Webvisualisierung von Prozesskomponenten in der Bildgebenden Qualitätskontrolle

Studienarbeit 2

des Studienganges Elektrotechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

von
Andreas Braig

02.01.2025

Bearbeitungszeitraum:	07.01.2025 - 07.04.2025
Matrikelnummer, Kurs:	6481829, TEL22AT1
Ausbildungsfirma:	ABB AG
Abteilung:	PAPI-EAM
Betreuer der Dualen Hochschule:	Prof. Dr.-Ing. Bozena Lamek-Creutz

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: „Webvisualisierung von Prozesskomponenten in der Bildgebenden Qualitätskontrolle“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Vorwort

Die vorliegende Studienarbeit wurde im Rahmen des Studiengangs Elektrotechnik an der DHBW erstellt.

An dieser Stelle möchte ich mich herzlich bei meiner Betreuerin Prof. Dr.-Ing. Bozena Lamek-Creutz und Herrn Shobit Agarwal für ihre wertvolle Unterstützung bedanken.

Zusammenfassung

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

CNN Convolutional Neural Network
API Application Programming Interface
PCB Printed Circuit Board
CP-Lab Cyber-Physical Lab
JSON JavaScript Object Notation
REST Representational State Transfer
HTTP Hypertext Transfer Protocol
URL Uniform Resource Locator
HTML Hypertext Markup Language
WSGI Web Server Gateway Interface
MVC Model-View-Controller
PC Personal Computer
USB Universal Serial Bus
SPS Speicherprogrammierbare Steuerung

1. Problemstellung und Ziel dieser Arbeit

Die zunehmende Automatisierung industrieller Prozesse erfordert zuverlässige Qualitätskontrollsysteme, insbesondere in der Fertigung von elektronischen Baugruppen wie PCBs. Im letzten Semester wurde in einer Machbarkeitsstudie untersucht ob ein KI-basiertes System umsetzbar ist, welches mittels TensorFlow Convolutional Neural Networks (CNNs) Defekte auf PCBs erkennt. Dieses System basiert auf einer Online-Implementierung Benutzeroberfläche.

Aktuell bestehen drei zentrale Herausforderungen: Erstens bietet die Online- Implementierung keine lokale Kontrolle über Parameter oder Daten, was die Flexibilität limitiert. Zweitens fehlt eine intuitive Schnittstelle zur Visualisierung von Klassifizierungsergebnissen, was die Benutzerinteraktion erschwert. Drittens soll die Leistung der bisher verwendeten CNN-Architektur evaluiert werden und weitere Architekturen oder Optimierungstechniken sollen getestet werden, um die Erkennungsgenauigkeit zu verbessern. Hierfür stehen eine Reihe neuer PCBs zur Verfügung, die in einer zu dieser Arbeit parallelen Studienarbeit entwickelt wurden.

Ziel dieser Arbeit ist es, die bestehende Lösung in eine lokale Anwendung zu überführen, die folgende Kernkomponenten integriert: Eine zentrale Parametrisierung. Diese ermöglicht die flexible Steuerung aller Modell- und Systemparameter, während eine modularisierte Codebasis die Wartbarkeit und Wiederverwendbarkeit des Python-Codes verbessert. Zusätzlich soll eine Webanwendung mit einer Python Application Programming Interface (API) entwickelt werden, die eine Darstellung in echtzeit von PCBs Klassifizierungsergebnissen bietet. Parallel erfolgt eine systematische Modellre-Evaluation, bei der das aktuelle CNN mit alternativen Architekturen oder Optimierungstechniken verglichen wird.

Durch diese Maßnahmen soll die Darstellung der industriellen Anwendbarkeit im FESTO CP Lab gestärkt werden. Die neue Webvisualisierung soll greifbar machen, was bildverarbeitende Qualitätskontrolle bedeutet, indem sie in echtzeit Analyse und Ergebnisse der PCBs anschaulich darstellt. Benutzer können direkt sehen, dass Defekte erkannt und klassifiziert werden, was die Transparenz und Nachvollziehbarkeit des gesamten Prozesses erhöht.

2. Theoretische Grundlagen

In diesem Abschnitt werden die theoretischen Grundlagen erläutert, die für das Verständnis der späteren Kapitel notwendig sind. Dazu gehören die Funktionsweise von Machine Learning und Computer Vision, sowie die Grundlagen einer API. Wie bereits im Ersten Kapitel (siehe Kapitel 1) beschrieben, baut diese Studienarbeit auf der Arbeit des letzten Semesters auf. Die theoretischen Grundlagen für Machine Learning und Computer Vision werden hier nur kurz erläutert, da sie bereits im letzten Semester ausführlich behandelt wurden.

Die Funktionsweise einer API wird in diesem Kapitel genauer erläutert, da sie eine zentrale Rolle in dieser Studienarbeit spielt. Mithilfe einer Web-API werden die Daten des Python programmes an die entwickelte Webanwendung übertragen.

2.1. Machine Learning und Klassifikationsprobleme

Grundlegend, bevor die Datensätze in Form von Bildern durch Convoliutional Neural Networks CNN analysiert werden können, müssen die Daten verarbeitet werden.

Bildverarbeitung beschäftigt sich mit der Manipulation und Analyse digitaler Bilder durch Algorithmen und bildet die Grundlage für komplexere Verfahren. Ein digitales Bild wird als mehrdimensionale Matrix gespeichert, wobei farbige Bilder als dreidimensionale Tensoren dargestellt werden, deren Dimensionen Höhe, Breite und Farbkanäle repräsentieren. Diese Repräsentation ermöglicht die Anwendung verschiedener Transformationen wie Filterung, Kontrastverbesserung oder geometrische Verzerrungen, die entweder zur Bildverbesserung oder als Vorverarbeitungsschritte für nachfolgende Analysen durch fortschrittlichere Techniken wie Machine Learning und Computer Vision dienen [1].

Convolutional Neural Networks CNN sind spezialisierte Deep Learning Modelle, die für die Verarbeitung von Bilddaten optimiert sind. Sie bilden die Grundlage für zahlreiche moderne Computer Vision Anwendungen, wie Gesichtserkennung und autonome

Fahrzeuge. Die Architektur eines CNN nutzt die räumliche Struktur von Bildern effizient, um visuelle Muster zu erkennen und zu klassifizieren. Ein CNN besteht hauptsächlich aus Faltungsschichten und Linearen Ebenen (siehe Abbildung 2.1). Die Convolutional Layers verwenden kleine Filtermatrizen (Kernel), die über das Bild gleiten und visuelle Merkmale wie Kanten und Texturen erkennen. Diese Filter werden während des Trainings optimiert, um die relevantesten Merkmale zu extrahieren [1] [2].

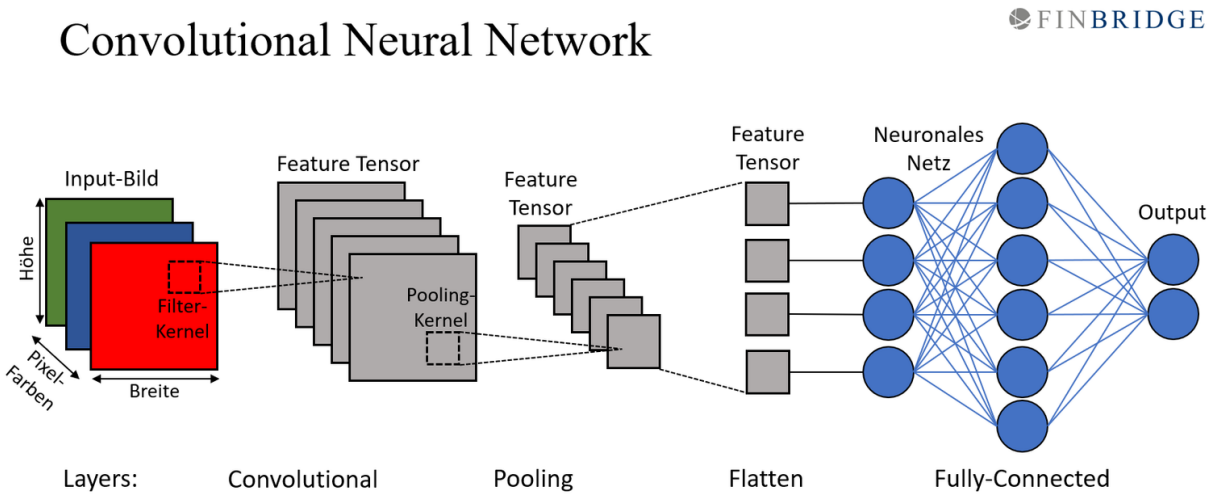


Abbildung 2.1.: Aufbau eines Convolutional Neural Networks [1].

Die Fully-Connected Layers am Ende des Netzwerks (siehe Abbildung 2.1) wandeln die extrahierten Merkmale in eine endgültige Klassifikation oder Vorhersage um. Diese Schichten sind ähnlich wie traditionelle neuronale Netzwerke aufgebaut, wobei jedes Neuron mit allen Neuronen der vorherigen Schicht verbunden ist.

Die wichtigsten Lernansätze im maschinellen Lernen sind überwachtes, unüberwachtes und verstärkendes Lernen. In dieser Arbeit wird überwachtes Lernen genutzt, bei dem ein Algorithmus mit gelabelten Daten trainiert wird, um aus diesen Beispielen zu lernen und Vorhersagen zu treffen.

