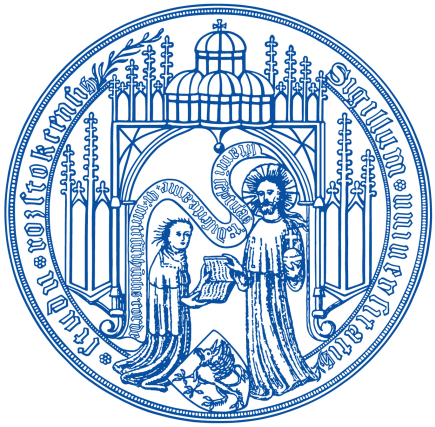
Clustering und Klassifikation von Trading Cards zur Bestimmung des Kartenspiels

Masterarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von: Tillmann Milinski
Matrikelnummer: 218202967
geboren am: 24. März 2000 in Rostock
Erstgutachter: Dr.-Ing. Hannes Grunert
Zweitgutachter: M.Sc. Felix Hauptmann
Abgabedatum: 20. April 2025

Inhaltsverzeichnis

Kapitel 1

Einleitung

Sammelkartenspiele (engl. Trading Card Games, kurz TCG) sind Kartenspiele mit einer großen, meist fortlaufend wachsenden Menge an Karten. Durch die Vielzahl an Karten und Varianten gibt es zu meist keine endgültig abgeschlossenen Kartensammlungen, sodass eine Klassifizierung komplexer ist. Die Klassifizierung zur Bestimmung der exakten Karte soll jedoch nicht Ziel dieser Arbeit sein, sondern die Bestimmung, um welches Kartenspiel es sich handelt.

Die Karten innerhalb eines Spiels haben in der Regel einen einheitlichen Aufbau und entsprechend ein ähnliches Design. Meist gibt es dabei unterschiedliche Kartenarten (z. B. Charaktere/Figuren und Energien/Fähigkeiten), die sich im Design stärker voneinander unterscheiden. Ein Großteil der Karten unterscheidet sich jedoch primär durch das abgedruckte Bild, welches meist eine Hälfte der Karte einnimmt und die Texte in den übrigen Bereichen.

Daraus stellen sich folgende Forschungsfragen für diese Arbeit:

1. Sind die Design-Features von Sammelkarten ähnlich genug innerhalb eines Kartenspiels und ausreichend unterschiedlich genug, um eine Klassifizierung zu ermöglichen?
2. Ist der Trainingsprozess zur Klassifizierung standardisierbar, sodass weitere Karten und Klassen nachträglich hinzufügbar sind?
3. Kann der Prozess sich selbst trainieren und evaluieren?
4. Kann der Prozess so designet werden, dass zukünftige Entwicklungen integrierbar sind?
5. Wie werden unbekannte Karten erkannt bzw. klassifiziert?

Diese Masterarbeit wurde durch den Lehrstuhl Datenbank- und Informationssysteme sowie dem Lehrstuhl Wirtschaftsinformatik betreut und ist eine Kooperation mit dem Start-up InSleeve, welches an einer Online-Plattform für Sammler von TCGs arbeitet.

Kapitel 2

Stand der Technik

In diesem Kapitel geht es um den Forschungsstand zur Objekterkennung und Klassifikation, welche miteinander verwandte Themengebiete sind. In der Forschung finden sich im Bezug auf die Klassifikation von Sammelkarten bisher primär Anwendung zur Objekterkennung von Beschädigungen [NIAV24] oder der Klassifizierung exakter Karten [MGC+24] bzw. spielspezifischer Eigenschaften [ZPL18].

Diese Forschungsarbeiten haben alle gemeinsam, dass sie zur Objekterkennung bzw. Klassifikation auf Convolutional Neural Networks (CNN) zurückgreifen, die für die beiden genannten Aufgaben als aktueller Stand der Technik gelten. Entsprechend fokussiert sich dieses Kapitel auf den Stand der Technik in diesen allgemeineren Forschungsgebieten, um die Grundlagen für die nachfolgenden Kapitel zu liefern, in denen diese Techniken zur Anwendung kommen.

2.1 Objekterkennung/Segmentierung

In diesem Abschnitt werden die Techniken zur Objekterkennung beschrieben. Dazu wird zunächst allgemein die Erkennung von Merkmalen (Feature Detection) in einem Bild erklärt und anschließend im Bezug auf Convolutional Neural Networks näher erläutert, da diese den aktuellen Stand der Technik darstellen.

2.1.1 Feature Detection

Ein Algorithmus zur Objekterkennung versucht in einem eingegebenen Bild ihm bekannte Objekte zu finden. Dazu muss der Algorithmus zunächst einmal lernen, wie die gesuchten Objekte aussehen.

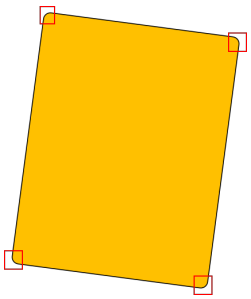


Abbildung 2.1: Vereinfachte Darstellung für Feature Detection

Der klassische Lösungsansatz für diese Aufgabe basiert auf der Ermittlung eindeutiger Features [Low04]. Features stellen dabei markante Bildstellen dar, die bspw. aus einer markanten Anhäufung von Ecken bestehen können, wie in Abb. 2.1 zu sehen.

Diese gefundenen Features werden dann weiter verarbeitet, sodass nur die deutlichsten betrachtet werden und durch diese Reduktionsschritte auch Rotationen und Skalierungen toleriert werden können. Als nächstes wird eine Funktion benötigt, die die Ähnlichkeit zwischen zwei oder mehr Featuremengen vergleichen kann, sodass die zuvor ermittelten Features mit Features bekannter Objekte (z. B. aus einer Datenbank) verglichen werden können.

Um Objekte in einem Bild nun zu finden, werden von dem Bild die Features bestimmt und mit denen in der Datenbank verglichen. Wenn eine Mindestanzahl an Features den in der Datenbank gespeicherten stark genug ähnelt, gibt der Algorithmus aus, dass in dem Bildbereich das Objekt gefunden wurde. Die Ausgabe ist dabei üblicherweise in Form einer Box, die um den ermittelten Bildbereich gezogen wird.

2.1.2 R-CNN

Region-Based Convolutional Neural Networks (R-CNN) [RHGS16] gehen anders vor, indem aus dem Bild kleinere Regionen ausgeschnitten werden und in diesen Bildausschnitten bewertet wird, wie wahrscheinlich sich eines der bekannten Objekte darin befindet.

Das Suchen der Regionen (RoI / Regions of Interest) übernimmt ein Region Proposal Network (RPN). Dazu wird zunächst ein kleinerer Bildausschnitt (Sliding Window) gewählt, in dem unterschiedlich förmige Bildausschnitte (Anchor Boxes) untersucht werden, wie in Abb. 2.2 zu sehen. Das Sliding Window wird dann über das Bild hinweg verschoben, sodass immer wieder neue Bildausschnitte untersucht werden, die sich auch mit vorherigen Anchor Boxes überschneiden können. Dabei werden die Anchor Boxes auf relevante Features untersucht. Für die weitere Suche nach Objekten müssen dann nur Anchor Boxes beachtet werden, in denen relevante Features gefunden wurden.

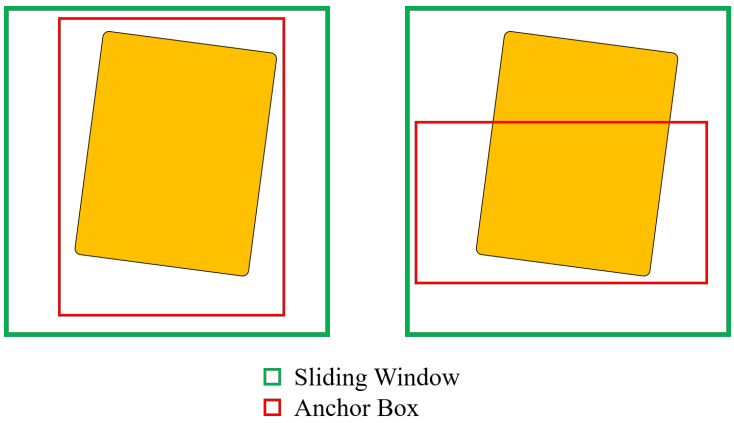


Abbildung 2.2: Vereinfachte Darstellung für das Testen unterschiedlich geformter Anchor Boxes

Beim Training wird zur Bewertung die Intersection-over-Union (IoU) verwendet. Dazu wird die Vereinigung und die Schnittmenge zwischen den Trainingsdaten und der Ausgabe des RPN gebildet, wie in Abb. 2.3 zu sehen:

$$\frac{A \cap B}{A \cup B} = IoU$$

Ein hoher IoU-Wert entsteht entsprechend, wenn Vereinigung und Schnittmenge möglichst ähnlich sind. Das bedeutet wiederum, dass die Ausgabe sehr nah an den Trainingsdaten ist.

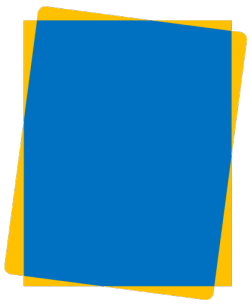


Abbildung 2.3: Beispiel für Intersection over Union zwischen 2 Formen. Die orange Fläche sollte kleinstmöglich gegenüber der blauen Fläche sein.

Beim Training des RPN werden dabei negativ Regionen betrachtet, die einen Wert von unter 0.3 haben. Regionen mit einem Wert von mehr als 0.7 werden wiederum positiv betrachtet. Die Regionen mit einem Wert zwischen 0.3 und 0.7 werden nicht zum Training verwendet.

Dieser Ansatz hilft jedoch nur Objekte ähnlicher Größe zu finden, da die Objekte innerhalb einer Anchor Box liegen müssen. Daher werden zusätzlich die Sliding Windows auf unterschiedliche Größen skaliert, um Objekte beliebiger Größe finden zu können.

Für die Klassifikation der Objekte wird ein Fast Region-based Convolutional Network (Fast R-CNN) verwendet [Gir15]. Das Fast R-CNN hat dazu eine Liste bekannter Objekte und ihrer Features, die zuvor beim Training des Netzwerkes erlernt wurden. Bei der Klassifikation werden die Features innerhalb der Anchor Box untersucht und bewertet, wie sehr diese zu den erlernten Objekten passt. Das Netzwerk gibt dann eine Wahrscheinlichkeit und eine Bounding Box zu den ermittelten Objekten aus.

Bei Faster R-CNN [RHGS16] wurden dann diese beiden Schritte miteinander verwoben. Dadurch muss das Fast R-CNN im 2. Schritt nicht erneut die Features bestimmen, sondern kann die bereits ermittelten Features aus dem RPN im 1. Schritt für die Klassifikation verwenden.

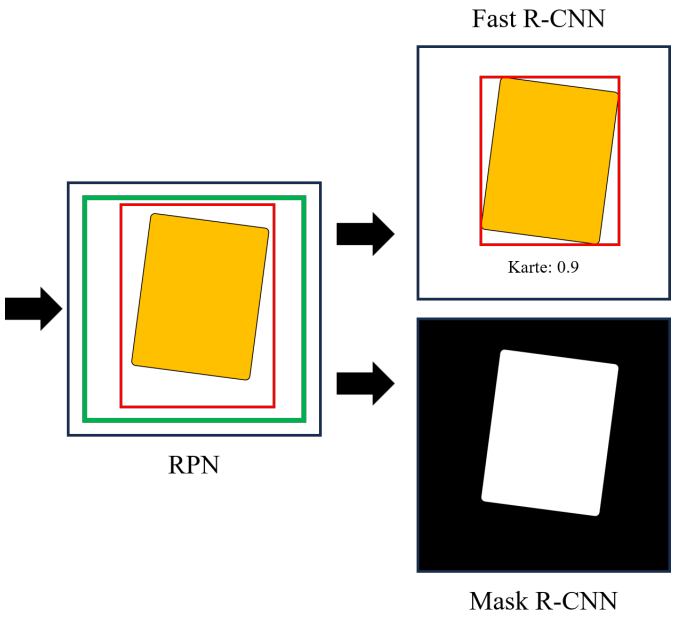


Abbildung 2.4: Prozess in Mask R-CNN

Dieses Framework wurde dann in Mask R-CNN [HGDG17] zur Image Segmentation weiterentwickelt. Dazu wird im zweiten Schritt parallel zur Klassifikation des Objektes pixelweise das Bild untersucht, um aus den Features zu bestimmen, wo das Objekt liegt, wie in Abb. 2.4 dargestellt. Jedes Pixel wird dabei mit einem Wahrscheinlichkeitswert bewertet, sodass mit Hilfe eines Schwellwerts für jede Klasse eine binäre Bildmaske erstellt werden kann. Da die Klassifikation der Anchor Box parallel stattfindet, kann nach der Klassifizierung die passende Bildmaske für das Objekt ausgegeben werden, sodass das Objekt pixelgenau aus dem Bild ausgeschnitten werden kann.

Durch die Generierung der Bildmaske kann darüber hinaus eine genauere Bounding Box erstellt werden, da diese exakt um die Maske gezogen werden kann. Eine exakte Bildmaske liefert dann entsprechend eine perfekte Bounding Box.

2.1.3 YOLO

You Only Look Once (YOLO) [RDGF16] ist ein anderer Ansatz zur Objekterkennung, der den mehrstufigen Prozess bei R-CNN reduziert.

Der Fokus bei YOLO liegt auf der Geschwindigkeit der Erkennung, um Echtzeit-Anwendungen zu ermöglichen. Dafür ist die Genauigkeit der Bounding Boxes geringer. Ein weiterer Vorteil liegt außerdem in der globalen Betrachtung des Bildes, was wiederum Fehler, die durch die Betrachtung eines Bildausschnitts entstehen, reduziert.

YOLO überspringt dazu den ersten Schritt der R-CNNs und betrachtet das gesamte Bild wie eine Anchor Box. Das neuronale Netz vereint Feature Detection, Bildung der Bounding Boxes, Klassifizierung und Bewertung in einem Schritt. Als abschließender Schritt müssen nur die Bounding Boxes aussortiert werden, die in der Bewertung den Schwellwert nicht überschreiten.

2.2 Vorverarbeitung zur Klassifikation

In diesem Abschnitt werden unterschiedliche Algorithmen erläutert, die zwischen der Object Detection und Klassifizierung verwendet werden. Das Ziel ist dabei aus dem Bildausschnitt mit der Karte, der durch die Image Segmentation bestimmt wurde, ein möglichst entzerrtes Bild der Sammelkarte für die Klassifikation zu erzeugen. Dazu werden Funktionen aus OpenCV[1] betrachtet, da die dort implementierten Algorithmen bereits auf einheitliche Datentypen standardisiert sind, die gut mit den Datentypen der KI-Modelle kompatibel sind.

2.2.1 Umwandlung der Bildmaske in ein Polygon

Das Ergebnis aus Mask R-CNN ist eine Bildmaske, in der die Sammelkarten durch weiße Pixel auf einem schwarzen Untergrund dargestellt sind. Um die Eckpunkte der Karte zu bestimmen, ist es für den folgenden Schritt notwendig, die Maske in ein Polygon umzuwandeln.

Dazu gibt es in der OpenCV-Bibliothek den Algorithmus aus [Sb85]. Dieser sucht zusammenhängende Bereiche aus weißen Pixeln und kann ggf. auch ausschließlich den äußeren Rahmen ausgeben. Auf geraden Strecken können Zwischenpunkte weggelassen werden, da diese nicht die Form des Polygons verändern.

Das Ergebnis ist ein Polygon, welches durch die vielen Unebenheiten der Bildmaske, sehr viele Eckpunkte besitzt. Daher muss für die folgenden Schritte das Polygon weiter verarbeitet werden, um auf vier Ecken reduziert zu werden.

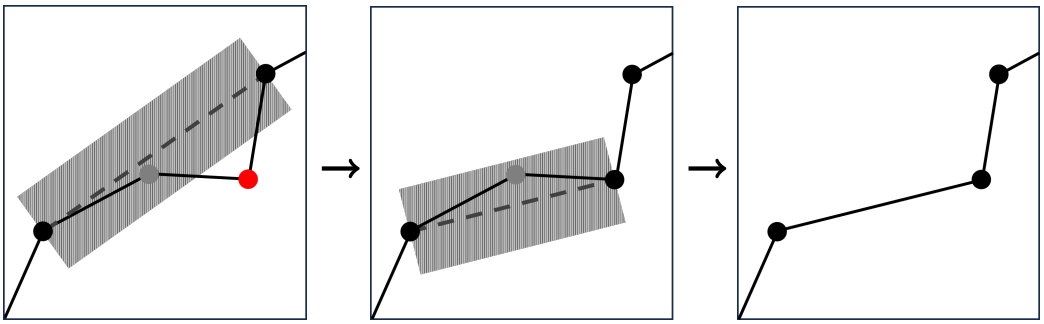
2.2.2 Glättung des Polygons

Zur Glättung des Polygons werden an dieser Stelle zwei Algorithmen vorgestellt, die auch aus OpenCV stammen. Durch die Integration in der Software-Bibliothek wurden ausschließlich diese beiden Verfahren betrachtet. In der späteren Funktion gibt es je nach Qualität der zuvor errechneten Daten Anwendungsfälle, in denen jeweils einer der beiden Algorithmen besser geeignet ist, weshalb an dieser Stelle beide Verfahren vorgestellt werden.

Der erste Algorithmus ist der Ramer-Douglas-Peucker Algorithmus [DP73], welcher die Punkte des Polygons mit Hilfe eines Schwellwerts reduziert (s. Abb. 2.5). Dazu wird zwischen einem Start- und einem Endpunkt eine Gerade gezogen (gestrichelte Linie). Zu dieser Geraden wird für die dazwischenliegenden Punkt die Distanz berechnet. Für den am weitesten entfernten Punkt wird geprüft, ob dieser innerhalb des Schwellwerts (gestrichelter Bereich in Abb. 2.5a und 2.5b) liegt. Ist dies der Fall, können alle dazwischenliegenden Punkte entfernt werden (s. Abb. 2.5b zu 2.5c). Andernfalls werden mit dem entferntesten Punkt (s. Abb. 2.5a) als Ziel- bzw. Startpunkt zu den ursprünglichen beiden Punkten zwei neue Geraden gebildet und der Vorgang wiederholt.

Diese Methode wurde ursprünglich für die Vereinfachung von Karten entwickelt, es lassen sich aber auch allgemein Polygone damit vereinfachen. Ein Nachteil dabei ist jedoch, dass immer auf bereits vorhandene Punkte zurückgegriffen wird.

[1] OpenCV-Dokumentation: `https://docs.opencv.org/4.x/`, zuletzt aufgerufen am 14.04.2025

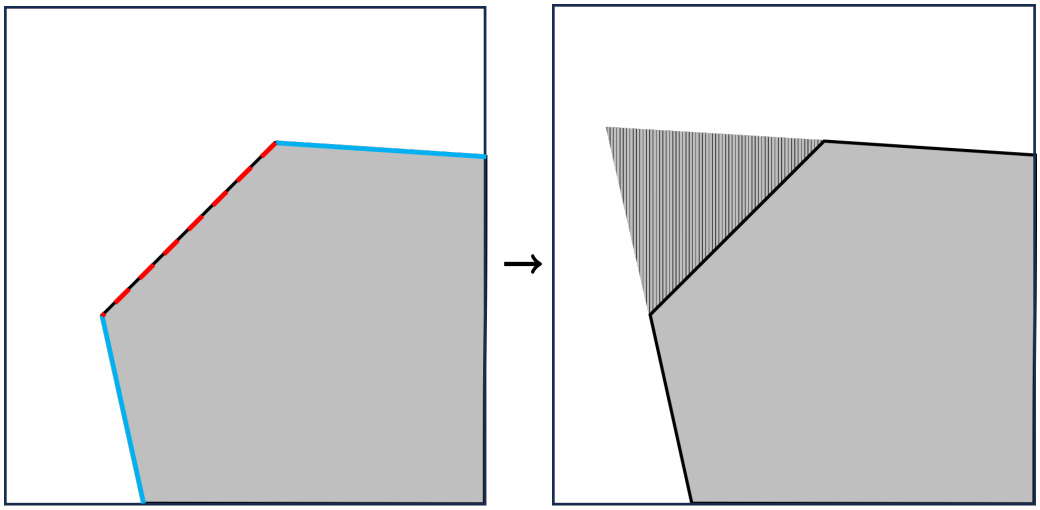


(a) Punkt außerhalb des Schwellwerts (b) Punkt innerhalb des Schwellwerts (c) geglättetes Ergebnis

Abbildung 2.5: Algorithmus nach [DP73]

Der zweite Algorithmus [Ili02] versucht hingegen mit einer gegebenen Anzahl an Kanten, eine möglichst eng anliegende Hülle um das Polygon zu ziehen (s. Abb. 2.6). Dazu werden nacheinander Kanten des Polygons entfernt, indem die anliegenden Kanten (Abb. 2.6a) bis zu dem Punkt verlängert werden, an dem diese sich kreuzen (Abb. 2.6b). Die Kanten werden dabei in der Reihenfolge entfernt, nach der sie am wenigsten den Flächeninhalt erhöhen.

Das entspricht quasi einer Bounding Box, die jedoch im Gegensatz zur Ausgabe von Mask R-CNN mit dem Polygon gedreht bzw. verzerrt sein kann, wodurch diese enger anliegt. Gegenüber Ramer-Douglas-Peucker hat dieser Algorithmus den Vorteil, dass neue Eckpunkte gebildet werden, wenn diese besser geeignet sind.



(a) Verlängerung benachbarter Kanten (b) erweitertes Polygon

Abbildung 2.6: Algorithmus nach [Ili02]

2.3 Klassifikation

Die Klassifikation von Bildern ist bereits eine Teilaufgabe der Objekterkennung. Entsprechend finden sich für diese Aufgabe ähnliche Verfahren, da Convolutional Neural Networks auch hier die besten Ergebnisse liefern. Für die Klassifikation werden jedoch eigene Modelle entwickelt, die mit Referenz-Datensätzen auf ihre Genauigkeit und Effizienz untersucht werden [CLB+21].

Neben den Modellen zur Bildklassifikation gibt es darüber hinaus auch Modelle zur Dokumentenklassifikation. Der entscheidende Unterschied dabei ist, dass die extrahierten Features nicht aus markanten Ecken und Kanten bestehen, sondern der Text mittels OCR extrahiert wird und die Worte als Features (bzw. Tokens) verwendet werden [KBLK10]. Dieses Klassifikationsverfahren ist für Sammelkarten interessant, da diese im Gegensatz zu Spielkarten in der Regel kurze Texte enthalten und sich (je nach Spiel) ggf. wiederkehrende Textelemente finden lassen können. Modelle zur Dokumentenklassifikation werden in dieser Arbeit jedoch nicht näher betrachtet, da die Überschneidungen zwischen den Karten eines Spiels nicht garantiert werden können und die Bildklassifikation für diese Aufgabenstellung besser geeignet ist.

2.3.1 Klassifikation von Bildern mit CNNs

Bei der Klassifikation von Bildern geht es im Gegensatz zur Objekterkennung darum, ein Bild aufgrund seiner Features einer Klasse zuzuordnen. Der Unterschied liegt darin, dass eine Gesamtbewertung des Bildes berechnet wird und nicht nach einzelnen Objekten innerhalb des Bildes gesucht wird. Es entfällt entsprechend der Schritt, der zur Lokalisierung des Objektes parallel zur Klassifikation durchgeführt wird. Darüber hinaus wird das Bild auch nicht in Ausschnitte geteilt, wenn mehrere Objekte erkannt werden, da eine globale Bewertung ausgegeben wird.

Trotzdem ist der allgemeine Ablauf einer Klassifikation ähnlich zur Objekterkennung. Allgemein wird in [CLB+21] der Prozess in drei Schritten bzw. Layern dargestellt. Im ersten Schritt werden mit Convolutional Layern Features extrahiert. Diese Features werden jeweils in einer Pooling Layer gefiltert, um die Komplexität zu reduzieren und abschließend im Fully Connected Layer klassifiziert.

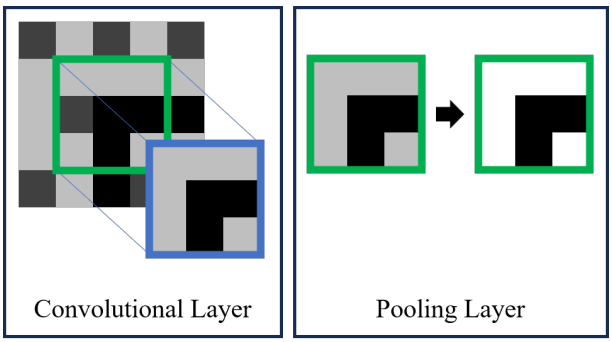


Abbildung 2.7: Vereinfachte Darstellung der Convolutional und Pooling Layer

In der Convolutional Layer werden die Features durch Verschieben von Kernels über das Bild ermittelt (Abb. 2.7). Die Kernels sind dabei quadratische Matrizen, die unterschiedliche Größen haben. In einer Matrix finden sich Gewichte, die zur Ermittlung von gesuchten Features benötigt werden und durch Backpropagation erlernt wurden. Die unterschiedlichen Größen ermöglichen es dabei, unterschiedliche Merkmale (wie Ecken, Kanten und Texturen) besser zu erkennen.

Die Pooling Layer reduziert die erkannten Merkmale, um unnötige Features zu filtern, Redundanzen zusammenzufassen und durch die Reduktion die Features auch mit leichten Veränderungen (wie Skalierungen und Rotationen) wieder zuerkennen. Dazu gibt es unterschiedliche mathematische Methoden wie Max Pooling und Average Pooling, bei denen aus den zusammenzufassenden Feldern das Maximum bzw. der Durchschnitt gebildet wird.

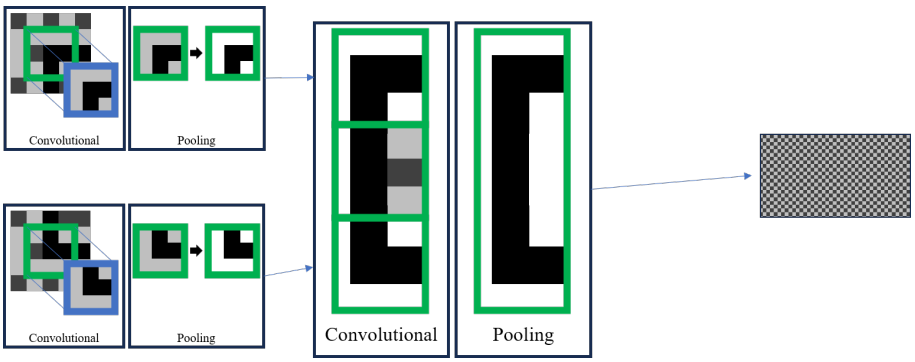


Abbildung 2.8: Convolutional und Pooling Layer finden mehrfach hintereinander statt, um die kleinen Details zu abstrakteren Features zusammenzusetzen.

