Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Artur Voskanyan

Entwicklung eines dateneffizienten Deep Learning Verfahrens zur automatisierten Sendungsverfolgung in der Logistik

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Artur Voskanyan

Entwicklung eines dateneffizienten Deep Learning Verfahrens zur automatisierten Sendungsverfolgung in der Logistik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Jörg Dahlkemper
Zweitgutachter: Dr. rer. pol. Ralph Grothmann

Eingereicht am: 11. Oktober 2024

Artur Voskanyan

Thema der Arbeit
Entwicklung eines dateneffizienten Deep Learning Verfahrens zur automatisierten Sendungsverfolgung in der Logistik

Stichworte
Deep Learning, Sendungsverfolgung, Logistik, OCR, Automatisierung

Kurzzusammenfassung
Diese Arbeit befasst sich mit der Entwicklung eines dateneffizienten Deep-Learning-Verfahrens zur Erkennung von Nummern der Versandeinheit, das als Teil einer umfassenderen Lösung zur automatisierten Sendungsverfolgung dient. Ziel ist es, die Erkennungsgenauigkeit bestehender Lösungen mit minimalem Datenaufwand zu übertreffen. Dazu werden aktuelle Methoden analysiert und eine Lösung auf Basis fortschrittlicher Techniken wie Convolutional Neural Networks und Objekterkennungsmodellen vorgestellt. Die abschließende Bewertung zeigt, dass der entwickelte Ansatz in Bezug auf Genauigkeit der bestehenden Lösung überlegen ist.

Artur Voskanyan

Title of the paper
Development of a Data-Efficient Deep Learning Approach for Automated Shipment Tracking in Logistics

Keywords
Deep Learning, Shipment Tracking, Logistics, OCR, Automation

Abstract
This thesis focuses on the development of a data-efficient deep learning method for recognizing Serial Shipping Container Codes, which is part of a more comprehensive solution for automated tracking. The aim is to surpass the recognition accuracy of existing solutions while using minimal data. To achieve this, current methods are analyzed, and a solution based on advanced techniques such as Convolutional Neural Networks and object detection models is presented. The final evaluation demonstrates that the developed approach outperforms the existing solution in terms of accuracy.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Die steigende Komplexität moderner Lieferketten stellt Unternehmen vor erhebliche Herausforderungen im Bereich der Logistik. Insbesondere in einem globalen Markt, in dem Effizienz, Genauigkeit und Flexibilität zunehmend zu entscheidenden Wettbewerbsfaktoren werden, spielt die Automatisierung logistischer Prozesse eine zentrale Rolle bei deren Optimierung. [10]
Die Integration neuer Technologien in Logistiksysteme hat zu einem deutlichen Fortschritt in der Automatisierung und Überwachung logistischer Abläufe geführt. Digitale Verfolgungssysteme, wie etwa Barcode- und RFID-Technologien, haben es ermöglicht, den physischen Fluss von Waren präzise und nahezu in Echtzeit zu überwachen. Trotz dieser Fortschritte gibt es bei der Automatisierung in einigen spezifischen Anwendungsfällen signifikante Hürden. Ein aktuelles Beispiel des in dieser Arbeit zu behandelnden Kundenprojekts zeigt einen solchen Anwendungsfall.
Im Logistiklager des Kunden werden Paletten mit Versandetiketten versehen, die eine eindeutige 18-stellige Nummer der Versandeinheit (NVE)[1] tragen. Diese Nummern müssen identifiziert und für die weitere Verarbeitung erfasst werden. Der Einsatz von automatisierten Barcode-Scannern ist hier nur eingeschränkt umsetzbar, da der Transportprozess der Paletten manuell erfolgt und die Position der Versandetiketten stark variiert. RFID-Technologien bieten zwar eine Alternative zur vollautomatischen Identifizierung, sind jedoch mit hohen Implementierungskosten verbunden und nicht für alle Unternehmen wirtschaftlich tragbar.
In Anbetracht der kontinuierlichen Weiterentwicklung der Kameratechnologien sowie fortschrittlicher Bildverarbeitungsalgorithmen haben sich bildbasierte Zeichenerkennungssysteme (OCR) hierfür als kostengünstige und vielversprechende Lösung erwiesen. Allerdings zeigen Open-Source-OCR-Lösungen in spezifischen Anwendungsfällen Einschränkungen hinsichtlich ihrer Leistungsfähigkeit. Faktoren wie ungünstige Licht-

[1] Im Rahmen dieser Arbeit wird der Begriff “Nummer der Versandeinheit (NVE)” verwendet, um die eindeutige Identifikationsnummer der Paletten zu bezeichnen.

verhältnisse, Bewegungsunschärfe und umgeknickte oder unebene Etiketten können die Erkennungsgenauigkeit erheblich beeinträchtigen. Im vorliegenden Anwendungsfall, bei dem NVE teilweise auf bewegten Paletten während des Transports erfasst werden müssen, zeigen sich die genannten Herausforderungen in ausgeprägter Form.
Daher ist eine maßgeschneiderte OCR-Lösung erforderlich, die an die spezifischen Bedingungen des Anwendungsfalls angepasst werden kann, um eine hohe Erkennungsgenauigkeit und Zuverlässigkeit zu gewährleisten.

1.2 Zielsetzung der Arbeit

Diese Arbeit wird im Rahmen eines Kundenprojekts der Siemens AG[2] durchgeführt und ist Bestandteil einer bereits bestehenden Gesamtlösung zur automatisierten Sendungsverfolgung. Der Fokus liegt primär auf der Optimierung der Genauigkeit bei der Erkennung von NVE.
Die aktuell implementierte Lösung basiert auf EasyOCR, einem Open-Source-System zur optischen Zeichenerkennung, entwickelt von JaidedAI [43]. Im aktuellen System werden Bilder von Versandetiketten mittels einer Industriekamera aufgenommen und nach diversen Vorverarbeitungsschritten einer NVE-Erkennung durch EasyOCR unterzogen. Die Limitationen der bestehenden OCR-Lösung hinsichtlich der Zeichenerkennung unter variierenden Bedingungen erfordern die Entwicklung einer spezifisch angepassten Lösung, die diese Herausforderungen adressiert.
Ziel dieser Arbeit ist die Entwicklung einer Deep-Learning-basierten OCR-Lösung, die eine höhere Erkennungsgenauigkeit als die bestehende Lösung bietet. Da die Datenerhebung in der Praxis oft eingeschränkt ist, wird ein Modell angestrebt, das auch mit begrenztem Datenumfang hohe Erkennungsraten erzielt. Dazu werden verschiedene Deep-Learning-Methoden sowie Vorverarbeitungs- und dateneffiziente Trainingsstrategien evaluiert und kombiniert.

[2] Siemens ist ein führendes Technologieunternehmen, das auf Automatisierung und Digitalisierung spezialisiert ist.

2 Grundlagen und Stand der Technik

Das vorliegende Kapitel gibt einen Überblick über die gängigen Methoden der optischen Zeichenerkennung. Darauf aufbauend werden die für das Verständnis dieser Arbeit wesentlichen Grundlagen neuronaler Netze und Objekterkennungsmodelle vermittelt. Abschließend werden dateneffiziente Verfahren sowie die Funktionsweise von EasyOCR erläutert.

2.1 Methoden zur optischen Zeichenerkennung

OCR ist die Abkürzung für den englischen Begriff Optical Character Recognition (optische Zeichenerkennung). Diese Technologie ermöglicht, Texte oder Zeichen aus Bildern zu extrahieren und in maschinenlesbare Daten umzuwandeln. OCR findet in zahlreichen Bereichen Anwendung, von der Automatisierung von Geschäftsprozessen, über Sofortübersetzungen und Bildsuche, bis hin zur Szenenanalyse und Navigation für Sehbehinderte [29] [37] [41] [44]. Die Ursprünge der OCR-Technologie reichen bis ins späte 19. Jahrhundert. Seitdem hat sie beachtliche Fortschritte gemacht, die sowohl traditionelle Methoden als auch moderne, auf künstlicher Intelligenz basierende Ansätze umfassen [17]. Im Folgenden werden drei verschiedene Methoden der optischen Zeichenerkennung näher betrachtet.

2.1.1 Template-Matching-Verfahren

Das Template-Matching-Verfahren zählt zu den einfachsten und ersten Methoden der Zeichenerkennung. Seine Ursprünge liegen in den frühen Phasen der Computerentwicklung in den 1950er Jahren [30]. Das Verfahren beruht darauf, Licht durch mechanische Schablonen zu leiten, die bestimmten Zeichen entsprechen. Das durch die Schablone hindurchgelassene Licht trifft auf das Papier. In der Folge wird das Licht von einem hinter dem Papier befindlichen Photodetektor erfasst. Wenn die Schablone genau mit dem Zeichen auf dem Papier übereinstimmt, wird das Licht blockiert und erreicht den Detektor nicht, wodurch die Maschine das Zeichen identifizieren kann.

Heutzutage erfolgt der Prozess im Gegensatz zu den früheren opto-mechanischen Methoden vollständig softwarebasiert und unter Verwendung digitaler Bilder. In diesem Kontext existieren diverse Ansätze. Ein einfacher Ansatz besteht darin, Vorlagebilder aus einer Datenbank über das zu untersuchende Textbild gleiten zu lassen und die Ähnlichkeit der Bildbereiche zu berechnen [21]. Eine gängige Methode der Berechnung ist die Summe der quadratischen Differenzen (SSD) der Intensitätswerte der Bildpixel. Hierbei werden die Differenzen zwischen den Intensitätswerten der Vorlage und des Zielbildes quadriert und anschließend aufsummiert. Ein kleinerer SSD-Wert weist auf eine höhere Übereinstimmung hin, da er auf geringere Unterschiede zwischen Vorlage und Zielbild hinweist.
Ein Nachteil dieser Methode ist ihre geringe Flexibilität und hohe Empfindlichkeit gegenüber Variationen. Template Matching erfordert exakte Übereinstimmungen zwischen dem zu erkennenden Objekt und den gespeicherten Vorlagen. Daher ist es anfällig für Unterschiede in Schriftart, Größe, Ausrichtung, Rotation und Helligkeit der Zeichen.

2.1.2 Feature-Extraction-Verfahren

Das Feature-Extraction-Verfahren beschreibt den Prozess, bei dem aus einem digitalen Bild spezifische Merkmale extrahiert werden, die zur Identifizierung der Zeichen dienen. Das Verfahren kann in vier Hauptschritte unterteilt werden:

- Vorverarbeitung,
- Segmentierung,
- Merkmalsextraktion und
- Klassifizierung[1]. [47]

Im ersten Schritt der Vorverarbeitung wird das farbige oder graustufige Bild binarisiert, indem ein geeigneter Schwellenwert für die Intensitätswerte der Pixel festgelegt wird. Die Binarisierung erzeugt ein Bild, das nur aus Schwarz- und Weißwerten besteht, wodurch die Komplexität deutlich reduziert wird. Mittels verschiedener Techniken wie morphologischer Operationen und Filtermethoden wird Rauschen reduziert und Kanten werden verstärkt. Im Anschluss wird das Bild für die weitere Verarbeitung normalisiert, um ein einheitliches Format zu gewährleisten. Im nächsten Schritt werden mithilfe

[1] "Klassifizierung" ist der Prozess, bei dem die aus einem Bild gewonnenen Merkmale analysiert und einem vordefinierten Zeichen, wie einem Buchstaben oder einer Zahl, zugeordnet werden.

von Konturerkennungsmethoden Zeichen oder Symbole aus dem Bild segmentiert. Aus den segmentierten Bildbereichen werden charakteristische Merkmale wie z. B. Umfang, Fläche oder Dichte extrahiert, die dann zu einem Merkmalsvektor[2] zusammengefasst werden. [53]
Für die Klassifizierung kommen verschiedene Algorithmen zur Anwendung, wie z. B. k-Nearest Neighbour (k-NN) oder Support Vector Machines (SVM). Sie nutzen die Merkmalsvektoren, um Zeichen auf Grundlage gemeinsamer Merkmale zuzuordnen. [47]
Zwar ist das Feature-Extraction-Verfahren robuster im Vergleich zu Template-Matching-Verfahren hinsichtlich Bildvariationen. Sein größter Nachteil ist jedoch die Abhängigkeit der Klassifizierungsgenauigkeit von der Auswahl der Merkmale. Ein falscher oder unzureichender Merkmalsauswahlprozess kann zu einer fehlerhaften Klassifizierung führen. Zudem erfordert die Merkmalsauswahl für jeden Anwendungsfall eine individuelle Herangehensweise, die große Erfahrung und Expertise verlangt.

2.1.3 Deep Learning-basierte OCR-Ansätze

Ein wesentlicher Fortschritt in der Entwicklung von OCR-Technologien auf Deep-Learning-Basis wird durch die Arbeit von Yann LeCun im Jahr 1998 erzielt, in der er das LeNet-5-Modell vorstellt. Dieses auf Convolutional Neural Networks (CNN) basierende Netzwerk ist speziell für die Erkennung handgeschriebener Ziffern konzipiert. LeNet-5 gehört zu den ersten CNNs, die erfolgreich in der Bildverarbeitung eingesetzt werden. Durch das Training[3] auf dem bekannten MNIST-Datensatz[4], erreicht das Modell hohe Erkennungsraten und ebnet den Weg für moderne Deep-Learning-Ansätze, die heute in der optischen Zeichenerkennung verwendet werden. Ein exemplarischer Ausschnitt aus dem MNIST-Datensatz ist in der Abbildung 2.1 dargestellt.

[2] Ein “Merkmalsvektor“ ist eine Liste von aus einem Bild extrahierten Eigenschaften, die ein Zeichen eindeutig beschreiben.
[3] Training im Deep Learning bezeichnet den Prozess, bei dem ein neuronales Netzwerk anhand vieler Beispiele lernt, Muster zu erkennen und Vorhersagen zu verbessern
[4] MNIST steht für Modified National Institute of Standards and Technology database, besteht aus 70.000 handgeschriebenen Ziffern (0-9) in 28x28-Pixel-Bildern und wird häufig zur Bewertung der Leistung von Klassifikationsalgorithmen verwendet.

Abbildung 2.1: Beispiele aus dem MNIST-Datensatz [7]

Im Gegensatz zu den Feature-Extraction-Verfahren sind die Deep-Learning-basierten OCR-Ansätze in der Lage, wichtige und oft auch für Menschen schwer greifbare Merkmale automatisch aus den Bildern zu extrahieren. Im Trainingsprozess solcher Ansätze werden in der Regel große Mengen annotierter[5] Textbilder vorgegeben, wodurch das Modell die Beziehungen zwischen visuellen Eingaben und den entsprechenden Klassen, beispielsweise Zeichen, Symbole oder ganze Wörter, erlernt. [26]
Der größte Vorteil von Deep-Learning-basierten OCR-Methoden ist ihre Fähigkeit, Zeichen auch unter stark variierenden Bedingungen, wie niedriger Auflösung, unterschiedlichen Schriftarten, Größen, Ausrichtungen oder komplexen Hintergründen, mit hoher Genauigkeit zu erkennen. Auch zur Erkennung handschriftlicher Texte haben sie sich als äußerst leistungsfähig erwiesen. Allerdings erfordert die Implementierung von Deep-Learning-Ansätzen in der Regel umfangreiche Rechenressourcen, viel Zeit und große Mengen annotierter Daten.

[5] Unter “annotierten“ oder “gelabelten“ Daten versteht man Daten, die mit Informationen versehen sind, die das gewünschte Ergebnis oder die Klasse angeben.

2.2 Grundlagen Künstliche neuronale Netzwerke

Dieser Abschnitt behandelt den Aufbau und die Funktionsweise künstlicher neuronaler Netzwerke. Es beginnt mit einer grundlegenden Erklärung der Neuronen und ihrer Funktion, da diese das Fundament für das Verständnis neuronaler Netzwerke bilden. Darauf aufbauend wird die Struktur mehrschichtiger neuronaler Netzwerke erläutert. Abschließend wird die Funktionsweise von Convolutional Neural Networks beschrieben, die sich aufgrund ihrer speziellen Architektur besonders für die Verarbeitung von Bilddaten eignen.

2.2.1 Aufbau und Funktionsweise eines Perzeptrons

Die Forschung und Nutzung künstlicher neuronaler Netze (KNN) findet bereits im frühen 20. Jahrhundert ihren Anfang. Ihre breite Bekanntheit erlangen sie jedoch erst später, als fortschrittlichere Rechenmaschinen entwickelt werden, deren Leistung ausreicht, um die Anforderungen an die Rechenkapazität künstlicher neuronaler Netze zu erfüllen. [1] KNNs sind in Anlehnung an die Struktur und Funktionsweise biologischer neuronalen Netze des Gehirns konzipiert. Ähnlich wie das biologische neuronale Netz besteht ein KNN aus einer Ansammlung von Neuronen, die auf eine bestimmte Weise miteinander verbunden sind.
Die Aufgabe eines biologischen Neurons besteht im Wesentlichen in der Aufnahme von Informationen in Form elektrischer Signale, deren Verarbeitung und anschließender Weiterleitung. Das Neuron setzt sich im Allgemeinen aus vier Komponenten zusammen, Dendriten, synaptische Verbindungen, Zellkörper und verzweigten Axonen (siehe Abb. 2.2).

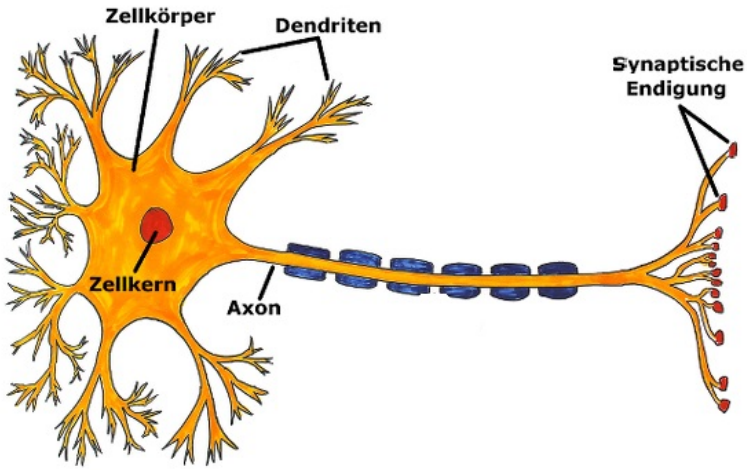


Abbildung 2.2: Aufbau einer Nervenzelle (veränderte Darstellung) [4]

Am Zellkörper befinden sich mehrere Fortsätze, die Dendriten, welche elektrische Signale von anderen, meist mehreren Neuronen über synaptische Verbindungen empfangen. Synaptische Verbindungen entstehen dabei, wenn die synaptischen Endigungen eines Neurons mit den Dendriten eines anderen Neurons verknüpft werden (siehe Abb. 2.2). Diese Signale werden proportional zur Leitfähigkeit der synaptischen Verbindungen aufsummiert. Die Leitfähigkeit kann man auch als Gewichtung der Verbindung betrachten. Je größer die Leitfähigkeit und damit die Gewichtung der synaptischen Verbindung, desto größer ist ihre Bedeutung für das Neuron. Dadurch können die elektrischen Eingangssignale unterschiedlich stark abgeschwächt werden. Erreicht die Summe der gewichteten Signale eine im Zellkörper festgelegte Schwelle, sendet das Neuron das verarbeitete Signal durch das Axon an die mit seinen synaptischen Endigungen verbundenen Neuronen weiter[6]. [14]

Künstliche Neuronen sind mathematische Modelle, die das funktionale Verhalten biologischer Neuronen simulieren. Die Abbildung 2.3 abstrahiert das Modell eines künstlichen Neurons, in Analogie zum biologischen, auf mathematischer Ebene.

[6] Hier wird die Funktionsweise eines Neurons vereinfacht dargestellt, um den Zusammenhang zwischen biologischen und künstlichen Neuronen nachvollziehbar zu machen, da die biologischen Prozesse in Wirklichkeit weitaus komplexer sind.

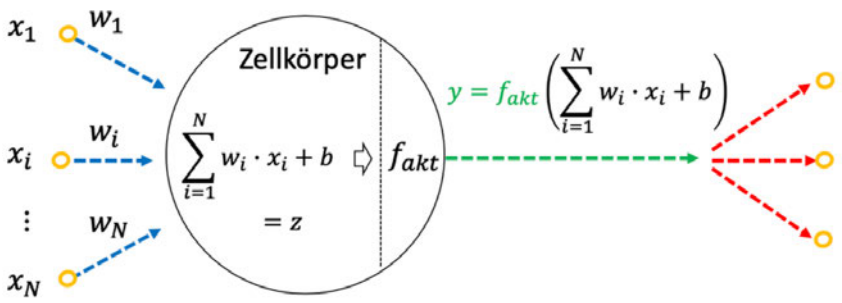


Abbildung 2.3: Aufbau eines künstlichen Neurons [2]

Die durch die Dendriten empfangenen Signale werden hier durch die Eingabewerte x_i dargestellt. Im Gegensatz zum biologischen Modell des Neurons wird ein Bias b eingeführt, der als Offset dient und die Aktivierung des künstlichen Neurons durch Verschiebung beeinflusst [2]. Die Bias-Einheit sorgt zudem dafür, dass eine Aktivierung im Neuron auch dann erfolgt, wenn alle Eingaben Null sind, da sie unabhängig von den gewichteten Eingangssignalen wirkt [31]. Die bereits erwähnten synaptischen Leitfähigkeiten entsprechen den Gewichtswerten w_i wobei die Summe der gewichteten Eingangssignale sowie des Bias durch die Aktivierung z repräsentiert wird. Das Ausgangssignal y ergibt sich aus der Transformation der Aktivierung z durch die Aktivierungsfunktion und stellt die finale Ausgabe dar. Der mathematische Ausdruck dieses Prozesses lautet:

$$y = f_{\text{akt}}\left(\sum_{i=1}^{N} w_i \cdot x_i + b\right) \tag{2.1}$$

Dabei stellt f_{akt} die Aktivierungsfunktion dar. Diese beschreibt, wie das Neuron auf die gewichtete Eingangssignale mit seinem Ausgangssignal reagiert. Bei mehrschichtigen neuronalen Netzen ist es üblich, nichtlineare Aktivierungsfunktionen auszuwählen. Mit rein linearen Funktionen kann das Netz nur lineare Abhängigkeiten zwischen den gewichteten Eingabewerten und den Ausgabewerten erzeugen. Um jedoch komplexe Strukturen in den Daten erkennen zu können, sind nichtlineare Funktionen erforderlich. [31] [2]
Das beschriebene Modell eines künstlichen Neurons bildet die Grundlage für ein einfaches neuronales Netz wie das Perzeptron. Perzeptron besteht aus einer Eingabeschicht und einem Neuron in der Ausgabeschicht. Die Anzahl der Eingabeknoten eines Perzeptrons entspricht dabei der Anzahl der Merkmale, die es verarbeitet. [16]
In der Literatur findet sich eine Vielzahl von Aktivierungsfunktionen. Die einfachste Variante davon ist die Sprungfunktion (siehe Abb. 2.4). Sie wird wie folgt definiert:

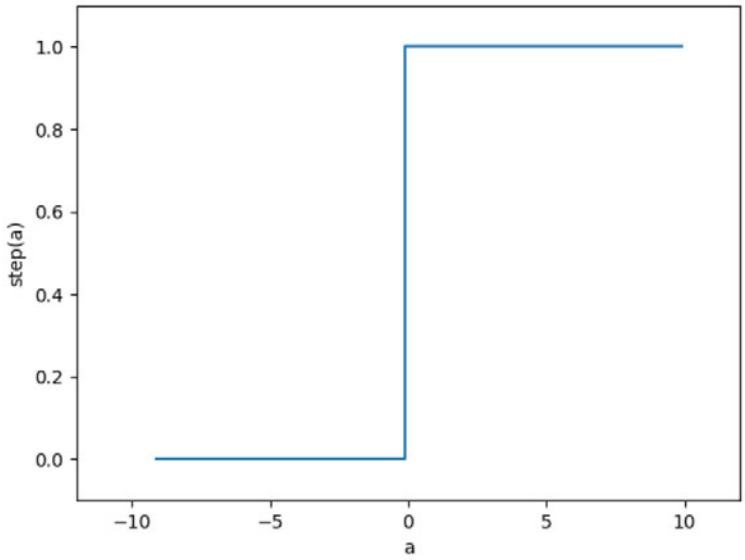


Abbildung 2.4: Sprungfunktion [16]

$$f_{\text{Sprung}}(z) := \begin{cases} 0 & \text{wenn } z < 0 \\ 1 & \text{wenn } z \geq 0 \end{cases} \tag{2.2}$$

Die Sprungfunktion ist eine binäre, diskrete und nicht-differenzierbare Aktivierungsfunktion, bei der das Neuron entweder “0” oder, ab einem bestimmten Schwellwert, “1” ausgibt [14]. Diese Aktivierungsfunktion wird häufig in einfachen Perzeptrons verwendet, die binäre Klassifikationsprobleme lösen, wie die Unterscheidung zwischen zwei Klassen.

2.2.2 Multi-Layer-Perzeptron

Wie bereits erläutert, verbinden sich biologische Neuronen über ihre synaptischen Endigungen mit den Dendriten anderer Neuronen. Sind mehrere Neuronen auf diese Weise miteinander verknüpft, entsteht ein neuronales Netz. Die Modellierung künstlicher neuronaler Netze erfolgt nach einem ähnlichen Prinzip: Hier werden die Neuronen in Schichten (Layers) angeordnet (siehe Abb. 2.5).

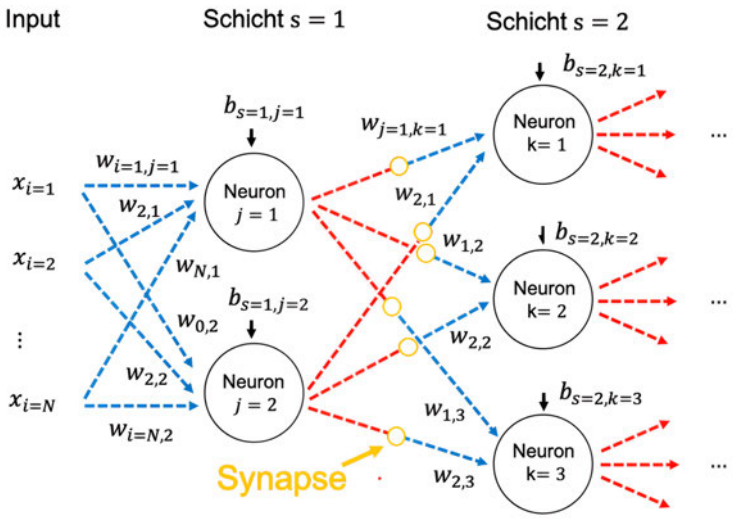


Abbildung 2.5: Multi-Layer-Perzeptron (veränderte Darstellung) [2]

In diesem Beispiel werden die Ausgänge der Neuronen einer Schicht mit allen Eingängen der Neuronen in der nächsten Schicht verknüpft. Die erste Schicht bildet die Eingabeschicht, gefolgt von einer oder mehreren versteckten Schichten und der Ausgabeschicht. In jeder Schicht wird der Eingabewert, ähnlich wie bei einem einzelnen Neuron, mit einem Gewicht $w_{j,k}$ multipliziert. Wenn zwei Schichten mit jeweils j- und k-Neuronen vollständig verbunden sind, entstehen $j \cdot k$ Verbindungen, die in einer Gewichtsmatrix W zusammengefasst werden. Zusätzlich erhält jedes Neuron einer Schicht ein Biasgewicht $b_{s,j}$. [2]
Diese Art von neuronalen Netzen wird Fully Connected Neural Networks (FCNN) oder Multi-Layer-Perzeptron (MLP) genannt. Die Anzahl der Neuronen in einer Schicht kann dabei beliebig sein und ist unabhängig von der Anzahl der Neuronen in anderen Schichten.
Externe Eingangssignale werden an die Eingabeknoten der Eingabeschicht angelegt, während die Neuronen in der Ausgabeschicht die durch die versteckten Schichten verarbeiteten Informationen ausgeben. Die versteckten Schichten in MLPs transformieren die Eingabedaten schrittweise in zunehmend abstraktere Repräsentationen. Jede Schicht lernt, Kombinationen von Merkmalen aus der vorherigen Schicht zu erkennen und zu verarbeiten. Während die vorderen Schichten tendenziell einfachere Muster erfassen, können die hinteren Schichten komplexere, nichtlineare Zusammenhänge in den Daten

modellieren. Diese zunehmende Abstraktion ermöglicht es dem Netzwerk, vielfältige und subtile Muster in den Eingabedaten zu erkennen und zu klassifizieren.

2.2.3 Trainingsmethoden und Optimierungsalgorithmen

Die Grundidee neuronaler Netze beim Lernen besteht darin, sich für eine bestimmte Aufgabe anhand von Trainingsbeispielen anzupassen. Für die Erkennung von Ziffern muss das neuronale Netz zunächst Beispiele dieser Ziffern sehen. Man unterscheidet hierbei zwischen überwachtem und unüberwachtem Lernen. Der wesentliche Unterschied zwischen beiden Lernverfahren liegt darin, dass beim überwachten Lernen annotierte Daten verwendet werden, um Vorhersagen zu treffen (z. B. bei Klassifikationsproblemen), wobei das unüberwachte Lernen ohne annotierte Daten auskommt. Die Anwendung des Letzteren findet meist in der Entdeckung von Mustern oder zur Gruppierung von Daten (z. B. Clustering) statt. [28] [32]
In dieser Arbeit wird hauptsächlich das überwachte Lernen eingesetzt. Dabei werden kleine Änderungen an den Gewichten und dem Bias des neuronalen Netzes vorgenommen und es wird beobachtet, in welche Richtung und in welchem Ausmaß sich die Ausgangsgröße verändert. Da bei dieser Lernmethode der erwartete Ausgangswert bekannt ist, ist das Ziel somit, durch gezielte Beeinflussung der Parameter die Abweichung zwischen den Soll- und Ist-Ausgangswerten zu minimieren. Dafür wird eine Fehlerfunktion definiert, die den Unterschied zwischen der tatsächlichen Ausgabe des Netzes und dem erwarteten Wert quantifiziert.
Eine häufig vorkommende Fehlerfunktion ist die mittlere quadratische Abweichung, auch MSE (mean squared error) genannt. Für einen einfachen Perzeptron und mehrere Trainingsbeispiele wird sie wie folgt definiert:

$$J(w_i) = \frac{1}{2N} \sum_{k=1}^{N} \left(y^{(k)} - \hat{y}^{(k)} \right)^2 \quad \text{mit} \quad k \in \mathbb{N} \tag{2.3}$$

Dabei steht k für das jeweilige Trainingsbeispiel und N für die Anzahl der Trainingsdaten. Die mittlere quadratische Abweichung $J(w_i)$ drückt aus, wie sehr sich die vorhergesagten Werte $\hat{y}^k$ im Durchschnitt von den tatsächlichen Werten y^k in Abhängigkeit von den Gewichten w_i unterscheiden. Der Faktor $\frac{1}{2}$ dient hier zur Vereinfachung bei späteren Berechnungen. Zusammengefasst müssen die Gewichte so eingestellt werden, damit der Gesamtfehler J möglichst klein wird. Dieser Prozess soll automatisch laufen. [25]
Werden exemplarisch zwei Gewichte (w_o und w_1) betrachtet, so spannt deren Abhängig-

keit vom Fehler J die in der Abbildung 2.6 dargestellte zweidimensionale Fehlerlandschaft auf.

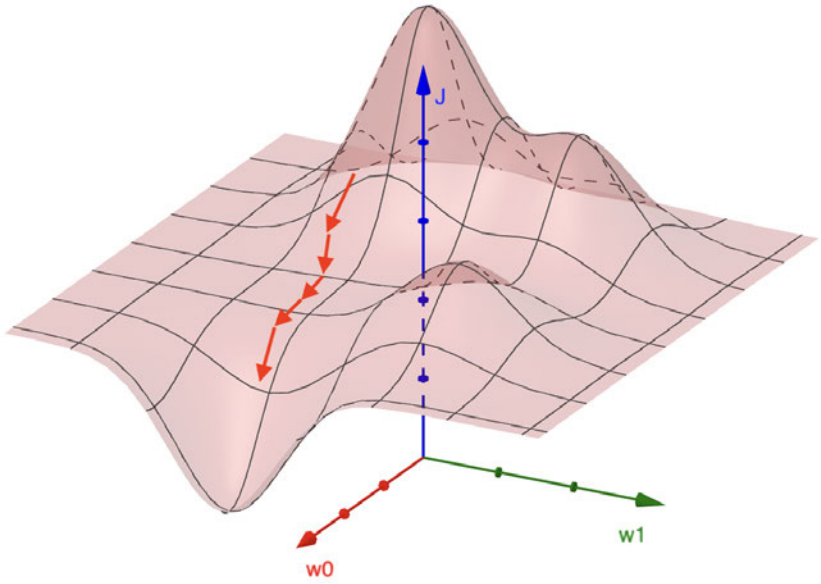


Abbildung 2.6: Abhängigkeit zwischen zwei Gewichten (w_o und w_1) und Fehler J [25]

Der zu minimierende Gesamtfehler J wird dabei entlang der Ordinate dargestellt. Das bedeutet, dass beim Lernprozess der tiefste Punkt (Minimum) dieser Landschaft gesucht wird.
Für diesen Zweck können verschiedene Algorithmen bzw. Optimierungsverfahren verwendet werden. Die Basis vieler Optimierungsverfahren bildet das Gradientenabstiegsverfahren (Gradient Descent). Der Gradient ist im Grunde eine mehrdimensionale Steigung der Fehlerfunktion. Die Idee besteht darin, dem negativen Gradienten zu folgen, indem man durch Anpassung der Parameter in die entgegengesetzte Richtung geht, da dieser die Richtung des stärksten Anstiegs vorgibt. Damit nähert man sich dem Minimum der Fehlerfunktion. Dies illustrieren auch die roten Pfeile in der Abbildung 2.6.
Ein einfacher Ansatz zur Fehlerminimierung, welcher auf dem Gradientenabstiegsverfahren basiert, ist die Delta-Lernregel. Diese passt die Gewichte schrittweise an, indem das alte Gewicht um einen Faktor verringert wird, der proportional zum Fehler sowie der jeweiligen Eingabe ist. Dies ist für einlagige Netzwerke (wie das einfache Perzeptron) geeignet und lässt sich einfach ableiten.