Beim überwachten Lernen gibt es zwei Hauptmodelle: Klassifikation und Regression. Regression beschreibt kontinuierliche Zusammenhänge zwischen Eingangs- und Ausgangsdaten [3]. Klassifikation teilt Daten in diskrete Gruppen ein. Es sollen keine Kontinuierlichen werte nachgebildet werden. Am Ende des Netzwerks wird mithifge einer $argmax()$ Funktiion eine Klasse fest zugeordnet [4, S. 450]. Die binäre Klassifikation, welche in dieser Arbeit angewandt wird ist eine spezielle Form der Klassifikation, bei der nur zwei Klassen unterschieden werden.

Es gibt verschiedene CNN-Architekturen wie ResNet und MobileNet, die für spezifische Aufgaben besonders gut geeignet sind. Oft werden vortrainierte Modelle verwendet und für spezifische Anwendungsfälle feinabgestimmt, eine Technik bekannt als Transfer Learning, welche auf dem überwachten Lernen basiert. [1].

2.2. Die Funktionsweise einer API

APIs stellen das Herzstück moderner Softwareentwicklung dar und ermöglichen es Programmen, miteinander zu kommunizieren und Daten auszutauschen. Dieser Datenaustausch ist wesentlich für vielfältige Anwendungen, wie etwa das Abrufen von Wetterdaten oder das Interagieren mit sozialen Netzwerken innerhalb einer fremden Anwendung. In der Python Entwicklungsumgebung machen Bibliotheken wie "requests"oder "http.client"den Einstieg in die API-Entwicklung besonders zugänglich [5]. Die in dieser Studienarbeit verwendete FLASK Bibliothek ermöglicht es, eine komplexere API zu erstellen, die Daten aus dem Python Programm an die Webanwendung überträgt.

Anwendungsprogrammierschnittstellen API sind Software-Vermittler (Abbildung 2.2) ihre Aufgabe besteht darin, Anwendungen die Kommunikation untereinander zu ermöglichen. Diese subtilen Vermittler sind allgegenwärtig im täglichen Leben, ob bewusst wahrgenommen oder nicht [6].

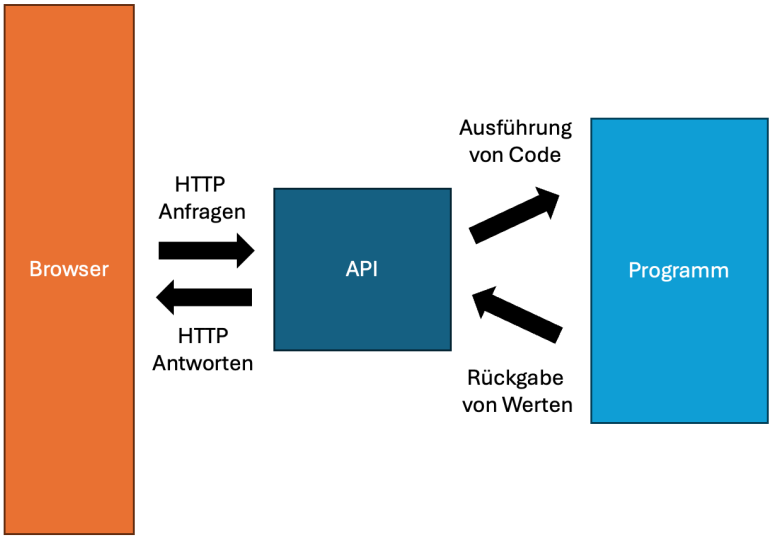


Abbildung 2.2.: Schematischer Aufbau der API Kommunikation

Im Kontext von Webanwendungen bieten APIs den Zugriff auf Daten und Funktionali-

täten von Drittanbietern. Dies ermöglicht es Entwicklern, ihre Anwendungen um Funktionen wie Wetterinformationen, Sportergebnisse, Filmlisten, Tweets, Suchmaschinenergebnisse und Bildverarbeitung zu erweitern. Die Eigenentwicklung solcher Funktionen würde erhebliche Ressourcen beanspruchen, während die Nutzung von APIs eine schnelle und effiziente Integration ermöglicht[7].

2.2.1. Anfragearten einer API

Die Grundbausteine einer API sind Anfragen (Requests) und Antworten (Responses). Diese basieren oft auf dem Hypertext Transfer Protocol (HTTP)-Protokoll, das die Grundlage des Internets bildet [5]. HTTP-Anfragen sind die Art und Weise, wie das Web funktioniert. Jedes Mal, wenn man zu einer Webseite navigiert, stellt der Browser mehrere Anfragen an den Server der Webseite. Der Server antwortet dann mit allen Daten, die für die Darstellung der Seite erforderlich sind, woraufhin der Browser die Seite darstellt [7].

Der generische Prozess der API-Kommunikation lässt sich wie folgt beschreiben: Ein Client, in dieser Studienarbeit der Browser, sendet Daten an eine Uniform Resource Locator Uniform Resource Locator (URL). Der Server unter dieser URL liest die Daten, entscheidet, was damit zu tun ist. Intern wird das passende Python Skript ausgeführt und gibt eine Antwort an den Client zurück. Schließlich verarbeitet der Client die empfangenen Daten entsprechend seiner Programmlogik [7].

Ein wesentlicher Teil der Anfrage ist die Hypertext Transfer Protocol HTTP-Methode. Einige der gebräuchlichsten Methoden sind [8]:

- **GET** Dient dem Abrufen von Daten, ohne Änderungen auf dem Server vorzunehmen
- **POST** Wird verwendet, um neue Daten an den Server zu senden
- **PUT** Aktualisiert vorhandene Daten auf dem Server
- **DELETE** Entfernt Daten vom Server

Diese Methoden bilden die Grundlage des Representational State Transfer Representational

State Transfer (REST)ful API-Designs, das in modernen Webanwendungen weit verbreitet ist [8].

2.2.2. FLASK

Für diese Studienarbeit ist keine Umfängliche WEB-API notwendig. Es soll lediglich eine einzige Hypertext Markup Language (HTML) Datei von der Python Anwendung an den Browser übertragen werden. Es ist für diese Anforderung kein Größeres Framework wie Beispielsweise Django notwendig. FLASK ist ein Mikroframework für Python, das sich auf einfache und schnelle Entwicklung konzentriert. Es ist besonders gut geeignet für kleine bis mittelgroße Projekte, bei denen die Verwendung eines größeren Frameworks übertrieben wäre. FLASK bietet eine Vielzahl von Erweiterungen, die die Entwicklung von Webanwendungen erleichtern. Es ist einfach zu erlernen und bietet eine Vielzahl von Funktionen, die für die Entwicklung von Webanwendungen erforderlich sind.

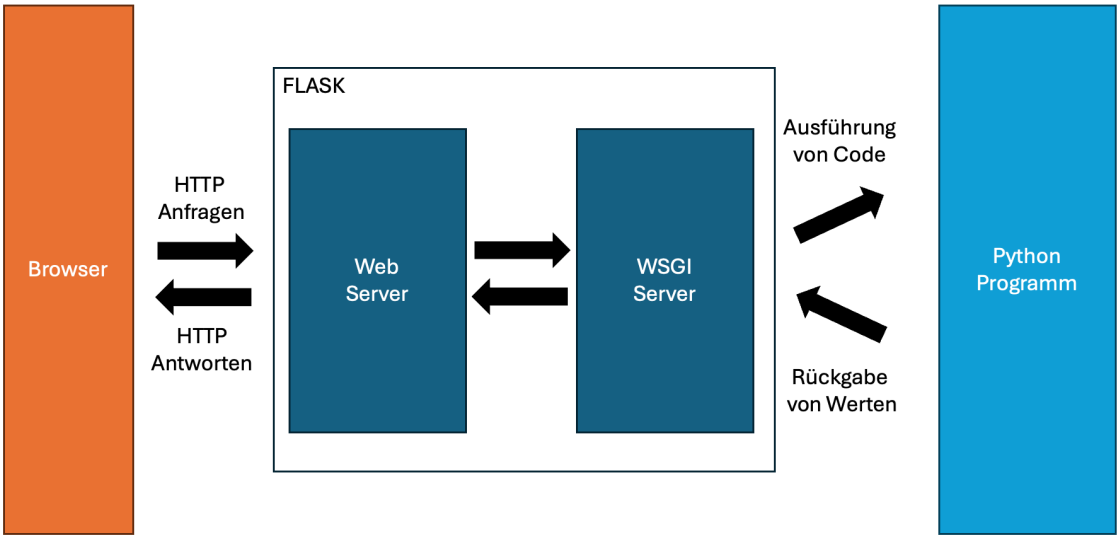


Abbildung 2.3.: Erweiterung der API Darstellung auf Basis von FLASK

FLASK Nutzt die Web Server Gateway Interface (WSGI) Schnittstelle von Python [9] und Abbildung 2.3 und Stellt in Verbindung mit der in dieser Studienarbeit Verwendeten Waitress Bibliothek den Webserver zur Verfügung.

Eine Besondere Anforderung der im Folgenden vorgestellten Webanwendung liegt in der Übertragung der neu erstellten Fotos von der Kamera des FESTO Cyber-Physical Lab (CP-Lab) an die Webanwendung. Um eine einfachere Benutzbarkeit zu gewährleisten soll der Browser nicht jedes mal neu geladen werden müssen, wenn ein Foto geschossen und klassifiziert wird.