Convolutional und Pooling Layers werden dabei mehrfach übereinander gelegt, wie genauer in [ON15] beschrieben, sodass von kleineren Details zu komplexeren Objekten über mehrere Ebenen hinweg eine immer größere Abstraktion erreicht werden kann, wie in Abb. 2.8 zu sehen. Da die Gewichte jeweils durch Backtracking durch das Netzwerk beim Lernen angepasst werden, wie bei der Convolutional Layer bereits erwähnt, ergibt sich schlussendlich eine deutlich reduzierte Feature-Menge, die dann als Input in der Fully Connected Layer klassifiziert wird.

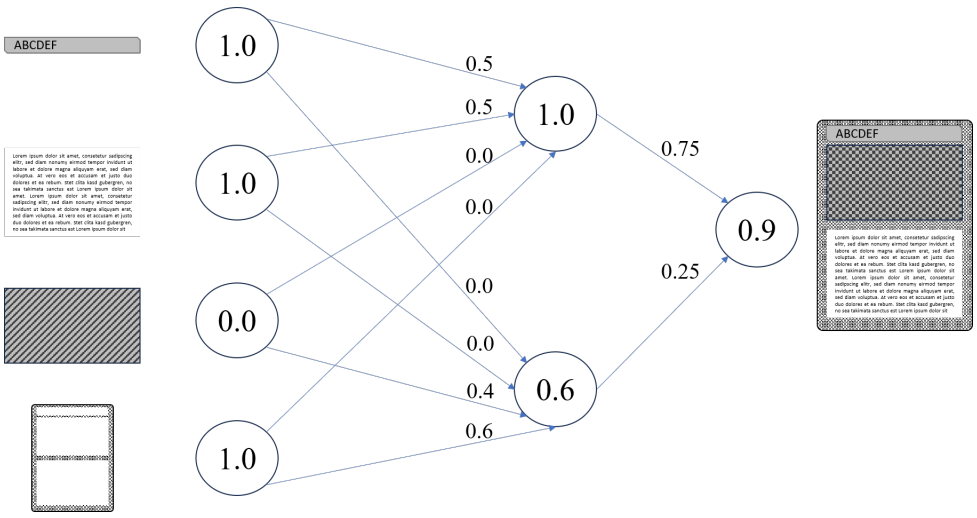


Abbildung 2.9: Vereinfachte Darstellung der Fully Connected Layer

In der Fully Connected Layer werden dann schlussendlich die Features den jeweiligen Klassen zugeordnet, auf die das Netzwerk trainiert wurde (Abb. 2.9). Diese Zuordnung wird, wie der Name bereits sagt, durch ein mehrschichtiges Netzwerk aus Knoten ermöglicht, bei denen alle Knotenpaare zwischen jeweils zwei Schichten verbunden sind. Entsprechend eines Neuronalen Netzes haben die Kanten jeweils Gewichte, die beim Training durch Backtracking so trainiert wurden, dass sich der Wert am Zielknoten der Klasse erhöht, wenn ein zur Klasse zugehöriges Feature eingegeben wurde. Folglich verringert sich der Wert auch, wenn dieses nicht vorhanden ist. Das Ergebnis ist dadurch ein Wahrscheinlichkeitswert (zwischen 0 und 1), der im jeweiligen Knoten der Klasse ausgegeben wird.

Da jede Klasse einen Zielknoten hat, kann das Gesamt-Ergebnis auch als Vektor ausgegeben werden. Der höchste Wahrscheinlichkeitswert im Vektor deutet entsprechend daraufhin, dass das eingegebene Bild zu der Klasse gehört, die zur Position im Vektor gehört. Das Ergebnis im Vektor kann durch eine Loss-Funktion noch einmal evaluiert werden, um zu bestimmen, wie sicher die Vorhersage des Netzwerks war. Wenn es bspw. mehrere ähnlich hohe Vorhersagen für verschiedene Klassen gibt, deutet dies auf ein unsicheres Ergebnis hin, was wiederum bedeutet, dass weiteres Training notwendig ist.

2.3.2 Semi-Supervised Learning

Im Bereich der Bild-Klassifizierung ist Semi-Supervised Learning eine unterstützende Technik, um den manuellen Aufwand zur Bildklassifizierung zu reduzieren. Dazu gibt es mehrere Möglichkeiten, wie in [VEH20] beschrieben. Eine der Techniken ist dabei das Self-Training, welches verwendet werden kann, wenn nur ein kleiner Anteil der Trainingsdaten klassifiziert wurde.

Dazu wird zunächst das neuronale Netz mit den klassifizierten Bildern trainiert. Nach dem Training werden die nicht-klassifizierten Bilder evaluiert und die Bilder, die eindeutig klassifiziert werden können, mit der entsprechende Klasse gelabelt. Das neuronale Netz wird nun erneut mit dem erweiterten Trainingsdatensatz trainiert und der ganze Vorgang wird wiederholt, bis keine nicht-klassifizierten Bilder übrig sind.

Diese Methode erfordert jedoch, dass das neuronale Netz eine sichere Voraussage treffen kann und die Qualität der Trainingsdaten sich nicht durch falsche Klassifikationen verschlechtert.

2.3.3 Aktuelle Algorithmen zur Bildklassifizierung mit CNNs

In [CLB+21] werden verschiedene CNNs zur Bildklassifizierung verglichen. Dabei lässt sich bereits bei dem 2012 vorgestelltem AlexNet CNN feststellen, dass durch die Verwendung von GPUs zur Parallelisierung ein technischer Stand erreicht wurde, bei dem das CNN durch selbstständiges Lernen von Features überlegen gegenüber Modellen mit manuell definierten Features war. In den darauffolgenden Jahren wurden unterschiedliche Modelle entwickelt, die unterschiedliche Optimierungen an den Netzwerken zur Beschleunigung, Parallelisierung und Verbesserung der Qualität der Ergebnisse ermöglichten.

2.3.4 Dokumentenklassifizierung

Für die Dokumenten-Klassifizierung gibt es unterschiedliche Ansätze, die sich hauptsächlich mit der Klassifizierung von eingescannten Briefen, Rechnungen und ähnlichen Geschäftsdokumenten auseinandersetzen. Dazu finden sich unterschiedliche Ansätze, die unter anderem auf eine reine Bildklassifizierung mittels CNN [KKY+14] setzen oder gemeinsam mit einer Texterkennung via OCR als zusätzlichen Input für das CNN [AHSV19] arbeiten.

Einen weiteren Ansatz ohne die Verwendung von OCR stellt Donut (Document understanding transformer) [KHY+22] dar. Dabei wird ein Dokument in Bildform in ein Encoder-Netzwerk gegeben, welches die Bilder mit einem textbasierten Decoder-Netzwerk umwandelt, sodass der Text in strukturierter Form (standardmäßig JSON) bzw. dessen Eigenschaften, wie die Klassifizierung ausgegeben werden.

Die direkte Umwandlung von Bild zu Text soll dabei bessere und schnellere Ergebnisse liefern als die der OCR-basierten Modelle, da diese einen zusätzlichen Schritt benötigen. Diese Modelle müssen zunächst den Text lokalisieren (wie bei der Objekterkennung beschrieben) und dann in diesen Bildausschnitten mit Hilfe von einem OCR-Modell den Text ermitteln.

Kapitel 3

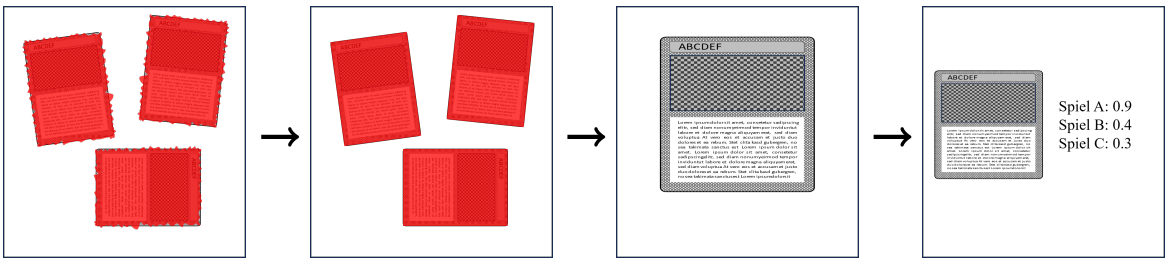
Konzept

In diesem Kapitel soll der Aufbau der Pipeline zur Erkennung und Klassifikation der Spiel- und Sammelkarten erklärt werden. Das Programm erhält als Eingabe Fotos von Sammelkarten, wobei auf einem Bild mehrere Karten vorhanden sein können, und soll als Ausgabe diese Karten mit einem Viereck markieren, welches einem Kartenspiel zugeordnet wird. Dabei sollen auch Karten von bisher unbekannten Kartenspielen erkannt und als solche klassifiziert werden.

Anschließend wird noch auf das Training der CNNs eingangen und abschließend wird das Verfahren zur Klassifizierung unbekannter Karten erklärt.

3.1 Bildverarbeitung

Dieser Abschnitt beschreibt die Schritte, in denen ein Bild in der Pipeline verarbeitet wird. Grundsätzlich würde es für diese Aufgabe ausreichen, ein CNN zur Objekterkennung zu trainieren, bei dem jedes Kartenspiel einer Klasse zugeordnet wird. Dieser Aufbau würde es jedoch nicht ermöglichen, Karten aus Kartenspielen zu finden, die bisher nicht bekannt sind. Da die Features dieser Karten keiner der Klassen zuzuordnen sind, würden diese als Hintergrund aus der Ausgabe raus fallen.



(a) Segmentierung (b) Glättung (c) Entzerrung (d) Klassifikation

Abbildung 3.1: Vier Phasen der Bildverarbeitung

Daher wird eine komplexere Pipeline gebildet, um mit Hilfe von zwei CNNs eine Pseudo-Klasse für unbekannte Karten erstellen zu können. Jedes Bild durchläuft daher nacheinander die in Abb. 3.1 dargestellten vier Schritte:

1. Objekterkennung mit Bildsegmentierung, welche allgemein nach Spielkarten sucht.
2. Glättung des Polygons zu einem Rechteck, dessen Kanten möglichst eng an den Kanten der Karte anliegen.
3. Projektion des Rechtecks auf ein Quadrat, sodass der Bildausschnitt der Karte einheitlich entzerrt wird.
4. Klassifikation des entzerrten Bildes nach Kartenspiel.

3.1.1 Objekterkennung und -segmentierung

Die Objekterkennung (Abb. 3.1a) dient primär dazu, in dem eingegebenen Bild die Anzahl und Position der Karten zu ermitteln. Die nachfolgenden Schritte nehmen dabei an, dass in dem Bildausschnitt nur eine Karte zu finden ist, die den Bildbereich ausfüllt. Entsprechend werden diese Schritte für jede der ermittelten Karten einzeln durchlaufen und am Ende zu einem Gesamtergebnis wieder zusammengeführt.

Für diese Aufgabe würde auch ein klassischer Objekterkennungsalgorithmus ausreichen, der nur eine Bounding Box ermittelt. Daraufhin müssten jedoch die Ecken und Kanten in einem nächsten Schritt bestimmt werden, wie in [LM23] vorgestellt. Algorithmen zur Eckenerkennung sind jedoch mit Sammel- und Spielkarten aufgrund der abgerundeten Ecken schwieriger anwendbar.

Alternativ dazu wird in dieser Pipeline jedoch eine Objekterkennung verwendet, welche nicht nur die Bounding Box bestimmt, sondern eine Bildmaske generiert. Diese Bildmaske liegt zunächst als Schwarz-Weiß-Rastergrafik vor, welche aber für den nächsten Schritt einfach vektorisiert werden kann.

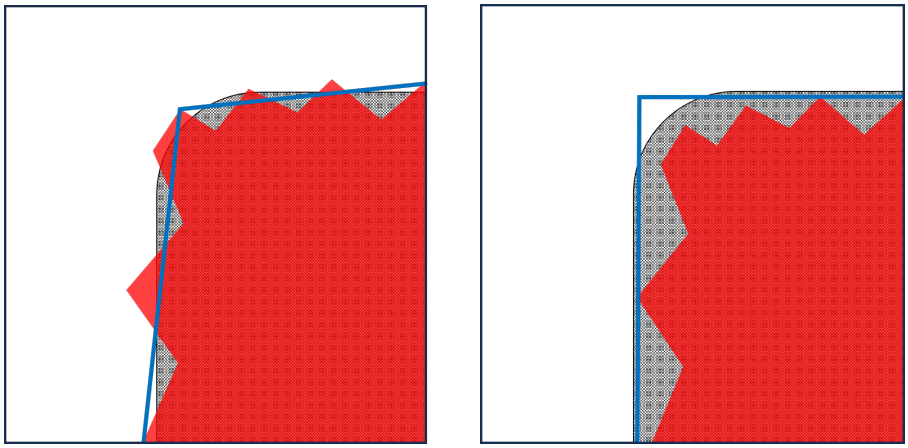
3.1.2 Glättung des Polygons

Die vektorisierte Bildmaske hat aufgrund der Bildpixel unnötig viele Eckpunkte, die in dieser Feinheit nicht notwendig sind. Da insbesondere für den nachfolgenden Schritt nur vier Eckpunkte notwendig sind, die zur Bestimmung der Kartenposition benötigt werden, wird in diesem Schritt das Polygon geglättet bzw. vereinfacht (Abb. 3.1b).

Dazu wurden in Unterabschnitt 2.2.2 bereits zwei Algorithmen vorgestellt, welche in unterschiedlichen Fällen für diese Aufgabe besser geeignet sind. Mit dem ersten Algorithmus können die Bildmasken besser geglättet werden, wenn die Unebenheiten gleichmäßig um die Kante verteilt sind, wie in Abb. 3.2a veranschaulicht. Dabei muss jedoch beachtet werden, dass Spiel- und Sammelkarten üblicherweise abgerundete Ecken haben. Da der Algorithmus nur gegebene Eckpunkte auswählen kann, wird einer der Punkte an der Abrundung ausgewählt, was zur Folge hat, dass immer ein kleiner Teil der Kartenkante abgeschnitten ist.

Der zweite Algorithmus ist besser geeignet, wenn die Polygonfläche in der Regel innerhalb der Karte liegt, wie in Abb. 3.2b zu sehen. Dieser Algorithmus kann auch besser mit den abgerundeten Kartenecken umgehen, da bei der Kantenentfernung neue Eckpunkte gebildet werden können.

Um ein Viereck zu bilden, muss beim ersten Algorithmus auch in der Regel durch mehrere Schwellwerte iteriert werden, bis das resultierende Polygon genau vier Ecken bzw. Kanten besitzt. Beim zweiten Algorithmus kann hingegen direkt die gewünschte Kantenanzahl eingegeben werden und der Algorithmus terminiert, wenn genug Kanten entfernt wurden.



(a) Glättung mit Schwellwert (b) Glättung durch Hülle

Abbildung 3.2: Vergleich der Glättungsverfahren im Kontext von Sammelkarten

3.1.3 Bildtransformation

Das entstandene Rechteck wird nun als neuer Bildausschnitt weiterverwendet, um einheitliche Eingabedaten für die Klassifikation zu erhalten. Dazu muss die Größe des Bildausschnitts untersucht werden, um sicherzustellen, dass nicht versehentliche kleine Bildflecken als eigenständige Karte erkannt wurden. Dazu wird eine Mindestkantengröße definiert, sodass die Distanz zwischen den jeweiligen Eckpunkten überprüft werden kann. Diese Methodik hat den Vorteil, dass dadurch auch abgeschnittene Karten erkannt werden können, die entsprechend eine dreieckige Form haben und dadurch in der Regel eine kurze Kante besitzen.

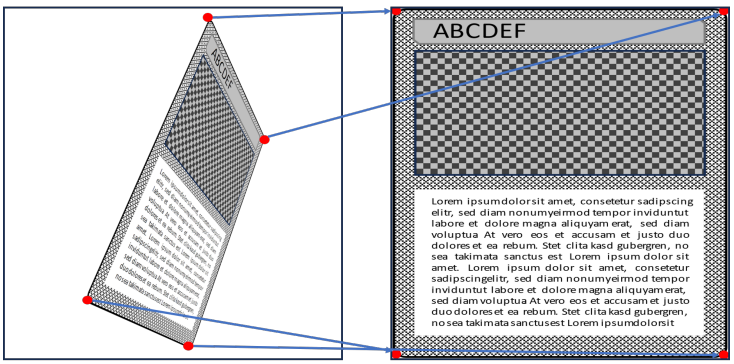
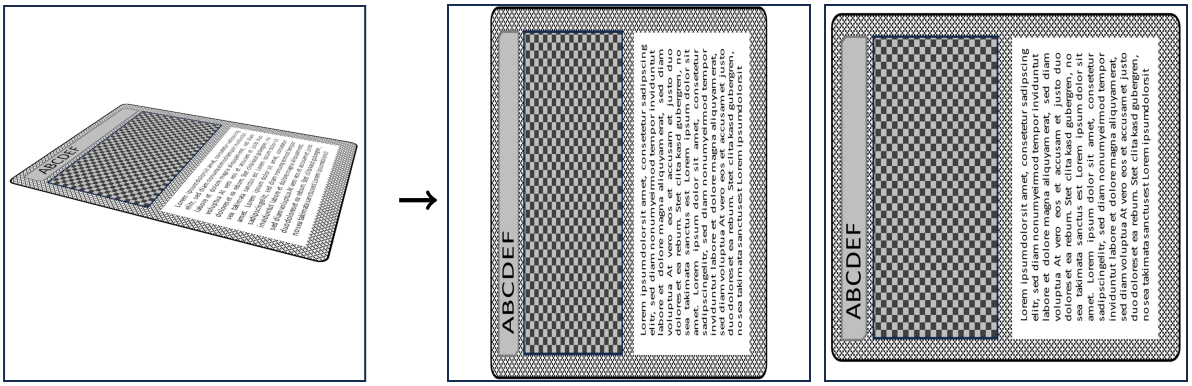


Abbildung 3.3: 4-Punkt Entzerrung

Anschließend wird ein quadratisches Bild erzeugt, indem jedem der Eckpunkte des Bildausschnitts einem Eckpunkt des Quadrats zugeordnet wird (Abb. 3.3). Mit Hilfe der Transformations-Funktionen `getPerspectiveTransform()` und `warpPerspective()` von OpenCV wird dann das Bild auf das Quadrat entzerrt (Schritt 3 bei Abb. 3.1c). Dabei verliert die Karte ihre rechteckige Original-Form. Da jedoch nicht eindeutig ist, welche Kanten des Rechtecks die kurze bzw. lange Seite bilden (aufgrund des unbekannten Aufnahmewinkels), wird so auch bei rotierten Bildern immer ein einheitliches Ergebnis (s. Abb. 3.4c) erzielt. Andernfalls könnten die langen und kurzen Seiten vertauscht werden, was die Features stark verzerren würde (s. Abb. 3.4b).



(a) Verdrehte Karte (b) Korrektes Seitenformat mit vertauschten Kanten (c) Quadratische Entzerrung

Abbildung 3.4: Beispiel einer verdrehten Karte

3.1.4 Klassifikation

Zur Klassifikation (Abb. 3.1d) wird ein zweites CNN trainiert, welches die quadratischen Bilder als Eingabedaten erhält und die Kartenspiele als Klassen und damit Ausgabewerte ausgibt. Das Klassifikations-Netzwerk gibt dabei den gesamten Vektor der Zielknoten aus, da zur Klassifikation unbekannter Karten alle Ergebniswerte relevant sind.

Als Ergebnis wird eine unbekannte Klasse ausgegeben, wenn alle Ergebnisse im Vektor unterhalb eines festgesetzten Schwellwerts liegen und die Karte damit nicht eindeutig einem Spiel bzw. einer Klasse zuzuordnen ist. Liegt mindestens ein Wert über dem Schwellwert, so wird die Klasse bzw. das Spiel mit dem höchsten Wert als Ergebnis ausgegeben.

3.2 Training der CNNs

Innerhalb des Algorithmus werden zwei verschiedene CNNs verwendet, welche vorher mit Trainingsdaten trainiert werden müssen: Eines zur Objekterkennung im ersten Schritt (Abb. 3.1a) und eines zur Klassifikation im letzten Schritt (Abb. 3.1d).

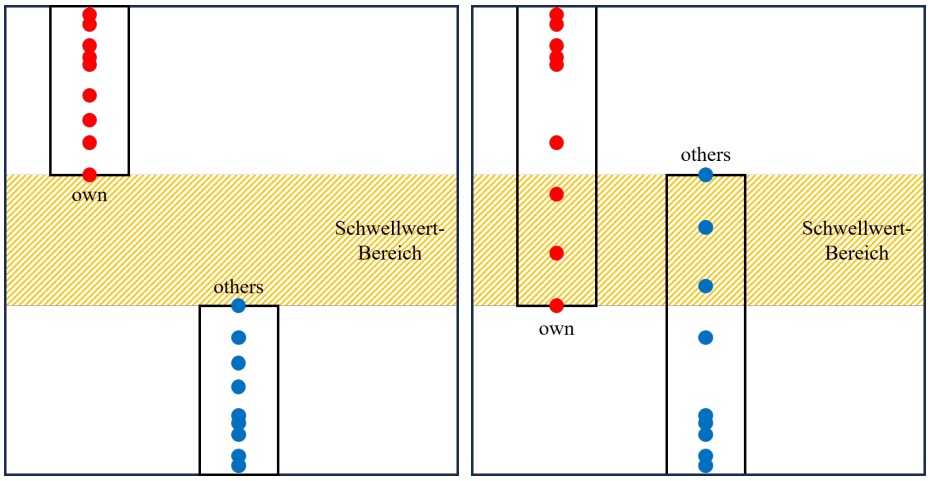
Daher ist für beide Schritte ein Trainings- und Testdatensatz notwendig, mit dem die Netzwerke trainiert werden können. Der modulare Aufbau der Pipeline erlaubt es jedoch, die Netze unabhängig voneinander zu trainieren. Lediglich zum Erstellen der Trainingsdaten für die Klassifizierung muss bereits die Pipeline bis zum vorherigen Schritt funktionieren, damit die Karten aus dem Bild bereits extrahiert werden können.

Für die Objekterkennung müssen als Trainingsdaten zu jedem Bild Objektmasken erstellt werden. Eine Objektmaske ist dabei ein Bild, welches einen schwarzen Hintergrund hat und jedes Objekt, also jede Karte, mit einem unterschiedlichen Grauwert markiert. Aus Datensicht bedeutet das, dass ein zweidimensionales Array in den Dimensionen des Bildes erstellt wird. In dem Array hat dann jedes Pixel einen Wert von 0 (Hintergrund bzw. kein Objekt) bzw. den Index des Objektes (Abb. 4.2b). Für den Index können die Objekte auf dem Bild einfach durchnummeriert werden. Da bei der Objekterkennung keine Klassifizierung notwendig ist, werden alle Objekte beim Einlesen einfach einer gemeinsamen Klasse hinzugefügt, weshalb keine Zuordnung zwischen Index und Klasse notwendig ist.

Für die Klassifizierung wiederum können die von der Pipeline generierten Bildquadrate exportiert und anschließend in Klassen einsortiert werden. Dazu muss jedem der Eingabebilder lediglich die Klasse zugeordnet werden. Dazu können die Bilder einfach in unterschiedliche Ordner sortiert werden und am Anfang des Trainings automatisiert indexiert und mit Hilfe des Ordnernamens klassifiziert werden. Dieser Aufbau ermöglicht es bei einer späteren Erweiterung der Klassen sehr einfach den Trainingsdatensatz erweitern zu können.

3.3 Klassifikation unbekannter Karten

Zur Klassifikation unbekannter Karten werden bei der Evaluation die direkt vom Klassifikationsnetzwerk ausgegebenen Daten analysiert. Da im Evaluationsdatensatz auch Karten aus unbekannten Kartenspielen enthalten sind, können deren Klassifizierungswerte beispielhaft verwendet werden. Dazu wird für jede der Klassen des Netzes untersucht, wie die Karten (sortiert nach Kartenspiel) bewertet wurden.



(a) eindeutiger Schwellwert (b) uneindeutiger Schwellwert

Abbildung 3.5: Schwellwert-Bereichsbestimmung durch Extremwerte

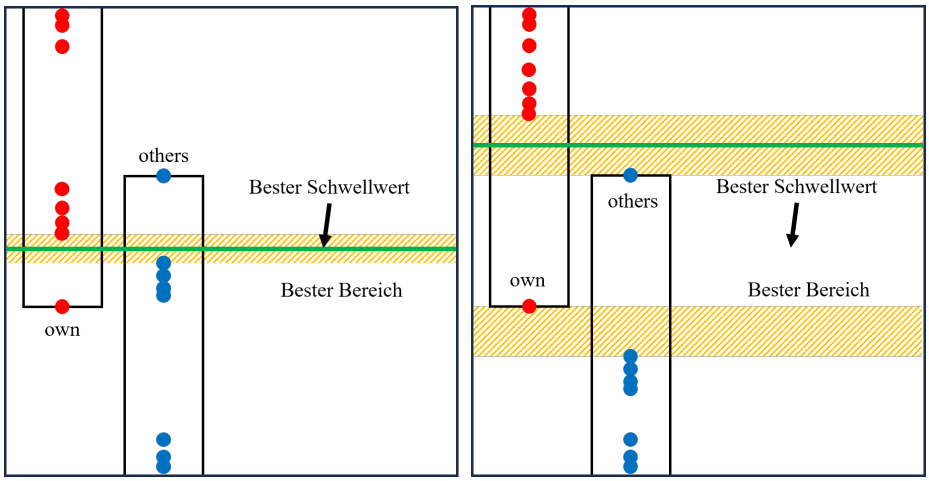
Wenn sich dabei erkennen lässt, dass die Karten, die korrekt zu dieser Klasse gehören („own"), immer eindeutig höhere Werte erhalten, als die Karten anderer Kartenspiele („others") (Abb. 3.5a), kann ein eindeutiger Schwellwertbereich bestimmt werden. Aus dem Bereich wird in diesem Fall der Mittelpunkt gebildet, um einen gleichmäßigen Abstand zwischen den Datenpunkten als Schwellwert zu erhalten.

Der Schwellwert würde dazu führen, dass wenn eine Karte dieser Klasse zugeordnet wird (weil der Wert dieser Klasse das Maximum im Ausgabevektor des Netzes ist), diese Karte nur dann der Klasse

zugeordnet wird, wenn der Wert oberhalb des Schwellwerts liegt. Liegt der Wert jedoch darunter, wird die Klasse durch die Pseudoklasse „unknown" ersetzt, auch wenn in einer anderen Klasse möglicherweise ein Wert über dem Schwellwert liegen könnte.