Wird die mittlere quadratische Abweichung aus der Formel 2.3 auf ein Trainingsbeispiel vereinfacht, so resultiert folgende quadratische Fehlerfunktion:

$$J(w_i) = \frac{1}{2}(y - \hat{y})^2 \tag{2.4}$$

wobei $\hat{y}$ Output des Perzeptrons und y Zielwert (Output) ist . Der Gradient dieser Fehlerfunktion in Bezug auf das Gewicht w_i wird wie folgt berechnet:

$$\frac{\partial J}{\partial w_i} = (y - \hat{y}) \cdot \frac{\partial \hat{y}}{\partial w_i} \tag{2.5}$$

Da $\hat{y}$ durch die Aktivierungsfunktion des Perzeptrons erzeugt wird, die auf die gewichtete Summe der Eingaben[7] x_i angewendet wird, gilt es:

$$\frac{\partial \hat{y}}{\partial w_i} = f'(z) \cdot x_i \tag{2.6}$$

wobei $f'(z)$ die Ableitung der Aktivierungsfunktion $f(z)$ ist und $z = \sum w_i \cdot x_i + b$ die gewichtete Summe der Eingaben beschreibt. Somit ergibt sich der vollständige Gradient als:

$$\frac{\partial J}{\partial w_i} = (y - \hat{y}) \cdot f'(z) \cdot x_i \tag{2.7}$$

Das Gewicht w_i wird dann durch die Delta-Regel mit dem Gradientenabstiegsverfahren aktualisiert:

$$w_i^{\text{neu}} = w_i^{\text{alt}} - \eta \cdot \frac{\partial J}{\partial w_i} \tag{2.8}$$

Die Lernrate (η) bestimmt dabei die Größe der Schritte, die das Modell bei jeder Iteration unternimmt, um den Fehler zu minimieren. Eine höhere Lernrate führt zu größeren Anpassungen der Gewichte, was den Lernprozess beschleunigt, aber auch das Risiko birgt, das Optimum zu überschreiten. Eine kleinere Lernrate sorgt für einen langsameren, aber stabileren Lernprozess. Wird als Aktivierungsfunktion die Identität $f(z) = z$ verwendet, so ist ihre Ableitung $f'(z) = 1$ [25]. Damit vereinfacht sich die Delta-Regel auf:

$$w_i^{\text{neu}} = w_i^{\text{alt}} - \eta \cdot (y - \hat{y}) \cdot x_i \tag{2.9}$$

[7] Häufig wird unter dem Begriff „gewichtete Summe der Eingaben“ auch der Bias berücksichtigt, wie es auch in dieser Arbeit der Fall ist.

Durch diese Herleitung ist ersichtlich, dass sowohl die Fehlerfunktion als auch die Aktivierungsfunktion differenzierbar sein müssen, da die Gradientenberechnung ansonsten nicht möglich ist. Da die Sprungfunktion eine unstetige Funktion ist, erfüllt sie diese Anforderung nicht. Ein anderes Beispiel für eine Aktivierungsfunktion ist die Sigmoid-Funktion (siehe Abb. 2.7).

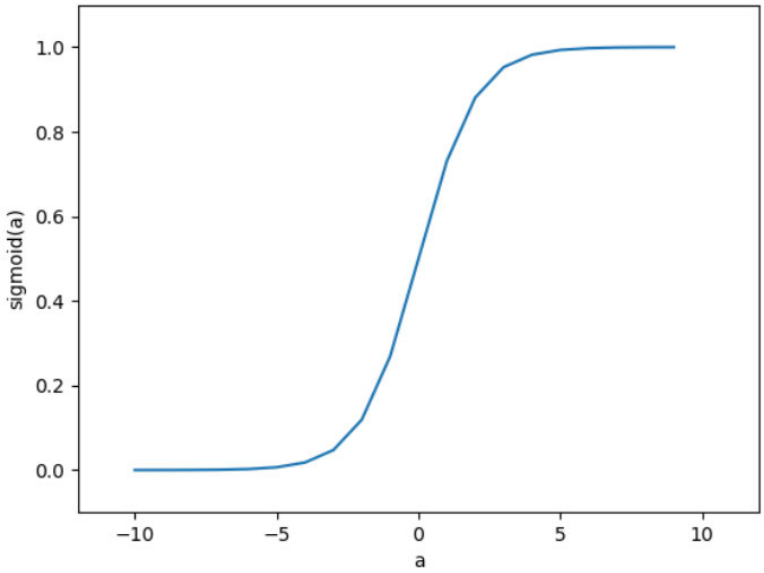


Abbildung 2.7: Sigmoid-Funktion [16]

$$f_{\text{Sigmoid}}(z) = \frac{1}{1 + e^{-z}} \tag{2.10}$$

Die Sigmoid-Funktion ermöglicht eine stetige Ausgabe in Abhängigkeit von den gewichteten Eingangssignalen und approximiert die beiden Aktivitätszustände eines Neurons. Bei schwachen Signalen ist die Ausgabe nahe dem Minimum, bei starken Signalen nahe dem Maximum. Im mittleren Bereich steigt die Kurve steil an. Dadurch löst die Sigmoid-Funktion das Problem der Differenzierbarkeit. Sie hat jedoch den Nachteil, dass bei extremen Signalwerten kaum Lernfortschritte gemacht werden. Dieser Effekt wird auch Vanishing Gradient, oder auf Deutsch das Problem des verschwindenden Gradienten genannt. [16]
Eine häufig verwendete Aktivierungsfunktion, die sowohl das Problem des verschwindenden Gradienten als auch das Problem der Nicht-Differenzierbarkeit löst, ist die

ReLU-Funktion (Rectified Linear Unit). Siehe hierzu auch die Abbildung 2.8 respektive die Formel 2.11.

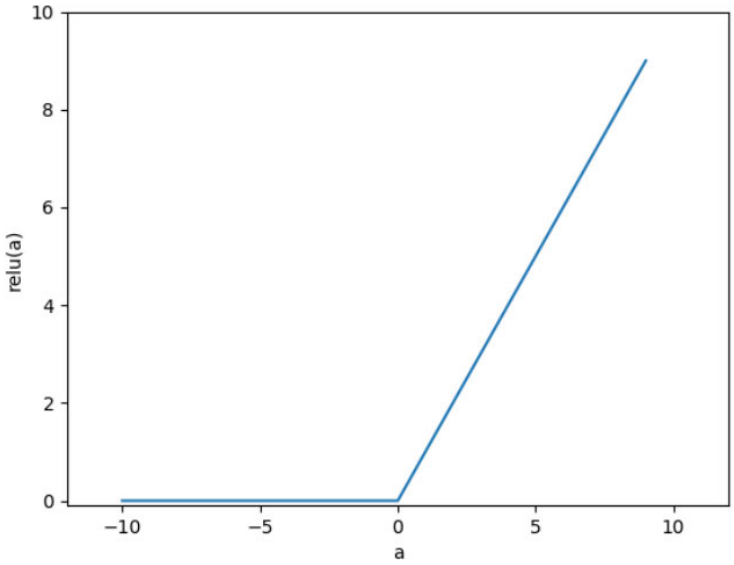


Abbildung 2.8: ReLU-Funktion [16]

$$f_{\text{ReLU}}(z) = \max(0, z) \tag{2.11}$$

Sie gibt den Input direkt zurück, wenn dieser positiv ist, und 0, wenn der Input negativ ist.

Die Anwendung der Delta-Regel ist in mehrschichtigen Netzwerken mit versteckten Schichten nicht möglich, da eine direkte Übertragung des Fehlers auf die Neuronen in den versteckten Schichten nicht realisierbar ist. In diesem Kontext findet das Backpropagation-Verfahren Anwendung. Backpropagation propagiert den Fehler rückwärts durch das Netzwerk, beginnend bei der Ausgabeschicht bis zu den Eingabeschichten. Es berechnet die Gradienten der Fehlerfunktion in Bezug auf die Gewichte in jeder Schicht. Durch die Verwendung des Gradientenabstiegsverfahrens werden diese Gewichte schrittweise angepasst, um den Fehler zu minimieren. Aktivierungsfunktionen wie ReLU unterstützen diesen Prozess besonders gut, da sie die Berechnung der Gradienten durch ihre einfache Ableitung, insbesondere in tiefen Netzwerken, erleichtern. [32]

2.2.4 Convolutional Neural Networks (CNN)

Obwohl klassische neuronale Netzwerke in diversen Anwendungsfeldern bemerkenswerte Erfolge erzielt und ihre Leistungsfähigkeit unter Beweis gestellt haben, stellt ihre Anwendung auf hochdimensionale Problemstellungen, wie beispielsweise die Bilderkennung, eine signifikante Herausforderung dar. Wie bereits im Abschnitt 2.2.1 erwähnt, hängt die Anzahl der Eingabeknoten in der Eingabeschicht eines Perzeptrons von der Anzahl der Merkmale ab. Für neuronale Netze mit mehreren Schichten gilt dasselbe. Problematisch wird es dadurch, dass der Intensitätswert jedes Pixels eines Bildes für ein KNN mindestens ein Eingangsmerkmal darstellt (bei RGB-Bildern sogar drei).
Angenommen wird ein KNN mit einer versteckten Schicht und einem Ausgabeneuron, das ein einfaches Klassifizierungsproblem mit nur zwei Klassen löst. Ein solches KNN hat bei einem Graustufenbild mit 120x240 Pixeln 28.800 Eingabeknoten in der Eingabeschicht. Die Anzahl der Neuronen in der ersten versteckten Schicht wird üblicherweise auf das 2- bis 3-Fache der Eingabeknoten festgelegt, sodass bei 2-facher Anzahl 57.600 Neuronen in der ersten versteckten Schicht resultieren.
Da bei diesem Modell jeder Eingabeknoten mit jedem Neuron der versteckten Schicht und jedes Neuron der versteckten Schicht mit dem Ausgabeneuron verbunden ist, entstehen $(28.800 \times 57.600) + 57.600 = 1.658.937.600$ Verbindungen. Zusätzlich hat jedes Neuron in der versteckten Schicht sowie das Ausgabeneuron einen Bias, was weitere 57.601 Bias-Gewichte ergibt. Insgesamt resultieren damit $1.658.937.600 + 57.601 = 1.658.995.201$ zu trainierende Parameter.
Aus diesem Beispiel wird deutlich, dass sowohl die Berechnung als auch das Training neuronaler Netze für Bilderkennungsaufgaben äußerst aufwändig sind. Ein weiteres Problem besteht darin, dass klassische neuronale Netze die räumliche Struktur der Bilddaten nicht explizit berücksichtigen. Diese räumlichen Informationen sind jedoch maßgeblich, da sie wichtige Merkmale des Bildes enthalten.
Durch Experimente zum Verständnis der visuellen Verarbeitung im Gehirn gelangen David H. Hubel und Torsten Wiesel im Jahr 1959 zu der Erkenntnis, dass bestimmte Neuronen im visuellen Kortex des Gehirns auf spezifische Merkmale reagieren, wie z. B. horizontale oder diagonale Linien. Die visuelle Verarbeitung beginnt auf lokaler Ebene. Das bedeutet, dass das Gehirn zunächst kleine, sich überlappende Bereiche der Netzhaut, auch Wahrnehmungsfelder genannt, verarbeitet. In mehreren aufeinanderfolgenden Schritten werden die Informationen aus diesen kleinen Bereichen kombiniert, um immer komplexere und umfassendere Merkmale zu erkennen, wie zum Beispiel Formen oder Bewegungen. Diese Erkenntnisse haben zur Entwicklung von Convolutional Neural

Networks (CNNs), auf Deutsch Faltungsnetzwerke, beigetragen.
Die Funktion von CNNs basiert auf einem ähnlichen Prinzip wie der visuelle Kortex des Gehirns. Eingaben aus einem lokalen Bereich des Eingabebildes (oder der vorherigen Schicht) werden von einem einzelnen Neuron in der nächsten Schicht verarbeitet. Dadurch wird die Anzahl der zu trainierenden Gewichte stark verringert. [25]
Ein CNN besteht typischerweise aus einer Eingabeschicht und einer Abfolge von Faltungs-, Pooling- und Aktivierungsschichten. Am Abschluss des Netzwerks befinden sich oft eine oder mehrere vollständig verbundene Schichten (Fully Connected Layers), die gewöhnlich von einem Softmax-Klassifikator gefolgt werden (siehe Abb. 2.9).

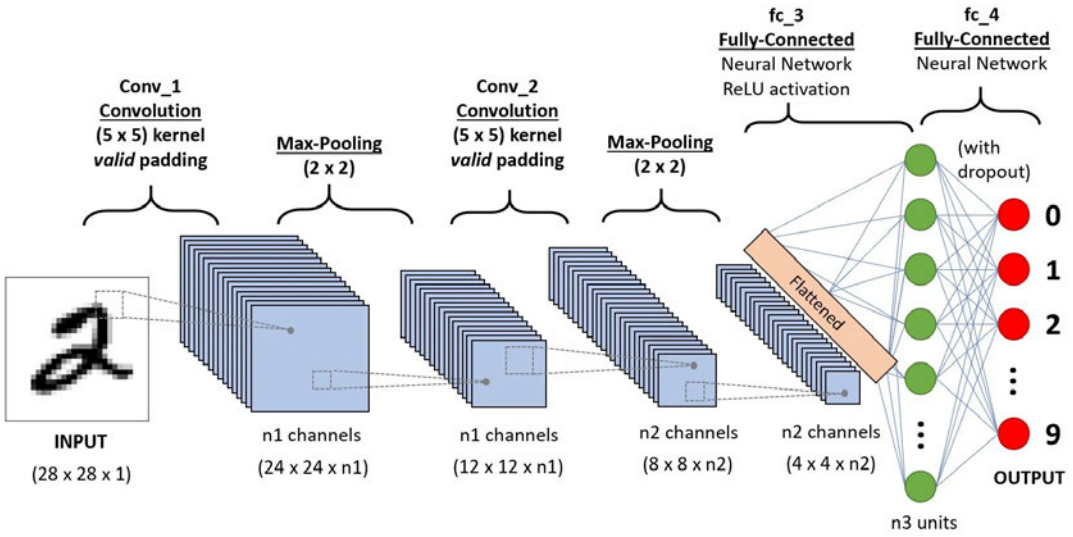


Abbildung 2.9: CNN zur Klassifizierung handgeschriebener Ziffern [42]

Die Faltungsschicht, auch als Convolutional Layer bezeichnet, ist meist die erste Schicht nach der Eingabeschicht. Sie besteht aus einem Stapel von Merkmalskarten, sogenannten Feature Maps. Diese werden durch die Anwendung von Faltungsoperationen auf die Ausgabe der vorherigen Schicht erzeugt. Bei der Faltung wird ein Filterkern, der im Grunde eine Matrix unterschiedlicher Dimensionen darstellt und Gewichte beinhaltet, schrittweise über das Bild geschoben. (siehe Abb. 2.10).

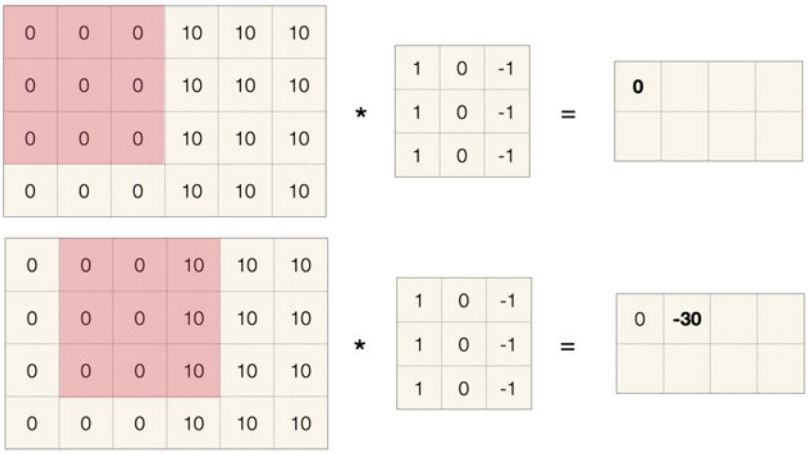


Abbildung 2.10: Faltungsoperation auf ein Bild mit einem Filterkern (1) [25]

Die große Matrix auf der linken Seite der Abbildung stellt dabei das Eingangsbild bzw. die Ausgabe der vorherigen Schicht dar. In der Mitte befindet sich der Filterkern, und auf der rechten Seite die Ausgabematrix. Die Gewichte des Filterkerns werden mit den entsprechenden Pixeln des Bildausschnitts multipliziert. Anschließend werden die Produkte summiert. Dies kann als Analogie zur gewichteten Summe der Eingaben in klassischen neuronalen Netzen gesehen werden, da in beiden Fällen eine gewichtete Kombination der Eingabewerte erfolgt. Die Faltungsformel für den ersten Bildausschnitt aus der oberen Abbildung lautet:

$$I^1 * F = \sum_{i=1}^{3} \sum_{j=1}^{3} I^1_{i,j} F_{i,j} \tag{2.12}$$

Wobei I^1 das erste 3x3-Feld des Bildes und F den Filterkern repräsentiert. Das Ergebnis der Faltung über alle Bereiche des Bildes ist eine Matrix der Dimension 2x4 (siehe Abb. 2.11).

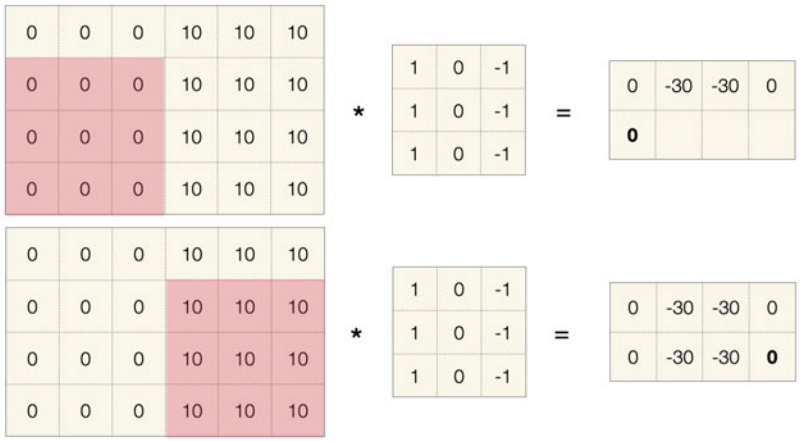


Abbildung 2.11: Faltungsoperation auf ein Bild mit einem Filterkern (2) [25]

Ähnlich wie Neuronengruppen im visuellen Kortex des Gehirns reagieren alle Neuronen eines CNNs, die einen gemeinsamen Filterkern teilen, nur auf bestimmte Merkmale. Dabei ist jedes einzelne Neuron für ein bestimmtes Wahrnehmungsfeld aus der vorherigen Schicht zuständig.
Auf welche Merkmale Neuronengruppen reagieren, hängt von den Gewichten sowie dem Bias-Term des Filterkerns ab, die, ähnlich wie in mehrschichtigen neuronalen Netzwerken, durch Training angepasst werden. Das bedeutet, dass das CNN während des Trainings durch Anpassung der Filterparameter selbst entscheidet, welche Merkmale für die Erkennung relevant sind. Beispielsweise ist der in Abbildung 2.10 eingesetzte Filter für die Erkennung vertikaler Kanten zuständig. [25]
Um die Dimension von Feature Maps zu reduzieren und somit den Rechenaufwand zu minimieren, wird Pooling eingesetzt. Analog zur Faltung wird auch hier ein Filter über die vorherige Schicht geschoben. Im Gegensatz zur Faltung werden hier auf den Wahrnehmungsfeldern jedoch einfache Operationen ausgeführt, wie z. B. die Bestimmung des Maximums (Max Pooling) oder die Berechnung des Durchschnitts (Average Pooling) (siehe Abb. 2.12).

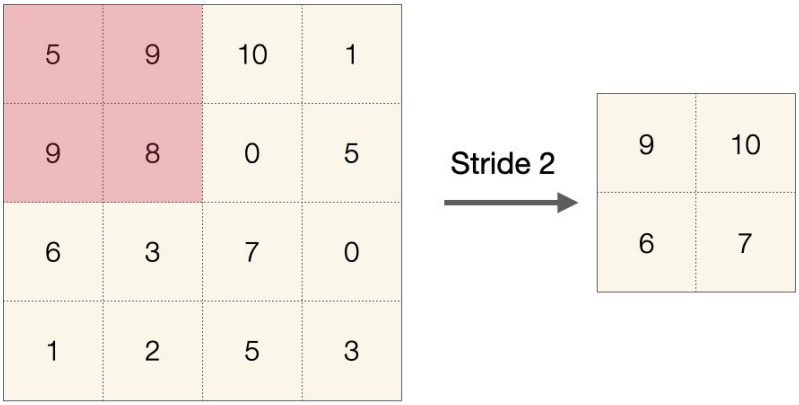


Abbildung 2.12: Anwendung des Max Pooling-Filters auf eine Eingabematrix mit Stride=2. Quelle: Veränderte Darstellung [25]

Obwohl die Dimension der Feature Maps durch Pooling reduziert wird, bleiben die für die Erkennung relevanten Informationen erhalten. Darüber hinaus verbessert Pooling die Invarianz gegenüber Verschiebungen und kleineren Veränderungen in der Position der Objekte, indem es die wesentlichen Merkmale betont und unwichtige Details herausfiltert. Die so vereinfachten Feature Maps werden als Stapel zum Pooling Layer zusammengeführt.
Für das oben beschriebene Beispiel der Faltung (siehe Abb. 2.10) wird eine stark vereinfachte Formel 2.2.4 verwendet. Betrachtet man die formale Definition der Faltungsschicht, so werden zusätzlich die Kanäle der Eingabe, die Filtergrößen und der Bias-Term berücksichtigt. Diese vollständige Formel ermöglicht die Berechnung der Aktivierungen im CNN, indem die gewichteten Summen der Filteroperationen über alle Eingabekanäle und Bereiche hinweg durchgeführt werden:

$$z_{i,j,m}^{(l)} = \sum_{c=1}^{n_c} \sum_{h=0}^{f-1} \sum_{k=0}^{f-1} a_{i+h,j+k,c}^{(l-1)} w_{h,k,c}^{(l),<m>} + b^{(l),<m>} \tag{2.13}$$

Dabei bedeutet:

- $z_{i,j,m}^{(l)}$: Die Aktivierung des Neurons an der Position (i, j) im Feature Map m der l-ten Schicht.
- n_c: Die Anzahl der Kanäle in der vorherigen Schicht (z.B. RGB-Kanäle).
- f: Die Größe des Filterkerns (z.B. 3x3).

- $a^{(l-1)}_{i+h,j+k,c}$: Die Aktivierung an der Position $(i+h, j+k)$ im Kanal c der vorherigen Schicht $l-1$.
- $w^{(l),<m>}_{h,k,c}$: Das Gewicht des Filterkerns m in der aktuellen Schicht l bei Position (h,k) und Kanal c.
- $b^{(l),<m>}$: Der Bias-Term für das Feature Map m in der Schicht l.

Dazu kommt noch die Aktivierungsfunktion, welche auf die Aktivierung $z^{(l)}_{i,j,m}$ angewendet wird, um die Ausgabe des Neurons $a^{(l)}_{i,j,m}$ an der Position (i,j) im Feature Map m der l-ten Schicht zu berechnen:

$$a^{(l)}_{i,j,m} = g(z^{(l)}_{i,j,m}) \tag{2.14}$$

Als Aktivierungsfunktion wird in CNNs meist die ReLU-Funktion verwendet. Wie bereits im letzten Abschnitt erwähnt, setzt die ReLU-Funktion alle negativen Werte auf 0 und lässt positive Werte unverändert. Dadurch werden die Berechnungen im Vergleich zu anderen nichtlinearen Funktionen vereinfacht.
In CNNs folgen den Faltungs- und Pooling-Schichten häufig die Fully Connected Layers (FCL). Diese Schichten verarbeiten die zuvor extrahierten Merkmale weiter, indem sie die mehrdimensionalen Ausgaben der vorherigen Schichten zu einem eindimensionalen Vektor abflachen, der als Eingabe für die Neuronen der Fully Connected Layers dient. Diese Schichten kombinieren die Merkmale, um abstrakte Repräsentationen zu erzeugen, die schließlich durch einen Klassifikator, oft mit einer Softmax-Aktivierungsfunktion, zur endgültigen Klassifizierung verwendet werden.

2.3 Funktionsprinzip von Objekterkennungsmodellen

Die Objekterkennung stellt ein wichtiges Teilgebiet der Deep-Learning-basierten Bildverarbeitung dar und umfasst die Lokalisierung und Erkennung von Objekten in digitalen Bildern. Zeichenerkennung kann dabei als eine spezielle Form der Objekterkennung verstanden werden, da sie sich auf die Lokalisierung und Erkennung einzelner Zeichen oder Symbole innerhalb eines Bildes konzentriert. Der folgende Abschnitt stellt verschiedene Ansätze der Objekterkennung vor, die die Grundlage für den konzeptionellen Teil bilden.

2.3.1 Region-based Convolutional Neural Networks (R-CNN)

Im Jahr 2014 präsentieren Ross Girshick et al. das Objekterkennungsmodell Region-based Convolutional Neural Network (R-CNN) (siehe Abb. 2.13). Das Modell beruht auf der Idee, das Eingangsbild mithilfe des sogenannten Selective-Search-Algorithmus in etwa 2000 Regionen zu unterteilen, in denen sich potenzielle Objekte befinden können. Diese Bereiche werden auch als Region Proposals (Regionsvorschläge) bezeichnet. Für jedes Region Proposal extrahiert ein Convolutional Neural Network (CNN) Merkmale, die zu Merkmalsvektoren zusammengefasst werden. Diese Vektoren dienen dann einer Support Vector Machine (SVM) zur Objektklassifizierung.

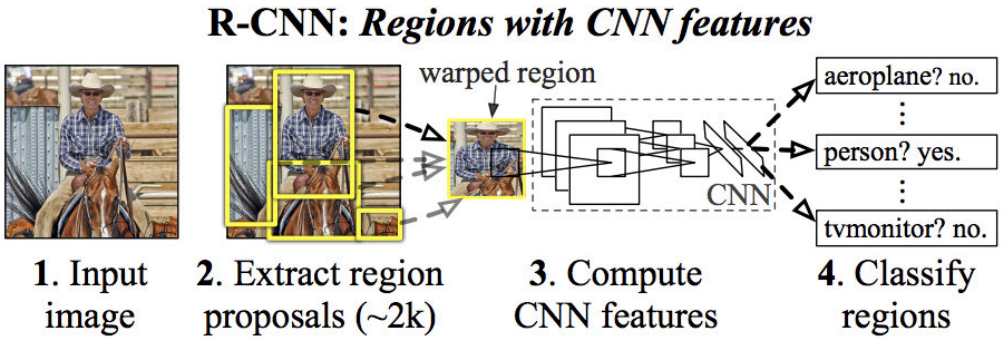


Abbildung 2.13: Die R-CNN-Architektur [19]

Zur Verbesserung der Genauigkeit der Objektdetektion wird eine Bounding-Box-Regression durchgeführt, die die Koordinaten der vorgeschlagenen Regionen anpasst, um die Grenzen der erkannten Objekte präziser zu bestimmen. Diese angepassten Regionen werden als Bounding Boxes bezeichnet. Bounding Boxes sind rechteckige Rahmen, die die Position und Größe eines erkannten Objekts im Bild festlegen. R-CNN zeigt trotz guter Ergebnisse bei der Objekterkennung wesentliche Nachteile. Der Hauptkritikpunkt ist die hohe Anzahl von Region Proposals, die durch Selective Search erzeugt werden. Dies führt zu einem erheblichen Rechenaufwand und verlangsamt den Erkennungsprozess deutlich, mit einer durchschnittlichen Verarbeitungszeit von etwa 50 Sekunden pro Bild. Ein weiterer Nachteil ist das mehrstufige Training der Pipeline. CNN, SVM und der Selective-Search-Algorithmus müssen separat und nacheinander trainiert werden, was einen großen Trainingsaufwand bedeutet. Diese Faktoren begrenzen die praktische Anwendbarkeit von R-CNN, insbesondere in Szenarien, die eine schnelle Verarbeitung

oder ressourceneffiziente Implementierung erfordern.[19]
Eine optimierte Version des R-CNN, die sogenannte Fast R-CNN, wird im Jahr 2015 von Ross Girshick vorgestellt (siehe Abb. 2.14).

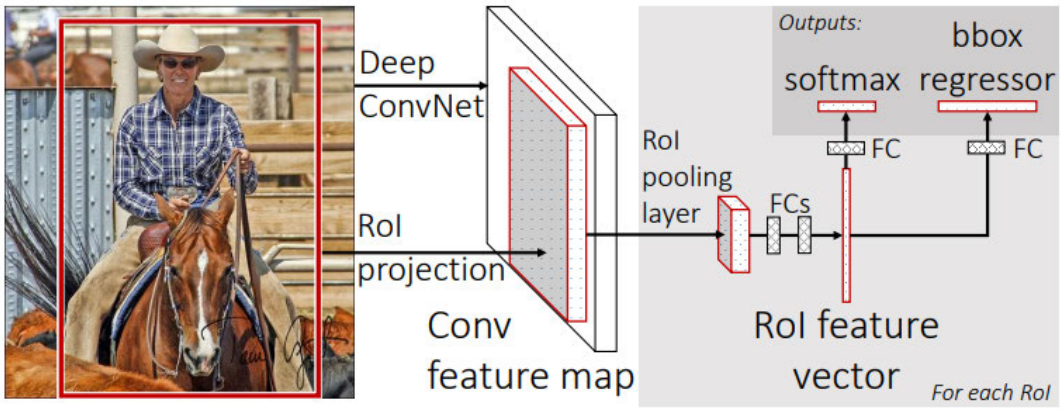


Abbildung 2.14: Fast R-CNN-Architektur [18]

Beim neuen Ansatz werden zum Beginn durch CNN einmalig Feature Maps aus dem gesamten Eingangsbild erzeugt. Gleichzeitig wird, ähnlich wie bei dem R-CNN, der Selective-Search-Algorithmus auf dem Eingangsbild durchgeführt, um Region Proposals zu erzeugen. Im Gegensatz zum R-CNN werden hier die resultierenden Region Proposals jedoch auf Future Maps projiziert. Dies bedeutet, dass für jede Region Proposal die entsprechenden Merkmalsregionen auf den Feature Maps ausgewählt werden. Fortan werden diese Regionen als Regions of Interest (RoI) bezeichnet.
Um die unterschiedlichen Größen der RoIs zu bewältigen und sie in ein einheitliches Format für die weitere Verarbeitung zu bringen, verwendet Fast R-CNN eine spezielle Schicht namens RoI-Pooling-Layer. Dieser Layer teilt jede RoI in ein festes Gitter von Zellen auf, beispielsweise 7x7, unabhängig von der ursprünglichen Größe der RoI. Innerhalb jeder dieser Zellen wird eine Pooling-Operation durchgeführt, meist Max-Pooling, um die dominanten Merkmalswerte zu extrahieren und damit nachfolgende Verarbeitungen zu vereinfachen. Das Ergebnis ist ein fixierter Merkmalsvektor mit einheitlicher Größe für jede RoI. Die so gewonnenen Merkmalsvektoren werden anschließend durch vollständig verbundene Schichten (Fully Connected Layers) zur Klassifizierung und Lokalisierung weitergeleitet.
Schließlich resultieren zwei Ausgaben. Hierbei handelt es sich zum einen um Softmax-

Wahrscheinlichkeiten, die die Zugehörigkeit zu einer der Objektklassen bestimmen, einschließlich einer „Hintergrund“-Klasse. Zum anderen handelt es sich um vier Werte pro Klasse, die die Position und Größe der Bounding-Box für die jeweilige Klasse präzisieren. Daraus resultiert eine signifikante Verbesserung der Erkennungsgenauigkeit und eine erhebliche Verkürzung der Verarbeitungszeit. Zudem entfällt das mehrstufige Training, da das Modell in einem einzigen Schritt trainiert werden kann, was den Rechenaufwand zusätzlich verringert. [18]
Trotz der Verbesserungen bleibt der Selective-Search-Algorithmus weiterhin ein maßgeblicher Nachteil des Fast R-CNNs, da nach wie vor eine eine große Anzahl von Proposals erzeugt wird. Darüber hinaus sind sie nicht in das neuronale Netzwerk integriert, was bedeutet, dass die Generierung der Regionsvorschläge nicht von den während des Trainings gelernten Merkmalen profitieren kann. Aus diesen Gründen erfolgt im Jahr 2016 eine Weiterentwicklung des Modells. Der Name des überarbeiteten Modells lautet Faster R-CNN (siehe Abb. 2.15).

