INSTITUT FÜR INFORMATIK

Fachbereich Informatik und Mathematik

Masterarbeit

Maschinelles Lernen angewendet auf Bilder antiker Münzen

Antje Elisabeth Loyal

Abgabe am 12. Juli 2018

eingereicht bei
Prof. Dott.-Ing. Roberto V. Zicari
Datenbanken und Informationssysteme

Erklärung zur Abschlussarbeit

gemäß Ordnung für den Masterstudiengang Informatik §35 ABS.16 vom 13. Juli 2015

Hiermit erkläre ich Herr / Frau

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Ebenso bestätige ich, dass diese Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass die von mir abgegebenen schriftlichen gebundenen Versionen meiner Masterarbeit mit der auf dem Datenträger abgegebenen elektronischen Version übereinstimmen.

Frankfurt am Main, den 12. Juli 2018 Antje Elisabeth Loyal

Abstract

The examination of ancient coins is an important tool for the exploration of different cultures. At the same time, it is often difficult and time-consuming task, especially for old coins with a high level of wear and corrosion.

The classification of Roman imperial coins from Augustus (31 BC) to Zeno (491 AD) is represented in a standard work called Roman Imperial Coinage (RIC). The classification task can take a long time, even for experienced numismatics. This time could be better used for the analysis and interpretation of those coins. Apart from the time factor this task leads to high costs.

This thesis examines the use of machine learning methods in the field of numismatics for roman imperial coins. It is based on concepts of artificial neural networks for computer vision. For this purpose, different architectures and methods of training are compared.

The identification of the portraits on the front of the coins as a part of the classification task. Due to a division into two consecutive classification aims, portrait identification and classification into OCRE Classes (Online Coins of the Roman Empire, based on RIC), it also becomes possible to understand the functionality of the created neural networks and to gain an insight in the usually intransparent decision making of the neural network.

In the course of this work two datasets (*CoinPortraitData* and *OCRECoinClasses*) are created. These datasets can be used for further examination of other methods of artificial intelligence.

The result of this thesis shows that neural networks can support the classification task in the field of numismatics. Beside that they can be used to detect wrong classifications and can help to clarify further research questions.

Zusammenfassung

Die Untersuchung von Fundmünzen ist ein wichtiges Instrument bei der wissenschaftlichen Erforschung verschiedenster Kulturen. Zugleich handelt es sich bei der Untersuchung vieler Münzen, insbesondere jener aus älteren Kulturen mit starker Abnutzung oder Korrosion, um eine sehr schwierige und langwierige Aufgabe. Die Klassifizierung der Münzen der Kaiserzeit, von Augustus im Jahre 31 v. Chr. bis zu Zenon im Jahre 491 n. Chr., wird in einem der wichtigsten Standardwerke für römische Münzen, dem *Roman Imperial Coinage* (kurz RIC), abgebildet. Diese Klassifizierung kann selbst bei einem geübten Numismatiker mehrere Stunden in Anspruch nehmen. Dies führt zu einem großen Zeitaufwand, Zeit die der Auswertung der Funde im Wege steht, da die Forscher zuerst die Münzen klassifizieren müssen, bevor sie diese auswerten können. Außerdem entstehen durch diese zeitintensive Aufgabe hohe Kosten.

Die vorliegende Masterarbeit setzt sich mit dem Einsatz von Methoden des maschinellen Lernens im Bereich der Numismatik bei römisch imperialen Fundmünzen auseinander. Die Basis der Arbeit bilden Konzepte künstlicher neuronaler Netze für die Bilderkennung. Hierzu werden verschiedene Architekturen und Trainingsmethoden verglichen.

Bei römischen Münzen ist die Identifizierung des Porträts auf der Münzvorderseite ein wichtiger Teilschritt in der Klassifizierung der Münzen. Durch eine Unterteilung in zwei aufeinander aufbauende Klassifizierungsziele, der Identifizierung des Porträts und der Klassifizierung in die OCRE Klasse (Online Coins of the Roman Empire, dessen Klassen auf RIC basieren), ist es außerdem möglich, die Funktionsweise der erstellten neuronalen Netze besser nachzuvollziehen und Einblicke in die sonst undurchsichtige Entscheidungsfindung eines neuronalen Netzes zu erhalten.

Im Zuge dieser Arbeit sind auch zwei Datensets, *CoinPortraitData* und *CoinTypeData*, entstanden. Diese können im Anschluss dieser Arbeit für weitere Untersuchungen anderer Methoden der künstlichen Intelligenz dienen.

Die Resultate dieser Arbeit zeigen, dass es möglich ist die Arbeit der Numismatiker durch den Einsatz von Methoden des maschinellen Lernens zu unterstützen. Außerdem können die neuronalen Netze dazu dienen fehlerhafte Klassifizierungen aufzudecken oder bei weiteren Forschungsfragen behilflich sein.

INHALTSVERZEICHNIS

1 Einleitung **1**

2 Theoretische Grundlagen **3**
2.1 Maschinelles Lernen . . . 3
2.1.1 Unüberwachtes und überwachtes Lernen . . . 4
2.1.2 Maschinelles Sehen . . . 5
2.1.3 Unter- und Überanpassung . . . 5
2.1.4 Validierungsmenge . . . 6
2.2 Neuronale Netze . . . 6
2.2.1 Menschliche Neuronen . . . 7
2.2.2 Einschichtiges Perzeptron . . . 7
2.2.3 Mehrschichtiges Perzeptron . . . 8
2.2.4 Fehlerrückführung . . . 9
2.3 Convolutional Neural Networks . . . 10
2.3.1 Motivation . . . 11
2.3.2 Grundstruktur . . . 11
2.3.3 Convolution . . . 12
2.3.4 Pooling . . . 14
2.3.5 Normalisierung und Dropout . . . 14
2.3.6 Vollständig verbundene Schicht . . . 16
2.4 Evaluationsverfahren . . . 16
2.4.1 Accuracy, Precision und Recall . . . 16
2.4.2 Confusion Matrix . . . 18
2.4.3 Visualisierung durch Grad-CAM . . . 20
2.5 Frameworks und Bibliotheken für neuronale Netze . . . 20
2.5.1 Caffe . . . 20
2.5.2 Nvidia Digits . . . 21

3 Daten: Römisch-imperiale Münzen **23**
3.1 Münzen als Propagandamittel und Zeugen der Vergangenheit . . . 23
3.2 Struktur römischer Münzen . . . 24
3.3 OCRE, RIC und Klassenbildung . . . 25
3.4 Herausforderungen . . . 26

4 Methodik **29**
4.1 Aufgabenstellung und Zielsetzung . . . 29
4.1.1 Überblick . . . 31
4.1.2 Vorgehensweise . . . 31
4.2 Voruntersuchungen . . . 33
4.2.1 Erstellung erster Datensets . . . 33
4.2.2 Erste Durchläufe . . . 34

4.2.3 Ergebnisse der Voruntersuchungen 35
4.3 Entwicklung 36
4.3.1 Definition der Ein- und Ausgabe 37
4.3.2 Selektion einer geeigneten Architektur 39
4.3.3 Auswahl der Hyperparameter 43
4.4 Experimente und Validierung 46
4.4.1 Experiment 1: Maximale Bildanzahl pro Klasse 48
4.4.2 Experiment 2: AlexNet vs. VGG16 und ImageNet Vortraining vs. eigenes Vortraining 54
4.4.3 Experiment 3: Hyperparameter 58

5 Implementierung **63**
5.1 Erstellung der Datensets 63
5.1.1 Extraktion der Datenpaare 63
5.1.2 Vorverarbeitung und Datenset Erstellung mit Java und Nvidia Digits 65
5.2 Arbeitsumgebung 65
5.3 Architekturen 66
5.4 Hyperparameter in Digits 69
5.5 Evaluations Programm 70
5.5.1 Ausführen der trainierten Netze 70
5.5.2 Implementierung der Evaluationsmetriken 72

6 Evaluation und Ergebnisse **73**
6.1 Ziel 1: Porträt Klassen 73
6.1.1 Testmenge 73
6.1.2 Testdurchführung 73
6.1.3 Testergebnis 73
6.2 Ziel 2: OCRE Klassen 77
6.2.1 Testmenge 77
6.2.2 Testdurchführung 77
6.2.3 Testergebnis 77

7 Zusammenfassung und Ausblick **79**

A Abbildungen, Tabellen und Algorithmen **81**

Index **86**

Literaturverzeichnis **88**

KAPITEL

EINS

Einleitung

Einleitung

Münzen sind eine der wichtigsten Quellen für die Erforschung der Wirtschafts- und Kulturgeschichte verschiedener Epochen. Damit sie jedoch der Geschichtswissenschaft dienen können, müssen sie zuerst analysiert werden. Die zugehörige Wissenschaft nennt sich Numismatik.

Die Analyse antiker Münzen stellt eine besondere Herausforderung dar, da die Münzen meist stark abgenutzt oder korrodiert sind. Aufgrund der Ikonographien auf Vorder- und Rückseite und weiteren Merkmalen, wie Inschriften, Material oder Münzstempeln, gelingt die Datierung der Münzen erfahrenen Numismatikern trotzdem sehr genau. Für Sammler war lange Zeit das Porträt und die Inschrift auf der Münzvorderseite das Hauptmerkmal einer Münze [36]. Die Identifizierung des Porträts erfordert jedoch ein tiefes Expertenwissen und viel Erfahrung. Speziell bei großen Funden nimmt die Sichtung und Klassifizierung der Münzen längere Zeit in Anspruch. Durch diesen langwierigen Prozess entstehen zum Teil sehr hohe Kosten (Beispielrechnung in 3.4).

Die Aufgabenstellung dieser Arbeit beinhaltet die Untersuchung von Methoden des maschinellen Lernens, zur automatischen Identifizierung von Münzen anhand des abgebildeten Porträts. Im Verlauf der Arbeit stellte sich außerdem die Klassifizierung in die in OCRE (`http://numismatics.org/ocre/`) verwendeten Klassen als weiteres Klassifizierungsziel heraus. Neben dem Porträt gibt es hier noch weitere Details, wie das Material oder die Kennbuchstaben der Münzstätte, die gemeinsam den in OCRE gegebenen Münztyp bestimmen. Bei dieser Klassifizierung handelt es sich um eine komplexere Aufgabe, verglichen mit der Identifizierung der Porträts. Für beide Ziele wird der Einsatz von sogenannten Convolutional Neural Networks (kurz CNN) zur Klassifizierung der Münzen prototypisch untersucht.

Abschließend wird die Performance der Netze analysiert und ein Ausblick auf weitere Einsatzgebiete gegeben sowie mögliche Modifikationen der Netze erläutert.

Struktur der Arbeit

Der Hauptteil dieser Arbeit ist in vier Abschnitte unterteilt. Im ersten Abschnitt, in Kapitel 2, werden die theoretischen Grundlagen zum maschinellen Lernen und neuronaler Netze erläutert. Es wird dazu in Kapitel 2.2 auf die Entwicklung neuronaler Netze eingegangen. Anschließend werden die heute häufig verwendeten Convolutional Neural Networks in Kapitel 2.3 erklärt. Abschließend werden in Kapitel 2.5 zwei Frameworks bzw. Bibliotheken für die Implementierung neuronaler Netze vorgestellt.

Im zweiten Teil, Kapitel 3, wird auf das für diese Arbeit relevante Anwendungsgebiet eingegangen. Hierzu wird in Abschnitt 3.1 erläutert, welche Bedeutung die Münzen im römischen Reich hatten. Des Weiteren wird auf den Aufbau der Vorder- und Rückseite und auf für die Klassifizierung wichtige Eigenschaften von römischen Münzen in 3.2 eingegangen. Die Bildung der OCRE Klassen ist im nachfolgenden Kapitel 3.3 erklärt. Abschließend werden die besonderen Herausforderungen in diesem Bereich erläutert.

Im dritten Abschnitt des Hauptteils, Kapitel 4, wird die wissenschaftliche Vorgehensweise dargelegt. Der Abschnitt beginnt dazu mit der Definition der Ziele und Anforderungen. Anschließend werden die einzelnen Schritte zur Erstellung eines neuronalen Netzes aufgeführt, diese lassen sich grob in die Voruntersuchungen (Kapitel 4.2), die Entwicklung des neuronalen Netzes (Kapitel 4.3) und in die durchgeführten Experimente (Kapitel 4.4) unterteilen.

Der letzte Abschnitt, Kapitel 5, beschreibt die konkrete Erstellung der Datensets, die Arbeitsumgebung und die Implementierung der verwendeten Programme und Skripte.

Den Abschluss dieser Arbeit bilden Kapitel 6 und 7, in denen die Ergebnisse präsentiert werden und ein Ausblick auf künftige Verfahren und Anwendungsgebiete gegeben wird.

KAPITEL

ZWEI

Theoretische Grundlagen

Dieses Kapitel behandelt die theoretischen Grundlagen der Themen dieser Arbeit. Es wird mit einer kurzen Einführung in das maschinelle Lernen begonnen und anschließend werden die Funktionsweisen neuronaler Netze und speziell die Convolutional Neural Networks vorgestellt. Sofern nicht anders vermerkt stammen die Informationen dieser Kapitel aus dem Buch *Deep Learning* von Goodfellow [26]. Im Anschluss werden verwendete Evaluationsverfahren und Frameworks erläutert.

2.1 Maschinelles Lernen

Beim maschinellen Lernen handelt es sich um ein Forschungsgebiet der Informatik. Gegenstand der Forschung sind lernfähige Systeme. Diese sollen in der Lage sein, komplexe Probleme, nach einer Lernphase, selbst zu lösen. Um die Problemstellung und die verschiedenen Lernverfahren zu verstehen, ist die folgende Definition von Mitchell [43] hilfreich:

> A computer program is said to learn from experience $\boldsymbol{E}$ with respect to some class of tasks $\boldsymbol{T}$ and performance measure $\boldsymbol{P}$, if its performance at tasks in $\boldsymbol{T}$, as measured by $\boldsymbol{P}$, improves with experience $\boldsymbol{E}$.

Task $\boldsymbol{T}$ beschreibt die Aufgabe, welche durch den Algorithmus gelöst werden soll. Klassische Aufgaben des maschinellen Lernens sind **Klassifikation**, **Regression** oder **Clustering** [20].

Bei der **Klassifikation** soll jede Eingabe einer bestimmten Klasse zugeordnet werden. Bei den Klassen handelt es sich um eine diskrete (und bekannte) Menge. Ein Beispiel hierfür ist die Zeichenerkennung von Zahlen. Eingabe ist zum Beispiel eine handgeschriebene Ziffer und das System soll die Klasse dieser Ziffer ausgeben. Die Klassen sind in diesem Beispiel die Ziffern 0 bis 9. Ein bekanntes Datenset hierfür ist die MNIST Datenbank[17]. Diese enthält 60.000 Trainingsbilder zu handgeschriebenen Ziffern von 0 bis 9.

Wenn es sich bei der Ausgabe um eine oder mehrere kontinuierliche Variablen handelt, dann nennt sich diese Aufgabe **Regression**. Ein Beispiel hierfür ist die Vor-

hersage des Kaufpreises einer Immobilie. Die Variablen der Eingabe sind dann beispielsweise die Größe, die Lage und das Baujahr.

Beim **Clustering** werden die Daten in vorher nicht bekannte Klassen gruppiert. Ziel ist es, eine gegebene Datenmenge so in Gruppen zu unterteilen, dass sich ähnliche Objekte in einer Gruppe befinden und Objekte verschiedener Cluster möglichst unähnlich sind.

Bei der Aufgabenstellung dieser Arbeit handelt es sich um zwei Klassifikationsaufgaben, da die Eingabe eine endliche Menge an Münzbildern ist und die Ausgabe die zugehörige Klasse (entweder Porträt Identifizierung oder OCRE-Typ) sein soll.

Um den Algorithmus zu evaluieren, muss seine Performance $\boldsymbol{P}$, bei der Ausführung der Aufgabe, messbar sein. Wichtig ist, dass sich der Begriff Performance im Bereich des maschinellen Lernens nicht auf die Rechenleistung, sondern auf die Güte oder auch Qualität der Ausgabe bezieht. Bei einer Klassifikation ist dies zum Beispiel das Verhältnis von korrekt klassifizierten Eingaben zur gesamten Menge. Es gibt auch noch weitere Methoden die zur Evaluation verwendet werden können. Welche Verfahren zur Verfügung stehen und in dieser Arbeit Verwendung finden, ist in Abschnitt 2.4 und Kapitel 6 genauer erläutert.

Verwendet wird der Algorithmus nach dem Training üblicherweise auf neuen Daten, zu denen noch keine Labels vorliegen. Aufgrund dieser Tatsache ist es sinnvoll, eine Unterteilung des Datensets in **Trainings-** und **Testmenge** durchzuführen. Die Testmenge sollte Daten enthalten, welche während des Trainings nicht verwendet wurden. Die Performancemessung **P** erfolgt dann auf der Testmenge. Dies dient dazu, eine mögliche **Über-** oder **Unteranpassung** auf die Trainingsdaten zu verhindern. Genaueres zu den Problemen der Unter- bzw. Überanpassung ist in Abschnitt 2.1.3 zu lesen.

Bei der Erfahrung $\boldsymbol{E}$ handelt es sich um die Daten, die während des Trainings zur Verfügung stehen. Je nachdem, ob die richtige Ausgabe dem Algorithmus während des Trainings zur Verfügung steht oder nicht, spricht man von **unüberwachtem** oder **überwachtem Lernen**. Im Folgenden werden die beiden Kategorien genauer erläutert.

2.1.1 Unüberwachtes und überwachtes Lernen

Beim **unüberwachten Lernen** erhält der Algorithmus eine Menge an Eingabedaten und erzeugt daraus ein Modell, das die Eingabe beschreibt. Das so erzeugte Modell soll dazu fähig sein, Vorhersagen über neue Daten zu treffen. Ein Beispiel für eine Aufgabe des unüberwachten Lernens ist das Clustering, da die Klassen zuvor unbekannt sind und so dem Algorithmus während des Trainings nicht zur Verfügung stehen [48].

Beim **überwachten Lernen** erhält der Algorithmus neben den Eingabedaten auch die Ausgabedaten. Mit Hilfe der Datenpaare aus Ein- und Ausgabe kann der Algorithmus die Parameter so anpassen, dass er die Ausgabe am Besten reproduzieren kann [18] und zur einer Generalisierung auf neue Daten fähig ist. Ein mögliches

Verfahren dieser Modellanpassung ist die Fehlerrückführung, die in Kapitel 2.2.4 erläutert wird.

2.1.2 Maschinelles Sehen

Beim **maschinellen Sehen** handelt es sich um eine Teildisziplin des maschinellen Lernens. Gegenstand der Forschung sind hier Algorithmen und Techniken, die zur Analyse und Interpretation von Bildern dienen. In diesem Teilbereich sind die Eingabedaten auf Bilder oder Videos beschränkt. Dazu werden oft Bildverarbeitungsalgorithmen und Algorithmen der Mustererkennung verwendet. Typische Problemstellungen sind zum Beispiel die *Gesichts-* oder *Objekterkennung*, aber auch die *Szenenanalyse* oder *Stimmungserkennung*.

Erfolgreich verwendet werden Techniken des maschinellen Sehens bereits in der automatischen Produktkontrolle diverser Produktionen [9] oder bei der optischen Zeichenerkennung [33].

2.1.3 Unter- und Überanpassung

Eine zentrale Herausforderung des maschinellen Lernens ist die Fähigkeit der Algorithmen zur **Generalisierung**. Eines der ersten dazu fähigen Modelle, das auf den Grundlagen der menschlichen Neuronen basierte, war das **Perzeptron**. Seine Funktionsweise wird in Kapitel 2.2.2 erläutert. Unter Generalisierung versteht man in diesem Kontext, dass nicht nur die Fehlerrate auf den Trainingsdaten, sondern auch auf ungesehenen, neuen Daten, möglichst gering sein sollte. Die Fehlerrate auf den Trainingsdaten wird allgemein als *Trainingsfehler* und auf Testdaten als *Testfehler* bezeichnet.

Bei der Reduzierung des Trainingsfehlers handelt es sich um ein Optimierungsproblem. Ziel des maschinellen Lernens ist ein möglichst kleiner Trainingsfehler, bei gleichzeitig kleinem Testfehler. Zwei typische Verläufe der Fehlerraten ist in Abbildung 2.1 zu sehen. Diese zwei Ziele korrespondieren direkt mit zwei Herausforderungen beim Training der Algorithmen:

- Unteranpassung
- Überanpassung

Bei der **Unteranpassung** ist die Fehlerrate auf der Trainingsmenge hoch und das Modell ist auch nicht zur Generalisierung fähig. Während eine Unteranpassung bereits am Trainingsfehler ersichtlich ist, stellt die Überanpassung eine größere Herausforderung dar.

Bei einer **Überanpassung** ist der Trainingsfehler zwar niedrig, jedoch ist der Testfehler im Kontrast dazu hoch. Die Herausforderung besteht darin, den Zeitpunkt im Training zu finden, bei dem Test- und Trainingsfehler am niedrigsten sind.

Eine weitere Methode um eine Unter- oder Überanpassung aufzudecken ist die Kostenfunktion (auch Verlustfunktion oder engl. loss genannt). Diese Funktion gilt es in Optimierungsproblemen zu minimieren. Nähere Informationen dazu sind in Kapitel 2.2.4 zu finden. In Experiment 4.4.3 kann man außerdem den Verlauf der Kostenfunktion für verschiedene Netze dieser Arbeit sehen.

Zusammenfassend lässt sich sagen, dass ein niedriger Trainingsfehler noch keinen guten Algorithmus auszeichnet. Im Gegenteil, ist die Fehlerrate beim Training sehr niedrig, der Testfehler allerdings hoch, deutet dies auf eine Überanpassung hin und sollte zu einer Überarbeitung des Algorithmus führen.

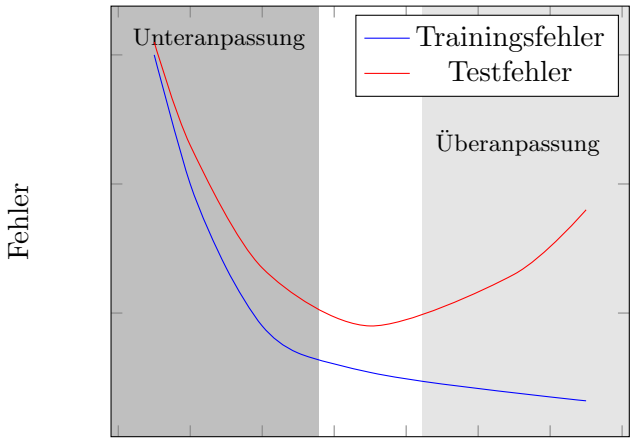


Abbildung 2.1: *Typische Verläufe des Trainings- und Testfehlers*

2.1.4 Validierungsmenge

Kontrollierbare Parameter der Algorithmen des maschinellen Lernens, speziell von neuronalen Netzen, werden auch **Hyperparameter** genannt. Diese werden nicht vom Algorithmus erlernt, sondern vom Entwickler gewählt [52]. Dies sind zum Beispiel die Lernrate, die Anzahl der Netzwerkschichten oder die Breite dieser Schichten.

Da die Performance eines Algorithmus dadurch beeinflusst wird, kann es hier ebenfalls zu einer Überanpassung auf die Trainingsdaten kommen. Um das zu vermeiden, wird, zur Auswahl der Hyperparameter, die Trainingsmenge weiter in Trainings- und **Validierungsmenge** unterteilt. Die Validierungsmenge dient ausschließlich zur Auswahl der Hyperparameter und wird nicht zum Training genutzt [27].

2.2 Neuronale Netze

Im folgenden Abschnitt 2.2.1 werden die biologischen Hintergründe zu neuronalen Netzen kurz erläutert. Anschließend wird das einschichtige Perzeptron in Kapitel

2.2.2 betrachtet und die Weiterentwicklung zum mehrschichtigen Perzeptron in 2.2.3 erklärt. Dieses Perzeptron ist Grundlage heutiger neuronaler Netze. Außerdem wird noch auf das Erlernen der Parameter mittels Fehlerrückführung in Abschnitt 2.2.4 eingegangen.

2.2.1 Menschliche Neuronen

Neuronale Netze sind nach dem Vorbild der menschlichen **Neuronen** entwickelt worden. Im menschlichen Gehirn läuft die Reizverarbeitung in Nervenzellen ab. Dazu treffen Reize an der Inputzone eines Neurons ein, den sogenannten *Dendriten*. Von dort aus wird der Reiz über den Zellkörper zum *Axonhügel* geleitet. Hier geschieht eine Bündelung der Signale bis es zum Überschreiten eines Schwellenwertes kommt. Wird dieser erreicht, wird ein Aktionspotential ausgelöst. Es erfolgt eine Reizaussendung an angebundene Zellen [37].

1943 wurde mit der **McCulloch-Pitts-Zelle**, der Vorgänger heutiger künstlicher Neuronen, entwickelt. Rosenblatt entwickelte daraufhin 1957 das **Perzeptron**. Es ist bis heute Grundlage künstlicher neuronaler Netze.

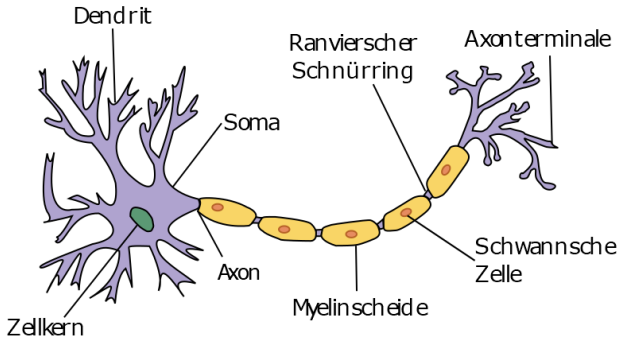


Abbildung 2.2: *Aufbau eines menschlichen Neurons aus Dendriten als Inputzone, Axon zur Signalbündelung und Axonterminale als Outputzonen [10]*

2.2.2 Einschichtiges Perzeptron

Ein einfaches Perzeptron (Abbildung 2.3) verfügt über n Eingangsvariablen $x_1, x_2, \ldots, x_n \in \mathbb{R}$ und Gewichte $w_1, w_2, \ldots, w_n \in \mathbb{R}$. Für die Berechnung der Ausgabe wird zuerst die Eingangsfunktion σ berechnet. Dabei handelt es sich um die gewichtete Summe der Eingabewerte:

$$\sigma = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

Die Ausgabe $o \in \mathbb{R}$ des Perzeptrons wird anschließend mit Hilfe der Aktivierungsfunktion φ bestimmt.

$$o = \varphi(\sigma) \tag{2.2}$$

Die Aktivierungsfunktion ist im ursprünglichen Modell von Rosenblatt eine einfache Stufenfunktion, die sogenannte *Heaviside*-Funktion:

$$f_\theta(x) = \begin{cases} 1, & \text{falls } x \geq \theta, \\ 0, & \text{sonst.} \end{cases} \tag{2.3}$$

Die Aktivierungsfunktion soll die Bündelung der Signale, wie sie im Axonhügel geschieht, nachahmen. Der Nachteil der Heaviside-Funktion ist, dass sie nicht differenzierbar ist. Für linear separierbare Daten ist dies noch unproblematisch. Bei Daten die nicht linear separierbar sind, kann die Heaviside-Funktion jedoch nicht verwendet werden. Heute werden aus diesem Grund andere Aktivierungsfunktionen, wie die sogenannten Rectified Linear Units (kurz ReLU) verwendet. Ein Überblick über verschiedene Aktivierungsfunktionen ist in [53] zu sehen.

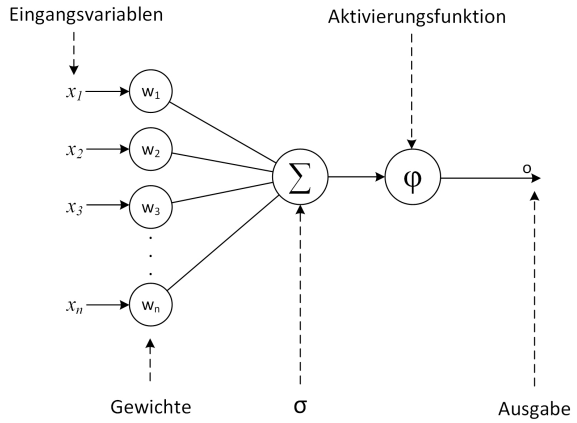


Abbildung 2.3: *Schematische Darstellung der Funktionsweise eines Einlagigen Perzeptrons mit Eingangsvariablen $x_1, x_2, \ldots, x_n$, Gewichten $w_1, w_2, \ldots, w_n$, Eingangsfunktion σ als gewichtete Summe und Aktivierungsfunktion φ*

Beim Training eines Perzeptrons geht es um das iterative Inkrementieren oder Dekrementieren der Gewichte w_i mit $i \in \{1, ..., n\}$, sodass die Menge an Trainingsdaten möglichst gut abgebildet werden kann.

Bei einem einschichtigen Perzeptron handelt es sich um eine Menge an p Perzeptrons, die jeweils mit allen Eingangsvariablen $x_1, x_2, \ldots, x_n \in \mathbb{R}$ verbunden sind. Diese Neuronen Schicht bildet auch gleichzeitig die Ausgabeschicht. Das Netz besitzt also p Ausgaben. Im Folgenden wird das mehrschichtige Perzeptron betrachtet.

2.2.3 Mehrschichtiges Perzeptron

Verknüpft man Perzeptrone in mehr als einer Schicht miteinander, wird dieses Netz **mehrschichtiges Perzeptron** (engl. *multilayer perceptron*, kurz *MLP*) genannt. Man unterscheidet bei allen mehrschichtigen Netzen drei verschiedene Arten von Schichten:

- **Eingabeschicht**: Beinhaltet die Eingangsvariablen

- **Ausgabeschicht**: Die Ausgabe des Netzes
- **Versteckte Schichten**: Alle Schichten zwischen Ein- und Ausgabeschicht

Bei den MLPs handelt es sich um vorwärts gerichtete, vollständig verbundene Netze. Das bedeutet, alle Neuronen einer Schicht, sind jeweils mit allen Neuronen der nachfolgenden Schicht verbunden (siehe Abbildung 2.4).

Ein großer Nachteil einer solchen Architektur ist die Skalierbarkeit. Ein Bild aus dem CIFAR-10 Datenset hat eine Größe von $32 \times 32 \times 3$ (Höhe, Breite und RGB-Farbraum), und ein einzelnes Neuron der ersten versteckten Schicht hätte somit $32 \times 32 \times 3 = 3072$ Eingangsgewichte. Hinzu kommt, dass solch ein Netz nicht nur aus einem, sondern aus vielen solcher Schichten besteht.