Im Gegensatz dazu ist der Fall schwieriger, wenn die Punkte sich nicht genau zwischen „own" und „others" aufteilen lassen und damit keine eindeutige Grenze bestimmbar ist (Abb. 3.5b). In diesem Fall wird der Bereich zwischen Minimum der „own" und Maximum der „others"-Werte, sowie jeweils zu den nächsten angrenzenden Punkten um den Bereich untersucht. Dazu werden alle Datenpunkte in dem Bereich erfasst, um mögliche Schwellwertpunkte zu bestimmen (da ein bestmglicher Schwellwert immer auf oder zwischen einem Datenpunkt liegt). Als möglicher Schwellwert zwischen zwei Datenpunkten wird wieder der Mittelpunkt gebildet.

Die erzeugte Liste an möglichen Schwellwertpunkten wird dann durchgetestet und nach Anzahl der fehlerhaft klassifizierten Datenpunkte bewertet. Dabei müssen jedoch nur die zuvor ermittelten Datenpunkte innerhalb der Region getestet werden, da alle außerhalb liegenden Punkte bereits korrekt eingeordnet werden und somit keinen Fehler erzeugen können. Die Schwellwertpunkte mit der niedrigsten Anzahl an falsch einsortierten Datenpunkten werden dann gesammelt und es wird überprüft, ob mehrere dieser Punkte nebeneinander liegen und somit eine gemeinsame Region bilden. Sollte nur auf exakt einem Punkt der Bestwert liegen, wird dieser Start- und Endpunkt der Region.



(a) bester Schwellwert zwischen Extrempunkten (b) bester Schwellwert über Extrempunkt

Abbildung 3.6: Schwellwertbestimmung bei Überschneidung

Aus den neuen Regionen wird dann die größtmögliche Region ermittelt, deren Mittelpunkt dann der bestmögliche Schwellwert ist (Abb. 3.6). Dabei kann diese Region entweder innerhalb des untersuchten Bereichs liegen, wenn viele Punkte innerhalb des Überschneidungsbereichs liegen (Abb. 3.6a) oder es gibt nur sehr wenige Ausreißer, wodurch eine geringere Fehlerzahl ober- bzw. unterhalb des Bereichs erreicht werden kann (Abb. 3.6b).

Kapitel 4

Implementierung

Die Implementierungen in dieser Arbeit sind mit Python umgesetzt, da für Python mit PyTorch[1] und OpenCV[2] bereits Frameworks für einzelne Teilaufgaben existieren. Die Implementierung teilt sich, wie im Kapitel 3 bereits, in die Pipeline, welche bei der Ausführung verwendet wird, und das Training der beiden Neuronalen Netze.

4.1 Programmiersprachen und Frameworks

In diesem Abschnitt werden kurz die in diesem Projekt verwendeten Tools und Programmiersprachen aufgelistet:

Programmiersprache
Python 3.12.1

Framework für neuronale Netze
PyTorch: 2.5.1+cu124
Torchvision: 0.20.1+cu124

Framework für Algorithmen aus der Zwischenverarbeitung
OpenCV 4.11.0.86

Framework , welches Datenstrukturen für OpenCV und PyTorch erzeugt
numpy 2.1.3

Framework zum Generieren von Grafiken
matplotlib 3.9.3

Tool zum Zeichnen und Sammeln der Trainingsdaten zur Objekterkennung
label-studio

[1] PyTorch-Dokumentation: `https://pytorch.org/docs/stable/index.html`, zuletzt aufgerufen am 14.04.2025
[2] OpenCV-Dokumentation: `https://docs.opencv.org/4.x/`, zuletzt aufgerufen am 14.04.2025

4.2 Pipeline

In der Funktion `simplifiedPipeline()` (Algorithmus 4.1) wird ein Bild eingelesen, welches durch die vortrainierten neuronalen Netze verarbeitet wird und abschließend eine Liste aus Karten ausgegeben. Für jede Karte werden die Eckpunkte des umliegenden Vierecks und das erkannte Kartenspiel bzw. `unknown` für unbekannte Karten ausgegeben.

Quellcodeausschnitt 4.1: Algorithmus der gesamten Pipeline

```
1  def simplifiedPipeline(imgLocation, classifierName, classifierFolder,
   ↪ classes, thresholdMethod, thresholds):
2      img = readImageFromFile(imgLocation)
3
4    # Bildsegmentierung
5      pred = generatePredictions(img, loadObjDetectorModel())
6
7    # Umwandlung der Predictions in eine Segmentierungsmaske
8      mask = generateOutputMaskImage(img, pred)
9
10   # Vektorisierung der Bildmaske
11     polygonList = convert_mask_to_polygon(mask)
12
13   # Reduzierung der Eckpunkte
14     polygonList = approxCardOuter(polygonList)
15
16   # Filterung nach Kantenlänge
17     polygonList = filterPolysByEdgeLengths(polygonList)
18
19   # Zuschneiden der Bilder
20     transformedImages = transformImageToCards(imgLocation, polygonList,
       ↪ size=classifier.getResizeForNet(classifierName))
21
22   # Klassifizierung
23     classification = getClassifications(transformedImages,
       ↪ loadClassificationModel(classifierName, folder=classifierFolder),
       ↪ classifier.getTransformForNet(classifierName), classes,
       ↪ thresholdMethod, thresholds)
24
25   result = []
26
27     for i in range(len(polygonList)):
28         result.append({
29             "box": polygonList[i].tolist(),
30             "class": classification[i]
31         })
32
33     return result
```

Datenstruktur 4.2: Ausgabe von `simplifiedPipeline()`

```
[
  {
    "box": [
      [Int, Int],        # Eckpunkt der Karte im Eingabebild
      ...                # für jede Karte werden vier Eckpunkte ausgegeben
    ],
    "class": <Klasse>    # Name der Klasse
  },
  ...
]
```

4.2.1 Bildsegmentierung

Der Code für die Bildsegmentierung basiert auf der Implementierung von PyTorch [PyT24a], welche auf den Anwendungsfall reduziert und angepasst wurde. Für die Bildsegmentierung wird mit den Funktionen aus Ausschnitt 4.3 das vortrainierte PyTorch-Modell geladen und das Bild evaluiert. PyTorch verarbeitet das Bild in Form eines „Tensors", welches ein mehrdimensionales Array ist, das jedoch auf der GPU schneller verarbeitet werden kann. Das Bild wird dazu als dreidimensionales Array eingelesen (RGB-Farbkanäle und XY-Koordinaten der Pixel).

Quellcodeausschnitt 4.3: Objekterkennung

```
def loadObjDetectorModel():
    model = torch.load('object_detector.pth', weights_only=False)
    return model.eval()

def readImageFromFile(path):
    return read_image(path)

def generatePredictions(image, model):
    with torch.no_grad():
        x = eval_transform(image)
        # convert RGBA -> RGB and move to device
        x = x[:3, ...].to(device)
        predictions = model([x, ])
        thisPred = predictions[0]
    return thisPred
```

Die Prediction wird anschließend durch die Funktion `generateOutputMaskImage()` (Ausschnitt 4.4) zu einem Schwarz-Weiß-Bild umgewandelt. Die Funktion `pred["masks"] > 0.7` wandelt dabei den bisher als Float ausgegebenen Wert in einen Boolean-Wert um und stellt praktisch einen Schwellwert dar. Das Pixel muss entsprechend mit einer Wahrscheinlichkeit von mind. 70 % als dem Objekt zugehörig vom CNN bewertet worden sein. Der Schwellwert lässt sich ändern, sodass ein höherer Schwellwert zu weniger falsch erkannten Objekten, dafür jedoch kleineren Bildmasken führt. Der aktuelle Wert (70 %) wurde dabei als Standardeinstellung beibehalten, da diese Grenze auch beim Training (s. Abschnitt 2.1.2) üblich ist und mit einem höheren Wert (80 %) zu wenig von der Kartenfläche erkannt wurde.

Ein Problem in den ersten Tests der Programmierung war, dass sich in einigen Fällen das CNN unsicher war, wie sich eine unbekannte Karte zusammensetzt und dabei überschneidende Bildmasken erzeugt hat, wie in Abb. 4.1 zu sehen ist. Es wurde daher eine Optimierung durchgeführt, welche annimmt, dass der Nutzer die Karten separiert voneinander fotografiert, sodass aus überlappenden Bildmasken eine Vereinigung gebildet wird. Bei übereinanderliegenden Karten hätte dies jedoch zur Folge, dass diese als zusammenhängendes Objekt weiterverarbeitet werden.

Abbildung 4.1: Bildmaske aus dem anfänglichen Training

Für die Optimierung werden durch die Funktion `any(0)` die booleschen Werte aus den Dimensionen der erkannten Objekte miteinander durch Disjunktion (OR) zu einer Dimension reduziert. Der PyTorch eigenen `draw_segmentation_masks()`-Funktion wird dadurch der Tensor so übergeben, als wäre nur ein Objekt erkannt worden. Dieses Objekt besitzt überall dort ein Pixel, wo in mindestens einer der Dimensionen ein `True` berechnet wurde.

Quellcodeausschnitt 4.4: Erzeugung der Bildmasken

```
1 def generateOutputMaskImage(image, pred):
2     size = image.shape
3     output_image = torch.from_numpy(np.zeros((size[0], size[1], size[2]),
      ↪ np.uint8))
4     output_image = draw_segmentation_masks(output_image[:3, ...],
      ↪ (pred["masks"] > 0.7).any(0).squeeze(1), alpha=1, colors="white")
5     return output_image
```

4.2.2 Zwischenverarbeitung

In der Zwischenverarbeitung geht es darum, die als Rastergrafik vorliegende Bildmaske in ein Polygon umzuwandeln und dieses soweit zu vereinfachen, dass es nur noch aus vier Punkten besteht. Diese Eckpunkte des Vierecks werden anschließend zur Entzerrung der Karte auf ein standardisiertes Quadrat verwendet.

In die Funktion `convert_mask_to_polygon()` (Ausschnitt 4.5) wird eine Bildmaske eingegeben und zunächst aus dem Tensor-Format in ein Graustufenbild umgewandelt. Für die Umwandlung in ein Polygon wird die Funktion `findContours()` aus OpenCV verwendet. Diese gibt eine Liste an Polygonen zurück, welche wiederum als eine Liste aus Eckpunkt-Koordinaten ausgegeben wird.

Der Parameter `cv2.RETR_EXTERNAL` führt in der Funktion dazu, dass bei verschachtelten Polygonen, die bspw. ein Loch enthalten, nur das äußerste Polygon ausgegeben wird. Durch den Parameter `cv2.CHAIN_APPROX_SIMPLE` werden bereits Eckpunkte weggelassen, die zwischen zwei Eckpunkten in einer geraden Linie liegen und dadurch sowieso keinen Einfluss auf die Form des Polygons haben. Ausgegeben werden am Ende nur die Liste der Polygone, da die Hierarchie unwesentlich ist. Daher wird nur das erste Element aus `findContours()` ausgegeben.

Quellcodeausschnitt 4.5: Umwandlung der Rastergrafik in ein Polygon

```
1 def convert_mask_to_polygon(im):
2     im = im.permute(1, 2, 0).numpy()
3     imgray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
4     contours = cv2.findContours(imgray, cv2.RETR_EXTERNAL,
      ↪ cv2.CHAIN_APPROX_SIMPLE)
5     poly = contours[0]
6     return poly
```

Anschließend stehen für die Reduktion der Eckpunkte zwei Algorithmen zur Auswahl, welche im Abschnitt 3.1.3 bereits ausführlicher erklärt wurden. Der erste Algorithmus, welcher die bestehenden Eckpunkte reduziert, wurde in der Funktion `approxCard()` (Ausschnitt 4.6) implementiert. In der zweiten Funktion `approxCardOuter()` (Ausschnitt 4.7) findet sich die Implementierung des anderen Algorithmus, welcher die Fläche des Polygons erweitert, bis eine viereckige Hülle entstanden ist.

Die Funktionen ähneln sich dabei in ihrer Vorgehensweise: In einem Array werden alle reduzierten Polygone gesammelt, die bei der Iteration durch die eingegebenen Polygone entstanden sind. Die Polygone müssen dazu exakt vier Ecken aufweisen, da die weiteren Schritte ein Viereck zur Verarbeitung erwarten. Diese Eigenschaft ist bei Sammelkarten üblicherweise gegeben, sodass durch diese Filterung nur fehlerhafte Erkennungen kleiner Pixelflecken verloren gehen.

In der Funktion `approxCard()` wird das Polygon solange mit einem schrittweise ansteigenden Schwellwert geglättet, bis das Polygon nur noch vier Ecken besitzt. Der Bereich durch den iteriert wird, ist dabei absichtlich großzügig gewählt, damit Karten auch bei einer sehr groben Bildmaske korrekt geglättet werden können. Die Schleife bricht jedoch bei einem vorzeitigen Ergebnis bereits vorher ab. Es wurde dabei bewusst keine unendlich laufende Schleife verwendet, damit bei fehlerhaften Bildmasken eine zügige Terminierung sichergestellt ist und diese Bildmaske durch den anschließenden Test verworfen wird.

Quellcodeausschnitt 4.6: Erster Algorithmus zur Reduzierung der Polygoneckpunkte

```
def approxCard(poly):
    finished = []
    for i in poly:
        oldLen = len(i)
        newApprox = None
        squareEps = None
        for j in range(10, 15000, 10):
            step = j / 100
            newApprox = cv2.approxPolyDP(i, step, True)
            if len(newApprox) <= 4:
                squareEps = step
                break
        if len(newApprox) == 4:
            finished.append(newApprox)
    return finished
```

Für die zweite Funktion `approxCardOuter()` ist diese schrittweise Iteration nicht notwendig, sodass direkt vom Algorithmus das gesuchte Ergebnis berechnet werden kann und anschließend nur noch auf die korrekte Eckenzahl überprüft wird. Dabei ist jedoch vorher zu prüfen, ob das Rechteck bereits vier Ecken besitzt oder bei geringer Eckenanzahl verworfen wird, da von einer unvollständig erfassten Karte oder einem Artefakt auszugehen ist.

Quellcodeausschnitt 4.7: Zweiter Algorithmus zur Reduzierung der Polygoneckpunkte

```
def approxCardOuter(poly):
    finished = []
    for i in poly:
        oldLen = len(i)
        if oldLen > 4:
            try:
                newApprox = cv2.approxPolyN(i, 4, )
            except:
                print("APPROX_CARD_OUTER ERROR ON", i)
            else:
                finished.append(newApprox[0])
        elif oldLen == 4:
            finished.append([i[0][0], i[1][0], i[2][0], i[3][0]])
    return finished
```

Die Kanten des Vierecks werden anschließend noch einmal in ihrer Länge überprüft, sodass sowohl stark verzerrte, als auch zu kleine Vierecke rausgefiltert werden. In beiden Fällen ist davon auszugehen, dass keine vollständige Karte erkannt bzw. das Bild so stark entzerrt werden muss, dass die Features im Bild nicht mehr genau erkennbar sind. Dazu wird eine Mindestkantengröße `x` definiert, welche von jeder der Kanten erreicht werden muss. Sobald eine der Kanten kleiner ist, wird das Viereck nicht weiter in der Pipeline verarbeitet.

Die Funktion `calcDistance()` (Ausschnitt 4.8) berechnet dazu die Distanz zwischen zwei Punkten nach dem Satz von Pythagoras. Durch die Funktion `calcEdgeLengths()` wird diese Distanz für jede der Kanten des Vierecks berechnet, sodass diese in der Funktion `filterPolysByEdgeLengths()` zur Überprüfung der eingegebenen Vierecke verwendet werden kann. Zurückgegeben wird damit nur eine Liste an Vierecken, die die Mindestanforderungen an alle Kanten erfüllen.

Quellcodeausschnitt 4.8: Filterung nach Kantenlänge

```
1  def calcDistance(p1, p2):
2    return round(math.sqrt(((p1[0] - p2[0]) ** 2) + ((p1[1] - p2[1]) ** 2)))
3
4  def calcEdgeLengths(i):
5    return [calcDistance(i[0], i[1]), calcDistance(i[1], i[2]),
     ↪ calcDistance(i[2], i[3]), calcDistance(i[3], i[0])]
6
7  def filterPolysByEdgeLengths(polys, x=150):
8    finished = []
9    for p in polys:
10     dists = calcEdgeLengths(p)
11     edgeToShort = False
12     for i in dists:
13       if i <= x:
14         edgeToShort = True
15         break
16     if not edgeToShort:
17       finished.append(p)
18   return finished
```

Um das Bild zu entzerren, werden durch die Funktion `transformImageToCards()` (Ausschnitt 4.9) zwei Algorithmen aus OpenCV genutzt, die die Entzerrung zwischen Ausgangs- und Zielbild berechnen. Dazu wird mit `cv2.getPerspectiveTransform()` zunächst eine Transformationsmatrix berechnet, welche durch `cv2.warpPerspective()` angewendet wird. Die Quellkoordinaten sind dabei die vier Eckpunkte des Vierecks, während die Zielkoordinaten aus den Eckpunkten des Zielquadrats gebildet werden. Das Zielquadrat hat dabei immer eine einheitliche Größe für alle Kartenbilder (welche im Parameter `size` übergeben wird), sodass alle Karten auf ein einheitliches Format entzerrt werden.

Quellcodeausschnitt 4.9: Erzeugung der Bildausschnitte

```
1  def transformImageToCards(imageLocation, inputCoordList, size=300,
   ↪ reverse=False):
2      imgFile = cv2.imread(imageLocation)
3      if reverse:
4          destCoords = np.float32([[0, 0], [0, size - 1], [size - 1, size -
           ↪ 1], [size - 1, 0]])
5      else:
6          destCoords = np.float32([[0, 0], [size - 1, 0], [size - 1, size -
           ↪ 1], [0, size - 1]])
7      transformedImages = []
8      for i in inputCoordList:
9          srcCoords = np.float32([i[0], i[1], i[2], i[3]])
10         matrix = cv2.getPerspectiveTransform(srcCoords, destCoords)
11         transformedImages.append(cv2.warpPerspective(imgFile, matrix,
           ↪ (size, size), flags=cv2.INTER_LINEAR))
12     return transformedImages
```

4.2.3 Klassifikation

Für die Klassifikation werden die entzerrten Bilder verwendet, welche durch die Vorverarbeitung als quadratische Bilder vorliegen. Mit der Funktion `loadClassificationModel()` (Ausschnitt 4.10) wird das ausgewählte neuronale Netz in den Zwischenspeicher geladen. Die Klassifizierungsfunktionen `rawClassification()` und `getClassifications()` werden in der Pipeline ausgeführt, wenn die Bildausschnitte des Eingabebildes klassifiziert werden.

Die Funktion `rawClassification()` lädt dazu die Bildausschnitte in ein Tensor-Objekt, sodass sie im Zwischenspeicher mit dem Klassifizierungsnetz liegen. Die Klassifizierung wird durch die Zeile `torch.narrow(classificationModel(classifyTensor), 1, 0, len(classes))` durchgeführt. Dabei wird der gerade erstellte Tensor vom Modell bewertet und die Bewertung auf die Anzahl der Klassen gekürzt (falls ein vortrainiertes Netz verwendet wurde, dessen Klassenanzahl nicht reduziert wurde).

Anschließend werden in `classification`, `classificationScore` und `classificationSoftMax` die Ergebnisse für die Bildausschnitte berechnet. Da alle Bildausschnitte gleichzeitig bewertet wurden, sind in diesen Variablen Arrays gespeichert, wobei der Index im Array dem Index des Bildausschnitts entspricht.

In `classification` wird die vom CNN berechnete Klasse durch die `argmax`-Funktion berechnet, indem diese den Index des maximalen Werts zurückgibt. In `classificationScore` werden die direkten Werte je Klasse vom CNN gespeichert, während in `classificationSoftMax` durch die `softMax`-Funktion aus `torch.nn.Softmax(dim=1)` die relative Wahrscheinlichkeit aus allen Werten berechnet wird. Diese Werte sind wichtig zur Bestimmung, ob eine Karte einer unbekannten Klasse angehört, was im Abschnitt 4.5 genauer erklärt wird.

Die gesammelten Ergebnisse werden am Ende der Funktion in einem Array den einzelnen Bildern zugeordnet, welches dann an die Funktion `getClassifications()` übergeben wird. Dort wird das Ergebnis auf den Namen der Klasse reduziert. Sofern Schwellwerte in der Variable `thresholds` berechnet und übergeben wurden, kann die Funktion mit Hilfe des Schwellwerts der Klasse bewerten, ob die Karte wahrscheinlich genug dazugehört und andernfalls die Klasse `unknown` ausgeben.

Quellcodeausschnitt 4.10: Klassifizierung

```
1  def loadClassificationModel(model_name, folder=""):
2      path = folder + 'classifier_' + model_name + '.pth'
3      model = torch.load(path, weights_only=False)
4      model.to(device)
5      return model.eval()
6
7  def rawClassification(transformedImages, classificationModel,
   ↪ classifyTransformer, classes):
8      softMax = torch.nn.Softmax(dim=1)
9      classifyTensor = torch.Tensor().to(device)
10     for i in transformedImages:
11         classifyTensor = torch.cat((classifyTensor,
           ↪ classifyTransformer(i).unsqueeze(0).to(device)))
12     classify = torch.narrow(classificationModel(classifyTensor), 1, 0,
       ↪ len(classes))
13     classification = classify.argmax(1).tolist()
14     classificationScore = classify.tolist()
15     classificationSoftMax = softMax(classify).tolist()
16     results = []
17     for i in range(len(transformedImages)):
18         thisScores = {}
19         thisSoftMax = {}
20         for classId in classes:
21             thisScores[classes[classId]] =
               ↪ classificationScore[i][int(classId)]
22             thisSoftMax[classes[classId]] =
               ↪ classificationSoftMax[i][int(classId)]
23         results.append({
24             "Score": thisScores,
25             "SoftMax": thisSoftMax,
26             "predicted": classes[str(classification[i])]
27         })
28     return results
29
30 def getClassifications(transformedImages, classificationModel,
   ↪ classifyTransformer, classes, thresholdMethod, thresholds):
31     returnClasses = []
32     for i in rawClassification(transformedImages, classificationModel,
       ↪ classifyTransformer, classes):
33         thisClass = i["predicted"]
34         if thresholdMethod == "SoftMax" and thresholds[thisClass] is not
           ↪ None and i["SoftMax"][thisClass] <= thresholds[thisClass]:
```

```
35              returnClasses.append("unknown")
36          elif thresholdMethod == "Score" and thresholds[thisClass] is not
            ↪ None and i["Score"][thisClass] <= thresholds[thisClass]:
37              returnClasses.append("unknown")
38          else:
39              returnClasses.append(thisClass)
40      return returnClasses
```

4.3 Training

In der Pipeline werden zwei verschiedene CNNs verwendet, welche vor der Verwendung trainiert werden müssen. Um diese Neuronalen Netze unabhängig voneinander trainieren zu können, wurden jeweils eigene Trainingsdaten und -programme erstellt. Dadurch können die CNNs auch weiter optimiert oder komplett ausgetauscht werden.

Zur Vereinheitlichung in dieser Implementierung wird PyTorch bzw. TorchVision als Framework verwendet, in dem bereits einige Funktionen und Modelle implementiert wurden. PyTorch kann sowohl auf der CPU als auch (NVIDIA-)GPU laufen, wobei letzteres eine deutliche Beschleunigung ermöglicht.

4.3.1 Objekterkennung

Zur Objekterkennung wird Mask R-CNN verwendet, für das es durch PyTorch bereits eine Referenzprogrammierung [PyT24a] gibt. Die Implementierung (Ausschnitt C.1) wurde auf einen eigenen Trainingsdatensatz angepasst und die Anzahl der Klassen wurde auf eine reduziert, da an dieser Stelle allgemein alle Sammelkarten erkannt werden sollen.

Zur Erstellung der Trainingsdaten wurde Label-Studio[3] verwendet, welches es ermöglicht Objekte in importierten Bildern zu markieren. Da für das Training möglichst realistische Bilder verwendet werden sollten, wurden reale Karten auf unterschiedlichen Hintergründen fotografiert. Auf den Bildern waren jeweils eine ein- bis zweistellige Anzahl an separat voneinander liegenden Karten zu sehen, so wie sie auch in einem Sammelalbum zu finden wären. Auf den Bildern wurde anschließend in Label-Studio ein Polygon aus acht Ecken erstellt (um aufgrund der abgerundeten Ecken möglichst nah am Umriss der Karte zu bleiben). Die Bildersammlung wurde anschließend im COCO-Format[4] exportiert, wodurch zu der Bildersammlung auch eine passende JSON-Datei erstellt wurde.

In der JSON-Datei finden sich die Metadaten zu den Bildern, insbesondere Dateiname, Breite und Höhe, sowie die Koordinaten der manuell erstellten Polygone. Diese wurden verwendet, um in der Funktion `generateMaskImg()` (Ausschnitt 4.11) Bildmasken (Abb. 4.2b) zu generieren, bei denen die Objekte in unterschiedlichen Grautönen auf schwarzem Hintergrund markiert sind. Die unterschiedlichen Grautöne dienen dazu, die Objekte voneinander abzugrenzen, wenn diese zu nah aneinander liegen.

[3] Dokumentation: `https://labelstud.io/guide/get_started`, zuletzt aufgerufen am 15.04.2025

[4] Format, welches vom COCO-Datensatz verwendet wird. `https://cocodataset.org/`, zuletzt aufgerufen am 18.04.2025

Quellcodeausschnitt 4.11: Generierung der Bildmasken

```
1 def generateMaskImg(width, height, polygonList):
2   maskImg = np.zeros((width, height, 1), np.uint8)
3   counter = 0
4   for k in polygonList:
5     cv2.fillPoly(maskImg, pts=[np.array(k)], color=[counter])
6     counter += 1
7   return maskImg
```

Mit diesen generierten Masken wird das CNN zunächst einmal mit einer kleinen Menge an Trainingsdaten trainiert, bis die ausgegebenen Daten (nach Optimierung durch die Verarbeitungsschritte in der Pipeline) nah an den manuell markierten Masken liegen. Die weiteren Trainingsdaten wurden dann mit Hilfe des CNNs selbst generiert.

4.3.2 Selbsttrainierende Objekterkennung

Mit Hilfe der Funktionen aus der Pipeline lässt sich auch eine Objekterkennungs-Funktion bauen, die vor der Klassifikation endet (Algorithmus C.3). Die Ausgaben sind dabei Vierecke, die eine die Karten umgebende Box (Abb. 4.3a) ergeben. Diese Vierecke können wiederum erneut verwendet werden, um Bildermasken (Abb. 4.3b) zu erzeugen. Dadurch lassen sich bei einem bereits gut trainierten Netzwerk neue Trainingsdaten einfach erzeugen, was bei einer späteren Anwendung mit Nutzerdaten zum automatisierten Training verwendet werden kann.

