Automatische Erkennung von Klausurdaten mittels Bildverarbeitung und Machine Learning

Masterarbeit

von

Fabian Mersch

aus

Düsseldorf

vorgelegt am

Lehrstuhl für Rechnernetze
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

März 2024

Betreuer:
Dr. Markus Brenneis

Zusammenfassung

Nachdem eine Klausur korrigiert wurde, müssen die Ergebnisse in ein digitales System eingetragen werden. Die einzutragenden Daten auf einem Klausurdeckblatt umfassen den Vor- und Nachname des Studenten, sowie die Matrikelnummer. Darüber hinaus müssen die erreichten Punkte des Studenten eingetragen werden, welche in Tabellenform aufgeführt sind. Diese Übertragung der Ergebnisse ist ein langwieriger und fehleranfälliger Prozess. Aktuell werden diese Daten händisch in ein System eingetragen. Dies ist gerade bei Pflichtmodulen ein aufwändiger Vorgang, da hunderte Namen mit unterschiedlichen Handschriften erkannt werden müssen. Zusätzlich müssen knapp zehn verschiedene Punktzahlen übertragen werden.

Daher wurde im Rahmen dieser Arbeit der Prozess automatisiert. In dieser Masterarbeit wurde ein Programm erstellt, das diese Informationen mittels einer Kamera automatisch aufnimmt, mit Machine Learning verarbeitet und die Daten übertragen kann. Dadurch konnte eine monotone Arbeit, für die in der Regel zwei Personen mehrere Stunden benötigen, deutlich angenehmer gemacht werden. Der Zeitbedarf für das Eintragen mit dem Programm dieser Arbeit sinkt um 35 %.

Das erstellte Programm ist in verschiedene Module aufgeteilt, die über eine API miteinander kommunizieren. Die verschiedenen Module sind für die Erkennung verschiedener Datentypen erstellt, wie zum Beispiel die Namenserkennung, oder die Punkteerkennung. Vorteil dieses modularen Aufbaus mit einer definierten API ist der einfache Tausch der Unterprogramme, um die Erkennung in Zukunft weiter zu optimieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In den meisten Informatikmodulen an der Heinrich-Heine-Universität (HHU) wird zur Leistungserhebung eine Klausur geschrieben. Nach der Korrektur müssen alle relevanten Daten in ein System eingetragen werden. Diese umfassen den Vor- und Nachnamen des Studierenden, seine Matrikelnummer sowie die erreichten Punkte für jede Aufgabe und die Gesamtpunktzahl. Besonders bei Pflichtmodulen gestaltet sich die Eintragung langwierig, da hunderte Namen mit unterschiedlichen Handschriften manuell erfasst werden müssen, ebenso wie rund zehn Teilpunkte und die Gesamtpunktzahl für jede Klausur übertragen werden müssen, siehe Abbildung 1.1. In diesem Kapitel wird darauf eingegangen, wie dieser Vorgang optimiert werden könnte und welche bestehenden Ansätze dafür bereits existieren.

1.1 Motivation

Um den Prozess des Eintragens zu optimieren wurde im Rahmen dieser Arbeit ein System entwickelt, das alle relevanten Informationen automatisch erkennt und verarbeitet. Dazu wird das Deckblatt der Klausur unter eine Dokumentenkamera platziert und aufgenommen. Durch Bildverarbeitung und maschinelles Lernen werden die benötigten Daten erkannt. Anschließend können diese Informationen in das zentrale System der Universität eingetragen werden. Dieser neue Vorgang kann somit eine monotone Arbeit, für die normalerweise zwei Personen mehrere Stunden benötigen, erheblich effizienter gestalten.

Für die Handschriftenerkennung werden in dieser Arbeit Machine-Learning-Modelle ver-

wendet. Es werden zwei verschiedene Modelle eingesetzt: Für die Erkennung von Namen wird das SimpleHTR-Modell [16] verwendet, während für die Zahlenerkennung bei den Aufgabenpunktzahlen und den Matrikelnummern ein eigenes Modell zum Einsatz kommt.

Durch den modularen Aufbau des Systems und definierte APIs können die Erkennungsmodelle in Zukunft bei Bedarf einfach ausgetauscht werden. Darüber hinaus ermöglicht dies eine Erweiterung des Systems, wenn zukünftig weitere Daten erfasst werden sollen.

Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Dr. Markus Brenneis

08. Februar 2022

Hauptklausur

Programmierung
WS 21/22

Nachname:	Mustermann	Vorname:	Max
Matrikelnummer:	3141593	Sitzplatznummer:	26
Zusätzliche Blätter:	0	Unterschrift:	Max Mustermann

Hinweise:

- Diese Klausur enthält 20 nummerierte Klausurseiten. Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält. Sie dürfen die Heftung der Klausur nicht auftrennen.
- Sie erhalten außerdem von uns leere Blätter. Sollten Sie auf diesen Blättern für die Korrektur relevante Teile Ihrer Lösung notieren, markieren Sie dies deutlich an der entsprechenden Aufgabe und auf dem zusätzlichen Blatt. Schreiben Sie auf alle zusätzlichen Blätter Ihren Namen. Geben Sie außerdem oben auf dem Deckblatt an, wie viele zusätzliche Blätter Sie zur Korrektur abgeben. Wenn Sie weiteres Papier benötigen, melden Sie sich bitte.
- Alle Fachbegriffe in dieser Klausur werden wie in der Vorlesung definiert verwendet. Alle Fragen beziehen sich auf die in der Vorlesung vorgestellte Java-Version 11. Programmcode muss in Java geschrieben werden.
- Antworten dürfen auf Deutsch oder Englisch gegeben werden. Sofern keine ausformulierten Sätze verlangt sind, reichen nachvollziehbare Stichworte als Antwort.
- Zugelassene Hilfsmittel: eine beidseitig beschriebene oder bedruckte DIN-A4-Seite, Wörterbuch (Wörterbücher müssen vor Beginn der Klausur den Aufsichtspersonen zur Kontrolle vorgelegt werden.)
- Schalten Sie Ihr Mobiltelefon aus. Täuschungsversuche führen zum sofortigen Ausschluss von der Klausur. Die Klausur wird dann als nicht bestanden gewertet.
- Schreiben Sie **nicht** mit radierbaren Stiften und auch **nicht** mit rot!

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	9	Σ
Punktzahl	7	5	2	13	13	8	9	6	27	90
Erreicht	6	4	1	12	11	7	6	5	24	76

1

Abbildung 1.1: Ein ausgefülltes Deckblatt einer Klausur. In blau sind die Informationen des Studenten geschrieben, in rot sind Teilpunkte jeder Aufgabe, sowie die Gesamtpunktzahl der Klausur geschrieben. Relevant für die Auswertung ist die handschriftliche Matrikelnummer, sowie der Vor- und Nachname und alle roten handschriftlichen Punkte.

1.2 Bestehende Projekte

In einem vergleichbaren Projekt [1] wurde eine ähnliche Problematik gelöst. Ein Lehrer an einer vietnamesischen Schule verbrachte viel Zeit damit, die erreichten Punkte der Schüler nach jedem Test manuell zu erfassen. Um diesen Prozess zu beschleunigen, entwickelte ein Student ein Programm, das diesen Vorgang automatisierte. Auf einem Testblatt, wie in Abbildung 1.2 dargestellt, sind der Name des Schülers, eine eindeutige Schüler-ID und die erzielte Punktzahl vermerkt.

Zunächst wird ein Bild des vietnamesischen Tests aufgenommen. Hierzu wird ein Smartphone mit einer IP-Kamera-App verwendet, die die Bilddatei an das Programm auf dem Computer sendet. Dort wird das Bild durch Feature-Mapping ausgerichtet, wobei dieser Vorgang in Kapitel 2.2 genauer erläutert wird. Anschließend können die relevanten Bereiche ausgeschnitten werden. Der Kontrast dieser Ausschnitte wird mithilfe der sogenannten *Top-Hat- und Black-Hat*-Methode erhöht. Danach werden die ausgeschnittenen Bereiche segmentiert, um beispielsweise den Vor- und Nachnamen getrennt vorhersagen zu können. Diese Ausschnitte werden in ein neuronales Netzwerk eingegeben, das den entsprechenden Text vorhersagt. Das neuronale Netz setzt sich aus Convolutional Neural Layern, Attention Layern, Rekurrenten neuronalen Layern und Connectionist Temporal Classification Layern zusammen.

Dieses Projekt kann für die an der HHU verwendeten Deckblätter nicht genutzt werden. Das neuronale Netz wurde beispielsweise auf anderen Daten trainiert, wodurch die Vorhersage von Namen aus dem europäischen Raum erschwert werden. Darüber hinaus unterscheiden sich die Punkte auf Tests von den Deckblättern der HHU. Auf den Deckblättern der HHU müssen Punkte für jede Aufgabe erfasst werden, auf den Tests nur eine Punktzahl. Aufgrund vieler unterschiedlicher Anforderungen wurde deshalb ein angepasstes System im Rahmen dieser Arbeit entwickelt, um verschiedene Deckblätter der HHU zu verarbeiten.

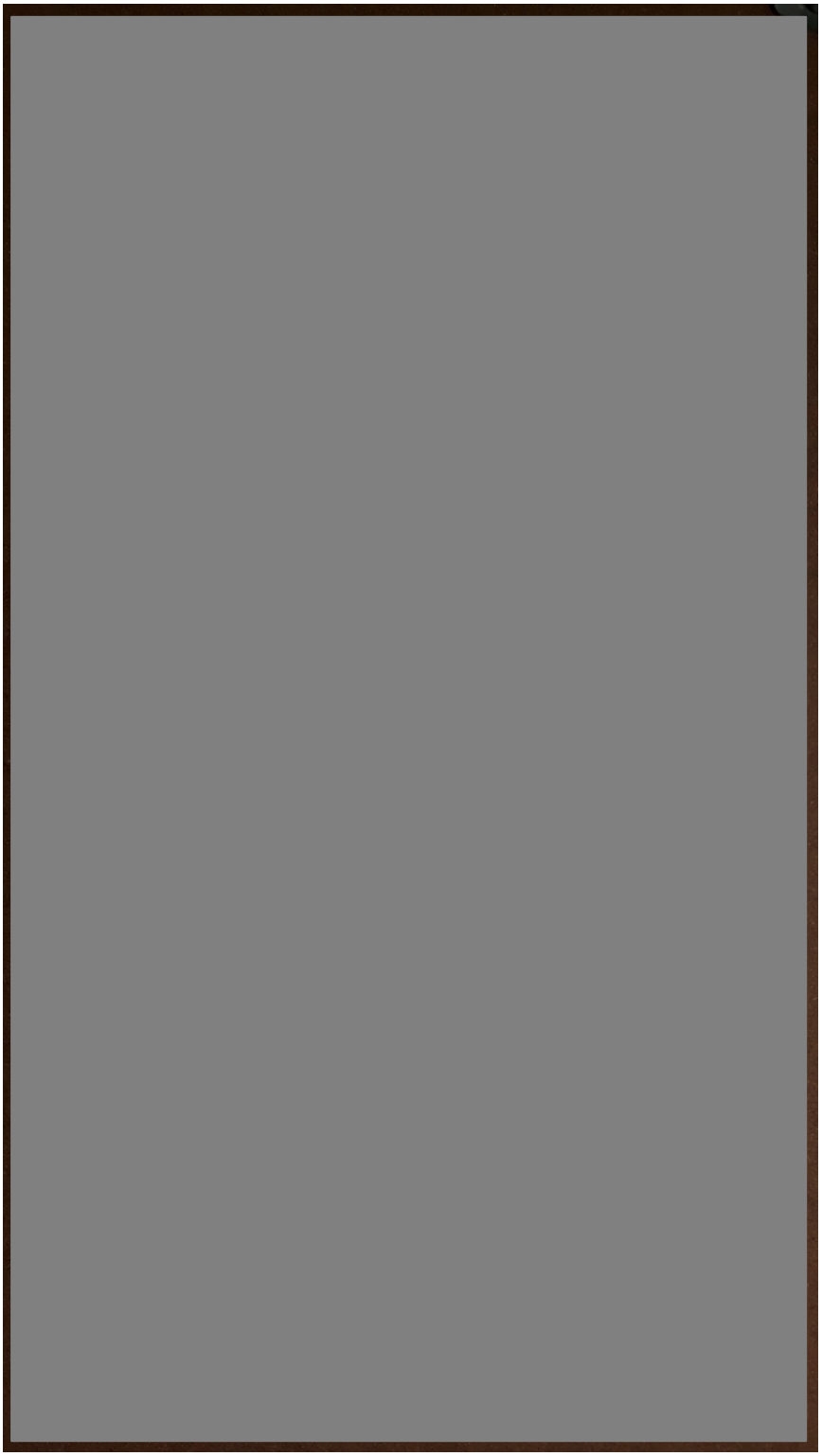


Abbildung 1.2: Beispiel eines Tests der in dem Projekt [1] verwendet wird. Die mit blauem Kugelschreiber geschriebenen Daten werden erfasst.

Ein anderer Ansatz, um die Klausurergebnisse automatisch zu übertragen, umfasst eine angepasste Punktetabelle. Dieses System wurde an der HHU entwickelt und wurde 2016 bis 2020 getestet. In eine angepasste Punktetabelle wird dabei nicht die Punktzahl handschriftlich geschrieben. Es existieren verschiedene kleine Kästchen in einer Punktebox. Jedes Kästchen steht dabei für eine bestimmte Punktzahl und kann ausgemalt werden. Durch ein Programm wird erfasst, welche Kästchen ausgemalt wurden und entsprechend werden die Punkte vergeben.

Dieser Vorgang konnte sich in der Praxis allerdings nicht durchsetzten. Grund dafür war zum einen die schlechte Korrekturfähigkeit. Wenn ein falsches Kästchen markiert wurde, musste die Box aufwändig mit Tippex rekonstruiert werden. Darüber hinaus war die Verwendung dieser Kästchen nicht sehr intuitiv, wodurch dieses System verworfen wurde.

1.3 Aufbau der Arbeit

In den kommenden Abschnitten wird zunächst die erforderliche Theorie für diese Arbeit dargelegt. Danach wird aufgezeigt, wie die relevanten Bereiche auf einem Deckblatt identifiziert werden können. Anschließend wird erläutert, welche Methoden zur Erkennung der Informationen in den relevanten Bereichen angewendet werden können. Im Anschluss daran erfolgt eine Beschreibung der Softwarearchitektur, welche die zuvor beschriebenen Prozesse vereint. Abschließend erfolgt eine Auswertung des Projekts anhand praktischer Tests, sowie ein Ausblick auf mögliche zukünftige Arbeiten.

Kapitel 2

Theorie

In diesem Kapitel werden die theoretischen Hintergründe des Projektes erläutert. Zum einen werden die verwendeten neuronalen Netze für die Informationsvorhersage erklärt, zum anderen das Feature Mapping welches zur Lokalisierung der relevanten Daten dient. Darüber hinaus wird in diesem Projekt OpenCV zur Bildverarbeitung genutzt, sowie die Levensthein Distanz um Texte miteinander zu vergleichen.

2.1 Machine Learning

Die Erkennung von Handschrift stellt eine Form der Mustererkennung dar. Menschen sind in der Lage visuell Muster sehr schnell zu erkennen und darauf zu reagieren. Unser Gehirn verarbeitet Informationen, die durch das Auge bereitgestellt werden, in Bruchteilen von Sekunden. Diese Leistung wird dabei durch etwa 86 Milliarden Neuronen, die über Synapsen miteinander verbunden sind, ermöglicht [9].

In der Informatik kann das gleiche Prinzip der Mustererkennung durch Machine Learning angewendet werden. Im Folgenden werden die Arten von neuronalen Netzen vorgestellt, die für die Klassifizierung in dieser Arbeit verwendet werden. Als Grundlage um Merkmale in Abbildungen zu erkennen werden in allen neuronalen Netzen dieser Arbeit Convolutional Neural Networks verwendet. Die Zahlenerkennung gelang mit nur Convolutional Neural Networks gut. Für handschriftliche Namen werden weitere neuronale Netze verwendet, rekurrente neuronale Netze und anschließend ein Connectionist Temporal Classification Decoder werden in der Verarbeitung nachgeschaltet.

2.1.1 Convolutional Neural Networks

Gezielt für die Verarbeitung von Bildern und Mustern wurde die Klasse der Convolutional Neural Networks (CNNs) entwickelt. Ursprünglich wurden diese Netzwerke in den 1980er Jahren entwickelt [6] und erleben in den letzten Jahren eine Wiederbelebung, getrieben durch die Verfügbarkeit großer Datensätze und die Fortschritte in der Hardware. Der Aufbau der CNNs in der Handschrifterkennung basiert auf den drei folgenden Hauptkomponenten, die in dieser Reihenfolge angewendet werden:

Convolution Layer

Das Herzstück eines Convolutional Layers besteht aus einer Reihe von Filtern oder Kernels, die während des Trainings des Netzes angelernt werden. Diese Filter sind kleine, rechteckige Matrizen von Gewichtungen, die über die Eingabedaten gefaltet werden, um Feature Maps zu erzeugen. Jeder Filter ist darauf ausgerichtet, bestimmte Merkmale, die Features, in den Eingabedaten zu erkennen. Dies können beispielsweise Kanten, Texturen oder definierte Formen sein. Durch die Anwendung verschiedener Filter können Convolutional Layers eine Hierarchie von Merkmalen lernen, die von einfachen, lokalen Mustern bis hin zu komplexen, globalen Strukturen reichen.

Die Faltung, die den Kern der Convolutional Layer bildet, ist eine lokale, gewichtete Summenoperation. Während der Faltung wird der Filter über die Eingabedaten verschoben, und an jeder Position werden die Elemente des Filters mit den entsprechenden Elementen des Eingabedatenausschnitts elementweise multipliziert und summiert. Das Ergebnis dieser Faltung ist eine Feature Map, die die Reaktion des Filters auf bestimmte Merkmale oder Muster in den Eingabedaten darstellt.

Pooling Layer

Nach den Convolution Layern folgen oft Pooling Layer. Ihre Hauptfunktion besteht darin, die räumliche Dimensionalität der Feature Maps zu reduzieren, indem sie die Informationen in den Feature Maps zusammenfassen und die Größe der darin enthaltenen Merkmale

verkleinern. Dies hilft, die Anzahl der Parameter im Netzwerk zu reduzieren und die Berechnungseffizienz zu verbessern, während wichtige Merkmale beibehalten werden.

In dieser Arbeit wird Max-Pooling verwendet. Bei Max-Pooling wird innerhalb eines bestimmten Bereichs der Feature Map der maximale Wert ausgewählt und beibehalten, während alle anderen Werte verworfen werden. Dadurch wird die dominante Aktivierung in diesem Bereich beibehalten und gleichzeitig die Dimensionalität der Feature Map reduziert.

Fully Connected Layer

Fully Connected Layer werden oft am Ende eines Convolutional Neural Networks (CNNs) verwendet, um die extrahierten Merkmale zu klassifizieren oder zu interpretieren. Convolutional Layer und Pooling Layer dienen dazu Merkmale auf verschiedenen Abstraktionsebenen zu extrahieren und zu lokalisieren. Die Fully Connected Layer aggregieren diese Merkmale und führen eine Klassifikation durch.

Nachdem die Eingabedaten durch die Convolutional- und Pooling Layer verarbeitet wurden, werden die Feature Maps in einen Vektor umgeformt, der dann den Fully Connected Layers zugeführt wird. Jede Einheit in einem Fully Connected Layer ist mit jeder Einheit im vorherigen Layer verbunden, wodurch eine vollständige Verbindung entsteht. Diese Struktur ermöglicht es dem Netzwerk, komplexe nichtlineare Beziehungen zwischen den extrahierten Merkmalen zu erlernen und Entscheidungen auf höherer Ebene zu treffen.

Fully Connected Layers werden in dieser Arbeit genutzt, um die extrahierten Merkmale in Klassenlabels umzuwandeln. Bei der Bilderkennung können Fully Connected Layers beispielsweise die Wahrscheinlichkeiten für verschiedene Ziffern angeben.

2.1.2 Rekurrentes neuronales Netz

Rekurrente neuronale Netzwerke (RNNs) dienen zur Verarbeitung von sequenziellen Daten. In diesem Projekt werden rekurrente neuronale Netze für die Handschrifterkennung der Namen verwendet. Im Anwendungsfall der Handschrift sind die sequenziellen Daten die aufeinander folgenden Buchstaben eines Wortes.

RNNs haben eine Feedback-Schleife, die es ihnen ermöglicht, Informationen über vorherige Zeitschritte zu speichern und zu nutzen, um kontextabhängige Vorhersagen[1] zu treffen. Diese vergangene Zeitschritte sind in diesem Zusammenhang die handschriftlichen Zeichen links vom aktuell betrachteten Abschnitt. Das grundlegende Bauelement eines RNNs ist das rekurrente Neuron, das eine interne Zustandsrepräsentation besitzt, die sich mit jedem neuen Zeitschritt ändern kann. Die Berechnungen in einem RNN erfolgen schrittweise für jede Eingabe im Zeitverlauf. Zu jedem Zeitpunkt werden sowohl die Eingabedaten als auch der interne Zustand des Netzwerks verwendet, um eine Ausgabe zu generieren und den neuen Zustand des Netzwerks zu aktualisieren.

2.1.3 Connectionist Temporal Classification

Die Technik der Connectionist Temporal Classification (CTC) wird hier verwendet, um die Sequenzdaten des Rekurrenten neuronalen Netzes zu verarbeiten. Der theoretische Rahmen von CTC basiert auf der Idee der Wahrscheinlichkeitsberechnung von Sequenzen. Im Gegensatz zu traditionellen Ansätzen, bei denen eine genaue Zuordnung zwischen Eingabe und Ausgabe erforderlich ist, erlaubt CTC die Modellierung von Ausgabesequenzen variabler Länge und die Berücksichtigung von Auslassungen und Wiederholungen in den Daten.

