Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

LEAN PRODUCTION MONITORING

Masterarbeit

Zürcher Hochschule für Angewandte Wissenschaften

eingereicht bei:
Prof. Dr. Thilo Stadelmann

Vorgelegt von: David Frey

Matrikelnummer: 13566872

Studiengang: Master of Advanced Studies (MAS) in Data Science

Ort, Datum: Galgenen, 28.06.2023

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

Erklärung betreffend das selbständige Verfassen einer Masterarbeit

Mit der Abgabe dieser Masterarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. Der/die unterzeichnende Studierende erklärt, dass alle verwendeten Quellen (auch Internetseiten) im Text oder Anhang korrekt ausgewiesen sind, d.h. dass die Masterarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind. Bei Verfehlungen aller Art treten Paragraph 39 und Paragraph 40 der Rahmenprüfungsordnung für die Bachelor- und Masterstudiengänge an der Zürcher Hochschule für Angewandte Wissenschaften vom 29. Januar 2008 sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Zusatz über die Nutzung generativer KI-Systeme

Die für diese Arbeit erstellten Python Skripte wurden teilweise mit der KI bekannt als Chat GPT überprüft, ergänzt und verbessert. Mit dieser Erklärung bezieht sich der Verfasser dieser Arbeit auf die E-Mail des Leiters für Weiterbildung an der ZHAW, Herrn Markus Marti, verfasst am 05.04.2023 um 11:13 Uhr. Diese besagt, dass die Verwendung generativer KI-Systeme erlaubt ist, aber vom Verfasser/-in der Arbeit deklariert werden muss.

Ort, Datum: Unterschrift:

Galgenen, 28.06.2023 ..... D. Frey .....

Abstract/Zusammenfassung

In Anbetracht der wachsenden Bedeutung von Echtzeitdaten in der Fertigungsindustrie und der zunehmenden Digitalisierung von Produktionsstätten hat die Sensirion AG ein internes Projekt namens 'Lean Production Monitoring' initiiert, mit dem Ziel, die Verfügbarkeit von Echtzeitdaten in ausgewählten Produktionslinien zu verbessern und diese Daten für die Optimierung von Prozessen nutzbar zu machen. Trotz des grossen Potentials des maschinellen Lernens in diesem Kontext wurden die Möglichkeiten bei der Sensirion AG bisher nicht vollständig ausgeschöpft.

Diese Masterarbeit adressiert diese Lücke und zielt darauf ab, ein nicht-invasives System zur Erfassung von Produktionsdaten in Echtzeit zu schaffen, das durch maschinelles Lernen gestützt wird. Die vorliegende Masterarbeit beschreibt den Aufbau dieses Systems detailliert Schritt für Schritt. Sämtliche beschriebenen Schritte wurden im Praxisunternehmen experimentell getestet und angewendet. Als Teil des Systems wird ein Raspberry Pi in Kombination mit einer Kamera zur Bilderfassung verwendet. Darüber hinaus wurde eine Datenpipeline vom Raspberry Pi zu Databricks eingerichtet und ein Labeling Prozess für die erfassten Bilder entwickelt. Zudem wurden zwei konvolutionelle neuronale Netzwerke, eines mit der Rolle als Klassifikator und eines zur Objekterkennung, in diesem System als schnell einsetzbare und erweiterbare Modelle implementiert. Das ganze System zielt darauf ab, mit möglichst wenig Aufwand in Bezug auf Kamerasetup, Zeit zum Labeln der Daten und Modelltraining zurechtzukommen. Zuletzt wurde ein Prozess zur Qualitätsüberwachung etabliert, der in regelmässigen Abständen zufällige Testbilder extrahiert und einem manuellen Überprüfungsprozess zuweist. Dieser Prozess garantiert die Gewährleistung der Modellgüte.

Die Ergebnisse aus den Experimenten zeigen, dass das System als Ganzes effizient funktioniert und erfolgreich zur Erfassung von Produktionsdaten eingesetzt werden kann. Zwei Testläufe, einer mit dem CNN-Klassifikator und einer mit dem Modell zur Objekterkennung wurden in unterschiedlichen Produktionslinien eingesetzt. Die Auswertung zeigt, dass beide Modelle mit einem Labeling Aufwand von etwa einer Arbeitsstunde für die jeweiligen Daten bereits gute Leistungen erbringen, insbesondere der Klassifikator. Die Modelleistung wird durch das Nutzen von Hyperopt, einem Paket zur Optimierung von Hyperparametern, maximiert. Das implementierte Qualitätsmonitoring ermöglicht es, eine langfristige Übersicht über die Modellperformance zu behalten und gleichzeitig weitere Trainingsdaten für künftige Trainingsläufe zu generieren, womit das Fazit gezogen werden kann, dass das Gesamtsystem erfolgreich implementiert wurde. Es wird der Produktionsleitung von Sensirion die Empfehlung abgegeben, das implementierte System beizubehalten und weitere Ressourcen für den Unterhalt und mögliche Weiterentwicklungen bereitzustellen.

Inhaltsverzeichnis

Vorwort **2**
1 Einleitung **3**
1.1 Anforderungen 5
1.2 Zielsetzung und inhaltliche Abgrenzung dieser Masterarbeit 7
1.3 Struktur dieser Masterarbeit 8
2 Grundlagen **9**
2.1 Modelle für Bildklassifikation 12
2.2 Modelle für Objekterkennung 13
3 Methoden und Implementierung **15**
3.1 Kamerasystem Anbindung mit Raspberry Pi und Balena OS inklusive Datenpipeline 15
3.2 Aufbau Databricks Infrastruktur 18
3.3 Labeling der Bilder 20
3.4 CNN-Modell Klassifikator für Ampelsignale 21
3.5 Ausweitung des Systems auf weitere Aufgaben: CNN-Modell Objekterkennung 24
3.6 Qualitätsmonitoring 27
3.7 Zusammenfassung und Übersicht 33
4 Resultate **35**
4.1 Testlauf VGG16 Modell zur Klassifikation von Ampelsignalen 35
4.1.1 Training und Resultate 37
4.1.2 Hyperparameter Tuning 44
4.1.3 Qualitätsmonitoring 46
4.1.4 Dashboard Darstellung 48
4.2 Testlauf Retinanet Modell zur Objekterkennung 50
4.2.1 Training und Resultate 51
4.2.2 Hyperparameter Tuning 58
4.2.3 Qualitätsmonitoring und Dashboard Darstellung 60
5 Fazit **61**
6 Quellenverzeichnis **66**
7 Abbildungsverzeichnis **68**
8 Tabellenverzeichnis **69**
9 Anhang **70**

Vorwort

Die vorliegende Masterarbeit entstand im Rahmen meiner Weiterbildung, dem MAS Data Science an der Zürcher Hochschule für Angewandte Wissenschaften. Die Idee für diese Arbeit ergab sich aus meinem persönlichen Interesse für maschinelle Bilderkennung und dem Wunsch, mein gelerntes Wissen in der Praxis anwenden zu können.

Ein besonderer Dank geht an meinen Betreuer, Prof. Dr. Thilo Stadelmann für seine Unterstützung und seine hilfreichen Tipps für die Vervollständigung meiner Arbeit. Ein weiterer Dank geht an meinen innerbetrieblichen Betreuer, Salomon Diether, welcher mir dieses Thema vermittelt hat und während der ganzen Dauer dieser Arbeit tatkräftig und mit viel Motivation zur Seite stand. Des Weiteren bedanke ich mich bei meinen Arbeitskollegen Jürg Eugster, Philipp Kübler und meinem Chef, Lukas Steinmann, welche mich in dieser Zeit unterstützt haben. Ebenfalls möchte ich mich bei meinem Arbeitgeber, der Sensirion AG bedanken, welcher mir die nötige Infrastruktur, Material und Tools zur Verfügung gestellt hat.

Mein letzter Dank geht an meine Familie und besonders an meine liebevolle Partnerin, die mich in dieser intensiven Phase stets unterstützt und mir viel Kraft gegeben haben.

1 Einleitung

Hintergrund

Die Sensirion AG ist ein weltweit führender Entwickler und Hersteller von Sensoren und Sensormodulen. Im Laufe der letzten 15 Jahre hat sich ein beachtlicher Maschinenpark in den Produktionsgebäuden angesammelt, der jedes Jahr Millionen von Produkten ausstossen. Aufgrund des steigenden Bedarfs an Produktionskapazitäten hat die Produktionsleitung diverse Massnahmen zur Optimierung von Produktionsprozessen und Kapazitätssteigerungen ergriffen. Ein Produktionsprozess besteht gewöhnlich aus mehreren Arbeitsschritten. Einige davon sind manuell, andere Arbeiten werden vollautomatisiert von Maschinen übernommen. Das Ziel einer jeden Produktionsabteilung ist es, sämtliche Arbeitsschritte so aufeinander abzustimmen, dass so wenig Leerlaufzeiten wie möglich und ein Maximum an Produktionsleistung erreicht werden kann. Um eine vollständige Übersicht über einen laufenden Produktionsprozess zu erhalten, braucht es Echtzeitdaten von allen Prozessschritten, welche Probleme rasch anzeigen und Engpässe (Engl. «Bottlenecks») identifizierbar machen. Ein Sammelbegriff aus der Literatur für das Optimieren von Produktionsprozessen, sowie von Materialflüssen ist 'Lean Production', oder auf Deutsch übersetzt 'schlanke Produktion' (IPH Hannover, 2023).

Modernste Produktionsmaschinen generieren bereits grosse Mengen an Daten, allerdings verfügt die Sensirion AG teilweise über ältere Maschinen, die entweder keine oder nur wenige Daten in Echtzeit zur Verfügung stellen können. Zudem fehlen bei vielen manuellen Arbeitsschritten, zum Beispiel dem Bestücken von Leiterplatinen, Lötprozessen, manuellem Nachführen von Material und dem händischen Zusammenbau von Sensormodulen wichtige Informationen bezüglich Materialverfügbarkeit und Prozesszeiten.

In Zusammenarbeit mit der Produktionsplanung wurde firmenintern ein Projekt namens 'Lean Production Monitoring' gestartet. Das Hauptziel dieses Projektes ist es, die Verfügbarkeit von Echtzeitdaten an möglichst vielen Prozessschritten von ausgewählten Produktionslinien in einem temporären Zeitraum zu verbessern und die aggregierten Daten in Form eines Dashboards einfach zugänglich zu machen. Das neue Lagebild dient dazu, die Schwachstellen in einer Prozesskette zu detektieren und geeignete Massnahmen zur Verbesserung des Prozesses zu ergreifen. Das System dafür muss schnell einsetzbar und modular auf unterschiedliche Anwendungen erweiterbar sein. Anhand dieser Daten kann dann die Produktionsplanung geeignete Massnahmen zur Optimierung von Prozessschritten ergreifen, was eine Steigerung der Produktionskapazitäten bewirken soll.

Motivation

Angesichts der dargelegten Herausforderungen besteht ein offensichtlicher Bedarf an praktischen Tools zur Verbesserung der Verfügbarkeit von Echtzeitdaten im Produktionsprozess. Hier kommt die Rolle des maschinellen Lernens in Spiel.

Die Bedeutung von maschinellem Lernen in Produktionsstätten kann in unserer heutigen Zeit nicht hoch genug eingeschätzt werden. Verschiedenste Methoden von maschinellem Lernen tragen entscheidend zur digitalen Transformation in der Fertigungsindustrie bei. Dies wird hauptsächlich durch die Optimierung von Fertigungsabläufen, Material- und Mitarbeiterplanung erreicht. Um langfristig erfolgreich zu bleiben, müssen Produktionsfirmen ihr Produktionspotential mit sogenannten 'Smart Manufacturing Systems' (SMS) vollumfänglich ausschöpfen. Das beinhaltet unter anderem das Digitalisieren sämtlicher Glieder einer Prozesskette, sowie der nötigen Infrastruktur, um die gewonnenen Daten aufzubereiten und auszuwerten (Lu et al., 2016).

Zahlreiche Studien wurden in den letzten Jahren durchgeführt, deren Fokus auf dem Einfluss von maschinellem Lernen in verschiedenen Industriesektoren lag. Eine Studie untersuchte den Einfluss von maschinellem Lernen zur Erkennung von Anomalien und Signalverarbeitung in Echtzeit, um den Defekt einer Produktionsmaschine frühzeitig vorherzusagen. Diese Studie zeigt, dass der Einsatz von maschinellem Lernen zur Reduktion von Wartungskosten und Effizienzsteigerung von Produktionsprozessen führen kann (Zhao et al., 2019).

Trotz des enormen Potenzials, das maschinelles Lernen für die Produktionsoptimierung hat, wurden seine Anwendungsmöglichkeiten bei der Sensirion AG noch nicht vollständig ausgeschöpft, insbesondere aufgrund der Komplexität im Bereich des Modellaufbaus, Trainings und Tunings. Diese Arbeit ist daher von entscheidender Bedeutung, da sie eine Plattform für diverse CNN-Modelle schafft, die individuell auf unterschiedliche Produktionsprozesse zugeschnitten werden können. Ziel dieser Arbeit ist daher, unter Berücksichtigung der gegebenen Voraussetzungen eine universell einsetzbare Plattform zu erstellen. Diese Plattform stellt der Produktionsleitung die nötigen Instrumente zur Optimierung von Material- und Personalmanagement zur Verfügung und trägt somit zur Steigerung der Effizienz und Produktivität bei, damit die Sensirion AG nachhaltig wettbewerbsfähig bleibt.

1.1 Anforderungen

Dieses Unterkapitel behandelt die Anforderungen von Seiten des Projektteams sowie die inhaltliche Abgrenzung dieser Masterarbeit.

Geleistete Vorarbeit

Zum Zeitpunkt des Starts dieser Masterarbeit war eine Komponente des 'Lean Production Monitoring' Projekts bereits implementiert. Der Ansatz besteht darin, mit einem Beschleunigungsmesser die Durchsatzzahl an einem Fliessband oder einer Produktionsmaschine mit Hilfe von Mustererkennung zu berechnen. Zum Beispiel kann eine Maschine, die fertige Sensormodule verschweisst, mit einem solchen Beschleunigungsmesser ausgerüstet werden. Das jeweilige Pulsmuster, das beim Verschweissungsvorgang vom Sensor erfasst wird, kann mit maschinellem Lernen detektiert und die Anzahl Pulse aufaddiert werden. Für die Datenübertragung der Rohsignale wird ein Raspberry Pi mit einem WLAN-Netz verbunden und die Daten via Confluent Kafka auf eine Databricks Cloud geladen. Kafka ist ein Streaming-System, das von Balena unterstützt wird und bietet die Möglichkeit, grosse Datenmengen in Echtzeit zu verarbeiten und zwischen verschiedenen Systemen zu streamen. Auf Databricks findet dann die Mustererkennung und weitere Datenverarbeitung in Form von regelmässigen Jobs statt. Aufgrund der positiven Rückmeldungen von Seiten Produktionsplanung wurde eine Erweiterung dieses Projekts beschlossen. Dieses Unterkapitel dient zur Übersicht über die bereits geleistete Vorarbeit.

Abbildung 1: Prozessschema für initiales Projekt.

Anforderungen an das Projekt

Fast jede Produktionsmaschine bei der Sensirion AG verfügt über ein Ampelsystem, welches den Produktionsmitarbeitern optische Signale zum aktuellen Status der Maschine vermittelt. Wie bereits erwähnt, werden diese Ampelsignale oder Statusmeldungen teilweise (insbesondere bei älteren Maschinen) nicht systematisch als Daten erfasst und hochgeladen. Andere Maschinen haben aufgrund unterschiedlicher Softwares keine einheitliche Integration in das Datenmanagement System des Unternehmens.

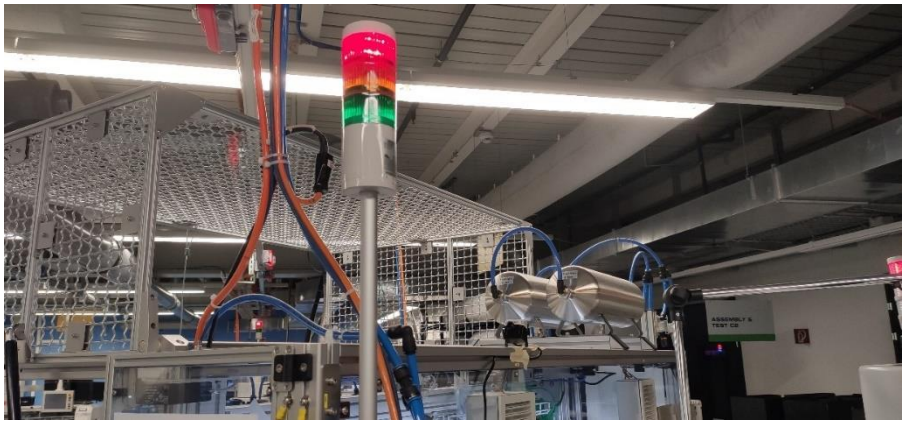

Abbildung 2: Ampelsystem an einer Produktionsmaschine der Sensirion AG.

Der erste Auftrag der Produktionsleitung ist, die optischen Signale ausgewählter Produktionsmaschinen zu digitalisieren und den aktuellen Status sämtlicher Maschinen auf einem Dashboard zu visualisieren. Eine Maschine, die nichts produziert kann entweder ausgeschaltet sein, einen Fehler oder Defekt haben oder einfach auf den manuellen Input eines Produktionsmitarbeiters warten. Da keine einheitlichen Ampelsysteme verwendet werden, muss die Lösung mit unterschiedlichen Ampeltypen (Grösse, Farben) zurechtkommen. Zudem dürfen keine invasiven Eingriffe an den Produktionsmaschinen, wie beispielweise das Anzapfen der Kabel für die Ampeln, vorgenommen werden. Das System muss trotz unterschiedlicher Ampeltypen in maximal 5 Minuten an oder in der Nähe des Ampelsystems eingerichtet sein.

Eine weitere Anforderung von der Leitung des ‘Lean Production Monitoring’ Projekts sind unter anderem die Nutzung eines Raspberry Pi 3 oder 4 als Plattform für die Datenerfassung und Übermittlung. Aufgrund deren Verwendung und Bewährung in früheren Projekten verfügt die Sensirion AG über genügend Stückzahlen von diesen Einplatinencomputern. Zuletzt muss das System zur Klassifikation von Ampelsignalen erweiterbar auf andere Aufgaben sein, beispielweise dem Erkennen und Zählen von Objekten oder dem Klassifizieren von weiteren Objekten und Zuständen.

Die Anforderungen für dieses System wurden vom Leiter des 'Lean Production Monitoring' Projektes in der folgenden Tabelle 1 festgehalten:

Tabelle 1: Auftrag und Anforderungen an das Projekt.

System	Anforderung Sensirion
Hardware: System zur Erkennung von Ampelsignalen/Produktions-material	➢ Nicht invasiv (es dürfen keine Kabel oder Datenströme angezapft werden) ➢ Das System muss mit diversen Typen von Ampeln funktionieren (Grösse, Anzahl Farben.) ➢ Als Plattform für die Datenübermittlung wird ein Raspberry Pi 3 oder 4 mit einer Balena Host OS benutzt. ➢ Einfach einzurichten (keine Spezialgeräte oder Werkzeuge) ➢ Maximaler Aufwand für das Einrichten pro Ampel: 5 Minuten (Anbringen der Hardware, aufsetzen Datenpipeline) ➢ Die gleiche Plattform ist auf weitere Anwendungen, z.B. dem Detektieren von Produktionsmaterial in einer Prozesslinie erweiterbar.
Datenpipeline/Software	➢ Die Daten müssen auf einer Datenbank oder Cloud redundant gespeichert werden. ➢ Die Daten dürfen keine sensitiven oder vertraulichen Inhalte haben. ➢ Kontinuierliches Qualitätsmonitoring. Die Genauigkeit des Systems muss in regelmässigen Abständen überprüft und Genauigkeitsverluste erkannt werden

1.2 Zielsetzung und inhaltliche Abgrenzung dieser Masterarbeit

Basierend auf den Anforderungen wurde die Entscheidung gefällt, eine Machine Learning (Abk. ML) Lösung mit Kameras zu implementieren. Die Bilder von den Ampelsystemen werden mit Hilfe eines ML-Modells in Statusmeldungen übersetzt (z.B. «Wartet auf Input», oder «Maschine ausgeschaltet»), welche dann ein Echtzeitbild des Maschinenstatus darstellen. Dieser Ansatz ermöglicht das Verwenden der gleichen Hardware (Kamera, Raspberry Pi) für verschiedene Ampelmodelltypen. Ein weiterer grosser Vorteil ist, dass der Aufgabenbereich mit zusätzlichen ML-Modellen erweitert werden kann. Die Kameras werden nach dem Einrichten mit einem Raspberry Pi verbunden, welches sich aufgrund seines Balena Host OS automatisch mit dem firmeninternen WLAN verbindet und die Bilder via Python Skript auf die Databricks Cloud lädt.

Die Eigenleistung für diese Masterarbeit beinhaltet folgende Punkte:

I. Aufbau einer Datenpipeline vom Raspberry Pi nach Databricks und Einbettung der Kamera auf dem Raspberry Pi mit Hilfe eines Python Skriptes.

II. Aufbau einer Plattform auf Databricks, welches Modellverwaltung, Datenverwaltung und Labeling sowie Qualitätsmonitoring integriert.

III. Aufbau eines Labeling Prozesses für die erfassten Bilder.

IV. Erstellen und Tuning eines CNN-Klassifikators. Dieses Modell wird zur Klassifikation von Ampelsignalen eingesetzt, was die Kernanforderung dieses Projektes ist. Zudem muss es die Anforderungen der einfachen Erweiterbarkeit auf neue Angaben, kurze Zeit zum Einrichten und die Skalierbarkeit erfüllen.

V. Erstellen und Tuning eines CNN-Modells zur Objekterkennung. Dieses Modell dient als Erweiterung und kann Aufgaben übernehmen, für die ein Klassifikator nicht geeignet ist. Wie bereits in Punkt IV erwähnt, muss auch dieses Modell die Anforderungen der Erweiterbarkeit, Zeit zum Einrichten und Skalierbarkeit erfüllen.

1.3 Struktur dieser Masterarbeit

Diese Arbeit gliedert sich in fünf Hauptkapitel. Nach der Einleitung, welche einen Überblick über das Thema, Motivation, Zielsetzung und inhaltliche Abgrenzung dieser Arbeit verschafft, folgt der Grundlagen Teil in Kapitel 2. Dieser Teil enthält die, für dieses Thema relevante Literatur und das Basiswissen über Elemente eines konvolutionellen neuronalen Netzwerks. Im dritten Teil wird die Implementation des Gesamtsystems und deren Subkomponenten beschrieben. Darunter sind Themen wie der Aufbau der Datenpipeline, Datenablage, Modellwahl und Qualitätsmonitoring. Kapitel 4 beschreibt die durchgeführten Experimente und präsentiert die Ergebnisse. Eine Diskussion über die Resultate und deren Interpretation folgt am Ende des Kapitels. Das fünfte Kapitel beinhaltet das Fazit, einen Rückblick über das Erreichte und die Auseinandersetzung mit kritischen Aspekten dieser Arbeit.

2 Grundlagen

Die Zielsetzung dieser Arbeit, beschrieben in Kapitel 1.2, beinhalten mehrere Themenbereiche. Punkte I, II und III beinhalten die Datenpipeline, den Aufbau der Databricks Infrastruktur und die Implementierung eines Labeling Prozesses. Diese Punkte liegen nicht im Hauptfokus dieser Arbeit und werden in diesem Grundlagenteil nicht weiter behandelt. Eine Beschreibung dieser drei Punkte erfolgt in den Kapiteln 3.1, 3.2 und 3.3.

Aufgrund der Anforderungen an das firmeninterne Projekt, beschrieben in Kapitel 1.1, wird eine Lösung mit einem konvolutionellen neuronalen Netzwerk in Kombination mit einem Kamerasystem zur Klassifikation der Ampelzustände angestrebt. Dieses Thema betrifft die Punkte IV und V der Zielsetzung. Die theoretischen Grundlagen hierfür werden nachfolgend beschrieben. Zuerst werden die wichtigsten Komponenten von konvolutionellen neuronalen Netzwerken erklärt. Anschliessend werden zwei Modelle aus der Literatur vorgestellt, die sich zur Klassifikation eignen (Punkt IV). Diese beiden Modelle zeichnen sich durch ihre in der Fachwelt einschlägige Reputation aus, weshalb sie im Unterkapitel 2.1 erläutert werden. Punkt V wird in Unterkapitel 2.2 thematisiert und beschreibt mehrere CNN-Modelle aus der Literatur, die sich für Objekterkennung eignen.

Konvolutionelle Neuronale Netzwerke

Konvolutionelle neuronale Netzwerke, abgekürzt als CNN (Convolutional Neural Networks), sind eine besondere Form eines künstlichen neuronalen Netzwerks. Sie eignen sich aufgrund ihrer Architektur besonders für Bild- und Spracherkennung. Herkömmliche CNNs bestehen aus unterschiedlichen Schichten (Engl. Layer), den sogenannten Faltungs- oder Convolutional Layern, den Pooling Layern und den Fully Connected Layern (Litzel & Luber, 2019).

Die Aufgaben der einzelnen Layer sind wie folgt:

Convolutional Layer

Dieser Layer bildet die Faltungsebene. Er wird dazu genutzt, um eine oder mehrere Feature Maps aus einem Tensor zu extrahieren. Ein Tensor ist ein mathematisches Objekt, das als eine Verallgemeinerung von Skalaren, Vektoren und Matrizen angesehen werden kann. Ein Farbbild kann als Tensor der Ordnung 3 angesehen werden mit einer Dimension für die Höhe, Breite und für die Farbkanäle (z.B. RGB). Ein Filter oder sogenannter Kernel ist eine Matrix, die zur Generierung von Feature Maps verwendet wird (Litzel & Luber, 2019). Die folgende Abbildung beschreibt, wie ein 3x3 Kernel eine multidimensionale Feature Map aus einem 6x6x3 Tensor extrahiert. Der Stride bestimmt die Schrittgrösse des Kernels bzw. um wie viele Reihen und

Spalten dieser sich verschiebt. Ein Stride von 1 bedeutet, dass der Kernel sich jeweils um eine Reihe oder Spalte verschiebt (Prijono, 2018).

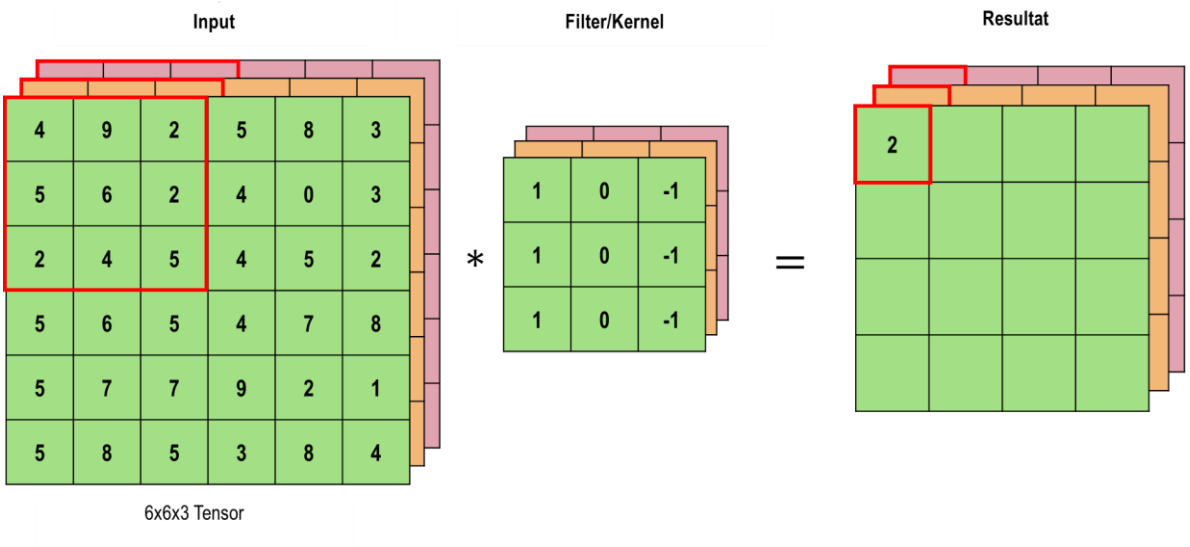

Abbildung 3: Funktionsweise eines Convolutional Filters/Kernels mit der Grösse 3x3 und Stride 1. Eigene Darstellung, Inhalt übernommen von Prijono (2018).

Das Ändern der Zahlen im Kernel ändert dessen Filterverhalten und ergibt als Resultat eine andere Feature Map. Daher ist es für CNN üblich, mit mehreren unterschiedlichen Kernels und Feature Maps zu arbeiten (Prijono, 2018).

Pooling Layer

Der Pooling Layer wird normalerweise nach einem Convolutional Layer angewandt. Er dient mehreren Zwecken. Zum einen wird er zur Dimensionsreduzierung und somit zur Verkleinerung der Gesamtzahl der Parameter und Berechnungen im Netzwerk verwendet. Ein weiterer Vorteil ist, dass durch die Reduzierung der Dimension sich das Modell mehr auf die hochrangigen Merkmale konzentriert, da diese nach mehreren Pooling Layers noch vorhanden sind (Litzel & Luber, 2019). Wie bereits beim Kernel sind die Filtergrösse sowie der Stride variabel. Die folgende Abbildung beinhaltet eine Filtergrösse von 2x2 und einem Stride von 2.

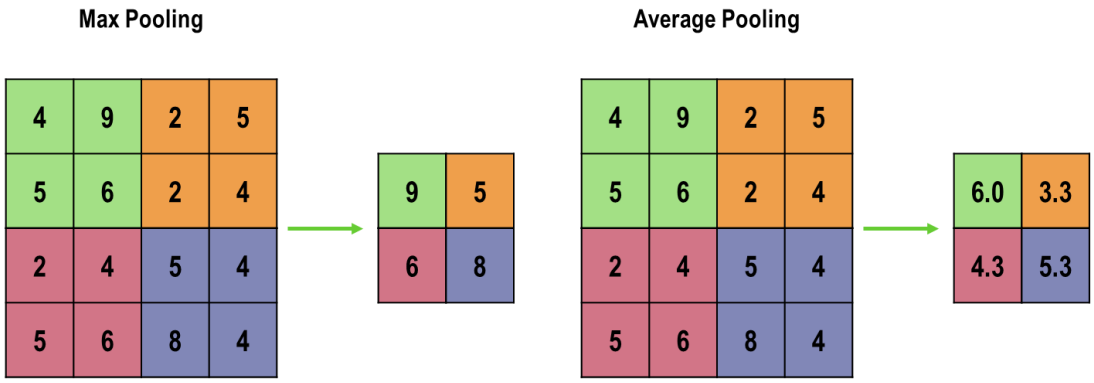

Abbildung 4: Max Pooling und Average Pooling.
Eigene Darstellung, Inhalt übernommen von (Prijono, 2018).

Fully Connected Layer

Der Fully Connected Layer dient zur Durchführung einer Klassifikation oder Regression, nachdem die vorangehenden Convolutional und Pooling Layers die relevanten Features aus den Inputdaten extrahiert haben. Typischerweise werden die Feature Maps aus dem letzten Convolutional oder Pooling Layer in einen eindimensionalen Tensor transformiert und dann an die Fully Connected Layers übergeben. Viele CNN-Architekturen verwenden eine oder mehrere Fully Connected Layers am Ende des Netzwerks, allerdings haben mittlerweile einige moderne Architekturen die Verwendung von Fully Connected Layers entweder reduziert oder ganz weggelassen. Die Anzahl Neuronen im letzten Layer eines CNNs dient dem Output einer Wahrscheinlichkeit (Klassifikation) oder dem Vorhersagen von Koordinaten einer Bounding Box (Regression) (o.A., 2023).

