleich zu den Experimenten des vorherigen Abschnitts 3.4, bei dem die besten Objekterkennungsgenauigkeiten für die CNNs mit 1070 Klassen bei 0, 65070 und 0, 77120 (vgl. Tabelle 3.5), liegen sind sie hier im 13. Experiment bei 0, 68170 für den dritten Block und 0, 63440 im 15. Experiment für den vierten Block. Für das CNN mit nur drei Blöcken bzw. ohne zusätzliche Dense-Layer ist die Objekterkennungsgenauigkeit hier besser, für das Andere allerdings das CNN aus dem vorherigen Abschnitt.

Die Experimente sind auch nicht genau miteinander vergleichbar, da die Anzahl an Klassen von 1070 auf 1286 gestiegen ist. Trotzdem sind einige interessante Dinge zu bemerken. So sind zum Einen die CNNs der Experimenten des vorherigen Abschnitts mit zusätzlichen Dense-Layer oft besser als die ohne. Im Gegensatz dazu ist bei den oben beschrieben Experimenten das Gegenteil der Fall. Zwar sind hier keine Dense-Layer verwendet worden sondern Convolutional-Layer. Da diese aber so gewählt wurden, das sie wie Dense-Layer funktionieren, sind die Netze trotzdem miteinander vergleichbar.

Zusammenfassend kann hier gesagt werden, dass die zusätzlichen Transformationen der Bilder die Ergebnisse verbessern, dass die Anzahl an Filtern (32, 64, 128) kaum einen Einfluss auf die Testgenauigkeit hat, aber bei der Objekterkennung wesentlich deutlicher hervortritt, dass die kleineren Netze schlechtere Ergebnisse liefern, auch schon bei nur zwei Convolutional-Layer weniger, und dass mehr Trainingsepochen bessere Ergebnisse liefern, dies aber mehr Zeit benötigt.

3.6. Going Deeper

Im folgenden Abschnitt werden die letzten Experimente dieser Arbeit vorgestellt. Diese drehen sich dabei alle um das Thema *Going Deeper*, also tiefere Netze zu trainieren.

Zum Trainieren wurde der zweite Datensatz benutzt. In jedem Experiment wurden zwei CNNs trainiert, das erste mit 6063 Bildern und 1286 Klassen, das Zweite mit 16421 Bildern und 6465 Klassen. Es wurde ebenfalls wieder auf den RGB-Werten trainiert.

Wie auch bei den vorherigen Experimenten wurden die Bilder während des Trainings, bevor sie ins Netz gegeben wurden, mit Hilfe des `ImageDataGenerator` augmentiert. Es wurde die selbe Strategie wie in den Experimenten 14 bis 19 verfolgt. Die Bilder wurden somit zufällig um bis zu 20° rotiert, um bis zu $0,3rad$ geschert und in der Höhe oder Breite um bis zu einen Anteil von $0,2$ der Höhe oder Breite verschoben. Der einzige Unterschied besteht darin, dass die zufällige Positionierung der Hieroglyphen in den Eingabebildern nicht mehr stattfand, stattdessen sind alle Hieroglyphen zentriert. Auch ist die Größe der Eingabebilder von 48×48 Pixel auf 32×32 Pixel reduziert worden.

Im Gegensatz zu den vorherigen Experimenten werden bei den Folgenden unterschiedliche Netzstrukturen untersucht, die es ermöglichen, tiefere Netze Ende-zu-Ende zu trainieren. Als Vergleich zu den vorherigen Experimenten dient das 20. Experiment, das der gleichen blockweisen Trainingsmethode folgt wie in den Experimenten zehn bis 19. Das 21. Experiment ist eine Abwandlung des 20., das Ende-zu-Ende trainiert wurde. Die Netze in den Experimenten 22. und 23. sind sogenannte ResNets (vgl. Abschnitt 3.6.1) und im Letzten sind es Densely-Connected CNNs (vgl. Abschnitt 3.6.2).

Die CNNs im 20. Experiment bestehen aus drei Blöcken, jeder Block hat drei Convolutional-Layer mit einem 3×3 Filter, gefolgt von einem Convolutional-Layer mit 3×3 Filter und einer *stride* von 2×2 und anschließend ein Dropout-Layer mit $p = 0,25$. Das letzte Layer ist ein Convolutional-Layer mit Softmax-Aktivierung, das die Klassifikation vollzieht. Es hat dementsprechend entweder 1286 Filter oder 6465 Filter. Die anderen Convolutional-Layer haben im ersten Block 64 Filter, im Zweiten 128 und im Dritten 256. Es hat somit die selbe Netzstruktur wie die Experimente 10., 11., 12., 13., 14., 16. und 18. (vgl. Tabelle 3.8).

Die CNNs im 21. Experiment unterscheiden sich insofern, dass jedes Convolutional-Layer über Batch Normalization (vgl. Abschnitt 2.1.8 verfügt und das kein Dropout mehr vorhanden ist. Durch die Batch Normalization ist auch das blockweise Trainieren nicht mehr erforderlich, sondern es kann Ende-zu-Ende trainiert werden.

Bereits in ersten vorläufigen Tests mit Batch Normalization hat sich gezeigt, dass das Training mit einer fest vorgegebenen Anzahl an Epochen zu sehr unterschied-

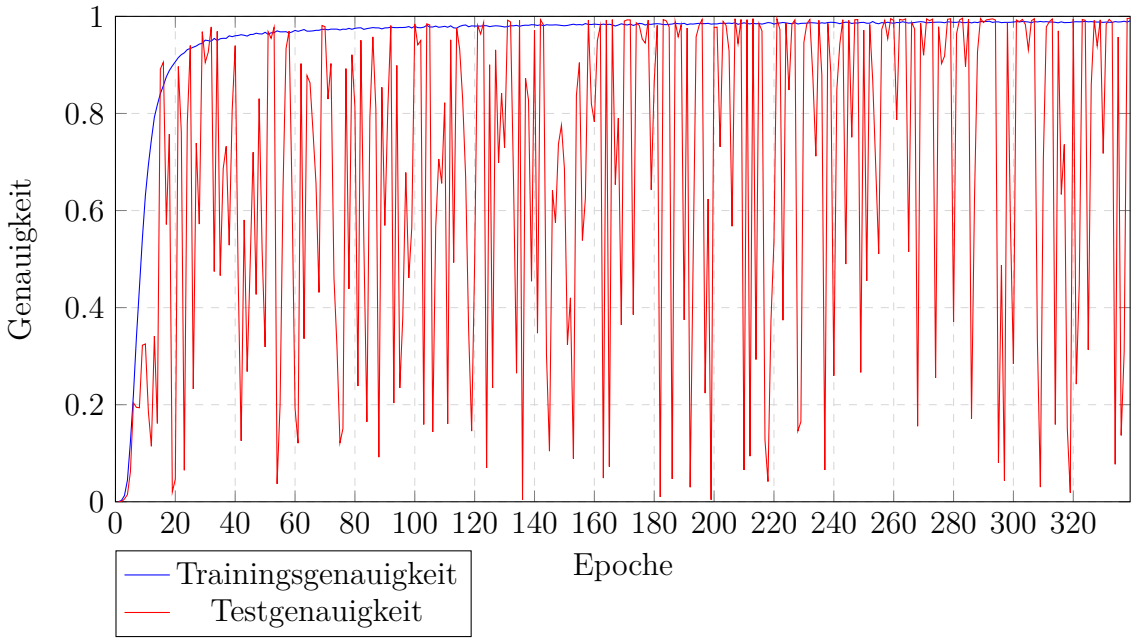


Abbildung 3.5.: Trainingsgenauigkeit und Testgenauigkeit für das CNN mit 6465 Klassen des 21. Experimentes.

lichen Ergebnissen führt. In Abbildung 3.5 ist die Trainingsgenauigkeit und die Testgenauigkeit für das CNN mit 6465 Klassen des 21. Experimentes abgebildet. Während die Trainingsgenauigkeit kontinuierlich absinkt, weist die Testgenauigkeit wilde Sprünge auf. Würde hier für eine feste Anzahl an Epochen trainiert, kann es sein, dass die Testgenauigkeit zufällig gerade gute Werte aufweist oder schlechte. Aus diesem Grund wurden die Netze der Experimente 21 bis 24 nicht mit einer festen Anzahl an Epochen trainiert, sondern so lange, bis ein Testloss von $\leq 0,01$ erreicht wurde. Einzige Ausnahme hierzu ist das zweite Netz (6465 Klassen) im 24. Experiment, hier wurde abgebrochen bei einem Testloss von $\leq 0,015$, da sonst die Zeitbeschränkung von sieben Tagen pro Job auf dem Cluster überschritten worden wäre.

Der Grund für die wilden Sprünge der Testgenauigkeit ist wohl darin zu finden, dass die Trainingsbilder mit ihren Transformationen zu unterschiedlich von den Testbildern sind und dies besonders durch die Regulierung, die Batch Normalization durchführt, verstärkt wird.

Ein Seiteneffekt dieser Trainingsstrategie ist, dass die Genauigkeits- und Testwerte sowohl für Training als auch Test sehr nahe beieinanderliegen und dadurch zum Vergleich keine allzu große Aussagekraft mehr haben.

Die CNNs mit 6465 Klassen haben zwei Objekterkennungsgenauigkeiten. Es wurde zusätzlich zu der Objekterkennung die auch für die CNNs mit 1286 Klassen

durchgeführt wurde noch eine zweite durchgeführt die Hieroglyphen aus allen 6465 Klassen haben. Beide Datensätze für die Objekterkennung haben 10.000 Bilder mit jeweils 10 Hieroglyphen.

3.6.1. Residual Neural Network (ResNet)

Ein Residual Neural Network (ResNet)[10] ist ein CNN, das aus sogenannten Residual-Blöcken besteht. In Abbildung 3.6 ist ein solcher Residual-Block einmal dargestellt. Die Funktionsweise eines Residual-Blockes ist dabei wie folgt: Ausgehend von einer Eingabe x, entweder dem Eingangsbild oder der Ausgabe eines anderen Convolutional-Layer, wird durch das erste Convolutional-Layer, inklusive Batch Normalization und Aktivierung (SReLU), gegeben und dann durch das zweite Convolutional-Layer. Bevor die Werte durch die Batch Normalization und die Aktivierung (SReLU) des zweiten Convolutional-Layer gegeben werden, wird die Eingabe x hinzuaddiert und anschließend durch die Batch Normalization und Aktivierung (SReLU) gegeben. Das erste Convolutional-Layer berechnet ganz normal $f(x)$ das zweite Convolutional-Layer berechnet hingegen $f(x) + x$. Der Name Residual rührt daher, dass hier der Rest (engl. residual) mit berechnet wird. Die größere Tiefe, die mit ResNets möglich ist, kommt dadurch zustande, dass der Gradient auch durch die zusätzliche Verbindung zurückgegeben wird und dadurch langsamer kleiner wird.

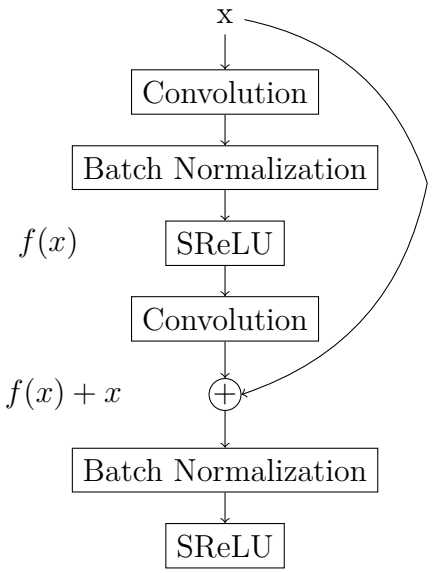


Abbildung 3.6.: Schematische Darstellung eines Residual-Blocks.