Der Prozess sollte jedoch überwacht werden, da das CNN bei anfänglichem Training fehlerhafte Masken erzeugen kann, bzw. neue Spiele nicht garantiert korrekt erkannt werden (Abb. 4.1) oder zu qualitativ schlechte Bilder in die Trainingsdaten gelangen. Insbesondere ist dabei das Verhalten bei überlappenden Polygonen abzuwägen. In der Pipeline wird angenommen, dass der Nutzer seine Karten getrennt voneinander hingelegt hat, sodass Überlappungen zu einem gemeinsamen Polygon vereinigt werden (siehe Abschnitt 4.2.1).

(a) Beispielbild (b) generierte Bildmaske

Abbildung 4.2: Maskengenerierung aus manuell markierten Karten

(a) Polygon auf Karte markiert (b) von Pipeline errechnete Bildmaske

Abbildung 4.3: automatische Maskengenerierung durch die Funktionen aus der Pipeline

4.4 Klassifikation

Für die Klassifikation stehen unterschiedliche Modelle in `torchvision.models` bereit, die sich dadurch leicht zum Training importieren lassen. Es ist jedoch auch möglich, sein eigenes CNN als Klasse zu definieren, wie in [PyT24b] vorgestellt und im Ausschnitt 4.12 gezeigt.

Quellcodeausschnitt 4.12: Definition eines eigenen CNN zur Klassifizierung

```
1  class OwnCNN(nn.Module):
2    def __init__(self, classes):
3      super().__init__()
4      self.conv1 = nn.Conv2d(3, 60, 5)
5      self.pool = nn.MaxPool2d(2, 2)
6      self.conv2 = nn.Conv2d(60, 16, 5)
7      self.fc1 = nn.Linear(16 * 72 * 72, 120)
8      self.fc2 = nn.Linear(120, 84)
9      self.fc3 = nn.Linear(84, classes)
10
11   def forward(self, x):
12     x = self.pool(F.relu(self.conv1(x)))
13     x = self.pool(F.relu(self.conv2(x)))
14     x = torch.flatten(x, 1)  # flatten all dimensions except batch
15     x = F.relu(self.fc1(x))
16     x = F.relu(self.fc2(x))
17     x = self.fc3(x)
18     return x
```

Das zugehörige Trainingsdatenset besteht aus durch die Pipeline generierten Bildern, die in einer Auflösung von 300×300 Pixeln vorliegen. Einige der PyTorch-Modelle benötigen jedoch eine Auflösung von 224×224 Pixeln, sodass für diese das Bild vorher runterskaliert werden muss. PyTorch liefert außerdem vortrainierte Gewichtungen für das Netzwerk, welche auf dem ImageNet-Datensatz[5] trainiert wurden. Dieser Datensatz beinhaltet Bilder, die in 1000 Klassen vorsortiert wurden, weshalb die vortrainierten Netze auf eine Ausgabe von 1000 Klassen vorkonfiguriert sind.

Diese Konfiguration lässt sich dadurch anpassen, dass die letzte Fully-Connected-Layer mit einer angepassten Klassenanzahl neu initialisiert und trainiert wird. Dadurch können die vortrainierten Gewichte in den vorherigen Ebenen für bessere Trainingsergebnisse verwendet werden, da das Zusammensetzen von Features nicht komplett neu trainiert werden muss. Andererseits können aber die Features neu den Klassen des Karten-Datensatzes zugeordnet werden, sodass die letzte Ebene (welche die Features den Klassen zuordnet) ohne Bias durch den ImageNet-Datensatz trainiert ist.

Das Ergebnis der Klassifikation ist ein Vektor aus den Knotenwerten der letzten Fully-Connected Layer. Das bedeutet, dass es für jede der trainierten Klassen einen Wahrscheinlichkeitswert gibt. Die übliche Methode zur Bestimmung der Klasse ist, aus diesen Werten das Maximum zu bestimmen und durch die Position im Vektor die zugehörige Klasse zu ermitteln und auszugeben. Diese Methode ist gut geeignet zur Bestimmung der Klasse, wenn alle möglichen Klassen bekannt sind.

[5] `https://image-net.org/`, zuletzt aufgerufen am 18.04.2025

Eine andere Methode, die etwas mehr über die Genauigkeit des Ergebnisses aussagt, ist diese Werte in Relation zueinander zu setzen. Mit Hilfe der `softMax()`-Funktion werden die Werte miteinander verglichen und für jede der Klassen eine relative Wahrscheinlichkeit ausgegeben, bei der alle Wahrscheinlichkeiten der Klassen aufsummiert 1.0 bzw. 100% ergeben. Diese Methode ist besser geeignet, um Unsicherheiten zwischen bekannten Klassen zu erkennen, da unsichere Ergebnisse zu einem niedrigeren Wert beim Maximum führen. Dafür lassen sich jedoch nicht mehr Unsicherheiten auf die einzelne Klasse bezogen erkennen, da ein klassenfremdes Bild auch durch leichte Ähnlichkeiten zu einer bestehenden Klasse eine hohe relative Ähnlichkeit hat.

4.5 Klassifizierung unbekannter Karten

Für die Auswahl des besten Klassifikations-Netzwerks soll neben der Korrektheit der Ergebnisse bei korrekten Karten auch die Bewertung unbekannter Karten evaluiert werden. Daher werden für den Evaluationsdatensatz auch Karten von Kartenspielen aufgenommen, auf die das Netzwerk zuvor nicht trainiert wurde. Die Evaluation soll neben dem besten Netzwerk auch die bestmöglichen Schwellwerte für jede Klasse ausgeben und ermitteln, ob die relativen Wahrscheinlichkeiten aus der `softMax()`-Funktion oder die absoluten Wahrscheinlichkeiten direkt aus dem Vektor besser geeignet sind.

Dazu wird ein mehrstufiger Prozess entwickelt, um die Komplexität der Aufgabe zu reduzieren und durch die Teilschritte bzw. -ergebnisse eine vollständige, aufwendige Neuevaluierung nicht immer notwendig zu machen. Die Teilergebnisse müssen jedoch vollständig neu berechnet werden, wenn Anpassungen an dem Netzwerk gemacht werden, da dies die Evaluierung des Datensatzes verändert.

Der Evaluierungsprozess sieht folgende Schritte vor, auf deren Implementierung (Ausschnitt C.5) nachfolgend genauer eingegangen wird:

classifyTestFiles() vollständige Klassifizierung aller Trainingsdaten, aufgeschlüsselt nach tatsächlicher Klasse

analyzeResults() Auswertung der gesammelten Daten zur Aufschlüsselung nach tatsächlicher und berechneter Klasse

accumulateResults() Zusammenfassen der Ergebnisse je Klasse nach richtigen und falschen Klassifikationen

calculateThresholdArea() Bestimmung des möglichen Schwellwert-Bereichs aus Maximum der falsch klassifizierten Karten und Minimum der korrekt klassifizierten Karten

- **analyzeThresholdProblem()** Bestimmung eines bestmöglichen Schwellwerts, wenn keine perfekte Lösung möglich ist

evaluateThreshold() Bewertung des ermittelten bestmöglichen Schwellwerts

calculateRanking() Bewertung der Klassifikationsnetze nach falschen Bewertungen

Diese Schritte werden für alle trainierten Modelle und Varianten (mit/ohne Gewichten bei Initialisierung bzw. mit neu-initialisierter Full-Connected Layer) durchgeführt, sodass am Ende ein Ranking aller verfügbaren Netze entsteht.

4.5.1 Klassifizierung der Testdaten

Mit der Funktion `classifyTestFiles()` (Algorithmus C.6, Ausgabe 4.13) werden für den eingegebenen Evaluierungsdatensatz und das Klassifizierungsnetz alle Evaluierungsdaten gesammelt. Dazu werden alle Bilder nacheinander in den Zwischenspeicher geladen und von dem Modell klassifiziert.

Die gemessene Zeit bezieht sich dabei ausschließlich auf die Dauer der Klassifizierung und beinhaltet nicht das Laden der Bilder. Der errechnete Vektor wird dann mit Hilfe der `torch.nn.Softmax()`-Funktion in ein relatives Ergebnis umgewandelt und beide Ergebnisse werden für die weiteren Schritte gespeichert.

Datenstruktur 4.13: Ausgabe von `classifyTestFiles()`

```
{
  <Tatsächliche Klasse>: [
    {
      "score": {
        "<berechnete Klasse>": Float, # berechneter Wahrscheinlichkeitswert
        ...
      },
      "softMax": {
        "<berechnete Klasse>": Float, # relativer Wahrscheinlichkeitswert
        ...
      },
      "prediction": <Klasse>,         # berechnete Klasse
      "duration": Float               # Dauer der Klassifizierung
    },
    ...
  ],
  ...
}
```

4.5.2 Auswertung der gesammelten Daten

Die vorherigen Daten werden dann durch die Funktion `analyzeResults()` (Algorithmus C.7, Ausgabe 4.14) nach tatsächlicher und berechneter Klasse gruppiert. Dadurch ergibt sich eine erste Statistik, die jedoch in den weiteren Schritten noch stärker zusammengefasst wird. Dabei werden für die berechneten Klassen jeweils immer ein Minimum, Maximum und Durchschnitt berechnet. Mit Hilfe der Variable `length` kann in den folgenden Schritten die Berechnung des Durchschnitts weiter zusammengefasst werden.

Datenstruktur 4.14: Ausgabe von `analyzeResults()`

```
{
  <Tatsächliche Klasse>: {
    "length": Int             # Anzahl der Karten
    "evaluations": {
      <berechnete Klasse> : {
        "minScore": Float,    # Minimum der gefundenen Wahrscheinlichkeiten
        "maxScore": Float,    # Maximum der gefundenen Wahrscheinlichkeiten
        "avgScore": Float,    # Durchschnitt der gefundenen
        ↪ Wahrscheinlichkeiten
        "minSoftMax": Float,  # Minimum der gefundenen relativen
        ↪ Wahrscheinlichkeiten
        "maxSoftMax": Float,  # Maximum der gefundenen relativen
        ↪ Wahrscheinlichkeiten
        "avgSoftMax": Floaf,  # Durchschnitt der gefundenen relativen
        ↪ Wahrscheinlichkeiten
      },
      ...
    },
    "predictable": Boolean,   # gibt an, ob diese Klasse unter den
    ↪ trainierten Klassen ist
    "minDuration": Float,     # Minimum der Klassifizierungsdauerwerte
    "maxDuration": Float,     # Maximum der Klassifizierungsdauerwerte
    "avgDuration": Float,     # Durchschnitt der Klassifizierungsdauerwerte
  },
  ...
}
```

4.5.3 Zusammenfassen der Ergebnisse je Klasse

Die Funktion `accumulateResults()` (Algorithmus C.8, Ausgabe 4.15) sortiert die Daten nach Klassen, um alle Extremwerte für die Bestimmung des Schwellwerts zusammenzufassen. An dieser Stelle werden unter `others` alle berechneten Werte zusammengefasst, die für diese Klasse unter allen anderen tatsächlichen Klassen gefunden wurden. `own` beschreibt die Werte, die für diese Klasse unter allen tatsächlich zu dieser Klasse dazugehörenden Karten gefunden wurden.

Datenstruktur 4.15: Ausgabe von `accumulateResults()`

```
{
  <Klasse>: {
    "own": {                # Werte für alle Karten, die tatsächlich zu
    ↪ dieser Klasse gehören
      "minScore": Float,    # Minimum der gefundenen Wahrscheinlichkeiten
      "maxScore": Float,    # Maximum der gefundenen Wahrscheinlichkeiten
      "avgScore": Float,    # Durchschnitt der gefundenen
      ↪ Wahrscheinlichkeiten
      "minSoftMax": Float,  # Minimum der gefundenen relativen
      ↪ Wahrscheinlichkeiten
      "maxSoftMax": Float,  # Maximum der gefundenen relativen
      ↪ Wahrscheinlichkeiten
      "avgSoftMax": Float,  # Durchschnitt der gefundenen relativen
      ↪ Wahrscheinlichkeiten
      "totalCards": Int,    # Anzahl der Karten
    },
    "others": {             # Werte für alle Karten, die tatsächlich nicht
    ↪ zu dieser Klasse gehören
      "minScore": Float,    # Minimum der gefundenen Wahrscheinlichkeiten
      "maxScore": Float,    # Maximum der gefundenen Wahrscheinlichkeiten
      "avgScore": Float,    # Durchschnitt der gefundenen
      ↪ Wahrscheinlichkeiten
      "minSoftMax": Float,  # Minimum der gefundenen relativen
      ↪ Wahrscheinlichkeiten
      "maxSoftMax": Float,  # Maximum der gefundenen relativen
      ↪ Wahrscheinlichkeiten
      "avgSoftMax": Float,  # Durchschnitt der gefundenen relativen
      ↪ Wahrscheinlichkeiten
      "totalCards": Int,    # Anzahl der Karten
    }
  },
  ...
}
```

4.5.4 Bestimmung des Schwellwert-Bereichs

Nach allen Zusammenfassungen der Messwerte kann nun durch die Funktion `calculateThresholdArea()` (Algorithmus C.9, Ausgabe 4.16) für jede Klasse ein Bereich bestimmt werden, in dem der ideale Schwellwert liegt. Dazu werden der Maximum-Wert der `others`-Klassen und der Minimum-Wert der `own`-Klasse verwendet. Dabei ist entscheidend, welcher Wert größer ist:

Ist das Minimum der `own`-Klasse größer als das Maximum aus den `others`-Klassen, so gibt es einen Wertebereich dazwischen, in dem im Evaluations-Datensatz keine Messwerte aufgetaucht sind und der bestmögliche Schwellwert kann in die Mitte dieses Bereichs gesetzt werden. Dies entspricht der Darstellung in Abbildung 3.5a.

Wenn das Minimum von `own` jedoch kleiner ist als das Maximum von `others`, gibt es im Bereich dazwischen Werte, die nicht eindeutig einer der beiden Gruppen zugeordnet werden können. Daraus ergibt sich die in Abbildung 3.5b dargestellte Situation, sodass der ideale Schwellwert nur angenähert werden kann und die Suche nach dem Schwellwert zu einem Optimierungsproblem wird.

Das Ergebnis dieses Tests wird in der Variable `safeThreshold` durch ein `True` (Abb. 3.5a: eindeutiger Schwellwert) bzw. `False` (Abb. 3.5b: uneindeutiger Schwellwert) ausgegeben.

Datenstruktur 4.16: Ausgabe von `calculateThresholdArea()`

```
1  {
2    <Klasse>: {
3      "totalCards": Int,                # Anzahl der Karten (diese sollte
       ↪ der Gesamtanzahl des Evaluations-Datensatzes entsprechen)
4      "score": {
5        "safeThreshold": Bool,          # gibt an, ob der Schwellwert
         ↪ eindeutig ist
6        "thresholdArea": (Float, Float) # gibt den Bereich an, in dem der
         ↪ Schwellwert liegen kann
7      },
8      "softMax": {
9        "safeThreshold": Bool,
10       "thresholdArea": (Float, Float)
11     }
12   },
13   ...
14 }
```

4.5.5 Bestimmung eines bestmöglichen Schwellwerts

Wenn der Schwellwert im vorherigen Schritt uneindeutig ist (s. Abb. 3.5b), wird die Funktion `classifyThresholdProblem()` (Algorithmus C.10) aufgerufen, die potentielle Schwellwerte aus den in diesem Bereich liegenden Messwerten berechnet nach der Methode aus Abschnitt 3.3. Dazu werden die Werte aus allen Messungen für die untersuchte Klasse gesammelt und danach gefiltert, ob diese zwischen den Extrempunkten liegen (Abb. 3.6a) bzw. an diese anliegen (Abb. 3.6b).

Die gesammelten Messpunkte werden sortiert, doppelte Messpunkte entfernt und zusätzliche Zwischenpunkte zwischen den gemessenen Punkten gebildet. Jeder dieser Punkte wird anschließend als potentieller Schwellwert untersucht und es wird berechnet, wie viele Messpunkte falsch zugeordnet werden. Die Messpunkte mit den geringsten Fehlern werden gesammelt und darauf untersucht, ob sie direkt nebeneinander liegen, wodurch sie einen gemeinsamen Bereich bilden. Abschließend wird für den größten Bereich der Mittelpunkt gebildet, welcher als bester Schwellwert ausgegeben wird.

4.5.6 Evaluierung des bestimmten Schwellwerts

Zur Überprüfung wird der berechnete Schwellwert noch einmal mit allen im ersten Schritt erhobenen Messwerten überprüft. Dazu wird von der Funktion `evaluateThreshold()` (Algorithmus C.11, Ausgabe 4.17) eine Evaluation mit der üblichen Einteilung in True/False Positive/Negative durchgeführt, wie in Tabelle 4.1 zu sehen.

	gehört tatsächlich zur Klasse	**gehört tatsächlich nicht zur Klasse**
wurde der Klasse zugeordnet	True Positive	False Positive
wurde nicht der Klasse zugeordnet	False Negative	True Negative

Tabelle 4.1: Einteilung der Klassifizierungsevaluation

Anhand des berechneten Schwellwerts werden noch einmal alle Messwerte überprüft und entsprechend des Ergebnisses gezählt. Zusätzlich werden noch die Precision- und Recall-Werte für die Klasse ausgegeben. In der Bestimmung der Klasse sollen jedoch falsche Klassifikationen ausschlaggebend sein. Daher wird dafür im nächsten Schritt die Anzahl der Fehler (False Positive + False Negative) evaluiert.

Zur Klassifikation wird zunächst die Klasse mit dem höchsten Wert bestimmt. Wenn für diese Klasse der Schwellwert nicht erreicht wurde, gehört das evaluierte Bild keiner Klasse an und wird als unbekannt klassifiziert, auch wenn möglicherweise eine schlechter bewertete Klasse durch einen niedrigeren Schwellwert zugeordnet werden könnte.

Datenstruktur 4.17: Ausgabe von `evaluateThreshold()`

```
{
  <Klasse>: {
    "totalCards": Int,                  # Anzahl der Karten (diese sollte
    ↪ der Gesamtanzahl des Evaluations-Datensatzes entsprechen)
    "score": {
      "safeThreshold": Bool,            # gibt an, ob der Schwellwert
      ↪ eindeutig ist
      "thresholdArea": (Float, Float),  # gibt den Bereich an, in dem der
      ↪ Schwellwert liegen kann
      "recommendedThreshold": Float,    # der berechnete ideale Schwellwert
      "evaluation": {
        "truePositive": Int,            # Anzahl korrekt dieser Klasse
        ↪ zugeordnete Karten
        "falsePositive": Int,           # Anzahl falsch dieser Klasse
        ↪ zugeordnete Karten
        "trueNegative": Int,            # Anzahl korrekt dieser Klasse
        ↪ nicht zugeordnete Karten
        "falseNegative": Int,           # Anzahl falsch dieser Klasse
        ↪ nicht zugeordnete Karten
        "precision": Float,             # truePositive / (truePositive +
        ↪ falsePositive)
        "recall": Float                 # truePositive / (truePositive +
        ↪ falseNegative)
      }
    },
    "softMax": {
      "safeThreshold": Bool,
      "thresholdArea": (Float, Float),
      "recommendedThreshold": Float,    # der berechnete ideale Schwellwert
      "evaluation": {
        "truePositive": Int,            # Anzahl korrekt dieser Klasse
        ↪ zugeordnete Karten
        "falsePositive": Int,           # Anzahl falsch dieser Klasse
        ↪ zugeordnete Karten
        "trueNegative": Int,            # Anzahl korrekt dieser Klasse
        ↪ nicht zugeordnete Karten
        "falseNegative": Int,           # Anzahl falsch dieser Klasse
        ↪ nicht zugeordnete Karten
        "precision": Float,             # truePositive / (truePositive +
        ↪ falsePositive)
        "recall": Float                 # truePositive / (truePositive +
        ↪ falseNegative)
      }
    }
  },
  ...
}
```

4.5.7 Bewertung aller Klassifikatoren

Zur Bewertung der Netzwerke werden diese nacheinander von den Funktion in Ausschnitt C.12 evaluiert und nach der Anzahl der Fehler (False Positive + False Negative) sortiert. Jedes Netzwerk wird dabei einmal mit dem SoftMax- und dem Score-Wert evaluiert, sodass am Ende eine sortierte Liste mit empfohlenen Parametern entsteht (Ausgabe 4.18).

Datenstruktur 4.18: Ausgabe von `orderRanking()`

```
[
  {
    "place": Int,                 # Platzierung
    "mistakes": Int,              # Anzahl der Fehler
    "modelName": <Modelname>,     # Metadaten des Models
    "trainingType": <"mitGewichten" |
    ↪ "mitGewichtenUndAngepassterKlassenanzahl" | "ohneGewichte">,
    "evalType": <SoftMax | Score>,
    "duration": {                 # Messwerte aus der Evaluation
      "minDuration": Float,
      "maxDuration": Float,
      "avgDuration": Float
    },
    "thresholds": {               # berechnete Schwellwerte
      <Klasse>: Float,
      ...
    }
  },
  ...
]
```

Kapitel 5

Evaluation

In diesem Kapitel werden die trainierten CNNs mit einem Evaluationsdatensatz ausgewertet. Dabei soll auch untersucht werden, wie die Modelle mit unbekannten Karten aus nicht trainierten Kartenspielen umgehen.

5.1 Objekterkennung

Zur Evaluation der Objekterkennung wurde zwischen einem manuell gezeichneten achteckigen Polygon und den Berechnungen der Pipeline die Intersection-over-Union gebildet, wie in Abschnitt 2.1.2 bereits vorgestellt (Algorithmus C.2). Dabei wurde einerseits direkt zwischen dem manuell gezeichneten Polygon und dem Ergebnis von Mask R-CNN verglichen (CNN-Prediction), als auch nach den Optimierungsschritten, die die Prediction zu einem Viereck umwandeln. Für einen fairen Vergleich wurde dabei das manuell gezeichnete Polygon auch zu einem Viereck optimiert.

Wie in Tabelle 5.1 zu sehen, sind die Ergebnisse bei den nicht-trainierten Kartenspielen erkennbar schlechter. Insgesamt lässt sich beobachten, dass die berechneten Objektmasken kleiner als die tatsächlichen sind. Dies ist jedoch nicht für die Klassifizierungsergebnisse kritisch, da nur Teile des äußeren Rahmens abgeschnitten werden. Aus den Ergebnissen ist außerdem zu erkennen, dass die Zwischenverarbeitungsschritte das Ergebnis weiter verbessern.

Tabelle 5.1: Evaluationsergebnisse der Objekterkennung

	CNN-Prediction			auf Viereck optimiert			
Kartenspiel	Min	Avg	Max	Min	Avg	Max	Karten
Pokemon	87.54 %	88.86 %	90.31 %	90.97 %	92.25 %	93.39 %	23
Final Fantasy	90.96 %	91.42 %	91.88 %	94.21 %	94.45 %	94.69 %	18
Spielkarten[1]	87.2 %	87.79 %	88.38 %	90.21 %	91.5 %	92.79 %	16
Yu-Gi-Oh	86.34 %	87.14 %	87.95 %	90.04 %	91.0 %	91.97 %	16
Magic – The Gathering*	82.1 %	84.34 %	86.64 %	84.06 %	86.43 %	88.54 %	20
Disney Lorcano*	80.58 %	82.04 %	83.02 %	83.91 %	85.24 %	86.1 %	50

* Kartenspiel ist nicht Teil des Trainingsdatensatzes gewesen.
[1] Spielkarten beschreibt ein französisches Blatt mit 55 Karten, d.h. inkl. Jokern

5.2 Polygonkanten-Filterung

An dieser Stelle wird evaluiert, wie lang die Seiten der Polygone in den Testdaten sind, um eine Mindestgrenze für die Filterung zu bestimmen. Aus Tabelle 5.2 geht hervor, dass die kleinste erkannte Kantenlänge 163px lang war. Es wird daher empfohlen, eine Mindestkantenlänge von 150 Pixeln einzustellen, damit alle Karten erkannt und kleine Artefakte gefiltert werden.

Tabelle 5.2: Evaluationsergebnisse der erfassten Kantenlängen

Minimum	**163px**
Durchschnitt	516px
Maximum	1472px
Kantenanzahl	572

5.3 Klassifikation

Zur Evaluierung der Klassifikation wurden aus den Trainingsbildern die Bildausschnitte für die Karten generiert und nach Klassen einsortiert. Mit diesen Daten wurden dann alle zuvor trainierten Klassifizierungsnetze getestet, mögliche Schwellwerte zur Bestimmung unbekannter Klassen errechnet und nach abschließender Evaluation mit den Schwellwerten nach Anzahl der Fehler (False Positive + False Negative) sortiert (s. Abschnitt 4.5).

Die Schwellwerte wurden mit den Evaluationsdaten berechnet, da im Trainingsdatensatz alle Karten einer bekannten Klasse angehört haben und somit unbekannte Klassen erst an dieser Stelle evaluiert werden können. Des Weiteren ist zu beachten, dass im Evaluationsdatensatz nur beispielhaft Karten aus zwei unbekannten Spielen aufgenommen wurden, um das Verhalten der Klassifizierung zu evaluieren. Aufgrund der Vielzahl an unbekannten Karten und Kartenspielen kann jedoch nicht allgemeingültig eine korrekte Klassifikation unbekannter Kartenspiele sichergestellt werden.

Alle Klassifizierungsnetze wurden mit den gleichen Parametern trainiert: 20 Durchläufe mit dem `torch.optim.SGD`-Optimizer. Die dabei gemessene durchschnittliche Dauer bezieht sich auf den reinen Klassifizierungsvorgang und bezieht nicht mit ein, dass das Netzwerk und Bild in den Zwischenspeicher geladen werden müssen. Die Netzwerke wurden aus den in PyTorch verfügbaren Netzwerken nach den dort angegebenen Messwerten[1] zur Klassifikation des Imagenet-Datensatzes und der Größe des Netzwerks ausgewählt. Die Netzwerke wurden auf drei Arten trainiert und gemessen:

Ohne Gewichte Das Netzwerk wurde ohne vortrainierte Daten initialisiert.

Mit Gewichten Das Netzwerk wurde mit den Klassen und Gewichten aus PyTorch initialisiert. Diese Gewichte wurden durch den Imagenet-Datensatz trainiert. Beim Training wurden die ersten Klassen den Kartenspielen zugeordnet und die Ergebnisse der weiteren Klassen bei der Evaluation verworfen.

Mit angepassten Gewichten Das Netzwerk wurde mit den Gewichten aus PyTorch initialisiert. Vor dem Training wurde die letzte Fully-Connected-Layer durch eine neuinitialisierte Version ersetzt, deren Zielknotenanzahl auf die Anzahl der Klassen angepasst wurde. Dadurch können die vortrainierten Gewichtungen innerhalb des Netzwerks verwendet werden und lediglich die letzte Schicht wird komplett neu trainiert.