Dies geschieht durch die Verwendung von Websockets. Websockets sind eine Technologie, die es ermöglicht, eine bidirektionale Verbindung zwischen einem Client und einem Server herzustellen. FLASK SocketIO ist eine Erweiterung für FLASK, die die Verwendung von Websockets in FLASK Anwendungen ermöglicht und wird in dieser Studienarbeit verwendet, um die Übertragung der Fotos zu realisieren. Auf der gegenseite im HTML Code wird die Bibliothek SocketIO.js verwendet. Diese Basiert auf der Programmiersprache JavaScript und ermöglicht die Kommunikation zwischen dem Browser und dem Server.

Im Folgenden wird die neue Softwarearchitektur vorgestellt und auch die Implementierung der Webanwendung erläutert.

3. Softwarekonzept

Dieser Studienarbeit liegt der, in der letzten Studienarbeit beschriebene Code, zugrunde. Dieser wurde im Rahmen einer Machbarkeitsstudie entwickelt und untersucht. Zentraler Fokus des Softwarekonzepts liegt auf der Präsentierparkeit und Wiederverwenbarkeit dieser Machbarkeitsstudie als Anwendung für die FESTO CP-Lab.

Um eine größere Auswahl an Möglichkeiten, sowie eine bessere Kontrolle über die Software zu erhalten, wird die Software in eine lokale Umgebung verschoben. Zusätzlich sollen alle einstellbaren Parameter zentral aufgeführt werden. Dies ermöglicht eine einfache Konfiguration der Software und entspricht dem Stand der Technik[10] [11].

3.1. Programmstruktur

Erster Teil der Softwarekonzeption ist der Entwurf der Struktur. Die Struktur einer Software ist entscheidend für die Wartbarkeit und Erweiterbarkeit. Zwei zentrale Kriterien, deren Erhaltung im gesamten Entwurfsprozess berücksichtigt werden muss.

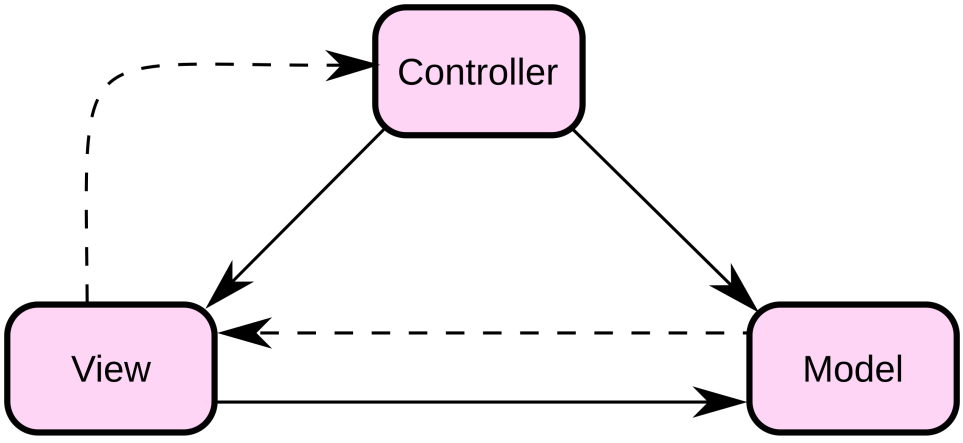


Abbildung 3.1.: Schematische Darstellung der MVC Struktur [12]

Im Rahmen dieser Studienarbeit gibt es keine externen Anforderungen an die Struktur. Daher wurde sich für eine vereinfachte Model-View-Controller (MVC) Struktur entschie-

den (Abbildung 3.1). Ziel dieser Struktur ist es die Software in drei Teile zu unterteilen. Diese drei Teile sollen eigenständige Aufgaben übernehmen und so verhindern, dass das Programm zu einem monolithischen Codeblock wird.

Der Model-Teil ist für die (Bild-) Datenverarbeitung zuständig und enthält die Datenstrukturen sowie die Logik, die die Daten verarbeitet. Der View-Teil ist für die Darstellung der Daten verantwortlich und umfasst die Benutzeroberfläche sowie die Logik, die die Daten darstellt. Der Controller-Teil steuert die Daten und koordiniert die Kommunikation zwischen Model und View, indem er die entsprechende Logik enthält.

Die vereinfachte Version der MVC Struktur kombiniert die Funktionalitäten von View und Controller in der API (Siehe Abbildung 3.1) und trennt die Datenverarbeitung in einem eigenes Modul ab. Dieses eigene Modul wird im Programmcode `Pycore` genannt und enthält die Funktionen, welche die Daten für das Neuronale Netzwerk aufbereiten und zur Verfügung stellen.

3.2. Konfiguration der Programmparameter

Aus der Aufgabenstellung (siehe Kapitel 1) lässt sich ein weiterer Teil des Softwarekonzepts, die Einstellbarkeit, ableiten. Durch die Einstellbarkeit soll gewährleistet sein, dass die Software flexibel an die Anforderungen des Benutzers angepasst werden kann, ohne dass der Benutzer ein tiefes Verständnis der Software haben muss.

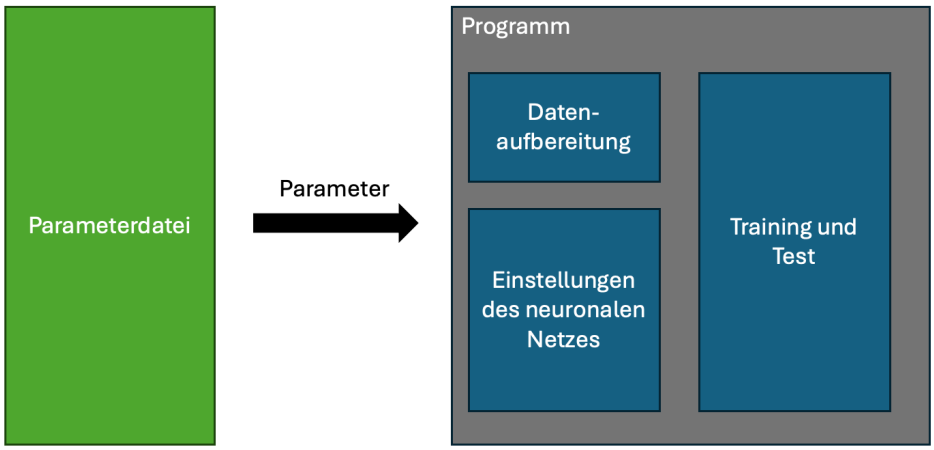


Abbildung 3.2.: Konzept der Parametergruppierung- und Verteilung auf das Programm

Ziel ist es das der Benutzer eine einzige Datei öffnet und dort alle Einstellungen vornehmen kann. Diese Datei soll im JSON-Format vorliegen, da es ein weit verbreitetes Format ist und von vielen Programmiersprachen unterstützt wird [10].

3.3. Die Weboberfläche mittels Python Web API

Das zentrale Element des neuen Programmentwurfs stellt die Webanwendung dar. Dieses neue Interface zwischen Benutzer und Programm ermöglicht es neue Bilder des FESTO CP-Lab darzustellen, zu klassifizieren und die Ergebnisse auf seiner Weboberfläche visualisieren. Zusätzlich koordiniert sie die Überwachung des Ordners für neue Bilder und die Klassifizierung dieser Bilder in Defekt oder nicht Defekt.

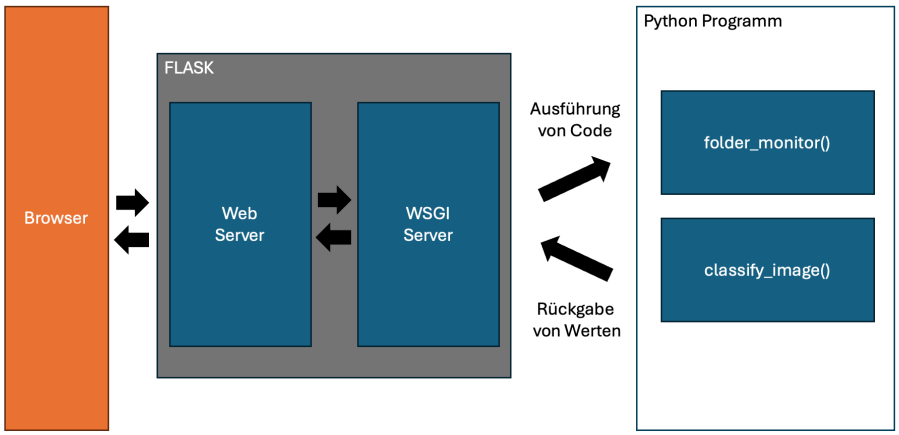


Abbildung 3.3.: Aufschlüsselung der API Funktionen

Der Aufbau des Programmteils folgt dem funktionalen Prinzip einer API, welche in Abschnitt 2.2 beschrieben ist. Die Abbildung 3.3 erweitert die Abbildung 2.3 um die beiden Haputfunktionen, welche die API innerhalb des Python Programmes ausführt. Der Benutzer wird in der Lage sein das aktuelle Werkstück im Browser zu sehen, ohne sich mit dem Programmcode auseinandersetzen zu müssen. Hierfür sollen parallel mehrere Prozesse gestartet werden, welche die API, überwachung des Ordners und klassifizierung des aktuellen Bildes übernehmen.

4. Implementierung

Nachdem im letzten Kapitel (siehe Kapitel 3) das Konzept der Lösung der Aufgabenstellung beschrieben wurde, wird in diesem Kapitel die Implementierung der Software beschrieben. Ziel ist es, alle beschriebenen Aufgabenteile in Python umzusetzen und eine Software zu entwickeln, die in der Lage ist, die Anforderungen der Aufgabenstellung zu erfüllen. Ausschnitte aus dieser Software sind im Kapitel des Konzepts bereits beschrieben und werden hier weiter ausgeführt.