Zu beachten ist dabei, dass, wenn eines der Convolutional-Layer eine Dimensionsreduktion durchführt (*stride* ≥ 2), die Addition nicht mehr möglich ist bzw.

hier ebenfalls eine Dimensionsreduktion durch geführt werden muss, da sonst die Dimensionen nicht mehr übereinstimmen. Im Paper werden dazu zwei Methoden vorgestellt. Die erste Methode passt die Dimensionen aneinander an und füllt gegebenenfalls mit Nullen auf. Die Zweite führt eine Projektion durch, diese benötigt aber zusätzliche Parameter. Die Experimente in dieser Arbeit benutzen eine Abwandlung der ersten Methode, die durch den Lambda-Ausdruck `lambda t : K.repeat_elements(t[:, :, :: 2, :: 2], 2, 1)` beschrieben wird.

Es wird, da die *stride* immer 2×2, jeder zweite Wert des Ausgangs genommen und für die Anzahl der Filter, die sich verdoppelt, werden, anstatt mit Nullen aufzufüllen, die Werte wiederholt.

Die CNNs in den Experimenten 22 und 23 sind ResNets. Es gibt insgesamt drei Convolutional-Layer mit einer *stride* von 2×2 pro Netz. Alle Convolutional-Layer die nach einem Convolutional-Layer mit *stride* kommen verdoppeln die Anzahl der Filter, beginnend mit 64 Filtern (8×64, 8×128, 8×256, 9×512). Insgesamt haben die vier Residual Netze 34 Convolutional-Layer.

Die beiden ResNets des 23. Experimentes (einmal mit 1286 Klassen und einmal mit 6465 Klassen) haben zusätzlich noch eine extra Verknüpfung über alle Convolutional-Layer mit der gleichen Filteranzahl. Dass heißt zum Beispiel, dass die Ausgabe des achten Convolutional-Layer, welches das Letzte mit 64 Filtern ist, die Ausgabe des ersten Convolutional-Layer dazu addiert bekommt.

3.6.2. Densely-Connected CNN

Ein Densely-Connected CNN[15] besteht aus Densely-Connected Blöcken. In Abbildung 3.7 ist ein solcher Block bestehend aus vier Convolutional-Layer dargestellt. In einem Densely-Connected Block hat jedes Convolutional-Layer jedes vorherige Convolutional-Layer als Eingabe. Es werden dabei die Filter aneinandergehängt, so dass wenn alle Convolutional-Layer zum Beispiel zehn Filter berechnen, die Eingabe im dritten Convolutional-Layer 20 Filter sind, im Vierten 30 und so weiter. Im Gegensatz zu den Residual-Blöcken ist es nicht möglich, innnerhalb eines Densely-Connected-Block Convolutional-Layer mit einer *stride* ≥ 2 zu haben, mit Ausnahme des letzten Convolutional-Layer.

Durch diese vielen zusätzlichen Verbindungen zwischen den Convolutional-Layer ist es möglich, dass ein Convolutional-Layer sehr wenige Filter benötigt. Im Paper wurden zwei Varianten getestet, einmal mit 12 Filter und einmal mit 24 Filter. Convolutional-Layer der Netze im 24. Experiment haben 16 Filter und insgesamt 69 Convolutional-Layer, vier Blöcke mit 16 Convolutional-Layer plus drei Convolutional-Layer mit *stride* 2×2 und jeweils eines zur Ein- und Ausgabe.

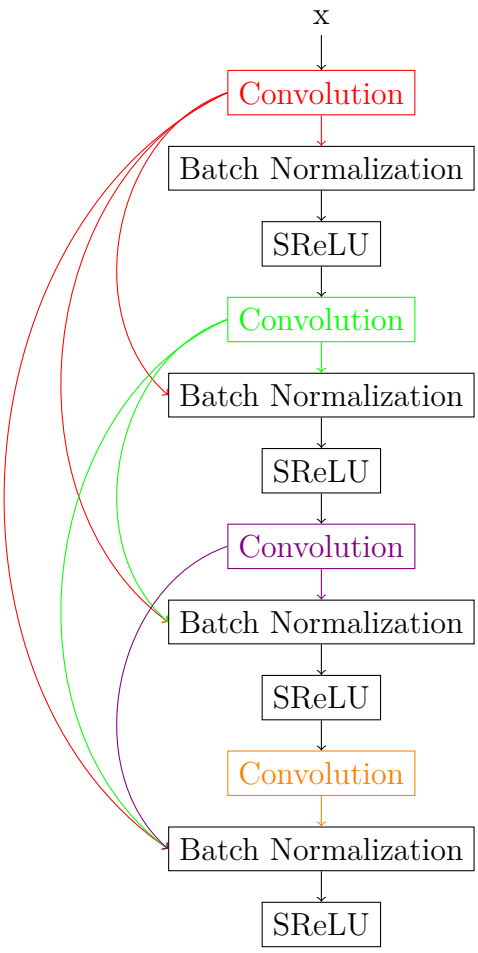


Abbildung 3.7.: Schematische Darstellung eines Densely-Connected-Blocks.

3.6.3. Ergebnisse

In Tabelle 3.13 sind für die Experimente 20 bis 24 die Anzahl an Parameter und die Anzahl der Convolutional-Layer aufgeführt. Die Netze werden dabei immer tiefer. Die ResNets im 22. und 23. Experiment haben dabei die meisten Parameter, einmal abgesehen von dem CNN des ersten Blocks im 20. Experiment mit 6465 Klassen.

In Tabelle 3.14 sind die Ergebnisse des Trainings der Experimente 20 bis 24 aufgelistet und in Tabelle 3.15 die Ergebnisse der Tests und der Objekterkennung.

Im 20. Experiment wurden zwei CNNs, einmal mit 1286 Klassen und einmal 6465 Klassen, trainiert. Die CNNs wurden blockweise trainiert, jeweils mit drei Blöcken. Jeder Block wurde 100 Epochen trainiert. Das Training für das CNN mit 1286 Klassen dauerte insgesamt $4,12h$ ($1,58h$, $1,26h$, $1,28h$), für das andere CNN mit

Experiment	Klassen	Block	#Parameter	#Convolution	
20.	1286	1.	v1.0.7	22.035.654	5
		2.	v1.0.7	12.267.142	9
		3.	v1.0.7	8.565.190	13
	6465	1.	v1.0.7	106.893.569	5
		2.	v1.0.7	54.698.689	9
		3.	v1.0.7	29.783.553	13
21.	1286		v1.0.7	8.570.834	13
	6465		v1.0.7	29.799.555	13
22.	1286		v1.0.7	40.027.602	34
	6465		v1.0.7	82.469.507	34
23.	1286		v1.0.8	40.027.602	34
	6465		v1.0.8	82.469.507	34
24.	1286		v1.0.8	20.331.234	69
	6465		v1.1.0	42.885.779	69

Tabelle 3.13.: Konfigurationen der CNNs der Experimente 20 bis 23.

Experiment	Klassen	Block	Epochen	Loss	Genauigkeit	Zeit
20.	1286	1.	100	1,01748	0,74600	1,58h
		2.	100	0,37632	0,89345	1,26h
		3.	100	0,29814	0,90830	1,28h
	6465	1.	100	0,91696	0,77072	14,16h
		2.	100	0,40400	0,88649	8,26h
		3.	100	0,43930	0,87163	5,84h
21.	1286		1240	0,02200	0,99175	20,43h
	6465		339	0,03491	0,99080	23,84h
22.	1286		2109	0,02920	0,98812	128,16h
	6465		920	0,04233	0,98660	167,39h
23.	1286		1018	0,02550	0,98911	49,66h
	6465		316	0,04371	0,98776	57,13h
24.	1286		1355	0,02511	0,99010	160,10h
	6465		403	0,04704	0,98435	142,79h

Tabelle 3.14.: Ergebnisse des Trainings der CNNs der Experimente 20 bis 23.

Experiment	Klassen	Block	Loss	Genauigkeit	Objekterkennung	
20.	1286	1.	0,26433	0,93221	0,70935	
		2.	0,05801	0,97988	0,78732	
		3.	0,03989	0,98681	0,78797	
	6465	1.	0,15116	0,95591	0,74197	0,75291
		2.	0,06213	0,98027	0,83154	0,84252
		3.	0,08889	0,97150	0,76121	0,77257
21.	1286		0,00908	0,99555	0,90948	
	6465		0,00977	0,99592	0,91257	0,92362
22.	1286		0,00975	0,99505	0,65953	
	6465		0,00881	0,99592	0,63585	0,62720
23.	1286		0,00952	0,99505	0,79183	
	6465		0,00976	0,99641	0,75976	0,77792
24.	1286		0,00919	0,99588	0,78272	
	6465		0,01478	0,99501	0.77229	0,79094

Tabelle 3.15.: Ergebnisse der Tests und der Objekterkennung der CNNs der Experimente 20 bis 23.

6465 Klassen insgesamt $28,26h$ ($14,16h$, $8,26h$, $5,84h$). Das Training ist damit etwas schneller als bei den Experimenten im vorherigen Abschnitt, was dadurch zu begründen liegt, dass die Eingabebilder kleiner sind.

Das CNN mit 1286 Klassen hat eine Trainingsgenauigkeit von $0,90830$ und eine Testgenauigkeit von $0,98681$. Die Objekterkennungsgenauigkeit liegt bei $0,78797$ und ist damit besser als bei allen vorherigen Experimenten.

Die Trainingsgenauigkeit für das CNN mit 6465 Klassen liegt bei $0,87163$ und die Testgenauigkeit liegt bei $0,97150$. Die Objekterkennungsgenauigkeit liegt bei $0,76121$ und $0,77257$. Hier ist die Objekterkennung besser für die 6465 Klassen als für die 1286 Klassen.

Das Training der CNNs im 21. Experiment dauert wesentlich länger als im vorherigen Experiment. Das CNN mit 1286 Klassen braucht $20,43h$ mit 1240 Epochen und das CNN mit 6465 Klassen $23,84h$ mit 339 Epochen. Die wesentlich längere Trainingsdauer liegt daran, dass anstatt für eine fest vorgegebene Anzahl an Epochen trainiert wurde, solange trainiert wurde, bis der Testloss $\leq 0,01$ war. Dadurch haben beide CNNs ebenfalls wesentlich bessere Trainings- und Testgenauigkeiten. Die Trainingsgenauigkeit liegt bei $0,99175$ und die Testgenauigkeit bei $0,99555$ für das CNN mit 1286 Klassen und für das CNN mit 6465 Klassen ist die Trainingsgenauigkeit $0,99080$ und die Testgenauigkeit $0,99592$.

Die Objekterkennungsgenauigkeit ist ebenfalls wesentlich besser und liegt bei $0,90948$ für das erste CNN und für das Zweite bei $0,91257$ und $0,92362$. Wie auch

schon im vorherigen Experiment ist die Genauigkeit für das CNN mit 6465 Klassen auf dem zweiten Datensatz besser. Zusätzlich sind aber auch beide Genauigkeiten des zweiten CNNs besser als die des Ersten.

Die beiden ResNets im 22. Experiment haben noch längere Trainingszeiten. Das Training des ResNet mit 1286 Klassen dauert $128,16h$ mit 2109 Epochen und des ResNet mit 6465 Klassen $167,39h$ mit 920 Epochen.

Das erste ResNet (1286 Klassen) hat eine Trainingsgenauigkeit von $0,98812$ und eine Testgenauigkeit von $0,99505$. Die Objekterkennungsgenauigkeit liegt aber nur bei $0,65953$.

Das zweite ResNet (6465 Klassen) hat eine Trainingsgenauigkeit von $0,98660$ und Testgenauigkeit von $0,99592$ und eine Objekterkennung von $0,63585$ und $0,62720$.

Das Training für die ResNets des 23. Experimentes braucht wesentlich weniger Zeit. Die Trainingszeiten liegen bei $49,66h$ mit 1018 Epochen für das ResNet mit 1286 Klassen und $57,13h$ mit 316 Epochen für das ResNet mit 6465 Klassen.