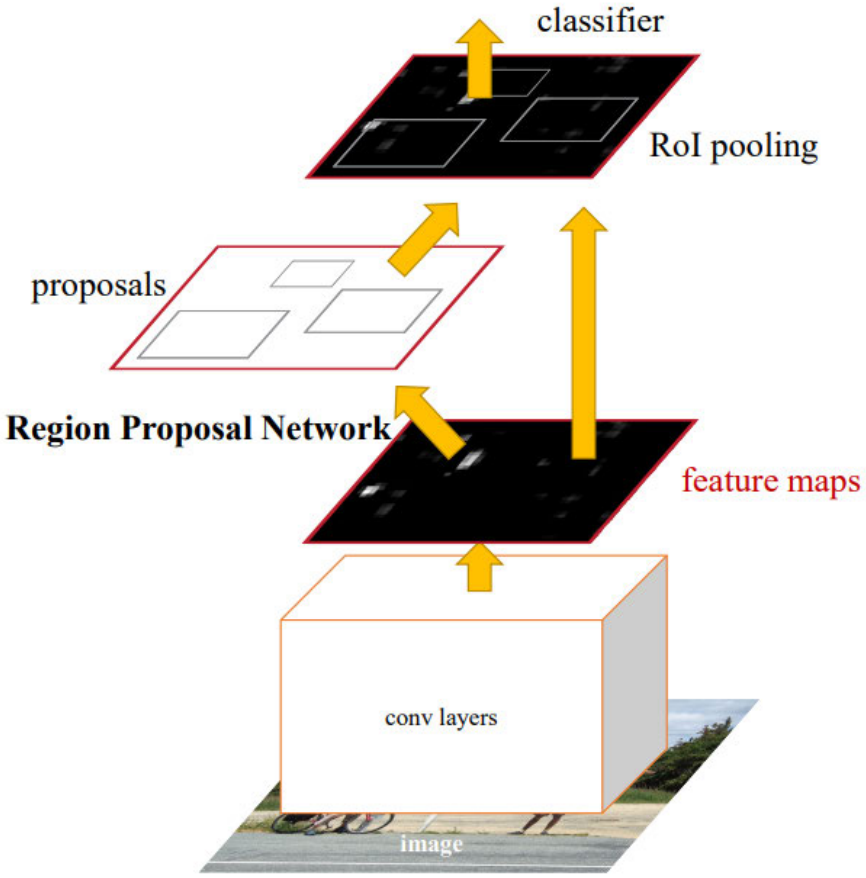


Abbildung 2.15: Faster R-CNN-Architektur [39]

Bei Faster R-CNN wird die Aufgabe des Selective Search Algorithmus durch das sogenannte Region Proposal Network (RPN) übernommen. In ähnlicher Weise wie bei Fast R-CNN werden hier Feature Maps aus dem Eingangsbild erzeugt. RPN erstellt dabei Ankerpunkte, die gleichmäßig über die gesamte Feature Maps verteilt sind. An jedem dieser Punkte werden mehrere Ankerboxen mit vordefinierten Skalen und Seitenverhältnissen gesetzt, die als Referenzrahmen für potenzielle Objekte dienen.
Das Region Proposal Network (RPN) nutzt diese Ankerboxen und die zugehörigen Merkmalsinformationen als Eingabe. Für jede Ankerbox berechnet es eine Objektwahrscheinlichkeit und führt eine Bounding-Box-Regression durch, um die Position und Größe der Ankerbox an die tatsächliche Objektform anzupassen. Durch diese Prozesse werden die Ankerboxen verfeinert, um präzisere RoIs zu erstellen. [39]

Die restlichen Schritte werden analog zu Fast R-CNN durchgeführt. Durch das RPN wird die Objekterkennung deutlich schneller, da die Generierung der Regionsvorschläge direkt im Netzwerk integriert wird. Zudem kann das RPN im Gegensatz zu Selective Search die Merkmalsinformationen der Feature Maps beim Training nutzen, um die Regionsvorschläge dynamisch zu optimieren und präzisere RoIs zu erzeugen.
Die Tabelle 2.1 verdeutlicht, dass mit jeder Version sowohl die Genauigkeit, gemessen in mAP[8], als auch die Verarbeitungsgeschwindigkeit erheblich verbessert werden

Tabelle 2.1: Vergleich der Erkennungsgenauigkeit und -geschwindigkeit zwischen R-CNN, Fast R-CNN und Faster R-CNN bei Training mit VOC 2012. [19] [18] [39]

Modell	**Geschwindigkeit**	**mAP**
R-CNN	49 s	53,3 %
Fast R-CNN	2.32 s	65,7 %
Faster R-CNN	0.2 s	67 %

Faster R-CNN bietet dabei die beste Kombination aus beiden Aspekten und übertrifft seine Vorgänger deutlich.

[8] Die mAP (mean Average Precision) ist eine Kennzahl in der Objekterkennung, die den Durchschnitt der Genauigkeiten über alle erkannten Objekte und deren verschiedenen Schwellenwerte hinweg berechnet. Sie gibt an, wie gut ein Modell Objekte korrekt identifiziert und lokalisiert.

2.3.2 You Only Look Once (YOLO)

You Only Look Once (YOLO) ist ein weiterer Ansatz, der sich durch seine schnelle Erkennungsgeschwindigkeit auszeichnet. Dabei behandelt es die Objekterkennung als ein Regressionsproblem und nicht als ein Klassifikationsproblem.
YOLOv1 war die allererste Version und wurde im Jahr 2016 von Joseph Redmon et al. veröffentlicht. Im Gegensatz zu Methoden wie R-CNN, die potenzielle Objekte durch Region Proposals ermitteln, bevor eine Klassifizierung stattfindet, betrachtet YOLO das gesamte Bild in einem Durchgang und nutzt die darin enthaltenen Merkmale, um Bounding Boxes und Klassifikationen gleichzeitig zu erstellen. Dadurch wird der gesamte Prozess erheblich beschleunigt.
YOLOv1 besteht aus 24 Convolutional Layers, gefolgt von zwei vollverbundenen Schichten. Abwechselnd eingesetzte 1×1-Convolutional Layers komprimieren dabei den Feature-Raum und reduzieren den Rechenaufwand. Das finale Ausgabeformat des Netzwerks ist ein $7 \times 7 \times 30$-Tensor, der die Vorhersagen enthält (siehe Abb. 2.16).

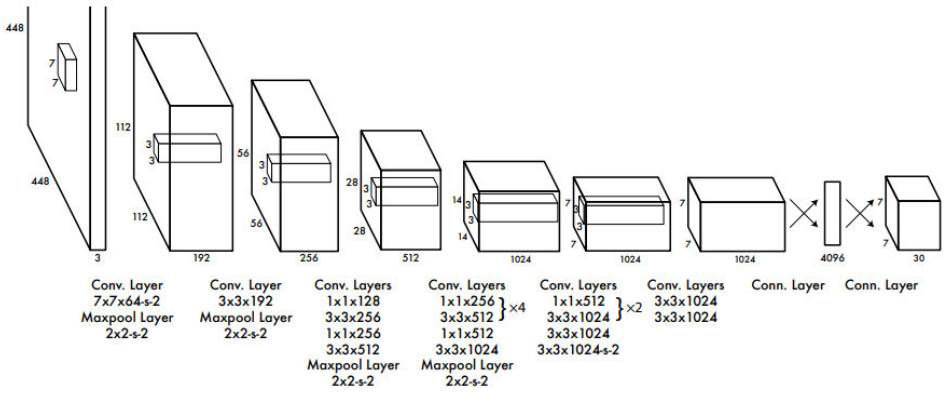


Abbildung 2.16: Netzwerkarchitektur von YOLOv1 [38]

Das Modell teilt das Eingabebild in S x S-Raster auf (siehe Abb. 2.17).

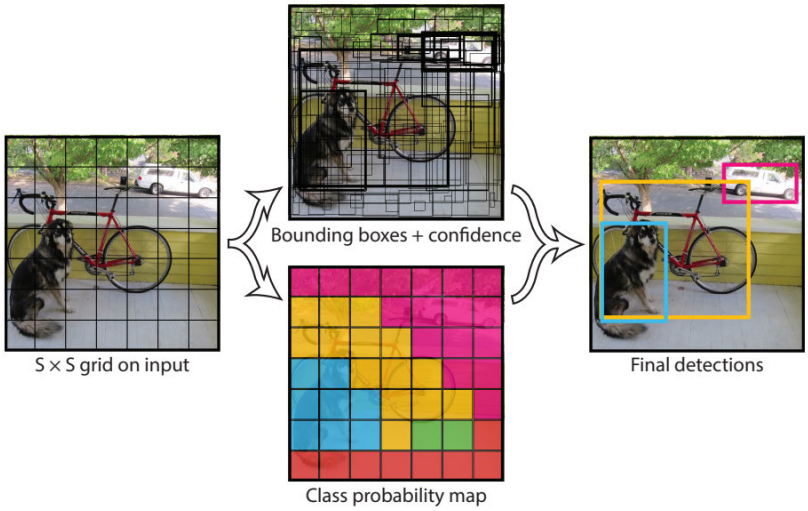


Abbildung 2.17: Objekterkennungsprozess von YOLO [38]

Jedes Rasterquadrat ist dafür verantwortlich, Objekte zu erkennen, deren Mittelpunkt in dieses Quadrat fällt. Diese lokale Verantwortung erleichtert die Zuordnung von Objekten zu bestimmten Bereichen im Bild. Für jedes Rasterquadrat werden dabei mehrere Bounding Boxes (für die Erkennung von mehr als ein Objekt) mit Confidence Scores erstellt. Confidence Score gibt an, wie sicher sich das Modell ist, dass die entsprechende Bounding Box ein Objekt enthält und wie genau seine Begrenzung ist. Sie wird wie folgt definiert:

$$Pr(\text{Object}) \cdot \text{IOU}_{\text{pred}}^{\text{truth}} \tag{2.15}$$

Falls kein Objekt in der Bounding Box vorhanden ist, wird der Confidence Score auf null gesetzt. Andernfalls entspricht er dem Intersection over Union (IoU) zwischen der vorhergesagten Bounding Box und der Ground Truth (Tatsächliche Position des Objektes).
IoU ist ein Maß, das die Überlappung zwischen zwei Bounding Boxes quantifiziert. Es wird berechnet, indem der Schnitt (die überlappende Fläche) der vorhergesagten Bounding Box und der Ground Truth durch deren Vereinigung (die gesamte Fläche beider Boxen) geteilt wird (siehe Abb. 2.18). Der Wert liegt zwischen null und eins, wobei eins für eine perfekte Übereinstimmung steht.

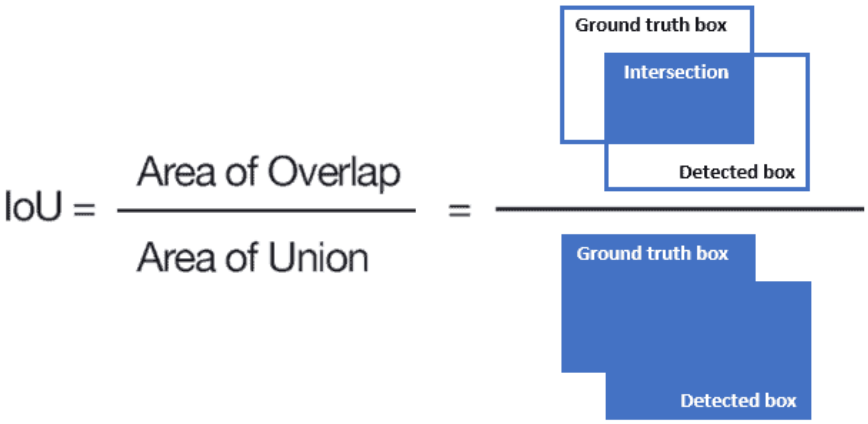


Abbildung 2.18: Intersection over Union (IoU) [9]

Gleichzeitig zu den Bounding Boxes sagt das Modell für jedes Rasterquadrat die bedingten Klassenwahrscheinlichkeiten für jede der möglichen Klassen voraus.

$$Pr(\text{Klasse}_i|\text{Objekt}) \tag{2.16}$$

Diese Wahrscheinlichkeiten werden unter der Annahme berechnet, dass sich in der jeweiligen Rasterzelle tatsächlich ein Objekt befindet, und geben an, zu welcher Klasse das erkannte Objekt wahrscheinlich gehört. Um den klassenspezifischen Confidence Score für jede Bounding Box zu berechnen, wird der entsprechende Confidence Score mit den bedingten Klassenwahrscheinlichkeiten multipliziert.

$$Pr(\text{Class}_i|\text{Object}) \cdot Pr(\text{Object}) \cdot \text{IOU}_{\text{pred}}^{\text{truth}} = Pr(\text{Class}_i) \cdot \text{IOU}_{\text{pred}}^{\text{truth}} \tag{2.17}$$

Das Ergebnis ist die Wahrscheinlichkeit, dass sich ein Objekt einer bestimmten Klasse in der jeweiligen Bounding Box befindet. Bounding Boxes mit niedrigen Konfidenzwerten werden verworfen, um die Anzahl der falsch-positiven Ergebnisse zu verringern. Da es vorkommen kann, dass mehrere Bounding Boxes dasselbe Objekt erkennen, wird das Verfahren der Non-Maximum Suppression (NMS) angewendet, um überlappende Bounding Boxes zu entfernen. Dabei bleibt die Box mit dem höchsten Confidence Score bestehen, während andere Boxen, die eine starke Überlappung mit dieser aufweisen, verworfen werden. [38]
Trotz seiner Geschwindigkeit bringt YOLO einige Herausforderungen mit sich. Aufgrund der festen Rasteraufteilung kann es Schwierigkeiten haben, kleine Objekte genau zu

lokalisieren. Neuere Versionen von YOLO, wie YOLOv3 und YOLOv4, haben jedoch erhebliche Verbesserungen eingeführt, um diese Herausforderungen zu adressieren, einschließlich besserer Handhabung kleinerer Objekte und gesteigerter Genauigkeit bei der Erkennung in komplexen Szenen. Mittlerweile existieren auch fortschrittlichere Versionen wie YOLOv8 und YOLOv9, die weitere Optimierungen und eine noch höhere Erkennungsgenauigkeit bieten.

2.4 Dateneffiziente Ansätze im Deep Learning

In der Regel erfordert das Training von Deep Learning-Modellen große Mengen an annotierten Daten. Die Beschaffung solcher Daten ist in vielen Anwendungsbereichen jedoch zeitaufwendig und kostspielig. Daher gewinnen dateneffiziente Ansätze im Deep Learning zunehmend an Bedeutung. Diese Ansätze zielen darauf ab, die Menge an erforderlichen Trainingsdaten zu reduzieren, indem sie bereits vorhandene Modelle und Daten effektiver nutzen. In diesem Abschnitt werden drei wesentliche dateneffiziente Techniken vorgestellt: Transfer Learning, Few-shot Learning und Data Augmentation. [55]

2.4.1 Transfer Learning

Die Kernidee von Transfer Learning ist die Übertragung des bereits erworbenen Wissens aus einem Bereich (Source Domain) auf eine neue Aufgabe (Target Domain). Dieser Ansatz wird auch von Menschen und Tieren verwendet, da sie Erfahrungen und Kenntnisse aus einem Bereich oft auf andere, ähnliche Bereiche übertragen. Zum Beispiel hilft das Erlernen eines Instruments, ein anderes schneller zu erlernen. Wenn jedoch die Domänen nur wenig gemeinsam haben, kann der Wissenstransfer ineffektiv sein oder sogar zu sogenannten negativen Transfer führen. Das Erlernen des Fahrradfahrens hilft nicht dabei, Klavierspielen schneller zu erlernen. [55]
Für Deep Learning bedeutet das, anstatt ein Modell von Grund auf neu zu trainieren, wird ein vortrainiertes Modell verwendet. Grundsätzlich lassen sich zwei Implementierungsansätze unterscheiden. Der erste Ansatz sieht vor, ein vortrainiertes Modell direkt auf die neue Aufgabenstellung anzuwenden, ohne es erneut zu trainieren. Diese Vorgehensweise ist sinnvoll, wenn die Klassen der neuen Aufgabenstellung eine Teilmenge der Klassen des vortrainierten Modells darstellen. Der zweite Ansatz besteht darin, die letzten Schichten des Modells neu zu trainieren, während die Gewichte der frühen Schichten entweder eingefroren oder nur teilweise angepasst werden. Dies ist besonders

dann vorteilhaft, wenn die Klassen der neuen Aufgabenstellung denen des vortrainierten Modells ähnlich sind, sodass die allgemeinen Merkmale, die das Modell bereits gelernt hat, weiterhin genutzt werden können. [40]

2.4.2 Few-shot Learning

Few-shot Learning ist eine weitere dateneffiziente Methodik. Das Ziel von Few-shot Learning besteht darin, auf neue, während des Trainings nicht gesehene Klassen zu generalisieren, obwohl nur eine geringe Anzahl von Beispielen pro neuer Klasse verfügbar ist.
Im Gegensatz zu klassischen Deep-Learning-Ansätzen teilt man beim Few-shot Learning die Daten in ein gelabeltes Support-Set und ein Query-Set. Das Support-Set fungiert dabei als eine Art Katalog. Ziel ist es, für jedes Beispiel im Query-Set ein Label vorherzusagen, das aus den Labels des Support-Sets stammt (siehe Abb. 2.19).

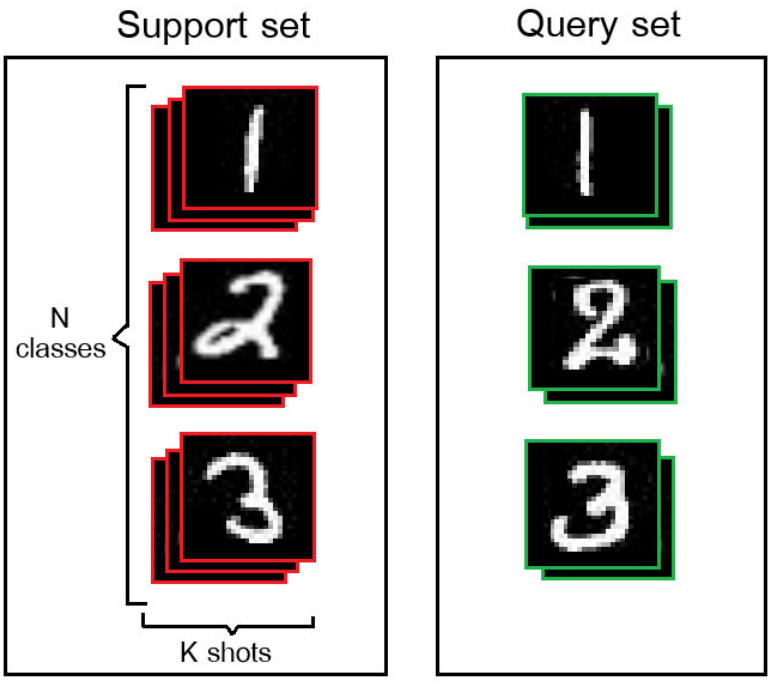


Abbildung 2.19: Support und Query sets in Few-shot Learning (eigene Darstellung)

Das Beispiel in der Abbildung zeigt eine sogenannte 3-way 3-shot Klassifikationsaufgabe. “3-way” bedeutet die Anzahl der Klassen und “3-shot” die Anzahl der Beispiele pro

Klasse. Beide Angaben beziehen sich auf das Support-Set. Somit muss das Modell in der Lage sein, während des Trainings nicht gesehene Klassen anhand eines kleinen Katalogs zuzuordnen.
Es gibt unterschiedliche Ansätze im Few-shot Learning. Die meisten davon basieren auf metrischen Methoden. Sie projizieren sowohl das Support-Set als auch das Query-Set in einen Merkmalsraum und klassifizieren die Daten aus dem Query-Set, indem sie diese mit dem Support-Set vergleichen.
Ein häufig verwendeter und leicht zu implementierender Ansatz ist das Prototypical Network. Ein auf CNN basierendes vortrainiertes Modell wie z. B. ResNet18 oder VGG16 dient dabei als Backbone für das Prototypical Network. Die Aufgabe des Backbones ist die Merkmalextraktion der Daten aus dem Query- und Support-Set in Form von Merkmalsvektoren. Diese Merkmalsvektoren werden dann in einen Merkmalsraum projiziert (siehe Abb. 2.20). In diesem Raum liegen Merkmalsvektoren derselben Klasse näher beieinander, während Merkmalsvektoren unterschiedlicher Klassen weiter voneinander entfernt sind (in der Abbildung 2.20 farblich gekennzeichnet).

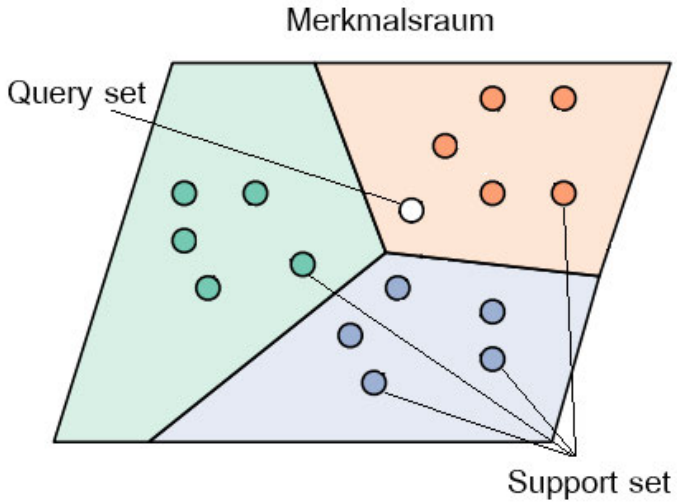


Abbildung 2.20: Projektion von Query Set und Support Set in den Merkmalsraum (veränderte Darstellung) [48]

Da Backbone-Modelle bereits auf eine große Anzahl von Daten vortrainiert sind, sind sie meist auch ohne erneutes Training in der Lage, relevante Merkmale von nicht gesehenen Klassen zu extrahieren. Aus den Merkmalsvektoren jeder Klasse des Support-Sets

wird im nächsten Schritt jeweils ein Durchschnittsvektor berechnet. Diese werden auch Prototypen genannt (siehe Abb. 2.21). Somit repräsentieren die Prototypen die jeweilige Klasse aus dem Support-Set. [48]

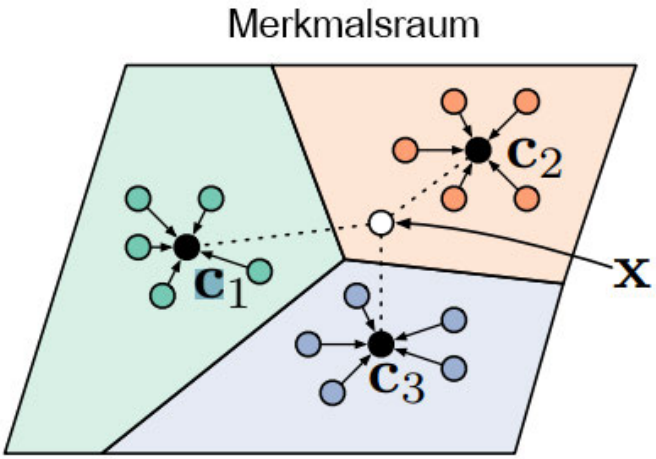


Abbildung 2.21: Darstellung der Prototypen im Merkmalsraum (veränderte Darstellung) [48]

Die Zuordnung von Merkmalsvektoren aus dem Query-Set zu einem Prototyp erfolgt durch Berechnung der Distanz im Merkmalsraum. Eine gängige Methode ist die Ermittlung der euklidischen Distanz. Dabei wird der jeweilige Merkmalsvektor aus dem Query-Set dem Klassenprototyp zugeordnet, zu dem er die kürzeste Distanz aufweist.
Obwohl ein Prototypical Network auch ohne zusätzliches Training klassifizieren kann, muss für eine gute Genauigkeit das vorgeschaltete CNN-Modell trainiert werden.
Im Gegensatz zu klassischen Trainingsmethoden erfolgt das Training hier episodisch. Bei einem episodischen Training werden zufällig generierte Klassifikationsszenarien bzw. Episoden verwendet. Diese Episoden beinhalten, genauso wie eine Few-shot-Klassifizierungsaufgabe, Support- und Query-Sets mit festgelegter Anzahl an Klassen und Beispielen pro Klasse, die für jede Episode zufällig ausgewählt werden. In der Regel werden fürs Training Klassen verwendet, die nicht für spätere Klassifizierungsaufgaben genutzt werden. Zum Beispiel wird das Modell auf Bilder mit bestimmten Hunderassen trainiert, muss jedoch durch die Benutzung des Support-Sets andere, während des Trainings nicht gesehene Hunderassen klassifizieren können. Ein anderes Beispiel, das in Abb.

2.22 illustriert wird, sind Alphabete. Das Modell kann auf Buchstaben einer Sprache trainiert werden, jedoch mit Support-Sets Buchstaben anderer Sprachen erkennen. [48]

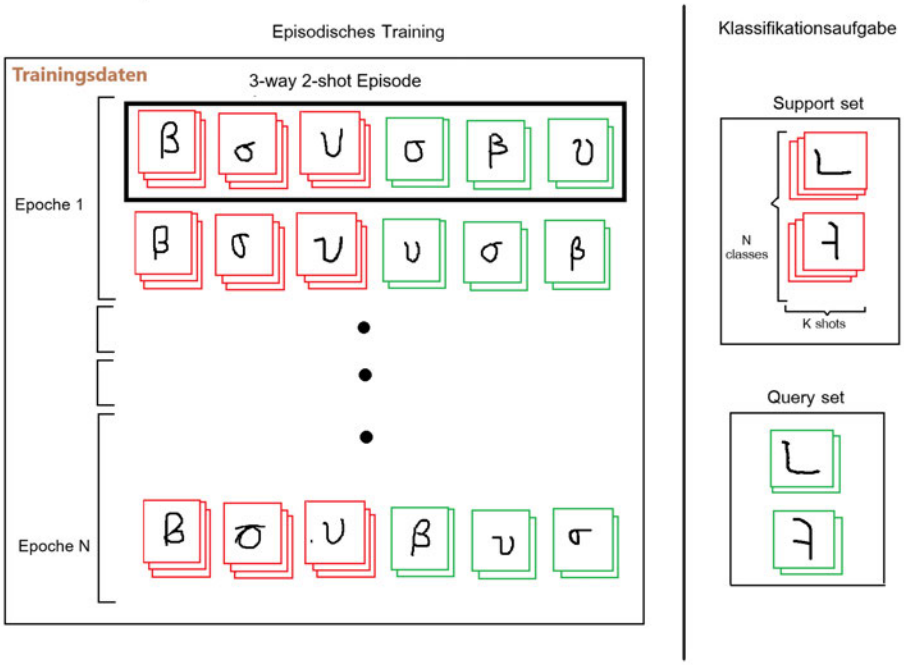


Abbildung 2.22: Episodisches Training (eigene Darstellung)

Während des Trainings durchläuft das Modell viele Episoden. Eine Epoche beim episodischen Training kann dabei als eine bestimmte Anzahl von Episoden definiert werden. Beispielsweise könnte eine Epoche aus 1.000 Episoden bestehen. Der Zweck dieser Art von Training ist es, das Modell darin zu schulen, aus wenigen Beispielen zu generalisieren. Durch die wiederholte Exposition gegenüber verschiedenen Kombinationen von Klassen und Beispielen lernt das Prototypical Network, robuste Merkmalsrepräsentationen zu erzeugen, indem die Gewichte des Backbones angepasst werden. Ein Vorteil des Few-shot Learning liegt auch darin, dass man ohne erneutes Training die Klassen flexibel entfernen bzw. hinzufügen kann.
Obwohl man wenig Daten aus den zu klassifizierenden Klassen benötigt, braucht das Modell beim Training eine große Anzahl an fremden Klassen aus einem ähnlichen Klassenbereich. Deshalb ist diese Methode nur in bestimmten Szenarien anwendbar.

2.4.3 Data Augmentation

Data Augmentation-Techniken zielen darauf ab, die Größe und Vielfalt der Trainingsdaten künstlich zu erhöhen. Zu diesem Zweck werden unterschiedliche Methoden eingesetzt, darunter geometrische Transformationen, Farbtransformationen oder generative Methoden wie GANs (Generative Adversarial Networks).
Geometrische Transformationen umfassen einfache Manipulationen wie Rotation, Spiegelung oder Skalierung von Bildern. Diese Techniken sind leicht umzusetzen und erhalten die ursprüngliche Klassifizierung der Daten.
Farbtransformationen, wie Anpassungen der Helligkeit, des Kontrasts oder der Sättigung, ermöglichen es Modellen, unterschiedliche Lichtverhältnisse zu verarbeiten und die Leistung auf vielfältigen Datensätzen zu verbessern.
Generative Methoden, insbesondere GANs, gewinnen zunehmend an Bedeutung in Data Augmentation. GANs erstellen neue, realistische Daten, indem sie die zugrundeliegende Verteilung der Trainingsdaten erlernen. Mithilfe von GANs können qualitativ hochwertige synthetische Bilder generiert werden, die die Diversität der Trainingsdaten erheblich erhöhen und dadurch die Modellleistung verbessern.

2.5 EasyOCR: Funktionsweise

Ein exemplarisches Beispiel für ein fortschrittliches, auf Deep Learning basierendes OCR-Tool ist EasyOCR. Es unterstützt mehr als 80 Sprachen und zeichnet sich durch eine relativ einfache Implementierung aus. Aufgrund des bereits erfolgten Training von EasyOCR auf einer großen Anzahl von Daten ist seitens des Benutzers weder ein rechenintensives Training des Modells noch die Beschaffung von annotierten Daten erforderlich. [43]
Obgleich die Anwendung als einfach zu bezeichnen ist, weist der interne Aufbau eine beachtliche Komplexität auf. Der Erkennungsprozess von EasyOCR ist in mehrere Verarbeitungsschritte unterteilt und basiert auf der Verwendung zweier neuronaler Netze (siehe Abb. 2.23). Die Textlokalisierung erfolgt durch ein erstes neuronales Netz, während ein zweites Netz für die Erkennung zuständig ist.

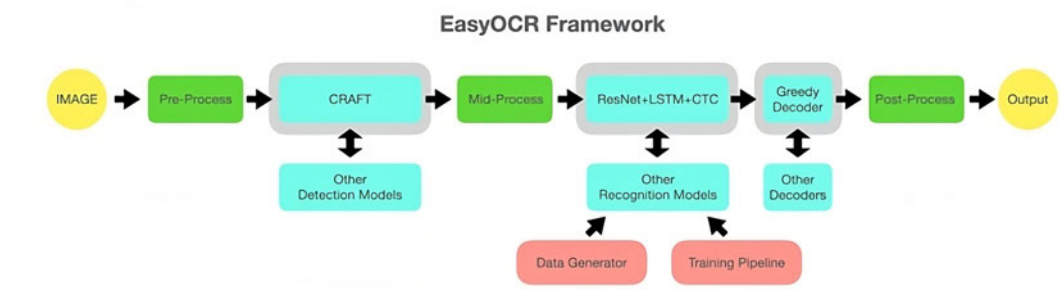


Abbildung 2.23: EasyOcr Framework [22]

Als Erstes durchläuft das zu betrachtende Eingangsbild eine Vorverarbeitung. Zur Lokalisierung der Bereiche im Bild, die Text oder Zeichen enthalten, wird ein auf CNN basiertes neuronales Netz, wie der CRAFT-Algorithmus (Character Region Awareness for Text Detection), eingesetzt. Der CRAFT-Algorithmus berechnet dabei einen sogenannten "Region Score" und "Affinity Score". Der Region Score dient zur Lokalisierung einzelner Zeichen im Bild, während der Affinity Score zur Gruppierung der isolierten Zeichen zu Textinstanzen verwendet wird, indem er Verbindungen zwischen benachbarten Zeichen schafft. [8]
Neben CRAFT können auch andere Lokalisierungsmodelle eingesetzt werden, da jedoch diese Methode auf Zeichenebene arbeitet, ist sie im Gegensatz zu anderen Algorithmen, die nur auf Wortebene lokalisieren, weniger anfällig für gekrümmte, schräge oder unregelmäßig platzierte Zeichen.(siehe Abb. 2.24).

