HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Deep Learning mit TensorFlow

Automatisches erkennen von defekten Packungen

Florian Bitterlin, Cyrill Schenkel

Projektdokumentation
21. Dezember 2016

Autoren: Florian Bitterlin, Cyrill Schenkel
Betreuer: Prof. Hansjörg Huser
Projektpartner: Intigena Produktions AG, Mettmenstetten

Inhaltsverzeichnis

Abstract

Ausgangslage

In der Intigena Produktions AG werden Produktverpackungen, die von den Verpackungsstationen nicht korrekt verschlossen werden oder sonstige Schäden aufweisen von Hand aussortiert. Das manuelle Aussortieren verlangsamt den Produktionsprozess. Deshalb soll dieser Arbeitsschritt automatisiert werden.

Vorgehen

Dazu werden im Rahmen dieser Machbarkeitsstudie die genauen Anforderungen an eine Software für die Klassifizierung von Packungen in defekte und intakte, zusammengetragen und ein Prototyp erstellt, der diese Anforderungen so gut wie möglich erfüllt. Der Prototyp analysiert Aufnahmen der Packungen und verwendet einen Deep Learning Algorithmus für die Klassifizierung.

Ergebnis

Dieses Dokument präsentiert einen Prototyp einer solchen Software, der mittels TensorFlow und Inception v3 sehr nahe an die im ersten Teil erarbeiteten Anforderungen heran kommt. Der Prototyp kann 300 Packungen in der Minute klassifizieren. Damit wird die Mindestanforderung von 120 Packungen in der Minute übertroffen. Die geforderte Genauigkeit wird nicht vollständig erreicht. Insbesondere werden zuviele Packungen als defekt eingestuft.

Management Summary

Ausgangslage

Die Intigena Produktions AG stellt verschiedene Hygieneartikel her. Der gesamte Produktionsprozess, inklusive verpacken und verschweissen der Produkte in Plastikfolie, ist automatisiert. Da beim Verschweissen der Packungen Ausschuss (siehe: Abbildung 1a und Abbildung 1b) entstehen kann, werden diese heute von Hand kontrolliert.

Um ein höheres Produktionsvolumen erreichen zu können, soll die Überprüfung der Packungen nun ebenfalls automatisiert werden. Gleichzeitig soll wenn möglich auch die Genauigkeit erhöht werden. Im Rahmen dieses Projekt soll überprüft werden, ob sich mit Hilfe von Machine Learning und state of the art Image Processing Methoden dieses Problem lösen lässt.

Vorgehen/Technologien

Um sich für ein Lösungskonzept entscheiden zu können, wurden drei bekannte Frameworks ausgewählt und evaluiert. Es wurden mit VLFeat, OpenCV und TensorFlow erste Vorversuche gemacht um ihre Vorzüge und Nachteile zu erschliessen. Der Entscheid fiel schliesslich für TensorFlow von Google.

Genutzt wird ein bestehendes Modell, Inception v3, von TensorFlow welches darauf trainiert wird, den Inhalt eines Bildes zu erkennen. Dieses Modell wird so umtrainiert, dass es im Stande ist intakte und defekte Packungen zu erkennen. Im Folgenden wird versucht, über Parameteranpassungen und Optimierungen an den verwendeten Bildern, die erzielten Resultate zu verbessern.

Ergebnisse

Mit dieser Machbarkeitsstudie wird aufgezeigt, dass es möglich ist die Überprüfung der Packungen zu automatisieren. Das Ergebnis der Arbeit ist ein Prototyp, welcher mit Bildern von Packungen trainiert werden kann und in der Lage ist, dass trainierte Modell zu testen. Die erreichte Genauigkeit des Prototypen kommt sehr nahe an die geforderten Werte für den produktiven Einsatz heran

(a) Offensichtlicher Defekt (b) Intakt

Abbildung 1: Beispiel defekte und intakte Packungen

und kann noch weiter optimiert werden. Die geforderte Geschwindigkeit wurde bereits mit einem etwas besseren Desktoprechner um die Hälfte unterboten.

Ausblick

Der erstellte Prototyp ist nicht für den Einsatz im produktiven Umfeld gedacht. Vielmehr soll er als Grundlage dienen für Tests von weiteren Optimierungen.