Die Trainingsgenauigkeiten sind $0,98911$ und $0,98776$ und die Testgenauigkeiten sind bei $0,99505$ und $0,99641$. Die Objekterkennungsgenauigkeiten liegen bei $0,79183$ für das ResNet mit 1286 Klassen und $0,75976$ bei $0,77792$ für das ResNet mit 6465 Klassen.

Die zusätzlichen Verbindungen in den ResNets des 23. Experimentes haben sowohl das Training erheblich beschleunigt, als auch zu besseren Ergebnissen in der Objekterkennung geführt.

Im 24. Experiment wurden abschließend zwei Densely-Connected CNNs trainiert. Hier hat das Trainings $160,10h$ mit 1355 Epochen für das erste CNN (1286 Klassen) gedauert und $142,79h$ mit 403 Epochen für das Zweite (6465 Klassen), wobei beim zweiten CNN schon bereits bei einem Testloss von $\leq 0,015$ abgebrochen wurde.

Das CNN mit 1286 Klassen hat eine Trainingsgenauigkeit von $0,99010$, eine Testgenauigkeit von $0,99588$ und die Objekterkennungsgenauigkeit liegt bei $0,78272$ und das CNN mit 6465 Klassen hat eine Trainingsgenauigkeit von $0,98435$, eine Testgenauigkeit von $0,99501$ und die Objekterkennungsgenauigkeit liegt bei $0,77229$ und $0,79094$. Die Objekterkennungsgenauigkeiten liegen hier auf einem ähnlichen Niveau wie beim vorherigen Experiment, liegen aber ins gesamt näher beieinander. Auch ist die Objekterkennungsgenauigkeit des Densely-Connected CNN mit 6465 Klassen für den zweiten Datensatz die Beste wie auch schon im 21. Experiment.

Abschließend ist festzustellen, dass das Ende-zu-Ende Trainieren der CNNs im 21. Experiment mit $0,92362$ die beste Objekterkennungsgenauigkeit liefert und somit besser ist als das blockweise Trainieren. Die ResNets und die Densely-Connected CNNs haben sehr lange Trainingszeiten und liefern keine besseren Ergebnisse, sondern nur vergleichbare mit dem blockweisen Trainieren. Trotzdem ist es so möglich, sehr tiefe CNNs (mit bis zu 69 Convolutional-Layer) zu trainieren, die gute Ergebnisse liefern.

4. Zusammenfassung und Ausblick

4.1. Zusammenfassung

Im Rahmen dieser Arbeit wurden unterschiedliche Architekturen von Convolutional Neural Networks (CNN) vorgestellt und auf ihre Eignung für die Objekterkennung von ägyptischen Hieroglyphen hin untersucht. Zunächst wurden die Grundlagen von künstlichen neuronalen Netzen erläutert mit Schwerpunkt auf die Convolutional-Layer, die die wichtigsten Bestandteile von CNNs sind.

Die CNN wurden mit Hilfe der Bibliothek Keras (vgl. Abschnitt 2.3.3) erstellt, die während der Entstehung der Arbeit konstant weiterentwickelt wurde. Dadurch wurden unterschiedliche Experimente mit unterschiedlichen Versionen erstellt.

Eine besondere Schwierigkeit dieser Arbeit lag in der sehr dünnen Datenlage. Es wurden zwei Datensätze im Verlauf dieser Arbeit betrachtet. Der erste Datensatz (vgl. Abschnitt 2.2.1) hat insgesamt 13980 Bilder mit 6671 Klassen. Für einige Experimente wurde nur ein Teil dieses Datensatzes betrachtet, bei dem mindestens drei Bilder pro Klasse vorhanden sind, dieser umfasst 3210 Bilder mit 1070 Klassen. Beim zweiten Datensatz (vgl. Abschnitt 2.2.1) wurden lediglich nur zwei Teildatensätze betrachtet. Der Eine hat mindestens drei Bilder pro Klasse (6063 Bilder mit 1286 Klassen), der Andere nur zwei (16421 Bilder mit 6465 Klassen). Im Vergleich zu anderen Datensätzen wie z.B. dem ImageNet LSVRC-2012 Datensatz mit 1,2 Million Bildern mit 1000 Klassen ist dass sehr wenig. Zwei Schwierigkeiten die damit einhergehen und berücksichtigt werden müssen: Zum Einen muss das neuronale Netz ausreichend Kapazität aufweisen, um überhaupt lernen zu können. Zum Anderen, dass sehr leicht ein Overfitting stattfinden kann. Die Bilder für die Objekterkennung wurden künstlich erzeugt (vgl. Abschnitt 3.2).

In den ersten acht Experimenten (vgl. Abschnitt 3.4) wurde damit begonnen, einen Autoencoder zu trainieren, der lernen sollte, die Bilder wieder zu rekonstruieren. Anschließend wurden die Gewichte des Autoencoder, genauer die Gewichte des Encoders, in die CNNs als Initialgewichte kopiert. Es wurde dabei pro Experiment ein Autoencoder und vier CNNs trainiert. In diesen Experimenten wurde vor Allem untersucht, ob sich Pooling-Layer (vgl. Abschnitt 2.1.6) durch Convolutional-Layer ersetzen lassen (vgl. Abschnitt 2.1.6 und Abschnitt 3.1.2). Die CNNs haben hier neun oder zwölf Convolutional-Layer und wurden mit 1070 oder 6671 Klassen trainiert. Zudem wurde untersucht ob zwischen dem letzten Convolutional-Layer

und der Softmax-Klassifizierung zusätzliche Dense-Layer erforderlich sind und wie sich unterschiedliche Anzahlen an Trainingsepochen auf die Ergebnisse auswirken. Die besten Ergebnisse wurden im fünften Experiment erzielt. Das CNN mit 1070 Klassen hat eine Objekterkennungsgenauigkeit von $0,77120$ und das CNN mit 6671 Klassen $0,78200$, beide haben Max-Pooling-Layer und verfügen über zusätzliche Dense-Layer. Die besten Ergebnisse für CNNs mit Convolutional-Layer anstatt Max-Pooling-Layer wurden im achten Experiment erreicht mit $0,65070$ und $0,72870$, jeweils ohne zusätzliche Dense-Layer und mit 30 Trainingsepochen mehr. Damit ist es grundsätzlich möglich Max-Pooling-Layer durch Convolutional-Layer zu ersetzen, es bedarf aber einer längeren Trainingsphase.

Im zweiten Teil der Experimente (vgl. Abschnitt 3.5) wurde lediglich zur Referenz ein Autoencoder trainiert. Ansonsten wurden nur CNNs, insgesamt zehn, trainiert. Die CNNs wurden alle blockweise trainiert. Untersucht wurden unterschiedliche Parameter des `ImageDataGenerator` von Keras, unterschiedliche Anzahlen an Convolutional Filter und zwei verschiedene Netzarchitekturen. Die CNNs haben hier elf, 13 oder 15 Convolutional Layer und 1286 Klassen. Die beste Objekterkennungsgenauigkeit wurde im 13. Experiment mit $0,68170$ erreicht und ist somit schlechter als im ersten Teil. Aber es konnte gezeigt werden, dass durch die zusätzlichen Transformationen des `ImageDataGenerator` die Netze bessere/robustere Features lernen und dadurch eine besseren Objekterkennung erreicht wird.

Die Experimente im dritten und letzten Teil (vgl. Abschnitt 3.6) stehen unter dem Motto *Going Deeper*. Es wurden unterschiedliche Netzarchitekturen untersucht, die es ermöglichen, tiefere Netze Ende-zu-Ende zu trainieren. Es wurde zunächst zur Referenz ein CNN mit 13 Convolutional Layer blockweise trainiert, anschließend wurde ein ähnliches Netz Ende-zu-Ende trainiert. Es folgen zwei Experimente mit ResNets (34 Convolutional Layer) und ein Experiment mit Densely-Connected CNNs (69 Convolutional Layer). Es wurden jeweils zwei Netze pro Experiment trainiert, einmal mit 1286 und einmal mit 6465 Klassen. Die beste Objekterkennungsgenauigkeit wurde im 21. Experiment mit $0,92362$ für das CNN mit 6465 Klassen erreicht, dass 13 Convolutional-Layer hat und Ende-zu-Ende trainiert wurde. Dies ist auch die beste Objekterkennungsgenauigkeit aller Experimente. Die tieferen Netze liefern vergleichbare Ergebnisse mit denen anderer Experimente, brauchen aber bis zu sieben Tagen für das Training.

Ziel dieser Arbeit war es, eine Objekterkennung für ägyptische Hieroglyphen zu entwickeln. Dies wurde mittels mehrerer CNNs erreicht.

4.2. Ausblick

Die in dieser Arbeit benutzte Objekterkennungsmethode ist recht unflexibel, bildet aber eine gute Grundlage für weitergehende Verfahren. Eine Hieroglyphe muss na-

hezu komplett im Eingabebild des CNNs liegen, damit sie richtig erkannt werden kann. Bei einem Schiebefensteransatz (engl. sliding window) ist es dadurch schwierig, herauszufinden, ob dieselbe Hieroglyphe mehrfach erkannt wurde, oder ob sie mehrfach vorkommt. Aus diesem Grund sollten Verfahren, die auf Basis von rekurrenten neuronalen Netzen (RNN) arbeiten, näher untersucht werden. Ebenfalls sind reale Sequenzen mit Hieroglyphen komplexer, da diese in-, über- und untereinander angeordnet sein können.

Hinsichtlich des Ziels der Erkennung von Hieroglyphen aus den vorliegenden digitalisierten Büchern der frühen Ägyptologie um 1900, bildet dies eine wichtige Vorarbeit und Grundlage, auf der aufgebaut werden kann. Besonders wenn es um die Erkennung der Hieroglyphen in den Büchern geht, ist ein Ansatz mit RNNs zu bevorzugen, da diese in der Regel auf realen Sequenzen trainiert werden und dadurch auch ein Stück weit ein Sprachmodell mit lernen. Mit einer erfolgreichen Erkennung können auch komplexere neuronale Netze trainiert werden, für Aufgaben wie Sprachverständnis und automatische Übersetzung.

Bildquellenverzeichnis

[1] Campus Inform. *Universitaet Leipzig*. 11. Juni 2015. URL: http://www.campusinform.de/wp-content/uploads/2013/01/uni_leipzig.jpg.

Literaturquellenverzeichnis

[2] Frédéric Bastien u. a. *Theano: new features and speed improvements*. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop. 2012.

[3] James Bergstra u. a. „Theano: a CPU and GPU Math Expression Compiler“. In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. Austin, TX, Juni 2010.

[4] François Chollet. *Keras*. https://github.com/fchollet/keras. 2015.

[5] Michael Copeland. *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?* Aug. 2016. URL: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

[6] Yann N. Dauphin u. a. „Identifying and attacking the saddle point problem in high-dimensional non-convex optimization“. In: *Advances in Neural Information Processing Systems 27*. Hrsg. von Z. Ghahramani u. a. Curran Associates, Inc., 2014, S. 2933–2941. URL: http://papers.nips.cc/paper/5486-identifying-and-attacking-the-saddle-point-problem-in-high-dimensional-non-convex-optimization.pdf.

[7] John Duchi, Elad Hazan und Yoram Singer. „Adaptive subgradient methods for online learning and stochastic optimization“. In: *Journal of Machine Learning Research* 12.Jul (Juli 2011), S. 2121–2159.

[8] Ian J. Goodfellow u. a. „Pylearn2: a machine learning research library“. In: *arXiv preprint arXiv:1308.4214* (2013). URL: http://arxiv.org/abs/1308.4214.

[9] S.S. Haykin. *Neural Networks: A Comprehensive Foundation*. International edition. Prentice Hall, 1999. ISBN: 9780132733502. URL: https://books.google.de/books?id=bX4pAQAAMAAJ.