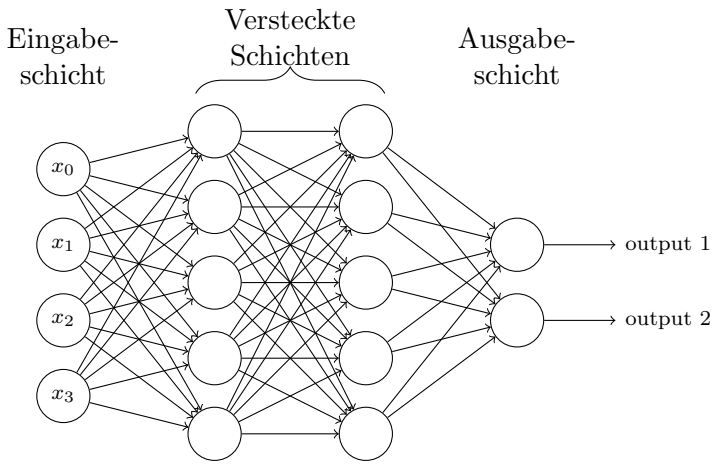


Abbildung 2.4: *Vollständige Verbundenheit bei einem mehrschichtigen Perzeptron.*

Diese komplexere Struktur führt dazu, dass einfaches Inkrementieren oder Dekrementieren der Gewichte nicht mehr funktioniert, da der Ursprung des Fehlers nicht direkt ersichtlich ist. Mehrschichtige Perzeptrone werden deshalb mit Hilfe der **Fehlerrückführung** trainiert. Auch in den heutigen neuronalen Netzen wird diese Art des Trainings noch verwendet. Anknüpfend hieran wird im nächsten Abschnitt die Fehlerrückführung erläutert.

2.2.4 Fehlerrückführung

Ziel des Trainings neuronaler Netze ist es, die Gewichte so zu wählen, dass die Ausgabe des Netzes den tatsächlichen Sollwerten (engl. ground truth) entspricht. Um die besten Gewichte zu finden, benötigt es eine Funktion, die Informationen über die Güte der aktuellen Gewichte liefert. Eine Funktion die diese Aufgabe übernimmt ist die **Kostenfunktion**. Diese ermittelt die Abweichungen der Ausgabe des Netzes von den Sollwerten. Während des Trainings wird versucht diese Kostenfunktion zu minimieren.

Ein Verfahren was der Minimierung der Kostenfunktion dient, ist das sogenannte **Fehlerrückführungsverfahren**. Es basiert auf der **Gradientenabstiegsmethode**.

Angenommen $f(x) = y$ sei eine beliebige Kostenfunktion und x und y reellwertig. Die Ableitung dieser Funktion $f'(x)$ bzw. $\frac{dy}{dx}$ gibt an, wie x verändert werden muss, um kleine Verbesserungen von y zu erhalten. Wir minimieren die Kostenfunktion $f(x)$, indem wir x in kleinen Schritten, entgegen der Richtung des steilsten Anstiegs (Richtung des Gradientens), abändern. Diese Methode nennt sich **Gradientenabstieg**[26]. Das eigentliche Ziel des Abänderns ist das Auffinden eines **globalen Minimums** der Kostenfunktion, jedoch kann häufig nur ein lokales Minima garantiert werden.

Im Kontext neuronaler Netze und speziell bei hochdimensionalen Eingaben (wie Bilder), gestaltet sich das Auffinden eines globalen Minimums jedoch oft als sehr schwierig, da es viele *lokale Minima*, *Sattelpunkte* und *flache Bereiche* geben kann. Oft genügt es daher den Optimierungsverfahren einen kleinen Wert von $f(x)$ zu finden, der nicht zwingend der minimale Wert ist.

Mehrdimensionale Eingaben führen außerdem dazu, dass die **partiellen Ableitungen** verwendet werden müssen. Die partielle Ableitung $\frac{\partial}{\partial x_i} f(x)$ misst, wie f sich verändert, wenn nur die Variable x_i in Punkt x erhöht wird. Der Gradient von f ist dann ein Vektor, $\vec{\nabla}_{x_i}$, der alle partielle Ableitung enthält und in Richtung der stärksten Steigung zeigt. Bei Element i des Gradienten handelt es sich um die partielle Ableitung von f bezüglich x_i.

Der Hyperparameter der Lernrate steht in direktem Bezug zur Gradientenabstiegsmethode und bezieht sich auf die Schrittgröße der Abänderung (in Richtung stärkster Steigung). Die Kostenfunktion wird an der Ausgabeschicht berechnet und die Änderung der Gewichte erfolgt von der Ausgabeschicht in Richtung Eingabeschicht. Aus diesem Grund nennt sich das Verfahren auch Fehlerrückführung (engl. backward propagation of errors)[26].

2.3 Convolutional Neural Networks

2012 gewannen Krizhevsky et al. [40] die *ImageNet Large Scale Visual Recognition Challenge* (kurz *ILSVRC*) mit ihrem **Convolutional Neural Network**, *AlexNet*. Das Netz erreichte eine Top-5 Fehlerrate von 15,3%, welche fast halb so groß war wie die des zweitplatzierten Teams (26,2%). Dieser Sieg war ausschlaggebend für das heute große kommerzielle Interesse an Convolutional Neural Networks (kurz *CNN*). Eingesetzt wurden diese jedoch schon vorher erfolgreich. 1998 entwickelten Yann LeCun et al. [41] ein System, das Schecks einlesen konnte. Ende der 1990 Jahre wurden bereits 10% aller Schecks in den USA von diesem System eingelesen [26].

Im Folgenden werden die Vorteile von CNNs erläutert. Es werden außerdem die Bausteine dieser Architektur erklärt, durch die diese Vorteile zustande kommen. In Kapitel 4.3.2 wird der Einsatz dieser Architektur in dieser Arbeit erklärt und welche

Vorteile sich daraus bei den Münzdatensets ergeben.

2.3.1 Motivation

Wie bereits in 2.4 erwähnt, ist ein Nachteil der klassischen neuronalen Netze, dass sie nicht skalierbar sind. Mit steigenden Bildgrößen gelangen solche Netze schnell an die Grenzen der Rechen- und Speicherkapazität. Trotz gestiegener Kapazitäten, wäre ein langwieriges Training und Überanpassung die Folge [58].

Neben der Anzahl an Gewichten reagieren klassische neuronale Netze auch empfindlich auf Verschiebungen in den Bildern. Bei den heutigen Einsatzgebieten neuronaler Netze (z.B. Objekt- oder Gesichtserkennung), müssen diese jedoch **translationsinvariant** sein [26].

Stand der Technik sind heute Convolutional Neural Networks [21][42][40], was auf deutsch als faltende neuronale Netzwerke übersetzt werden kann (in der Fachliteratur wird vornehmlich der bereits eingeführte Begriff des Convolutional Neural Networks aus dem angelsächsischen Sprachraum verwendet). Die drei größten Vorteile dieser Netze, gegenüber klassischen Netzen wie dem MLP, sind:

- lokale Konnektivität
- geteilte Gewichte
- Translationsinvarianz

Die nachfolgenden Kapitel beleuchten die Grundstruktur in Abschnitt 2.3.2 und die Convolution Operation und Schicht in 2.3.3. Daran anschließend werden das Pooling in 2.3.4, die Normalisierung in 2.3.5 und die vollständig verbundenen Schichten in 2.3.6 erläutert.

2.3.2 Grundstruktur

Grundsätzlich bestehen CNNs, ähnlich wie die MLPs, aus drei verschiedenen Schichten. Der Eingabeschicht, den versteckten Schichten und der Ausgabeschicht (siehe 2.4). Der Unterschied liegt in der mittleren Schicht, die bei CNNs mindestens eine Convolution Operation (siehe 2.3.3) enthält [26] und in der nicht jedes Neuron mit jedem vorangegangenen Neuron verbunden ist. Die Convolution Schichten bestehen außerdem aus drei verschiedenen Grundbausteinen:

- Convolution Operation und Aktivierungsfunktion (siehe 2.3.3)
- Pooling (siehe 2.3.4) und
- Normalisierung und Dropout (siehe 2.3.5)

Diese drei Bausteine können sich je nach Architektur beliebig oft wiederholen. Bei den letzten Schichten eines Convolutional Neural Network handelt es sich häufig um vollständig verbundene Schichten (siehe dazu Abschnitt 2.3.6). In Abbildung 2.5 ist

die Grundstruktur von *LeNet-5* zu sehen. Dieses CNN wird erfolgreich zur Ziffernerkennung eingesetzt [41]. Dort sieht man auch die Verwendung der verschiedenen Bausteine und den Aufbau in Schichten.

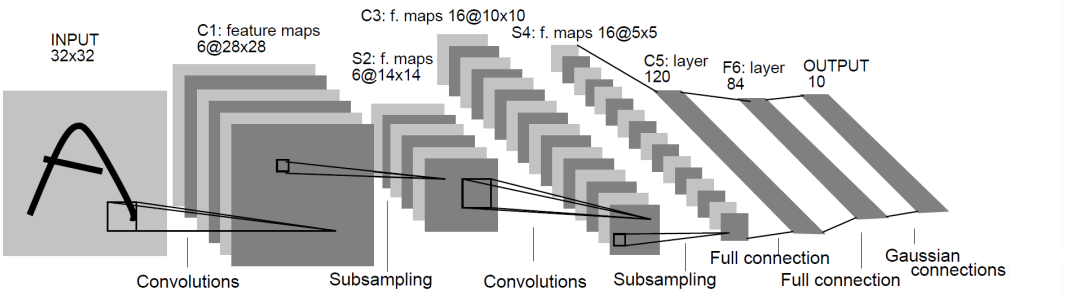


Abbildung 2.5: *Architektur von LeNet-5 zur Erkennung von Ziffern mit fünf lernbaren Convolution Schichten (C1 bis C5), gefolgt von einer vollständig verbundenen Schicht (F6). Subsampling entspricht Pooling [41]*

2.3.3 Convolution

Die ursprüngliche Idee der Convolutional Neural Networks war inspiriert vom biologischen Konzept des menschlichen **visuellen Systems**. Dieses ist hierarchisch aufgebaut, wobei einfache Strukturen, wie zum Beispiel Kanten, zuerst extrahiert werden und schließlich zu komplizierteren Strukturen zusammengefügt werden [23]. Die Anwendung des Prinzips der rezeptiven Felder geschieht bei CNNs dadurch, dass nicht alle Neuronen mit allen vorangehenden Neuronen verknüpft werden. Daraus ergibt sich der erste Vorteil dieser Architektur, die sogenannte **lokale Konnektivität**. Dadurch wird sichergestellt, dass die Neuronen bei bestimmten lokalen Eingabemustern am stärksten aktiviert werden.

In Convolutional Neural Networks werden diese Filter mittels **Faltungsoperation**[1] (auch Kernel genannt) und einem mehrschichtigen Aufbau des Netzes erreicht. In der Theorie sollten Neuronen der vorderen Schichten ebenfalls einfache Strukturen erkennen. Je näher man der Ausgabeschicht kommt, desto komplexer werden die Strukturen auf die die jeweiligen Neuronen reagieren [41].

Die Faltungsmatrix $K \in \mathbb{R}^{k_b \times k_h}$, wobei k_b die Breite und k_h die Höhe der Eingabe ist, wird über die gesamte Eingabe $I \in \mathbb{R}^{m \times n}$ gefaltet, um die Ausgabe I' zu erzeugen[28]. Ist die Eingabe ein zweidimensionales Bild, wird jeder Pixel $I'(x, y)$ der Ausgabe mit der Convolution Funktion wie folgt berechnet:

$$I'(x,y) = (K * I) = \sum_m \sum_n I(x-m, y-n) K(m,n) \tag{2.4}$$

[1]Eigentlich gibt es einen Unterschied zwischen dem Faltungsoperator und der bei neuronalen Netzen häufig implementierten Kreuzkorrelation. Da die Gewichte jedoch vom Netz gelernt werden und somit beide Operationen durch geeignete Gewichte die gleichen Ergebnisse liefern, werden beide Begriffe im Bereich neuronaler Netze synonym verwendet

Hinzu kommt außerdem die Tiefe d^{i-1} der Eingabeschicht $(i-1)$, welche jedoch in der Literatur zur Vereinfachung meist weggelassen wird[52].

In der Bildverarbeitung werden diese Filter zum Beispiel in Weichzeichnern oder Kantenfiltern angewendet. Die Anwendung eines Laplace-Filters, zur Kantendetektion, ist in Abbildung 2.6 zu sehen. Anhand dieser Abbildung wird auch ersichtlich, dass mit Hilfe solcher Faltungsmatrizen einfache Strukturen, wie Kanten oder Farbflächen, extrahiert werden können.

Abbildung 2.6: *Auswirkung eines Laplace-Filters mit Achterumgebung (siehe Anhang A.1 für Filtermatrixeinträge)*

Eine Convolution Schicht erhält als Eingabe eine Menge an Merkmalen, diese werden auch Aktivierungskarten genannt. Als Ausgabe werden n Aktivierungskarten produziert, wobei es sich bei n um die Anzahl der angewendeten Faltungsmatrizen in dieser Schicht handelt. Die Gewichte der Filter werden von den gegebenen Trainingsdaten gelernt.

Wichtig ist außerdem, dass die Gewichte *eines* Filters, der auf die gesamte Eingabe angewendet wird, identisch sind. Dies führt zu einem wichtigen Merkmal von CNNs, den **geteilten Gewichten**. Durch diese gelingt es dem Netz, bestimmte Merkmale zu extrahieren und diese an jeder Stelle der Eingabe wieder entdecken zu können. Daraus ergibt sich der Vorteil der **Translationsinvarianz** gegenüber klassischer MLPs.

Die Anzahl n der Filter und die Größe $k_b \times k_h$ dieser, sind Hyperparameter der Schicht und werden vom Entwickler festgelegt.

Weitere Hyperparameter sind die Schrittgröße (engl.: stride) $s \in \mathbb{N}_{\geq 1}$, welche angibt, in welchem Abstand die Filtermatrizen auf die Eingabe angewendet werden sowie der Umgang mit den Rändern. Würde man die Ränder nicht erweitern, würde die Ausgabe kleiner als die Eingabe werden. Um dies zu vermeiden, kann das sogenannte *Zero-Padding* verwendet werden. Hierbei wird die Eingabe mit Null-Werten umrandet, sodass die Ausgabe die Größe der Eingabe (ohne Null-Werte) besitzt [6].

Mit Hilfe der Aktivierungsfunktion wird nach Anwendung der Faltungsmatrix die Ausgabe jedes Neurons berechnet. Häufige Anwendung finden hier die sogenannten

Rectified Linear Units (ReLUs) [40]. ReLUs sind definiert als:

$$f(x) = \max(0, x) \tag{2.5}$$

2.3.4 Pooling

Nachdem nun die Convolution und Aktivierungsfunktionen betrachtet wurden, folgt das sogenannte Pooling. Hierbei handelt es sich um eine Art Zusammenfassung eines $p \times p$ Bereiches der vorherigen Ausgabe. Wie bei der Convolution, sind auch beim Pooling die Schrittgröße s ($s \in \mathbb{N}_{\geq 1}$) und die Filtergröße p ($p \in \mathbb{N}_{\geq 1}$) Hyperparameter.

Für das Pooling stehen verschiedene Varianten, wie das max-Pooling oder das mean-Pooling, zur Verfügung. In Abbildung 2.7 ist die Anwendung des max-Poolings visualisiert.

Das Pooling sorgt in erster Linie für eine Reduktion der Daten (bei $s \geq 1$). Dadurch wird weniger Speicher benötigt und die Berechnungsgeschwindigkeit gesteigert. Weitere Gründe für den Einsatz der Pooling Operation sind die damit einhergehende lokale Translationsinvarianz und die Robustheit gegenüber kleinen lokalen Änderungen. Zu sehen ist dies in Abbildung 2.8. Dort wurde die Eingabe an dieser Stelle um eine Stelle nach rechts verschoben. Dadurch ändert sich die Eingabe zwar an jeder Stelle, die Ausgabe wird jedoch nur an einer Stelle beeinflusst. Des Weiteren kann dadurch das Pooling gegen eine Überanpassung (siehe 2.1.3) helfen.

Häufig gewählte Werte sind $p \in \{2, 3, 4, 5\}$ und $s = 2$, wie in AlexNet [40] und VGG16 [47].

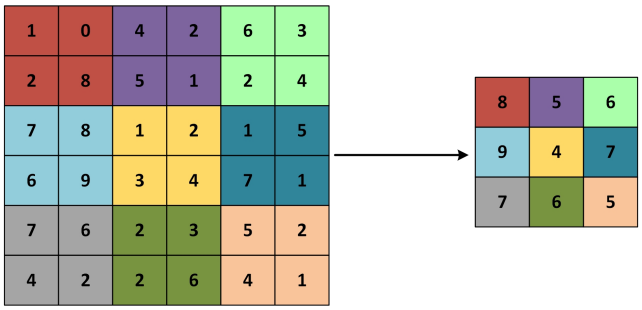


Abbildung 2.7: *Anwendung eines* 2×2 *max-Pooling Filters auf eine* 6×6 *große Eingabe mit einer Schrittgröße von* $s = 2$*. Die Anwendung des max-Poolings führt zu einer kleineren* (3×3) *Ausgabe.*

2.3.5 Normalisierung und Dropout

Sobald nur einige wenige Eingaben in einer ReLU Aktivierungsfunktion positiv sind, wird dieses Neuron einen Lernprozess vollziehen [40]. Um die Fähigkeit der Generalisierung jedoch noch weiter zu unterstützen, dienen erneut die menschlichen

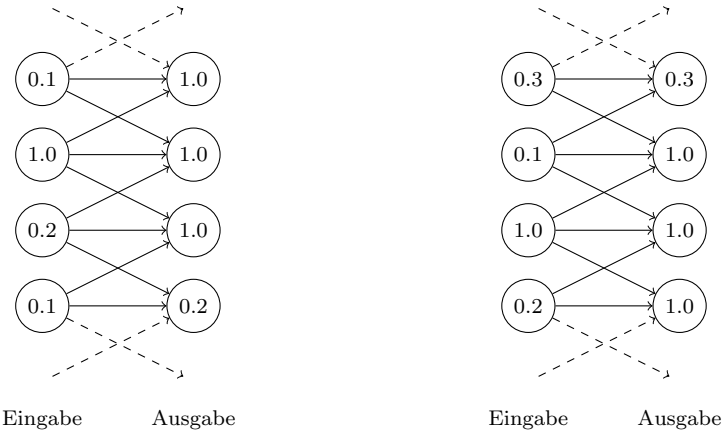


Abbildung 2.8: *Ausschnitt einer max-Pooling Operation: Durch Verschiebung ändert sich die Eingabe an allen Stellen, während die Ausgabe nur an der Hälfte der Stellen sich ändert. Bild nach [29]*

Neuronen als Inspiration. Bei menschlichen Neuronen gibt es das Prinzip der lateralen Hemmung [8]. Hierbei führt eine Reizung eines Neurons dazu, dass umliegende Neuronen ihre Aktivität verringern. Dieser so erzeugte Kontrast der Stimulation erhöht die sensorische Wahrnehmung. Anwendung findet die sogenannte *local response* **Normalisierung** (kurz *LRN*) zum Beispiel in AlexNet (siehe 4.3.2).

Sei $a^i_{x,y}$ die Aktivität eines Neurons, nachdem die Filtermatrix i an Position (x, y) berechnet und anschließend die Aktivierungsfunktion ReLU verwendet wurde, dann wird die LRN Aktivität $b^i_{x,y}$ wie folgt berechnet[40]:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta} \tag{2.6}$$

Dabei wird die Summe der n adjazenten Aktivierungs-Maps an der gleichen Position berechnet, wobei N die gesamte Anzahl an Filtern in dieser Schicht ist. Bei den Konstanten k, n, α und β handelt es sich um Hyperparameter. Im Falle von AlexNet wurde $k = 2, n = 5, \alpha = 10^{-4}$ und $\beta = 0.75$ gewählt.

Auf dem ImageNet Datenset haben Simonyan et al. [47] Experimente mit der LRN Normalisierung durchgeführt und kamen zu dem Ergebnis, dass die Performanz des Models dadurch nicht merklich gesteigert werden konnte. Jedoch führte die Benutzung zu einem höheren Speicherverbrauch und längerer Rechenzeit, weswegen in VGG16 (siehe 4.3.2) lediglich eine LRN Normalisierung verwendet wird.

Ziel des Trainings der Neuronen ist es, dass jedes Neuron, unabhängig von anderen, ein wichtiges Merkmal erkennt und von diesem aktiviert wird. Es kann jedoch passieren, dass es zu einer Co-Adaption kommt und ein einzelnes Neuron nicht in der Lage ist, solch ein Merkmal zu detektieren und stattdessen nur in Kombination mit anderen Neuronen dazu fähig ist. Dies kann wiederum zu einer Überanpassung führen, da Merkmale nicht zwingend immer in solch einer Kombination auftreten müssen [38].

Eine Möglichkeit eine solche Co-Adaption zu verhindern, ist das sogenannte **Dropout**. Hierbei werden beim Training zeitweise eine bestimmte Anzahl an Neuronen (samt deren Verbindungen) zufällig entfernt.

2.3.6 Vollständig verbundene Schicht

Am Ende eines CNNs befinden sich die sogenannten **vollständig verbundenen Schichten**. Hier sind, wie bei einem mehrschichtigen Perzeptron, alle Neuronen mit den Neuronen der vorangehenden Schicht verbunden. Die Anzahl der Neuronen der letzten Schicht entspricht der Anzahl an Klassen bei einer Klassifikationsaufgabe. Die Neuronen der vorangegangen Schichten haben die Eigenheiten (auch Features) jeder Klasse gelernt und werden beim Auftreten einer solchen aktiviert. Die Ausgabe repräsentiert dann, welche Klasse am wahrscheinlichsten ist, auf Basis der vorherigen Aktivierung der Neuronen.

Eine häufig genutzte Funktion ist die sogenannte **Softmax Funktion**. Da diese jedoch einen Vektor als Eingabe erwartet, muss die Ausgabe einer Convolution Schicht zuerst umgewandelt werden. Die Ausgabe ist dann eine Wahrscheinlichkeitsverteilung zwischen 0 und 1, wobei die Summe aller Ausgabewerte 1 ergibt. Der Vorteil gegenüber einer einfachen *max*-Funktion ist, dass auch bei einer sehr unsicheren Ausgabe (z.B. 0.49 und 0.51) nicht einfach nur die höchste ausgegeben wird.

2.4 Evaluationsverfahren

Die Evaluation der trainierten Modelle geschieht auf der Testmenge. Hierzu stehen verschiedene Metriken zur Verfügung, welche auch später in der Entwicklung und Evaluation genutzt wurden:

- Accuracy
- Precision
- Recall
- F-Maß
- Confusion Matrix
- Visualisierung der Schichten

Die genannten Metriken sollen in diesem Kapitel vorgestellt werden. Dazu wird auf die Berechnung der genannten Metriken eingegangen und die Visualisierungsmethode mit Hilfe von Heatmaps unter der Verwendung von Grad-CAM erläutert.

2.4.1 Accuracy, Precision und Recall

Um die Güte der Algorithmen zu ermitteln gibt es verschiedene Metriken die genutzt werden können. Eine davon ist die sogenannte **Accuracy** (dt.: Klassifikati-

onsgenauigkeit). Sie wird mit Hilfe der Klassifikationsergebnisse des Modells auf der Testmenge bestimmt. Die Accuracy gibt das Verhältnis der richtigen Klassifizierungen zur Gesamtheit der Klassifizierungen an [46]. Um dieses Verhältnis bestimmen zu können, benötigt es die Werte einer sogenannten Confusion Matrix. Bei einer binären Klassifikation in positive und negative Werte (z.B. Ultraschallbilder und die Klassifizierung in *krank* (positiv) und *nicht krank* (negativ)) ergeben sich dann anhand der Ausgabe des Modells und der Ground-Truth (die wahren Klassen der Bilder) die Werte für:

- Richtig Positiv (das Modell gibt Positiv aus, die Ground-Truth ist Positiv)
- Richtig Negativ (das Modell gibt Negativ aus, die Ground-Truth ist Negativ)
- Falsch Positiv (das Modell gibt Positiv aus, die Ground-Truth ist Negativ)
- Falsch Negativ (das Modell gibt Negativ aus, die Ground-Truth ist Positiv)

Die Confusion Matrix beinhaltet jeweils die Anzahl dieser vier (siehe Abbilung 2.1). Die Formel 2.7 für die Berechnung der Accuracy ergibt sich aus der Anzahl der korrekten Klassifikationen (RP und RN), welche durch die Anzahl aller Klassifizierungen (RP, RN, FP und FN) geteilt wird:

		Modell	
		Positiv	Negativ
Ground-Truth	Positiv	Richtig Positiv (RP)	Falsch Negativ (FN)
	Negativ	Falsch Positiv (FP)	Richtig Negativ (RN)

Tabelle 2.1: Allgemeine Confusion Matrix für den Fall einer binären Klassifikation

$$Accuracy = \frac{RP + RN}{RP + RN + FP + FN} \tag{2.7}$$

Im oben genannten Beispiel mit den Ultraschallbildern lässt sich allerdings bereits ein Nachteil der Accuracy erklären. Angenommen es liegen 100 Bilder in der Testmenge vor, davon sind 90 *nicht krank* und 10 *krank*. Die Confusion Matrix eines Modells, welches bei allen 90 *nicht Kranken* korrekt liegt, jedoch nur einen *Kranken* erkennt ist in Tabelle 2.2 zu sehen.

		Modell	
		Positiv	Negativ
Ground-Truth	Positiv	1 (RP)	0 (FP)
	Negativ	9 (FN)	90 (RN)

Tabelle 2.2: Confusion Matrix des Modells, das 90 *nicht Kranke* erkennt, jedoch nur einen *Kranken*

Das Ergebnis der Accuracy dieses Modells ist 91%. Ein Modell, dass bei jeder Eingabe immer *nicht krank* ausgibt, besitzt allerdings auch schon eine Accuracy von

90%. Das Problem welches hier vorliegt nennt sich **Accuracy Paradox**. Zu diesem kommt es durch ein großes Ungleichgewicht der Klassen. Sobald eine Klasse deutlich häufiger vorkommt, als eine andere, eignet sich die Accuracy alleine nicht mehr als Metrik um die Güte der Algorithmen zu ermitteln.

Aus diesem Grund gibt es neben der Accuracy auch noch **Precision** (dt.: Präzision) und **Recall** (dt.: Sensitivität, Trefferquote)[46]. Beide lassen sich ebenfalls aus der Confusion Matrix ableiten:

$$Precision = \frac{RP}{RP + FP} \tag{2.8}$$

$$Recall = \frac{RP}{RP + FN} \tag{2.9}$$

Für das Beispiel mit den Ultraschallbildern ergibt sich für das Modell mit einer Accuracy von 91%, eine Precision von 1 und ein Recall von $0,1$. Für das Modell, dass immer *nicht krank* ausgibt ergibt sich jedoch eine Precision von 0 und ein Recall von ebenfalls 0.

Sollte also ein Ungleichgewicht der Klassen vorliegen, müssen sowohl Recall, als auch Precision zusätzlich bestimmt werden, um die Güte des Modells beurteilen zu können. An dieser Stelle sei außerdem noch erwähnt, dass es häufig zu einem Trade-Off zwischen Precision und Recall kommen kann. Ein Maß, welches sowohl Precision als auch Recall mit einbezieht ist das **F-Maß** (auch F1-Score genannt). Hierbei handelt es sich um das harmonische Mittel aus Precision und Recall:

$$\text{F-Maß} = 2 \cdot \frac{(Precision \cdot Recall)}{(Precision + Recall)} \tag{2.10}$$

Außerdem gibt es noch den sogenannten **Top-1 Error**. Dabei handelt es sich um die Fehlerrate des wahrscheinlichsten Ergebnisses bei der Ausgabe des Netzes. Da CNNs eine Wahrscheinlichkeitsverteilung über die möglichen Klassen ausgeben, wird auch häufig der Top-5 Error genutzt. Dieser gibt ein Maß dafür, wie häufig die korrekte Klasse in, den 5 Klassen mit höchster Wahrscheinlichkeit, der Ausgabe liegt.

Die obigen Beispiele waren für den Fall einer binären Klassifikation, in vielen Fällen gibt es jedoch mehr als zwei Klassen. Da sich alle Werte aus der Confusion Matrix errechnen lassen und die Confusion Matrix auch noch weitere Evaluationsmöglichkeiten bietet, beschreibt der nächsten Abschnitt das Erstellen einer Confusion Matrix mit mehreren Klassen.

2.4.2 Confusion Matrix

Viele Evaluationsverfahren und Qualitätskriterien neuronaler Netze zur Klassifikation lassen sich aus der **Confusion Matrix** ableiten. Bei einer Confusion Matrix für mehr als zwei Klassen, handelt es sich um eine Matrix $c_{ij} \in \mathbb{N}_{\geq 0}^{K \times K}$, wobei $K \in \mathbb{N}_{>2}$

die Anzahl der Klassen ist. Der Eintrag c_{ij} gibt an, wie oft Klasse i als Klasse j klassifiziert wurde. Korrekte Klassifizierungen befinden sich auf der Diagonalen C_{ii}. Bei der Summe $\sum_{i=1}^{K}\sum_{j=1}^{K} C_{ij}$ handelt es sich um die Anzahl evaluierter Daten und $\frac{\sum_{i=1} c_{ii}}{\sum_{i=1}^{K}\sum_{j=1}^{K} C_{ij}}$ ergibt die, bereits für den binären Fall erklärte, Accuracy des evaluierten Modells [52]. Der Vorteil der Confusion Matrix liegt darin, dass man eine Überanpassung sehr schnell erkennen kann: Sobald die Werte abseits der Diagonalen bei einer Klasse höher liegen. Bei einem Klassifizierer, der zwischen den Klassen *snake, cat, dog* und *bird* unterscheiden soll und die Ergebnisse in Tabelle 2.3 liefert, lässt sich anhand der Confusion Matrix eine deutliche Überanpassung auf die Klasse *cat* erkennen.

		Modell			
		snake	cat	dog	bird
Ground-Truth	snake	3	7	0	0
	cat	0	10	0	0
	dog	0	6	3	0
	bird	0	9	0	1

Tabelle 2.3: Beispiel Ergebnisse des Klassifizierers, die Zeilen geben die Ausgabe wieder, während die Spalten der Ground Truth entsprechen. Die Confusion Matrix für diesen Klassifizierer ist in Abbildung 2.9 zu sehen.

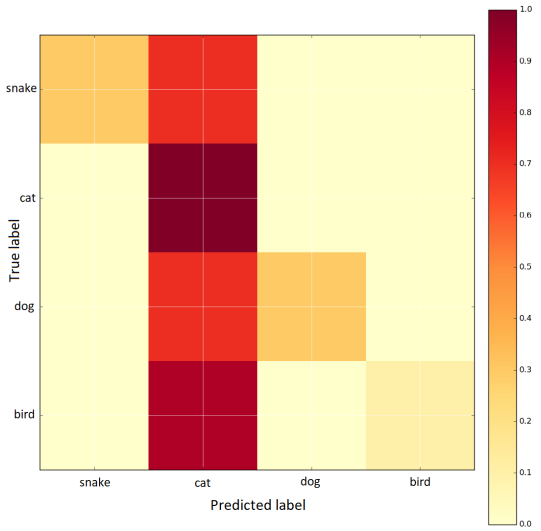


Abbildung 2.9: *Confusion Matrix (erstellt mit [15]) für einen überangepassten Klassifizierer. Das Modell gibt zu oft die Klasse cat aus, was anhand der vertikalen deutlich zu erkennen ist.*

2.4.3 Visualisierung durch Grad-CAM

Bei Grad-CAM handelt es sich um einen auf den Gradienten basierenden Visualisierungsansatz. Grad-CAM soll CNNs transparenter machen, indem es *wichtige* Bereiche der Eingabe für das Modell hervorhebt. Dazu werden die Gradienten Informationen der letzten Netzschicht verwendet. Es bietet sich für den Einsatz bei Modellen zur Klassifikation an, da es dem Nutzer Einsicht darin gibt, wie bestimmte Entscheidungen des Netzes gefällt werden.