Teil I

Projektplan

Kapitel 1

Einleitung

1.1 Zweck

Dieses Dokument beschreibt die Planung der Studienarbeit *Automatisches Erkennen von defekten Paketen,* welche an der Hochschule für Technik Rapperswil (HSR) im Herbstsemester 2016 von den Autoren durchgeführt wird.

1.2 Gültigkeitsbereich

Dieses Dokument gilt während der gesamten Dauer des Projektes vom Datum seiner Erstellung bis zum Abschluss des Projektes im Verlaufe der letzten Semesterwoche des Herbstsemesters 2016. Inhaltlich beschränken sich die nachfolgenden Ausführungen auf die Planung der Studienarbeit.

1.3 Referenzen

Die Referenzen für dieses Dokument finden sich in den separaten Verzeichnissen, dem “Abbildungsverzeichnis”, dem “Abkürzungsverzeichnis” und dem “Literaturverzeichnis” im Anhang dieses Dokumentes.

Kapitel 2

Projektübersicht

Im Rahmen der Studienarbeit an der HSR wird, in Zusammenarbeit mit der Intigena Produktions AG, eine Lösung zur automatischen Erkennung von defekten Paketen entwickelt.

Das Produkt soll in der automatisierten Produktionsstrasse für Inkontinenzeinlagen für Damen eingesetzt werden. Die einzelnen Einlagen werden dabei in Plastikpakete abgefüllt und verschweisst. Beim Schweissen entsteht Ausschuss, welcher heute manuell aussortiert wird. Dieser Schritt soll vollständig automatisiert werden.

2.1 Zweck und Ziel

Mittels Bilderkennung soll die Verpackung geprüft werden, dass keine Risse oder beschädigte Stellen sichtbar sind. Im Weiteren soll auch geprüft werden ob die Beutel richtig verschlossen sind. Um die Produktionsstrasse nicht aufzuhalten, muss die Überprüfung einer einzelnen Verpackung innerhalb einer bestimmten Zeitspanne erfolgen können. Bei einem Volumen von ca. 120 Verpackungen/-min müsste eine Überprüfung in ca. 0.5s erfolgen können.

Das Ziel des Projekts ist es einen Prototypen zu entwickeln, welcher defekte Pakete auf Fotos erkennt. Vorgängig werden verschiedene Verfahren und Bibliotheken zur Bildverarbeitung und Machine Learning evaluiert und der Prototypen aufgrund der daraus gewonnenen Erfahrungen aufgebaut.

2.2 Lieferumfang

Der Lieferumfang der Studienarbeit umfasst:

- Lauffähiger SW-Prototyp welcher das evaluierte Verfahren umsetzt
- Quellcode des Prototypen inkl. Build-Scripts
- Benutzerdokumentation des Prototypen
- Studienarbeitsbericht nach Vorgaben der HSR
- Abstract zusätzlich im Online-Tool der HSR erfasst
- Poster in elektronischer Form

2.3 Annahmen und Einschränkungen

Zum jetzigen Zeitpunkt sind uns keine weiteren Einschränkungen als jene, welche uns bereits durch die Aufgabenstellung gegeben wurden, bekannt.

Kapitel 3

Projektorganisation

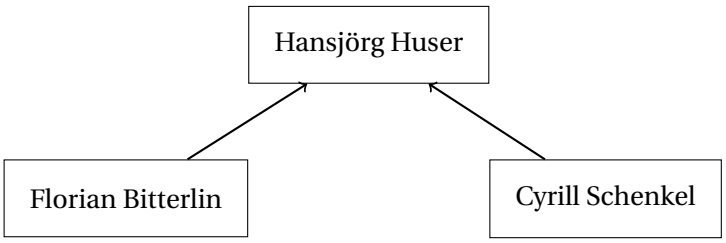


Abbildung 3.1: Organigramm

Name	**Rolle**	**E-Mail**
Hansjörg Huser	Betreuer	hansjoerg.huser@hsr.ch
Florian Bitterlin	Student	florian.bitterlin@hsr.ch
Cyrill Schenkel	Student	cyrill.schenkel@hsr.ch

Tabelle 3.1: Rollen

Kapitel 4

Zeitliche Planung

Im Zeitplan werden alle Phasen, Iterationen und Meilensteine beschrieben. Als Übersicht dient die Abbildung 4.1.