[10] Kaiming He u. a. „Deep Residual Learning for Image Recognition“. In: *ArXiv e-prints* (Dez. 2015). arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[11] Kaiming He u. a. „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification“. In: *ArXiv e-prints* (Feb. 2015). arXiv: 1502.01852 [cs.CV]. URL: https://arxiv.org/abs/1502.01852.

[12] Kaiming He u. a. „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification“. In: *CoRR* abs/1502.01852 (2015). URL: http://arxiv.org/abs/1502.01852.

[13] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[14] Sepp Hochreiter. „Untersuchungen zu dynamischen neuronalen Netzen“. In: *Diploma, Technische Universität München* (1991), S. 91.

[15] Gao Huang, Zhuang Liu und Kilian Q. Weinberger. „Densely Connected Convolutional Networks“. In: *ArXiv e-prints* (Aug. 2016). arXiv: 1608.06993 [cs.CV]. URL: https://arxiv.org/abs/1608.06993.

[16] Hal Daumé III. *A Course in Machine Learning*. 2015.

[17] Sergey Ioffe und Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *ArXiv e-prints* (Feb. 2015). arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[18] Xiaojie Jin u. a. „Deep Learning with S-shaped Rectified Linear Activation Units“. In: *CoRR* abs/1512.07030 (2015). URL: http://arxiv.org/abs/1512.07030.

[19] Diederik P. Kingma und Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *CoRR* abs/1412.6980 (Dez. 2014). arXiv: 1412.6980 [cs.LG]. URL: http://arxiv.org/abs/1412.6980.

[20] Alex Krizhevsky und Geoffrey Hinton. „Learning multiple layers of features from tiny images“. In: (2009).

[21] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Advances in Neural Information Processing Systems 25*. Hrsg. von F. Pereira u. a. Curran Associates, Inc., 2012, S. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-w.

[22] Yann LeCun u. a. „Backpropagation applied to handwritten zip code recognition“. In: *Neural computation* 1.4 (1989), S. 541–551.

[23] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[24] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze u. a. *Introduction to information retrieval*. Bd. 1. Cambridge university press Cambridge, 2008.

[25] WarrenS. McCulloch und Walter Pitts. „A logical calculus of the ideas immanent in nervous activity“. English. In: *The bulletin of mathematical biophysics* 5.4 (1943), S. 115–133. ISSN: 0007-4985. DOI: 10.1007/BF02478259. URL: http://dx.doi.org/10.1007/BF02478259.

[26] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. „Learning representations by back-propagating errors“. In: *Cognitive modeling* 5.3 (1988), S. 1.

[27] Karen Simonyan und Andrew Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition“. In: *ArXiv e-prints* (Sep. 2014). arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[28] Jost Tobias Springenberg u. a. „Striving for Simplicity: The All Convolutional Net“. In: *ArXiv e-prints* (Dez. 2014). arXiv: 1412.6806 [cs.LG]. URL: https://arxiv.org/abs/1412.6806.

[29] Nitish Srivastava u. a. „Dropout: A simple way to prevent neural networks from overfitting“. In: *The Journal of Machine Learning Research* 15.1 (2014), S. 1929–1958.

[30] Christian Szegedy u. a. „Going Deeper with Convolutions“. In: *ArXiv e-prints* (Sep. 2014). arXiv: 1409.4842 [cs.CV]. URL: https://arxiv.org/abs/1409.4842.

[31] Theano. *Ops for neural networks*. Aug. 2016. URL: http://deeplearning.net/software/theano/library/tensor/nnet/nnet.html%5C#theano.tensor.nnet.nnet.categorical_crossentropy.

[32] Nicolas Usunier u. a. „Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks“. In: *ArXiv e-prints* (Sep. 2016). arXiv: 1609.02993 [cs.AI]. URL: https://arxiv.org/abs/1609.02993.

[33] Curtis VanWerkhoven. „Performance assessment of short-term hydrological forecasts in small, coastal watersheds with complex terrain using fully-distributed hydrological and meteorological models“. Diss. University of British Columbia, 2015. DOI: http://dx.doi.org/10.14288/1.0166617. URL: https://open.library.ubc.ca/cIRcle/collections/24/items/1.0166617.

[34] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. Bd. 1. John Wiley & Sons, 1994.

[35] Kôsaku Yoshida. *Functional analysis* . Berlin: Springer, 1965, XI, 458 S. URL: https://katalog.ub.uni-leipzig.de/Record/0000111773.

[36] Matthew D. Zeiler und Rob Fergus. „Visualizing and Understanding Convolutional Networks“. In: *ArXiv e-prints* (Nov. 2013). arXiv: 1311.2901 [cs.CV]. URL: https://arxiv.org/abs/1311.2901v3.

A. YAML-Konfiguration eines CNN in Pylearn

Listing A.1: YAML-Konfiguration des CNN

```
!obj:pylearn2.train.Train {
    dataset: &train !obj:pylearn2.datasets.hieroglyph.Hieroglyph {},
    model: !obj:pylearn2.models.mlp.MLP {
        batch_size: %(batch_size)i,
        input_space: !obj:pylearn2.space.Conv2DSpace {
            shape: [200, 200],
            num_channels: 1
        },
        layers: [ !obj:pylearn2.models.mlp.ConvRectifiedLinear {
                    layer_name: 'h2',
                    output_channels: %(output_channels_h2)i,
                    irange: .05,
                    kernel_shape: [5, 5],
                    pool_shape: [4, 4],
                    pool_stride: [2, 2],
                    max_kernel_norm: 1.9365
                }, !obj:pylearn2.models.mlp.ConvRectifiedLinear {
                    layer_name: 'h3',
                    output_channels: %(output_channels_h3)i,
                    irange: .05,
                    kernel_shape: [5, 5],
                    pool_shape: [4, 4],
                    pool_stride: [2, 2],
                    max_kernel_norm: 1.9365
                }, !obj:pylearn2.models.mlp.ConvRectifiedLinear {
                    layer_name: 'h4',
                    output_channels: %(output_channels_h4)i,
                    irange: .05,
                    kernel_shape: [5, 5],
                    pool_shape: [4, 4],
                    pool_stride: [2, 2],
                    max_kernel_norm: 1.9365
                }, !obj:pylearn2.models.mlp.Sigmoid {
                    layer_name: 'h5',
                    dim: 400,
                    sparse_init: 15
                }, !obj:pylearn2.models.mlp.Sigmoid {
                    layer_name: 'h6',
                    dim: 400,
                    sparse_init: 15
                }, !obj:pylearn2.models.mlp.Softmax {
                    max_col_norm: 1.9365,
                    layer_name: 'y',
                    n_classes: 20,
                    istdev: .05
                }
               ],
    },
    algorithm: !obj:pylearn2.training_algorithms.sgd.SGD {
        batch_size: %(batch_size)i,
        learning_rate: .01,
        learning_rule: !obj:pylearn2.training_algorithms.learning_rule.Momentum {
            init_momentum: .5
        },
        monitoring_dataset: {
            'valid' : !obj:pylearn2.datasets.hieroglyph.Hieroglyph {},
            'test'  : !obj:pylearn2.datasets.hieroglyph.Hieroglyph {}
        },
        cost: !obj:pylearn2.costs.cost.SumOfCosts { costs: [
            !obj:pylearn2.costs.cost.MethodCost {
                method: 'cost_from_X'
```

```
62              }, !obj:pylearn2.costs.mlp.WeightDecay {
63                  coeffs: [ .00005, .00005, .00005, .00005, .00005, .00005 ]
64              }
65              ]
66          },
67          termination_criterion: !obj:pylearn2.termination_criteria.EpochCounter {
68              max_epochs: %(max_epochs)i
69          },
70      },
71      extensions:
72          [ !obj:pylearn2.training_algorithms.learning_rule.MomentumAdjustor {
73              start: 1,
74              saturate: 10,
75              final_momentum: .99
76          }
77      ],
78      save_path: "%(save_path)s/convolutional_network_test_4.pkl",
79      save_freq: 1
80  }
```