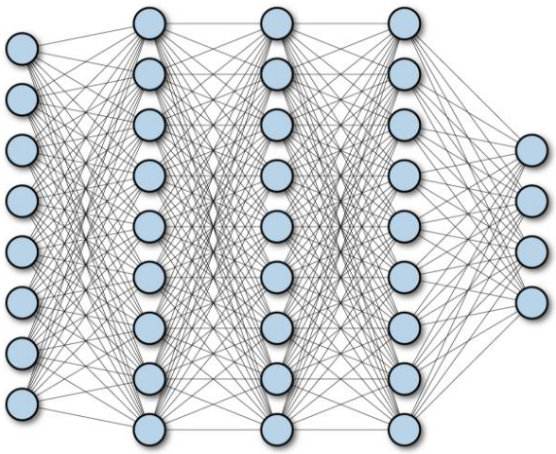

Abbildung 5: Fully Connected Layers mit der Tiefe 5. Jedes Neuron eines Layers ist mit jedem Neuron des nachfolgenden Layer verbunden. Der letzte Layer ist der Output Layer des Modells (Mahajan, 2020).

2.1 Modelle für Bildklassifikation

Dieses Unterkapitel beschreibt die, für Punkt IV (Siehe Kapitel 1.2) relevanten CNN-Modelle.
Klassifikation ist nach der Regression der zweite Modellierungsansatz im maschinellen Lernen (Supervised Learning). Klassifizierung kann als Zuordnung oder Gruppierung von Beobachtungen in Klassen, also vordefinierten Kategorien, bezeichnet werden. Ein Programm oder eine Funktion, welche Daten in Klassen ordnet, nennt man Klassifikator. Ein Label eines Datenpunkts enthält Informationen darüber, welche Klasse dieser Datenpunkt vom Klassifikator erhalten hat (Müller, 2022).

Ein tiefes neuronales Netzwerk, das zur Klassifikation von Bildern verwendet wird, ist das ResNet50, einer speziellen Variante der ResNet-Architektur (Residual Network). Es besteht aus 50 Layern und wurde im Jahr 2015 in einem Wissenschaftlichen Artikel veröffentlicht. Das Hauptmerkmal von ResNet ist die Einführung von sogenannten Residualblöcken, die ein Problem im Maschinellen Lernen, nämlich dem des verschwindenden bzw. explodierenden Gradienten, umgehen kann. In einem Residualblock wird die Eingabe direkt zum Ausgang dazu addiert, wodurch eine sogenannte Skip-Connection entsteht, was während des Modelltrainings einen positiven Effekt erzielt. Die Vorteile von ResNet50 sind insbesondere seine Leistung. Aufgrund seiner tiefen Architektur ist es fähig, komplexere Muster in Daten zu lernen (He et al., 2015). Resnet50 hat beispielsweise den ImageNet Large Scale Visual Recognition Challenge Wettbewerb im Jahr 2015 gewonnen und neue Standards für die Bilderkennung gesetzt. Aufgrund seiner Leistung wird es häufig als Backbone für komplexere Modelle eingesetzt (Tsang, 2018).

Ein weiteres verbreitetes CNN-Modell ist das VGG16-Modell, welches durch seine guten Leistungen beim ImageNet Large Scale Visual Recognition Challenge Wettbewerbs im Jahr 2014 Bekanntheit erlangte (ILSVRC, 2014). Es nutzt in seinen Convolutional Layern kleinere Kernel als damals üblich mit der Grösse 3x3, was zu einer geringeren Zahl an Gewichten führt, womit das Modell effizienter wird. Beispielweise entsprechen zwei aufeinanderfolgende 3x3 Konvolutionen effektiv einer 5x5 Konvolution. Allerdings haben die aufeinanderfolgenden 3x3 Konvolutionen weniger Gewichte und fügen zusätzliche Nichtlinearitäten hinzu, was die Ausdrucksstärke des Modells verbessern kann (Anwar, 2019). Das VGG16 ermöglicht tiefe Netzwerke mit bis zu 16 Layern und ist in der Lage, komplexe Muster in den Daten zu erfassen (Simonyan & Zisserman, 2014).

Die Modellwahl inklusive Begründung ist in Kapitel 3.4 beschrieben.

2.2 Modelle für Objekterkennung

Dieses Unterkapitel beschreibt die, für Punkt V (Siehe Kapitel 1.2) relevanten CNN-Modelle, die zur Objekterkennung eingesetzt werden können.

Im Kontext der maschinellen Zustandserkennung ist nicht nur die Klassifikation oder Identifikation eines einzelnen Objekts oder Merkmals von Interesse, sondern die Erkennung von mehreren Objekten oder Zuständen zur gleichen Zeit. Im Laufe der Zeit wurden dafür verschiedene Ansätze entwickelt. Das R-CNN Netzwerk (Region-based Convolutional Neural Networks) wurde speziell zur Objekterkennung in Bildern entwickelt. Es nutzt Region Proposal, was die Aufteilung des Eingabebilds in verschiedene Regionen bedeutet, die potenzielle Objekte enthalten könnten. Anschliessend wird jeder der vorgeschlagenen Bereiche durch ein CNN geführt, um Features zu extrahieren. Schlussendlich werden die Features durch einen Klassifikator geleitet, um zu bestimmen, welches Objekt in den jeweiligen Bereichen liegt (Girshick et al., 2013).

Mit der Weiterentwicklung zu Fast R-CNN wurden die Geschwindigkeit und Effizienz gesteigert. Fast R-CNN integriert die Bounding-Box-Regression und Klassifikation in einem einzigen Netzwerk, was bedeutet, dass beide Teile gleichzeitig trainiert werden können. Die Genauigkeit von Fast-R-CNN konnte mit der Verwendung einer Technik namens ROI-Pooling (Region of Interest Pooling) gesteigert werden. Trotz dieser Verbesserungen hatte das Fast R-CNN Netzwerk immer noch Engpässe bei der Geschwindigkeit, da das Erstellen der Region Proposals immer noch separat und relativ langsam war. (Girshick, 2015).

Mit Faster R-CNN wurde eine weitere, verbesserte Variante von R-CNN der Fachwelt vorgestellt. Die Integration der Region Proposal Generierung in das Hauptnetzwerk hat die Objekterkennung deutlich schneller gemacht als bei den älteren Versionen. Mit der Einführung des Region Proposal Network (RPN) wurde ein lernfähiges Modul in das Haupt-CNN integriert. Es teilt sich die Convolutional Layers und generiert Region Proposals direkt aus den Feature Maps, was eine Steigerung der Geschwindigkeit zur Folge hat. Mit diesen Verbesserungen kann das Faster R-CNN Objekterkennung in Echtzeit durchführen (Ren et al., 2015).

Ein weiterer möglicher Ansatz ist der Single Shot MultiBox Detector (SSD), der von (Liu et al., 2015) entwickelt wurde. Der Name ‘Single-Shot’ deutet schon darauf hin, dass die Objekterkennung in einem einzelnen Durchlauf durchgeführt wird. Dies ist eine Besonderheit, da es sich von zweistufigen Ansätzen, wie den R-CNN Varianten unterscheidet. Die zweistufigen Varianten schlagen zuerst eine Menge von

Regionen vor und führen dann eine Klassifikation und Bounding-Box-Regression auf diesen Regionen durch. SSD verwendet sogenannte Multi-Scale Feature Maps mit unterschiedlichen Grössen, was es effizient für das Erkennen von Objekten unterschiedlicher Grösse macht (Liu et al., 2015).

Mit Retinanet wurde 2017 ein weiteres leistungsfähiges Modell zur multiplen Objektdetektion entwickelt. Retinanet besteht aus zwei Hauptteilen, dem Backbone Netzwerk und der Feature Pyramid Network (FPN). Das Backbone Netzwerk besteht aus einem vortrainierten CNN-Modell, z.B. ein VGG16 oder Resnet50 Netzwerk, das zur Extraktion von Features aus dem Eingabebild verwendet wird. Das Backbone Netzwerk erzeugt eine Reihe von Feature Maps auf verschiedenen Skalen, die dann vom restlichen Retinanet Modell genutzt werden. Der zweite Teil, das Feature Pyramid Network (FPN), wird zum Erkennen von Objekten verschiedener Grössen genutzt. Es bietet eine effektive Methode zur Erkennung von Objekten verschiedenster Grössen, indem es die Merkmale aus verschiedenen Ebenen des Netzwerks extrahiert und kombiniert. Das FPN erzeugt eine Pyramide von Feature Maps auf verschiedenen Skalen, indem es starke Features aus den unteren hochauflösenden Bildern mit schwachen Features aus den oberen niedrig aufgelösten Feature Maps kombiniert (Lin et al., 2017).
Eine weitere Besonderheit von Retinanet ist die Focal Loss-Funktion. Sie ist eine spezielle Verlustfunktion und kann schwer zu klassifizierenden Beispielen mehr Gewicht geben, was das Modell dazu veranlasst, diese Beispiele besser zu lernen. Gemäss Aussage der Autoren übertrifft Retinanet bezüglich Genauigkeit (AP) Faster R-CNN auf dem COCO test-dev Datensatz (Lin et al., 2017).

Die Modellwahl inklusive Begründung ist in Kapitel 3.5 beschrieben.

3 Methoden und Implementierung

Dieses Kapitel beschreibt die Umsetzung des erarbeiteten Lösungsansatzes in die Praxis, basierend auf den Anforderungen in Kapitel 1.1 und den einzelnen Punkten in der Zielsetzung, beschrieben in Kapitel 1.2. Zuerst wird die Kamerainstallation, sowie der Aufbau der Datenpipeline (Punkt I) beschrieben. Anschliessend folgt das Speichern der Bilder auf Databricks in Form von Deltatabellen (Punkt II), sowie die Einbindung eines effizienten Labeling Prozesses (Punkt III). Als nächstes folgen die beiden Modelle (Klassifikator (Punkt IV), Objekterkennung (Punkt V)) und deren Umsetzung als Python Files in Databricks. Zuletzt werden die Themen Bilderablage, Labeling und Modelle mit dem Qualitätsmonitoring vereint (Punkt III).

3.1 Kamerasystem Anbindung mit Raspberry Pi und Balena OS inklusive Datenpipeline

Dieses Unterkapitel behandelt Punkt I aus der Zielsetzung, beschrieben in Kapitel 1.2.

Installation der Kamera

Der initiale Lösungsansatz bestand darin, die Kamera auf einem Metallprofil anzubringen, das direkt mit dem Ampelsystem verbunden ist. Die Distanz zwischen Kamera und Ampel beträgt knapp 30 cm. Das Anbringen des Metallprofils mit einer Schlauchschelle ist simpel, erfordert kein Spezialwerkzeug und dauert pro Kamera weniger als die geforderten 5 Minuten pro Ampel. Erste Testläufe waren vielversprechend in Bezug auf Modellgüte. Beim Betrachten der ersten Trainingsbilder wurde schnell klar, dass dieses Setup eine kritische Schwachstelle hat. Einige Bilder zeigten Produktionsmitarbeiter im Hintergrund, was ein Datenschutzrechtliches Problem darstellt. Möglichkeiten, wie das Einführen von Scheuklappen, wurden diskutiert und mit Prototypen getestet, jedoch war der Aufwand für das Einrichten und die Stabilität dieser Vorrichtung nicht zufriedenstellend.

Der verfolgte Lösungsansatz besteht im Nutzen von USB Endoskop Kameras, die mit Hilfe von Plastikschienen direkt an der Ampel befestigt werden. Das Nutzen einer Plastikschiene bringt folgende Vorteile:

- Datenschutz: Keine sensitiven Bilder von Personen und Einrichtungen.
- Ähnliche Lichtverhältnisse bei allen Kameras: Kein Einfluss von Raumhelligkeit und Tageszeit, da die Schiene nur das Licht von der Ampel durchlässt.
- Preis: Die USB Endoskop Kameras und Plastikprofile sind günstig und kosten weniger als das Metallprofil Setup.
- These: Das Modell fokussiert sich beim Training mehr auf relevante Features, anstelle von Features im Hintergrund.

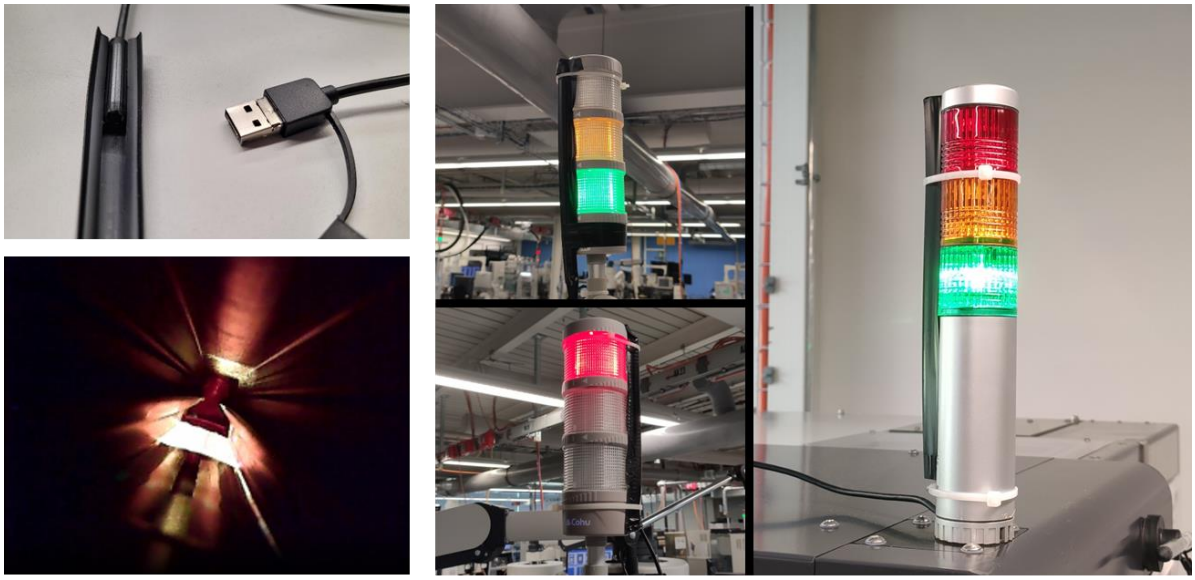

Abbildung 6: Links oben: Endoskop Kamera in einer Plastikschiene. Links unten: Bild aus der Plastikschiene. Rechte Bildhälfte: 3 unterschiedliche Ampeltypen ausgestattet mit Plastikschiene und Kamera.

Erste Tests mit diesem Setup zeigten, dass dieser Ansatz sämtliche Anforderungen in Bezug auf Datenschutz, Zeit zum Einrichten und Einfachheit erfüllt.

Aufbau Datenpipeline

Wie bereits in Kapitel 1.1 beschrieben, ist die Verwendung eines Raspberry Pi eine Anforderung der Projektleitung. Als Betriebssystem wird ein Balena OS Image verwendet. Balena ist ein Unternehmen, das auf Dienstleistungen im Bereich IoT Geräte (Internet of Things) spezialisiert ist. Das Unternehmen stellt konfigurierbare OS-Images für Raspberry Pi Geräte zur Verfügung und bietet eine Umfassende Geräteverwaltung online an. Prinzipiell bietet die Nutzung folgende Vorteile:

- Skalierbarkeit: Die Anzahl Geräte kann schnell erweitert werden.
- Containerbasierte Umgebung: Keine Probleme mit Python Paketinkompatibilitäten.
- Online-Verwaltung aller Geräte: Alle Geräte können webbasiert überwacht, neu gestartet und mit neuen Updates versehen werden. Ein Entwicklermodus unterstützt das Erweitern von bestehenden Skripten. Die Verknüpfung mit einem GitHub Repository ermöglicht es, Änderungen an Python Skripten automatisch auf alle Geräte gleichzeitig zu übernehmen (Balena, n.d.).

Ein Python Skript, welches in der Balena OS als Dauerschleife ausgeführt wird, sorgt für die Anbindung der USB Endoskop Kamera auf dem Raspberry Pi. Das Skript sucht mit jeder Iteration nach USB-Geräten und lokalisiert den angeschlossenen USB-Port, da mehrere Kameras pro Raspberry Pi angeschlossen werden können. Anschliessend wird pro Kamera ein Foto gemacht und das Streaming der Bilder mit Kafka vorbereitet.

Die Kamerabilder werden via Kafka auf einer Cloud für 7 Tage zwischengespeichert. Eine Databricks Anwendung greift auf den Kafka Zwischenspeicher zu und holt sich die Daten mit Hilfe von sogenannten Spark Workers. Schlussendlich werden die Bilder auf einer Deltatabelle in Databricks gespeichert. Eine Beschreibung über Deltatabellen, sowie dem Delta Lake Speichersystem folgt im nächsten Unterkapitel.

3.2 Aufbau Databricks Infrastruktur

Dieses Unterkapitel beschreibt die Infrastruktur, die auf Databricks implementiert ist und behandelt Punkt II, beschrieben in Kapitel 1.2. Es beschreibt den Aufbau des Databricks Speichersystems, in welches die Bilder gestreamt werden. Zudem wird erläutert, wie das Datenhandling, CNN-Modelle und Qualitätsmonitoring auf Databricks implementiert sind.

Delta Lake

Databricks arbeitet standardmässig mit einem eigens entwickelten Speicherformat namens Delta Lake. Delta Lake ist ein Open-Source-Projekt zur optimierten Speicherung von grossen Datenmengen. Es unterstützt ACID (Atomicity Consistency Isolation Durability) Transaktionen, was in herkömmlichen Big Data-Umgebungen oft fehlt. Dies sorgt für Zuverlässigkeit und Konsistenz der Daten und ermöglicht gleichzeitige Lese- und Schreibvorgänge. Delta Lake ist für den Einsatz mit grossen Datenmengen optimiert und bietet Funktionen, welche die Leistung von Datenabfragen verbessern. Zudem ermöglicht Data Versioning oder auch umgangssprachlich Zeitreise genannt, dass Benutzer ältere Versionen der Daten abrufen können (o.V., 2023b).

Deltatabellen und Datenformat

Delta Lake basiert auf der Idee von Deltatabellen, einer speziellen Art von Tabellen für die Speicherung und den Zugriff auf Daten. Eine Deltatabelle bildet die Versionierungsschicht auf den eigentlichen Daten, die im Delta Lake gespeichert sind. Sie sind in der Lage, verschiedene Versionen der Daten zu speichern. Eine Änderung an den Daten wird als neue Version in der Deltatabelle gespeichert, was eine vollumfängliche Rückverfolgung aller Änderungen ermöglicht (o.V., 2023b).

Die von der Endoskop Kamera aufgenommenen Bilder werden im Format eines Binärstrings in einer Deltatabelle auf Databricks gespeichert. Die Deltatabelle beinhaltet folgende Spalten:

Tabelle 2: Übersicht Datentypen des Bilder Streams in der Deltatabelle.

Spaltenname	Datentyp	Beschreibung
Timestamp	Timestamp	Ein Zeitstempel zum Zeitpunkt der Bildaufnahme.
Image	Struct	Das aufgenommene Bild, verpackt als struct. Dieser beinhaltet nebst den Bilderdaten in Form eines Binärstrings Informationen zu Höhe, Breite und Anzahl Farbkanäle.
ComponentId	String	Ein zufällig generierter einmalig vorhandener String, der zur Identifikation einer Kamera dient.
Year	Integer	Eine vierstellige Zahl, die das Jahr der Bildaufnahme beinhaltet.
Month	Integer	Eine zweistellige Zahl, die den Monat der Bildaufnahme beinhaltet.

Die Delta Lake Architektur richtet sich nach dem Prinzip von verschiedenen Schichten, jede davon bestehend aus Deltatabellen. Zweck ist es, die Daten über mehrere Schichten hinweg aufzubereiten und für eine visuelle Darstellung oder weitere Datenanalysen vorzubereiten. Folgende Tabellen werden im Rahmen dieser Masterarbeit verwendet:

Bronze Tabelle: Die Bronze Tabelle bildet die unterste Schicht im Delta Lake und enthält die unverarbeiteten Rohdaten, in diesem Falle die Bilder von den Kameras. Sie sind als Binärstrings formatiert und haben nebst den in Tabelle 2 beschriebenen Informationen keinen weiteren Informationsgehalt.

Silver Tabelle: Die Silver Tabelle ist die mittlere Schicht im Delta Lake und beinhaltet sämtliche Informationen aus dem Bronze Layer (ohne die Bilddaten), sowie die Modellvorhersagen des aktuell verwendeten CNN-Modells. Die Modellvorhersagen in dieser Tabelle werden nicht nur für die Visualisierung, sondern auch für die regelmässigen Qualitätstests verwendet.

Gold Tabelle: Die Gold Tabelle stellt die letzte Aggregationsform dar und beinhaltet Informationen, die zur Darstellung auf dem Dashboard benötigt werden. Im Fall des zu erarbeitenden Ampelsignal Systems enthält es Informationen zum Prozentuellen Anteil der einzelnen Klassen oder Ampelsignalen (Lee & Heintz, 2019).

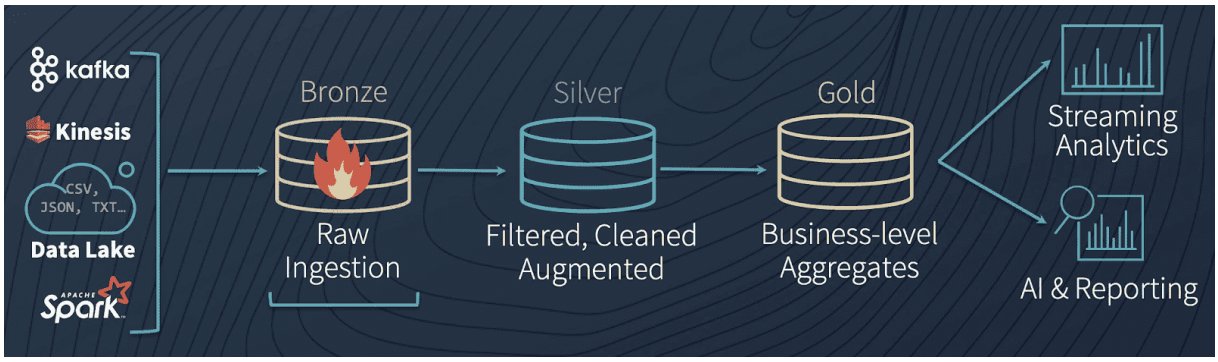

Abbildung 7: Delta Lake Tabellenschema (Lee & Heintz, 2019).

3.3 Labeling der Bilder

Dieser Unterkapitel behandelt Punkt III aus der Zielsetzung, beschrieben in Kapitel 1.2 und beschreibt den Aufbau des Labelingsystems.

Für das Generieren von Trainingsdaten wird eine Anbindung von Databricks auf Labelbox verwendet. Labelbox ist eine Plattform für das Labeln und Verwalten von Trainingsdaten für Machine Learning Modelle. Die zu kennzeichnenden Bilder werden nicht direkt auf einen Labelbox Server geschickt, sondern auf einem sogenannten Blob-Speichercontainer (Binary Large Object) auf Databricks abgelegt. Jedes Bild auf diesem Blob erhält eine eindeutige URL, mit welcher auf das Bild zugegriffen werden kann (Labelbox, 2023b). Durch eine API (Application Programming Interface) Anbindung zwischen Databricks und Labelbox können Informationen in beide Richtungen gesendet werden. Zunächst werden Bildinformationen, u.a. die URL der zu kennzeichnenden Bilder an Labelbox geschickt (Labelbox, 2023c). Anschliessend werden die Bilder von einem Nutzer angeschaut und direkt annotiert. Ist ein Labeling Projekt fertig, werden die annotierten Daten (z.B. Class Label, Bounding Box Koordinaten etc.) mit einem Python Skript direkt importiert.

Model Assisted Labeling

Ein wichtiger Bestandteil von Labelbox ist das sogenannte Model Assisted Labeling (MAI). Es erlaubt das Einbinden eines Machine Learning Modells in den Labeling-Prozess. Die Vorhersagen des Modells werden zusammen mit den Bilddaten an Labelbox geschickt und müssen dann manuell überprüft werden. Einfach gesagt, werden die Bilder mitsamt den Labeln übermittelt (Labelbox, 2023a). Dies bringt mehrere Vorteile mit sich. Der Aufwand für das Labeling verringert sich, besonders bei guter Modellperformance. Gleichzeitig werden im Rahmen dieses Projektes die daraus erzeugten Daten als neue Trainingsdaten und als Qualitätsindex für die Modellgüte der beiden zu erarbeitenden Modelle erhoben.

3.4 CNN-Modell Klassifikator für Ampelsignale

Nach dem Aufbau der Datenpipeline und dem Anbinden von Labelbox als Plattform für das Labeling der Bilder folgt als nächster Schritt der Aufbau eines CNN-Modells für die Klassifikation von Ampelsignalen. Das Erkennen von Ampelsignalen ist Hauptbestandteil des Projekts 'Lean Production Monitoring' und in den Anforderungen in Kapitel 1.1 sowie als Punkt IV in der Zielsetzung (Kapitel 1.2) festgehalten.

Modellwahl

Für den Aufbau des Basismodells zur Klassifikation stehen, wie in Kapitel 2.1 beschrieben, zwei performante Modelle zur Auswahl: ResNet50 und VGG16. Der Entscheid war daher schwierig, fiel schlussendlich aber auf VGG16. Trotz seines fortgeschrittenen Alters bietet VGG16 viele gute Gründe, als Klassifikator eingesetzt zu werden. Mit seiner Architektur aus 16 Layern in Kombination mit den 3x3 Kernels, mit einem Stride von 1, und den 2x2 Max-Pooling Layern mit einem Stride von 2, folgt es über die gesamte Modellarchitektur dem gleichen Aufbau. Im Gegensatz zu komplexeren Modellen hat VGG16 wenige Hyperparameter und erfordert weniger Tuning als komplexere Modelle. Des Weiteren ist VGG16 auf dem ImageNet Datensatz trainiert und kann mit vortrainierten Gewichten genutzt werden, was der angedachten Rolle eines universell einsetzbaren Klassifikators entgegenkommt (Thakun, 2023).

In diesem Projekt wird das VGG16 Modell ohne die Fully Connected Layers implementiert. Da es für individuelle Klassifikationsaufgaben eingesetzt wird, werden die Fully Connected Layers während des automatischen Modellaufbaus der Anzahl möglicher Objekte/Klassen angepasst. Die 13 Convolutional Layers werden mit den ImageNet Gewichten übernommen. Diese Methode wird als Transfer Learning bezeichnet und ist in der Machine Learning Community verbreitet, da die für ein Training benötigte Rechenleistung reduziert wird (Cassimiro, 2021). Transfer Learning ist auch ganz im Sinne des Projekts 'Lean Production Monitoring', da die Modelle mit möglichst wenig Trainingsbildern auskommen müssen.

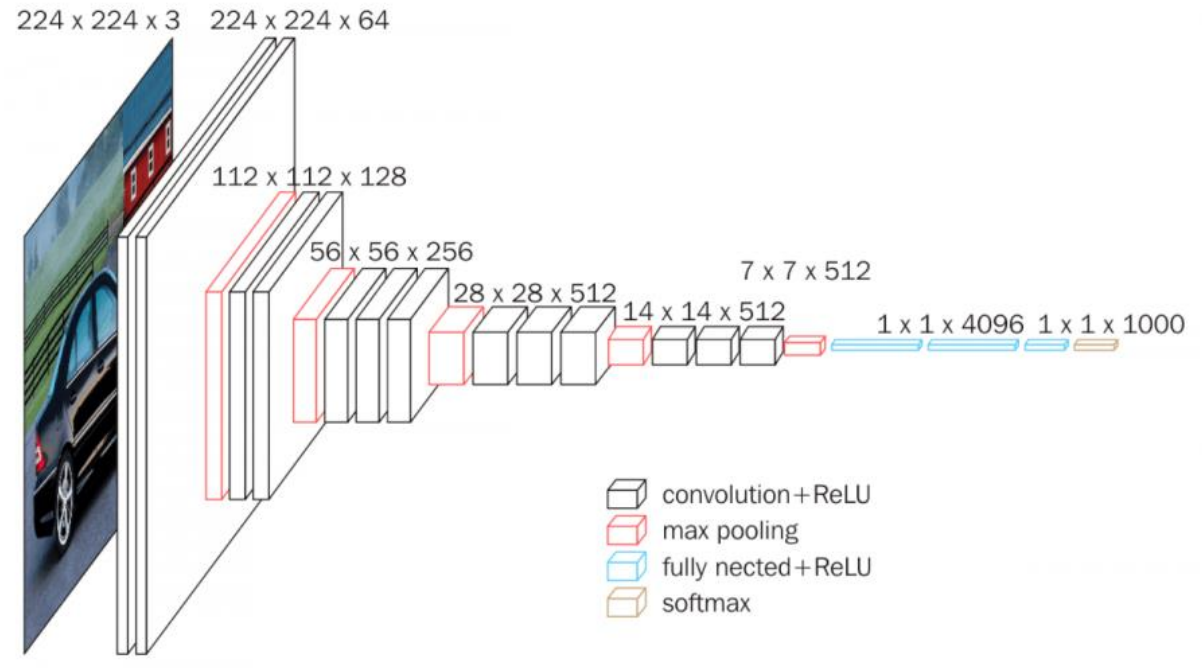

Abbildung 8: Aufbau des VGG16 Netzwerks (Simonyan & Zisserman, 2014).

Abbildung 8 zeigt den Aufbau des VGG16 Netzwerks. Das Eingabebild wird über insgesamt 13 Convolutional Layers verarbeitet (Bezeichnet als 'convolution + ReLU'), mit 5 Max Pooling Layers zwischen den Convolutional Layern. Die 3 blau eingefärbten Schichten bilden die Fully Connected Layers, die in diesem Bild zur Klassifikation von 1000 Objekten eingesetzt werden (Simonyan & Zisserman, 2014).

Implementierung als MLFlow Modell auf Databricks

Die Modellerstellung, Modelltraining und Verwaltung wird mit dem Paket MLflow durchgeführt. MLflow ist wie Delta Lake eine Entwicklung von Databricks und ermöglicht eine bessere Organisation von Machine Learning Workflows. Es umfasst 4 Hauptkomponenten:

- MLflow Tracking: Eine Benutzeroberfläche inklusive API zum Aufzeichnen und Abrufen von Experimenten mit verschiedenen Parametern und Metriken.
- MLflow Projects: Hierbei handelt es sich um die Möglichkeit, wiederverwendbaren Code in einem konsistenten Format zu verpacken und diesen mit anderen zu Teilen.
- MLflow Models: Die Möglichkeit, ML-Modelle in ein Standardformat zu verpacken.
- MLflow Model Registry: Ein zentrales Modell-Repository, welches alle registrierten Modelle und deren Versionen über den gesamten Lebenszyklus verfolgen kann (o.V., 2023a).

Die Modellimplementierung erfolgt in diesem Projekt in einem Python File, welches eine *mlflow.pyfunc.PythonModel* Klasse enthält. Die Klasse beinhaltet Funktionen, die für das Training (fit) und die Vorhersage (predict) vorgesehen sind.

Für das Training eines Modells wird eine Instanz dieser Klasse erzeugt, wobei modellspezifische Hyperparameter als Dictionary übergeben werden. Nach erfolgreichem Training wird das Modell als MLflow Modell registriert und kann mit einem Befehl aus der Model Registry geladen werden. Dieser Ansatz bringt viele Vorteile. Zum einen kann die Klasse mit unterschiedlichen Trainingsdaten zum Erzeugen von beliebig vielen Modellen mit unterschiedlichen Aufgaben verwendet werden. Des Weiteren bietet die MLflow Model Registry die Möglichkeit, Modelle einfach zu speichern, laden und zu verwalten (o.V., 2023a). Zu Guter Letzt erlaubt die Implementierung als MLFlow Klasse das Einbinden einer Vorverarbeitungs-Pipeline für Bilder bei Modellvorhersagen. Das bedeutet, dass das trainierte Modell Bilder mit unterschiedlichen Grössen akzeptiert, da es diese automatisch skaliert und für das VGG16 Modell kompatibel aufbereitet.

