



Technische Hochschule Nürnberg
Georg Simon Ohm
Fakultät Informatik
Studiengang Medieninformatik

Abschlussarbeit
zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

Automatische Klassifizierung von digitalisierten Dokumenten durch Verwendung eines neuronalen Netzes

Vorgelegt von Tobias Velke
am 28. Februar 2017
Erstprüfer Prof. Dr. Ing. Florian Gallwitz
Zweitprüfer Prof. Dr. Jens Albrecht
Betreuer Dipl.-Ing. Holger Velke

mit Unterstützung der mediendesign AG





Abstract

Das Ziel der vorliegenden Bachelorarbeit ist es, für die Firma mediendesign AG eine prototypische Anwendung zu entwickeln, um automatisiert digitale Dokumente zu sortieren. Wurden in der Vergangenheit oftmals Methoden der Textanalyse bei der Klassifikation von Dokumenten eingesetzt, soll in dieser Arbeit dagegen die Machbarkeit auf Basis eines künstlichen neuronalen Netzes untersucht werden. Um ein bestmögliches Ergebnis zu erzielen, werden zwei unterschiedliche Lösungsstrategien verfolgt. Für den ersten Ansatz wird auf Transferlearning zurückgegriffen, für den zweiten werden eigene Modelle erstellt.

Nachdem an das Problem herangeführt wurde, werden die wesentlichen Aspekte künstliche neuronale Netze und speziell Convolutional Neural Networks im Detail erläutert.

Die Vorgehensweise bei der Implementierung der Anwendung in der Programmiersprache Python, und dabei insbesondere die Erstellung der Klassifikatoren mit Tensorflow und Keras sowie deren Evaluation, bilden den Schwerpunkt dieser Arbeit. Dabei wird gezeigt, dass durch iterative Optimierung der Klassifikatoren ein vergleichbar gutes Ergebnis mit beiden Lösungsansätzen erzielt wird. Im Anschluss an die Evaluation wird eine Einschätzung zu den sehr guten Ergebnissen beider Lösungsstrategien sowie eine Empfehlung für den produktiven Einsatz gegeben.



Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	VI
Verzeichnis der Listings	VII
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Aufbau der Arbeit	2
2 Anwendungsszenario	4
2.1 Klassifikation	4
2.1.1 Klassifikation von Mustern	4
2.1.2 Merkmale zur Klassifikation	5
2.1.3 Vorgehensweise bei der Klassifizierung	7
2.2 Beschreibung der Klassifikations-Aufgabe	8
2.3 Lösungsansatz	8
2.3.1 Genereller Aufbau des Klassifikations-Systems	9
2.3.2 Vortrainierter Klassifikator	10
2.3.3 Mehrstufiger Klassifikator	11
3 Dokumente	12
3.1 Datengrundlage und Aufteilung der Stichprobe	12
3.2 Stichprobe aus Originaldokumenten	15
3.3 Stichprobe aus RVL-CDIP Datensatz	17
4 Stand der Technik	19
4.1 Künstliche neuronale Netze	19
4.1.1 Künstliche Neuronen	19
4.1.2 Struktur künstlicher neuronaler Netze	20
4.1.3 Trainingsphase	21
4.1.4 Überanpassung und Unteranpassung	22
4.2 Convolutional Neural Network	23
4.2.1 Struktur eines Convolutional Neural Network	23
4.2.2 Faltung	24



4.2.3	Aktivierungsfunktion	27
4.2.4	Pooling	28
4.2.5	Batch Normalization	29
4.3	Aktuelle Ansätze zur Klassifikation von Dokumenten	30
5	Vewendete Frameworks	33
5.1	Die Programmiersprache Python	33
5.2	Machine Learning Framework	33
5.2.1	TensorFlow	34
5.2.2	Keras	35
6	Prototypische Umsetzung	37
6.1	Modellbildung und Training des Netzes	37
6.1.1	Vortrainierter Klassifikator	38
6.1.2	Mehrstufiger Klassifikator	41
6.2	Implementierung einer Anwendung zur Klassifikation	44
6.2.1	Server-Anwendung zur Klassifikation	44
6.2.2	Client-Anwendung	45
7	Evaluation der Klassifikatoren	47
7.1	Evaluation des neuronalen Netzes	47
7.1.1	Kenngrößen zur Evaluation	47
7.1.2	Evaluation der Klassifikatoren	49
7.2	Auswahl des Klassifikators	51
7.3	Hypothese	52
8	Schlussbetrachtung	54
8.1	Fazit	54
8.2	Ausblick	55
Literatur		56
A	Anhang	ii
A.1	Architekturen der Modelle des Mehrstufigen Klassifikators	ii
A.2	Testergebnisse ohne Rotation und Scherung von Dokumenten	v
A.3	Trainingsverlauf ohne Batch-Normalization-Layer	vii
Eidesstattliche Erklärung		viii



Abkürzungverzeichnis

API Application Programming Interface

CNN Convolutional Neural Network

ERP Enterprise-Resource-Planning

FKR Falschklassifikationsrate

GUI Graphical User Interface

KKR Korrektklassifikationsrate

MLP Multilayer Perzeptron

OCR Optical Character Recognition

ReLU Rectified Linear Unit

RVL-CDIP Ryerson Vision Lab Complex Document Information Processing



Abbildungsverzeichnis

2.1	Strukturelle Merkmale	6
2.2	Prinzipieller Aufbau des Systems	9
3.1	Beispiel eines Dokuments mit verringelter Auflösung	13
3.2	Aufteilung der Stichprobe	14
3.3	Originaldokumente	16
3.4	RVL-CDIP Dokumente	17
3.5	Varianz der Muster	18
3.6	Ähnlichkeit der Muster	18
4.1	Multilayer Perzeptron	21
4.2	Dropout	23
4.3	Topologie eines Convolutional Neural Network	23
4.4	Prinzip der Faltung	24
4.5	Rezeptives Feld	25
4.6	Vergrößertes rezeptives Feld	25
4.7	Zero-Padding	26
4.8	Aktivierungsfunktion	28
4.9	Max-Pooling	28
5.1	TensorFlow Graph	34
6.1	Filter-Ersetzung im InceptionV3	38
6.2	Architektur InceptionV3	39
6.3	Trainingsverlauf „Vortrainierter Klassifikator“	40
6.4	Trainingsverlauf „Mehrstufiger Klassifikator“	43
6.5	Unteranpassung Intern-Klassifikator	44
6.6	Sequenzdiagramm der Anwendung	46
7.1	Konfusionsmatrix	47
7.2	Konfusionsmatrix „Vortrainierter Klassifikator“	50
7.3	Konfusionsmatrix „Mehrstufiger Klassifikator“	51
7.4	Angebotsschreiben und Leistungsbeschreibung in 150×150 -Pixel Auflösung	53
A.1	Architektur Intern-Extern-Modell	ii



A.2 Intern-Modell	iii
A.3 Extern-Modell	iv
A.4 Konfusionsmatrix „Vortrainierter Klassifikator“	v
A.5 Konfusionsmatrix „Mehrstufiger Klassifikator“	vi
A.6 Trainingsverlauf ohne Batch-Normalization	vii
A.7 „Mehrstufiger Klassifikator“ ohne Batch-Normalization	vii



Tabellenverzeichnis

3.1	Anzahl Originaldokumente	15
6.1	Architektur „Mehrstufiger Klassifikator“	42
7.1	Auswertung „Vortrainierter Klassifikator“	49
7.2	Auswertung des „Mehrstufigen Klassifikators“	50
7.3	Korrektklassifizierungsrate, Falschklassifizierungsrate	52
A.1	Auswertung „Vortrainierter Klassifikator“ ohne Veränderungen . . .	v
A.2	Auswertung „Mehrstufiger Klassifikator“ mit Rotation & Scherung .	vi



Verzeichnis der Listings

5.1 Keras Beispiel	35
6.1 Erstellen und zweimaliges Trainieren eines Modells mit der Klasse „Finetuner“	40
6.2 Erstellen eines Modells mit der Klasse „Basemodel“	41
6.3 REST Schnittstelle	44
6.4 Keras Predict Methode	45



1 Einleitung

1.1 Motivation

Bis vor einigen Jahren wurden maschinelle Lernverfahren wie künstliche neuronale Netze vorwiegend von großen Unternehmen mit großen Rechenzentren erprobt und eingesetzt. Dank steigender Rechenleistung und immer günstigerer Hardware, wird es auch für kleine Unternehmen einfacher möglich, maschinelle Lernverfahren einzusetzen [RM18]. Dadurch ergeben sich neue Bereiche und Produkte, in welchen diese Verfahren verwendet werden. Weiter entstanden in jüngster Zeit auch mächtige Programm-Bibliotheken, welche die Implementierung dieser Verfahren vereinfachen. Diese Voraussetzungen führen dazu, dass sich das maschinelle Lernen als eines der aktuell spannendsten Betätigungsfelder für Informatiker entwickelt.

Durch die zunehmende Digitalisierung werden Dokumente immer häufiger elektronisch versendet. Allerdings werden nach wie vor auch immer noch viele wichtige Dokumente auf dem Postweg zugestellt. Diese Dokumente werden in Unternehmen nach der Zustellung digitalisiert. So kann der Schriftverkehr zusammen mit allen weiteren Informationen eines Projektes oder eines Kunden zentral auf einer Plattform verwaltet werden. Auch für eine effiziente Archivierung der Schriftstücke ist deren Digitalisierung sinnvoll. Deshalb müssen die auf dem Postweg zugestellten Sendungen nach der Digitalisierung je nach Art des Dokuments in eine Ordnerstruktur eingesortiert oder in ein Enterprise-Resource-Planning (ERP)-System eingepflegt werden.

Bei der Firma mediendesign AG werden bestimmte Postsendungen wie Rechnungen, Vertragspapiere oder Bestellungen eingescannt. Anschließend werden die Scans von einem Bearbeiter nochmals geöffnet und gesichtet, bevor sie händisch einem Projekt oder einem Kunden zugeordnet werden.

Wünschenswert wäre hier ein System, welches diesen manuellen Vorgang unterstützt. Ein solches System müsste Dokumente bewerten können, um Aussagen zu treffen, um welche Art Dokumente es sich handelt. Das System müsste also die vorab digitalisierten Dokumente klassifizieren können.

In dieser Arbeit soll die zentrale Fragestellung bearbeitet werden, ob es mithilfe eines neuronalen Netzes möglich ist, digitalisierte Dokumente anhand eines Bildes der ersten Seite des jeweiligen Dokuments vorgegebenen Dokumentenklassen zuzuordnen.



1.2 Zielsetzung

Ziel der Arbeit ist es, ein geeignetes künstliches neuronales Netz zu evaluieren, um digitalisierte Dokumente anhand der ersten Seite zuverlässig klassifizieren zu können. Im Anschluss soll eine prototypische Anwendung entwickelt werden, die für einen Benutzer eine automatische Klassifikation von Dokumenten durchführt und dazu das künstliche neuronale Netz als Klassifikator verwendet.

Durch die Geschäftsführung der Firma mediendesign AG wurden folgende Klassen festgelegt: *Leistungsbeschreibung*, *Angebotsschreiben*, *Eingangsrechnung*, *Leistungsnachweis_md*, *Leistungsnachweis_sy* und *Ausgangsrechnung*.

Die zu entwickelnde Anwendung muss nach der Auswahl und dem Training des Netzes Dokumente, welche der Applikation als Bild- oder Text-Dateien übergeben werden, zuverlässig in die vorgegebenen Klassen einordnen. Nach der Klassifizierung sollen die Dateien in eine, den Klassennamen entsprechende Ordnerstruktur eingesortiert werden.

In dieser Arbeit wird darauf verzichtet, den Inhalt der einzelnen Dokumente semantisch zu bewerten. Vielmehr sollen die einzelnen Dokumentklassen anhand von Struktur- und Layoutelementen erlernt und unterschieden werden können. Aus diesem Grund wird auch nicht der Ansatz verfolgt, die einzelnen Dokumente gleich bestimmten Kunden oder Projekten zuzuordnen, sondern sie vorerst in eine Ordnerstruktur zu verschieben, welche die einzelnen Klassen repräsentiert.

1.3 Aufbau der Arbeit

Das zweite Kapitel wird zunächst an das Thema heranführen, indem der Begriff der Klassifikation sowie die gegebene Problemstellung detailliert erläutert werden. Daran anschließend werden Strategien zur Lösung der Aufgabe skizziert.

Im dritten Teil der Arbeit werden die zugrunde liegenden Daten zum Training des Klassifikators beschrieben. Es werden dabei Grundlagen für die Erstellung von Stichproben erläutert. Außerdem werden die zwei Stichproben, die in dieser Arbeit verwendet werden, genauer betrachtet.

Das vierte Kapitel behandelt Neuronale Netze und wird dabei den Schwerpunkt auf *Convolutional Neural Networks (CNNs)* legen, da diese Typen von künstlichen neuronalen Netzen zur Klassifikation von Bildern eingesetzt werden. Ebenfalls in



diesem Kapitel wird auf aktuelle Arbeiten zum Thema der Dokumentenklassifikation eingegangen. Dabei werden Parallelen sowie Abgrenzungen zu dieser Arbeit hergestellt.

Vor der Beschreibung der Implementierung und dem Training des Klassifikators, werden im fünften Kapitel die dazu verwendeten Werkzeuge vorgestellt. Anschließend werden im sechsten Kapitel die Modellbildung des Klassifikators und die Entwicklung der späteren Anwendung den Schwerpunkt bilden.

Im siebten Kapitel werden die trainierten Modelle bewertet und der Klassifikator für den produktiven Einsatz im Unternehmen ausgewählt.

Abschließend werden die Ergebnisse der Arbeit zusammengefasst, bevor ein Ausblick gegeben wird, wie das System eingesetzt und gegebenenfalls weiterentwickelt werden kann.



2 Anwendungsszenario

2.1 Klassifikation

Die Sortierung der Dokumente ist eine Klassifikations-Aufgabe, deshalb soll in diesem Abschnitt zunächst der Begriff der Klassifikation definiert werden. Anschließend werden semantische und strukturelle Merkmale an Beispielen erklärt, bevor die eigentliche Klassifikations-Aufgabe erläutert wird.

2.1.1 Klassifikation von Mustern

Die Klassifikation ist ein Teilgebiet der Mustererkennung und besteht darin, einen Teil der gesamten Umwelt in disjunkte Bereiche zu unterteilen und die zu untersuchenden Testobjekte diesen Teilbereichen zuzuordnen [Nie03].

Definition 2.1. Ein *Problemkreis* Ω bezeichnet einen für eine konkrete Anwendung spezifischen Teil der Umwelt U [Nie03]. Alle Objekte eines Problemkreises können als Funktionen ${}^{\varrho}\mathbf{f}(\mathbf{x})$ beschrieben werden und bilden die Menge:

$$\Omega = \{{}^{\varrho}\mathbf{f}(\mathbf{x}) \mid \varrho = 1, 2, \dots\} \subset U \quad (2.1)$$

Der Problemkreis Ω bildet innerhalb der gesamten Umwelt U einen Merkmalsraum. In diesem liegen alle Funktionen, die Elemente des Problemkreises beschreiben.

Definition 2.2. Die Elemente eines Problemkreises Ω werden als *Muster* bezeichnet. Das bedeutet, jedes Muster stellt eine Funktion \mathbf{f} dar [Nie03].

$$\mathbf{f}(\mathbf{x}) = \begin{cases} f_1(x_1 \dots x_n) \\ \vdots \\ f_m(x_1 \dots x_n) \end{cases} \quad (2.2)$$

Der in dieser Arbeit behandelte Problemkreis Ω ist die Menge aller digitaler Bilder von Dokumenten, wobei jedes Dokument eines der zu untersuchenden Muster darstellt. Innerhalb des Problemkreises Ω müssen Partitionen für die Dokument-Typen gebildet werden. Diese Partitionen werden als *Klassen* bezeichnet.



Definition 2.3. Durch Zerlegung des Problemkreises Ω in k oder $k + 1$ Teilmengen ergeben sich Klassen [Nie03]. Dabei repräsentiert jede Klasse Ω_κ eine Menge an Mustern des Problemkreises mit:

$$\begin{aligned} \Omega_\kappa &\neq \emptyset, \kappa = 1 \dots k \\ \Omega_\kappa \cap \Omega_\lambda &= \emptyset, \lambda \neq \kappa \\ \text{entweder } \bigcup_{\kappa=1}^k \Omega_\kappa &= \Omega \text{ oder } \bigcup_{\kappa=0}^k \Omega_\kappa = \Omega \end{aligned} \tag{2.3}$$

Der Index k wird durch die Anzahl der Klassen bestimmt, die das System unterscheiden soll. Bei $k + 1$ bildet Ω_0 die Rückweisungsklasse.

Definition 2.4. Bei der *Klassifikation einfacher Muster* wird jedes Muster einzeln als Ganzes betrachtet. Das betrachtete Muster muss genau einer der definierten Klassen Ω_κ zugeordnet werden. Ist dies nicht möglich, muss das Muster der Rückweisungsklasse Ω_0 zugeordnet werden. [Nie03]

Die Klassifikation einfacher Muster kann nach obigen Definitionen als Abbildung betrachtet werden, die einem Muster $f(x)$ eine Zahl $\kappa \in 0, 1 \dots k$ als Klasse des partitionierten Problemkreises Ω_κ zuordnet [Nie03].

2.1.2 Merkmale zur Klassifikation

Die große Zahl diskreter Abtastwerte bei der Digitalisierung eines Musters macht eine direkte Klassifikation in den meisten Fällen unmöglich [Nie03]. Im Fall des in dieser Arbeit besprochenen Klassifikations-Problems müsste der Klassifikator, bei einer Verkleinerung der Dokumente auf 150×150 -Pixel, aufgrund von 22.500 Abtastwerten entscheiden, um welchen Dokumenttyp es sich handelt. Ein Großteil dieser enthält jedoch keine für die Klassifikation wichtigen Informationen. Es ist deshalb sinnvoll Merkmale innerhalb der Muster einer Klasse zu suchen, welche für diese Klasse spezifisch sind und sie von Mustern anderer Klassen abgrenzen.

Um Muster verschiedener Klassen unterscheiden zu können, genügt es, anstatt aller Funktionswerte eines Musters, nur wesentliche Merkmale zu bewerten. Können solche spezifischen Merkmale identifiziert werden, kann die Performance eines Klassifikators verbessert werden. Denn die zu untersuchende Datenmenge wird verringert und somit auch die benötigten Parameter eines Klassifikators [Nie03]. Ebenso kann die Güte des Klassifikators erhöht werden, denn wenige trennscharfe Merkmale



bilden im Merkmalsraum einen kompakteren Bereich, was eine bessere Trennung der Klassenbereiche erlaubt. Je kompakter der Merkmalsraum einer Klasse ist, umso besser kann ein Klassifikator diese Klasse abstrahieren.

Für numerische Klassifikatoren, zu welchen auch künstliche neuronale Netze zählen, werden die Merkmale eines Musters in einem n -dimensionalen Vektor zusammengefasst. Diese können bei der Klassifikation digitaler Dokumente anhand semantischer oder struktureller Merkmale gebildet werden.

Semantische Merkmale

Bei der semantischen Klassifikation wird aufgrund des Inhalts eines Dokuments entschieden, welcher Klasse es angehört. Hier bilden Begriffe oder Phrasen die zu unterscheidenden Merkmale. Ein Dokument wird zunächst mit *Optical Character Recognition (OCR)* abgescannt. Anschließend werden aus den extrahierten Wörtern, Satzfragmenten oder Zahlen die Merkmalsvektoren gebildet.

Strukturelle Merkmale

Neben semantischen Merkmalen können auch die Struktur oder Layout-Elemente der Dokumente Merkmale bieten, die eine Klassifizierung ermöglichen. Dies ist darin begründet, dass Dokumente gleicher Klassen immer ähnlichen Regeln im Aufbau folgen [Afz+15, S. 1111-1115].