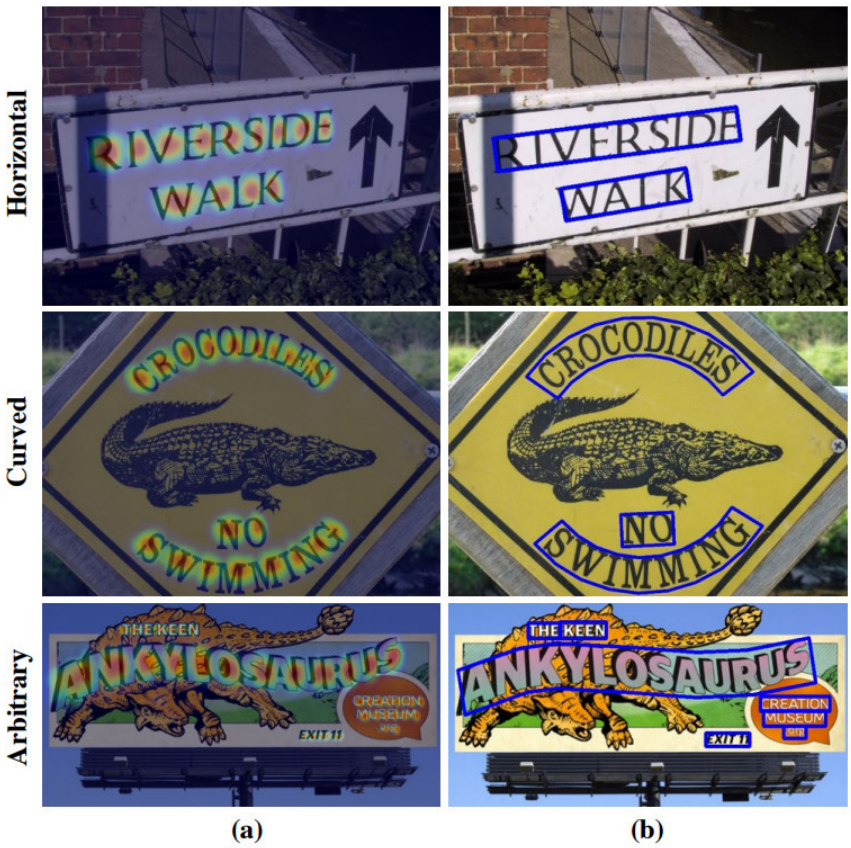


Abbildung 2.24: Visulisierung der Lokalisierung mit CRAFT. (a) Darstellung des "Region Score" als Heatmap (b) Darstellung der Endergebnisse [8]

Nach der Lokalisierung der Zeichenbereiche wird CRNN (Convolutional Recurrent Neural Network) angewendet (siehe Abb. 2.25). Dieses Modell besteht aus drei Hauptkomponenten:

- Merkmalsextraktion,
- Sequenzbeschriftung und
- Transkription

Das Besondere an CRNN ist, dass es Text in beliebiger Länge erkennen kann und keine vorherige Segmentierung (Trennung) der Zeichen benötigt. Außerdem kann es sowohl ohne vordefinierte Wörterbücher (lexikonfrei) als auch mit Wörterbüchern (lexikonbasiert) arbeiten. [45]

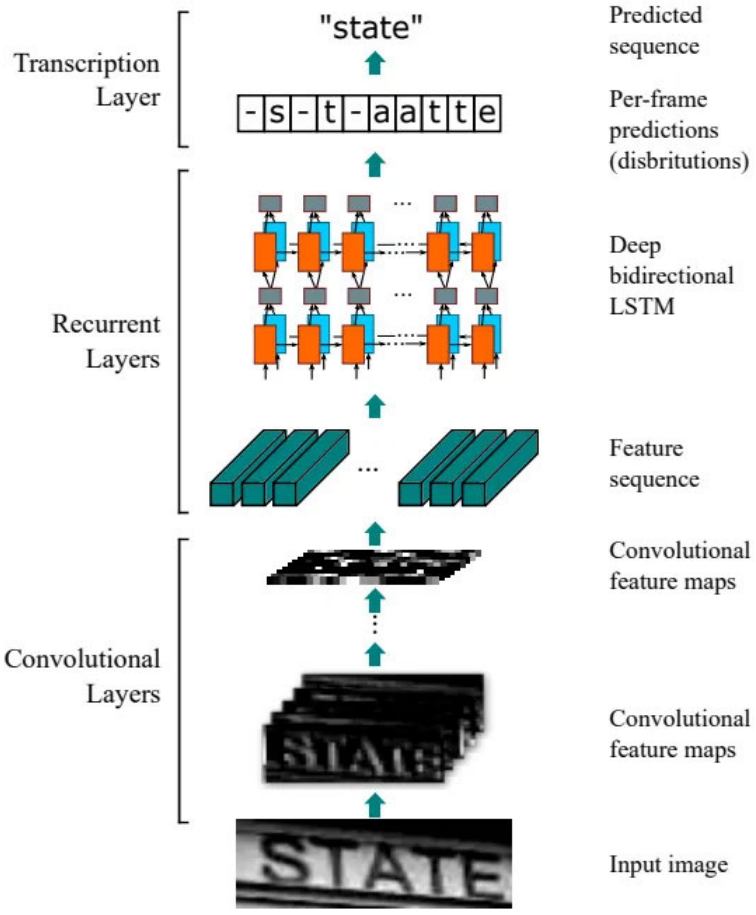


Abbildung 2.25: CRNN Netzwerkarchitektur [45]

Ein auf CNN basiertes neuronales Netz, wie z. B. ResNet (Residual-Netzwerk) wird verwendet, um Merkmalssequenzen aus den lokalisierten Textbereichen zu extrahieren. Verschiedene Strukturen werden von diesem Netzwerk erkannt und die gewonnenen Informationen werden als Eingang für nachfolgende Schritte verwendet. Die Merkmalssequenzen werden dabei von einem RNN (rekurrenten neuronalen Netz) bearbeitet. Das CRNN verwendet hier bidirektionale LSTM-Schichten (Long Short-Term Memory), welche Informationen von links nach rechts sowie umgekehrt einbeziehen. Dies ist nützlich, weil es oft einfacher ist, Buchstaben im Zusammenhang zu identifizieren. Eine Vorhersage darüber, welches Zeichen dort erscheinen könnte, und eine entsprechende Wahrscheinlichkeitsverteilung werden an jedem Punkt der Merkmalssequenz gemacht. Diese Vorhersage ist jedoch oft ungenauer, weil das LSTM tendenziell mehrere

leere oder gleiche Zeichen in einer Sequenz vorhersagen kann. Der CTC-Algorithmus (Connectionist Temporal Classification) wird daher verwendet. CTC stellt sicher, dass die Rohvorhersagen durch das Entfernen leerer Zeichen und die Zusammenfassung wiederholter Vorhersagen zu einem abschließenden, lesbaren Text werden. Nachdem der Sequenzbeschriftung wird ein "Greedy Decoder" oder ein anderer Dekodierungsansatz verwendet, um die höchstwahrscheinliche Zeichenfolge aus den bestehenden Vorhersagen zu ziehen. Die erkannten Zeichen und Texte werden abschließend nachgearbeitet, um die Genauigkeit der Erkennung zu erhöhen. [45] [20]

2.6 Aufbau von Nummern der Versandeinheit

Die Nummer der Versandeinheit ist eine eindeutige Identifikationsnummer zur Kennzeichnung und Nachverfolgung von Logistikeinheiten wie Paletten. Die NVE ermöglicht das Nachvollziehen von Informationen über die Art der Waren, deren Herkunft und Zielort sowie den Status der Lieferung. Die Nummern der Versandeinheit ist flexibel gestaltet. Sie besteht aus einer Reserveziffer, einer variablen Basisnummer, einer anpassbaren fortlaufenden Nummerierung und einer Prüfziffer (siehe Abb. 2.26).

Nummer der Versandeinheit (SSCC/NVE)			
Reserve-Ziffer*	Basisnummer der GLN	Serielle Bezugsnummer	Prüfziffer
3	4 0 1 2 3 4 5	0 0 0 0 0 0 0 0 1	7
3	4 2 1 2 3 4 5 6	0 0 0 0 0 0 0 1	9
3	4 3 1 2 3 4 5 6 7	0 0 0 0 0 0 1	2

* Reserveziffer kann mit Ziffern von 0 – 9 belegt werden

Abbildung 2.26: Beispiele für Nummern der Versandeinheit [13]

Die Reserveziffer wird vom Unternehmen selbst festgelegt. Die Basisnummer identifiziert das Unternehmen und kann je nach Länge sieben bis neun Ziffern umfassen. Die fortlaufende Nummer wird verwendet, um die einzelnen Versandeinheiten eindeutig zu kennzeichnen und wird üblicherweise durch eine aufsteigende Zählweise vom Unternehmen vergeben. Die Prüfziffer wird abschließend aus den vorherigen Ziffern nach einem festgelegten Verfahren berechnet, um die Richtigkeit der NVE zu überprüfen. [13]

3 Ausgangssituation und Anforderungsanalyse

Dieses Kapitel widmet sich der Beschreibung des aktuellen Systems sowie der darin eingesetzten Technologien und Methoden. Zudem wird eine Analyse der Leistungsfähigkeit des bestehenden Systems vorgenommen, um Optimierungspotenziale zu identifizieren und die Anforderungen an das neue System zu spezifizieren.

3.1 Beschreibung der bestehenden Gesamtlösung

Wie bereits dargelegt, ist die vorliegende Arbeit Teil einer bereits existierenden Gesamtlösung zur Erfassung von NVE. Die aktuelle Lösung besteht aus mehreren Verarbeitungsschritten und basiert auf einem zweistufigen Deep-Learning-Ansatz. Sie umfasst die in der Abbildung 3.1 dargestellten Schritte.

Abbildung 3.1: Ablauf der NVE-Erkennung (eigene Darstellung)

Der Prozess beginnt mit der Erfassung hochauflösender Bilder der Paletten durch installierte Industriekameras. Diese Bilder werden an ein Deep-Learning-Modell übergeben, das die Positionen der Etiketten im Bild bestimmt. Die erkannten Bereiche werden dabei ausgeschnitten und begradigt. Danach werden diese Bilder zur NVE-Erkennung an das Zeichenerkennungssystem übergeben. Im Nachbearbeitungsprozess werden irrelevante Erkennungen entfernt und die erkannten Ergebnisse validiert. Abschließend werden die validierten NVE zur weiteren Verarbeitung an externe Systeme übermittelt.

3.2 Verwendete Technologien und Methoden

Die installierten Industriekameras erfassen Bilder mit einer Auflösung von 3840 x 2160 Pixeln. Aufgrund dieser hohen Auflösung werden die Bilder für die Verarbeitung durch das neuronale Netz auf eine Zielgröße von 640 x 384 Pixeln skaliert. Für die Bildverarbeitung und die Lokalisierung der Etiketten kommt das Framework TensorFlow zum Einsatz, während Python als Programmiersprache dient.
Die Lokalisierung der Etiketten erfolgt mithilfe eines FCOS-Ansatzes (Fully Convolutional One-Stage Object Detection) mit einem MobileNetV3Large-Backbone. Dieser Ansatz basiert auf einer Methode, die Vorhersagen für jedes Bildraster trifft. Anstelle von vordefinierten Ankerboxen berechnet das Modell an jeder Position im Raster zwei zentrale Vorhersagen: Zum einen die Centerness, die angibt, wie nah eine Position am Zentrum eines Objekts (hier des Etiketts) liegt, und zum anderen die relativen Koordinaten der vier Ecken der Bounding Box in Bezug auf die Position. [52] Diese Vorhersagen werden auf die Originalgröße des Bildes zurückskaliert, indem die berechneten Koordinaten mit den entsprechenden Skalierungsfaktoren multipliziert werden, die das Verhältnis zwischen der Zielgröße und der Auflösung des Originalbildes widerspiegeln.
EasyOCR wird im Programm ausschließlich zur Erkennung von Ziffern konfiguriert, um irrelevante Symbole von vornherein herauszufiltern. Da auf dem Etikett jedoch neben den NVE auch andere Ziffern vorhanden sind, werden diese im Erkennungsfall durch speziell im Code implementierte Regeln ignoriert. Anschließend werden die Ergebnisse durch heuristische Methoden korrigiert und validiert, wobei nicht valide Ergebnisse ausgeschlossen werden.
Zur Übermittlung der erkannten und validierten NVE wird das MQTT-Protokoll (Message Queuing Telemetry Transport) verwendet. MQTT ist ein einfaches Nachrichtenprotokoll, das auf einem System basiert, bei dem Nachrichten an eine zentrale Stelle (Broker) gesendet werden, der diese dann an alle interessierten Abonnenten weiterleitet. In diesem

Fall überträgt MQTT die NVE an externe Systeme, wie beispielsweise Datenbanken oder SAP-Systeme, damit diese weiterverarbeitet und nachverfolgt werden können. [49]

3.3 Leistungsfähigkeit der bestehenden Lösung

Die Untersuchung der Leistungsfähigkeit und Schwächen des bestehenden Systems ist ein zentraler Schritt in der Entwicklung einer verbesserten Lösung. Da sich diese Arbeit ausschließlich auf die Optimierung der Zeichenerkennung konzentriert, wird die Analyse auf die Leistung von EasyOCR beschränkt.
Für das Projekt stehen etwa 500 Bildaufnahmen von Paletten zur Verfügung. Nach einer manuellen Bereinigung, bei der Doppelbilder sowie Aufnahmen ohne Versandetikett oder mit unleserlichen Etiketten aussortiert werden, verbleiben 250 Bilder. Die Beschaffung dieser Datenmenge ist mit einem erheblichen Aufwand verbunden und kann den regulären Betriebsablauf beim Kunden potenziell beeinträchtigen. Um dieses Problem bei zukünftigen Implementierungen des Konzepts zu minimieren, wird eine dateneffiziente Umsetzung angestrebt.
Vor diesem Hintergrund konzentriert sich diese Arbeit auf die Simulation von Szenarien mit weiter reduzierter Trainingsdatenmenge. Dieser Ansatz zielt darauf ab, Methoden zu entwickeln und zu evaluieren, die auch bei begrenzter Datenverfügbarkeit effektive Ergebnisse liefern können.
Typischerweise werden Datensätze für das Training neuronaler Netze in Trainings-, Validierungs- und Testdaten aufgeteilt. Die Trainingsdaten dienen der Anpassung der internen Parameter des neuronalen Netzes, wie Gewichten und Biases. Die Validierungsdaten werden während des Trainings verwendet, um die Modellleistung auf zuvor nicht gesehenen Daten zu evaluieren und eine mögliche Überanpassung (Overfitting) frühzeitig zu erkennen. Overfitting tritt auf, wenn ein Modell die Trainingsdaten zu genau erlernt und dadurch seine Fähigkeit zur Generalisierung auf neue Daten verringert (siehe Abb. 3.2).

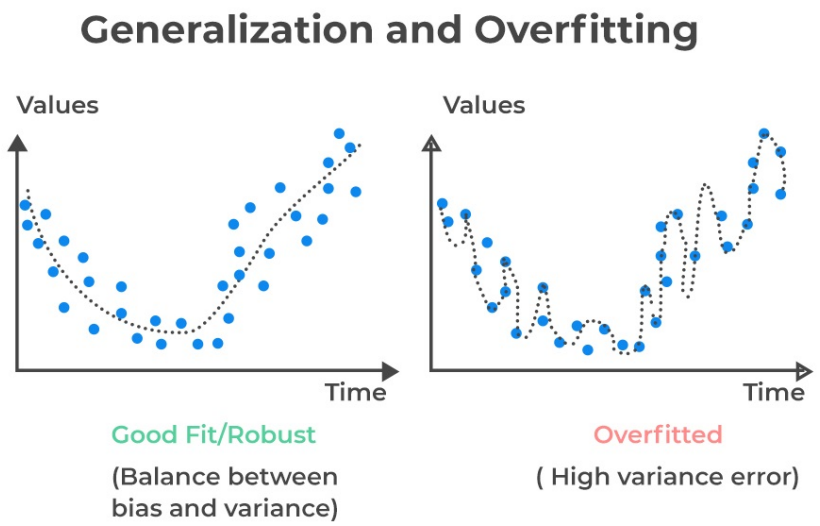


Abbildung 3.2: Vergleich zwischen gute Generalisierung und Overfitting (veränderte Darstellung) [5]

Nach Abschluss des Trainings- und Validierungsprozesses erfolgt die finale Evaluation des Modells anhand der Testdaten. Diese Testdaten repräsentieren einen zuvor vollständig ungesehenen Datensatz, der ausschließlich zur Beurteilung der endgültigen Modellleistung herangezogen wird.
Die Aufteilung des Gesamtdatensatzes in diese drei Kategorien folgt üblicherweise bestimmten Proportionen, die sich in der Praxis bewährt haben. Eine gängige Verteilung für kleine bis mittlere Datensätzen ist in der Tabelle 3.1 dargestellt.

Tabelle 3.1: Verhältnis der Trainings-, Validierungs- und Testdaten bei kleinen bis mittleren Datensätzen (10.000-500.000 Daten) [34].

Trainingsdaten	Validierungsdaten	Testdaten
70%	10%	20%

Aufgrund der begrenzten Größe des verfügbaren Datensatzes und der Absicht, Szenarien mit weiter eingeschränkten Datenmengen zu untersuchen, wird in dieser Arbeit von der üblichen Aufteilung abgewichen.
Zunächst werden 100 zufällige Bilder mit einem entwickelten Python-Skript ausgewählt und als fester Testdatensatz festgelegt. Dieser Satz bleibt während der gesamten Untersu-

chung unverändert, um eine konsistente Basis für die Leistungsbewertung verschiedener Modellkonfigurationen zu schaffen. Die Aufteilung der übrigen Bilder in Trainings- und Validierungsdaten wird im Konzeptionellen Teil der Arbeit erörtert.
Zur Bewertung der Leistungsfähigkeit der Zeichenerkennung in der bestehenden Lösung wird die NVE-Erkennungsrate als primäre Metrik herangezogen. Diese Kennzahl quantifiziert den prozentualen Anteil der korrekt erkannten NVE an der Gesamtheit aller im Datensatz vorhandenen NVE. Eine Erkennung wird als korrekt gewertet, wenn alle 18 Ziffern der NVE vollständig und fehlerfrei erkannt werden.

$$\text{NVE-Erkennungsrate} = \left(\frac{\text{True-NVE}}{\text{True-NVE} + \text{False-NVE}}\right) \times 100 \tag{3.1}$$

Dabei steht:

- True-NVE für die Anzahl der korrekt erkannten NVE,
- False-NVE für die Anzahl der falsch erkannten NVE.

Die Testdaten werden zur Lokalisierung der Etiketten durch das erste neuronale Netz verarbeitet. Die resultierenden ausgeschnittenen und begradigten Bilder werden anschließend in einem lokalen Ordner zur weiteren Analyse gespeichert
Da der Fokus dieser Untersuchung ausschließlich auf der Leistungsfähigkeit von EasyOCR liegt, ist es essenziell, potenzielle Engpässe in anderen Systemkomponenten auszuschließen. Zu diesem Zweck werden die vorverarbeiteten Etikettenbilder einer zusätzlichen manuellen Überprüfung unterzogen. Diese Qualitätskontrolle dient der Sicherstellung, dass etwaige Erkennungsfehler eindeutig der OCR-Komponente zugeordnet werden können.
Die Überprüfung des Testdatensatzes ergibt, dass in keinem der Fälle eine fehlerhafte Lokalisierung der Etiketten vorliegt. Folglich können alle 100 Testbilder für die Evaluierung herangezogen werden. Darüber hinaus ist hervorzuheben, dass in sämtlichen 100 Beispielen die NVE-Ziffern für das menschliche Auge lesbar sind. Diese Feststellung ist bedeutsam, da sie einen Referenzpunkt für die erwartete Leistung des automatisierten Systems bietet.
Der durchgeführte Test offenbart eine NVE-Erkennungsrate von 77% ohne Anwendung heuristischer Reparaturmethode. In der praktischen Anwendung bedeutet dies, dass etwa 23% der Nummern der Versandeinheiten nicht korrekt identifiziert werden. Durch Verwendung von heuristischen Reparatur wird die NVE-Erkennungsrate auf 81% Verbessert.

Obwohl diese Erkennungsrate auf den ersten Blick kritisch erscheinen mag, ist es wichtig, den praktischen Kontext zu berücksichtigen. In realen Anwendungsszenarien werden in der Regel mehrere Aufnahmen einer Palette gemacht. Dies eröffnet die Möglichkeit, im Falle einer Falscherkennung durch einen Prüfziffervergleich den Fehler zu identifizieren und auf nachfolgende Bildaufnahmen derselben Palette zurückzugreifen, um einen erneuten Erkennungsversuch zu starten.
Die Messung der Verarbeitungsgeschwindigkeit von EasyOCR liefert ebenfalls wichtige Erkenntnisse für die Bewertung der Systemleistung. Die Leistungsmessung ergibt eine durchschnittliche Inferenzgeschwindigkeit von 3,3 Bildern pro Sekunde. Eine Verarbeitungsrate von 3,3 Bildern pro Sekunde ist für den vorliegenden Anwendungsfall mehr als ausreichend. Diese Geschwindigkeit ermöglicht eine zügige Verarbeitung der Versandetiketten, ohne den logistischen Workflow zu beeinträchtigen.

3.4 Schwachstellen der bestehenden Lösung

Eine Analyse der nicht erkannten NVE ermöglicht es, spezifische Schwachstellen in der Leistung von EasyOCR zu identifizieren. Die Untersuchung zeigt, dass das System insbesondere bei unscharfen und dunklen Bildern sowie geknickten Etiketten Schwierigkeiten aufweist (siehe Abb. 3.3)
Diese Limitationen lassen sich auf den ursprünglichen Entwicklungsfokus von EasyOCR zurückführen. Das System ist primär für die Erkennung strukturierter Dokumente wie PDF-Dateien, Rechnungen und Quittungen konzipiert [33]. Folglich ist es weniger auf die Bewältigung der spezifischen Herausforderungen optimiert, die bei der Erfassung von Versandetiketten in realen Logistikumgebungen auftreten.

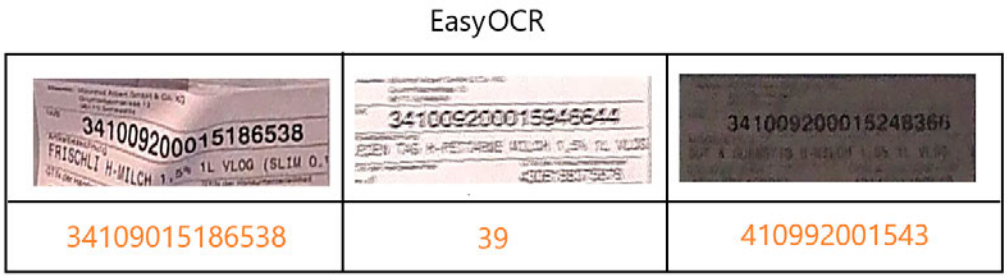


Abbildung 3.3: 3 Beispiele für fehlerhafte Erkennung mit EasyOCR: Oben der Bildausschnitt, unten die von EasyOCR erkannte NVE (eigene Darstellung)

Das neu zu entwickelnde Modell zielt darauf ab, die identifizierten Defizite von EasyOCR zu adressieren und somit die bestehende Lücke in der Zeichenerkennungsleistung zu schließen. Der Fokus liegt dabei auf der Optimierung des Systems nicht nur für ideale, sondern insbesondere für die herausfordernden Bedingungen, die in der praktischen Anwendung häufig auftreten.

3.5 Anforderungen an das neue System

Dieser Abschnitt widmet sich der Ableitung der Anforderungen und technischen Rahmenbedingungen für das neu zu entwickelnde System. Die Grundlage hierfür bildet die vorangegangene Analyse der bestehenden Lösung, kombiniert mit den spezifischen Zielsetzungen dieser Arbeit

3.5.1 Funktionale und nicht-funktionale Anforderungen

Funktionale und nicht-funktionale Anforderungen definieren zusammen, was das System leisten soll und wie diese Leistungen unter verschiedenen Bedingungen erbracht werden. Sie bestimmen sowohl die konkreten Aufgaben des Systems als auch die Bedingungen, die für die erfolgreiche Umsetzung dieser Aufgaben erforderlich sind.

Funktionalität und Leistung

Das primäre Ziel des neuen Systems ist es, eine signifikante Verbesserung gegenüber der bestehenden EasyOCR-basierten Lösung zu erreichen. Konkret wird eine Steigerung der NVE-Erkennungsrate um mindestens 5% angestrebt. Diese Anforderung resultiert aus der Notwendigkeit, die Genauigkeit und Zuverlässigkeit des Gesamtsystems substanziell zu erhöhen, um den Anforderungen in der Logistikbranche gerecht zu werden. Die Fähigkeit zur Klassifizierung aller Ziffern von null bis neun ist dabei eine fundamentale Voraussetzung für die Erkennung der NVE. Gleichzeitig muss das System in der Lage sein, zusätzliche Symbole oder irrelevante Ziffern aus den Ergebnissen herauszufiltern. Diese Anforderung ergibt sich aus der Komplexität realer Versandetiketten, die neben den relevanten NVE oft weitere Informationen enthalten.

Dateneffizienz

Ein zentraler Aspekt dieser Arbeit ist die Entwicklung eines dateneffizienten Verfahrens. Daher muss das neue Modell in der Lage sein, die oben genannten Leistungsmerkmale

mit einem begrenzten Trainingsdatensatz[1] von lediglich 50 Bildern zu erreichen. Wie bereits erwähnt ist diese Anforderung besonders wichtig, da in vielen praktischen Anwendungsszenarien der Zugang zu großen Mengen annotierter Daten oft limitiert und Aufwendig ist.

Verarbeitungsgeschwindigkeit

Das neue Modell muss eine Mindestverarbeitungsgeschwindigkeit von einem Bild pro Sekunde erreichen. Diese Anforderung wurde unter Berücksichtigung der praktischen Anwendungsbedürfnisse und der Leistung des bestehenden Systems festgelegt.
Obwohl diese Verarbeitungsrate keine Echtzeitverarbeitung im strengen Sinne darstellt, ist sie für den spezifischen Anwendungskontext in der Logistik ausreichend dimensioniert. Das Hauptziel dieser Geschwindigkeitsvorgabe ist es, potenzielle Verzögerungen im operativen Ablauf zu vermeiden und einen reibungslosen Arbeitsfluss zu gewährleisten.

Zusammenfassung und Priorisierung der Anforderungen

Die folgende Tabelle fasst die definierte Anforderungen zusammen und ordnet sie nach ihrer Priorität:

[1] Damit werden je nach Modell auch Validierungsdaten einbezogen

Tabelle 3.2: Priorisierung der Systemanforderungen

Priorität	Anforderung
1	Fähigkeit zur korrekten Klassifizierung aller Ziffern von 0 bis 9
2	Steigerung NVE-Erkennungsrate um mindestens 5% gegenüber der bestehenden EasyOCR-Lösung
3	Leistung auch bei begrenztem Trainingsdatensatz (50 Bilder)
4	Filterung irrelevanter Symbole und Ziffern
5	Verarbeitung von mindestens einem Bild pro Sekunde
6	Zuverlässige Leistung unter verschiedenen Bildbedingungen (unscharf, dunkel, geknickt)

3.5.2 Technische Rahmenbedingungen

Technische Rahmenbedingungen legen die spezifischen technischen Vorgaben fest, unter denen das System entwickelt und betrieben wird. Diese betreffen Aspekte wie die verwendete Hardware, Software, Programmiersprachen und Betriebssysteme, um sicherzustellen, dass das System unter den definierten technischen Voraussetzungen funktioniert.

Entwicklungsumgebung

Das neue System muss in Python entwickelt werden, um eine nahtlose Integration in die vorhandene Codebasis zu gewährleisten. Die Wahl des Deep-Learning-Frameworks ist dabei auf TensorFlow-kompatible Lösungen beschränkt, wobei alternative Frameworks nur in Betracht gezogen werden können, wenn eine Konvertierung zu TensorFlow geeignetes Format möglich ist.

Systemumgebung

Die Zielplattform für das neue System ist ein Windows-Server, was direkte Auswirkungen auf die Entwicklung und Auswahl von Komponenten hat. Alle verwendeten Bibliotheken und Tools müssen vollständig Windows-kompatibel sein.

Hardware-Anforderungen

Die Unterstützung von NVIDIA-GPUs und CUDA-Technologie ist für das neue System vorgegeben. Diese Anforderung basiert auf der vorhandenen Hardware-Infrastruktur und zielt darauf ab, die Rechenleistung für Training und Inferenz[2] zu maximieren.

Zusammenfassung und Priorisierung der technischen Rahmenbedingungen

Die folgende Tabelle fasst die technischen Rahmenbedingungen zusammen und ordnet sie nach ihrer Priorität:

Tabelle 3.3: Priorisierung der technischen Rahmenbedingungen

Priorität	Technische Rahmenbedingung
1	Entwicklung in Python
2	NVIDIA-GPU und CUDA-Unterstützung
4	TensorFlow-Kompatibilität
3	Lauffähigkeit auf Windows-Servern

[2] Inferenz im maschinellen Lernen bezieht sich auf den Prozess, bei dem ein trainiertes Modell auf neue, unbekannte Daten angewendet wird, um Vorhersagen zu treffen.

4 Konzeption

Im konzeptionellen Teil dieser Arbeit wird auf Basis der festgelegten Anforderungen die Auswahl und Integration der technischen Werkzeuge und Methoden zur Umsetzung des neuen Systems beschrieben. Die Programmiersprache und das Deep-Learning-Framework werden ausgewählt, verschiedene Ansätze und Tools zur Zeichenerkennung diskutiert und verglichen. Basierend auf diesen Entscheidungen wird die Systemkombination und ihre grobe Funktionsweise erläutert. Anschließend wird das Konzept für die Evaluierung vorgestellt, wobei die zu simulierenden Szenarien beschrieben werden.

4.1 Auswahl der Hardware

Das System wird auf einem von Siemens bereitgestellten Laptop entwickelt, der die folgenden Hardwaresystemkonfigurationen aufweist:

- **Betriebssystem:** Windows 11 Pro
- **CPU:** Intel Core i7-8850H
- **GPU:** Nvidia Quadro P2000

Diese Hardware erfüllt die Anforderungen der Windows-Kompatibilität und ermöglicht die Entwicklung des Systems mit NVIDIA-GPU und CUDA-Unterstützung, wie sie in den zuvor definierten technischen Rahmenbedingungen beschrieben sind. Die Konfiguration stellt sicher, dass die nötige Rechenleistung für Trainings- und Inferenzprozesse zur Verfügung steht und die ausgewählten Softwarebibliotheken vollständig genutzt werden können.

4.2 Auswahl der Programmiersprache und des Frameworks

Die Auswahl der Programmiersprache und des Deep-Learning-Frameworks bildet das Fundament für die Struktur und Umsetzung des Projekts. In Übereinstimmung mit den

definierten Anforderungen fällt die Wahl der Programmiersprache auf Python.
Bei der Wahl des Deep-Learning-Frameworks stehen primär PyTorch und TensorFlow zur Diskussion. Beide Frameworks bieten leistungsstarke Werkzeuge zur Erstellung und zum Training neuronaler Netze und haben sich in der Praxis bewährt.
Ein zentrales Konzept, das sowohl PyTorch als auch TensorFlow zugrunde liegt, ist der Berechnungsgraph. Dieser Graph repräsentiert den Datenfluss und die Abhängigkeiten zwischen den Operationen in einem neuronalen Netzwerk auf abstrakte Weise. Er visualisiert den komplexen Prozess, wie Daten von der Eingabeschicht durch die verschiedenen Ebenen des Netzwerks propagieren, dabei schrittweise transformiert werden und letztendlich in der Ausgabeschicht resultieren.
Im Gegensatz zu statischen Berechnungsgraphen, wie sie in TensorFlow verwendet werden, erlaubt PyTorch die Verwendung dynamischer Berechnungsgraphen (Dynamic Computational Graphs, DCGs). Diese dynamischen Graphen legen die Beziehungen zwischen den Operationen erst zur Laufzeit fest. Sie ermöglichen es, das Modell während der Entwicklung anzupassen und zu debuggen, ohne den gesamten Graphen neu zu kompilieren. Dies reduziert die Komplexität bei der Entwicklung und erleichtert die Fehlersuche, da einzelne Modellkomponenten schnell iteriert und überprüft werden können. Diese Flexibilität ist einer der Hauptvorteile von PyTorch, insbesondere in der Forschung und im Prototyping. [11]
Ein weiterer wesentlicher Vorteil liegt in der starken Präsenz von PyTorch im aktuellen Forschungsumfeld. Eine Vielzahl moderner Deep-Learning-Modelle werden primär in PyTorch implementiert und publiziert. Dieser Umstand eröffnet den Zugang zu einem breiten Spektrum vortrainierter Modelle und State-of-the-Art-Implementierungen. Die Nutzung dieser Ressourcen kann zu einer deutlichen Reduktion der Entwicklungszeit führen. [35]
Obwohl im aktuellen System TensorFlow verwendet wird, bietet PyTorch durch das Open Neural Network Exchange (ONNX)-Format eine Lösung für eine nahtlose Kompatibilität. ONNX ermöglicht den Austausch von Modellen zwischen PyTorch und TensorFlow, sodass ein in PyTorch entwickeltes Modell problemlos in das TensorFlow-basierte System integriert werden kann. Dadurch lassen sich die Vorteile von PyTorch während der Entwicklungsphase voll ausschöpfen, ohne bei der abschließenden Integration in das bestehende System signifikante Kompromisse eingehen zu müssen. [6] Die Gegenüberstellung beider Frameworks ist in der Tabelle 4.2 dargestellt.

Tabelle 4.1: Vergleich von PyTorch und TensorFlow

Aspekt	PyTorch	TensorFlow
Berechnungsgraphen	+++	++
Debugging	+++	+
Forschungsfreundlichkeit	+++	++
Produktionsreife	++	+++
Flexibilität	+++	++
Community	++	+++
Integration in das bestehende System	++	+++

Im Rahmen dieser Arbeit wird in den Tabellen ein Bewertungssystem verwendet, bei dem die Symbolik +, ++ und +++ zur Einordnung der jeweiligen Leistungsmerkmale dient. Dabei steht +++ für die bestmögliche Ausprägung eines Aspekts, ++ für eine mittlere und + für eine geringere Leistung. Das Symbol '-' signalisiert, dass die jeweilige Eigenschaft nicht erfüllt wird.
Nach sorgfältiger Abwägung dieser Faktoren fällt die Wahl auf PyTorch als primäres Framework für dieses Projekt. Diese Entscheidung repräsentiert einen optimalen Kompromiss zwischen der erforderlichen Flexibilität in der Entwicklungsphase und der notwendigen Kompatibilität mit dem bestehenden TensorFlow-System. Sie ermöglicht es, die Stärken beider Frameworks zu nutzen und dabei die projektspezifischen Anforderungen optimal zu erfüllen.

4.3 Auswahl des Ansatzes

Das Konzept des neuen OCR-Systems lässt sich grundsätzlich in einstufige und zweistufige Ansätze unterteilen.
Beim einstufigen Ansatz wird Klassifizierung der Zeichen in einem einzigen Schritt durchgeführt. Diese Modelle führen keine explizite Lokalisierung der Zeichen durch. Stattdessen betrachten sie das gesamte Bild und versuchen, direkt die Zeichen zu erkennen. Dieser Ansatz zeichnet sich durch seine vergleichsweise einfache Implementierung und schnellere Verarbeitungszeit aus. Allerdings geht dies oft mit Einbußen in der Erkennungsgenauigkeit einher, da das Modell Muster aus dem gesamten Bild extrahiert, einschließlich jener Bereiche, die für die Erkennung von NVE irrelevant sind. Dies erfor-

dert in der Regel mehr Trainingsdaten, um eine relativ hohe Erkennungsgenauigkeit zu erreichen.
Der zweistufige Ansatz unterteilt den Prozess der Zeichenerkennung in zwei Phasen. In der ersten Phase werden die Zeichen im Bild lokalisiert und markiert. Anschließend erfolgt in der zweiten Phase die separate Klassifizierung jedes lokalisierten Zeichens. Dieser Ansatz ist zwar in der Implementierung und im Training komplexer, bietet jedoch die Möglichkeit einer präziseren Anpassung an komplexe Szenarien. Ein wesentlicher Vorteil des zweistufigen Ansatzes liegt in der gezielten Lokalisierung einzelner Zeichen. Dies ermöglicht es dem Klassifizierungsmodell, sich auf die Merkmale dieser begrenzten Segmente zu konzentrieren, was insbesondere bei begrenzten Trainingsdatensätzen oder komplexen Hintergrundszenarien zu überlegenen Ergebnissen führt. Eine Gegenüberstellung dieser Ansätze ist in der Tabelle 4.2 dargestellt.

Tabelle 4.2: Vergleich der einstufigen und zweistufigen Ansätze zur Zeichenerkennung

Aspekt	**Einstufiger Ansatz**	**Zweistufiger Ansatz**
Genauigkeit [Prio. 1]	+	+++
Datenbedarf [Prio. 2]	++	+++
Geschwindigkeit [Prio. 3]	+++	+
Komplexität [Prio. 4]	+++	++

Unter Berücksichtigung der Anforderungen des Projekts, insbesondere der Notwendigkeit einer hohen Genauigkeit bei einem begrenzten Datensatz, wird ein zweistufiger Ansatz als optimal erachtet. Der zweistufige Ansatz bietet einen ausgewogenen Kompromiss zwischen dem Schwierigkeitsgrad der Implementierung und der Genauigkeit, wobei die Möglichkeit, beide Stufen unabhängig voneinander zu optimieren, in einigen Szenarien sogar als Vorteil betrachtet werden kann.