Hyperparameter Tuning

Für die optimale Auswahl bestimmter Hyperparameter wird das Paket Hyperopt verwendet. Hyperopt ist in der Lage, die ideale Kombination für definierte Hyperparameter eines ML-Modells in mehreren Durchläufen zu ermitteln. Für das VGG16 Modell wurden folgende Hyperparameter für die Optimierung ausgewählt:

- Learning Rate: Diese bestimmt, wie stark das Modell während des Trainings aktualisiert wird. Eine hohe Learning Rate führt dazu, dass die Gewichte stärker verändert werden. Dies kann zu schnellerem Lernen führen, aber auch schädlich für die Modellqualität sein, da das Modell die optimalen Gewichte überspringt (Jordan, 2018).
- Anzahl Neuronen pro Layer: Dies bestimmt, wie viele Neuronen in einem Layer eingesetzt werden. Ein Modell mit zu wenig Neuronen läuft Gefahr, aufgrund der zu geringen Komplexität ein Muster nicht lernen zu können. Ein Modell mit zu viel Komplexität kann mit Problemen des Overfitting zu kämpfen haben (Krishnan, 2021).
- Anzahl Fully Connected Layer im Modell (Modelltiefe): Eine Erhöhung der Anzahl an Fully Connected Layer im Modell kann ebenfalls zu mehr Modellkomplexität führen. Nachteile bei mehr Schichten sind einerseits die Gefahr von Overfitting sowie das Problem des Vanishing/Exploding Gradient. Ein Problem, bei dem insbesondere tiefe Netzwerke während des Trainings der Gradient entweder klein oder extrem grossen werden kann, was das Training erschwert (Wang, 2019).

3.5 Ausweitung des Systems auf weitere Aufgaben: CNN-Modell Objekterkennung

Eine Anforderung der Projektleitung des 'Lean Production Monitoring' Projekts ist es, das System zur Ampelsignalerkennung so flexibel zu gestalten, dass es auch weitere Aufgaben übernehmen kann. Das VGG16 Modell, beschrieben in Kapitel 3.4, kann zwar für Klassifikationsaufgaben eingesetzt werden, jedoch ist es nicht dafür ausgelegt, mehrere Objekte gleichzeitig in einem Bild zu lokalisieren und richtig zu klassifizieren. Ein weiteres Modell mit diesen Fähigkeiten kann zum Erkennen und Zählen von Produktionsmaterial in manuellen Arbeitsschritten eingesetzt werden oder dazu dienen, Materialengpässe, z.B. das Fehlen von Nachschubmaterial beim Materialfeeder einer Maschine zu erkennen. Im Gespräch mit einer Fachperson, welche sich auf Lean Consulting spezialisiert hat, ist eine der wichtigsten Massnahmen das Überwachen von Materialflüssen und deren Verfügbarkeit bei den Produktionsmaschinen. Das Einführen eines weiteren, komplexeren CNN-Modells ermöglicht es, die Möglichkeiten mit optischer Bilderkennung innerhalb dieses Projekts zu erweitern. Das CNN-Modell beschrieben in diesem Unterkapitel deckt Punkt V in der Zielsetzung (Kapitel 1.2).

Modellwahl

Für das Erkennen von mehreren Objekten in einem Bild wird in diesem Projekt ein Retinanet Modell verwendet. Aufgrund seiner Focal Loss-Funktion und seiner Leistungsfähigkeit im Detektieren von verschiedenen Objekten mit unterschiedlichen Grössen, wurde Retinanet als Modell zur Objekterkennung im Projekt 'Lean Production Monitoring' ausgewählt. Da auch für dieses CNN-Modell der Trainingsaufwand so tief wie möglich gehalten werden soll, wird ein Retinanet Modell mit einem vortrainierten Resnet50 Netzwerk als Basismodell, oder auch Backbone genannt, verwendet. Dieses Modell stammt aus dem offiziellen Modellpaket von der Softwarebibliothek Tensorflow (tf-models-official).

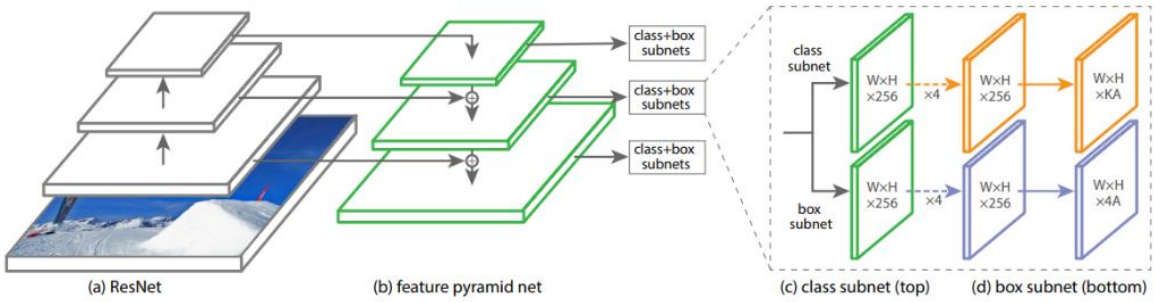

Abbildung 9: Aufbau des Retinanet Modells (Lin et al., 2017).

Die Funktionsweise von Retinanet erfolgt in folgenden Schritten. Ein Resnet50 Netzwerk verarbeitet das Eingabebild und generiert eine Pyramide von Feature Maps. Die niedrigen Ebenen der Pyramide haben eine höhere Auflösung, während die höheren Ebenen eine niedrige Auflösung, dafür grössere

Kontextinformationen beinhalten. Diese Kombination von Feature Maps mit unterschiedlichen Auflösungen und Skalen ermöglicht es, unterschiedliche Objekte aller Grössen und Formen zu erkennen. Diese Ansammlung an Feature Maps werden in einem nächsten Schritt durch zwei separate Subnetzwerke geleitet. Das erste dient zur Klassifikation (Siehe Abbildung 9 'class subnet') und führt eine Klassifikation für jede Position im Bild durch. Dieses Subnetzwerk besteht aus mehreren Convolutional Layern, welche in Kapitel 2 erklärt werden. Das zweite Subnetzwerk ist für das Schätzen der Bounding Box Koordinaten mit einer Regression zuständig und verwendet ebenfalls mehrere Convolutional Layers. Die Ausgabe des Modells ist die Information über die Klasse und Position des Objekts(Lin et al., 2017).

Implementierung als MLFlow Modell auf Databricks

Wie bereits in Kapitel 3.4 beschrieben, wird auch dieses Modell mit dem MLflow Paket als *mlflow.pyfunc.PythonModel* Klasse implementiert. Die Vorteile von MLflow werden in besagtem Kapitel aufgeführt. Wichtig zu erwähnen ist, dass beide Modelle als eigene Klasse in einem separaten Python File gespeichert sind. Aufgrund seiner Integration in Tensorflow war es deutlich aufwändiger, Retinanet mit dem vortrainierten Resnet50 Backbone im gleichen Stil wie das VGG16 Netzwerk zu implementieren. Insbesondere das Speichern der Trainingsbilder in einem Spezialformat (TFRecord) sowie die Schwierigkeiten beim Hinzufügen von Bildaugmentationen haben die Integration dieses Modells viel Ressourcenaufwändiger gemacht, wie das VGG16 Netzwerk (Tensorflow, 2023a). Dem normalen Nutzer fällt dies nicht auf, da das Modell die Trainingsdaten im gleichen Format übergibt, wie beim VGG16 Modell.

Hyperparameter Tuning

Auch für dieses Modell wird mit dem Paket Hyperopt gearbeitet. Es erlaubt, die Hyperparameter bei einem neuen Modell mit wenig manuellem Aufwand zu ermitteln, um das Maximum an Modellperformanz zu erzielen. Retinanet verwendet eine Loss Funktion namens Focal Loss, die dafür entwickelt wurde, das Problem des Klassenungleichgewichts bei der Objekterkennung zu lösen. Die Focal Loss Funktion ist eine modifizierte Variante der Cross Entropy Loss Funktion, welche häufig bei Klassifikationen eingesetzt wird. Die Formel ist:

$$FL = -\sum_{i=1}^{i=n} \alpha_i (1 - p_i)^{\gamma} log(p_i)$$

Wobei p die vorhergesagte Wahrscheinlichkeit der Klasse i ist. Die Parameter Alpha und Gamma werden im folgenden Abschnitt erklärt (Nayak, 2022).

Bei der Objekterkennung gibt es oft das Problem, dass zwischen Hintergrundklassen (kein Objekt) und tatsächlichen Objekten ein Ungleichgewicht gibt. Die Focal Loss Funktion hat zwei Parameter, um dieses Problem zu behandeln.

- **Alpha:** Der Alpha Hyperparameter ist ein Skalierungsfaktor, der zur Ausgleichung dieses Ungleichgewichts gedacht ist. Mit Alpha lässt sich steuern, wie viel Gewichtung das Modell den positiven oder negativen Beispielen (wahre oder falsche Vorhersagen) beimisst. Dies ist besonders nützlich, wenn die Klassen unausgewogen sind, es also viel mehr Beispiele einer Klasse gibt als der anderen (Tsang, 2019).
- **Gamma:** Gamma ist neben Alpha der zweite Hyperparameter der Focal Loss Funktion und dient zum Steuern des Verhältnisses zwischen der Gewichtung von schwierigen und einfachen Beispielen. Ein höherer Gamma Wert sorgt dafür, dass schwierig zu klassifizierende Beispiele mehr Gewicht erhalten als einfache (Tsang, 2019).

Folgende weitere Parameter werden in der Optimierung berücksichtigt:

- **Aspect Ratios**: Die Aspect Ratios beziehen sich auf die Seitenverhältnisse von Ankerboxen. Ankerboxen werden in der Objekterkennung in verschiedenen Grössen und Seitenverhältnissen über das ganze Eingabebild verteilt. Anschliessend wird jede Box auf mögliche Objekte untersucht. Die Seitenverhältnisse werden üblicherweise so gewählt, dass eine möglichst grosse Variation an Objekten damit abgedeckt werden kann (Christiansen, 2018).
- **Anchor Size**: Die Anchor Size bezieht sich auf die Grösse der Ankerboxen. Diese wird in Bezug auf die Feature Stride definiert. Die Stride einer Feature Map beschreibt, wie viele Pixel des Eingabebilds in einer Zelle der Feature Map ist. Hat ein Eingabebild zum Beispiel 224x224 Pixel und die Feature Map eine Grösse von 56x56 Pixel, dann ist der Feature Stride 4 (Brownlee, 2019). Die Anchor Size ist ein Skalierungsfaktor abhängig vom Stride. Eine Anchor Size von 1 bedeutet, dass die Ankerbox gleich gross ist wie der Stride auf der jeweiligen Ebene einer Feature Map (Tensorflow, 2023b). Ein Wert von 2 bedeutet, dass die Ankerbox doppelt so gross ist, wie der Stride.
- **L2 Weight Decay/L2-Regularisierung**: Hierbei handelt es sich um eine Technik zur Verhinderung von Overfitting. Hierbei werden grosse Gewichte in den Modellparametern bestraft, indem ein Regularisierungsterm zur ursprünglichen Loss Funktion hinzugefügt wird, der proportional zum Quadrat der L2-Norm (also dem quadratischen Betrag) der Gewichte ist (Mishra, 2020).
- **Box Loss Weight**: Dieser Hyperparameter steuert das Gleichgewicht zwischen dem Classification Loss und dem Box Loss. Der Classification Loss misst, wie gut das Modell die Klassen von Objekten vorhersagt. Der Box Loss misst, wie gut das Modell die Bounding Boxes

der Objekte vorhersagt. Über diesen Hyperparameter waren praktisch keine Informationen verfügbar, da dieser im Tensorflow Modell des Retinanet nur erwähnt wird. Beim Betrachten des Quellcodes auf Github stellt sich heraus, dass ein höherer Box Loss Weight Wert dazu führt, dass der Box Loss mehr Einfluss auf den Gesamtverlust (Classification Loss & Box Loss) hat und somit den Fokus auf eine genauere Bounding Box Vorhersage legt (o.A., 2021).

- **Optimizer**: Der Optimizer ist zuständig für die Aktualisierung der Gewichte in einem CNN-Modell während des Trainings mit dem Ziel, die Loss-Funktion zu minimieren. Für dieses Modell werden zwei Optimizer in Betracht gezogen, zum einen den Stochastic Gradient Descent (SGD) und zum anderen den Adaptive Moment Estimation (Adam) Optimizer, welcher bereits im VGG16 Klassifikator eingesetzt wird. Die Frage, welcher Optimizer besser ist wie der andere, ist schwierig zu beantworten und wird in der Fachwelt eifrig diskutiert. Zum Teil wird die Meinung vertreten, dass der SGD Optimizer im Allgemeinen besser abschneidet als Adam. Zwar konvergiere Adam schneller, jedoch sei der SGD besser verallgemeinert als Adam und führe somit zu einer besseren Endleistung. Andere Meinungen besagen, dass ein fein abgestimmter Adam Optimizer immer noch besser abschneide als SGD (Park, 2021). Diese Diskussionen heben die Wichtigkeit von gut abgestimmten Hyperparametern hervor, was die Anwendung von Hyperopt umso wichtiger macht.
- **Learning Rate**: Diese bestimmt, wie stark das Modell während des Trainings aktualisiert wird. Eine hohe Learning Rate führt dazu, dass die Gewichte stärker verändert werden. Dies kann zu schnellerem Lernen führen, aber auch schädlich für die Modellqualität sein, da das Modell die optimalen Gewichte überspringt (Jordan, 2018).

3.6 Qualitätsmonitoring

Dieses Unterkapitel deckt Punkt II in der Zielsetzung (Kapitel 1.2) in Bezug auf Qualitätsmonitoring ab. Eine weitere Anforderung der Projektleitung ist es, die Modellgenauigkeit über die gesamte Lebensdauer aller aktiven Modelle zu überwachen und bei abfallender Genauigkeit eines Modells, dieses neu zu trainieren. Ein gestecktes Ziel bezüglich Modellgenauigkeit für das Modell zur Klassifikation von Ampelsignalen ist ein F-Score von mindestens 95% auf allen möglichen Farbkombinationen, ungeachtet des Ampeltyps. Die Anforderung, wie oft die Qualitätsüberprüfung stattfinden soll, sind nicht gegeben. Bereits in Kapitel 3.3 wird das sogenannte Model Assisted Labeling beschrieben, eine Möglichkeit, die Bilder mitsamt Modellvorhersage an Labelbox zu senden. In dessen Interface können die Annotationen eines bestehenden Modells von einem Nutzer überprüft und im Zweifelsfall korrigiert werden. Die Vorhersagedaten für die Bilder sind, wie in Kapitel 3.2 beschrieben, in der Silver Deltatabelle abgelegt, der Zwischenschicht zwischen den Rohbildern in der Bronze Deltatabelle und den aggregierten Daten für

das Dashboard in der Gold Deltatabelle. Das gesamte Qualitätsmonitoring besteht aus einem Python Funktionsfile, welches in Abständen von 3 bis 4 Tagen mit einem Databricks-Job eine zufällige Stichprobe von 20 Bildern pro Kamera entnimmt und diese per API-Anbindung mitsamt der Modellvorhersage an Labelbox sendet. Diese Zeitspanne wird zukünftig erweitert, sobald sich zeigt, dass das Modell auch über längere Zeiten stabil bleibt. Eine E-Mail von Databricks informiert die verantwortliche Person, dass der Upload Job ausgeführt wurde. Die zu überprüfenden Bilder sind bereits im richtigen Projekt in Labelbox hinterlegt und können vom User per Mausklick bestätigt oder überarbeitet werden. Die eigentliche Qualitätsüberprüfung findet in einem weiteren Job statt, der alle Labels der vergangenen 4 Tage aus Labelbox importiert und anschliessend mit den Modellvorhersagen aus der Silver Deltatabelle vergleicht. In diesem Schritt werden nur Label aus Labelbox importiert, die vom User manuell überprüft wurden und somit als wahre Label gelten. Je nach Modelltyp (Klassifikator oder Objekterkennung) wird die passende Qualitätsmetrik auf den Daten berechnet, welche in den beiden nachfolgenden Abschnitten erläutert wird. Im letzten Schritt werden die berechneten Qualitätsmetriken in die 'Quality' Deltatabelle geschrieben.

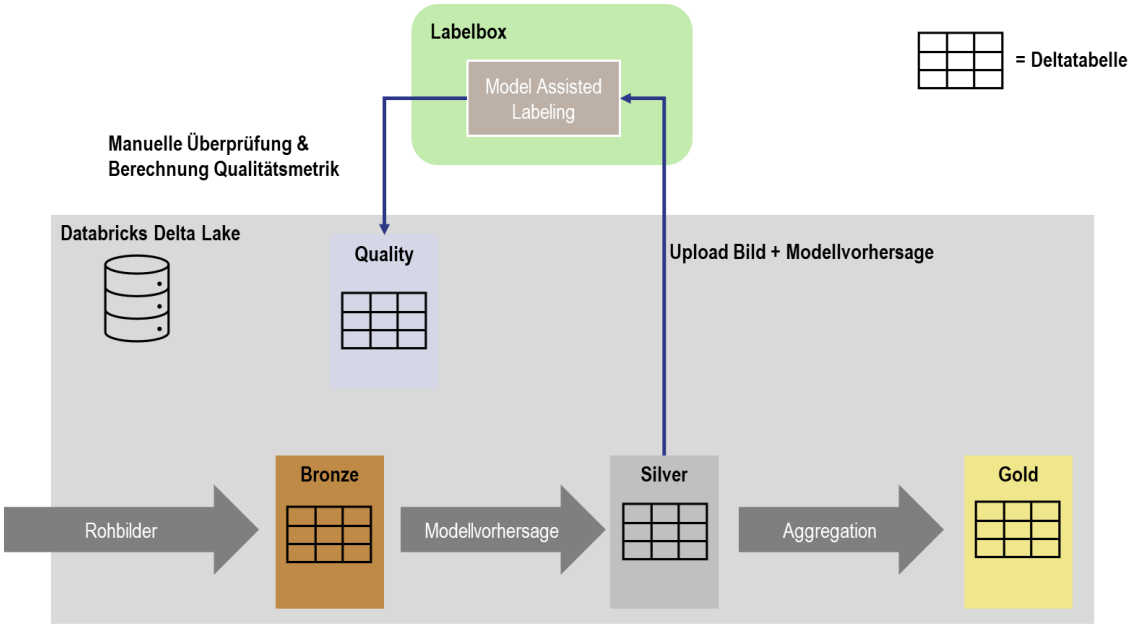

Abbildung 10: Schema Qualitätsmonitoring.

Qualitätsmetrik für Klassifikator

Für alle Klassifizierungsmodelle, das heisst alle Instanzen des VGG16 Klassifikator Modells (siehe Kapitel 3.4), werden die Metriken Accuracy, Precision, Recall und F1-Score verwendet. Die Definitionen dieser Metriken lassen sich mit einer Konfusionsmatrix erklären und sind wie folgt:

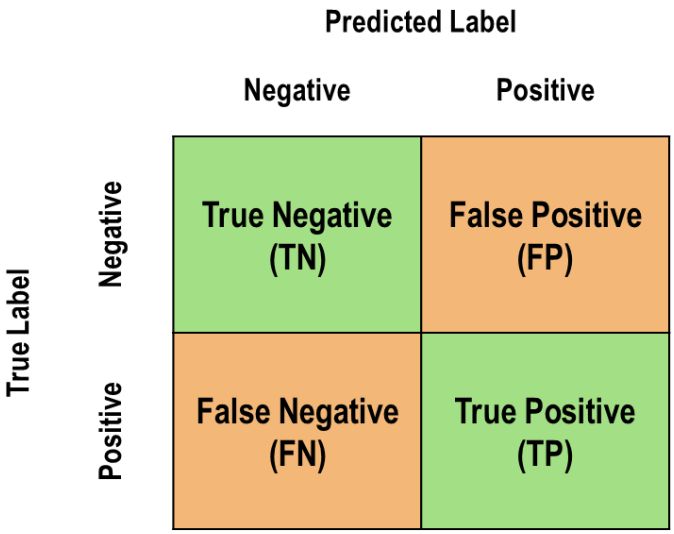

Abbildung 11: Konfusionsmatrix für Modellvorhersagen.
Eigene Darstellung, Inhalt übernommen von NB (2019).

Accuracy: Beschreibt den Prozentsatz der Vorhersagen, die korrekt waren.

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

Precision: Beschreibt den Anteil der wahren positiven Vorhersagen an allen positiven Vorhersagen, oder vereinfacht gesagt, die Genauigkeit (Precision) der positiven Vorhersagen des Modells. Ein hoher Precision Wert bedeutet, dass die meisten oder alle vorhergesagten positiven Fälle tatsächlich positiv sind.

$$Precision = \frac{TP}{TP + FP}$$

Recall (Sensitivity): Beschreibt das Verhältnis der richtig erkannten positiven Vorhersagen zur Gesamtzahl aller tatsächlich positiven Ergebnissen. Ein hoher Recall Wert bedeutet, dass das Modell die Fähigkeit hat, die meisten positiven Fälle richtig zu erkennen. Diese Metrik ist insbesondere dann wichtig, wenn keine positiven Fälle übersehen werden sollten, zum Beispiel beim Erkennen von Krankheiten.

$$Recall = \frac{TP}{TP + FN}$$

F1-Score: Der F1-Score beschreibt das harmonische Mittel von Precision und Recall. Diese Metrik ist nützlich, wenn Precision und Recall beiderseits wichtig sind und ein Ausgleich zwischen ihnen erforderlich ist.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Sämtliche Formeln und Beschreibungen sind in der Literatur bekannt und wurden übernommen NB (2019).

Für die Qualitätsüberprüfung sämtlicher Klassifikationsmodelle werden folgende Informationen in die Deltatabelle geschrieben:

- **ComponentId**: Der ID-Code, der jede Kamera eindeutig zuordnet.
- **F1-Score**: Der F1 Score, berechnet aus dem aktuellen Datensatz.
- **Recall**: Der Recall Score, berechnet aus dem aktuellen Datensatz.
- **Precision**: Der Precision Score, berechnet aus dem aktuellen Datensatz.
- **Accuracy**: Der Accuracy Score, berechnet aus dem aktuellen Datensatz.
- **ImageCount**: Die Anzahl Bilder im zu überprüfenden Datensatz.
- **TimeStart**: Ein Zeitstempel, wann das älteste Bild im zu überprüfenden Datensatz entstanden ist.
- **TimeEnd**: Ein Zeitstempel, wann das jüngste Bild im zu überprüfenden Datensatzes entstanden ist.
- **QADate**: Ein Zeitstempel, wann das Skript zur Qualitätssicherung ausgeführt wurde

Qualitätsmetrik für Objekterkennung

Für alle Objekterkennungsmodelle, also allen Instanzen des Retinanet Modells (siehe Kapitel 3.5) wird als Qualitätsmetrik die Mean Average Precision (mAP) verwendet. Für das Bestimmen der mAP muss eine weitere wichtige Metrik in der Objekterkennung vorgestellt werden, nämlich die Intersection over

Union (IoU). Sie beschreibt den Bereich, in dem sich die vorhergesagte Bounding Box und die tatsächliche (Ground Truth) Bounding Box überschneiden. Je höher der IoU, desto genauer kann ein Modell ein Objekt in einem Bild lokalisieren (Rosebrock, 2016).

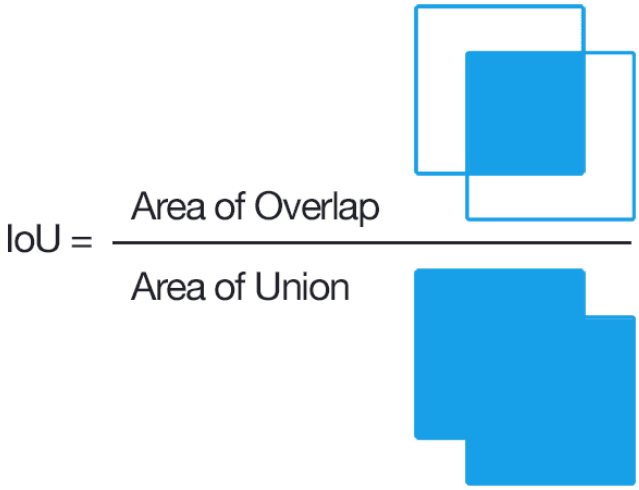

Abbildung 12: Definition der Intersection over Union (Rosebrock, 2016).

Die Berechnung der Mean Average Precision erfolgt in mehreren Schritten:

1. Alle vorhergesagten Bounding Boxen werden mit den tatsächlichen Bounding Boxen verglichen und der IoU Wert für alle Kombinationen berechnet.
2. Zuordnung der Bounding Boxen und Klasse basierend auf dem höchsten IoU Wert zu den tatsächlichen Bounding Boxen und Klassen. Typischerweise werden IoU Werte von 0.5 oder 0.75 als Kriterium für eine Übereinstimmung (Match) angenommen.
3. Berechnung der Precision und Recall für jede Objektklasse. Dabei gilt, jedes korrekt erkannte Objekt ist ein True Positive. Ein Objekt, das nicht erkannt wurde, wird als False Negative eingestuft. Wenn ein Objekt vom Modell erkannt wird, das in Wahrheit nicht an dieser Position existiert, wird es als False Positive eingestuft.
4. Berechnung Average Precision (AP) für jede Objektklasse für den jeweiligen IoU Wert (0.5). Dafür wird die sogenannte Precision-Recall Curve erstellt, welche aus den sortierten Precision und Recall Werten besteht. Die AP wird als die Fläche unter der Precision-Recall Curve berechnet.
5. Berechnung Mean Average Precision anhand aller vorhandenen Average Precision Werte für jede Objektklasse.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

Sämtliche Formeln und Beschreibungen sind in der Literatur bekannt und wurden übernommen von Shah, (2022).

Für die Qualitätsüberprüfung sämtlicher Modelle zur Objekterkennung werden folgende Informationen in die Deltatabelle geschrieben:

- **ComponentId**: Der ID-Code, der jede Kamera eindeutig zuordnet.
- **Mean Average Precision**: Der mAP-Score, berechnet aus dem aktuellen Datensatz.
- **ImageCount**: Die Anzahl Bilder im zu überprüfenden Datensatz.
- **TimeStart**: Ein Zeitstempel, wann das älteste Bild im zu überprüfenden Datensatz entstanden ist.
- **TimeEnd**: Ein Zeitstempel, wann das jüngste Bild im zu überprüfenden Datensatzes entstanden ist.
- **QADate**: Ein Zeitstempel, wann das Skript zur Qualitätssicherung ausgeführt wurde

3.7 Zusammenfassung und Übersicht

Dieses Unterkapitel fasst die in diesem Projekt erarbeitete Plattform zusammen und beschreibt nochmals die einzelnen Subkomponenten.

Die Bilder werden vom Raspberry Pi mit einem Python Skript aufgenommen und via Kafka in die Bronze Deltatabelle gestreamt. Die Bronzetabelle ist Teil des Speichersystems auf Databricks. Für das Erzeugen und Trainieren eines neuen Modells nutzt man je nach Aufgabenstellung eine der beiden bereitgestellten Modellklassen, dem VGG16 Klassifikator oder dem Retinanet Objekt Detektor. Über ein Funktionsfile wird eine definierte Anzahl an randomisierten Trainingsbildern aus der Bronze Tabelle auf einem internen Zwischenspeicher (Blob) abgelegt und die URLs nach Labelbox geschickt. Auf Labelbox wird je nach Situation ein neues Projekt angelegt, oder ein bestehendes weiter genutzt, um die neuen Bilder zu labeln. Anschliessend wird auf Databricks eine Instanz von einem der beiden Modellklassen erzeugt und mit den Trainingsbildern und den aus Labelbox importierten Label trainiert. Die Befehle zum Laden der Bilder und Annotationen aus Labelbox erfolgen alle durch Funktionen, welche in einem Python File geschrieben sind. Das Modell wird direkt nach dem Training in die MLflow Registry gespeichert. Existiert bereits ein gleichnamiges Modell mit dieser Aufgabe, wird eine neue Version erstellt. Ein Databricks Job verarbeitet die Bilder aus der Bronze Deltatabelle und lädt automatisch die aktuelle Modellversion aus der MLflow Registry. Die Modellvorhersagen werden in die Silver Deltatabelle geschrieben. Ein weiterer Databricks Job extrahiert in regelmässigen Abständen eine fest definierte Anzahl Samples aus der Silver Deltatabelle und schickt die Daten inklusive Annotation ins gleiche Labelbox Projekt, wie zum initialen labeln der Rohdaten genutzt wurde. Nach manueller Überprüfung werden die Qualitätsmetriken im letzten gültigen Zeitraum berechnet und als Einträge in die Deltatabelle Quality abgelegt. Die Daten zur Qualitätsüberwachung als auch die aggregierten Daten aus der Silver Deltatabelle werden auf einem Databricks Dashboard dargestellt. Die Erstellung dieses Systems umfasst massgebliche Arbeiten in Datengewinnung und Aufbereitung, Modellerstellung und Skripting. Es wurde der zeitintensive Ansatz von Test and Fail verwendet, was jedoch kontinuierlich zur Verbesserung der Plattform und der Modellgüte beigetragen hat.
Die folgende Abbildung 13 beschreibt den Ablauf, beschrieben in diesem Unterkapitel noch einmal grafisch.

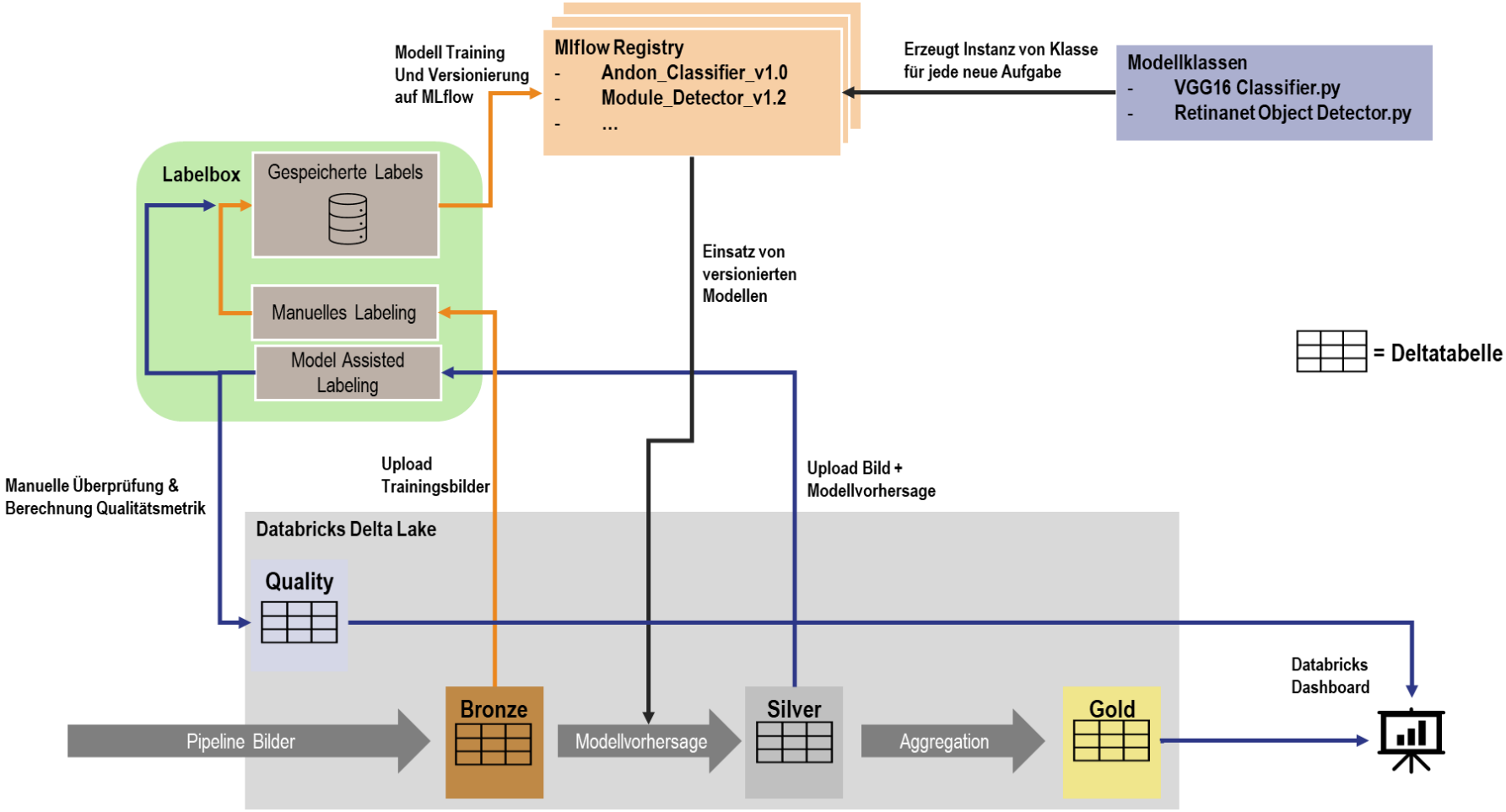

Abbildung 13: Übersicht Gesamtsystem.