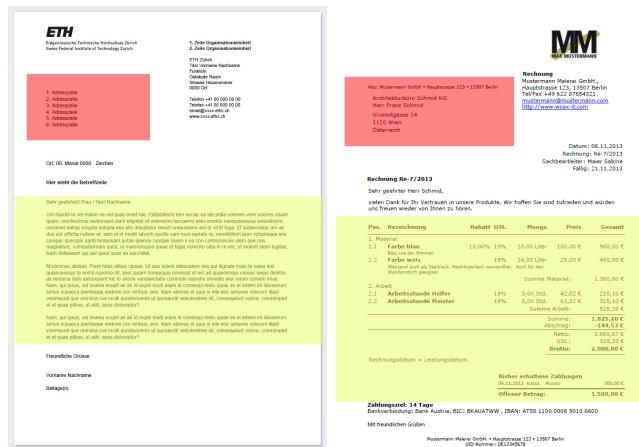


Abbildung 2.1: Beispiel Geschäftsbrief und Rechnung



Die Abbildung 2.1 zeigt ein Beispiel für einen Geschäftsbrief und eine Rechnung. Dokumente dieser beiden Klassen werden in ihrem jeweiligen Layout immer Unterschiede aufweisen. Diese sind in der Abbildung durch die gelben Bereiche gekennzeichnet. Bei Geschäftsbriefen ist zentral ein größerer zusammenhängender Textblock vorhanden, dagegen sind Rechnungen in diesem Bereich mehrheitlich tabellarisch aufgebaut. Allgemein sind in Rechnungsdokumenten mehr strukturierende Elemente wie Trennlinien zu finden, als dies in Geschäftsbriefen der Fall ist. Diese Unterschiede können als Merkmale zur Klassifikation dienen.

Die roten Flächen in Abbildung 2.1 markieren Bereiche, die in ähnlicher Form in den meisten maschinell erstellten Dokumenten zu finden sind. Solche Elemente, wie Empfängeradresse oder fett gedruckte Betreffzeile, liefern also keine eindeutigen Merkmale.

In Textdokumenten ist die Fläche mit gedruckten Zeichen im Verhältnis zur Gesamtfläche des Dokuments klein. Folglich sind große Bereiche in den Bildern weiß. Im Gegensatz zu „natürlichen Bildern“ enthalten Bilder digitaler Dokumente in vielen Pixeln somit weniger Information. Auch die Merkmale in Dokument-Bildern sind weniger vielfältig als in „natürlichen Bildern“, so sind senkrechte und waagrechte Kanten deutlich häufiger vorhanden als runde Formen oder geschwungene Linien.

2.1.3 Vorgehensweise bei der Klassifizierung

Die prinzipielle Vorgehensweise aller Systeme zur Lösung von Klassifikations-Aufgaben beinhalten immer die folgenden gleichen Schritte [Nie03].

- **Datenerfassung:** Aufnahme der Daten und Wandlung in ein für das Klassifikationssystem verarbeitbares Format.
- **Vorverarbeitung:** Aufbereitung und Verbesserung der Qualität der Daten.
- **Merkmalsextraktion:** Spezifische Merkmale in den Daten suchen und extrahieren, um sie dem Klassifikator zuzuführen.
- **Klassifikation:** Der Klassifikator entscheidet auf Grund der ihm zugeführten Merkmale über die Klassenzugehörigkeit eines Objekts.

Damit der Klassifikator eine Entscheidung treffen kann, müssen ihm die Klassengrenzen bekannt sein [Nie03]. In einer Trainingsphase müssen diese aus den Daten einer repräsentativen Stichprobe bestimmt werden.



2.2 Beschreibung der Klassifikations-Aufgabe

Nachdem der Begriff der Klassifikation erläutert und die grundsätzliche Vorgehensweise bei Klassifikations-Aufgaben und der Merkmalsbildung beschrieben wurde, soll nun die konkret zu lösende Aufgabe beschrieben werden.

Ziel ist es, einen Klassifikator zu evaluieren, welcher in der Lage sein muss, ein Bild eines Dokuments einer der in Abschnitt 1.2 festgelegten Klassen zuzuordnen. Der Klassifikator muss für ein Muster die Wahrscheinlichkeit liefern, mit der dieses Muster einer der definierten Klassen angehört.

Da es sich bei den einzelnen Dokumenten um potentiell betriebswirtschaftlich wichtige Dokumente handelt, muss sichergestellt werden, dass nach der Klassifizierung keines der Dokumente für die weitere Verarbeitung unberücksichtigt bleibt.

Kein Klassifikator ist in der Lage, eine absolute Genauigkeit zu erreichen. Somit muss für nicht eindeutig klassifizierbare Muster eine Rückweisungsklasse gebildet werden. Dadurch ist es möglich, einen Grenzwert festzulegen, bis zu welcher Vorhersage-Wahrscheinlichkeit ein Dokument dieser Rückweisungsklasse zuzuordnen ist. Nur wenn der Klassifikator für ein Dokument eine Klassenwahrscheinlichkeit über dem vorgegebenen Grenzwert ermittelt, soll das Dokument in diese Klasse eingesortiert werden. Andernfalls ist das Dokument der Rückweisungsklasse zuzuordnen und muss im Anschluss von einem Bearbeiter händisch klassifiziert werden.

Ein weiterer Aspekt, der bei der Klassifizierung von Unternehmensdokumenten zu berücksichtigen ist, ist der Datenschutz. Die Dokumente enthalten sensible, teils personenbezogene Daten, weshalb sichergestellt werden muss, dass die Dokumente durch das System vertraulich behandelt werden. Um dies zu erreichen, darf dem Klassifikator nicht das Originaldokument, sondern nur eine anonymisierte Kopie übergeben werden.

2.3 Lösungsansatz

Dieser Abschnitt wird zunächst ein grundsätzliches Konzept zur Gestaltung des gesamten Systems liefern und im Weiteren Strategien zur Lösung der gestellten Aufgabe vorstellen.



2.3.1 Genereller Aufbau des Klassifikations-Systems

Das System wird in zwei Teilen entwickelt. Ein Teil wird dabei clientseitig die Dokumente entgegen nehmen und in ein einheitliches Format wandeln. Der zweite Teil beinhaltet das CNN als Klassifikator und wird auf einem Server die Klassifikation der Dokumente vornehmen. Abbildung 2.2 zeigt den schematischen Aufbau des Systems.

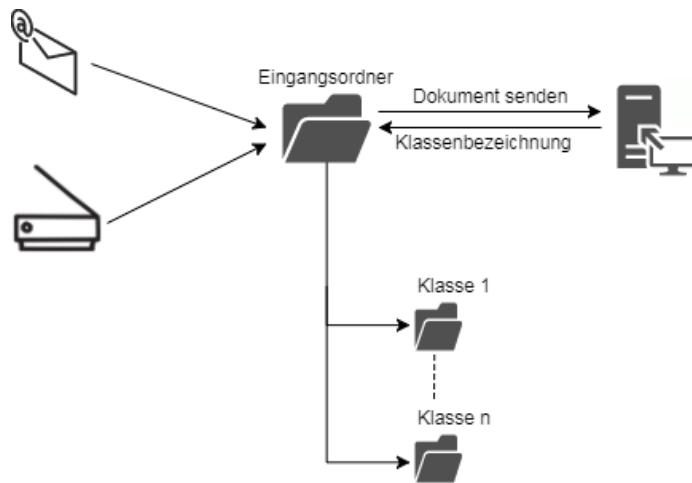


Abbildung 2.2: Prinzipieller Aufbau des Systems

Ein Benutzer kann einen Ordner auf einem Laufwerk angeben, dessen Inhalt von dem Klassifikator bewertet werden soll. Der Inhalt des Ordners wird nach elektronischen Dokumenten durchsucht und Kopien dieser werden an den Server geschickt, wo sie klassifiziert werden. Als Antwort erhält die Client-Anwendung eine Aussage über die Klassenzugehörigkeit des Dokuments, woraufhin das Originaldokument in einen entsprechenden Ordner verschoben wird.

Die Aufteilung bietet den Vorteil, dass clientseitig nur eine relativ kleine Anwendung installiert werden muss. Die Server-Anwendung, welche das Modell des CNNs beinhaltet, muss nur einmalig installiert werden. Dadurch kann der gesamte Klassifikator auch ausgetauscht werden, falls die Klassen angepasst werden sollen und sich dadurch das Modell verändert. Da die Modelle eines CNNs schnell sehr groß werden können, bildet das Laden des Modells im Vergleich zur eigentlichen Klassifikation meist den zeitlichen Flaschenhals. Auf dem Server kann der Klassifikator durchgehend im Speicher gehalten werden und muss nicht für jede Klassifikation neu geladen werden. Auf diese Weise kann der Engpass umgangen werden.



Diese prinzipielle Aufteilung der Anwendung soll in jedem Fall beibehalten werden. Denn dadurch wird es auch möglich, unternehmensweit Dokumente durch das CNN klassifizieren zu lassen. So kann jedem Scanner vorgegeben werden, in welchen Ordner ein abgescanntes Dokument gespeichert werden soll. Oder ein Benutzer kann für sein E-Mail-Postfach eine Regel erstellen, um Anhänge automatisch immer in den selben Ordner zu speichern. Diese Ordner können durch die Anwendung im Hintergrund überwacht werden und beim Eintreffen einer neuen Datei kann diese automatisch klassifiziert werden.

Nachdem der grundsätzliche Aufbau der Anwendung erläutert wurde, werden nun zwei Ansätze zum Aufbau des eigentlichen Klassifikators skizziert.

2.3.2 Vortrainierter Klassifikator

Bei der Verwendung von CNNs hat es sich bewährt, auf bereits vortrainierte Modelle zurückzugreifen und diese für den eigenen Anwendungszweck zu adaptieren. Ein solches Vorgehen wird als *Transferlernen* (engl. *Transferlearning*) bezeichnet. Die Methode ist insbesondere dann vorteilhaft, wenn für ein Klassifikations-Problem nur eine geringe Anzahl an Trainingsdaten zur Verfügung steht. Denn diese Modelle wurden auf sehr großen oder auf für bestimmte Problemstellungen speziell erstellten Datensätzen trainiert.

Ein Standard-Datensatz ist das 2009 vorgestellte *ImageNet* [Den+09, S. 248-255]. Das ImageNet ist eine der größten öffentlich zugänglichen Bilddatenbanken mit 14.197.122 annotierten Bildern in 1.000 Objektkategorien. Modelle, die auf diesem Datensatz trainiert wurden, können aufgrund der großen Menge an diversen Daten sehr generelle Merkmale erlernen.

Für diesen Lösungsansatz wird ein solches Modell zusammen mit den bereits gelernten Merkmalen verwendet. Dazu werden die vorderen Schichten samt Gewichte beibehalten, da diese für die Extraktion der Merkmale verantwortlich sind. Nur die letzten Schichten müssen in diesem Ansatz auf das gegebene Klassifikations-Problem angepasst werden.

Als Architektur wird ein *InceptionV3*-Modell verwendet, da dieses Modell mit einer Korrektklassifikationsrate (KKR) von 94,4% sehr gute Ergebnisse bei der *Large Scale Visual Recognition Challenge (ILSVRC)* im Jahr 2014 erzielt hat [Rus+15, S. 211–252]. Trotz einer großen Anzahl an Parametern kann das Modell noch gut auf einem normalen PC verarbeitet werden.



Die folgende Hypothese ist ausschlaggebend für diese Vorgehensweise. Die Gewichte des vortrainierten InceptionV3-Modells wurden auf Basis des ImageNet Datensatzes ermittelt. Dadurch wurde eine Vielzahl genereller Merkmale gelernt. Nun wird dieses Modell mit Dokumenten des *Ryerson Vision Lab Complex Document Information Processing (RVL-CDIP)* Datensatzes, der in Abschnitt 3.3 genauer betrachtet wird, erneut trainiert. So werden Merkmale extrahiert, die für maschinell erstellte Dokumente typisch sind. Anschließend wird das angepasste Modell ein zweites Mal auf den letztendlichen Dokumenten des Unternehmens trainiert. Aufgrund dieses mehrfachen Trainings sollen, trotz der relativ geringen Menge an zur Verfügung stehender Dokumente des Unternehmens, gute Ergebnisse bei der Klassifikation erzielt werden.

2.3.3 Mehrstufiger Klassifikator

Als zweite Variante eines Klassifikators soll ein System aus mehreren nacheinander geschalteten Modellen getestet werden. Die Klassen werden für diesen Lösungsansatz in interne und externe Dokumente unterteilt.

Für die erste Stufe des Klassifikators wird ein Modell trainiert, welches Dokumente der Klassen *Intern* und *Extern* unterscheiden soll. Abhängig vom Ergebnis dieser ersten Klassifikation, wird das Dokument an einen weiteren Klassifikator weitergegeben. Diese zweite Stufe entscheidet dann über die spezifischere Klassenzugehörigkeit des entsprechende Dokuments.

Das CNN, an welches die als *Intern* klassifizierten Dokumente übergeben wurden, muss diese nun in die Klassen *Leistungsbeschreibung*, *Leistungsnachweis-md*, *Angebotschreiben* und *Ausgangsrechnung* einteilen. Werden in der ersten Stufe Dokumente als *Extern* gekennzeichnet, müssen diese an ein Modell übergeben werden, welches die Klassen *Leistungsbeschreibung-sy* und *Eingangsrechnung* erkennen kann.

Die Hypothese hinter diesem Ansatz ist, dass durch die Aufteilung der Klassen die einzelnen Klassifikations-Aufgaben vereinfacht werden. Ein Modell muss maximal vier anstatt sechs Klassen unterscheiden. So sollen drei Modelle erstellt werden, die zu einem Klassifikator kombiniert werden und die gestellte Klassifikations-Aufgabe zuverlässig lösen.



3 Dokumente

3.1 Datengrundlage und Aufteilung der Stichprobe

Im Abschnitt 2.1.1 wurde gezeigt, dass es sich bei der Klassifikation um das Vergleichen von Merkmalen eines Objekts mit für eine Klasse spezifischen Merkmalen handelt. Während der Lernphase muss der Klassifikator möglichst generelle Merkmale der einzelnen Klassen lernen, um diese im späteren Einsatz auf die Eingangsmuster anzuwenden. Dazu wird eine repräsentative Stichprobe der Klassen benötigt. Bei der Erstellung dieser Stichprobe ist es wichtig, dass die darin enthaltenen Muster auch die relevanten Merkmale abbilden und in ausreichend großer Anzahl vorliegen.

Ein Trainingsmuster ist nach Definition 2.2 eine Funktion des Problemkreises Ω , für das die Klasse bekannt ist. Die gesamte Stichprobe besteht also aus einer Menge an Mustern mit bekannter Klassenaussage. Jedes Muster kann als Vektor betrachtet werden, dessen Komponenten die einzelnen Merkmale widerspiegeln. Durch diese Repräsentation kann ein Klassifikator diese Merkmale systematisch für die einzelnen Klassen erlernen.

Je mehr Trainingsdaten zur Verfügung stehen, umso robustere Ergebnisse des Klassifikators sind zu erwarten. Allerdings erhöht eine große Trainingsstichprobe auch den zeitlichen Aufwand des Lernens. Während der Erstellung der Stichprobe muss darauf geachtet werden, dass keine Muster falsch annotiert werden, da sonst falsche Klassenmerkmale erlernt werden.

Bei einer geringen Menge an Trainingsdaten oder bei sehr gleichartigen Mustern innerhalb einer Klasse steigt das Risiko der Überanpassung [RM18]. Deshalb wird durch gezielte Bearbeitung der Muster versucht, die Anzahl sowie die Varianz der Daten zu vergrößern. Dazu können Operationen wie Rotation, Spiegelung, Scherung oder Hinzufügen von Rauschen auf die Elemente der Stichprobe angewandt werden.

Um den Klassifikator trainieren zu können, müssen aus den Dokumenten, welche als Textdateien vorliegen, die jeweils ersten Seiten extrahiert und in ein Bilddateiformat gewandelt werden. Für die gängigsten Formate „.doc“, „.docx“ und „.pdf“ wird diese Umwandlung implementiert, da Dokumente dieser Formate auch im späteren produktiven Einsatz klassifiziert werden sollen. Bei der Entwicklung der dafür notwendigen Methoden wird darauf geachtet, eine spätere Wiederverwendung im clientseitigen Teil der Anwendung zu ermöglichen.



Zum Trainieren der Modelle werden in dieser Arbeit zwei unterschiedliche Datenquellen verwendet. Der erste Datensatz besteht aus originalen Dokumenten des Unternehmens, der zweite Datensatz ist ein öffentlich zugänglicher Datensatz.

mediendesign

mediendesign, Bartholomäusstraße 26, DE-90499 Nürnberg

[Company Name]
 [Name]
 [Street No]
 [Area Code]/[Town]
 [Country]

2012-XX-X
Offer
[Project Name]
[XXXXX-P12XX]

Dear Mr./Mrs [Name],
 we appreciate your interest in our services.
 Our offer is based on the enclosed specification, dated
 2012-XX-XX, in which we have evaluated all necessary tasks.

Costs and Expenses

Task	Expenses in man-days
[Task]	XX,XX
Total Expenses	XX,XX

Total expenses add up to [XX,XX] man-days. We charge
 € XXX,- per man-day, so total costs would be € [XX,XXX,-].

Optional:
 [We offer a reduced project price of € XX,XXX,- for all specified tasks.]

Travel Costs
 Travel costs are already included in the project price.
 or
 Travel costs will be reimbursed based on receipts; travels by car
 will be charged with € 0,60 per kilometer.

mediendesign - Aktiengesellschaft für Informationsmanagement und Unternehmenskommunikation, Bartholomäusstraße 26, 90499 Nürnberg, Deutschland
 Tel. +49 911-2 39 06 39, Fax +49 911-2 39 06 394, E-Mail: info@mediendesign.de, Aufenthaltsverordnungs-Nr. Thüring. Günzburg 2010 Jörg Meier, Deutsche Bank Nürnberg, BIZ 700 702 12, Kvits. 0176 148 140, BIC DEUTMMFF, IBAN DE34 760 120 079 148 00, Geschäftsbank Nürnberg, BIZ 0178334, Kvits. 134 476 660, Anregeramt Nürnberg, 0902 181 111, Deutsches Taxidienst 010 944 02 28 900 999999

Task	Expenses in man-days
[Task]	XX,XX
Total Expenses	XX,XX

Total expenses add up to [XX,XX] man-days. We charge
 € XXX,- per man-day, so total costs would be € [XX,XXX,-].

[We offer a reduced project price of € XX,XXX,- for all specified tasks.]

Task	Expenses in man-days
[Task]	XX,XX
Total Expenses	XX,XX

Total expenses add up to [XX,XX] man-days. We charge
 € XXX,- per man-day, so total costs would be € [XX,XXX,-].

[We offer a reduced project price of € XX,XXX,- for all specified tasks.]

Abbildung 3.1: Beispiel eines Dokuments mit verringelter Auflösung



Die Forderung, den Inhalt unkenntlich zu machen, wird durch Verkleinerung der Dokumente gewährleistet, wie in Abbildung 3.1 demonstriert. Der rot umrandete Ausschnitt des Dokuments wird jeweils nach der Verkleinerung auf 150px, 200px und 300px dargestellt. Ziel ist die Verwendung der kleinsten Auflösung, um ein möglichst hohes Maß an Datenschutz zu bieten. Die verkleinerte Darstellung zeigt die Invarianz der Struktur und des Layouts eines Dokuments gegen Skalierung. Diese Darstellung erlaubt die Unterscheidung der Dokumente, da Strukturelemente wie Textblöcke, Trennelemente oder Tabellen noch erkennbar sind, während inhaltlich keine Analyse der Dokumente mehr möglich ist.

Aufteilung der Stichproben

Um bewerten zu können, wie gut ein Klassifikator ein gegebenes Klassifikationsproblem löst, muss dieser nach der Trainingsphase mit unbekannten Daten getestet werden. Die Stichprobe wird dazu jeweils in eine Trainingsstichprobe (80%), eine Validierungsstichprobe (10%) und eine Teststichprobe (10%) aufgeteilt. Die einzelnen Muster werden randomisiert den Teilstichproben zugeordnet. Trainingsstichprobe und Validierungsstichprobe werden während der Trainingsphase verwendet, um die Merkmale zu bestimmen und den Klassifikator zu optimieren.

Die Teststichprobe dient dazu, den Klassifikator nach dem Lernvorgang zu evaluieren. Sie wird vor dem Lernprozess abgespalten und darf nicht zum Trainieren des Klassifikators eingesetzt werden, wie schematisch in Abbildung 3.2 gezeigt. So kann sichergestellt werden, dass die Ergebnisse des Tests nicht auf auswendig gelernten Merkmalen der Stichprobe während des Trainings beruhen.