Ein Konzept in CTCs ist die Einführung von sogenannten *Blanks* oder Leerzeichen, die dazu dienen, die Möglichkeit von Auslassungen in der Ausgabesequenz zu berücksichtigen. Durch die Berücksichtigung von Leerzeichen können CTC-Modelle Sequenzen variabler Länge korrekt zuordnen und ermöglichen so eine flexible Modellierung von Daten. Diese variable Länge wird in dieser Arbeit benötigt, da Namen unterschiedlich viele Buchstaben haben.

Mit Hilfe des CTC kann der Ausgabe des neuronalen Netzes ein Studentenname zugeordnet werden. Dazu existieren verschiedene Ansätze, die näher in Kapitel 4.1.2 erläutert werden.

[1] Mit Vorhersage ist das Ergebnis gemeint, das durch das neuronale Netz ausgegeben wird.

2.2 Feature Mapping

Die aufgenommenen Deckblätter sind nicht ausgerichtet, daher bedarf es eines Algorithmus, der zuverlässig die Deckblätter zur Verarbeitung orientiert. In dieser Arbeit wird dazu Feature Mapping verwendet. Feature Mapping ist ein Konzept in der Computer Vision, das zur Extraktion und Korrelation von Merkmalen aus Bildern genutzt werden kann. Wenn auf zwei verschiedenen Bilder identische Features gefunden werden, kann das eine Bild zu dem anderen ausgerichtet werden, sodass gleiche Features bei denselben absoluten Koordinaten liegen. Für das Feature Mapping wird hier der ORB-Algorithmus verwendet, welcher aus aus drei grundlegenden Schritten besteht.

2.2.1 ORB-Algorithmus

Im ersten Schritt des ORB-Algorithmus (Oriented FAST and Rotated BRIEF)[15] müssen mögliche Features gefunden werden. Features sind in diesem Fall besondere Bildbereiche. Diese werden mit Hilfe des FAST-Algorithmus (Features from Accelerated Segment Test)[14] entdeckt.
Im zweiten Schritt werden diese Features beschrieben. Dazu wird der BRIEF-Algorithmus (Binary Robust Independent Elementary Features) [3] verwendet. Dieser Algorithmus erstellt Deskriptoren, die die Features beschreiben.
Im letzten Schritt werden Feature-Paare gebildet. Dazu wird verglichen, ob die Deskriptoren von zwei Features eine hohe Ähnlichkeit aufweisen. Anschließend kann eine Homografie Matrix erstellt werden, um das auszurichtende Bild so zu projizieren, dass die Features bei beiden Bildern auf denselben absoluten Koordinaten liegen.

FAST-Algorithmus

Der FAST-Algorithmus [21] iteriert über alle Pixel in einem Graustufenbild. Das jeweils analysierte Pixel wird als *Testpixel* bezeichnet. Ein zentraler Aspekt des FAST-Algorithmus ist ein Schwellwertvergleich. Hierbei werden die Helligkeitswerte des Testpixels mit den Helligkeitswerten seiner benachbarten Pixel auf einem kreisförmigen Muster um das Testpixel herum verglichen, wie in Abbildung 2.1 dargestellt. Der Algorithmus verwendet einen festge-

legten Schwellwert, um zu bestimmen, wie sich die Helligkeitswerte der Nachbarpixel vom Helligkeitswert des Testpixels unterscheiden müssen, damit der Testpixel als Schlüsselpunkt erkannt wird. Der Wert W kann mit der Formel 2.1 ermittelt werden und beschreibt, wie stark sich das Testpixel von den Nachbarpixeln unterscheidet. In der Formel repräsentiert $Kreis(p)$ die Pixel, die einen Kreis mit einem bestimmten Radius r schneiden. In Abbildung 2.1 sind diese Pixel x mit $x \in Kreis(p)$ weiß markiert. Der Wert $I(x)$ beschreibt die Helligkeit des Pixels x.

$$W = \sum_{x \in \{Kreis(p)\}} |I(x) - I(p)| \tag{2.1}$$

Damit der Algorithmus eine Skalierungsinvarianz hat, wird dieser Vorgang für verschiedene Bildgrößen wiederholt. Dazu wird eine Gauß-Pyramide verwendet, siehe Abbildung 2.2. Das hochauflösende Bild zunächst durch einen Gaußschen Weichzeichner geglättet und anschließend herunter skaliert. Durch diesen Vorgang kann ein Feature auch nach einer Skalierung erkannt werden.

Abbildung 2.1: Visualisierung der Pixel für den FAST-Algorithmus, Abbildung aus [15]. Die weißen Zahlen in der rechten Abbildung sind für diese Arbeit ohne Bedeutung.

Abbildung 2.2: Visualisierung der Schritte einer Gaußpyramide. Die Abbildung stammt aus [4].

BRIEF-Algorithmus

Im nächsten Schritt wird durch den BRIEF-Algorithmus [3] das gefundene Feature beschrieben, es wird ein Deskriptor erstellt. Der Deskriptor wird benötigt, um Featurepaare zu bilden. Dazu können die Deskriptoren der Features miteinander verglichen werden, weisen sie eine sehr hohe Ähnlichkeit auf, so sind sie mit hoher Wahrscheinlichkeit die selben Features. Um Deskriptoren zu erstellen, werden um das durch den FAST-Algorithmus gefundene Testpixel

herum Punktepaare gebildet. Für die Punktepaare werden die x und y Koordinaten[2] proportional zu einer Gaußverteilung mit den Parametern $(x,y) \sim Gaussian(0, \frac{1}{25}, S^2)$ gewählt, wobei S die Höhe und Breite des Feldes p ist, aus dem die Paare gewählt werden. Für ein Punktepaar ist eine Testfunktion $\tau(x,y)$ vorhanden:

$$\tau(x,y) := \begin{cases} 1 & \text{falls } I(x) < I(y), \\ 0 & \text{sonst} \end{cases} \tag{2.2}$$

Durch diese Testfunktion kann sogenannter BRIEF-Deskriptor erstellt werden. Aus n_d Pixelpaaren wird ein Bitstring f_{n_d} geformt:

$$f_{n_d} := \sum_{1 \le i \le n_d} 2^{i-1} \tau(x_i, y_i) \tag{2.3}$$

Ein Problem des BRIEF-Deskriptor ist die fehlende Rotationsinvarianz des BRIEF-Algorithmus, es werden keine Informationen über die Rotation gespeichert. Wird eine Eingabe rotiert, ist die Ausgabe des BRIEF-Algorithmus unterschiedlich. Um auch rotierte Features zu erkennen, wird im ORB-Algorithmus der BRIEF-Deskriptor mehrmals berechnet. Dabei wird das zu beschreibende Bild in Abständen von 12° gedreht, es werden entsprechend 30 Deskriptoren berechnet. Diese Anzahl bietet einen bewährten Kompromiss aus Rechenzeit und Speicherbedarf, sowie Robustheit. Die Deskriptoren werden anschließend in einer Lookup-Table gespeichert.

Nachdem auf beiden Bildern Features gefunden und diese als Deskriptoren beschrieben worden sind, können die Features abgeglichen werden. Dazu wird in diesem Projekt ein Brute-Force-Matcher verwendet werden. Der Brute-Force-Matcher ist ein einfaches Verfahren in der Computer Vision. Er nimmt den Deskriptor eines Merkmals im ersten Satz und vergleicht ihn mit allen anderen Merkmalen im zweiten Satz mithilfe der Hamming-Distanz. Anschließend wird das am nächsten liegende Merkmal zurückgegeben. Dieser Vorgang wird für alle Deskriptoren wiederholt. Die Laufzeit dieses Algorithmus ist trotz Brute-Force-Ansatz ausreichend schnell. Das Ausführen des gesamten Feature-Mappings gelingt so schnell, dass das Programm dieses Projektes durch die Bereitstellung neuer Bilder der Kamera limitiert ist und nicht durch die Algorithmen.

[2] In dieser Arbeit wird die waagerechte Achse als X-Achse interpretiert und die senkrechte Achse als Y-Achse

2.2.2 Homografie-Matrix

Nachdem die verschiedenen Features gefunden und gematcht worden sind, wird eine Homografie-Matrix gebildet, um die Feature-Paare auf dieselben, absoluten Koordinaten zu projizieren. Um einen Punkt $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ mit einer Homografie-Matrix H zu projizieren, kann die Formel 2.4 verwendet werden. In der Formel ist die Homografie-Matrix H elementweise dargestellt.

$$\begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} \tag{2.4}$$

Diese Matrix wird mithilfe des RANSAC-Algorithmus [5] berechnet. RANSAC steht für *Random Sample Consensus* und ist ein iterativer Algorithmus, der verwendet wird, um Ausreißer in Daten zu identifizieren und Modelle zu schätzen, wenn die Daten von Ausreißern oder Rauschen beeinflusst werden können. Ausreißer sind im Zusammenhang von Featuremapping falsch gemappte Features.
Die grundlegende Idee hinter RANSAC besteht darin, dass eine zufällige Teilmenge der Daten auswählt wird, ein Modell anhand dieser Teilmenge geschätzt wird und anschließend überprüft wird, wie gut das Modell zu den übrigen Daten passt. Datenpunkte, die gut zum geschätzten Modell passen, werden als Inlier betrachtet, während Datenpunkte, die nicht gut passen, als Outlier betrachtet werden.
Der Algorithmus wiederholt diesen Prozess sehr oft und wählt das Modell aus, das die größte Anzahl von Inlier-Datenpunkten aufweist. Dieses ausgewählte Modell wird dann als das endgültige geschätzte Modell betrachtet.
RANSAC eignet sich gut für Probleme, bei denen eine robuste Schätzung eines Modells erforderlich ist, insbesondere wenn die Daten durch Rauschen oder Ausreißer beeinflusst werden. Diese Eigenschaft wird hier benötigt, um falsch gemappte Features zu ignorieren. Um mit diesem Algorithmus die Homographie-Matrix zu finden, wird der Projektionsfehler E in Formel 2.5 minimiert.

$$E = \sum_{i=1,2} \left(x'_i - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y'_i - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 \tag{2.5}$$

2.3 OpenCV

OpenCV (Open Source Computer Vision Library) [2] ist eine Open-Source-Bibliothek, die für die Bildverarbeitung und Computer Vision entwickelt wurde. Sie bietet eine Vielzahl von Funktionen und Algorithmen für die Verarbeitung von Bildern, von grundlegenden Operationen bis hin zu fortgeschrittenen Techniken. Da das Modul auch in dieser Arbeit zum Einsatz kommt, wird in den folgenden Abschnitten auf die entsprechenden Techniken und Algorithmen eingegangen.

2.3.1 Konturen

Für die Erkennung der Punktebox in diesem Projekt wird auf die Konturenerkennung der OpenCV-Bibliothek verwendet. Der Algorithmus hinter der Konturenerkennung lässt sich durch mehrere essenzielle Schritte beschreiben.

Zu Beginn erfolgt die Bildvorverarbeitung, welche die Transformation des Eingangsbildes in ein Graustufenformat beinhaltet. Eine zusätzliche Glättung des Bildinhaltes mittels Filtern, wie dem Gauß-Filter, trägt zur Rauschunterdrückung bei.
Im Anschluss wird ein Schwellwertverfahren angewendet, um die Pixel in zwei Gruppen zu separieren: Diejenigen über und diejenigen unter einem festgelegten Schwellwert. Diese Binarisierung des Bildes ermöglicht eine klare Unterscheidung zwischen Objekt und Hintergrund.

Die eigentliche Konturenerkennung erfolgt durch Algorithmen, die zusammenhängende Pixelgruppen identifizieren und als Konturen extrahieren. Die in OpenCV verwendete Funktion `findContours` nutzt hierfür einen *Contours-Tracing-Algorithmus*. In dem Algorithmus werden Verbindungen zwischen benachbarten Pixeln betrachtet, um geschlossene Konturen zu formen. Während der Konturenerkennung gibt die Funktion `findContours` nicht nur die Konturpunkte zurück, sondern auch Hierarchieinformationen. Diese Informationen beschreiben die räumliche Struktur der Konturen in der Hierarchie und können beispielsweise Auskunft darüber geben, ob eine Kontur eine innere oder äußere Kontur ist.

2.3.2 Schwellwertverfahren

Schwellwertverfahren können benutzt werden, um den Kontrast von Bildern ins Extreme zu erhöhen. Dazu werden alle Bildpunkte, die über einen Schwellwert liegen, weiß gezeichnet und alle Bildpunkte, die unter einem Schwellwert liegen, schwarz gezeichnet. In dieser Arbeit werden verschiedene Schwellwertverfahren genutzt um die Bildaten für die Weiterverarbeitung bestmöglich zu verändern. Die verschiedenen Methoden unterscheiden sich dabei in der Wahl des Schwellwertes. Dieser kann entweder global für das gesamte Bild gewählt werden, wie beispielsweise bei dem binären Schwellwertverfahren. Er kann auch individuell für bestimmte Bildbereiche, wie im adaptiven Gausschen Schwellwertverfahren, ermittelt werden.

Binäres Schwellwertverfahren

Das binäre Schwellwertverfahren ist ein globales Schwellwertverfahren. Das bedeutet, das ein einziger Schwellwert für alle Bildpunkte eines Bildes verwendet wird. Sofern 0 die minimale mögliche Helligkeit eines Bildpunktes ist und 1 die maximal mögliche Helligkeit, so kann für jeden Bildpunkt g und dem Schwellwert t die Helligkeit T nach Anwendung des Schwellwertverfahrens wie folgt berechnet werden:

$$T_{\text{global}}(g) = \begin{cases} 0 & \text{falls } g < t \\ 1 & \text{falls } g \geq t \end{cases} \tag{2.6}$$

Dieses simple Verfahren bietet einige Vor- und Nachteile. Ein Nachteil ist, dass dieses Verfahren nur unzureichend bei Bildern mit einem Helligkeitsverlauf funktioniert, siehe Abbildung 2.3. Bei kleineren Bildausschnitten ist der gesamte Helligkeitsverlauf meist geringer, wodurch sich das binäre Schwellwertverfahren bei diesen Fällen besser eignet. Ein Vorteil dieses Verfahrens ist die geringe Anzahl an Parametern. Es kann nur ein Schwellwert festgelegt werden, wodurch eine manuelle Einstellung des Parameters sehr einfach ist.

Adaptives Gaussche Schwellwertverfahren

Das adaptive Gaussche Schwellwertverfahren benutzt nicht einen Schwellwert für alle Bildpunkte, sondern berechnet für jeden Bildpunkt an den Position (x,y) einen eigenen Schwellwert. Dieser Schwellwert hängt von den Bildpunkten in der quadratischen Nachbarschaft mit einer Kantenlänge von Größe N ab:

$$T_{\text{dynamisch}}(x,y) = \begin{cases} 0 & \text{falls } g(x,y) < t(F(x,y)) \\ 1 & \text{falls } g(x,y) \geq t(F(x,y)) \end{cases} \tag{2.7}$$

Die Funktion F repräsentiert eine gewichtete Summe über die Nachbarschaft für einen Bildpunkt, minus einer Konstante C:

$$F(x,y) = -C + \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{(i-\frac{N}{2})^2+(j-\frac{N}{2})^2}{2\sigma^2}} \cdot \text{Pixel}(x+i, y+j) \tag{2.8}$$

mit der in Open-CV definierten Verteilungsbreite

$$\sigma = 0.3 \cdot ((N-1)0.5-1) + 0,8 \tag{2.9}$$

Dieses Verfahren bietet den Vorteil, dass es für Bilder mit einem Helligkeitsverlauf geeignet ist, siehe Abbildung 2.3. Details bleiben erhalten, sofern Sie sich von der Nachbarschaft signifikant genug unterscheiden. Es werden somit mehr Details erhalten als bei dem globalen Schwellwertverfahren. Ein Nachteil bei diesem Verfahren ist jedoch die hohe Anzahl an Parametern. Die Nachbarschaftsgröße kann angepasst werden, es können manuell Werte für σ angegeben werden sowie die Auswahl einer konstante C.

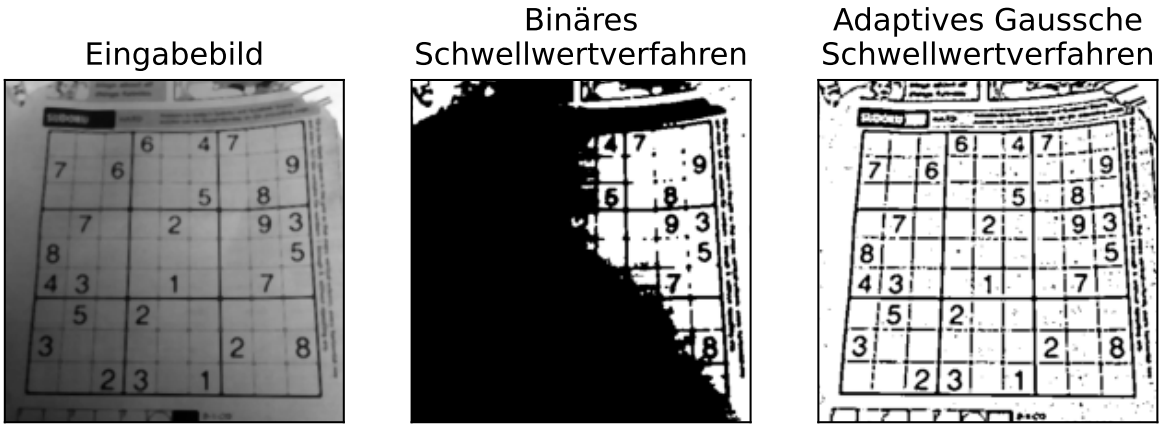


Abbildung 2.3: Vergleich der beiden Schwellwertverfahren. Links ist das unverarbeitete Bild, in der Mitte das Bild nach Anwendung des binären Schwellwertverfahrens mit dem Schwellwert $0,5$ und rechts das Bild nach Anwendung des adaptiven Gaussche Schwellwertverfahren. (eigene Abbildung)

2.4 Levenshtein-Distanz

Mit der Levenshtein-Distanz kann die Ähnlichkeit zweier Texte bestimmt werden. Diese Fähigkeit ist nützlich, wenn zum Beispiel in diesem Projekt geprüft werden soll, ob eine vorhergesagte Matrikelnummer ähnlich genug zu der eines Studenten ist. Die Levenshtein-Distanz zwischen zwei Texten wird als ganze Zahl beschrieben, je niedriger dieser Wert ist, desto ähnlicher sind die beiden Texte.

Die Levenshtein-Distanz kann durch dynamische Programmierung mit einer Matrix D bestimmt werden. Dabei sind u und v die beiden Eingangstexten mit den Längen $m = |u|$ und $n = |v|$. Entsprechend wird die Matrix die Größe $(m+1) \times (n+1)$ haben. Die Matrix kann zu Beginn mit folgenden Werten gefüllt werden:

$$D_{0,0} = 0 \tag{2.10}$$

$$D_{i,0} = i, 1 \leq i \leq m \tag{2.11}$$

$$D_{0,j} = j, 1 \leq j \leq n \tag{2.12}$$

Anschließend können die restlichen Elemente durch folgende rekursive Formel berechnet werden:

$$D_{i,j} = \min \begin{Bmatrix} D_{i-1,j-1} + 0, \text{ falls } u_i = v_j \\ D_{i-1,j-1} + 1 \\ D_{i,j-1} + 1 \\ D_{i-1,j} + 1 \end{Bmatrix} \tag{2.13}$$

Diese Formel kann so interpretiert werden, dass bei zwei gleichen Buchstaben $u_i = v_j$ eines Wortes die Levenshtein-Distanz nicht steigt. Der Term $D_{i-1,j-1} + 1$ beschreibt die Ersetzung eines Buchstabens, der Term $D_{i,j-1} + 1$ das Einfügen und der Term $D_{i-1,j} + 1$ die Löschung eines Buchstaben. Ein Beispiel zur Berechnung der Levenshtein-Distanz in in Tabelle 2.1 dargestellt.

	ε	**2**	**8**	**1**	**1**	**3**	**1**	**9**
ε	0	1	2	3	4	5	6	7
2	1	0	1	2	3	4	5	6
6	2	1	1	2	3	4	5	6
1	3	2	2	1	2	3	4	5
8	4	3	2	2	2	3	4	5
1	5	4	3	2	2	3	3	4
3	6	5	4	3	3	2	3	4
1	7	6	5	4	3	3	2	3

Tabelle 2.1: Beispiel zur Berechnung der Levenshtein-Distanz zwischen den beiden Matrikelnummern 2811319 und 2618131. Die Levenshtein-Distanz beträgt in diesem Fall 3 und kann unten rechts abgelesen werden.

Kapitel 3

Detektion relevanter Bereiche auf dem Deckblatt

Die Machine-Learning-Modelle müssen die handschriftlichen Texte auf dem Deckblatt, siehe Abbildung 1.1, mit möglichst hoher Präzision vorhersagen können. Dazu müssen geeignete Eingangsdaten bereitgestellt werden. Dies bedeutet, dass das Modell idealerweise einen Ausschnitt erhält, der ausschließlich die Handschrift enthält, ohne dass zusätzlicher gedruckter Text die Vorhersage beeinflusst. Darüber hinaus sollte der Text möglichst passend ausgeschnitten werden, da große weiße Ränder um den Text herum zu einer schlechteren Vorhersage führen können. Mit einem breiten weißen Rand sind weniger relevante Informationen auf der Abbildung vorhanden, die zur Vorhersage genutzt werden können.