B. Vortrainieren mittels Autoencoder

Listing B.1: Übersicht der Experimente mit Keras v0.3.3

```
1  "experiment","depth","height","width","nb_conv2d","filters","nb_pool2d","nb_dense","optimizer","activation","dataset
       ","train_set","nb_classes","nb_train","nb_epoch","loss","accuracy","runtime","test_set","nb_test","test_loss","
       test_accuracy","test_runtime","recognize_accuracy"
2  "experiment000_autoencoder_#0","9","48","48","3","[64, 128, 256]","1","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
       ","autoencoder","13980","13980","40","0.000150044939525","0.00478212924232","5361.055","test
       ","13980","4.50542730285e-05","0.0","32.398",""
3  "experiment000_autoencoder_#1","17","48","48","6","[64, 128, 256]","2","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
       ","autoencoder","13980","13980","80","0.000192449243692","0.0151376967242","16339.455","test
       ","13980","0.000113015165657","5.96089624681e-06","49.951",""
4  "experiment000_autoencoder_#2","25","48","48","9","[64, 128, 256]","3","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
       ","autoencoder","13980","13980","120","0.000419136735418","0.0194340130142","31443.965","test
       ","13980","0.000230078831205","0.000131139717846","76.189",""
5  "experiment000_cnn000_#0","10","48","48","3","[64, 128, 256]","1","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","40","4.22719756911","0.196884735212","814.634","test
       ","3210","4.29386296822","0.191900310997","4.678",""
6  "experiment000_cnn000_#1","18","48","48","6","[64, 128, 256]","2","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","80","0.451498804991","0.872585668371","2475.588","test
       ","3210","0.140494939949","0.962616830229","7.245",""
7  "experiment000_cnn000_#2","26","48","48","9","[64, 128, 256]","3","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","120","0.121365202028","0.968535840326","4634.569","test
       ","3210","0.0452346968714","0.985669790027","9.756","0.5686000000000009"
8  "experiment000_cnn001_#0","14","48","48","3","[64, 128, 256]","1","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","40","16.103031711","0.000934579418363","1015.642","test
       ","3210","16.103031711","0.000934579418363","5.491",""
9  "experiment000_cnn001_#1","22","48","48","6","[64, 128, 256]","2","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","80","0.350490002153","0.890965733387","2708.182","test
       ","3210","0.200823316403","0.940498441551","7.697",""
10 "experiment000_cnn001_#2","30","48","48","9","[64, 128, 256]","3","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","120","0.215088080742","0.932710283271","4877.513","test
       ","3210","0.0851972214907","0.97352025962","10.036","0.45599999999999935"
11 "experiment000_cnn002_#0","10","48","48","3","[64, 128, 256]","1","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","6671","13980","40","3.89403516029","0.320529329248","4665.315","test
       ","13980","3.59824315741","0.360801147055","25.41",""
12 "experiment000_cnn002_#1","18","48","48","6","[64, 128, 256]","2","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","6671","13980","80","0.312874623975","0.912374819638","11724.186","test
       ","13980","0.0732072095489","0.977753943896","34.134",""
13 "experiment000_cnn002_#2","26","48","48","9","[64, 128, 256]","3","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","6671","13980","120","0.0969958103412","0.974320467364","20745.234","test
       ","13980","0.0315125757433","0.989914172282","43.851","0.6025"
14 "experiment000_cnn003_#0","14","48","48","3","[64, 128, 256]","1","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","6671","13980","40","1.12063761569","0.673247495605","4668.144","test
       ","13980","0.729586748393","0.782188836938","25.004",""
15 "experiment000_cnn003_#1","22","48","48","6","[64, 128, 256]","2","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","6671","13980","80","0.215576569218","0.939341915268","11915.697","test
       ","13980","0.0946115188818","0.972889854417","34.296",""
16 "experiment000_cnn003_#2","30","48","48","9","[64, 128, 256]","3","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","6671","13980","120","0.126874523101","0.962446357048","21484.034","test
       ","13980","0.04048765707","0.985693857735","44.475","0.6656000000000003"
17 "experiment001_autoencoder_#0","10","48","48","4","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
       ","autoencoder","13980","13980","40","0.00395150935092","0.0726558773345","4365.731","test
       ","13980","0.000443908637541","0.169587504305","29.341",""
18 "experiment001_autoencoder_#1","19","48","48","8","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
       ","autoencoder","13980","13980","80","0.00364472015057","0.0575762998435","14456.161","test
       ","13980","0.0003524340711","0.104166664183","53.82",""
19 "experiment001_autoencoder_#2","28","48","48","12","[64, 128, 256]","0","0","adamax","SReLU","
       jsesh_bbaw_erichsen_24x24","autoencoder","13980","13980","120","0.003699654635","0.0833258828972","30381.561","
       test","13980","0.000458687660951","0.125028314231","81.283",""
20 "experiment001_cnn000_#0","11","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","40","4.72963738516","0.143613706752","480.651","test
       ","3210","4.70618780677","0.1457943927","3.222",""
21 "experiment001_cnn000_#1","20","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","80","0.8110071131","0.774766351761","1574.065","test
       ","3210","0.391774341325","0.890654203676","5.715",""
22 "experiment001_cnn000_#2","29","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
       train","1070","3210","120","0.226807694244","0.929906545398","3252.966","test
       ","3210","0.0871834475975","0.973831789516","8.524","0.6006000000000004"
```

```
23 "experiment001_cnn001_#0","15","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","40","6.14530408568","0.0252336443538","614.46","test
      ","3210","6.27438923726","0.031152647743","3.831",""
24 "experiment001_cnn001_#1","24","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","80","1.60149567298","0.571028029436","1666.951","test
      ","3210","0.777236753906","0.783800622199","6.028",""
25 "experiment001_cnn001_#2","33","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","120","0.313209832365","0.905919005009","3372.831","test
      ","3210","0.114987279877","0.961370710643","8.856","0.4943000000000002"
26 "experiment001_cnn002_#0","11","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","4.37651925981","0.259942776591","3193.307","test
      ","13980","3.99750672185","0.309513592648","18.969",""
27 "experiment001_cnn002_#1","20","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","0.513493566938","0.86144491898","8027.089","test
      ","13980","0.132281554306","0.960300435558","27.476",""
28 "experiment001_cnn002_#2","29","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.157454825397","0.956652364847","15056.791","test
      ","13980","0.064155097821","0.981044361179","38.595","0.6115000000000002"
29 "experiment001_cnn003_#0","15","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","5.9415961035","0.117668096765","4411.386","test
      ","13980","5.71034723699","0.138125894631","24.078",""
30 "experiment001_cnn003_#1","24","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","0.762037506303","0.79020028394","8221.885","test
      ","13980","0.20970059329","0.933619456145","28.524",""
31 "experiment001_cnn003_#2","33","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.215543555141","0.935050073014","15138.418","test
      ","13980","0.102302036583","0.966523609789","39.38","0.4627999999999998"
32 "experiment002_autoencoder_#0","9","48","48","3","[64, 128, 256]","1","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","40","9.08217763197e-05","0.887197480949","5345.877","test
      ","13980","0.0346163925055","0.0","32.25",""
33 "experiment002_autoencoder_#1","17","48","48","6","[64, 128, 256]","2","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","80","0.000114563749155","0.929351457028","17015.803","test
      ","13980","0.0699393564411","0.0","54.255",""
34 "experiment002_autoencoder_#2","25","48","48","9","[64, 128, 256]","3","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","120","0.000367836454173","0.920231283137","31479.2","test
      ","13980","0.0488205546239","0.0","76.875",""
35 "experiment002_cnn000_#0","10","48","48","3","[64, 128, 256]","1","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","40","3.81763453201","0.257320871513","817.637","test
      ","3210","12.2360212558","0.000934579418363","4.662",""
36 "experiment002_cnn000_#1","18","48","48","6","[64, 128, 256]","2","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","80","0.437682655556","0.874454830294","2491.536","test
      ","3210","9.52865040265","0.0208722736768","7.261",""
37 "experiment002_cnn000_#2","26","48","48","9","[64, 128, 256]","3","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","120","0.106241896093","0.966043616381","4649.03","test
      ","3210","5.25139083743","0.197819315524","9.807","0.2035999999999992"
38 "experiment002_cnn001_#0","14","48","48","3","[64, 128, 256]","1","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","40","2.13073336224","0.422429908102","1017.924","test
      ","3210","16.0146582773","0.000934579418363","5.491",""
39 "experiment002_cnn001_#1","22","48","48","6","[64, 128, 256]","2","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","80","0.470671585973","0.845794391966","2722.432","test
      ","3210","2.27628843844","0.522118377537","7.753",""
40 "experiment002_cnn001_#2","30","48","48","9","[64, 128, 256]","3","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","120","0.187986148388","0.941121499115","4914.112","test
      ","3210","9.74315019411","0.0912772582327","10.126","0.1124999999999968"
41 "experiment002_cnn002_#0","10","48","48","3","[64, 128, 256]","1","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","3.8114314799","0.335264665886","4658.754","test
      ","13980","14.1597321951","0.000429184539763","25.184",""
42 "experiment002_cnn002_#1","18","48","48","6","[64, 128, 256]","2","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","0.365019826674","0.899284692622","11609.956","test
      ","13980","11.471133887","0.0176680969402","34.183",""
43 "experiment002_cnn002_#2","26","48","48","9","[64, 128, 256]","3","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.0983248672292","0.973319034379","20961.011","test
      ","13980","9.50991042796","0.0581545063903","44.319","0.0964999999999998"
44 "experiment002_cnn003_#0","14","48","48","3","[64, 128, 256]","1","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","0.980848661479","0.704434906208","4632.089","test
      ","13980","15.8464117719","0.00100143059278","24.844",""
45 "experiment002_cnn003_#1","22","48","48","6","[64, 128, 256]","2","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","0.20788891395","0.937267528668","11830.214","test
      ","13980","8.22404580737","0.152288984513","34.581",""
46 "experiment002_cnn003_#2","30","48","48","9","[64, 128, 256]","3","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.119951710534","0.962446355769","21672.013","test
      ","13980","11.3773503979","0.043991415936","44.968","0.07570000000000014"
47 "experiment003_autoencoder_#0","10","48","48","4","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","40","0.00614503616979","0.737593887275","4315.532","test
      ","13980","0.0524653652963","0.0","28.799",""
48 "experiment003_autoencoder_#1","19","48","48","8","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","80","0.00612648759232","0.827689855822","14208.946","test
      ","13980","0.0535753508057","0.0","52.631",""
49 "experiment003_autoencoder_#2","28","48","48","12","[64, 128, 256]","0","0","adamax","SReLU","
      jsesh_bbaw_erichsen_24x24","autoencoder","13980","13980","120","0.00614287264216","0.830649442854","29924.703","
      test","13980","0.0539812985088","0.0","79.723",""
50 "experiment003_cnn000_#0","11","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","40","4.04737313886","0.214018692735","478.781","test
```

```
      ","3210","11.8441280205","0.000934579418363","3.16",""
51 "experiment003_cnn000_#1","20","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","80","0.659796605229","0.814018684199","1560.234","test
      ","3210","12.4717053981","0.00155763236394","5.595",""
52 "experiment003_cnn000_#2","29","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","120","0.254352525526","0.923676014689","3221.65","test
      ","3210","14.5158247665","0.00872274123805","8.334","0.008899999999999986"
53 "experiment003_cnn001_#0","15","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","40","4.51133134432","0.149221184005","610.238","test
      ","3210","12.1103121321","0.000934579418363","3.752",""
54 "experiment003_cnn001_#1","24","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","80","0.919418977614","0.737694704273","1649.406","test
      ","3210","14.5246884771","0.00186915883673","5.902",""
55 "experiment003_cnn001_#2","33","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","120","0.268980520444","0.91619937665","3332.696","test
      ","3210","15.2546004952","0.0024922117823","8.675","0.004100000000000001"
56 "experiment003_cnn002_#0","11","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","3.91251682314","0.309155939837","3172.723","test
      ","13980","12.9095380494","0.000858369079526","18.691",""
57 "experiment003_cnn002_#1","20","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","0.453009126695","0.86981402037","7935.535","test
      ","13980","14.2530938914","0.00586552204343","26.889",""
58 "experiment003_cnn002_#2","29","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.155220811584","0.957510732615","14916.467","test
      ","13980","11.1100593984","0.037052932298","37.777","0.07660000000000006"
59 "experiment003_cnn003_#0","15","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","3.84711190696","0.309871246481","4377.13","test
      ","13980","13.7193870695","0.000357653783136","23.711",""
60 "experiment003_cnn003_#1","24","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","0.454150456717","0.867381973754","8103.173","test
      ","13980","15.1089325497","0.00128755361929","27.843",""