4.4 Methoden zur Lokalisierung der Ziffern

Das Grundlagenkapitel präsentierte bereits verschiedene Methoden, die zur Lokalisierung der Ziffern in Frage kommen. darunter der CRAFT-Algorithmus, der die Basis von EasyOCR bildet, sowie Objekterkennungsmodelle wie Faster R-CNN und YOLO, welche die Ziffern als einzelne Objekte behandeln können. Ein weiterer Ansatz ist die klassische Bildverarbeitungsmethode mit OpenCV in Python, die durch einfache Bildverarbei-

tungsoperationen in Verbindung mit Konturerkennungsalgorithmen die Lokalisierung von Ziffern ermöglicht.
Das CRAFT-Modell verwendet einen durchdachten Ansatz zur Lokalisierung von Texten. Obwohl das nachgeschaltete CRNN-Modell in EasyOCR mit Textregionen arbeitet und nicht mit einzelnen Zeichen (vgl. Abschnitt 2.5), kann CRAFT nach einigen Anpassungen am Quellcode prinzipiell die Koordinaten einzelner Zeichen ausgeben. Ein signifikanter Vorteil dieses Modells liegt in seiner Fähigkeit, Zeichen auch unter anspruchsvollen Bedingungen zuverlässig zu lokalisieren, etwa bei schrägen oder gekrümmten Zeichenfolgen. Dies macht CRAFT zunächst zu einem vielversprechenden Kandidaten für die Zeichenlokalisierung. Allerdings erfordert CRAFT für eine optimale Leistung bei Bildern minderer Qualität ein spezifisches Training. Hier tritt eine entscheidende Einschränkung zutage: Die Entwickler des CRAFT-Modells stellen aus Gründen des IP-Schutzes (Schutz des geistigen Eigentums) den vollständigen Quellcode für das Training nicht zur Verfügung. Diese Limitation beschränkt den Einsatz auf das bereits trainierte Modell.
Die OpenCV-Bibliothek in Python stellt eine weitere Option für die Zeichenlokalisierung dar. Diese Methode nutzt grundlegende Bildverarbeitungsoperationen, um das Bild so aufzubereiten, dass eine einfache Konturerkennung zur Lokalisierung der Ziffern ausreicht. Ein bedeutender Vorteil dieses Ansatzes liegt darin, dass er kein Training erfordert. Dies vereinfacht die Implementierung und reduziert den Bedarf an umfangreichen Trainingsdaten. Allerdings erfordert diese Flexibilität, dass die Bildverarbeitungsoperationen für jeden spezifischen Anwendungsfall individuell angepasst werden müssen. Die größte Einschränkung dieser Methode zeigt sich in ihrer begrenzten Leistungsfähigkeit bei Bildern von minderer Qualität. In Szenarien mit unscharfen, kontrastarmen oder verzerrten Aufnahmen kann die Zuverlässigkeit der Zeichenlokalisierung deutlich abnehmen.
Eine alternative Herangehensweise zur Zeichenerkennung besteht darin, Zeichen als eigenständige Objekte zu betrachten. In diesem Kontext können Objekterkennungsmodelle wie Faster R-CNN, YOLO oder SSD (Single Shot Detector) für die Zeichenlokalisierung eingesetzt werden. SSD ist dabei ein zu YOLO vergleichbares Objekterkennungsmodell. Seine Architektur basiert auf einem Backbone-Netzwerk (typischerweise VGG16) mit zusätzlichen Convolutional Layers, die schrittweise die räumliche Auflösung der Feature Maps reduzieren. [27]
Obwohl diese Modelle neben der Lokalisierung auch eine Klassifizierung der Zeichen ermöglichen und somit potenziell als einstufiger Ansatz für das neue System in Betracht kommen könnten, zeigt die Erfahrung, dass ihre Fähigkeit zur korrekten Zeichenklassifizierung in der Regel weniger ausgeprägt ist als ihre Objektlokalisierungsfähigkeit.

Basierend auf diesen Erkenntnissen wird bei den genannten Objekterkennungsmodellen vorrangig deren Lokalisierungsfähigkeit genutzt. Dennoch kann auch ihre Klassifizierungsleistung in die Evaluierung einbezogen werden, da dies meist keinen signifikanten Zusatzaufwand erfordert und möglicherweise eine umfassendere Bewertung der Modelle ermöglicht.
Der zusammengefasste Vergleich der Methoden ist in der Tabelle 4.3 dargestellt.

Tabelle 4.3: Vergleich der Methoden zur Zeichenlokalisierung

Aspekt	**CRAFT**	**OpenCV**	**Objekterkennung**
Umsetzbarkeit [Prio. 1]	-	+++	+++
Genauigkeit [Prio. 2]	+++	+	+++
Datenbedarf [Prio. 3]	++	+++	++
Geschwindigkeit [Prio. 4]	++	+++	+++
Komplexität [Prio. 5]	++	+++	++

Die Analyse der vorgestellten Methoden unter Berücksichtigung der Projektanforderungen identifiziert Objekterkennungsmodelle als die optimale Wahl für den vorliegenden Anwendungsfall. Die Genauigkeit der Lokalisierung stellt dabei einen entscheidenden Faktor dar. OpenCV zeigt deutliche Schwächen bei der Verarbeitung komplexer Bilder und unter variablen Beleuchtungsbedingungen. Der CRAFT-Algorithmus bietet zwar eine hohe Leistungsfähigkeit, ist jedoch aufgrund des fehlenden Zugangs zum vollständigen Trainingscode in seiner Anpassungsfähigkeit eingeschränkt. Im Gegensatz dazu zeichnen sich Objekterkennungsmodelle wie YOLO, SSD oder Faster R-CNN durch eine gute Leistung unter verschiedenen Bedingungen aus. Ihre hohe Anpassungsfähigkeit durch Training ist ein weiterer signifikanter Vorteil. Diese Eigenschaften machen sie besonders geeignet, um den variierenden Anforderungen im logistischen Bereich gerecht zu werden.
Im nächsten Schritt wird die Auswahl eines geeigneten Objekterkennungsmodells vorgenommen. Die YOLO-Familie bietet eine Vielzahl von Versionen, die stetig weiterentwickelt werden. Um die neuesten Fortschritte in der Objekterkennung zu nutzen, konzentriert sich diese Analyse auf die aktuellen Versionen YOLOv8 und YOLOv9. Diese neueren Versionen weisen im Vergleich zu ihren Vorgängern signifikante Verbesserungen in der Genauigkeit auf.

Als Referenzpunkt wird zusätzlich YOLOv3 in die Betrachtung einbezogen. Eine Gegenüberstellung der Leistungsmerkmale dieser Modelle findet sich in der Tabelle 4.4.

Tabelle 4.4: Vergleich der Objekterkennungsmodelle [50] [24] [23] [51].

Aspekt	**SSD512**	**Faster R-CNN**	**YOLOv3**	**YOLOv8**	**YOLOv9**
Genauigkeit: mAP [Prio. 1]	26,8	35	33	53,9	55,6
Geschwindigkeit [Prio. 2]	+	++	+++	+++	+++
Implementierungs-komplexität [Prio. 3]	++	+	++	+++	+++

Der Vergleich von Objekterkennungsmodellen anhand des COCO 2017-Datensatzes[1] zeigt deutliche Unterschiede in Leistung und Geschwindigkeit. SSD512, eine Weiterentwicklung der SSD-Architektur, bietet eine hohe Verarbeitungsgeschwindigkeit, erreicht jedoch mit einem mAP-Wert von 26,8 eine relativ geringe Genauigkeit im Vergleich zu neueren Ansätzen. Trotzdem findet SSD aufgrund seiner Geschwindigkeit oft Anwendung in Echtzeit-Szenarien. Faster R-CNN erzielt mit einem mAP von 35 eine höhere Genauigkeit als SSD, weist jedoch im Vergleich zu den restlichen Modellen eine geringe Verarbeitungsgeschwindigkeit auf. Die komplexe Architektur von Faster R-CNN resultiert zudem in einem erhöhten Implementierungsaufwand.
YOLOv3 erreicht mit einem mAP von 33 eine ähnliche Genauigkeit wie Faster R-CNN, bietet aber eine höhere Verarbeitungsgeschwindigkeit. Es bleibt jedoch in der Präzision hinter den neueren Modellen YOLOv8 und YOLOv9 zurück, die deutlich höhere mAP-Werte von 53,9 und 55,6 erreichen.
Für die vorliegende Aufgabenstellung wird YOLOv9 ausgewählt. Es bietet die höchste Detektionsgenauigkeit und hohe Inferenzgeschwindigkeiten bei gleichzeitig einfacher Implementierung, was den Anforderungen des Projekts am besten entspricht.

[1] Der COCO 2017-Datensatz (Common Objects in Context) ist ein bekannter Datensatz zur Bildverarbeitung, der Bilder mit markierten Objekten in realen Szenen enthält. Er wird häufig für das Training und die Bewertung von Objekterkennungsmodellen eingesetzt.

4.5 Methoden zur Erkennung der lokalisierten Ziffern

In der zweiten Stufe des Ansatzes erfolgt die Erkennung der Ziffern basierend auf Bounding Boxes die durch das Objekterkennungsmodell identifiziert werden . Zunächst findet die Extraktion der Ziffern statt. Hierbei werden die Bildbereiche, die durch die Bounding Boxes definiert sind, aus dem Gesamtbild herausgeschnitten. Anschließend werden diese extrahierten Ziffern einem Klassifikationsmodell zugeführt. Die Aufgabenstellung für dieses zweite Modell vereinfacht sich dadurch zu einem vergleichsweise simplen Klassifikationsproblem mit zehn Klassen, ähnlich dem bekannten MNIST-Datensatz (vgl. Abschnitt 2.1.3).
Im Kapitel 2 wurden bereits verschiedene Methoden zur Klassifizierung diskutiert. Eine etablierte Methode für Klassifizierungsprobleme ist die Feature Extraction mit anschließender Klassifikation, beispielsweise durch Softmax oder K-Nearest Neighbors (KNN). Dieser Ansatz zeichnet sich durch seine schnelle Erkennungsleistung aus, was ihn zunächst attraktiv erscheinen lässt. Jedoch stellt er hohe Anforderungen an die Merkmalsauswahl. Wie bereits im Grundlagenkapitel beschrieben, resultiert dieser Ansatz häufig in suboptimaler Klassifikationsgenauigkeit. Die Herausforderung liegt darin, dass die manuell ausgewählten Merkmale möglicherweise nicht alle relevanten Aspekte der Ziffern erfassen oder nicht robust genug gegenüber Variationen in den Eingabedaten sind.
Eine vielversprechendere Alternative bietet der Einsatz klassischer neuronaler Netze. Diese Modelle können komplexe Muster in den Daten erlernen, ohne dass eine manuelle Merkmalsextraktion erforderlich ist. Allerdings, wie bereits im Abschnitt 2.2.4 anhand eines anschaulichen Beispiels demonstriert, führt der Einsatz solcher Modelle bei mehrdimensionalen Aufgaben wie der Bilderkennung zu einer großen Anzahl zu trainierender Parameter. Diese hohe Parameteranzahl kann dazu führen, dass das Modell schnell an seine Grenzen stößt.
CNNs haben sich bereits als optimale Lösung für die Ziffernklassifizierung erwiesen, wie das Modell von Yann LeCun am MNIST-Datensatz zeigt. Ein bedeutender Vorteil von CNNs liegt in der Verfügbarkeit umfangreicher Data Augmentation-Techniken, die nahtlos in PyTorch integriert sind. Dies spielt eine entscheidende Rolle im Kontext der Dateneffizienz.
Als alternative Methode kann das Prototypical Network in Betracht gezogen werden, das speziell für Few-Shot-Szenarien entwickelt wurde. Diese Modelle sind jedoch primär für Klassifizierungsprobleme konzipiert, bei denen zahlreiche zusätzliche, nicht zu klassi-

fizierende Klassen für das Training zur Verfügung stehen. Zwar besteht die Möglichkeit, einen Teil der Trainingsdaten zur Anpassung des Feature-Extractors und den anderen Teil als Support-Set zu verwenden, doch ist diese Vorgehensweise unkonventionell. In der Praxis führt sie zu einer geringeren Erkennungsgenauigkeit im Vergleich zu klassischen CNNs.

Tabelle 4.5: Vergleich der Klassifikationsmethoden im Kontext der NVE-Erkennung

Aspekt	**Feature Extraction**	**MLP**	**CNN**	**Prototypical Network**
Genauigkeit [Prio. 1]	+	++	+++	+
Datenbedarf [Prio. 2]	++	+++	++	++
Geschwindigkeit [Prio. 3]	+++	+	++	+
Implementierungskomplexität [Prio. 4]	++	+++	+++	++

Die vorhergehende Analyse führt zu dem Schluss, dass das CNN den optimalsten Kandidat für die Implementierung der Ziffernklassifizierung darstellt. Es erfüllt die priorisierten Kriterien in optimaler Weise. Es bietet die höchste Genauigkeit im Vergleich zu den anderen Methoden. Der Datenbedarf ist dabei moderat, und die Verarbeitungsgeschwindigkeit liegt im akzeptablen Bereich. Die Implementierungskomplexität ist überschaubar und ermöglicht eine relativ einfache Integration in das Gesamtsystem. Das CNN stellt somit die geeignetste Methode aus der Liste.

4.6 Auswahl des Labeltools

Die Entwicklung von Modellen zur Objekterkennung erfordert die Durchführung eines spezifischen Trainings, welches auf Basis annotierter Daten erfolgt. Das Ziel besteht darin, dem Modell die Fähigkeit zur präzisen Lokalisierung der Ziffern beizubringen, während die exakte Klassifizierung in dieser Phase von untergeordneter Bedeutung ist. Für die Erstellung der Annotationen ist ein geeignetes Labeling-Tool auszuwählen. Die Wahl des Tools hat dabei keinen direkten Einfluss auf die Erkennungsgenauigkeit des Modells, jedoch ist die Qualität der erstellten Annotationen entscheidend. Daher ist es von Vorteil, ein benutzerfreundliches Tool auszuwählen. Dies muss unter anderem die benötigten Dateiformate für das jeweilige Modell unterstützt.

Für das Training von YOLO wird in der Regel das YOLO-spezifische Format verwendet. Jedes annotierte Objekt wird bei diesem Format durch eine Zeile beschrieben, die die Klasse und die Position des Objekts relativ zur Bildgröße definiert. Dabei werden keine absoluten Pixelwerte verwendet, sondern alle Angaben sind normiert und liegen zwischen "0" und "1". In der Tabelle 4.6 werden verschiedene Tools zur Bildannotation miteinander verglichen.

Tabelle 4.6: Vergleich der Tools zur Bildannotation für Objekterkennung [12] [54]

Aspekt	**LabelImg**	**LabelMe**	**RectLabel**	**CVAT**
Benutzerfreundlichkeit	+++	++	++	+++
Unterstützte Formate	YOLO, Pascal VOC	YOLO, COCO, Pascal VOC	YOLO, Pascal VOC	YOLO, COCO, Pascal VOC
Anpassungsmöglichkeiten	+	++	++	+++
Plattformkompatibilität	Windows, macOS, Linux	Web-basiert	macOS	Web-basiert, Docker, Cloud
Kosten	Kostenfrei	Kostenfrei	Kosten-pflichtig	Kostenfrei

Nach dem Vergleich erweist sich CVAT als die geeignetste Option für die Annotation. Es zeichnet sich durch eine hohe Benutzerfreundlichkeit aus und unterstützt verschiedene Dateiformate. Zudem ermöglicht die webbasierte Plattform eine plattformunabhängige Nutzung, wodurch die Flexibilität und Zugänglichkeit der Annotation erheblich verbessert werden.

4.7 Strategien zur Optimierung der Dateneffizienz

Die begrenzte Verfügbarkeit von annotierten Trainingsdaten erfordert den Einsatz verschiedener Strategien zur Optimierung der Dateneffizienz, wie etwa Transfer Learning oder Data Augmentation. Ein vortrainiertes CNN kann dabei als Ausgangspunkt dienen, da es bereits in der Lage ist, allgemeine Merkmale wie Linien und Kanten zu extrahieren. Dadurch verringert sich in der Regel der Bedarf an Trainingsdaten.

Zur künstlichen Vergrößerung des Trainingsdatensatzes und zur Erhöhung der Modellrobustheit können zusätzlich die im Grundlagenkapitel beschriebenen Datenaugmentation-Techniken angewendet werden. In der Evaluierung wird dabei analysiert, ob diese Methoden Vorteile hinsichtlich Genauigkeit für diesen spezifischen Anwendungsfall bieten.

4.8 Beschreibung des neuen Systems

Die ausgewählte Kombination für das neue System besteht aus YOLOv9 und einem CNN. In der ersten Stufe wird YOLOv9 zur Lokalisierung der Ziffern im Bild eingesetzt. Zur Eliminierung redundanter Detektionen kommt ein NMS-Algorithmus zum Einsatz, dessen Funktionsweise im Kapitel 5 näher erläutert wird.
Anschließend werden die Bounding Boxes gefiltert, um sicherzustellen, dass nur NVE-Ziffern an das Klassifikationsmodell weitergeleitet werden. Die von den verbleibenden Bounding Boxes umschlossenen Bereiche werden extrahiert und einzeln in die zweite Stufe zugeführt.
In der zweiten Stufe führt das CNN die Klassifizierung durch. Die einzelnen Ergebnisse des CNN-Modells werden zusammengeführt. Die resultierenden Ziffern werden, wie in der bestehenden Lösung, durch Heuristik korrigiert und validiert. Abschließend gibt das System eine Zeichenkette aus, die die NVE enthält (siehe Abb. 4.1).

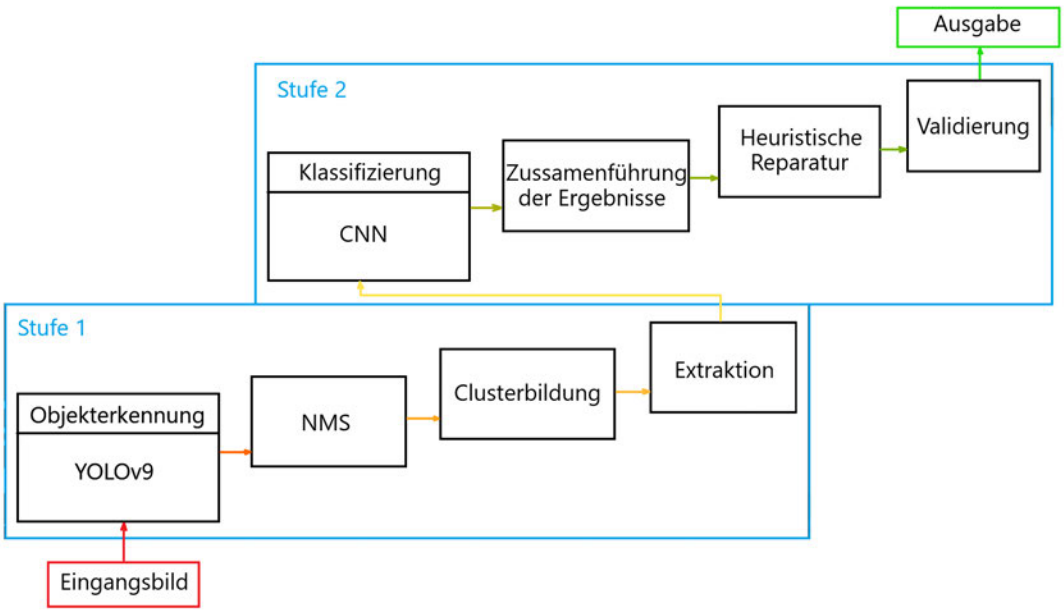


Abbildung 4.1: Gesamtkonzept des neuen OCR-Systems (eigene Darstellung)

Um den Entwicklungsprozess zu optimieren und den Implementierungsaufwand zu minimieren, wird das neue System zunächst unabhängig vom bestehenden Gesamtsystem entwickelt. Diese Strategie ermöglicht eine fokussierte Entwicklung und Evaluierung. Die Integration in die bestehende Lösung kann nach einer gründlichen Evaluierung und Selektion der Modellarchitektur erfolgen.

4.9 Konzept der Evaluierung

Die Evaluierung des Systems zielt auf eine quantitative Bewertung der Leistungsfähigkeit der Modellkombination ab. Wie im Abschnitt 3.3 dargelegt, stehen für diese Arbeit 250 aussortierte Bildaufnahmen zur Verfügung, wovon 150 Bilder für den Trainingsprozess und 100 Bilder für die Evaluierung vorgesehen sind. Obwohl im Rahmen dieser Arbeit verschiedene Szenarien hinsichtlich der Datenmenge untersucht werden, bleibt für einen repräsentativen Vergleich die Anzahl der Testbilder konstant bei 100.
Es sollen dabei folgende Szenarien untersucht werden (siehe Tabelle 4.7).

Tabelle 4.7: Zu untersuchende Testszenarien

Trainings- und Validierungsdaten	Testdaten
25	100
50	100
100	100
150	100

Das Ziel ist es, herauszufinden, ob das Konzept auch in einer datenarmen Umgebung eine bessere Genauigkeit erzielen kann. Besondere Aufmerksamkeit gilt dem Szenario mit 50 Bildern, da das neue System gemäß den Anforderungen mindestens eine NVE-Erkennungsrate von 82% bei 50 Trainingsbildern erreichen muss.
Wie bereits im Abschnitt 3.3 erläutert, erfordert die Trainingsphase neuronaler Netze den Einsatz von Validierungsdaten. Diese spielen eine wesentliche Rolle bei der Beurteilung der Modellleistung und der Auswahl optimaler Gewichtungen. Folglich ist es notwendig, einen Teil des ursprünglichen Trainingsdatensatzes für Validierungszwecke zu reservieren. Die entsprechende Aufteilung ist der Tabelle 4.8 zu entnehmen.

Tabelle 4.8: Aufteilung von Trainings- und Validierungsdaten

Trainings- und Validierungsdaten	Training	Validierung
30	20	10
50	35	15
100	75	25
150	120	30

Die Evaluierung gliedert sich für den entsprechenden Szenarien in drei Phasen:

1. Bewertung des YOLOv9-Modells .
2. Bewertung des CNN-Modells.
3. Vergleich zwischen EasyOCR und der entwickelten OCR-Lösungen hinsichtlich der NVE-Erkennungsrate (vgl. Abschnitt 3.3).

Die durchzuführende Evaluierung dient der objektiven Beurteilung, inwieweit die implementierte Lösung den definierten Projektanforderungen entspricht.

5 Entwicklung und Implementierung

In diesem Kapitel wird die praktische Umsetzung des konzipierten Systems beschrieben. Es umfasst die Datenvorbereitung, das Training der Modelle, deren Implementierung sowie die Nachbearbeitungsprozesse.

5.1 Vorbereitung der Daten

Die Vorbereitungsphase beinhaltet die Aufteilung der verfügbaren Daten in verschiedene Szenarien, wie in der Tabelle 4.7 dargestellt. Der Testdatensatz wurde bereits im Abschnitt 3.3 definiert. Für die Aufteilung der verbleibenden Daten wird ein Python-Skript entwickelt, das gemäß Tabelle 4.8 arbeitet. Dieses Skript führt eine zufällige, stufenweise Aufteilung des 150 Bilder umfassenden Datensatzes in vier unterschiedlichen Szenarien durch. Der Prozess ist so konzipiert, dass in jedem Schritt eine Teilmenge der nächstgrößeren Datenmenge generiert wird, wodurch die für das Training und die Validierung verfügbare Datenmenge in den aufeinanderfolgenden Szenarien sukzessive abnimmt. Dies ermöglicht eine repräsentative Vergleichbarkeit der unterschiedlichen Datengrößen.
Für das Training des YOLOv9-Modells werden alle Daten mit dem Labeltool-CVAT annotiert. Dies stellt den aufwendigsten Schritt in der Datenvorbereitung dar, da jede einzelne NVE-Ziffer manuell mit einer Groundtruth-Box und der entsprechenden Klasse versehen wird. Die Klassenbezeichnungen entsprechen dabei den Ziffern selbst (siehe Abb. 5.1).

Abbildung 5.1: Manuelle Annotation von NVE-Ziffern mit CVAT, wobei jede Ziffer durch eine Groundtruth-Box markiert ist (eigene Darstellung)

Am Ende des Annotationsprozesses wird für jedes Bild eine Textdatei erstellt. Diese Dateien enthalten die genauen Koordinaten der Groundtruth-Boxes und die zugehörigen Klassen für jede markierte NVE-Ziffer. Ein Beispiel einer solchen Textdatei ist im Anhang C zu finden. Die so erstelle Textdateien werden in Kombination mit Bildern für den Trainingsprozess der ersten Stufe verwendet.
Die vorliegenden Daten eignen sich in ihrer ursprünglichen Form nicht für das Training von CNN. So wie es konzipiert ist erfordern diese Modelle als Eingabe einzelne Ziffern samt ihrer Klassenbezeichnung. Zur Lösung dieses Problems wird ein zusätzliches Skript erstellt, welches die annotierten Daten durchläuft und die in den Groundtruth-Boxes enthaltenen Ziffern als separate Bilddateien extrahiert. Diese Ziffern werden anschließend anhand der Annotationen den entsprechenden Klassen zugeordnet. Die Klassenbezeichnungen werden durch die Ordnerstruktur abgebildet, wobei jede Ziffer in den dazugehörigen Klassenordner verschoben wird (siehe Abb. 5.2).

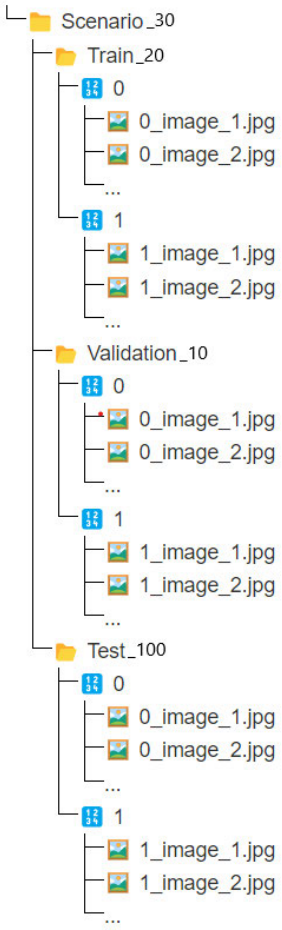


Abbildung 5.2: Datensatzstruktur für die Ziffernerkennung (eigene Darstellung).

Dies ermöglicht eine strukturierte Datenbasis, die sowohl für den Trainingprozess als auch für Evaluierung des CNN-Modells verwendet wird.
Angesichts des begrenzten Datenumfangs, der durch die Aufteilung in Trainings- und Validierungsdatensätze zusätzlich reduziert wird, erweist sich die Anwendung von Datentransformationen als zweckmäßig. Zu diesem Zweck wird die in PyTorch implementierte Funktion `transforms.Compose` eingesetzt. Diese Funktion wird sowohl für Transformationen wie Normalisierung, Skalierung und Umwandlung der Bilder in ein für CNN kompatibles Format, nämlich Tensoren[1], verwendet als auch für Data Augmentation.

[1] Tensoren sind multidimensionale Datenstrukturen, die in neuronalen Netzen zur effizienten Verarbeitung und Darstellung von Eingaben, Gewichten und Aktivierungen genutzt werden. Sie ermöglichen die Berechnung komplexer Operationen in verschiedenen Dimensionen.

Die Vielseitigkeit dieser Funktion zeigt sich auch in ihrer Fähigkeit, sowohl vordefinierte als auch benutzerdefinierte Transformationen zu integrieren.
Zum Zweck der Data Augmentation werden folgende Transformationen auf das CNN-Modell angewendet:

- `transforms.RandomGrayscale(p=0.2)`: Wandelt die Eingabebilder mit einer Wahrscheinlichkeit von 20 % in Graustufen um. Dies hilft, Farbabhängigkeiten im Modell zu reduzieren.
- `transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)`: Passt Helligkeit, Kontrast, Sättigung und Farbton der Bilder zufällig an, um das Modell gegenüber unterschiedlichen Beleuchtungsbedingungen widerstandsfähiger zu machen.
- `transforms.GaussianBlur(kernel_size=5, sigma=(0.1, 2.0))`: Wendet einen Gaußschen Weichzeichner an, um Unschärfe im Bild zu simulieren. Dies trainiert das Modell auf unscharfen oder weniger detailreichen Bildern.
- `ElasticTransform(alpha=74, sigma=3)`: Eine benutzerdefinierte elastische Verzerrung, die geometrische Veränderungen im Bild einführt, um das Modell auf Verzerrungen vorzubereiten.
- `CustomShiftTransform(shifts=[-1, 0, 1])`: Eine benutzerdefinierte Transformation, die das Bild in eine zufällige Richtung verschiebt, um das Modell auf Objekte vorzubereiten, die nicht zentral im Bild platziert sind.
- `transforms.RandomRotation(10)`: Rotiert das Bild um bis zu 10 Grad, um das Modell auf variierende Objektorientierungen zu trainieren.

Diese Transformationen werden sequentiell auf alle Trainingsbilder angewendet, wobei die Parameter der Transformationen experimentell festgelegt sind.
Die in Abbildung 5.3 gezeigtes Beispiel veranschaulicht den Einfluss der oben genannten Transformationen auf ein Eingabebild.

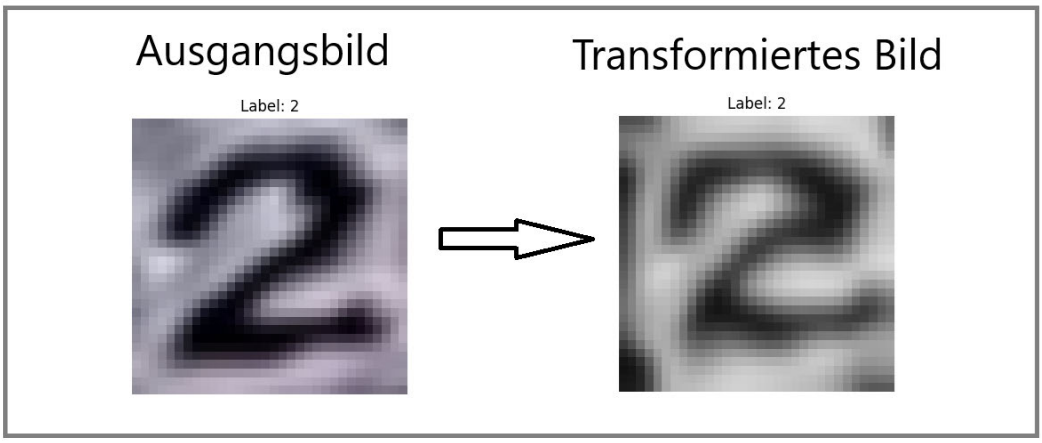


Abbildung 5.3: Vergleich eines Ausgangsbildes mit einer durch Transformation erzeugten Variante (eigene Darstellung)

Während des Trainingsprozesses wird für jedes Bild in jeder Epoche[2] eine transformierte Version generiert. Die Gesamtmenge der neu erzeugten Daten hängt von der Anzahl der Trainingsdaten und der Epochen ab. Der Zufallsfaktor in den Transformationen stellt dabei sicher, dass in jeder Epoche unterschiedliche Bildvarianten entstehen, was Overfitting minimiert. Die Abbildung 5.4 verdeutlicht den Prozess der Bildtransformation über mehrere Epochen. Die fünf Originalbilder werden in jeder Epoche durch Transformationen modifiziert, wodurch pro Epoche fünf zusätzliche Varianten entstehen. Nach drei Epochen werden insgesamt 15 transformierte Bildvarianten generiert.

[2]Eine Epoche ist ein vollständiger Durchlauf durch den gesamten Trainingsdatensatz, wobei das Modell seine Parameter anpasst; in der Regel wird das Modell über mehrere Epochen hinweg trainiert, um seine Leistung zu optimieren.

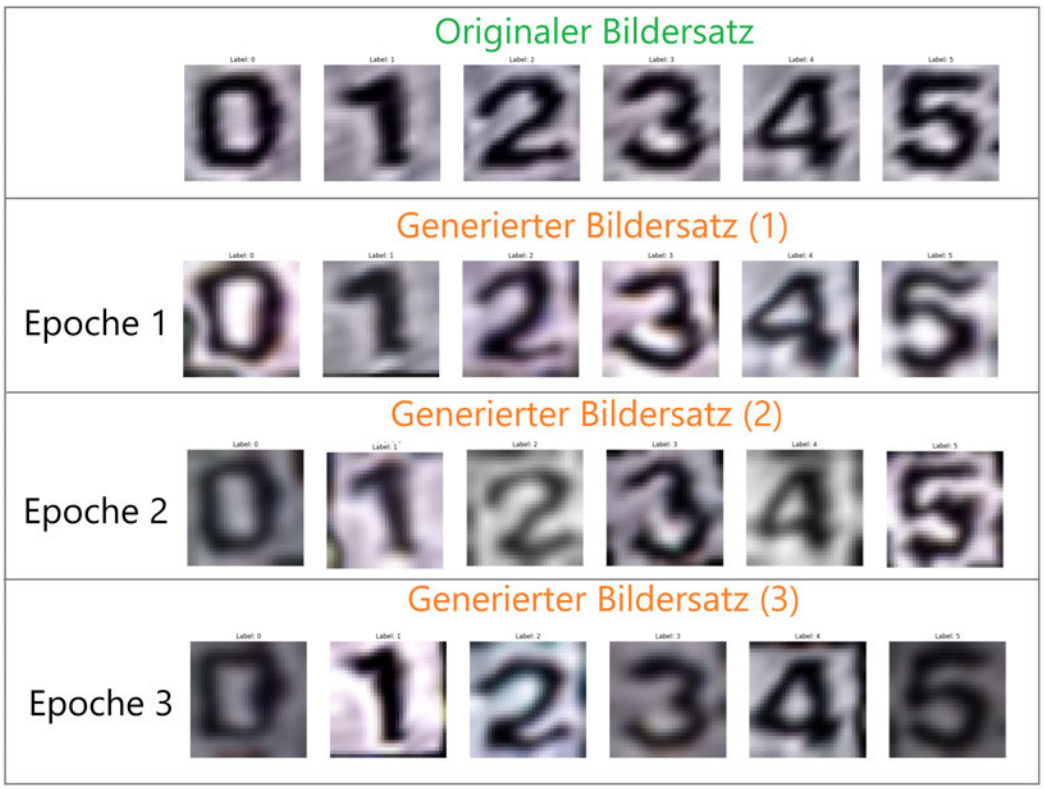


Abbildung 5.4: Originaler Bildersatz und durch Transformationen generierte Bildvarianten über drei Epochen (eigene Darstellung)

Auch für das YOLOv9-Modell kommen Transformationen zum Einsatz, allerdings mit der Einschränkung, dass Ultralytics keine benutzerdefinierten Transformationen unterstützt. Daher ist man hier auf die vorhandenen Optionen angewiesen.