Um solche gut geeigneten Ausschnitte zu extrahieren, wurden verschiedene Möglichkeiten verglichen. Einerseits wurden Kontureneigenschaften der Handschrift genutzt, um diese auf dem Deckblatt zu erkennen, andererseits wurde Feature-Mapping für diesen Vorgang eingesetzt.

Die Feature-Mapping-Methode erwies sich als effektiver als die alleinige Verwendung der Kontureneigenschaften. Ihre Anwendung zeigte sich zudem als robuster, insbesondere gegenüber Rotationen der Eingangsbilder. Aus diesem Grund wurde daher das Feature Mapping in der finalen Version genutzt.

3.1 Detektion durch Kontureneigenschaften

In einer frühen Version des Projekts wurden Eigenschaften der Konturen der Handschrift genutzt. Zuerst wurde das Deckblatt grob in einige Bereiche aufgeteilt, da die Punktetabelle meist im unteren Drittel des Blattes zu finden ist. Dieser untere Bereich kann für einen Algorithmus verwendet werden, der die Punktzahlen aus der Tabelle extrahiert. Der eingesetzte Algorithmus ist robust genug, um die einzelnen Punkte aus einem relativ großen Ausschnitt zu extrahieren, wie später in Kapitel 3.3 näher erläutert wird.

Um die Namen und Matrikelnummern durch Eigenschaften der Konturen zu erkennen, wurde, wie in Kapitel 2.3 beschrieben, auf die Open-CV zurückgegriffen. Die Linie, auf der die Namen und Matrikelnummern geschrieben wurden, hat eine signifikant längere Breite als andere Konturen, wie in Abbildung 3.1 dargestellt. Wenn eine Kontur mit einer horizontalen Länge entdeckt wird, die länger als ein vorgegebener Schwellwert ist, kann davon ausgegangen werden, dass die Kontur der Linie entspricht, auf der handschriftlich etwas eingetragen wird. Durch die relativen Positionen zueinander kann herausgefunden werden, wo der Vorname steht, wo der Nachname steht und über welcher Kontur die Matrikelnummer niedergeschrieben ist.

Diese Methodik birgt einige Nachteile. Je nach Auflösung und Qualität der Kamera kann es dazu führen, dass lange Wörter als Kontur interpretiert werden. Deshalb ist es möglich, nur Teilbereiche des Deckblatts zu verwenden. Zum Beispiel werden der Nachname und die Matrikelnummer in dem zweiten vertikalen linken Viertel eingetragen und der Vorname auf der entsprechend rechten Seite. Mit dieser Lösung gehen jedoch Nachteile einher. Das Deckblatt muss unter der Kamera möglichst gleich ausgerichtet werden, da ansonsten die Felder nicht mehr in den entsprechenden Bereichen vorhanden sind und somit nicht gefunden werden können. Darüber hinaus muss die relative Position der Felder für jedes Klausurlayout einzeln konfiguriert werden.

Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Dr. Markus Brenneis

08. Februar 2022

Hauptklausur

Programmierung
WS 21/22

Nachname: _____ Vorname: _____

Matrikelnummer: _____ Sitzplatznummer: _____

Zusätzliche Blätter: _____ Unterschrift: _____

Hinweise:

- Diese Klausur enthält 20 nummerierte Klausurseiten. Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält. Sie dürfen die Heftung der Klausur nicht auftrennen.
- Sie erhalten außerdem von uns leere Blätter. Sollten Sie auf diesen Blättern für die Korrektur relevante Teile Ihrer Lösung notieren, markieren Sie dies deutlich an der entsprechenden Aufgabe und auf dem zusätzlichen Blatt. Schreiben Sie auf alle zusätzlichen Blätter Ihren Namen. Geben Sie außerdem oben auf dem Deckblatt an, wie viele zusätzliche Blätter Sie zur Korrektur abgeben. Wenn Sie weiteres Papier benötigen, melden Sie sich bitte.
- Alle Fachbegriffe in dieser Klausur werden wie in der Vorlesung definiert verwendet. Alle Fragen beziehen sich auf die in der Vorlesung vorgestellte Java-Version 11. Programmcode muss in Java geschrieben werden.
- Antworten dürfen auf Deutsch oder Englisch gegeben werden. Sofern keine ausformulierten Sätze verlangt sind, reichen nachvollziehbare Stichworte als Antwort.
- Zugelassene Hilfsmittel: eine beidseitig beschriebene oder bedruckte DIN-A4-Seite, Wörterbuch (Wörterbücher müssen vor Beginn der Klausur den Aufsichtspersonen zur Kontrolle vorgelegt werden.)
- Schalten Sie Ihr Mobiltelefon aus. Täuschungsversuche führen zum sofortigen Ausschluss von der Klausur. Die Klausur wird dann als nicht bestanden gewertet.
- Schreiben Sie **nicht** mit radierbaren Stiften und auch **nicht** mit rot!

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	9	Σ
Punktzahl	7	5	2	13	13	8	9	6	27	90
Erreicht										

1

Abbildung 3.1: Das Deckblatt einer Klausur. Die Daten zur Identifikation der Studenten werden auf die waagerechten Linien geschrieben, die Punkte werden in der Tabelle eingetragen.

3.2 Bildausrichtung durch Feature Mapping

Eine weitere Option zur Extraktion der Handschrift vom Deckblatt ist das Feature Mapping. Dazu wird zunächst mit einem Configurator-Programm des Projekts ein Referenzbild von einem leeren Deckblatt aufgenommen. Dieses Deckblatt sollte leer sein, damit auf der Handschrift keine Features gefunden werden, die nicht auf einem anderen Deckblatt vorhanden sind. Die Anzahl an geeigneten Features erhöht sich also. Anschließend können relevante Bereiche wie Vorname, Nachname, Matrikelnummer und die Punktetabelle eingezeichnet werden. Die absoluten Koordinaten dieser Bereiche werden zusammen mit dem aufgenommenen Bild gespeichert. Das gespeicherte Bild dient so als Referenzbild.

Um nun ein anderes, beschriebenes Deckblatt auszurichten, werden auf dem Referenzbild sowie auf dem ausgefüllten Deckblatt Features nach dem in Kapitel 2.2 beschriebenen FAST-Algorithmus gesucht. Die Features werden durch den BRIEF-Algorithmus beschrieben und können anschließend durch den Brute-Force-Matcher Algorithmus gematcht werden. Die Matches können beispielsweise wie in Abbildung 3.2 aussehen, gematchte Features sind dort durch Linien miteinander verbunden.

Sofern ausreichend viele Features zugeordnet werden können, wird eine affine Transformation auf dem ausgefüllten Deckblatt durchgeführt. Die Koordinaten der relevanten Bereiche wurden zuvor auf dem Referenzbild bestimmt. Nach der Transformation befinden sich diese relevanten Bereiche auf dem ausgefüllten Deckblatt an den gleichen Koordinaten, sodass sie ausgeschnitten werden können.

Durch diesen Prozess können die Bereiche, in denen zum Beispiel der Name oder die Matrikelnummer steht, sehr präzise extrahiert werden. Der Algorithmus funktioniert in der Praxis sehr gut. Wenn ein Großteil des Deckblattes unter der Kamera liegt, kann die Transformation durchgeführt werden.

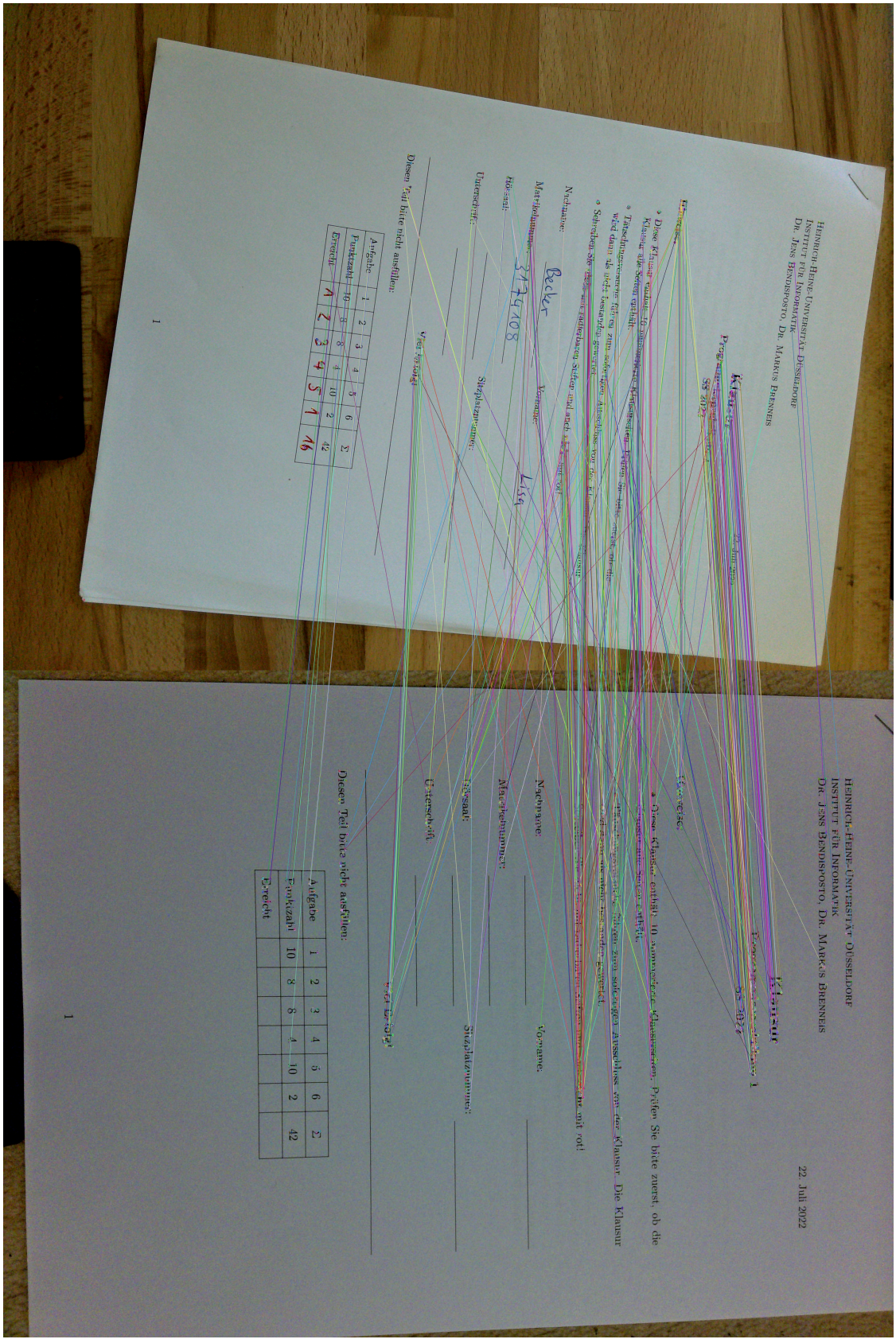
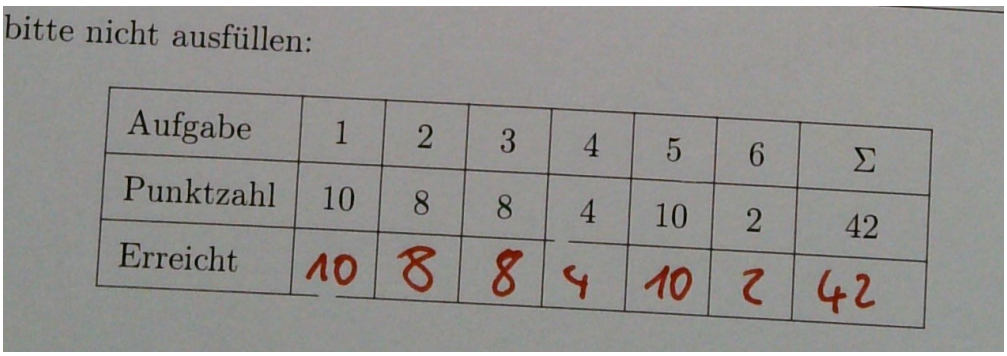


Abbildung 3.2: Feature Mapping, Zuordnung der Features des zu lesenden Blatts, oben, zur Referenz, unten.

3.3 Lokalisierung der Punkte

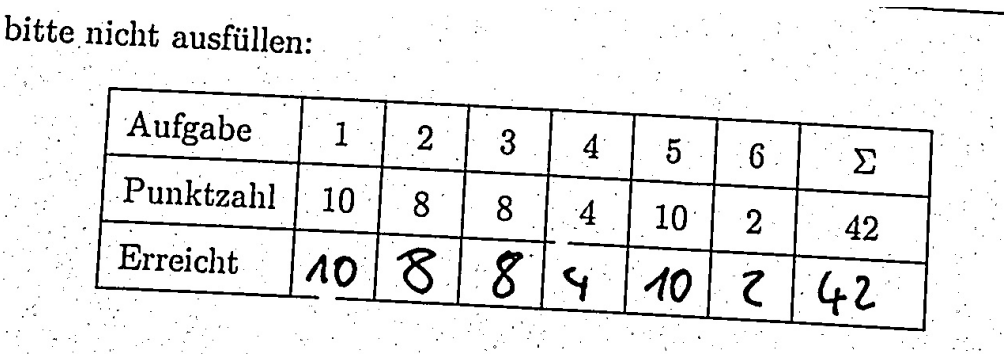
Um die eingetragenen Ergebnisse, kurz Punkte, zu ermitteln, muss zunächst die Punktetabelle lokalisiert werden. Dazu wird ein Bildausschnitt des ungefähren Bereichs verwendet, wie zum Beispiel in Abbildung 3.3. Dieses Bild ist ein Farbbild und die gesamte Punktetabelle ist abgebildet. Die Punktetabelle kann, wie dargestellt, auch verkippt sein oder leichte Defekte aufweisen. Defekte können in diesem Fall nichtdurchgängige Linien sein. Solche Fälle sind in Abbildung 3.3 in der Spalte der Aufgabe 1 und der Aufgabe 4 zu sehen, wo jeweils eine Linie unterbrochen ist.


bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	Σ
Punktzahl	10	8	8	4	10	2	42
Erreicht	10	8	8	4	10	2	42

Abbildung 3.3: Eingangsbild für den Algorithmus der die Punktzahlen extrahiert. Es sind Defekte in der Spalte mit Aufgabe 1 und Aufgabe 4 vorhanden.

Zunächst wird für folgende Konturenerkennung der Kontrast des Eingangsbilds erhöht. Dies geschieht durch das adaptive Gaussche Schwellwertverfahren. Die Konturenerkennung von Open-CV funktioniert auf dem kontrastreicheren Ergebnisbild, welches in Abbildung 3.4 dargestellt ist, besser.


bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	Σ
Punktzahl	10	8	8	4	10	2	42
Erreicht	10	8	8	4	10	2	42

Abbildung 3.4: Eingangsbild nach Konstrasterhöhung durch einen adaptiven Schwellwert-Filter.

Anschließend wird die Box mit der Punktetabelle waagerecht ausgerichtet. Dazu werden Konturen gesucht, die ein bestimmtes Höhen-Seitenverhältnis aufweisen und in einer Liste gespeichert. Der Parameter für das Höhen-Seitenverhältnis wird zu Beginn des Programms eingegeben. Anschließend wird der Medianwert dieser Flächeninhalte bestimmt. Dies dient dazu, die reale Fläche einer Box zu ermitteln, ohne dass dieser Wert durch andere Konturen verfälscht wird, die zufällig ein ähnliches Höhen-Seitenverhältnis besitzen.

Der Flächenwert wird anschließend verwendet, um die Kontur der gesamten Tabelle zu ermitteln. Diese ist 20- bis 80-mal größer und kann zuverlässig über den Flächeninhalt gefunden werden. Dieser Vorgang ist sehr robust und funktioniert auch bei verschiedenen Auflösungen der Kamera, ohne dass das Programm neu konfiguriert werden muss.

Nachdem die Kontur der gesamten Tabelle erkannt wurde, wird die Tabelle waagerecht ausgerichtet. Dazu wird die Kontur so gedreht, dass die Fläche eines umschließenden Rechtecks minimiert wird. Dieser Vorgang erleichtert spätere Koordinatenbestimmungen, da alle Kästchen in einer Zeile nahezu genau die gleiche Y-Koordinate haben und die Spalten die gleiche X-Koordinate. Die rotierte Tabelle ist in Abbildung 3.5 dargestellt.

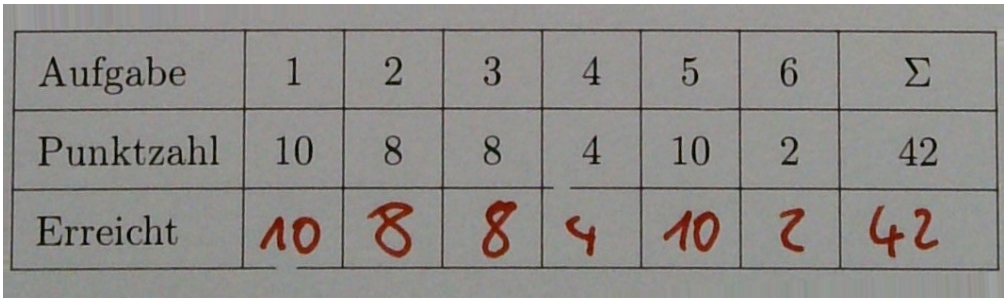


Abbildung 3.5: Rotiertes Eingangsbild, die Tabelle ist ausgerichtet.

Als nächstes werden die kleineren Boxen den Zeilen und Spalten zugeordnet. Dazu wird durch das Gaussche Schwellwertverfahren der Kontrast des rotierten Bildes erhöht. Es werden wieder Konturen gesucht, die das definierte Höhen-Seitenverhältnis und die per Algorithmus bestimmte Medianfläche aufweisen. Diese gefundenen Konturen werden nun in Spalten aufgeteilt, indem sie nach ihren X-Koordinaten sortiert werden. Wenn die aufeinander folgenden Koordinaten der Boxen sich nicht signifikant unterscheiden, werden sie derselben Spalte zugeordnet, ansonsten werden sie der nächsten Spalte zugeordnet. Die Konturen in jeder Spalte werden dann nach ihrer Y-Koordinate sortiert. Nach diesem Vorgang liegt eine zweidimensionale Liste vor, welche die Tabelle repräsentiert.

Es kann vorkommen, dass in einer Spalte nicht alle Boxen direkt erkannt werden. Da die Boxen jedoch nahezu identische Größen haben, können die Koordinaten von nicht erkannten Boxen annähernd berechnet werden. Wenn die oberste Box einer Spalte erkannt wurde, können die Koordinaten einer fehlenden Box in dieser Spalte berechnet werden. Da die Punktetabelle waagerecht ausgerichtet wurde, ist die X-Koordinate der gefundenen Box in dieser Spalte für die restlichen Boxen in dieser Spalte gleich. Lediglich die Y-Koordinate unterscheidet sich. Diese kann durch die Boxhöhe und die Dicke der Ränder einer Box berechnet werden. Die erkannten Boxen sind in Abbildung 3.6 visualisiert. In der Spalte mit Aufgabe 4 wurden beispielsweise zwei Boxen rekonstruiert.

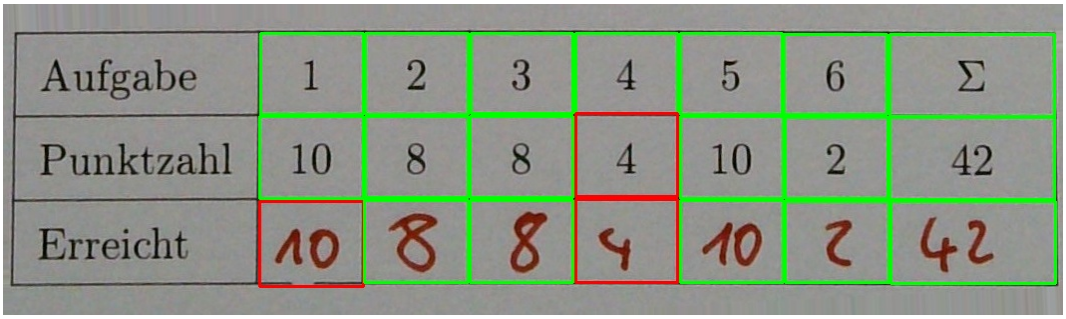


Abbildung 3.6: Erkannte Boxen des Eingangsbilds. Felder mit grünen Boxen wurden direkt erkannt, bei den roten Boxen wurden Daten rekonstruiert, sodass die Felder mit Handschrift detektiert werden. Die Feld mit *Aufgabe*, *Punktzahl* und *Erreicht* wurden nicht detektiert, da sie nicht das korrekte Höhen-Seiten Verhältnis aufweisen.

Nachdem die Boxen detektiert worden sind, können die Bildbereiche mit handschriftlichen Punkten ausgeschnitten und weiter bearbeitet werden. Die Handschrift ist ein der untersten Spalte. Diese Bildausschnitte sehen beispielsweise wie in Abbildung 3.7 aus. Durch die vorherige Aufteilung der Boxen in Spalten und Zeilen kann jedem Ausschnitt eindeutig eine Aufgabe zugeordnet werden.

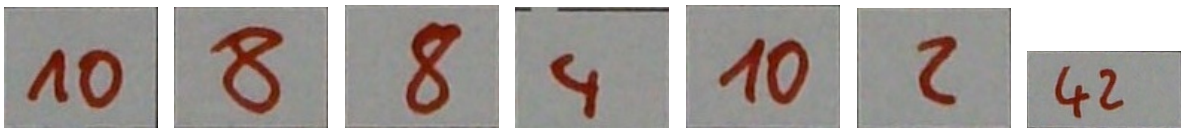


Abbildung 3.7: Ausgeschnittene Boxen mit den jeweiligen erreichten Punktzahlen der Aufgaben.