4 Resultate

Dieses Kapitel beschreibt die durchgeführten Experimente und Resultate im Rahmen des 'Lean Production Monitoring' Projektes. Das korrekte Erkennen von Ampelsignalen ist die Hauptanforderung in Bezug auf diese Arbeit. Für diese Aufgabe wurde eine Instanz des VGG16 Klassifikators verwendet, beschrieben nachfolgend in Kapitel 4.1 bis 4.1.4. Dieses Modell ist Bestandteil der Zielsetzung aus Punkt IV (Siehe Kapitel 1.2).

Für das Testen des Retinanet Objekterkennungsmodell wurde ein Testlauf an einem Zwischenlager für manuelle Prozesse mit Sensormodulen durchgeführt, was in Kapitel 4.2 bis 4.2.3 beschrieben wird. Dieses Modell ist Bestandteil der Zielsetzung aus Punkt V (Siehe Kapitel 1.2).

4.1 Testlauf VGG16 Modell zur Klassifikation von Ampelsignalen

Für den ersten Testlauf wurden 4 Ampelsysteme mit unterschiedlichem Typ mit Endoskop Kameras ausgestattet, darunter befinden sich zwei der Ampelsysteme an Maschinen in einem Reinraum für Siliziumwafer Tests. Die Anbringung der Kameras erfolgte, wie in Kapitel 3.1 beschrieben, mit Plastikschienen und Endoskop Kameras. Die Kameras sammelten 14 Tage lang Testdaten und ein erster Satz von 300 randomisierten Bildern wurde auf Labelbox hochgeladen und mit einer durchschnittlichen Zeit von 4 Sekunden pro Bild gelabelt. Im Verlauf der nächsten zwei Wochen wurden weitere 600 Bilder gelabelt. Die folgende Tabelle beschreibt die Klassen, welche jeweils für eine der möglichen Farbkombinationen steht. Die Kombinationen Grün/Rot und Gelb/Rot sind nicht vorhanden, da diese Zustände auf den vorhandenen Maschinen nicht möglich sind.

Ampelstatus	Klasse/Label
Aus	Off
Grün	Green
Gelb	Amber
Rot	Red
Grün/Gelb	Green/Amber
Grün/Gelb/Rot	Green/Amber/Red

Tabelle 3: Klassenlabel für Ampelzustände

Die folgende Abbildung zeigt eine Auswahl an Bildern, die von den Endoskop Kameras im Inneren der Plastikschiene während des Testlaufs aufgezeichnet wurden. Die Ampelsysteme unterscheiden sich teilweise deutlich voneinander was Lichtintensität und Farbspektrum angehen.

Abbildung 14: Aufnahmen von den Endoskop Kameras innerhalb der Plastikschiene.

4.1.1 Training und Resultate

Für die Evaluierung des Modells wurde zuerst ein Testdatensatz bestehend aus 344 Bildern aller Klassenlabel generiert. Dieser wird für die Evaluierung verschiedener Modelle verwendet und soll Auskunft geben, wie gut das Modell in der Praxis abschneidet. Der Datensatz wurde mit der der Bedingung generiert, dass maximal 33% aller Bilder aus einer Klasse für die Generierung verwendet werden dürfen, da die übrigen Bilder sonst nicht für einen guten Trainings- und Validierungsdatensatz ausreichen. Die folgende Abbildung zeigt die Verteilung der Klassen im Testdatensatz. Aufgrund der geringen Anzahl an Trainingsbildern für die Klassen Grün/Gelb und Grün/Gelb/Rot konnte keine gleichmässige Verteilung der Klassen erreicht werden.

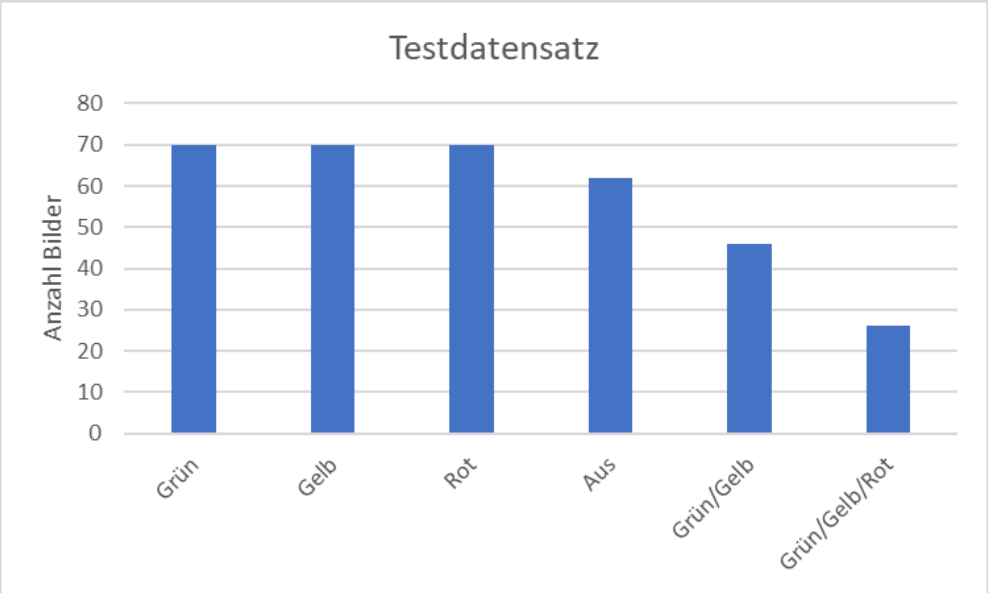

Abbildung 15: Klassenverteilung im Testdatensatz für Ampelsignale.

Das Modelltraining erfolgte über mehrere Trainingsläufe mit variierender Anzahl Trainings- und Validierungsbilder. Nebst den Loss Werten aus dem Training wurde nach jedem Trainingslauf das Modell mit dem Testdatensatz evaluiert. Dafür wurden die Metriken Precision, Accuracy, Recall und F1-Score pro Klasse berechnet, sowie Informationen über die Confidence Score des Modells, also wie sicher sich das Modell in seiner Vorhersage ist, evaluiert. Des Weiteren wurde das Modell jeweils einmal mit den vortrainierten Imagenet Gewichten für die Convolutional Layers und einmal mit zufälligen Gewichten trainiert. Ziel dieses Tests war es, die möglichen Vorteile des Imagenet Modells in Bezug auf benötigte Anzahl Trainingsbilder zu untersuchen.

Verwendete Modellparameter

Für das Modelltraining wurden folgende Modellparameter verwendet:

- Modell: VGG16 (Instanz der VGG16.py Modellklasse, beschrieben in Kapitel 3.4)
- Learning Rate: 0.00086 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.1.2)
- Anzahl Fully Connected Layers: 1 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.1.2)
    - Dropout Layer mit 40%, um Overfitting entgegenzuwirken.
- Anzahl Neuronen: 250 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.1.2)
- Loss Funktion: Categorical Cross Entropy
    - Die Categorical Cross Entropy Loss Funktion wird häufig zur Klassifikation von multiplen Objekten eingesetzt. Die Formel zur Berechnung lautet:

$$CE = -\sum_{i=1}^{i=n} Y_i log(p_i)$$

      Wobei Y das wahre Label und p die vorhergesagte Wahrscheinlichkeit für die Klasse i ist (Nayak, 2022).
- Optimizer: Adam
- Batch Size: 32
- Anzahl Trainingsepochen: 50

Zusätzlich zu jedem Trainingsdurchlauf wurden die Metriken Precision, Recall, Accuracy und F-Score auf dem Testdatensatz pro Klasse durchgeführt.

Resultate

Nebst Qualitätsmetriken wurden auch die sogenannten Probability Scores, also wie hoch das Modell die Wahrscheinlichkeit für eine Klasse einschätzt, aufgezeichnet. Ein hoher Probability Score bedeutet nicht, dass das Modell richtig liegt, ist aber ein vorsichtiges Indiz, wie gut das Modell trainiert ist. Die folgende Abbildung 16 und Abbildung 17 zeigen die F1-Scores aus den beiden Testläufen mit vortrainierten Imagenet Gewichten und randomisierten Gewichten. Eine detaillierte Übersicht aller Scores inklusive aller generierten Plots und Tabellen ist im Anhang aufgeführt.

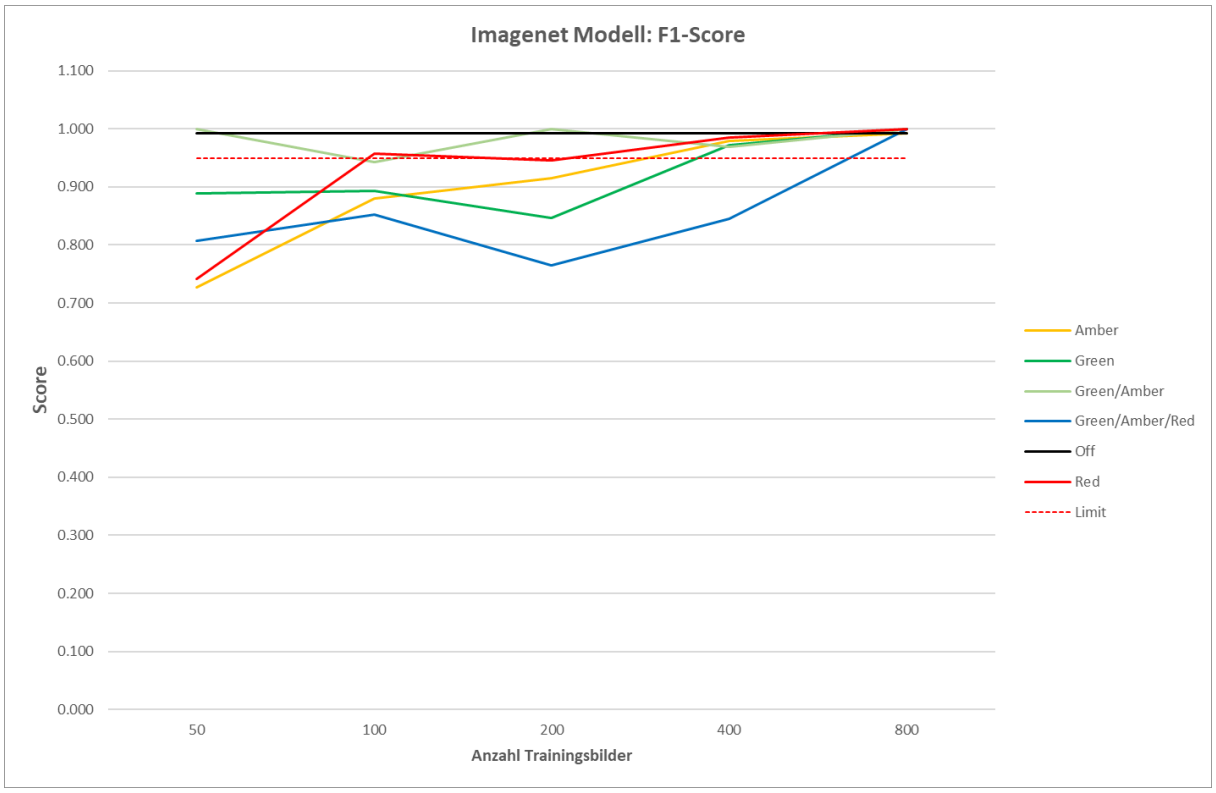

Abbildung 16: F1-Score des vortrainierten Imagenet Modells. Das Limit von 95% ist eine Anforderung an die Modellqualität.

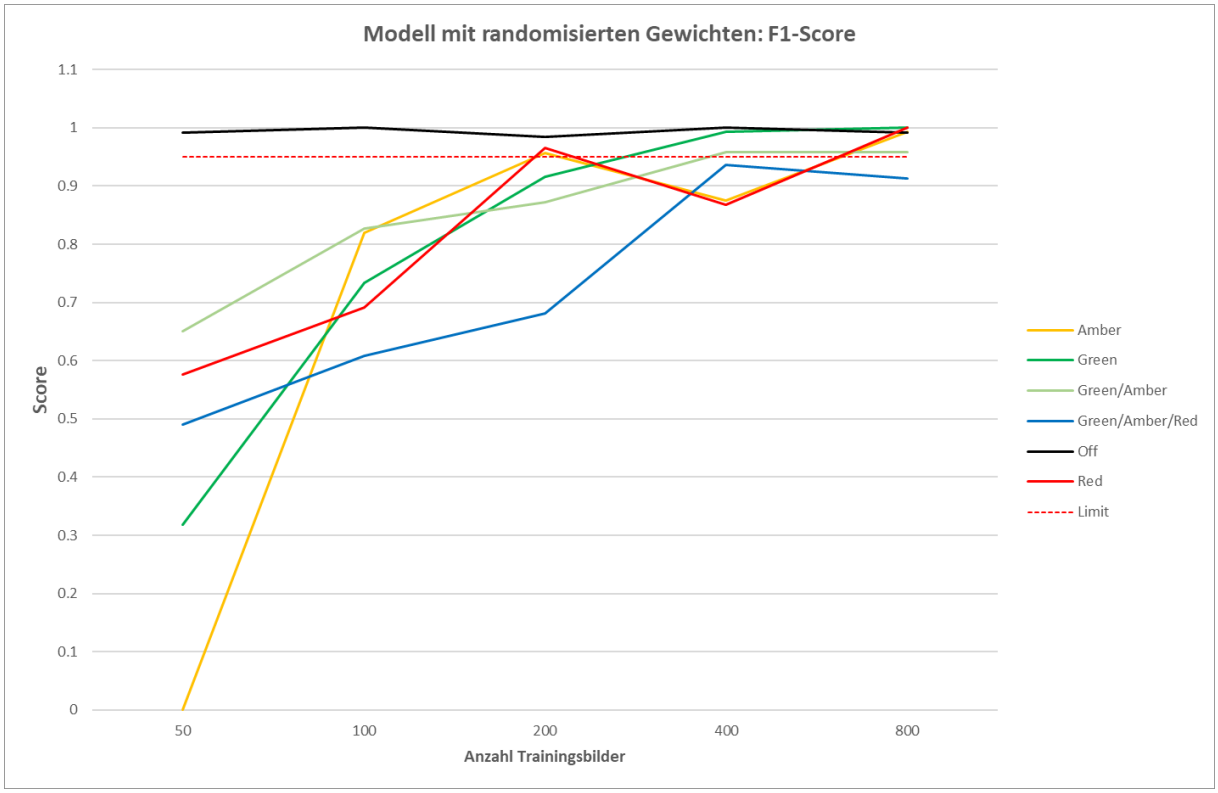

Abbildung 17: F1-Score des Modells mit randomisierten Gewichten. Das Limit von 95% ist eine Anforderung an die Modellqualität.

Der Vergleich der beiden Abbildungen zeigt, dass das vortrainierte Imagenet Modell besser abgeschnitten hat als das Modell mit randomisierten Gewichten. Das war aufgrund des grossen Imagenet Datensatzes, der für das Training dieser Gewichte verwendet wurde, keine grosse Überraschung. Aufgrund der besonderen Aufgabe, nämlich dem Klassifizieren von vielfarbigen Lichtpunkten in einer dunklen Plastikschiene, galt es sicherzustellen, dass ein untrainiertes Modell nicht doch besser geeignet wäre. Um diese Effekte zu erzielen, müssten jedoch mehr Trainingsepochen und idealerweise viel mehr Trainingsbilder verwendet werden, womit das Ziel eines schnell einsetzbaren Modells mit verhältnismässig wenig Aufwand verfehlt wäre. Das Imagenet Modell zeigt ab bereits 100 Trainingsbildern eine gute Performanz, verliert bei 200 Bildern jedoch in den Farben Grün und Grün/Gelb/Rot deutlich an Genauigkeit. Dies könnte damit zusammenhängen, dass mit mehr Trainingsbildern auch mehr Komplexität dazukommt. Ab 800 Bildern liegt die Modellgenauigkeit für alle Farben bei über 95% F-Score, womit die Modellperformanz als bestanden angesehen wird. Bei einer durchschnittlichen Zeit von 4 Sekunden für das Labeln eines Bildes kommt man bei 800 Bildern auf weniger als eine Stunde für den gesamten Datensatz, was für eine solche Modellperformanz verhältnismässig wenig Aufwand ist.

Die folgende Abbildung 18 zeigt die Loss Werte aus dem Modelltraining. Pre-Trained steht für das Imagenet Modell und Untrained bezeichnet das Modell mit randomisierten Gewichten.

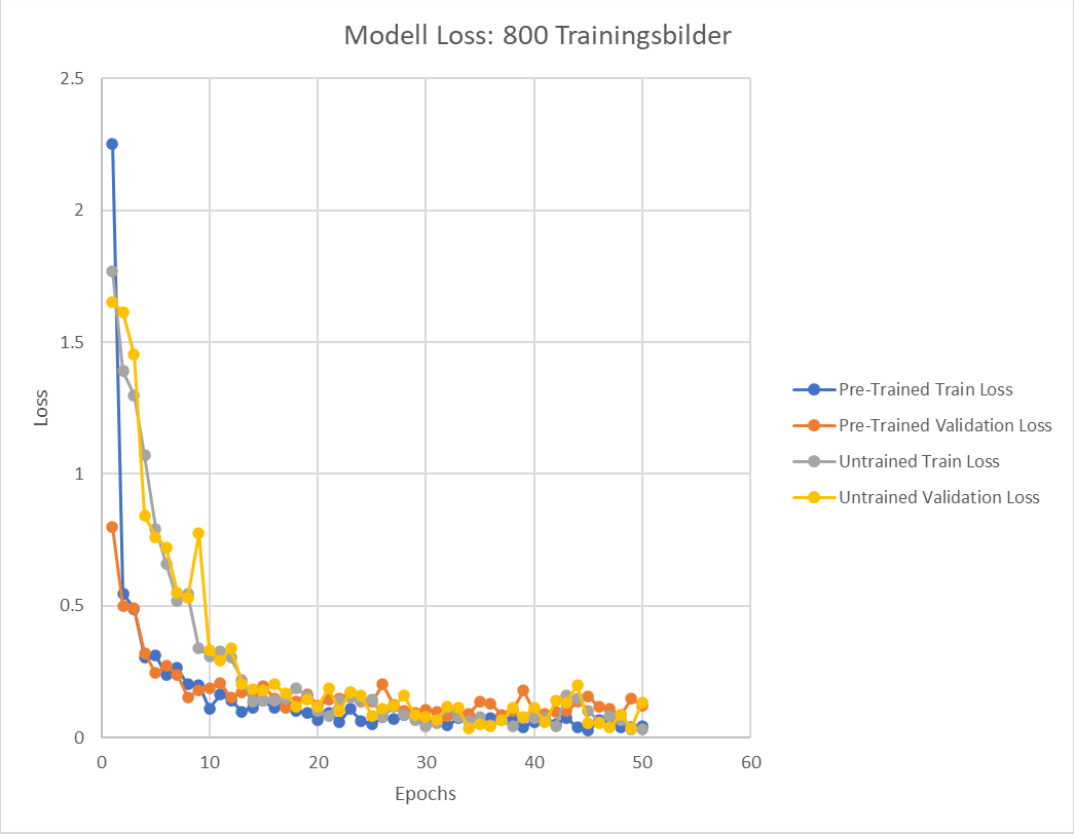

Abbildung 18: Modell Loss mit 800 Trainingsbildern.

Beim Betrachten der Loss-Plots fällt auf, dass die Kurven des Validation Loss teilweise tiefer ausfallen als der Training Loss, insbesondere bei Trainingsläufen mit weniger Bildern. Auf den ersten Blick wirkt dies nicht intuitiv, lässt sich jedoch damit erklären, dass das implementierte VGG16 Model Dropout Layers zwischen den Fully Connected Layers hat. Diese verhindern während des Trainings, dass sich das Modell zu fest den Trainingsdaten anpasst (Overfitting), indem es abwechselnd unterschiedliche Gewichte einfriert (Soleymani, 2022).

Eine Betrachtung der Konfusionsmatrizen in der folgenden Abbildung 19 und Abbildung 20 gibt mehr Aufschluss über die Modellgüte.

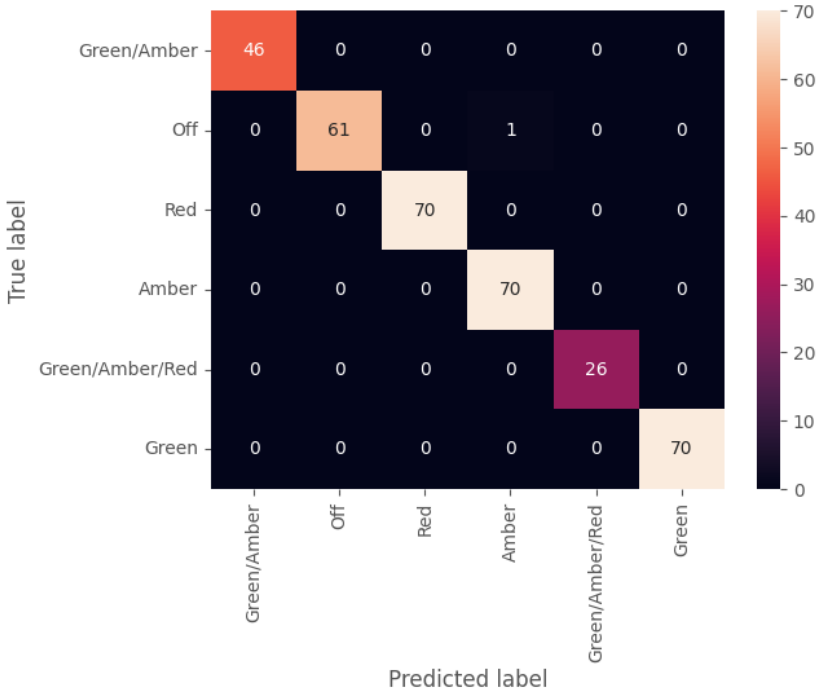

Abbildung 19: Konfusionsmatrix für Modellvorhersagen (Imagenet Modell) auf Testdatensatz.

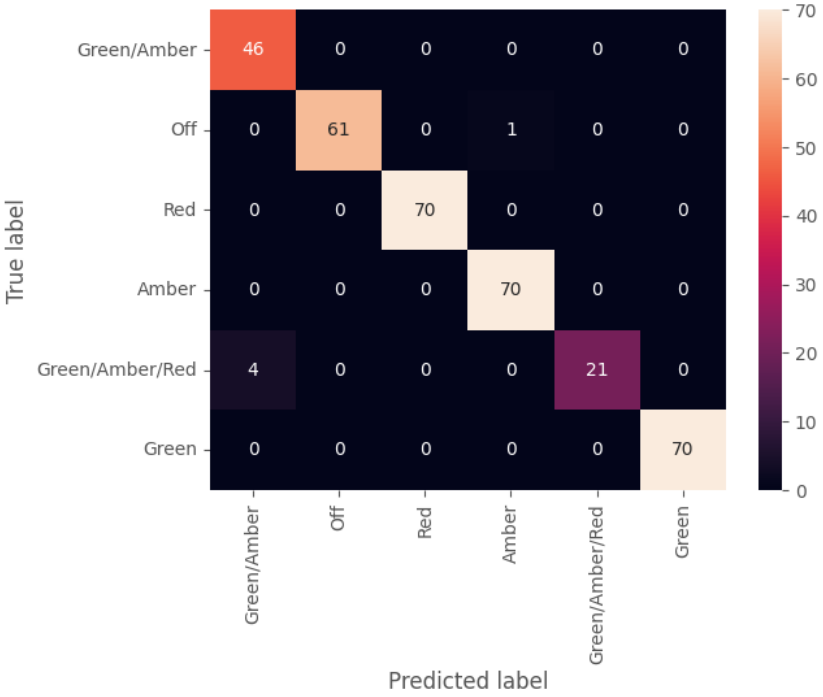

Abbildung 20: Konfusionsmatrix für Modellvorhersagen (Modell mit randomisierten Gewichten) auf Testdatensatz.

Die Konfusionsmatrizen, dargestellt in Abbildung 19 und Abbildung 20, geben Auskunft darüber, welche Label das Modell gut vorhersagt und welche nicht. Auf der vertikalen Achse befinden sich die wahren Label (True Label), die im Labeling Prozess entstanden sind und auf der horizontalen Achse die

vorhergesagten Label (Predicted Label), die vom Modell stammen. Das Modell gilt als perfekt, wenn alle vorhergesagten Label mit den wahren Labeln übereinstimmen. Interessant ist, dass das Imagenet Modell lediglich ein Bild fälschlicherweise als Orange (Amber) klassifiziert hat, es aber in Wahrheit kein Licht angezeigt hat (Off). Dies könnte daran liegen, dass einige Bilder einen fliessenden Übergang zwischen einer Farbe und dem ausgeschalteten Status haben. Teilweise ist ein Restglühen zu beobachten, wie in Abbildung 21 ersichtlich ist.

Abbildung 21: Ausgeschaltete Ampel mit schwachem orangefarbenem Lichteinfall. Das Bild entstand kurz nachdem die orange Ampel ausgeschaltet wurde.

Schwachstellen des Modells

Eine Schwachstelle des Modells ist, dass es nicht erkennen kann, ob die Kamera auf die Ampel zeigt, oder diese verrutscht ist. Da das Modell nur 6 mögliche Klassen hat, wird es auch im Fall einer abgefallenen Kamera weiterhin Klassen vorhersagen, auch wenn gar keine Ampel vorhanden ist. Diese Schwäche kann durch das Qualitätsmonitoring abgefedert werden, da eine Person Stichproben der Bilder regelmässig überprüft und solch ein Problem schnell erkennen würde. Es ist zudem möglich, dass so eine Situation einen Abfall der Probability Scores mit sich ziehen würde. Aufgrund eines unteren Limits von 85% für den Probability Score würden diese Bilder im Dashboard teilweise als ungültige Daten dargestellt werden. Dies ist aber nicht in jeder Situation garantiert. Eine Kamera, deren Linse verdeckt ist, wird weiterhin mit grosser Zuversicht den Ampelstatus als ausgeschaltet interpretieren.

Eine weitere Schwachstelle ist, dass eine hohe Abtastrate zu einer grossen Datenmenge führt. Dies ist technisch zwar kein Problem, führt aber zu höheren Kosten auf Databricks. Für die Pilotphase wurde daher die Bilderrate auf 60 Bilder pro Stunde gesetzt. Dies ermöglicht immer noch eine gute Gesamtübersicht, kann jedoch schnelle Wechsel von Ampelsignalen nicht mehr detektieren.

4.1.2 Hyperparameter Tuning

Die Hyperparameter für das Hyperopt Skript sind in Kapitel 3.4 beschrieben und werden automatisch durch ein Skript ermittelt. Ein Test mit 50 Durchläufen, jeweils 30 Trainingsepochen und 400 Trainingsbildern wurde durchgeführt, um die optimalen Hyperparameter für die in Kapitel 4.1.1 beschriebenen Trainingsläufen bereitzustellen. Hyperopt benötigt einen sogenannten Search Space für jeden zu testenden Hyperparameter. Über mehrere Iterationen hinweg testet das Skript unterschiedliche Hyperparameter Einstellungen und ermittelt basierend auf einer definierten Evaluationsmetrik, die optimalen Hyperparameter. Als Evaluationsmetrik für diese Testläufe wird der Validation Loss genutzt.

Tabelle 4: Hyperparameter mit Search Space für das Hyperopt Skript.

Hyperparameter	**Suchbereich**
Learning Rate	0.000001 – 0.01
Anzahl Neuronen pro Layer	50 – 500 (Schrittgrösse 50)
Anzahl Fully Connected Layers	1 – 4 (Schrittgrösse 1)

Die Resultate des Hyperopt Testlaufs sind in der nachfolgenden Tabelle 5 zusammengefasst. Die Tabelle listet die besten 15 Resultate absteigend geordnet nach Validation Loss auf.

Tabelle 5: Top 15 Resultate Hyperparameter Tuning für Ampelzustände.

Learning Rate	Anzahl Fully Connected Layers	Anzahl Neuronen pro Layer	Validation Loss
0.00086	1	250	0.0125
0.00093	1	250	0.0188
0.00640	1	250	0.0206
0.00010	1	250	0.0325
0.00173	1	450	0.0450
0.00188	1	450	0.0529
0.00008	1	450	0.0661
0.00490	1	400	0.0718
0.00610	1	350	0.0753
0.00024	3	500	0.0782
0.00830	1	250	0.0875
0.00018	3	400	0.1017
0.00005	1	400	0.1159
0.00063	2	300	0.1296
0.00904	1	200	0.1386

Die Resultate zeigen ein deutliches Bild in Bezug auf Anzahl Layer und Neuronen, da die besten 4 Durchläufe alle die gleichen Werte aufweisen. Die beiden besten Durchläufe weisen auch in Bezug auf Learning Rate ähnliche Werte auf. Die Testläufe haben es ermöglicht, mit wenig Aufwand viele Möglichkeiten auszutesten und die optimalen Hyperparameter zu ermitteln. Aufgrund der guten Performanz wurden weitere Hyperparameter, wie der standardmässig genutzte Optimizer Adam, nicht in die Hyperopt Evaluation miteinbezogen.

4.1.3 Qualitätsmonitoring

Wie bereits in Kapitel 3.6 beschrieben, wird alle 3-4 Tage eine Stichprobe von 20 Bildern pro Kamera erhoben, manuell überprüft und anschliessend der F1-Score, Recall, Precision und Accuracy auf den Testbildern berechnet. Das Resultat wird in die Deltatabelle 'Quality' geschrieben. Die folgenden beiden Abbildungen zeigen die Scores aus der Qualitätsprüfung von zwei Kameras. Die Scores sind hierbei für eine bessere Übersicht jeweils nach Kamera (ComponentId) gegliedert. Es fällt auf, dass ein Zeitraum von 10 Tagen keine Qualitätsprüfung aufweist, was mit einem Fehler des Upload-Jobs verbunden war.