Im Kontext der Münzen ist Grad-CAM speziell bei den Porträts vielversprechend. Ein Grund dafür ist, dass die Ausgabe von einem Numismatiker überprüft und die Plausibilität der Klassifizierung beurteilt werden kann.

Ein weiterer Grund ist die bei den Porträts mögliche Überprüfung, ob tatsächlich nur die abgebildeten Köpfe genutzt werden oder zusätzlich die Beschriftung (oder andere Details). Anhand dieser Ergebnisse kann dann auch entschieden werden, ob ein Zuschneiden der Bilder nötig ist.

2.5 Frameworks und Bibliotheken für neuronale Netze

Für das Training und die Anwendung neuronaler Netze gibt es verschiedene Frameworks und Bibliotheken. Die bekanntesten Frameworks sind Tensorflow und Caffe. Allen voran steht die Entwicklung einer Architektur. Zur Umsetzung bieten alle Frameworks unterschiedliche Möglichkeiten an.

Tensorflow arbeitet hierzu mit Graphen, die Ein- und Ausgaben der einzelnen Rechenschritte sind die Kanten in diesen Graphen. Die Knoten beinhalten die Operationen. Die Kommunikation zwischen den Knoten erfolgt mit Datenarrays, den sogenannten Tensoren. Tensorflow wird vom Google Brain Team entwickelt und ist als Open-Source Bibliothek verfügbar[16].

Auf dem in dieser Arbeit verwendeten Computer mit leistungsstarken Grafikkarten ist nur **Caffe** installiert und aufgrund fehlender Berechtigungen können keine weiteren Frameworks installiert werden. Deshalb wird für die Erstellung und das Training Caffe verwendet. Dieses Framework wird im nächsten Kapitel genauer erläutert.

2.5.1 Caffe

Das *Caffe Framework* wurde vom Berkeley Vision and Learning Center (BVLC) entwickelt [3]. Es ist ein speziell auf das Training und die Verwendung von CNNs zugeschnittenes Framework. Caffe ist in `C++` implementiert und nutzt das Plaintext Protocol Buffer Schema (`.prototxt`) für die Definition der Netztopologie und der Trainingsparameter.

Caffe besitzt unter anderem Schnittstellen für Python und Matlab. Des Weiteren ermöglicht Caffe das Training von CNNs sowohl auf der CPU als auch auf der GPU.

Caffe selbst besitzt keine grafische Benutzeroberfläche, sondern wird mittels Konsolenbefehlen gesteuert. Für die Definition der Netzstruktur und Trainingsparameter werden mehrere Dateien im .prototxt Format übergeben:

- solver.txt: Orchestriert die Modelloptimierung, enthält die Hyperparameter des Trainings
- train_val.prototxt: Enthält den strukturellen Aufbau des Netzes, wie er für das Training und die Validierung verwendet wird
- deploy.prototxt: Enthält den strukturellen Aufbau des Netzes, wie er für das Deployment von trainierten Modellen genutzt wird

Die solver.txt dient lediglich zur Definition der Hyperparameter, welche nicht bereits in der Netzdefinition enthalten sind und sich speziell auf den Trainingsprozess beziehen (z.B. Lernrate oder Iterationsanzahl). Außerdem unterscheidet man zwischen der Trainings- und Validierungsphase und der Ausführungsphase (oder auch Deployment). Beide Netzdefinitionen sind gleich, bis auf die letzten Schichten, die während des Trainings die Ausgabe und den *loss* bzw. die *accuracy* berechnen. Diese werden für das Deployment ersetzt, sodass nur die Klassenwahrscheinlichkeiten ausgegeben werden.

Neben diesen .prototxt Dateien wird während des Trainings eine weitere Datei erstellt, die .caffemodel Datei. Hier werden in regelmäßigen Abständen (Abstand wird im Solver definitiert) die Gewichte gespeichert. Die .caffemodel Datei und die deploy.prototxt werden außerdem für eine spätere Implementierung des Models in eine Softwareumgebung benötigt.

Dank des modularen Aufbaus von Caffe können die verwendeten Algorithmen auf dem aktuellen Stand gehalten oder ersetzt werden.

2.5.2 Nvidia Digits

Beim *NVIDIA Deep Learning GPU Training System* (kurz Digits) handelt es sich um ein Open Source Projekt, welches das Erstellen von Datensets, das Training neuronaler Netze und die Visualisierung dieser vereinfachen soll [13]. Digits arbeitet Job-basiert und besitzt eine Web Applikation, die die Steuerung von Caffe mittels Konsolenbefehlen ersetzt.

Da bestimmte Modelle feste Eingabegrößen voraussetzen, ermöglicht Digits das Zuschneiden oder Skalieren der Bilder während der Datenseterstellung. Des Weiteren gibt es die Option *fill*, welche genutzt werden kann, um Bilder in eine bestimmte Größe zu bringen, ohne sie zu verzerren oder zuzuschneiden. Es stehen außerdem verschiedene Datenbankformate wie *HDF5* und *LMDB* zur Verfügung. Die Erstellung von Datensets geschieht somit durch Digits und erfordert keine zusätzlichen Skripte vom Benutzer.

Eine weitere Vereinfachung der Handhabung von Caffe ist das automatische Generierung des Solvers, nachdem der Benutzer die Einstellungen dazu in der Modelldefinition ausgewählt hat. Auch die Erstellung der entsprechenden Dateien für die

Netzdefinitionen geschieht automatisch. Des Weiteren gibt es die Option Modelle zu visualisieren, wie es ausschnitthaft in Abbildung 2.10 zu sehen ist.

Durch die Abbildung von Echtzeit Informationen zu Accuracy und Loss laufender Jobs, ist es möglich, nötige Änderungen der Hyperparameter schnell aufzudecken. Somit wird der Prozess des Findens passender Hyperparameter beschleunigt.

Ein Nachteil ist jedoch, dass Digits auf einem *fork* von Caffe arbeitet und Neuerungen nicht immer eingepflegt werden.

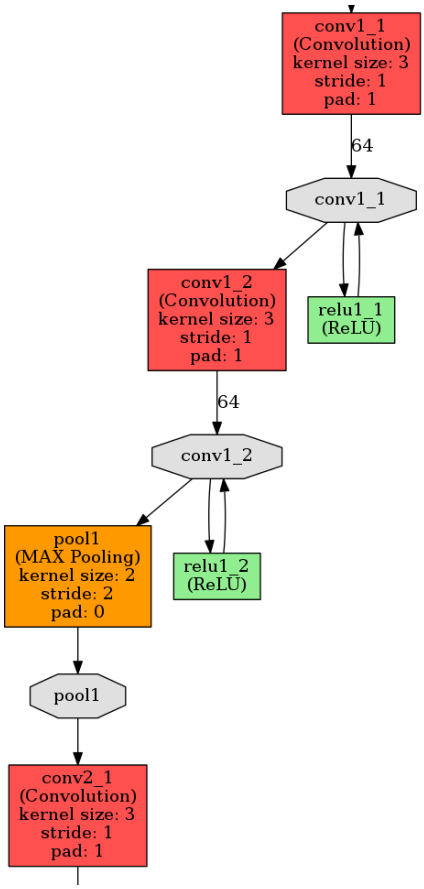


Abbildung 2.10: *Visualisierung der ersten drei Convolutionsschichten der VGG16 Architektur in Digits. Eine detailliertere Übersicht der Visualisierung der VGG16 Architektur ist im Anhang A.2 zu finden.*

KAPITEL

DREI

Daten: Römisch-imperiale Münzen

Münzen gehörten zu den ersten historischen Artefakten die untersucht wurden [56]. Hauptsächlich lag dies am steigenden Interesse der Antiquitätenhändler der Renaissance. Münzen waren und sind außerdem wichtige wirtschaftliche Komponenten jeder Regierung. In diesem Zusammenhang steht auch die Interpretation der Münze als offizielles Dokument, welches Entscheidungen und Handlungen der Regierung verkörpert. Im Kontext der römischen Kaiser lassen sich über Münzen Rückschlüsse auf Regierungszeit, politische und militärische Ereignisse und sogar Verschwörungen ziehen.

In den folgenden Kapiteln wird auf die Bedeutung der Münzen im römischen Reich eingegangen und die strukturellen Besonderheiten und Herausforderungen römischer Münzen erläutert.

3.1 Münzen als Propagandamittel und Zeugen der Vergangenheit

Bei den Münzen des Römischen Reichs handelte es sich um ein wichtiges Kommunikationsmittel der Regierung mit dem Volk. Besonders das Militär und die Bürger konnten so erreicht werden. Münz-Schenkungen erreichten spezielle Zielgruppen sehr schnell und umfassend. Gleichzeitig konnten über den allgemeinen Geldumlauf auch Informationen in entfernte Provinzen verteilt werden [45].

Außerdem sind Münzen zum Teil lange im Umlauf, weshalb Informationen auch für die Nachwelt sichtbar bleiben. Dadurch konnten Kaiser ihr Bild für die nachfolgenden Generationen prägen. Diese Bedeutungen war den Kaisern durchaus bewusst und daher wurden Münzen auch zu Propagandazwecken verwendet. Berühmt ist die Prägung von Marcus Iunius Brutus (siehe Abbildung 3.1). Auf der Vorderseite ist ein Porträt Brutus zu sehen, während auf der Rückseite zwei Dolche und eine Freiheitsmütze abgebildet sind. Außerdem ist die Inschrift “EID MAR“ zu lesen, welche “Iden des März“ bedeutet und auf den 15. März 44 v. Chr. hinweist. An diesem Datum kam es während einer Senatssitzung zur Ermordung von Gaius Julius Caesar. Die Münze soll die Befreiung vom Diktator Julius Caesar durch Brutus zeigen [11].

Abbildung 3.1: *Römische Münze mit Brutus auf der Vorderseite und der Inschrift "EID MAR" und zwei Dolchen als Hinweis auf das Attentat gegen Caesar [1][11]. Außerdem sind die Auswirkungen eines verschobenen Prägestempels zu sehen.*

Durch solche Prägungen konnten Prägeherren sowohl politische als auch militärische Errungenschaften unter ihrer Herrschaft propagieren und für die Nachwelt konservieren. Sie prägten damit also absichtlich das Bild ihrer Selbst beim Volk.

3.2 Struktur römischer Münzen

Römische Münzen besitzen, ähnlich wie heutige Münzen, eine Vorder- und Rückseite, welche bei allen Münzen eine sehr ähnliche Struktur aufweist. In der Numismatik wird die Vorderseite *Avers* und die Rückseite *Revers* genannt. Zu den wichtigsten Merkmalen der römischen Münzen zählen:

- Legende bzw. Inschrift
- Porträt
- Rückseitenbild
- Münzzeichen
- Münzstätte und Münzmeister
- Wertigkeit (Denomination)
- Material und Gewicht

Bei der **Legende** handelt es sich um die Schrift, die auf Vorder- und Rückseite zu finden ist und meist die Abbildungen umrahmt. Die Legende enthält unter anderem Informationen zum Kaiser, zur Autorität (in wessen Namen die Münze geprägt wurde) oder zur Stadt in der die Münze geprägt wurde [24].

Das **Porträt** auf der Vorderseite zeigt in vielen Fällen Kaiser, in einigen Fällen können aber auch Münzen mit dem Porträt vorangegangener Kaiser oder zu Familienmitgliedern geprägt worden sein. Oft stellten sich Kaiser auch mit Merkmalen von Göttern und Heroen dar, auf die sie ihre Herkunft zurückführen. Die Abbildung auf der Rückseite der Münzen nennt man **Rückseitenbild**. Hier können verschiedene Themengebiete behandelt werden. Abgebildet werden zum Beispiel Symbolbilder

(Bauwerke, Szenen etc.), Götter oder auch abstrakte Muster. Häufig ist die Personifikation von Städten (zum Beispiel die Darstellung der Roma als Personifikation Roms) oder die Darstellung politischer und militärischer Erfolge zu Propagandazwecken.

Das **Münzzeichen** ist eine Kombination verschiedener Abkürzungen und Zeichen. Es enthält meistens Informationen über die Stadt in der die Münze geprägt wurde und die **Prägestätte** der Münzserie. Das Münzzeichen kann auch noch weitere Merkmale enthalten, die zum Beispiel Rückschlüsse auf den **Münzmeister** oder die Prägezeit geben können [36]. Es ermöglichte bei Fehlern der Prägung, zum Beispiel falsche Gewichte der Münzen, die Münzen zurückzuverfolgen, um weitere Fehlprägungen zu verhindern. Nicht alle Münzen besitzen jedoch ein Münzzeichen.

Das **Material** und das **Gewicht** sind ebenfalls wichtige Merkmale römischer Münzen, da sie zusammen das Nominal bestimmen. Das Gewicht und die Reinheit der Silbermünzen sank über die Zeit im römischen Reich und hilft damit, bei einer Datierung der Münzen [7][35].

3.3 OCRE, RIC und Klassenbildung

Bei *Online Coins of the Roman Empire* (kurz OCRE) handelt es sich um ein Projekt welches die Identifikation, Katalogisierung und Erforschung römischer Münzen als Ziel hat. Die Katalogisierung erfasst die Münzen nach der Systematik der *Roman Imperial Coinage* (kurz RIC), dessen Katalog das Standardwerk auf diesem Gebiet ist. Es umfasst kaiserzeitliche Münzen von 31 v. Chr. (Kaiser Augustus) bis 491 n. Chr. (Kaiser Zenon).

Die Klassifizierung umfasst 10 Bände, das Schema ist jedoch innerhalb dieser Bände nicht konsistent. In den meisten Bänden (Band 1-5 und 10) sind die Münzen nach den Kaisern sortiert. Dort bildet sich der Klassenname aus dem Zusammensetzen von RIC Band, Kaisernamen und RIC Nummer. So entsteht zum Beispiel die Klasse *RIC III Antonius Pius 202b*.

In den anderen Bänden sind die Münzen jedoch nach Prägeort sortiert, was zu Klassen wie *RIC IX Constantinople 71D* führt. Des Weiteren gibt es Fälle in denen die Frauen der Kaiser eigene Einträge besitzen (*RIC V Salonina 68*) oder beim entsprechenden Kaiser als Autorität hinter der Münze eingeordnet werden (*RIC III Antonius Pius 327 (denarius)*).

Bei den OCRE Klassen handelt es sich um die gleiche Klassifizierung wie in den RIC Bänden. Die URI einer Münze setzt sich aus der URL `http://numismatics.org/ocre/id/` und der RIC Klasse (z.B. *ric.3.ant.202b*) zusammen [1].

[1] `http://numismatics.org/ocre/id/ric.3.ant.202b`

3.4 Herausforderungen

Im Gegensatz zu heutigen Münzen, sind römische Münzen mit zwei Münzstempeln von Hand geprägt worden, weswegen es zu einer großen Varianz innerhalb eines Münztyps kommen kann [24]. Je nach Abnutzung des Stempels, Abgegriffenheit und Oxidierung des Materials, unterscheiden sich Münzen nicht nur in ihrer Prägung, sondern auch in ihrem Erhaltungszustand. Ebenso konnte es durch das Prägeverfahren dazu kommen, dass der Prägestempel nicht zentriert auf dem Schrötling landete und es so zu Verschiebungen und fehlenden Bereichen des Stempelreliefs auf der Münze kam. Betrachtet man Abbildung 3.1 ist auf der Vorderseite die Legende unter dem Kopf von Brutus nicht mehr auf den Schrötling geprägt worden. Solche Verschiebungen können hier zu einer Herausforderung werden.

Des Weiteren liegt eine hohe Inter-Class Variation vor: Die Münzen innerhalb einer Klasse unterscheiden sich zum Teil stark voneinander. Gleichzeitig gibt es jedoch auch eine hohe Intra-Class Variation: Die Münzen verschiedener Klassen können sich sehr ähneln. Gut zu sehen ist dies in Abbildung 3.2.

Abbildung 3.2: *Beispiele aus den genutzten Daten für eine hohe Inter- und Intra-Class Variation. Münzen mit gleichem Kaiserporträt können sich stark unterscheiden, während Münzen mit Porträts verschiedener Kaiser eine starke Ähnlichkeit aufweisen können. Dies ist hier an den Geschwistern Caracalla und Geta zu sehen.*

Diese Herausforderungen führen auch zur zeit- und kostenintensiven Bearbeitung bei der manuellen Klassifizierung: Bei einem Arbeitstag von 8 Stunden, 250 Arbeitstagen im Jahr und einer Vergütung von 300€ pro Tag ergeben sich folgende Kosten:

Minuten pro Münze	15 min	30 min	60 min
Münzen pro Stunde	4	2	1
Münzen pro Tag	32	16	8
Münzen pro Jahr	8.000	4.000	2.000
Arbeitstage 1.000 Münzen	31,25	62,5	125
Kosten für 1000 Münzen	9.375 €	18.750 €	37.500 €

Tabelle 3.1: Übersicht über die entstehenden Kosten für den Klassifizierungsprozess bei verschieden schwierigen Münzen (Tabelle entnommen aus [25]).

KAPITEL

VIER

Methodik

In diesem Kapitel wird die Vorgehensweise zur Erstellung eines funktionsfähigen neuronalen Netzes vorgestellt. Dabei werden verschiedene Methoden und die Daten aus den vorangegangenen Kapiteln verwendet. Den Hauptteil dieses Kapitels bilden die Entwicklung der neuronalen Netze und die durchgeführten Experimente.

4.1 Aufgabenstellung und Zielsetzung

In einer vorangegangenen Arbeit [34] wurde der Einsatz von OpenCV, skaleninvarianter Merkmalstransformation (kurz *SIFT*) und dem Speeded Up Robust Features-Algorithmus (kurz *SURF*) für die Münzklassifizierung untersucht. Hierbei handelt es sich um Methoden des maschinellen Lernens. Der Vorteil dieser ist, dass sie sehr robust gegenüber Translation sind. Um weitere Methoden des maschinellen Lernens zu untersuchen, wird in der vorliegenden Arbeit der Einsatz neuronaler Netze für die Klassifizierung römischer Münzen untersucht. Dazu werden verschiedene Prototypen entwickelt, die anschließend miteinander verglichen werden. Die gewählte Architektur der Convolutional Neural Networks bringt außerdem ebenfalls den Vorteil der Robustheit gegenüber Translation mit sich und die Performance kann mit der vorangegangenen Arbeit verglichen werden.

Da neuronale Netze sehr viele Gewichte besitzen, ist es für einen Menschen nicht nachvollziehbar wie das Netz seine Entscheidungen fällt. Vergleichbar sind solche Systeme mit Black Box Systemen. Um eine Einsicht in die Funktionsweise neuronaler Netze zu erhalten, wird eine Unterteilung in zwei Aufgaben durchgeführt:

- **Ziel 1**: Porträt Identifizierung
- **Ziel 2**: OCRE-Typ Klassifizierung

Bei **Ziel 1** handelt es sich um eine, für Numismatiker, Teilaufgabe der OCRE-Typ Bestimmung. Selbst Laien gelingt es, wichtige Merkmale und Unterschiede der Porträts zu identifizieren. Nach dem Training eines Netzes ist es möglich, mit Hilfe von *Grad-CAM* (siehe Kapitel 2.4.3) zu visualisieren, welche Bereiche am stärksten zur Klassenbestimmung beigetragen haben. Bei der Verwendung in dieser Arbeit

dienen die Visualisierungen zur Feststellung, ob das Netz ebenfalls dazu fähig ist diese Merkmale zu erkennen. Außerdem bieten sie eine Möglichkeit, Einsicht in die Funktionsweise der neuronaler Netze zu bekommen.

Eine Alternative ist die Betrachtung der Legende oder Identifizierung des Münzzeichens. Jedoch wird an dieser Stelle das Porträt gewählt, da es einige Vorteile mit sich bringt.

Zum einen befindet sich das Porträt in der Mitte der Münze und ist somit fast immer komplett erhalten. Die Legende, die sich um das Porträt befindet, ist öfter abgenutzt oder komplett verschwunden und eignet sich daher weniger. Ein weiterer Grund der gegen die Legende spricht ist, dass die Legende zwar in OCRE abgeschrieben ist, also als Label vorliegt, jedoch keinerlei Markierung der Buchstaben im Bild existiert. Für die Zeichenerkennung liegen die Label aber häufig samt einer Markierung in den Bildern vor. Des Weiteren enthält die Legende oft den Name des Kaisers und somit entsteht eine Korrelation: Bilder mit gleichem Kaisernamen in der Legende haben häufig auch das gleiche Porträt. Aus diesem Grund kann nicht ausgeschlossen werden, dass das Porträt nicht ebenfalls vom neuronalen Netz genutzt wird. Eine Maßnahme, um diesen Effekt zu verhindern, wäre das Zuschneiden der Bilder, sodass entweder nur die Legende oder nur das Porträt übrig bleibt. Da die Legende als Halbkreis um das Porträt liegt, ist in diesem Fall ein Zuschneiden auf das Porträt einfacher.

Das zweite Ziel (**Ziel 2**) ist die Klassifizierung der Münzen nach den OCRE-Klassen. Bei dieser Aufgabe handelt es sich um eine wesentlich komplexere, da nicht nur das Porträt zur Klassenbildung beiträgt, sondern auch alle weiteren Details auf der Münzvorderseite und zusätzlich Merkmale der Münzrückseite.

Dieses Ziel wird gewählt, um zu evaluieren, ob neuronale Netze auch bei dieser komplexeren Aufgabenstellung in dieser Domäne erfolgreich verwendet werden können.

Da neuronale Netze als Methode verwendet werden, ist außerdem noch zu erwähnen, dass die *Annahme zur Weltabgeschlossenheit* (engl. *Closed-World-Assumption*) in dieser Arbeit gemacht wird. Konkret bedeutet das, dass das System nicht dazu fähig ist, ungesehene Klassen zu identifizieren und eine Klassifizierung nur in Klassen geschehen kann, die im Training bereits enthalten waren.

4.1.1 Überblick

Hauptziele:

- **Ziel 1 – Identifikation der Porträts:** Unterstützung der Vorsortierung für den Klassifizierungsprozess der Numismatiker
- **Ziel 2 – Klassifizierung der Münzen in ihre OCRE Typen:** Direkte Unterstützung des Klassifizierungsprozess der Numismatiker

Nebenergebnisse:

- **Datenset 1:** Bilder der Münzvorderseite und den Namen der abgebildeten Person (Ziel 1)
- **Datenset 2:** Bilder der Vorder- und Rückseite der Münzen und der zugehörigen OCRE Klasse (Ziel 2)

Nutzen (beider Ziele):

- Zeitersparnis durch den Einsatz neuronaler Netze (siehe 3.1)
- Angabe der Güte (durch die Wahrscheinlichkeit) der Ausgabe
- Visualisierung wichtiger Bildbereiche für das neuronale Netz (siehe 2.4.3)
- Evaluation der Performance (siehe 2.4.1) der Netze
- Verbesserung der Performance (siehe 2.4.1) gegenüber SIFT und SURF

4.1.2 Vorgehensweise

Beim Training eines neuronalen Netzes handelt es sich um eine sehr Zeitintensive Aufgabe. Die Trainingsdauer der bekannten AlexNet Architektur wird von den Entwicklern dieses Convolutional Neural Network, auf zwei Nvidia GTX 580 mit 3GB Grafikspeicher, mit fünf bis sechs Tagen angegeben [40]. Um die Funktionsweise verschiedener Architekturen zu verstehen und möglichen Restriktionen und Herausforderungen bei der Parameterwahl aufzudecken, sollte das vollständige Training auf allen vorhandenen Daten also vorerst vermieden werden. Aus diesem Grund setzt sich das Vorgehen bei der Erstellung der CNNs in dieser Arbeit aus drei Stufen zusammen: den Voruntersuchungen, der Entwicklung und schließlich der Implementierung. Eine schematische Darstellung dieser Stufen ist in Abbildung 4.1 zu sehen.

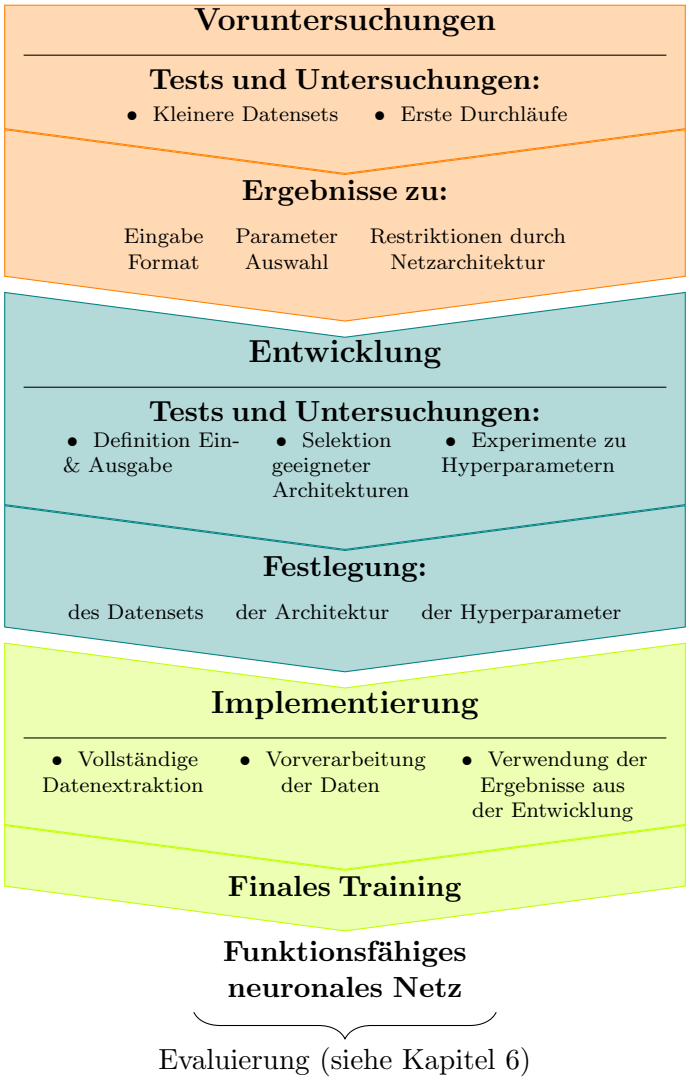


Abbildung 4.1: *Vorgehensweise zur Erstellung des funktionsfähigen neuronalen Netz*

Für die Voruntersuchungen wird lediglich ein kleiner Teil der zur Verfügung stehenden Bilddateien der Datensets verwendet. In den Voruntersuchungen sind unter anderem Restriktionen bei den Eingabedaten und ein Klassenungleichgewicht aufgedeckt worden. Genaueres zum Ablauf der Voruntersuchgen und zu weiteren Ergebnissen dieser ist in 4.2 zu lesen.

Nach den Voruntersuchungen folgt die Entwicklung. Hierbei werden alle vorhandenen Bilddateien genutzt. Auf Basis dieser, werden zwei verschiedene Architekturen (AlexNet und VGG16) untersucht und Experimente zur Wahl der Hyperparameter durchgeführt. Ziel der Entwicklung ist es, die passende Architektur und deren Hyperparameter zu finden, die die Anforderungen aus 4.1.1 am Besten erfüllen. Die Validierung geschieht hier mit der Validierungsmenge. Die Durchführung und Ergebnisse dieser Stufe sind in 4.3 nachzulesen.

In der letzten Stufe, der Implementierung, erfolgt dann das abschließende Training

mit der vorher gewählten Architektur und den ermittelten Parametern. Das Kapitel 5 enthält alle Informationen zu den hierfür entwickelten Programmen, zur Arbeitsumgebung und zum Trainingsprozess.

4.2 Voruntersuchungen

Wie bereits erwähnt, dienen die Voruntersuchungen dazu, bestimmte Restriktionen der Parameter aufzudecken und einen ersten Eindruck der Funktionsweise von CNNs in Nvidia Digits und Caffe zu erhalten. Um die Voruntersuchungen so einfach wie möglich zu halten, wird das in Nvidia Digits bereits vordefinierte AlexNet mit den vorgeschlagenen Hyperparametern trainiert.

Für das Training wird ein Datenset für jedes Ziel benötigt. Da solche Datensets bisher nicht existierten, entstehen im Zuge dieser Arbeit zwei Datensets. Zur Erstellung dieser ist es nötig die Datenpaare, bestehend aus Bild und zugehöriger Klasse, herunterzuladen. Dazu ist eine leicht abgeänderte Java Applikation aus [34] verwendet worden. Je nach Datenset muss in dieser Applikation die *SPARQL* und die Ordnerstruktur verändert werden. Beides gleicht in der Voruntersuchung, bis auf die Menge der Daten, der tatsächlichen Implementierung, welche in in Kapitel 5.1.1 beschrieben wird.

4.2.1 Erstellung erster Datensets

AlexNet erwartet als Eingabe Bilder der Größe 227×227 Pixel. Für **Ziel 1** müssen dazu die bereits quadratischen Bilder der Münzvorderseite von OCRE heruntergeladen werden und auf die benötigte Größe skaliert werden. Die Skalierung kann in Nvidia Digits durchgeführt werden. Die für Ziel 1 entstanden Datensets sind mit *CoinPortraitData* benannt[1].

Abbildung 4.2: *fill-Operation angewendet auf Münze der Klasse RIC VII Antioch 88.*

Für **Ziel 2** muss das Bild einer Münzvorderseite mit dem Bild der entsprechenden Rückseite zusammengeführt werden. Das so erzeugte Bild ist jedoch nicht quadratisch. Ein Zuschneiden bringt einen Informationsverlust mit sich, da in jedem Fall ein Teil der Münzen abgeschnitten werden muss. Aus diesem Grund wird die in Digits verfügbare Option *fill* bei der Erstellung des Datensets genutzt. Diese Option skalierte das Bild auf eine Breite von 227 Pixel und füllte die fehlenden Pixel in der Höhe mit zufälligem Rauschen auf, was zu einem 227×227 Pixel großen Bild führt. Ein so erzeugtes Bild kann man in Abbildung 4.2 sehen.

[1] die Datensets die im Anschluss dieser Arbeit verfügbar sind, beinhalten die Bilder in ihrer Originalgröße, sodass kein Informationsverlust entsteht und anknüpfende Arbeiten die Skalierung selbst wählen können

Der aufgefüllte Bereich enthält keine Muster, die vom Netz verwendet werden können und die Performance daher nicht wesentlich beeinflussen. Das Datenset für Ziel 2 ist mit *OCRECoinClasses* benannt.