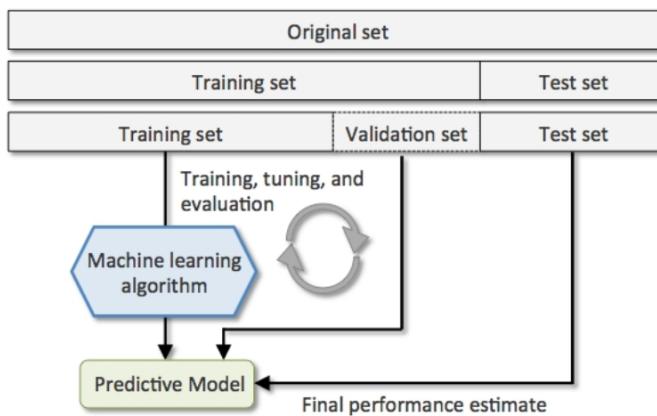


Abbildung 3.2: Aufteilung in Trainings-, Validierungs-, Teststichprobe [RM18]



Diese Aufteilung hat den Nachteil, dass die Muster der Teststichprobe dem Klassifikator vorenthalten werden und darin enthaltene Merkmale nicht erlernt werden können [RM18]. Die *k-fache-Kreuzvalidierung* bietet ein Verfahren, um dieses Problem zu umgehen. Denn die Muster der Trainings- und Validierungsstichprobe werden hierbei während des Trainings variiert [RM18]. Das Verfahren verlängert die Dauer der Trainingsphase deutlich, weshalb es für diese Arbeit nicht verwendet werden kann.

3.2 Stichprobe aus Originaldokumenten

Diese Stichprobe stellt die Daten dar welche im späteren produktiven Einsatz vom Klassifikator erkannt werden sollen. Sie besteht aus reellen Dokumenten des Unternehmens und muss zum Teil digitalisiert werden. Zum Teil liegen die Dokumente bereits in elektronischer Form vor und können aus dem ERP-System der Firma mediendesign AG exportiert werden.

Klasse	Anzahl
Ausgangsrechnung (AR)	1141
Eingangsrechnung (ER)	1438
Leistungsnachweis-md (LN_md)	250
Leistungsnachweis-sy (LN_sy)	425
Leistungsbeschreibung (LB)	217
Angebotsschreiben (AG)	410

Tabelle 3.1: Anzahl Originaldokumente

In der Tabelle 3.1 ist die Anzahl der einzelnen Dokumente je Klasse notiert. Es ist zu erkennen, dass die Anzahl der Muster der einzelnen Klassen sehr unterschiedlich ist. Die Klassen *Ausgangsrechnung* und *Eingangsrechnung* sind in deutlich größerer Zahl vorhanden als Muster der Klassen *Leistungsbeschreibung* oder *Leistungsnachweis-md*.

Dieses Ungleichgewicht in der Verteilung der Daten kann ein Problem darstellen. Der Klassifikator wird aufgrund der a-priori-Wahrscheinlichkeit der Daten implizit auf die Vorhersage der überrepräsentierten Daten optimiert [RM18]. Dadurch kann selbst bei guten Trainings- und Testergebnissen nicht bewertet werden, wie gut der Klassifikator gelernt hat, die einzelnen Klassen zu abstrahieren.



Um dem entgegen zu wirken, stehen drei Möglichkeiten zur Verfügung. Erstens können die überrepräsentierten Daten reduziert werden, um eine ausgewogene Verteilung zu erreichen [RM18]. Zweitens kann versucht werden, die Zahl der unterrepräsentierten Daten künstlich zu erhöhen, indem Methoden geschaffen werden, um neue Muster einer Klasse zu erstellen. Die dritte Möglichkeit besteht darin, während des Trainings eines Modells die unterrepräsentierten Klasse stärker zu gewichten.

Abbildung 3.3: Original Trainingsmuster v.l.n.r Leistungsbeschreibung, Angebotsschreiben, Leistungsnachweis-md, Ausgangsrechnung, Leistungsnachweis-sy, Eingangsrechnung

Die Abbildung 3.3 zeigt je ein Dokument der zu unterscheidenden Klassen. Darin ist eine große Ähnlichkeit in den Klassen *Leistungsbeschreibung*, *Angebotsschreiben* und *Ausgangsrechnung* zu erkennen, die durch das Unternehmens-Erscheinungsbild (engl. Corporate Design) gegeben ist.



3.3 Stichprobe aus RVL-CDIP Datensatz

Der RVL-CDIP Datensatz wurde von *Harley, Ufkes und Derpanis* im Rahmen ihrer Arbeit „*Evaluation of deep convolutional nets for document image classification and retrieval*“ [HUD15, S. 991-995] erstellt. In dem Datensatz sind 400.000 Dokumente aufgeteilt in 16 Klassen enthalten. Die Information über die Klassenzugehörigkeit der einzelnen Dateien ist für den gesamten Datensatz nur textuell vorhanden. Für eine einfachere Wiederverwendung müssen die Dokumente noch in eine entsprechende Ordnerstruktur sortiert werden.

Aus diesem Datensatz werden sechs Klassen (siehe Abbildung 3.4) mit je 6.000 Beispiel-Dokumenten verwendet, da sie die größten Ähnlichkeiten zu den Originaldokumenten aus Abschnitt 3.2 haben. Eine Verwendung aller 400.000 Dateien ist sehr rechen- und zeitintensiv und im Rahmen dieser Arbeit, mit mehreren Ansätzen für Klassifikatoren, nicht umsetzbar. Die insgesamt 36.000 Dokumente werden im Verhältnis 80%-10%-10% in Trainings-, Validierungs- und Teststichprobe aufgeteilt.

Abbildung 3.4: RVL-CDIP Dokumente v.l.n.r budget, form, handwritten, invoice, letter, specification

Wie in Abschnitt 2.1.2 erwähnt ist es wichtig, dass die Merkmale einer Klasse einen möglichst kompakten Merkmalsraum bilden, um dem Klassifikator eine Generalisierung der Klassen zu erlauben. Kritisch an diesem Datensatz ist unter anderem die große Varianz der Daten innerhalb der Klassen, wie in Abbildung 3.5 gezeigt.



Abbildung 3.5: Varianz innerhalb der Klasse “budget“

Ebenso kritisch ist die Ähnlichkeit von Mustern unterschiedlicher Klassen, wie die Beispiele der Klassen "budget" und "specification" in Abbildung 3.6 zeigen.

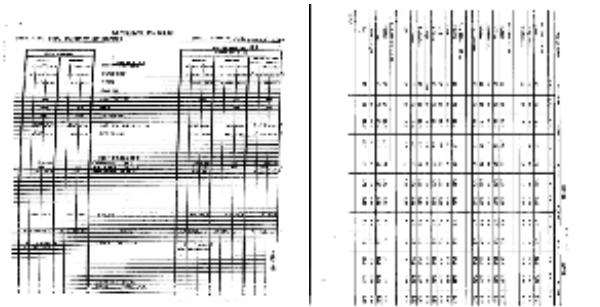


Abbildung 3.6: Ähnlichkeit von Mustern der Klassen “budget” und “specification”

Die beiden beschriebenen Probleme sind darauf zurückzuführen, dass die Dokumente über einen längeren Zeitraum hinweg händisch von verschiedenen Personen annotiert wurden. Weiter handelt es sich um größtenteils mehrseitige Dokumente, weshalb Elemente, wie Tabellen, in Mustern vieler Klassen auftreten. Da der Datensatz nur dafür verwendet werden soll, um dem Klassifikator generelle, in Dokumenten häufig auftretende, Merkmale erlernen zu lassen, werden diese Einschränkungen akzeptiert.



4 Stand der Technik

4.1 Künstliche neuronale Netze

Dieser Abschnitt behandelt zunächst die Grundlagen von künstlichen neuronalen Netzen. Da für diese Arbeit ein CNN als Klassifikator verwendet wird, werden diese im Anschluss betrachtet.

Stark vereinfacht bestehen biologische neuronale Netze aus einer Vielzahl hochgradig verknüpfter Neuronen [Kru+15]. Diese akkumulieren empfangene Eingangsreize und geben entsprechende Ausgangsreize weiter. So können Sinneswahrnehmungen ausgewertet und Informationen weitergeleitet werden. Diese Verarbeitung von Reizen läuft parallel im gesamten Nervensystem ab.

Auch ein künstliches neuronales Netz soll aufgrund von Reizen in Form von numerischen Werten eine Ausgabe erzeugen. Wird das Netz, wie in dieser Arbeit, zur Klassifikation von digitalen Bildern verwendet, soll es die zugehörigen Klassenwahrscheinlichkeiten für ein betrachtetes Muster ausgeben.

4.1.1 Künstliche Neuronen

In einem künstlichen neuronalen Netz wird das Konzept der biologischen Neuronen nachempfunden, indem der Ausgabewert in Abhängigkeit der Höhe eines Eingangswertes gesetzt wird. Ein künstliches Neuron ist eine aus n Eingängen x_1, \dots, x_n und einem Ausgang z bestehende Einheit. Jeder Eingang x_j hat eine bestimmte Gewichtung w_{ij} . Aus allen Eingangswerten und den korrespondierenden Gewichten wird eine Summe gebildet [Nie17].

$$z = \sum_{j=1}^n x_j w_{ij} \quad (4.1)$$

Der Ausgangswert, *Aktivitätsniveau* genannt, des Neurons wird gebildet, indem man die Summe der Eingangsgrößen über eine *Aktivierungsfunktion* θ leitet. Zusätzlich kann das Aktivitätsniveau durch ein *Bias*-Neuron beeinflusst werden. Diese spezielle Art von Neuron besitzt keinen Input und hat ein konstantes Aktivitätsniveau gleich Eins. Durch gezieltes Verändern der Verbindungs-Gewichte eines Bias-Neurons können andere Neuronen entweder gehemmt oder verstärkt werden [Nie17].



So ergibt sich zur Berechnung des Aktivitätsniveaus eines künstlichen Neurons folgende Formel (4.2) [GBC16].

$$\text{output} = \theta(z) = \theta\left(\sum_{j=1}^n x_j w_{ij} - b\right) \quad (4.2)$$

4.1.2 Struktur künstlicher neuronaler Netze

Das einfachste künstliche neuronale Netz ist das Perzeptron [Ros58, S. 386–408]. Es besteht in seiner einfachsten Form lediglich aus einem künstlichen Neuron. Da ein solches einschichtiges Perzeptron nur in der Lage ist, linear separable Probleme zu lösen, müssen ein oder mehrere Zwischenschichten von Neuronen hinzugefügt werden, um beliebige Funktionen approximieren zu können. Dadurch entsteht ein verknüpftes Netzwerk aus Neuronen-Schichten, auch Multilayer Perzeptron (MLP) genannt.

Ein solches Netzwerk besteht aus mindestens zwei Schichten, der Eingangsschicht (*Input-Layer*) und der Ausgangsschicht (*Output-Layer*). Dazwischen können beliebig viele Zwischenschichten (*Hidden-Layer*) liegen. Netze mit mehr als einer Zwischenschicht werden als *tiefe künstliche neuronale Netze* bezeichnet, man spricht in diesem Zusammenhang von *Deep Learning* [RM18].

Die Anzahl der Input-Neuronen entspricht der Anzahl der Komponenten des zugeführten Merkmalsvektors. Im Output-Layer entspricht die Anzahl der Neuronen der Anzahl der Klassen. Die Zahl der Neuronen in den Hidden-Layern kann beliebig gewählt werden. Ihre Anzahl gibt die Breite eines Netzes an. Meist wird die Struktur des Netzes in den Hidden-Layern zunächst verbreitert, indem mehr Neuronen als im Input-Layer verwendet werden. Dadurch wird zunächst die Anzahl der Parameter des Netzes erhöht. In späteren Hidden-Layern wird die Anzahl der Neuronen wieder verringert und dadurch auch die Anzahl der trainierbaren Parameter. Das Netz muss somit auf weniger, aber abstraktere Merkmale zurückgreifen, was eine bessere Generalisierung zur Folge hat.

Bei dem in Abbildung 4.1 dargestellten MLP handelt es sich um ein *Feed-Forward-Netzwerk*. Vom Input-Layer aus werden die Eingangsdaten durch das Netz propagiert. In jeder Schicht werden die Aktivitätsniveaus der Neuronen berechnet, welche wiederum Eingangswerte der folgenden Schicht sind [Nie17]. Der Ausgangswert des Output-Layers bildet die Aussage des Netzes zu einem ihm präsentierten Muster.

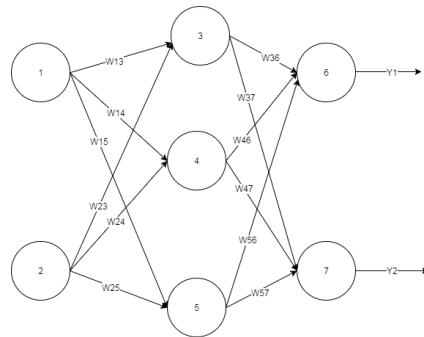


Abbildung 4.1: MLP mit einem Hidden-Layer

Eine alternative Topologie bieten *rekurrente Netze*. Diese Netze können einen gewissen Kontextbezug herstellen, da sie Rückkopplungen in den Verbindungen der Neuronen erlauben [Nie17].

4.1.3 Trainingsphase

Das Lernen in künstlichen neuronalen Netzen findet durch iterative Veränderung der Gewichte und der Bias-Werte statt [Nie17]. Diese werden zu Beginn des Trainings zunächst mit zufälligen, sehr kleinen Werten initialisiert [RM18]. Anschließend präsentiert man dem Netz die Trainingsmuster und ändert die Gewichte sowie die Bias-Werte in Abhängigkeit der Ausgabe. Die Gewichte werden während des Trainings iterativ so oft angepasst bis entweder die Genauigkeit oder der Fehler, den das Netz bei der Vorhersage macht, konvergiert.

Beim *überwachten Lernen* wird eine *Straffunktion J* (engl. *loss*) definiert, welche während des Trainings optimiert wird. Sie ist ein Maß dafür, wie sehr die Ausgabe des Netzes von dem bekannten Sollwert eines Trainingsmusters abweicht [Nie17]. Je besser das neuronale Netz den Sollwert approximiert, umso kleiner ist der Wert der Kosten-Funktion. Ziel ist es deshalb, diesen Fehler durch ein Optimierungsverfahren zu minimieren. Ein solches bietet das *Gradientenabstiegsverfahren*.

Der Gradient der Kosten-Funktion ∇J beschreibt deren Steigungsverhalten für die aktuellen Gewichte und Biaswerte [RM18]. Nun kann die Richtung bestimmt werden, in welche die Gewichtskoeffizienten geändert werden müssen, um den Fehler zu verringern. Die Größe der Änderung wird durch die *Lernrate* η bestimmt. Sie kann während des Trainings variiert werden und bestimmt dadurch die Lerngeschwindigkeit.

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, \text{ wobei } \Delta \mathbf{w} = -\eta \nabla J(\mathbf{w}) \quad (4.3)$$



Beim Gradientenabstiegsverfahren werden alle Trainingsmuster durchlaufen und der Fehler summiert. Am Ende einer solchen Epoche werden die Gewichte geändert. Bei großen Datenmengen kann dies sehr ineffizient sein. Deshalb wird zur Optimierung häufig das *stochastische Gradientenabstiegsverfahren* verwendet [RM18]. Hierbei wird der Gradient nicht für alle Muster der Stichprobe auf einmal berechnet. Die Stichprobe wird hier in kleinere Teilmengen (*Batches*) aufgeteilt und der Gradient pro Teilmenge bestimmt. Anstatt die Gewichte nur einmal je Epoche zu verändern, werden sie beim stochastischen Gradientenabstiegsverfahren mehrfach pro Epoche angepasst. Durch die häufigere Aktualisierung konvergiert dieses Verfahren schneller als der normale Gradientenabstieg.

Beim *unüberwachten Lernen* dagegen sind keine Soll-Ausgaben bekannt. Das neuronale Netz verändert die Gewichte hierbei aufgrund von Ähnlichkeiten innerhalb der Trainingsmuster [Kru+15].

Das *bestärkende Lernen* ist ein Lernverfahren, bei dem die Parameter verändert werden, je nachdem, ob ein Verarbeitungsschritt hin zu einem gewünschten Ergebnis oder weg davon geführt hat [Kru+15].

4.1.4 Überanpassung und Unteranpassung

Überanpassung und Unteranpassung resultieren aus mangelnder Generalisierung des Netzes während des Trainings. Als Folge kann ein Netz für neue Muster keine zuverlässigen Aussagen treffen.

Die Überanpassung hat ihre Ursache häufig in zu vielen trainierbaren Parametern eines Modells im Verhältnis zu wenigen Trainingsdaten [RM18]. Überanpassung kann also vermieden werden, indem die Menge an Trainingsmustern vergrößert wird. Alternativ, falls dies nicht oder nur begrenzt möglich ist, kann die Netzkomplexität verringert werden. Eine Methode dazu bietet *Dropout* [Sri+14, S. 1929-1958]. Dabei wird die Anzahl der Parameter eines Modells durch zufälliges Kappen von Verbindungen verringert, wie dies in Abbildung 4.2 dargestellt wird.

Von einer Unteranpassung spricht man dann, wenn ein Modell das gegebene Problem aufgrund einer zu geringen Anzahl an Parametern nicht ausreichend generalisieren kann und daher zu bestimmten Ergebnissen tendiert [RM18]. In diesem Fall muss die Komplexität des Modells erhöht werden.

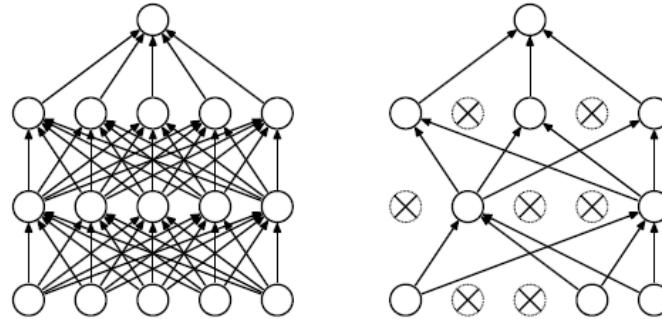


Abbildung 4.2: Links, Modell ohne Dropout. Rechts, mit Dropout [GBC16]

4.2 Convolutional Neural Network

Ein CNN gehört zu der Familie der künstlichen neuronalen Netze und stellt bei der Klassifikation von Bild-Daten den momentan leistungsfähigsten Klassifikator dar. Im Rahmen dieser Arbeit wird ein CNN verwendet, um die Dokumente zu klassifizieren.

4.2.1 Struktur eines Convolutional Neural Network

Die vorderen Schichten eines CNNs dienen zur Extraktion der Merkmale eines Musters. Sie bestehen typischerweise aus einem *Convolutional-Layer*, einer Aktivierungs-Funktion, gefolgt von einem *Pooling-Layer*. Ein Convolutional-Layer besteht in der Regel aus mehreren Filtern. Die letzten Schichten bestehen aus mindestens einem oder mehreren Fully-Connected-Layern, gefolgt von einem Output-Layer, wie in Abschnitt 4.1.2 beschrieben.