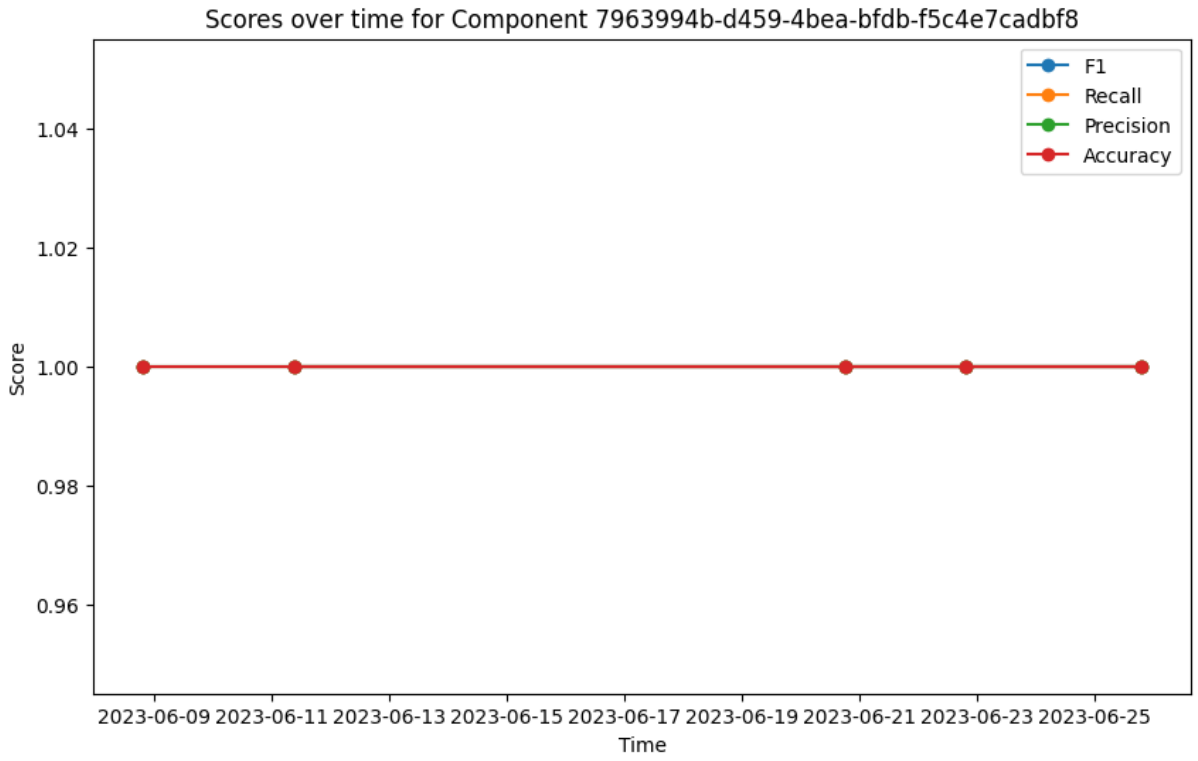

Abbildung 22: Qualitätsübersicht über die vergangenen 3 Wochen für eine überwachte Ampel.

Das Qualitätsmonitoring erfüllt seinen Zweck gut, da es die aktiven Kameras mit einem verhältnismässig kleinen Aufwand überwacht und auch zum Erkennen von Defekten, wie beispielsweise die nachfolgende Abbildung 23 zeigt, wo die Plastikschiene sich teilweise vom Ampelgehäuse gelöst hat. Die 4 Kameras, die zu Beginn des Projektes installiert wurden, weisen alle einen F1-Score von über 95% auf.

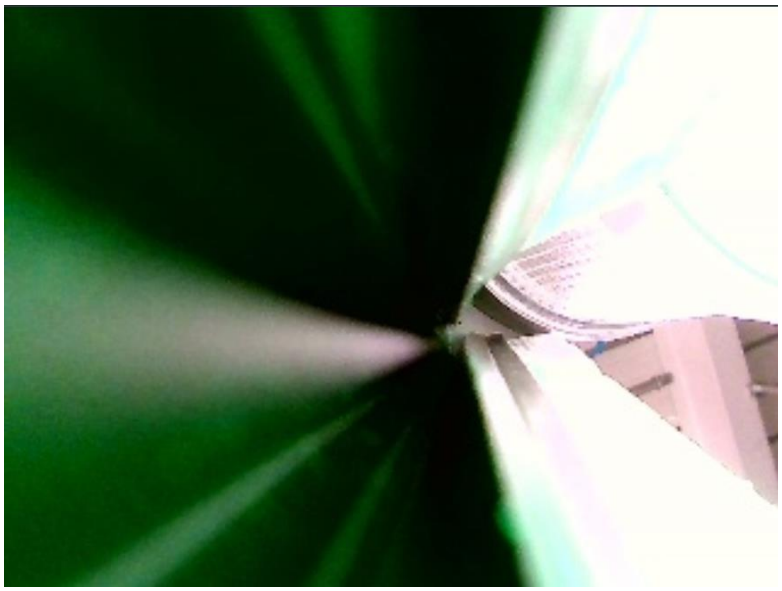

Abbildung 23: Abgelöste Plastikschiene vom Ampelsystem.

4.1.4 Dashboard Darstellung

Die aggregierten Daten aus der Gold Deltatabelle (siehe Kapitel 3.2) werden auf Databricks in Form einer Dashboard Darstellung visualisiert. Diese grafische Oberfläche erlaubt es einer Person, ohne Kenntnisse in Python oder SQL, die Daten schnell und einfach zu betrachten und die nötigen Schlüsse zu ziehen. Das Dashboard ist frei skalierbar und bietet die Möglichkeit, die geplotteten Daten herunterzuladen oder in einen spezifischen Zeitraum hineinzuzoomen. Aus Gründen des Datenschutzes und Firmengeheimnissen werden in dieser Arbeit nur ausgewählte Plots aus dem Dashboard verwendet. Eine beliebte Form der Darstellung ist die Aggregation aller Ampelzustände pro Tag oder auf Stundenbasis, ersichtlich in der folgenden Abbildung 24.

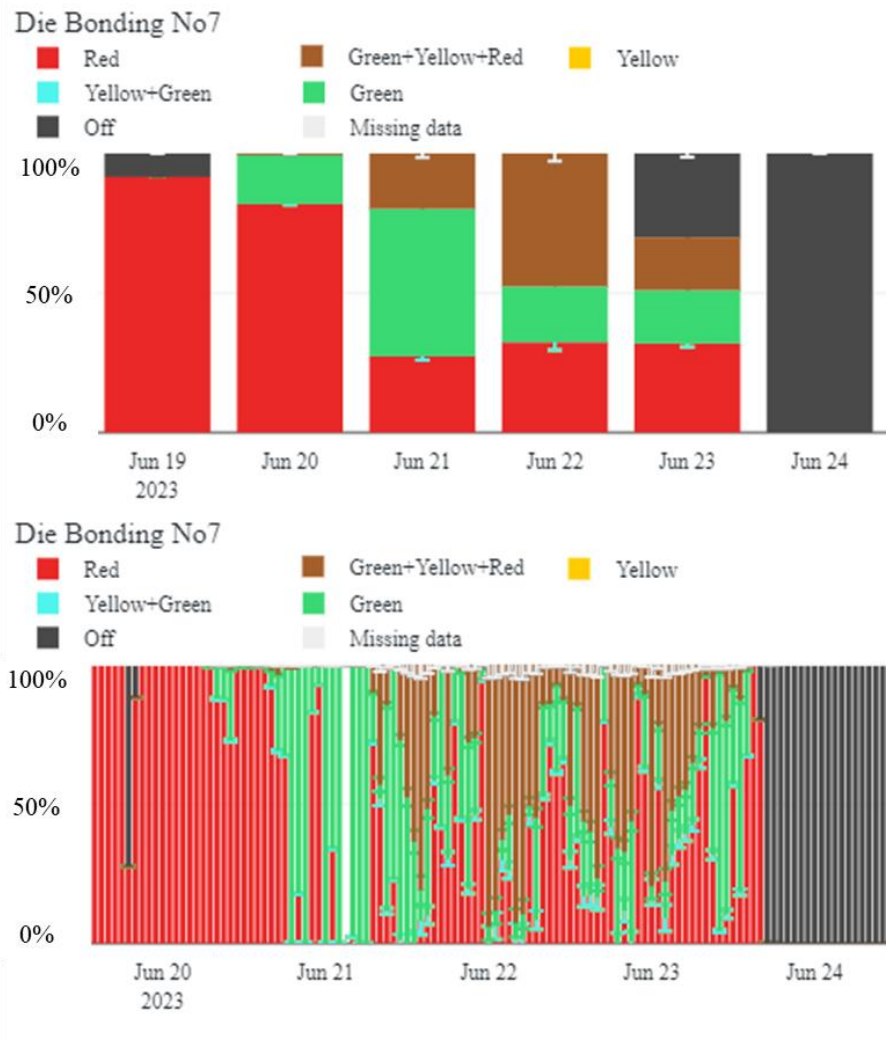

Abbildung 24: Tages und Stundendarstellung des Maschinenstatus anhand der Ampel an einer Die Bonding Maschine.

Diese Dashboard Plots werden in Kombination mit Informationen über Produktionszahlen, ermittelt mit dem Verfahren der Mustererkennung eines Vibrationssensors und, beschrieben in Kapitel 1, für die

Produktionsplaner dargestellt. Sie ermöglichen es, Informationen über den Status sämtlicher überwachten Produktionsmaschinen zu vermitteln. Die folgende Abbildung 25 zeigt den Maschinenstatus anhand des Ampelzustands und auf der unteren Bildhälfte den Produktionsausstoss der Maschine. Solche Darstellungen ermöglichen es, Einbrüche in der Produktionsleistung zu erkennen und zu analysieren.

Abbildung 25: Dashboard Darstellung mit Ampelstatus und berechnetem Produktionsausstoss.

4.2 Testlauf Retinanet Modell zur Objekterkennung

Eine Erweiterung, die das VGG16 Modell und dessen Möglichkeiten ergänzen soll, ist das Retinanet CNN, beschrieben in Kapitel 3.5. Die Aufgabenbereiche dieses Modells sollen hauptsächlich im Erkennen und Unterscheiden von unterschiedlichen Sensormodulen, welche teilweise auf der gleichen Produktionslinie verbaut werden, liegen. Eine solche Fähigkeit könnte dazu beitragen, den Materialfluss in einem manuellen Produktionsprozess zu überwachen, indem zum Beispiel Komponenten in einem Materiallager erkannt und gegebenenfalls auch gezählt werden können.

Um die Möglichkeiten dieses Modells besser zu erforschen, wurde ein Testversuch in einer experimentellen Produktionslinie gestartet, wo Sensormodule in einem manuellen Prozessschritt zusammengebaut und Komponenten angelötet werden. Die Sensormodule liegen in einem schwarzen Tablar mit einem dazugehörigen Laufzettel mit Produktspezifischen Details und Auftragsinformationen. Dieses wird in einem kleinen Materialzwischenlager abgelegt und die Sensormodule werden von den Produktionsmitarbeitern herausgenommen und verarbeitet. Die fertigen Teile werden nach Vollendung in einem weiteren Tablar abgelegt und zum nächsten Arbeitsschritt weitergeleitet.

Abbildung 26: Materialtablar mit Sensormodulen. Oben Links befindet sich ein Laufzettel mit Produktspezifischen Informationen.

Der Testlauf besteht darin, mit einer Kamera das Zwischenlager mit den Sensortablaren zu beobachten und Testbilder für das Modelltraining zu sammeln. Ziel ist es, die Sensormodule in einem Tablar zu erkennen und die korrekte Anzahl verbleibender Sensoren zu übermitteln. Da die Sensormodule eine weisse Schutzmembran auf der Oberseite haben, sind sie in einem schwarzen Tablar gut zu erkennen.

Während 10 Tagen wurden Testbilder des Zwischenlagers gesammelt. Das Labeling erfolgte ebenfalls durch Labelbox, allerdings wurden für diese Testbilder eine neue Variante des Labeling verwendet, nämlich die Bounding Box. Diese besteht, wie der Name bereits impliziert, aus einem Rechteck, welche um das zu erkennende Objekt gezogen wird. Sie beschreibt, wo sich ein Objekt im Bild befindet, und bildet zusammen mit einem Klassenlabel die Basis für alle Modelle zur Objekterkennung. Die Bounding Box Koordinaten werden zusammen mit den Testbildern sowie den Labeln an das Modell übergeben. Die folgende Abbildung 27 zeigt zwei Testbilder mit den dazugehörigen Bounding Box Annotationen.

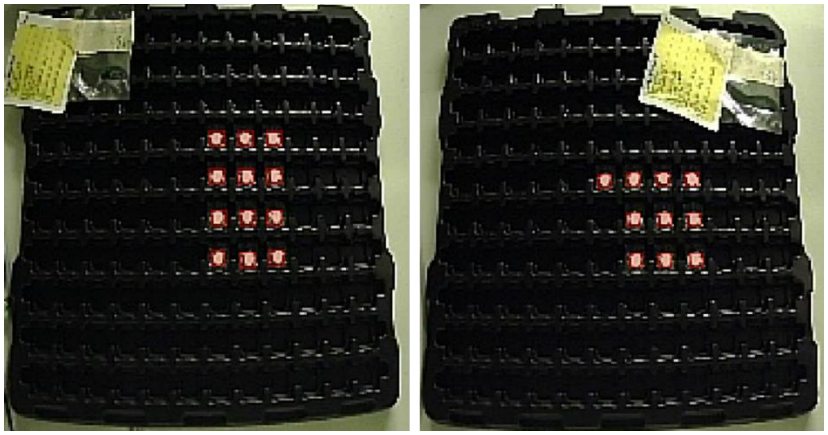

Abbildung 27: Bilder des Materialzwischenlagers mit eingezeichneten Bounding Box Annotationen (Rot).

4.2.1 Training und Resultate

Ein Testdatensatz von 80 Bildern wurde für die Evaluierung des Modells generiert. Da in diesem Arbeitsschritt die Hardware der Sensormodule immer gleichbleibt, ist nur eine Objektklasse nötig. Trotzdem ist der Labeling Aufwand bedeutend grösser als beim Labeln für Klassifikationen, da jedes zu identifizierende Objekt manuell mit einer Bounding Box markiert werden muss.

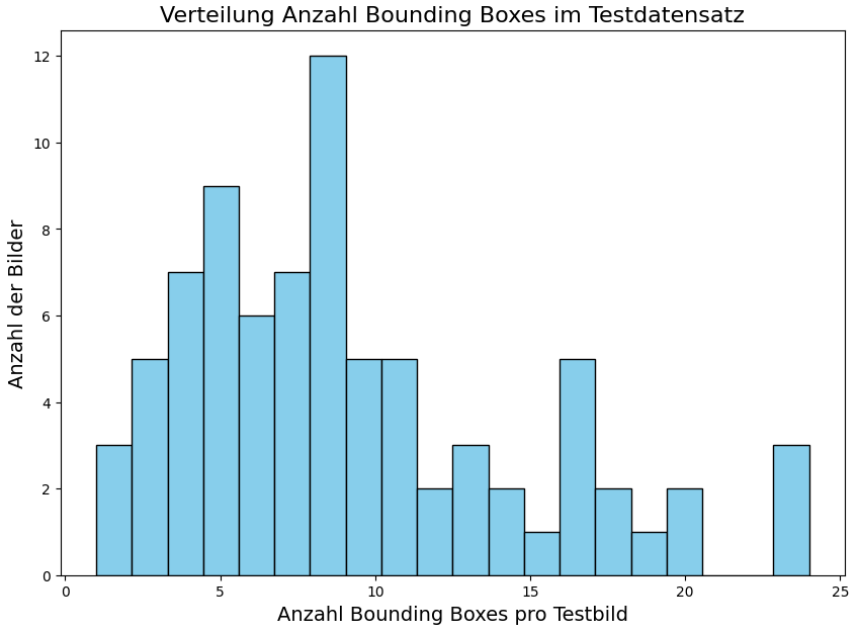

Abbildung 28: Anzahl Bounding Boxes (Anzahl Objekte) pro Bild im Testdatensatz (80 Bilder).

Über mehrere Trainingsläufe wurden anschliessend mit variierender Anzahl Trainings- und Validierungsbilder die Modellevaluation durchgeführt. Wie in Kapitel 3.6 beschrieben, wird als Qualitätsmetrik die Mean Average Precision für eine IoU von 50% gewählt. Das bedeutet, alle Objekte, deren vorhergesagte Bounding Box Fläche zu mindestens 50% innerhalb der korrekten (Ground Truth) Bounding Box Fläche liegen, gelten als korrekt identifiziertes Objekt. Zusätzlich werden alle Modellvorhersagen mit einer Genauigkeit von weniger als 30% ignoriert.

Folgende Definitionen gelten in Bezug auf die Begriffe True Positive, False Positive und False Negative.
True Positive: Die Bounding Box der Modellvorhersage überlappt eine wahre Bounding Box mindestens mit dem gesetzten IoU Wert von 0.5. In diesem Fall gilt die Vorhersage als korrekt.
False Positive: Die Bounding Box der Modellvorhersage findet keine Übereinstimmung mit einer wahren Bounding Box. Das bedeutet, das Modell hat ein Objekt erkannt, das in den Testdaten nicht markiert ist.
False Negative: Eine Bounding Box aus den Testdaten, die mit keiner, der vorhergesagten Bounding Boxes den gesetzten IoU Wert überschreitet. Das Objekt existiert gemäss Testdaten, wurde aber nicht vom Modell korrekt entdeckt.

Verwendete Modellparameter

Für das Modelltraining wurden folgende Modellparameter verwendet:

- Modell: Retinanet (Instanz der Retinanet_Object_Detector.py Modellklasse, beschrieben in Kapitel 3.5)
- Loss Funktion: Focal Loss (Erklärt in Kapitel 3.5 Hyperparameter Tuning)
    - Alpha: 0.305 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
    - Gamma: 2.78 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
- Learning Rate: 0.011 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
- L2 weight decay: 1.156e-05 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
- Box Loss Weight: 50 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
- Anchor Size: 2 (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
- Batch Size: Mindestgrösse des Validationsdatensatzes, maximal 32.
- Aspect Ratios: [0.3, 0.6, 1.2] (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
- Optimizer: Stochastic Gradient Descent (SGD) (Ermittelt durch Hyperparameter Tuning, Kapitel 4.2.2)
- Anzahl Trainingsepochen: 50

Resultate

Die folgende Abbildung 29 zeigt die Precision-Recall Kurven berechnet auf den 80 Testbildern aus insgesamt 6 Trainingsläufen mit 20, 40, 60, 150, 300 und 449 Trainings- und Validierungsbildern. Der Mean Average Precision Wert entspricht der Fläche unter der Kurve. Die rot gestrichelten Linien beschreiben die Form eines perfekten Modells. Für die Berechnung wurde ein Grenzwert von 30% auf die vorhergesagte Wahrscheinlichkeit angewandt. Alle Vorhersagen, die eine niedrigere Wahrscheinlichkeit haben, wurden nicht in die Berechnung einbezogen.

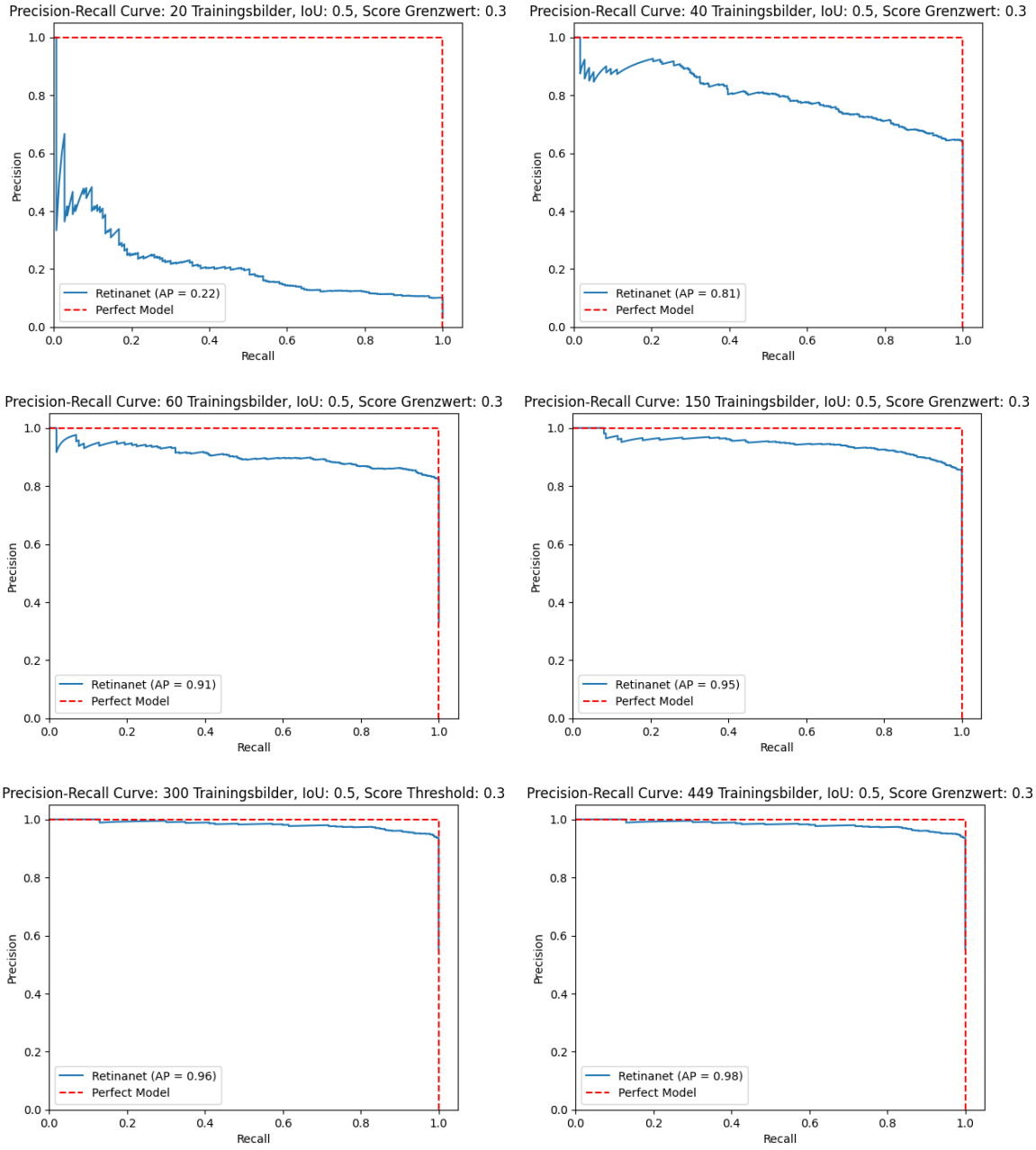

Abbildung 29: Precision-Recall Curve auf den Testdatensatz mit unterschiedlicher Anzahl Trainingsbildern. Als Kriterium für ein Match gilt ein IoU Wert von 0.5.

Die Plots zeigen, dass das Modell nach bereits 60 Trainingsbildern einen hohen mAP Score von 0.91 auf den Testdatensatz von 80 Bildern erzielt. Hierbei ist nochmals zu erwähnen, dass die Auswertung nur Vorhersagen mit einer Wahrscheinlichkeit von über 30% einbezog, sowie mit einem IoU Grenzwert von 0.5 erfolgte. Die folgende Abbildung 30 zeigt die mAP Scores zusammengefasst auf einem Plot gegenüber der Anzahl Trainingsbilder.

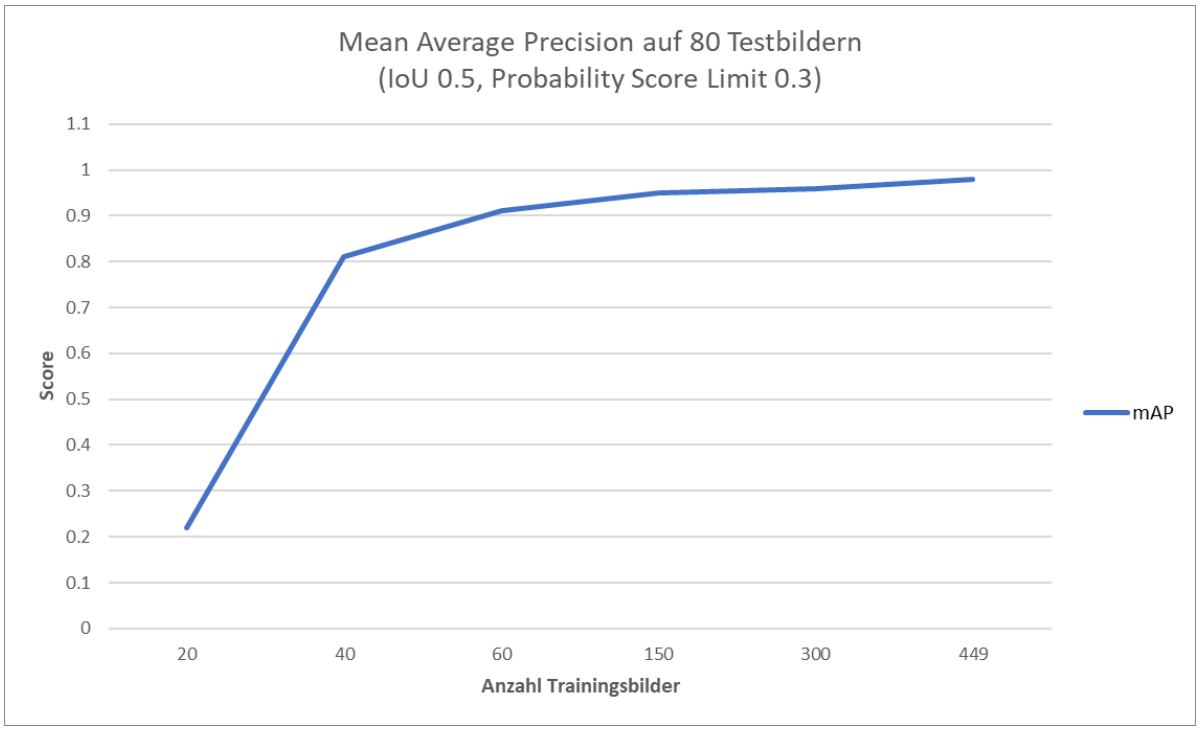

Abbildung 30: Mean Average Precision Score gegenüber Anzahl Trainingsbilder für das Modelltraining.

Da der mAP keine völlige Transparenz über die Modellgüte gibt, wird in der nachfolgenden Abbildung 31 die durchschnittliche Anzahl False Negatives und False Positives pro Testbild geplottet. Diese geben weiteren Aufschluss darüber, wie präzise das Modell Objekte erkennt und wie viele fehlerhafte Vorhersagen dieses macht. Es zeigt sich, dass bei 20 Trainingsbildern das Modell im Durchschnitt fast 8 Objekte pro Bild nicht erkennt und 16 Objekte im Bild erkennt, die nicht vorhanden sind. Bei einer Durchschnittlichen Anzahl von 9 Objekten pro Bild ist die Modellgüte noch zu ungenau. Bei 40 Trainingsbildern sinken insbesondere die False Positives stark ab auf ca. 3 Objekte pro Testbild. Die Modellgüte nimmt im Laufe der steigenden Trainingsbildern stets zu und erreicht mit 449 Trainingsbildern eine gute Leistung von weniger als 1 False Positive und False Negative pro Testbild.

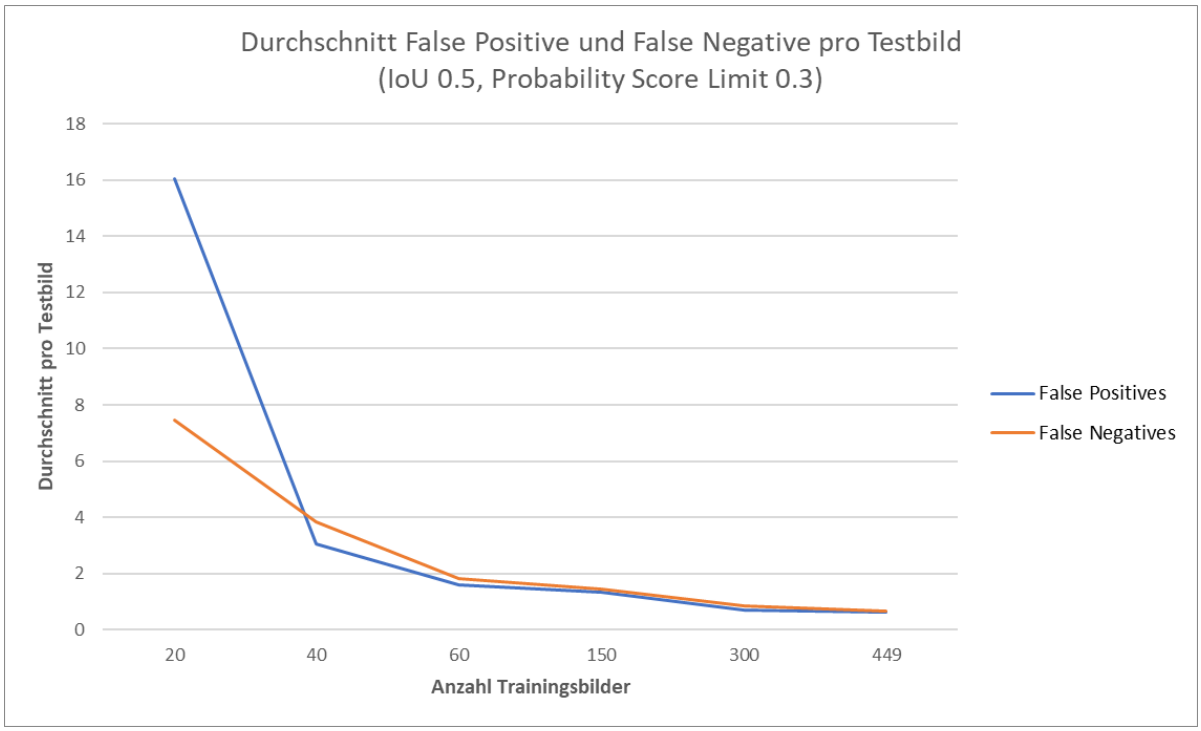

Abbildung 31: Die Durchschnittliche Anzahl False Positives und False Negatives pro Testbild

Die nachfolgenden Abbildungen Abbildung 32 bis Abbildung 35 beschreiben den Anstieg der Modellqualität insbesondere zwischen 20 und 60 Trainingsbildern deutlich.

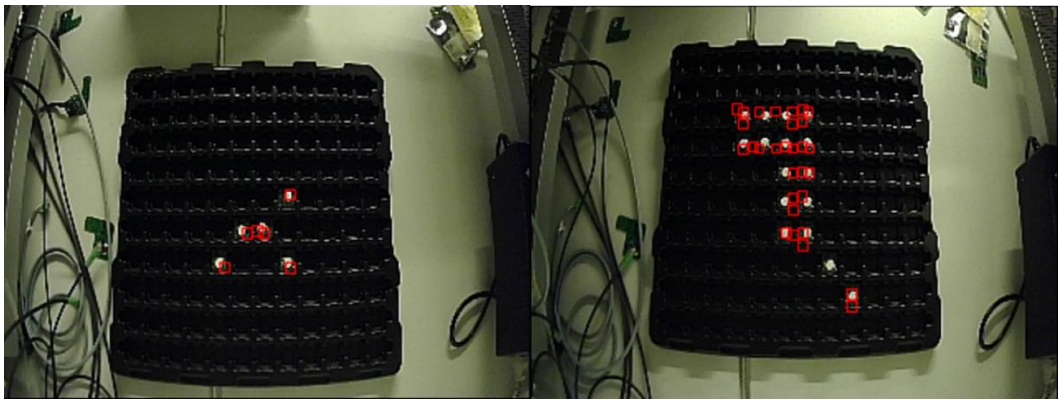

Abbildung 32: Modellvorhersage auf zwei Testbilder mit 20 Trainingsbildern.