4.2.2 Erste Durchläufe

Für die Voruntersuchungen kann noch nicht der Computer mit den Titan X Grafikkarten genutzt werden, welcher für das spätere Training verwendet wird (siehe Kapitel 4.3). Stattdessen wird ein Computer mit folgender Konfiguration verwendet:

- Arbeitsspeicher: 24GB
- Grafikkarte: Nvidia GTX 780 (2GB)
- Prozessor: Intel Core i7-4770 (3.40GHz)

Während der ersten Anwendungen von AlexNet auf den Daten für **Ziel 1** stellte sich heraus, dass einige wenige Klassen viel mehr Münzen enthalten als die restlichen Klassen und einige Klassen nur sehr wenige Bilder enthalten. Dieses sogenannte **Klassen Ungleichgewicht** ist im Datenset *CoinPortraitData* am größten: Die Klasse *Constantine I* zum Beispiel enthält 3085 Bilder, während die restlichen Klassen durchschnittlich 173 Bilder enthalten. Beim Datenset für **Ziel 2** (*OCRECoinClasses*) ist das Ungleichgewicht deutlich geringer. Außerdem sind bei diesem Datenset insgesamt nur 7438 Bilder enthalten (bei *CoinPortraitData* sind es 21.137).

Ein Ungleichgewicht kann schnell zu einer Überanpassung (siehe 2.1.3) führen [30][54]. Ein Entfernen von Daten, um ein Gleichgewicht zu erzeugen, führt jedoch auch zum Verlust von Trainingsdaten.

Um die Auswirkungen des Ungleichgewichts bei **Ziel 1** zu untersuchen, werden zwei Datensettypen erstellt:

- **Datenset Ziel 1a**: *CoinPortraitData* ohne Begrenzung der maximalen Bilder
- **Datenset Ziel 1b**: *CoinPortraitData* mit maximaler Bilderanzahl pro Klasse

Wobei es sich bei der maximalen Bilderanzahl in *1b* um einen Parameter handelt, der in Experiment 1 (siehe 4.4.1) untersucht wird. Da das *OCRECoinClasses* Datenset für **Ziel 2** bereits sehr viel weniger Bilder enthält und das Ungleichgewicht nicht so groß ist wie im Datenset für Ziel 1, wird beim zweiten Datenset darauf verzichtet Bilder zu entfernen.

Eine weitere bereits aufgedeckte Herausforderung ist, dass ein Training von AlexNet ohne vortrainierte Gewichte aufgrund der Trainingszeit nicht möglich ist. Das Training mit eigener Initialisierung der Gewichte hätte den Vorteil, dass AlexNet auch auf anderen Bildgrößen verwendet werden könnte.

Das Training ohne vortrainierte Gewichte zeigt jedoch selbst nach 16 Stunden noch keine nennenswerten Veränderungen bei Kostenfunktion und Accuracy. Das Trai-

ning wird daher abgebrochen und ein **vortrainiertes Caffemodel** verwendet (siehe 4.3.2).

Trotz der Verwendung dieses vortrainierten Caffemodels, dauert ein Trainingsdurchlauf mit 30 Epochen mehrere Tage. Unter anderem kann nicht auf der Grafikkarte gerechnet werden: Der benötigte Speicherplatz für dieses Netz ist größer als der verfügbare Grafikkartenspeicher.

4.2.3 Ergebnisse der Voruntersuchungen

Neben den Restriktionen der Bildgröße und dem hohen Speicherbedarf, die bereits vor und während dem ersten Training auffallen, liefern die Ergebnisse des ersten Trainings weitere Hinweise zur Verbesserung.

Bei **Ziel 1** fällt auf, dass viele der Klassen nicht richtig klassifiziert werden. Im Gegensatz dazu, werden einige wenige sehr häufig erkannt und auch zu oft ausgegeben. Bei den Klassen, die gar nicht erkannt werden, handelte es sich um die Klassen, bei denen es nur sehr wenige Bilder in der Trainingsmenge gibt. Erst ab einer Menge von ca. 60 bis 70 Bildern innerhalb der Klasse, werden diese Klassen auch ausgegeben. Da im späteren Verlauf die gesamten Daten verwendet werden und außerdem noch eine Aufspaltung in Trainings-, Test- und Validierungsmenge vorgenommen wird, muss an dieser Stelle eine minimale Anzahl an Bildern im Porträt Datenset von 100 pro Klasse gesetzt werden.

Beim Datenset für **Ziel 2** (OCRE Klassen), sind durchschnittlich deutlich weniger Bilder in den Klassen vorhanden. Die Mindestanzahl an Bildern pro Klasse wird hier auf 20 gesetzt. Dadurch können sehr viele Klassen erhalten bleiben und trotzdem bereits in den Voruntersuchungen ähnliche Ergebnisse wie bei den Voruntersuchungen zu Ziel 1 erzielt werden.

In Abbildung 4.3 sind außerdem die Ergebnisse der Voruntersuchung in Stichpunkten zusammengefasst.

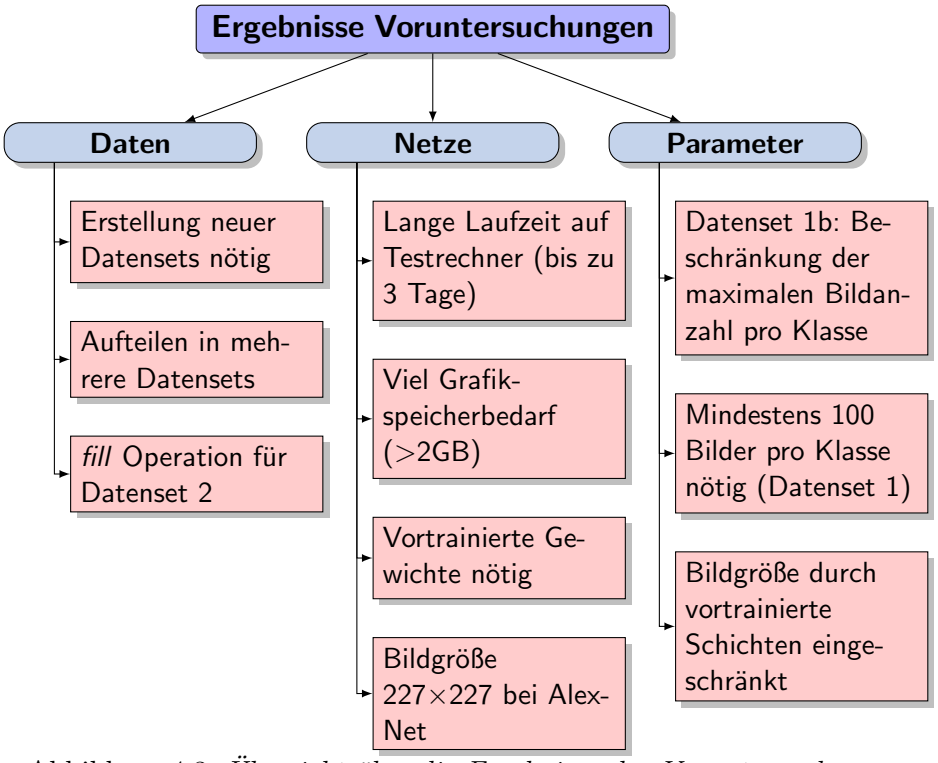


Abbildung 4.3: *Übersicht über die Ergebnisse der Voruntersuchungen*

Nachdem die Schritte der Voruntersuchung und die resultierenden Ergebnisse erklärt wurden, folgt im nächsten Kapitel die anknüpfende konkrete methodische Vorgehensweise.

4.3 Entwicklung

In diesem Abschnitt wird auf die methodische Vorgehensweise bei der Entwicklung eines Modells eingegangen, ehe in den anschließenden Kapiteln die Implementation und Evaluation gezeigt wird. Die Entwicklung eines Systems, welches auf maschinellem Lernen basiert, umfasst üblicherweise [22][26] folgende Schritte:

1. Definition der Ein- und Ausgabe (Auswahl oder Erstellung eines Datensets)
2. Selektion geeigneter Architekturen
3. Auswahl der Hyperparameter
4. Validierung
5. Modell Evaluation

Je nachdem welche Ergebnisse die einzelnen Schritte liefern, kann es nötig sein, den gesamten Vorgang oder Teile zu iterieren.

Da für die Entwicklung das vollständige Datenset verwendet wird, wird für das Training in der Entwicklung ein leistungsstärkerer Computer verwendet. Genaueres zu der verwendeten Arbeitsumgebung ist in Kapitel 5.2 nachzulesen.

Nun im Detail zu den einzelnen Schritten.

4.3.1 Definition der Ein- und Ausgabe

Bei den Ein- und Ausgabedaten handelt es sich um die Daten, die während des Trainings, der Validierung und der Evaluation genutzt werden. Die Eingabe ist in diesem Fall ein Bild einer Münze, während die Ausgabe, je nach Ziel, der Name des Kaisers des Porträts oder die OCRE Klasse ist. In den Voruntersuchungen ist festgestellt worden, dass es derzeit keine fertigen Datensets gibt, sondern sowohl für **Ziel 1** als auch **Ziel 2** neue Datensets erstellt werden müssen. Die Erstellung eines Datensets beinhaltet die folgenden Teilschritte, welche anschließend konkret erklärt werden:

- Festlegung der Datenmenge
- Vorverarbeitungsschritte
- Aufteilung in Trainings-, Validierungs- und Testmenge

Festlegung der Datenmenge
Die Menge der Daten ist abhängig vom später gewählten Ansatz, generell lässt sich aber sagen, dass genügend Daten vorhanden sein müssen, damit es dem Modell gelingen kann Rückschlüsse zu ziehen [26]. Für die Menge benötigter Bilder werden vergleichbare Datensets als Referenzwert verwendet. In Tabelle 4.1 sind dazu verschiedene Datensets aufgeführt. Keines dieser Datensets enthält weniger als 50.000 Bilder in den Trainingsdaten und bei allen Datensets handelt es sich um Datensets, welche auch für Wettbewerbe genutzt werden. In den letzten Jahren gewannen außerdem immer neuronale Netze diese Wettbewerbe [17][5], weswegen sich diese als Referenzwerte eigneten.

Datenset	Klassen	Trainingsmenge	Testmenge	Bildgröße
MNIST [17]	10	60.000 Bilder	10.000 Bilder	28×28
ImageNet	21.841	14.197.122 Bilder	keine Angaben	variiert
CIFAR-10 [5]	10	50.000 Bilder	10.000 Bilder	32×32
CIFAR-100 [5]	100	50.000 Bilder	10.000 Bilder	32×32
CoinPortraitData	69	21.137 Bilder	7045 Bilder	variiert
OCRECoinClasses	195	4638 Bilder	1400 Bilder	variiert

Tabelle 4.1: Übersicht über die Anzahl an Klassen, Bildern pro Menge und Bildgrößen verschiedener Datensets

In der Realität sind aber oft nicht unbegrenzt viele Daten verfügbar, weswegen zuerst geprüft werden muss, wie viele Datenpaare (bestehend aus Münzbild und

Klassenbezeichnung) überhaupt gesammelt werden können. Bei einer Anzahl von mindestens 100 Bildern im Porträt Datenset (**Ziel 1**) ergibt die Extraktion der Daten eine Menge von insgesamt 35.227 Bildern. Beim Datenset für die OCRE Klassen (**Ziel 2**) sind es bei mindestens 20 Bildern pro Klasse insgesamt 7438 Bilder. Diese Daten sind ebenfalls in Tabelle 4.1 eingetragen und zeigen den Unterschied zu den Referenzwerten der anderen Datensets. In Tabelle 4.2 und 4.3 sind außerdem die restlichen Werte zu allen Datensets dieser Arbeit gelistet.

Wegen dem Ungleichgewicht des *CoinPortraitData* Datensets, werden hier mehrere Versionen mit verschiedenen Begrenzung der maximalen Bildanzahl erstellt. Bei der Begrenzung handelt es sich um einen Hyperparameter, welcher nicht vorhergesagt werden kann, sondern durch Experimente festgelegt werden muss. Details zu den Ergebnissen dieser Experimente ist im Abschnitt 4.4.1 Experiment 1 nachzulesen.

Vorverarbeitungsschritte
Neben der Datenmenge ist auch die Datenqualität ein wichtiges Kriterium zur Auswahl der Daten. Bei Bilddaten muss die Auflösung hoch genug sein, damit wichtige Merkmale erkennbar sind. Die Voruntersuchungen (siehe 4.2) ergaben jedoch, dass die Wahl eines vortrainierten Models zu einer Einschränkung der Bildgröße führt. Im weiteren Verlauf werden AlexNet und VGG16 mit vortrainiertem Caffemodel verwendet und Bilder in der Größe 227×227 (für AlexNet) und 224×224 (für VGG16) benötigt.

Da es sich bei den Quellen der Münzbilder um wissenschaftliche Quellen handelt, kann davon ausgegangen werden, dass das Format der Bilder und die Position der Münzen einer Quelle nicht sehr stark variiert. Lediglich bei einigen Quellen lassen sich hier Abweichungen finden. Ein Beispiel hierfür ist in Abbildung 4.4 zu sehen. Diese Unterschiede können zu einem Bias führen und vom Netz als Merkmal für die-

Abbildung 4.4: *Unterschied der Münzgröße und der Hintergrundfarbe verschiedener Quellen. Die linke Münze (weißer Hintergrund, minimaler Rand) ist aus dem British Museum, während die rechte Münze (grauer Hintergrund mit Skala, größerer Rand) Harvard Art Museum stammt [14]*

se Klasse von Münzen gewertet werden. Eine Möglichkeit um solchen oder ähnlichen Abweichungen vorzubeugen, ist der Bezug aus einer Quelle, anstatt aus mehreren. Im Falle der OCRE Münzen werden nur Bilder der Website Numismatics.org verwendet (URLs zu Bilder die mit `http://numismatics.orgcollectionimages/` beginnen). Da aber auch bei dem Bezug aus einer Quelle zum Teil unterschiedliche Auflösungen vorliegen, musste die Auflösung vereinheitlicht werden.

Aufteilung in Trainings-, Validierungs- und Testmenge
Das Verhältnis von Trainings-, Validierungs- und Testmenge ist je nach Datenset sehr unterschiedlich, hierbei handelt es sich um ein eigenes Forschungsgebiet im maschinellen Lernen [31]. Deshalb wird an dieser Stelle die Empfehlung aus [44] genutzt. Das Verhältnis der einzelnen Mengen ist dabei 60% Trainings-, 20% Validierungs- und 20% Testdaten.

Bei den erstellten Datensets handelt es sich zum einen um die Datensets für **Ziel 1**, mit verschiedener Begrenzung der maximalen Bildanzahl pro Klasse, um die Auswirkungen auf eine mögliche Über- oder Unteranpassung durch diese Begrenzungen zu untersuchen. Da die Testdaten immer gleich sein müssen, wird das oben genannte Verhältnis für das Datenset mit allen Bildern bestimmt und so auch für die Bilder der kleineren Datensets beibehalten:

Max. Bildanzahl pro Klasse	200	300	1000	alle Bilder
Klassen	69	69	69	69
Min. Bildanzahl pro Klasse	100	100	100	100
Trainingsmenge	10.006	12.956	19.739	21.137
Validierungsmenge	7045	7045	7045	7045
Testmenge	7045	7045	7045	7045

Tabelle 4.2: **Datenset Ziel 1a und 1b:** *CoinPortraitData_min100_noMax* und *CoinPortraitData_min100_max200*, *max300* und *max1000*

Und zum anderen um das Datenset für **Ziel 2**, die Klassifizierung der Münzen in die OCRE Klassen:

Bilder (insg.)	7438
Klassen	195
Durchschnittliche Anzahl an Bildern	173
Klasse mit den meisten Bildern	ric.5.tet_i.270 (235)
Min. Bildanzahl pro Klasse	20
Trainingsmenge	4638
Validierungsmenge	1400
Testmenge	1400

Tabelle 4.3: **Datenset Ziel 2:** *OCRECoinClasses*

4.3.2 Selektion einer geeigneten Architektur

Viele Entwickler stellen ihre trainierten Modelle samt der Gewichte online zur Verfügung. Für Caffe gibt es den *Model Zoo* [55], in dem viele bekannte Modelle aufgelistet sind

und sich auch die Gewichte dieser Modelle finden lassen. In den Voruntersuchungen wurden erste Ergebnisse mit AlexNet erzielt, weswegen AlexNet die erste Wahl ist. AlexNet gewann außerdem 2012 die ImageNet Large Scale Visual Recognition Challenge. Heute wird AlexNet seltener eingesetzt, der Vergleich mit der VGG16 Architektur, die diese Challenge im Jahr 2014 gewann, ist jedoch aufgrund der Unterschiede interessant. Diese Architektur wird ebenfalls für diese Arbeit ausgewählt.

Weitere heute verbreitete Architekturen sind ResNet [32] und Inception v4 [50], diese basieren jedoch auf dem Konzept der Residual Networks. Diese ermöglichen eine noch größere Tiefe der neuronalen Netze. Bei dieser Arbeit soll die Machbarkeit des Einsatzes neuronaler Netze auf römisch-imperialen Fundmünzen untersucht werden, weswegen die einfache AlexNet und die direkt vergleichbare VGG16 Architektur gewählt werden.

Eine Voraussetzung an die Architekturen ist die Verfügbarkeit vortrainierter Gewichte. Worum es sich dabei handelt, wird nun im nächsten Abschnitt erklärt.

Vortrainierte Modelle

In den Voruntersuchungen dauerte das Training von 30 Epochen mit AlexNet ohne vortrainierte Gewichte mehrere Tage. Da sich bis dahin keine Veränderungen in der Kostenfunktion und der Accuracy zeigten, wurde das Training abgebrochen. Die Autoren sprechen selbst von einer Trainingszeit von fünf bis sechs Tagen (bei einem parallelisierten Training auf zwei GPUs mit jeweils 3GB Speicher).

Bei einem vortrainierten Model werden für die Initialisierung der Gewichte der ersten Schichten, Gewichte eines bereits gelernten Models verwendet. Das sogenannte *Transferlernen* ist ein übliches Vorgehen in diesem Bereich, da die Neuronen der ersten Schichten auf einfache Strukturen (oder Farbflächen) reagieren und sich diese in vielen Domänen ähneln [19][57][58].

Sogar wenn das Training mit selbst initialisierten Gewichten erfolgreich ist, kann die Nutzung von vortrainierten Gewichten zu einer besseren Generalisierungsfähigkeit des Netzes führen [57].

In Caffe und Nvidia Digits werden die Gewichte der Modelle in der `.caffemodel` Datei gespeichert. Die Gewichte und Architekturdefinitionen der ausgewählten Architekturen, AlexNet und VGG16, sind ebenfalls im Caffe Model Zoo zu finden.

AlexNet

Durch AlexNet begann das kommerzielle Interesse an CNNs stark zu steigen und immer mehr und auch komplexere CNN Variationen entstanden[26][40].

Architektur
AlexNet besteht aus 5 Convolutionsschichten, gefolgt von 3 vollständig verbundenen Schichten und einer 1000 Klassen Softmax Ausgabe.

Die erste Convolutionsschicht erhält als Eingabe $227 \times 227 \times 3$ Bilder. Die Filtergröße der ersten Schicht ist $F = 11$ und die Schrittgröße des Filters ist $s = 4$. Die Anzahl der Neuronen in dieser Schicht ist $55 \times 55 \times 96 = 290.400$ [2]. Die Anzahl der Parameter lässt sich aus der Filtergröße und der Anzahl dieser errechen: $(11 \times 11 \times 3 + 1) \times 96 = 34.944$. Insgesamt besitzt AlexNet 62 Millionen Parameter. Der Aufbau der folgenden Schichten ist in Tabelle 4.4 zu sehen.

Als Aktivierungsfunktion wurde ReLU gewählt, da diese, bei einem CNN mit vier Schichten, sechsmal so schnell konvergiert wie die *tanh*-Aktivierungsfunktion.

Zusätzlich wurde die max-Pooling Methode verwendet. Das Pooling geschieht nach der ersten bzw. zweiten ReLU Aktivierungsfunktion der ersten bzw. zweiten Convolution und vor der ersten vollständig verbundenen Schicht. Außerdem wurde nach der ersten und zweiten vollständig verbundenen Schicht jeweils die Dropout Technik verwendet.

Zwar ist dank Verwendung von ReLUs keine Normalisierung mehr nötig, trotzdem wurde nach der ersten und zweiten Convolution (nach der ReLU und vor dem max-Pooling) eine local response Normalisierung (kurz LRN) durchgeführt[40].

VGG16

Eine weiteres erfolgreiches CNN ist VGG16 von der Visual Geometry Group der Universität Oxford[47]. Mit dieser Architektur untersuchten die Autoren die Auswirkung der Tiefe bei gleichzeitig kleinerer Filtergröße der Convolutionsschichten. Neben VGG16 gibt es auch VGG19, wobei die Zahlen in den Namen sich auf die Netztiefe beziehen.

Architektur
VGG16 besteht aus 13 Convolutionsschichten, ebenfalls gefolgt von 3 vollständig verbundenen Schichten und einer 1000 Klassen Softmax Ausgabe.

Bei der Eingabe dieses Netzes handelt es sich um Bilder der Größe $224 \times 224 \times 3$. Die Filtergröße der ersten Convolution ist $F = 3$ und das Padding wurde mit 1 angegeben. Die Anzahl der Filter der ersten Schicht ist demnach 64. Daraus ergeben sich für die Anzahl der Parameter in der ersten Schicht $(3 \times 3 \times 3) \times 64 = 1728$. In dieser ersten Schicht sorgen die kleineren Filter für weniger Parameter als in AlexNet. Dadurch, dass in VGG16 jedoch viel mehr Schichten hintereinander liegen, führt dies zu insgesamt 138 Millionen Parametern und damit mehr als doppelt so vielen Parametern als in AlexNet.

In allen Convolutionsschichten wurde ebenfalls die ReLu Aktivierungsfunktion und für das Pooling die max-Pooling Methode verwendet. Jedoch wurde auf die Normalisierung durch LRN komplett verzichtet. Die Autoren fanden in ihren Experimenten heraus, dass die Verwendung dieser Normalisierung zu einem höheren Speicherverbrauch und einem längeren Training führte, während die Ergebnisse dadurch nicht verbessert wurden[47]. Der genaue Aufbau von VGG16 ist in Tabelle 4.4 zu sehen.

[2] $\frac{227-11}{4+1} = 55$

	AlexNet				VGG16		
Größe	**Schicht**	**Filter**	**Parameter**	**Größe**	**Schicht**	**Filter**	**Parameter**
[227x227x3]	Input			[224x224x3]	Input		
[55x55x96]	Conv1	11x11		[224x224x64]	Conv1_1	3x3	
	ReLu		34.944		ReLu		1.728
[27x27x96]	Max Pooling	3x3		[224x224x64]	Conv1_2	3x3	
[27x27x96]	Norm1				ReLu		36.864
[27x27x256]	Conv2	5x5		[112x112x64]	Max Pooling	2x2	
	ReLu		614.656	[112x112x128]	Conv2_1	3x3	73.728
[13x13x256]	Max Pooling	3x3			ReLu		
[13x13x256]	Norm2			[112x112x128]	Conv2_2	3x3	
[13x13x384]	Conv3	3x3			ReLu		147.456
	ReLu		885.120	[56x56x128]	Max Pooling	2x2	
[13x13x384]	Conv4	3x3		[56x56x256]	Conv3_1	3x3	
	ReLu		1.327.488		ReLu		294.912
[13x13x256]	Conv5	3x3		[56x56x256]	Conv3_2	3x3	
	ReLu		884.992		ReLu		589.824
[6x6x256]	Max Pooling	3x3		[56x56x256]	Conv3_3	3x3	
	Dropout				ReLu		589.824
[4096]	FC6			[28x28x256]	Max Pooling	2x2	
	ReLu		37.748.736	[28x28x512]	Conv4_1	3x3	
	Dropout				ReLu		1.179.648
[4096]	FC7			[28x28x512]	Conv4_2	3x3	
	ReLu		16.777.216		ReLu		2.359.296
	Dropout			[28x28x512]	Conv4_3	3x3	
[1000]	FC8		4.096.000		ReLu		2.359.296
				[14x14x512]	Max Pooling	2x2	
Insgesamt:			**62.369.152**	[14x14x512]	Conv5_1	3x3	
					ReLu		2.359.296
				[14x14x512]	Conv5_2	3x3	
					ReLu		2.359.296
				[14x14x512]	Conv5_3	3x3	
					ReLu		2.359.296
				[7x7x512]	Max Pooling	2x2	
				[4096]	FC6		
					ReLu		102.760.448
					Dropout		
				[4096]	FC7		
					ReLu		16.777.216
					Dropout		
				[1000]	FC8		4.096.000
				Insgesamt:			**138.344.128**

Tabelle 4.4: Vergleich des Aufbaus und der Parameter der Schicht in AlexNet[40] und VGG16[47]

Unterschiede und Gemeinsamkeiten beider Architekturen

Nachdem der Aufbau beider Architekturen erläutert wurde, kann nun auf die Unterschiede und Gemeinsamkeiten beider eingegangen werden.

Filtergröße
Bei AlexNet sind die Filter der ersten beiden Convolutionsschichten größer als die Filter der VGG16 Architektur, die eine einheitliche Größe von 3×3 besitzen (AlexNet: Conv1 11×11 bzw. Conv2 5×5). Wobei weiterhin noch zu erwähnen ist, dass bei einem 3×3 Filter und einer Stride von $s = 1$ bereits bei einer Tiefe von zwei, ein rezeptives Feld von 5×5 erreicht wird. Bei einer Tiefe von drei ist es bereits ein rezeptives Feld der Größe 7×7.

Tiefe des Netzes
Nicht nur die Filtergröße unterscheidet sich, sondern auch die Anzahl der Convolutionsschichten. Bei AlexNet gibt es insgesamt fünf, während es bei VGG16 13 sind. Die Tiefe des Netzes beeinflusst unter anderem die Trainingsdauer, weswegen das Training von AlexNet durchschnittlich schneller ist, als das Training von VGG16. In den letzten Jahren gab es einen Trend dazu, die Tiefe der Netze immer größer werden zu lassen. Es kann allerdings nicht davon gesprochen werden, dass Tiefere Netze grundsätzlich besser sind, denn diese werden auch anfälliger für eine Überanpassung [49][51].

Struktur (Normalisierung, Pooling und Aktivierungsfunktion)
VGG16 verzichtet komplett auf LRN, während in AlexNet sowohl nach der ersten und nach der zweiten Convolution LRN angewendet wird. Weiterhin unterscheidet sich die Position und Anzahl der Pooling Operationen. Als Pooling wird bei beiden das max-Pooling verwendet. Eine weitere Gemeinsamkeit beider Architekturen ist die Verwendung der ReLU Aktivierungsfunktion.

Gründe für die Auswahl

Der Hauptgrund für die Auswahl ist das Vorhandensein der .prototxt Architekturdefinition und vortrainierter .caffemodel Gewichte. Dadurch wird das zeitaufwändige Training mit selbstinitialisierten Gewichten eingespart. Dies kann außerdem zu einer bessere Generalisierungsfähigkeit der Netze führen.

Ein weiterer Grund ist die Platzierung beider Architekturen bei der ILSVRC 2012 bzw. 2014 auf den vordersten Plätzen.

Da sich die Tiefe und auch die Größe der Filter beider Netze unterscheidet, kann so außerdem geprüft werden, welche Auswirkungen dies bei der Anwendung auf römische Münzen und ihre speziellen Herausforderungen (siehe Kapitel 3.4) hat.

Außerdem kann zusätzlich geprüft werden, ob beide Netze unterschiedlich auf das hohe Klassenungleichgewicht reagieren. Dies kann einen Hinweis auf die nötige Tiefe geben, falls beide Netze nicht die erwünschten Ergebnisse erzielen und ein weiteres Netz erstellt werden muss.

4.3.3 Auswahl der Hyperparameter

Nachdem im vorangegangenen Abschnitt die Definition der Ein- und Ausgabe sowie eine Selektion geeigneter Architekturen erfolgte, handelt dieser Abschnitt von der Auswahl der noch fehlenden Hyperparameter. Speziell die Parameter des Trainingsprozesses werden genauer erklärt und deren Auswahl für die Implementierung erläutert. Diese Trainingsparameter werden im sogenannten Solver spezifiziert (siehe 2.5.1). Digits bietet dafür in seiner Web Applikation verschiedene Optionen, anhand derer später der Solver automatisch generiert wird. Für die Problemstellung dieser Arbeit wichtige Parameter sind:

- Trainings Epochen und Intervall der Zwischenspeicherung
- Validierungsintervall
- Batchgröße
- Optimierungsmethode
- Basis Lernrate
- Anpassung der Lernrate über die Zeit

Welche Optionen hier zur Verfügung stehen und welche Auswirkungen die einzelnen Parameter auf das Training haben, ist in den nächsten Abschnitten erklärt. Für die Auswahl der besten Parameter werden mehrere Experimente durchgeführt. Diese sind im Kapitel 4.4.3 beschrieben.

Trainings Epochen und Intervall der Zwischenspeicherung
Beim ersten Parameter handelt es sich um die Anzahl an Epochen, also wie oft die Trainingsdaten durch das Netz propagiert werden. Bei einer Epoche handelt es sich um *einen* vollständigen Durchlauf aller Trainingsdaten durch das Netz.

Ist die Anzahl der Epochen zu gering, bricht das Training ab, bevor die besten Filter bzw. Gewichte gelernt wurden. Es kommt zur Unteranpassung. Dieser Effekt zeigt sich auch schon in den Voruntersuchungen: Die Accuracy aller Modelle steigt auch nach mehreren Epochen noch an.

Ein weiterer Effekt, der mit steigender Anzahl an Epochen einhergeht, ist jedoch die Überanpassung. Ist die Anzahl der Epochen zu hoch, steigt zwar die Accuracy auf der Trainingsmenge weiter an, jedoch sinkt die Accuracy auf der Validierungs- und Testmenge.

In Digits gibt es die Möglichkeit festzulegen, in welchem Intervall (in Epochen) eine Zwischenspeicherung erfolgen soll. Diese Zwischenspeicherung enthält alle wichtigen Informationen, um das Netz zu testen und später auch einzusetzen. Sollte es zu einer Überanpassung kommen, könnten dadurch frühere Epochen extrahiert werden, in denen noch keine Überanpassung stattgefunden hat. Da der verfügbare Speicherplatz ausreichend ist, um auch bei 30 Epochen eine Zwischenspeicherung nach jeder Epoche durchzuführen, wird an dieser Stelle ein Intervall von einer Epoche gewählt.

Die Voruntersuchung ergab, dass bei beiden Zielsetzungen (**Ziel 1** und **Ziel 2**) eine Anzahl von höchstens 15 Epochen ausreichend ist. Für das tatsächliche Training wird hier ebenfalls eine Epochenanzahl von 15 gewählt. In Abbildung 4.5 sind die Verläufe der Accuracy auf der Validierungsmenge mit steigender Epochenanzahl zu sehen. Anhand dem Verlauf der Accuracy auf der Validierungsmenge wird das Modell der entsprechenden Epoche mit der höchsten Accuracy ausgewählt. Gleichzeitig sollte die Verlustfunktion (im Bild als Loss bezeichnet) möglichst gering sein, um einer Überanpassung vorzubeugen.