5.2 Umsetzung der ersten Stufe

Wie bereits im konzeptionellen Teil erläutert, setzen sich die Stufen des neuen Systems aus mehreren Komponenten zusammen. Wie der Abbildung 5.5 zu entnehmen ist, besteht die erste Stufe aus Objekterkennung und Nachbearbeitungsprozessen.

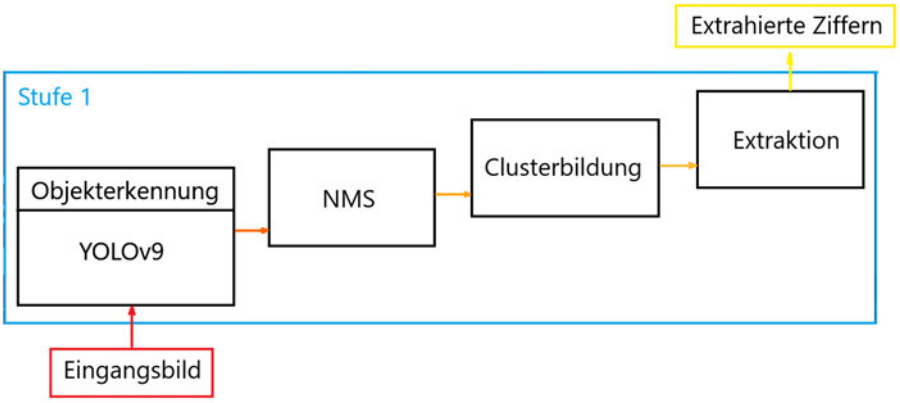


Abbildung 5.5: Aufbau der ersten Stufe (eigene Darstellung)

Das YOLOv9-Modell erfordert jedoch ein Training. Dieser Prozess umfasst die Implementierung und Anpassung des YOLOv9-Modells an die Trainingsdaten. Als Ergebnis resultiert ein Modell, das für die Lokalisierung der NVE-Ziffern eingesetzt wird. Anschließend werden Nachbearbeitungsmechanismen eingesetzt, die auf die vom YOLOv9-Modell generierten Ergebnisse angewandt werden. Als Output dieser ersten Stufe werden 18 extrahierte Ziffern bereitgestellt, welche als Input für das zweite Modell der nachfolgenden Stufe dienen.

5.2.1 Training

Für die Implementierung des YOLO-Modells wird die von Ultralytics entwickelte Version herangezogen, die in das PyTorch-Framework integriert ist. Nach der Erstellung der notwendigen Konfigurationsdateien zur Definition der Datenstruktur und Verzeichnisse gestaltet sich der Trainingsprozess unkompliziert. Dem Modell werden lediglich die Konfigurationsdatei und die Anzahl der Epochen übergeben.
Vorverarbeitungsschritte wie Skalierung, Normalisierung und Transformationen sowie die den Lernprozess steuernden Modellparameter sind standardmäßig festgelegt. Es besteht jedoch die Option, viele dieser Parameter anzupassen. Eine vollständige Auflistung der

vordefinierten Parameter findet sich im Anhang D.
Der Trainingsprozess beginnt mit dem Laden der Trainingsdaten anhand der in einer config.yaml-Datei spezifizierten Pfade und Klassendefinitionen. Dies umfasst das Einlesen der Bilder und zugehörigen Labels. Um die Genauigkeit zu verbessern, wird das Modell mit Gewichten eines vortrainierten Modells initialisiert. Diese Gewichte werden während des Trainings mittels Backpropagation basierend auf einer Verlustfunktion angepasst. Die Verlustfunktion setzt sich aus mehreren Komponenten zusammen:

- Bounding Box Loss: Beschreibt die Differenz zwischen den vorhergesagten und tatsächlichen Positionen der Bounding Boxes.
- Classification Loss: Beschreibt die Abweichung zwischen der vorhergesagten und der tatsächlichen Objektklasse.
- Objectness Loss: Bemisst den Fehler bei der Einschätzung der Objektpräsenz innerhalb der Bounding Box.

Standardmäßig kommen während des Trainings diverse Mechanismen zur Optimierung des Lernprozesses zum Einsatz:

- Adam-Optimierer: Ermöglicht eine dynamische Anpassung der Lernrate im Trainingsverlauf, was die Konvergenz verbessert.
- Early Stopping: Initiiert einen automatischen Trainingsabbruch bei Ausbleiben weiterer Verbesserungen.
- L2-Regularisierung: Dient der Prävention von Überanpassung durch Bestrafung übermäßig großer Gewichte.

Zur Evaluierung der Generalisierungsfähigkeit auf unbekannte Daten erfolgt nach dem Training eine Überprüfung des Modells anhand der Validierungsdaten. Im Anschluss werden die Modellgewichte mit der besten Validierungsleistung gespeichert. [3]

5.2.2 Inferenz

Die Implementierung der Inferenz erfolgt ebenfalls in einfacher Weise. Das trainierte Modell wird geladen und mit zwei wesentlichen Argumenten aufgerufen: dem zu analysierenden Bild und dem Confidence-Threshold. Der Confidence-Threshold bestimmt den Schwellenwert der Modellzuversicht bzw. Confidence Score, ab dem ein erkanntes Objekt als gültig betrachtet wird. Objekte, deren Confidence Score unter diesem Schwellenwert

liegt, werden nicht ausgegeben. [38]
Das Modell gibt für jedes erkannte Objekt eine Bounding Box aus, die durch vier Koordinaten (x, y, Breite, Höhe) beschrieben wird. Zusätzlich werden die zugehörige Objektklasse und der Confidence Score ausgegeben.
Im Kontext des zweistufigen Ansatzes ist die Klassifizierungsgenauigkeit des YOLO-Modells zweitrangig. Der Fokus liegt vielmehr auf die Vorhersage von Bounding Boxes. Um zu verhindern, dass das Objekterkennungsmodell zum limitierenden Faktor im System wird, muss sichergestellt werden, dass möglichst für alle NVE-Ziffern Bounding Boxes ausgegeben werden, auch wenn dabei die Klassifizierungszuversicht gering ist. Aus diesem Grund wird der Confidence-Threshold reduziert, wodurch die Empfindlichkeit des Modells bei der Bounding-Box-Vorhersage erhöht wird. In der Abbildung 5.6 ist der Einfluss niedriger und hoher Confidence-Threshold-Werte auf die Ausgabe der Bounding Boxes dargestellt.

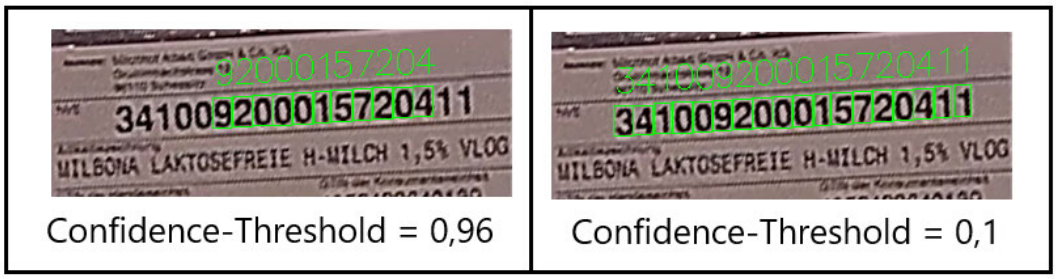


Abbildung 5.6: Einfluss des Confidence-Thresholds auf die Erkennung von NVE (eigene Darstellung)

5.2.3 Non-Maximum Suppression (NMS)

Wie im Grundlagenkapitel erläutert, wird bei YOLO-Modellen der NMS-Algorithmus eingesetzt, um redundante Detektionen zu eliminieren. Der Standard-NMS-Algorithmus funktioniert jedoch klassenbasiert. Dies bedeutet, dass NMS ausschließlich auf Bounding Boxes angewendet wird, die derselben Klasse zugeordnet sind [38]. Überlappende Bounding Boxes mit unterschiedlichen Klassenvorhersagen werden nicht eliminiert. Obwohl dieser Ansatz auf den ersten Blick plausibel erscheint, erweist er sich im Kontext des vorliegenden Anwendungsfalls als problematisch.
Die erhöhte Empfindlichkeit des Modells führt zur Ausgabe zahlreicher irrelevanter

Bounding Boxes, die teilweise mit den relevanten überlappen. Da diese überlappenden Bounding Boxes nicht immer derselben Klasse zugeordnet werden - etwa wenn das Modell in einem Bereich nahe einer Ziffer fälschlicherweise eine andere Ziffer erkennt - eliminiert der standardmäßig implementierte NMS-Algorithmus diese Bounding Boxes trotz Anpassungen des IoU-Thresholds nicht. Dies lässt sich deutlich in Abbildung 5.7 beobachten.

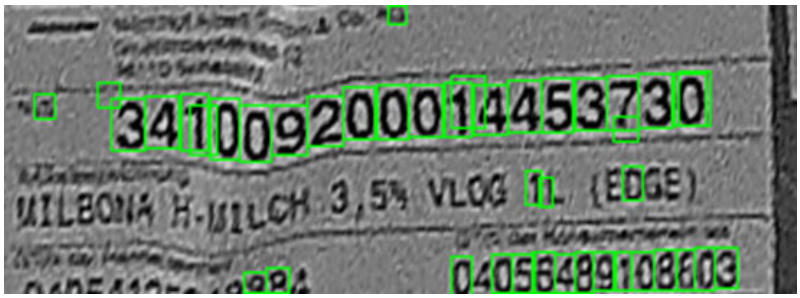


Abbildung 5.7: Ausgabe irrelevanter und überlappender Bounding Boxes aufgrund erhöhter Empfindlichkeit des Modells (eigene Darstellung)

Aufgrund dieser Problematik wird ein benutzerdefinierter NMS-Algorithmus implementiert, der klassenunabhängig arbeitet. Im Gegensatz zum klassenbasierten NMS, das nur Bounding Boxen derselben Klasse vergleicht, berücksichtigt der klassenunabhängige Ansatz lediglich die räumliche Überlappung der Boxen, unabhängig von der Klassenzuordnung.
Die Funktionsweise des klassenunabhängigen NMS-Algorithmus basiert auf der Berechnung der IoU zwischen jeder Bounding Box. Zunächst wird die Bounding Box mit dem höchsten Confidence Score ausgewählt und beibehalten. Anschließend wird die IoU zwischen dieser Box und allen verbleibenden Boxen berechnet. Boudning Boxes, deren IoU-Wert eine festgelegte Schwelle überschreitet, werden verworfen, da sie als Duplikate der priorisierten Box betrachtet werden. Im nächsten Schritt wird die Box mit dem nächsthöchsten Confidence Score ausgewählt und der Prozess wiederholt. Dieser iterative Prozess wird fortgesetzt, bis alle relevanten Boxen überprüft wurden, wodurch am Ende nur die Bounding Boxes mit der höchsten Zuversicht und minimaler Überlappung erhalten bleiben. Die Abbildung 5.8 veranschaulicht den Effekt des klassenunabhängigen NMS-Algorithmus.

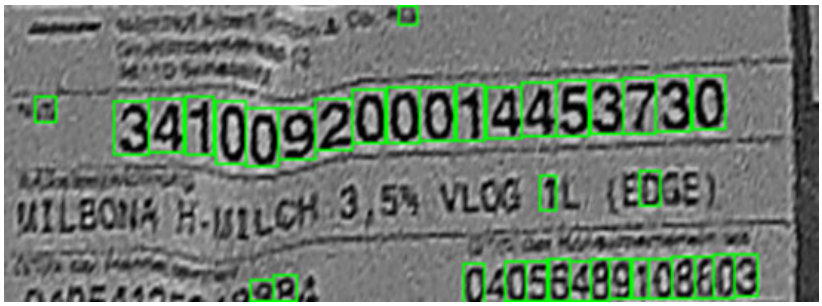


Abbildung 5.8: Filterung irrelevanter und überlappender Bounding Boxes (eigene Darstellung)

5.2.4 Clusterbildung

Wie aus der Abbildung 5.8 ersichtlich wird, werden nach Anwendung des NMS-Algorithmus überlappende Bounding Boxes erfolgreich eliminiert. Dennoch verbleiben weiterhin für die Erkennung irrelevante Bounding Boxes. Um zu gewährleisten, dass ausschließlich die NVE-Ziffern ausgegeben werden, ist eine zusätzliche Filterung erforderlich.
Hierfür wird der DBSCAN (Density-Based Spatial Clustering of Applications with Noise)-Algorithmus aus der sklearn-Bibliothek verwendet. DBSCAN ist ein Clustering-Verfahren, das Datenpunkte basierend auf ihrer Dichte in Gruppen bzw. Cluster zusammenfasst. In diesem Fall werden die Bounding Boxes anhand ihrer Mittelpunkte gruppiert.
Zunächst wird eine Funktion implementiert, die für jede Bounding Box die Koordinaten des Mittelpunkts berechnet, welche dann als Grundlage für die Clusterbildung dienen. Anschließend wird der DBSCAN-Algorithmus auf diese berechneten Zentren angewendet. Der Algorithmus verwendet zwei wesentliche Parameter. Der ε-Parameter definiert den Radius, innerhalb dessen Punkte als benachbart gelten. Der MinPts-Parameter bestimmt die Mindestanzahl an Punkten, die innerhalb dieses Radius liegen müssen, damit ein Punkt als Kernpunkt eines Clusters klassifiziert wird. Kernpunkte fungieren als Zentren von Clustern. Punkte mit einer geringeren Nachbaranzahl werden entweder als Rauschen eingestuft oder als Randpunkte eines Clusters betrachtet. Randpunkte, obgleich selbst keine Kernpunkte, gehören zum Cluster, sofern sie innerhalb des festgelegten Radius von einem Kernpunkt erreichbar sind. Punkte, die weder ausreichend Nachbarn aufweisen noch von einem Kernpunkt erreichbar sind, werden als Rauschen kategorisiert und bei der Clusterbildung unberücksichtigt gelassen. [15]
Zur Bestimmung eines optimalen ε-Parameters wird ein Python-Skript implementiert.

Dieses ermittelt den maximalen Pixelabstand zwischen benachbarten NVE-Boxes in den annotierten Trainingsbeispielen. Basierend auf dieser Analyse wird ein ε-Parameter von 23 Pixeln festgelegt, der zusätzlich eine Sicherheitsreserve beinhaltet. Dies gewährleistet, dass auch Ziffern, die aufgrund von Verformungen des Etiketts weiter voneinander entfernt liegen, dennoch dem NVE-Cluster zugeordnet werden
Der MinPts-Parameter wird auf zwei festgelegt, wodurch die mittleren 16 NVE-Ziffern als Kernpunkte und die äußeren zwei als Randpunkte gelten, was der typischen Struktur einer NVE entspricht. Die durch DBSCAN gebildeten Cluster und die als Rauschen identifizierten Bounding Boxen sind in Abbildung 5.9 zu sehen.

Abbildung 5.9: Clusterbildung durch DBSCAN (eigene Darstellung)

Nach der Clusterbildung wird das NVE-Cluster identifiziert, indem das größte Cluster (in Abbildung 5.9 rot markiert) ausgegeben wird. Dies stellt in diesem Fall eine geeignete Herangehensweise dar, da die NVE stets die größte zusammenhängende Folge von Ziffern bildet. Auf diese Weise werden irrelevante Bounding Boxes aussortiert, sodass nur die NVE-Ziffern übrig bleiben. Dieser Prozess wird anschaulich in Abbildung 5.10 illustriert

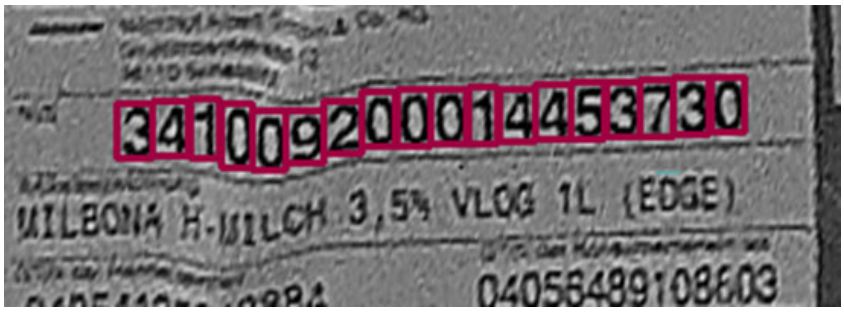


Abbildung 5.10: Filterung der Ergebnisse durch Auswahl des größten Clusters (eigene Darstellung)

5.2.5 Extraktion

Nach Abschluss des Filterungsprozesses erfolgt eine Sortierung der Ziffern entsprechend ihrer räumlichen Position im Bild. Dafür werden die entsprechenden Bounding Boxes nach ihrer x-Koordinate sortiert. Im Anschluss erfolgt die Extraktion der NVE-Ziffern. Dieser Schritt umfasst das Ausschneiden und Skalieren der von den Bounding Boxes umschlossenen Bildbereiche (siehe Abb. 5.11).

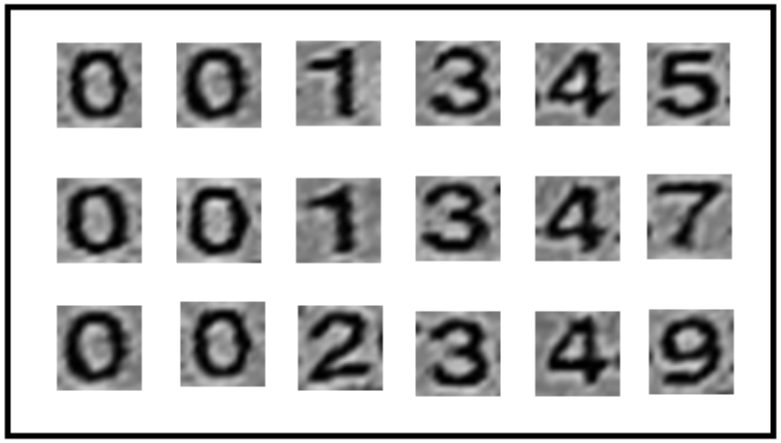


Abbildung 5.11: Extrahierte NVE-Ziffern (eigene Darstellung)

Die extrahierten Ziffern werden temporär gespeichert und anschließend für die Klassifizierung verwendet.

5.3 Umsetzung der zweiten Stufe

Die zweite Stufe des Systems baut auf der ersten Stufe auf und umfasst weitere wichtige Komponente (siehe Abb. 5.12).

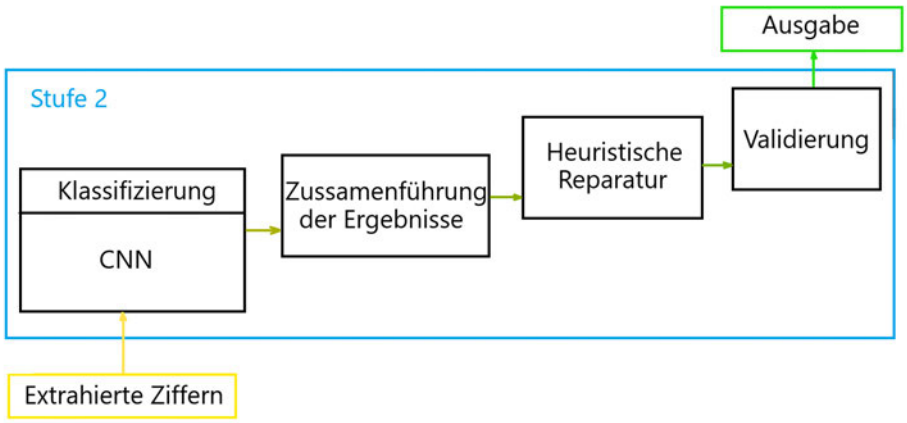


Abbildung 5.12: Aufbau der zweiten Stufe (eigene Darstellung)

Ähnlich wie das Objekterkennungsmodell muss auch das CNN-Modell vor seinem Einsatz zur Inferenz trainiert werden. Die Erkennung der NVE erfolgt sequenziell, wobei die NVE-Ziffern nacheinander dem Modell zugeführt und anschließend zusammengeführt werden. Im Anschluss werden im bestehenden System bereits implementierte Funktionen zur heuristischen Reparatur und Validierung eingesetzt.

5.3.1 Training

In dieser Arbeit werden die Netzwerkarchitekturen VGG16 und LeNet-5 verwendet. Während VGG16 im PyTorch-Framework vorimplementiert ist und direkt eingesetzt werden kann, muss die Architektur von LeNet-5 manuell definiert werden.
VGG16 und LeNet-5 unterscheiden sich erheblich in ihrer Architektur und Parameteranzahl. VGG16 akzeptiert Eingaben in einer Größe von 224x224 Pixeln mit drei Farbkanälen (RGB). Die Architektur besteht aus insgesamt 16 gewichteten Schichten, darunter 13 Convolutional-Schichten und 3 Fully-Connected-Schichten. Die Convolutional-Schichten verwenden 3x3 Filter. Nach jeweils zwei oder drei Convolutional-Schichten folgt eine Max-Pooling-Schicht, die die Auflösung der Feature Maps um die Hälfte reduziert. Am Ende des Netzwerks befinden sich drei vollständig verbundene Schichten: die erste mit

4096 Neuronen, die zweite ebenfalls mit 4096 Neuronen und die letzte Schicht mit 1000 Neuronen für die Klassifikation im ursprünglichen ImageNet-Datensatz (1000 Klassen). Für die spezifische Aufgabenstellung in dieser Arbeit wird die letzte Schicht an die Anzahl der zu klassifizierenden Klassen angepasst. Mit seiner tiefen Architektur ist VGG16 primär für komplexe Bildklassifizierungsaufgaben und große Datenmengen geeignet. Bei kleineren Datensätzen kann die Tiefe der Architektur jedoch zum Overfitting führen. Um auch bei kleinen Datensätzen eine gute Generalisierbarkeit zu erreichen, sind zusätzliche Anpassungen an der Modellstruktur des Netzes erforderlich. [46] Die Struktur der VGG16-Architektur ist in der Abbildung 5.13 dargestellt.

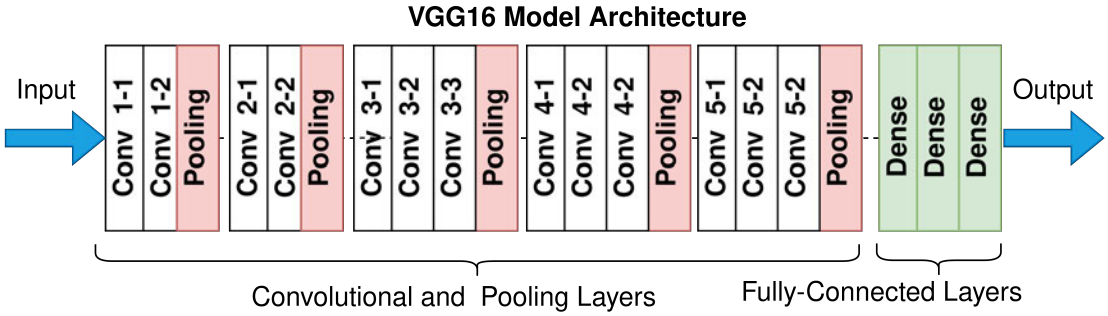


Abbildung 5.13: Aufbau der VGG16-Architektur [36]

LeNet-5 besteht aus 5 gewichteten Schichten, darunter zwei Convolutional-Schichten, zwei vollständig verbundene Schichten und eine Output-Schicht. Es akzeptiert Eingaben mit einer Größe von 32x32 Pixeln in Graustufen. Die Convolutional-Schichten verwenden 5x5 Filter und sind zur reduzierung des Feature Maps jeweils mit einer Pooling-Schicht verbunden. Nach den beiden Convolutional-Schichten folgen zwei vollständig verbundene Schichten mit 120 bzw. 84 Neuronen. Die letzte Schicht dient der Klassifikation und besteht ursprünglich aus 10 Neuronen. Obwohl LeNet-5 aufgrund seiner einfacheren Netzwerkarchitektur bei komplexeren Aufgaben nicht die Genauigkeit von VGG16 erreicht, liefert es in datenarmen Umgebungen dennoch solide Ergebnisse. [26]
Die Struktur der LeNet-5-Architektur ist in Abbildung 5.14 dargestellt.

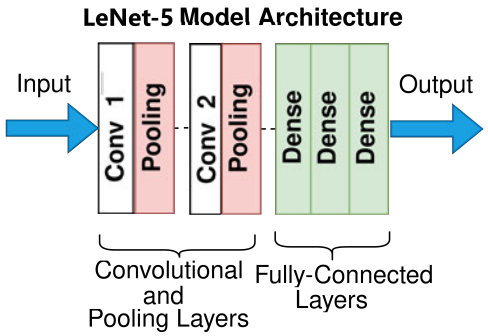


Abbildung 5.14: Aufbau der LeNet-5-Architektur (veränderte Darstellung) [36]

Das VGG16-Modell wird für vergleichende Zwecke in zwei Varianten implementiert: In der ersten Variante wird ein vortrainiertes Modell verwendet, bei dem die vorderen Schichten eingefroren und nur die hinteren Schichten neu trainiert werden. In der zweiten Variante wird das Modell vollständig von Grund auf neu trainiert. Das LeNet-5-Modell wird nur von Grund auf implementiert, da keine vortrainierte Version verfügbar ist.
Der Klassifikator des VGG16-Netzes wird an das Klassifizierungsproblem mit 10 Ziffernklassen (Ziffern “0” bis “9”) angepasst. Dafür werden die hinteren Schichten modifiziert (Dense Schcihten in der Abbildung 5.13). In der neuen Struktur werden ReLU-Aktivierungsfunktionen eingeführt sowie Dropout-Mechanismen zur Regularisierung verwendet. Der Dropout-Ansatz wirkt dem Overfitting entgegen, indem während des Trainings zufällig ausgewählte Neuronen in den vollverknüpften Schichten deaktiviert werden. Dies reduziert die Abhängigkeit des Modells von einzelnen Neuronen, da das Netzwerk gezwungen ist, robustere Merkmalsrepräsentationen zu lernen, was die Generalisierungsfähigkeit verbessert.
In beiden Modellen wird als Verlustfunktion die Cross-Entropy Loss verwendet, welche eine Softmax-Aktivierung mit dem negativen Log-Likelihood-Verlust für Mehrklassen-Klassifikationsprobleme kombiniert. Für das Training kommt der RMSprop-Optimierer zum Einsatz. RMSprop basiert auf dem Gradientenabstiegsverfahren und bietet durch die Berücksichtigung der Schwankungen der Gradienten eine adaptive Anpassung der Lernrate.
Die Genauigkeit der Modelle wird nach jeder Epoche während des Trainings sowohl auf den Trainings- als auch auf den Validierungsdaten berechnet. Dabei wird für jede Vorhersage die Klasse mit der höchsten Output-Wahrscheinlichkeit gewählt und mit

den tatsächlichen Labels verglichen. Die Gewichte des Modells werden im Anschluss basierend auf der höchsten Validierungsgenauigkeit ausgewählt und gespeichert.

5.3.2 Inferenz

Für die Inferenz werden die Modelle in ähnlicher Weise wie bei YOLOv9 geladen. Beim LeNet-5-Modell ist eine explizite Definition der Netzwerkarchitektur erforderlich, da es sich um ein benutzerdefiniertes Modell handelt. Im Gegensatz dazu ist VGG16 bereits in PyTorch implementiert, wodurch eine erneute Definition der Architektur entfällt.
Vor der Durchführung der Inferenz werden die Modelle in den Evaluierungsmodus versetzt. Die Vorhersage der Ziffern erfolgt durch einen einfachen Modellaufruf, bei dem das zu klassifizierende Bild als Argument übergeben wird. Dabei durchläuft das Bild zunächst die gleiche Vorverarbeitung wie die Validierungs- und Testdaten während der Trainingsphase. Dieser Prozess umfasst die Skalierung, Normalisierung und Umwandlung des Bildes in einen Tensor.
Nach Abschluss der Inferenz generiert das Modell für jede mögliche Klasse eine Softmax-Wahrscheinlichkeit. Diese Wahrscheinlichkeiten repräsentieren die vom Modell eingeschätzte Zugehörigkeit des Bildes zu den jeweiligen Klassen. Als endgültige Vorhersage wird die Klasse mit der höchsten Wahrscheinlichkeit ausgewählt.

5.3.3 Zusammenführung der Ergebnisse

Nach der Erkennung der einzelnen Ziffern einer NVE werden die Ergebnisse zur vollständigen Nummer zusammengeführt. Das Modell verarbeitet zunächst die extrahierten Ziffern in der Reihenfolge ihrer Position im Bild und sortiert nach der X-Koordinate. Die Vorhersage für jede Ziffer wird gespeichert, wobei die zuvor festgelegte Sortierung die korrekte Reihenfolge sicherstellt. Im nächsten Schritt werden die Vorhersagen der einzelnen Ziffern zu einer vollständigen NVE verbunden. Diese Ziffernfolge stellt die vom Modell erkannte NVE dar.
Bei der Auswertung der Ergebnisse auf Testdaten wird die erkannte NVE mit der erwarteten Nummer verglichen, die im Dateinamen des Bildes kodiert ist. Dieser Vergleich ermöglicht eine Bewertung der Genauigkeit des Modells.

5.3.4 Heuristische Reparatur

Zur Korrektur der Ergebnisse wird eine im bestehenden System bereits implementierte Methode verwendet. Diese überprüft zunächst, ob 18 Ziffern erkannt wurden. Falls dies

zutrifft, werden die ersten 9 Ziffern der Nummer automatisch durch das festgelegte Standardpräfix „341009200“ ersetzt. Dieses Präfix setzt sich aus der Kombination der Reservierungsziffer und der Basisnummer der NVE zusammen (vgl. Abschnitt 2.6). Bei der Erkennung von weniger als 18 Ziffern bleiben diese unverändert. Der Reparaturprozess stellt sicher, dass bekannte, feste Teile der Nummer korrekt sind, selbst wenn bei der Erkennung Fehler auftreten.

5.3.5 Validierung

Nach der Erkennung und heuristischen Korrektur der NVE wird eine Validierung mittels Prüfziffer durchgeführt. Die letzte Ziffer der 18-stelligen NVE dient dabei als Prüfziffer. Der Prozess der Validierung sieht wie folgt aus:

1. Die ersten 17 Ziffern der erkannten NVE werden extrahiert.
2. Diese Ziffern werden einer spezifischen Gewichtung unterzogen. Dabei werden die Ziffern abwechselnd mit 1 und 3 multipliziert, beginnend mit der ersten Ziffer.
3. Die Produkte werden aufsummiert.
4. Basierend auf dieser Summe wird eine neue Prüfziffer berechnet. Dies geschieht, indem die Summe durch 10 geteilt und der Rest bestimmt wird. Die Differenz zwischen 10 und diesem Rest ergibt die neue Prüfziffer. Falls diese Differenz 10 beträgt, wird die Prüfziffer auf 0 gesetzt.

Im Anschluss wird die berechnete Prüfziffer mit der ursprünglich erkannten Prüfziffer verglichen. Bei Übereinstimmung wird die NVE als valide eingestuft und für die weitere Verarbeitung freigegeben. Stimmen die Prüfziffern nicht überein, wird das Ergebnis als nicht valide klassifiziert und von der weiteren Verarbeitung ausgeschlossen.

5.4 Zusammenführung beider Stufen

Die Integration des YOLOv9-Modells für die Ziffernlokalisierung und des CNN für die Ziffernerkennung wird in einem End-to-End-System umgesetzt. Hierfür wird eine Verarbeitungspipeline entworfen, die die Bilddaten aufnimmt, das YOLOv9-Modell zur Lokalisierung aufruft, die Ergebnisse filtert und die extrahierten Ziffern an das CNN zur Erkennung weiterleitet (siehe Abb. 5.15).