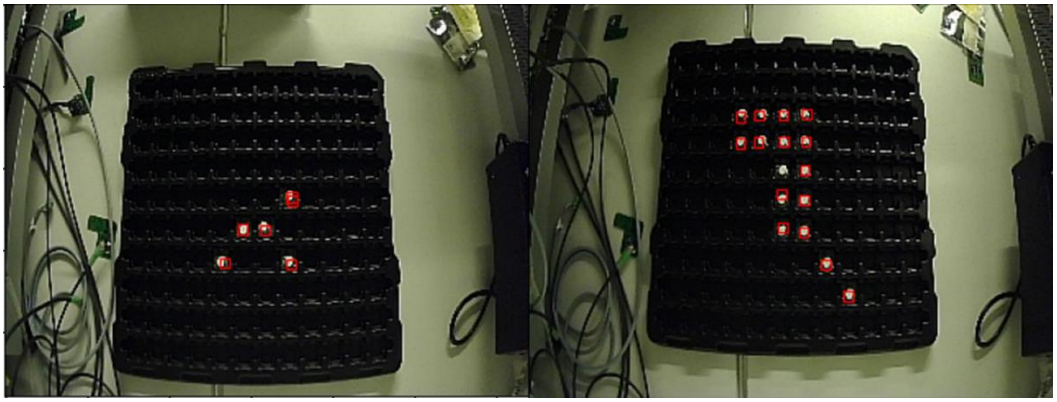

Abbildung 33: Modellvorhersage auf zwei Testbilder mit 40 Trainingsbildern.

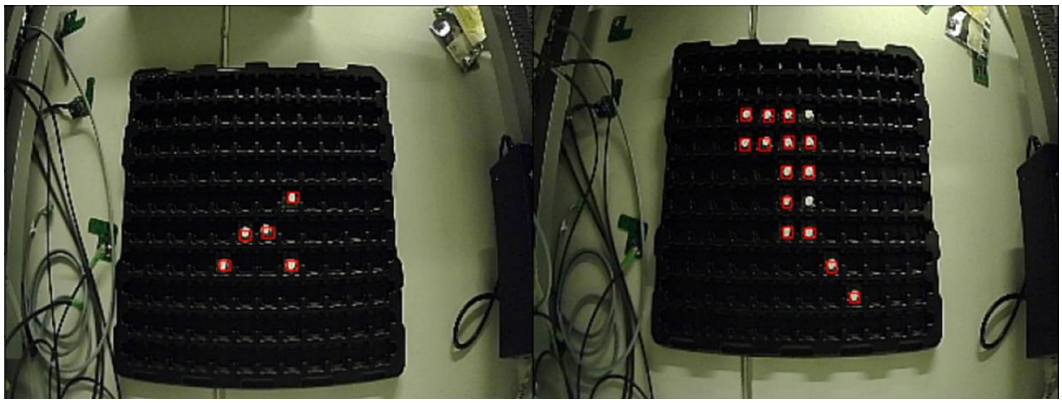

Abbildung 34: Modellvorhersage auf zwei Testbilder mit 60 Trainingsbildern.

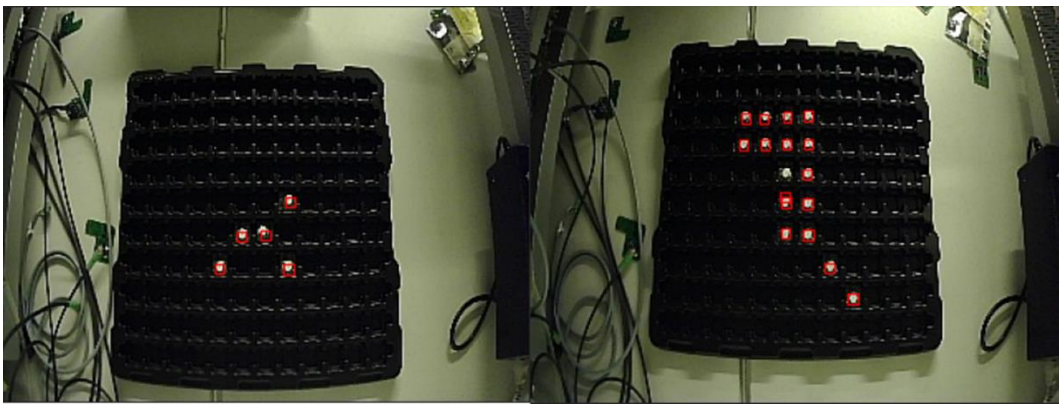

Abbildung 35: Modellvorhersage auf zwei Testbilder mit 449 Trainingsbildern.

4.2.2 Hyperparameter Tuning

Die Hyperparameter für das Hyperopt Skript sind in Kapitel 3.5 beschrieben und werden automatisch durch ein Skript ermittelt. Ein Test mit 170 Durchläufen, jeweils 50 Trainingsepochen und 400 Trainingsbildern wurde durchgeführt. Hyperopt benötigt einen sogenannten Search Space für jeden zu testenden Hyperparameter. Über mehrere Iterationen hinweg testet das Skript unterschiedliche Hyperparameter Einstellungen und ermittelt basierend auf einer definierten Evaluationsmetrik die optimalen Hyperparameter. Als Evaluationsmetrik für diese Testläufe wird der Validation Loss genutzt.

Tabelle 6: Hyperparameter für Retinanet Modell mit Search Space für das Hyperopt Skript.

Hyperparameter	**Suchbereich**
L2 Weight Decay	0.00001 – 0.001
Box Loss Weight	20 – 80 (Schrittgrösse 10)
Alpha	0.1 – 0.8
Gamma	1 – 2.5
Learning Rate	0.001 – 0.1
Aspect Ratios	Option 1: [0.5, 1.0, 2.0] Option 2: [0.3, 0.6, 1.2] Option 3: [0.8, 1.6, 3.0]]
Anchor Size	[0.5, 1, 2, 4]
Optimizer	SGD, Adam

Die Resultate des Hyperopt Testlaufs sind in der nachfolgenden Tabelle 7 zusammengefasst. Die Tabelle listet die besten 15 Resultate aufsteigend geordnet nach Validation Loss auf.

Tabelle 7: Resultate Hyperparameter Tuning Retinanet Objekterkennung.

Anchor Size	Aspect Ratios	Box Loss Weight	Alpha	Gamma	L2 Weight Decay	Learning Rate	Optimizer	Validation Loss
2	[0.3, 0.6, 1.2]	50	0.305	2.777	1.1556E-05	0.0110	SGD	0.310
0.5	[0.3, 0.6, 1.2]	40	0.229	2.736	6.4311E-05	0.0193	SGD	0.578
0.5	[0.3, 0.6, 1.2]	40	0.241	1.011	1.0269E-05	0.0054	Adam	0.592
0.5	[0.3, 0.6, 1.2]	50	0.232	2.748	5.9575E-05	0.0184	SGD	0.602
2	[0.5, 1.0, 2.0]	80	0.264	2.681	1.6719E-05	0.0011	SGD	0.656
4	[0.5, 1.0, 2.0]	80	0.323	2.349	5.0163E-05	0.0082	SGD	0.664
2	[0.3, 0.6, 1.2]	80	0.253	2.695	3.9909E-05	0.0122	SGD	0.709
2	[0.5, 1.0, 2.0]	70	0.477	2.439	8.9230E-05	0.0031	SGD	0.742
2	[0.5, 1.0, 2.0]	100	0.279	2.798	1.3455E-05	0.0129	SGD	0.754
2	[0.3, 0.6, 1.2]	50	0.298	2.126	2.6562E-05	0.0279	SGD	0.760
0.5	[0.3, 0.6, 1.2]	30	0.362	1.381	2.5038E-05	0.0100	Adam	0.794
0.5	[0.3, 0.6, 1.2]	30	0.328	1.489	1.0240E-05	0.0122	Adam	0.855
0.5	[0.5, 1.0, 2.0]	40	0.338	2.365	1.4491E-05	0.0674	SGD	0.860
0.5	[0.3, 0.6, 1.2]	40	0.203	2.800	1.2132E-04	0.0086	SGD	0.869
0.5	[0.3, 0.6, 1.2]	40	0.236	1.049	9.2920E-05	0.0010	Adam	0.881

Die Resultate des Hyperopt Durchlaufs in Tabelle 7 zeigen interessante Informationen. Zum einen schneidet der Adam Optimizer offensichtlich nicht so gut ab wie der SGD, zudem benötigt dieser einen anderen Gamma Wert als der SGD Optimizer. Generell betrachtet sind die besten Durchläufe mit höheren Gamma Werten verbunden, was darauf hindeutet, dass das Modell erst dann eine bessere Leistung zeigt, wenn es sich besonders auf die schweren Beispiele fokussiert. Ebenfalls überraschend zu sehen ist, dass die Anchor Size des besten Durchlaufs trotz der kleinen Objekte im Bild die Grösse 2 hat. Insgesamt scheint die Anchor Size beim Optimieren der Hyperparameter keinen starken Effekt auf den Modell Loss zu haben. Zusätzlich fällt auf, dass der L2 Weight Decay, der als Regularisierungsterm agiert, kleine Werte bei den besten Trainingsläufen hat.

4.2.3 Qualitätsmonitoring und Dashboard Darstellung

Da dieser Testlauf ein Experiment zur Erforschung der Möglichkeiten von Retinanet und dem Testen des Modellaufbaus ist, wurden keine Einträge in der Qualitätsmonitoring Deltatabelle erfasst und auch keine Modellvorhersagen zur regelmässigen Überprüfung hochgeladen. Die Funktionen zum Hochladen der Modellvorhersagen sowie der Berechnung der Qualitätsmetrik wurden als Teil dieses Projektes aber erstellt, getestet und in den Funktionsfiles abgelegt. Das Gleiche gilt für die Databricks Dashboard Darstellung, auf die in diesem Experiment ebenfalls verzichtet wurde.

5 Fazit

Im Zuge der vorliegenden Masterarbeit konnten die Anforderungen, beschrieben in Kapitel 1.2 erfolgreich umgesetzt werden. Zu den einzelnen Anforderungen wird nachfolgend das Fazit beschrieben.

I. **Aufbau der Datenpipeline und Anbindung der Kamera mit einem Raspberry Pi**: Der Aufbau der Pipeline vom Raspberry Pi zu Databricks und die Einbettung der Kamera mit Balena OS haben erfolgreich funktioniert. Während des gesamten Zeitraumes gab es keine Übertragungslücken oder vergleichbare Probleme mit Bezug auf Stabilität und Datenqualität. Die Wahl von Balena als Host Image für das Raspberry Pi war zu Beginn zwar eine Herausforderung, da insbesondere das Aufsetzen und Debugging des Codes auf dem Raspberry Pi mit Schwierigkeiten verbunden war. Rückblickend stellt es sich als richtige Entscheidung heraus, da sich das Management der gesamten Raspberry Pi Flotte einfach und intuitiv gestaltet.

II. **Aufbau einer Plattform auf Databricks, welches Modellverwaltung, Datenverwaltung und Labeling sowie Qualitätsmonitoring integriert**: Der Aufbau der Plattform auf Databricks, das Modellverwaltung- und Training sowohl als auch eine Anbindung zur Labeling Plattform Labelbox und ein Qualitätsmonitoring ermöglicht, hat sich als effektiv erwiesen. Das Nutzen der Databricks Umgebung bringt Vorteile, wie z.B. das redundante Speichern von Daten, das Nutzen eines einfach zugänglichen Datenspeichersystems mit Deltatabellen, das Einbinden von MLflow für die Versionierung von Modellen, das Nutzen von Spark für Big Data Themen und das einfache Einrichten von neuen Jobs. Neue Trainingsdaten können mit wenigen Funktionsaufrufen generiert, hochgeladen oder deren Annotationen von Labelbox importiert werden. Modelltrainings gestalten sich ebenfalls als effizient, da die Mlflow.pyfunc Klassen den gleichen Aufbau haben und die Trainingsdaten in einem einheitlichen Format akzeptieren.

III. **Aufbau eines Labeling Prozesses für die erfassten Bilder**: Die Anbindung zu Labelbox sowie das Nutzen von Model Assisted Labeling reduzieren die Zeit fürs Labeling auf ein Minimum. Zudem bringt das Qualitätsmonitoring nebst dem Überwachen der Modellgenauigkeit den Vorteil, weitere Trainingsdaten für neue Modellversionen zu generieren. Insbesondere die Anbindung zum Model Assisted Labeling gestaltete sich zu Beginn schwierig, hat sich rückblickend jedoch als richtige Entscheidung herausgestellt, da der Vorteil für das Qualitätsmonitoring den Nachteil des aufwändigen Einrichtens überwiegt.

IV. **Erstellen und Tuning eines CNN-Klassifikators für Ampelsignale**: Zusammenfassend lässt sich sagen, dass der Machine Learning Ansatz für das Projekt 'Lean Production Monitoring' ein Erfolg ist. Sämtliche Anforderungen, wie zum Beispiel die Erweiterbarkeit des Systems auf neue Aufgaben, die Zeit zum Einrichten und die Skalierbarkeit des Systems sind mit diesem Ansatz erfüllt und bieten eine gute Plattform, die mit neuen Modellen erweitert werden kann. Weitere Produktionsmaschinen können innerhalb kürzester Zeit mit einem Raspberry Pi und einer Kamera ausgestattet werden, der Rest kann mit wenigen Mausklicks über das Balena Webinterface erledigt werden. Die bereitgestellte VGG16 Modellklasse ermöglicht es, neue Modelle und Trainingsdaten für weitere Aufgaben schnell und ohne weiteren Scripting Aufwand zu erstellen. Grundsätzlich ist zu erwähnen, dass für die Aufgabe, ein Ampelsignal richtig zu erkennen, es viele einfachere Ansätze gegeben hätte als ein CNN-Modell. Der Punkt jedoch ist, dass das ganze System den Vorteil hat, flexibel und adaptiv auf alle möglichen Aufgaben zu richten. Ein System mit einem Lichtsensor würde zwar das Erkennen von Ampelsignalen deutlich einfacher gestalten, kann jedoch nur auf einem kleinen Aufgabenbereich eingesetzt werden.

V. **Erstellen und Tuning eines CNN-Modells zur Objekterkennung**: Die Implementierung des Retinanet Modells stellt sich ebenfalls als Erfolg heraus, da beim Testlauf eine gute Leistung erzielt wurde. Insbesondere das Hyperparameter Tuning ermöglicht es, das Maximum aus jedem Modell herauszuholen. Das Labeling der Daten war aufgrund der gleichen Labelbox Anbindung nicht schwierig, gestaltete sich jedoch arbeitsintensiver als bei den Labels für Klassifikationen. Jede Bounding Box muss vom Nutzer genau eingezeichnet werden, da schlechte Labels die Modellqualität beeinträchtigen. Mit einer durchschnittlichen Labelingzeit von 1 Minute pro Bild ist die Hemmschwelle zum Nutzen dieses Modells höher. Zusätzlich ist zu erwähnen, dass in diesem Testlauf lediglich eine Objektklasse verwendet wurde. Die Leistung des Modells in ähnlicher Umgebung auf mehreren Produkten ist noch zu Testen. Eine wichtige Erkenntnis aus diesem Testlauf ist zudem, dass die Kamera nach dem Einrichten nicht mehr bewegt werden darf, da Distanz, Lichtverhältnisse und andere Hintergrundobjekte einen Einfluss auf das Modelltraining haben können. Trotz dieser Schwierigkeiten zeigt sich, dass Retinanet ein universell einsetzbares und performantes neuronales Netzwerk ist und auch in einer Produktionsumgebung mit Erfolg zum Einsatz kommt.

Abschliessend kann gesagt werden, dass der Projektauftrag zum Aufbau eines performanten ML-Systems zum Zweck der Datengenerierung-, Verarbeitung und Analyse von Produktionsdaten im Praxisunternehmen umgesetzt wurde und ein erfolgreiches Projekt darstellt.

Weiterführende Arbeiten

Weiterführende Arbeiten auf diesem Bereich (mittel- und langfristig) könnten folgende Themen behandeln:

Mittelfristig: Aktuell werden die Bilder auf Databricks gespeichert und dort mit einem CNN-Modell analysiert. Dies hat den Nachteil, dass grosse Mengen an Bildern gestreamt und gespeichert werden müssen. Mittelfristig bietet es sich an, das Modell auf das Gerät vor Ort zu implementieren und nur noch die Modellvorhersagen und einen Bruchteil der Bilder zu Qualitätszwecken abzuspeichern (z.B. 10 % aller Bilder). Es gibt CNN-Modelle, die für den Einsatz auf leistungsschwächeren, mobilen Geräten entwickelt wurden, wie z.B. MobileNet, die sich für einen solchen Einsatz eignen würden. Die bestehende Infrastruktur mit einem Balena Host OS müsste dafür angepasst werden.
Das Hinzufügen von neuen CNN-Modellen könnte die Leistungsfähigkeit und Einsatzmöglichkeiten auf weitere Aufgaben weiter ansteigen lassen, daher empfiehlt es sich, weitere Modelle für den Einsatz zu prüfen. Insbesondere vortrainierte Modelle zur Objekterkennung (Transfer Learning) würden die Nachteile von längeren Labelingzeiten ausgleichen und die Anwendung dieser Modelle vereinfachen.

Langfristig: Als langfristige Möglichkeit könnte das manuelle Überprüfen der Stichproben durch ein Ensemble Modell abgelöst werden. Ein Ensemble Modell besteht aus mehreren unterschiedlichen ML-Modellen, die in ihrer Gesamtheit besser abschneiden als es jedes Modell individuell tut. Ein Ensemble Verfahren könnte aufgrund seiner besseren Performanz gegenüber dem Standardmodell zur Überprüfung der Stichproben eingesetzt werden und somit den manuellen Labeling Aufwand weiter reduzieren.

Moralische/Ethische Aspekte dieser Arbeit

Diese Arbeit ermöglicht es, ein verbessertes Lagebild von Produktionsprozessen zu erlangen. Ziel davon ist es, Möglichkeiten zum Steigern der Produktivität und Qualität aufzuzeigen. Solch ein System wirft jedoch auch moralische und ethische Fragen auf, insbesondere im Hinblick auf Datenschutz, Persönlichkeitsschutz und Überwachung von Mitarbeitern. Datenschutz ist ein grundlegendes Recht eines jeden Menschen. Mit dem Aufbau eines solchen Systems werden auch Daten gesammelt, die zur Erstellung von Profilen von Personen und deren Produktivität genutzt werden könnten. Daher ist es von entscheidender Bedeutung, dass das Unternehmen Massnahmen zum Schutz der Privatsphäre der Mitarbeiter ergreift. Die Kameras, insbesondere diejenigen auf den manuellen Prozessschritten sollten keine Mitarbeiter filmen, sondern lediglich auf Produktionsmaterial und Werkzeuge fokussieren. Des Weiteren ist es wichtig, das neue System den Mitarbeitern vor Ort klar zu kommunizieren. Die Mitarbeiter müssen darüber informiert werden, für was diese Kameras eingesetzt werden und was mit den Bildern passiert. Fehlende Informationen können schnell zu falschen Schlüssen, Misstrauen und einem schlechteren Arbeitsklima führen.

Persönliche Reflexion und Schlusswort

Diese Arbeit ermöglichte es mir, mein angeeignetes Wissen aus dem MAS Data Science zu festigen und gebündelt in einem Projekt umzusetzen. Meine Kenntnisse über Databricks und die Möglichkeiten dieser Plattform, wie das Nutzen von MLflow oder das Arbeiten mit Deltatabellen haben sich stark verbessert als auch das Arbeiten mit Python Funktionsfiles. Besonders beeindruckt haben mich die Vorzüge von Hyperopt, was mein Verständnis der einzelnen Hyperparameter stark verbessert hat. Der Ansatz, ein Modell als MLflow Python Klasse zu implementieren war mir ebenfalls unbekannt, hat mir jedoch eine neue und effiziente Art aufgezeigt, unterschiedlichste Modelle in einem einheitlichen Format zu verpacken und für weniger erfahrene Nutzer einfach zugänglich zu machen.

Rückblickend gab es einige Hürden, wie beispielsweise das Implementieren des Model Assisted Labeling, das Verpacken des Retinanet Modells als Mlflow.pyfunc Klasse oder das Einarbeiten mit Balena und dem damit verbundenen Aufwand des Python Debuggings für die Kameraanbindung, deren Lösungsfindung mich oft in meinem Zeitplan zurückgeworfen haben. Auch das Arbeiten mit Databricks und Spark war für mich zu Beginn eine Herausforderung, da die Debugging Tools auf dieser Plattform längst nicht so gut sind, wie die Möglichkeiten einer lokalen IDE wie Pycharm.

In einem nächsten Mal würde ich mein Vorgehen folgendermassen verbessern: Reduktion des Arbeitspensums auf 80%, da ich diese Arbeit hauptsächlich in meiner Freizeit schreiben und deren Inhalt erarbeiten musste, was eine andauernde Stresssituation bedeutete. Ein weiterer Punkt, den ich heute anders machen würde, ist das Einsetzen von Hyperopt für die Modelloptimierung. Dieses Paket kam meiner Meinung nach zu spät zum Einsatz und hätte mir zu Beginn des Modellaufbaus- und Trainings viel Zeit erspart. Zuletzt hätte ich die KI Chat GPT früher zum Debuggen von Code Segmenten einsetzen sollen, da insbesondere Python Skripte mit Spark Anwendungen für unerfahrene Spark Benutzer zeitaufwändig sein können.

Trotz dieser intensiven und belastbaren Zeit war die Arbeit an diesem Projekt und das Verfassen dieser Masterarbeit eine grosse Bereicherung für mich und ich blicke voller Stolz auf das Erreichte zurück.

6 Quellenverzeichnis

Anwar, A. (2019, June 7). *Difference between AlexNet, VGGNet, ResNet, and Inception*. Towards Data Science. https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

Balena. (n.d.). *BalenaOS Architecture Overview*. Retrieved June 25, 2023, from https://os-docs.balena.io/architecture

Brownlee, J. (2019, April 19). *A Gentle Introduction to Padding and Stride for Convolutional Neural Networks*. Deep Learning for Computer Vision. https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/

Cassimiro, G. (2021, June 16). *Transfer Learning with VGG16 and Keras*. https://towardsdatascience.com/transfer-learning-with-vgg16-and-keras-50ea161580b4

Christiansen, A. (2018, September 15). *Anchor Boxes - The key to quality object detection*. Towards Data Science. https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9

Girshick, R. (2015). *Fast R-CNN*. http://arxiv.org/abs/1504.08083

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013). *Rich feature hierarchies for accurate object detection and semantic segmentation*. http://arxiv.org/abs/1311.2524

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. http://arxiv.org/abs/1512.03385

ILSVRC. (2014). *ILSVRC2014 Results*. https://image-net.org/challenges/LSVRC/2014/results

IPH Hannover. (2023). *Lean Production als Teil des Lean Managements*. https://www.iph-hannover.de/de/information/lean-production/lean-production/

Jordan, J. (2018, March 1). *Setting the learning rate of your neural network*.

Krishnan, S. (2021, September 8). *How do determine the number of layers and neurons in the hidden layer?* Geek Culture. https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3

Labelbox. (2023a, April). *Import annotations as pre-labels*. https://docs.labelbox.com/docs/model-assisted-labeling

Labelbox. (2023b, May 25). *Microsoft Azure Blob Storage*. https://docs.labelbox.com/docs/microsoft-azure-blob-storage

Labelbox. (2023c, June 8). *Access, storage, and security*. https://docs.labelbox.com/docs/access-storage

Lee, D., & Heintz, B. (2019, August 14). *Productionizing Machine Learning with Delta Lake*. Engineering Blog. https://www.databricks.com/blog/2019/08/14/productionizing-machine-learning-with-delta-lake.html

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). *Focal Loss for Dense Object Detection*. http://arxiv.org/abs/1708.02002

Litzel, N., & Luber, S. (2019, February 25). *Was ist ein Convolutional Neural Network?* Was Ist Ein Convolutional Neural Network? https://www.bigdata-insider.de/was-ist-ein-convolutional-neural-network-a-801246/

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2015). *SSD: Single Shot MultiBox Detector*. https://doi.org/10.1007/978-3-319-46448-0_2

Lu, Y., Morris, K., & Frechette, S. (2016). *Current Standards Landscape for Smart Manufacturing Systems*. https://doi.org/10.6028/NIST.IR.8107

Mahajan, P. (2020, October 23). *Fully Connected vs Convolutional Neural Networks*. The Startup. https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5

Mishra, D. (2020, May 9). *Weight Decay == L2 Regularization?* Towards Data Science. https://towardsdatascience.com/weight-decay-l2-regularization-90a9e17713cd

Müller, S. (2022, May 18). *Klassifikation im Maschinellen Lernen*. https://lamarr-institute.org/de/ml-klassifikation/

Nayak, R. (2022). Focal Loss: A better alternative for Cross-Entropy. *Towards Data Science*. https://towardsdatascience.com/focal-loss-a-better-alternative-for-cross-entropy-1d073d92d075

NB, H. (2019, December 10). *Confusion Matrix, Accuracy, Precision, Recall, F1 Score*. Analytics Vidhya. https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

o.A. (2021, August 27). *Github Google Automl*. https://github.com/google/automl/blob/master/efficientdet/det_model_fn.py

o.A. (2023, January 21). *Activation function for Output Layer in Regression, Binary, Multi-Class, and Multi-Label Classification*. https://androidkt.com/activation-function-for-output-layer-in-regression-binary-multi-class-and-multi-label-classification/

o.V. (2023a). *What is MLflow?* https://www.run.ai/guides/machine-learning-operations/mlflow

o.V. (2023b, May 24). *What is Delta Lake?* What Is Delta Lake? https://docs.databricks.com/delta/index.html

Park, S. (2021, June 21). *A 2021 Guide to improving CNNs-Optimizers: Adam vs SGD*. https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008

Prijono, B. (2018, March 7). *Student Notes: Convolutional Neural Networks (CNN) Introduction*. Student Notes: CNN Introduction. https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. http://arxiv.org/abs/1506.01497

Rosebrock, A. (2016, November 7). *Intersection over Union for object detection*. Pyimagesearch. https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Shah, D. (2022, March 7). *Mean Average Precision (mAP) Explained: Everything you need to know*. https://www.v7labs.com/blog/mean-average-precision#h2

Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*.

Soleymani, A. (2022, April 8). *Your validation loss is lower than your training loss? This is why!* Towards Data Science. https://towardsdatascience.com/what-your-validation-loss-is-lower-than-your-training-loss-this-is-why-5e92e0b1747e#:~:text=The%20regularization%20terms%20are%20only,loss%20than%20the%20training%20set.

Tensorflow. (2023a, January 22). *Object detection with Model Garden*. https://www.tensorflow.org/tfmodels/vision/object_detection

Tensorflow. (2023b, March 31). *tfm.vision.models.RetinaNetModel*. https://www.tensorflow.org/api_docs/python/tfm/vision/models/RetinaNetModel

Thakun, R. (2023, March 9). *Beginner's Guide to VGG16 Implementation in Keras*. https://builtin.com/machine-learning/vgg16

Tsang, S.-H. (2018, September 15). *Review: ResNet - Winner of ILSVRC 2015 (Image Classification, Localization, Detection)*. https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8

Tsang, S.-H. (2019, January 24). *Review: RetinaNet - Focal Loss (Object Detection)*. Towards Data Science. https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4

Wang, C.-F. (2019, January 8). *The Vanishing Gradient Problem*. Towards Data Science.

Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2019). Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, *115*, 213–237. https://doi.org/10.1016/J.YMSSP.2018.05.050

7 Abbildungsverzeichnis

Abbildung 1: Prozessschema für initiales Projekt. 5
Abbildung 2: Ampelsystem an einer Produktionsmaschine der Sensirion AG. 6
Abbildung 3: Funktionsweise eines Convolutional Filters/Kernels mit der Grösse 3x3 und Stride 1. 10
Abbildung 4: Max Pooling und Average Pooling. 11
Abbildung 5: Fully Connected Layers mit der Tiefe 5. Jedes Neuron eines Layers ist mit jedem Neuron des nachfolgenden Layer verbunden. Die 4 Neuronen des letzten Layers könnten zur Klassifikation von 4 Objekten, oder der Vorhersage von 4 Bounding Box Koordinaten dienen (Mahajan, 2020). 11
Abbildung 6: Links oben: Endoskop Kamera in einer Plastikschiene. Links unten: Bild aus der Plastikschiene. 16
Abbildung 7: Delta Lake Tabellenschema (Lee & Heintz, 2019). 20
Abbildung 8: Aufbau des VGG16 Netzwerks (Simonyan & Zisserman, 2014). 22
Abbildung 9: Aufbau des Retinanet Modells (Lin et al., 2017). 24
Abbildung 10: Schema Qualitätsmonitoring. 28
Abbildung 11: Konfusionsmatrix für Modellvorhersagen. 29
Abbildung 12: Definition der Intersection over Union (Rosebrock, 2016). 31
Abbildung 13: Übersicht Gesamtsystem. 34
Abbildung 14: Aufnahmen von den Endoskop Kameras innerhalb der Plastikschiene. 36
Abbildung 15: Klassenverteilung im Testdatensatz für Ampelsignale. 37
Abbildung 16: F1-Score des vortrainierten Imagenet Modells. Das Limit von 95% ist eine Anforderung an die Modellqualität. 39
Abbildung 17: F1-Score des Modells mit randomisierten Gewichten. Das Limit von 95% ist eine Anforderung an die Modellqualität. 39
Abbildung 18: Modell Loss mit 800 Trainingsbildern. 40
Abbildung 19: Konfusionsmatrix für Modellvorhersagen (Imagenet Modell) auf Testdatensatz. 42
Abbildung 20: Konfusionsmatrix für Modellvorhersagen (Modell mit randomisierten Gewichten) auf Testdatensatz. 42
Abbildung 21: Ausgeschaltete Ampel mit schwachem orangefarbenem Lichteinfall. Das Bild entstand kurz nachdem die orange Ampel ausgeschaltet wurde. 43
Abbildung 22: Qualitätsübersicht über die vergangenen 3 Wochen für eine überwachte Ampel. 46
Abbildung 23: Abgelöste Plastikschiene vom Ampelsystem. 47
Abbildung 24: Tages und Stundendarstellung des Maschinenstatus anhand der Ampel an einer Die Bonding Maschine. 48
Abbildung 25: Dashboard Darstellung mit Ampelstatus und berechnetem Produktionsausstoss. 49
Abbildung 26: Materialtablar mit Sensormodulen. Oben Links befindet sich ein Laufzettel mit Produktspezifischen Informationen. 50
Abbildung 27: Bilder des Materialzwischenlagers mit eingezeichneten Bounding Box Annotationen (Rot). 51
Abbildung 28: Anzahl Bounding Boxes (Anzahl Objekte) pro Bild im Testdatensatz (80 Bilder). 52
Abbildung 29: Precision-Recall Curve auf den Testdatensatz mit unterschiedlicher Anzahl Trainingsbildern. Als Kriterium für ein Match gilt ein IoU Wert von 0.5. 54
Abbildung 30: Mean Average Precision Score gegenüber Anzahl Trainingsbilder für das Modelltraining. 55
Abbildung 31: Die Durchschnittliche Anzahl False Positives und False Negatives pro Testbild 56
Abbildung 32: Modellvorhersage auf zwei Testbilder mit 20 Trainingsbildern. 57
Abbildung 33: Modellvorhersage auf zwei Testbilder mit 40 Trainingsbildern. 57
Abbildung 34: Modellvorhersage auf zwei Testbilder mit 60 Trainingsbildern. 57
Abbildung 35: Modellvorhersage auf zwei Testbilder mit 449 Trainingsbildern. 57
Abbildung 36: Qualitätsmetrik aus Testlauf für Ampelsignalerkennung mit vortrainierten Imagenet Gewichten 74
Abbildung 37: Qualitätsmetrik aus Testlauf für Ampelsignalerkennung mit zufälligen Gewichten 76
Abbildung 38: Modell Loss mit 50 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten. 77
Abbildung 39: Modell Loss mit 100 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten. 77
Abbildung 40: Modell Loss mit 200 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten. 78

Abbildung 41: Modell Loss mit 400 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten. 78
Abbildung 42: Modell Loss mit 800 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten. 79
Abbildung 43: Konfusionsmatrix für Imagenet Modell trainiert mit 50 Trainingsbildern und getestet mit dem Testdatensatz. 80
Abbildung 44: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 50 Trainingsbildern und getestet mit dem Testdatensatz. 80
Abbildung 45: Konfusionsmatrix für Imagenet Modell trainiert mit 100 Trainingsbildern und getestet mit dem Testdatensatz. 81
Abbildung 46: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 100 Trainingsbildern und getestet mit dem Testdatensatz. 81
Abbildung 47: Konfusionsmatrix für Imagenet Modell trainiert mit 200 Trainingsbildern und getestet mit dem Testdatensatz. 82
Abbildung 48: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 200 Trainingsbildern und getestet mit dem Testdatensatz. 82
Abbildung 49: Konfusionsmatrix für Imagenet Modell trainiert mit 400 Trainingsbildern und getestet mit dem Testdatensatz. 83
Abbildung 50: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 400 Trainingsbildern und getestet mit dem Testdatensatz. 83
Abbildung 51: Konfusionsmatrix für Imagenet Modell trainiert mit 800 Trainingsbildern und getestet mit dem Testdatensatz. 84
Abbildung 52: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 800 Trainingsbildern und getestet mit dem Testdatensatz. 84
Abbildung 53: Modellvorhersagen Retinanet (trainiert auf 449 Bildern) auf den Testbildern. 85

8 Tabellenverzeichnis

Tabelle 1: Auftrag und Anforderungen an das Projekt. 7
Tabelle 2: Übersicht Datentypen des Bilder Streams in der Deltatabelle. 19
Tabelle 3: Klassenlabel für Ampelzustände 35
Tabelle 4: Hyperparameter mit Search Space für das Hyperopt Skript. 44
Tabelle 5: Top 15 Resultate Hyperparameter Tuning für Ampelzustände. 45
Tabelle 6: Hyperparameter für Retinanet Modell mit Search Space für das Hyperopt Skript. 58
Tabelle 7: Resultate Hyperparameter Tuning Retinanet Objekterkennung. 59
Tabelle 8: Resultate aus Testlauf für Ampelsignalerkennung mit vortrainierten Imagenet Gewichten 73
Tabelle 9: Resultate aus Testlauf für Ampelsignalerkennung mit zufälligen Gewichten 75

9 Anhang

Der Quellcode dieser Arbeit ist in zensierter Form (ohne firmeninterne Passwörter, Speicherorte, API Keys oder ähnliches) auf folgendem Github Verzeichnis hochgeladen:

https://github.com/daveyd92/Masterarbeit_David_Frey/tree/main

Der Code über die Datenpipeline wurde aufgrund sicherheitstechnischer Bedenken und gemeinsamer Nutzung mit anderen Projekten nicht einbezogen.