Die Implementierung erfolgt in mehreren Schritten:

1. **Umzug auf lokale Programmierung**
2. **Einpflegen der JSON-Parameter**
3. **Entwicklung der API**
4. **Installation im Labor**

In den folgenden Kapiteln werden die einzelnen Schritte genauer beschrieben.

4.1. Umzug auf lokale Programmierung

Die Struktur dieser Implementierung folgt der Konzeption der Programmstruktur nach dem MVC-Prinzip Abbildung 3.1, welches in Abschnitt 3.1 beschrieben ist. Dieses Modell trennt die Datenverarbeitung, die Darstellung und die Steuerung der Software in drei verschiedene Module auf. Die Datenverarbeitung wird in einem eigenen Modul abgetrennt, Darstellung und Steuerung sind in der API zusammengefasst.

Das Pycore-Modul, welches diese benötigten, selbst entwickelten Bibliotheken zur Verfügung stellt, ist auch in Abbildung 4.1 als eigener Ordner mit Python-Skripten in seinen Unterverzeichnissen dargestellt. Es beinhaltet die Funktionen, welche den Datensatz

aufbereiten und dem neuronalen Netzwerk zum Training zur Verfügung stellen. Die wichtigsten Bestandteile sind die Konvertierung in Graustufen, die Aufteilung des Datensatzes in Trainings- und Testdaten sowie der Download des untrainierten neuronalen Netzwerks und dessen Parametrisierung bezüglich der Klassifikationsaufgabe.

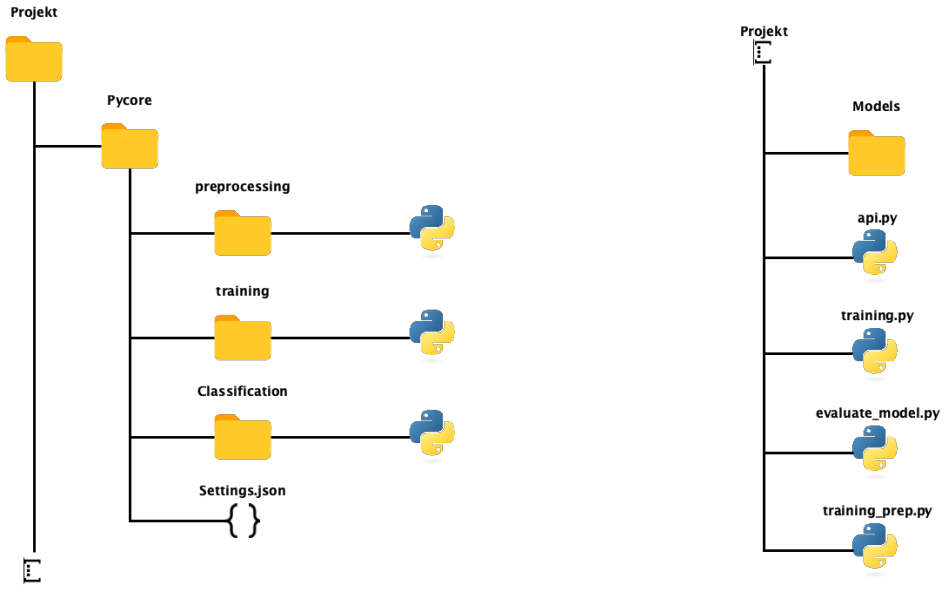


Abbildung 4.1.: Projektstruktur der Software im Dateiverzeichnis

Die in der Grafik Abbildung 4.1 dargestellte Dateistruktur repräsentiert ebenso die Struktur der Software. Die Funktionen werden in Bibliotheken im Pycore-Verzeichnis zusammengefasst. Sämtliche Daten werden in einem ausgegliederten Verzeichnis innerhalb des Projektverzeichnisses abgelegt.

Im Projekthauptverzeichnis gibt es, neben des Pycore-Moduls und der Verzeichnisse für Trainings- und Testdaten, vier Python-Skripte. Diese können ausgeführt werden, um einzelne Teile der Software auszuführen. `api.py` wird ausgeführt, um die Weboberfläche zu starten. `train.py` wird ausgeführt, um das Modell zu trainieren. `evaluate_model.py` wird ausgeführt, um das Modell zu evaluieren und die in Unterabschnitt 5.1.2 beschriebenen Werte zu erhalten. `training_prep.py` kann ausgeführt werden, um den Datensatz zu verarbeiten, ohne das Modell zu trainieren.

Diese Skripte laden den jeweils benötigten Programmteil und die benötigten Parameter aus dem Pycore-Verzeichnis, führen die entsprechenden Funktionen aus und legen die Ergebnisse im Dateisystem ab. Keines dieser Skripte ist über die Webanwendung zu

erreichen, da sie nur für die interne Verwendung gedacht sind.

4.2. Einpflegen der JSON-Parameter

Die Umsetzung des Konzeptes der Konfiguration der Programmparameter (siehe Abschnitt 3.2) erfolgt in Python mittels einer JSON-Datei. Diese Datei enthält alle Parameter, die für die Software benötigt werden.

```
1   {
2       "filepaths": {
3           "good": "Bilder/Good_Pictures",
4           "bad": "Bilder/Bad_Pictures",
5           "good_gray": "Bilder/Good_Grayscale",
6           "bad_gray": "Bilder/Bad_Grayscale",
7           "train": "Bilder/train",
8           "test":"Bilder/test",
9           "validate":"Bilder/validate",
10          "new": "Bilder/new"
11      }
12  }
```

Listing 4.1: Beispiel einer JSON-Datei mit Parametern des mobilnet Modells

Die Struktur der JSON-Datei ist in Listing 4.1 dargestellt und wird in Python mittels des `json` Moduls eingelesen. Der hier dargestellte Ausschnitt beinhaltet alle wichtigen Dateipfade, um dem Python-Programm Zugriff zu den Trainings- und Testdaten sowie dem überwachten Ordner für neue Bilder zu gewähren. Er dient beispielhaft für das gesamte Programm. Ein Beispiel für das Einlesen der Datei ist in Listing 4.2 dargestellt.

```
1   import json
2
3   config_path = "pycore/setings.json"
4   cf = json.load(open(config_path, 'r'))
5
6   # Uebergabe der Parameter an die Funktionen
7   uic.folder_to_grayscale(cf["filepaths"]["good"],cf["filepaths"]["good_gray"])
8   uic.folder_to_grayscale(cf["filepaths"]["bad"],cf["filepaths"]["bad_gray"])
```

Listing 4.2: Einlesen der JSON-Datei

In diesem Codeausschnitt wird der Pfad zur Konfigurationsdatei festgelegt (Zeile 3) und die Datei wird eingelesen (Zeile 4). Die Parameter werden dann an die erste Funktion übergeben, welche die Bilder in Graustufen umwandelt und in einem separaten Ver-

zeichnis ablegt (Zielverzeichnis siehe Listing 4.2). Die Syntax für das Anwählen des Parameters ist `cf["filepaths"]["good"]`, wobei `cf` die Variable ist, in der die JSON-Datei eingelesen wurde. Die JSON-Datei wird als Array eingelesen, wobei der Zugriff auf die einzelnen Parameter über den Schlüssel in Textform erfolgt. In diesem Fall wird der Pfad zum Ordner mit den guten Bildern über den Schlüssel `"good"` angesprochen, während `"filepaths"` den Zugriff auf die passende Sektion des Arrays ermöglicht.

Angenommen der Benutzer wünscht ein anderes Verzeichnis für die Bilder, so kann er dies in der JSON-Datei ändern und die Software erneut ausführen, ohne zu wissen, wo die Funktion, welche den Datensatz generiert, abliegt. Der gesamte Code dieses Projekts wurde nach diesem Schema aufgebaut, um eine einfache und einheitliche Konfiguration zu gewährleisten.

4.3. Entwicklung der API

In diesem Abschnitt wird die Entwicklung der API beschrieben, welche die Kommunikation zwischen der Weboberfläche und der Backend-Logik der Software ermöglicht. Es wird sichergestellt, dass die Entwicklung dem Konzept der Aufgabenstellung Abschnitt 3.3 entspricht.

Hierfür werden in zwei verschiedenen Threads zwei Funktionen ausgeführt werden. Die erste Funktion lädt das trainierte Modell in den Zwischenspeicher, um es für die Klassifizierung nutzbar zu machen (Zeile 3 Listing 4.3). Im zweiten Thread wird der Watchdog gestartet, welcher den Ordner für neue Bilder überwacht und bei neuen Bildern die Klassifizierung startet (Zeile 6).

```
1  if __name__ == '__main__':
2
3      model_thread = threading.Thread(target=load_model, daemon=True)
4      model_thread.start()
5
6      watchdog_thread = threading.Thread(target=start_watchdog, daemon=True)
7      watchdog_thread.start()
8
9      socketio.run(app, host="127.0.0.1", port=5000,debug=True,use_reloader=False)
```

Listing 4.3: Start der Weboberfläche und der Threads in der api.py

Ziel des Threadings ist es, den Ordner für neue Bilder permanent zu überwachen, ohne zyklisch abzufragen. Dies spart Rechenleistung und ermöglicht eine schnellere Reaktion auf neue Bilder [13].