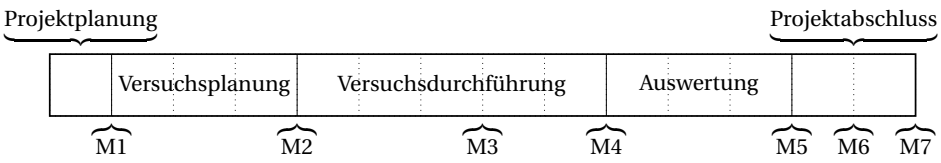


Abbildung 4.1: Phasen und Iterationen Plan

4.1 Phasen

Es folgen Beschreibungen aller Projektphasen.

4.1.1 Projektplanung

In der Projektplanungsphase wird dieses Dokument verfasst und die Infrastruktur, die für das Projekt gebraucht wird vorbereitet.

4.1.2 Versuchsplanung

In dieser Phasen muss die Problemstellung analysiert und Lösungsansätze formuliert werden.

Die Problemstellung soll zuerst in funktionale und nicht-funktionale Anforderungen umformuliert werden. Sind diese Anforderungen korrekt formuliert, kann bei der Auswertungs-Phase eine klare Aussage gemacht werden, wie gut das Erreichte mit den im Voraus festgelegten Anforderungen übereinstimmt und welche Anforderungen nicht umgesetzt werden konnten.

Drei Lösungsansätze sollen formuliert und verfolgt werden. Am Ende dieser Phase muss feststehen, welcher der drei geprüften Lösungsansätze am vielversprechendsten ist und in den folgenden Phasen weiterbearbeitet wird.

4.1.3 Versuchsdurchführung

In dieser Phase soll der vielversprechendste der zuvor geprüften Lösungsansätze weiterverfolgt werden.

4.1.4 Auswertung

In dieser Phase soll ausgewertet werden, wie gut die Anforderungen erreicht wurden und wo die grössten Probleme lagen. Das Ziel der Auswertung ist, Aussagen darüber machen zu können, was bei einer allfälligen Implementation für den Einsatz im realen Produktionsprozess beachtet werden müsste.

4.1.5 Projektabschluss

In dieser letzten Phase soll das Projekt auf alle formalen Anforderungen an die Arbeit geprüft und die Präsentation vorbereitet werden.

4.2 Iterationen

Eine Iteration dauert jeweils eine Woche. Die Einteilung von Iterationen auf die Projektphasen können der Abbildung 4.1 entnommen werden.

Zu Beginn jeder Iteration, wird geplant, welche Arbeit während dieser erledigt werden soll. Die Iterationen werden jeweils am Montag morgen geplant.

4.3 Meilensteine

Im folgenden werden alle Meilensteine aufgeführt. Sie sind terminiert und mit einer Beschreibung versehen.

4.3.1 M1: Projektplan

Datum:
23. September

Beschreibung:
Der Projektplan ist fertiggestellt.

4.3.2 M2: Versuchsplanung abgeschlossen

Datum:
14. Oktober

Beschreibung:
Alle Anforderungen wurden ausformuliert und ein Lösungsansatz, der weiterverfolgt werden soll, wurde evaluiert. Zu diesem Zeitpunkt sollten auch alle Notwendigen Daten für den Versuch vorliegen (Lerndaten etc.)

4.3.3 M3: Zwischenresultat (erster Prototyp)

Datum:
4. November

Beschreibung:
Es existiert ein Prototyp, der dem gewählten Lösungsansatz entspricht und bis zum Ende der Projektphase verfeinert werden kann. Der Prototyp muss in dieser Phase noch nicht alle Anforderungen erfüllen, sondern bloss den Weg zu deren Erfüllung aufzeigen, bzw. zeigen, welche Anforderungen schwierig zu erreichen sind.

4.3.4 M4: Versuchsdurchführung abgeschlossen

Datum:
18. November

Beschreibung:
Die Versuchsdurchführung ist abgeschlossen, alle Qualitätskriterien für den Versuch (Programmierrichtlinien, Dokumentation, etc.) und die Daten für die folgende Auswertung liegen vor.