Um die handschriftlichen Punktzahlen mithilfe von Machine Learning vorherzusagen, werden die Bildausschnitte mit den Punktzahlen nachbearbeitet. Zunächst werden alle farbigen Bildausschnitte in Schwarz-Weiß umgewandelt. Nach diesem Prozess wird ein binäres Schwellwertverfahren angewandt. Der Schwellwert kann dynamisch im Programm angepasst werden, um bei verschiedenen Beleuchtungen und Schriftdicken einen geeigneten Wert bereitzustellen. Durch visuelles Feedback erkennt der Nutzer, ob ein geeigneter Schwellwert gewählt wurde.

Nach der Schwellwertanpassung wird der Bildausschnitt mit den Punktzahlen weiter zugeschnitten. Dazu wird zunächst der weiße Rand des Ausschnitts an allen Seiten entfernt. Nun ist ein Bild vorhanden, in dem die handschriftliche Zahl jeden Rand berührt. Anschließend wird entweder links und rechts oder oben und unten ein jeweils gleich dicker, weißer Rand hinzugefügt, sodass das Bild ein Höhen-Seitenverhältnis von eins besitzt. Zuletzt wird das Bild so skaliert, dass es die vom Machine Learning Model benötigte Auflösung besitzt. Die Bildausschnitte sehen nach der Bearbeitung wie in Abbildung 3.8 dargestellt aus.

Abbildung 3.8: Die zugeschnittenen Punktzahlen mit Anwendung eines binären Schwellwerts. Es wurde ein schwarzer Rahmen um das Bild gelegt, um die Dimensionen zu verdeutlichen.

Kapitel 4

Vorhersage der relevanten Informationen

Es existieren zwei verschiedene Arten der Handschrifterkennung mit Machine Learning: die sogenannte *online* Erkennung und die *offline* Erkennung. Der Begriff *online* und *offline* bezieht sich dabei auf die Art der bereitgestellten Daten.

Um Online-Daten von Handschrift zu erhalten, muss das Schreiben des Wortes aufgezeichnet werden, beispielsweise mithilfe von Tablets oder Kameras. Bei der *online* Handschrifterkennung sind so zusätzliche Informationen verfügbar, insbesondere die Reihenfolge, in der der Text erstellt wurde. Diese Reihenfolge kann beispielsweise durch Vektorpfeile wie in Abbildung 4.1 dargestellt werden. Diese zusätzlichen Daten liefern relevante Informationen, die ein Modell nutzen kann, um eine höhere Genauigkeit zu erzielen.

Bei der *offline* Handschrifterkennung liegt lediglich ein Bild des geschriebenen Textes vor. Bei der Handschrift auf den Deckblättern ist genau dies der Fall. Die Klausuren werden Aufgabe für Aufgabe korrigiert und nacheinander physisch auf Papier niedergeschrieben. Dadurch ist es nicht möglich den Vorgang des Schreibens aufzunehmen. Das Projekt bekommt als Eingabe die bereits beschriebenen Deckblätter, welche *offline* Handschriftdaten enthalten. Die Konvertierung von *offline* Handschriftdaten in *online* Handschriftdaten ist nicht möglich, es können keine Informationen im nachhinein zugefügt werden.

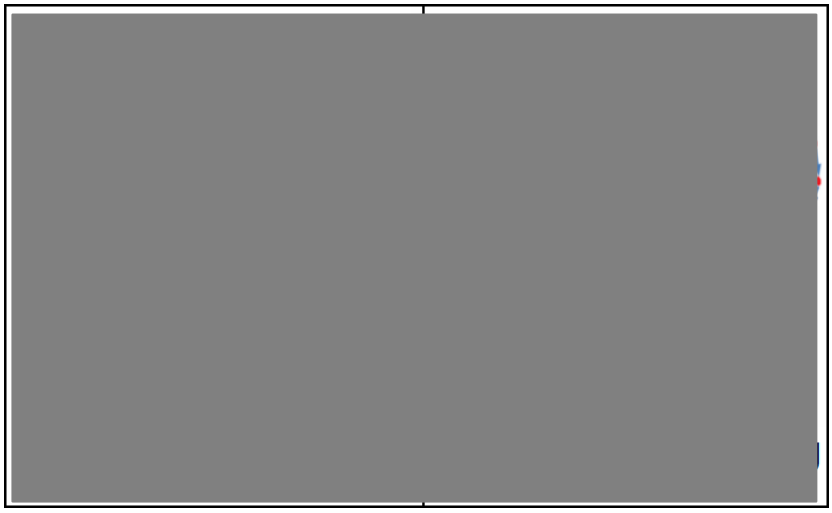


Abbildung 4.1: Links *offline* Daten für Handschrift, rechts *online* Daten für die gleiche Handschrift. Die Abbildung stammt aus [13].

4.1 Namenserkennung

Als Modul zur Erkennung handgeschriebener Namen auf den Deckblättern wird SimpleHTR verwendet [16]. SimpleHTR ist ein Projekt zur *offline*-Erkennung handgeschriebener Texte. Dazu wird ein Machine-Learning-Modell sowie ein angepasster CTC-Decoder verwendet. SimpleHTR ist in Python geschrieben und verwendet TensorFlow v1 für das Machine Learning.

Um handgeschriebene Daten mit SimpleHTR zu erkennen, werden die Eingabedaten zunächst bearbeitet. Das Machine-Learning-Modell hat eine Eingabegröße von 128 mal 32 Pixeln. Die Eingabedaten werden auf diese Größe skaliert, indem das Eingabebild so vergrößert oder verkleinert wird, dass entweder die Länge 128 Pixel oder die Höhe 32 Pixel beträgt und der andere Wert entsprechend kleiner ist. Die fehlenden Bereiche des Bildes werden anschließend mit weißen Pixeln aufgefüllt. Danach wird das Bild in Graustufen umgewandelt.

Das Machine-Learning-Modell von SimpleHTR ist, wie in Abbildung 4.2 dargestellt, aufgebaut. Es besteht zunächst aus fünf Convolutional Neural Network (CNN) Layern. Die ersten beiden Layer haben eine Kernelgröße von fünf mal fünf, und die drei folgenden Layer haben eine Kernelgröße von drei mal drei. Nach jedem dieser fünf Layer wird eine nichtlineare ReLU-Funktion angewendet. Diese Funktion ermöglicht die Extraktion nichtlinearer

Eigenschaften, was besonders bei der Erkennung von Handschrift wichtig ist. Handschriften weisen oft komplexe Muster auf, die nicht durch lineare Beziehungen beschrieben werden können. Anschließend wird ein Pooling Layer verwendet, um nicht benötigte Dimensionen zu reduzieren. Dieser gesamte Vorgang wird insgesamt fünfmal mit verschiedenen Kernelgrößen und zunehmender Feature Map Größe wiederholt. Im Anschluss wird ein rekurrentes neuronales Netzwerk (RNN) angewendet, das aus zwei Layern mit jeweils 256 Features pro Schritt besteht.

Schlussendlich folgt ein CTC, um aus der Ausgabe des Netzes Namen vorherzusagen. In dem Projekt ist es möglich, zwischen drei verschiedenen CTC Modellen zu wählen: einem *Best-Path-Decoder*, einem *Beamsearch-Decoder* und dem *CTC Word Beam Search Decoding Algorithm*. Darüber hinaus wurde im Rahmen dieser Arbeit ein Lexicon-Search Decoder implementiert.

Abbildung 4.2: Aufbau des Modells aus SimpleHTR. Die Abbildung stammt aus [16].

4.1.1 Datensatz

Ein gut geeigneter Datensatz für das Training des Handschriftmodells von SimpleHTR muss mehrere wichtige Eigenschaften erfüllen. Zunächst sollte der Datensatz eine große Anzahl an Wörtern enthalten. Ein weiter Umfang an Daten stellt sicher, dass allgemeine Eigenschaften der Handschrift gelernt werden können und weniger Overfitting stattfindet. Dabei ist auch die Vielfalt der Schriften entscheidend, da ein Datensatz, der nur von einer Person geschrie-

bene Wörter enthält, einen begrenzten Schrifstil repräsentiert. Ein Datensatz mit Beiträgen verschiedener Autoren hingegen bietet eine größere Vielfalt an Schriftstilen und ermöglicht es dem Modell, allgemeinere Merkmale der Handschrift zu erfassen.

Die Qualität der Daten ist ebenfalls wichtig. Die Trainingsdaten sollten eine Ähnlichkeit mit den später zu erkennenden Namen haben. Daten mit chinesischen Schriftzeichen oder anderen nicht relevanten Inhalten eignen sich nicht für das Training. Es ist auch wichtig zu beachten, dass einzelne Buchstaben möglicherweise nicht ausreichen, um ein robustes Modell zu trainieren, da die Kontextinformationen, die durch die Reihenfolge der Buchstaben bereitgestellt werden, fehlen können. Daher ist es ratsam, Wörter oder sogar ganze Sätze als Trainingsdaten zu verwenden, um ein realistischeres Szenario der Handschrifterkennung zu simulieren.

Es gibt nur wenige umfangreiche Datensätze, die einen Großteil dieser charakteristischen Merkmale aufweisen. Ein Beispiel für einen solchen Datensatz ist die Arbeit von Smith et al. [10], welche eine Sammlung von 5000 geschrieben amerikanischen Städten und 5000 geschriebenen Staaten umfasst. Neben seiner Umfangsbeschränkung in Bezug auf die Textvielfalt, bietet dieser Datensatz nur begrenzte Eignung für das Training von Modellen, da er lediglich 10.000 verschiedene Wörter enthält.

Ein weiterer relevanten Datensatz ist die *NIST Special Database 19* [20]. Diese Datenbank besteht aus handgeschriebenen Texten von 3600 Autoren und umfasst insgesamt 800.000 Buchstaben und Zahlen. Die Vielzahl der Autoren stellt einen Vorteil dar, der für das Training von Modellen relevant ist. Jedoch ist ein signifikanter Nachteil dieses Datensatzes die Tatsache, dass er ausschließlich einzelne Buchstaben und Zahlen enthält, anstatt vollständige Wörter. Angesichts der Notwendigkeit, in diesem Projekt ganze Namen zu erkennen, ist es ein großer Nachteil.

Eine Lösung wäre die Buchstaben der Wörter auf dem Deckblatt einzeln zu segmentieren und sie dann nach der Erkennung durch das Modell zu einem Wort zusammenzusetzen. Diese Segmentierung gestaltet sich allerdings sehr schwierig. Außerdem bieten Modelle, die mit ganzen Wörtern arbeiten, wie zum Beispiel welche die auf Recurrent Neural Networks (RNNs) basieren, einige Vorteile und erkennen die Namen mit einer höheren Wahrscheinlichkeit.

Eine alternative Strategie besteht darin, synthetische Wörter zu generieren, indem die einzelnen Buchstaben aneinandergesetzt werden. Jedoch führt dieser Ansatz zu unnatürlich aussehenden Wörtern, wie in Abbildung 4.3 dargestellt. In diesen synthetischen Wörtern sind verschiedene Schriftarten und -stile vorhanden, und die Buchstaben weisen unterschiedliche Dicken auf, was dazu führt, dass eine Schreibschrift nicht authentisch wiedergegeben werden kann. Aufgrund dieser Nachteile wird dieser Datensatz nicht verwendet.

Abbildung 4.3: Auf der linken Seite das Wort *Meeting* aus der *IAM Handwriting Database*, auf der rechten Seite das Wort *Meeting* synthetisch erzeugt, aus Buchstaben der *NIST Special Database 19*.

Die IAM Handwriting Database [12] in der Version 3.0 stellt einen weiteren verbreiteten Datensatz für handgeschriebene Wörter dar. Diese Datenbank ist in zwei Varianten verfügbar, einmal mit ganzen Textzeilen, die aus mehreren Wörtern bestehen, und einmal mit einzelnen Wörtern. Die enthaltenen Texte stammen von 657 verschiedenen Autoren und umfassen insgesamt 115.320 einzelne Wörter. Im Vergleich zum zuvor erwähnten Datensatz [10] ist der Umfang dieser Datenbank erheblich größer und vergleichbar mit der NIST Special Database 19. Aufgrund dieser großen Menge an Trainingsdaten besteht theoretisch die Möglichkeit, ein qualitativ hochwertiges Modell zu entwickeln. Des Weiteren zeichnet sich die IAM Handwriting Database durch die gute Qualität ihrer Daten aus, da sie aus vollständigen Wörtern besteht, die weder segmentiert noch zusammengefügt werden müssen. Die Datenbank enthält verschiedene Schriftstile, darunter Schreib- und Druckschrift. Zudem weisen die enthaltenen Wörter aus englischen Texten ausreichend Ähnlichkeit mit den zu erkennenden Namen auf. Obwohl deutsche Sonderzeichen wie ä, ö und ü fehlen, können diese aufgrund der geringen optischen Unterschiede durch a, o und u ersetzt werden.

Aufgrund des umfangreichen Datensatzes und der hochwertigen Datenqualität wurde die IAM Handwriting Database für das Training des Handschrifterkennungsmodells ausgewählt.

4.1.2 CTC-Decoder für SimpleHTR

Ein CTC-Decoder übersetzt die Ausgabe des neuronalen Netzes in diesem Fall in lesbaren Text. In dem SimpleHTR Projekt sind drei CTC-Decoder implementiert, ein *best path decoding Decoder*, ein *vanilla beam search Decoder* und ein *word beam search Decoder*. Diese drei Decoder eignen sich nur bedingt für den hier benötigten Einsatzzweck, da sie nicht auf ein vordefiniertes Wörterbuch beschränkt sind. Das Wörterbuch ist in diesem Zusammenhang die Namensliste der Teilnehmer. Es können fälschlicherweise Sonderzeichen oder Zahlen erkannt werden, die möglicherweise keine relevanten Namen darstellen. Um diese Problem zu lösen, wurde eine Schnittstelle zu einem *lexicon search* Decoder implementiert. Dieser Decoder ist auf Wörter aus einem Wörterbuch beschränkt. Dadurch eignet er sich hervorragend für den vorliegenden Einsatzzweck, da nur Namen aus einem vordefinierten Namensverzeichnis erkannt werden müssen.

4.2 Nummernkennung

Im europäischen und amerikanischen Raum werden Buchstaben ähnlich geschrieben, jedoch gibt es Unterschiede bei der Darstellung von Ziffern. Insbesondere weichen die Eins und die Sieben im amerikanischen Schriftstil von ihren europäischen Entsprechungen ab, wie in den Abbildungen 4.4 und 4.5 dargestellt. In der europäischen Schulausgangsschrift wird die Eins mit zwei Strichen geschrieben, wobei der obere rechte Strich einen Winkel bildet. Im amerikanischen Schriftstil hingegen ist die Eins lediglich ein nahezu senkrechter Strich. Der Hauptunterschied zwischen der europäischen und amerikanischen Sieben liegt in dem mittleren waagerechten Strich.

Abbildung 4.4: Schulausgangsschrift aus Deutschland. Die Abbildung stammt aus [19].

Abbildung 4.5: Schulausgangsschrift aus den USA, Abbildung abgeändert aus [8].

Diese Unterschiede können dazu führen, dass eine europäisch geschriebene Eins leicht mit einer amerikanisch geschriebenen Sieben verwechselt werden kann, da beide die gleiche Anzahl von Strichen aufweisen. Nach einer geringfügigen Drehung sind die beiden Ziffern kaum zu unterscheiden, was die Erkennung erschwert. Dies stellt ein Problem bei der Identifizierung von handgeschriebenen Daten dar, da die Auswahl des korrekten Datensatzes für die Erkennung bestimmter Ziffern entscheidend ist.

In den Abbildungen 4.6 und 4.7 sind die Ziffern Eins und Sieben dargestellt. Es ist zu erkennen, dass die Eins aus Abbildung 4.6 der Sieben aus Abbildung 4.7 ähnelt.

Abbildung 4.6: Links die Ziffer Eins aus dem MNIST Datensatz, rechts die Ziffer Eins aus dem Repository Numbers [17].

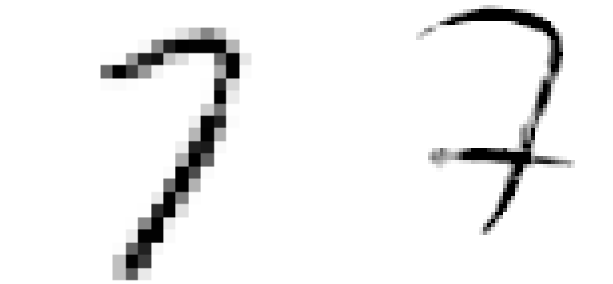


Abbildung 4.7: Links die Ziffer Sieben aus dem MNIST Datensatz, rechts die Ziffer Sieben aus dem Repository Numbers [17].

4.2.1 Datensatz

Der MNIST-Datensatz (Modified National Institute of Standards and Technology) [11] ist ein etablierter Datensatz für Ziffernerkennung, der 60.000 Trainingsbilder und 10.000 Validierungsbilder umfasst. Da dieser Datensatz in den USA erstellt wurde, entsprechen die enthaltenen Ziffern dem dortigen Schrifstil, wie in Abbildung 4.5 dargestellt. Bei der Verwendung dieses Datensatzes besteht die Herausforderung darin, dass das Modell häufig Schwierigkeiten hat, die Ziffern Eins und Sieben im europäischen Schriftstil zu unterscheiden, insbesondere aufgrund der ähnlichen Darstellung in diesem Schrifstil. Aus diesem Grund wurde die Suche nach einem alternativen Datensatz mit Ziffern im europäischen Stil vorgenommen.

Eine Datenbank mit einer großen Anzahl von handgeschriebenen Ziffern im europäischen Stil ist im Repository Numbers [17] verfügbar. Diese Datenbank enthält 16.394 handgeschriebene Ziffern, die dem Stil entsprechen, wie in Abbildung 4.4 dargestellt. Zusätzlich enthält das Repository über 800.000 weitere Ziffern, jedoch sind diese nicht vollständig korrekt gelabelt. Es gibt auch fehlerhafte Bilder sowie einige gedruckte anstelle von handgeschriebenen Ziffern. Des Weiteren sind die Ziffern nicht gleichmäßig verteilt, wobei die Null mit 323.945 Bildern am häufigsten vorkommt und die Neun mit 34.571 Vorkommen am seltensten ist. Trotzdem ist der Anteil ungeeigneter Daten gering, sodass die Verwendung des umfangreicheren Datensatzes mit einer größeren Vielfalt an verschiedenen Ziffern zu einer deutlichen Verbesserung der Modellleistung führt.

4.2.2 Punktzahlerkennung durch Machine Learning

Die Punktzahlenerkennnung wurde für zwei verschiedene Deckblattvorlagen entwickelt. Eine Version ist für Klausuren, bei denen nur ganze Punkte erreicht werden können. Dabei umfassen die möglichen Punktzahlen alle ganzen Zahlen von 0 bis 99. In der Version für Klausuren, bei denen halbe Punkte vergeben werden können, wird vorher eine Segmentierung der Zahlen zu Ziffern durchgeführt, sodass beliebige Punktzahlen in 0,5er-Schritten erkannt werden können. Für dieses System müssen die Punktzahlen korrekt in eine angepasste Tabelle, siehe Abbildung 4.8 eingetragen werden. Die Strukturen der neuronalen Netze ist bei den beiden Systemen prinzipiell gleich. Lediglich das Output Layer wurde angepasst. Bei dem Netzwerk für die ganzen Zahlen von 0 bis 99 existieren 100 Output-Neuronen, bei dem Netzwerk für Dezimalzahlen existieren entsprechend 10 Output-Neuronen für eine Ziffer.

Aufgabe	1	2	3	4	5	6	7	8	Summe
Punkte	19	7	18	6	7	12	5	26	100
Erreicht									

Abbildung 4.8: Modifizierte Punktetabelle für Punkte als Dezimalzahlen mit einer Nachkommastelle. Es sind drei Zellen für die erreichten Punkte jeder Aufgabe vorhanden und entsprechend vier Zellen für das Ergebnis, sofern es mindestens 100 zu erreichende Punkte gibt.

Die Abbildungen 4.9 und 4.10 zeigen die Struktur des eigens erstellten neuronalen Netzes. Das vorliegende neuronale Netzwerk folgt einer sequenziellen Architektur, die zur Bilderkennung und Klassifizierung entwickelt wurde. Es beginnt mit einem Eingangslayer, auf die CNN-Layer folgen. Diese Layer sind für das Extrahieren von Merkmalen aus den Eingangsbildern zuständig. Das erste CNN-Layer besteht aus 64 Filtern mit einer Größe von 3x3 und nutzt die ReLU-Aktivierungsfunktion, um Nichtlinearitäten zu behandeln.

Anschließend folgt ein weiteres CNN-Layer mit denselben Eigenschaften, um weitere Merkmale zu identifizieren. Nach jedem CNN-Layer wird ein Max Pooling Layer verwendet, welche die räumliche Dimension der Aktivierungen reduziert und damit die Rechenlast verringert. Das Muster setzt sich fort, wobei die Anzahl der Filter in den CNN-Layern erhöht wird: von 128 zu 256 und schließlich zu 512.

Nach den CNN-Layern wird die Ausgabe des letzten Max Pooling Layer in einen flachen Vektor umgeformt. Dieser Vektor wird dann einer vollständig verbundenen Layer mit 512 Neuronen zugeführt, die die Merkmale aggregiert und die Dimensionalität reduziert.

Abschließend wird die Ausgabe dieses Layers über ein weiteres Dense Layer mit zehn Neuronen, beziehungsweise 100 Neuronen geleitet. Es wird eine Softmax Aktivierungsfunktion verwendet, um Wahrscheinlichkeiten für die zehn, beziehungsweise 100 verschiedenen Klassen zu erzeugen.