[1] siehe `https://pytorch.org/vision/main/models.html`, zuletzt aufgerufen am 15.04.2025

Das Ergebnis der Evaluation zeigt, dass im Vergleich die mobilenet_v2 Varianten mit angepassten Gewichten bzw. mit den Standard-Gewichten am besten abgeschnitten haben. Die vollständigen Ergebnisse finden sich in den Tabellen B.1 und B.2. An dieser Stelle sollen noch einmal die Messergebnisse von mobilenet_v2 mit angepassten Gewichten genauer betrachtet werden.

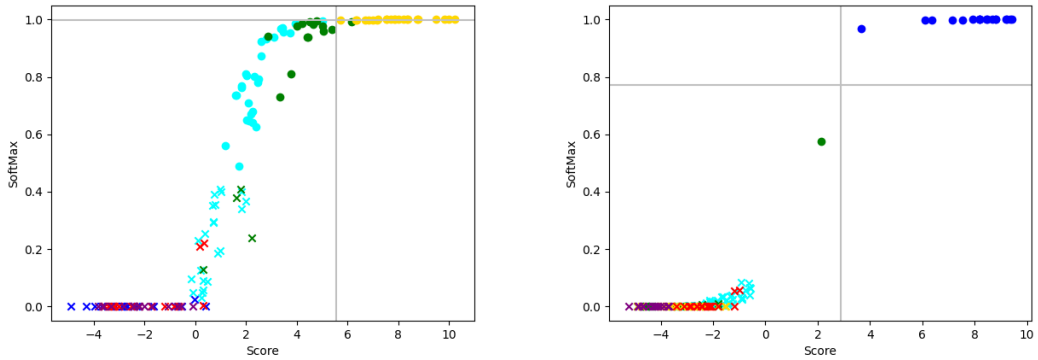
Dazu wurden die Messwerte in den nachfolgenden Abbildungen je Klasse in einem 2D-Raum dargestellt, mit dem Score-Wert auf der x-Achse und dem SoftMax-Wert auf der y-Achse. Die beiden grauen Linien stellen dabei jeweils den Schwellwert dar, der berechnet wurde, um die nicht-klassifizierbaren Karten auszusortieren. Jeder Datenpunkt hat eine Farbe, die einer Klasse zugeordnet wurde. Ein Kreis bedeutet dabei, dass die Karte dieser Klasse zugeordnet wurde, ein Kreuz, dass die Karte in einer anderen Klasse liegt.

Das bedeutet, dass ohne Schwellwerte alle Datenpunkte mit einem Kreis durch den Maximalwert zu dieser Klasse zugeordnet wurden. Durch die Schwellwerte können nun jedoch Datenpunkte, die unterhalb bzw. links des Schwellwerts liegen als „unbekannt" aussortiert werden. Bei einem idealen Bild würden also die Datenpunkte klar getrennt im oberen-rechten bzw. unteren-linken Quadranten liegen. Die Punkte im oberen-rechten Quadranten sollten dann alle eine Farbe (die der korrekten Klasse) haben und alle anderen Punkte unten-links sollten in den anderen Farben sein.

Wie sich auf den Abbildungen 5.1 nun erkennen lässt, hat dies mit dem mobilenet_v2 Modell bereits gut geklappt. Die Datenpunkte sind jedoch oft noch recht nah an den Schwellwerten, sodass diese Trennung mit dem aktuellen Trainingsergebnis noch nicht zuverlässig genug zum Aussortieren der unbekannten Karten ist. Darüber hinaus lässt sich auch erkennen, dass bestimmte Kartenspiele für das Klassifizierungsnetz ähnlicher zueinander aussehen, als andere. Bei den Datenpunkten in Abb. 5.1b ist die Trennung bspw. deutlich eindeutiger als in Abb. 5.1a.

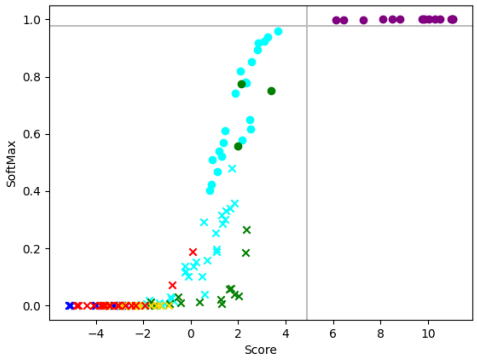
Im Vergleich dazu ist in Abb. 5.2 das Ergebnis für mobilenet_v2 mit Gewichten zu sehen, das etwas schlechtere Ergebnisse produziert. Hier sind einige Punkte um die Schwellwerte herum verteilt. Als Gegensatz dazu ist in Abb. 5.3 jedoch mit mnasnet0_5 ohne Gewichte (einem der letzten Plätze in Score und Softmax) in den Ergebnissen nur eine geringe Separierung der Datenpunkte nach Klassen zu erkennen. Dabei hat das Bestimmen des Maximums außerdem dazu geführt, dass alle Karten als Spielkarten eingestuft wurden, da dort die höchsten Werte (sichtbar an den x-Achsen) erreicht wurden. Hierbei führt der Schwellwert in 5.3d dazu, dass der Großteil der Karten als „unbekannt" eingestuft wird.

Damit lässt sich insgesamt feststellen, dass die Klassifizierung von Kartenspielen möglich ist. Für zuverlässige Ergebnisse sind jedoch sowohl mehr Trainings- als auch Evaluationsdaten notwendig. Insbesondere die Klassifizierung der „unbekannt"-Pseudoklasse ist schwierig, da die Vielzahl unbekannter Karten(-spiele) auch eine unterschiedliche Ähnlichkeit zu den anderen Klassen mit sich bringt.

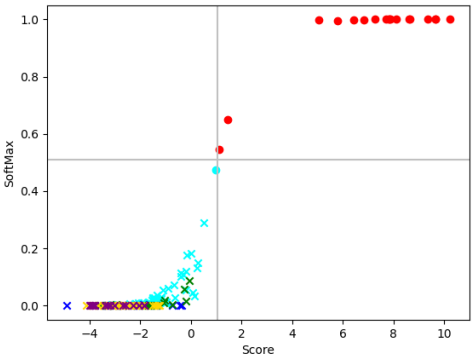


(a) Pokemon

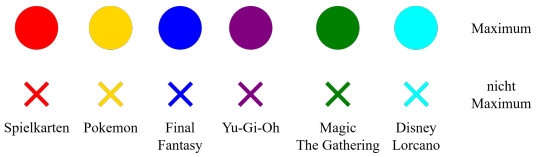
(b) Final Fantasy



(c) Yu-Gi-Oh

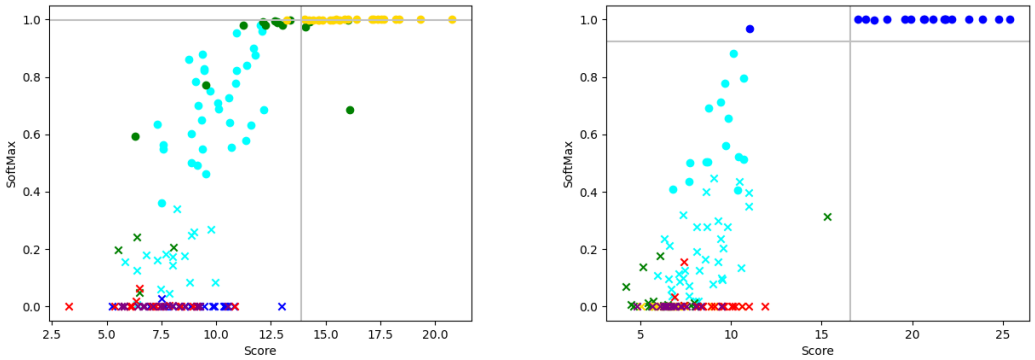


(d) Spielkarten



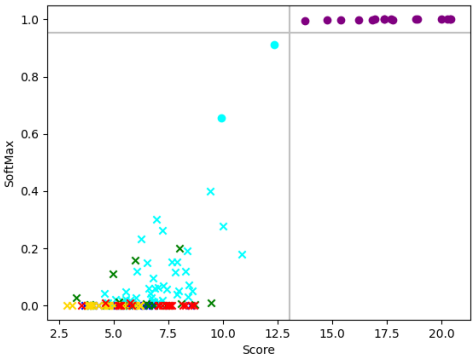
(e) Farblegende

Abbildung 5.1: Messwerte von mobilenet_v2 mit angepassten Gewichten (bestes Modell)

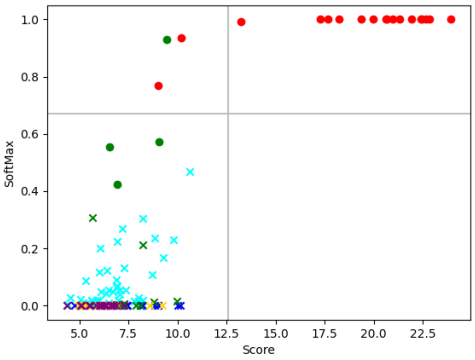


(a) Pokemon

(b) Final Fantasy



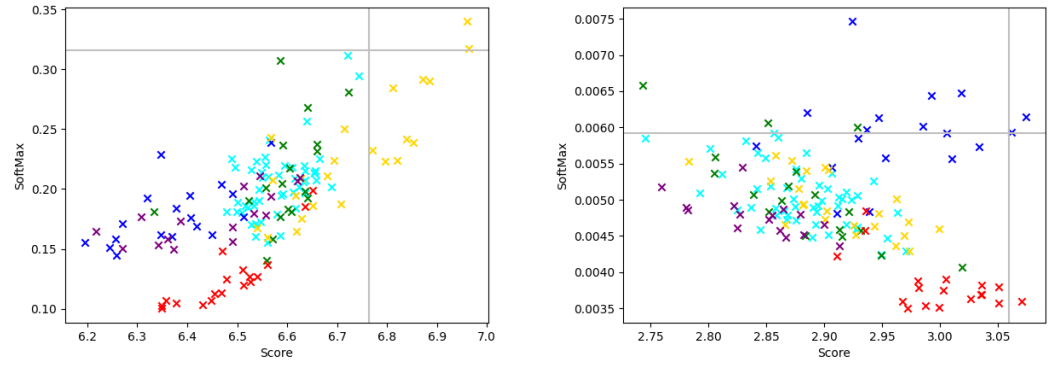
(c) Yu-Gi-Oh



(d) Spielkarten

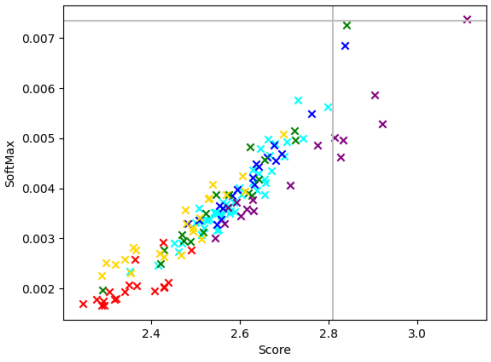
(e) Farblegende

Abbildung 5.2: Messwerte von mobilenet_v2 mit Gewichten (zweitbestes Modell)

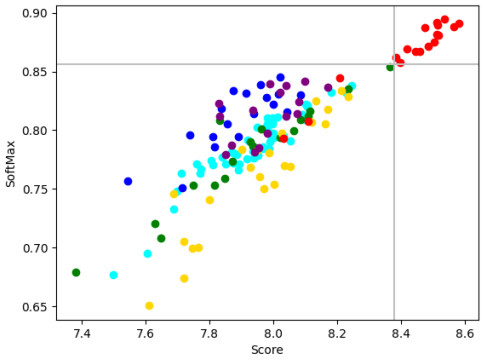


(a) Pokemon

(b) Final Fantasy



(c) Yu-Gi-Oh



(d) Spielkarten

(e) Farblegende

Abbildung 5.3: Messwerte von mnasnet0_5 ohne Gewichte (eines der schlechtesten Modelle)

Kapitel 6

Zusammenfassung und Ausblick

In dieser Arbeit wurden Techniken zur Objekterkennung und Klassifizierung primär mit Convolutional Neural Networks vorgestellt und angewendet. Dabei konnte gezeigt werden, dass durch einen mehrstufigen Prozess die Ergebnisse verbessert werden konnten, indem durch eine zunächst allgemeine und später spezifischeren Klassifizierung mehr Objekte erkannt und als unbekannte Objekte klassifiziert werden konnten.

Im Bezug auf die Forschungsfragen vom Anfang der Arbeit lässt sich damit feststellen:

1. **Sind die Design-Features von Sammelkarten ähnlich genug innerhalb eines Kartenspiels und ausreichend unterschiedlich genug, um eine Klassifizierung zu ermöglichen?**
   Mit den getesteten Kartenspielen konnte eine erfolgreiche Klassifizierung mit einigen der getesteten Klassifizierungsmodelle durchgeführt werden. Bei unbekannten Karten ist das Ergebnis jedoch nicht eindeutig genug, um allgemein zu sagen, dass eine Klassifizierung möglich ist. Diese Ergebnisse können aber mit mehr Trainings- und Evaluationsdaten verbessert werden.

2. **Ist der Trainingsprozess zur Klassifizierung standardisierbar, sodass weitere Karten und Klassen nachträglich hinzufügbar sind?**
   Mit der Implementierung konnte ein Prozess aufgebaut werden, der um zusätzliche Trainingsdaten und Klassen leicht erweiterbar ist. Die Objekterkennung kann fortlaufend mit weiteren Trainingsdaten noch mehr trainiert werden, wobei durch die Erstellung von Trainingsdaten mit dem Modell gezeigt werden konnte, dass dieser Prozess zukünftig leichter ist, da nur noch wenige bis keine Bilder manuell markiert werden müssen.
   Für die Klassifizierung wurde ein standardisierter Prozess implementiert, der aus mehreren Klassifizierungsnetzen das am besten geeignete Netz bestimmen kann.

3. **Kann der Prozess sich selbst trainieren und evaluieren?**
   Für die Objekterkennung konnte eine semiautomatische Lösung umgesetzt werden, bei der die Trainingsdaten einfach erstellt werden können. Der Prozess erfordert aktuell jedoch noch eine manuelle Kontrolle, da das Ergebnis nicht überprüft werden kann, sodass bei schwer erkennbaren Karten die Qualität der Trainingsdaten leiden kann. Hier ist zukünftig also noch eine automatisierte Qualitätskontrolle notwendig.
   Für die Klassifikation ist noch kein automatisches Training unbekannter Karten implementiert. Die generierten Bildausschnitte der Pipeline können jedoch einfach manuell oder durch die Klassifizierungsfunktion sortiert werden. Die daraus entstehenden Ordner in der Testfunktion entsprechen daher der Trainingsdatenstruktur. Dabei fehlt jedoch noch ein zuverlässiger automatischer Evaluationsprozess, zumal die Erkennung unbekannter Kartenspiele noch nicht sicher genug ist und die unbekannten Karten sowieso manuell klassifiziert werden müssten.

4. **Kann der Prozess so designet werden, dass zukünftige Entwicklungen integrierbar sind?**
   Innerhalb des Projektes wurde primär auf das Framework PyTorch gesetzt, um eine einheitliche Programmierung zu ermöglichen. PyTorch hat zum Zeitpunkt der Entwicklung zwar bereits mehrere Klassifizierungsmodelle, von denen eine Auswahl evaluiert wurde. Bei der Objekterkennung ist aktuell jedoch nur Mask R-CNN verfügbar, wodurch andere Modelle mit einem höheren Entwicklungsaufwand integriert werden müssten.
   Insofern ist der Aufwand zur Erweiterung mit Modellen aus dem Framework gering und für andere Modelle bzw. Frameworks individuell zu evaluieren. Unabhängig davon wurde aber ein allgemeiner Prozess zur Klassifizierung von Kartenspielen in dieser Arbeit aufgezeigt, der sich auf andere Frameworks und Entwicklungsumgebungen mit Modellen zur Objekterkennung und Klassifizierung abbilden lässt.

5. **Wie werden unbekannte Karten erkannt bzw. klassifiziert?**
   Unbekannte Karten konnten mit nur geringfügig schlechteren Ergebnissen von der Objekterkennung erkannt werden, die eine Klassifizierung nicht signifikant beeinflussen sollte. Die Klassifizierung stellt sich jedoch als schwieriger heraus, da einige Kartenspiele ähnliche Designs aufweisen. Daher ist insbesondere die Klassifizierung ein Thema, welches zukünftig noch verbessert werden kann.

Literaturverzeichnis

[AHSV19] Audebert, Nicolas, Catherine Herold, Kuider Slimani und Cédric Vidal: *Multimodal deep networks for text and image-based document classification.* In: *Joint European conference on machine learning and knowledge discovery in databases*, Seiten 427–443. Springer, 2019.

[CLB+21] Chen, Leiyu, Shaobo Li, Qiang Bai, Jing Yang, Sanlong Jiang und Yanming Miao: *Review of Image Classification Algorithms Based on Convolutional Neural Networks.* Remote Sensing, 13(22):4712, 2021.

[DP73] Douglas, David H und Thomas K Peucker: *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature.* Cartographica: the international journal for geographic information and geovisualization, 10(2):112–122, 1973.

[Gir15] Girshick, Ross: *Fast R-CNN.* In: *2015 IEEE International Conference on Computer Vision (ICCV)*, Seiten 1440–1448, 2015.

[HGDG17] He, Kaiming, Georgia Gkioxari, Piotr Dollár und Ross Girshick: *Mask R-CNN.* In: *Proceedings of the IEEE international conference on computer vision*, Seiten 2961–2969, 2017.

[Ili02] Ilie, Kok-Lim Low Adrian: *View Frustum Optimization To Maximize Object's Image Area.* Citeseer, 2002.

[KBLK10] Khan, Aurangzeb, Baharum Baharudin, Lam Hong Lee und Khairullah Khan: *A Review of Machine Learning Algorithms for Text-Documents Classification.* Journal of advances in information technology, 1(1):4–20, 2010.

[KHY+22] Kim, Geewook, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han und Seunghyun Park: *OCR-free Document Understanding Transformer.* In: *European Conference on Computer Vision*, Seiten 498–517. Springer, 2022.

[KKY+14] Kang, Le, Jayant Kumar, Peng Ye, Yi Li und David Doermann: *Convolutional Neural Networks for Document Image Classification.* In: *2014 22nd international conference on pattern recognition*, Seiten 3168–3172. IEEE, 2014.

[LM23] Lenhart, Stephan und Tillmann Milinski: *Eckenerkennung von Dokumenten.* NEidI, Universität Rostock, Institut für Informatik, 2023.

[Low04] Lowe, David G: *Distinctive Image Features from Scale-Invariant Keypoints.* International journal of computer vision, 60:91–110, 2004.

[MGC+24] Mittal, Khushi, Kanwarpartap Singh Gill, Rahul Chauhan, Manish Sharma und G Sunil: *Playing Cards Classification and Detection Using Sequential CNN Model Through Machine Learning Techniques Using Artificial Intelligence.* In: *2024 International Conference on E-mobility, Power Control and Smart Systems (ICEMPS)*, Seiten 1–4, 2024.

[NIAV24] Nahar, Lutfun, Md Saiful Islam, Mohammad Awrangjeb und Rob Verhoeve: *Edge Grading in Trading Cards Using Transfer Learning: Methods, Experiments, and Evaluation.* In: *2024 IEEE International Conference on Big Data (BigData)*, Seiten 2005–2012. IEEE, 2024.

[ON15] O'shea, Keiron und Ryan Nash: *An Introduction to Convolutional Neural Networks.* arXiv preprint arXiv:1511.08458, 2015.

[PyT24a] PyTorch: *TorchVision Object Detection Finetuning Tutorial.* `https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html`, 2024. Abgerufen am 20. Janaur 2025.

[PyT24b] PyTorch: *Training a Classifier.* `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`, 2024. Abgerufen am 04. April 2025.

[RDGF16] Redmon, Joseph, Santosh Divvala, Ross Girshick und Ali Farhadi: *You Only Look Once: Unified, Real-Time Object Detection.* In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, Seiten 779–788, 2016.

[RHGS16] Ren, Shaoqing, Kaiming He, Ross Girshick und Jian Sun: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* IEEE transactions on pattern analysis and machine intelligence, 39(6):1137–1149, 2016.

[Sb85] Suzuki, Satoshi und KeiichiA be: *Topological Structural Analysis of Digitized Binary Images by Border Following.* Computer Vision, Graphics, and Image Processing, 30(1):32–46, 1985.

[VEH20] Van Engelen, Jesper E und Holger H Hoos: *A survey on semi-supervised learning.* Machine learning, 109(2):373–440, 2020.

[ZPL18] Zilio, Felipe, Marcelo Prates und Luis Lamb: *Neural Networks Models for Analyzing Magic: the Gathering Cards.* In: *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13–16, 2018, Proceedings, Part II 25*, Seiten 227–239. Springer, 2018.

Anhang A

Installations- und Ausführungshinweise

In diesem Kapitel gibt es eine kurze Anleitung, wie das Programm verwendet werden kann.
Alle Links auf dieser Seite wurden zuletzt am 18.04.2025 aufgerufen.

A.1 Einrichtung

Um das Programm verwenden zu können, müssen vor der ersten Nutzung folgende Programme installiert werden:

- **Python 3.12**
  `https://www.python.org/downloads/release/python-3121/`
- **PyTorch 2.5.1**
  Bitte den Installationsanweisungen unter `https://pytorch.org/get-started/locally/` folgen und die auf dem Gerät vorhandene Plattform auswählen, um den optimalen Installationsbefehl zu erhalten.
- **CUDA Toolkit** *optional, aber empfohlen*
  Sofern das Gerät eine CUDA-fähige Grafikkarte besitzt, sollten die CUDA-Toolkits von `https://developer.nvidia.com/cuda-downloads` installiert werden. Andernfalls laufen die KI-Netze auf der CPU, wodurch die Programme langsamer laufen. Weitere Informationen dazu finden sich auch in der PyTorch-Anleitung.

Abschließend sollten in dem Python-Projekt bzw. der venv-Umgebung mit `pip install -r requirements.txt` die verbliebenen Abhängigkeiten installiert werden.

Bitte beachte außerdem, dass die Ausführung beim ersten Durchgang etwas länger dauern wird, da noch Modelle / Gewichte durch PyTorch nachgeladen werden. Diese sind notwendig, um die Modelle neu trainieren zu können.

A.2 Übersicht über die Programme

An dieser Stelle gibt es eine kurze Übersicht über Programme, die direkt ausgeführt werden können und eine kurze Beschreibung, was in dem Programm passiert.

- **`pipeline.py`**
  Dieses Programm enthält die Pipeline (Abschnitt 4.2) und ermöglicht damit die Objekterkennung und Klassifizierung von Bilddateien bzw. Ordner mit Bildern. Das Programm gibt bei direkter Ausführung über die Kommandozeile die Koordinaten der Eckpunkte und das bestimmte Kartenspiel für jede erkannte Karte aus.

- **`obj_detector.py`**
  Dieses Programm trainiert die Objekterkennung bei direkter Ausführung (Ausschnitt C.1). Wenn kein Netz unter `object_detector.pth` gefunden wurde, wird ein neues Netz initialisiert. Andernfalls wird das bisherige Netz mit Zeitstempel gesichert und weiter trainiert.

- **`self_training.py`**
  Dieses Programm erstellt mit Hilfe der Objekterkennung weitere Trainingsdaten aus den Bildern, die unter `data/cardsSelfTraining/input` eingegeben werden (Ausschnitt C.3). Die Trainingsdaten können dann mit den generierten Bildern aus `data/cardsSelfTraining/images` und `data/cardsSelfTraining/masks` ergänzt werden.

- **`evaluate_objdetector.py`**
  Dieses Programm evaluiert die Objekterkennung mit Hilfe des Evaluationsdatensatzes unter `data/eval_classes` (Ausschnitt C.2).

- **`classifier.py`**
  Dieses Programm trainiert die Klassifizierung bei direkter Ausführung (Ausschnitt C.4). Der Klassifizierungstrainingsdatensatz in `data/classesExpanded` muss vorhanden sein.

- **`classifier_threshold.py`**
  Dieses Programm evaluiert mit Hilfe der Evaluationsdaten aus `data/eval_classes` die Klassifizierungsmodelle (Abschnitt C.5). Die Ergebnisse werden in `classifier_evaluation/` ausgegeben.

Das Programm **`pipeline.py`** besitzt einige Kommandozeilenargumente, mit denen die Klassifizierung genauer gesteuert werden kann. Die Argumente sind jeweils optional, da die Statistiken, die durch **`classifier_threshold.py`** erstellt wurden, verwendet werden, um das jeweils beste Modell nach den übergebenen Parametern auszuwählen. Als `source` kann eine Bilddatei oder ein Ordner mit Bilddateien übergeben werden. Die möglichen Parameter für das Programm können über das Kommando `python pipeline.py -help` abgefragt werden.

Anhang B

Evaluationsergebnisse der Klassifizierung

In diesem Kapitel finden sich in Tabelle B.1 die vollständigen Ergebnisse der Klassifizierung, sowie in Tabelle B.2 die dabei berechneten Schwellwerte. In beiden Tabellen ist jeweils in der ersten Spalte ein Identifizierungscode zu finden, durch den sich die Werte aus den beiden Tabellen zueinander zuordnen lassen.