61 "experiment003_cnn003_#2","33","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.163941365082","0.950214594645","14956.991","test
      ","13980","13.8707849915","0.00972818291995","38.599","0.034700000000000154"
62 "experiment004_autoencoder_#0","10","48","48","3","[64, 128, 256]","1","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","40","0.00155609118145","0.00723950886123","5387.091","test
      ","13980","0.000659289255952","0.0","32.56",""
63 "experiment004_autoencoder_#1","18","48","48","6","[64, 128, 256]","2","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","80","0.00133549790957","0.0111081306772","16978.744","test
      ","13980","0.00057753969253","6.2589410915e-05","54.741",""
64 "experiment004_autoencoder_#2","26","48","48","9","[64, 128, 256]","3","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","120","0.00147525869105","0.0151809132726","32966.986","test
      ","13980","0.000782686352293","0.125973114043","73.694",""
65 "experiment004_cnn000_#0","11","48","48","3","[64, 128, 256]","1","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","40","5.47926641773","0.0669781926938","817.079","test
      ","3210","5.5599254522","0.0601246104782","4.692",""
66 "experiment004_cnn000_#1","19","48","48","6","[64, 128, 256]","2","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","80","0.987294145277","0.728037378313","2518.845","test
      ","3210","0.483391500615","0.864797504334","7.283",""
67 "experiment004_cnn000_#2","27","48","48","9","[64, 128, 256]","3","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","120","0.279835872097","0.914641747408","4682.1","test
      ","3210","0.0673320177345","0.977881631747","9.886","0.6100999999999993"
68 "experiment004_cnn001_#0","15","48","48","3","[64, 128, 256]","1","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","40","16.103031711","0.000934579418363","1027.906","test
      ","3210","16.1030317408","0.000934579418363","5.531",""
69 "experiment004_cnn001_#1","23","48","48","6","[64, 128, 256]","2","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","80","0.700430626438","0.791900306662","2750.114","test
      ","3210","0.334323695005","0.893457943034","7.758",""
70 "experiment004_cnn001_#2","31","48","48","9","[64, 128, 256]","3","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","1070","3210","120","0.331672110429","0.894080993542","4893.514","test
      ","3210","0.1175804614","0.959813087157","10.172","0.7711999999999962"
71 "experiment004_cnn002_#0","11","48","48","3","[64, 128, 256]","1","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","6.74720289233","0.051430614937","4666.07","test
      ","13980","6.57062672068","0.0638054362998","25.374",""
72 "experiment004_cnn002_#1","19","48","48","6","[64, 128, 256]","2","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","1.00613477975","0.732117308823","11974.395","test
      ","13980","0.258208845061","0.918240342955","34.335",""
73 "experiment004_cnn002_#2","27","48","48","9","[64, 128, 256]","3","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.389031450146","0.884120171503","20276.247","test
      ","13980","0.0899955091814","0.96924177942","44.449","0.7252999999999998"
74 "experiment004_cnn003_#0","15","48","48","3","[64, 128, 256]","1","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","40","16.1146365686","0.000214592269882","4674.212","test
      ","13980","16.1146365686","0.000214592269882","24.997",""
75 "experiment004_cnn003_#1","23","48","48","6","[64, 128, 256]","2","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","80","0.637831901532","0.810371958988","12221.057","test
      ","13980","0.219463836892","0.927253220555","34.735",""
76 "experiment004_cnn003_#2","31","48","48","9","[64, 128, 256]","3","3","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
      train","6671","13980","120","0.402099162948","0.876680972494","20892.95","test
      ","13980","0.116250562009","0.958583692624","45.184","0.7819999999999974"
77 "experiment005_autoencoder_#0","11","48","48","4","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
      ","autoencoder","13980","13980","40","0.00512945918814","0.0967960178787","4343.387","test
      ","13980","0.00114930052465","0.356154624399","29.316",""
```

```
78 "experiment005_autoencoder_#1","20","48","48","8","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
     ","autoencoder","13980","13980","80","0.004638228735","0.0943878157772","14620.747","test
     ","13980","0.00108879432315","0.211476214987","53.682",""
79 "experiment005_autoencoder_#2","29","48","48","12","[64, 128, 256]","0","0","adamax","SReLU","
     jsesh_bbaw_erichsen_24x24","autoencoder","13980","13980","120","0.00459214186","0.0828057944671","30375.23","
     test","13980","0.00119084608225","0.389547566402","81.001",""
80 "experiment005_cnn000_#0","12","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","40","4.42973936978","0.164174456091","483.134","test
     ","3210","4.49053947948","0.155763239875","3.225",""
81 "experiment005_cnn000_#1","21","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","80","1.16389108484","0.690342684215","1589.058","test
     ","3210","0.485655903004","0.855451708271","5.711",""
82 "experiment005_cnn000_#2","30","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","120","0.448523918695","0.865732082882","3257.514","test
     ","3210","0.105839517722","0.966355149991","8.483","0.5959000000000004"
83 "experiment005_cnn001_#0","16","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","40","5.31145291016","0.075700934338","617.263","test
     ","3210","5.496419663","0.0601246103505","3.827",""
84 "experiment005_cnn001_#1","25","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","80","1.13263100516","0.686292830843","1673.103","test
     ","3210","0.478885644507","0.858878499994","6.028",""
85 "experiment005_cnn001_#2","34","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","120","0.465289286642","0.852647974112","3372.162","test
     ","3210","0.168037715893","0.944548285639","8.829","0.49270000000000014"
86 "experiment005_cnn002_#0","12","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","40","5.95898761736","0.101931330342","3204.073","test
     ","13980","5.65017237411","0.122889842913","18.953",""
87 "experiment005_cnn002_#1","21","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","80","1.13144658563","0.700929901631","8009.9","test
     ","13980","0.261522955819","0.915236054487","27.454",""
88 "experiment005_cnn002_#2","30","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","120","0.570333758606","0.833690986101","15061.935","test
     ","13980","0.153312140617","0.948998569504","38.444","0.6166000000000003"
89 "experiment005_cnn003_#0","16","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","40","6.09332626572","0.0944921317269","4409.81","test
     ","13980","5.9026386482","0.108226036957","24.096",""
90 "experiment005_cnn003_#1","25","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","80","1.11377126047","0.701788269484","8196.355","test
     ","13980","0.300846978468","0.903361944596","28.454",""
91 "experiment005_cnn003_#2","34","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","120","0.476174889567","0.85600858219","15122.154","test
     ","13980","0.122401894242","0.958941349912","39.336","0.5653999999999998"
92 "experiment006_autoencoder_#0","11","48","48","4","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
     ","autoencoder","13980","13980","50","0.00505113574171","0.0715933477331","5426.008","test
     ","13980","0.00107894330059","0.0227810564332","29.318",""
93 "experiment006_autoencoder_#1","20","48","48","8","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
     ","autoencoder","13980","13980","100","0.00452600852307","0.0937961971286","18250.024","test
     ","13980","0.00104679579629","0.258179840271","53.607",""
94 "experiment006_autoencoder_#2","29","48","48","12","[64, 128, 256]","0","0","adamax","SReLU","
     jsesh_bbaw_erichsen_24x24","autoencoder
     ","13980","13980","150","0.00455751587225","0.0898500837211","37954.532","test
     ","13980","0.00116266662675","0.376143007618","80.945",""
95 "experiment006_cnn000_#0","12","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","50","4.09233309844","0.197196260466","602.964","test
     ","3210","4.20847351306","0.182866044226","3.221",""
96 "experiment006_cnn000_#1","21","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","100","0.939379169934","0.728971965216","1983.415","test
     ","3210","0.485682034534","0.854517133436","5.703",""
97 "experiment006_cnn000_#2","30","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","150","0.35207693497","0.895638626127","4064.723","test
     ","3210","0.0974318864654","0.96947041516","8.484","0.6048000000000001"
98 "experiment006_cnn001_#0","16","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","50","4.62724759943","0.12616822481","770.604","test
     ","3210","4.60843712667","0.131464174325","3.816",""
99 "experiment006_cnn001_#1","25","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","100","1.05864761329","0.69750778456","2090.459","test
     ","3210","0.397061407195","0.875077881546","6.022",""
100 "experiment006_cnn001_#2","34","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","1070","3210","150","0.414448709781","0.870404982493","4218.62","test
     ","3210","0.0961765813354","0.965420562159","8.812","0.5353000000000017"
101 "experiment006_cnn002_#0","12","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","50","5.12017125565","0.166523605532","4002.993","test
     ","13980","4.91573327123","0.177825465329","18.938",""
102 "experiment006_cnn002_#1","21","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","100","0.991196729339","0.732188840707","9999.113","test
     ","13980","0.241537359662","0.919027184809","27.419",""
103 "experiment006_cnn002_#2","30","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","150","0.408690334176","0.882474965548","18800.802","test
     ","13980","0.125814708054","0.957296140204","38.35","0.7145999999999986"
104 "experiment006_cnn003_#0","16","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
     train","6671","13980","50","5.31949265252","0.147925608212","5514.06","test
     ","13980","5.02985663748","0.169957082266","24.033",""
```

```
105 "experiment006_cnn003_#1","25","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","100","0.902032976669","0.752789702965","10236.107","test
        ","13980","0.247883903121","0.921316165354","28.422",""
106 "experiment006_cnn003_#2","34","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","150","0.431946590905","0.866452073164","18870.58","test
        ","13980","0.115451868448","0.960085839673","39.282","0.6398999999999989"
107 "experiment007_autoencoder_#0","11","48","48","4","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
        ","autoencoder","13980","13980","60","0.00503208314885","0.0667471388591","6413.773","test
        ","13980","0.00101991130636","0.0","28.942",""
108 "experiment007_autoencoder_#1","20","48","48","8","[64, 128, 256]","0","0","adamax","SReLU","jsesh_bbaw_erichsen_24x24
        ","autoencoder","13980","13980","120","0.00452202928182","0.106294706851","21565.093","test
        ","13980","0.000931807396799","0.0","52.901",""
109 "experiment007_autoencoder_#2","29","48","48","12","[64, 128, 256]","0","0","adamax","SReLU","
        jsesh_bbaw_erichsen_24x24","autoencoder
        ","13980","13980","180","0.00444977888031","0.0874761568963","44830.433","test
        ","13980","0.000973594274694","0.0668779715752","79.934",""
110 "experiment007_cnn000_#0","12","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","1070","3210","60","3.60971389307","0.261370717458","712.591","test
        ","3210","3.6959791748","0.240809968815","3.18",""
111 "experiment007_cnn000_#1","21","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","1070","3210","120","0.798376292454","0.780062301879","2345.812","test
        ","3210","0.345581887203","0.896261678492","5.627",""
112 "experiment007_cnn000_#2","30","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","1070","3210","180","0.330556360175","0.900000009396","4808.347","test
        ","3210","0.0976522088028","0.965732093541","8.375","0.650699999999999"
113 "experiment007_cnn001_#0","16","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","1070","3210","60","4.23924324","0.175700935062","912.399","test
        ","3210","4.62831913274","0.122741432669","3.772",""
114 "experiment007_cnn001_#1","25","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","1070","3210","120","0.937660516608","0.732710283122","2469.607","test
        ","3210","0.547930989217","0.834267909282","5.937",""
115 "experiment007_cnn001_#2","34","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","1070","3210","180","0.360130604554","0.888473519841","4989.949","test
        ","3210","0.0781833873083","0.974143310128","8.707","0.5657000000000005"
116 "experiment007_cnn002_#0","12","48","48","4","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","60","4.35503203606","0.238483547845","4731.36","test
        ","13980","3.83217666316","0.296494993913","18.675",""
117 "experiment007_cnn002_#1","21","48","48","8","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","120","0.866388859155","0.765164520352","11820.373","test
        ","13980","0.20493583751","0.932618027081","27.055",""
118 "experiment007_cnn002_#2","30","48","48","12","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","180","0.416976468715","0.875679544625","22245.906","test
        ","13980","0.0852751500627","0.970886989639","37.96","0.7286999999999971"
119 "experiment007_cnn003_#0","16","48","48","6","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","60","4.45426133467","0.223390558044","6512.299","test
        ","13980","4.04379777847","0.264663806289","23.801",""
120 "experiment007_cnn003_#1","25","48","48","10","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","120","0.839011384054","0.769027181023","12098.913","test
        ","13980","0.203187605976","0.933047211136","28.015",""
121 "experiment007_cnn003_#2","34","48","48","14","[64, 128, 256]","0","1","adamax","SReLU","jsesh_bbaw_erichsen_24x24","
        train","6671","13980","180","0.392943708426","0.87775393293","22335.633","test
        ","13980","0.0944691445167","0.96745351055","38.823","0.6429999999999989"
```