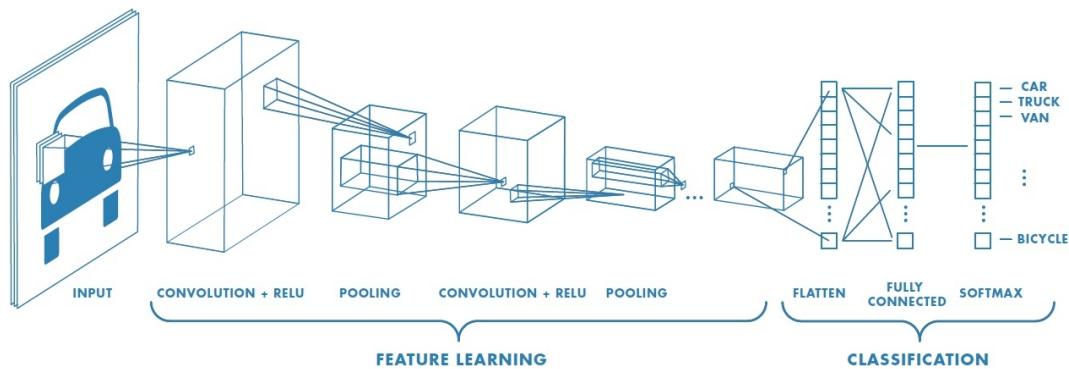


Abbildung 4.3: Topologie eines CNN [Net17]

4.2.2 Faltung

Ein digitales Bild besteht aus einer zweidimensionalen Matrix, deren Elemente je einem Pixel eines Bildes entsprechen und die Farbinformation für jeden Bildpunkt enthalten. Analog zu Techniken der herkömmlichen filterbasierten Bildverarbeitung, werden in einem Convolutional-Layer Filter mit dieser Matrix gefaltet. Dazu wird ein Filter zeilen- und spaltenweise über das Bild verschoben [GBC16]. Bei der Faltung (engl. *convolution*) werden die Koeffizienten eines Filters und die Pixel-Werte eines Bildes paarweise multipliziert und addiert. Daher können die Filter eines Convolutional-Layers als Neuronen betrachtet werden, deren Eingänge die Pixel und deren Gewichte die Koeffizienten der Filtermaske sind. In der Trainings-Phase werden die Koeffizienten der Filter bestimmt.

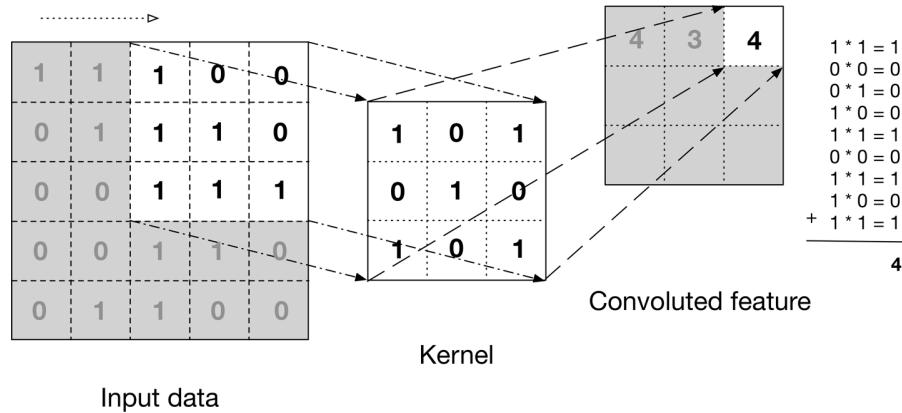


Abbildung 4.4: Prinzip der Faltung einer Matrix mit einem Filterkern [Pat17]

Abbildung 4.4 zeigt das Prinzip der Faltung. Formel 4.4 beschreibt die Faltung einer zweidimensionalen Matrix I mit einem Filter K der selben Dimension [GBC16].

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \quad (4.4)$$

In einem herkömmlichen neuronalen Netz mit m Neuronen in einer Schicht und n Neuronen in der folgenden Schicht ist zur Berechnung des Output-Werts eine Matrixmultiplikation mit $m \times n$ Parametern notwendig [GBC16]. Jedes Neuron einer Schicht nimmt also Einfluss auf alle folgenden. In einem CNN dagegen ist jedes Neuron der folgenden Schicht nur mit einem Teil der Neuronen der vorangegangenen Schicht verknüpft, dem sogenannten *rezeptiven Feld*. Jeder Filter bildet das rezeptive Feld eines Neurons der folgenden Schicht, wie in Abbildung 4.5 dargestellt.

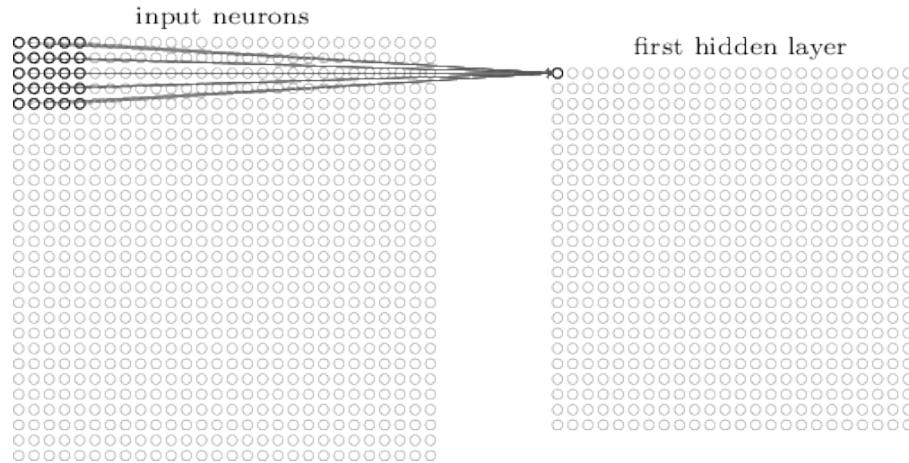


Abbildung 4.5: Rezeptives Feld [Nie17]

Dadurch wird die Anzahl der notwendigen Parameter bei einer Filter-Größe k auf $k \times n$ verringert [GBC16]. Somit muss nicht jeder Eingangswert mit jedem Ausgang verrechnet werden, wodurch sich der Berechnungsaufwand deutlich verringert. Dies ist möglich, da die gesuchten Merkmale in Bezug auf die gesamte Größe eines Bildes vergleichsweise klein sind und deshalb auch die Filter nur wenige Elemente beinhalten müssen. In höheren Schichten vergrößert sich das rezeptive Feld, da Neuronen indirekt durch mehr Neuronen vorhergehender Schichten beeinflusst werden, wie in Abbildung 4.6 veranschaulicht.

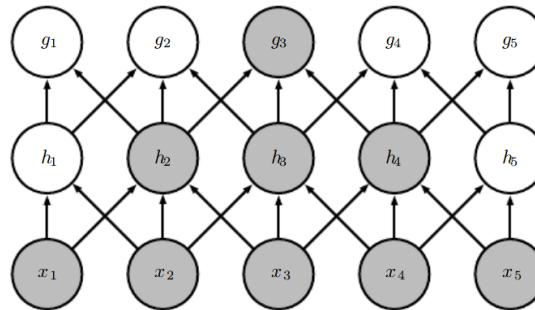


Abbildung 4.6: Vergrößertes rezeptives Feld in tieferen Schichten [GBC16]

Ein weiterer Vorteil, der sich aus der Verwendung von Convolutional-Layern ergibt, ist, dass einmal gelernte Gewichte wiederverwendet werden. Dies ist in dem Begriff *Parameter sharing* oder *Shared Weights* zusammengefasst [GBC16]. Durch das Verschieben des Filters werden die Koeffizienten des Filters mit jedem Bildpunkt verrechnet. Dadurch können Merkmale überall in den Eingangsdaten detektiert werden.



Die Gewichte zum Detektieren eines bestimmten Merkmals müssen daher nur einmal für das gesamte Bild bestimmt werden, während in einem normalen neuronalen Netz für mehrfach auftretende Merkmale die Gewichte jeweils bestimmt werden müssen. Zusätzlich werden CNNs dadurch invariant gegen Translationen, denn Merkmale müssen nicht an bestimmten Positionen auftreten, um detektiert werden zu können.

Als Ergebnis entsteht nach der Faltung eine Matrix, welche als *Feature-Map* bezeichnet wird [GBC16]. Darin ist gespeichert, an welcher Position in den Eingangsdaten ein entsprechender Filter ein Merkmal detektieren konnte. Die Größe der Feature-Map wird durch die Behandlung der Randbereiche eines Bildes (*Padding*) und dem Verschieben des Filters (*Stride*) auf dem Bild beeinflusst.

Üblich ist ein Stride=1, also ein pixelweises Verschieben des Filters. Es können aber auch beliebige andere Werte verwendet werden. Ein größerer Stride führt zu einer stärkeren Verkleinerung der Feature-Map und dadurch zur Verschlechterung der Auflösung zur Detektion eines Merkmals [GBC16].

Bei der Faltung muss der komplette Filter-Kern innerhalb des Bildbereiches liegen. Somit wird bei einer Bildbreite m und einer Filtergröße k die Feature-Map pro Convolutional-Layer um $m - k + 1$ Pixel verringert [GBC16]. Dieser Effekt ist in Abbildung 4.5 zu sehen. Folgen mehrere Convolutional-Layer aufeinander, führt dies zu kleinen und wenig aussagekräftigen Feature-Maps. Man kann diesen Effekt durch Verkleinerung der Filter zwar verlangsamen, allerdings werden die Feature-Maps dennoch bei vielen Schichten deutlich verkleinert. Außerdem werden Merkmale, die direkt in den Randbereichen liegen, nicht berücksichtigt.

Das sogenannte *Zero-Padding* verhindert die Verkleinerung der Feature-Maps. Zudem werden damit auch auf dem Rand liegende Merkmale detektiert [GBC16]. Dabei wird der Rand eines Bildes mit so vielen Nullen (schwarze Punkte in Abbildung 4.7) aufgefüllt, dass bei der Faltung der Mittenzwischenwert des Filters auf der jeweils äußersten Pixelreihe des Bildes liegt (Siehe Abbildung 4.7).



Abbildung 4.7: Zero-Padding [GBC16]



Ein Convolutional-Layer besteht üblicherweise aus einer Reihe von Filtern, wobei jeder unterschiedlich aufgebaut ist und somit auch unterschiedliche Merkmale bestimmen kann.

Während in den vorderen Convolutional-Layern grundlegende Merkmale wie Ecken, Kanten, Farb- oder Helligkeitsübergänge detektiert werden, werden diese in tieferen Schichten zu abstrakteren Merkmalen verknüpft.

4.2.3 Aktivierungsfunktion

Ebenso wie in normalen künstlichen neuronalen Netzen wird in einem CNN zur Bestimmung des Ausgangswertes eines Neurons, in diesem Fall eines Filters, eine Aktivierungsfunktion verwendet.

Meist wird die *logistische Regression*, wie die *Sigmoid*-Funktion (4.5) oder die *Softmax*-Funktion (4.6) im Output-Layer verwendet [Nie17]. Diese Funktionen sind effizient differenzierbar was die Verwendung des *Backpropagation-Algorithmus* erlaubt.

$$\theta(z) = \text{sigmoid}(z) = \frac{1}{1 + e^{-z}} \quad (4.5)$$

$$\theta(z) = \text{softmax}(z)_i = \frac{\exp z_i}{\sum_j \exp z_i} \quad (4.6)$$

Bei $z \rightarrow \infty$ geht $\theta(z) \rightarrow 1$ und bei $z \rightarrow -\infty$ gegen $\theta(z) \rightarrow 0$ [RM18]. Es können also damit große Werte auf den Bereich von $0 \dots 1$ abgebildet werden. Die Sigmoid-Funktion wird bei der binären Klassifikation verwendet. Ihre Ausgabe kann als bedingte Wahrscheinlichkeit $\theta(z) = P(y = 1 | \mathbf{x}, \mathbf{w})$ interpretiert werden. Für diese gilt, dass bei den beobachteten Merkmalen \mathbf{x} und den Gewichtungen \mathbf{w} das aktuell betrachtete Muster zur Klasse 1 gehört. Die Softmax-Funktion kann bei Klassifikations-Aufgaben mit mehreren Klassen verwendet werden, da sie für jede Klasse eine Wahrscheinlichkeit liefert.

In den anderen Schichten wird die *Rectified Linear Unit (ReLU)* als Aktivierungsfunktion eingesetzt.

$$\theta(z) = \max(z, 0) \quad (4.7)$$

Die ReLU-Funktion verhält sich wie eine Schwellwert-Funktion und kann performant berechnet werden. Sie begrenzt das Aktivitätslevel eines Neurons für negative Eingangswerte auf Null und hat im restlichen Verlauf einen konstanten Gradienten. Dies ermöglicht ebenso die Verwendung des Backpropagation-Algorithmus, allerdings mit dem Vorteil, dass die konstante Steigung der Funktion nicht zum *Vanishing-Gradient-Problem* [GBC16] führt. Dieses resultiert aus der Begrenzung des Aktivitätslevels der Sigmoid- und der Softmax-Funktion.

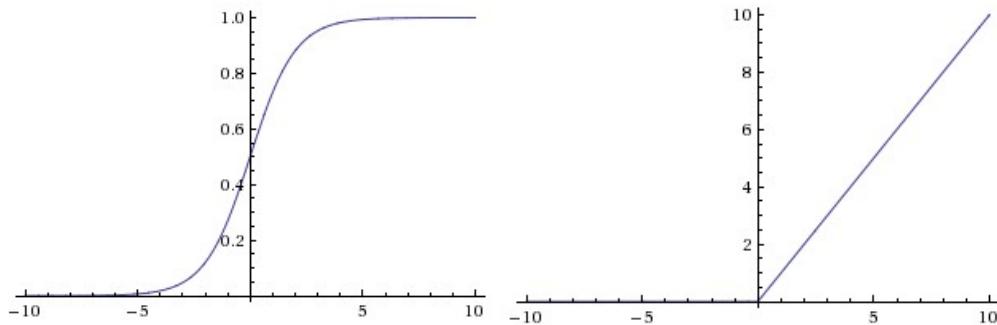


Abbildung 4.8: Aktivierungsfunktion Sigmoid, ReLU [Vis17]

4.2.4 Pooling

Auf die meisten Convolutional-Layer folgt ein *Pooling-Layer*. Ein Pooling-Layer hat eine feste Größe und wird ebenso wie die Filter in den Convolutional-Layer mit einem vorgegebenen Stride über die Eingangsdaten verschoben [GBC16]. Die Pooling-Operation fasst die Eingangswerte ihres rezeptiven Feldes zusammen. Dies kann auf unterschiedliche Weise geschehen und soll hier anhand des häufig verwendeten *Max-Poolings* erläutert werden. Abbildung 4.9 zeigt schematisch das Vorgehen beim Max-Pooling.

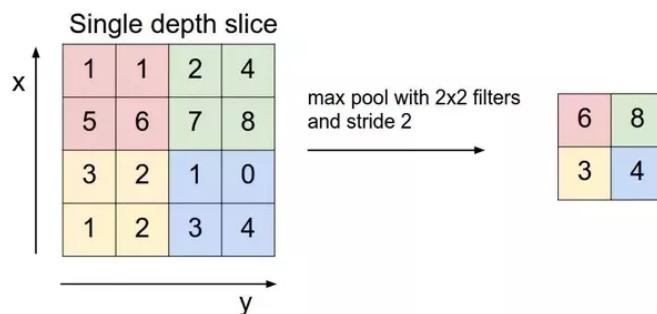


Abbildung 4.9: Beispiel Max-Pooling mit stride=2 [Vis17]



Das Max-Pooling fungiert als Maximalwert-Filter. Aus dem Receptive-Field des Pooling-Layers wird der höchste Wert zur Weiterverarbeitung gespeichert. Die restlichen Werte werden verworfen. In der Feature-Map eines Pooling-Layers werden also nur die stärksten Merkmalswerte notiert, wobei deren exakte vorherige Lokalisierung in den Daten verloren geht.

Die Anwendung von Pooling-Operationen hat mehrere Vorteile. Zunächst reduziert sich die Anzahl der benötigten Parameter deutlich. In der Abbildung 4.9 wird durch Pooling die Anzahl der Parameter um 75% verringert. Dies führt zu einer Steigerung der Verarbeitungsgeschwindigkeit bei gleichzeitig verringertem Speicherbedarf [GBC16].

Weiter wird durch die Pooling-Operationen ein CNN annähernd invariant gegenüber Translationen [GBC16], da für die Klassifikation das Vorhandensein eines Merkmals wichtiger ist als dessen genaue Lokalisierung.

Zudem beugt die Konzentration auf eine geringere Anzahl an Parametern einer Überanpassung vor [GBC16]. Durch die Verringerung der Anzahl der Parameter des Netzes wird die Wahrscheinlichkeit für ein Auswendiglernen des Netzes reduziert.

4.2.5 Batch Normalization

Mit der internen Kovarianzverschiebung (engl. *Internal Covariance Shift*) wird die Veränderung der Parameter einer Schicht in Abhängigkeit der Aktivitätslevel der vorangegangenen Schicht bezeichnet [IS15]. Je größer die Änderung einer Schicht ist, umso stärker ist die Veränderung der folgenden. *Batch-Normalization* ist ein Mittel, den Betrag der internen Kovarianzverschiebung zu begrenzen.

Beim Training eines Netzes werden für jedes Batch der Trainingsdaten \mathcal{B} , Durchschnitt μ und Varianz σ für jeden Merkmalswert $x_{i \dots m}$ berechnet.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad (4.8)$$

$$\sigma_{\mathcal{B}} \leftarrow \sqrt{\frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2} \quad (4.9)$$



Jeder Eingangswert wird anschließend mit diesen beiden Werten normalisiert, wie in folgender Formel (4.10) veranschaulicht wird.

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (4.10)$$

Batch-Normalization verringert die Veränderung der Parameter einer Schicht, die durch eine Kovarianzverschiebung in früheren Schichten entstehen kann. So wird der überproportionale Einfluss, den sehr große Merkmalswerte auf die Gewichtsanpassung in der Trainingsphase haben, reduziert. Dadurch können höhere Lernraten eingesetzt und das Training beschleunigt werden [Fir17].

Als positiver Nebeneffekt wird die Überanpassung reduziert, da bei der Normalisierung in jeder Schicht ein kleiner Anteil an Rauschen entsteht [Fir17].

4.3 Aktuelle Ansätze zur Klassifikation von Dokumenten

Nachdem die grundlegenden Techniken künstlicher neuronaler Netze erläutert wurden, werden in diesem Abschnitt Veröffentlichungen zu dem Thema „Analyse und Klassifikation von digitalen Dokumenten“ behandelt, die zur Klassifikation ein CNN einsetzen.

Die Architekturen der aktuell leistungsfähigsten CNNs zur Klassifikation von Bildern wurden ausgelegt, um möglichst gute Ergebnisse auf „natürlichen Bildern“ zu erzielen [TM17]. Diese unterscheiden sich jedoch stark von Bildern digitaler Dokumente. Deshalb stellen *Tensmeyer und Martinez* in ihrer Arbeit „*Analysis of Convolutional Neural Networks for Document Image Classification*“ die Frage nach der Eignung dieser Netze zur Klassifikation von Dokumenten und beleuchten, welche Faktoren bei der Auslegung wichtig sind. Für ihre Experimente werden Abwandlungen der *AlexNet*-Architektur mit jeweils zufällig initialisierten Gewichten eingesetzt. Dazu verwenden sie, wie in dieser Arbeit, den RVL-CDIP-Datensatz [TM17].

Die Autoren konvertieren die Bilder in unterschiedliche Farträume und untersuchen, wie sich diese Repräsentationen auf die Ergebnisse auswirken. Die besten Ergebnisse erzielen sie mit der Kombination von RGB-Bildern und dem SURF-Algorithmus. Dagegen werden die schlechtesten Ergebnisse mit Binär-Bildern erzielt. Weiter erörtern sie, welche Schritte zur Vergrößerung der Datenmenge für diesen Anwendungszweck sinnvoll sind. Dabei kommen sie zu dem Ergebnis, dass die Scherung dafür am besten



geeignet ist, da diese die horizontale und vertikale Struktur des Dokumentes erhält. Sie kommen weiter zu dem Schluss, dass die Scherung ein CNN robuster gegenüber Veränderungen der Seitenverhältnisse eines Dokumentes macht. Aufgrund dieses Ergebnisses wird die Scherung auch als Vorverarbeitungsschritt auf Teile der Daten dieser Arbeit angewendet.

Um den Einfluss der Netz-Architektur zu testen, variieren die Autoren systematisch die Tiefe des Netzes sowie die Breite einzelner Schichten. Sie beobachten dabei, dass die Vorhersage-Genauigkeit mit zunehmender Netz-Tiefe abnimmt. Je weniger Daten für das Training verwendet werden, umso drastischer ist diese Abnahme. Ihren Ergebnissen nach führt auch eine Vergrößerung der Datenmenge nicht automatisch zu besseren Ergebnissen. So stellen sie bei einer Vergrößerung der Datenmenge um 50% lediglich einen Performance-Gewinn von 1-2% fest. Für eine geringere Menge an Trainingsdaten empfehlen die Autoren eine geringere Breite der einzelnen Schichten, wobei Verschmälern der Convolutional-Layer die Performance negativer beeinflusst, als das Verschmälern der Fully-Connected-Layer. Diese Hinweise zur Auslegung der Architektur werden für den unter 2.3.3 vorgestellten zweiten Lösungsansatz berücksichtigt.