4.3.5 M5: Auswertung abgeschlossen

Datum:
9. Dezember

Beschreibung:
Der Versuch wurde vollständig Ausgewertet und es wurden Schlüsse daraus gezogen.

4.3.6 M6: Abgabe I (Systemtest abgeschlossen)

Datum:
16. Dezember

Beschreibung:
Das Abstract ist fertig, das Poster ist fertig und die Grundlegenden Inhalte der Präsentation wurden festgelegt.

4.3.7 M7: Abgabe II (Abgabe des Berichts)

Datum:
23. Dezember

Beschreibung:
Der Bericht ist fertiggestellt und die Arbeitsplätze in den Ursprungszustand zurückversetzt.

4.4 Meetings

Jeden Montag Morgen um 10:10 gibt es ein Meeting bei dem die letzte Iteration nachbesprochen und die neue geplant wird.

Meetingstermin mit Betreuer unklar. Folgt...

Kapitel 5

Risikomanagement

Es folgt eine Einschätzung und der geplante Umgang mit den Risiken, die während des Projekts auftreten könnten.

5.1 Risikoanalyse

In den folgenden Abschnitten, werden die wichtigsten Risiken beschrieben eingeschätzt und Massnahmen zur Vorbeugung, sowie zur Schadensbewältigung definiert.

5.1.1 Zu wenige Lerndaten

Beschreibung

Zu wenige Lerndaten um Funktionstüchtigkeit des Algorithmus zu beweisen.

Schaden

max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden
16	70%	11.2

Vorbeugung

Erfrühte Anfrage an Intigena nach Beispielen oder Beispielbilder.

Verhalten beim Eintreten

Dokumentation wieso die Funktionstüchtigkeit des Algorithmus nicht genügend bewiesen werden konnte.

5.1.2 Zu geringe Qualität der Bilder

Beschreibung

Bilder haben zu geringe Qualität(Auflösung, Ausleuchtung) um dem Anwendungsfall zu genügen.

Schaden

max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden
4	40%	1.6

Vorbeugung

Genaue Definition was die Anforderungen an die benötigten Bilder sind.

Verhalten beim Eintreten

Neue Bilder werden anhand der ausgehändigten Beispiel-Packungen neu erstellt.

5.1.3 Bilder zeigen nötige Informationen nicht

Beschreibung

Bilder zeigen nicht nötige Informationen. Es ist z.B. nicht erkennbar ob Schweissnähte zusammen oder Löcher darunter vorhanden sind.

Schaden

max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden
8	30%	2.4

Vorbeugung

Es sind keine Vorbeugungsmassnahmen möglich.

Verhalten beim Eintreten

Dokumentation der Möglichkeit, dass dieser Fall aufgrund der gelieferten Lerndaten eintreten kann.

5.1.4 Evaluation der Libraries

Beschreibung

Evaluation der Libraries benötigt länger als geplant.

Schaden

max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden
32	60%	19.2

Vorbeugung

Es sind keine Vorbeugungsmassnahmen möglich.

Verhalten beim Eintreten

Die zeitliche Planung muss angepasst werden. M2 - Versuchsplanung wird nach hinten verschoben.

5.1.5 Unvorhergesehene Schwierigkeiten mit der Library

Beschreibung

Bugs oder anderweiteige Schwierigkeiten in der gewählten Library erschweren die Umsetzung.

Schaden

max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden
16	40%	6.4

Vorbeugung

Es sind keine Vorbeugungsmassnahmen möglich.

Verhalten beim Eintreten

Ein Workaround wird gesucht. Bei eintreten in genügend früher Phase der Umsetzung kann auch ein Ersetzen der Library in betracht gezogen werden.

5.2 Umgang mit Risiken

Die aufgeführten Risiken können grob in zwei Kategorien unterteilt werden. Manche der Risiken können durch technische Massnahmen und definierte Vorgehen bei der Entwicklung mitigiert werden. Andere Punkte sind zum Beispiel Teil eines externen Systems und befinden sich deshalb ausserhalb unserer Systemgrenze. Auf sie kann nur begrenzt Einfluss genommen werden. Als Beispiel wäre hier die verwendete Hardware zu nennen.