Der Code ist in folgender Struktur auf dem Github Repository abgelegt:

Masterarbeit_David_Frey

- Databricks_Platform
    - -data/train_test
        - Get_data.py
    - models
        - Retinanet_Object_Detector.py
        - VGG16_Single_Classifier.py
    - quality_assurance
        - quality_assurance.py
    - Classifier_AndonDetector_QualityCheck.py
    - Classifier_andon_detector_generate_and_train_model.py
    - Classifier_andon_detector_hyperopt.py
    - Object_detector_KLI_generate_and_train_model.py
    - Object_detector_KLI_hyperopt.py
    - QA_Upload_Model_Predictions.py

Anbei folgt eine kurze Erklärung sämtlicher vorhandener .py Files:

- **Get_data.py**: Dieses Python File beinhaltet sämtliche Funktionen, die zum Hochladen von zufälligen Testbildern auf Labelbox oder dem Importieren von Annotationen von Labelbox oder dem Laden von Trainingsbildern inklusive Annotationen dienen.
- **Retinanet_Object_Detector.py**: Dieses Python File beinhaltet das Retinanet Modell als mlflow.pyfunc.PythonModel Klasse. Es beinhaltet sämtliche Funktionen, die zum Erstellen und

Trainieren des Modells nötig sind, sowie weitere Hilfsfunktionen zum Augmentieren von Bildern und dem Vorverarbeiten von Bildern für die Funktion zur Vorhersage (predict). Ein neues Modell wird erstellt, indem eine Instanz dieser Klasse erzeugt wird. Die Instanz erbt sämtliche Funktionen dieser Klasse und kann mit Mlflow als Modell registriert werden.

- **VGG16_Single_Classifier.py**: Dieses Python File beinhaltet das VGG16 Modell als mlflow.pyfunc.PythonModel Klasse. Es beinhaltet sämtliche Funktionen, die zum Erstellen und Trainieren des Modells nötig sind, sowie weitere Hilfsfunktionen zum Augmentieren von Bildern und dem Vorverarbeiten von Bildern für die Funktion zur Vorhersage (predict). Ein neues Modell wird erstellt, indem eine Instanz dieser Klasse erzeugt wird. Die Instanz erbt sämtliche Funktionen dieser Klasse und kann mit Mlflow als Modell registriert werden.
- **quality_assurance.py**: Dieses Python File beinhaltet sämtliche Funktionen, die für das Model Assisted Labeling verwendet werden. Beispielsweise lädt die Funktion 'periodic_check' eine Auswahl zufälliger Bilder, die bereits eine Modellvorhersage haben, in einem spezifischen Zeitraum inklusive deren Bildannotation im Labelbox Format hoch, wo sie dann von einem Nutzer manuell überprüft wird.
- **Classifier_AndonDetector_QualityCheck.py**: Dieses Python File entstammt einem Jupyter Notebook File und ist für das Qualitätsmonitoring des Modells zur Ampelerkennung (AndonDetector) zuständig. Es lädt die Annotationen aus dem Model Assisted Labeling und vergleicht diese mit den Modellvorhersagen in der Silver Deltatabelle. Auf diese Testdaten werden dann die Qualitätsmetriken F1-Score, Precision, Recall und Accuracy berechnet und in die Deltatabelle zur Qualitätsüberwachung geschrieben. Die letzten Codezeilen dienen zum Visualisieren einzelner Metriken, die unter anderem auch Abbildungen in dieser Arbeit generiert haben.
- **Classifier_andon_detector_generate_and_train_model.py**: Dieses Python File entstammt einem Jupyter Notebook File und wird zum Erzeugen des Modells zur Ampelerkennung verwendet. Das File lädt die Testdaten basierend auf dem Labelbox Projekt, welches die Trainingsbilder der Ampeln beinhaltet, erzeugt eine Instanz des VGG16 Klassifikators aus dem 'models/VGG16_Single_Classifier.py' File, übergibt die relevanten Hyperparameter und trainiert das Modell. Anschliessend wird das Modell auf Mlflow registriert. Die nächste Sektion lädt das soeben registrierte Modell aus Mlflow und berechnet, basierend auf dem Testdatensatz, die Qualitätsmetriken und generiert weitere Plots, welche auch in dieser Arbeit genutzt wurden.
- **Classifier_andon_detector_hyperopt.py**: Dieses Python File entstammt einem Jupyter Notebook File und wird zum Ermitteln der optimalen Hyperparameter für das Modell zur Ampelerkennung verwendet. Der Code lädt das aktuelle Modell und die Trainingsdaten und führt

eine definierte Anzahl an Trainingsläufen mit einem definierten Search Space durch. Schlussendlich werden die Testresultate dargestellt und als .csv File gespeichert.

- **Object_detector_KLI_generate_and_train_model.py**: Dieses Python File entstammt einem Jupyter Notebook File und wird zum Erzeugen des Retinanet Objekt Detektors verwendet. Das File lädt die Testdaten basierend auf dem Labelbox Projekt, welches die Trainingsbilder der Ampeln beinhaltet, erzeugt eine Instanz des Retinanet Modells aus dem 'models/Retinanet_Object_Detector.py' File, übergibt die relevanten Hyperparameter und trainiert das Modell. Anschliessend wird das Modell auf Mlflow registriert. Die nächste Sektion lädt das soeben registrierte Modell aus Mlflow und berechnet, basierend auf dem Testdatensatz, die Qualitätsmetriken (mAP) und generiert weitere Plots, welche auch in dieser Arbeit genutzt wurden.
- **Object_detector_hyperopt.py**: Dieses Python File entstammt einem Jupyter Notebook File und wird zum Ermitteln der optimalen Hyperparameter für das Retinanet Modell zur Objekterkennung verwendet. Der Code lädt das aktuelle Modell und die Trainingsdaten und führt eine definierte Anzahl an Trainingsläufen mit einem definierten Search Space durch. Schlussendlich werden die Testresultate dargestellt und als .csv File gespeichert.
- **QA_Upload_Model_Predictions.py:** Dieses Python File entstammt einem Jupyter Notebook File und wird zum Hochladen von zufälligen Bildern verschiedener Kameras über einen definierten Zeitraum genutzt. Es benutzt die Funktionen, die im quality_assurance.py File implementiert sind. Dieses Notebook wird als Databricks Job alle 3 Tage ausgeführt und inkludiert alle Kameras, die in diesem Notebook festgelegt sind. Der Code selbst kann unterscheiden, ob es sich um Klassifikationsdaten oder um Bounding Boxes handelt.
-

Tabelle 8: Resultate aus Testlauf für Ampelsignalerkennung mit vortrainierten Imagenet Gewichten

Vortrainiertes Imagenet Modell		Qualitätsmetrik				Modell Probability Scores				
Trainingsbilder	Label	Precision	Recall	F1-Score	Overall Accuracy	Min Score	Max Score	Avg Score	Median	Labellingaufwand (min)
50	Amber	0.712	0.743	0.727	0.852	0.484	0.993	0.805	0.785	3.3
50	Green	0.923	0.857	0.889	0.852	0.386	0.998	0.709	0.668	3.3
50	Green/Amber	1.000	1.000	1.000	0.852	0.387	0.981	0.742	0.874	3.3
50	Green/Amber/Red	0.808	0.808	0.808	0.852	0.456	0.675	0.615	0.636	3.3
50	Off	1.000	0.984	0.992	0.852	0.995	0.998	0.998	0.998	3.3
50	Red	0.726	0.757	0.741	0.852	0.504	0.995	0.885	0.966	3.3
100	Amber	1.000	0.786	0.880	0.924	0.706	0.936	0.834	0.851	6.7
100	Green	0.838	0.957	0.893	0.924	0.539	1.000	0.892	0.969	6.7
100	Green/Amber	1.000	0.891	0.943	0.924	0.507	0.990	0.858	0.980	6.7
100	Green/Amber/Red	0.743	1.000	0.852	0.924	0.417	0.996	0.860	0.994	6.7
100	Off	1.000	0.984	0.992	0.924	0.999	0.999	0.999	0.999	6.7
100	Red	0.944	0.971	0.958	0.924	0.601	0.997	0.945	0.983	6.7
200	Amber	1.000	0.843	0.915	0.919	0.625	0.999	0.975	0.994	13.3
200	Green	0.917	0.786	0.846	0.919	0.459	0.998	0.900	0.942	13.3
200	Green/Amber	1.000	1.000	1.000	0.919	0.677	0.999	0.976	0.995	13.3
200	Green/Amber/Red	0.619	1.000	0.765	0.919	0.513	0.998	0.930	0.995	13.3
200	Off	1.000	0.984	0.992	0.919	0.997	1.000	0.999	0.999	13.3
200	Red	0.908	0.986	0.945	0.919	0.460	0.998	0.935	0.991	13.3
400	Amber	0.959	1.000	0.979	0.971	0.971	0.999	0.971	0.995	26.7
400	Green	0.946	1.000	0.972	0.971	0.501	1.000	0.978	0.998	26.7
400	Green/Amber	0.939	1.000	0.968	0.971	0.506	0.999	0.946	0.997	26.7
400	Green/Amber/Red	1.000	0.731	0.844	0.971	0.504	0.990	0.962	0.988	26.7
400	Off	1.000	0.984	0.992	0.971	1.000	1.000	1.000	1.000	26.7
400	Red	1.000	0.971	0.986	0.971	0.509	1.000	0.981	0.998	26.7
800	Amber	0.986	1.000	0.993	0.997	0.460	1.000	0.992	1.000	53.3
800	Green	1.000	1.000	1.000	0.997	0.971	1.000	0.999	1.000	53.3
800	Green/Amber	1.000	1.000	1.000	0.997	0.973	1.000	0.999	1.000	53.3
800	Green/Amber/Red	1.000	1.000	1.000	0.997	0.928	1.000	0.986	0.999	53.3
800	Off	1.000	0.984	0.992	0.997	1.000	1.000	1.000	1.000	53.3
800	Red	1.000	1.000	1.000	0.997	0.986	1.000	0.999	1.000	53.3

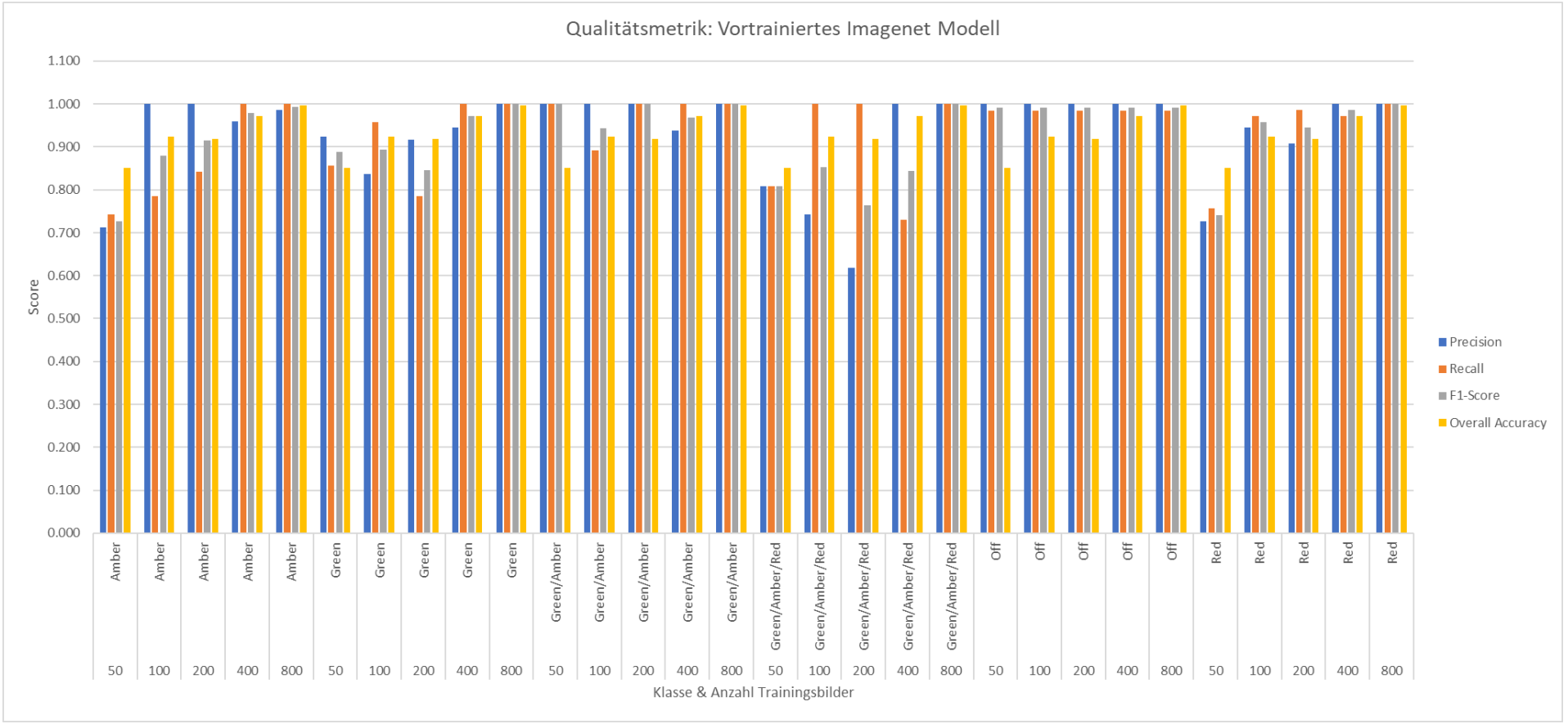

Abbildung 36: Qualitätsmetrik aus Testlauf für Ampelsignalerkennung mit vortrainierten Imagenet Gewichten

Tabelle 9: Resultate aus Testlauf für Ampelsignalerkennung mit zufälligen Gewichten

Modell mit zufälligen Gewichten		Qualitätsmetrik				Modell Probability Scores				
Trainingsbilder	Label	Precision	Recall	F1-Score	Overall Accuracy	Min Score	Max Score	avg Score	Median	Labellingaufwand (min)
50	Amber	0.000	0.000	0.000	0.554	0.230	0.276	0.269	0.276	3.3
50	Green	0.778	0.200	0.318	0.554	0.259	0.746	0.610	0.704	3.3
50	Green/Amber	0.519	0.870	0.650	0.554	0.367	0.629	0.533	0.573	3.3
50	Green/Amber/Red	0.329	0.960	0.490	0.554	0.233	0.814	0.517	0.523	3.3
50	Off	1.000	0.984	0.992	0.554	0.192	0.197	0.194	0.194	3.3
50	Red	0.477	0.729	0.576	0.554	0.332	0.630	0.451	0.381	3.3
100	Amber	0.725	0.943	0.820	0.793	0.506	0.986	0.817	0.918	6.7
100	Green	0.810	0.671	0.734	0.793	0.390	0.851	0.534	0.505	6.7
100	Green/Amber	0.878	0.783	0.828	0.793	0.730	0.998	0.971	0.998	6.7
100	Green/Amber/Red	0.444	0.960	0.608	0.793	0.415	0.862	0.656	0.608	6.7
100	Off	1.000	1.000	1.000	0.793	0.864	0.996	0.992	0.994	6.7
100	Red	1.000	0.529	0.692	0.793	0.423	0.535	0.519	0.523	6.7
200	Amber	0.971	0.943	0.957	0.927	0.666	1.000	0.974	0.998	13.3
200	Green	0.892	0.943	0.917	0.927	0.382	0.994	0.829	0.899	13.3
200	Green/Amber	0.854	0.891	0.872	0.927	0.380	1.000	0.893	0.998	13.3
200	Green/Amber/Red	0.789	0.600	0.682	0.927	0.370	0.602	0.546	0.590	13.3
200	Off	1.000	0.984	0.984	0.927	0.892	0.985	0.967	0.981	13.3
200	Red	0.945	0.945	0.965	0.927	0.397	0.988	0.861	0.970	13.3
400	Amber	0.851	0.900	0.875	0.936	0.443	0.998	0.998	0.954	26.7
400	Green	1.000	0.986	0.993	0.936	0.574	1.000	0.944	0.991	26.7
400	Green/Amber	0.920	1.000	0.958	0.936	0.780	1.000	0.967	0.999	26.7
400	Green/Amber/Red	1.000	0.880	0.936	0.936	0.970	0.993	0.990	0.992	26.7
400	Off	1.000	1.000	1.000	0.936	0.928	1.000	0.997	0.998	26.7
400	Red	0.894	0.843	0.868	0.936	0.516	0.998	0.871	0.894	26.7
800	Amber	0.986	1.000	0.993	0.985	0.985	1.000	1.000	1.000	53.3
800	Green	1.000	1.000	1.000	0.985	0.977	1.000	0.999	1.000	53.3
800	Green/Amber	0.920	1.000	0.958	0.985	0.820	1.000	0.993	1.000	53.3
800	Green/Amber/Red	1.000	0.840	0.913	0.985	0.579	1.000	0.932	1.000	53.3
800	Off	1.000	0.984	0.992	0.985	1.000	1.000	1.000	1.000	53.3
800	Red	1.000	1.000	1.000	0.985	0.994	1.000	1.000	1.000	53.3

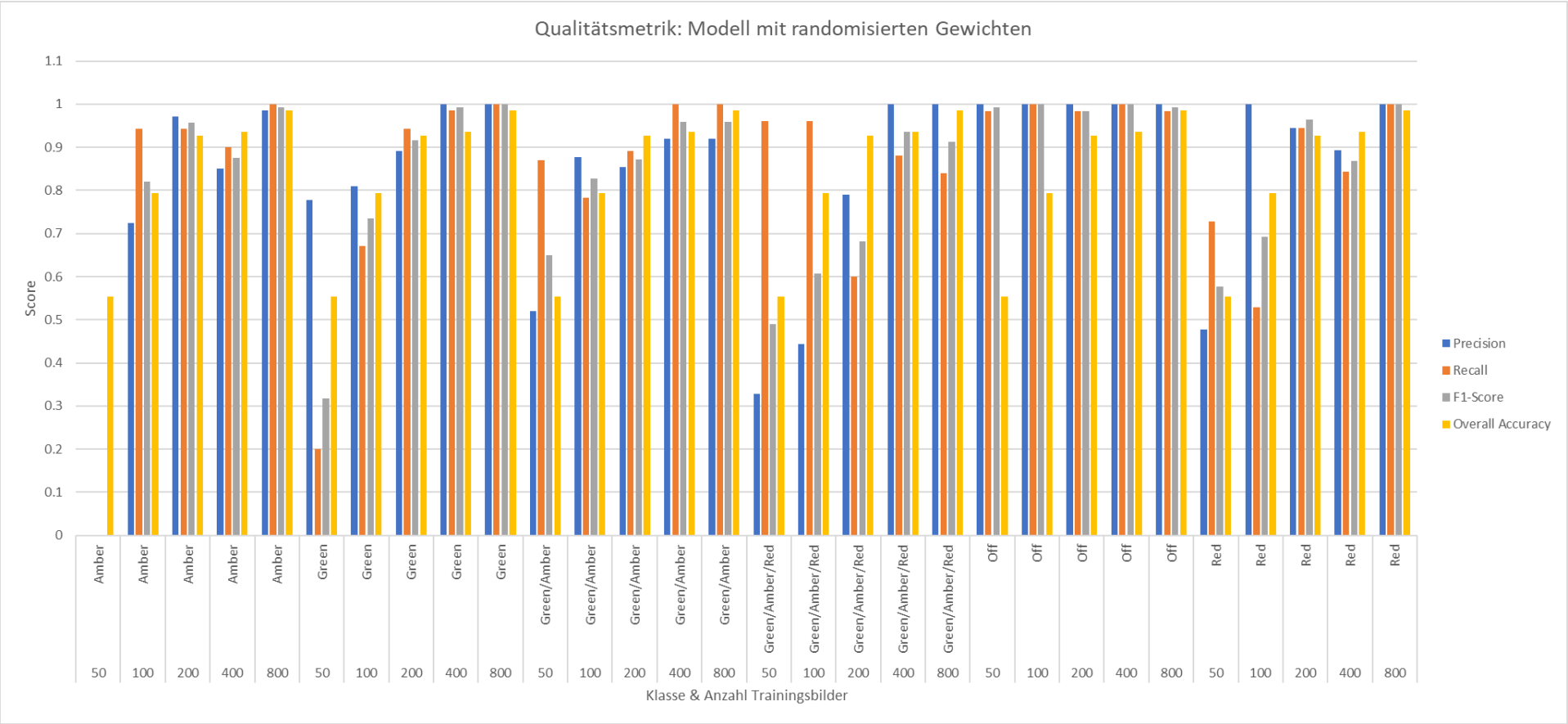

Abbildung 37: Qualitätsmetrik aus Testlauf für Ampelsignalerkennung mit zufälligen Gewichten

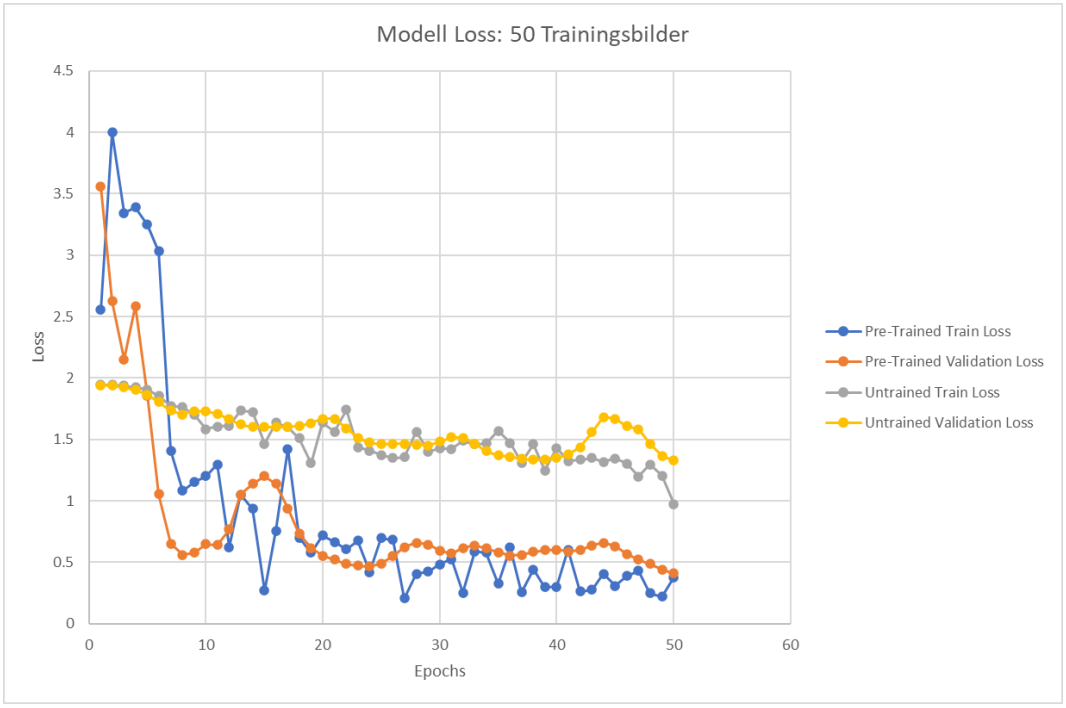

Abbildung 38: Modell Loss mit 50 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten.

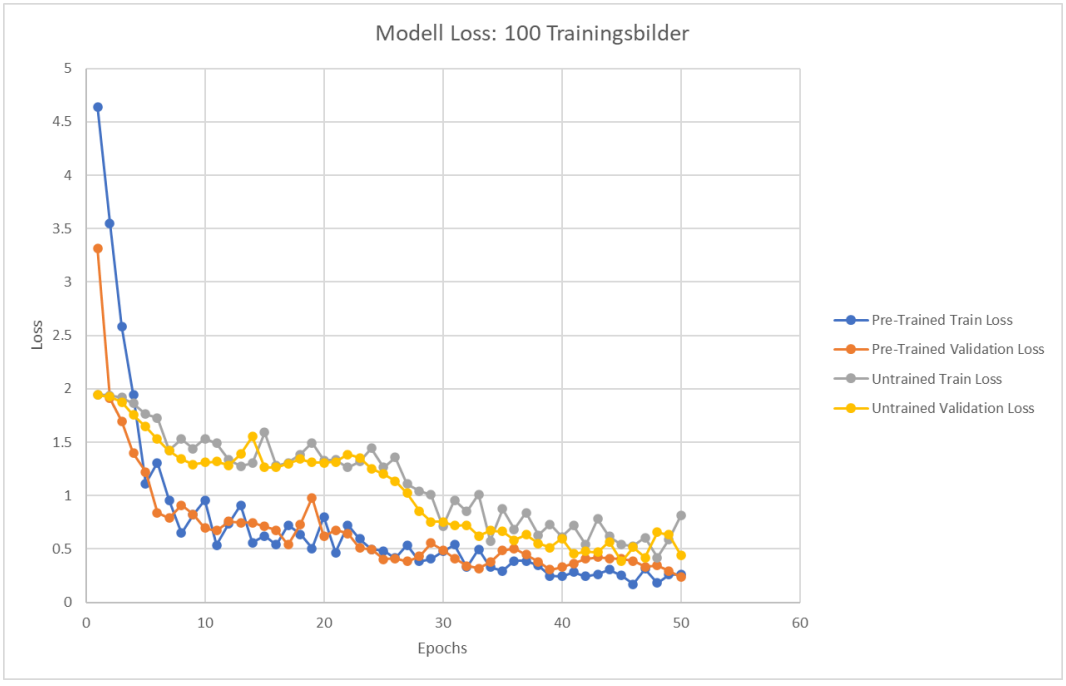

Abbildung 39: Modell Loss mit 100 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten.

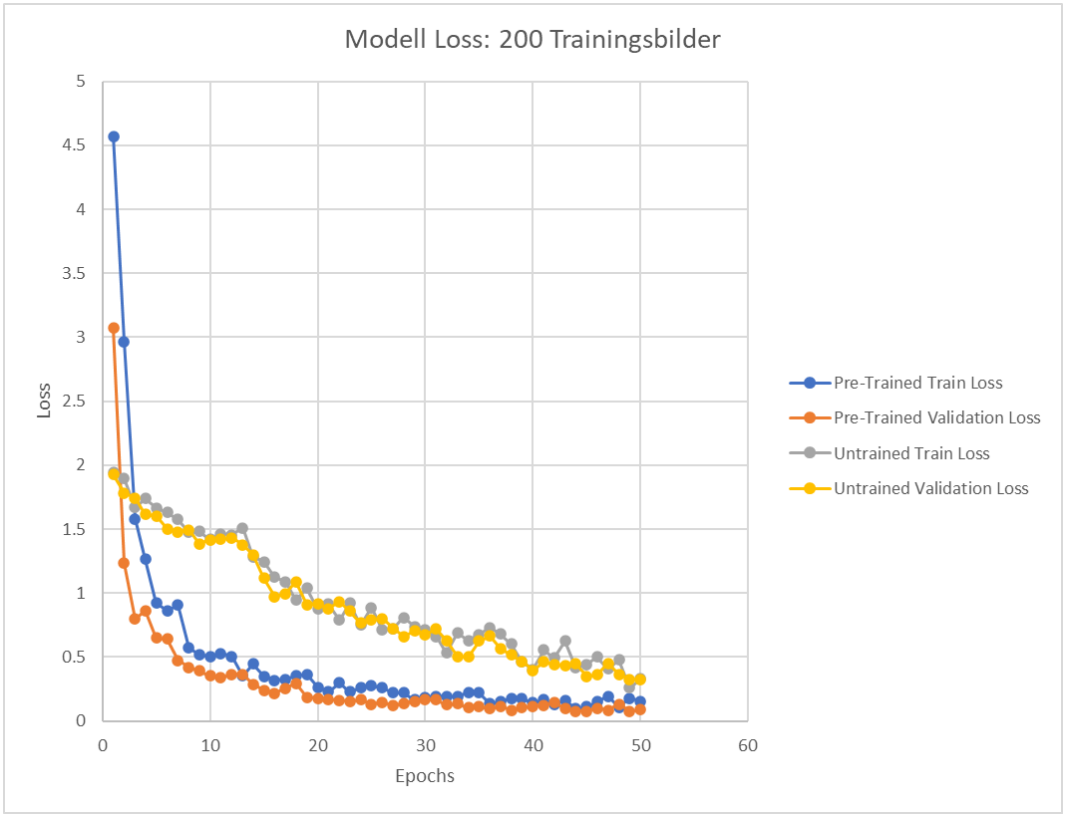

Abbildung 40: Modell Loss mit 200 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten.