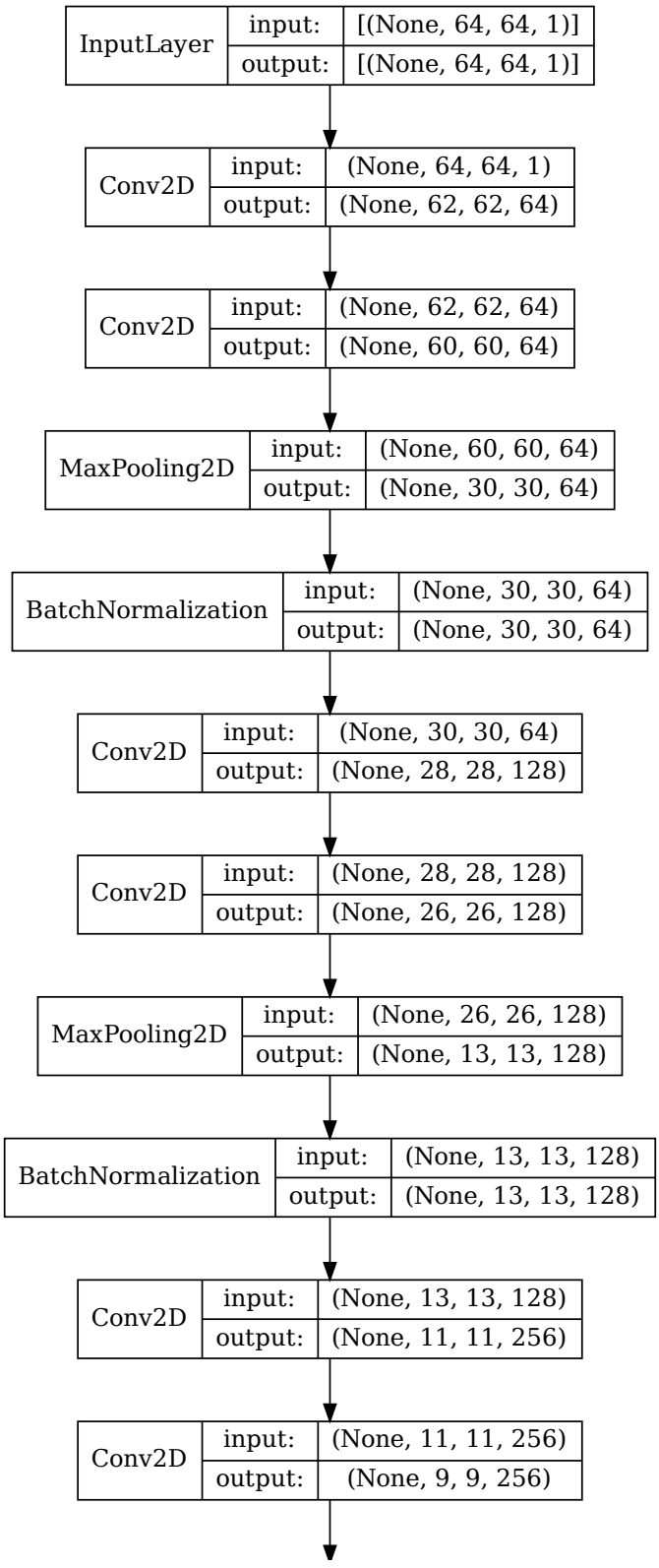


Abbildung 4.9: Erste Hälfte des Aufbaus des neuronalen Netzes. Auf der linken Seite jedes Layers steht der Typ, auf der rechten Seite die Eingabedimensionen und die Ausgabedimensionen.

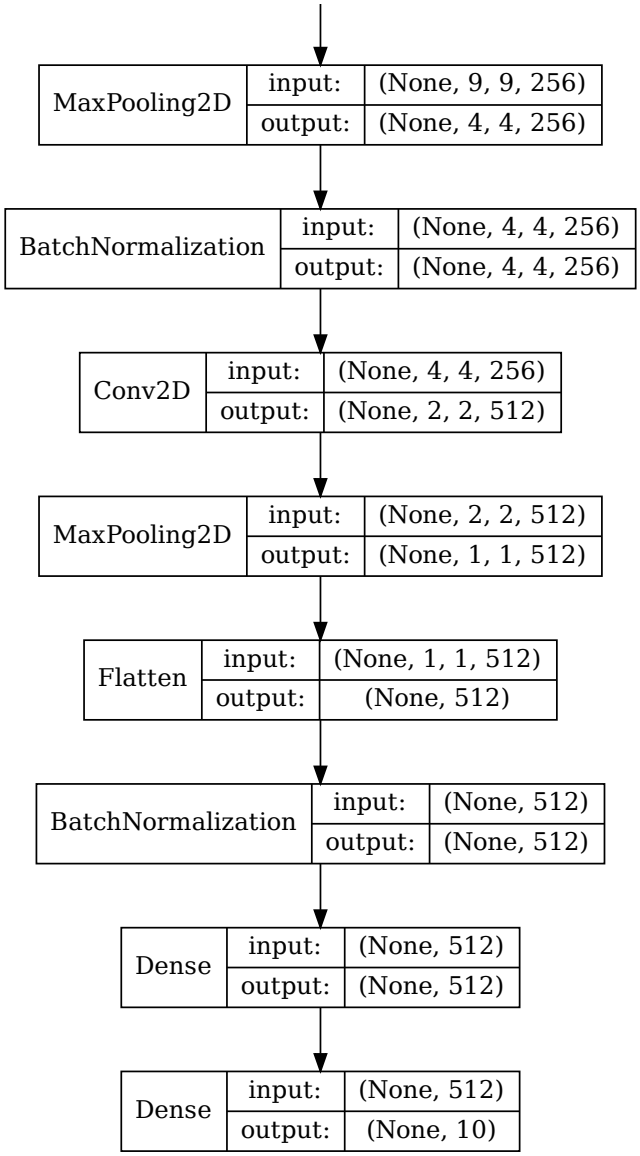


Abbildung 4.10: Zweite Hälfte des Aufbaus des neuronalen Netzes. Auf der linken Seite jedes Layers steht der Typ, auf der rechten Seite die Eingabedimensionen und die Ausgabedimensionen.

Punktzahlen als ganze Zahlen von 0 bis 99

Eine Option dieses Projekts besteht darin, ganze Zahlen im Bereich von 0 bis 99 zu erkennen. Zu diesem Zweck wurde das neuronale Netzwerk, wie im vorherigen Abschnitt beschrieben, mit 100 Ausgabeneuronen verwendet.

Für das Training des neuronalen Netzwerks werden Trainingsdaten benötigt, die den zu prognostizierenden Bildern möglichst ähnlich sind. Ausgehend vom Datensatz [17] wurden daher synthetische Zahlen im Bereich von 0 bis 99 erstellt. Diese Zahlen sind nach dem europäischen Schriftstil konzipiert. Dabei wurde die größere Datensatz an Ziffern ausgewählt, die möglicherweise gedruckte Zahlen sowie einige falsch kategorisierte Ziffern enthält. Obwohl in diesem Datensatz potenziell fehlerhafte Daten vorhanden sind, erwies sich ihre Verwendung in der Praxis als vorteilhafter als die Verwendung des kleineren Datensatzes mit fehlerfreier Sortierung und weniger fehlerhaften Ziffern. Dies liegt an der höheren Variabilität der Daten, die es dem Modell ermöglicht, länger zu trainieren, ohne Overfitting zu erleiden. Letztendlich führt die Variabilität der Trainingsdaten zu einer höheren Erkennungsrate bei den tatsächlichen Daten.

Im folgenden Abschnitten wird das Verfahren zur Generierung der Trainingsbilder mit zweistelligen ganzen Zahlen beschrieben. Um diese Bilder zu erstellen, wird zunächst eine Zufallszahl innerhalb des entsprechenden Bereichs generiert. Anschließend werden die Bilder der ersten und zweiten Ziffer aus dem Datensatz als Graustufen-Bilder geladen, wie in Abbildung 4.11 dargestellt. Dabei entspricht der hellste mögliche Pixelwert 255 und der dunkelste Wert ist 0.

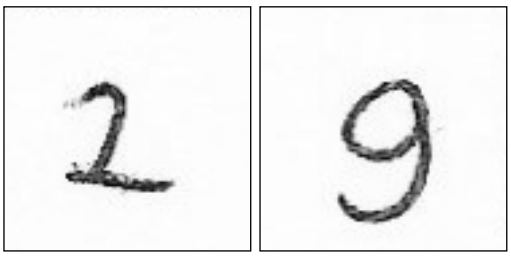


Abbildung 4.11: Zwei Abbildungen aus dem Datensatz [17] als Graustufen-Bilder.

Im nächsten Schritt wird ein binäres Schwellwertverfahren angewendet, wobei der Schwellwert auf 70 festgelegt wird. Dies bedeutet, dass alle Bildpunkte mit einer Helligkeit unter 70 auf den Wert 0 gesetzt werden, während alle anderen Bildpunkte auf den Wert 255 gesetzt werden. Nach dieser Operation sieht das vorherige Bild wie in Abbildung 4.12 aus.

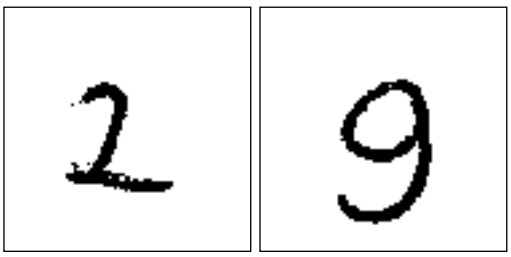


Abbildung 4.12: Die Ziffern nach Anwendung des binären Schwellwertverfahrens.

Die Abbildungen im Datensatz stammen aus verschiedenen Quellen und haben möglicherweise unterschiedliche Auflösungen. Um sicherzustellen, dass die Bilder die Dimensionen des Input Layers des Machine-Learning-Modells erfüllen, werden sie skaliert. Das Beispielbild nach dem Skalieren auf 64 mal 64 Pixel ist in Abbildung 4.13 dargestellt.

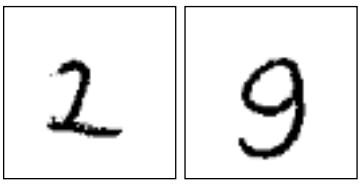


Abbildung 4.13: Die Ziffern nach dem Skalieren auf die Eingangsdimension des Netzwerkes.

Um aus den einzelnen Ziffern zweistellige Zahlen zu bilden, wird die Ziffer, die die Zehnerstelle beschreibt, nach links verschoben, während die Einer-Ziffer nach rechts verschoben wird, damit die Ziffern nebeneinander stehen. Die Weite der Verschiebung beträgt eine zufällige Distanz zwischen einem Viertel und einem Achtel der Bildgröße. Nach diesem Vorgang sehen die Ziffern wie in Abbildung 4.14 dargestellt aus.

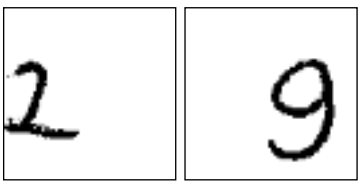


Abbildung 4.14: Die beiden verschobenen Ziffern.

Zuletzt müssen die beiden Bilder der Ziffern zu der zweistelligen Zahl kombiniert werden. Hierzu wird ein neues Bild erstellt, wobei die beiden einzelnen Abbildungen überlagert werden, siehe Abbildung 4.15.

Abbildung 4.15: Das finale Bild, wie es zum Trainieren des Netzes genutzt werden kann.

Punktzahlen als Dezimalzahlen mit Nachkommastellen

Für Klausuren, bei denen auch halbe Punkte erreicht werden können, wird eine angepasste Punktetabelle verwendet, wie in Abbildung 4.8 dargestellt. Dabei sind die Felder für die Punktzahl durch zwei oder drei vertikale gestrichelte Linien geteilt. Beim Eintragen der erreichten Punkte sollte ganz rechts jeweils eine 5 eingetragen werden, sofern ein halber Punkt erreicht wurde. In die übrigen Felder wird die Zahl rechtsbündig eingetragen, sodass jeweils eine Ziffer in einem Feld steht. Dies ermöglicht eine Segmentierung, wodurch die einzelnen Ziffern ausgeschnitten und einzeln vorhergesagt werden können. Das Machine-Learning-Modell muss somit lediglich Zahlen im Wertebereich von 0 bis 9 klassifizieren. Zu diesem Zweck wurde ein Modell mit der gleichen Architektur wie zuvor beschrieben erstellt. Lediglich das Output Layer wurde angepasst und auf zehn Kategorien reduziert.

Die hierzu benötigten Trainingsdaten werden aus demselben Datensatz wie bei den ganzen Zahlen entnommen. Das Data-Preprocessing unterschiedet sich allerdings, da unterschiedliche Zahlenabbildungen für das Training des Models benötigt werden. In diesem Fall werden lediglich einstellige Zahlen vorhergesagt, die mehrstelligen Zahlen sind in Boxen aufgeteilt. Daher werden einzelne Ziffern, entsprechend einer Box, aus dem Datensatz geladen und anschließend daraus entfernt, um sicherzustellen, dass keine Bilder doppelt verwendet werden. Ein Beispiel für ein solches Bild ist in Abbildung 4.16 dargestellt.

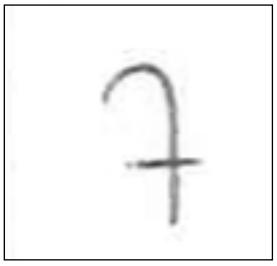


Abbildung 4.16: Schwarz weiß Bild aus dem Datensatz [17].

Im nächsten Schritt wird ein binäres Schwellwertverfahren angewendet, wobei der Schwellwert auf 70 festgelegt wird. Das bedeutet, dass alle Bildpunkte mit einer Helligkeit unter 70 auf den Wert 0 gesetzt werden, während alle anderen Bildpunkte auf den Wert 255 gesetzt werden. Das resultierende Bild sieht dann beispielsweise wie in Abbildung 4.17 aus.

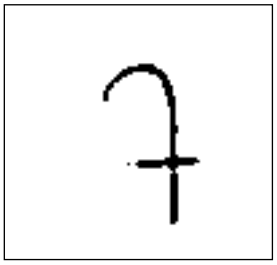


Abbildung 4.17: Die Ziffer nach Anwendung der binären Schwellwertverfahrens.

Anschließend wird bei der Hälfte der Bilder durch Dilatation die Dicke einer Ziffer vergrößert. Dabei wird per Zufallsgenerator bestimmt, ob und wie stark eine Kontur dilatiert wird. Nach einer Dilatation sieht das Bild dann beispielsweise wie in Abbildung 4.18 aus.

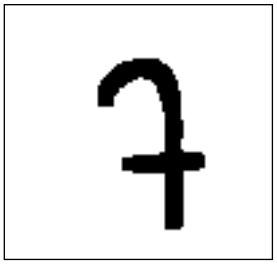


Abbildung 4.18: Die Ziffer nach Dilatation.

Im nächsten Schritt wird das Bild der Ziffer zugeschnitten. Dazu werden alle Zeilen und Spalten entfernt, die komplett weiß sind. Dieser Vorgang wird an jedem Rand des Bildes wie-

derholt, bis jeder Rand mindestens ein nicht weißes Pixel enthält. Das Bild sieht anschließend wie in Abbildung 4.19 dargestellt aus.

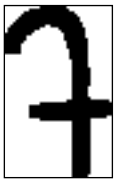


Abbildung 4.19: Die Ziffer nachdem das Bild zugeschnitten wurde.

Im Anschluss wird die Dimension des Bildes angepasst, sodass sie mit der Eingangsdimension des Machine-Learning-Modells übereinstimmt. Zunächst wird das Bild skaliert, sodass eine Seitenlänge mit der Eingangsdimension übereinstimmt. Danach wird ein Rand hinzugefügt, sodass sowohl die Höhe als auch die Breite alle Bedingungen erfüllen. Nach diesen Anpassungen sieht das Bild wie in Abbildung 4.20 dargestellt aus.

Abbildung 4.20: Skalierung und Anpassung der Ziffer.

Training des Machine Learning Modells

Die so erzeugten Bilder werden anschließend normiert, sodass der maximal mögliche Wert 1 und der minimal mögliche Wert eines Bildpunktes 0 beträgt. Durch die vorher beschriebenen Vorgänge werden für zweistellige Zahlen insgesamt 1.000.000 Bilder von Zahlen mit entsprechenden Labels erstellt. Dabei werden keine Abbildungen aus dem Datensatz entfernt, da sonst nur eine geringere Anzahl an Zahlen möglich wäre. Wird eine zweistellige Zahl erstellt, so wird die erste und zweite Ziffer aus den ca. 800.000 Ziffern großen Datensatz gewählt. Die Wahrschienlichkeit, dass zweimal dieselbe Kombination gewählt wird, ist gering. Falls dies doch der Fall ist, ist die Verschiebung der Ziffer wahrscheinlich unterschiedlich, wodurch es nahezu unmöglich ist, deckungsgleiche Abbildung zu erhalten.

Für die Dezimalzahlen werden nach der Erstellung einer Abbildung die Zahlen aus dem Datensatz entfernt, sodass keine doppelten Abbildungen vorhanden sind. So werden über 345.000 verschiedene Abbildungen erzeugt.

Für das Training des beschriebenen neuronalen Netzes wurden 90% des Datensatzes als Trainingsdaten verwendet, während die restlichen 10% des Datensatzes als Validierungsdaten verwendet wurden. Aufgrund der hohen Anzahl an Trainingsdaten genügen bereits zwei bis drei Epochen, um ein sehr gutes Modell zu erhalten. Die Fehlerrate bei der Vorhersage der Validierungsdaten beträgt bei den zweistelligen Zahlen etwa 3%. Weiteres Training verbessert diese Rate nicht mehr. Dies liegt daran, dass etwa 3% der Daten fehlerhaft sind, beispielsweise aufgrund zu geringer Verschiebung der Ziffern, wodurch sie durch Überlagerung unleserlich werden. Bei den einstelligen Zahlen liegt die Fehlerrate bei weit unter 1% und kann nahezu alle Validierungsdaten korrekt vorhersagen.

4.2.3 Matrikelnummererkennung durch Machine Learning

Matrikelnummern bestehen an der HHU aus sieben verschiedenen Ziffern. Das neuronale Netz zur Vorhersage von Matrikelnummern hat eine gleiche interne Struktur wie die Modelle für Punktzahlen. Lediglich die Ausgabelayer wurde für die Matrikelnummern angepasst. Das letzte Dense Layer auf 7 mal 10 Klassen erweitert und anschließend in sieben Klassen mit jeweils 10 Unterklassen umgeformt. Dabei entspricht jede Klasse einer Ziffer der Matrikelnummer, wobei es 10 verschiedene Möglichkeiten, also Unterklassen für jede Ziffer gibt.

Das Training des neuronalen Netzes erfordert Trainingsdaten, die teilweise synthetisch generiert werden. Dazu dient der große Datensatz aus dem Repository [17] als Grundlage, aus dem synthetische Matrikelnummern erstellt werden. Die Erzeugung dieser Matrikelnummern wird im Folgenden beschrieben.

Zunächst werden sieben Ziffern zufällig aus dem Datensatz ausgewählt. Diese Ziffern können verschiedene Auflösungen und Schreibstile aufweisen. Die Abbildungen liegen als Farbbilder vor und werden zuerst in Graustufenbilder umgewandelt. Anschließend werden sie durch ein binäres Schwellwertverfahren angewandt. Die ausgewählten Ziffern für eine Matrikelnummer können nach diesem Schritt wie in Abbildung 4.21 aussehen.

Abbildung 4.21: Schwarz weiß Bilder aus dem Datensatz [17]. Die verschiedenen Ziffern haben unterschiedliche Auflösungen.

Nachdem die einzelnen Ziffern zufällig ausgewählt wurden, müssen möglicherweise vorhandene Ränder an der linken und rechten Seite entfernt werden. Dazu werden Spalten entfernt, die nur weiße Pixel enthalten. Anschließend können die einzelnen Ziffern wie in Abbildung 4.22 aussehen.

Abbildung 4.22: Die Ziffern nach dem Zuschneiden.

Nach dem Zuschneiden der Abbildungen müssen diese auf eine einheitliche Größe skaliert werden, damit sie später aneinander gefügt werden können. Dabei wird die Höhe der Abbildung der vertikalen Eingabedimension des neuronalen Netzes angepasst. Die Breite wird entsprechend proportional skaliert. Nach diesem Vorgang haben alle Abbildungen die gleiche Höhe, aber möglicherweise unterschiedliche Breiten, wie in Abbildung 4.23 dargestellt.

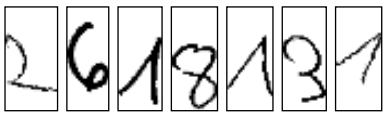


Abbildung 4.23: Die Ziffern nach dem Skalieren.

Der nächste Schritt dient als Vorbereitung, um die einzelnen Ziffern zu einer Matrikelnummer zusammenzufügen. Dazu wird links von den Abbildungen jeweils ein weißer Rand hinzugefügt, damit die Bilder anschließend überlagert eine Matrikelnummer ergeben. Die Breite der ersten Ziffer bleibt dabei unverändert. An den linken Rand der zweiten Ziffer wird ein weißer Bereich eingefügt, dessen Breite der Breite der ersten Ziffer entspricht. Dieser Vorgang wird für alle übrigen Ziffern wiederholt, bis die Abbildungen wie in Abbildung 4.24 aussehen.