Parallel zu den beiden Threads wird die Weboberfläche in Zeile 9 des Listing 4.3 gestartet. Bei Aufruf der Adresse, welche ebenfalls in Zeile 9 dargestellt ist, wird eine vorher definierte HTML-Seite an den Browser des Clients mittels GET Anfrage (siehe Unterabschnitt 2.2.1) gesendet. Diese Seite enthält ein JavaScript, welches die Verbindung zum Websocket des Python-Programms herstellt und einen bidirektionalen Datenaustausch ermöglicht.

```
1  <script>
2      var socket = io();
3      socket.on('update_image', function(data) {
4          console.log("Neues Bild-Event empfangen:", data);  // Debugging
5          document.getElementById('latestImage').src = data.image_url + "?t=" + new Date
               ().getTime();
6      });
7      socket.on('classification_result', function(data) {
8      document.getElementById("classification-result").innerText = data.result;
9      });
10 </script>
```

Listing 4.4: JavaScript Code der Weboberfläche, welcher die Bilder des Websockets empfängt

Sobald ein neues Bild in den Ordner `Bilder/new` gelegt wird, wird ein Event ausgelöst, welches sowohl eine Klassifizierung herbeiführt als auch das Bild mittels Websocket an das JavaScript-Backend der Weboberfläche sendet.

4.4. Installation im Labor

Die Installation verläuft in drei Phasen: Übertragung, Installation und Tests. Zunächst wird die Software auf einem Universal Serial Bus (USB)-Stick in das Labor gebracht und auf den Laborrechner kopiert. Der Laborrechner ist Teil des Lieferumfangs des FESTO CP-Lab Systems und steht dieser Studienarbeit zur Verfügung.

Dieser Personal Computer (PC) wird ausschließlich für die Steuerung der FESTO-Anlage verwendet und verwaltet die Datenbank, die Konfiguration der Stationen und

die Programmierung der Speicherprogrammierbare Steuerung (SPS)en. Python ist auf diesem Rechner bereits installiert, jedoch fehlen die benötigten Pakete für die Software. Alle benötigten Pakete werden in einem Python Virtual Environment installiert, um die Installation dieser Studienarbeit zu isolieren und die Funktionalität des bestehenden Systems nicht zu beeinträchtigen [14].

Um eine Speicherung der von der Kamera aufgenommenen Bilder zu ermöglichen, muss innerhalb der Konfigurationssoftware der Kamera ein Archivierungspfad festgelegt werden. Dieser Pfad wird in der Software als Zielverzeichnis für die Klassifizierung genutzt.

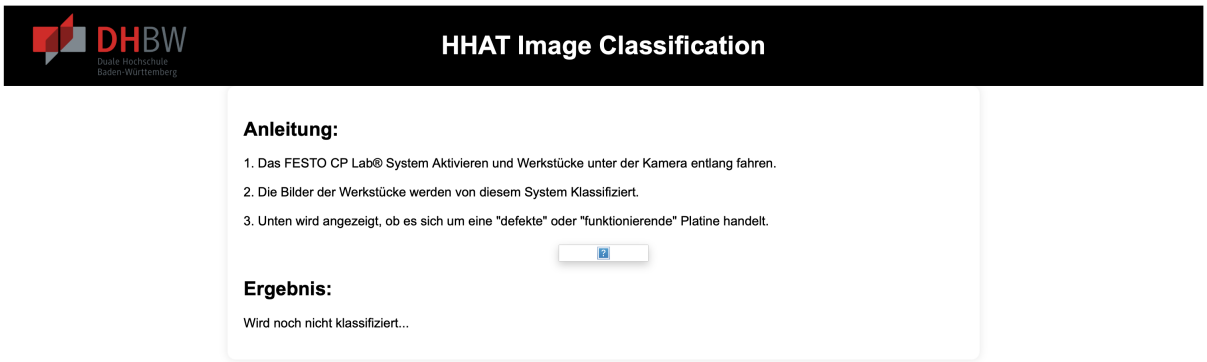


Abbildung 4.2.: Darstellung der Weboberfläche im Zustand ohne Klassifikation eines aktuellen PCBs.

Nach der Installation der benötigten Pakete wird die Software getestet. Die Interpretation des übertragenen Python-Codes ist fehlerfrei und der Testbetrieb kann aufgenommen werden. Die finale Gestaltung der Weboberfläche ist in Abbildung 4.2 dargestellt.

4.4.1. Aufgetretene Probleme

Während der Installation und des Testbetriebs traten zwei Probleme auf, deren Lösung im Folgenden erklärt wird.

Im Testbetrieb fällt auf, dass die Software Bilder nicht korrekt an die Webanwendung übermittelt. Der Grund für das Auftreten dieses Fehlers war ein überschnelles Reagieren des Watchdogs auf neue Bilder. Dieser erkennt eine Datei, bevor sie vollständig in den Ordner übermittelt wurde. Dieses Problem wird durch eine Verzögerung der

Bildübermittlung an den Webserver und den Klassifikator behoben. Die Übermittlung startet eine halbe Sekunde nach dem Auslösen des Events.

Der Vision-Sensor, welcher ebenfalls im Lieferumfang des FESTO CP-Lab Systems enthalten ist, kann mittels der Visor Vision Software verbunden werden [15]. Diese Software ermöglicht eine Anzeige und Konfiguration des Sensors über den Laborrechner. Eine automatisierte Übertragung mittels direkter Verbindung zwischen Laborrechner und Vision-Sensor ist nicht möglich, da dies die Konfiguration des Sensors und damit die Funktionalität des CP-Lab Systems beeinträchtigen würde.

Um diesen Umstand zu umgehen, wird die Archivierung in der SensoView Software aktiviert. Der hierdurch entstehende Nachteil ist, dass die SensoView Software gestartet und angemeldet sein muss, um die Bilder zu speichern (siehe Anhang Abschnitt A). Zum jetzigen Stand ist eine permanente Lösung ohne Hilfe von FESTO nicht möglich.

4.4.2. Neuer Datensatz

Parallel zu dieser Studienarbeit wurde eine weitere Studienarbeit durchgeführt, welche sich mit dem Design neuer PCBs beschäftigt. Die detaillierten Änderungen und der Hergang des Designprozesses sind in der Studienarbeit von Herrn Lucas Weyland beschrieben und können dort nachgelesen werden. In dieser Studienarbeit wird diese Änderung als gegeben betrachtet und der neue Datensatz wird für die Klassifikation verwendet. Entsprechend dazu müssen neue Trainingsdaten generiert und das Modell neu trainiert werden.

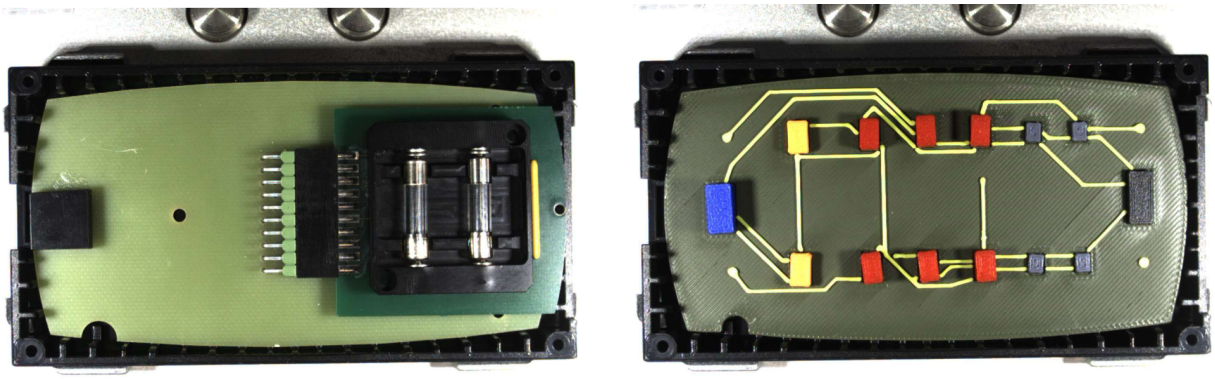


Abbildung 4.3.: Gegenüberstellung der alten und neuen PCBs

Wie auch auf der Abbildung 4.3 zu sehen ist, bieten die neuen PCBs weit mehr Details

und sind daher komplexer als die PCBs, die für die Machbarkeitsstudie verwendet wurden. Dies führt zu mehr Möglichkeiten, die Möglichkeiten des Modells zu testen und die Klassifikation zu verbessern. Die Reevaluierung des Modells mit den neuen Daten wird in Abschnitt 5.2 beschrieben.

5. Softwaretests

In diesem Kapitel wird auf die Evaluierungs- und Testvorgäne für die Sicherstellung der Güte der entwickelten Software eingegangen. Da es sich bei der entwickelten Software um ein Machine-Learning-Modell handelt, wird in diesem Kapitel auf die Evaluierung des Modells eingegangen. Die Funktionen des interpretierten Python Skripts entwickeln keine neuen Algorithmen, die auf ihre Fehleranfälligkeit oder Korrektheit getestet werden müssen.

5.1. Metriken zur Evaluierung

Um zu verstehen wie ein Machine Learning Modell evaluiert werden kann ist es zunächst wichtig die Arten von Kriterien nachzuvollzihen, die für die binäre Klassifikation notwendig sind. Im Folgenden wird allgemein von Instanzen gesprochen, es handelt sich dabei um die Bilder, der PCBs die klassifiziert werden sollen.

- **True Positive (TP)**: Die Anzahl der korrekt klassifizierten positiven Instanzen.
- **True Negative (TN)**: Die Anzahl der korrekt klassifizierten negativen Instanzen.
- **False Positive (FP)**: Die Anzahl der falsch klassifizierten positiven Instanzen.
- **False Negative (FN)**: Die Anzahl der falsch klassifizierten negativen Instanzen.