B.1 Ergebnisse

Tabelle B.1: Evaluationsergebnisse der Klassifizierung

ID	Fehler	Model	Trainingsart	Parameter	durch-schnitt. Dauer
01AM	9	regnet_y_16gf	mit angepassten Gewichten	SoftMax	0.021s
01AS	8	regnet_y_16gf	mit angepassten Gewichten	Score	0.021s
01BM	12	regnet_y_16gf	mit Gewichten	SoftMax	0.023s
01BS	14	regnet_y_16gf	mit Gewichten	Score	0.023s
01CM	32	regnet_y_16gf	ohne Gewichte	SoftMax	0.021s
01CS	43	regnet_y_16gf	ohne Gewichte	Score	0.021s
02AM	8	vit_b_16	mit angepassten Gewichten	SoftMax	0.007s
02AS	8	vit_b_16	mit angepassten Gewichten	Score	0.007s
02BM	22	vit_b_16	mit Gewichten	SoftMax	0.007s
02BS	14	vit_b_16	mit Gewichten	Score	0.007s
02CM	42	vit_b_16	ohne Gewichte	SoftMax	0.007s
02CS	44	vit_b_16	ohne Gewichte	Score	0.007s
03AM	14	efficientnet_v2_m	mit angepassten Gewichten	SoftMax	0.037s
03AS	15	efficientnet_v2_m	mit angepassten Gewichten	Score	0.037s
03BM	15	efficientnet_v2_m	mit Gewichten	SoftMax	0.036s
03BS	21	efficientnet_v2_m	mit Gewichten	Score	0.036s
03CM	35	efficientnet_v2_m	ohne Gewichte	SoftMax	0.036s
03CS	39	efficientnet_v2_m	ohne Gewichte	Score	0.036s
04AM	5	efficientnet_v2_s	mit angepassten Gewichten	SoftMax	0.026s

04AS	3	efficientnet_v2_s	mit angepassten Gewichten	Score	0.026s
04BM	7	efficientnet_v2_s	mit Gewichten	SoftMax	0.025s
04BS	25	efficientnet_v2_s	mit Gewichten	Score	0.025s
04CM	18	efficientnet_v2_s	ohne Gewichte	SoftMax	0.026s
04CS	30	efficientnet_v2_s	ohne Gewichte	Score	0.026s
05AM	10	efficientnet_b4	mit angepassten Gewichten	SoftMax	0.024s
05AS	13	efficientnet_b4	mit angepassten Gewichten	Score	0.024s
05BM	7	efficientnet_b4	mit Gewichten	SoftMax	0.023s
05BS	18	efficientnet_b4	mit Gewichten	Score	0.023s
05CM	38	efficientnet_b4	ohne Gewichte	SoftMax	0.024s
05CS	44	efficientnet_b4	ohne Gewichte	Score	0.024s
06AM	71	efficientnet_b3	mit angepassten Gewichten	SoftMax	0.019s
06AS	34	efficientnet_b3	mit angepassten Gewichten	Score	0.019s
06BM	7	efficientnet_b3	mit Gewichten	SoftMax	0.018s
06BS	12	efficientnet_b3	mit Gewichten	Score	0.018s
06CM	27	efficientnet_b3	ohne Gewichte	SoftMax	0.02s
06CS	31	efficientnet_b3	ohne Gewichte	Score	0.02s
07AM	6	regnet_y_1_6gf	mit angepassten Gewichten	SoftMax	0.028s
07AS	7	regnet_y_1_6gf	mit angepassten Gewichten	Score	0.028s
07BM	5	regnet_y_1_6gf	mit Gewichten	SoftMax	0.027s
07BS	6	regnet_y_1_6gf	mit Gewichten	Score	0.027s
07CM	34	regnet_y_1_6gf	ohne Gewichte	SoftMax	0.028s
07CS	46	regnet_y_1_6gf	ohne Gewichte	Score	0.028s
08AM	6	efficientnet_b2	mit angepassten Gewichten	SoftMax	0.017s
08AS	7	efficientnet_b2	mit angepassten Gewichten	Score	0.017s
08BM	5	efficientnet_b2	mit Gewichten	SoftMax	0.016s
08BS	14	efficientnet_b2	mit Gewichten	Score	0.016s
08CM	34	efficientnet_b2	ohne Gewichte	SoftMax	0.018s
08CS	63	efficientnet_b2	ohne Gewichte	Score	0.018s
09AM	11	efficientnet_b1	mit angepassten Gewichten	SoftMax	0.016s
09AS	10	efficientnet_b1	mit angepassten Gewichten	Score	0.016s
09BM	7	efficientnet_b1	mit Gewichten	SoftMax	0.016s
09BS	13	efficientnet_b1	mit Gewichten	Score	0.016s
09CM	24	efficientnet_b1	ohne Gewichte	SoftMax	0.018s
09CS	39	efficientnet_b1	ohne Gewichte	Score	0.018s
10AM	5	regnet_y_800mf	mit angepassten Gewichten	SoftMax	0.012s
10AS	5	regnet_y_800mf	mit angepassten Gewichten	Score	0.012s
10BM	6	regnet_y_800mf	mit Gewichten	SoftMax	0.012s
10BS	21	regnet_y_800mf	mit Gewichten	Score	0.012s
10CM	32	regnet_y_800mf	ohne Gewichte	SoftMax	0.013s
10CS	42	regnet_y_800mf	ohne Gewichte	Score	0.013s
11AM	10	efficientnet_b0	mit angepassten Gewichten	SoftMax	0.011s
11AS	10	efficientnet_b0	mit angepassten Gewichten	Score	0.011s
11BM	4	efficientnet_b0	mit Gewichten	SoftMax	0.011s
11BS	8	efficientnet_b0	mit Gewichten	Score	0.011s
11CM	29	efficientnet_b0	ohne Gewichte	SoftMax	0.012s
11CS	33	efficientnet_b0	ohne Gewichte	Score	0.012s
12AM	6	regnet_y_400mf	mit angepassten Gewichten	SoftMax	0.014s

12AS	8	regnet_y_400mf	mit angepassten Gewichten	Score	0.014s
12BM	6	regnet_y_400mf	mit Gewichten	SoftMax	0.013s
12BS	14	regnet_y_400mf	mit Gewichten	Score	0.013s
12CM	48	regnet_y_400mf	ohne Gewichte	SoftMax	0.014s
12CS	59	regnet_y_400mf	ohne Gewichte	Score	0.014s
13AM	41	shufflenet_v2_x1_5	mit angepassten Gewichten	SoftMax	0.009s
13AS	26	shufflenet_v2_x1_5	mit angepassten Gewichten	Score	0.009s
13BM	7	shufflenet_v2_x1_5	mit Gewichten	SoftMax	0.009s
13BS	15	shufflenet_v2_x1_5	mit Gewichten	Score	0.009s
13CM	20	shufflenet_v2_x1_5	ohne Gewichte	SoftMax	0.009s
13CS	29	shufflenet_v2_x1_5	ohne Gewichte	Score	0.009s
14AM	**0**	**mobilenet_v2**	**mit angepassten Gewichten**	**SoftMax**	**0.007s**
14AS	1	mobilenet_v2	mit angepassten Gewichten	Score	0.007s
14BM	2	mobilenet_v2	mit Gewichten	SoftMax	0.008s
14BS	8	mobilenet_v2	mit Gewichten	Score	0.008s
14CM	16	mobilenet_v2	ohne Gewichte	SoftMax	0.008s
14CS	28	mobilenet_v2	ohne Gewichte	Score	0.008s
15AM	6	shufflenet_v2_x1_0	mit angepassten Gewichten	SoftMax	0.009s
15AS	9	shufflenet_v2_x1_0	mit angepassten Gewichten	Score	0.009s
15BM	9	shufflenet_v2_x1_0	mit Gewichten	SoftMax	0.01s
15BS	24	shufflenet_v2_x1_0	mit Gewichten	Score	0.01s
15CM	21	shufflenet_v2_x1_0	ohne Gewichte	SoftMax	0.009s
15CS	41	shufflenet_v2_x1_0	ohne Gewichte	Score	0.009s
16AM	16	mnasnet0_5	mit angepassten Gewichten	SoftMax	0.007s
16AS	19	mnasnet0_5	mit angepassten Gewichten	Score	0.007s
16BM	16	mnasnet0_5	mit Gewichten	SoftMax	0.007s
16BS	33	mnasnet0_5	mit Gewichten	Score	0.007s
16CM	60	mnasnet0_5	ohne Gewichte	SoftMax	0.008s
16CS	60	mnasnet0_5	ohne Gewichte	Score	0.008s
17AM	5	shufflenet_v2_x0_5	mit angepassten Gewichten	SoftMax	0.009s
17AS	6	shufflenet_v2_x0_5	mit angepassten Gewichten	Score	0.009s
17BM	19	shufflenet_v2_x0_5	mit Gewichten	SoftMax	0.009s
17BS	32	shufflenet_v2_x0_5	mit Gewichten	Score	0.009s
17CM	18	shufflenet_v2_x0_5	ohne Gewichte	SoftMax	0.009s
17CS	40	shufflenet_v2_x0_5	ohne Gewichte	Score	0.009s
18AM	16	squeezenet1_1	mit angepassten Gewichten	SoftMax	0.003s
18AS	30	squeezenet1_1	mit angepassten Gewichten	Score	0.003s
18BM	16	squeezenet1_1	mit Gewichten	SoftMax	0.003s
18BS	34	squeezenet1_1	mit Gewichten	Score	0.003s
18CM	36	squeezenet1_1	ohne Gewichte	SoftMax	0.003s
18CS	44	squeezenet1_1	ohne Gewichte	Score	0.003s
19CM	19	OwnCNN	ohne Gewichte	SoftMax	0.001s
19CS	21	OwnCNN	ohne Gewichte	Score	0.001s

B.2 Schwellwerte

Tabelle B.2: Schwellwerte für die Klassifizierung

	Schwellwerte			
ID	Pokemon	Final Fantasy	Yu-Gi-Oh	Spielkarten
01AM	0.9989386349916458	0.9424833059310913	0.8649406433105469	0.889292374253273
01AS	7.100507974624634	2.8948771953582764	1.5446373224258423	5.856348276138306
01BM	0.9976820945739746	0.9504522085189819	0.9074264168739319	0.10157676786184311
01BS	12.009497165679932	14.049155950546265	12.830270290374756	9.134831190109253
01CM	0.9995683431625366	0.961906909942627	0.8496924042701721	0.1971425311639905
01CS	50.61533784866333	19.566923141479492	22.090784549713135	17.253950119018555
02AM	0.9994583576917648	0.9999288618564606	0.996718555688858	0.6366128344088793
02AS	7.040209650993347	8.428579807281494	6.195665121078491	2.916752703487873
02BM	0.9999733865261078	0.9882000088691711	0.9956058412790298	0.030598179437220097
02BS	18.619333267211914	17.01204013824463	16.248483657836914	11.590078830718994
02CM	0.9999958574771881	0.9999781847000122	0.9206459373235703	0.6533658802509308
02CS	17.638218879699707	16.646467685699463	15.216109991073608	13.306296348571777
03AM	0.9998154044151306	0.9950726926326752	0.6994303464889526	0.7698690257966518
03AS	7.3271530866622925	6.661548614501953	2.773954153060913	5.024505943059921
03BM	0.9993148148059845	0.9978995025157928	0.9528068602085114	0.7392018586397171
03BS	14.858628034591675	17.729186534881592	15.616386890411377	15.010821342468262
03CM	0.989920124411583	0.9789745211601257	0.9765691459178925	0.35358189791440964
03CS	18.73610258102417	21.988179206848145	21.85477924346924	13.008601427078247
04AM	0.9993527680635452	0.9867774546146393	0.9444147348403931	0.8905657976865768
04AS	7.25513768196106	4.185597896575928	5.591425895690918	4.27686333656311
04BM	0.9999717772006989	0.9476840496063232	0.5372827760875225	0.996047317981 72
04BS	18.255249500274658	15.816365957260132	13.637338638305664	17.076040029525757
04CM	0.9971962571144104	0.9894243478775024	0.9718454033136368	0.8208905905485153
04CS	18.57530117034912	17.337364196777344	15.384127140045166	12.299389362335205
05AM	0.9511955678462982	0.9933392256498337	0.9143721759319305	0.7941862940788269
05AS	2.4737282395362854	4.975522041320801	3.778579831123352	2.633926570415497
05BM	0.9272169321775436	0.9958361387252808	0.5841504037380219	0.9154578596353531
05BS	16.602067947387695	17.044976711273193	13.337191104888916	12.655282258987427
05CM	0.9997649788856506	0.9537189900875092	0.9100875556468964	0.0146501634735614 06
05CS	86.81975078582764	20.888477325439453	13.645681858062744	12.406671285629272
06AM	0.9989658892154694	0.964605987071991	0.9658345580101013	0.8980412036180496
06AS	6.326011776924133	6857.8514404296875	4.517811298370361	1.9093201160430908
06BM	0.9986028969287872	0.9977815449237823	0.9040522277355194	0.7717311382293701
06BS	13.961510181427002	13.23567008972168	9.390264511108398	9.755962610244751
06CM	0.9837480038404465	0.9976103901863098	0.31811708211898804	0.7655830979347229
06CS	16.11234736442566	23.30740737915039	13.346506118774414	19.094937324523926
07AM	0.9946698248386383	0.9789060205221176	0.9875660240650177	0.24549663439393044
07AS	5.43265426158905	5.320613861083984	5.069467306137085	0.32786399126052856
07BM	0.9935116171836853	0.9990759044885635	0.7869478762149811	0.6399846822023392
07BS	13.794759511947632	17.918716430664062	13.46117877960205	10.554019451141357
07CM	0.9822266399860382	0.670409768819809	0.9992582350969315	0.33754612132906914
07CS	1302.4300537109375	20.221745491027832	1088.2359313964844	16.048624992370605

08AM	0.9962406605482101	0.8467212617397308	0.863727867603302	0.8670150339603424
08AS	6.283023476600647	4.126323223114014	3.158142924308777	4.186129212379456
08BM	0.9950493574142456	0.9260243475437164	0.5606604516506195	0.9712251424789429
08BS	13.443284749984741	12.376537322998047	10.6331701278686 52	12.475377082824707
08CM	0.9835682064294815	0.9976425170898438	0.25078166276216507	0.8941333740949631
08CS	20.439262866973877	18.946543216705322	57.1388521194458	43.770259857177734
09AM	0.997705489397049	0.8102246075868607	0.9844028949737549	0.9466474056243896
09AS	6.235562324523926	3.4596455097198486	4.771796226501465	5.056131720542908
09BM	0.9878658503293991	0.998598262667656	0.9873551428318024	0.9323467314243317
09BS	13.06957721710205	15.993719339370728	13.389980792999268	13.06696605682373
09CM	0.9649250209331512	0.9566545784473419	0.9113965481519699	0.5748145580291748
09CS	23.685548782348633	17.252294063568115	18.41145086288452	12.06111741065979
10AM	0.9985409379005432	0.9155672639608383	0.9433572888374329	0.2739705219864845
10AS	5.621941804885864	3.771154761314392	4.524105548858643	2.5795645117759705
10BM	0.9848922491073608	0.9986533373594284	0.9583773910999298	0.12276023998856544
10BS	14.369809865951538	18.60725212097168	14.481934070587158	13.245416641235352
10CM	0.9962211698293686	0.9251366853713989	0.9842527210712433	0.0734611724037677
10CS	18.488571166992188	17.625274658203125	37.757094860076904	14.867987394332886
11AM	0.9995252788066864	0.9678388088941574	0.9618628025054932	0.8153506442904472
11AS	6.5349873304367065	6.317530035972595	4.977881669998169	4.479865461587906
11BM	0.9984471350908279	0.9607379734516144	0.9742526710033417	0.8365371227264404
11BS	14.56239366531372	14.174836158752441	15.301518440246582	16.143070936203003
11CM	0.9851557612419128	0.9832884669303894	0.974155992269516	0.745589985512197
11CS	20.138415336608887	17.575451850891113	15.364356994628906	13.456592321395874
12AM	0.9981570392847061	0.6194669604301453	0.9915194511413574	0.016286600148305297
12AS	6.491758346557617	6.37028956413269	5.610537886619568	-0.9889419674873352
12BM	0.9922939389944077	0.7710413932800293	0.9958204925060272	0.1483667530119419
12BS	13.959071636199951	10.390103816986084	14.428779602050781	12.119608640670776
12CM	0.9999657869338989	0.9985433369874954	0.7842129915952682	0.0857263458892703
12CS	147.14674377441406	26.393288612365723	227.62348556518555	14.263521432876587
13AM	0.9960931539535522	0.9999926090240479	0.898729994893074	0.815892182290554
13AS	4.865909814834595	728.7379150390625	2.4538318514823914	3.6663938760757446
13BM	0.9976014345884323	0.8064341247081757	0.8991116881370544	0.15590491518378258
13BS	13.9832022190094	10.953817367553711	10.511841297149658	11.571646690368652
13CM	0.9712688028812408	0.8218776881694794	0.9946702271699905	0.14467357844114304
13CS	16.383405208587646	17.642577648162842	16.90117073059082	14.01008939743042
14AM	0.9974055588245392	0.7726157009601593	0.9782227575778961	0.5107774883508682
14AS	5.569255828857422	2.8865489959716797	4.8980947732925415	1.0619264543056488
14BM	0.9984252750873566	0.925284206867218	0.9544469118118286	0.6709683537483215
14BS	13.884147644042969	16.585097551345825	13.050317287445068	12.572020292282104
14CM	0.9460242092609406	0.9690606594085693	0.9947630017995834	0.9471950531005859
14CS	17.73927593231201	23.430306434631348	17.691388607025146	16.368196964263916
15AM	0.9350037723779678	0.9942279160022736	0.9639542102813721	0.15808145701885223
15AS	4.205085039138794	5.6400134563446045	3.7333954572677612	-0.061444565653800964
15BM	0.9993962347507477	0.9952504336833954	0.9216533303260803	0.7881437391042709
15BS	19.187979221343994	20.77259111404419	23.269397735595703	13.386234283447266
15CM	0.9672467559576035	0.9666747897863388	0.9937185347080231	0.501469112932682
15CS	25.03475570678711	20.021219730377197	20.24223232269287	15.3250412940979

16AM	0.710451066493988	0.9560292214155197	0.884491428732872	0.6381560526788235
16AS	2.114127457141876	2.9874053597450256	2.766380786895752	1.8097846806049347
16BM	0.5846158564090729	0.964978888630867	0.8618860840797424	0.45843181759119034
16BS	14.365485191345215	17.45291566848755	16.94884490966797	15.396888971328735
16CM	0.31565170735120773	0.005918790935538709	0.007344080833718181	0.8564821183681488
16CS	6.764025807380676	3.059348464012146	2.8100398182868958	8.380005836486816
17AM	0.9625945687294006	0.8560273945331573	0.9624761939048767	0.5228070318698883
17AS	3.3526123762130737	2.3493728637695312	4.248353004455566	0.6432503759860992
17BM	0.9997575432062149	0.9839821606874466	0.2662598788738251	0.8495966345071793
17BS	24.321136474609375	21.425963401794434	16.003632068634033	17.7727632522583
17CM	0.9970768392086029	0.9827109575271606	0.9641820788383484	0.27478655613958836
17CS	24.80864191055298	20.80399513244629	21.71045684814453	14.714615106582642
18AM	0.9999832212924957	0.9998945593833923	0.807391494512558	0.7880479916930199
18AS	29.089799404144287	20.398065090179443	19.670652389526367	14.659780502319336
18BM	0.9999993145465851	0.9999952614307404	0.8115347474813461	0.48546575504587963
18BS	39.53803253173828	40.92510986328125	42.415894508361816	21.62769603729248
18CM	0.9800111055374146	0.7637479826807976	0.9995829910039902	0.8664387464523315
18CS	96.72990989685059	30.49292516708374	57.97839641571045	15.293956518173218
19CM	0.9999966323375702	0.9999794960021973	0.9998516291379929	0.01885204203426838
19CS	16.456707000732422	15.76789927482605	10.393659830093384	0.714169055223465

Anhang C

Programmcode

C.1 Training der Objekterkennung

Quellcodeausschnitt C.1: Training der Objekterkennung

```
1  import datetime
2  import math
3  import shutil
4
5  import torch
6  import torchvision
7  from torchvision.transforms import v2 as T
8  from torchvision.transforms.v2 import functional as F
9  from torchvision import tv_tensors
10 from torchvision.io import read_image
11 from torchvision.ops.boxes import masks_to_boxes
12 from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
13 from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
14
15 import os
16
17 import create_poly
18 import helpers.utils as utils
19 from helpers.engine import train_one_epoch, evaluate
20
21 import cv2
22 import numpy as np
23 from pathlib import Path
24 import json
25
26 class cardsDataset(torch.utils.data.Dataset):
27     # based on class PennFudanDataset from
      ↪ https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html
28     def __init__(self, root, transforms):
29         self.root = root
```

```
30          self.transforms = transforms
31          # load all image files, sorting them to
32          # ensure that they are aligned
33          self.imgs = list(sorted(os.listdir(os.path.join(root, "images"))))
34          self.masks = list(sorted(os.listdir(os.path.join(root, "masks"))))
35
36      def __getitem__(self, idx):
37          # load images and masks
38          img_path = os.path.join(self.root, "images", self.imgs[idx])
39          mask_path = os.path.join(self.root, "masks", self.masks[idx])
40          img = read_image(img_path)
41          mask = read_image(mask_path)
42          # instances are encoded as different colors
43          obj_ids = torch.unique(mask)
44          # first id is the background, so remove it
45          obj_ids = obj_ids[1:]
46          num_objs = len(obj_ids)
47
48          # split the color-encoded mask into a set
49          # of binary masks
50          masks = (mask == obj_ids[:, None, None]).to(dtype=torch.uint8)
51
52          # get bounding box coordinates for each mask
53          boxes = masks_to_boxes(masks)
54
55          # there is only one class
56          labels = torch.ones((num_objs,), dtype=torch.int64)
57
58          image_id = idx
59          area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
60          # suppose all instances are not crowd
61          iscrowd = torch.zeros((num_objs,), dtype=torch.int64)
62
63          # Wrap sample and targets into torchvision tv_tensors:
64          img = tv_tensors.Image(img)
65
66          target = {}
67          target["boxes"] = tv_tensors.BoundingBoxes(boxes, format="XYXY",
            ↪ canvas_size=F.get_size(img))
68          target["masks"] = tv_tensors.Mask(masks)
69          target["labels"] = labels
70          target["image_id"] = image_id
71          target["area"] = area
72          target["iscrowd"] = iscrowd
73
74          if self.transforms is not None:
75              img, target = self.transforms(img, target)
76
77          return img, target
78
79      def __len__(self):
```

```
80          return len(self.imgs)
81
82
83  def get_model_instance_segmentation(num_classes):
84      # load an instance segmentation model pre-trained on COCO
85      model =
        ↪ torchvision.models.detection.maskrcnn_resnet50_fpn(weights="DEFAULT")
86
87      # get number of input features for the classifier
88      in_features = model.roi_heads.box_predictor.cls_score.in_features
89      # replace the pre-trained head with a new one
90      model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
        ↪ num_classes)
91
92      # now get the number of input features for the mask classifier
93      in_features_mask =
        ↪ model.roi_heads.mask_predictor.conv5_mask.in_channels
94      hidden_layer = 256
95      # and replace the mask predictor with a new one
96      model.roi_heads.mask_predictor = MaskRCNNPredictor(
97          in_features_mask,
98          hidden_layer,
99          num_classes
100     )
101
102     return model
103
104 def createImageMasksForPreTaggedData():
105     folder = Path("data\\cards\\")
106
107     with open(folder / 'result.json', 'r') as file:
108         resultJson = json.load(file)
109
110     images = {}
111     for i in resultJson["images"]:
112         images[i["id"]] = {
113             "width": i["width"],
114             "height": i["height"],
115             "file_name": i["file_name"],
116             "annotations": []
117         }
118
119     categories = {}
120     for i in resultJson["categories"]:
121         categories[i["id"]] = {
122             "name": i["name"]
123         }
124
125     annotations = {}
126     for i in resultJson["annotations"]:
127         images[i["image_id"]]["annotations"].append({
```

```
128             "category_id": i["category_id"],
129             "category_name": categories[i["category_id"]],
130             "segmentation": i["segmentation"],
131             "bbox": i["bbox"],
132         })
133
134     for i in images:
135         img = images[i]
136         mask = np.zeros((img["height"], img["width"], 1), np.uint8)
137         counter = 0
138         for annotation in img['annotations']:
139             poly = []
140             for k in range(0, len(annotation["segmentation"][0]), 2):
141                 poly.append([int(annotation["segmentation"][0][k]),
                    ↪ int(annotation["segmentation"][0][k + 1])])
142                 cv2.fillPoly(mask, pts=[np.array(poly)], color=[counter])
143                 counter += 1
144
145         (folder / "masks").mkdir(parents=True, exist_ok=True)
146         filename = (folder / img["file_name"]).stem + ".png"
147         cv2.imwrite(str(folder / "masks" / filename), mask)
148
149 def get_transform(train):
150     transforms = []
151     if train:
152         transforms.append(T.RandomHorizontalFlip(0.5))
153     transforms.append(T.ToDtype(torch.float, scale=True))
154     transforms.append(T.ToPureTensor())
155     return T.Compose(transforms)
156
157
158 def trainObjDetector(num_epochs=10, trainingFactor=0.8,
159                      lr=0.005, momentum=0.9, weight_decay=0.0005,
160                      step_size=3, gamma=0.1):
161     # train on the GPU or on the CPU, if a GPU is not available
162     device = torch.device('cuda') if torch.cuda.is_available() else
        ↪ torch.device('cpu')
163
164     # our dataset has two classes only - background and cards
165     num_classes = 2
166     # use our dataset and defined transformations
167     dataset = cardsDataset('data/cards', get_transform(train=True))
168     dataset_test = cardsDataset('data/cards', get_transform(train=False))
169
170     setLength = len(dataset)
171     trainSetLength = math.ceil(setLength * trainingFactor)
172     trainingIndex = trainSetLength * (-1)
173
174     createImageMasksForPreTaggedData()
175
176     # split the dataset in train and test set
```

```
177     indices = torch.randperm(len(dataset)).tolist()
178     dataset = torch.utils.data.Subset(dataset, indices[trainingIndex:])
179     dataset_test = torch.utils.data.Subset(dataset_test,
        ↪ indices[:trainingIndex])
180
181     # define training and validation data loaders
182     data_loader = torch.utils.data.DataLoader(
183         dataset,
184         batch_size=1,
185         shuffle=False,
186         collate_fn=utils.collate_fn
187     )
188
189     data_loader_test = torch.utils.data.DataLoader(
190         dataset_test,
191         batch_size=1,
192         shuffle=False,
193         collate_fn=utils.collate_fn
194     )
195
196     # get the model using our helper function
197     #model = get_model_instance_segmentation(num_classes)
198     datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
199     shutil.copyfile('object_detector.pth', 'object_detector_before' +
        ↪ str(datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')) + '.pth')
200     model = torch.load('object_detector.pth', weights_only=False)
201
202     # move model to the right device
203     model.to(device)
204
205     # construct an optimizer
206     params = [p for p in model.parameters() if p.requires_grad]
207     optimizer = torch.optim.SGD(
208         params,
209         lr=lr,
210         momentum=momentum,
211         weight_decay=weight_decay
212     )
213
214     # and a learning rate scheduler
215     lr_scheduler = torch.optim.lr_scheduler.StepLR(
216         optimizer,
217         step_size=step_size,
218         gamma=gamma
219     )
220
221     lastStats = [None, None, None,None, None, None,None, None, None,None,
        ↪ None, None]
222
223     for epoch in range(num_epochs):
224         # train for one epoch, printing every 10 iterations
```

```
225         training_return = train_one_epoch(model, optimizer, data_loader,
            ↪ device, epoch, print_freq=10)
226         print("1/3", training_return)
227         # update the learning rate
228         lr_scheduler.step()
229         print("2/3")
230         # evaluate on the test dataset
231         evaluator = evaluate(model, data_loader_test, device=device)
232         print("3/3")
233         curStats = evaluator.coco_eval["segm"].stats
234         print(curStats)
235         isEqual = True
236         for i in range(len(curStats)):
237             if curStats[i] != lastStats[i]:
238                 isEqual = False
239         if isEqual:
240             break
241         torch.save(model, 'object_detector_bckup.pth')
242
243     print("That's it!")
244     torch.save(model, 'object_detector.pth')
245     print("Saved!")
246     return model
247
248
249 if __name__ == "__main__":
250     trainObjDetector()
```

Hinweis: Die Funktion wurde zum leichteren Verständnis gegenüber der tatsächlichen Implementierung gekürzt. Die implementierte Version speichert Statistiken des Trainings in einer Variable zwischen, damit diese über eine Web-Oberfläche ausgelesen werden können. Diese Funktion ist zum Training jedoch nicht notwendig.