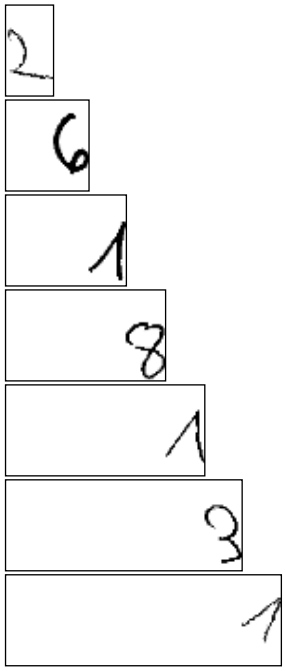


Abbildung 4.24: Anpassung der Breite der Ziffern, sodass sie zusammengefügt werden können. Die erste Ziffer der Matrikelnummer ist das oberste Bild in der Abbildung.

Im folgenden Schritt werden die Abbildungen zu einer Matrikelnummer zusammengefügt. Die einzelnen Abbildungen können als Matrizen interpretiert werden. Dadurch ist es möglich, die Matrikelnummerabbildung als das Maximum aller entsprechenden Bildpunkte der einzelnen Abbildungen zu ermitteln. Es entsteht eine Abbildung mit der gesamten Matrikelnummer, wie beispielsweise in Abbildung 4.25 dargestellt.

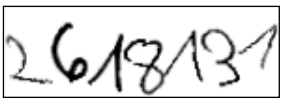


Abbildung 4.25: Die zusammengefügten Ziffern.

Das Machine-Learning-Netz benötigt feste Eingabedimensionen. Die Höhe der Abbildung wurde in einem vorherigen Schritt bereits angepasst. Um die Breite des neuronalen Netzes zu erfüllen, wird links und rechts von der Abbildung ein Rand hinzugefügt. Es ist zufällig, welcher Anteil von Rand links und rechts hinzugefügt wird. Nach diesem Schritt kann eine Matrikelnummer wie in Abbildung 4.26 aussehen.

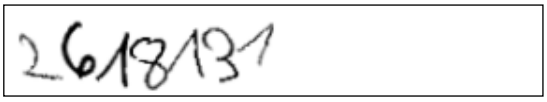


Abbildung 4.26: Anpassung an die für das Machine Learning benötigte Auflösung.

Auf den Klausurdeckblättern werden die Matrikelnummern auf eine horizontale Linie geschrieben. Damit das neuronale Netz Matrikelnummern in Bildern mit dieser Linie vorhersagen kann, wird auch in den Trainingsdaten eine Linie hinzugefügt. Diese hat eine zufällige Dicke und Neigung. Sie ist stets im unteren Bereich des Bildes gezeichnet. Die Matrikelnummer nach dem Hinzufügen der Linie kann wie in Abbildung 4.27 aussehen.

Abbildung 4.27: Die Trainingsdaten nach dem Hinzufügen der Linie zur Matrikelnummer.

Als letzter Schritt wird dem Bild der Matrikelnummer etwas Noise hinzugefügt, damit das neuronale Netz resistent gegen unsaubere Bilder ist. Es wird eine zufällige Anzahl von schwarzen Punkten mit einer zufälligen Dicke über das Bild verteilt. Das endgültige Bild für das Training des neuronalen Netzes kann wie in Abbildung 4.28 aussehen.

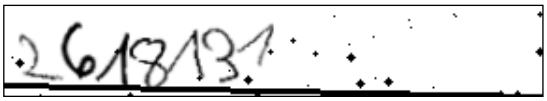


Abbildung 4.28: Die Trainingsdaten nach dem Hinzufügen von Noise.

Die so erzeugten Bilder werden normiert, sodass der höchstmögliche Bildwert 1 beträgt und der kleinste mögliche Wert 0 ist. Für das Training des neuronalen Netzes wurden 1.000.000 Matrikelnummern nach dem vorher beschriebenen Schema samt Label erstellt. Diese Anzahl an Matrikelnummern genügt um auch Matrikelnummern vorherzusagen, die nicht in den Trainingsdaten vorhanden sind. Das liegt an der Struktur des neuronalen Netzes mit den 7 Klassen und 10 Unterklassen. Es sollte im Training jede Ziffer an jeder Stelle vorkommen. Dadurch können im Anschluss auch neue Kombinationen von Ziffern erkannt werden.

Jede einzelne Ziffer des Datensatzes kann mehrmals verwendet werden, da die Wahrscheinlichkeit, zwei identische Matrikelnummern zu erhalten, sehr gering ist. Als Testdaten dienen 90% der Daten, die restlichen 10% werden als Validierungsdaten verwendet. Nach drei Trainingsepochen beträgt die Fehlerrate 17%, eine signifikante Verbesserung wird durch weiteres Training nicht erzielt. Die Fehlerrate bedeutet, dass von den 700.000 Ziffern der 100.000 Validierungs-Matrikelnummern des Validierungsdatensates 119000 Ziffern falsch erkannt wurden.

Dieser Wert erscheint im ersten Moment hoch. Allerdings kann es zu verschiedenen Fehlern bei der Vorhersage kommen, beispielsweise wenn eine Ziffer nicht erkannt wurde und die restlichen Ziffern sich um jeweils eine Position verschieben. Die berechnete Fehlerrate ist in diesem Fall hoch, jedoch wurde nur eine Zahl fehlerhaft erkannt. In dieser Arbeit werden die vorhergesagten Matrikelnummern per Levinsthein-Distanz mit realen Matrikelnummern abgeglichen. Bei diesem Vorgang wäre die Distanz bei einer nicht erkannten Ziffer gering. Die Matrikelnummer kann korrekt zugordnet werden, obwohl die berechnete Fehlerrate hoch wäre.

4.3 QR-Codes

Aufgrund der Fehlerquote in den Praxistests bei der Erkennung handschriftlicher Matrtikelnummern und Namen wurden QR-Codes als Alternative getestet. Diese Informationen können auch als QR-Code codiert auf der Klausur vorhanden sein. In QR-Codes ist eine Fehlerkorrektur vorhanden.
Für die Verarbeitung von QR-Codes wird eine bereits bestehende Bibliothek von OpenCV verwendet. Aus der Klasse *QRCodeDetector* wird die Funktion *detectAndDecodeMulti* verwendet. Diese Funktion erhält als Eingabeparameter einen Bildausschnitt, in dem der QR-Code enthalten ist. Die Funktion erkennt alle QR-Codes in dem Bildausschnitt. Als Rückgabeparameter liefert sie einen Boolean-Wert, der angibt, ob mindestens ein QR-Code decodiert wurde, einen String mit dem decodierten Text sowie Bildinformationen über den entdeckten QR-Code.

4.4 Fehlerkorrektur der erkannten Werte

Die neuronalen Netze in diesem Projekt erkennen Handschriften nur mit einer gewissen Wahrscheinlichkeit. Dementsprechend kann es passieren, das beispielsweise eine Zahl falsch erkannt wird und statt einer Eins eine Sieben vorhergesagt wird. Solche Fehler sollen nach Möglichkeit vermieden werden und dürfen nicht in die Ergebnisliste eingetragen werden. Um die Fehlerrate zu minimieren, werden verschiedene Prüfungen und Algorithmen angewandt, die im Folgenden erläutert werden.

4.4.1 Namen und Matrikelnummer

Die korrigierten Klausuren werden in der Regel alphabetisch vorsortiert. Dies ist eine große Hilfe für der Fehlerkorrektur, da es die möglichen Namen auf dem Blatt stark einschränkt. Es wird jedoch davon ausgegangen, dass einige Klausuren nicht korrekt alphabetisch sortiert sind, sondern mit alphabetisch benachbarten Klausuren vertauscht sind. Aus diesem Grund wird eine definierte Anzahl von Studenteneinträgen zur Namensauswahl bereitgestellt, ein Sliding-Window. Dadurch sinkt die Fehlerate signifikant, da der Pool von möglichen Namen geringer ist.

Ein weiterer Faktor, um die Fehlerrate zu vermindern ist der Ansatz, dass Vor- und Nachname erkannt werden müssen. Sofern Vor- und Nachname zueinander passen, steigt die Wahrscheinlichkeit erheblich, dass der Student korrekt identifiziert wurde. Das neuronale Netz sagt den niedergeschriebenen Vor- oder Nachnamen mit einer höheren Wahrscheinlichkeit vorher, als falsche Vor- oder Nachnamen.

Geht man davon aus, die Teilliste mit potzentiellen Namen 20 Studenten umfasst und die Wahrscheinlichkeit einen niedergeschriebenen Namen korrekt zu erkennen bei 70 % liegt, so liegt die Wahrscheinlichkeit, dass der richtige Student identifiziert wird sofern Vor- und Nachname zueinander passen bei über 99 %.

Um die Fehlerrate weiter zu verringern, kann ein zusätzlich vorhergesagter Wert zugezogen werden, die Matrikelnummer. Sofern ein Student durch den vorhergesagten Vor- und Nachnamen identifiziert wurde, kann zusätzlich nun geprüft werden, ob die erkannte Matrikelnummer der des Studenten entspricht. Alle Ziffern der Matrikelnummer können durch das

neuronale Netz nur selten korrekt vorhergesagt werden. In den meisten Fällen sind Ziffern fehlerhaft erkannt oder vertauscht. Deshalb wird die Levinsthein Distanz, siehe Kapitel 2.4 zwischen der realen Matrikelnummer und der erkannten Matrikelnummer untersucht. Ist diese Distanz geringer als ein konfigurierbarer Schwellwert, so wird der Student als korrekt identifiziert interpretiert. Dieser Schritt senkt die Fehlerrate weiter ab.

4.4.2 Punktzahlen

Bei der erkannten Punktzahlen können Fehler auftreten. Jede einzelne Punktzahl könnte falsch erkannt werden und so zu inkorrekten Informationen führen. Um solche falsch erkannten Zahlen zu detektieren, wird hier die erkannte Gesamtpunktzahl zur Kontrolle verwendet. Die einzelnen Punkte für jede Aufgabe werden summiert. Anschließend wird die Summe über all die erkannten Punkte gebildet. Wenn die berechnete Summe mit der erkannten Summe übereinstimmt, wird davon ausgegangen, dass alle Zahlen richtig erkannt wurden.

Ein weitere Fehlerkorrektur beinhaltet die Überprüfung der maximal erreichbaren Punktzahl für jede Aufgabe. Die Höchstpunktzahl für jede Aufgabe wird in einer Config-Datei angegeben. Sobald eine höhere Punktzahl für eine Aufgabe vorhergesagt wird, kann davon ausgegangen werden, dass nicht alle Punkte korrekt identifiziert werden konnten.

Die beiden Ansätze garantieren die Erkennung eines Fehlers. Bei mehr als einem Fehler ist es nicht garantiert, dass der Fehler erkannt wird. Allerdings zeigen die praktische Tests, siehe Kapitel 6, dass es nur selten vorkommt, dass zwei oder mehr Zahlen fehlerhaft detektiert werden und sich diese Fehler so ausgleichen, dass die erkannte Summe und die berechnete Summe übereinstimmen. Durch die beiden Fehlerkorrekturen können die Punktzahlen verlässlich erkannt werden.

4.4.3 QR-Codes

In diesem Projekt können Daten per QR-Codes gelesen werden. Der Vorteil dabei ist, dass QR-Codes eine eingebaute Fehlerkorrektur besitzen. Es muss nicht ein eigener Algorithmus zur Validierung der Daten erstellt werden. Die hinter QR-Codes implementierte Fehlerkorrektur beruht auf dem Reed-Solomon-Code [7], einem algorithmischen Verfahren zur Erkennung und Wiederherstellung von Daten in beschädigten oder gestörten Codes. QR-Codes verwenden dieses Verfahren, und fügen Redundanz in Form von zusätzlichen Datenblöcken hinzu. Diese Blöcke enthalten Informationen, die es dem Decoder ermöglichen, Fehler zu identifizieren und zu korrigieren. Durch die Redundanz kann der QR-Code selbst bei Beschädigung oder Verschmutzung noch erfolgreich decodiert werden. Je nach Konfiguration des QR-Codes variiert die Fehlerkorrekturfähigkeit. Codes mit höherer Fehlerkorrektur können größere Beschädigungen überwinden, während Codes mit geringerer Fehlerkorrektur empfindlicher auf Störungen reagieren. Höhere Fehlerkorrektur resultiert gleichzeitg auch mit größeren QR-Codes. Durch die bereits vorhandene Fehlerkorrektur der QR-Codes wird keine Fehlerkorrektur der dadurch erkannten Daten benötigt. Die so erfassten Daten werden in diesem Projekt als fehlerfrei angenommen.

Kapitel 5

Softwarearchitektur

Das gesamte System ist in mehrere, modulare Komponenten unterteilt, darunter ein Hauptprogramm mit einer grafischen Benutzeroberfläche. Mit diesem Hauptprogramm kann der Benutzer aktiv interagiert. Zusätzlich gibt es zwei Programme für Vorhersage von handschriftlichen Daten. Durch diese modulare Struktur ist es möglich, nur Bereiche einzelner Module zu warten und Anpassungen an die eingesetzte Hardware vorzunehmen, beispielsweise an nutzbaren Grafikkarten. Darüber hinaus bietet sie aufgrund der asynchron parallel ablaufenden Module einen Geschwindigkeitsvorteil, da verschiedene Prozessorkerne parallel genutzt werden können. Die Module können als Microservice in einem Docker-Container genutzt werden, entsprechend ist keine langwierige Einrichtung und Installation der Python-Packages nötig.

Im Folgenden werden die einzelnen Programmmodule erläutert. Die Module kommunizieren miteinander über die in Kapitel 5.5 vorgestellte API.

5.1 Configurator

Das Configurator-Programm spielt eine wichtige Rolle beim Einrichten des Hauptprogramms. Es ermöglicht das Einlesen des Deckblatts der Klausur, indem es zunächst ein Bild mit der angeschlossenen Kamera aufnimmt. Auf diesem Referenzbild können dann die Bereiche markiert werden, aus denen später Werte vorhergesagt werden sollen. Im Configurator-Programm können die relevanten Bereiche durch Ziehen eines Rechtecks mit der Maus über den entsprechenden Bereich markiert werden, wie in Abbildung 5.1 dargestellt.

Nachdem alle relevanten Bereiche markiert worden sind, können die Informationen gespeichert werden. Dazu gehört zum einen das aufgenommene Bild als Referenzbild im jpg-Format für das Feature-Mapping, siehe Kapitel 2.2. Zum anderen werden die Koordinaten der relevanten Bildbereiche als Python-Dictionary in einer txt-Datei gespeichert. Diese gespeicherten Informationen werden dann vom Hauptprogramm genutzt, um die erforderlichen Vorhersagen durchzuführen.

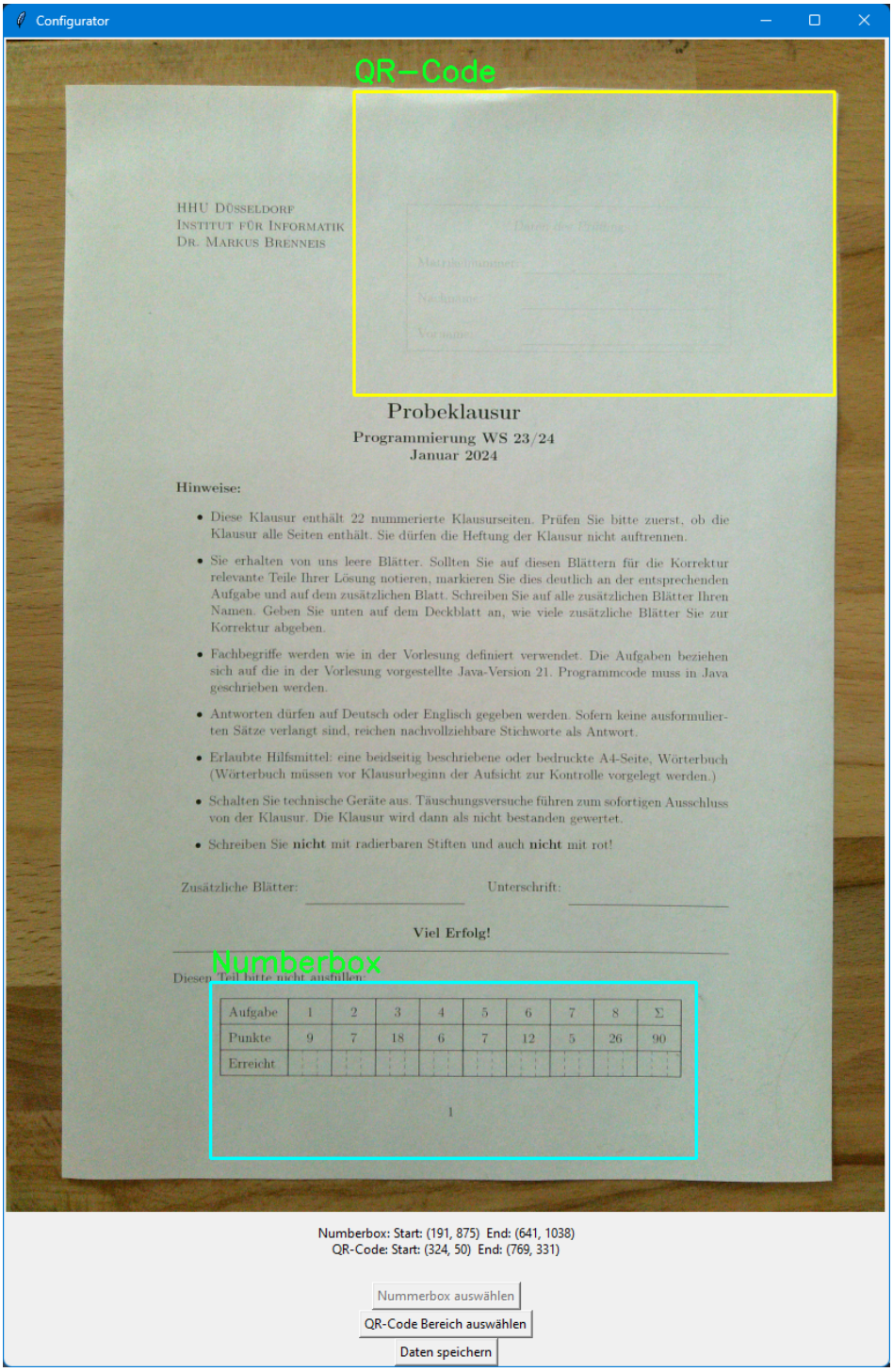


Abbildung 5.1: Configurator-Programm. Dieses Programm muss zu Beginn ausgeführt werden. Es wird ein Referenzbild für die Ausrichtung der ausgefüllten Deckblätter aufgenommen. Anschließend können relevante Bereiche markiert werden.

5.2 Hauptprogramm

Das Hauptprogramm beherbergt diverse essenzielle Funktionen. Der Benutzer interagiert nur mit diesem Programm. Die Oberfläche sieht bei der Nutzung wie in Abbildung 5.2 aus. Beim Start des Hauptprogramms wird zunächst eine Teilnehmerliste angefordert. Die Teilnehmerliste stammt von einem Portal der Universität und liegt im Excel- oder Open Document Format vor. Anschließend erfolgt die kontinuierliche Aufnahme von Bildern mithilfe der voreingestellten Kamera. Durch Feature Mapping werden die aufgenommenen Bilder entsprechend einer Vorlage ausgerichtet. Diese Vorlage wird durch das Configurator-Programm bereitgestellt. Neben der Vorlage werden Koordinatenbereiche in dem Configurator Programm definiert, in denen relevante Informationen stehen, wie Vor- und Nachname, Matrikelnummer, QR-Code oder die Punktetabelle. Die entsprechenden Bildbereiche werden in dem Hauptprogramm ausgeschnitten und gespeichert, um sie verschiedenen Unterprozessen zugänglich zu machen.

Die Prozesse laufen parallel und asynchron. Sie kommunizieren mit den verschiedenen Modulen, wie dem Number-Detector aus Kapitel 5.3, oder dem SimpleHTR Modul, aus Kapitel 5.4. Durch den asynchronen Ablauf können Vorhersagen parallel ausgeführt werden. Es muss beispielsweise nicht auf die Fertigstellung einer Namensvorhersage gewartet werden, bis die ein Punktevorhersage gestartet werden kann. Den Prozessen wird ein Bild übergeben und als Rückgabe wird der vorhergesagte Wert gegeben.

Im Hauptprogramm wird als nächstes geprüfen, ob die Werte valide sind. Der Student muss korrekt identifiziert werden und die korrekte Punktzahl muss vorliegen, damit die Daten eines Teilnehmers in die Ergebnisliste eingetragen werden können. Das Programm stellt dabei zwei verschiedene Routinen bereit. Zum einen gibt es die Möglichkeit die Studenten über einen QR-Code zu identifizieren, zum anderen über handschriftliche Namen und Matrikelnummern.

Bei der Identifikation über den QR-Code wird die Matrikelnummer des Studenten in einem QR-Code abgebildet. Wird der QR-Code erfolgreich decodiert, so wird davon ausgegangen, dass die Informationen korrekt sind. Es ist durch die Fehlerkorrektur des QR-Codes unwahrscheinlich, das ein falscher Text decodiert wird. Deshalb kann in der Praxis auf das Sliding Window, siehe Kapitel 4.4.1, verzichtet werden. Sofern ein Student mit dieser Matrikelnummer in der Teilnehmerliste vorhanden ist, wird dieser Teilnehmer als aktueller Teilnehmer

gespeichert. Gleichzeitig wird geprüft, ob eine korrekte Vorhersage der Punkte vorhanden ist. Kriterium dafür ist, dass die Summe der erreichten Punkte der einzelnen Aufgaben gleich der vorhergesagten Gesamtpunktzahl ist und keine Punktzahl höher als die erreichbare Punktzahl ist. Sofern diese Kriterien erfüllt sind, werden die Punkte als aktuelle Punktzahl gespeichert. Wenn nun ein aktueller Teilnehmer und eine aktuelle Punktzahl vorhanden sind, werden die Daten nach händischer Bestätigung in die Ergebnisliste eingetragen. Parallel wird der Studenteneintrag aus der Liste möglicher Studenten entfernt. Außerdem wird die aktuelle Punktzahl und der aktuelle Teilnehmername zurückgesetzt.