In der Arbeit „*Convolutional Neural Networks for Document Image Classification*“ verfolgen Kang u. a. den Ansatz, Dokumente anhand der Struktur zu klassifizieren [Kan+14, S. 3168-3172]. Sie verwenden dazu ebenfalls ein CNN und stellen dessen Architektur sowie die damit erzielten Ergebnisse vor. Ihr Netz besteht aus zwei Convolutional-Layern mit einer ReLU-Funktion als Aktivierung, gefolgt von jeweils einem Max-Pooling-Layer. Die letzten Schichten vor der Ausgabeschicht bilden zwei Fully-Connected-Layer. Der Output-Layer verwendet als einzige Schicht eine Softmax-Aktivierungsfunktion. Zum Training verwenden die Autoren Teile des *TOBACCO*-Sets [Lew+06] sowie Teile des *NIST*-Datensets [Gro95].

Das CNN akzeptiert Bilder der Größe 150×150 Pixel. Diese Größe ist deshalb so gewählt, da hierbei die meisten Textinhalte nicht mehr erkennbar sind, die Struktur des Dokuments aber erhalten bleibt. Obwohl sehr wenige Muster zum Training verwendet werden, können die Autoren zeigen, dass ihr Ansatz bessere Ergebnisse erzielt als Klassifikatoren, welche nicht auf CNNs basieren. Aufgrund der Ergebnisse von [Kan+14] wird als Größe eines Eingangsbildes in dieser Arbeit ebenfalls 150×150 Pixel gewählt. Die von ihnen vorgeschlagene Architektur wird bei der Umsetzung des zweiten Lösungsansatzes (vgl. Abschnitt 2.3.3) getestet.



Die Autoren *Afzal u. a.* in „*Deepdocclassifier: Document classification with deep Convolutional Neural Network*“ [Afz+15, S. 1111-1115] verfolgen ebenfalls einen Ansatz mit wenigen Trainingsmustern. Anders als *Kang u. a.* verwenden sie ein vortrainiertes AlexNet-Modell. Dieses Modell hat fünf Convolutional-Layer, jeder gefolgt von einem Max-Pooling-Layer und zwei Fully-Connected-Layern. Als Aktivierungsfunktion wird in den Hidden-Layern eine ReLU-Funktion, im Output-Layer eine Softmax-Funktion eingesetzt. Ursprünglich wurden Gewichte des Modells auf RGB-Bildern trainiert. Dabei wurden spezifische Features in den einzelnen Farbkanälen gelernt. Die Autoren nutzen dies, indem sie die Bilder in Grauwert-Bilder konvertieren und diese dem Netz zuführen. Jeder der drei Farbkanäle kann dadurch unterschiedliche Merkmale in einem Bild detektieren. Insgesamt führt dies zu einer größeren Zahl lernbarer Merkmale und somit zu einer verbesserten Vorhersage-Genauigkeit. Ein ähnlicher Ansatz, bereits ein vortrainiertes Modell zu verwenden, wird im ersten Lösungsansatz in Abschnitt 2.3.2 verfolgt. Anders als in der Arbeit von *Afzal u. a.* werden jedoch RGB-Bilder verwendet, da diese insgesamt ein besseres Ergebnis erwarten lassen.



5 Vewendete Frameworks

5.1 Die Programmiersprache Python

Python ist eine Multiparadigmensprache, mit welcher Objektorientierung, funktionale sowie aspektorientierte Programmierung umgesetzt werden kann [Tut18]. Da Python Programme interpretiert werden, findet eine dynamische Typisierung zur Laufzeit statt. Die Programme sind durch Blöcke strukturiert, welche mit Einrückungen realisiert werden, anstatt wie bei vielen anderen Sprachen mit Klammerstrukturen. Methoden und Variablen sind in Python selbst Objekte. Dadurch ist es möglich, Funktionen als Parameter zu übergeben oder einer Variablen als Wert eine Funktion zuzuweisen.

Im Umfeld des maschinellen Lernens hat sich Python als eine der meist eingesetzten Programmiersprachen etabliert. Dies liegt zum einen an der einfachen Syntax, die ein schnelles Lernen dieser Sprache ermöglicht, als auch an der Vielzahl von Bibliotheken, die für diese Sprache entwickelt wurden.

NumPy und *SciPy* sind zwei Erweiterungen zur Python Standardbibliothek, die eine effiziente Verwendung von mehrdimensionalen Arrays in Python erlauben[vCV11, S. 22–30]. *Scikit-learn* ist eine mächtige Bibliothek mit Implementierungen verschiedener maschineller Lernalgorithmen [Ped+11, S. 2825–2830]. Die Bibliothek bietet zudem noch Methoden zur Vorbereitung der Trainingsdaten sowie zur Evaluation eines eigenen Modells.

Zur Entwicklung der Graphical User Interface (GUI) wird in dieser Arbeit *PyQt* verwendet. PyQt sind Bindings, die den Einsatz des *Qt*-Frameworks der Firma „The Qt Company“ in Python möglich machen [PyQ18]. Qt selbst ist eine Bibliothek zur plattformübergreifenden Entwicklung von Bedienoberflächen.

5.2 Machine Learning Framework

Für die Implementierung des CNN in Python wird *Tensorflow* als Framework zusammen mit *Keras* verwendet. Beide Werkzeuge werden in diesem Abschnitt vorgestellt.



5.2.1 TensorFlow

TensorFlow ist eine von Google 2015 veröffentlichte Open-Source-Bibliothek für maschinelles Lernen [Zac16]. Die Bibliothek unterstützt sowohl Python als auch C++. Modelle werden in TensorFlow als Graphen repräsentiert. Die einzelnen Operationen des Modells bilden die Knoten des Graphen. Die Kanten des Graphen sind die Eingangs- beziehungsweise die Ausgangsdaten der Knoten. Die Daten, auf welchen die Operationen ausgeführt werden, sind in TensorFlow durch Tensoren repräsentiert. Jeder Knoten erhält als Eingangsdatum einen Tensor und produziert als Ausgangsdatum ebenfalls einen Tensor. Der Rang von Eingangs- und Ausgangstensor kann je nach ausgeführter Operation variieren.

TensorFlow trennt die Definition eines Modells von der Ausführung [Zac16]. Dazu muss, nach der Erstellung eines Graphen, ein Session-Objekt erzeugt werden. Dieses Objekt kapselt die Umgebung, in welcher der Graph ausgeführt wird. Das Session-Objekt stellt sicher, dass zur Ausführung eines Knotens alle benötigten Daten vorliegen. Die Berechnungen einzelner Teilgraphen können dadurch parallel in unterschiedlichen Prozessen auf mehreren Prozessoren ausgeführt werden.

```

1 import tensorflow as tf
2
3 x = tf.constant(3, dtype=tf.int8)
4 y = tf.constant(2, dtype=tf.int8)
5
6 op1 = tf.add(x, y)
7 op2 = tf.mul(x, y)
8 op3 = tf.pow(op2, op1)
9
10 with tf.Session() as sess:
11     op3 = sess.run(op3)

```

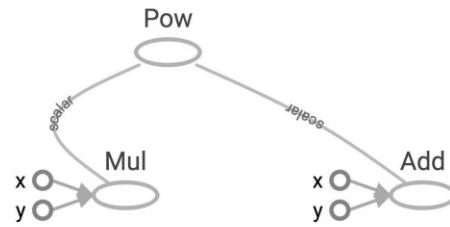


Abbildung 5.1: TensorFlow Programm mit Graph

Abbildung 5.1 zeigt die Definition eines Graphen in TensorFlow. In diesem werden zunächst drei Knoten definiert. Anschließend wird der Graph mit dem Session-Objekt ausgeführt.

Graphen großer Modelle können sehr komplex werden und sind aus dem Programmcode schlecht zu erschließen. Ein Werkzeug zur Visualisierung solcher Graphen bietet TensorFlow mit *TensorBoard*. Dieses Werkzeug bietet zudem auch die Möglichkeit, den Lernfortschritt, den Verlauf der Fehlerrate sowie eigene erstellte Parameter während des Trainings eines Modells darzustellen.



TensorFlow erlaubt die Auslagerung von Berechnungen auf die Prozessoren der Grafikkarte (GPU). Dies ist deutlich effektiver als die Verwendung normaler CPUs und reduziert den zeitlichen Aufwand beim Training eines Modells massiv. Mit der Verwendung einer von Tensorflow unterstützten Grafikkarte wird die Dauer einer Trainings-Epoche des InceptionV3 Modells mit den RVL-CDIP-Daten von über 20 Minuten auf sechs Minuten reduziert.

5.2.2 Keras

Zur Erstellung der Modelle in dieser Arbeit wird *Keras* als High-Level-API für TensorFlow verwendet [Cho16]. Keras wurde in Python entwickelt und unterstützt neben TensorFlow noch *Theano* und das von Microsoft entwickelte *Cognitive Toolkit (CNTK)*.

Modelle in Keras werden durch Konkatenieren einzelner Schichten erstellt, wie in 5.1 dargestellt. Jeder Layer, der mit Keras einem Modell hinzugefügt werden kann, nimmt als Eingang einen Tensor entgegen und gibt ebenfalls einen Tensor als Ausgabe weiter. Sowohl die Form des Tensors des Input-Layers als auch die des Output-Layers müssen angegeben werden. Der Input-Tensor entspricht der Größe eines Bildes, in dieser Arbeit $150 \times 150 \times 3$. Die Drei gibt die Anzahl der Farbkanäle eines Bildes an, die 150 stehen für dessen Abmessung in Pixel. Der Output-Tensor entspricht der Anzahl der zu unterscheidenden Klassen. Für die Schichten dazwischen ist dies nicht notwendig, da deren Eingänge auf die vorhergehenden Ausgänge angepasst werden.

```

1 model = Sequential()
2 model.add(Conv2D(64, (3,3), activation='relu', input_shape=(150,150,3)))
3 model.add(MaxPooling2D(pool_size=(2, 2)))
4 model.add(Conv2D(64, (3,3), activation='relu'))
5 model.add(Flatten())
6 model.add(Dense(128, activation='relu'))
7 model.add(Dense(1, activation='sigmoid'))
8
9 model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
10 model.fit(data, labels, epochs=10, batch_size=32)

```

Listing 5.1: Keras Beispiel

Listing 5.1 zeigt das Modell eines CNNs, bestehend aus zwei Convolutional-Layern, einem Max-Pooling-Layer und einem Fully-Connected sowie einem Output-Layer. Die beiden Convolutional-Layer bestehen aus jeweils 64 Filtern mit einer Filterkern-Größe von 3×3 Elementen. Als Aktivierungsfunktionen werden in den vorderen Schichten ReLU und im Output-Layer eine Sigmoid-Funktion verwendet (siehe Abschnitt 4.2.3).



Der Befehl *Flatten()* überführt die Ausgangsdaten des Convolution-Blocks in ein eindimensionales Array (1d-Tensor). Anschließend können diese Werte an die 128 Neuronen des Fully-Connected-Layer übergeben werden.

Nachdem ein Modell in Keras definiert wurde, muss dieses kompiliert werden (siehe Listing 5.1). Dazu muss die *loss*-Funktion sowie ein Optimierungs-Verfahren (*optimizer*) als Parameter vorgegeben werden. In dieser Arbeit wird als Optimierungs-Verfahren der in Abschnitt 4.1.3 beschriebene stochastische Gradientenabstieg verwendet. Anschließend kann das Modell durch den Befehl *model.fit()* trainiert werden.

Die *Datagenerator*-API von Keras liefert eine Schnittstelle, um Bilder einzulesen, diese auf eine gewünschte Auflösung zu skalieren sowie eine Vielzahl von Möglichkeiten, die Bilder zu bearbeiten.

Mit *Callbacks* bietet Keras die Möglichkeit, die Lernhistorie zu speichern, die Lernrate während des Trainings zu variieren oder ein Modell mit Gewichtung zu speichern. Callbacks ermöglichen es auch, Schwellwerte vorzugeben, um das Training vorzeitig zu beenden, falls der Lernfortschritt frühzeitig konvergiert.

Zudem bietet Keras auch eine Schnittstelle an das Scikit-learn-Modul. Das erleichtert die Evaluation von Modellen und erlaubt einen schnellen Vergleich verschiedener Modelle.



6 Prototypische Umsetzung

6.1 Modellbildung und Training des Netzes

Wie in Abschnitt 2.3 beschrieben, werden zunächst zwei Strategien verfolgt, um das gestellte Klassifizierungs-Problem möglichst zuverlässig zu lösen. Die Implementierung und das Training dieser beiden Klassifikatoren werden hier beschrieben.

Anschließend werden nur die Modelle und Trainingsverläufe vorgestellt, welche die besten Ergebnisse lieferten. Diese werden mit Dokument-Bildern in den Auflösungen 150px sowie 200px trainiert, um den Einfluss der Auflösungen bewerten zu können.

Beim Training der Modelle werden als Straffunktionen die *binary_crossentropy* sowie die *categorical_crossentropy* verwendet. Dabei handelt es sich um logistische Regression, wie in Abschnitt 4.2.3 bereits vorgestellt [RM18]. Für eine genauere Betrachtung der logistischen Regression als Straffunktion wird auf „*Machine Learning mit Python und Scikit-learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*“ von Raschka und Mirjalili verwiesen.

Für das Einlesen der Bilder kann die Imagedatagenerator-API verwendet werden. Die API bietet eine komfortable, aber unperformante Möglichkeit, die Daten einzulesen. Denn bildbearbeitende Operationen, wie die Skalierung, sind sehr rechen- und daher zeitintensiv. Das Training kann insgesamt beschleunigt werden, wenn diese Operationen vorab durchgeführt werden.

Es wird deshalb ein Unterprogramm entwickelt, welches vor Trainingsbeginn alle Bilder als vorzeichenloses 8-Bit-Array einliest und nach dem Zufallsprinzip in eine Trainings- und Validierungsstichprobe aufteilt. Bereits zuvor werden 10% der Dokumente gesondert als Teststichprobe vom Rest der Dokumente abgespalten (vgl. Abschnitt 3.1).

Das Unterprogramm nutzt eine Klasse, in welcher Methoden zur Scherung oder Rotation eines Bildes implementiert sind. Beide Operationen werden zufällig auf 10% der Dokumente angewendet. Die Scherung wird aufgrund der positiven Aussagen von Tensmeyer und Martinez in [TM17] ausgewählt. Die Rotation um 180° soll das CNN robuster gegen eine unterschiedliche Ausrichtung der Dokumente machen, die beispielsweise beim Scannen entstehen kann.

Dem Problem der unausgeglichenen Datenmenge wird begegnet, indem unterrepräsentierte Klassen stärker gewichtet werden (vgl. Abschnitt 3.1). Die unterschiedlichen Gewichte der Klassen werden aus den Verhältnissen der Dokumente pro Klasse zur Gesamtzahl aller Dokumente gebildet.

6.1.1 Vortrainierter Klassifikator

Als Basismodell wird eine InceptionV3-Architektur verwendet. Dieses Modell zeichnet sich neben den guten Ergebnissen beim ILSVRC2014 [Rus+15, S. 211–252], durch seine effiziente Verarbeitung aus.

Die Entwickler des Modells konnten durch Verringerung der Parameter und Parallelisierung den Berechnungsaufwand gegenüber anderen vergleichbar großen Modellen reduzieren. Um die Anzahl der Parameter des Netzes zu verringern, werden die Filterkerne der Convolutional-Layer verkleinert. Da dies eine Verschlechterung der Güte zur Folge hat, werden ganze Convolutional-Layer durch kleine CNNs ersetzt [Sze+15]. Diese bezeichnen die Entwickler als *Inception*-Module. So wird beispielsweise ein 5×5 Convolutional-Layer durch zwei 3×3 Convolutional-Layer ersetzt. Diese Vorgehensweise reduziert den Berechnungsaufwand gegenüber dem ursprünglichen 5×5 Filter um 28%, ohne dabei die Genauigkeit des Netzes negativ zu beeinflussen. Jedes Inception-Modul hat dasselbe rezeptive Feld und dieselbe Ausgangsgröße, wie der ersetzte Convolutional-Layer.

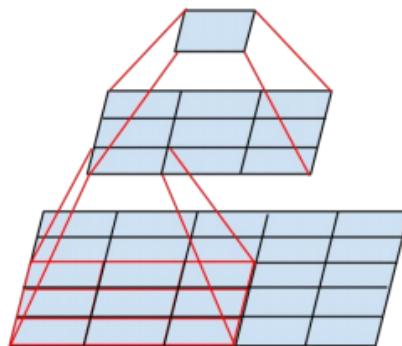


Abbildung 6.1: Ein 5×5 Filter durch zwei 3×3 Filter ersetzt [Sze+15]

Die Inception-Module werden parallel verarbeitet, sodass die Performanz trotz größerer Netztiefe und -breite, verglichen zu den Vorgängern, verbessert wird.

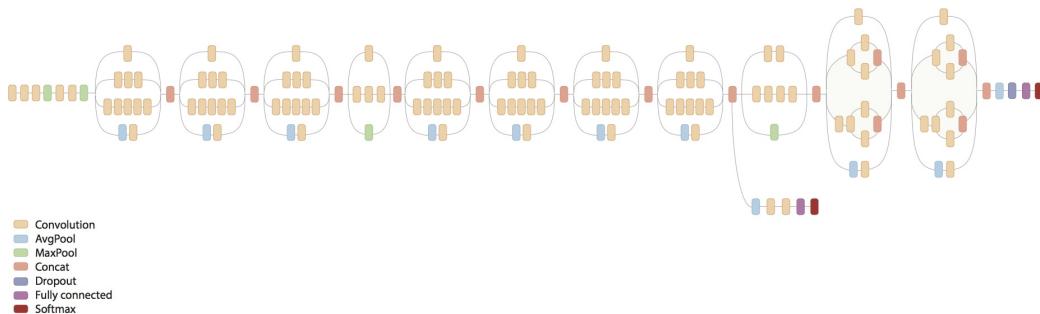


Abbildung 6.2: Architektur InceptionV3 [Ten18]

Mit dem *Applications*-Modul bietet Keras Methoden, eine InceptionV3-Architektur (Abbildung 6.2) samt ImageNet-Gewichten zu laden. Um das Modell für diese Arbeit zu verwenden, werden die letzten beiden Schichten des Modells entfernt. Anstatt dieser, wird ein Fully-Connected-Layer mit 1024 Neuronen sowie ein Output-Layer mit sechs Neuronen für die zu unterscheidenden Dokumenten-Klassen verwendet. Als Aktivierungsfunktion des Output-Layers wird eine Softmax-Funktion (vgl. Abschnitt 4.2.3) verwendet.

Training

Das Modell wird entsprechend der in Abschnitt 2.3.2 beschriebenen Vorgehensweise zweimal trainiert. Zunächst auf den 36.000 Dokumenten des angepassten RVL-CDIP-Datensatzes und anschließend auf den 3.881 Originaldokumenten.

Hierfür wird eine Python-Klasse „Finetuner“ implementiert. Diese bietet Methoden, um Modelle zu laden, anzupassen und erneut zu trainieren, wie das Programmbeispiel in Listing 6.1 zeigt. Die beiden Trainingsdurchläufe können so direkt hintereinander ausgeführt werden, ohne dass ein erneutes Anstoßen durch einen Benutzer erforderlich wird.