Batchgröße
Der Speicher der Grafikkarten ist nur begrenzt, deshalb kann nicht die gesamte Trai-

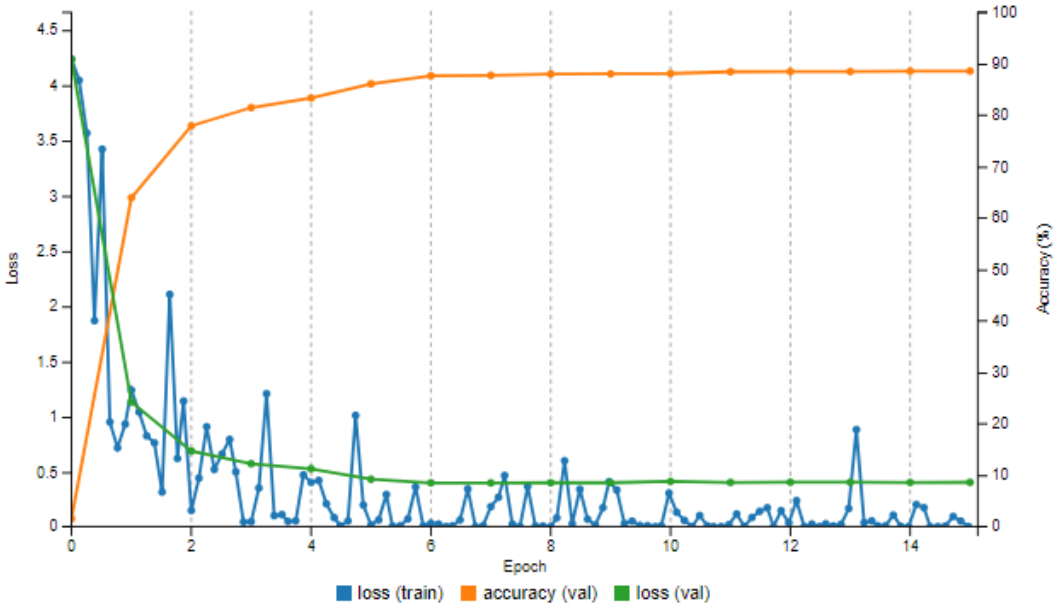


Abbildung 4.5: *Verlauf der Accuracy auf der Validierungsmenge und der Verlustfunktion auf Validierungsmenge und Trainingsmenge für* ***Ziel 1***

ningsmenge auf einmal durch das Netz propagiert werden. Aus diesem Grund ist eine Unterteilung der Trainingsmenge in mehrere Teile nötig. Die durch die Unterteilung entstehenden Teile werden auch *Batches* genannt. Die Batchgröße gibt hierbei an, wie viele Bilder in einem Batch enthalten sind. Daraus ergibt sich auch der Begriff der Iteration: Diese gibt an, wie viele Batches benötigt werden, um die gesamte Trainingsmenge einmal durch das Netz zu propagieren (eine Epoche). Größere Batches führen zu einer schnelleren Trainingszeit. Die Batchgröße ist jedoch durch den Grafikkartenspeicher begrenzt.

Nach Durchlauf eines Batches findet eine Aktualisierung der Parameter statt. Ist die Batchgröße zu klein liegen zu wenige Informationen vor, um die Parameter anzupassen. Bei einer zu großen Batchgröße und zu wenigen Gesamtdaten, reichen die wenigen Aktualisierungen der Parameter möglicherweise nicht aus. Die Batchgröße und die Lernrate stehen bei Optimierungsmethoden wie *Stochastic Gradient Descent* in direktem Zusammenhang [39]. Aus diesem Grund wird eine Batchgröße von 16 für VGG16 und 128 für AlexNet gewählt. Dabei handelt es sich um die von Digits selbst gesetzten Größen. Mit Hilfe von Experiment 3 (siehe 4.4.3) kann die Lernrate für diese Batchgröße angepasst werden.

Optimierungsmethode
Ein wichtiger Parameter für den Trainingsprozess ist die Optimierungsmethode. Diese führt sozusagen die vorher genannten Parameter des Trainings zusammen. Mögliche Methoden sind:

- Stochastic Gradient Descent
- AdaDelta

- Adam
- Nesterov's Accelerated Gradient
- RMSprop

Beim Stochastic Gradient Descent (kurz SGD) handelt es sich um eine stochastische Approximation der Gradientenabstiegsmethode (siehe 2.2.4). Anstatt des genauen Gradienten wird dieser in jeder Iteration geschätzt. SGD bestimmt die Änderung der Gewichte W durch eine Linearkombination des negativen Gradientens $\nabla L(W)$ und der vorangegangenen Gewichtsänderung V_t. Die Lernrate α ist hier das Gewicht des negativen Gradientens und das Momentum μ das Gewicht der vorangegangenen Änderung.

Für Iteration $t+1$ ergibt sich dann folgende Gewichtsänderung [4]:

$$\begin{aligned} V_{t+1} &= \mu V_t - \alpha \nabla L(W_t) \\ W_{t+1} &= W_t + V_{t+1} \end{aligned} \tag{4.1}$$

SGD ist für diese Arbeit die erste Wahl, da diese Methode ebenfalls in der Original AlexNet Implementierung eingesetzt wurde. Es werden jedoch Experimente zu den anderen Methoden durchgeführt. Näheres dazu ist in Experimente 3 (siehe 4.4.3) nachzulesen.

Basis Lernrate
Bei der Lernrate handelt es sich, bei Gradientenverfahren, um die Schrittgröße der Änderung der Gewichte. Eine zu kleine Lernrate führt zu einem sehr langwierigen Trainingsprozess und die Gefahr steigt in einem lokalen Minimum zu bleiben. Eine zu große Lernrate könnte jedoch das Minimum gar nicht finden oder durch ständiges Hin- und Herspringen zu einem nicht konvergenten Verhalten der Anpassung führen. Abbildung 4.6 zeigt die Auswirkungen einer zu großen oder zu kleinen Lernrate und ein weiteres Beispiel, in dem die Lernrate passend gewählt wurde.

Anpassung der Lernrate über die Zeit
Zu Beginn kann es sinnvoll sein, die Lernrate größer zu wählen und diese während des Trainings zu verringern. Die Anpassung geschieht entweder dynamisch, zum Beispiel je nachdem ob sich die Accuracy auf der Validierungsmenge weiter verbessert oder stagniert. Oder statisch wobei die Anpassung nach einer festen Epochenanzahl geschieht oder durch eine Funktion beschrieben (z.B. Sigmoid) wird. Für die Lernrate und ihre Anpassung werden in Experiment 3 in Abschnitt 4.4.3 verschiedene Varianten untersucht.

4.4 Experimente und Validierung

Für das Auffinden und die Validierung der besten Netzarchitektur und deren Hyperparameter werden drei Experimente durchgeführt. Jedes Experiment besteht aus mehreren Untersuchungen. Für die Untersuchungen werden unter anderem Accuracy, Recall und Precision berechnet, Confusion Matrizen erstellt und Grad-CAM

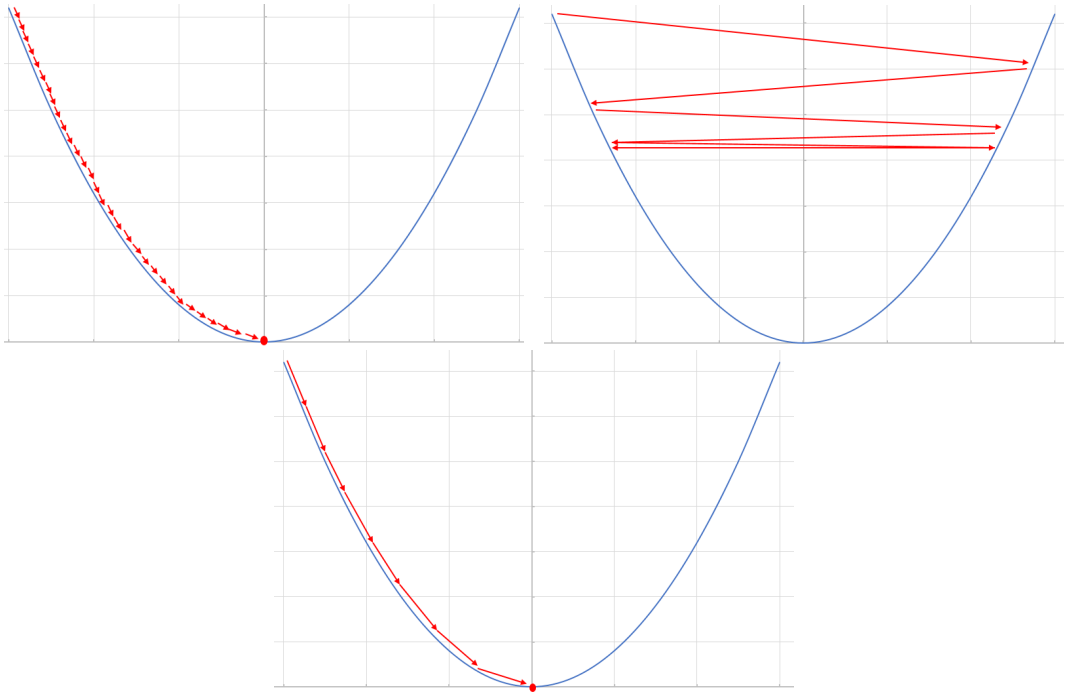


Abbildung 4.6: *Auswirkungen einer zu kleinen Lernrate (oben links), einer zu großen Lernrate (oben rechts) und einer passenden Lernrate*

zur Visualisierung verwendet. Alle Untersuchungen zu den Hyperparametern werden entweder auf den Trainingsdaten oder auf den Validierungsdaten durchgeführt, damit es zu keiner Anpassung auf die Testdaten kommt.

Folgende Experimente wurden durchgeführt:

- **Experiment 1:** Maximale Bildanzahl pro Klasse - Abschnitt 4.4.1
- **Experiment 2:** AlexNet vs. VGG16 und ImageNet Vortraining vs. eigenes Vortraining - Abschnitt 4.4.2
- **Experiment 3:** Hyperparameter - Abschnitt 4.4.3

Experiment 1 wird ausschließlich für **Ziel 1** durchgeführt, da es hier ein großes Ungleichgewichts der Klassen gibt. In **Experiment 2** werden die Ergebnisse von Experiment 1 genutzt und zusätzlich weitere Untersuchungen für **Ziel 1** und **Ziel 2** durchgeführt. **Experiment 3** untersucht die Hyperparameter für das Training für **Ziel 1**. Nachdem diese dafür festgelegt wurden, werden die Hyperparameter auch für **Ziel 2** validiert und, wenn nötig, angepasst.

Üblicherweise werden die Experimente iterativ durchgeführt. Zum Beispiel kann ein Anpassen der Hyperparameter dazu führen, dass eine andere Architektur besser abschneidet, als zum Beispiel in Experiment 2 gewählt. Diese Arbeit dient jedoch der Untersuchung der Machbarkeit und aufgrund der zeitlichen Begrenzung werden die Experimente lediglich einmal sequenziell durchgeführt.

4.4.1 Experiment 1: Maximale Bildanzahl pro Klasse

Ziel

Die Auswirkungen des Klassenungleichgewichts (siehe 4.2.2) in **Ziel 1** bei beiden ausgewählten Architekturen, AlexNet und VGG16, zu untersuchen.

Verfahren

Für das Experiment werden AlexNet und VGG16 jeweils viermal mit unterschiedlicher Begrenzung der gesamten Daten trainiert. Bei den Begrenzungen handelt es sich um 200, 300 und 1000 maximale Bilder pro Klasse und einem Datenset ohne Begrenzung (siehe Tabelle 4.2).

Mit Hilfe des Evaluationsskriptes (für Details zur Funktionsweise und Implementierung dieses, siehe Kapitel 5.5) werden nach dem Training die Performance der unterschiedlichen Netze auf der Testmenge evaluiert. Des Weiteren werden Confusion Matrizen für die verschiedenen Modelle erstellt und anschließend die Heatmaps des Modells mit einer Begrenzung von 200 und dem Modell ohne Begrenzung untersucht.

Auswertung

Das Datenset ohne Begrenzung und der VGG16 Architektur erreicht die höchste Accuracy von $91,51\%$. Die Begrenzung von 1000 Bildern pro Klasse führt allerdings nur zu einem um $0,25\%$ schlechteren Ergebnis (siehe Tabelle 4.5) .

Begrenzung	**Architektur**	**Accuracy**	**Top1-Error**	**∅ Precision**	**∅ Recall**
keine	VGG16	$90,51\%$	$9,49\%$	$0,91$	$0,91$
keine	AlexNet	$68,3\%$	$31,70\%$	$0,71$	$0,68$
max. 1000	VGG16	$90,24\%$	$9,76\%$	$0,90$	$0,90$
max. 1000	AlexNet	$66,74$	$33,26\%$	$0,69$	$0,67$
max. 300	VGG16	$86,96\%$	$13,04\%$	$0,88$	$0,87$
max. 300	AlexNet	$61,09\%$	$38,91\%$	$0,68$	$0,61$
max. 200	VGG16	$84,77\%$	$15,23\%$	$0,86$	$0,85$
max. 200	AlexNet	$58,94\%$	$41,06\%$	$0,64$	$0,59$

Tabelle 4.5: Ergebnisse des ersten Experiments für **Ziel 1** mit unterschiedlicher Begrenzung der Bildanzahl pro Klasse (für die Definitionen von Accuracy, Precision und Recall und Top-1 Error siehe Kapitel 2.4.1)

Dadurch, dass die Testdaten zuvor randomisiert aus den Gesamtdaten ausgewählt wurden, kommt es auch bei diesen zu einem Klassenungleichgewicht. Dieses führte dazu, dass das Accuracy Paradox (siehe Kapitel 2.4.1) nicht auszuschließen ist. Deshalb ist an dieser Stelle die Accuracy nicht ausreichend, um die Auswirkungen der Begrenzungen zu ermitteln. Trotz hoher Accuracy kann eine Überanpassung nicht ausgeschlossen werden.

Um eine Überanpassung auszuschließen, muss für alle Modelle Confusion Matrizen erstellt werden. Hier äußert sich eine etwaige Überanpassung mit höheren Werten in der der überangepassten Klasse zugeordneten Spalte (siehe 2.4.2).

Abbildung 4.7 zeigt die Confusion Matrix des Modells ohne Begrenzung und des Modells mit einer Begrenzung von maximal 200 Bildern pro Klasse. In beiden kann eine Überanpassung auf eine Klasse ausgeschlossen werden, da keine Spalte mit höheren Werten vorliegt.

Der Vergleich der Matrizen untereinander zeigt außerdem, dass es bei dem Modell, das auf dem Datenset ohne Begrenzung trainiert wurde, seltener zu Verwechslungen verschiedener Klassen kommt. Die Confusion Matrizen der anderen Begrenzungen sind im Anhang A.3 und A.4 zu finden.

Das Modell ohne Begrenzungen schneidet auch bei den Ergebnissen des Evaluationsskriptes in Top1-Error und Precision (bzw. Recall) besser ab. Alle Werte dazu sind in Tabelle 4.5 zu sehen. Im Anhang A.5 sind außerdem die Ergebnisse des F1-Scores für die einzelnen Klassen zu sehen. Diese zeigen ebenfalls ein besseres Abschneiden des Modells ohne Begrenzung. Die Werte des F1-Scores bei dem Modell ohne Begrenzung liegen fast immer über denen des Modells mit einer Begrenzung. Lediglich bei 9 Klassen liegt der F1-Score des Modells mit Begrenzung höher (bei *Aurelian, Claudius, Julia Domna, Maximinus Thrax, Theodosius II, Tiberius, Valens und Valentian I*).

Da diese Klassen große Unterschiede in der Menge der Bilder aufweisen, kann es sich dabei nicht um den ausschlaggebenden Faktor handeln. Gemeinsam haben sie alle, dass sie, ohne Begrenzung, weniger als 200 Bilder besitzen. In den restlichen 60 Klassen gibt es aber mehr Klassen (mit unter 200 Bildern), bei denen das Modell ohne Begrenzung einen höheren F1-Score erreicht. Daher ist die Menge der Bilder in diesen Klassen nicht der ausschlaggebende Faktor, der zu einem besseren Abschneiden bei dem Modell mit Begrenzung führt.

Der Unterschied bei diesen Klassen ist auch in der Confusion Matrix zu sehen. In der Abbildung ist die Klasse *Valens* mit einem roten Kreis gekennzeichnet.

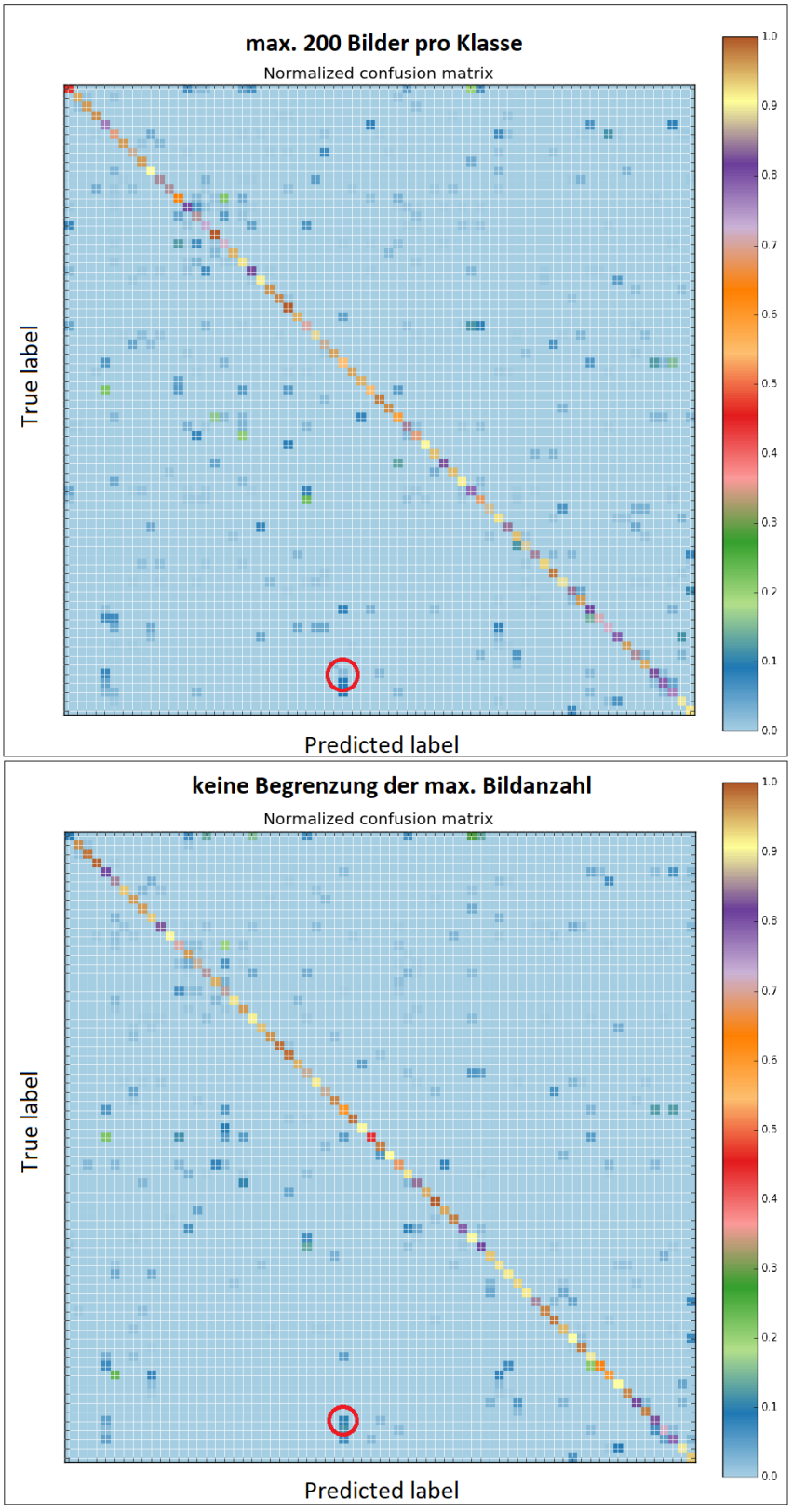


Abbildung 4.7: *VGG16: Confusion Matrix (normalisiert) für das VGG16 Modell mit dem Datenset mit Begrenzung von max. 200 Bildern pro Klasse (oben) und ohne Begrenzung (unten). Das Modell ohne Begrenzung weist niedrigere Werte abseits der Diagonalen auf. Eine Ausnahme ist die Klasse Valens (roter Kreis).*

Betrachtet man die Bilder, bei denen das Modell mit einer Begrenzung von 200 Bildern besser abschneidet, fällt auf, dass es viele Bilder sind, bei denen durch Abnutzung und Oxidation sehr wenig zu erkennen ist. Bei den restlichen Bildern, bei denen das Porträt gut erhalten ist, gibt das Modell ohne Begrenzung sehr ähnliche Klassen aus. Dies ist zum Beispiel bei Valentinian I und Valens der Fall.

Valentinian I *Valens*

Abbildung 4.8: *Vergleich von zwei sehr ähnlichen Münzen. Das Modell ohne Begrenzung gab bei dem Porträt von Valentinian I Valens aus, das Modell mit einer Begrenzung von max. 200 Bildern die korrekte Klasse.*

Die Münze in Abbildung 4.8 zeigt das Porträt von Valentinian I, bei der das Modell ohne Begrenzung Valens ausgab und zum Vergleich eine Münze mit dem Porträt von Valens. In der Confusion Matrix (siehe Abbildung 4.9) ist das bessere Abschneiden des Modells mit Begrenzung bei Valens und Valentinian I ebenfalls zu sehen.

max. 200 *keine Begrenzung*

Abbildung 4.9: *Vergleich der Confusion Matrizen bei den drei sehr ähnlichen Münzporträts von Valens, Valentinian I und Valentinian II.*

Betrachtet man zusätzlich die Heatmaps beider Modelle (siehe Abbildung 4.10), fällt auf, dass das Modell ohne Begrenzung zwar prägnante Bereiche betrachtet (Büste und Kopfschmuck), diese jedoch bei dem Porträt von Valentinian I und Valens sehr ähnlich sind. Das Modell ohne Begrenzung betrachtet zusätzlich noch den Haaransatz und die Legende. Interessant ist auch der geschichtliche Hintergrund. Bei Valens handelt es sich um den jüngeren Bruder Valentinians I. Beide Modelle verwechseln zusätzlich in einigen Fällen Valens und Gratian. Bei Gratian handelt es sich um einen Sohn von Valentinian I.

max. 200 *keine Begrenzung*

Abbildung 4.10: *Heatmap Vergleich der Modelle mit Begrenzung (max. 200) und ohne Begrenzung bei Valentinian I. Das Modell ohne Begrenzung gibt die ähnliche Klasse Valens aus.*

Schlussfolgerung

Es ist keine Begrenzung der maximalen Bildanzahl im Porträt Datenset für **Ziel 1** nötig.

Zwar schneidet bei einigen Bildern das Modell mit Begrenzung besser ab, jedoch handelt es sich hier lediglich um 11 Bilder in der Testmenge (von insg. 7047 Bildern). Bei der Mehrheit der Bilder liegt das Modell ohne Begrenzung richtig. Diese Ergebnisse zeigen, dass eine Begrenzung der maximalen Bildanzahl pro Klasse bei den gewählten Architekturen und Daten nicht nötig ist. Im Gegenteil, eine zu starke Begrenzung führt sogar zu einer schlechteren Performance dieser.

Im Hinblick auf die Option der Erweiterbarkeit der Daten, welches zu einem noch größeren Ungleichgewicht führen könnte, sollte jedoch das Modell mit einer Begrenzung genutzt werden. Bei der Weiterverwendung der Modelle dieser Arbeit, sollte das Modell ohne Begrenzung verwendet werden.

Weitere Ergebnisse

Da die Ähnlichkeit verschiedener Münzen in der Confusion Matrix zu sehen ist, ergibt sich die Möglichkeit, ähnliche Porträts herauszufiltern. Damit könnte man weitere Forschungsfragen beantworten, z.B. die Frage nach Ähnlichkeiten bei Münzen von Prägeherren, die keine familiären Verzweigungen besitzen. Hier wäre zu untersuchen, ob und welche anderen Gründe es gab, sich auf den Münzen ähnlich darzustellen.

Die Evaluation zeigt auch, dass es zwei Klassen gibt, die lediglich eine numerische ID besitzen (diese befinden sich in der Confusion Matrix in der ersten und zweiten Zeile). Die erste Klasse ist die ID 58396. Dabei handelt es sich auf der Webseite des British Museums um die ID zum römischen Konzept des *Genius*. Die Münzen in dieser Klasse besitzen nicht immer das gleiche Porträt. Unter anderem sind Porträts von Maximian, Maximinus Thrax und Diocletian enthalten. Genius ist auf der Rückseite dieser Münzen abgebildet. Bei der Bestimmung des Porträts liegen keine Informationen über die Rückseite vor, weswegen der F1-Score der Klasse 58396

im Modell ohne Begrenzung bei 0.16 liegt. Betrachtet man die einzelnen Ausgaben des Netzes für diese Klasse, fällt auf, dass häufig eine Verwechslung mit Maximian, Maximinus Thrax und Diocletian vorliegt. Hierbei handelt es sich um die eigentlich korrekten Klassen und das Ergebnis kann genutzt werden, um diesen Fehler in OCRE aufzudecken.

Die zweite Klasse, die lediglich eine ID besitzt, ist die Klasse mit der ID 60208. Hier liegt der F1-Score jedoch bei 0.97. Das bessere Ergebnis dieser Klasse lässt sich damit erklären, dass es sich bei dieser ID um die Personifikation der Stadt Rom als Roma handelt. Die Göttin Roma hat wie die anderen Porträts wiederkehrende Merkmale, wie einen Helm oder den Ansatz eines Umhangs bzw. einer Rüstung an der Büste. Dadurch lässt sich auch das bessere Abschneiden des Netzes bei dieser ID erklären.

4.4.2 Experiment 2: AlexNet vs. VGG16 und ImageNet Vortraining vs. eigenes Vortraining

Ziel

Die Auswirkungen der Unterschiede der beiden gewählten Architekturen zu untersuchen und eine Entscheidung für die Implementierung zu treffen.

Verfahren

In diesem Experiment werden AlexNet und VGG16 hinsichtlich ihrer Unterschiede in der Performance untersucht. Neben der Ermittlung der Performance auf dem Porträt Datenset (**Ziel 1**) wird ebenfalls die Performance auf dem OCRE Datenset (**Ziel 2**) mit dem Evaluationsskript evaluiert. Weiterhin werden Confusion Matrizen und verschiedene Heatmaps für **Ziel 2** erstellt, um die Unterschiede beider Netze herauszufiltern.

Zusätzlich werden anschließend noch Untersuchungen dazu durchgeführt, ob die Performance von VGG16 für **Ziel 2**, durch Verwendung der vortrainierten Gewichte aus der Klassifizierung der Porträts (Ziel 1), noch weiter verbessert werden kann.

Auswertung

Aus Tabelle 4.5 ist bereits ersichtlich, dass für das Porträt Datenset VGG16 deutlich besser abschneidet, als AlexNet. Bei der Evaluation der Modelle der OCRE Klassen werden mit der VGG16 Architektur ebenfalls bessere Werte erreicht. Hier ist der Abstand beider Architekturen jedoch deutlich größer. Zu sehen sind die Ergebnisse in Tabelle 4.6. Dieses Ergebnis zeigt sich auch in den Confusion Matrizen (siehe Anhang A.6).

Durch die Verwendung der Gewichte aus dem vorangegangen Training zur Klassifizierung der Porträts, kann die Accuracy des Modells weiter auf $67,36$ gesteigert werden.

Architektur	**Accuracy**	**Top1-Error**	**∅ Precision**	**∅ Recall**
VGG16 (ImageNet)	$66,24\%$	$33,76\%$	$0,73$	$0,66$
VGG16 (Porträt)	$67,36\%$	$32,64\%$	$0,72$	$0,67$
AlexNet	$18,16\%$	$81,84\%$	$0,18$	$0,18$

Tabelle 4.6: Ergebnisse VGG16 bei **Ziel 2** (OCRE Klassen): Vortrainierte Gewichte auf ImageNet und den Porträts (für die Definitionen von Accuracy, Precision und Recall und Top-1 Error siehe Kapitel 2.4.1)

Bei den Heatmaps zeigt sich, dass AlexNet den Fokus bei vielen Münzen auf die Legende legt und weniger die Ikonographien betrachtet. In Tabelle 4.7 sind die Heatmaps basierend auf der letzten vollständig verbunden Schicht beider Netze zu sehen.

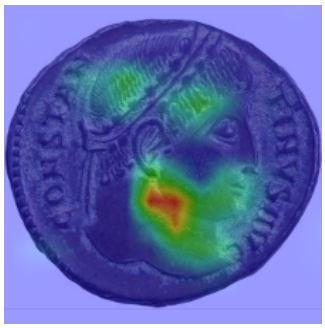


Abbildung 4.11: *Heatmap des VGG16 Modells zur Klassifizierung der Porträts (Constantine I).*

Auch bei den beiden Varianten von VGG16 sind deutliche Unterschiede zu sehen. Vergleicht man zum Beispiel die Heatmaps der Klasse RIC.7.THES.153 aus der Tabelle 4.7 mit der Heatmap von VGG16 zur Klassifizierung der Porträts (Constantine I) in Abbildung 4.11, sieht man die Auswirkungen der vortrainierten Gewichte. Ohne die domänenspezifischen Gewichte aus einem vorangegangenen Training wird die Klassifizierung stärker anhand der Rückseite durchgeführt und auf der Vorderseite ein anderer Bereich betrachtet. Wenn die Gewichte aus dem Porträt Modell verwendet werden, werden die Einzelheiten der Vorderseite ähnlich wie im Porträt Modell betrachtet und fließen verstärkt in die Klassifizierung mit ein.

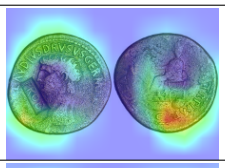
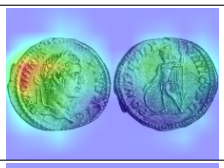
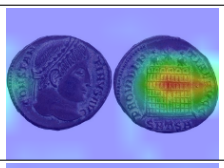
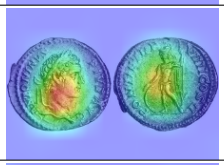
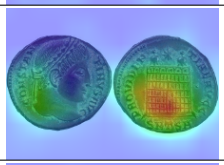
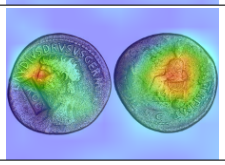
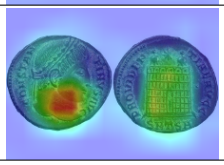
Modell	RIC.1(2).CL.109	RIC.4.CRL.83a	RIC.7.THES.153
• Original Bild			
• AlexNet			
• VGG16 • Vortrainiert: ImageNet			
• VGG16 • Vortrainiert: Porträts			

Tabelle 4.7: Heatmaps basierend auf der letzten vollständig verbundenen Schicht für AlexNet und VGG16 für **Ziel 2**.