Diese wesentlichen Typen teilen die Klassifizierten Daten nach dem Test in vier diskrete Kategorien ein. Diese Kategorien bilden die in Unterabschnitt 5.1.2 beschriebene Confusion Matrix. Mit Ihnen können aber auch die Einfacheren Werte Loss und Accuracy berechnet werden. Im Folgenden wird die Mathematik hinter den Methoden vorgestellt.

5.1.1. Accuracy und Loss

Die Accuracy ist eine der einfachsten Metriken zur Evaluierung eines Machine-Learning-Modells. Sie gibt an, wie viele der Instanzen korrekt klassifiziert wurden. Die Accuracy wird vereinfacht wie folgt berechnet:

$$\text{Accuracy} = \frac{n_{\text{richtig klassifiziert}}}{n_{\text{gesamt}}} \tag{5.1}$$

Bezug auf die in Abschnitt 5.1 beschriebenen Kategorien, kann die Accuracy wie folgt berechnet werden [16]:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

Die Accuracy wird typischerweise in Prozent angegeben und liegt zwischen 0 und 100%. Sie ermöglicht eine schnelle Einschätzung der Modellgüte. Allerdings kann die Accuracy irreführend sein, da sie die Anzahl der falsch klassifizierten Instanzen nicht berücksichtigt. Ein Modell, das alle Instanzen als negativ klassifiziert, könnte eine hohe Accuracy aufweisen, obwohl es nicht leistungsfähig ist.

Dennoch ist die Accuracy einfacher zu Interpretieren als der hier vorgestellte Loss. Der Loss ist eine Metrik, die die Güte eines Modells anhand der Wahrscheinlichkeiten der Klassifikationen bewertet. Für binäre Klassifikationen wird der Binary Crossentropy Loss verwendet, der wie folgt berechnet wird [16]:

$$\text{Loss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{5.3}$$

Symbol	Bedeutung
n	Anzahl der Trainingsbeispiele
k	Anzahl der Klassen (bei Mehrklassen-Klassifikation)
y_i	Wahre Klasse (0 oder 1) für das i-te Beispiel
$\hat{y}_i$	Vorhergesagte Wahrscheinlichkeit für Klasse 1

In jeder Epoche des Trainings werden beide Werte berechnet und stellen so die Verbesserung des Modells dar. Der Loss wird dabei minimiert, während die Accuracy maximiert wird. Bereits in der Letzten Studienarbeit wurden die geteste Modelle anhand dieser Metriken evaluiert.

5.1.2. Confusion Matrix und F1 Score

Eine aussagekräftigere Methode zur Evaluierung ist der F1 Score. Um diesen nachzuvollziehen ist zunächst eine aufschlüsselung der Klassifizierten Daten notwendig. Die Confusion Matrix ist eine Tabelle, die die Anzahl der korrekten und falschen Klassifikationen für jede Klasse anzeigt. Die Confusion Matrix hat die folgende Form [17]:

	Vorhergesagt: Positiv	**Vorhergesagt: Negativ**
Tatsächlich Positiv	TP	FN
Tatsächlich Negativ	FP	TN

Hier sind die Werte TP, FP, TN und FN die in Abschnitt 5.1 bereits eingeführt wurden wieder zu finden. Auf der Hauptdiagonale dieser $2x2$ Matrix befinden sich die korrekt klassifizierten Instanzen, während auf der nebendiagonale die falsch klassifizierten Instanzen zu finden sind.

Für jeden Evaluierungsaufruf lässt sich diese Matrix bestimmen. Vorteilhaft an dieser Darstellung ist, dass die Möglichtkeit besteht dieses Modell auf nicht binäre Klassifikation zu erweitern.

Aus dieser Matrix lassen sich nun weitere Metriken ableiten. Eine davon ist der F1 Score. Der F1 Score ist das harmonische Mittel zwischen Precision und Recall. Precision gibt an, wie viele der als positiv klassifizierten Instanzen tatsächlich positiv sind, während Recall angibt, wie viele der tatsächlich positiven Instanzen korrekt klassifiziert wurden. Der F1 Score wird wie folgt berechnet [17]:

$$\text{F1} = \frac{2TP}{2TP + FP + FN} \tag{5.4}$$

Beispielhafte Confusion Matritzen werden in Abschnitt 5.2 vorgestellt.

5.2. Reevaluierung des Modells

In diesem Kapitel soll das Ergebnismodell der letzten Studienarbeit, MobileNet V1 erneut Evaluiert werden. Diesmal mithilfe der neuen Metrinken (Unterabschnitt 5.1.2) und anhand eines neuen Datensatzes von PCBs (siehe Unterabschnitt 4.4.2).

Für diese Reevaluierung wurden mehrere Modelle getestet. In dieser Arbeit werden drei Vertreter dieser Testreihe vorgestellt. Die Ergebnisse der Reevaluierung sind in Abbildung 5.1 dargestellt. Da es sich bei Mobilnet V1 und MobilenetV3 um vortrainierte Modelle von Tensorflow handelt, soll auch ein selbstgeschriebenes Modell evaluiert werden. Dieses Modell "Custom CNN"wurde im Rahmen der Vorlesung Bildverarbeitung entwickelt und fließt mit in diese Tests ein.

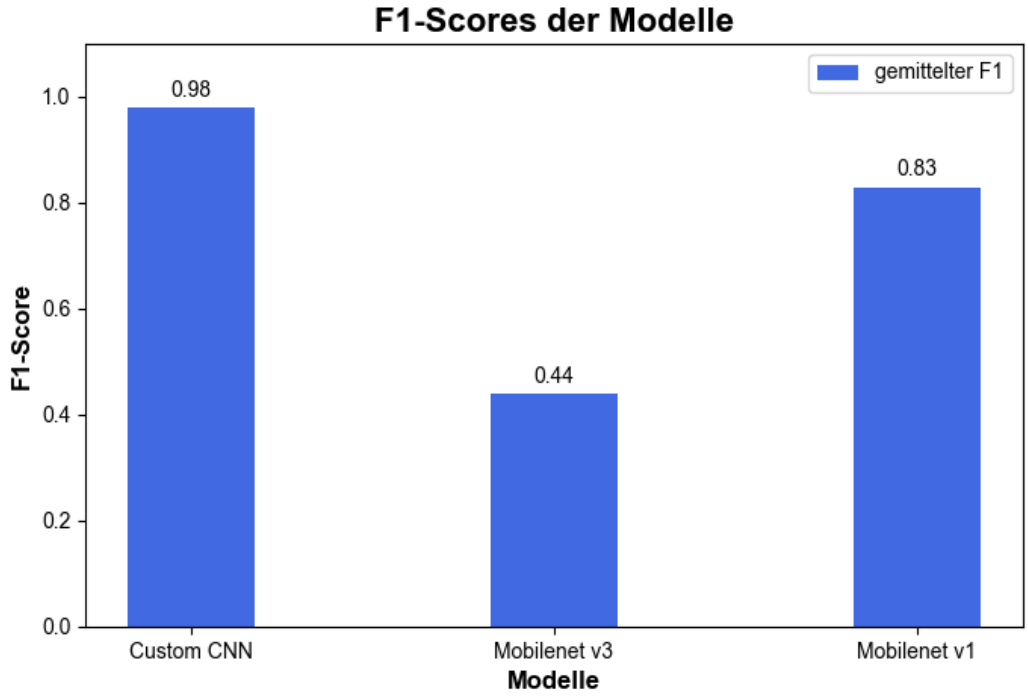


Abbildung 5.1.: Vergleich der Modelle anhand des gemittelten F1 Scores über beide Klassen

Die besten Ergebnisse liefert das Custom CNN, während Mobilenet Version3 auch nach mehreren Optimierungsversuchen schlecht abschneidet. Dies ist Abbildung 5.1 zu entnehmen. Hier sind die gemittelten F1 Scores über beide Klassen für die verschiedenen Modelle dargestellt ist. Berechnungsgrundlage ist die in Unterabschnitt 5.1.2 vorgestellte Confusion Matrix.

Ein detaillierter Vergleich der Modelle zeigt, dass das Custom CNN auch in allen anderen Metriken überlegen ist. Aus Gründen der Übersichtlichkeit enfallen die genauen Tabellen mit allen Daten. Die Tabellen mit allen Werten sind dem Anhang Abschnitt B zu entnehmen. MobileNet V1 liefert dennoch gute Ergebnisse und ist aufgrund der Einfachheit der Integration und der guten Ergebnisse nicht zu vernachlässigen.

Für die folgenden Betrachtungen wurd MobileNet V3 ausgeschlossen. Seine schlechten Ergebnisse lassen auf eine schlechte Anpassung an den Datensatz oder falsche Einstellung schließen. Im Direkten Vergleich haben sowohl das Custom CNN als auch MobileNet V1 tendenziell mehr Probleme mit False Positive Klassifikationen. Dies ist in Abbildung 5.2 zu sehen.