Innerhalb der Systemgrenze und damit in der Verantwortung des Projekts:

- Permanente Qualitätsprüfung während der Entwicklung durch Code Reviews, Unit Testing, statische Codeanalyse und Issue Tracking.
- Installationsanleitung mit Empfehlungen bezüglich Qualität der verwendeten Bilder und eingesetzten Hardware.

Kapitel 6

Arbeitspakete

Die folgenden Arbeitspakete konnten bis zu diesem Zeitpunkt ausgemacht werden.

6.1 M2: Versuchsplanung Abgeschlossen

Die Folgenden Arbeitspakete dienen der Vorbereitung für den Versuch. Der Milestone ist erreicht, sobald mit der Umsetzung der Lösung begonnen werden kann.

6.1.1 Ausgangslage (Kontext) beschreiben

Gemäss dem Dokument *Anleitung: Dokumentation Studien- und Bachelorarbeiten* soll folgendes beschrieben werden:

- "Beschreibung des Typs der Arbeit (Bsp. Fokus Lösungserstellung oder Machbarkeitsanalyse)"
- "Fachliche Domäne, Zielgruppe, heutige Praktiken bzw. Lösungen (Methoden, Tools, etc.)"

6.1.2 Motivation beschreiben

Gemäss dem Dokument *Anleitung: Dokumentation Studien- und Bachelorarbeiten* soll folgendes beschrieben werden: "Motivation für die Arbeit, z.B. aus den Schwächen der heutigen Praktiken bzw. Lösungen". Dafür ist es notwendig, dass die Ausgangslage bereits bekannt ist.

6.1.3 Funktionale Anforderungen definieren

Alle funktionalen Anforderungen müssen ausreichend definiert werden. Die Anforderungen müssen die Genauigkeit des Klassifizierers, sowie die Klassifizierungskriterien präzise spezifizieren.

6.1.4 Nicht Funktionale Anforderungen definieren

Alle NFA müssen überprüfbar sein und mit der Aufgabenstellung übereinstimmen.

6.1.5 3 Lösungsansätze entwerfen

Die Lösungsansätze sollen verschiedene Pfade zur Lösung des Problems erkunden. Das schliesst sowohl verschiedene Technologien als auch Abweichungen von der Ausgangslage, die mit dem Betreuer besprochen wurden, ein.

6.1.6 Lösungsansätze evaluieren

Die Lösungsansätze werden miteinander verglichen und ihre Machbarkeit erkundet. Dazu müssen die Verwendeten Technologien teilweise ausprobiert werden. Der vielversprechendste Lösungsansatz wird dann ausgewählt und die Auswahl gut begründet.

6.1.7 Voraussetzungen für die Umsetzung definieren

Jeder Lösungsansatz wird zur Umsetzung eine gewisse Menge und Form von Inputdaten und evtl. spezielles Equipment (Grafikkarten, Cluster, etc.) benötigen. Da nun der umzusetzende Lösungsansatz feststeht, soll definiert werden, welche Daten und welches Equipment dazu notwendig sind.

6.1.8 Umsetzung Planen (Architektur, etc.)

Es soll eine Architektur für die Umsetzung des gewählten Lösungsansatzes erstellt werden.

6.1.9 Build für Dokumente aufsetzen

Auf dem Buildserver muss ein Build konfiguriert werden, der als Reaktion auf neue Commits die geänderten Dokumente neu buildet und für alle am Projekt beteiligten zugänglich macht.

6.2 M3: Zwischenresultat

Das Zwischenresultat ist ein Prototyp des gewählten Lösungsansatzes. Dem Zwischenresultat soll abgelesen werden können, ob die Problemstellung mit der gegebenen Ausgangslage überhaupt lösbar ist. Dieser Milestone ist entscheidend dafür, wie die Arbeit weiter verlaufen wird.

6.2.1 Projekt für Lösung Aufsetzen

Das Softwareprojekt für die Lösung soll eingerichtet werden. Dabei muss dokumentiert werden, wie das Projekt kompiliert und wie die Entwicklungsumgebung eingerichtet werden kann. Dazu gehört auch das Code Repository zu erstellen.

6.2.2 Build für Lösung Aufsetzen

Zum Aufsetzen des Builds für das Softwareprojekt gehört auch noch das Konfigurieren der statischen Code Analyse.