Da die Performance beider VGG16 Varianten nahe beieinander liegt, werden zwei Klassen identifiziert, bei denen die Netze den größten Unterschied in Precision und Recall aufweisen. Bei diesen zwei Klassen handelt es sich um RIC.6.SIS.232b und

RIC.9.THES.26b. Der Vergleich (siehe Tabelle 4.8) zeigt, dass bei diesen Münzen die Entscheidungen aufgrund verschiedener Bereiche gefällt wird. Bei der Münze der Klasse RIC.9.THES.26b liegt die ImageNet Variante richtig und gewichtet, im Vergleich zur Porträt Variante, die Rückseite schwerer. Bei der Münze der Klasse RIC.6.SIS.232b ist dies umgekehrt und die Rückseite ist für die Klassifizierung nicht so wichtig, wie die Vorderseite.

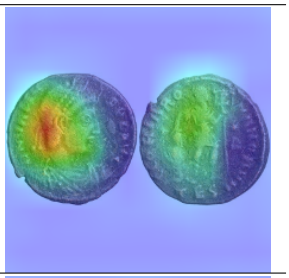
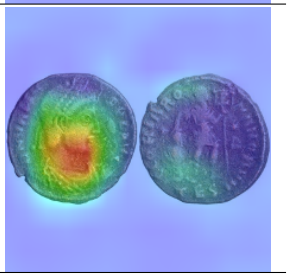
Modell	Korrekte Klasse	RIC.6.SIS.232b	RIC.9.THES.26b
• Original Bild			
• VGG16 • ImageNet	RIC.9.THES.26b		
• VGG16 • Porträts	RIC.6.SIS.232b		

Tabelle 4.8: Heatmaps basierend auf der letzten vollständig verbundenen Schicht für AlexNet und VGG16.

Schlussfolgerung

Für die Nutzung als Hilfsmittel der Numismatik eignet sich nach diesen Experimenten eine tieferes neuronales Netz, wie VGG16.

Bei den Porträts schneidet AlexNet zwar schlechter ab, jedoch kann auch AlexNet eine Accuracy von fast 70% erreichen. Im direkten Vergleich mit der Klassifizierung der OCRE Klassen aus Ziel 2 handelt es sich bei den Porträts aus Ziel 1 also um eine wesentlich einfachere Aufgabe für AlexNet. Die Tiefe der Architektur und die Kombination mehrerer kleinerer Filter in VGG16, führt zu einem deutlich besseren Abschneiden. Hier kann eine Top1-Error Rate von $33,76\%$ erreicht werden im

Vergleich zur Top1-Error Rate von 81, 84% von AlexNet (siehe Tabelle 4.6).

Durch die Verwendung eigens vortrainierter Gewichte kann auch die Performance auf den OCRE Klassen weiter gesteigert werden. VGG16 mit selbst vortrainierten Gewichten, auf dem Porträt Datensatz, erreicht hier eine durchschnittliche Accuracy von 67, 36%. Die Ergebnisse der beiden Varianten von VGG16 liegen allerdings so nahe beieinander, dass hier keine Verallgemeinerung gemacht werden kann. Dazu bedarf es weiterer Experimente, die auch die Auswirkungen anderer vortrainierter Gewichte (z.B. auf der Legende oder der Rückseite) untersuchen.

4.4.3 Experiment 3: Hyperparameter

Ziel

Die Hyperparamter des Trainings für die zuvor gewählte VGG16 Architektur festlegen.

Verfahren

In diesem Experiment wird das Training der VGG16 Architektur, auf dem Datenset mit einer Begrenzung von 1000 Bildern pro Klasse für **Ziel 1**, mit verschiedenen Optimierungsmethoden durchgeführt. Anschließend wird der Verlauf der Accuracy über die Epochen der verschiedenen Varianten verglichen. Für die beste Optimierungsmethode werden zusätzlich weitere Variationen der Hyperparameter untersucht. Als letzten Schritt werden die besten Hyperparameter für die Porträts auf das Training der OCRE Klassen (**Ziel 2**) übertragen, um zu prüfen, ob die Parameter auch hier gute Ergebnisse erzielen können oder angepasst werden müssen.

Auswertung

Nvidia Digits bietet verschiedenen Optimierungsmethoden an:

- Stochastic Gradient Descent (SGD)
- AdaGrad
- Adam
- Nesterov's Accelerated Gradient (NAG)
- RMSprop

In Caffe und Digits gibt es keine Möglichkeit die original Optimierungsmethode von VGG16 zu verwenden. Daher wird zuerst die Methode von AlexNet eingesetzt. Bei AlexNet wird SGD mit einer initialen Lernrate von 0.01 angewendet. Wie in Abbildung 4.12 in türkis zu sehen ist, funktioniert diese Variante jedoch am schlechtesten von allen. Nach dem Tutorial von Caffe [4] muss die Lernrate um einen konstanten Faktor reduziert werden. Die besten Ergebnisse werden mit einer Lernrate von 0.001 erzielt.

Bei den anderen Optimierungsmethoden wird direkt mit einer Lernrate von 0.001 begonnen und AdaGrad und NAG können vergleichbare Ergebnisse wie SGD erzielen. Adam schneidet hier am schlechtesten ab. Der Verlauf von RMSprop lässt vermuten, dass für diese Methode die Lernrate zu Beginn noch weiter reduziert werden muss, da zwischen Epoche 6 und 8 ein Anstieg der Accuracy zu sehen ist, nachdem in Epoche 5 die Lernrate auf 0.0001 reduziert wurde.

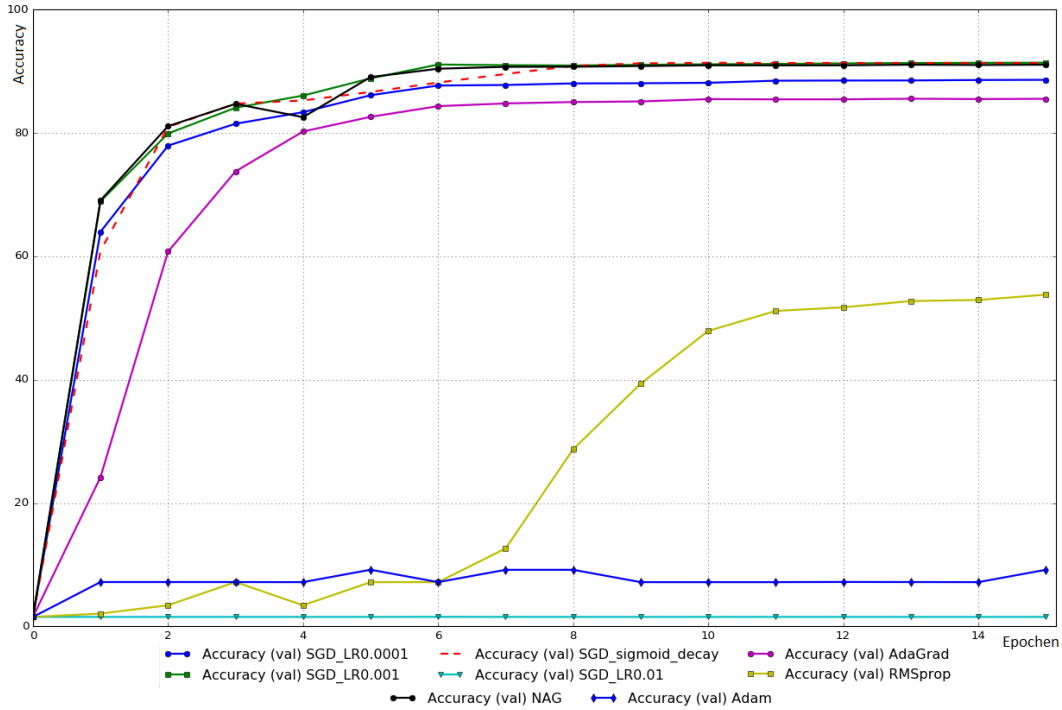


Abbildung 4.12: *Vergleich der Accuracy auf der Validierungsmenge von **Ziel 1** bei Verwendung verschiedener Optimierungsmethoden im Training*

Da SGD und NAG die besten Ergebnisse erzielen und eine ähnliche Funktionsweise haben, werden weitere Experimente mit SGD durchgeführt. Eine weitere Verringerung der Lernrate führt hier jedoch zu einem schlechteren Ergebnis.

Nachdem SGD mit einer Basis Lernrate von 0.001 im ersten Teil dieses Experiments die besten Ergebnisse erzielte, werden weitere Untersuchungen zur Veränderung der Lernrate während des Trainings durchgeführt. Alle in Abbildung 4.12 gezeigten Varianten sind mit stufenweiser Verringerung der Lernrate erreicht worden. Abbildung 4.13 zeigt die Veränderung der Lernrate über die Epochen.

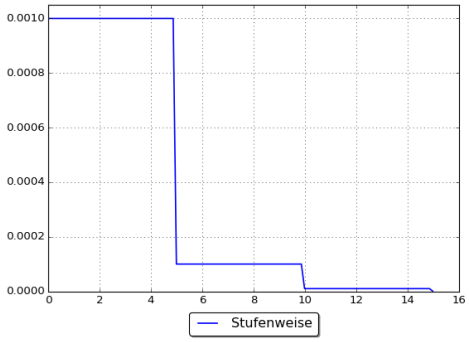


Abbildung 4.13: *Stufenweise Anpassung der Lernrate während des Trainings*

Eine Ausnahme stellt hier die schwarze Kurve dar. Sie entsteht durch das Trai-

ning von SGD mit sigmoider Verringerung der Lernrate (Lernratenveränderung in Abbildung 4.14). Durch dieses Verfahren kann eines der besten drei Ergebnisse (in Epoche 8 bis 15) erreicht werden. Vergleicht man den Verlauf der Verlustfunktion (auch Kostenfunktion) in Abbildung 4.15, liegt diese bei sigmoider Veränderung über den Ergebnissen der stufenweisen Anpassung von SGD.

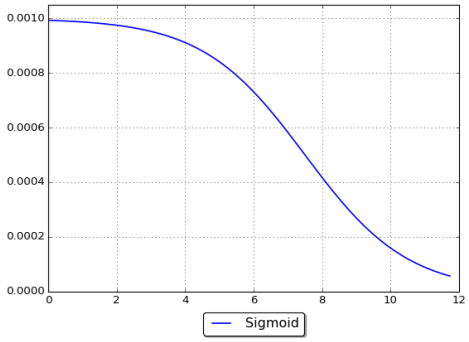


Abbildung 4.14: *Sigmoide Anpassung der Lernrate während des Trainings*

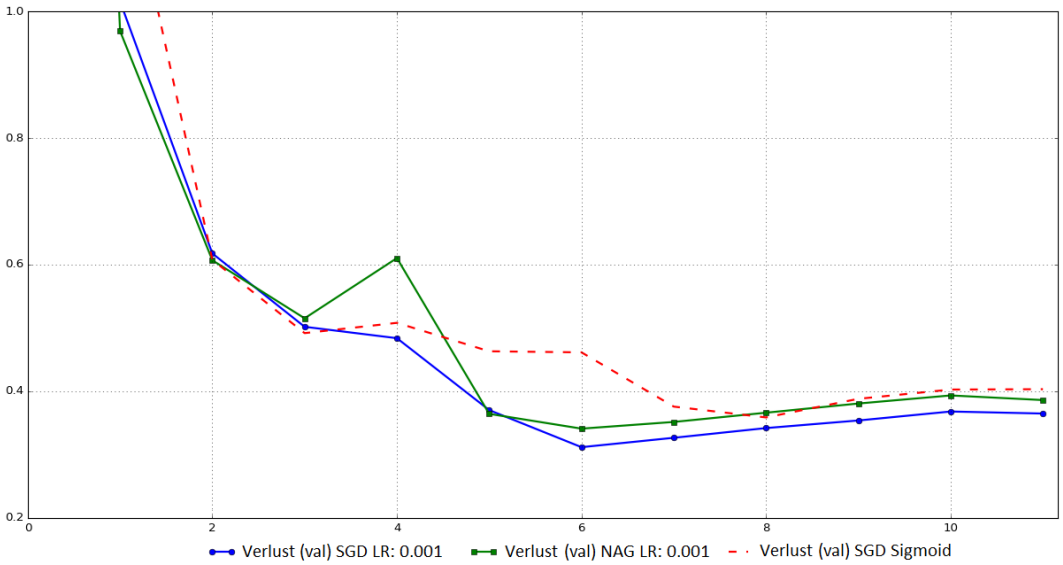


Abbildung 4.15: *Vergleich der Kostenfunktion der besten drei Modelle für* ***Ziel 1***

Bei Abbildung 4.15 handelt es sich um typische Verläufe der Kostenfunktion, wie sie bereits in Kapitel 2.1.3 über Unter- und Überanpassung erklärt wurden. Hier zeigt sich, dass SGD mit stufenweise Veränderung der Lernrate und einer initialen Lernrate von 0.001 in Epoche 6 die besten Ergebnisse erzielen kann.

Nachdem die Hyperparameter für die Porträt Klassen validiert worden sind, muss überprüft werden, ob diese auch bei den OCRE Klassen verwendet werden können. Die Ergebnisse zeigen, dass die Hyperparameter übertragen werden können. Eine Anpassung der benötigten Epochenanzahl muss jedoch vorgenommen werden. Für die SGD Optimierungsmethode mit stufenweiser Anpassung der Lernrate und in-

itialer Lernrate von 0.001 kann die höchste Accuracy bei gleichzeitig niedrigster Verlustfunktion in Epoche 12 erzielt werden.

Schlussfolgerung

Für die Verwendung von VGG16 bei den Münzen können durch den Verlauf des Trainings und der Ergebnisse auf der Validierungsmenge die Hyperparameter des Trainings festgelegt werden. Als Optimierungsmethode erzielen NAG und SGD die besten Ergebnisse, während SGD den geringsten Wert der Verlustfunktion besitzt. Dieser Wert wird in Epoche 6 erreicht. Zur Sicherheit wird für das finale Training eine Anzahl von 10 Trainingsepochen gewählt. Für das Training von VGG16 auf den OCRE Klassen wird eine Anzahl von 15 Epochen gewählt. Für die Lernrate wird bei beiden Zielen eine initiale Lernrate von 0.001 mit stufenweiser Verringerung gewählt.

KAPITEL

FÜNF

Implementierung

Basierend auf den Ergebnissen des vorhergehenden Kapitels wird in diesem Kapitel die Implementierung der entwickelten Programme vorgestellt. Speziell wird auf die Erstellung der Datensets mit Hilfe von Python, die Anpassung der Architekturen und Hyperparameter und die Evaluationsprogramme eingegangen.

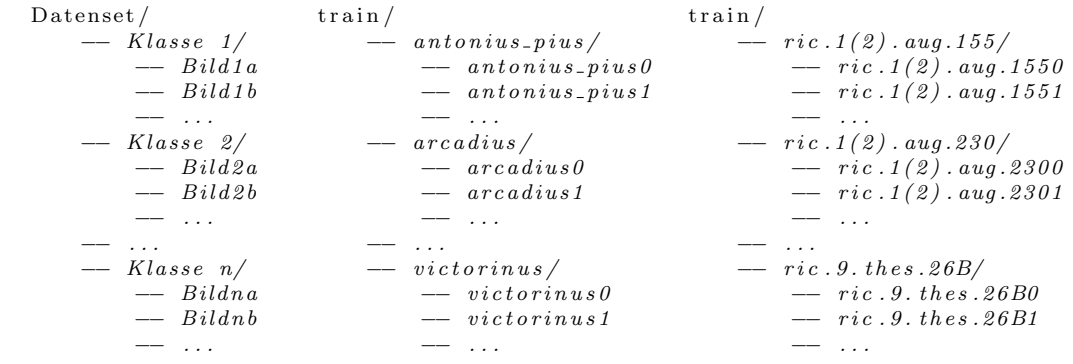
5.1 Erstellung der Datensets

Der erste Schritt der Implementierung ist die Erstellung der benötigen Datensets. Dieser Schritt besteht aus zwei aufeinander folgenden Teilschritten: Extraktion der Datenpaare und Vorverarbeitung dieser.

Für den ersten Teilschritt wird eine Java Applikation entwickelt. Diese wird im folgenden Kapitel genauer beschrieben. Im darauf folgenden Kapitel wird die automatische Vorverarbeitung mit Java und Nvidia Digits erklärt. Anschließend findet die Einstellung der Hyperparameter und daraufhin das Training statt.

5.1.1 Extraktion der Datenpaare

Zur Erstellung der Datensets, müssen zuerst die Datenpaare, bestehend aus Münzbild und Klasse, von der Webseite `http://numismatics.org` heruntergeladen werden. Für diesen Zweck wird die in [34] beschriebene Java Applikation verwendet. Da diese jedoch eine andere Ordnerstruktur erzeugt und auch die SPARQL-Abfrage nicht die benötigten Datenpaare extrahiert, muss die Applikation angepasst werden. Die benötigte Ordnerstruktur sieht wie folgt aus:



Allgemeine Struktur *Trainingsmenge* ***Ziel 1*** *Trainingsmenge* ***Ziel 2***

Abbildung 5.1: *Allgemeine Ordnerstruktur (links) und Beispielstruktur aus der Trainingsmenge von* ***Ziel 1*** *(mitte) und* ***Ziel 2*** *(rechts)*

An dieser Stelle erfolgt noch keine Aufteilung in Trainings-, Validierungs- und Testmenge. Diese wird anschließend, in Abhängigkeit von der Datenmenge, durch ein eigenes Programm durchgeführt. Nach dem Anpassen der Ordnerstruktur muss noch die SPARQL-Abfrage angepasst werden. Dabei ist es wichtig, dass sowohl für Ziel 1 (Porträt Identifizierung) als auch für Ziel 2 (OCRE-Typ Klassifizierung) alle benötigten Informationen (Klassen und Bilder der Vorder- bzw. Rückseite der Münzen) vorliegen.

Für Ziel 1 muss lediglich die URL zur Münzvorderseite (*obvRef*) und für die Klasse der Name der Person des Porträts (*portraitOBV*) zurückgegeben werden.

Für Ziel 2 muss sowohl die URL zum Bild der Münzvorderseite (*obvRef*) als auch die URL zur Münzrückseite (*revRef*) ausgegeben werden und als Klasse die OCRE-Klasse (*type*).

Die entsprechende SPARQL-Abfrage ist hier zu sehen:

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX nm:  <http://nomisma.org/id/>
PREFIX nmo: <http://nomisma.org/ontology#>
SELECT ?type ?coin ?obvRef ?revRef ?portraitOBV WHERE {
    ?type dcterms:source nm:ric.
        ?coin nmo:hasTypeSeriesItem ?type .
            { ?coin nmo:hasObverse ?obverse.
                ?obverse foaf:depiction  ?obvRef }
            { ?coin nmo:hasReverse ?reverse .
                ?reverse foaf:depiction ?revRef }
    ?coin nmo:hasTypeSeriesItem ?type .
        {?type nmo:hasObverse ?obv.
            ?obv nmo:hasPortrait ?portraitOBV}
}
```

Abbildung 5.2: *SPARQL-Abfrage zur Extraktion der Bilder und Klassen für beide Ziele*

Diese Abfrage führt zu dem in Abbildung 5.3 gezeigten JSON-String, der anschlie-

ßend im Programm weiterverarbeitet wird: Die URLs der Bilder wird zum Herunterladen der Bilder genutzt und die jeweiligen Klassen zum Erzeugen der Ordnerstruktur.


```
{"head": {
    "vars": [ "type" , "coin" , "obvRef" , "revRef" , "portraitOBV" ]} ,
  "results": {
    "bindings": [
      {"type": { "type": "uri" , "value":
        "http://numismatics.org/ocre/id/ric.1(2).aug.535" } ,
        "coin": { "type": "uri" , "value":
        "http://ikmk.smb.museum/object?id=18207249" } ,
        "obvRef": { "type": "uri" , "value":
        "http://ikmk.smb.museum/mk-edit/images/n7/7427/vs_opt.jpg" } ,
        "revRef": { "type": "uri" , "value":
        "http://ikmk.smb.museum/mk-edit/images/n7/7427/rs_opt.jpg" } ,
        "portraitOBV": { "type": "uri" , "value":
        "http://nomisma.org/id/augustus" }}
```


Abbildung 5.3: *Ein Teil der Ausgabe der SPARQL-Abfrage aus 5.2*

Im Anschluss erfolgt die Vorverarbeitung der Bilder.

5.1.2 Vorverarbeitung und Datenset Erstellung mit Java und Nvidia Digits

Nach dem Herunterladen der Daten im vorangegangen Schritt, erfolgt die Aufteilung in Trainings-, Validierungs- und Testmenge. Für die Aufteilung wurde eine weitere Java Applikation geschrieben. Der Grund hierfür ist, dass mehrere Datensets für die verschiedenen Experimente benötigt werden. Um auszuschließen, dass die Ergebnisse der Experimente durch verschiedene Testmengen beeinflusst werden und um die Vergleichbarkeit zu gewährleisten, muss sichergestellt werden, dass die Testmengen bei allen Experimenten gleich sind. Da es in Nvidia Digits nicht ohne Weiteres möglich ist, bei der automatischen Aufteilung der Daten, diese Voraussetzung einzuhalten, müssen die Daten bereits vorher aufgeteilt werden.

Eine weitere wichtige Voraussetzung ist die zufällige Auswahl von Bildern. Wählt der Benutzer selbst die Bilder der einzelnen Mengen aus oder wird ein festes Muster zur Auswahl verwendet, kann ein Bias nicht ausgeschlossen werden. Um diesen Bias so gering wie möglich zu halten, wird in der Java Applikation die ThreadLocalRandom .current() Methode verwendet.

Im Anschluss an die Aufteilung kann über die Digits Web-Schnittstelle die Erstellung der Datensets im LMDB Format durchgeführt werden.

5.2 Arbeitsumgebung

Das Training auf einem Computer ohne Grafikkarte oder einer Grafikkarte mit wenig Speicherplatz dauert mehrere Tage. Da jedoch verschiedene Experimente durch-

geführt werden müssen und dazu viele verschiedene Varianten trainiert werden, erfolgt dieses Training auf einem leistungsstärkeren Computer.

Die Konfiguration dieses Computers enthält:

- Arbeitsspeicher: 126GB
- Grafikkarte: 4× Nvidia Titan X Pascal (je 12GB, insg. 48GB)
- Prozessor: Intel Core i7-5930K (3.50GHz)

In Verbindung mit der cuDNN Bibliothek von Nvidia [12] kann die Trainingsdauer um bis zu 98% reduziert werden (∼38 Stunden vs. ∼53 Minuten).

Für das Training wird Nvidia Digits mit dem oben beschriebenen Computer verwendet. Da für das Ausführen der Netze zur Evaluation nur der leistungsschwächere Computer (beschrieben in 4.2.2) zur Verfügung steht, wird hierfür das pure Caffe verwendet. Diese Einschränkung führt zu Laufzeiten zwischen 6 (AlexNet, Ziel 2) und 18 (VGG16, Ziel 1) Stunden. Zur Erstellung der Heatmaps wird Grad-CAM verwendet und zur Erstellung der Confusion Matrizen die in Python verfügbare scikit-learn Bibliothek. Für die Ausführung dieser beiden Funktionen ist der leistungsschwächeren Computer ausreichend.

5.3 Architekturen

In Experiment 2 (siehe 4.4.2) wird VGG16 als Architektur gewählt. Der Aufbau ist im *Google Protocol Buffer* geschrieben. Die Definition der ersten Convolution Schicht und der darauf folgenden ReLU Schicht ist in Abbildung 5.1 gezeigt.

Listing 5.1: *Laden eines Netzes in Caffe (Python Code)*

```
layer {
  name: "conv1_1"
  type: "Convolution"
  bottom: "data"
  top: "conv1_1"
  convolution_param {
    num_output: 64
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu1_1"
  type: "ReLU"
  bottom: "conv1_1"
  top: "conv1_1"
}
```

Mit name wird der Name der aktuellen Schicht festgelegt, anschließend kann mit type die Art der Schicht (Convolution, ReLU, Pooling, Softmax etc.) festgelegt werden. Mit bottom und top wird jeweils festgelegt, welche Schicht der aktuellen vorangegangen ist bzw. welche folgt. Die anderen Schichten wurden in der gleichen Art definiert. Die gesamte Definition der Architektur ist im Anhang A.1 zu finden.

Wichtig ist, dass es einen Unterschied zwischen der im Training verwendeten Definition des Netzes und der später zur Anwendung des trainierten Netzes genutzten Defintion gibt. Für die Anwendung des trainierten Netzes ändert sich die Definition der ersten Schicht, der sogenannten *Data Schicht*. Für das Trai-

```
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 224
  dim: 224
}
```

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mean_file: "/jobs/Portrait_noMax
        /mean.binaryproto"
  }
  data_param {
    source: "/jobs/Portrait_noMax/
        train_db"
    batch_size: 16
    backend: LMDB
  }
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mean_file: "/jobs/Portrait_noMax
        /mean.binaryproto"
  }
  data_param {
    source: "/jobs/Portrait_noMaxa/
        val_db"
    batch_size: 16
    backend: LMDB
  }
}
```

Abbildung 5.4: *Unterschiede des Data Layers für das Training (rechts) und die Anwendung des trainierten Netzes (links).*

ning wird die Batchgröße und die Quelle der Daten festgesetzt. Im Gegensatz hierzu sind für die Anwendung des Netzes diese Werte variabel. Daher wird lediglich das Format der Bilder festgelegt. Der Unterschied der Data Schichten ist in Abbildung 5.4 zu sehen.

Auch die letzten Schichten unterscheiden sich. Hier erfolgt im Training die Berechnung der Accuracy und des Loss zur Anpassung der Gewichte. Für die Ausführung des trainierten Netzes ist dieser Schritt nicht nötig, stattdessen muss mittels Softmax die Klasse mit den höchsten Werten ermittelt werden.

Im Vergleich zur original Definition von VGG16, welche auf dem ImageNet Dataset in 1000 Klassen klassifizieren muss, wird in dieser Variante außerdem die letzte vollständig verbundene Schicht so angepasst, dass sie für Ziel 1 Werte zu 69 Klassen und für Ziel 2 Werte zu 195 Klassen ausgibt.

```
layer {
  name: "softmax"
  type: "Softmax"
  bottom: "fc8_69"
  top: "softmax"
}
```

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc8_69"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8_69"
  bottom: "label"
  top: "loss"
}
```

Abbildung 5.5: *Unterschiede der letzten Schichten für das Training (rechts) und die Anwendung des trainierten Netzes (links).*

Der Unterschied der letzten Schichten ist in Abbildung 5.5 zu sehen.

5.4 Hyperparameter in Digits

Die im vorangegangenen Abschnitt erläuterte Architektur wird über die Weboberfläche von Digits und der dort verfügbaren Caffe Schnittstelle definiert. Außerdem wird für Ziel 1 das mit Hilfe von Experiment 2 (siehe 4.4.1) ausgewählte Datenset ohne Begrenzung ausgewählt. Für Ziel 2 läuft die Einstellungen über die Weboberfläche gleich wie in der folgenden Beschreibung ab, lediglich das Datenset und die später gewählte Trainingsepoche unterscheiden sich.

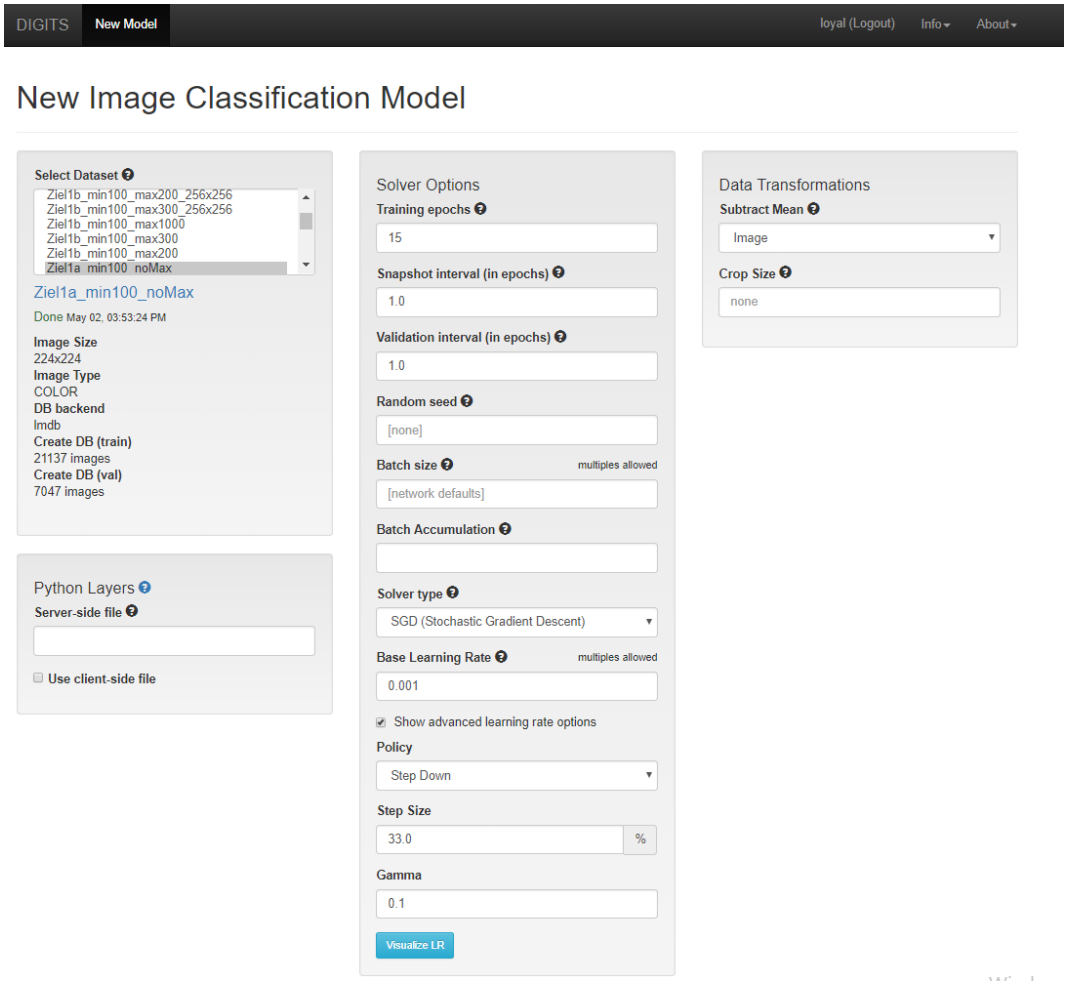


Abbildung 5.6: *Weboberfläche von Nvidia Digits*

Neben der Definition des Aufbaus des Netzes, benötigt es außerdem die Definition der Parameter des Trainings. Die Oberfläche von Digits zum Setzen der benötigen Trainingsparameter ist in Abbildung 5.6 zu sehen.

Digits generierte mit Hilfe der eingetragenen Parameter anschließend den **Solver** (siehe Kapitel 2.5.1). Anhand des 3. Experiments (siehe 4.4.3) werden die Hyperparameter ausgewählt und hier über die Weboberfläche eingetragen. Außerdem wird die Anzahl der zu nutzenden Grafikkarten auf 4 gesetzt (die maximal verfügbare

Anzahl in dieser Arbeitsumgebung) und anschließend das Training gestartet.

5.5 Evaluations Programm

Für die Evaluierung der neuronalen Netze auf der Testmenge müssen die Netze für alle Bilder in der Testmenge eine Klasse berechnen. Es erfolgt dann ein Vergleich mit der richtigen Klasse (der *ground truth*). Die Ergebnisse werden anschließend genutzt, um mit Hilfe von verschiedenen Evaluationsmetriken, die Güte der Netze zu bestimmen. Im ersten Abschnitt dieses Kapitels wird die Implementierung des Python Programms erklärt, das das Ausführen der trainierten Netze ermöglicht. Anschließend werden die Programme, welche die Evaluationsmetriken implementieren, erläutert.

5.5.1 Ausführen der trainierten Netze

Da Caffe über eine Python Schnittstelle verfügt, wird für die Tests im Zuge der Evaluation ein Python Programm geschrieben. Das Programm kann über die Kommandozeile aufgerufen werden.

Die Ausführung der trainierten Netze auf den Bildern der Testmenge geschieht in mehreren Batches, da die Anzahl der Bilder in der Testmenge (7000 Porträtbilder und 1400 OCRE Klassen Bilder) zu groß ist, um auf einmal verarbeitet zu werden. Zur Ausführung des Netzes müssen folgende Dateien vorliegen:

- Definition des Netzaufbaus in *deploy.prototxt*
- Gewichte des Netzes in *.caffemodel* Datei
- Durchschnittliche Pixelwerte in *.binaryproto*
- Textdatei mit allen Klassen in *labels.txt*
- Ordner der Bilder der Testmenge

Die Ordnerstruktur der Testmenge muss wie in Abschnitt 5.1.1 gegeben sein, damit ein Vergleich der Ground Truth mit der Ausgabe der Netze möglich ist.

Zu Beginn werden die Textdatei mit allen Klassen eingelesen und eine Liste mit allen Klassen erstellt. Anschließend wird mit der Datenstruktur eines Dictionarys eine Liste aller Bilder und deren Ground Truth erstellt.

Um Informationen über das Voranschreiten des Programms zur Laufzeit zu erhalten werden außerdem die benötigte Anzahl an Batches, abhängig von der gewählten Batch-Größe und der Anzahl an Testbildern, berechnet. Diese Anzahl an Batches und die Nummer des aktuellen Batches werden während des Ausführen des Netzes zu jedem neuen Batch ausgegeben. Im Falle eines Programmabbruchs kann so die Fehlerquelle leichter ausgemacht werden oder, im Falle von Speicherproblemen, die Testmenge aufgeteilt werden. Speziell beim Ausführen des komplexeren VGG16

Netzes ist diese Funktion wichtig, da die Berechnung eines Batches (250 Bilder) bei den Porträt Klassen bereits 10 Minuten dauert.

Die Hauptfunktion des Programmes ist das Ausführen des Netzes in Caffe. Dazu muss das entsprechende Netz und seine gelernten Gewichte geladen werden:

```
1 # loading neural net with trained weights
2 net = caffe.Net(MODEL_FILE, PRETRAINED, caffe.TEST)
3 in_shape = net.blobs['data'].data.shape
4 # reshaping data layer to process #images in one batch (bs) (with 3
      colors (RGB) and size 244x244)
5 net.blobs['data'].reshape(bs, 3, 224, 224)
6 net.reshape()
7 # Set the shape of the input for the transformer
8 transformer = caffe.io.Transformer({'data': in_shape})
9 # set mean from given mean file (previous converted from the .
      binaryproto)
10 transformer.set_mean('data', np.load('mean.npy').mean(1).mean(1))
11 # order of color channels
12 transformer.set_transpose('data', (2,0,1))
13 # RGB to BGR
14 transformer.set_channel_swap('data', (2,1,0))
```

Listing 5.2: Laden eines Netzes in Caffe (Python Code)

Das eigentliche Laden der Netze geschieht bereits in Zeile 1. Hier wird mit Hilfe des `caffe.Net` Interfaces festgelegt, dass es sich um die Test-Phase handelt, also keine Gewichte verändert werden und auch in der *deploy.prototxt* (MODEL_FILE) die Definition der Test-Phase verwendet wird. Bei PRETRAINED handelt es sich um das *.caffemodel* mit den Informationen zu den trainierten Gewichten.

Anschließend wird mit `net.blobs['data'].data.shape` festgelegt, dass die im *deploy.prototxt* als *data* benannte Schicht als Definition der Eingabe für dieses Netz gilt. Für diese Schicht muss danach noch angepasst werden, wie viele Bilder in einem Batch durch das Netz propagiert werden. Dies geschieht in Zeile 5.

Mit Hilfe des Transformers werden die vorher übergebenen Definitionen für Caffe geladen und das Bild so vorverarbeitet, dass es mit Caffe durch das Netz propagiert werden kann. Hierfür werden die Reihenfolge der Farben im Bild und die durchschnittlichen Pixelwerte wie für das Training gesetzt. Das ist nötig, da die Daten sonst nicht dem Format der im Training gelernten Daten entsprechen, da im Training ebenfalls die Reihenfolge der Farbchannels so verarbeitet und der durchschnittliche Pixelwert zur Zentrierung der Daten verwendet wird.

Nachdem das Netz geladen und alle nötigen Definitionen übergeben sind, wird in einer `for`-Schleife auf jedes Bild das Preprocessing (dt.: Vorverarbeitung) durchgeführt und anschließend in einen Caffe Blob (für mehr Informationen über die Datenstruktur siehe [2]) an die entsprechende Batchposition gespeichert.

```
1 # preprocess image
2 transformed_image = transformer.preprocess('data', img)
3 # put the image into i-th place in batch
```

```
4 net.blobs['data'].data[i,:,:,:] = transformed_image
```

Listing 5.3: Auszug einer For-Schleife über die Bilder in einem Batch: Vorverarbeitung der Eingabebilder und Einfügen in den Data Blob (Python Code)

Nachdem sowohl das Netz als auch alle Testbilder des aktuellen Batches geladen sind, kann ein vorwärtsgerichtetes Propagieren der Bilder durch das Netz durchgeführt werden.

```
1 # Forward pass of the images through the network
2 out = net.forward()
```

Listing 5.4: Die Bilder des aktuellen Batches werden durch das Netz propagiert (vorwärts: es geschieht keine Backpropagation) (Python Code)

Die Ergebnisse liegen in der *softmax* Schicht vor und die Klasse mit der höchsten Wahrscheinlichkeit wird mit Hilfe der `argmax()` Funktion abgerufen und anschließend als Ausgabe in einer Liste (in der Reihenfolge der Bilder im aktuellen Batch) gespeichert. Zusätzlich wird eine Liste mit den Ground Truth Werten in der gleichen Reihenfolge erstellt. Für jeden Batch wird diese Liste erweitert und am Ende mit Hilfe der in der Bibliothek `sklearn` verfügbaren Funktion `sklearn.metrics.confusion_matrix` in eine CSV Datei gespeichert, damit später auf die Ergebnisse zurückgegriffen werden kann.

5.5.2 Implementierung der Evaluationsmetriken

Für die Erstellung der Bilder der Confusion Matrizen wird `sklearn.metrics.confusion_matrix()` verwendet und für die Implementierung das Beispiel von [15] benutzt. Da die puren Werte in einer Confusion Matrix schwer zu erkennen sind, eignete sich die farbige Darstellung. Je nach Wert wird eine andere Farbe für die Zelle verwendet. Die Ausgabe als Bilddatei erfolgt mit Hilfe der `matplotlib.pyplot` Bibliothek. Die Erstellung der Confusion Matrizen erfolgt direkt im Anschluss an das Zwischenspeichern in eine CSV Datei.

Des Weiteren wird ein zweites Skript entwickelt, welches den Import der CSV Datei ermöglicht. Dieses Skript erstellt eine Auswertung pro Klasse für Accuracy, Precision und Recall, indem es die Anzahl der korrekten und falschen Klassifizierungen zählt und mit Hilfe der Funktion `sklearn.metrics.classification_report()` berechnet. Es speichert die Anzahl der falschen und korrekten Klassifizierungen pro Klasse außerdem in einem Dictionary.

Dieses Skript kann verwendet werden, um zusätzliche Evaluationsmetriken zu implementieren. Die Auswertung pro Klasse kann jedoch auch gleich im Evaluationsskript ausgegeben werden. Die Trennung in ein separates Skript erfolgt, um die Erweiterbarkeit zu ermöglichen und ein erneutes zeitaufwendiges Durchlaufen der Testdaten zu vermeiden.

KAPITEL

SECHS

Evaluation und Ergebnisse

In diesem Kapitel werden die Ergebnisse der beiden Klassifizierungsaufgaben vorgestellt. Dazu wird zuerst jeweils auf die Testmenge und die Durchführung eingegangen. Anschließend werden die Ergebnisse beider Aufgaben gezeigt und die Klassifizierung in die OCRE Klassen mit einer vorangegangenen Arbeit verglichen.

6.1 Ziel 1: Porträt Klassen

6.1.1 Testmenge

Die Testmenge besteht aus 7000 zufällig ausgewählten Bildern der gesammelten Daten. Es handelt sich bei den Bildern um 224×224 große Bilddateien, die die Vorderseite römischer Münzen zeigen. Die Bilder der Testmenge werden weder während des Trainings noch während der Validierung verwendet.

Zusätzlich zu den Münzen in der Testmenge sind noch sechs sehr stark abgenutzte oder zerbrochene Münzen durch das Netz zu klassifizieren. Dieser zusätzliche Test dient zur Feststellung, wie das Netz mit solchen Münzen umgeht.

6.1.2 Testdurchführung

Die Testdurchführung erfolgt mit dem in Kapitel 5.5 vorgestellten Evaluationsskript. Jedes Bild in der Testmenge wird durch das vorher trainierte Netz propagiert und es erfolgt die Berechnung von Accuracy, Precision und Recall für jede Klasse.

6.1.3 Testergebnis

Die in Tabelle 6.1 gezeigten Ergebnisse werden mit dem auf ImageNet vortrainierten und auf dem Porträt Datenset trainierten VGG16 ohne Begrenzung erzielt.

	Precision	Recall	F1-Score	Bilder		Precision	Recall	F1-Score	Bilder
58396[1]	0.67	0.09	0.16	45	julia_mamaea	1	0.91	0.95	33
60208[1]	0.97	0.97	0.97	66	julian_the_apostate	0.68	0.68	0.68	37
Constantinopolis	0.96	0.98	0.97	54	licinius	0.9	0.92	0.91	237
antoninus_pius	0.99	0.99	0.99	169	licinius_ii	0.78	0.84	0.81	38
arcadius	0.73	0.81	0.77	47	lucilla	1	0.96	0.98	23
augustus	0.88	0.85	0.86	167	lucius_verus	0.88	1	0.93	35
aurelian	0.96	0.94	0.95	80	magnentius	0.88	0.96	0.92	23
caracalla	0.93	0.96	0.95	138	marcus_aurelius	0.97	0.98	0.97	132
carinus	0.78	0.96	0.86	26	maxentius	0.95	0.79	0.86	48
claudius	0.67	0.94	0.78	33	maximian	0.83	0.91	0.87	200
claudius_ii_gothicus	0.97	0.83	0.89	135	maximinus_daia	0.82	0.82	0.82	119
commodus	0.99	0.9	0.94	103	maximinus_thrax	0.88	0.94	0.91	32
constans	0.74	0.7	0.72	159	nero	0.86	0.93	0.89	67
constantine_i	0.94	0.96	0.95	598	nerva	0.88	0.92	0.9	25
constantine_ii	0.9	0.87	0.88	219	philip_the_arab	0.94	0.93	0.94	73
constantius_chlorus	0.88	0.86	0.87	141	philippus_ii	1	0.92	0.96	25
constantius_gallus	0.88	0.95	0.91	62	postumus	0.97	0.85	0.91	40
constantius_ii	0.88	0.86	0.87	478	probus	0.98	0.97	0.98	114
cornelia_salonina	0.97	0.93	0.95	40	septimius_severus	0.95	0.98	0.96	112
crispus	0.9	0.96	0.93	103	severus_alexander	0.97	0.95	0.96	132
diocletian	0.93	0.92	0.92	224	tetricus_i	0.88	0.91	0.89	178
domitian	0.99	0.94	0.97	109	tetricus_ii	0.93	0.98	0.96	132
elagabalus	0.97	0.97	0.97	63	theodosius_i	0.85	0.9	0.88	59
faustina_i	1	0.99	0.99	85	theodosius_ii	0.9	0.64	0.75	14
faustina_ii	0.94	0.99	0.96	79	tiberius	0.47	0.6	0.53	25
galeria_valeria	0.95	0.95	0.95	21	titus	0.72	0.91	0.8	43
galerius	0.8	0.87	0.84	213	trajan	0.99	0.97	0.98	192
gallienus	0.97	0.92	0.95	198	trajan_decius	0.93	0.82	0.87	34
geta	0.89	0.87	0.88	46	trebonianus_gallus	0.94	0.98	0.96	45
gordian_iii	0.98	0.98	0.98	124	valens	0.79	0.82	0.8	40
gratian	0.5	0.61	0.55	33	valentinian_i	0.86	0.72	0.78	43
hadrian	1	0.99	0.99	217	valentinian_ii	0.77	0.79	0.78	42
helena	0.95	0.9	0.93	21	vespasian	0.96	0.9	0.93	108
honorius	0.89	0.44	0.59	18	victorinus	0.85	0.94	0.89	140
julia_domna	0.92	0.98	0.95	46	**avg/total**	**0.91**	**0.91**	**0.9**	**7000**

Accuracy:	0.91
Top-1 Error:	0.09

Tabelle 6.1: Ergebnisse der Klassifizierung der Porträts.

[1] ID Problem: siehe 4.4.1

Für die sechs sehr stark abgenutzten Münzen liefert das Netz bei vier Münzen die richtige Klassifizierung. Bei den restlichen zwei wird eine Münze in den Top 5 richtig klassifiziert. Die zerbrochene Münze mit dem Porträt von Faustina kann auch bei den Top 5 nicht korrekt klassifiziert werden.

Klasse	Bild	Netzausgabe (Top 5)
• Augustus		• Augustus (48%) • Tiberius (41%) • Nero (8%) • Claudius (< 1%) • Faustina I (< 1%)
• Hadrian		• Hadrian (60%) • Tiberius (16%) • Augustus (10%) • Nero (2%) • Antonius Pius (1%)
• Hadrian		• Hadrian (98%) • Titus (< 1%) • Tiberius (< 1%) • Nero (< 1%) • Trajan (< 1%)
• Trajan		• Trajan (88%) • Nerva (2%) • Tiberius (2%) • Nero (2%) • Hadrian (1%)
• Vespasian		• Constantine II (42%) • Vespasian (18%) • Licinius II (13%) • Valentinian II (5%) • Trajan (5%)
• Faustina I		• Claudius II Gothicus (61%) • Augustus (5%) • 58396 (5%) • Aurelian (3%) • Tiberius (1%)

Tabelle 6.2: Klassifizierung der Porträts für stark abgenutzte oder korrodierte Münzen

Die Ergebnisse zeigen, dass auf gut erhaltenen Münzen zur Klassifizierung der Porträts ein neuronales Netz unterstützend eingesetzt werden kann. Da es sich bei den neuronalen Netzen um ein Hilfsmittel bei der Klassifizierung handeln soll und die Entscheidung zur Klassifizierung somit letztendlich bei einem Numismatiker liegt, ist es vertretbar, dass stark abgenutzte Münzen seltener korrekt klassifiziert werden. Hinzu kommt, dass nicht nur das beste Ergebnis ausgegeben werden kann, sondern auch die besten fünf Ergebnisse. Die Top 5 Accuracy bei diesem Modell liegt bei **92%**, was stark für den Einsatz dieser Technik als Hilfsmittel in der Numismatik spricht.

6.2 Ziel 2: OCRE Klassen

6.2.1 Testmenge

Die Testmenge des zweiten Ziels enthält 1400 zufällig ausgewählte Bilder der gesamten Daten. Bei den Bildern der Testmenge handelt es sich ebenfalls um 224×224 große Bilder die jeweils die Münzvorderseite und Rückseite in einem Bild enthalten. Auch die Bilder dieser Testmenge werden weder während des Trainings noch während der Validierung verwendet.

6.2.2 Testdurchführung

Die Tests für die OCRE Klassen werden so durchgeführt wie die Tests für die Porträts, lediglich eine andere Testmenge wird verwendet. Um einen Vergleich mit der vorangegangen Arbeit [34], die den Einsatz von SIFT und SURF untersucht, zu ermöglichen, wird hier zusätzlich die Top-5 Accuracy ermittelt.

6.2.3 Testergebnis

Die Ergebnisse in Tabelle 6.3 werden mit dem auf ImageNet vortrainierten und anschließend auf dem OCRE Datenset trainierten VGG16 erzielt. Vergleicht man die Top-5 Accuracy mit der vorangegangenen Arbeit, kann die Accuracy auf mehr als das doppelte (von 43% auf 92%) gesteigert werden. Dieses Ergebnis liefert nur einen Richtwert, der tatsächliche Unterschied in der Performance kann nur durch einen direkten Vergleich, mit dem gleichen Testset, ermittelt werden.

	Precision	Recall	F1-Score	Bilder
ric.1(2).aug.155	0.75	0.67	0.71	9
ric.1(2).aug.230	0.93	0.87	0.9	15
ric.1(2).aug.427	0	0	0	6
ric.1(2).aug.435	0.3	0.5	0.37	6
ric.1(2).cl.109	0.67	1	0.8	2
ric.1(2).cl.96	1	0.89	0.94	9
ric.1(2).cl.99	1	1	1	3
ric.1(2).gai.38	0.67	0.5	0.57	4
ric.1(2).gai.58	1	0.86	0.92	7
ric.1(2).tib.30	1	1	1	9
ric.1(2).tib.81	0.83	1	0.91	5
ric.2.tr.128	1	0.67	0.8	3
ric.2_1(2).ves.772	1	1	1	6
ric.3.ant.344_denarius	1	1	1	5
ric.3.ant.351a	1	1	1	4
ric.3.ant.360a	1	0.25	0.4	8
ric.3.ant.362	1	1	1	1
ric.3.m_aur.677	0.4	1	0.57	2
ric.4.crl.130A_denarius	1	0.75	0.86	4
ric.4.crl.369_denarius	1	1	1	2
ric.4.crl.83a	1	1	1	9
ric.4.ge.13a_denarius	1	0.71	0.83	7
ric.4.ge.18	1	0.71	0.83	7
ric.4.mxs.13	1	1	1	2
ric.4.tr_d.59B	1	1	1	6
ric.4.tr_g.84	1	0.75	0.86	4
ric.5.aur.386	0.88	0.93	0.9	15
ric.5.car.325	1	1	1	12
ric.5.car.329	1	1	1	3
ric.5.car.463	1	1	1	9
ric.5.car.466	1	0.8	0.89	5
ric.5.cg.109	1	0.14	0.25	7
ric.5.cg.18	0.33	0.33	0.33	6
ric.5.cg.261	1	0.88	0.93	16
ric.5.cg.266	1	0.28	0.43	18
ric.5.cg.36	0.5	0.2	0.29	5
ric.5.cg.48	0.2	0.25	0.22	4
ric.5.cg.54	0.5	0.4	0.44	5
ric.5.dio.306	1	0.75	0.86	4
ric.5.dio.323	1	1	1	4
ric.5.dio.399	0.6	0.5	0.55	6
ric.5.dio.607	1	0.57	0.73	7
ric.5.dio.622	0.75	1	0.86	3
ric.5.gall(2).157k.1	0.75	0.86	0.8	7
ric.5.gall(2).160k.1	1	0.5	0.67	4
ric.5.gall(2).236kB.2	0.3	0.75	0.43	4
ric.5.gall(2).236kB.3	0	0	0	4
ric.5.gall(2).280k.2	0.67	0.89	0.76	9
ric.5.pro.921	1	1	1	3
ric.5.pro.922	0.92	1	0.96	12
ric.5.pro.925	0.88	1	0.93	7
ric.5.pro.927	0.91	0.91	0.91	11
ric.5.sala(1).29	0.5	0.67	0.57	3
ric.5.tet_i.100	0.8	0.94	0.87	35
ric.5.tet_i.126	1	0.2	0.33	5
ric.5.tet_i.127	0.33	0.33	0.33	3
ric.5.tet_i.136	1	0.7	0.82	10
ric.5.tet_i.141	1	0.86	0.92	14
ric.5.tet_i.148	0.71	1	0.83	12
ric.5.tet_i.224	0.67	1	0.8	2
ric.5.tet_i.248	0.55	0.67	0.6	9
ric.5.tet_i.255	0.33	0.67	0.44	6
ric.5.tet_i.258	0.5	0.73	0.59	11
ric.5.tet_i.259	0.75	0.67	0.71	9
ric.7.her.17	0	0	0	0
ric.5.tet_i.260	0.74	0.93	0.82	15
ric.5.tet_i.270	0.9	1	0.95	37
ric.5.tet_i.272	0.77	0.92	0.84	25
ric.5.tet_i.56	0.9	0.56	0.69	16
ric.5.tet_i.71	0.29	0.4	0.33	5
ric.5.tet_i.80	0.89	0.94	0.91	17
ric.5.tet_i.88	0.75	0.5	0.6	12
ric.5.tet_i.90	0.65	0.75	0.7	20
ric.5.val_ii.3	1	0.75	0.86	4
ric.5.vict.114	1	1	1	20
ric.5.vict.118	1	0.88	0.93	24
ric.5.vict.57	0.86	0.8	0.83	15
ric.5.vict.61	0.85	0.94	0.89	18
ric.5.vict.67	0.69	0.82	0.75	11
ric.5.vict.71	0.86	0.95	0.9	19
ric.5.vict.78	0.93	1	0.96	27
ric.6.alex.100a	0.83	0.38	0.53	13
ric.6.alex.101a	0	0	0	5
ric.6.alex.105a	0.35	0.8	0.48	10
ric.6.alex.107c	0.75	0.75	0.75	4
ric.6.alex.110	1	1	1	4
ric.6.alex.160b	0.75	1	0.86	6
ric.6.alex.32a	0	0	0	4
ric.6.alex.71	1	1	1	3
ric.6.alex.78	0.25	0.67	0.36	3
ric.6.alex.79	1	0.25	0.4	12
ric.6.alex.99a	0.43	0.43	0.43	7
ric.6.anch.52a	0.33	0.4	0.36	5
ric.6.anch.54a	0.6	0.43	0.5	7
ric.6.anch.54b	0.4	1	0.57	2
ric.6.anch.55a	0.67	1	0.8	2
ric.6.anch.59a	0	0	0	4
ric.6.cyz.10a	0.67	0.5	0.57	4
ric.6.cyz.12a	0.33	1	0.5	1
ric.6.cyz.16b	1	0.5	0.67	2
ric.6.her.17a	0.55	0.67	0.6	9
ric.6.her.19a	1	0.17	0.29	6
ric.6.ost.35	1	0.5	0.67	2
ric.6.rom.258	1	1	1	5
ric.6.sis.232a	1	0.43	0.6	7
ric.6.sis.232b	0	0	0	3
ric.6.sis.234a	1	0.75	0.86	4
ric.6.thes.60	0.92	0.58	0.71	19
ric.6.thes.61b	0.75	0.46	0.57	13
ric.7.alex.10	0.86	0.67	0.75	9
ric.7.alex.18	0.6	0.6	0.6	5
ric.7.alex.34	1	0.67	0.8	9
ric.7.alex.58	1	0.25	0.4	12
ric.7.alex.59	0.75	0.82	0.78	11
ric.7.alex.60	0.75	0.6	0.67	5
ric.7.alex.63	1	0.67	0.8	3
ric.7.alex.66	1	1	1	1
ric.7.anch.86	0.25	1	0.4	2
ric.7.anch.87	0.89	1	0.94	8
ric.7.anch.88	1	1	1	6
ric.7.cnp.59	0.5	0.75	0.6	4
ric.7.cnp.73	0.14	0.5	0.22	4
ric.7.cnp.80	0.29	0.25	0.27	8
ric.7.cyz.24	0	0	0	5
ric.7.cyz.34	0.5	0.67	0.57	3
ric.7.cyz.8	0.5	0.57	0.53	7
ric.7.cyz.9	0.6	0.6	0.6	10
ric.7.her.116	0.67	0.4	0.5	5
ric.7.her.60	1	0.25	0.4	4
ric.7.rom.19	0.56	0.42	0.48	12
ric.7.rom.23	0.6	0.6	0.6	5
ric.7.rom.237	0.75	0.6	0.67	5
ric.7.rom.27	0.24	0.83	0.37	6
ric.7.rom.287	0.5	1	0.67	2
ric.7.rom.30	1	0.33	0.5	3
ric.7.rom.33	0	0	0	5
ric.7.rom.40	0	0	0	2
ric.7.rom.57	0.5	0.8	0.62	5
ric.7.sis.15	0.22	0.5	0.31	4
ric.7.sis.163	0.5	0.5	0.5	2
ric.7.sis.17	0.67	0.67	0.67	6
ric.7.sis.181	1	0.5	0.67	2
ric.7.sis.182	0.75	1	0.86	3
ric.7.sis.5	0.36	0.57	0.44	7
ric.7.sis.7	0.33	0.2	0.25	5
ric.7.sis.8	0.58	0.73	0.65	15
ric.7.thes.117	0.67	0.44	0.53	9
ric.7.thes.123	0.43	0.75	0.55	4
ric.7.thes.153	0.55	0.75	0.63	8
ric.7.thes.19	0.5	0.8	0.62	5
ric.7.tic.167	0.55	1	0.71	6
ric.8.alex.12	0.8	0.8	0.8	10
ric.8.alex.14	0.67	0.5	0.57	4
ric.8.alex.15	0	0	0	3
ric.8.alex.19	0.5	0.33	0.4	3
ric.8.alex.21	0	0	0	2
ric.8.alex.27	0.8	1	0.89	4
ric.8.alex.32	0.69	1	0.82	9
ric.8.alex.34	0.4	0.67	0.5	3
ric.8.alex.4	0.5	1	0.67	3
ric.8.alex.6	0.29	0.67	0.4	3
ric.8.alex.72	0.57	0.4	0.47	10
ric.8.alex.75	0.04	0.1	0.06	10
ric.8.alex.78	0.7	0.64	0.67	11
ric.8.alex.79	0.33	0.5	0.4	4
ric.8.alex.80	0.03	1	0.05	1
ric.8.alex.9	1	1	1	4
ric.8.anch.113	0.89	0.67	0.76	12
ric.8.anch.132	0.38	0.27	0.32	11
ric.8.anch.135	0.2	0.08	0.12	12
ric.8.anch.144	0	0	0	5
ric.8.anch.148	0.33	0.12	0.18	8
ric.8.anch.187a	0.33	0.09	0.14	11
ric.8.anch.188	0.29	0.83	0.43	12
ric.8.anch.191	1	0.33	0.5	3
ric.8.anch.52	1	0.75	0.86	4
ric.8.anch.54	0.67	0.33	0.44	6
ric.8.cnp.106	0	0	0	7
ric.8.cnp.116	1	0.33	0.5	3
ric.8.cnp.121	0.25	1	0.4	2
ric.8.cnp.135	0.67	0.5	0.57	4
ric.8.cnp.24	1	0.25	0.4	4
ric.8.cnp.82	0.5	0.17	0.25	6
ric.8.cyz.104	1	0.2	0.33	5
ric.8.cyz.18	0.67	0.57	0.62	7
ric.8.cyz.92	1	0.09	0.17	11
ric.8.cyz.93a	0	0	0	4
ric.8.cyz.96	0	0	0	4
ric.8.her.82	0	0	0	5
ric.8.her.90	0.5	0.25	0.33	4
ric.8.nic.84	1	0.2	0.33	5
ric.8.rom.309	1	0.33	0.5	6
ric.9.sis.14A	1	0.75	0.86	4
ric.9.sis.15A	0.5	1	0.67	2
ric.9.thes.26B	1	0.5	0.67	2
avg/total	**0.72**	**0.66**	**0.66**	**1400**

Top-1 Accuracy:	0.66
Top-1 Error:	0.33
Top-5 Accuracy:	0.92

Tabelle 6.3: Ergebnisse der Klassifizierung der OCRE Klassen.

KAPITEL

SIEBEN

Zusammenfassung und Ausblick

Mit dieser Arbeit sollte untersucht werden, ob die Nutzung von neuronalen Netzen die Arbeit der Numismatiker bei der Klassifizierung von römisch imperialen Fundmünzen unterstützen kann. Hierzu wurden verschiedene Architekturen und Varianten neuronaler Netze getestet. Durch die Arbeit entstanden zwei neuronale Netze die verschiedene Klassifikationsaufgaben unterstützen können.

Das erste neuronale Netz identifiziert das Porträt auf der Münzvorderseite. Die Accuracy dieses Netzes liegt bei 91% (Top1). Die Top 5 Accuracy dieses Modells erreicht sogar 99%. Bei den Experimenten zu diesem Modell ergaben sich weitere Anwendungsgebiete. Nicht nur als Unterstützung bei der Klassifizierung, sondern auch bei der Untersuchung von Beziehungen der abgebildeten Personen könnten neuronale Netze helfen. Außerdem konnte durch die Wiederverwendung der in diesem Modell entstandenen Gewichte im zweiten Modell eine Performancesteigerung erreicht werden.

Das zweite Netz klassifiziert die Münzen in ihre OCRE Klassen und somit auch in die weitläufig genutzten RIC Klassen. Eine Top 1 Accuracy von 66% und eine Top 5 Accuracy von 92% zeigen eine deutliche Verbesserung im Vergleich zu der vorangegangenen Arbeit (SIFT Top 5 Accuracy 43%). Um die Performance beider Methoden direkt vergleichen zu können, müssten diese auf den gleichen Testdaten evaluiert werden. Jedoch lässt der große Unterschied in der Accuracy auf verschiedenen Testsets bereits ein besseres Abschneiden der neuronalen Netze vermuten.

Durch die Angabe der ersten 5 Ergebnisse in einer Hilfsapplikation oder einer Vorsortierung der Münzen könnte die Arbeit der Numismatiker unterstützt und beschleunigt werden. Der Einsatz einer sogenannten Siamese Architektur könnte hier sinnvoll sein. Bei dieser Architektur werden zwei Netze trainiert, ein Netz für die Vorderseite und ein Netz für die Rückseite. Ein sogenannter *Merge Layer* führt beide Ergebnisse zusammen und klassifiziert die Münzen basierend auf den Ergebnissen zu Vorder- und Rückseite.

Ein häufiger Kritikpunkt an neuronalen Netzen ist deren schwer nachvollziehbare Entscheidungsfindung. Die Verwendung von Visualisierungstools wie Grad-CAM machen die Klassifizierungen nachvollziehbarer. In Kombination mit einem für die Inschriften trainierten Netz könnten diese in den Bildern markiert werden und das

Ergebnis wiederrum ein weiteres Netz trainieren, das somit die Buchstaben erkennt. Dadurch ergeben sich weitere Anwendungsgebiete in der Geschichtswissenschaft, zum Beispiel die Inschriftenerkennung in Gemälden oder Statuen.

Kombiniert man die trainierten Netze mit großen Mengen an Fundmünzen, könnten diese mit Hilfe von Confusion Matrizen außerdem in Cluster aufgeteilt werden und so weitere Einblicke in den Geldumlauf oder die Abnutzung bestimmter Münzklassen geben. Durch den Einsatz der Confusion Matrizen konnten mit Hilfe dieser Arbeit auch fehlerhafte Einträge in der OCRE zugrundeliegenden Datenbank aufgedeckt werden. Der Einsatz zur Verifikation von manuellen Münzklassifizierungen vor allem aufgrund der Heatmaps könnte ein weiteres Einsatzgebiet von neuronalen Netzen darstellen.

Denkbar ist auch die Nutzung der in dieser Arbeit entstandenen neuronalen Netze als vortrainierte Gewichte für die Untersuchung anderer Kulturen oder Epochen.

Bei den neuronalen Netzen handelt es sich außerdem um ein derzeit sehr aktives Forschungsgebiet, das in den letzten Jahren viele neue Architekturen hervorbrachte. Es ist davon auszugehen, dass es schon bald neue Architekturen gibt, deren spezifisches Verhalten für die automatische Klassifizierung vielversprechend sein könnte. Auch die Untersuchung des Einsatzes von Residual Networks ist noch durchzuführen.

Aufgrund der in dieser Arbeit erzielten Ergebnisse und dem aufgedeckten Potential im Bereich der Numismatik und Geschichtswissenschaft, lohnt sich eine weitere Erforschung von Methoden des maschinellen Lernens. Speziell bei neuronalen Netzen handelt es sich um ein vielversprechendes Forschungsgebiet für die automatische Klassifizierung von Münzen.

ANHANG

A

Abbildungen, Tabellen und Algorithmen

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Abbildung A.1: *Für die Erzeugung der Abbildung 2.6 verwendete Laplace-Filtermatrix*

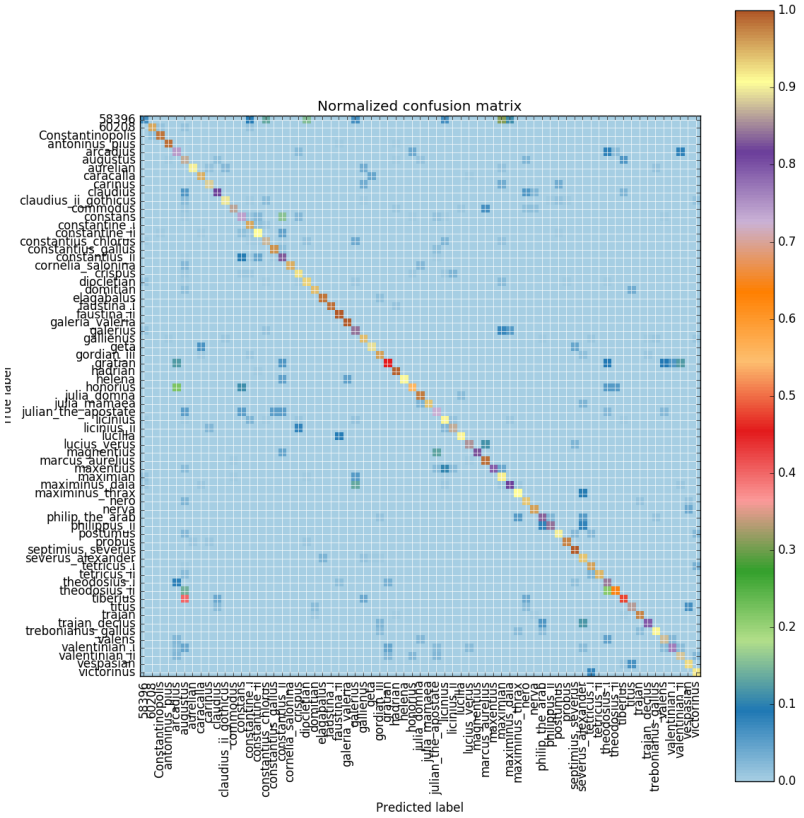


Abbildung A.3: *VGG16: Confusion Matrix (normalisiert) Begrenzung von max. 1000*

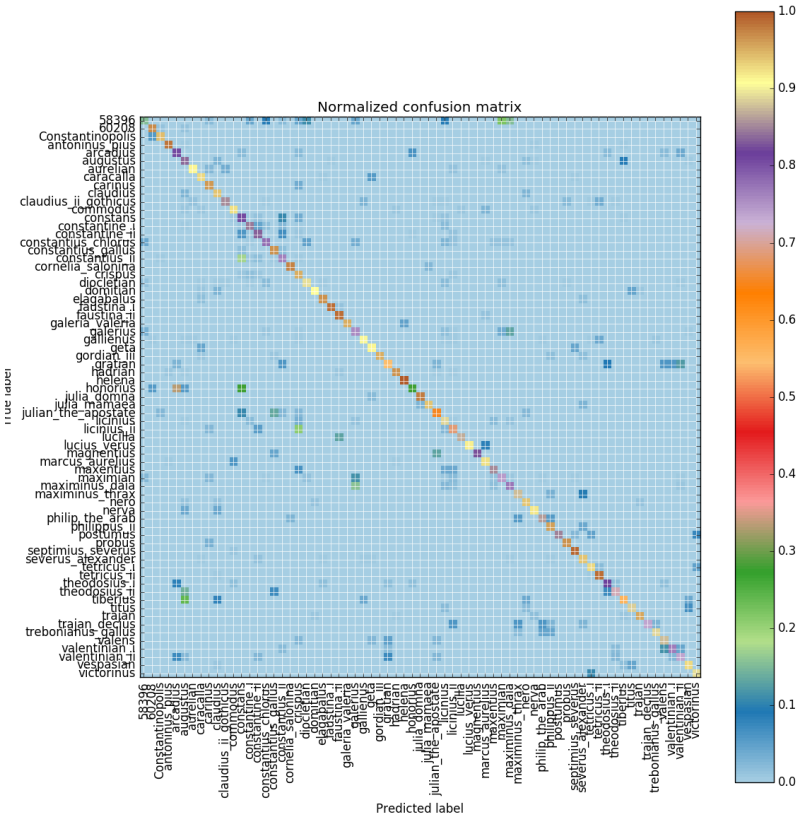


Abbildung A.4: *VGG16: Confusion Matrix (normalisiert) Begrenzung von max. 300*

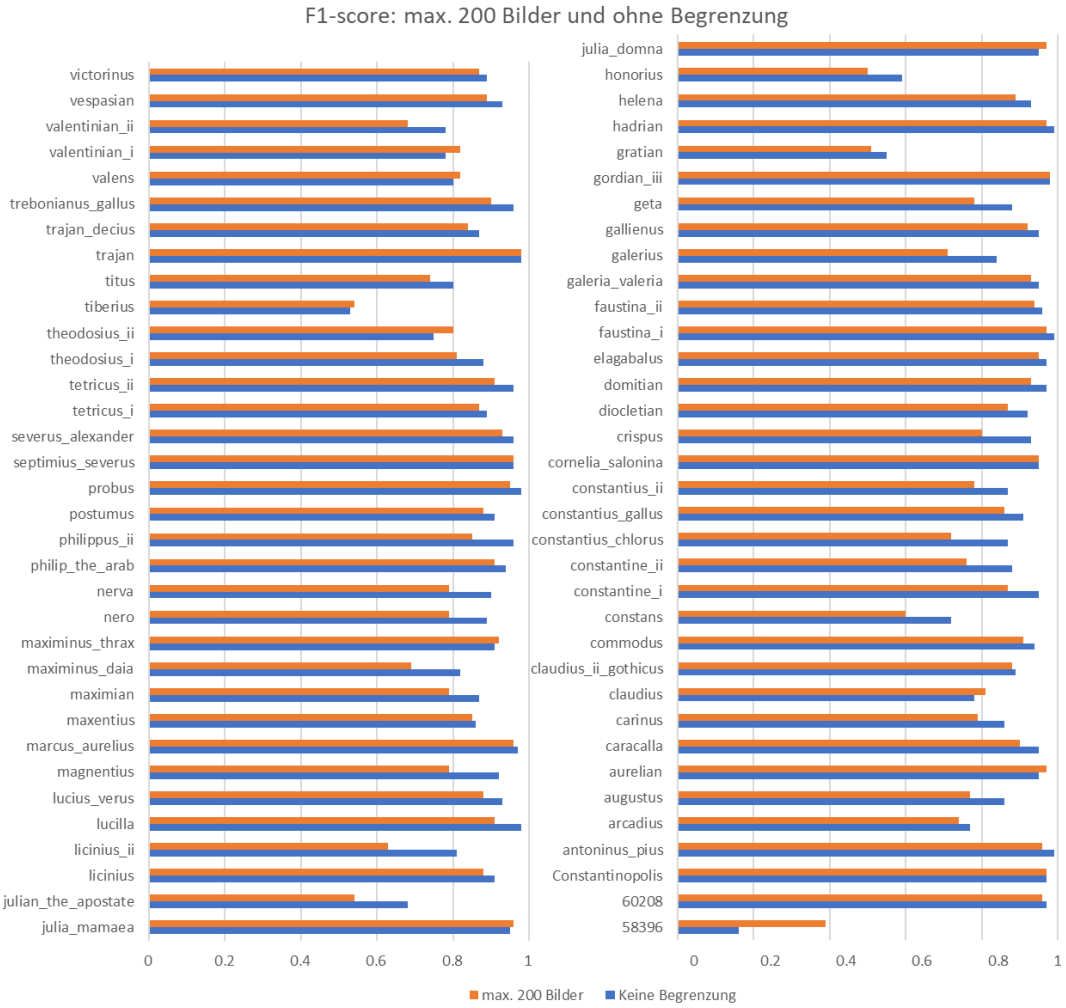


Abbildung A.5: *Vergleich zwischen den F1-Scores der VGG16 Modelle mit Begrenzung der maximalen Bildanzahl und ohne Begrenzung. In fast allen Klassen liegt der F1-Score des Modells ohne Begrenzung über dem mit Begrenzung.*

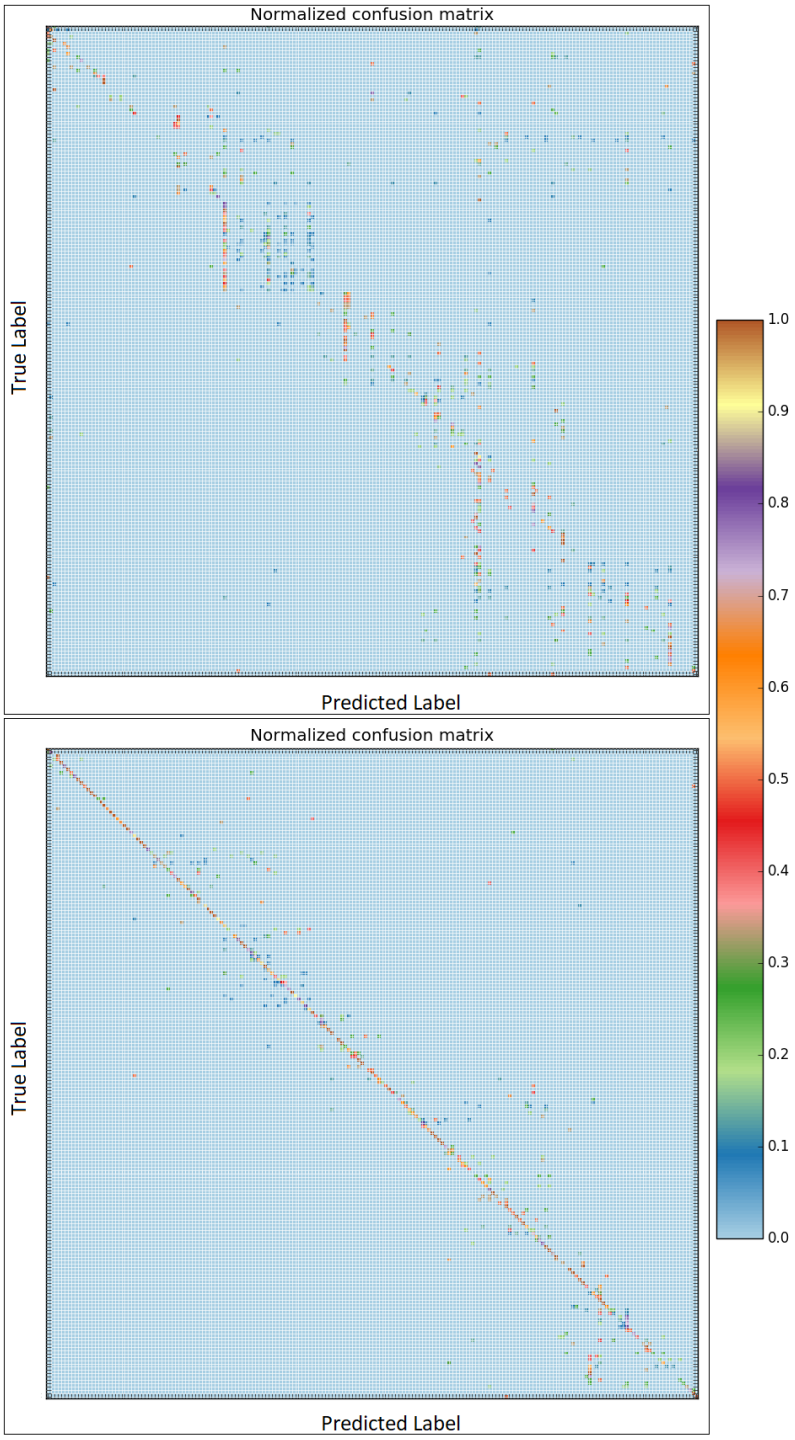


Abbildung A.6: *Confusion Matrizen von AlexNet (oben) und VGG16 (unten) auf dem OCRE Klassen Datenset.*

Listing A.1: Vollständige Definition der deploy.prototxt von VGG16 (Google Protocol Buffer)

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mean_file: "/
        jobs
        /20180502-154521-
        f76a/mean.
        binaryproto"
  }
  data_param {
    source: "/jobs
        /20180502-154521-
        f76a/
        train_db"
    batch_size: 16
    backend: LMDB
  }
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mean_file: "/
        jobs
        /20180502-154521-
        f76a/mean.
        binaryproto"
  }
  data_param {
    source: "/jobs
        /20180502-154521-
        f76a/val_db"
    batch_size: 16
    backend: LMDB
  }
}
layer {
  name: "conv1_1"
  type: "Convolution
      "
  bottom: "data"
  top: "conv1_1"
  convolution_param
      {
    num_output: 64
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu1_1"
  type: "ReLU"
  bottom: "conv1_1"
  top: "conv1_1"
}
layer {
  name: "conv1_2"
  type: "Convolution
      "
  bottom: "conv1_1"
  top: "conv1_2"
  convolution_param
      {
    num_output: 64
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu1_2"
  type: "ReLU"
  bottom: "conv1_2"
  top: "conv1_2"
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1_2"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv2_1"
  type: "Convolution
      "
  bottom: "pool1"
  top: "conv2_1"
  convolution_param
      {
    num_output: 128
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu2_1"
  type: "ReLU"
  bottom: "conv2_1"
  top: "conv2_1"
}
layer {
  name: "conv2_2"
  type: "Convolution
      "
  bottom: "conv2_1"
  top: "conv2_2"
  convolution_param
      {
    num_output: 128
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu2_2"
  type: "ReLU"
  bottom: "conv2_2"
  top: "conv2_2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2_2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv3_1"
  type: "Convolution
      "
  bottom: "pool2"
  top: "conv3_1"
  convolution_param
      {
    num_output: 256
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3_1"
  type: "ReLU"
  bottom: "conv3_1"
  top: "conv3_1"
}
layer {
  name: "conv3_2"
  type: "Convolution
      "
  bottom: "conv3_1"
  top: "conv3_2"
  convolution_param
      {
    num_output: 256
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3_2"
  type: "ReLU"
  bottom: "conv3_2"
  top: "conv3_2"
}
layer {
  name: "conv3_3"
  type: "Convolution
      "
  bottom: "conv3_2"
  top: "conv3_3"
  convolution_param
      {
    num_output: 256
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3_3"
  type: "ReLU"
  bottom: "conv3_3"
  top: "conv3_3"
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3_3"
  top: "pool3"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv4_1"
  type: "Convolution
      "
  bottom: "pool3"
  top: "conv4_1"
  convolution_param
      {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu4_1"
  type: "ReLU"
  bottom: "conv4_1"
  top: "conv4_1"
}
layer {
  name: "conv4_2"
  type: "Convolution
      "
  bottom: "conv4_1"
  top: "conv4_2"
  convolution_param
      {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu4_2"
  type: "ReLU"
  bottom: "conv4_2"
  top: "conv4_2"
}
layer {
  name: "conv4_3"
  type: "Convolution
      "
  bottom: "conv4_2"
  top: "conv4_3"
  convolution_param
      {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu4_3"
  type: "ReLU"
  bottom: "conv4_3"
  top: "conv4_3"
}
layer {
  name: "pool4"
  type: "Pooling"
  bottom: "conv4_3"
  top: "pool4"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv5_1"
  type: "Convolution
      "
  bottom: "pool4"
  top: "conv5_1"
  convolution_param
      {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu5_1"
  type: "ReLU"
  bottom: "conv5_1"
  top: "conv5_1"
}
layer {
  name: "conv5_2"
  type: "Convolution
      "
  bottom: "conv5_1"
  top: "conv5_2"
  convolution_param
      {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu5_2"
  type: "ReLU"
  bottom: "conv5_2"
  top: "conv5_2"
}
layer {
  name: "conv5_3"
  type: "Convolution
      "
  bottom: "conv5_2"
  top: "conv5_3"
  convolution_param
      {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu5_3"
  type: "ReLU"
  bottom: "conv5_3"
  top: "conv5_3"
}
layer {
  name: "pool5"
  type: "Pooling"
  bottom: "conv5_3"
  top: "pool5"
```

```
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "fc6"
  type: "
      InnerProduct"
  bottom: "pool5"
  top: "fc6"
  inner_product_param
      {
    num_output: 4096
  }
}
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "fc6"
  top: "fc6"
}
layer {
  name: "drop6"
  type: "Dropout"
  bottom: "fc6"
  top: "fc6"
  dropout_param {
    dropout_ratio:
        0.5
  }
}
layer {
  name: "fc7"
  type: "
      InnerProduct"
  bottom: "fc6"
  top: "fc7"
  inner_product_param
      {
    num_output: 4096
  }
}
layer {
  name: "relu7"
  type: "ReLU"
  bottom: "fc7"
  top: "fc7"
}
layer {
  name: "drop7"
  type: "Dropout"
  bottom: "fc7"
  top: "fc7"
  dropout_param {
    dropout_ratio:
        0.5
  }
}
layer {
  name: "fc8_69"
  type: "
      InnerProduct"
  bottom: "fc7"
  top: "fc8_69"
  inner_product_param
      {
    num_output: 69
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc8_69"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "
      SoftmaxWithLoss
      "
  bottom: "fc8_69"
  bottom: "label"
  top: "loss"
}
```