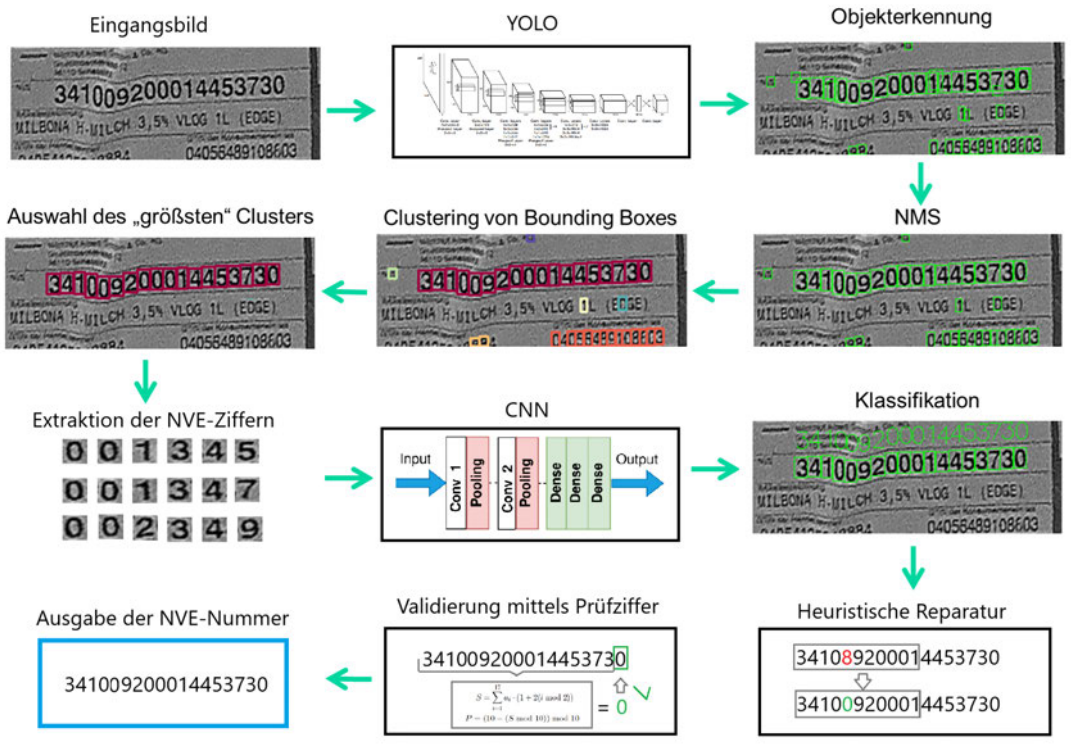


Abbildung 5.15: Verarbeitungsschritte des neuen Systems (veränderte Darstellung) [36] [38]

Nach Durchführung der Reparatur- und Validierungsschritte gibt das System die NVE als Zeichenkette aus.

6 Evaluierung und Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Modelle präsentiert und evaluiert. Anschließend erfolgt eine Zusammenfassung der Ergebnisse, auf deren Grundlage die beste Modellkombination ausgewählt wird.

6.1 Performance der Objektlokalisierung

Bei der Evaluierung der Leistungsfähigkeit von Objekterkennungsmodellen ist die Festlegung einer geeigneten Bewertungsmetrik maßgeblich. Es existiert eine Vielzahl von Metriken, deren Einsatz in Abhängigkeit vom spezifischen Anwendungsfall variiert. Die Auswahl einer geeigneten Metrik setzt ein Verständnis der zugrundeliegenden Prinzipien voraus, weshalb im Folgenden einige zentrale Begriffe und Definitionen erläutert werden. Zunächst wird die mehrklassige Objekterkennung betrachtet. Dabei wird zwischen vier Arten von Erkennungen unterschieden:

- **True Positive (TP)**: Ein Objekt wurde korrekt lokalisiert und richtig klassifiziert.
- **False Negative (FN)**: Ein tatsächliches Objekt wurde nicht erkannt.
- **False Positive (FP)**: Ein Bereich wurde fälschlicherweise als Objekt klassifiziert oder einem Objekt wurde die falsche Klasse zugeordnet.
- **True Negative (TN)**: Kein Objekt war vorhanden und der Bereich wurde korrekt als “Hintegrund” klassifiziert.

Ob eine Erkennung als True Positive oder False Positive gewertet wird, hängt dabei vom gewählten IOU-Schwellenwert ab. Dieser legt fest, ab welcher Überlappung zwischen richtig klassifiziertem und tatsächlichem Objekt eine Erkennung als korrekt gilt.
Die Abbildung 6.1 veranschaulicht die Arten der Objekterkennung am Beispiel der Ziffernerkennung. Grün umrandete Rechtecke stellen die vom Modell vorhergesagten Bounding Boxes dar, während schwarze Rechtecke die Ground-Truth-Boxen der jeweiligen Ziffern markieren. Oberhalb der vorhergesagten Boxen zeigen kleine Ziffern die

Klassifikation des Modells an, wobei grüne Ziffern korrekte und rote Ziffern falsche Klassifizierungen kennzeichnen.

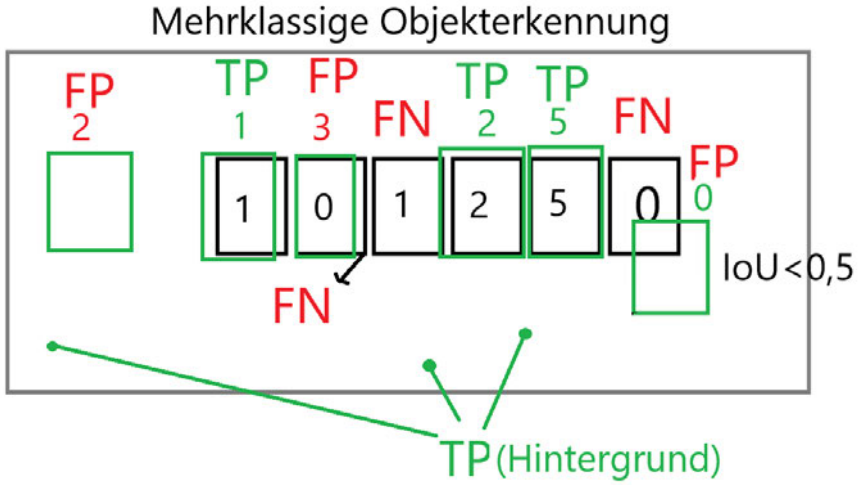


Abbildung 6.1: Fälle von mehrklassigen Objekterkennungen am Beispiel der Ziffererkennung (eigene Darstellung)

Diese Erkennungsarten finden ihre Anwendung in der sogenannten Confusion Matrix, welche in Abbildung 6.2 dargestellt ist. Sie zeigt, wie gut das Modell zwischen den verschiedenen Klassen unterscheidet und bildet die Basis für die Berechnung weiterer Leistungsmetriken.

		Vohersagte Klasse: Positive	Vohersagte Klasse: Negative
Tatsächliche Klasse	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Abbildung 6.2: Confusion Matrix (hier für binäre Klassifikation) (eigene Darstellung).

Zwei wesentliche Metriken, die sich aus diesen Erkennungsarten ergeben, sind Precision und Recall. Precision gibt den Anteil der korrekt vorhergesagten Objekte einer Klasse im Verhältnis zu allen als dieser Klasse zugeordneten Erkennungen an. Die Formel lautet:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.1}$$

Eine hohe Precision bedeutet, dass das Modell wenige False Positive Erkennungen hat, d.h. es identifiziert überwiegend echte Objekte korrekt, aber möglicherweise auf Kosten der Vollständigkeit.
Recall hingegen gibt den Anteil der korrekt erkannten Objekte einer Klasse im Verhältnis zu allen tatsächlichen Objekten dieser Klasse an. Er wird wie folgt definiert:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.2}$$

Als geeignete Metrik für das entwickelte Objekterkennungsmodell und den Anwendungsfall erscheint zunächst der Recall sinnvoll, da ein hoher Recall darauf hinweist, dass viele der tatsächlich vorhandenen Ziffern erkannt werden.
Wie bereits im Abschnitt 5.2.2 erläutert, wird für die Lokalisierung der NVE-Ziffern der Confidence Threshold bewusst niedrig gewählt, um möglichst alle Ziffern zu lokalisieren. Obgleich dies einen negativen Einfluss auf die Klassifizierung haben kann, spielt das im Rahmen des entwickelten Systems eine untergeordnete Rolle. Dies führt somit dazu, dass viele der lokalisierten Ziffern falsch erkannt werden.
Der Nachteil bei der Betrachtung des Recalls in der mehrklassigen Objekterkennung besteht darin, dass, wenn ein Objekt zwar richtig lokalisiert, aber falsch klassifiziert wird (was im Kontext des entwickelten Systems zulässig ist), dies nicht nur ein False Positive aufgrund der falschen Klassenzuordnung erzeugt, sondern auch ein False Negative für das Objekt (siehe Abb. 6.1). Die so entstandenen False Negatives wirken sich negativ auf den Recall-Wert aus, da sie fälschlicherweise als nicht erkannte Objekte gewertet werden, obwohl sie korrekt lokalisiert wurden und für die nachgelagerte Klassifikationsstufe zur Verfügung stehen.
Um dieses Problem zu umgehen, wird die Aufgabenstellung als einklassige Objekterkennung betrachtet. Das bedeutet, dass für das Modell alle Objekte, auch wenn sie zu unterschiedlichen Klassen gehören, als eine Klasse behandelt werden. Das Modell kann somit nur eine Klasse erkennen. Dies beseitigt die oben genannten Probleme, da nun bei korrekter Lokalisierung keine falsche Klassifizierung erfolgt und somit keine False

Negatives entstehen, die den Recall verfälschen könnten. Dies wird anschaulich in der Abbildung 6.3 dargestellt.

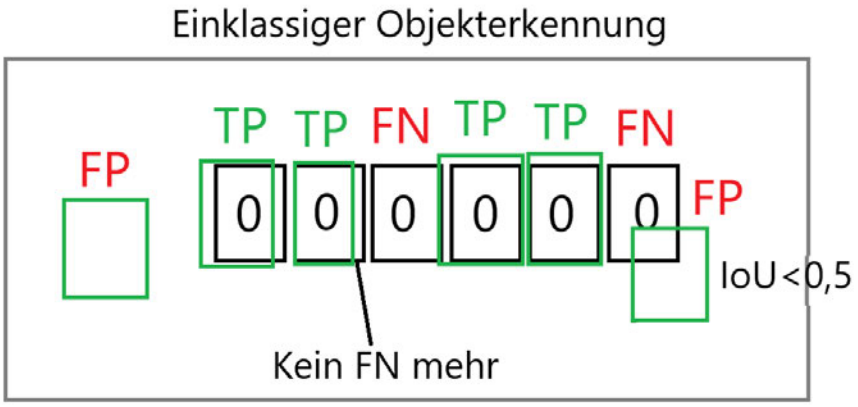


Abbildung 6.3: Fälle von einklassigen Objekterkennungen am Beispiel der Ziffererkennung (eigene Darstellung)

Um das YOLOv9-Modell auf diese Weise zu evaluieren, müssen zunächst alle Klassen in den annotierten Textdateien in eine einzige Klasse umgewandelt werden. Dafür wird ein Python-Skript entwickelt, das alle bestehenden Klassen automatisch durch die Klasse "0" ersetzt. Die Koordinaten der Bounding Boxes bleiben dabei unverändert.
Das YOLOv9-Modell wird für die im konzeptionellen Teil definierten Szenarien trainiert. Dabei gelten folgende Bedingungen:

- Die Anzahl der Epochen wird für alle Szenarien auf 200 festgelegt.
- Die Bewertung und Ermittlung des Recalls basiert ausschließlich auf den Testdaten.
- Für die Berechnung des Recalls wird ein IoU-Schwellenwert von 0,9 festgelegt.
- Confidence Threshold wird auf 0,02 festgelegt und bleibt bei diesem Wert auch für Evaluierung des Kombinierten Systems.

Die durchgeführte Test führen zu den folgenden Ergebnissen:

Tabelle 6.1: Recall-Werte bei unterschiedlichen Szenarien mit IoU=0,9 auf dem Testdatensatz

Szenario	Recall @IoU=0,9
30	0,90
50	0,92
100	0,94
150	0,95

Die Ergebnisse in der Tabelle 6.1 verdeutlichen, wie sich der Recall in Abhängigkeit von der Anzahl der Trainings- und Validierungsdaten entwickelt. Im ersten Szenario, bei dem 30 Trainingsbilder verwendet wurden, erreicht der Recall einen Wert von 0,90. Dies bedeutet, dass 90% der tatsächlich vorhandenen Ziffern unter oben genannten Bedingungen detektiert werden. Mit der Erhöhung der Trainingsdatenmenge auf 50 steigt der Recall-Wert auf 0,92. Dieser Trend setzt sich auch bei den Szenarien mit 100 und 150 Trainingsbildern fort, bei denen der Recall auf 0,94 bzw. 0,95 steigt.
Die Interpretation dieser Ergebnisse lässt darauf schließen, dass das Modell mit zunehmender Trainingsdatenmenge immer präziser bzw. zuverlässiger in der Lokalisierung der Ziffern wird. Dies steht im Einklang mit den Erwartungen, da ein Modell mit mehr Trainingsdaten in der Lage ist, besser generalisierende Merkmale zu erlernen, die zu einer höheren Lokalisierungsgenauigkeit führen. Ein relativ hoher IoU-Schwellenwert von 0,9 impliziert, dass das Modell nicht nur eine korrekte Lokalisierung der Ziffern erreichen muss, sondern auch, dass die Übereinstimmung zwischen den vorhergesagten und den tatsächlichen Bounding Boxes sehr präzise sein muss. Der hohe Recall-Wert bei 150 Trainingsbildern trotz des strengen IoU-Schwellenwertes deutet darauf hin, dass das Modell eine gute Fähigkeit zur präzisen Lokalisierung entwickelt hat. Die in Abbildung 6.4 dargestellten Ergebnisse der lokalisierten NVE-Ziffer unterstreichen die Leistungsfähigkeit des Modells.

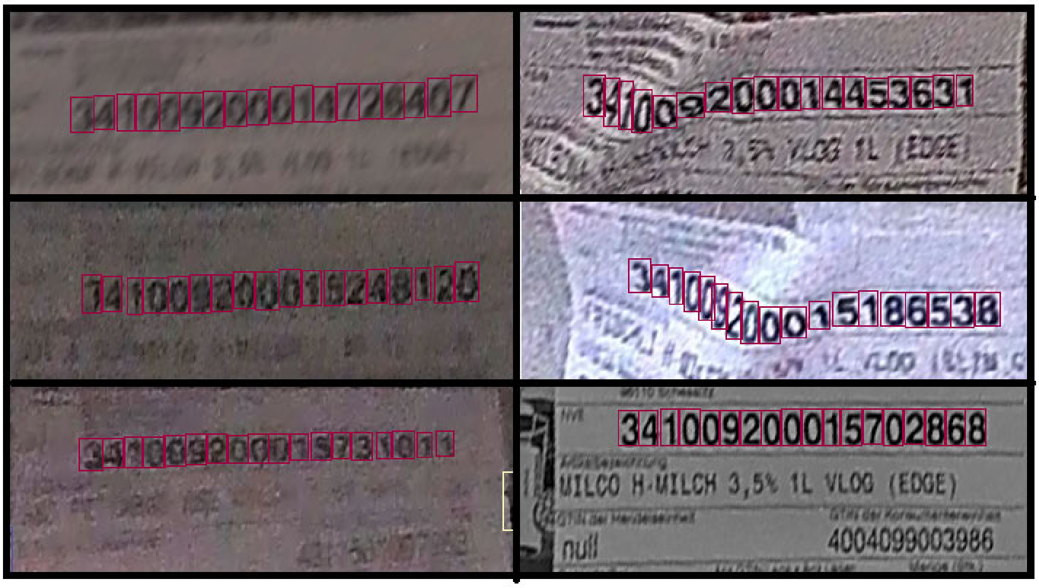


Abbildung 6.4: Lokalisierung von NVE-Ziffern aus dem Testdatensatz unter verschiedenen Bildqualitäten und geometrischen Bedingungen (eigene Darstellung)

Das entwickelte Modell zeigt sich Robust bei der Lokalisierung von Ziffern unter unterschiedlichen und herausfordernden Bedingungen. Wie in der Abbildung dargestellt, werden sowohl Bilder mit stark eingeschränkter Qualität als auch verformten Etiketten korrekt verarbeitet. Dies zeigt, dass das Modell in der Lage ist, auch bei erheblichen Abweichungen in der Bildqualität sowie bei unregelmäßigen geometrischen Formen zuverlässig zu arbeiten. Dies ist maßgeblich für das Gesamtsystem da die Leistungsfähigkeit des nachgelagerten CNN-Modells direkt von der Präzision der Objektlokalisierung abhängt.

6.2 Performance der Erkennung

Die Auswahl der Metrik für die Erkennungsmodelle hängt von der Klassifikationsaufgabe. Ein häufig verwendete Performance-Metrik ist die Genauigkeit bzw. Accuracy. Sie zeigt den Anteil der korrekt klassifizierten Datenpunkte im Verhältnis zur Gesamtzahl der Datenpunkte. Um eine aussagekräftige Bewertung zu liefern setzt diese Metrik voraus, dass die Klassen möglichst gleich verteilt sind. Bei unausgeglichenen Klassenverteilungen kann die Accuracy irreführend sein, da sie auch dann hoch ausfallen kann, wenn das

Modell die dominante Klasse bevorzugt und die kleinere Klasse komplett ignoriert. Die Abbildung 6.5 zeigt die Klassenverteilung der NVE-Ziffern im Testdatensatz.

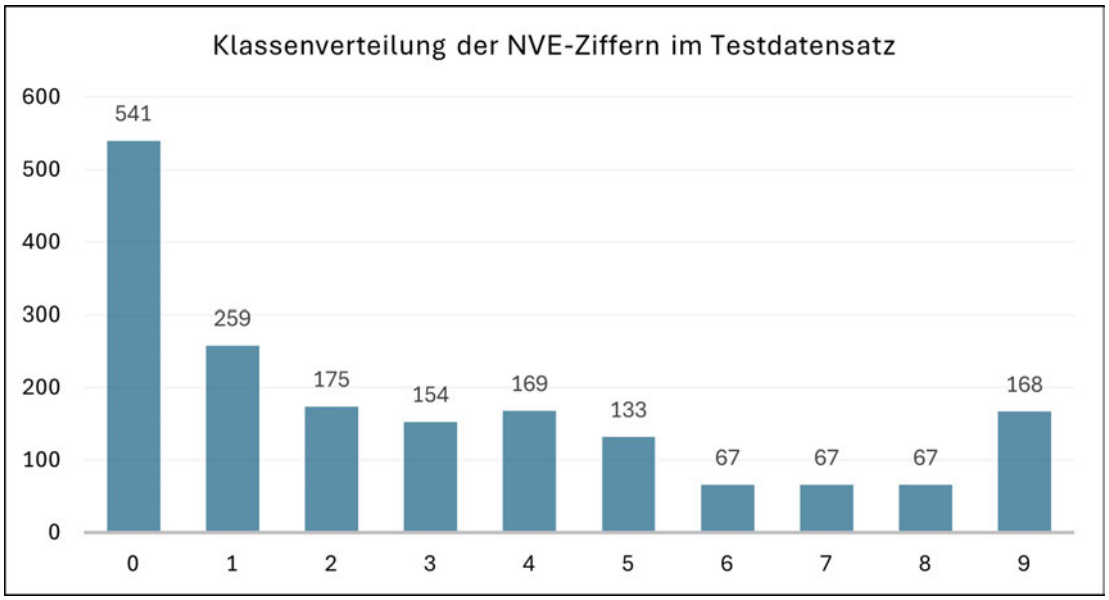


Abbildung 6.5: Klassenverteilung der NVE-Ziffern im Testdatensatz (eigene Darstellung)

Die Analyse der oberen Klassenverteilung offenbart eine signifikante Ungleichgewichtung der Ziffernklassen. Besonders auffällig ist die starke Überrepräsentation bestimmter Ziffern, wie etwa der “0” mit 541 Instanzen. Im Gegensatz dazu stehen deutlich unterrepräsentierte Klassen, darunter die Ziffern “6”, “7” und “8”, die jeweils nur 67 Mal im Datensatz vorkommen. Diese Ungleichheit in der Verteilung kann sich auf die Leistungsbewertung des Modells auswirken, insbesondere bei der Verwendung von Accuracy als Metrik. Das Modell kann dazu neigen, die dominante Klasse, in diesem Fall die Ziffer “0”, zu bevorzugen und dabei die selteneren Klassen zu vernachlässigen. Dies könnte zu einer hohen Genauigkeit führen, ohne dass das Modell tatsächlich eine adäquate Leistung bei der Klassifikation der weniger vertretenen Ziffern erbringt.
Angesichts der vorliegenden Klassenverteilung ist es sinnvoll, zusätzlich alternative Metriken in Betracht zu ziehen, um die Modellleistung differenzierter zu evaluieren. Bei ungleicher Klassenverteilung wird häufig die F1-Score-Metrik eingesetzt. Die F1-Score

wird mathematisch als harmonisches Mittel von Precision und Recall definiert und kann wie folgt formuliert werden:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.3}$$

Ein Wert von 1 indiziert eine perfekte Balance, während ein Wert von 0 darauf hinweist, dass entweder die Precision oder der Recall gleich null ist. Dies macht die F1-Score besonders wertvoll, wenn das Ziel darin besteht, die Leistung des Modells bei der Identifikation sowohl häufiger als auch seltener Klassen zu maximieren. Der F1-Score wird häufig auch in Prozent angegeben.
Die Macro-F1-Score wird verwendet, um den Einfluss der unterschiedlichen Klassenhäufigkeiten auf die Gesamtbewertung auszugleichen. Dabei wird der F1-Score für jede Klasse berechnet und anschließend das arithmetische Mittel dieser Werte gebildet, ohne Berücksichtigung der Klassenhäufigkeiten. Die Formel für die Macro-F1-Score lautet:

$$F1_{\text{macro}} = \frac{1}{n} \sum_{i=1}^{n} F1_i \tag{6.4}$$

Hierbei steht n für die Anzahl der Klasse und $F1_i$ für den F1-Score der Klase i.
Macro-F1-Score ist für diesen Fall besonders sinnvoll, da sie jeder Klasse das gleiche Gewicht bei der Bewertung der Modellleistung gibt. Dadurch wird verhindert, dass die Leistung auf häufig vertretenen Klassen die Gesamtbewertung dominiert, und die Modellleistung auf seltenen Klassen wird angemessen berücksichtigt. Aus diesem Grund wird hier Macro-F1-Score als primäre Metrik festgelegt.
Für die Evaluierung werden die im Abschnitt 5.3.1 beschriebenen Netzwerkarchitekturen für die im konzeptionellen Teil definierten Szenarien trainiert. Für diesen Abschnitt gelten folgende Bedingungen:

- Die Anzahl der Epochen wird für alle Szenarien auf 200 festgelegt.
- Die Lernrate wird auf 0,0004 festgelegt.
- Data Augmentation Transformationen werden beim Training verwendet.
- Als primäre Metrik wird der Macro-F1-Score eingesetzt, während die Accuracy als zusätzliche Vergleichsmetrik dient.
- Die Bewertung der Leistung erfolgt auf nicht vortrainierte Modelle und ausschließlich auf den Testdaten.

Die Ergebnisse der durchgeführten Tests werde in der Tabelle 6.2 dargestellt.

Tabelle 6.2: Performance-Vergleich zwischen VGG16 und LeNet-5

Szenario	**VGG16**		**LeNet-5**	
	Macro-F1-Score in %	**Accuracy in %**	**Macro-F1-Score in %**	**Accuracy in %**
30	98,23	98,83	98,24	98,94
50	98,55	99,05	99,12	99,27
100	98,82	99,11	99,13	99,33
150	98,97	99,22	99,22	99,39

Die obere Tabelle zeigt den Leistungsvergleich der Modelle VGG16 und LeNet-5 unter verschiedenen Trainingsszenarien.
VGG16 erreicht bei 30 Trainingsbeispielen einen Macro-F1-Score von 98,23% und eine Accuracy von 98,83%. Im Vergleich zu den ursprünglichen Erwartungen zeigt sich, dass das Modell auch bei einer geringeren Datenmenge eine solide Leistung erbringt. Mit einer wachsenden Anzahl an Trainingsdaten verbessert sich die Leistung kontinuierlich. Bei 150 Trainingsbeispielen erreicht es schließlich einen Macro-F1-Score von 98,97% und eine Accuracy von 99,22%.
LeNet-5 zeigt insgesamt etwas bessere Ergebnisse. Bei 30 Trainingsbeispielen erreicht es einen Macro-F1-Score von 98,24% und eine Accuracy von 98,94%, ähnlich wie VGG16. Mit zunehmender Trainingsdatenmenge erzielt LeNet-5 jedoch verbesserte Ergebnisse. Bei 150 Trainingsbeispielen erreicht es einen Macro-F1-Score von 99,22% und übertrifft damit VGG16 um 0,25%. Dies deutet auf eine gleichmäßigere Klassifikation über alle Klassen hin. Auch die Accuracy von LeNet-5 mit 99,39% liegt signifikant über der von VGG16, was zeigt, dass LeNet-5 entgegen den Erwartungen insgesamt eine höhere Genauigkeit erreicht, insbesondere bei einer größeren Anzahl an Trainingsdaten. Auch überraschend ist, dass die Modelle selbst in den Szenarien mit stark reduzierten Trainingsdaten (Szenarien 30 und 50) eine hohe Genauigkeit beibehalten. Dies deutet darauf hin, dass sie in der Lage sind, auch aus begrenzten Daten sinnvolle Muster zu lernen und diese erfolgreich auf die Testdaten anzuwenden. Mögliche Gründe dafür können die

effektiven Regularisierungsmechanismen sowie die angewandten Data-Augmentation-Transformationen sein, die die Generalisierungsfähigkeit der Modelle verbessern.
Obgleich die Ergebnisse auf den ersten Blick sehr hohe Werte aufweisen, muss berücksichtigt werden, dass selbst eine Genauigkeit von 99,39 % bei der Erkennung von NVE zu großen Abweichungen führen kann, da ein NVE nur dann als korrekt erkannt gilt, wenn alle 18 Ziffern fehlerfrei erkannt werden.
Für ein besseres Verständnis der Vorhersageleistung bei einzelnen Klassen bietet es sich an, die normalisierte Confusion-Matrix zu analysieren. Hierzu wird das LeNet-5-Modell im Szenario mit 30 Trainingsbildern betrachtet (siehe Abb. 6.6)

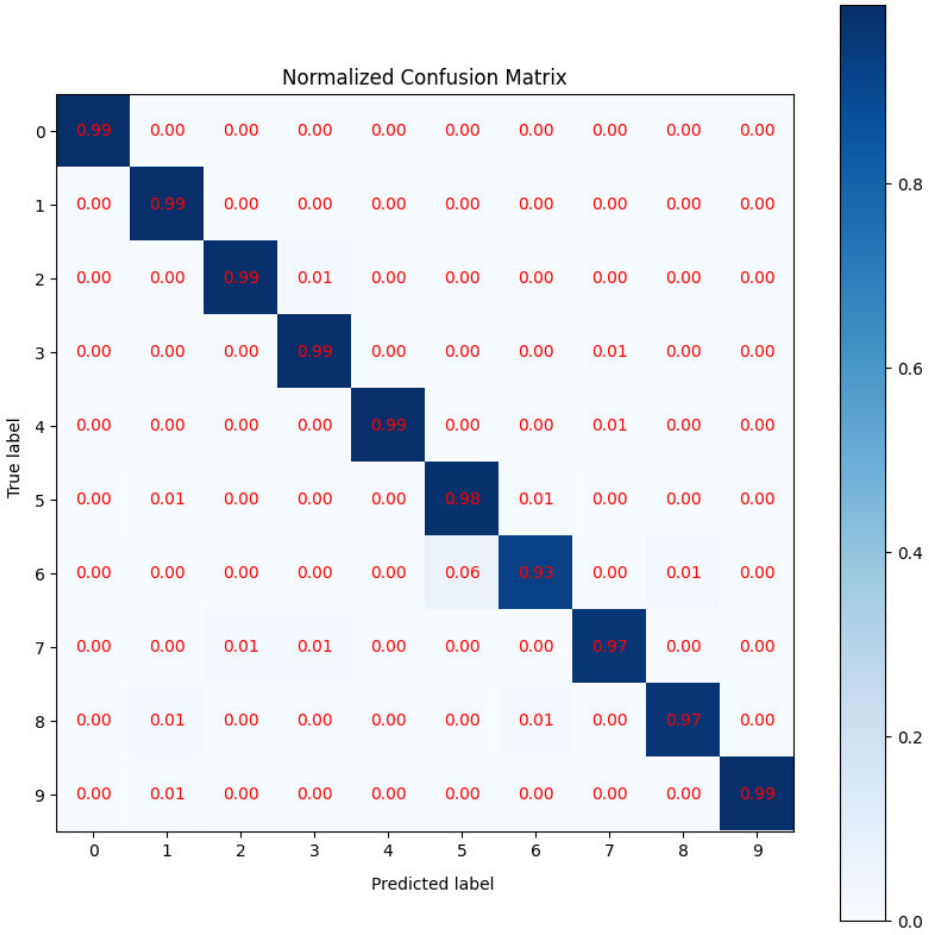


Abbildung 6.6: Normalisierte Confusion-Matrix im Szenario mit 30 Trainingsbildern und dem LeNet-5-Modell (eigene Darstellung)

Die Confusion Matrix zeigt, dass das Modell eine sehr hohe Vorhersagegenauigkeit bei den meisten Klassen aufweist, insbesondere bei den Ziffern “0” bis “5” sowie “9”, bei denen die Genauigkeit zwischen 98% und 99% liegt. Dies bedeutet, dass das Modell in diesen Fällen fast immer die richtige Klasse erkennt. Die Klasse “7” und “8” zeigen eine etwas niedrigere Genauigkeit von 97%. Eine Ausnahme bildet die Klasse “6”, bei der die Genauigkeit nur bei 93% liegt. Hier treten vermehrt Fehlklassifikationen auf, wobei 6% der Instanzen fälschlicherweise als Klasse “5” erkannt werden. Dies deutet darauf hin, dass das Modell insbesondere Schwierigkeiten hat, die Ziffern “5” und “6” korrekt zu unterscheiden. Dies ist auf sehr ähnliche Merkmale der beiden Klassen zurückzuführen. Das liegt unter anderem daran, dass die Anzahl der Instanzen der Ziffer “6” im Trainingssatz sehr gering ist. Ein ähnliches Bild zeigt auch der Confusion Matrix Im Szenario mit 150 Traingsdaten (siehe Abb. 6.7).

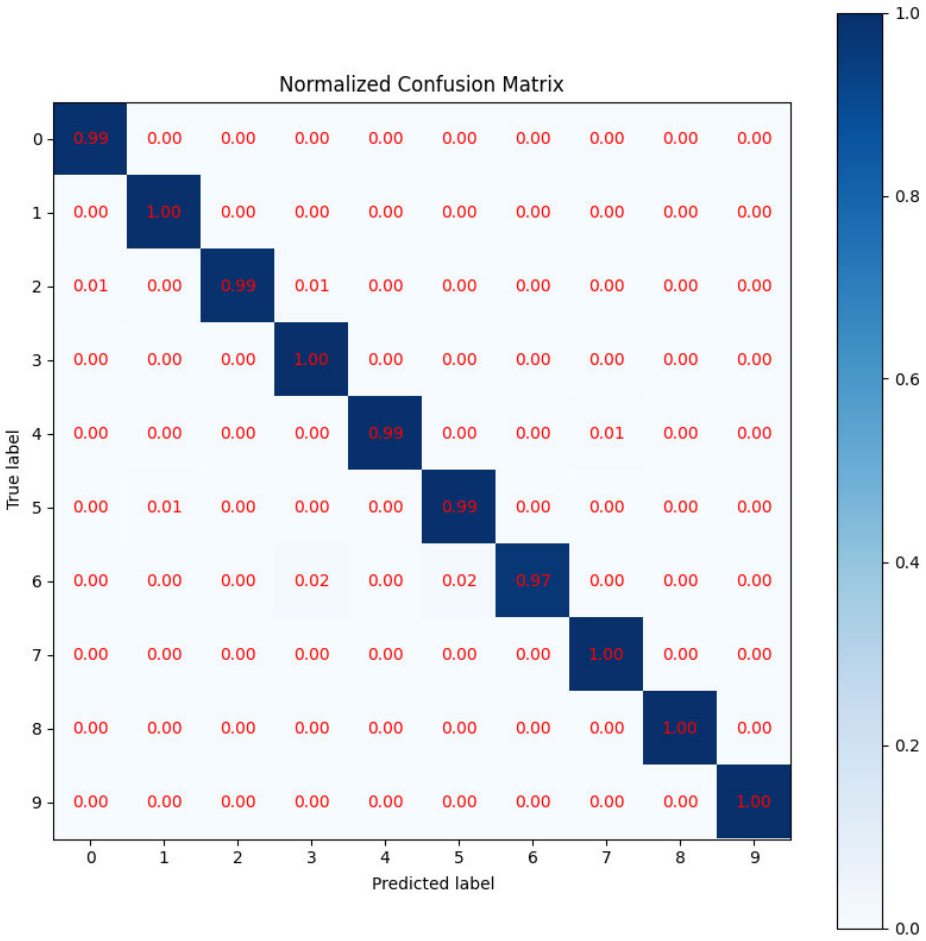


Abbildung 6.7: Normalisierte Confusion-Matrix im Szenario mit 150 Trainingsbildern und dem LeNet-5-Modell (eigene Darstellung)

Auch in diesem Szenario wird die zahl “6” unterrepräsentiert. Jedoch mit 97% in einem viel geringeren Maß als im vohherigen Fall. Bei allen andere Klassen zeigt das Modell gute Erkennungsgenauigkeit, sodass nur die Ziffer “6” als eine Engstelle Idetifiziert werden kann. Insgesamt sind die Fehlklassifikationen marginal und betreffen meist nur 1% der Instanzen.

6.3 Performance der kombinierten Ansätze

Um eine vergleichende Analyse mit dem bestehenden System zu ermöglichen, wird zur Bewertung der Gesamtleistung des neuen Systems die NVE-Erkennungsrate als Metrik verwendet. Diese Metrik wurde bereits im Abschnitt 3.3 erläutert. Zur Evaluierung werden drei Modelle in jedem der festgelegten Szenarien gegenübergestellt:

1. YOLOv9 in Kombination mit LeNet-5
2. YOLOv9 in Kombination mit VGG16
3. YOLOv9 allein

Anzumerken ist, dass für einen repräsentativen Vergleich die Tests ohne den Einsatz der heuristischen Reparatur durchgeführt werden.
Für diesen Abschnitt gelten folgende Bedingungen:

- Es werden die im Abschnitt 6.2 trainierten Modelle eingesetzt.
- Heuristische Reparatur wird nicht angewendet.
- Die Bewertung der Leistung erfolgt auf nicht vortrainierten Modellen und ausschließlich auf den Testdaten.
- Als Vergleichsmetrik wird die NVE-Erkennungsrate verwendet.

Die Tabelle 6.3 zeigt den Vergleich der drei OCR-Tools hinsichtlich der NVE-Erkennungsrate.