6.2.3 Minimales User Interface erstellen

Es muss ein CLI oder ein GUI erstellt werden. Die Entscheidung zwischen CLI und GUI hängt von den Anforderungen ab. Fordern die Anforderungen ein GUI, wird dies umgesetzt, sonst gibt es ein CLI.

6.2.4 Vorverarbeitung der Bilder implementieren

Die Bilder müssen in ein Format umgewandelt werden, dass sie von der verwendeten ML Library verarbeitet werden kann. Dabei soll die Möglichkeit offengelassen werden, die Bilder in der Optimierungsphase zu einem späteren Zeitpunkt zu verändern.

6.2.5 Parameter wählen

Für die Klassifizierung müssen Klassen sowie Parameter, anhand derer klassifiziert werden soll, festgelegt werden.

6.2.6 Abhängigkeit zwischen Definitions- und Zielmenge beweisen

Sobald das ML Modell fertig konfiguriert ist, können Tests gemacht werden, um eine Abhängigkeit oder Unabhängigkeit zwischen gewählten Parametern und der Klassenzugehörigkeit festzustellen. Wenn keine Abhängigkeit festgestellt werden kann, auch nachdem die Parameterwahl verändert wurde, muss davon ausgegangen werden, dass keine Abhängigkeit zwischen Definitions- und Zielmenge besteht. Das gilt es allerdings so gut wie möglich zu beweisen.

Es gibt daher zwei Szenarien. Szenario I tritt ein, wenn eine Abhängigkeit festgestellt werden kann. Szenario II tritt ein, wenn keine Abhängigkeit festgestellt werden konnte.

6.2.7 Schwierigkeiten bei der Umsetzung erläutern

Zu diesem Zeitpunkt können Aussagen darüber gemacht werden, wo die grössten Schwierigkeiten bei der Umsetzung liegen. Diese sollen dokumentiert werden.

6.3 M4: Versuchsdurchführung abgeschlossen (Szenario I)

Im Falle einer Abhängigkeit zwischen Definitions- und Zielmenge, muss die Lösung zunächst verfeinert werden und dann die Auswertung des Versuchs vorbereitet werden.

6.3.1 Performance Tests aufsetzen

Die Performance Tests sind die Basis für die Beurteilung einer der wichtigen Fragestellungen aus dem Projektantrag. Sie erlauben einerseits Aussagen über die Performance zu machen und andererseits die Performance gezielt zu erhöhen.

6.3.2 Performance optimieren

Aufgrund der Performance Tests soll die Lösung optimiert werden.

Vorverarbeitung erweitern

Eine Schraube für die Optimierung ist die Vorverarbeitung der Inputdaten.

Modell optimieren

Eine weitere Optimierungsmassnahme ist das tuning der Parameter des ML Modells.

Weitere Optimierungen

Falls andere Performanceprobleme gefunden werden, sollen diese ebenfalls behoben werden.

6.3.3 Reports für die Auswertung implementieren

Für die Auswertung der Versuchsdaten, ist es hilfreich, wenn die benötigten Daten in einem sinnvollen Format direkt aus der Applikation exportiert werden können.

6.3.4 Benutzerdokumentation schreiben

Das Benutzerinterface muss dokumentiert werden, sodass die Resultate einfach nachvollziehbar sind.

6.4 M4: Versuchsdurchführung abgeschlossen (Szenario II)

Besteht keine Abhängigkeit zwischen Definitions- und Zielmenge, muss exakt beschrieben werden, wie dieses Resultat erreicht wurde und alternative Lösungswege erkundet werden.

6.4.1 Beweis sauber erklären (Reproduzierbarkeit)

Der Beweis für die Unabhängigkeit zwischen Definitions- und Zielmenge muss von der Leserin bzw. vom Leser einfach reproduziert werden können.

6.4.2 Fehler suchen

Um alternative Lösungswege zu finden, muss zuerst analysiert werden, wieso die gewählte Lösung nicht funktioniert hat.

6.4.3 Alternative Lösungswege diskutieren

Zum Abschluss sollen andere Lösungswege gesucht und beschrieben werden.

6.5 M5: Auswertung abgeschlossen

Nach der Durchführung des Versuchs müssen die gesammelten Daten ausgewertet werden.

6.5.1 Alle gesammelten Daten auswerten

Die Daten müssen gegen die Anforderungen ausgewertet und nach Bedarf visualisiert werden.