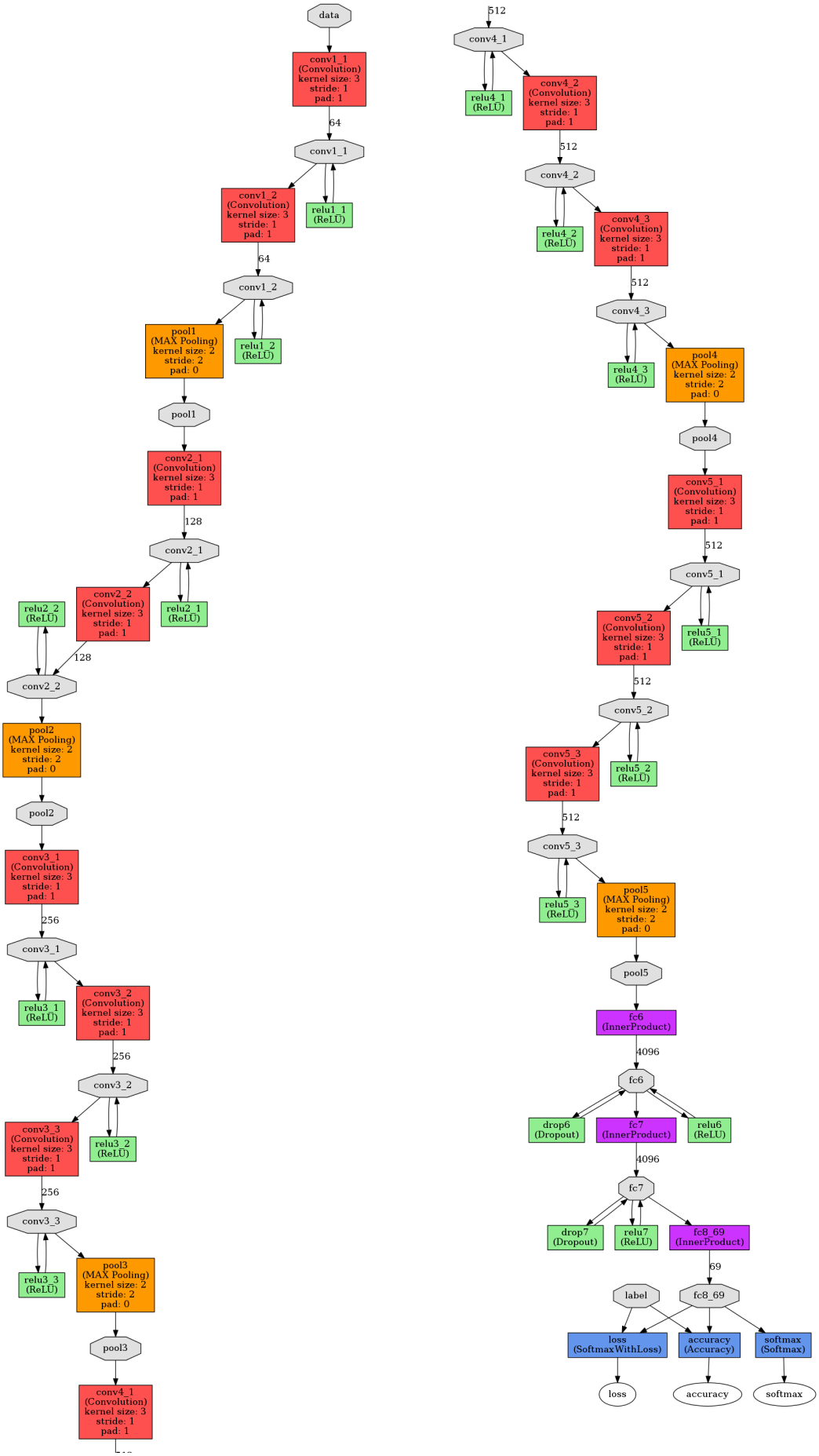


Abbildung A.2: *Visualisierung der VGG16 Architektur in Digits.*

LITERATURVERZEICHNIS

[1] Bild Caesarmörder Wikipedia. https://de.wikipedia.org/wiki/Caesarm\protect{\grmnU@D1ex{\setbox\z@\hbox{\char127}\dimen@-.45ex\advance\dimen@\ht\z@}\accent127\fontdimen5\font\grmnU@Do}\penalty\@M\hskip\z@skiprder. Accessed: 2018-04-14. 24

[2] Blobs, layers, and nets: anatomy of a caffe model. http://caffe.berkeleyvision.org/tutorial/net_layer_blob.html. Accessed: 2018-06-01. 71

[3] Caffe. http://caffe.berkeleyvision.org/. Accessed: 2018-04-01. 20

[4] Caffe: Solver. http://caffe.berkeleyvision.org/tutorial/solver.html. Accessed: 2018-04-10. 46, 58

[5] The CIFAR-10 and CIFAR-100 dataset. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2018-03-21. 37

[6] CS231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks/. Stanford CS class. Accessed: 2018-03-25. 13

[7] Das große Reppa Münzen-lexikon. https://www.reppa.de/lexikon/. Schlagwort: Officina, Münzbild, Münzstätte, Münzmeister . Accessed: 2018-01-10. 25

[8] Lexikon der Neurowissenschaften. http://www.spektrum.de/lexikon/neurowissenschaft/laterale-hemmung/6930. Schlagwort: laterale Hemmung. Spektrum Akademischer Verlag, Heidelberg. Accessed: 2018-03-24. 15

[9] Marktübersicht: Desktop-AOI-Systeme. http://www.all-about-test.info/desktop-aoi.html. Accessed: 2018-03-22. 5

[10] Menschliches Neuron. https://de.wiktionary.org/wiki/Neuron. Bildquelle. Accessed: 2018-03-27. 7

[11] Museum of Fine Arts Boston: Eid Mar Coin. http://www.mfa.org/entry/2016/eid-mar-coin-denarius-with-head-of-m-junius-brutus. Accessed: 2018-04-01. 23, 24

[12] NVIDIA cuDNN GPU Accelerated Deep Learning. https://developer.nvidia.com/cudnn. Accessed: 2018-06-03. 66

[13] NVIDIA DIGITS. https://developer.nvidia.com/digits. Accessed: 2018-04-01. 21

[14] OCRE RIC I (second edition) Augustus 263. http://numismatics.org/ocre/id/ric.1(2).aug.263. Accessed: 2018-04-02. 38

[15] Scikit-learn Beispielimplentierung: Confusion Matrix. http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html. Accessed: 2018-02-18. 19, 72

[16] Tensorflow. https://www.tensorflow.org/. Accessed: 2018-04-23. 20

[17] The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/. Accessed: 2018-03-21. 3, 37

[18] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012. 4

[19] Y. Bengio, F. Bastien, A. Bergeron, N. Boulanger–Lewandowski, T. Breuel, Y. Chherawala, M. Cisse, M. Côté, D. Erhan, J. Eustache, X. Glorot, X. Muller, S. P. Lebeuf, R. Pascanu, S. Rifai, F. Savard, and G. Sicard. Deep learners benefit more from out-of-distribution examples. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 164–172, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. 40

[20] C. M. Bishop. *Pattern Recognition and Machine Learning*, chapter 1. Introduction, pages 3–4. Springer-Verlag New York, Inc., 2006. 3

[21] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Two*, pages 1237–1242. AAAI Press, 2011. 11

[22] G. Dreyfus. *Neural Networks*. Springer-Verlag Berlin Heidelberg, 2005. 36

[23] K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1:119–130, 1988. 12

[24] R. Gebhardt and N. Backes. *Münzkatalog Römische Republik: von 290 v. Chr. bis Marcus Antonius*. Battenberg, 1998. 24, 26

[25] L. Gianazza. A web portal for the inventory of coin finds. *The 6th Joint Meeting of Nomisma.org and ECFN*, 2017. 27

[26] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. 3, 10, 11, 36, 37, 40, 90

[27] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, chapter 5.3 Hyperparameters and Validation Sets, pages 117–119. In [26], 2016. 6

[28] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, chapter 9.1 The Convolution Operation, pages 322–324. In [26], 2016. 12

[29] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, chapter 9.3 Pooling. In [26], 2016. 15

[30] X. Guo, C. Dong, G. Yang, G. Zhou, and Y. Yin. On the class imbalance problem. In *2008 Fourth International Conference on Natural Computation (ICNC)*, volume 04, pages 192–201, 10 2008. 34

[31] I. Guyon. A scaling law for the validation-set training-set size ratio. In *AT & T Bell Laboratories*, 1997. 39

[32] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 40

[33] R. Howard, B. Boser, J. Denker, H. Graf, D. Henderson, W. Hubbard, L. Jackel, Y. Le Cun, and H. S. Baird. Optical character recognition: a technology driver for neural networks. In *IEEE International Symposium on Circuits and Systems, 1990*, volume 3, pages 2433–2436, 1990. 5

[34] M. Jostock. Automatische Münzerkennung mit OpenCV. Bachelorarbeit. 29, 33, 63, 77

[35] H. Kahnt. *Das Grosse Münzlexikon von A bis Z.* Battenberg, 2005. 25

[36] R. Kankelfitz. *Römische Münzen: von Pompejus bis Romulus.* Battenberg, 1996. 1, 25

[37] C. Kirschbaum. *Biopsychologie von A bis Z.* Springer-Lehrbuch. Springer Berlin Heidelberg, 2008. Schlagwörter: Neuron, Soma, Axonhügel. 7

[38] K. Köppl. Dropout. `https://wiki.tum.de/display/lfdv/Dropout`. Convolutional Neural Networks for Image and Video Processing. TU München Wiki. Accessed: 2018-03-24. 15

[39] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. `https://arxiv.org/pdf/1404.5997.pdf`. Accessed: 2017-12-12. 45

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 10, 11, 14, 15, 31, 40, 41, 42

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998. 10, 12

[42] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *ISCAS*, pages 253–256. IEEE, 2010. 11

[43] T. Mitchell. *Machine Learning*, page 2. McGraw-Hill Science/Engineering/-Math, 1997. 3

[44] A. Ng, K. Katanforoosh, and B. Mourri. Train / Dev / Test sets. `https://www.coursera.org/learn/deep-neural-network/lecture/cxG1s/train-dev-test-sets`. Coursera Course: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization. Accessed: 2018-01-27. 39

[45] A. Pangerl. Propaganda auf Römischen Reichsprägungen - Politische Ereignisse und das Bild der kaiserlichen Dynastie. *Antike Welt*, 2012. 23

[46] Dr. Sabel and Prof. Dr. M. Schmidt-Schauss. Skript: Einführung in die methoden der künstlichen intelligenz. `http://www.ki.informatik.uni-frankfurt.de/lehre/SS2014/KI/skript/skript20Aug15.pdf`. Accessed: 2018-05-02. 17, 18

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 14, 15, 41, 42

[48] M. Sugiyama. *Introduction to Statistical Machine Learning*. Morgan Kaufmann Publishers Inc., 2016. 4

[49] S. Sun, W. Chen, L. Wang, and T. Liu. Large margin deep neural networks: Theory and algorithms. *CoRR*, abs/1506.05232, 2015. 43

[50] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. 40

[51] M. Telgarsky. Benefits of depth in neural networks. *CoRR*, abs/1602.04485, 2016. 43

[52] M. Thoma. Analysis and optimization of convolutional neural network architectures. *CoRR*, 2017. 6, 13, 19, 92

[53] M. Thoma. *Analysis and Optimization of Convolutional Neural Network Architectures*, chapter B. Hyperparameters: Tabelle B.6. Network Design, pages 84–85. In [52], 2017. 8

[54] S. Visa. Issues in mining imbalanced data sets - a review paper. In *Proceedings of the Sixteen Midwest Artificial Intelligence and Cognitive Science Conference*, pages 67–73, 2005. 34

[55] B. Vision and L. Center. Model zoo. `https://github.com/BVLC/caffe/wiki/Model-Zoo`. Accessed: 2018-03-03. 39

[56] R. Weiss. The study of ancient numismatics during the renaissance (1313-1517). In *The Numismatic Chronicle*, volume 8, pages 167–178. Royal Numismatic Society, 1968. 23

[57] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014. 40

[58] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, pages 818–833. Springer International Publishing, 2014. 11, 40