C. Ergebnisse der Experimente mit Keras v1.0.3

Listing C.1: Übersicht der Experimente mit Keras v1.0.3

```
1  "experiment","depth","height","width","nb_conv2d","filters","nb_pool2d","nb_dense","optimizer","activation","dataset
       ","train_set","nb_classes","nb_train","nb_epoch","loss","accuracy","runtime","test_set","nb_test","test_loss","
       test_accuracy","test_runtime","recognize_accuracy"
2  "experiment008_autoencoder","53","48","48","22","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","16788","16788","10","0.0740755922153","0.000346643098932","2087.416","train
       ","16788","0.0672749941283","0.0","73.53",""
3  "experiment009_cnn_#0","13","48","48","5","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","0.844261749987","0.793996367245","9040.049","train
       ","6063","0.674788110252","0.82005607604","32.592",""
4  "experiment009_cnn_#1","22","48","48","9","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","0.438366919564","0.869536530572","4847.247","train
       ","6063","0.149262371287","0.957941617487","18.535",""
5  "experiment009_cnn_#2","31","48","48","13","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","0.223471882186","0.92891307841","4242.494","train
       ","6063","0.0711506294013","0.974270172635","17.242","0.4998999999999995"
6  "experiment009_cnn_#3","35","48","48","15","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","0.300687774338","0.913904012901","9424.32","train
       ","6063","0.0966607417162","0.968992258512","34.956","0.4461000000000002"
7  "experiment010_cnn_#0","13","48","48","5","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.534761775254","0.868711857007","18230.733","train
       ","6063","0.356628481978","0.902853374628","33.144",""
8  "experiment010_cnn_#1","22","48","48","9","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.354037201972","0.901039089363","9907.262","train
       ","6063","0.124720366228","0.964539013089","18.959",""
9  "experiment010_cnn_#2","31","48","48","13","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.164581685294","0.948540327181","8648.568","train
       ","6063","0.0351651092579","0.987300028458","17.68","0.6006000000000002"
10 "experiment010_cnn_#3","35","48","48","15","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.236543482244","0.93963384936","19037.361","train
       ","6063","0.0961314401393","0.975094845561","35.006","0.47400000000000003"
11 "experiment011_cnn_#0","13","48","48","5","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","2.01971024526","0.514596728765","9149.118","train
       ","6063","1.41234794319","0.64357578954","32.394",""
12 "experiment011_cnn_#1","22","48","48","9","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","1.06094102067","0.711364016868","4929.223","train
       ","6063","0.288305023242","0.912089729012","18.743",""
13 "experiment011_cnn_#2","31","48","48","13","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","0.558064087299","0.830776840847","4303.422","train
       ","6063","0.112247466953","0.963384468121","17.178","0.6744999999999991"
14 "experiment011_cnn_#3","35","48","48","15","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","100","0.544357647287","0.831271651956","9550.466","train
       ","6063","0.130647054607","0.952168895352","35.404","0.5863999999999994"
15 "experiment012_cnn_#0","13","48","48","5","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.25569145762","0.683325085559","17872.464","train
       ","6063","0.669616474711","0.818241787826","32.659",""
16 "experiment012_cnn_#1","22","48","48","9","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.71799990783","0.797624938071","9785.919","train
       ","6063","0.160645540756","0.944416958714","18.597",""
17 "experiment012_cnn_#2","31","48","48","13","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.39891809433","0.878278079205","8578.308","train
       ","6063","0.0823012366799","0.972290951398","17.39","0.6816999999999991"
18 "experiment012_cnn_#3","35","48","48","15","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.382105696876","0.883226125759","18721.008","train
       ","6063","0.0830607851757","0.968497451925","34.582","0.5865000000000005"
19 "experiment013_cnn_#0","13","48","48","5","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","2.08344105103","0.502721421237","18186.414","train
       ","6063","1.06774613933","0.718621147656","32.367",""
20 "experiment013_cnn_#1","22","48","48","9","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.32305495485","0.643410853618","9776.386","train
       ","6063","0.366261457121","0.890978064756","18.295",""
```

```
21 "experiment013_cnn_#2","31","48","48","13","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.840145410606","0.761174335479","8548.146","train
       ","6063","0.150945118337","0.949694871165","17.079","0.6732000000000007"
22 "experiment013_cnn_#3","35","48","48","15","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.649508988751","0.802243112633","18950.773","train
       ","6063","0.159894121606","0.945406565449","34.859","0.6012000000000003"
23 "experiment014_cnn_#0","13","48","48","5","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","2.15911017211","0.494309745482","18111.619","train
       ","6063","1.09005748109","0.717796469323","33.353",""
24 "experiment014_cnn_#1","20","48","48","8","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.32824357551","0.642916044868","9311.13","train
       ","6063","0.460274632351","0.86293913964","17.607",""
25 "experiment014_cnn_#2","27","48","48","11","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.887510151118","0.746495137145","7782.172","train
       ","6063","0.16658969141","0.944746829379","15.657","0.6468999999999996"
26 "experiment014_cnn_#3","31","48","48","13","[64, 64, 64]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.717291093474","0.788058712973","18051.495","train
       ","6063","0.159944287824","0.940788390248","33.389","0.6343999999999993"
27 "experiment015_cnn_#0","12","48","48","5","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.71043581094","0.589807025556","36855.797","train
       ","6063","0.79733793683","0.783770408437","67.291",""
28 "experiment015_cnn_#1","20","48","48","9","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.14263506499","0.690087414871","20486.403","train
       ","6063","0.342648088932","0.889493652417","37.283",""
29 "experiment015_cnn_#2","28","48","48","13","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.640941105521","0.810654791485","17287.735","train
       ","6063","0.168021524882","0.943427349128","33.294","0.6624999999999999"
30 "experiment015_cnn_#3","32","48","48","15","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.637065332852","0.815272966342","30748.309","train
       ","6063","0.219577792435","0.92726373064","57.254","0.5576000000000009"
31 "experiment016_cnn_#0","13","48","48","5","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","2.00500703161","0.536203191645","33925.548","train
       ","6063","1.00405187424","0.735444500888","60.243",""
32 "experiment016_cnn_#1","20","48","48","8","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.38917076462","0.635493982139","17231.624","train
       ","6063","0.318182121219","0.899059868716","30.917",""
33 "experiment016_cnn_#2","27","48","48","11","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.934687757205","0.739732806161","13732.923","train
       ","6063","0.174041244735","0.9429325449","26.522","0.6468999999999994"
34 "experiment016_cnn_#3","31","48","48","13","[128, 128, 128]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.750364845919","0.769915886962","26159.515","train
       ","6063","0.151819072282","0.946561109286","47.88","0.6160000000000004"
35 "experiment017_cnn_#0","12","48","48","5","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","2.4060473934","0.431964371","10792.631","train
       ","6063","1.52237732741","0.607455054516","23.632",""
36 "experiment017_cnn_#1","20","48","48","9","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.24348371172","0.65726538335","6150.829","train
       ","6063","0.470367979875","0.859145637463","13.105",""
37 "experiment017_cnn_#2","28","48","48","13","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.698213484624","0.785584690998","5904.311","train
       ","6063","0.150082586716","0.949529938831","13.729","0.6412999999999995"
38 "experiment017_cnn_#3","32","48","48","15","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.613830178588","0.815932707869","15849.793","train
       ","6063","0.211656042877","0.930232561315","31.889","0.5609000000000007"
39 "experiment018_cnn_#0","13","48","48","5","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","2.72803256993","0.381659243591","10185.392","train
       ","6063","1.60661057197","0.607949858498","20.968",""
40 "experiment018_cnn_#1","20","48","48","8","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.67803978639","0.569684971181","5451.926","train
       ","6063","0.58883316125","0.843476828659","12.002",""
41 "experiment018_cnn_#2","27","48","48","11","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","1.28173478672","0.644565396423","5026.025","train
       ","6063","0.274628865571","0.919181923386","12.194","0.5979999999999996"
42 "experiment018_cnn_#3","31","48","48","13","[32, 32, 32]","0","0","adam","SReLU","hieroglyphs","train
       ","1286","6063","200","0.826579807036","0.751773051169","14346.669","train
       ","6063","0.146521987598","0.951179285766","28.343","0.5865000000000005"
```

D. Going Deeper

Listing D.1: Übersicht der Experimente mit Keras v1.0.7

```
1  "experiment","keras","input_shape","activation","optimizer","loss","depth","nb_conv2d","filters","dataset","train_set
       ","nb_classes","nb_train","nb_epoch","train_loss","train_accuracy","val_loss","val_accuracy","train_runtime","
       hieroglyphs_validation3pc_sequences_recognize_accuracy","hieroglyphs_validation2pc_sequences_recognize_accuracy"
2  "experiment000_cnn1286_#0","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","13","5","[64, 128, 256,
       512]","hieroglyphs","train3pc_32x32
       ","1286","6063","100","1.01747552901","0.746000329162","0.264328198255","0.932211776112","5705.784",
       "0.7093499999999942",""
3  "experiment000_cnn1286_#1","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","22","9","[64, 128, 256,
       512]","hieroglyphs","train3pc_32x32
       ","1286","6063","100","0.376319226305","0.893452085571","0.0580055720461","0.979877947876","4525.537",
       "0.7873199999999828",""
4  "experiment000_cnn1286_#2","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","31","13","[64, 128, 256,
       512]","hieroglyphs","train3pc_32x32
       ","1286","6063","100","0.298141779929","0.908296222471","0.0398862520904","0.986805211607","4592.919",
       "0.7879699999999834",""
5  "experiment000_cnn6465_#0","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","13","5","[64, 128, 256,
       512]","hieroglyphs","train2pc_32x32
       ","6465","16421","100","0.916955695976","0.770720419034","0.15115594913","0.955910115035","50973.925",
       "0.7419699999999881","0.7529099999999875"
6  "experiment000_cnn6465_#1","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","22","9","[64, 128, 256,
       512]","hieroglyphs","train2pc_32x32
       ","6465","16421","100","0.404004713917","0.886486815637","0.0621250859517","0.980269167529","29739.056",
       "0.8315399999999737","0.8425199999999667"
7  "experiment000_cnn6465_#2","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","31","13","[64, 128, 256,
       512]","hieroglyphs","train2pc_32x32
       ","6465","16421","100","0.439304830771","0.871627793664","0.0888928584115","0.97149990869","21037.49",
       "0.7612099999999891","0.7725699999999812"
8  "experiment001_cnn1286","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","41","13","[64, 128, 256,
       512]","hieroglyphs","train3pc_32x32
       ","1286","6063","1240","0.0219976373444","0.991753256795","0.00908206677798","0.995546758696","73558.183",
       "0.9094799999999581",""
9  "experiment001_cnn6465","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","41","13","[64, 128, 256,
       512]","hieroglyphs","train2pc_32x32
       ","6465","16421","339","0.0349053132041","0.990804457645","0.00976578298153","0.995919858717","85829.124",
       "0.9125699999999549","0.9236199999999563"
10 "experiment002_cnn1286","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","123","34","[64, 128, 256,
       512]","hieroglyphs","train3pc_32x32
       ","1286","6063","2109","0.0292009567126","0.988124690413","0.00974589052903","0.995051954478","461378.293",
       "0.6595299999999966",""
11 "experiment002_cnn6465","1.0.7","(3, 32, 32)","SReLU","adam","categorical_crossentropy","123","34","[64, 128, 256,
       512]","hieroglyphs","train2pc_32x32
       ","6465","16421","920","0.0423275728143","0.9866025211","0.0088138692663","0.995919858717","602606.224",
       "0.6358499999999998","0.6271999999999991"
12 "experiment003_cnn1286","1.0.8","(3, 32, 32)","SReLU","adam","categorical_crossentropy","129","34","[64, 128, 256,
       512]","hieroglyphs","train3pc_32x32
       ","1286","6063","1018","0.0254997088407","0.989114299852","0.00952104283504","0.995051954478","178760.049","
       0.7918299999999842",""
13 "experiment003_cnn6465","1.0.8","(3, 32, 32)","SReLU","adam","categorical_crossentropy","129","34","[64, 128, 256,
       512]","hieroglyphs","train2pc_32x32
       ","6465","16421","316","0.0437101632507","0.987759576127","0.00976421502773","0.996407039766","205663.492",
       "0.7597599999999851","0.7779199999999847"
14 "experiment004_cnn1286","1.0.8","(3, 32, 32)","SReLU","adam","categorical_crossentropy","279","69","[16, 16, 16,
       16]","hieroglyphs","train3pc_32x32
       ","1286","6063","1355","0.0251062898466","0.990103908976","0.00918873162796","0.995876628732","576350.786",
       "0.7827199999999813",""
15 "experiment004_cnn6465","1.1.0","(3, 32, 32)","SReLU","adam","categorical_crossentropy","279","69","[16, 16, 16,
       16]","hieroglyphs","train2pc_32x32
       ","6465","16421","403","0.0470374657355","0.984349308812","0.0147787746812","0.995006394251","514061.597",
       "0.7722899999999844","0.7909399999999803"
```