Bei der Identifikation des Studenten über handschriftliche Namen und Matrikelnummer verläuft der Prozess zur Bestimmung des aktuellen Teilnehmers anders. Der Prozess zur Ermittlung der Punkte bleibt gleich. Um einen Studenten zu identifizieren, wird zunächst ein Teil der Teilnehmerliste als Sliding Window geladen. Die Klausuren sind alphabetisch vorsortiert, allerdings können einzelne Klausuren vertauscht sein. Die Größe des Sliding Windows kann in einer Konfigurationsdatei angepasst werden. Wird ein Student korrekt identifiziert und die Punkte wurden entsprechend der vorher beschriebenen Überprüfungen als korrekt erkannt, werden die Daten in der Ergebnistabelle gespeichert und der Studenteneintrag wird aus der Sliding-Window-Liste entfernt. Damit die Sliding-Window-Liste stets die gleiche Größe hat, wird der alphabetisch nachfolgende Studenteneintrag, der noch nicht in dieser Liste ist, hinzugefügt. Durch das SimpleHTR-Modul wird der Studentenname aus dem Sliding-Window zurückgegeben, der mit der größten Wahrscheinlichkeit auf das Deckblatt geschrieben ist. Ist ein Studenteneintrag in der Liste, der den erkannten Vornamen und Nachnamen besitzt, so wird als nächstes die erkannte Matrikelnummer geprüft. Da diese in der Praxis nur selten fehlerfrei erkannt wird, wird die Levenshtein-Distanz, siehe Abschnitt 2.4, zwischen der vorhergesagten Matrikelnummer und der des Studenten mit erkannten Vor- und Nachnamen bestimmt. Ist die Distanz unter einem konfigurierten Schwellwert, so gilt der Student als korrekt identifiziert.

Alle Einstellungen für dieses Programm können in einer Konfigurationsdatei vorgenommen werden. Über diese Datei können Parameter wie die Auswahl der Kamera und die verwendete Auflösung festgelegt werden. Eine weitere Option betrifft die Größe des Sliding Windows für die Auswahl der Studenteneinträge. Weitere Einstellungen betreffen die Verbindungen zu den verschiedenen Modulen für die Vorhersage der Handschrift. Es kann festgelegt werden, welche Module verwendet werden sollen, beispielsweise ob QR-Codes verwendet werden sollen oder der Name in Handschrift erkannt werden soll. Falls ein bestimmtes Modul genutzt werden soll, muss für dieses Modul eine URL in der Config Datei hinterlegt werden, unter der der Service erreichbar ist.

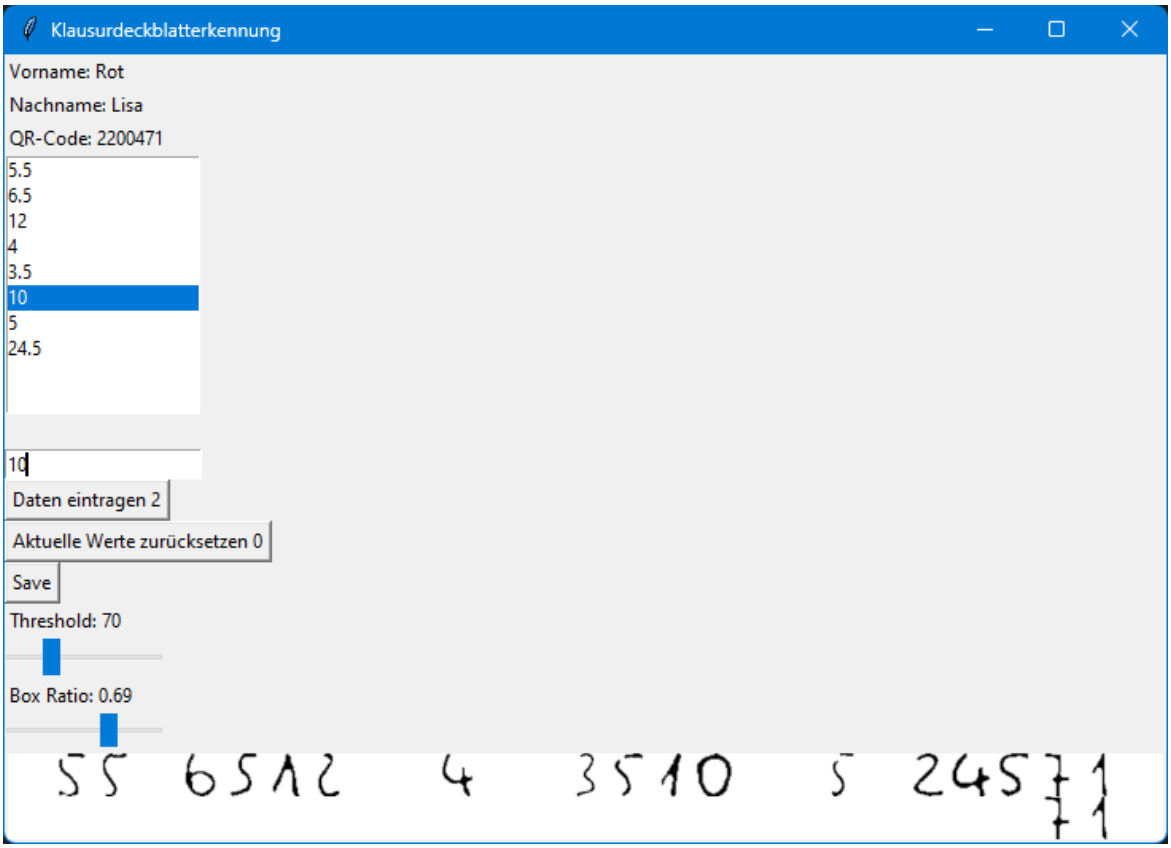


Abbildung 5.2: Die Oberfläche des Hauptprogramms.

5.3 Number-Detector

Das Programm zur Vorhersage von Zahlen ist der Number-Detector. Dieses Programm kann sowohl die erreichten Punktzahlen auf dem Deckblatt vorhersagen als auch die geschriebene Matrikelnummer erkennen. Als Eingabedaten benötigt das Programm ein Bild, auf dem der zu erkennende Text steht. Als Rückgabe liefert das Programm den erkannten Text sowie optional das bearbeitete Bild, das für die Vorhersage verwendet wurde sowie einen Boolean-Wert, der angibt, ob die Daten valide sein können.

Die Kommunikation mit dem Programm erfolgt über eine REST-API, die näher in Abschnitt 5.5 beschrieben wird. Ein Webserver, der durch das Modul Flask bereitgestellt wird, hört auf zwei verschiedenen URL-Endpunkten nach Daten. Ein Endpunkt ist für die Vorhersage der Matrikelnummern, der andere für die Vorhersage der Punktetabelle.

Für die Vorhersage der Matrikelnummer ist keine weitere Bildverarbeitung erforderlich. Das empfangene Bild hat vorher festgelegte Dimensionen und kann daher direkt durch das Machine-Learning-Modell aus Kapitel 4.2.3 vorhergesagt werden. Die vorhergesagte Matrikelnummer wird entsprechend den API-Bestimmungen an das Hauptprogramm zurückgesendet.

Die Vorhersage der erreichten Punkte umfasst verschiedene Schritte. Zuerst werden, wie für die Matrikelnummervorhersage, die Bilddaten entgegengenommen. Anschließend werden die in Kapitel 3.3 beschriebenen Preprocessingschritte durchgeführt. Der Programmablauf unterscheidet sich im nächsten Schritt je nachdem, ob die Klausurpunkte durch Dezimalzahlen oder ganze Zahlen beschrieben werden. Bei ganzen Zahlen müssen keine weiteren Schritte durchgeführt werden und die Abbildungen können durch das neuronale Netz klassifiziert werden. Bei der Verwendung von Dezimalzahlen enthält jede Abbildung der Punktzahl drei Felder. Darüber hinaus kann das Gesamtergebnis möglicherweise vier Zellen enthalten, sofern die erreichbare Punktzahl größer als 100 ist, wie in Abbildung 3.3 dargestellt. Die durch die Bildverarbeitung extrahierten Bilder werden vertikal mit äquidistanten Abständen in drei, beziehungsweise vier Felder geteilt, um die Ziffern zu segmentieren. Dabei werden die Felder exakt ausgeschnitten, es gibt keine Überlappung oder einen Rand zwischen benachbarten Zellen.

Im nächsten Schritt werden die Zahlenabbildungen durch das neuronale Netz klassifiziert. Dabei wird jeder möglichen Zahl eine Wahrscheinlichkeit zugeordnet, dass diese Zahl in der Abbildung vorhanden ist. Die Wahrscheinlichkeiten werden absteigend sortiert, und die beiden größten Werte werden betrachtet. Sofern die höchste Wahrscheinlichkeit über einem gewissen, in der Konfigurationsdatei konfigurierten Vertrauensschwellwert liegt, wird die entsprechende Ziffer als korrekt vorhergesagt interpretiert. Ist dieser Wert geringer als der Vertrauensschwellwert, werden beide zugehörigen Ziffern als mögliche Ergebnisse gespeichert. Dieser Vorgang wird für alle Abbildungen wiederholt, sodass eine Liste mit ein bis zwei Vorhersagen pro Abbildung existiert.

Für Dezimalzahlen müssen die zuvor segmentierten Zahlen wieder kombiniert werden. Jeweils drei Felder bilden dabei die Punktzahl einer Aufgabe. Das erste Feld, sofern vorhanden, beschreibt den Zehner-Wert, das zweite Feld den Einser-Wert und das dritte Feld die halben Punkte. Es wird angenommen, dass in der Zelle für die Einser-Werte immer eine Ziffer vorhanden ist. Wurden keine Punkte bei einer Aufgabe erreicht, so sollte in diese Zelle eine Null geschrieben werden. Je nach Vertrauensschwellwert und Konfidenz der Vorhersage werden ein oder zwei vorhergesagten Zahlen werden als mögliche Punkte für die Aufgabe gespeichert.

Zur Erkennung der Zehner-Werte wird ein konfigurierbarer Schwellwert für schwarze Pixel verwendet, um zu bestimmen, ob eine Zahl eingetragen wurde. Sofern eine Zahl in das Feld geschrieben wurde und der Schwellwert für die schwarzen Pixel überschritten ist, wird die Zahl durch das neuronale Netz vorhergesagt. Je nach Konfidenz der Vorhersage und Vertrauensschwellwert sind wieder ein bis zwei Vorhersagen vorhanden. Wenn nur eine valide Vorhersage ist, wird auf alle möglichen Punktzahlen für diese Aufgabe der Vorhergesagte Zehnerwert mit zehn multipliziert addiert. Ist die höchste Wahrscheinlichkeit der Vorhersage unter dem Vertrauensschwellwert, wird die zweitbeste Vorhersage hinzugezogen. Dazu wird die Liste mit möglichen Punktzahlen für die Aufgabe dupliziert. Auf die eine Liste wird die erste Vorhersage mit dem Faktor zehn multipliziert und addiert, auf die andere Liste der zweite Wert mit Zehn multipliziert und addiert.

Für die halben Punkte wird keine Vorhersage des neuronalen Netzes genutzt. In dem Feld steht eine fünf oder nichts. Es wird entsprechend nur geprüft, ob etwas in dieses Feld geschrieben wurde. Um dies zu prüfen, wird die Anzahl an schwarzen Bildpunkten gezählt. Sofern diese Anzahl über einem gewissen, in der Konfigurationsdatei festgelegten Schwellwert liegt, wird angenommen, dass ein halber Punkt vergeben wurde. Entsprechend wird für alle möglichen Ergebnisse der Aufgabe ein halber Punkt addiert.

Der folgende Schritt unterscheiden sich nicht mehr zwischen den Dezimalzahlen und den ganzen Zahlen. In beiden Fällen wurden Listen mit möglichen Punktzahlen für jede Aufgabe erstellt. Es werden nun alle Kombinationen an vorhergesagten Punktzahlen addiert und geprüft, ob diese Summe im Ergebnisfeld vorhergesagt wurde. Ist dies der Fall, wird über die REST-API die korrekte Vorhersage zurückgegeben, sowie ein Boolean, der auf wahr gesetzt ist. Falls nicht, wird die wahrscheinlichste Vorhersage mit einem Wahrheitswert, der auf falsch gesetzt ist, zurückgegeben.

5.4 SimpleHTR Modul

Eine weitere Funktionalität dieses Projekts besteht darin, handschriftliche Vor- und Nachnamen zu erkennen. Hierfür wurde ebenfalls ein Modul entwickelt, das über eine REST-API kommuniziert werden kann. Ähnlich wie beim Number-Detector erwartet auch dieses Modul als Eingabeparameter ein Bild, auf dem der vorherzusagende Text steht. Zusätzlich wird für eine höhere Wahrscheinlichkeit einer korrekten Vorhersage eine Namensliste der Studenten benötigt, die ebenfalls über die REST-API bereitgestellt wird. Als Rückgabewert liefert es den vorhergesagten Namen.

Für die Vorhersage der Namen wurde das Projekt SimpleHTR als Grundlage verwendet, welches bereits in der Literatur [16] beschrieben wurde. Das Projekt wurde erweitert durch einen Flask Server, der die REST-API implementiert und als Schnittstelle zwischen dem Hauptprogramm und diesem Modul agiert. Der Server nimmt bei jeder Anfrage das entsprechende Bild sowie eine Namensliste entgegen. Das Bild wird im Anschluss durch SimpleHTR vorhergesagt.

Nachdem SimpleHTR den Namen mit der höchsten Wahrscheinlichkeit auf dem Bild vorhergesagt hat, wird dieser Name über die REST-API dem Hauptprogramm zurückgeliefert. Neben dem Namen wird auch der Bildausschnitt mitgesendet, der für die Vorhersage verwendet wurde. Dieser Bildausschnitt kann für Debugging-Zwecke nützlich sein, um beispielsweise festzustellen, ob der richtige Bereich des Bildes für die Vorhersage ausgewählt wurde. Durch die Bereitstellung dieses Bildausschnitts kann der Benutzer eventuelle Probleme bei der Texterkennung identifizieren und beheben.

5.5 Kommunkation zwischen den Modulen

Für die Kommunikation der verschiedenen Module wird eine REST-API definiert. Daten werden im JSON Format versendet und empfangen. Es müssen zwischen den Modulen verschiedene Arten von Daten ausgetauscht werden, wie beispielsweise Bilder, vorhergesagte Namen, Namenslisten und vieles mehr. Im folgenden wird erläutert, wie die verscheidenen Datentypen vor dem Senden verarbeitet werden.

Bilddaten werden zu allen Subsystemen im gleichen Format gesendet. Die Bilddaten sollten zunächst als Numpy-array vorliegen, es werden nur Graustufenbilder unterstützt. Um nun das Bild per REST-API an ein anderes Subsystem zu senden, sollte das Bild-Aarray wie in Listing 5.1 konvertiert werden. Das zweidimensionale Array wird dabei zunächst in ein eindimensionales Array reduziert, da die Form eines Arrays nicht mit der verwendeten Funktion gespeichert wird. Anschließend wird die Höhe und Breite des Bildes an das Array gehängt, um im Subsystem das zweidimensionale Array rekonstruieren zu können. Im letzten Schritt wird das Array mit einer *base64* Kodierung in Bytes umgewandelt. Diese Bytes werden in einem JSON Paket unter dem Schlüssel *image* als POST Anfrage versendet.

```
1 height, width = image.shape
2 cut = np.reshape(image,(height*width))
3 cut = np.append(cut,height)
4 cut = np.append(cut,width)
5 cut = cut.astype(np.uint64)
6 array_bytes = cut.tobytes()
7 image_base64 = base64.b64encode(array_bytes).decode('utf-8')
8 return image_base64
```

Listing 5.1: Konvertierung der Bilddateien um sie an ein Subsystem zu senden

Um die Bilddaten in einem anderen System zu nutzen müssen die empfangenen Daten umgewandelt werden. Dazu werden die Daten zuerst von *base64* dekodiert werden, sodass sie als Numpy array vorliegen. Anschließend kann die zuvor angehängte Höhe und Breite des Bildes aus dem Array gelesen werden. Mit diesen beiden Informationen kann der eindimensionale Array in einen zweidimensionelen Array geformt werden, welches die Bilddaten repräsentiert. Dieser Vorgang kann in Python durch den Code in Listing 5.2 durchgeführt werden.

```
image=base64.b64decode(image.encode('utf-8'))
image_array = np.frombuffer(image, dtype=np.uint64)
h=image_array[-2]
w=image_array[-1]
image_array = image_array[:-2]
image_array = image_array.reshape((h, w))
return image_array
```

Listing 5.2: Konvertierung empfangener Daten zu einem Bild

Die Vorhersage eines Moduls stellt weitere Daten dar, die versendet werden müssen. Die Vorhersagen liegen als String vor, welche durch JSON unterstützt werden. Die Vorhersagen werden daher als String unter dem Schlüssel *prediction* versendet.

Das Number-Detecor Modul benötigt für die Vorhersage unter anderem ein Schwellwert für die Bildverarbeitung, sowie ein Höhen-Seitenverhältnis der Punkteboxen. Beide Parameter sind Zahlen, die in dieser Form von JSON unterstützt werden. unter dem Schlüssel *threshold* wird der Schwellwert versendet, unter dem Schlüssel *boxratio* das Höhen-Seitenverhältnis der Punktebox.

Das Number-Detection Modul liefert darüber hinaus einen Boolean, ob die erkannte Punktzahl valide sein kann, oder nicht. Auch Booleans werden in JSON nativ unterstützt, sodass unter dem Schlüssel *found* der entsprechende Paramter versendet werden kann.

Das SimpleHTR Modul benötigt eine Namensliste für die Vorhersagen. Um diese Vor- oder Nachnamen zu übertragen werden die einzelnen Namen in einen String zusammengefasst. Die Namen werden durch Leerzeichen getrennt. Anschließend wird der String unter dem Schlüssel *names* dem JSON Paket hinzugefügt.

In Abbildung 5.3 ist dargestellt, welche Daten zwischen welchen Modulen versendet werden. Das Hauptprogramm sendet stets zuerst ein JSON-Paket zu den jeweiligen Modulen als POST-Request. Als Antwort wird von den Modulen ein JSON-Paket mit den entsprechenden Schüsseln und Daten gesendet.

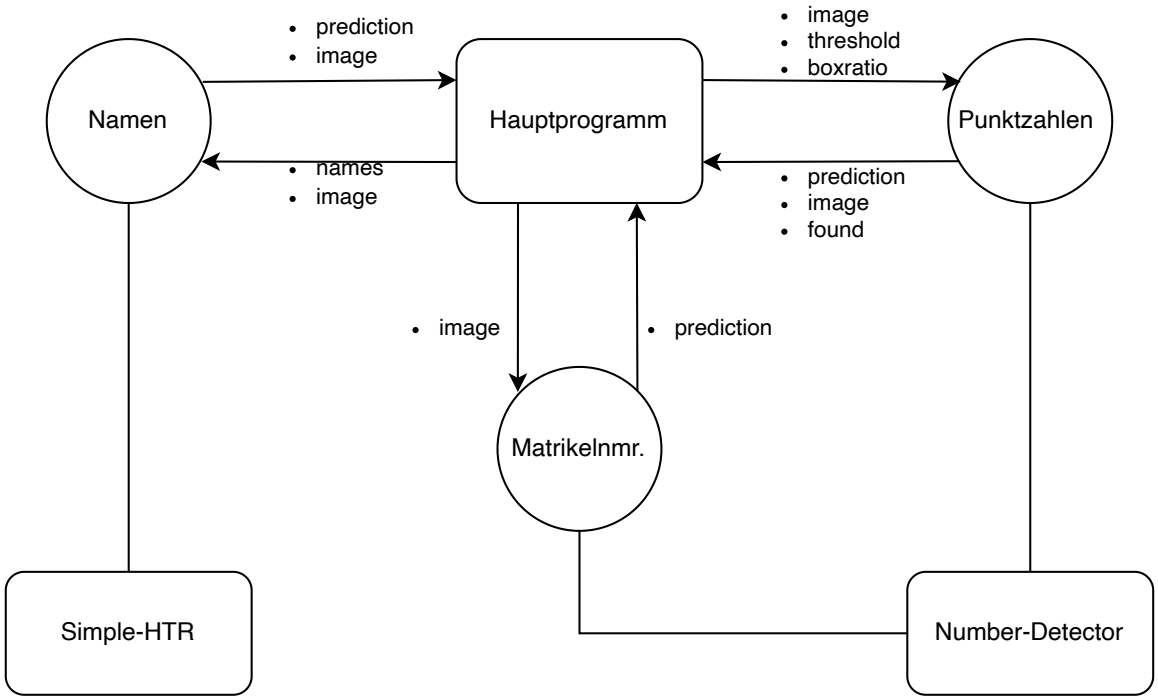


Abbildung 5.3: Kommunikation zwischen den Modulen in diesem Projekt. Die Rechtecke stellen jeweils ein Modul dar. Die Kreise repräsentieren die Endpunkte der per Linie verbundenen Module. Die Pfeile repräsentieren die Kommunikation zwischen den Modulen. Das jewilige JSON-Paket wird mit den auf den Pfeilen dargestellten Schlüssel und entsprechenden Daten versendet. Das Hauptprogramm sendet POST Anfragen, die Daten in Rückrichtung werden als Response gesendet.