Die große Anzahl an lernbaren Parametern des Modells, sowie die große Menge an Trainingsmustern führen zu einer Dauer von 400sec./Epoche. Aus diesem Grund wird im ersten Durchlauf das Training nach 150 Epochen abgebrochen. Die Bias-Werte und Gewichte werden gespeichert, um diese als Startwerte für ein erneutes Training des Modells zu verwenden. Dieses Modell kann damit als Ausgangspunkt für spätere Erweiterungen durch die Firma mediendesign AG genutzt werden.



```

1 finetuner = Finetuner(img_width, img_height, batchSize)
2
3 model = finetuner.prepareModelForBaseTraining(1024,6, 'Inception', 'softmax', optimizer=
    optimizer)
4 model = finetuner.trainTheModelWithGenerator(model, 150, trainGenerator, valGenerator,
    callbacks, stepsPerEpoch, validationSteps)
5 model = finetuner.loadSelfPreTrainedMobileNet('Base-Training_RVL-161-1.09-Lr-0.001.hdf5', ,
    InceptionV3')
6 model = finetuner.prepareModelForFineTuning(model, optimizer, lossMode='
    categorical_crossentropy')
7 model = finetuner.trainTheModelWithGenerator(model, 500, trainGenerator, valGenerator,
    callbacks, stepsPerEpoch, validationSteps)

```

Listing 6.1: Erstellen und zweimaliges Trainieren eines Modells mit der Klasse „Finetuner“

Im zweiten Durchlauf wird das Modell erneut geladen und mit den Bildern der Originaldokumente trainiert. Abbildung 6.3 zeigt den Verlauf der Korrektklassifizierungsrate („accuracy“) sowie des Fehlers („loss“) während des Trainings. Es ist zu erkennen, dass beide Werte bereits nach wenigen Iterationen konvergieren. Die Konvergenz auf einem hohen Wert bei der Korrektklassifizierungsrate, sowie einem niedrigen Wert des Fehlers zeigen, dass der Klassifikator in der Lage ist, die Muster der Validierungsstichprobe zuverlässig zu erkennen.

Das Training wird beendet, sobald in über 50 Epochen keine Verbesserung der Korrektklassifizierungsrate oder des Fehlers festzustellen ist.

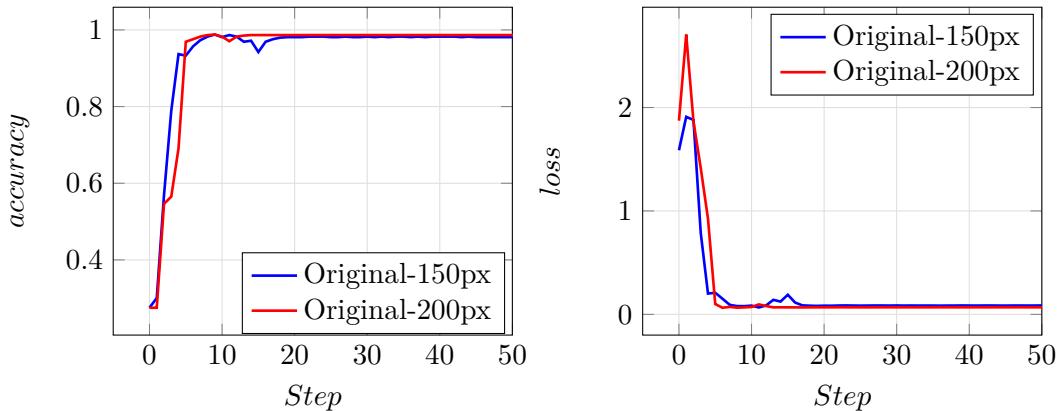


Abbildung 6.3: Trainingsverlauf „Vortrainierter Klassifikator“; Links, Korrektklassifizierungsrate Rechts, Fehlerrate



6.1.2 Mehrstufiger Klassifikator

Dieser Abschnitt wird die Architekturen sowie Trainingsverläufe der drei Modelle des „Mehrstufigen Klassifikators“ (vgl. Abschnitt 2.3.3) beschreiben. Bei der Erstellung der Modelle werden Ergebnisse der Arbeiten aus Abschnitt 4.3 berücksichtigt.

Um effizient mehrere Modelle mit alternativen Architekturen testen zu können, wird eine Python-Klasse entworfen, welche die Keras-Methoden zur Erstellung von Modellen nochmals abstrahiert. Ein Benutzer kann damit einzelne Schichten definieren. Für einen Convolutional-Layer muss die Anzahl und Größe der Filter festgelegt werden. Optional können die Aktivierungsfunktion, die Pooling-Size sowie ein Stride angegeben werden. Wird eine Pooling-Size von Null eingegeben, findet in dieser Schicht kein Pooling statt. Nach jedem Convolutional-Layer wird ein *Batch-Normalization-Layer* eingefügt. Bei Fully-Connected-Layern ist die Anzahl der Neuronen und ebenfalls die Aktivierungsfunktion sowie eine optionale Dropout-Rate anzugeben. Die definierten Schichten werden als Liste konkatiniert und einem Objekt der Klasse „Basemodel“ übergeben. In diesem wird das eigentliche Modell anhand der Vorgaben erstellt und kompiliert, bevor es anschließend trainiert werden kann.

Insgesamt wird der Programmcode so kürzer und lesbarer (siehe Listing 6.2). Weiter ist es möglich, Modelle systematisch zu erstellen und zu testen.

```

1 convlayer1=[64,3,3,'relu',2,1]
2 convlayer2=[64,3,3,'relu',2,1]
3 convlayer3=[32,3,3,'relu',2,1]
4
5 convlayers=[]
6 convlayers.append(convlayer1)
7 convlayers.append(convlayer2)
8 convlayers.append(convlayer3)
9
10 fc1=[512,'relu',.2]
11
12 fullyConnctedLayers=[]
13 fullyConnctedLayers.append(fc1)
14
15 modelCreater= BaseModel()
16 model= modelCreater.createModel(img_width,img_height,1, convlayers,fullyConnctedLayers)
17 model= modelCreater.compile_model(optimizer='adam', loss_mode='binary_crossentropy')
```

Listing 6.2: Erstellen eines Modells mit der Klasse „Basemodel“



Die Hyperparameter, wie Lernrate und Batch-Größe, werden vorab durch eine Rastersuche bestimmt. Dabei werden für jeden Hyperparameter Werte vorgegeben und alle daraus möglichen Kombinationen für wenige Epochen getestet [RM18]. Am effektivsten zeigt sich hier eine initiale Lernrate $\eta = 0.001$, sie kann während des Trainings auf einen minimalen Wert $\eta = 0.00001$ reduziert werden. Als Batch-Größen werden von der Rastersuche 16 Bilder je Batch für den Intern-Extern-Klassifikator und jeweils 32 Bilder für die beiden anderen Klassifikatoren ermittelt.

Zu Beginn wird ein Basismodell, bestehend aus fünf Convolutional-Layern mit jeweils 128 Filtern und gefolgt von zwei Fully-Connected-Layern sowie einem Output-Layer erstellt. Zwischen den einzelnen Convolutional-Layern werden Max-Pooling-Layer gesetzt. Ausgehend von diesem Basismodell entstehen die drei Modelle des Klassifikators, indem jeweils ein Modell zur Klassifikation der entsprechenden Klassen trainiert wird.

Nach jedem Trainingsdurchlauf findet eine Anpassung der Architektur statt und das reduzierte Modell wird erneut trainiert. Dies geschieht iterativ, bei ansonsten identischen Einstellungen, solange bis eine Verschlechterung der Ergebnisse festzustellen ist. Anschließend wird jedes Modell noch einmal mit einer Auflösung von 200×200 -Pixel trainiert. Tabelle 6.1 listet die Architekturen der optimierten Modelle auf. Eine detaillierte Darstellung der Architekturen ist im Anhang A.1 zu finden.

Intern-Extern-Modell	Intern-Modell	Extern-Modell
conv2d_1 (5x5x32)	conv2d_1 (5x5x32)	conv2d_1 (5x5x32)
conv2d_2 (5x5x16)	conv2d_2 (3x3x16)	max_pooling2d_1 (2x2)
max_pooling2d_1 (2x2)	conv2d_3 (3x3x16)	conv2d_3 (3x3x16)
conv2d_3 (3x3x16)	max_pooling2d_1 (2x2)	max_pooling2d_2 (2x2)
conv2d_4 (3x3x16)	dense_1 (256)	dense_1 (128)
conv2d_5 (3x3x16)	dense_2 (128)	dense_2 (64)
max_pooling2d_2 (2x2)	dense_3 (4)	dense_3 (4)
dense_1 (512)		
dense_2 (128)		
dense_3 (1)		

Tabelle 6.1: Architekturen der Modelle des „Mehrstufigen Klassifikators“

Training

Die Schwierigkeit des Klassifikations-Problems nimmt mit abnehmender Auflösung der Bilder zu. Der Klassifikator soll deshalb auf die ungünstigere Auflösung von 150×150 -Pixel optimiert werden, da bei größerer Auflösung von besseren Ergebnissen ausgegangen werden kann.

Auch hier kann beobachtet werden, dass die Modelle jeweils nach wenigen Epochen bereits auf hohem Niveau konvergieren, weshalb jeder Trainingsdurchlauf bereits nach 100 Epochen beendet werden kann.

Die Schwankungen von „accuracy“ und „loss“ beim Training des Intern-Klassifikators, deuten auf eine zu hohe Lernrate zu Beginn hin (blaue Kurve in Abbildung 6.4). Diese Lernrate wird während der Trainingsphase reduziert, wenn der Lernfortschritt stagniert oder große Schwankungen in „accuracy“ und „loss“ auftreten.

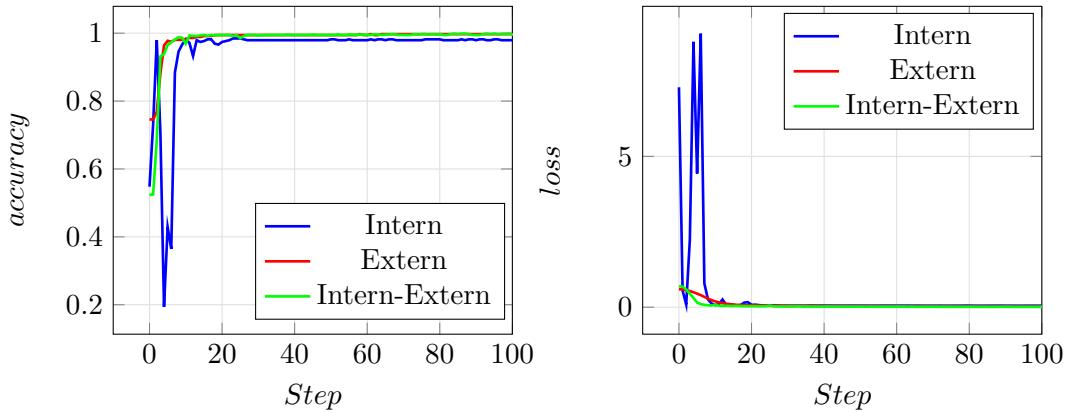


Abbildung 6.4: Trainingsverlauf der drei Modelle des „Mehrstufigen Klassifikators“

Das Modell zur Klassifikation interner Dokumente neigt zur Unteranpassung. Wie aus der niedrigen „accuracy“ und dem hohen „loss“ für Trainings- und Validierungsdaten in Abbildung 6.5 zu erkennen ist [RM18]. Zusätzliche Max-Pooling-Layer nach jedem Convolutional-Layer beheben das Problem. Jedoch unterscheiden sich die Modelle zur Klassifikation interner Dokumente dadurch.

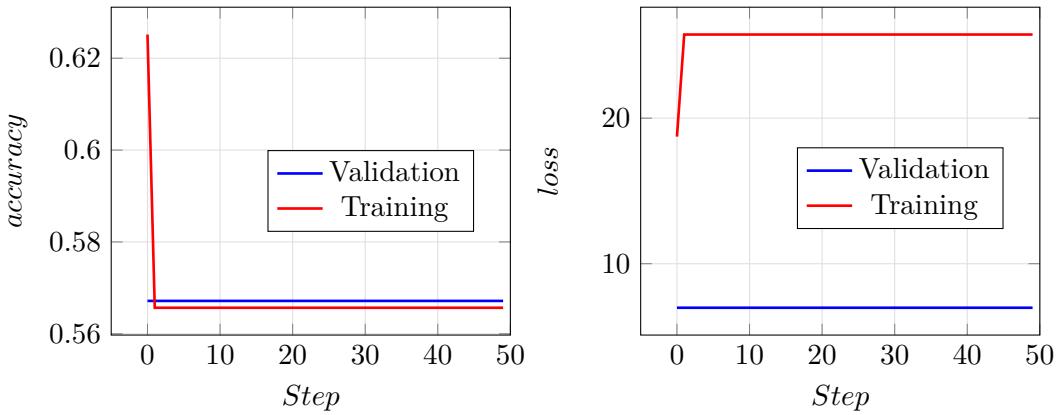


Abbildung 6.5: Unteranpassung des Intern-Klassifikators bei 200px großen Bildern

6.2 Implementierung einer Anwendung zur Klassifikation

6.2.1 Server-Anwendung zur Klassifikation

Der serverseitige Teil der Anwendung besteht aus zwei Teilen, einer REST-Schnittstelle sowie einer Classifier-Klasse, in welcher der eigentliche Klassifikator abstrahiert wird.

REST-Schnittstelle

Die Schnittstelle bietet als Dienst eine Methode an, die von einem Client per HTTP-Post-Request angesprochen werden kann. Darin schickt der Client ein beliebiges Dokument als JSON. Der Inhalt des JSON-Objekts wird in ein Integer-Array konvertiert, um es dem eigentlichen Klassifikator zu übergeben. Dieser antwortet mit einer Klassenaussage, die zurück an den Client übermittelt wird.

```

1 @app.route('/api/v1.0/classify', methods=['POST'])
2 def classify_data():
3     picture = {'data': request.json.get('data', "")}
4     try:
5         picture_as_array = picture['data']
6         document_image = np.asarray(picture_as_array, dtype=np.uint8)
7         data={'prediction': __prediction_to_class(classifier.classify_single_document(
8             document_image))}
8         return jsonify(data), 201
9     except:
10         abort(404)

```

Listing 6.3: REST Schnittstelle



Klassifikator Schnittstelle

Für jeden der beiden Lösungsansätze wird je eine Python-Klasse entworfen. Beide bieten mit einer *classify_single_document*-Methode die gleiche Schnittstellendefinition zur Klassifikation eines Dokumentes. Dadurch kann der Klassifikator in der Anwendung leicht ausgetauscht werden.

Die Modelle müssen dem jeweiligen Klassifikator bei Programmstart im Konstruktor übergeben werden. Das soll es dem Unternehmen ermöglichen, später neue Modelle für andere Dokumentklassen zu erstellen, ohne die Anwendung zur Klassifikation ändern zu müssen.

Bei dem „Mehrstufigen Klassifikator“ wird, wie in Abschnitt 2.3.3 beschrieben, die Vorhersage der Klassen durch die Kombination der drei Modelle gegeben.

Zur eigentlichen Klassifikation eines Dokuments wird für ein Modell die *predict*-API aufgerufen (siehe Listing 6.4). Diese antwortet je nach Aktivierungsfunktion im Output-Layer ein Array mit der vorhergesagten Klassenwahrscheinlichkeit.

```
1 prediction = self.extern_model.predict(document_img)
```

Listing 6.4: Keras Predict Methode

Der höchste Wahrscheinlichkeitswert, die Klasse als Integer-Wert sowie der Klassenname als String werden an den Aufrufer zurückgegeben.

6.2.2 Client-Anwendung

Der clientseitige Teil der Anwendung ist so konzipiert, dass er später in einen Windows-Dienst integriert werden kann, um im Hintergrund einen Ordner auf neu eintreffende Dokumente zu überwachen. Gleichzeitig wird die Schnittstelle verwendet, um zusammen mit einem GUI, einem Benutzer eine komfortable Möglichkeit zu bieten Dateien in Ordner auszuwählen, anzuzeigen und zu klassifizieren.

Nach Öffnen des Programms kann ein Benutzer einen Ordner auswählen, für dessen Inhalt eine Klassifikation durchgeführt werden soll. Der Ordner wird nach elektronischen Dokumenten durchsucht. Der Benutzer hat nun die Möglichkeit, die Dokumente einzeln klassifizieren zu lassen. Dabei wird jedes Dokument sowie die Klassenvorhersage für dieses Dokument angezeigt. Alternativ kann eine automatische Klassifikation aller Dokumente des ausgewählten Ordners angestoßen werden.



Der gesamte Ablauf zur Klassifikation eines einzelnen Dokuments wird durch das Sequenzdiagramm (Abbildung 6.6) veranschaulicht.

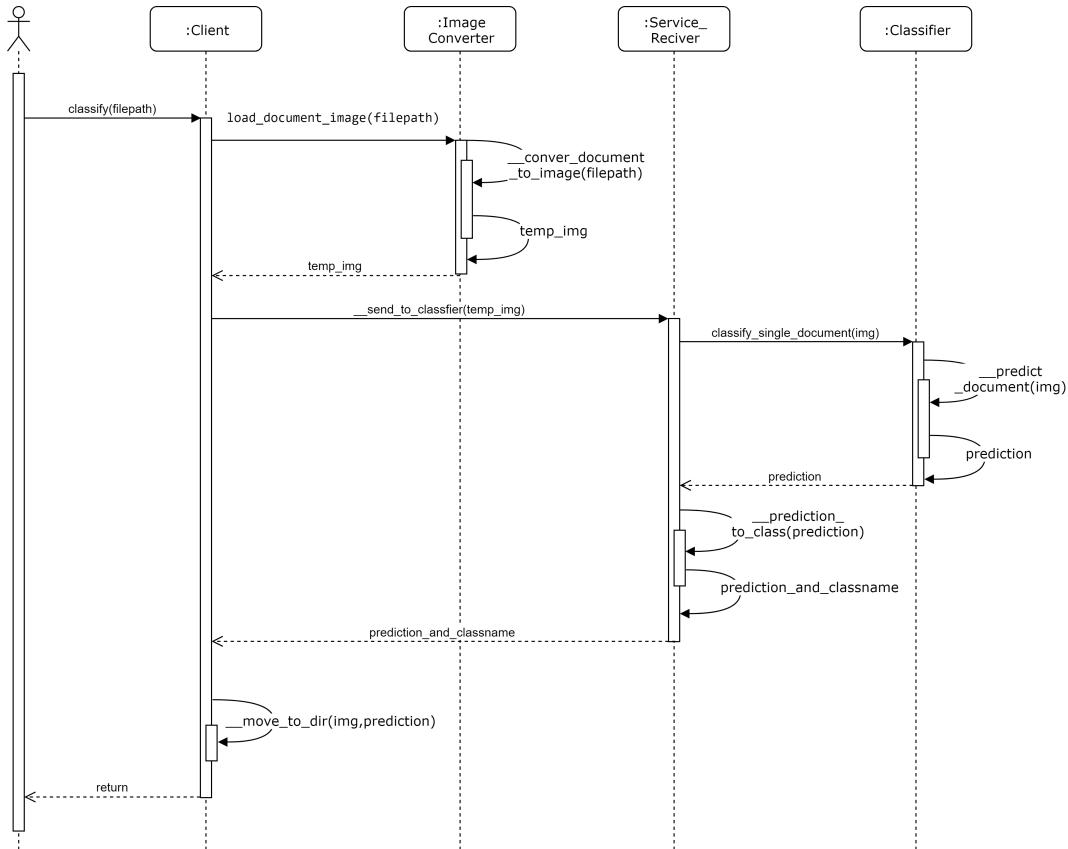


Abbildung 6.6: Sequenzdiagramm zur Klassifikation eines Dokuments

Zur Umwandlung der Dokumente nutzt der Client ein selbst implementiertes Unterprogramm, welches für ein übergebenes Dokument, abhängig von dessen Dateiformat, Methoden bereitstellt, um die erste Seite des Dokuments als PNG zu extrahieren. Das temporäre Bild wird in Form eines vorzeichenlosen 8Bit-Arrays zurück an den Aufrufer gegeben und anschließend gelöscht.

Der Client sendet das Integer-Array als JSON-Objekt per Post-Request an den severseitigen Teil der Anwendung. Die Antwort des Klassifikators wird mit einem Grenzwert verglichen. Ist für ein Dokument eine Klasse mit einer Wahrscheinlichkeit größer diesem Grenzwert vorhergesagt, verschiebt der Client das Dokument in einen entsprechenden Ordner. Andernfalls wird die Datei in ein gesondertes Verzeichnis verschoben, um von einem Bearbeiter händisch klassifiziert zu werden.



7 Evaluation der Klassifikatoren

7.1 Evaluation des neuronalen Netzes

Um zu entscheiden, welcher der beiden vorgestellten Lösungsansätze in der Anwendung verwendet werden soll, müssen diese nun verglichen werden. Eine Bewertung der einzelnen Modelle und Trainingsdurchläufe findet dabei nicht statt.