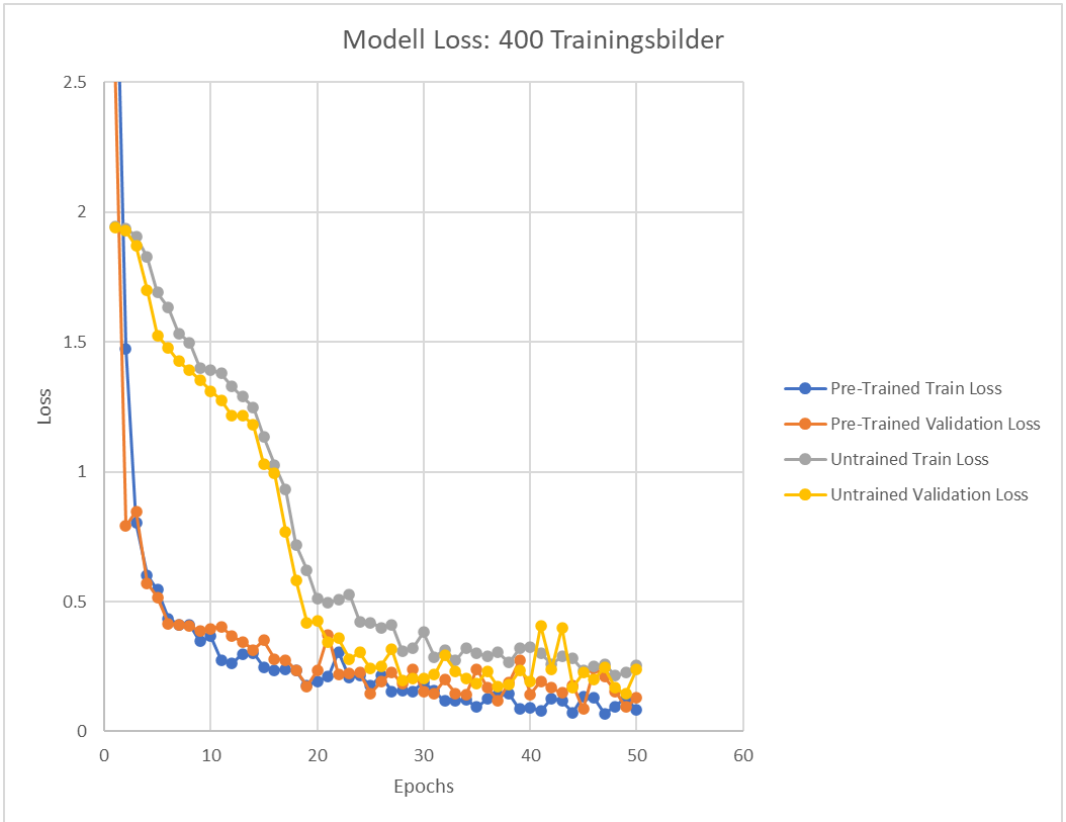

Abbildung 41: Modell Loss mit 400 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten.

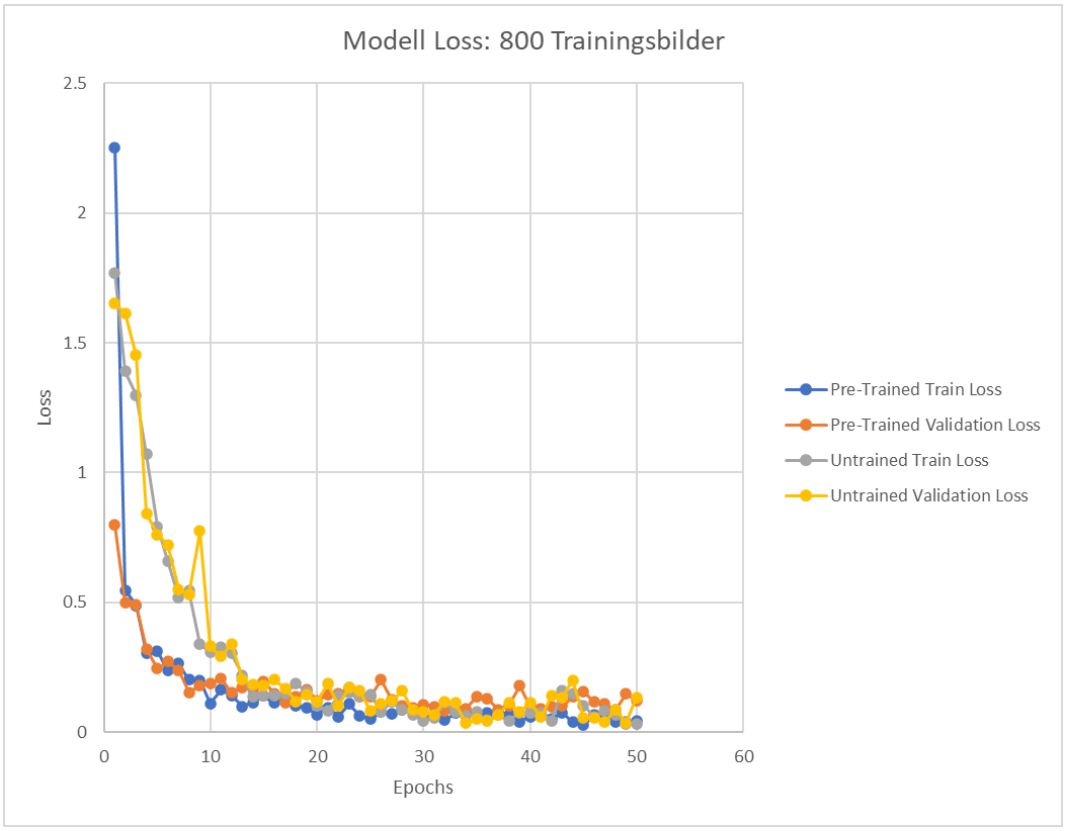

Abbildung 42: Modell Loss mit 800 Trainingsbildern. Pre-Trained = Imagenet Modell, Untrained = Modell mit randomisierten Gewichten.

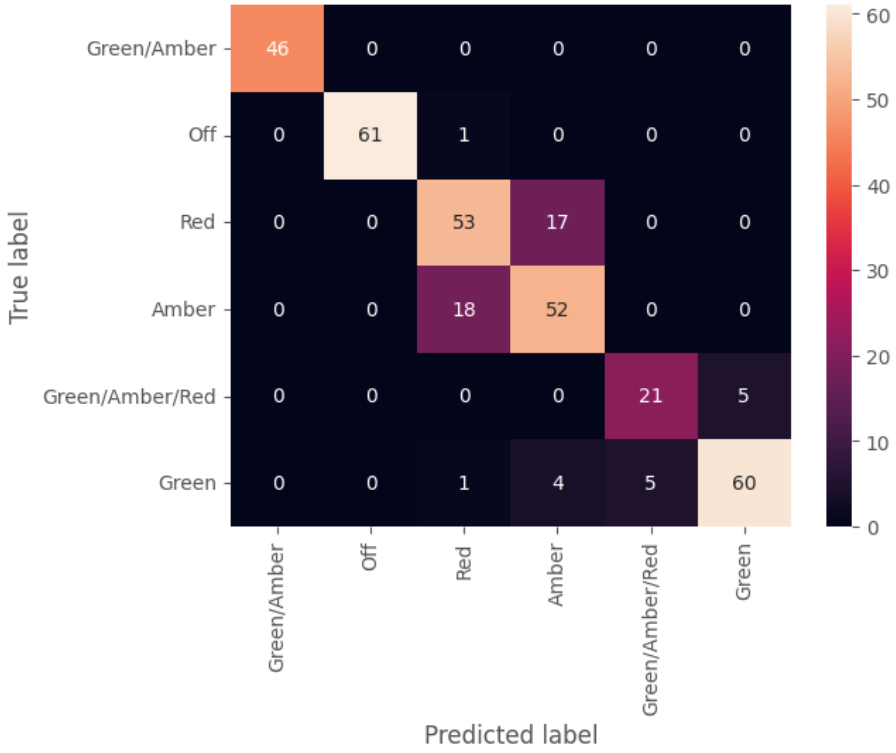

Abbildung 43: Konfusionsmatrix für Imagenet Modell trainiert mit 50 Trainingsbildern und getestet mit dem Testdatensatz.

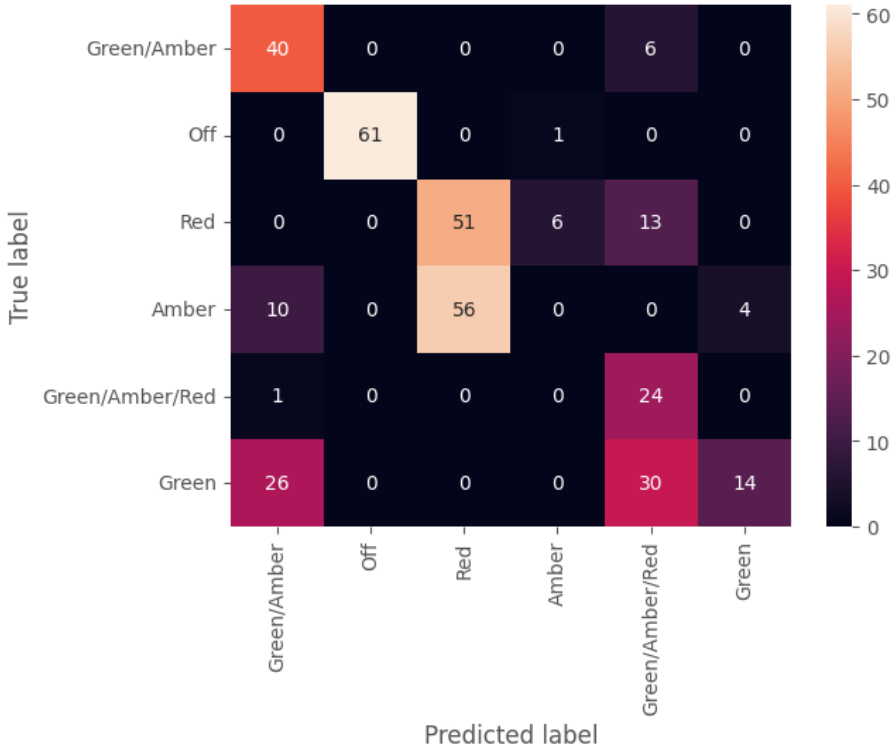

Abbildung 44: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 50 Trainingsbildern und getestet mit dem Testdatensatz.

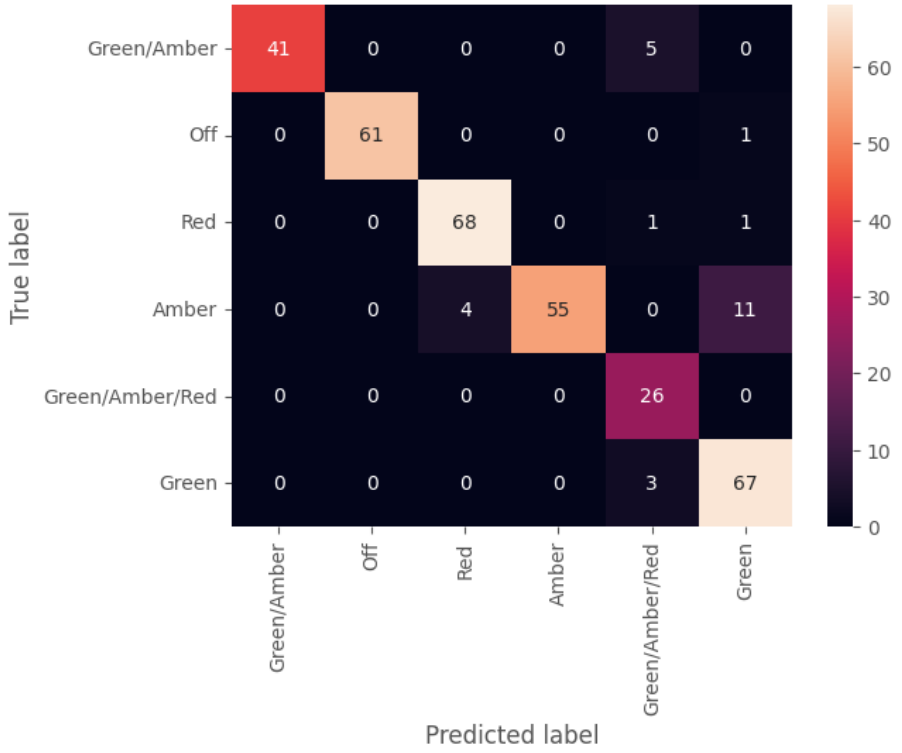

Abbildung 45: Konfusionsmatrix für Imagenet Modell trainiert mit 100 Trainingsbildern und getestet mit dem Testdatensatz.

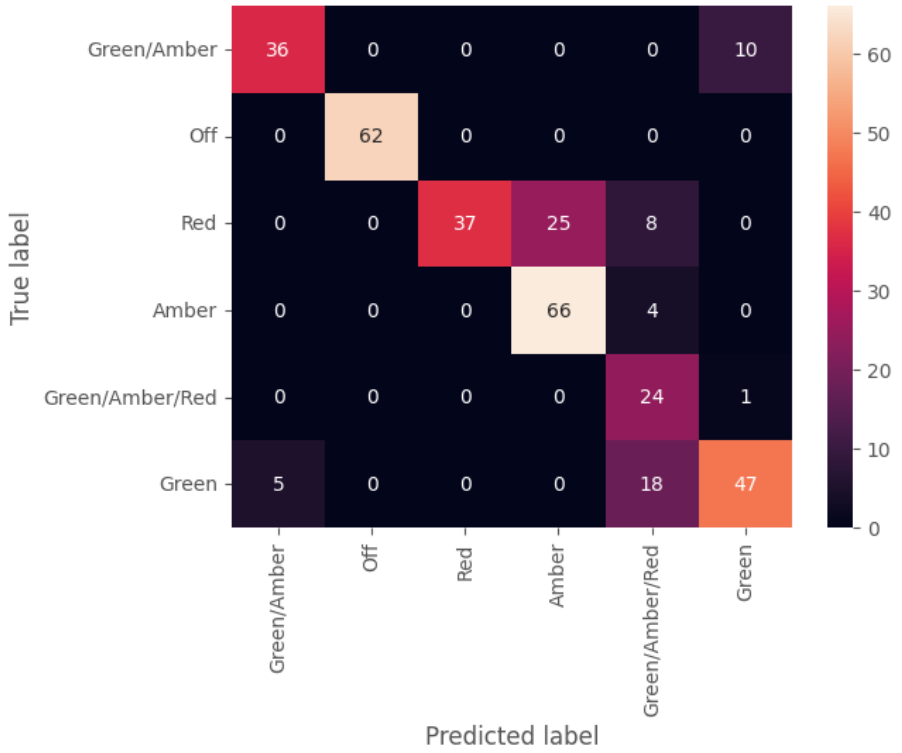

Abbildung 46: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 100 Trainingsbildern und getestet mit dem Testdatensatz.

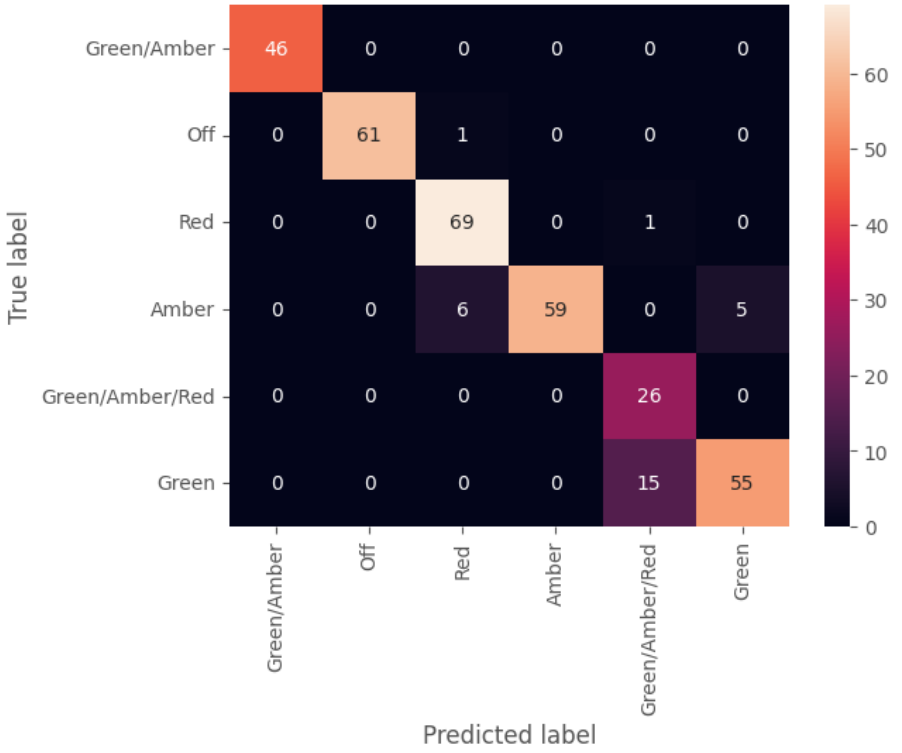

Abbildung 47: Konfusionsmatrix für Imagenet Modell trainiert mit 200 Trainingsbildern und getestet mit dem Testdatensatz.

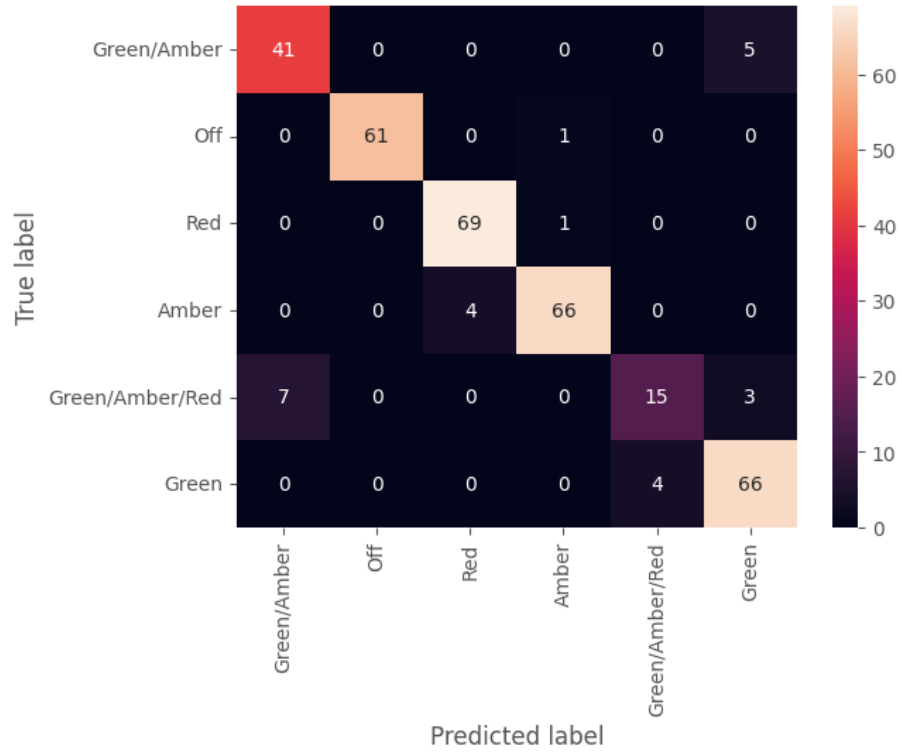

Abbildung 48: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 200 Trainingsbildern und getestet mit dem Testdatensatz.

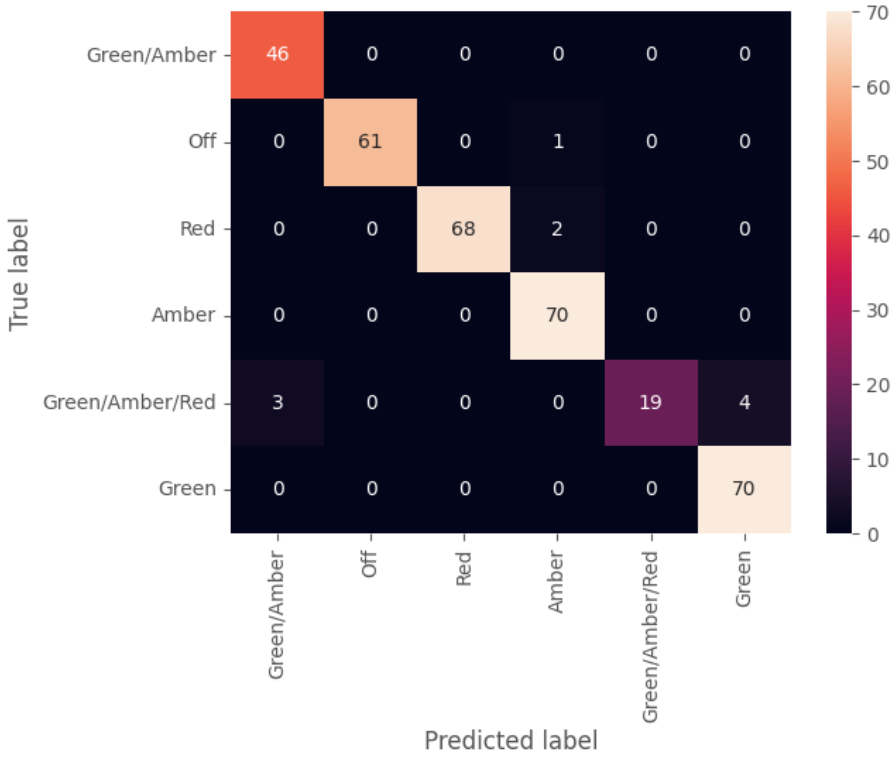

Abbildung 49: Konfusionsmatrix für Imagenet Modell trainiert mit 400 Trainingsbildern und getestet mit dem Testdatensatz.

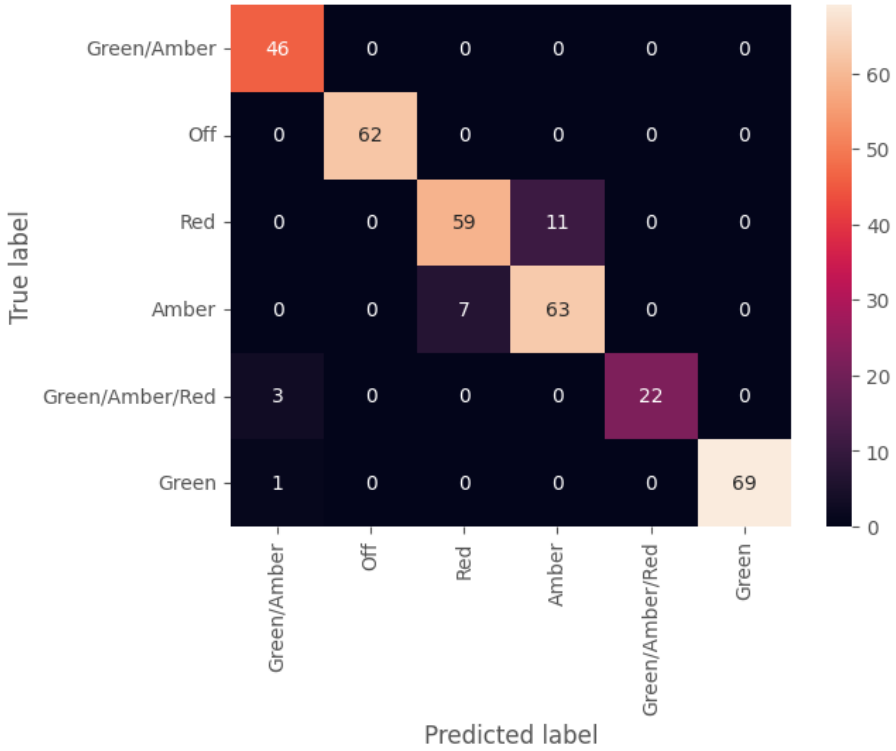

Abbildung 50: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 400 Trainingsbildern und getestet mit dem Testdatensatz.

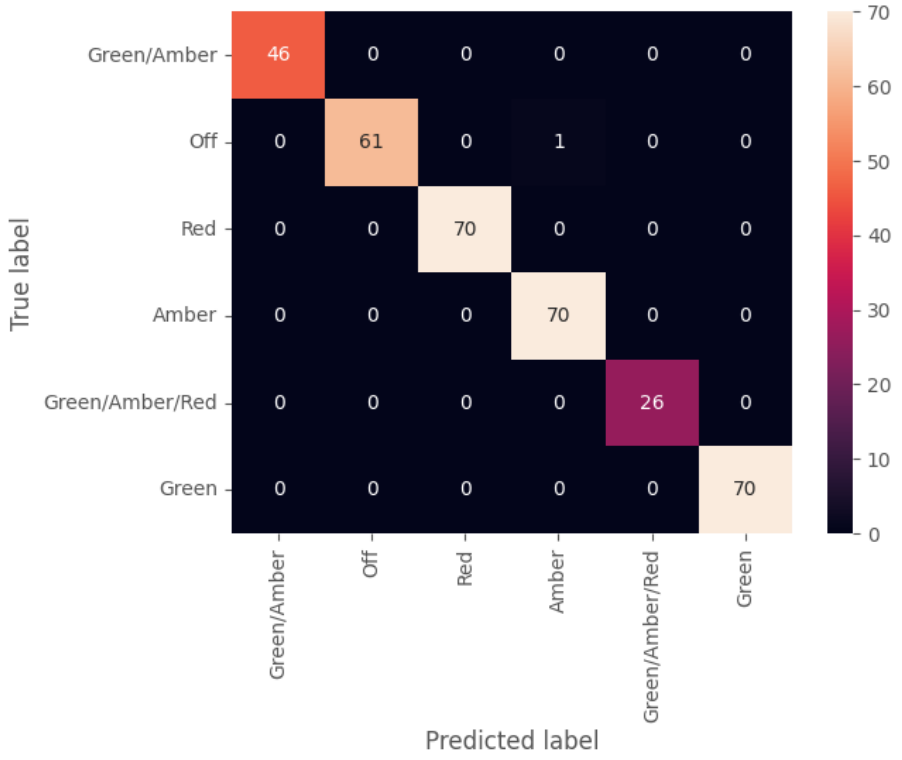

Abbildung 51: Konfusionsmatrix für Imagenet Modell trainiert mit 800 Trainingsbildern und getestet mit dem Testdatensatz.

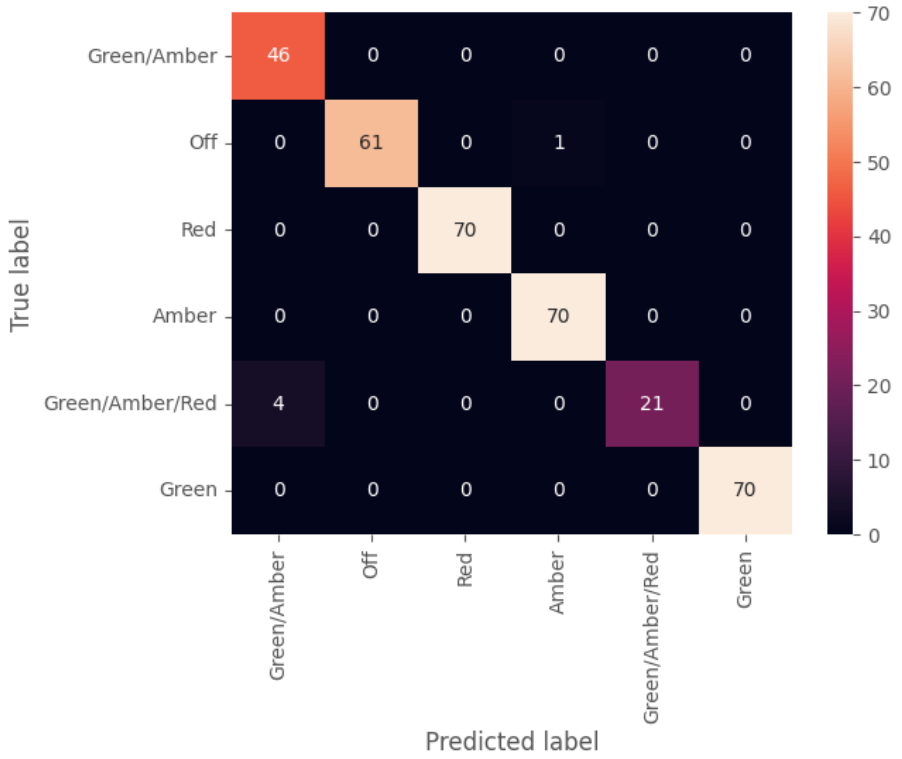

Abbildung 52: Konfusionsmatrix für Modell mit zufälligen Gewichten trainiert mit 800 Trainingsbildern und getestet mit dem Testdatensatz.

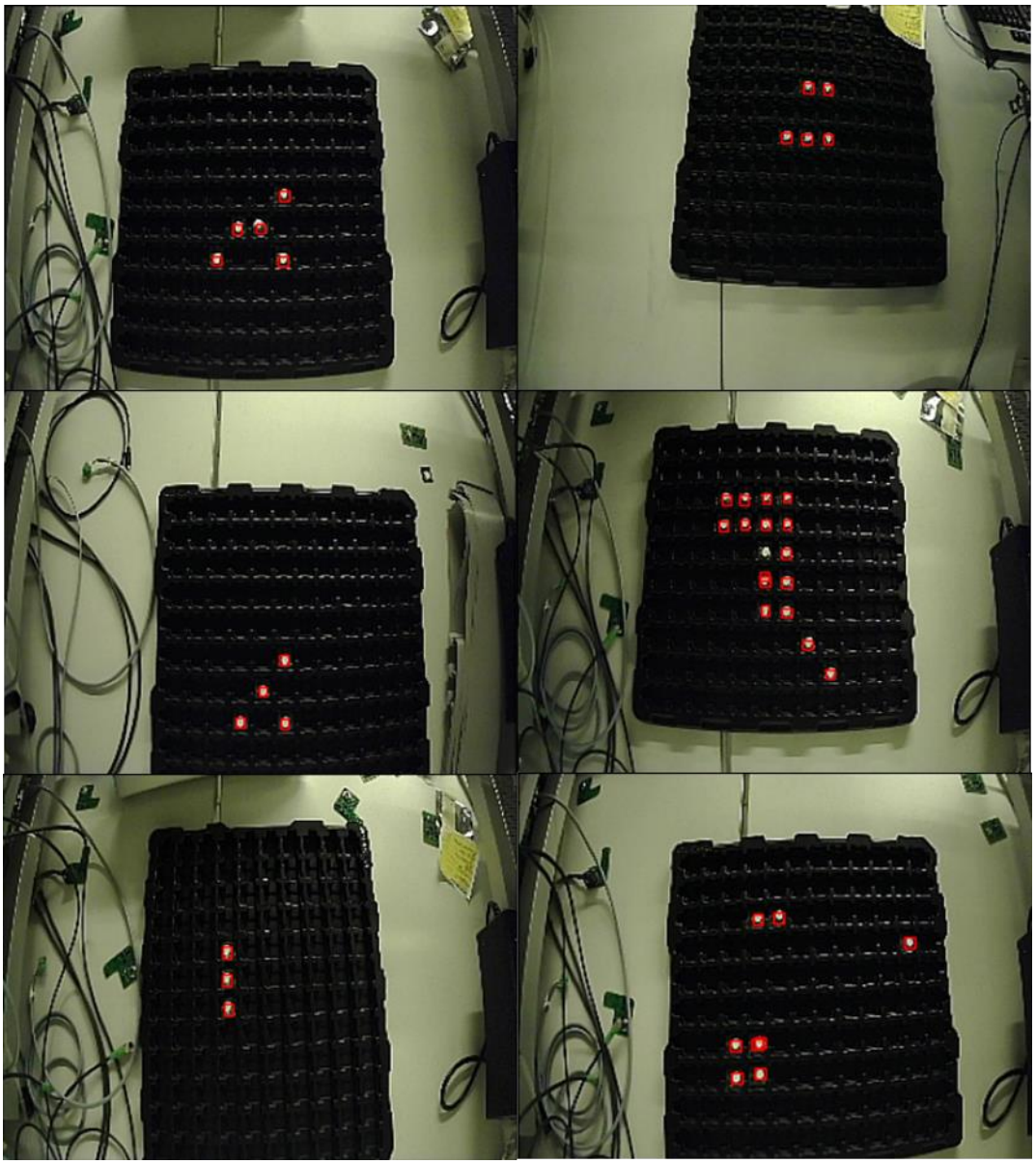

Abbildung 53: Modellvorhersagen Retinanet (trainiert auf 449 Bildern) auf den Testbildern.