Tabelle 6.3: Vergleich der Modelle anhand der NVE-Erkennungsrate

Szenario	**YOLOv9 + LeNet-5**	**YOLOv9 + VGG16**	**YOLOv9**
30	93 %	92 %	83 %
50	95 %	93 %	88 %
100	96 %	94 %	93 %
150	97 %	95 %	93 %

Die Ergebnisse zeigen, dass die Kombination aus YOLOv9 und LeNet-5 in allen Szenarien die höchsten Erkennungsraten erzielt, während YOLOv9 als alleinstehendes Modell

durchweg niedrigere Erkennungsraten aufweist.
Im Szenario mit der geringsten Datenmenge von 30 Trainingsbeispielen zeigt die Kombination mit VGG16 eine Erkennungsrate von 92%, die von LeNet-5 mit 93% knapp übertroffen wird. Beide Kombinationen übertreffen die Leistung von YOLOv9 um 9 bzw. 10 Prozentpunkte. Diese Diskrepanz verringert sich jedoch mit zunehmender Datenmenge, was auf eine höhere Sensitivität von YOLOv9 gegenüber limitierten Trainingsdaten im Vergleich zu den CNN-Modellen hindeutet.
Die höchste Genauigkeit wird bei beiden Kombinationen im datenreichsten Szenario mit 150 Trainingsbeispielen erreicht. Hierbei erzielt die Kombination aus YOLOv9 und LeNet-5 eine Genauigkeit von 97%, während YOLOv9 mit VGG16 eine Genauigkeit von 95% erreicht. Im Kontrast dazu erreicht YOLOv9 Modell seine Höchstleistung von 93% bereits im Szenario mit 100 Trainingsbildern und zeigt darüber hinaus keine weitere Verbesserung.
Die Evaluation der entwickelten Modelle offenbart eine deutliche Leistungssteigerung gegenüber der bestehenden EasyOCR-Lösung. Alle implementierten Modellkombinationen übertreffen die NVE-Erkennungsgenauigkeit von EasyOCR, die bei 77% liegt, signifikant. Besonders bemerkenswert ist die Leistung im Szenario mit 50 Trainingsbeispielen, wo die Kombination aus YOLOv9 und LeNet-5 eine Genauigkeit von 95% erreicht und damit 18 Prozentpunkte über dem Ergebnis von EasyOCR liegt. Diese Leistung übertrifft die ursprünglichen Anforderungen an die Erkennungsgenauigkeit deutlich. Die beste Gesamtleistung wird ebenfalls von der Kombination YOLOv9 und LeNet-5 erzielt, die eine Verbesserung von 20 Prozentpunkten gegenüber EasyOCR aufweist.
Die Abbildung 6.8 illustriert durch das neue System korrekt erkannten Beispiel von NVE-Bildern.

Abbildung 6.8: Korrekte Erkennung von NVE mit YOLOv9 und LeNet-5 (eigene Darstellung)

Die Abbildung zeigt, dass das System sowohl mit Bildern hoher Qualität, wie sie in der obersten Reihe zu sehen sind, als auch mit Aufnahmen von geringerer Qualität effektiv arbeiten kann. Bemerkenswert ist die Fähigkeit des Systems, NVE-Nummern auch unter ungünstigen Lichtverhältnissen zu erkennen. Ebenso stellen verschwommene Bilder oder verformte Zahlen keine größeren Hindernisse für das System dar.
Trotz der insgesamt guten Erkennungsraten gelingt es dem System nicht immer, die Ziffern vollständig zu erkennen. Die Abbildung 6.9 illustriert die drei Beispiele aus dem Szenario mit 150 Trainingsbildern, bei denen eine vollständige Erkennung nicht erreicht wurde

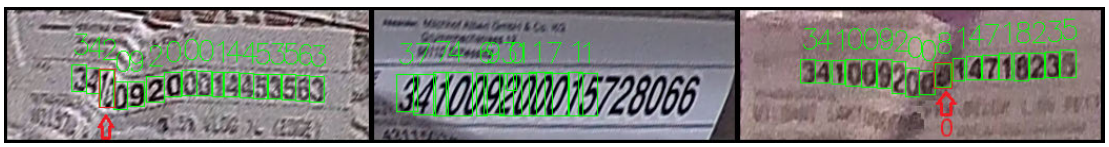


Abbildung 6.9: Fehlerhafte Erkennung von NVE mit YOLOv9 und LeNet-5 (eigene Darstellung)

Die roten Rechtecke in der Abbildung verdeutlichen die problematischen Bereiche für das System. Im ersten Bild scheitert das YOLOv9-Modell daran, die Ziffern 1 und 0 zu trennen, und lokalisiert sie fälschlicherweise als eine einzige Zahl. Dies zeigt eine Schwachstelle bei der Unterscheidung sehr eng beieinanderliegender Ziffern.
Das mittlere Beispiel offenbart ein vollständiges Versagen sowohl der Lokalisierung als auch der Erkennung. Die wenigen lokalisierten Ziffern konnten vom CNN-Modell nicht

korrekt erkannt werden. Der Grund hierfür liegt in der unzureichenden Begradigung des ausgeschnittenen Etiketts, was zu einer ungünstigen Perspektive führt. Da das YOLO-Modell nur gerade Bounding-Boxes vorhersagen kann, werden bei stark geneigten Bildern Ziffern oft mit Teilen anderer Ziffern zusammen extrahiert. Infolgedessen können diese vom CNN-Modell nicht korrekt erkannt werden.
Im letzten Beispiel wird die Ziffer 0 fälschlicherweise als 8 klassifiziert, was auf die starke Deformation der Ziffer zurückzuführen ist.
Die Messung der durchschnittliche Verarbeitungsgeschwindigkeit erfolgt mit der gleichen Hardwaresystemkonfiguration die im konzeptionellen Teil beschrieben wird.
Es ergeben sich die folgenden Ergebnisse:

Tabelle 6.4: Vergleich der Modelle anhand der Verarbeitungszeit in Bildern pro Sekunde

Szenario	**YOLOv9 + VGG16 Bilder/s**	**YOLOv9 + LeNet-5 Bilder/s**	**YOLOv9 Bilder/s**
30	1,92	2,44	5,26
50	2,04	2,33	5,56
100	1,85	2,44	5,00
150	1,96	2,38	5,56

Die in Tabelle 6.4 dargestellten Ergebnisse zeigen, dass YOLOv9 allein die schnellste Verarbeitungszeit erzielt und in allen Szenarien die höchste Rate an Bildern pro Sekunde erreicht. Im Vergleich dazu liegen die kombinierten Modelle mit VGG16 und LeNet-5 hinsichtlich der Verarbeitungszeit deutlich darunter, wobei LeNet-5 eine etwas höhere Geschwindigkeit als VGG16 aufweist. Mit 5,26 Bildern pro Sekunde übertrifft YOLOv9 auch die Verarbeitungsrate von EasyOCR, das eine Rate von 3,3 Bildern pro Sekunde erreicht. Dies zeigt, dass YOLOv9 hinsichtlich der Geschwindigkeit sowohl gegenüber den kombinierten Modellen als auch gegenüber EasyOCR überlegen ist. Zudem lässt sich aus den Ergebnissen ableiten, dass die Verarbeitungszeit keine Korrelation mit der Menge der Trainingsdaten aufweist.
Alle Modelle erfüllen die Anforderungen an Verarbeitungsgeschwindigkeit

6.4 Bewertung der Dateneffizienzstrategien

Wie bereits im Abschnitt 5.3.1 erläutert, erfolgt das Training des VGG16-Modells sowohl mit als auch ohne vortrainierte Gewichte. Bei der Verwendung des vortrainierten Modells werden die vorderen Schichten eingefroren, während nur die hinteren Klassifikationsschichten neu trainiert werden. Diese Methode zielt darauf ab, bereits gelerntes Wissen auf die spezifische Aufgabe der NVE-Erkennung zu übertragen. Die Tabelle 6.6 fasst die Ergebnisse der durchgeführten Tests zusammen.

Tabelle 6.5: Vergleich der Leistung von VGG16 mit und ohne vortrainierte Gewichte bei verschiedenen Trainingsdatensätzen

Szenario	**VGG16 ohne Vortraining**		**VGG16 vortrainiert**	
	Macro-F1-Score in %	**Accuracy in %**	**Macro-F1-Score in %**	**Accuracy in %**
30	98,23	98,83	75,41	82,08
50	98,55	99,05	80,11	84,87
100	98,82	99,11	80,35	85,22
150	98,97	99,22	80,58	85,55

Die Ergebnisse zeigen deutliche Unterschiede zwischen dem Training mit und ohne vortrainierte Gewichte. Entgegen der häufigen Annahme, dass Transfer Learning besonders bei begrenzten Datensätzen vorteilhaft ist, zeigt sich hier ein gegenteiliges Bild.
Das Modell ohne Vortraining erzielt in allen untersuchten Szenarien durchweg bessere Ergebnisse, sowohl hinsichtlich der Accuracy als auch des Macro-F1-Scores. Besonders auffällig ist der Leistungsunterschied im Szenario mit nur 30 Trainingsbildern. Das vortrainierte Modell erreicht lediglich einen Macro-F1-Score von 75,41% und eine Accuracy von 82,08%, was deutlich unter den Werten des Modells ohne vortrainierte Gewichte liegt. Mit steigender Anzahl der Trainingsbilder verbessert sich zwar die Leistung des vortrainierten Modells, erreicht aber maximal nur 80,58% für den Macro-F1-Score und 85,55% für die Accuracy. Diese Werte bleiben signifikant hinter den Ergebnissen des Modells ohne Vortraining zurück.
Die vergleichsweise niedrigen Macro-F1-Scores des vortrainierten Modells deuten darauf hin, dass es insbesondere Schwierigkeiten bei der Erkennung seltener Klassen hat. Dies könnte auf eine stark unausgewogene Repräsentation der Klassen im Trainingsprozess

hinweisen.
Die Analyse der Ergebnisse offenbart einen negativen Effekt des Transfer Learnings auf die Klassifizierung der NVE-Ziffern, was als Negativer Transfer bezeichnet wird. Dieser Effekt lässt sich darauf zurückführen, dass die NVE-Ziffern keine Teilmenge des ursprünglichen Trainingsdatensatzes darstellen, mit dem das vortrainierte Modell initial trainiert wurde.
Im Gegensatz dazu erweisen sich die angewandten Data-Augmentation-Techniken als vorteilhaft, wie das die Tabellen 6.6 und 6.7 demonstrieren.

Tabelle 6.6: Vergleich der Leistung von VGG16 mit und ohne Data Augmentation bei verschiedenen Trainingsdatensätzen

Szenario	**VGG16 mit Data Augmentation**		**VGG16 ohne Data Augmentation**	
	Macro-F1-Score in %	**Accuracy in %**	**Macro-F1-Score in %**	**Accuracy in %**
30	98,23	98,83	97,43	98,2
50	98,55	99,05	97,51	98,44
100	98,82	99,11	97,86	98,67
150	98,97	99,22	97,96	98,68

Die Ergebnisse der Tabelle 6.6 zeigen eindeutig den positiven Einfluss von Data Augmentation auf die Leistung des VGG16-Modells. In allen untersuchten Szenarien zeigt das Modell mit Data Augmentation sowohl eine höhere Genauigkeit als auch einen verbesserten Macro-F1-Score im Vergleich zum Modell ohne Augmentation. Die Leistungsdifferenz zwischen den Modellen mit und ohne Augmentation bleibt in allen Szenarien ähnlich, was darauf hinweist, dass Data Augmentation auch bei größeren Trainingsdatensätzen von Vorteil ist.

Tabelle 6.7: Vergleich der Leistung von LeNet5 mit und ohne Data Augmentation bei verschiedenen Trainingsdatensätzen

Szenario	LeNet5 mit Data Augmentation		LeNet5 ohne Data Augmentation	
	Macro-F1-Score in %	Accuracy in %	Macro-F1-Score in %	Accuracy in %
30	98,24	98,94	98,07	98,6
50	99,12	99,27	98,1	98,72
100	99,13	99,33	98,55	99,22
150	99,22	99,39	98,97	99,22

Die Analyse der Tabelle 6.7 offenbart ein ähnliches Muster für das LeNet5-Modell wie zuvor beim VGG16-Modell beobachtet. Die Anwendung von Data Augmentation führt durchgängig zu verbesserten Klassifikationsergebnissen, was die Effektivität dieser Technik über verschiedene Modellarchitekturen hinweg unterstreicht.

6.5 Validierung der Anforderungen

Die Systemanforderungen und technischen Rahmenbedingungen wurden während der Entwicklung und Implementierung kontinuierlich überprüft. Der Schwerpunkt lag auf der Verbesserung der NVE-Erkennungsrate und der Erfüllung der Anforderungen bei begrenztem Datenvolumen. Basierend auf den Evaluierungsergebnissen wurde die Kombination aus YOLOv9 und LeNet-5 als neues System ausgewählt. Die Tabelle 6.8 zeigt, dass dieses System fast alle funktionalen Anforderungen erfüllt.

Tabelle 6.8: Validierung der Systemanforderungen

Priorität	Anforderung	Erfüllungsgrad
1	Fähigkeit zur korrekten Klassifizierung aller Ziffern von 0 bis 9	Erfüllt
2	Steigerung der NVE-Erkennungsrate um mindestens 5% gegenüber der bestehenden EasyOCR-Lösung	Erfüllt
3	Leistung auch bei begrenztem Trainingsdatensatz (50 Bilder)	Erfüllt
4	Filterung irrelevanter Symbole und Ziffern	Erfüllt
5	Verarbeitung von mindestens einem Bild pro Sekunde	Erfüllt
6	Zuverlässige Leistung unter verschiedenen Bildbedingungen (unscharf, dunkel, geknickt)	Teilweise erfüllt

Das System klassifiziert zuverlässig alle Ziffern von 0 bis 9 und übertrifft die Erkennungsrate der EasyOCR-Lösung um 20%. Selbst mit einem Trainingsdatensatz von nur 50 Bildern erzielt das System eine um 18% höhere Erkennungsrate als EasyOCR. Die Verarbeitungsgeschwindigkeit von mindestens einem Bild pro Sekunde wird erreicht. Es nutzt verschiedene Filtermethoden, die sich als effektiv erwiesen haben. Das System bewältigt die meisten Bildbedingungen, zeigt jedoch Schwächen bei starken Verzerrungen und geneigten Bildern.
Alle technischen Rahmenbedingungen, wie die Entwicklung in Python, Unterstützung von NVIDIA-GPUs und die Kompatibilität mit TensorFlow, sind erfüllt (siehe Tabelle 6.9).

Tabelle 6.9: Validierung der technischen Rahmenbedingungen

Priorität	Technische Rahmenbedingung	Erfüllungsgrad
1	Entwicklung in Python	Erfüllt
2	NVIDIA-GPU und CUDA-Unterstützung	Erfüllt
4	TensorFlow-Kompatibilität	Erfüllt
3	Lauffähigkeit auf Windows-Servern	Erfüllt

Die Validierung bestätigt, dass die zentralen Anforderungen erfüllt sind. Die entwickelte Lösung zeigt eine hohe Leistungsfähigkeit und eignet sich für die praktische Anwendung.

7 Fazit

7.1 Zusammenfassung

In dieser Arbeit wurde eine Deep-Learning-basierte Lösung zur Erkennung von Nummern der Versandeinheit (NVE) entwickelt, die die bestehende EasyOCR-Lösung in der Erkennungsgenauigkeit deutlich übertrifft. Der Schwerpunkt lag auf der Steigerung der Erkennungsrate unter der Bedingung eines begrenzten Datenvolumens. Verschiedene Deep-Learning-Methoden wurden evaluiert, wobei sich eine Kombination aus YOLOv9 zur Objekterkennung und LeNet-5 zur Klassifikation der Ziffern als optimal herausstellte. Diese Lösung erreicht eine Erkennungsgenauigkeit von 97 %, was einer Verbesserung um 20 % gegenüber der EasyOCR-Lösung entspricht. Selbst mit einem reduzierten Trainingsdatensatz von nur 50 Bildern wird eine Erkennungsgenauigkeit von 95 % erzielt, was eine Steigerung von 18 % gegenüber der bestehenden Lösung darstellt. Die entwickelte Lösung erfüllt alle wesentlichen Anforderungen, darunter eine Verarbeitungsgeschwindigkeit von mindestens einem Bild pro Sekunde, und zeigt sich robust in den meisten Bildbedingungen. Damit bietet sie ein hohes Potenzial für den praktischen Einsatz in der Logistik.

7.2 Ausblick

Für zukünftige Arbeiten bietet sich eine gezielte Hyperparameteroptimierung an, um das volle Potenzial der Modelle weiter auszuschöpfen und die Erkennungsgenauigkeit sowie die Robustheit des Systems deutlich zu steigern. Bei der Hyperparameteroptimierung werden Parameter wie Lernrate, Batchgröße oder Anzahl der Schichten gezielt angepasst, um die Modellleistung zu maximieren. Diese Optimierung erfordert jedoch zusätzliche Rechenkapazität und ist mit einem erheblichen Zeitaufwand verbunden. Der Einsatz von Generative Adversarial Networks (GANs) könnte zudem spannende Möglichkeiten eröffnen, um das System gezielt auf herausfordernde Szenarien vorzubereiten. Durch die Generierung realistischer synthetischer Daten ließen sich schwierige Bedingungen wie stark verzerrte oder geneigte Bilder simulieren, ohne auf umfangreiche manuelle

Datenerhebungen angewiesen zu sein. Gleichzeitig könnte der Einsatz modernerer Architekturen wie Transformer-Modelle weitere vielversprechende Fortschritte bringen, die Erkennungsleistung auf ein neues Niveau heben und die Stabilität des Systems unter anspruchsvollen Bildbedingungen verbessern.

Literaturverzeichnis

[1] Einführung und geschichte neuronaler netze. https://www.desy.de/~guenterg/prosem/Einf_hrung_und_Geschichte_neuronaler_Netze.html. Last accessed: October 11, 2024.

[2] Grundlagen neuronaler netze. https://www.hochschule-trier.de/fileadmin/Hauptcampus/Fachbereich_Informatik/Fernstudium/Dokumente/Leseproben/bdl_grundlagen.pdf.

[3] Modell training mit ultralytics yolo. https://docs.ultralytics.com/de/modes/train/#augmentation-settings-and-hyperparameters. Last accessed: October 11, 2024.

[4] Nervenzelle - aufbau und funktion. https://www.abiblick.de/nervenzelle.

[5] Overfitting and methods of addressing it, March 2021. Last accessed: September 19, 2024. Verfügbar unter: https://analystprep.com/study-notes/cfa-level-2/quantitative-method/overfitting-methods-addressing/.

[6] torch.onnx, 2024. Last accessed: September 19, 2024. Verfügbar unter: https://pytorch.org/docs/stable/onnx.html.

[7] Activeloop. Machine learning datasets. Verfügbar unter: https://datasets.activeloop.ai/docs/ml/datasets/mnist/.

[8] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Character region awareness for text detection. arXiv preprint arXiv:1904.01941, 2019. Verfügbar unter: https://arxiv.org/pdf/1904.01941.

[9] Baeldung. Intersection over union for object detection. https://www.baeldung.com/cs/object-detection-intersection-vs-union, 2024. Letzte Aktualisierung: 18. März 2024, Überprüft von: Milos Simic.

[10] Helmut Baumgarten. Das beste in der logistik — auf dem weg zu logistischer exzellenz. In Logistik-Management: Herausforderungen, Chancen, Lösungen, pages 11–19. Springer, 2008. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-540-78405-0_1.

[11] Dave Bergmann and Cole Stryker. Was ist pytorch?, November 2023. Zugriff am: 19. September 2024. Verfügbar unter: https://www.ibm.com/de-de/topics/pytorch#:~:text=Ein%20Berechnungsgraph%20bildet%20den%20Datenfluss,in%20einem%20mathematischen%20System%20ab.

[12] Gaudenz Boesch. Labelimg for image annotation. https://viso.ai/computer-vision/labelimg-for-image-annotation/, February 2022. Last accessed: October 11, 2024.

[13] Heide Buhl and Ilka Machemer. Der sscc/nve und das gs1 transportetikett in der anwendung. https://www.gs1-germany.de/fileadmin/gs1/fachpublikationen/NVE_in_der_Anwendung.pdf, March 2020. Letztes Änderungsdatum: 01.03.2020, Deutsche Erstausgabe.

[14] Simon Johann Romedi Morpheus Cyrani. Neuronale netze: Grundlagen und einfaches anwendungsbeispiel. https://www.hans-riegel-fachpreise.com/fileadmin/hans-riegel-fachpreise/Module/ausgezeichnete-arbeiten/hans-riegel-fachpreise-seminararbeit-vwa-2019-cyrani.pdf, November 2018.

[15] D. Deng. DBSCAN Clustering Algorithm Based on Density. In 2020 7th International Forum on Electrical Engineering and Automation (IFEEA), pages 949–953, 2020. Verfügbar unter: https://ieeexplore.ieee.org/abstract/document/9356727.

[16] David Ebert. Bildklassifizierung mit neuronalen netzen. Masterarbeit, Fachbereich Ingenieur- und Naturwissenschaften, Hochschule Merseburg, October 2018. Verfügbar unter: https://opendata.uni-halle.de/bitstream/1981185920/14186/1/EbertDavid_Bildklassifizierung_mit_neuronalen_Netzen.pdf.

[17] Line Eikvil. Ocr: Optical character recognition. https://home.nr.no/~eikvil/OCR.pdf, December 1993. Zugriff am: 19. September 2024.

[18] Ross Girshick. Fast r-cnn. arXiv preprint arXiv:1504.08083, 2015. Verfügbar unter: https://arxiv.org/pdf/1504.08083.

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. Tech report (v5), UC Berkeley, 2014. Verfügbar unter: https://arxiv.org/abs/1311.2524.

[20] Mohamed Hasan. Ocr with deep learning in pytorch (easyocr) part 1: A beginner guide of ocr with python code, 2023. Verfügbar unter: https://eng-mhasan.medium.com/ocr-with-deep-learning-in-python-e443970d09e4.

[21] Nazanin Sadat Hashemi, Roya Babaie Aghdam, Atieh Sadat Bayat Ghiasi, and Parastoo Fatemi. Template matching advances and applications in image analysis, 2016. Verfügbar unter: https://arxiv.org/abs/1610.07231.

[22] JaidedAI. Easyocr. Verfügbar unter: https://github.com/JaidedAI/EasyOCR.

[23] Glenn Jocher. Ultralytics yolov8, 2024. Last accessed: September 19, 2024. Verfügbar unter: https://docs.ultralytics.com/models/yolov8/#can-i-benchmark-yolov8-models-for-performance.

[24] Glenn Jocher. Yolov9: A leap forward in object detection technology, 2024. Last accessed: September 19, 2024. Verfügbar unter: https://docs.ultralytics.com/models/yolov9/.

[25] Michael Kipp. Neuronale netze und deep learning. https://michaelkipp.de/deeplearning/Perzeptron.html.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, November 1998. Verfügbar unter: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. arXiv preprint arXiv:1512.02325, 2016. Verfügbar unter: https://arxiv.org/pdf/1512.02325.

[28] Jonah Lüdemann. Grundlagen neuronaler netze. https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2020/siw-20_luedemann_grundlagen-nn_ausarbeitung.pdf, August 2020.

[29] Sumit Mishra, Sree Devi, and Badri Narayanan. Technology dimensions of automation in business process management industry. International Journal of Engineering

and Advanced Technology (IJEAT), 8(6):2249–8958, August 2019. Verfügbar unter: https://d1wqtxts1xzle7.cloudfront.net/86296814/F8569088619-libre.pdf.

[30] S. Mori, C.Y. Suen, and K. Yamamoto. Historical review of ocr research and development. Proceedings of the IEEE, 80(7):1029–1058, 1992.

[31] A.V. Morozov, V.L. Levkivskyi, and D.D. Plechystyy. Activation functions in neural networks: Overview and comparison. Vernadsky Journals, 3, 2024. Verfügbar unter: https://tech.vernadskyjournals.in.ua/journals/2024/3_2024/part_1/24.pdf.

[32] Vladimir Nasteski. An overview of the supervised machine learning methods. Horizons, December 2017. Verfügbar unter: https://www.researchgate.net/publication/328146111_An_overview_of_the_supervised_machine_learning_methods.

[33] K.H. Nikoghosyan. Ocr engine comparison — tesseract vs. easyocr vs. keras-ocr, 2021. Russian-Armenian University, Armenia, Yerevan. Verfügbar unter: https://elibrary.ru/item.asp?id=48231604.

[34] Artem Oppermann. Auswahl der trainingsdaten, 2024. Last accessed: September 19, 2024. Verfügbar unter: https://artemoppermann.com/de/auswahl-von-trainingsdaten/.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703, 2019. Verfügbar unter: https://arxiv.org/pdf/1912.01703.

[36] K. M. Ankita Rai and Najme Zehra Naqavi. Brain tumor detection by fusion techniques. In Proceedings of the 2024 Conference, pages 563–580. Springer, 2024. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-981-97-3180-0_37.

[37] Sathiapriya Ramiah, Tan Yu Liong, and Manoj Jayabalan. Detecting text based image with optical character recognition for english translation and speech using android. In 2015 IEEE Student Conference on Research and Development (SCOReD). IEEE, November 2015. Verfügbar unter: https://www.researchgate.net/publication/301443412_Detecting_text_based_image_with_optical_character_recognition_for_English_translation_and_speech_using_Android.

[38] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2016. Verfügbar unter: https://arxiv.org/pdf/1506.02640.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv preprint arXiv:1506.01497, 2016. Verfügbar unter: https://arxiv.org/pdf/1506.01497.

[40] Ricardo Ribani and Mauricio Marengoni. A survey of transfer learning for convolutional neural networks. In 2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T), pages 47–57. IEEE, 2019. Verfügbar unter: https://ieeexplore.ieee.org/document/8920338.

[41] Udit Roy. Text recognition and retrieval in natural scene images. Master's thesis, International Institute of Information Technology, Hyderabad, 2015. Master's thesis. Verfügbar unter: https://cdn.iiit.ac.in/cdn/cvit.iiit.ac.in/images/Thesis/MS/UditRoy/Thesis.pdf.

[42] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way, December 2018. Verfügbar unter: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, Published in Towards Data Science, Last accessed: September 19, 2024.

[43] Ashutosh Saitwal. A quick guide to easy ocr. Verfügbar unter: https://klearstack.com/easy-ocr/.

[44] Arvind Sharma. Computer vision guided navigation system for visually impaired. Master's thesis, M.Tech Thesis, Advisor: Prakash Chand, June 2016. Verfügbar unter: https://www.researchgate.net/publication/306000500_COMPUTER_VISION_GUIDED_NAVIGATION_SYSTEM_FOR_VISUALLY_IMPAIRED.

[45] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. arXiv preprint arXiv:1507.05717, 2016. Verfügbar unter: https://arxiv.org/pdf/1507.05717.

[46] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556, 2014. Verfügbar unter: https://arxiv.org/pdf/1409.1556.

[47] Pritpal Singh and Sumit Budhiraja. Feature extraction and classification techniques in o.c.r. systems for handwritten gurmukhi script – a survey. International Journal of Engineering Research and Applications (IJERA), 1(4):1736–1739, 2011.

[48] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. arXiv preprint arXiv:1703.05175, 2017. Verfügbar unter: https://arxiv.org/pdf/1703.05175.

[49] Dipa Soni and Ashwin Makwana. A survey on mqtt: A protocol of internet of things (iot). In International Conference on Telecommunication, Power Analysis and Computing Techniques (ICTPACT - 2017). Bharath Institute of Higher Education and Research, Chennai, India, 2017. Verfügbar unter: https://www.researchgate.net/publication/316018571_A_SURVEY_ON_MQTT_A_PROTOCOL_OF_INTERNET_OF_THINGSIOT.

[50] Sergio Antonio Sánchez Hernández, H J Romero, and Alex David Morales Acosta. A review: Comparison of performance metrics of pretrained models for object detection using the tensorflow framework. IOP Conference Series: Materials Science and Engineering, 844:012024, June 2020. Verfügbar unter: https://www.researchgate.net/publication/342570032_A_review_Comparison_of_performance_metrics_of_pretrained_models_for_object_detection_using_the_TensorFlow_framework, Lizenz: CC BY 3.0.

[51] Juan R. Terven and Diana M. Cordova-Esparza. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. arXiv preprint arXiv:2304.00501v6, January 2024. Verfügbar unter: https://arxiv.org/abs/2304.00501.

[52] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. arXiv preprint arXiv:1904.01355, 2019. Verfügbar unter: https://arxiv.org/pdf/1904.01355.

[53] Rohit Verma. A survey of feature extraction and classification techniques in ocr systems. International Journal of Computer Applications & Information Technology, 1(3), November 2012. Verfügbar unter: https://www.researchgate.net/publication/348650070_A-Survey_of_Feature_Extraction_and_Classification_Techniques_in_OCR_Systems.

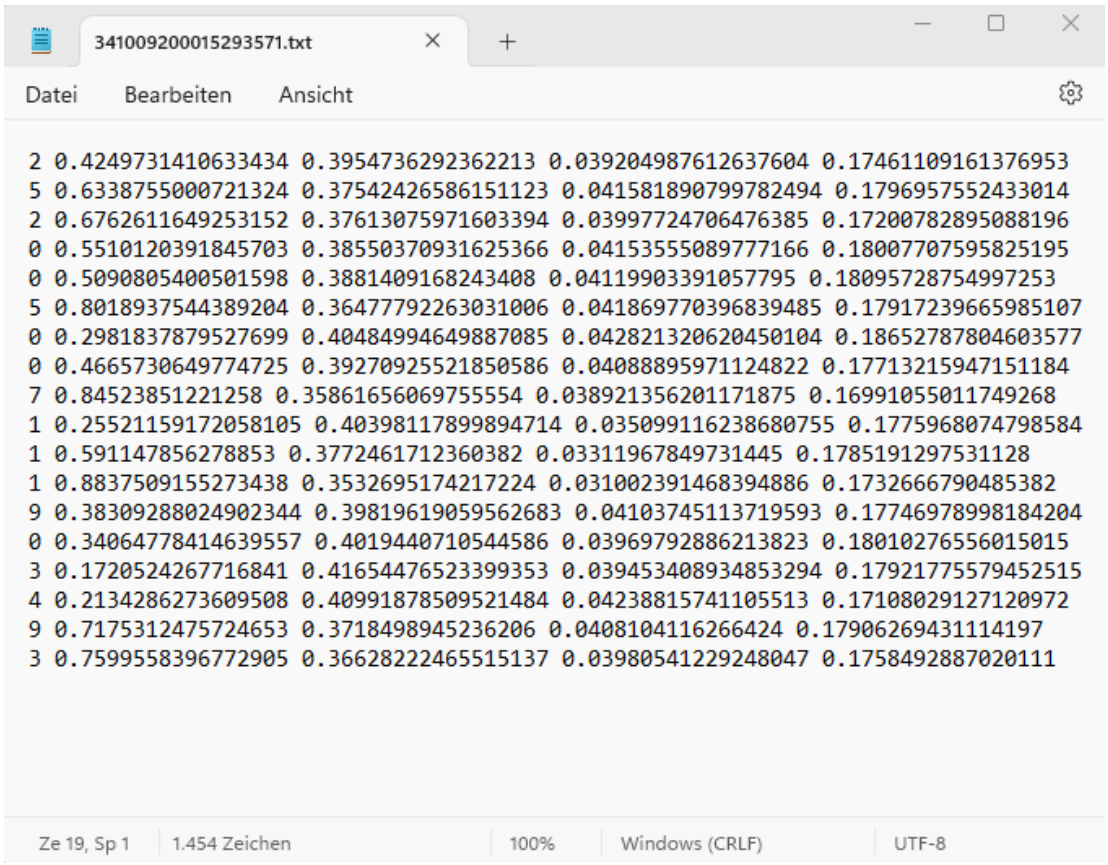
[54] wkentaro. labelme. https://github.com/wkentaro/labelme. Last accessed: October 11, 2024.

[55] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. arXiv preprint arXiv:1911.02685, 2020. Verfügbar unter: https://arxiv.org/pdf/1911.02685.

Anhang:

1. Anhang A: Thesis (PDF-Datei)
2. Anhang B: Quellcodes
3. Anhang C: Beispielannotation in YOLO-Format
4. Anhang D: Modellparameter YOLOv9e

Anhang C: Beispielannotation in YOLO-Format



Anhang D: Modellparameter YOLOv9e

```
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
bgr: 0.0
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
copy_paste_mode: flip
auto_augment: randaugment
erasing: 0.4
crop_fraction: 1.0
cfg: null
tracker: botsort.yaml
save_dir: runs/detect/train4
```

```
task: detect
mode: train
model: /content/drive/My Drive/yolov9e.pt
data: /content/drive/My Drive/config.yaml
epochs: 130
time: null
patience: 100
batch: 16
imgsz: 640
save: true
save_period: -1
cache: false
device: cuda
workers: 0
project: null
name: train4
exist_ok: false
pretrained: true
optimizer: auto
verbose: true
seed: 0
deterministic: true
single_cls: false
rect: false
cos_lr: false
close_mosaic: 10
resume: false
amp: true
fraction: 1.0
profile: false
freeze: null
multi_scale: false
overlap_mask: true
mask_ratio: 4
dropout: 0.0
val: true
split: val
save_json: false
save_hybrid: false
conf: null
iou: 0.7
max_det: 300
half: false
dnn: false
```

```
dnn: false
plots: true
source: null
vid_stride: 1
stream_buffer: false
visualize: false
augment: false
agnostic_nms: false
classes: null
retina_masks: false
embed: null
show: false
save_frames: false
save_txt: false
save_conf: false
save_crop: false
show_labels: true
show_conf: true
show_boxes: true
line_width: null
format: torchscript
keras: false
optimize: false
int8: false
dynamic: false
simplify: true
opset: null
workspace: 4
nms: false
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 7.5
cls: 0.5
dfl: 1.5
pose: 12.0
kobj: 1.0
label_smoothing: 0.0
nbs: 64
hsv_h: 0.015
hsv_s: 0.7
```

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Artur

Vorname: Voskanyan

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung eines dateneffizienten Deep Learning Verfahrens zur automatisierten Sendungsverfolgung in der Logistik

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

Hamburg (Ort) 11.10.2024 (Datum) [redacted]