Zunächst werden Metriken vorgestellt, welche eine Beurteilung der Gesamtergebnisse der beiden Klassifikatoren erlauben.

7.1.1 Kenngrößen zur Evaluation

Eine *Konfusionsmatrix* gibt eine einfache grafische Möglichkeit zur Bewertung eines Klassifikators [RM18]. In dieser quadratischen Matrix wird pro Klasse eine Spalte und Zeile generiert. Die Felder werden mit der Anzahl der Muster, die ein Klassifikator der jeweiligen Klasse zugeordnet hat, belegt. Aus einer Konfusionsmatrix können weitere wichtige Kenngrößen zur Bewertung abgeleitet werden.

		vorhergesagte Klassen	
		Klasse 1	Klasse 2
tatsächliche Klassen	Klasse 1	Richtig Positiv (RP)	Falsch Negativ (FN)
	Klasse 2	Falsch Positiv (FP)	Richtig Negativ (RN)

Abbildung 7.1: Konfusionsmatrix für Zweiklassen-Problem



Die Korrektklassifikationsrate (KKR) ist das Verhältnis der korrekten Klassenvorhersagen zu der Gesamtzahl aller Vorhersagen [RM18].

$$\text{Korrektklassifikationsrate} = \frac{RP + RN}{FP + FN + RP + RN} \quad (7.1)$$

Dementsprechend ist die Falschklassifikationsrate (FKR) der Anteil aller falsch klassifizierten Elemente [RM18].

$$\text{Falschklassifizierungsrate} = 1 - KKR = \frac{FP + FN}{FP + FN + RP + RN} \quad (7.2)$$

Die *Genauigkeit* (engl. *precision*) bildet das Verhältnis aus allen einer Klasse zugeordneten Elementen zu den korrekt dieser Klasse zugeordneten [RM18].

$$\text{Genauigkeit} = \frac{RP}{RP + FP} \quad (7.3)$$

Als *Trefferquote* (engl. *recall*) oder *Richtig-Positiv-Rate* wird das Verhältnis aller Elemente einer Klasse bezeichnet, die korrekt zugeordnet werden [RM18].

$$\text{Trefferquote} = \frac{RP}{FN + RP} \quad (7.4)$$

Genauigkeit und Trefferquote können mit dem *F1-Maß* kombiniert werden. Dieses bildet das gewichtete Mittel der beiden Werte.

$$F1 = 2 \frac{\text{Genauigkeit} \times \text{Trefferquote}}{\text{Genauigkeit} + \text{Trefferquote}} \quad (7.5)$$

Eine Verallgemeinerung für ein Mehrklassenproblem liefert der *Mikro-Durchschnitt*. Dieser kann verwendet werden, wenn die Testdaten ungleich verteilt sind aber jede Klasse gleich gewichtet werden soll [RM18]. Der *Makro-Durchschnitt* ist ein einfacher Durchschnitt der Werte über alle Klassen.

$$Gen_{micro} = \frac{RP_1 + \dots + RP_n}{RP_1 + \dots + RP_n + FP_1 + \dots + FP_n} \quad (7.6)$$

$$Rec_{micro} = \frac{RP_1 + \dots + RP_n}{RP_1 + \dots + RP_n + FN_1 + \dots + FN_n} \quad (7.7)$$



7.1.2 Evaluation der Klassifikatoren

Zur Bewertung werden die Modelle jeweils gegen die Dokumente der Teststichprobe in den Auflösungen 150px und 200px getestet und die Ergebnisse gegenübergestellt. Die Teststichprobe wird vergrößert, indem 10% der Dokumente kopiert und um 180° rotiert oder um 25° geschert werden.

Es wird jeweils eine Konfusionsmatrix erstellt, um die Vorhersagen der Klassifikatoren für die jeweiligen Klassen zu visualisieren. Außerdem werden Genauigkeit, Trefferquote und der F1-Wert für jede Klasse berechnet. Dies erlaubt eine Bewertung der Klassifikatoren hinsichtlich ihrer Fähigkeit die einzelnen Klassen zu erkennen.

Zur Auswahl des Klassifikators für den Einsatz in der Anwendung wird die Falschklassifizierungsrate als wichtigstes Bewertungskriterium herangezogen.

Vortrainierter Klassifikator

Tabelle 7.1 zeigt, dass Trefferquote und Genauigkeit bei beiden Modellen auf einem sehr hohen Niveau sind. In beiden Auflösungen werden alle Instanzen der Klasse *Ausgangsrechnung* richtig erkannt. Die niedrigste Trefferquote mit 0,85 ergibt sich bei einer Auflösung von 200px für *Leistungsbeschreibungen*.

Das Modell mit kleiner Auflösung macht für acht Dokumente falsche Vorhersagen. Bei größerer Auflösung werden aus der Teststichprobe sechs Dokumente fehlerhaft erkannt. Der F1-Wert über alle Klassen ist mit 0,988 bei 200px um 5% höher als bei 150px (0,983).

Klasse	precision		recall		F1	
	150px	200px	150px	200px	150px	200px
Ausgangsrechnung	0,99	0,97	1,00	1,00	1,00	0,99
Eingangsrechnung	1,00	0,99	0,99	1,00	0,99	0,99
Leistungsnachweis_md	0,91	1,00	0,97	1,00	0,94	1,00
Leistungsnachweis_sy	0,98	1,00	1,00	0,98	0,99	0,99
Leistungsbeschreibung	0,88	1,00	0,88	0,85	0,88	0,92
Angebotsschreiben	1,00	1,00	0,96	0,98	0,98	0,99

Tabelle 7.1: Auswertung „Vortrainierter Klassifikator“ mit Rotation & Scherung

Auffällig ist, dass trotz der stärkeren Gewichtung der unterrepräsentierten Klassen beim Training, die meisten Fehler dennoch bei Dokumenten der Klasse *Leistungsbeschreibung* gemacht werden, wie aus den Konfusionsmatrizen in Abbildung 7.2 abgelesen werden kann.

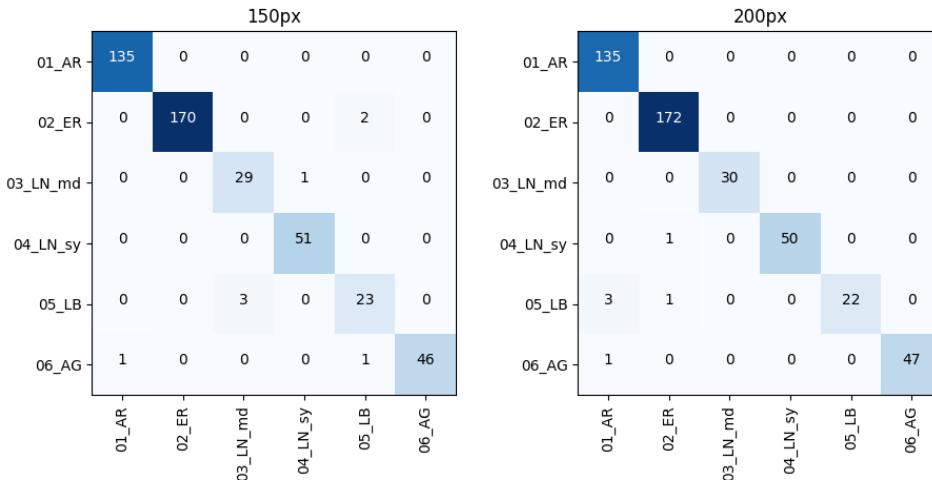


Abbildung 7.2: Konfusionsmatrix „Vortrainierter Klassifikator“ 150px & 200px

Mehrstufiger Klassifikator

Bei diesem Ansatz wird darauf verzichtet die einzelnen Modelle des Klassifikators für sich zu bewerten, da nur die Gesamtleistung ausschlaggebend ist.

Auch bei diesem Lösungsansatz können hohe Werte für Trefferquote und Korrektklassifizierungsrate in beiden Auflösungen festgestellt werden (siehe Tabelle 7.2).

Klasse	precision		recall		F1	
	150px	200px	150px	200px	150px	200px
Ausgangsrechnung	0,98	0,99	1,00	1,00	0,99	0,99
Eingangsrechnung	0,99	0,99	0,99	1,00	0,99	1,00
Leistungsnachweis_md	0,97	1,00	0,93	0,93	0,95	0,97
Leistungsnachweis_sy	1,00	1,00	1,00	1,00	1,00	1,00
Leistungsbeschreibung	1,00	0,92	0,92	0,88	0,96	0,90
Ausgangsrechnung	1,00	1,00	1,00	1,00	1,00	1,00

Tabelle 7.2: Auswertung des „Mehrstufigen Klassifikators“ mit Rotation & Scherung



Beide Klassifikatoren können alle *Ausgangsrechnungen*, *Leistungsnachweise_sy* und *Angebotsschreiben* fehlerfrei erkennen. Der niedrigste Wert für die Trefferquote wird in beiden Auflösungen bei *Leistungsbeschreibungen* ermittelt.

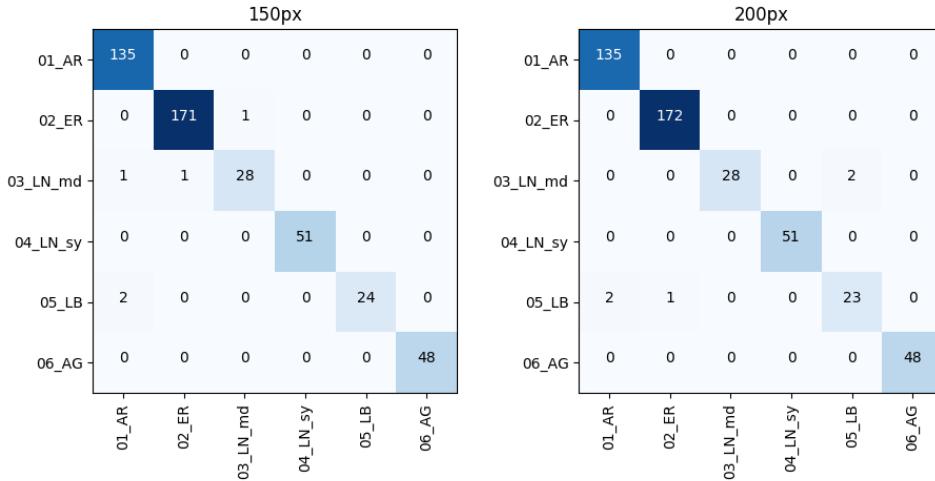


Abbildung 7.3: Konfusionsmatrix des „Mehrstufigen Klassifikators“ in 150px & 200px

Die Konfusionsmatrizen in Abbildung 7.3 zeigen für beide Auflösungen jeweils fünf fehlerhaft klassifizierte Dokumente. Wie bei dem Lösungsansatz mit dem „Vortrainierten Klassifikator“ werden die meisten Fehlentscheidungen in beiden Klassen mit wenigen Mustern gemacht.

7.2 Auswahl des Klassifikators

Der niedrigste Wert der Falschklassifizierungsrate wird mit dem „Mehrstufigen Klassifikator“ erreicht (Tabelle 7.3). In beiden Auflösungen werden, mit fünf Falschklassifizierungen, gleich viele Fehler gemacht.

Somit kann dieser Wert allein kein ausreichendes Entscheidungskriterium liefern. Als weiterer Entscheidungsfaktor wird aus diesem Grund der Makro-F1 Wert herangezogen. Dieser liegt bei kleinerer Auflösung mit 0,981 über dem bei größerer Auflösung mit 0,976. Daher wird in der Anwendung der „Mehrstufige Klassifikator“ bei einer Auflösung von 150-Pixeln eingesetzt. Die leicht besseren Werte des Modells sowie die Forderung nach möglichst großem Datenschutz sind ausschlaggebend für die Wahl.

Modell	KKR		FKR	
	150px	200px	150px	200px
Vortrainierter Klassifikator	0,983	0,987	0,017	0,013
Mehrstufiger Klassifikator	0,989	0,989	0,011	0,011

Tabelle 7.3: Korrektklassifizierungs- und Falschklassifizierungsrate beider Ansätze

Eine Ausgabe der höchsten Klassenwahrscheinlichkeit der falsch klassifizierten *Eingangsrechnung* liefert einen Wert von 38,3% bei *Leistungsnachweis_md*. Die beiden *Leistungsbeschreibungen* werden zu 83,9% sowie zu 76,9% als *Ausgangsrechnung* erkannt. Die hohen Wahrscheinlichkeitswerte können mit der starken Ähnlichkeit der Dokumente dieser Klassen erklärt werden. Eine Ausgabe der Vorhersagewahrscheinlichkeit korrekt klassifizierter Dokumente liefert bei wenigen Exemplaren ebenfalls Wahrscheinlichkeiten in diesem Bereich.

Im Anhang unter A.2 sind die Auswertungen nochmals ohne Rotation und Scherung abgebildet. Die dort gezeigten Ergebnisse unterstreichen die Aussage der Evaluation.

7.3 Hypothese

Ein Erklärungsversuch für die sehr guten Ergebnisse beider Ansätze zur Klassifikation könnte in den Daten begründet sein. Die Elemente der Teststichprobe sind zufällig ausgewählte Dokumente der Stichprobe (vgl. Abschnitt 3.1). Die Muster innerhalb einer Klasse sind in ihrem Erscheinungsbild sehr ähnlich, da sie, wie in Abschnitt 3.2 bereits erwähnt, immer den gleichen Regeln im Aufbau folgen. Somit unterscheiden sich die Muster der Teststichprobe von den Mustern der Trainingsstichprobe nur unwesentlich. Ein gewisses Maß an Überadaption kann dadurch nicht ausgeschlossen werden, obwohl mit Maßnahmen wie Dropout, Batch-Normalization und zufälliges Verändern der Daten versucht wird, dies zu vermeiden.

Sollte sich dieser Umstand wie beschrieben darstellen, könnte dies für die konkrete Aufgabe sogar einen positiven Einfluss haben. Denn auch im zukünftigen produktiven Einsatz des Klassifikators werden alle internen Dokumente immer den gleichen Regeln im Layout folgen und so sehr große Ähnlichkeit zu den Dokumenten der Stichprobe haben. Um die Fähigkeit der Generalisierung des Klassifikators bewerten zu können, ist dies suboptimal, könnte aber eine mögliche Erklärung der sehr guten Ergebnisse liefern.

Die Gleichartigkeit der Dokumente könnte ebenfalls als Erklärung für die Reduktion der einzelnen Modelle des „Mehrstufigen Klassifikators“ auf wenige und kleine Filter pro Schicht dienen. Abbildung 7.4 zeigt je ein Dokument der Klassen *Angebotsschreiben* und *Leistungsbeschreibung* in der Auflösung und Skalierung zur Klassifikation. In beiden Dokumenten sind wiederkehrende Elemente, wie zum Beispiel die Tabellen, zu erkennen. Eine Vermutung ist, dass als Folge der vielen Übereinstimmungen nur wenige Filter notwendig sind, um die einzelnen Merkmale zu detektieren. Zur Unterscheidung der Dokumente könnten demnach wenige Filter ausreichend sein, da diese nur verhältnismäßig wenige unterschiedliche Formen detektieren müssen, die auf ähnliche Weise in mehreren Klassen vorhanden sind.



Abbildung 7.4: Angebotsschreiben und Leistungsbeschreibung in 150×150 -Pixel Auflösung



8 Schlussbetrachtung

8.1 Fazit

Auf die zentrale Frage dieser Arbeit nach der Möglichkeit der automatischen Klassifikation von Dokumenten durch Verwendung eines CNNs kann eine positive Antwort gegeben werden. Mit einer Falschklassifizierungsrate von unter 2% erreichen alle der vorgestellten Klassifikatoren aus technischer Sicht eine sehr gute Erkennungsleistung. Die Anwendung zur Klassifikation wurde erfolgreich implementiert und kann im Rahmen der beschriebenen Problemstellung im Unternehmen eingesetzt werden.

Dass keiner der beiden vorgestellten Lösungsansätze gegenüber dem jeweils anderen einen signifikanten Vorteil hinsichtlich der Genauigkeit gezeigt hat, kann als bestimmendes Fazit dieser Arbeit herausgestellt werden. Aus diesem Grund ist die Evaluation sowie die anschließende Wahl des Klassifikators weniger deutlich ausgefallen, als dies noch zu Beginn der Arbeit erwartet wurde.

Dies wurde vor allem durch die iterative Optimierung des „Mehrstufigen Klassifikators“ erreicht. Hier ist besonders eine Leistungssteigerung durch die Verwendung von Batch-Normalization-Layern hervorzuheben. Der Einsatz dieser Technik erlaubte bei gleichbleibender Leistung eine deutliche Reduktion sowohl der Covolutional-Layer als auch der Anzahl der Filter pro Layer.

Ähnlich wie *Tensmeyer und Martinez* in ihrer Arbeit festgestellt haben, wurde auch hier ein positiver Einfluss durch die Reduktion der Schichten des Netzes, trotz der geringen Anzahl an Trainingsdaten, festgestellt. Zu Beginn der Optimierung wurden einzelne Klassen nur mit einer Korrektklassifizierungsrate von unter 80% erkannt. Die Verkleinerung der Modelle brachten die letztendlichen Werte von über 98%.

Eine weitere deutliche Verbesserung der Leistung wurde mit der Einführung der unterschiedlichen Gewichtungen der einzelnen Klassen erzielt. Dass in allen Klassifikatoren nach wie vor die niedrigste Trefferquote bei den unterrepräsentierten Klassen erreicht wurde, zeigt allerdings auch, dass hier noch weiteres Optimierungspotenzial besteht.



8.2 Ausblick

Wie gezeigt wurde, bietet ein Grenzwert, der überschritten werden muss, nur begrenzte Sicherheit gegen falsch sortierte Dokumente. Wird der Grenzwert zu hoch angesetzt, können zwar falsch eingesortierte Dokumente weitgehend vermieden werden. Dadurch werden jedoch auch korrekt klassifizierte Dokumente in die Rückweisungsklasse verschoben. Um diesem Problem zu begegnen, muss der Klassifikator weiter optimiert werden. Sinnvoll hierfür kann es sein, zur Klassifikation der Dokumente inhaltliche Elemente mit heranzuziehen. Dazu könnten vor Verkleinerung der Dokumente, Techniken der Textklassifikation eingesetzt werden, indem beispielsweise zusätzlich die Betreffzeile eines Dokuments ausgewertet wird.

Zur Verbesserung der Güte des Klassifikators können Tests mit einer alternativen Aufteilung der Klassen durchgeführt werden. Hier könnte es sinnvoll sein, die Klassen nicht nach inhaltlichen Gesichtspunkten aufzuteilen, sondern die Aufteilung so zu gestalten, dass möglichst unterschiedliche Klassen von den Klassifikatoren erkannt werden. Dies könnte das Klassifikations-Problem eines jeden Modells und so die Falschklassifikationsrate weiter verringern.

In dieser Arbeit wurde nicht untersucht, welche Ergebnisse ein Klassifikator bestehend aus nur einem selbst erstellten Modell zur Klassifikation der Dokumente liefern würde. Im Zuge der Weiterentwicklung wird dies durchgeführt.

Damit die automatische Klassifikation von Dokumenten für das Unternehmen in weiteren Einsatzbereichen nutzbar wird, muss zum Training eines Klassifikators eine Anwendung mit GUI entwickelt werden. Dies soll einem die Benutzer Auswahl der Daten und die Durchführung des Trainings eines Modells erleichtern. Auf diese Weise können dem bestehenden Klassifikator weitere Klassen hinzugefügt werden. Einzelnen Benutzern wird es damit möglich, eigene Klassifikatoren für ihre Anwendungszwecke zu erstellen.

Eine sinnvolle Erweiterung der Anwendung wäre die Implementierung des clientseitigen Teils als mobile Anwendung. Dies würde es Mitarbeitern erlauben, mit mobilen Endgeräten wie Smartphones oder Tablets, Bilder von Dokumenten zu machen und diese an den Klassifikator zu schicken.