Kapitel 6

Ergebnisse

Das Projekt wurde an drei verschiedenen Klausuren getestet. Die Deckblätter der ersten Klausur enthalten ganze Punktzahlen und handgeschriebene Namen, sowie Matrikelnummern. Die zweite und dritte Klausur haben eine angepasste Punktetabelle, sodass Dezimalzahlen detektiert werden können und die Studenten können über einen QR-Code identifiziert werden. Diese Tests sollen die Fehlerrate in der Praxis verdeutlichen, sowie auf mögliche Probleme hinweisen.

6.1 User Experience

Bei den praktischen Tests wurden einige Aspekte entdeckt, die die Benutzung der Anwendung vereinfachen oder effizienter gestalten könnten. Probleme traten auf, sobald die Punkteliste nicht korrekt identifiziert werden konnte und einzelne Punktzahlen nachträglich korrigiert werden müssen. Um die Punktzahlen einer Aufgabe zu ändern, muss zunächst auf die als fehlerhaft erkannte Punktzahl zur Auswahl geklickt werden. Im Anschluss kann in ein Feld die neue Zahl eingegeben.

Beim Versuch, das Feld zum Editieren der Punktzahl auszuwählen, ist es ein paarmal passiert, dass das unmittelbar darunterliegende Feld versehentlich ausgewählt wurde. Dieses Feld bestätigt die Korrektheit der Punkte und die falschen Punkte werden in die Ergebnistabelle eingetragen. Dieses Problem wurde auf zwei verschiedene Arten gelöst.

Zum einen wurde der Abstand zwischen dem Feld zum Editieren und dem Eintragen der Zahlen in der Benutzeroberfläche vergrößert. Durch die angemessene Abstandsgestaltung der Elemente wird die Anzahl versehentliche Klicke auf das falschen Feld minimiert. Unter Umständen kann es aber trotzdem passieren, dass eine inkorrekte Punktzahl eingetragen wird, und man diesen Fehler schnell bemerkt. Es existierte keine Möglichkeit diese Punktzahl bei Laufzeit des Programms zu entfernen oder eine neue Punktzahl für den Studenten ein zu tragen.

Deshalb wurde das Hauptprogramm so erweitert, dass nach Klick eines *Revert*-Button der letzte eingetragene Student wieder zur Liste der Studenten hinzugefügt wird, die auf dem Deckblatt erkannt werden können. Dadurch kann das Deckblatt erneut eingescannt werden, der Student identifiziert und die Punktzahl überschrieben werden. Ohne diese Funktion muss die Klausur bisher manuell beiseite gelegt werden. Nachdem alle übrigen Punkte eingetragen worden sind, kann die Ergebnisliste geöffnet werden und händisch die Punktzahl des Studenten korrigiert werden. Dieser langwierige Prozess ist durch den *Revert*-Button sehr viel effizienter geworden.

Ein weiterer Prozess, der optimiert werden konnte, ist das Editieren der Punkte, falls eine Punktzahl nicht erkannt wurde. Zu Beginn des Projektes wurde die nicht detektierte Punktzahl durch eine -1 am Ende der Punkteliste dargestellt. Es wurde keine Leerstelle eingefügt, sondern die jeweils nachfolgende Zahl. Um nun ein Aufgabe zu korrigieren, mussten alle folgenden Punkte ebenfalls neu eingetragen werden, was sehr viel Zeit benötigt. Dieser Fehler wurde behoben, indem Leerstellen an der richtigen Position eingefügt werden.

Ein weiterer Aspekt ist das Korrigieren von Punktzahlen auf dem Deckblatt. Wenn eine falsche Punktzahl geschrieben wird, wird diese meist durchgestrichen und daneben korrigiert. Diese Zahlen können durch die Zahlenerkennung nicht zuverlässig vorhergesagt werden. Um dieses Problem zu lösen wurde das Korrigieren von Punkten auf dem Deckblatt verändert. Die neue gültige Punktzahl sollte nicht in das selbe Feld wie die durchgestrichene Zahl geschrieben werden, sondern genau unter das Feld. Das Programm erkennt, sobald etwas unter einem Feld geschrieben ist und benutzt diese Daten für die Vorhersage. So können die Punktzahlen korrigiert werden und das Programm kann die Punkte trotzdem korrekt vorhersagen.

6.2 Fehlerrate

Neben der schnellen Erkennung der Handschrift auf dem Deckblatt ist eine geringe Fehlerrate von großer Bedeutung. Es sollten keine inkorrekten Daten in das System eingetragen werden, damit die Prüfungsergebnisse nicht falsch sind. Damit keine falschen Daten eingetragen werden sollten die erkannten Daten stets zusätzlich manuell überprüft werden. Durch diesen Vorgang können kritische Fehler erkannt und behoben werden. Kritische Fehler sind Fehler, die nicht von dem Programm identifiziert werden und somit zu nicht wahren Eintragungen in der Ergebnisliste führen.

Die kritischen Fehlerraten in den Praxistests waren gering, wenige Fehler waren jedoch vorhanden. In dem Praxistest mit handschriftlichen Namen und ganzen Punktzahlen wurden 110 Klausuren genutzt. Für die Namenserkennung wurde das Feature des Sliding-Window verwendet mit einem Umfang von 30 Studenten genutzt. Von den insgesamt 110 Studenten wurden 83 Studenten korrekt erkannt und das Programm hat die Eingabe der korrekten Daten ermöglicht. Bei den übrigen 27 Studenten konnte der Name nicht korrekt identifiziert werden und mit der Matrikelnummer abgeglichen werden. Es sind also keine kritischen Fehler aufgetreten, keinem Studenten wurden beispielsweise die Punkte eines anderen Studenten zugewiesen. Die Erkennungsrate lag somit bei $75,5\,\%$ und die kritische Fehlerrate bei $0\,\%$. Die Werte sind in der Vierfeldertafel in Tabelle 6.1 dargestellt.

Namen	erkannt	nicht erkannt	
bestätigt	83	0	83
nicht bestätigt	0	27	27
	83	27	110

Tabelle 6.1: Ergebnis des Praxistests einer Klausur mit handgeschriebenen Namen als Vierfelder-Tabelle dargestellt.

Für die nicht erkannten Namen gibt es verschiedene Gründe. Beispielsweise haben die Studenten ihre Namen teilweise nicht auf die dafür gedachte Linie geschrieben, sondern darunter oder rechts daneben. Dadurch konnten die Namen nicht von dem Hauptprogramm korrekt ausgeschnitten werden und die Vorhersage schlägt fehl.

Ein weiteres Problem tritt bei Doppelnamen auf. Die Namen können nur erkannt werden, sofern der Name auf dem Deckblatt dem Namen in der Teilnehmerliste entspricht. Wenn eine Studentin *Anna Lena Müller* heißt und in der Teilnehmerliste unter dem Vornamen *Anna Lena* und dem Nachnamen *Müller* eingetragen ist, so kann das Deckblatt von ihr nur identifiziert werden, wenn sie ihren vollen Namen schreibt. Lediglich *Anna* oder *Lena* als Vorname auf dem Deckblatt genügt nicht.

Ein weiteres Problem ist die stark variierende Schriftgröße der unterschiedlichen Studenten. In dem Praxistest wurde deutlich, wie unterschiedlich die Studenten ihre Namen schreiben. Dabei sind Namen mit einer großen Schriftgröße vorteilhaft zur Erkennung durch das Programm, da mehr Bildinformationen vorhanden sind. Schreibt ein Student seinen Namen zu klein, so kann es zu Problemen führen, da die Auflösung nicht reicht. Zwar werden die Namen ausgeschnitten und skaliert, jedoch funktioniert dieser Vorgang nur bis zu einem gewissen Maße, sodass zu kleine Namen nicht detektiert werden können.

Für die Punkteerkennung auf den Deckblättern wurde das Projekt in der Konfiguration für ganze Zahlen verwendet. Es wurden die selben Klausuren wie für die handschriftlichen Namen verwendet. Bei der Detektion der Daten durch dieses Projekt kam es zu einem kritischen Fehler. Es wurden zwei Zahlen falsch vorhergesagt, eine Punktzahl und die Summe wurden falsch erkannt. Beide Zahlen weichen um den gleichen Wert von der realen Zahl ab. Gleichzeitig war diese Abweichung noch im Bereich der möglichen Punkte, sodass es dem Programm nicht möglich war, den Fehler zu erkennen. Mit dieser fehlerhaften Eingabe liegt die kritische Fehlerrate bei $0,9\,\%$.

Die Erkennungsrate der Punkte liegt mit erkannten 65 von 110 korrekt erkannten Punktetabellen bei $59\,\%$. Die übrigen 44 Klausuren wurden durch verschiedene Gründe nicht erkannt. Auf 34 der Klausuren wurde die Punktzahl durchgestrichen und in dasselbe Feld etwas weiter rechts oder links erneut eingetragen. Diese Punkte konnten nicht durch das neuronale Netz korrekt vorhergesagt werden. Die Anzahl erkannter und nicht erkannter Punkte sind in der Vierfeldertafel in Tabelle 6.2 dargestellt.

Bei sieben der Klausuren wurde die Punktzahl nicht in das entsprechende Ergebnisfeld oder der Inhalt ist nicht mittig genug in die Zelle geschrieben. Die Zahlen touchieren den Rand der Zelle und werden dadurch abgeschnitten. Entsprechend konnte die Zahlen in diesen Fällen nicht erkannt werden. Es wird stets nur der Inhalt der entsprechenden Zelle für die Vorhersage genutzt. Sofern Informationen außerhalb dieses Bereichs vorhanden sind, können diese nicht zur Vorhersage genutzt werden und die Vorhersage hat eine erhöhte Wahrscheinlichkeit fehl zu schlagen.

Punkte	erkannt	nicht erkannt	
bestätigt	65	1	66
nicht bestätigt	0	44	44
	65	45	110

Tabelle 6.2: Ergebnis des Praxistest einer Klausur mit ganzen Punktzahlen als Vierfelder-Tabelle dargestellt.

Ein weiterer Test wurde mit einer anderen Klausur durchgeführt. Dort werden die Studenten durch QR-Codes identifiziert. Die Punktetabelle ist angepasst und in drei Segmente unterteilt, um auch Dezimalzahlen zu detektieren. Es wurden 81 Klausuren untersucht.

Die Identifikation der Studenten über den QR-Code gelang sehr schnell und zuverlässig. Die QR-Codes können an verschiedenen Positionen auf dem Deckblatt sein, rotiert werden oder teilweise bedeckt sein und trotzdem problemlos erkannt werden. Bei jeder der 81 Klausuren konnte der Student in wenigen Augenblicken identifiziert werden. Es ist zu keinem Fehlern gekommen.

Die Punkteidentifikation gelang überwiegend zuverlässig. 42 der 81 Klausuren wurden ohne Probleme erkannt und konnten in das System eingetragen werden. Die Punkte der restlichen 39 konnten aufgrund eines systematischen Fehlers nicht komplett erkannt werden. Die Ziffer Zwei konnte nur sehr selten korrekt erkannt wurde. Meistens wurde statt einer Zwei fälschlicherweise eine Acht erkannt. Dies war besonders der Fall, wenn die Zwei mit einer großen Schlinge links unten gezeichnet wurde. Vermutlich werden dadurch im neuronalen Netz Features detektiert, die durch diese geschlossene Schleife bei einer Acht vorhanden sind. Durch weiteres Training kann ein solcher systematischer Fehler vermutlich behoben werden, wie im Kapitel 7 diskutiert. Die nicht korrekt erkannten Punktzahlen wurden dabei in dem Programm korrigiert, sodass die Punkte auf der Ergebnisliste vorhanden sind.

6.3 Zeitersparnis

Ein wichtiger Faktor dieses Projektes ist die mögliche Zeitersparnis gegenüber der manuellen Eintragung der Werte in die Ergebnisliste. Um eine mögliche Verbesserung zu messen, wurde der Test mit den 81 gescannten Deckblättern wiederholt, wobei nicht dieses Projekt verwendet wurde, sondern die Punkte manuell in ein Liste eingetragen werden. Bei der Wiederholung des Tests haben zwei Personen die Daten verarbeitet. Eine Person liest zunächst den Namen des Studenten vor. Die zweite Person sucht den jeweiligen Namen in der Excel-Liste. Ist der Student gefunden, so ließt die erste Person die erreichten Punkte jeder Aufgabe vor und die zweite Person trägt die einzelnen Punktzahlen in die Tabelle ein. Durch eine Tabellenkalkulation wir die Summe der erreichten Punkte aller Aufgaben gebildet. Die zweite Person ließt diese Summe vor und die erste Person prüft, ob diese mit der berechneten Summe auf dem Deckblatt übereinstimmt. Ist dies der Fall, so wird das nächste Deckblatt mit dem gleichen Vorgehen bearbeitet. Weicht die Summe ab, so wird kontrolliert, ob einzelne Punkte falsch übertragen wurden oder auf der Klausur die Summe fehlerhaft berechnet wurde.
Für die Bearbeitung der 81 Klausuren haben zwei Personen 25 Minuten benötigt. Somit liegt der Zeitaufwand umgerechnet bei insgesamt 50 Personenminuten und die Geschwindigkeit $1,62$ Klausuren pro Minute.

Bei Verwendung dieses Projektes wird nur eine Person benötigt. Diese legt das jeweilige Deckblatt unter die Dokumentenkamera, prüft die Validität der erkannten Daten und korrigiert unter Umständen falsch erkannte Daten. Der Zeitaufwand bei dem Test betrug 37 Minuten und die Geschwindigkeit $2,19$ Klausuren pro Minute. Entsprechend beschleunigt dieses Projekt das Lesen und Eintragen der Daten um $35\,\%$.

Bei einer weiteren Klausur wurde die Geschwindigkeit bei Verwendung dieses Projektes untersucht. Die Studenten wurden wieder über QR-Codes identifiziert und es wurde die angepasste Punktetabelle für Dezimalzahlen verwendet. Für die 343 Klausuren wurden mit dem Projekt 174 Minuten benötigt, was in einer Geschwindigkeit von $1,97$ Klausuren pro Minuten resultiert.

Kapitel 7

Diskussion und Ausblick

Insgesamt gestaltet dieses Projekt den Vorgang des Eintragen der Punkte effizienter. Der Zeitaufwand sinkt bei der Verwendung des Projektes und es wird nur eine Person benötigt. Das Projekt ist universell in verschiedensten Voraussetzungen einsetzbar. Die Klausuren müssen nicht aufwendig für das Projekt angepasst werden. Außerdem funktioniert das Projekt unter verschiedenen Umgebungsbedingungen, mit verschiedenen Kameras und verschieden Lichtbedingungen. Das Configurator-Programm ermöglicht die automatische Anpassung an verschiedene Bildparameter. Durch den modularen Aufbau und der definierten API könnte durch zukünftige Modifikationen die Effizienz oder Fehlerkorrektur weiter noch gesteigert werden.

Die dokumentierte und einfach zu implementierende API ermöglicht die zukünftige Erweiterung oder Anpassung des Projektes. Falls beispielsweise in Zukunft ein besser geeignetes neuronales Netz oder eine andere für Handschrift zuverlässige Texterkennung auf anderer Basis, wie durch Verbessungen bei Tesseract [18], veröffentlicht wird, kann das Projekt ohne viel Aufwand modifiziert werden.

Besonders herausfordernd gestaltete sich in dieser Arbeit die Lokalisierung der relevanten Daten. Durch Kontureneigentschaften konnten die notwendigen Daten nicht zuverlässig lokalisiert werden. Die Lösung durch das Feature Mapping erwies sich als geeignete Alternative und stellt einen robusten Algorithmus dar, der schnell und präzise funktioniert.

Eine weitere Herausforderung war die Identifizierung der Studenten. Die handschriftlichen Namen und Matrikelnummern konnten nicht alle genau genug erkannt werden. Durch die Umstellung auf QR-Codes konnte dieses Problem gelöst werden.

Die Praxistests konnten zeigen, welche Teile des Projektes besonders zuverlässig funktionieren, und welche noch weiter verbessert werden können. Ein erkanntes Problem war die Erkennung der Ziffer Zwei. Die Ziffer wurde häufig mit einem großen Bogen an der linken Seite des unteren Balken gezeichnet. Durch diesen großen Bogen und dem Halbbogen weiter oben, hat das neuronale Netz die Zwei häufig fehlerhaft als Acht klassifiziert.

Darüber hinaus können je nach Schriftart weitere systematische Fehler auftreten, beispielsweise wenn ein Korrektor die Sieben in der amerikanischen Schulausgangsschrift schreibt, siehe Abbildung 4.5. Die Sieben könnte fehlerhaft als Eins vorhergesagt werden.

Durch ein angepasstes Fine-Tuning könnten diese Probleme in einer Folgearbeit wahrscheinlich behoben werden. Eine Möglichkeit an dafür gut geeignete Daten zu gelangen ist die Klausuren die, nicht korrekt erkannt wurden, zu verwenden. Es ist möglich, die Punktetabelle zu speichern, sobald Werte korrigiert werden. Aus diesen Bilddaten können die Zahlen wie in Kapitel 3.3 beschrieben, extrahiert werden und anschließend manuell gelabelt werden. Wird auf diesen Daten trainiert, so sollte das neuronale Netz anschließend ähnlich geschriebene Zahlen, im Gegensatz zu dem bisherigen Training, nun erfolgreich erkennen können.

In weiteren Folgearbeiten könnte zusätzlich die Benutzeroberfläche optimiert werden. Zum Beispiel kann elaboriert werden, wie das Hauptprogramm für eine effizientere Benutzung angepasst werden sollte. Es kann verändert werden, welche Elemente dem Nutzer gezeigt werden, um die erkannten Werte mit den realen Daten schneller abgleichen zu können. Außerdem könnte eine text-to-speech Unterstützung implementiert werden, damit der Nutzer die erkannten Daten intuitiver und schneller mit den realen Daten vergleichen kann.

Literatur

[1] Mai Chi Bao. *Automated-scoring-of-handwritten-test-papers*. 2022. URL: https://github.com/mrzaizai2k/Automated-scoring-of-handwritten-test-papers.

[2] G. Bradski. „The OpenCV Library“. In: *Dr. Dobb's Journal of Software Tools* (2000).

[3] Michael Calonder, Vincent Lepetit, Christoph Strecha und Pascal Fua. „Brief: Binary robust independent elementary features“. In: *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer. 2010, S. 778–792.

[4] Cmglee. *bildpyramide*. Okt. 2022. URL: https://de.wikipedia.org/wiki/Bildpyramide#/media/Datei:Image_pyramid.svg.

[5] Martin A Fischler und Robert C Bolles. „Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography“. In: *Communications of the ACM* 24.6 (1981), S. 381–395.

[6] Kunihiko Fukushima. „Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position“. In: *Biological cybernetics* 36.4 (1980), S. 193–202.

[7] Jie Gao. „Reed solomon code“. In: *Lectures on Wireless and Mobile Networks* (2007).

[8] GPodkolzin. *D'nealian*. Jan. 2021. URL: https://de.wikipedia.org/wiki/D%E2%80%99Nealian#/media/Datei:D%E2%80%99Nealian_Manuscript.svg.

[9] Suzana Herculano-Houzel. „The human brain in numbers: a linearly scaled-up primate brain“. In: *Frontiers in human neuroscience* (2009), S. 31.

[10] J.J. Hull. „A database for handwritten text recognition research“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.5 (1994), S. 550–554. DOI: 10.1109/34.291440.

[11] Y. Lecun, L. Bottou, Y. Bengio und P. Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), S. 2278–2324. DOI: 10.1109/5.726791.

[12] U-V Marti und Horst Bunke. „The IAM-database: an English sentence database for offline handwriting recognition“. In: *International Journal on Document Analysis and Recognition* 5 (2002), S. 39–46.

[13] Anil Chandra Naidu Matcha. *Handwriting recognition with ML (an in-depth guide)*. Juni 2023. URL: https://nanonets.com/blog/handwritten-character-recognition/.

[14] Edward Rosten und Tom Drummond. „Machine learning for high-speed corner detection“. In: *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer. 2006, S. 430–443.

[15] Ethan Rublee, Vincent Rabaud, Kurt Konolige und Gary Bradski. „ORB: An efficient alternative to SIFT or SURF“. In: *2011 International Conference on Computer Vision*. 2011, S. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[16] Harald Scheidl. „Build a handwritten text recognition system using tensorflow“. In: *Medium, 09-Aug-2020* (2019).

[17] Alex Schroeder. *Numbers*. 2017. URL: https://github.com/kensanata/numbers.

[18] R. Smith. „An Overview of the Tesseract OCR Engine“. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Bd. 2. 2007, S. 629–633. DOI: 10.1109/ICDAR.2007.4376991.

[19] Renate Tost. *Ausgangsschrift*. Mai 2020. URL: https://de.wikipedia.org/wiki/Ausgangsschrift#/media/Datei:Schulausgangsschrift_1968.png.

[20] R Allen Wilkinson, Jon Geist, Stanley Janet, Patrick J Grother, Christopher JC Burges, Robert Creecy, Bob Hammond, Jonathan J Hull, NORMANW Larsen, Thomas P Vogl u. a. *The first census optical character recognition system conference*. Bd. 184. US Department of Commerce, National Institute of Standards und Technology, 1992.

[21] Yinggang Xie, Quan Wang, Yuanxiong Chang und Xueyuan Zhang. „Fast Target Recognition Based on Improved ORB Feature“. In: *Applied Sciences* 12.2 (2022). ISSN: 2076-3417. DOI: 10.3390/app12020786. URL: https://www.mdpi.com/2076-3417/12/2/786.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 4. März 2024 Fabian Mersch