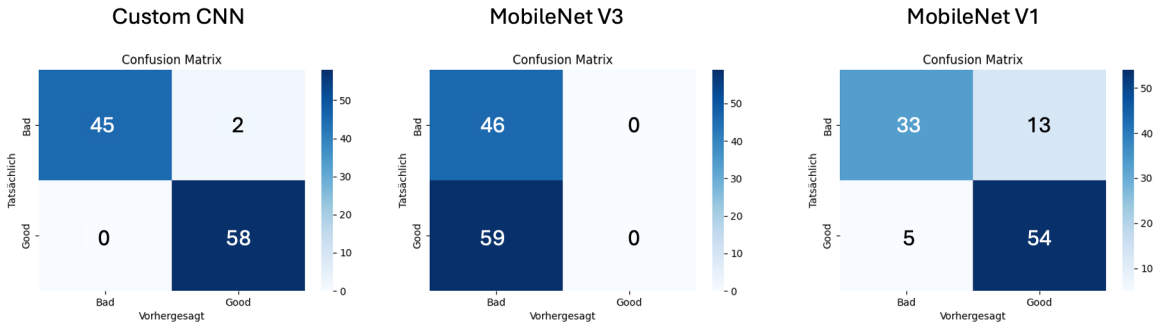


Abbildung 5.2.: Vergleich der Modelle anhand der Confusion Matrizen

Diese gemeinsamen Probleme lassen sich in diesem Fall auf den Datensatz zurückführen. Die falsch Positiv Klassifizierten Bilder sind ebenfalls für den Menschen nur schwer zu klassifizieren, da ein stück Leiterbahn fehlt oder eine schlecht belichtete Komponente fehlt (PCBs siehe Abbildung 4.3 Rechts). In den Praxistests während des Probebetriebs im Labor hat sich gezeigt, dass MobileNet V1 in allen beobachteten Fällen in der Lage ist eine korrekte Klassifikation durchzuführen.

Zusammenfassend lässt sich sagen, dass das Custom CNN in allen getesteten Metriken überlegen ist. Aufgrund der zeitlichen Einschränkungen und der bestehenden Programmpipeline wird jedoch weiterhin MobileNet V1 verwendet. Dieses Modell liefert gute Ergebnisse und ist einfacher zu integrieren.

6. Fazit und Ausblick

In dieser Arbeit wurde ein Ansatz zur Bildklassifikation mittels Convolutional Neural Networks (CNNs) untersucht und implementiert. Die Ergebnisse zeigen, dass CNNs eine hohe Genauigkeit bei der Klassifikation von Bildern erreichen können, insbesondere wenn sie auf spezifische Datensätze und Aufgaben zugeschnitten sind. Die durchgeführten Experimente haben die Bedeutung der Datenvorverarbeitung und der Hyperparameter-Optimierung hervorgehoben, um die Leistung des Modells zu maximieren.

Ein wesentlicher Bestandteil dieser Arbeit war die Entwicklung und Implementierung eines Custom CNN, das speziell auf die Anforderungen des gegebenen Datensatzes abgestimmt wurde. Die erzielten Ergebnisse bestätigen, dass maßgeschneiderte Architekturen, die auf die Besonderheiten der Daten eingehen, zu einer verbesserten Klassifikationsgenauigkeit führen können.

Trotz der erzielten Fortschritte gibt es mehrere Bereiche, die für zukünftige Arbeiten von Interesse sein könnten. Eine mögliche Erweiterung besteht darin, die Integration des Custom CNN in eine umfassendere Programmpipeline zu ermöglichen. Dies würde nicht nur die Leistung der Klassifikation weiter verbessern, sondern auch die Anwendbarkeit des Modells in realen Szenarien erhöhen.

Darüber hinaus könnten zukünftige Arbeiten die Erforschung und Implementierung von Transfer Learning Techniken umfassen. Durch die Nutzung vortrainierter Modelle könnte die Trainingszeit erheblich reduziert und die Genauigkeit weiter gesteigert werden. Auch die Untersuchung verschiedener Regularisierungsmethoden könnte dazu beitragen, Überanpassungen zu vermeiden und die Generalisierungsfähigkeit des Modells zu erhöhen.

Ein weiterer interessanter Ansatz wäre die Implementierung von Ensemble-Methoden, bei denen mehrere Modelle kombiniert werden, um die Klassifikationsergebnisse zu verbessern. Dies könnte insbesondere bei komplexen Datensätzen von Vorteil sein, bei denen einzelne Modelle möglicherweise nicht die bestmögliche Leistung erzielen.

Zusammenfassend lässt sich sagen, dass die in dieser Arbeit entwickelten Methoden und Modelle eine solide Grundlage für die Bildklassifikation bieten. Die vorgeschlagenen Erweiterungen und zukünftigen Forschungsrichtungen haben das Potenzial, die Leistung und Anwendbarkeit der Modelle weiter zu steigern und neue Anwendungsbereiche zu erschließen.

Zukünftige Arbeiten sollten sich darauf konzentrieren, die Integration des Custom CNN in die Programmpipeline zu ermöglichen, um die Leistung der Klassifikation weiter zu verbessern.

Literaturverzeichnis

[1] Finbridge.de. „Computer Vision für Finance [Teil 2]: Convolutional Neural Networks," Finbridge GmbH & Co KG. (8. Sep. 2022), Adresse: https://www.finbridge.de/ml-artikel/2022/09/08/computer-vision-fuer-finance-teil-2 (besucht am 01. 03. 2025).

[2] Intel. „Convolutional Neural Networks (CNN) und Deep Learning," Intel. (), Adresse: https://www.intel.com/content/www/de/de/internet-of-things/computer-vision/convolutional-neural-networks.html (besucht am 01. 03. 2025).

[3] „Machine Learning Regression," Mailchimp. (), Adresse: https://mailchimp.com/de/resources/machine-learning-regression/ (besucht am 10. 11. 2024).

[4] H. Süße und E. Rodner, *Bildverarbeitung und Objekterkennung: Computer Vision in Industrie und Medizin*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014, ISBN: 978-3-8348-2605-3 978-3-8348-2606-0. DOI: 10.1007/978-3-8348-2606-0. Adresse: https://link.springer.com/10.1007/978-3-8348-2606-0 (besucht am 15. 10. 2024).

[5] S. Mittelstand. „Erste Schritte mit Python HTTP-Anfragen für REST-APIs." (), Adresse: https://www.datacamp.com/tutorial/making-http-requests-in-python (besucht am 01. 03. 2025).

[6] K. Pykes. „Programmieren mit Python APIs: Ihr ultimativer Guide in einfachen Schritten." (), Adresse: https://www.software-mittelstand.info/apis-mit-python-programmieren-eine-schritt-fuer-schritt-anleitung/ (besucht am 01. 03. 2025).

[7] D. O. LLC. „Erste schritte mit der requests-bibliothek in python | DigitalOcean." (), Adresse: https://www.digitalocean.com/community/tutorials/how-to-get-started-with-the-requests-library-in-python-de (besucht am 01. 03. 2025).

[8] C. Rodríguez, M. Baez, F. Daniel u. a., „REST APIs: A large-scale analysis of compliance with principles and best practices," in *Web Engineering*, A. Bozzon, P. Cudre-Maroux und C. Pautasso, Hrsg., Cham: Springer International Publishing, 2016, S. 21–39, ISBN: 978-3-319-38791-8. DOI: 10.1007/978-3-319-38791-8_2.

[9] Flask. „Welcome to Flask — Flask Documentation (3.1.x).“ (), Adresse: https://flask.palletsprojects.com/en/stable/ (besucht am 01. 03. 2025).

[10] *Diskussion über die Projektstrukturen von Python ML Projekten*, unter Mitarb. von M. B. Gür, 8. Nov. 2024.

[11] P. Oliveira. „How to write a python configuration file,“ LambdaTest. Section: Selenium Python. (29. Sep. 2023), Adresse: https://www.lambdatest.com/blog/python-configuration-file/ (besucht am 02. 03. 2025).

[12] *Model View Controller*, in *Wikipedia*, Page Version ID: 248816402, 22. Sep. 2024. Adresse: https://de.wikipedia.org/w/index.php?title=Model_View_Controller&oldid=248816402 (besucht am 02. 03. 2025).

[13] „Threading — thread-based parallelism,“ Python documentation. (), Adresse: https://docs.python.org/3/library/threading.html (besucht am 11. 03. 2025).

[14] P. S. Foundation. „Venv — creation of virtual environments,“ Python documentation. (), Adresse: https://docs.python.org/3/library/venv.html (besucht am 12. 03. 2025).

[15] S. I. GmbH, „VISOR Benutzerhandbuch,“ Jan. 2019.

[16] A. Wiki. „Accuracy and loss | AI wiki.“ (17. Dez. 2019), Adresse: https://machine-learning.paperspace.com/wiki/accuracy-and-loss (besucht am 02. 03. 2025).

[17] Z. C. Lipton, C. Elkan und B. Narayanaswamy, *Thresholding Classifiers to Maximize F1 Score*, 14. Mai 2014. DOI: 10.48550/arXiv.1402.1892. arXiv: 1402.1892[stat]. Adresse: http://arxiv.org/abs/1402.1892 (besucht am 02. 03. 2025).

A. Anhang

A. Anleitung zur Verwendung

Nach der korrekten Installation des Programms, welches in Kapitel 4.4 beschrieben ist, kann das Programm wie folgt verwendet werden: Die Installation ist zum Zeitpunkt der Übergabe dieser Arbeit bereits auf dem FESTO-PC erfolgt.

Der Quellcode ist jederzeit unter folgender URL erreichbar: https://github.com/andreasbraig/DHBWMA_HHAT_Image-Classification

Im Branch main befindet sich die Version zum Zeitpunkt der Abgabe.