C.2 Evaluierung der Objekterkennung

Quellcodeausschnitt C.2: Evaluation des trainierten Mask R-CNN

```
1  import json
2  import pathlib
3  import torch
4  import cv2
5  import numpy as np
6  import pipeline
7
8  def calculateIoU(imageA, imageB):
9      combinedTensor = torch.stack([imageA, imageB], dim=0)
10     union = torch.any(combinedTensor, dim=0)
11     intersection = torch.all(combinedTensor, dim=0)
12     return int(torch.count_nonzero(intersection)) /
       ↪ int(torch.count_nonzero(union))
13
14 def importJson(evalFolder):
15     resultJson = json.load(open(evalFolder / 'result.json', 'r'))
16     images = {}
17     for i in resultJson["images"]:
18         images[i["id"]] = {
19             "width": i["width"],
20             "height": i["height"],
21             "file_name": i["file_name"],
22             "polygonList": []
23         }
24     for i in resultJson["annotations"]:
25         thisAnnotation = []
26         for pt in i["segmentation"]:
27             for k in range(0, len(pt), 2):
28                 thisAnnotation.append([int(pt[k]), int(pt[k + 1])])
29         images[i["image_id"]]["polygonList"].append(thisAnnotation)
30     return images
31
32 def generatePolyImage(width, height, polygonList):
33     mask = np.zeros((height, width, 1), np.uint8)
34     counter = 0
35     for annotation in polygonList:
36         cv2.fillPoly(mask, pts=[np.array(annotation)], color=[255])
37     return mask
38
39 def checkForExtremeValues(thisVal, minVal, maxVal, avgVal, length):
40     if minVal is None or thisVal < minVal:
41         minVal = thisVal
42     if maxVal is None or thisVal > maxVal:
43         maxVal = thisVal
44     if avgVal is None:
45         avgVal = 0.0
```

```
46      avgVal += thisVal / length
47      return minVal, maxVal, avgVal
48
49  def evalObjDetector(evalFolderPath="data\\eval\\"):
50      evalFolder = pathlib.Path(evalFolderPath)
51      objDetectorModel = pipeline.loadObjDetectorModel()
52      resultJson = importJson(evalFolder)
53
54      totalImages = len(resultJson)
55      totalCards = 0
56
57      minTotal = None
58      maxTotal = None
59      avgTotal = None
60
61      minOptimized = None
62      maxOptimized = None
63      avgOptimized = None
64
65      for i in resultJson:
66          thisImg = resultJson[i]
67          totalCards += len(thisImg["polygonList"])
68          actualMask = torch.from_numpy(
69              generatePolyImage(thisImg["width"], thisImg["height"],
                ↪ thisImg["polygonList"]))
70
71          img = pipeline.readImageFromFile(evalFolder / thisImg["file_name"])
72          pred = pipeline.generatePredictions(img, objDetectorModel)
73          mask = pipeline.generateOutputMaskImage(img, pred)
74
75          maskTensor = mask.permute(1, 2, 0).any(2, keepdim=True)
76          minTotal, maxTotal, avgTotal = checkForExtremeValues(
77              calculateIoU(actualMask, maskTensor), minTotal, maxTotal,
                ↪ avgTotal, totalImages)
78
79          optimizedMask = pipeline.convert_mask_to_polygon(mask)
80          optimizedMask = pipeline.approxCardOuter(optimizedMask)
81          optimizedMask = pipeline.filterPolysByEdgeLengths(optimizedMask)
82          optimizedMaskImg =
            ↪ torch.from_numpy(generatePolyImage(thisImg["width"],
            ↪ thisImg["height"], optimizedMask))
83
84          optimizedActual =
            ↪ pipeline.approxCardOuter(np.array(thisImg["polygonList"]))
85          optimizedActual =
            ↪ pipeline.filterPolysByEdgeLengths(optimizedActual)
86          optimizedActualImg =
            ↪ torch.from_numpy(generatePolyImage(thisImg["width"],
            ↪ thisImg["height"], optimizedActual))
87
88          minOptimized, maxOptimized, avgOptimized = checkForExtremeValues(
```

```
89              calculateIoU(optimizedMaskImg, optimizedActualImg),
                ↪ minOptimized, maxOptimized, avgOptimized, totalImages)
90
91      return {
92          "JustMask": (minTotal, avgTotal, maxTotal),
93          "Oprimized": (minOptimized, avgOptimized, maxOptimized),
94          "TotalImages": totalImages,
95          "TotalCards": totalCards,
96      }
97
98  if __name__ == "__main__":
99      print(evalObjDetector())
```

C.3 Selbsttrainierende Objekterkennung

Quellcodeausschnitt C.3: Generierung von Trainingsdaten mit Hilfe der Objekterkennung

```
1  import pathlib
2  import pipeline
3  import cv2
4  import numpy as np
5
6  def generateMaskImg(width, height, polygonList, white=False):
7      maskImg = np.zeros((width, height, 1), np.uint8)
8      counter = 0
9
10     for k in polygonList:
11         if white:
12             cv2.fillPoly(maskImg, pts=[np.array(k)], color=[255])
13         else:
14             cv2.fillPoly(maskImg, pts=[np.array(k)], color=[counter])
15             counter += 1
16     return maskImg
17
18 def paintPolygons(image, poly):
19     return cv2.drawContours(image, poly, -1, (255, 0, 0), 20)
20
21
22 def selfTraining():
23     baseFolder = pathlib.Path("data\\cardsSelfTraining")
24     imgFolder = baseFolder / "input"
25     imagesFolder = baseFolder / "images"
26     masks = baseFolder / "masks"
27     whiteMasks = baseFolder / "whiteMasks"
28     markedImage = baseFolder / "markedImage"
29     transformedImageFolder = baseFolder / "transformedImages"
30
31     imagesFolder.mkdir(parents=True, exist_ok=True)
32     masks.mkdir(parents=True, exist_ok=True)
33     whiteMasks.mkdir(parents=True, exist_ok=True)
34     markedImage.mkdir(parents=True, exist_ok=True)
35     transformedImageFolder.mkdir(parents=True, exist_ok=True)
36
37     maxFiles = None
38     fileCounter = 0
39     for imageFile in imgFolder.iterdir():
40         if imageFile.is_file():
41             cardFileCounter = 0
42             origFilename = str(imageFile.stem)
43             print("IMAGE", fileCounter, ":", origFilename)
44             img = pipeline.readImageFromFile(imageFile)
45             pred = pipeline.generatePredictions(img,
               ↪ pipeline.loadObjDetectorModel())
```

```
46              predictedCards = pred["masks"].size(0)
47              mask = pipeline.generateOutputMaskImage(img, pred)
48              polygonList = pipeline.convert_mask_to_polygon(mask)
49              predictedPolygons = len(polygonList)
50              polygonList = pipeline.approxCardOuter(polygonList)
51              rectangularPolygons = len(polygonList)
52              polygonList = pipeline.filterPolysByEdgeLengths(polygonList)
53              filteredPolygons = len(polygonList)
54              print("Detected Cards:", "OBJ_DETECT:", predictedCards,
                ↪ "POLY_OPENCV:", predictedPolygons, "SQUARE_POLY:",
55                    rectangularPolygons, "FILTERED_POLY:", filteredPolygons)
56
57              imgDimensions = list(img.size())
58
59              cv2.imwrite(
60                  str(masks / (origFilename + ".png")),
61                  generateMaskImg(imgDimensions[1], imgDimensions[2],
                    ↪ polygonList, white=False))
62
63              cv2.imwrite(
64                  str(whiteMasks / (origFilename + ".png")),
65                  generateMaskImg(imgDimensions[1], imgDimensions[2],
                    ↪ polygonList, white=True))
66
67              cv2.imwrite(
68                  str(markedImage / (origFilename + ".png")),
69                  paintPolygons(cv2.imread(imageFile, cv2.IMREAD_COLOR_BGR |
                    ↪ cv2.IMREAD_IGNORE_ORIENTATION), polygonList))
70
71              cv2.imwrite(
72                  str(imagesFolder / (origFilename + ".jpg")),
73                  cv2.imread(imageFile, cv2.IMREAD_COLOR_BGR |
                    ↪ cv2.IMREAD_IGNORE_ORIENTATION))
74
75              transformedImages = pipeline.transformImageArrayToCards(
76                  cv2.imread(imageFile, cv2.IMREAD_COLOR_BGR |
                    ↪ cv2.IMREAD_IGNORE_ORIENTATION),
77                  polygonList, size=300)
78              for i in transformedImages:
79                  filename = "img" + str(fileCounter) + "card" +
                    ↪ str(cardFileCounter) + ".png"
80                  cv2.imwrite(str(transformedImageFolder / filename), i)
81                  cardFileCounter += 1
82              fileCounter += 1
```

Hinweis: Die Funktion generiert in dieser Form zusätzliche Bilder, die zur menschlichen Evaluation dienen, jedoch für die Trainingsdaten nicht notwendig sind.

C.4 Training der Klassifizierungsnetze

Quellcodeausschnitt C.4: Training der Klassifizierung

```
import json
import os
import pathlib
import time
from pathlib import Path
import torch
import torchvision
from torchvision.io import read_image
from torchvision import transforms
from torchvision.models import (
    regnet_y_16gf, RegNet_Y_16GF_Weights,
    vit_b_16, ViT_B_16_Weights,
    efficientnet_v2_m, EfficientNet_V2_M_Weights,
    efficientnet_v2_s, EfficientNet_V2_S_Weights,
    efficientnet_b4, EfficientNet_B4_Weights,
    efficientnet_b3, EfficientNet_B3_Weights,
    regnet_y_1_6gf, RegNet_Y_1_6GF_Weights,
    efficientnet_b2, EfficientNet_B2_Weights,
    efficientnet_b1, EfficientNet_B1_Weights,
    regnet_y_800mf, RegNet_Y_800MF_Weights,
    efficientnet_b0, EfficientNet_B0_Weights,
    regnet_y_400mf, RegNet_Y_400MF_Weights,
    shufflenet_v2_x1_5, ShuffleNet_V2_X1_5_Weights,
    mobilenet_v2, MobileNet_V2_Weights,
    shufflenet_v2_x1_0, ShuffleNet_V2_X1_0_Weights,
    mnasnet0_5, MNASNet0_5_Weights,
    shufflenet_v2_x0_5, ShuffleNet_V2_X0_5_Weights,
    squeezenet1_1, SqueezeNet1_1_Weights,
)
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim


class CustomImageDataset(torch.utils.data.Dataset):
    def getFiles(self):
        files = []
        root = Path(self.img_dir)
        classes = {}
        if (root / "classes.json").exists():
            with open(root / "classes.json", "r") as f:
                classes = json.load(f)
        for i in root.iterdir():
```

```
47              if i.is_dir() and i.name not in classes.values():
48                  if len(classes) == 0:
49                      classId = 0
50                  else:
51                      classId = max(classes.keys()) + 1
52                  classes[classId] = i.name
53          json.dump(classes, open(root / "classes.json", "w"))
54          for c in classes:
55              class_dir = root / classes[c]
56              for f in class_dir.iterdir():
57                  if f.is_file():
58                      files.append({
59                          "name": f.name,
60                          "path": class_dir / f.name,
61                          "class_name": classes[c],
62                          "class_id": c
63                      })
64          return files, classes
65
66      def getClasses(self):
67          return self.classes
68
69      def __init__(self, img_dir, transform=None, target_transform=None):
70          self.img_dir = img_dir
71          self.img_labels, self.classes = self.getFiles()
72          self.transform = transform
73          self.target_transform = target_transform
74
75      def __len__(self):
76          return len(self.img_labels)
77
78      def __getitem__(self, idx):
79          img_path = self.img_labels[idx]["path"]
80          image = read_image(img_path)
81          label = self.img_labels[idx]["class_id"]
82          if self.transform:
83              image = self.transform(image)
84          if self.target_transform:
85              label = self.target_transform(label)
86          return image, int(label)
87
88
89  # DEFINE OWN CNN
90  class OwnCNN(nn.Module):
91      def __init__(self, num_classes=1000):
92          super().__init__()
93          self.conv1 = nn.Conv2d(3, 60, 5)
94          self.pool = nn.MaxPool2d(2, 2)
95          self.conv2 = nn.Conv2d(60, 16, 5)
96          self.fc1 = nn.Linear(16 * 72 * 72, 120)
97          self.fc2 = nn.Linear(120, 84)
```

```
98          self.fc3 = nn.Linear(84, num_classes)
99          self.fc = self.fc3
100
101     def forward(self, x):
102         x = self.pool(F.relu(self.conv1(x)))
103         x = self.pool(F.relu(self.conv2(x)))
104         x = torch.flatten(x, 1)  # flatten all dimensions except batch
105         x = F.relu(self.fc1(x))
106         x = F.relu(self.fc2(x))
107         x = self.fc3(x)
108         return x
109
110
111 def train(folder, net, netName, device, epochs=20, modifyLastLayer=True):
112     net.to(device)
113
114     print("TRAINING", netName, "ON", device)
115
116     transform = transforms.Compose(
117         [transforms.ToTensor(),
118          transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
119
120     training_data = CustomImageDataset(
121         img_dir=folder,
122         transform=getTransformForNet(netName)
123     )
124
125     trainloader = DataLoader(training_data, batch_size=4, shuffle=True)
126     classes = training_data.getClasses()
127
128     if modifyLastLayer:
129         if hasattr(net, "classifier"):
130             lastLinear = None
131             for i in range(len(net.classifier)):
132                 if isinstance(net.classifier[i], torch.nn.Linear):
133                     lastLinear = i
134             if lastLinear is not None:
135                 print("Modifiziere letzte Linear-Schicht um Klassenanzahl
                    ↪ anzupassen von",
136                       "in:", net.classifier[lastLinear].in_features,
                          ↪ "out:", net.classifier[lastLinear].out_features)
137                 net.classifier[lastLinear] = torch.nn.Linear(
                    ↪ in_features=net.classifier[lastLinear].in_features,
                    ↪ out_features=len(classes), bias=True).to(device)
138                 print("zu in:", net.classifier[lastLinear].in_features,
                    ↪ "out:", net.classifier[lastLinear].out_features)
139             else:
140                 last_conv2d = None
141                 for i in range(len(net.classifier)):
142                     if isinstance(net.classifier[i], torch.nn.Conv2d):
143                         last_conv2d = i
```

```
144                 if last_conv2d is not None:
145                     print("Modifiziere letzte conv2d-Schicht um
                        ↪ Klassenanzahl anzupassen von",
146                           "in:", net.classifier[last_conv2d].in_channels,
                              ↪ "out:",
147                           net.classifier[last_conv2d].out_channels)
148                     net.classifier[last_conv2d] = torch.nn.Conv2d(
                        ↪ net.classifier[last_conv2d].in_channels,
                        ↪ len(classes),
                        ↪ kernel_size=net.classifier[last_conv2d].kernel_size,
                        ↪ stride=net.classifier[last_conv2d].stride).to(device)
149                     print("zu in:",
                        ↪ net.classifier[last_conv2d].in_channels, "out:",
150                           net.classifier[last_conv2d].out_channels)
151                 else:
152                     print("Letzte Schicht ist kein Objekt von
                        ↪ torch.nn.Linear... Bitte manuell untersuchen")
153         elif hasattr(net, "fc"):
154             if isinstance(net.fc, torch.nn.Linear):
155                 print("Modifiziere letzte Schicht um Klassenanzahl
                    ↪ anzupassen von",
156                       "in:", net.fc.in_features, "out:",
                          ↪ net.fc.out_features)
157                 net.fc = torch.nn.Linear(in_features=net.fc.in_features,
                    ↪ out_features=len(classes), bias=True).to(
158                     device)
159                 print("zu in:", net.fc.in_features, "out:",
                    ↪ net.fc.out_features)
160             else:
161                 print("Letzte Schicht ist kein Objekt von
                    ↪ torch.nn.Linear... Bitte manuell untersuchen")
162         elif hasattr(net, "heads"):
163             seq = len(net.heads)
164             if isinstance(net.heads[seq - 1], torch.nn.Linear):
165                 print("Modifiziere letzte Schicht um Klassenanzahl
                    ↪ anzupassen von",
166                       "in:", net.heads[seq - 1].in_features, "out:",
                          ↪ net.heads[seq - 1].out_features)
167                 net.heads[seq - 1] =
                    ↪ torch.nn.Linear(in_features=net.heads[seq -
                    ↪ 1].in_features,
168                                                       out_features=len(classes),
169                                                       bias=True).to(device)
170                 print("zu in:", net.heads[seq - 1].in_features, "out:",
                    ↪ net.heads[seq - 1].out_features)
171             else:
172                 print("Letzte Schicht ist kein Objekt von
                    ↪ torch.nn.Linear... Bitte manuell untersuchen")
173         else:
174             print("Letzte Schicht ist kein Objekt von torch.nn.Linear...
                ↪ Bitte manuell untersuchen")
```

```
175
176     # if netName in ["efficientnet_v2_m"]:
177     #     net.classifier[1] =
        ↪ torch.nn.Linear(in_features=net.classifier[1].in_features,
        ↪ out_features=len(classes), bias=True).to(device)
178
179     criterion = nn.CrossEntropyLoss()
180     criterion.to(device)
181     optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
182
183     totalTrainingDuration = 0.0
184     statistics = {}
185
186     for epoch in range(epochs):  # loop over the dataset multiple times
187         net.train()
188         trainingDurationStart = time.time()
189         epoch_loss = 0.0
190         for i, data in enumerate(trainloader, 0):
191             # get the inputs; data is a list of [inputs, labels]
192             inputs, labels = data[0].to(device), data[1].to(device)
193
194             # zero the parameter gradients
195             optimizer.zero_grad()
196
197             # forward + backward + optimize
198             outputs = net(inputs)
199             loss = criterion(outputs, torch.stack(list(labels), dim=0))
200             loss.backward()
201             optimizer.step()
202
203             # print statistics
204             epoch_loss += loss.item()
205         print("[ EPOCH", epoch + 1, "] loss:", epoch_loss / len(classes))
206         epochTrainingDuration = (time.time() - trainingDurationStart)
207         totalTrainingDuration += epochTrainingDuration
208         net.eval()
209         evaluation = evaluateNet(folder, net, netName)
210         print("[EVAL]", network, "EPOCH TRAINING DURATION:",
            ↪ epochTrainingDuration, "EVALUATION:", evaluation)
211         statistics[epoch] = {
212             "EPOCH_TRAINING_DURATION": epochTrainingDuration,
213             "EPOCH_LOSS": epoch_loss,
214             "EVALUATION": evaluation
215         }
216     # print("Finished Training of", network, " TOTAL_DURATION:",
        ↪ totalTrainingDuration)
217     return net, statistics
218
219
220 def getResizeForNet(netName):
```

```
221     if netName in ["vit_b_16", "efficientnet_v2_m", "efficientnet_v2_s",
        ↪ "efficientnet_b4", "efficientnet_b3",
222                    "efficientnet_b2", "efficientnet_b1", ]:
223         return 224
224     else:
225         return 300
226
227
228 def getTransformForNet(netName):
229     imgSize = getResizeForNet(netName)
230     return transforms.Compose([
231         transforms.ToPILImage(),
232         transforms.Resize((imgSize, imgSize)),
233         transforms.ToTensor()
234     ])
235
236
237 def evaluateNet(folder, cnn, netName):
238     check_data = CustomImageDataset(
239         img_dir=folder,
240         transform=getTransformForNet(netName)
241     )
242     check_dataloader = DataLoader(check_data, batch_size=1, shuffle=True)
243
244     dataiter = iter(check_dataloader)
245     nextItem = next(dataiter, None)
246     correctCounter = 0
247     wrongCounter = 0
248     totalEvaluationTime = 0.0
249     itemCounter = 0
250
251     while nextItem is not None:
252         images, labels = nextItem[0].to(device), nextItem[1].to(device)
253         correctLabels = nextItem[1]
254         evaluateDurationStart = time.time()
255         pred = cnn(images)
256         evaluateDuration = (time.time() - evaluateDurationStart)
257         totalEvaluationTime += evaluateDuration
258         items = len(correctLabels)
259         predClasses = torch.argmax(pred, 1)
260         for i in range(items):
261             if (int(predClasses[i]) == int(correctLabels[i])):
262                 correctCounter += 1
263             else:
264                 wrongCounter += 1
265         itemCounter += items
266         nextItem = next(dataiter, None)
267     return {
268         "AVG_DURATION": (totalEvaluationTime / itemCounter),
269         "TOTAL_DURATION": totalEvaluationTime,
270         "CORRECT": correctCounter,
```

```
271         "WRONG": wrongCounter,
272         "ITEMS": itemCounter,
273         "CORRECT_RATE": round(((correctCounter / itemCounter) * 100), 2),
274     }
275
276 networkNames = [
277         "regnet_y_16gf",
278         "vit_b_16",
279         "efficientnet_v2_m",
280         "efficientnet_v2_s",
281         "efficientnet_b4",
282         "efficientnet_b3",
283         "regnet_y_1_6gf",
284         "efficientnet_b2",
285         "efficientnet_b1",
286         "regnet_y_800mf",
287         "efficientnet_b0",
288         "regnet_y_400mf",
289         "shufflenet_v2_x1_5",
290         "mobilenet_v2",
291         "shufflenet_v2_x1_0",
292         "mnasnet0_5",
293         "shufflenet_v2_x0_5",
294         "squeezenet1_1",
295         "OwnCNN"
296     ]
297
298 if __name__ == '__main__':
299     folder = "data\\classesExpanded"
300
301     classes = CustomImageDataset(
302         img_dir=folder,
303         transform=getTransformForNet("OwnCNN")
304     ).getClasses()
305
306     networksInit = {
307         "regnet_y_16gf": regnet_y_16gf(num_classes=len(classes)),
308         "vit_b_16": vit_b_16(num_classes=len(classes)),
309         "efficientnet_v2_m": efficientnet_v2_m(num_classes=len(classes)),
310         "efficientnet_v2_s": efficientnet_v2_s(num_classes=len(classes)),
311         "efficientnet_b4": efficientnet_b4(num_classes=len(classes)),
312         "efficientnet_b3": efficientnet_b3(num_classes=len(classes)),
313         "regnet_y_1_6gf": regnet_y_1_6gf(num_classes=len(classes)),
314         "efficientnet_b2": efficientnet_b2(num_classes=len(classes)),
315         "efficientnet_b1": efficientnet_b1(num_classes=len(classes)),
316         "regnet_y_800mf": regnet_y_800mf(num_classes=len(classes)),
317         "efficientnet_b0": efficientnet_b0(num_classes=len(classes)),
318         "regnet_y_400mf": regnet_y_400mf(num_classes=len(classes)),
319         "shufflenet_v2_x1_5": shufflenet_v2_x1_5(num_classes=len(classes)),
320         "mobilenet_v2": mobilenet_v2(num_classes=len(classes)),
321         "shufflenet_v2_x1_0": shufflenet_v2_x1_0(num_classes=len(classes)),
```

```
322         "mnasnet0_5": mnasnet0_5(num_classes=len(classes)),
323         "shufflenet_v2_x0_5": shufflenet_v2_x0_5(num_classes=len(classes)),
324         "squeezenet1_1": squeezenet1_1(num_classes=len(classes)),
325         "OwnCNN": regnet_y_16gf(num_classes=len(classes))
326     }
327
328     networksWeighted = {
329         "regnet_y_16gf":
            ↪ regnet_y_16gf(weights=RegNet_Y_16GF_Weights.DEFAULT),
330         "vit_b_16": vit_b_16(weights=ViT_B_16_Weights.DEFAULT),
331         "efficientnet_v2_m":
            ↪ efficientnet_v2_m(weights=EfficientNet_V2_M_Weights.DEFAULT),
332         "efficientnet_v2_s":
            ↪ efficientnet_v2_s(weights=EfficientNet_V2_S_Weights.DEFAULT),
333         "efficientnet_b4":
            ↪ efficientnet_b4(weights=EfficientNet_B4_Weights.DEFAULT),
334         "efficientnet_b3":
            ↪ efficientnet_b3(weights=EfficientNet_B3_Weights.DEFAULT),
335         "regnet_y_1_6gf":
            ↪ regnet_y_1_6gf(weights=RegNet_Y_1_6GF_Weights.DEFAULT),
336         "efficientnet_b2":
            ↪ efficientnet_b2(weights=EfficientNet_B2_Weights.DEFAULT),
337         "efficientnet_b1":
            ↪ efficientnet_b1(weights=EfficientNet_B1_Weights.DEFAULT),
338         "regnet_y_800mf":
            ↪ regnet_y_800mf(weights=RegNet_Y_800MF_Weights.DEFAULT),
339         "efficientnet_b0":
            ↪ efficientnet_b0(weights=EfficientNet_B0_Weights.DEFAULT),
340         "regnet_y_400mf":
            ↪ regnet_y_400mf(weights=RegNet_Y_400MF_Weights.DEFAULT),
341         "shufflenet_v2_x1_5":
            ↪ shufflenet_v2_x1_5(weights=ShuffleNet_V2_X1_5_Weights.DEFAULT),
342         "mobilenet_v2": mobilenet_v2(weights=MobileNet_V2_Weights.DEFAULT),
343         "shufflenet_v2_x1_0":
            ↪ shufflenet_v2_x1_0(weights=ShuffleNet_V2_X1_0_Weights.DEFAULT),
344         "mnasnet0_5": mnasnet0_5(weights=MNASNet0_5_Weights.DEFAULT),
345         "shufflenet_v2_x0_5":
            ↪ shufflenet_v2_x0_5(weights=ShuffleNet_V2_X0_5_Weights.DEFAULT),
346         "squeezenet1_1":
            ↪ squeezenet1_1(weights=SqueezeNet1_1_Weights.DEFAULT),
347     }
348
349     networksWeightedAndModified = {
350         "regnet_y_16gf":
            ↪ regnet_y_16gf(weights=RegNet_Y_16GF_Weights.DEFAULT),
351         "vit_b_16": vit_b_16(weights=ViT_B_16_Weights.DEFAULT),
352         "efficientnet_v2_m":
            ↪ efficientnet_v2_m(weights=EfficientNet_V2_M_Weights.DEFAULT),
353         "efficientnet_v2_s":
            ↪ efficientnet_v2_s(weights=EfficientNet_V2_S_Weights.DEFAULT),
```

```
354         "efficientnet_b4":
            ↪ efficientnet_b4(weights=EfficientNet_B4_Weights.DEFAULT),
355         "efficientnet_b3":
            ↪ efficientnet_b3(weights=EfficientNet_B3_Weights.DEFAULT),
356         "regnet_y_1_6gf":
            ↪ regnet_y_1_6gf(weights=RegNet_Y_1_6GF_Weights.DEFAULT),
357         "efficientnet_b2":
            ↪ efficientnet_b2(weights=EfficientNet_B2_Weights.DEFAULT),
358         "efficientnet_b1":
            ↪ efficientnet_b1(weights=EfficientNet_B1_Weights.DEFAULT),
359         "regnet_y_800mf":
            ↪ regnet_y_800mf(weights=RegNet_Y_800MF_Weights.DEFAULT),
360         "efficientnet_b0":
            ↪ efficientnet_b0(weights=EfficientNet_B0_Weights.DEFAULT),
361         "regnet_y_400mf":
            ↪ regnet_y_400mf(weights=RegNet_Y_400MF_Weights.DEFAULT),
362         "shufflenet_v2_x1_5":
            ↪ shufflenet_v2_x1_5(weights=ShuffleNet_V2_X1_5_Weights.DEFAULT),
363         "mobilenet_v2": mobilenet_v2(weights=MobileNet_V2_Weights.DEFAULT),
364         "shufflenet_v2_x1_0":
            ↪ shufflenet_v2_x1_0(weights=ShuffleNet_V2_X1_0_Weights.DEFAULT),
365         "mnasnet0_5": mnasnet0_5(weights=MNASNet0_5_Weights.DEFAULT),
366         "shufflenet_v2_x0_5":
            ↪ shufflenet_v2_x0_5(weights=ShuffleNet_V2_X0_5_Weights.DEFAULT),
367         "squeezenet1_1":
            ↪ squeezenet1_1(weights=SqueezeNet1_1_Weights.DEFAULT),
368     }
369
370     device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
371
372     for trainingType in ["ohneGewichte", "mitGewichten",
        ↪ "mitAngepasstenGewichten"]:
373         if trainingType == "ohneGewichte":
374             networks = networksInit
375             netFolder = "classifier_ohneGewichte\\"
376             pathlib.Path(netFolder).mkdir(parents=True, exist_ok=True)
377         elif trainingType == "mitGewichten":
378             networks = networksWeighted
379             netFolder = "classifier_mitGewichten\\"
380             pathlib.Path(netFolder).mkdir(parents=True, exist_ok=True)
381         else:
382             networks = networksWeightedAndModified
383             netFolder =
                ↪ "classifier_mitGewichtenUndAngepassterKlassenanzahl\\"
384             pathlib.Path(netFolder).mkdir(parents=True, exist_ok=True)
385
386         if Path("trainingStatistics.json").exists():
387             globalStatistics = json.load(open(netFolder +
                ↪ "trainingStatistics.json", "r"))
388         else:
389             globalStatistics = {}
```

```
390
391         for network in networks:
392             if network not in globalStatistics:
393                 if trainingType == "mitAngepasstenGewichten" and network
                    ↪ != "OwnCNN":
394                     modifyLastLayer = True
395                 else:
396                     modifyLastLayer = False
397
398                 cnn, netStatistics = train(folder, networks[network],
                    ↪ network, device, epochs=20,
399                                            modifyLastLayer=modifyLastLayer)
400
401                 torch.save(cnn, (netFolder + 'classifier_' + network +
                    ↪ '.pth'))
402                 globalStatistics[network] = netStatistics
403                 json.dump(globalStatistics, open(netFolder +
                    ↪ 'trainingStatistics.json', 'w'))
```