Literatur

- [Afz+15] M. Z. Afzal u. a. „Deepdocclassifier: Document classification with deep Convolutional Neural Network“. In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. Aug. 2015, S. 1111–1115. DOI: [10.1109/ICDAR.2015.7333933](https://doi.org/10.1109/ICDAR.2015.7333933).
- [Cho16] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2016. URL: <http://arxiv.org/pdf/1610.02357.pdf>.
- [Den+09] Jia Deng u. a. „ImageNet: A large-scale hierarchical image database“. In: *IEEE Conference on Computer Vision and Pattern Recognition, 2009*. Piscataway, NJ: IEEE, 2009, S. 248–255. ISBN: 978-1-4244-3992-8. DOI: [10.1109/CVPR.2009.5206848](https://doi.org/10.1109/CVPR.2009.5206848).
- [Fir17] Doukkali Firdaouss. *Batch normalization in Neural Networks*. 2017. URL: <https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c> (besucht am 13.02.2018).
- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [Gro95] Grother, Patrick J. National Institute of Standards and Technology. „NIST Handprinted Forms and Characters, NIST Special Database 19“. In: National Institute of Standards and Technology, 1995. DOI: [10.18434/T4H01C](https://doi.org/10.18434/T4H01C).
- [HUD15] Adam W. Harley, Alex Ufkes und Konstantinos G. Derpanis. „Evaluation of deep convolutional nets for document image classification and retrieval“. In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, S. 991–995. ISBN: 978-1-4799-1805-8. DOI: [10.1109/ICDAR.2015.7333910](https://doi.org/10.1109/ICDAR.2015.7333910).
- [IS15] Sergey Ioffe und Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: <http://arxiv.org/abs/1502.03167>.
- [Kan+14] Le Kang u. a. „Convolutional Neural Networks for Document Image Classification“. In: *2014 22nd International Conference on Pattern Recognition*. IEEE, 2014, S. 3168–3172. ISBN: 978-1-4799-5209-0. DOI: [10.1109/ICPR.2014.546](https://doi.org/10.1109/ICPR.2014.546).



- [Kru+15] Rudolf Kruse u. a. *Computational Intelligence*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015. ISBN: 978-3-658-10903-5. DOI: 10.1007/978-3-658-10904-2.
- [Lew+06] D. Lewis u. a. „Building a test collection for complex document information processing“. In: *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. Hrsg. von Efthimis N. Efthimiadis. New York, NY: ACM, 2006, S. 665. ISBN: 1595933697. DOI: 10.1145/1148170.1148307.
- [Net17] Convolutional Neural Network. *Convolutional Neural Network*. 2017. URL: <https://de.mathworks.com/discovery/convolutional-neural-network.html> (besucht am 08.01.2018).
- [Nie03] Heinrich Niemann. *Klassifikation von Mustern*. Erlangen: Springer, 2003.
- [Nie17] Micheal A. Nielsen. *Neural networks and deep learning*. 2017. URL: <http://neuralnetworksanddeeplearning.com/chap6.html> (besucht am 08.01.2018).
- [Pat17] Adam Gibson. Josh Patterson. *Deep Learning*. [S.l.]: O'Reilly Media, Inc, 2017. ISBN: 9781491924570.
- [Ped+11] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.
- [PyQ18] PyQt. *What is PyQt?* 2018. URL: <https://riverbankcomputing.com/software/pyqt/intro>.
- [RM18] Sebastian Raschka und Vahid Mirjalili. *Machine Learning mit Python und Scikit-learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*. 2., aktualisierte und erweiterte Auflage. Frechen: mitp, 2018. ISBN: 978-3-95845-733-1.
- [Ros58] F. Rosenblatt. „The perceptron: A probabilistic model for information storage and organization in the brain“. In: *Psychological Review* 65.6 (1958), S. 386–408. ISSN: 1939-1471. DOI: 10.1037/h0042519.
- [Rus+15] Olga Russakovsky u. a. „ImageNet Large Scale Visual Recognition Challenge“. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), S. 211–252. DOI: 10.1007/s11263-015-0816-y.
- [Sri+14] Nitish Srivastava u. a. „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“. In: *Journal of Machine Learning Research* 15 (2014), S. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.



-
- [Sze+15] Christian Szegedy u. a. *Rethinking the Inception Architecture for Computer Vision*. 2015. URL: <http://arxiv.org/pdf/1512.00567>.
 - [Ten18] Inception in TensorFlow. *Inception in TensorFlow*. 2018. URL: <https://github.com/tensorflow/models/tree/master/research/inception>.
 - [TM17] Chris Tensmeyer und Tony Martinez. *Analysis of Convolutional Neural Networks for Document Image Classification*. 2017. URL: <http://arxiv.org/pdf/1708.03273v1>.
 - [Tut18] The Python Tutorial. *The Python Tutorial — Python 3.6.4 documentation*. 2018. URL: <https://docs.python.org/3/tutorial/index.html>.
 - [vCV11] Stéfan van der Walt, S. Chris Colbert und Gaël Varoquaux. „The NumPy Array: A Structure for Efficient Numerical Computation“. In: *Computing in Science & Engineering* 13.2 (2011), S. 22–30. ISSN: 1521-9615. DOI: [10.1109/MCSE.2011.37](https://doi.org/10.1109/MCSE.2011.37).
 - [Vis17] CS231n Convolutional Neural Networks for Visual Recognition. *CS231n Convolutional Neural Networks for Visual Recognition*. 2017. URL: <http://cs231n.github.io/> (besucht am 08.01.2018).
 - [Zac16] Giancarlo Zaccone. *Getting started with TensorFlow: Get up and running with the latest numerical computing library by Google and dive deeper into your data!* Community experience distilled. Birmingham, UK: Packt Publishing Ltd, 2016. ISBN: 978-1-78646-906-9. URL: <http://lib.myilibrary.com/detail.asp?ID=944179>.



A Anhang

A.1 Architekturen der Modelle des Mehrstufigen Klassifikators

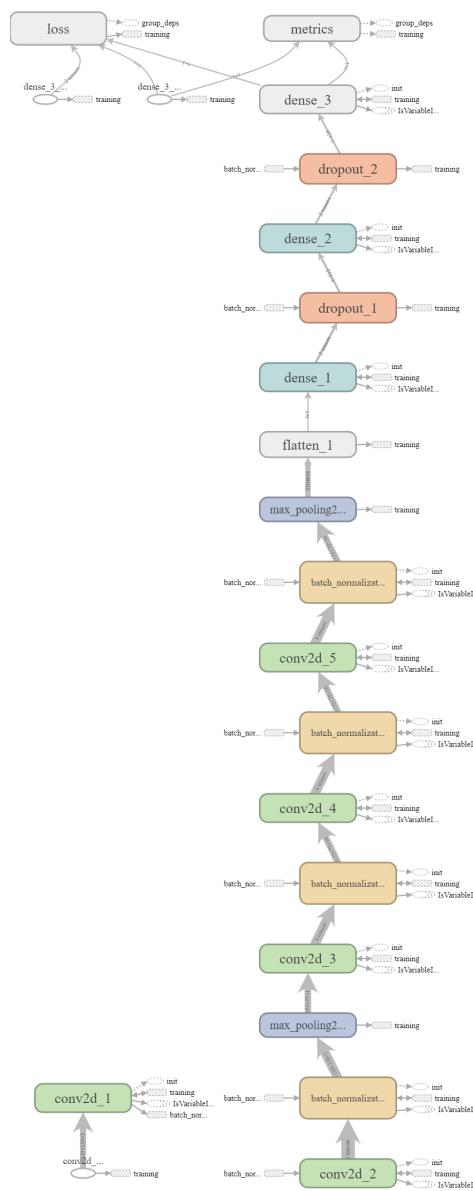


Abbildung A.1: Architektur Intern-Extern-Modell

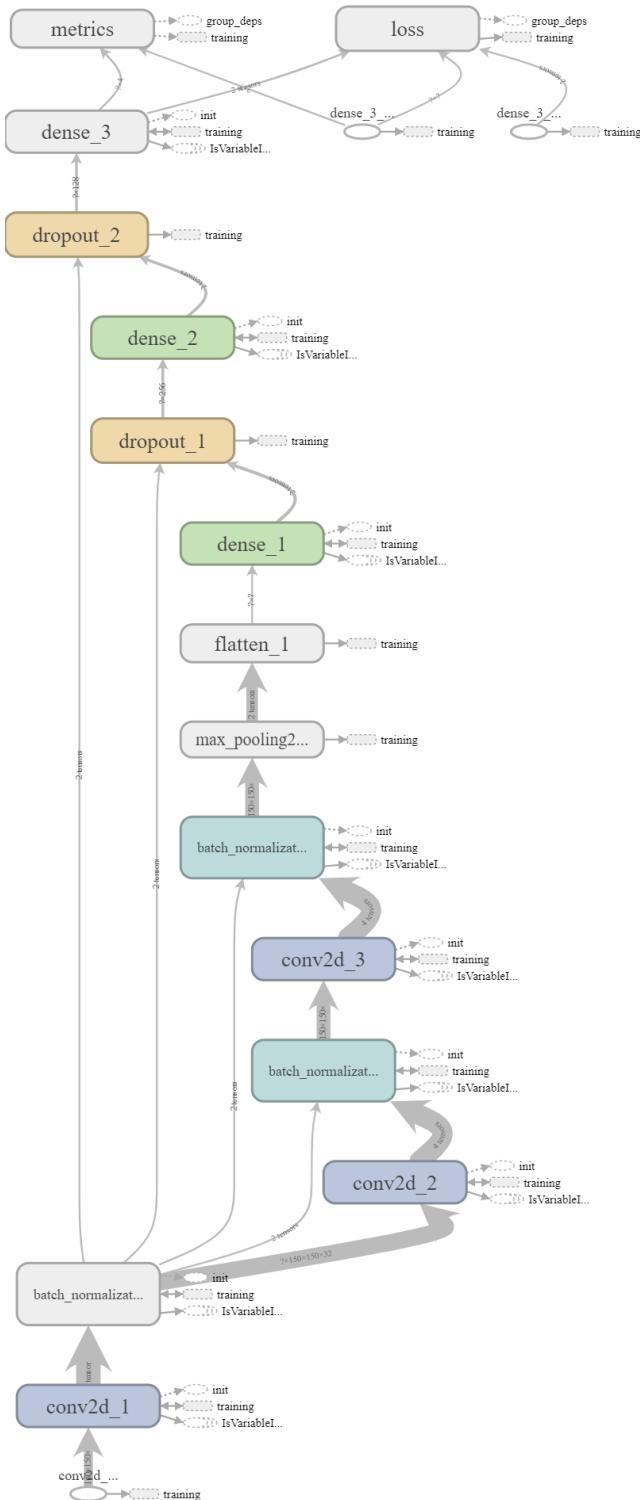


Abbildung A.2: Intern-Modell

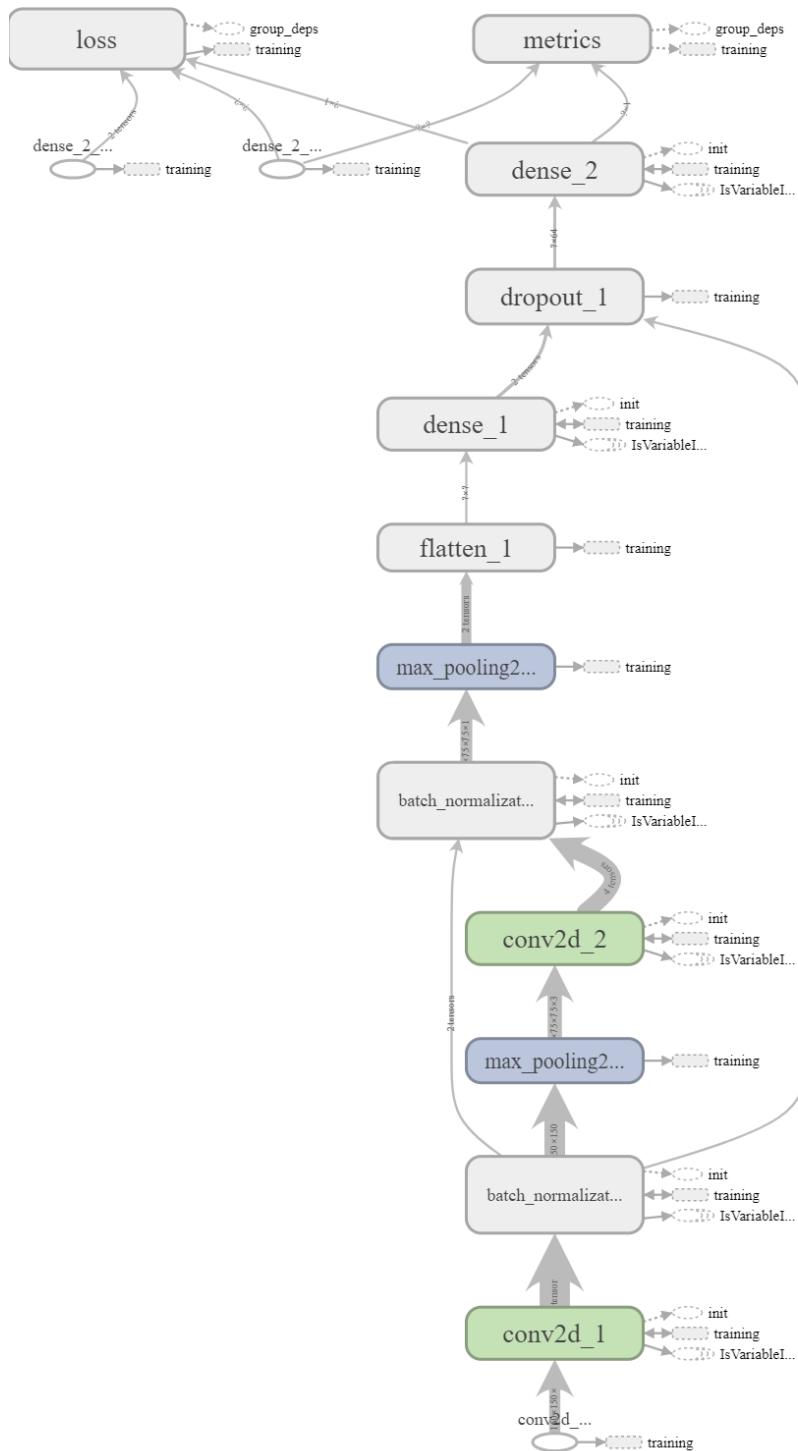


Abbildung A.3: Extern-Modell



A.2 Testergebnisse ohne Rotation und Scherung von Dokumenten

Vortrainierter Klassifikator

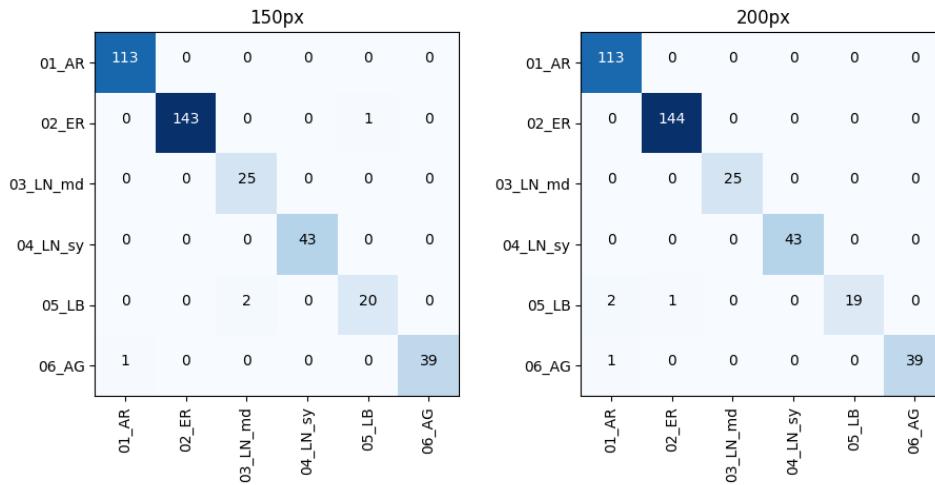


Abbildung A.4: Konfusionsmatrix „Vortrainierter Klassifikator“ 150px und 200px ohne Rotation & Scherung

Klasse	precision		recall		F1	
	150px	200px	150px	200px	150px	200px
Ausgangsrechnung	0,99	0,97	1,00	1,00	1,00	0,99
Eingangsrechnung	1,00	0,99	0,99	1,00	1,00	1,00
Leistungsnachweis_md	0,93	1,00	1,00	1,00	0,96	1,00
Leistungsnachweis_sy	1,00	1,00	1,00	1,00	1,00	1,00
Leistungsbeschreibung	0,95	1,00	0,91	0,86	0,93	0,93
Angebot	1,00	1,00	0,97	0,97	0,99	0,99

Tabelle A.1: Auswertung „Vortrainierter Klassifikator“ ohne Rotation & Scherung



Mehrstufiger Klassifikator



Abbildung A.5: Konfusionsmatrix „Mehrstufiger Klassifikator“ 150px und 200px ohne Rotation & Scherung

Klasse	precision		recall		F1	
	150px	200px	150px	200px	150px	200px
Ausgangsrechnung	0,99	0,99	1,00	1,00	1,00	1,00
Eingangsrechnung	0,99	0,99	0,99	1,00	0,99	1,00
Leistungsnachweis_md	0,96	1,00	0,96	0,96	0,96	0,98
Leistungsnachweis_sy	1,00	1,00	1,00	1,00	1,00	1,00
Leistungsbeschreibung	1,00	0,95	0,95	0,91	0,98	0,93
Angebot	1,00	1,00	1,00	1,00	1,00	1,00

Tabelle A.2: Auswertung „Mehrstufiger Klassifikator“ mit Rotation & Scherung

A.3 Trainingsverlauf ohne Batch-Normalization-Layer

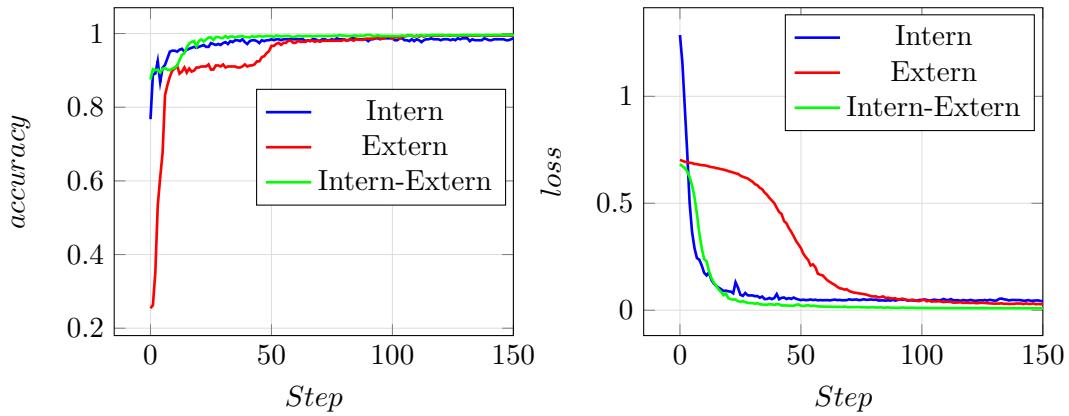


Abbildung A.6: „Mehrstufiger Klassifikator“ ohne Batch-Normalization

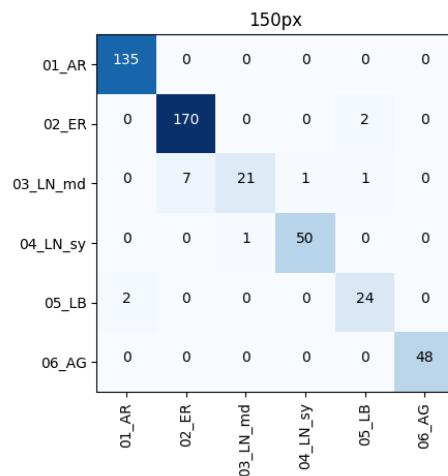


Abbildung A.7: Konfusionsmatrix „Mehrstufiger Klassifikator“ ohne Batch-Normalization



Eidesstattliche Erklärung

Ich, Tobias Velke, Matrikel-Nr. 2693554, versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema

Automatische Klassifizierung von digitalisierten Dokumenten durch Verwendung eines neuronalen Netzes

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Bachelorarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Prüfungsamt der Ohm-Hochschule abzugeben oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Nürnberg, den 27. Februar 2018

TOBIAS VELKE