Zunächst sollte auf folgende Punkte geachtet werden:

- Die Installation ist auf dem FESTO-PC erfolgt. Das Passwort ist bei Frau Prof. Dr.-Ing. Bozena Lamek-Creutz zu erfragen.
- Die verwendete Python Version ist 3.12.9.
- Alle verwendeten Bibliotheken sind in der requirements.txt Datei aufgelistet.
- Das Programmverzeichnis mit allen Daten liegt unter:

  `D:/Repo/DHBWMA_HHAT_Image-Classification`

Um das Programm auszuführen, bitte der folgenden Anleitung folgen:

1. Das PowerShell Skript auf dem Desktop mit dem Namen `Image_classification.ps1` ausführen.
2. Sobald in PowerShell die URL `http://127.0.0.1:5000` angezeigt wird, kann die Webseite im Browser geöffnet werden.

3. Wenn mit der VISOR Kamera der FESTO CP-Lab Kamera Fotos archiviert werden, sollten diese auf der Website mit Klassifizierungsergebnis dargestellt werden.

Im Folgenden wird in zwei Methoden erklärt, wie Fotos mit der VISOR Kamera archiviert werden können:

A.1. Manuelle Archivierung

1. Die FESTO CP-Lab Anlage muss mit Strom und Druckluft versorgt sein.

2. Auf dem FESTO-PC die Visor Software starten.

3. Zunächst Config öffnen, das Passwort ist `Admin` für den Benutzer `admin`.

4. Dieses Fenster kann direkt wieder geschlossen werden. (die Einstellungen sind bereits gespeichert, also `Yes` drücken)

5. Im Visor Fenster auf `View` klicken und unten links `Archiving` durch Klicken aktivieren.

6. Nun sollte die Kamera Fotos aufnehmen und diese in folgenden Ordner ablegen:

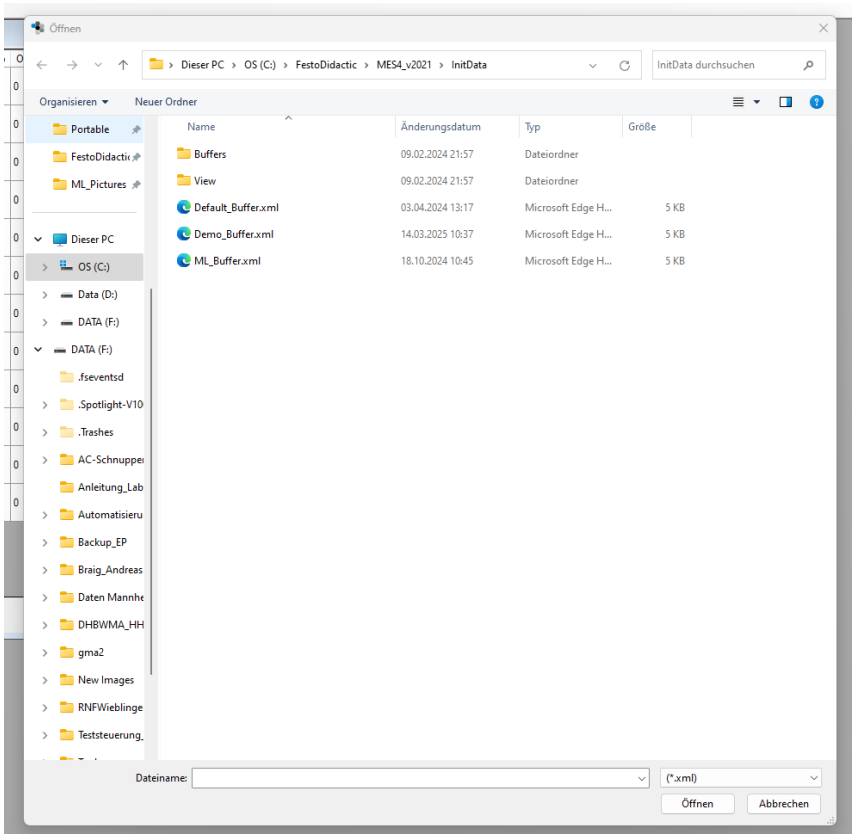
   `D:/Repo/DHBWMA_HHAT_Image-Classification/Bilder/new`

A.2. Automatische Archivierung

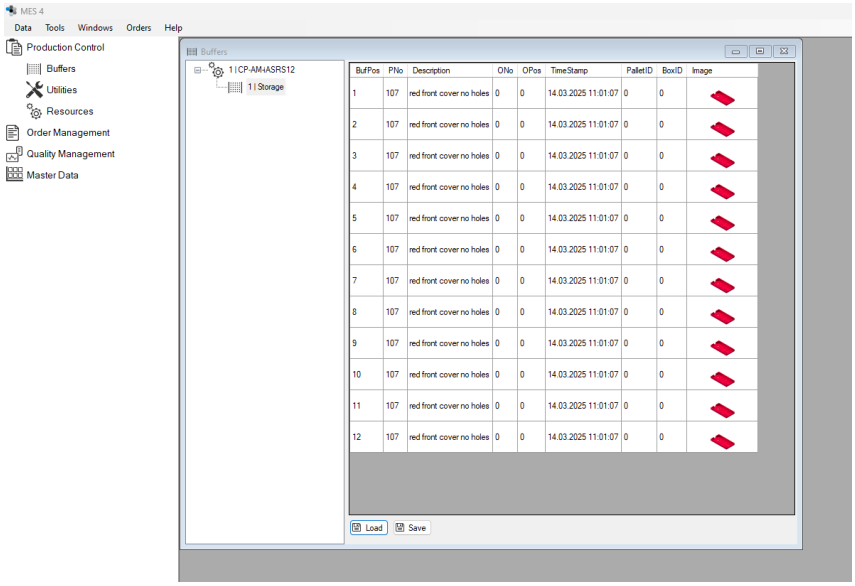
Für diese Methode wurde in der `MES 4` Software der FESTO CP-Lab Anlage eine neue Order mit einem neuen Buffer erstellt, welcher den Job für die Kamera enthält.

1. Die FESTO CP-Lab Anlage muss mit Strom und Druckluft versorgt sein.

2. Die Software MES 4 auf dem FESTO-PC starten.

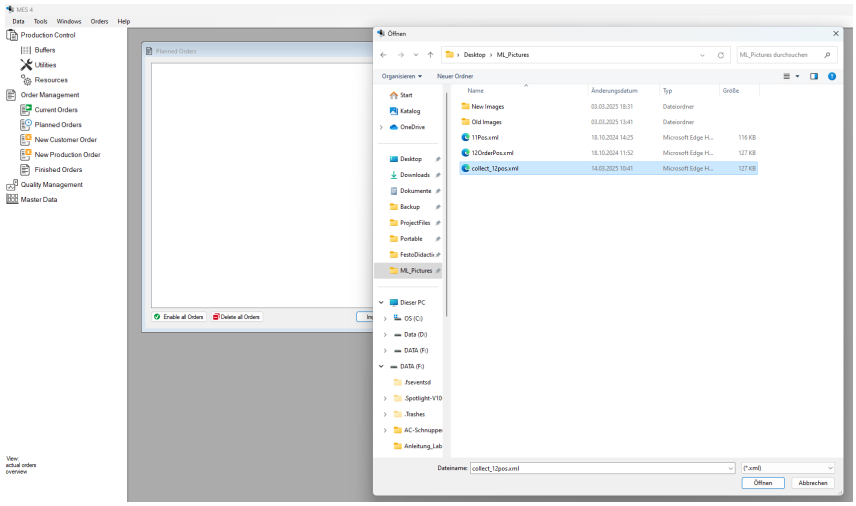


3. In der MES 4 Software einen neuen Buffer laden, dieser ist mit dem Namen `Demo_Buffer` versehen.



4. Der Buffer sollte jetzt mit pinken Boxen gefüllt sein.

5. Über das `Planned Orders` Menü eine neue Order importieren. Diese liegt unter `C:/Users/Festo/Desktop/ML Pictures` ab.



6. Die Order sollte jetzt in der MES 4 Software sichtbar sein und muss gestartet werden.

7. Nun sollte die Kamera Fotos aufnehmen und diese in folgenden Ordner ablegen:

   `D:/Repo/DHBWMA_HHAT_Image-Classification/Bilder/new`

8. Zur Kontrolle kann auf dem HMI der Kamerastation der aktuell ausgewählte Job betrachtet werden. Dieser sollte 7 sein.

B. Performance der Modelle

B.1. Pytorch Custom CNN

Label	F1-Score			Support
	Precision	Recall	F1-Score	
Bad	1.00	0.96	0.98	47
Good	0.97	1.00	0.98	58
Accuracy			0.98	105
Macro avg	0.98	0.98	0.98	105
Weighted avg	0.98	0.98	0.98	105

Tabelle A.1.: Performance der Custom CNN (PyTorch) Modell

B.2. MobileNet V3 (TensorFlow)

Label	F1-Score			Support
	Precision	Recall	F1-Score	
Bad	0.44	1.00	0.61	46
Good	0.00	0.00	0.00	59
Accuracy			0.44	105
Macro avg	0.22	0.50	0.30	105
Weighted avg	0.19	0.44	0.27	105

Tabelle A.2.: Performance der MobileNet V3 (TensorFlow) Modell

B.3. MobileNet V1 (TensorFlow)

Label	F1-Score			Support
	Precision	Recall	F1-Score	
Bad	0.87	0.72	0.79	46
Good	0.81	0.92	0.86	59
Accuracy			0.83	105
Macro avg	0.84	0.82	0.82	105
Weighted avg	0.83	0.83	0.83	105

Tabelle A.3.: Performance der MobileNet V1 (TensorFlow) Modell