C.5 Funktionen aus der Klassifizierungsevaluation

Hinweis: Die Funktion wurde zum leichteren Verständnis gegenüber der tatsächlichen Implementierung gekürzt. Die implementierte Version speichert Teilergebnisse in json-Dateien und liest diese, wenn bereits vorhanden, wieder ein, statt die Ergebnisse neu zu berechnen. Die hier dargestellte Version führt immer eine vollständige Berechnung aller Teilschritte durch.

Quellcodeausschnitt C.5: Hauptfunktion der Klassifizierungsevaluation

```
1  def classificationEvaluation():
2    imgFolder = Path("data\\classesExpanded")
3    evalFolder = Path("classifier_evaluation")
4    evalFolder.mkdir(parents=True, exist_ok=True)
5
6    classes = json.load(open(imgFolder / "classes.json"))
7    device = torch.device('cuda') if torch.cuda.is_available() else
     ↪ torch.device('cpu')
8
9    totalNetworksToBeEvaluated = 0
10   modelNameList = list(classifier.networks.keys())
11   evaluations = []
12
13   for trainingType in ["mitGewichten",
     ↪ "mitGewichtenUndAngepassterKlassenanzahl", "ohneGewichte"]:
14     for modelName in modelNameList:
15       netPath = Path("classifier_" + trainingType + "\\classifier_" +
         ↪ modelName + ".pth")
16       if netPath.is_file():
17         totalNetworksToBeEvaluated += 1
18         evaluations.append({
19           "modelName": modelName,
20           "trainingType": trainingType,
21           "netPath": netPath
22         })
23
24   networksToBeEvaluatedCounter = 0
25   ranking = {}
26   orderedRanking = []
27
28   for model in evaluations:
29     modelName = model["modelName"]
30     trainingType = model["trainingType"]
31
32     networksToBeEvaluatedCounter += 1
33     print("[INFO]", "[", networksToBeEvaluatedCounter, "/",
       ↪ totalNetworksToBeEvaluated, "]", "Evaluating network", trainingType,
       ↪ modelName, "...")
34     thisEvalFolder = (evalFolder / trainingType / modelName)
35     thisEvalFolder.mkdir(parents=True, exist_ok=True)
36
```

```
37      netPath = Path("classifier_" + trainingType + "\\classifier_" +
        ↪ modelName + ".pth")
38      net = torch.load(netPath)
39      net.eval()
40      transformer = classifier.getTransformForNet(modelName)
41      evaluation = classifyTestFiles(imgFolder, net, classes, transformer,
        ↪ device)
42      analyzation = analyzeResults(evaluation)
43      accumulation = accumulateResults(analyzation)
44      thresholds = calculateThresholdArea(accumulation)
45      for thisClass in thresholds:
46        for evalType in ["score", "softMax"]:
47          thresholdArea = thresholds[thisClass][evalType]["thresholdArea"]
48          if thresholds[thisClass][evalType]["safeThreshold"]:
49            thresholds[thisClass][evalType]["recommendedThreshold"] =
              ↪ (thresholdArea[0] + thresholdArea[1]) / 2
50          else:
51            searchThreshold = analyzeThresholdProblem(thisClass, evaluation,
              ↪ evalType, thresholds[thisClass][evalType]["thresholdArea"][0],
              ↪ thresholds[thisClass][evalType]["thresholdArea"][1])
52            thresholds[thisClass][evalType]["recommendedThreshold"] =
              ↪ (searchThreshold[0] + searchThreshold[1]) / 2
53          thresholds[thisClass][evalType]["evaluation"] =
            ↪ evaluateThreshold(thisClass, evaluation, evalType,
            ↪ thresholds[thisClass][evalType]["recommendedThreshold"])
54
55      thisRanking = calculateRanking(thresholds)
56
57      thisScoreRanking = thisRanking["mistakesScore"]
58      thisSoftmaxRanking = thisRanking["mistakesSoftmax"]
59      thisDuration = sumDurationsUp(analyzation)
60      bestThresholds = sumThresholdsUp(thresholds)
61
62      if thisScoreRanking not in ranking:
63        ranking[thisScoreRanking] = []
64        ranking[thisScoreRanking].append({
65          "modelName": modelName,
66          "trainingType": trainingType,
67          "evalType": "Score",
68          "duration": thisDuration,
69          "thresholds": bestThresholds["score"],
70        })
71
72      if thisSoftmaxRanking not in ranking:
73        ranking[thisSoftmaxRanking] = []
74      ranking[thisSoftmaxRanking].append({
75        "modelName": modelName,
76        "trainingType": trainingType,
77        "evalType": "SoftMax",
78        "duration": thisDuration,
79        "thresholds": bestThresholds["softMax"],
```

```
80      })
81    orderedRanking = orderRanking(ranking)
82    return orderedRanking
```

C.5.1 Klassifizierung der Evaluationsdaten

Quellcodeausschnitt C.6: Klassifizierung der Evaluationsdaten

```
def classifyTestFiles(imgFolder, net, classes, transform, device):
  result = {}
  softMax = torch.nn.Softmax(dim=1)
  for classFolder in imgFolder.iterdir():
    if classFolder.is_dir():
      combined = []
      for imageFile in classFolder.iterdir():
        if imageFile.is_file():
          with torch.no_grad():
            image = torchvision.io.read_image(imageFile)
            classifyTensor = transform(image).unsqueeze(0).to(device)
            startTime = time()
            classify = torch.narrow(net(classifyTensor), 1, 0, 4)
            duration = time() - startTime
            scoring = classify.tolist()
            classification = classify.argmax(1).tolist()
            classificationDetails = softMax(classify).tolist()
            score = {}
            for classId in range(len(scoring[0])):
              thisClass = classes[str(classId)]
              score[str(thisClass)] = scoring[0][classId]
            softMaxScore = {}
            for classId in range(len(classificationDetails[0])):
              thisClass = classes[str(classId)]
              softMaxScore[str(thisClass)] =
              ↪ classificationDetails[0][classId]
            combined.append({
              "score": score,
              "softMax": softMaxScore,
              "prediction": classes[str(classification[0])],
              "duration": duration
            })
      className = classFolder.stem
      result[className] = combined
  return result
```

C.5.2 Auswertung der gesammelten Daten

Quellcodeausschnitt C.7: Auswertung der gesammelten Daten

```
def analyzeResults(input_data):
  result = {}
  for correctClass in input_data:
    classLength = len(input_data[correctClass])

    classEvaluations = {}
    for i in input_data[correctClass][0]["score"]:
      classEvaluations[i] = {
        "minScore": None,
        "maxScore": None,
        "avgScore": None,

        "minSoftMax": None,
        "maxSoftMax": None,
        "avgSoftMax": None,
      }

    classPredictable = correctClass in input_data[correctClass][0]["score"]
    correctPredictions = 0
    minDuration = None
    maxDuration = None
    avgDuration = None

    for i in input_data[correctClass]:
      for predClass in i["score"]:
        if classEvaluations[predClass]["minScore"] is None or
        ↪ i["score"][predClass] < classEvaluations[predClass]["minScore"]:
          classEvaluations[predClass]["minScore"] = i["score"][predClass]
        if classEvaluations[predClass]["maxScore"] is None or
        ↪ i["score"][predClass] > classEvaluations[predClass]["maxScore"]:
          classEvaluations[predClass]["maxScore"] = i["score"][predClass]
        if classEvaluations[predClass]["avgScore"] is None:
          classEvaluations[predClass]["avgScore"] = 0.0
        classEvaluations[predClass]["avgScore"] += (i["score"][predClass]
        ↪ / classLength)

      for predClass in i["softMax"]:
        if classEvaluations[predClass]["minSoftMax"] is None or
        ↪ i["softMax"][predClass] <
        ↪ classEvaluations[predClass]["minSoftMax"]:
          classEvaluations[predClass]["minSoftMax"] =
          ↪ i["softMax"][predClass]
        if classEvaluations[predClass]["maxSoftMax"] is None or
        ↪ i["softMax"][predClass] >
        ↪ classEvaluations[predClass]["maxSoftMax"]:
```

```
38            classEvaluations[predClass]["maxSoftMax"] =
              ↪ i["softMax"][predClass]
39          if classEvaluations[predClass]["avgSoftMax"] is None:
40            classEvaluations[predClass]["avgSoftMax"] = 0.0
41          classEvaluations[predClass]["avgSoftMax"] +=
            ↪ (i["softMax"][predClass] / classLength)
42
43        if i["prediction"] == correctClass:
44          correctPredictions += 1
45
46        if minDuration is None:
47          minDuration = i["duration"]
48        elif i["duration"] < minDuration:
49          minDuration = i["duration"]
50
51        if maxDuration is None:
52          maxDuration = i["duration"]
53        elif i["duration"] > maxDuration:
54          maxDuration = i["duration"]
55
56        if avgDuration is None:
57          avgDuration = 0.0
58        avgDuration += (i["duration"] / classLength)
59
60      result[correctClass] = {
61        "length": classLength,
62        "evaluations": classEvaluations,
63        "predictable": classPredictable,
64        "minDuration": minDuration,
65        "maxDuration": maxDuration,
66        "avgDuration": avgDuration,
67      }
68
69      if classPredictable:
70        result[correctClass]["correctPredictions"] = correctPredictions
71        result[correctClass]["correctPredictionsPercent"] =
          ↪ (correctPredictions / classLength) * 100
72    return result
```

C.5.3 Zusammenfassen der Ergebnisse je Klasse

Quellcodeausschnitt C.8: Zusammenfassen der Ergebnisse je Klasse

```
def accumulateResults(analyzation):
    globalValues = {}
    ownClass = {}

    for correctClass in analyzation:
        globalValues[correctClass] = {
            "minScore": None,
            "maxScore": None,
            "avgScore": None,
            "minSoftMax": None,
            "maxSoftMax": None,
            "avgSoftMax": None,
            "totalCards": 0
        }
    for correctClass in analyzation:
        for predClass in analyzation[correctClass]["evaluations"]:
            if correctClass != predClass:
                globalValues[predClass]["totalCards"] +=
                ↪ analyzation[correctClass]["length"]
    for correctClass in analyzation:
        for predictedClass in analyzation[correctClass]["evaluations"]:
            if predictedClass == correctClass:
                ownClass[correctClass] =
                ↪ analyzation[correctClass]["evaluations"][predictedClass]
                ↪ .copy()
                ownClass[correctClass]["totalCards"] =
                ↪ analyzation[correctClass]["length"]
            else:
                if globalValues[predictedClass]["minScore"] is None or
                ↪ analyzation[correctClass]["evaluations"][predictedClass]
                ↪ ["minScore"] < globalValues[predictedClass]["minScore"]:
                    globalValues[predictedClass]["minScore"] =
                    ↪ analyzation[correctClass]["evaluations"]
                    ↪ [predictedClass]["minScore"]
                if globalValues[predictedClass]["maxScore"] is None or
                ↪ analyzation[correctClass]["evaluations"][predictedClass]
                ↪ ["maxScore"] > globalValues[predictedClass]["maxScore"]:
                    globalValues[predictedClass]["maxScore"] =
                    ↪ analyzation[correctClass]["evaluations"]
                    ↪ [predictedClass]["maxScore"]
                if globalValues[predictedClass]["avgScore"] is None:
                    globalValues[predictedClass]["avgScore"] = 0.0
                globalValues[predictedClass]["avgScore"] +=
                ↪ analyzation[correctClass]["evaluations"][predictedClass]
                ↪ ["avgScore"] * (analyzation[correctClass]["length"] /
                ↪ globalValues[predictedClass]["totalCards"])
```

```
32
33                  if globalValues[predictedClass]["minSoftMax"] is None or
                    ↪ analyzation[correctClass]["evaluations"][predictedClass]
                    ↪ ["minSoftMax"] <
                    ↪ globalValues[predictedClass]["minSoftMax"]:
34                      globalValues[predictedClass]["minSoftMax"] =
                        ↪ analyzation[correctClass]["evaluations"]
                        ↪ [predictedClass]["minSoftMax"]
35                  if globalValues[predictedClass]["maxSoftMax"] is None or
                    ↪ analyzation[correctClass]["evaluations"]
                    ↪ [predictedClass]["maxSoftMax"] >
                    ↪ globalValues[predictedClass]["maxSoftMax"]:
36                      globalValues[predictedClass]["maxSoftMax"] =
                        ↪ analyzation[correctClass]["evaluations"]
                        ↪ [predictedClass]["maxSoftMax"]
37                  if globalValues[predictedClass]["avgSoftMax"] is None:
38                      globalValues[predictedClass]["avgSoftMax"] = 0.0
39                  globalValues[predictedClass]["avgSoftMax"] +=
                    ↪ analyzation[correctClass]["evaluations"][predictedClass]
                    ↪ ["avgSoftMax"] * (analyzation[correctClass]["length"] /
                    ↪ globalValues[predictedClass]["totalCards"])
40
41      result = {}
42
43      for className in analyzation:
44          if analyzation[className]["predictable"]:
45              result[className] = {
46                  "own": ownClass[className],
47                  "others": globalValues[className],
48              }
49
50      return result
```

C.5.4 Bestimmung des Schwellwert-Bereichs

Quellcodeausschnitt C.9: Bestimmung des Schwellwertbereichs

```
def calculateThresholdArea(accumulation):
    result = {}
    sumUp = None
    for thisClass in accumulation:
        localSumUp = (accumulation[thisClass]["own"]["totalCards"] +
        ↪ accumulation[thisClass]["others"]["totalCards"])
        result[thisClass] = {
            "totalCards": localSumUp
        }
        if sumUp is None:
            sumUp = localSumUp
        elif localSumUp != sumUp:
            raise Exception("Etwas stimmt nicht mit der Summe bei",
            ↪ thisClass, "...: Ergebnis sollte", sumUp, "sein, ist aber",
                  localSumUp)

        for evalType in ["score", "softMax"]:
            if evalType == "score":
                ownMin = accumulation[thisClass]["own"]["minScore"]
                otherMax = accumulation[thisClass]["others"]["maxScore"]
            elif evalType == "softMax":
                ownMin = accumulation[thisClass]["own"]["minSoftMax"]
                otherMax = accumulation[thisClass]["others"]["maxSoftMax"]
            else:
                raise Exception("[ERROR] Unbekannter Evaluationstyp")

            if ownMin > otherMax:
                result[thisClass][evalType] = {
                    "safeThreshold": True,
                    "thresholdArea": (otherMax, ownMin)
                }
            else:
                result[thisClass][evalType] = {
                    "safeThreshold": False,
                    "thresholdArea": (ownMin, otherMax)
                }
    return result
```

C.5.5 Bestimmung eines bestmöglichen Schwellwerts

Quellcodeausschnitt C.10: Bestimmung des Schwellwerts bei uneindeutigen Bereichen

```
1  def analyzeThresholdProblem(className, evaluation, evalType, thresholdMin,
   ↪ thresholdMax):
2      ownValues = []
3      otherValues = []
4      lowestOwnValueAboveThreshold = None
5      highestOtherValueUnderThreshold = None
6      for curClass in evaluation:
7          for i in evaluation[curClass]:
8              thisValue = i[evalType][className]
9              if thresholdMin <= thisValue <= thresholdMax:
10                 if className == curClass:
11                     ownValues.append(thisValue)
12                 else:
13                     otherValues.append(thisValue)
14             elif thisValue > thresholdMax and className == curClass:
15                 if lowestOwnValueAboveThreshold is None or thisValue <
                   ↪ lowestOwnValueAboveThreshold:
16                     lowestOwnValueAboveThreshold = thisValue
17             elif thisValue < thresholdMin and className != curClass:
18                 if highestOtherValueUnderThreshold is None or thisValue >
                   ↪ highestOtherValueUnderThreshold:
19                     highestOtherValueUnderThreshold = thisValue
20     totalValues = (ownValues + otherValues)
21     if lowestOwnValueAboveThreshold is not None:
22         totalValues.append(lowestOwnValueAboveThreshold)
23     if highestOtherValueUnderThreshold is not None:
24         totalValues.append(highestOtherValueUnderThreshold)
25     totalValues.sort()
26
27     improvedTotalValues = []
28     curValue = None
29     while len(totalValues) > 0:
30         if curValue is None:
31             curValue = totalValues[0]
32             improvedTotalValues.append(totalValues.pop(0))
33         if totalValues[0] > curValue:
34             improvedTotalValues.append((curValue + totalValues[0]) / 2)
35             curValue = totalValues[0]
36             improvedTotalValues.append(totalValues.pop(0))
37         elif totalValues[0] == curValue:
38             totalValues.pop(0)
39         else:
40             print("Etwas stimmt nicht mit totalValues")
41     totalValues = improvedTotalValues
42
43     lowestMistakes = None
```

```
44      lowestMistakesThresholds = []
45      for pos in range(len(totalValues)):
46          thresh = totalValues[pos]
47          currentMistakes = 0
48          for i in ownValues:
49              if i < thresh:
50                  currentMistakes += 1
51          for i in otherValues:
52              if i >= thresh:
53                  currentMistakes += 1
54          if lowestMistakes is None:
55              lowestMistakes = currentMistakes
56              lowestMistakesThresholds.append(pos)
57          elif lowestMistakes == currentMistakes:
58              lowestMistakesThresholds.append(pos)
59          elif lowestMistakes > currentMistakes:
60              lowestMistakes = currentMistakes
61              lowestMistakesThresholds = [pos]
62
63      if len(lowestMistakesThresholds) <= 0:
64          print("Etwas ist schief gelaufen mit Threshold-Suche für",
            ↪ className, "im Bereich", thresholdMin, thresholdMax)
65      else:
66          thresholdAreas = []
67          curStart = None
68          curIndex = None
69          while len(lowestMistakesThresholds) > 0:
70              if curStart is None:
71                  curStart = lowestMistakesThresholds.pop(0)
72                  curIndex = curStart
73              elif lowestMistakesThresholds[0] == (curIndex + 1):
74                  curIndex = lowestMistakesThresholds.pop(0)
75              else:
76                  thresholdAreas.append((totalValues[curStart],
                    ↪ totalValues[curIndex]))
77                  curStart = lowestMistakesThresholds.pop(0)
78                  curIndex = curStart
79          if curStart is not None and curIndex is not None:
80              thresholdAreas.append((totalValues[curStart],
                ↪ totalValues[curIndex]))
81          else:
82              print("Etwas ist schief gelaufen mit Threshold-Suche für",
                ↪ className, "im Bereich", thresholdMin,
83                    thresholdMax, "(while hat nicht iteriert)")
84          bestDifference = None
85          bestThreshold = None
86          for i in thresholdAreas:
87              thisDifference = (i[1] - i[0])
88              if bestDifference is None:
89                  bestDifference = thisDifference
90                  bestThreshold = i
```

```
91              elif bestDifference < thisDifference:
92                  bestDifference = thisDifference
93                  bestThreshold = i
94          return bestThreshold
```

C.5.6 Evaluierung des bestimmten Schwellwerts

Quellcodeausschnitt C.11: Evaluierung der Schwellwerte

```
1  def evaluateThreshold(className, evaluation, evalType, thresholds,
   ↪ classes):
2      truePositive = 0
3      falsePositive = 0
4      trueNegative = 0
5      falseNegative = 0
6
7      for curClass in evaluation:
8          if curClass in classes.values():
9              correctPrediction = curClass
10         else:
11             correctPrediction = None
12         for i in evaluation[curClass]:
13             if i[evalType][i["prediction"]] >
               ↪ thresholds[i["prediction"]][evalType]
               ↪ ["recommendedThreshold"]:
14                 prediction = i["prediction"]
15             else:
16                 prediction = None
17
18             if correctPrediction == className:
19                 if prediction == className:
20                     truePositive += 1
21                 else:
22                     falseNegative += 1
23             else:
24                 if prediction == className:
25                     falsePositive += 1
26                 else:
27                     trueNegative += 1
28
29     if (truePositive + falsePositive) == 0:
30         precision = 0.0
31     else:
32         precision = truePositive / (truePositive + falsePositive)
33
34     if (truePositive + falseNegative) == 0:
35         recall = 0.0
36     else:
37         recall = truePositive / (truePositive + falseNegative)
38
39     return {
40         "truePositive": truePositive,
41         "falsePositive": falsePositive,
42         "trueNegative": trueNegative,
43         "falseNegative": falseNegative,
```

```
44          "precision": precision,
45          "recall": recall
46      }
```

C.5.7 Bewertung aller Klassifikationsnetze

Quellcodeausschnitt C.12: Bewertung der Klassifikationsnetze

```
def calculateRanking(thresholds):
    mistakesScore = 0
    mistakesSoftmax = 0
    for className in thresholds:
        thisScoreEval = thresholds[className]["score"]["evaluation"]
        mistakesScore += thisScoreEval["falsePositive"] +
        ↪ thisScoreEval["falseNegative"]
        thisScoreEval = thresholds[className]["softMax"]["evaluation"]
        mistakesSoftmax += thisScoreEval["falsePositive"] +
        ↪ thisScoreEval["falseNegative"]
    return {"mistakesScore": mistakesScore, "mistakesSoftmax":
    ↪ mistakesSoftmax}

def orderRanking(ranking):
    rankingScores = list(ranking.keys())
    rankingScores.sort()
    place = 1
    result = []
    for scoreRes in rankingScores:
        curPlace = place
        for i in ranking[scoreRes]:
            result.append({
                "place": curPlace,
                "score": scoreRes,
                "modelName": i["modelName"],
                "trainingType": i["trainingType"],
                "evalType": i["evalType"],
                "duration": i["duration"],
                "thresholds": i["thresholds"],
            })
            place += 1
    return result

def sumDurationsUp(analyzation):
    totalCards = 0
    minDuration = None
    maxDuration = None
    avgDuration = None
    for thisClass in analyzation:
        if minDuration is None or analyzation[thisClass]["minDuration"] <
        ↪ minDuration:
            minDuration = analyzation[thisClass]["minDuration"]
        if maxDuration is None or analyzation[thisClass]["maxDuration"] >
        ↪ maxDuration:
            maxDuration = analyzation[thisClass]["maxDuration"]
        totalCards += analyzation[thisClass]["length"]
```

```
42      if totalCards > 0:
43          avgDuration = 0.0
44      for thisClass in analyzation:
45          avgDuration += analyzation[thisClass]["avgDuration"] *
            ↪ (analyzation[thisClass]["length"] / totalCards)
46      return {
47          "minDuration": minDuration,
48          "maxDuration": maxDuration,
49          "avgDuration": avgDuration,
50      }
51
52  def sumThresholdsUp(thresholds):
53      softMax = {}
54      score = {}
55      for thisClass in thresholds:
56          score[thisClass] =
            ↪ thresholds[thisClass]["score"]["recommendedThreshold"]
57          softMax[thisClass] =
            ↪ thresholds[thisClass]["softMax"]["recommendedThreshold"]
58      return {
59          "softMax": softMax,
60          "score": score,
61      }
```

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 20. April 2025