Listing D.2: JSON-Konfiguration des CNN 6465 des 21. Experiments

```
{"config": {"input_layers": [["input", 0, 0]], "layers": [
    {"inbound_nodes": [], "name": "input", "config": {"batch_input_shape": [null, 3, 32, 32], "name": "input", "
        input_dtype": "float32"}, "class_name": "InputLayer"},
    {"inbound_nodes": [[["input", 0, 0]]], "name": "convolution2d0_0", "config": {"init": "glorot_uniform", "nb_filter
        ": 64, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "bias":
        true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "activity_regularizer":
        null, "nb_row": 3, "name": "convolution2d0_0", "b_constraint": null, "W_regularizer": null}, "class_name": "
        Convolution2D"},
    {"inbound_nodes": [[["convolution2d0_0", 0, 0]]], "name": "batchnormalization_14", "config": {"epsilon": 1e-06, "
        axis": 1, "name": "batchnormalization_14", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
        BatchNormalization"},
    {"inbound_nodes": [[["batchnormalization_14", 0, 0]]], "name": "srelu0_0", "config": {"t_left_init": "zero", "
        trainable": true, "a_left_init": "glorot_uniform", "name": "srelu0_0", "t_right_init": "glorot_uniform", "
        a_right_init": "one"}, "class_name": "SReLU"},
    {"inbound_nodes": [[["srelu0_0", 0, 0]]], "name": "convolution2d0_1", "config": {"init": "glorot_uniform", "
        nb_filter": 64, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
        bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
        activity_regularizer": null, "nb_row": 3, "name": "convolution2d0_1", "b_constraint": null, "W_regularizer":
         null}, "class_name": "Convolution2D"},
    {"inbound_nodes": [[["convolution2d0_1", 0, 0]]], "name": "batchnormalization_15", "config": {"epsilon": 1e-06, "
        axis": 1, "name": "batchnormalization_15", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
        BatchNormalization"},
    {"inbound_nodes": [[["batchnormalization_15", 0, 0]]], "name": "srelu0_1", "config": {"t_left_init": "zero", "
        trainable": true, "a_left_init": "glorot_uniform", "name": "srelu0_1", "t_right_init": "glorot_uniform", "
        a_right_init": "one"}, "class_name": "SReLU"},
    {"inbound_nodes": [[["srelu0_1", 0, 0]]], "name": "convolution2d0_2", "config": {"init": "glorot_uniform", "
        nb_filter": 64, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
        bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
        activity_regularizer": null, "nb_row": 3, "name": "convolution2d0_2", "b_constraint": null, "W_regularizer":
         null}, "class_name": "Convolution2D"},
    {"inbound_nodes": [[["convolution2d0_2", 0, 0]]], "name": "batchnormalization_16", "config": {"epsilon": 1e-06, "
        axis": 1, "name": "batchnormalization_16", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
        BatchNormalization"},
    {"inbound_nodes": [[["batchnormalization_16", 0, 0]]], "name": "srelu0_2", "config": {"t_left_init": "zero", "
        trainable": true, "a_left_init": "glorot_uniform", "name": "srelu0_2", "t_right_init": "glorot_uniform", "
        a_right_init": "one"}, "class_name": "SReLU"},
    {"inbound_nodes": [[["srelu0_2", 0, 0]]], "name": "convolution2d0_3", "config": {"init": "glorot_uniform", "
        nb_filter": 64, "subsample": [2, 2], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
        bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
        activity_regularizer": null, "nb_row": 3, "name": "convolution2d0_3", "b_constraint": null, "W_regularizer":
         null}, "class_name": "Convolution2D"},
    {"inbound_nodes": [[["convolution2d0_3", 0, 0]]], "name": "batchnormalization_17", "config": {"epsilon": 1e-06, "
        axis": 1, "name": "batchnormalization_17", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
        BatchNormalization"},
    {"inbound_nodes": [[["batchnormalization_17", 0, 0]]], "name": "srelu0_3", "config": {"t_left_init": "zero", "
        trainable": true, "a_left_init": "glorot_uniform", "name": "srelu0_3", "t_right_init": "glorot_uniform", "
        a_right_init": "one"}, "class_name": "SReLU"},
    {"inbound_nodes": [[["srelu0_3", 0, 0]]], "name": "convolution2d1_0", "config": {"init": "glorot_uniform", "
        nb_filter": 64, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
        bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
        activity_regularizer": null, "nb_row": 3, "name": "convolution2d1_0", "b_constraint": null, "W_regularizer":
         null}, "class_name": "Convolution2D"},
    {"inbound_nodes": [[["convolution2d1_0", 0, 0]]], "name": "batchnormalization_18", "config": {"epsilon": 1e-06, "
        axis": 1, "name": "batchnormalization_18", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
        BatchNormalization"},
    {"inbound_nodes": [[["batchnormalization_18", 0, 0]]], "name": "srelu1_0", "config": {"t_left_init": "zero", "
        trainable": true, "a_left_init": "glorot_uniform", "name": "srelu1_0", "t_right_init": "glorot_uniform", "
        a_right_init": "one"}, "class_name": "SReLU"},
    {"inbound_nodes": [[["srelu1_0", 0, 0]]], "name": "convolution2d1_1", "config": {"init": "glorot_uniform", "
        nb_filter": 128, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
        bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
        activity_regularizer": null, "nb_row": 3, "name": "convolution2d1_1", "b_constraint": null, "W_regularizer":
         null}, "class_name": "Convolution2D"},
    {"inbound_nodes": [[["convolution2d1_1", 0, 0]]], "name": "batchnormalization_19", "config": {"epsilon": 1e-06, "
        axis": 1, "name": "batchnormalization_19", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
        BatchNormalization"},
    {"inbound_nodes": [[["batchnormalization_19", 0, 0]]], "name": "srelu1_1", "config": {"t_left_init": "zero", "
        trainable": true, "a_left_init": "glorot_uniform", "name": "srelu1_1", "t_right_init": "glorot_uniform", "
        a_right_init": "one"}, "class_name": "SReLU"},
    {"inbound_nodes": [[["srelu1_1", 0, 0]]], "name": "convolution2d1_2", "config": {"init": "glorot_uniform", "
        nb_filter": 128, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
        bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
        activity_regularizer": null, "nb_row": 3, "name": "convolution2d1_2", "b_constraint": null, "W_regularizer":
         null}, "class_name": "Convolution2D"},
    {"inbound_nodes": [[["convolution2d1_2", 0, 0]]], "name": "batchnormalization_20", "config": {"epsilon": 1e-06, "
        axis": 1, "name": "batchnormalization_20", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
        BatchNormalization"},
    {"inbound_nodes": [[["batchnormalization_20", 0, 0]]], "name": "srelu1_2", "config": {"t_left_init": "zero", "
        trainable": true, "a_left_init": "glorot_uniform", "name": "srelu1_2", "t_right_init": "glorot_uniform", "
        a_right_init": "one"}, "class_name": "SReLU"},
    {"inbound_nodes": [[["srelu1_2", 0, 0]]], "name": "convolution2d1_3", "config": {"init": "glorot_uniform", "
        nb_filter": 128, "subsample": [2, 2], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
```

```
          bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
          activity_regularizer": null, "nb_row": 3, "name": "convolution2d1_3", "b_constraint": null, "W_regularizer":
           null}, "class_name": "Convolution2D"},
25    {"inbound_nodes": [[["convolution2d1_3", 0, 0]]], "name": "batchnormalization_21", "config": {"epsilon": 1e-06, "
          axis": 1, "name": "batchnormalization_21", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
          BatchNormalization"},
26    {"inbound_nodes": [[["batchnormalization_21", 0, 0]]], "name": "srelu1_3", "config": {"t_left_init": "zero", "
          trainable": true, "a_left_init": "glorot_uniform", "name": "srelu1_3", "t_right_init": "glorot_uniform", "
          a_right_init": "one"}, "class_name": "SReLU"},
27    {"inbound_nodes": [[["srelu1_3", 0, 0]]], "name": "convolution2d2_0", "config": {"init": "glorot_uniform", "
          nb_filter": 64, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
          bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
          activity_regularizer": null, "nb_row": 3, "name": "convolution2d2_0", "b_constraint": null, "W_regularizer":
           null}, "class_name": "Convolution2D"},
28    {"inbound_nodes": [[["convolution2d2_0", 0, 0]]], "name": "batchnormalization_22", "config": {"epsilon": 1e-06, "
          axis": 1, "name": "batchnormalization_22", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
          BatchNormalization"},
29    {"inbound_nodes": [[["batchnormalization_22", 0, 0]]], "name": "srelu2_0", "config": {"t_left_init": "zero", "
          trainable": true, "a_left_init": "glorot_uniform", "name": "srelu2_0", "t_right_init": "glorot_uniform", "
          a_right_init": "one"}, "class_name": "SReLU"},
30    {"inbound_nodes": [[["srelu2_0", 0, 0]]], "name": "convolution2d2_1", "config": {"init": "glorot_uniform", "
          nb_filter": 256, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
          bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
          activity_regularizer": null, "nb_row": 3, "name": "convolution2d2_1", "b_constraint": null, "W_regularizer":
           null}, "class_name": "Convolution2D"},
31    {"inbound_nodes": [[["convolution2d2_1", 0, 0]]], "name": "batchnormalization_23", "config": {"epsilon": 1e-06, "
          axis": 1, "name": "batchnormalization_23", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
          BatchNormalization"},
32    {"inbound_nodes": [[["batchnormalization_23", 0, 0]]], "name": "srelu2_1", "config": {"t_left_init": "zero", "
          trainable": true, "a_left_init": "glorot_uniform", "name": "srelu2_1", "t_right_init": "glorot_uniform", "
          a_right_init": "one"}, "class_name": "SReLU"},
33    {"inbound_nodes": [[["srelu2_1", 0, 0]]], "name": "convolution2d2_2", "config": {"init": "glorot_uniform", "
          nb_filter": 256, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
          bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
          activity_regularizer": null, "nb_row": 3, "name": "convolution2d2_2", "b_constraint": null, "W_regularizer":
           null}, "class_name": "Convolution2D"},
34    {"inbound_nodes": [[["convolution2d2_2", 0, 0]]], "name": "batchnormalization_24", "config": {"epsilon": 1e-06, "
          axis": 1, "name": "batchnormalization_24", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
          BatchNormalization"},
35    {"inbound_nodes": [[["batchnormalization_24", 0, 0]]], "name": "srelu2_2", "config": {"t_left_init": "zero", "
          trainable": true, "a_left_init": "glorot_uniform", "name": "srelu2_2", "t_right_init": "glorot_uniform", "
          a_right_init": "one"}, "class_name": "SReLU"},
36    {"inbound_nodes": [[["srelu2_2", 0, 0]]], "name": "convolution2d2_3", "config": {"init": "glorot_uniform", "
          nb_filter": 256, "subsample": [2, 2], "W_constraint": null, "activation": "linear", "b_regularizer": null, "
          bias": true, "dim_ordering": "th", "nb_col": 3, "border_mode": "same", "trainable": true, "
          activity_regularizer": null, "nb_row": 3, "name": "convolution2d2_3", "b_constraint": null, "W_regularizer":
           null}, "class_name": "Convolution2D"},
37    {"inbound_nodes": [[["convolution2d2_3", 0, 0]]], "name": "batchnormalization_25", "config": {"epsilon": 1e-06, "
          axis": 1, "name": "batchnormalization_25", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
          BatchNormalization"},
38    {"inbound_nodes": [[["batchnormalization_25", 0, 0]]], "name": "srelu2_3", "config": {"t_left_init": "zero", "
          trainable": true, "a_left_init": "glorot_uniform", "name": "srelu2_3", "t_right_init": "glorot_uniform", "
          a_right_init": "one"}, "class_name": "SReLU"},
39    {"inbound_nodes": [[["srelu2_3", 0, 0]]], "name": "convolution2d3_0", "config": {"init": "glorot_uniform", "
          nb_filter": 6465, "subsample": [1, 1], "W_constraint": null, "activation": "linear", "b_regularizer": null,
          "bias": true, "dim_ordering": "th", "nb_col": 4, "border_mode": "valid", "trainable": true, "
          activity_regularizer": null, "nb_row": 4, "name": "convolution2d3_0", "b_constraint": null, "W_regularizer":
           null}, "class_name": "Convolution2D"},
40    {"inbound_nodes": [[["convolution2d3_0", 0, 0]]], "name": "batchnormalization_26", "config": {"epsilon": 1e-06, "
          axis": 1, "name": "batchnormalization_26", "trainable": true, "mode": 0, "momentum": 0.99}, "class_name": "
          BatchNormalization"},
41    {"inbound_nodes": [[["batchnormalization_26", 0, 0]]], "name": "flatten_2", "config": {"name": "flatten_2", "
          trainable": true}, "class_name": "Flatten"},
42    {"inbound_nodes": [[["flatten_2", 0, 0]]], "name": "activation_2", "config": {"trainable": true, "name": "
          activation_2", "activation": "softmax"}, "class_name": "Activation"}
43 ], "output_layers": [["activation_2", 0, 0]], "name": "model_2"}, "keras_version": "1.0.7", "class_name": "Model"}
```

Erklärung

„Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.“

Ort Datum Unterschrift