6.5.2 Die Auswertung in den Kontext der Ausgangssituation stellen

Die Auswertung der Daten muss in den Kontext zur Ausgangssituation gestellt werden um Aussagen über die Geschwindigkeits- und Genauigkeitsunterschiede machen zu können.

6.5.3 Schlüsse für die Zukunft ziehen

Zuletzt sollen Schlüsse für zukünftige Projekte gezogen werden.

6.6 M6: Abgabe I

Der Inhaltliche Teil der Arbeit sollte zu diesem Zeitpunkt beinahe abgeschlossen sein. Es geht nun darum die Formalen Anforderungen an die Arbeit zu erfüllen und die Abgaben zu machen.

6.6.1 Abstract schreiben

Siehe entsprechende Vorlagen.

6.6.2 Poster gestalten

Siehe Vorgaben.

6.6.3 Outline der Präsentation erstellen

Der Ablauf der Präsentation sollte zu diesem Zeitpunkt bereits festgelegt werden. Auch die Kernbotschaft soll feststehen.

6.6.4 Abstract und Poster abgeben

Das Abstract und das Poster müssen fristgerecht abgegeben werden.

6.7 M7: Abgabe II

In der letzten Woche muss der Inhalt abgeschlossen sein und die Arbeit an die formalen Anforderungen angepasst werden.

6.7.1 Präsentation fertigstellen

Die Präsentation muss vor dem Präsentationstermin abgeschlossen werden.

6.7.2 Bericht finalisieren

Layoutprobleme müssen behoben werden und die Arbeit muss auf Schreibfehler kontrolliert werden.

6.7.3 Bericht abgeben

Der Bericht muss fristgerecht abgegeben werden.

6.7.4 Arbeitsplätze in Ursprungszustand zurückversetzen

Die Arbeitsplätze müssen gemäss der Vorgabe in den Ausgangszustand zurückversetzt werden.

Kapitel 7

Infrastruktur

Beide Teammitglieder verwenden ihre privaten Computer sowie die von der HSR zur Verfügung gestellten Workstations im Arbeitszimmer an der Schule für die Arbeit am Projekt. Für die verwendeten Server-Tools wird die private Server-Infrastruktur von Florian Bitterlin zur Verfügung gestellt. Für den Buildserver wird eine VM (SE2-BuildServer-Image) von der Schule zur Verfügung gestellt.

7.1 Übersicht der Tools

Die Verwendeten Tools sind unten in zwei Kategorien geteilt aufgeführt. Die *Server-Tools* werden im Multiuserbetrieb verwendet während die Entwicklungsumgebung lokal auf den Arbeitsmaschinen der Projektteilnehmer liegen.

7.1.1 Server-Tools

Folgende Tabelle listet die eingesetzten Tools auf. Zugangsdaten können bei den entsprechenden Administratoren angefragt werden.

Bezeichnung	**Version**	**Beschreibung**
GitLab CE / Git	8.*	Versionsverwaltungstool mit Weboberfläche für die Benutzerverwaltung und Codereviews. *Link:* [redacted] *Administrator:* Florian Bitterlin *Offizielle Webseite:* https://about.gitlab.com/ https://git-scm.com/
NextCloud	10.*	Filehosting für grössere Datenmengen *Link:* [redacted] *Administrator:* Florian Bitterlin *Offizielle Webseite:* http://www.nextcloud.com/
Jenkins	2.*	Buildserver und Continuous Integration *Link:* [redacted] *Administrator:* Cyrill Schenkel *Offizielle Webseite:* https://jenkins.io/

Tabelle 7.2: Server-Tools

7.1.2 Entwicklungsumgebung

Folgende Tabelle listet die eingesetzten Entwicklungsumgebungen und anderweitig eingesetzte Software auf.

Bezeichnung	Version	Beschreibung
Eclipse	Neon	Entwicklungsumgebung *Offizielle Webseite:* https://eclipse.org/
Emacs	25.1	Entwicklungsumgebung *Offizielle Webseite:* https://www.gnu.org/s/emacs/
Texlipse	1.5	LaTeX Editor Plugin for Eclipse *Offizielle Webseite:* http://texlipse.sourceforge.net/index.php
PyDev	5.3	Python Development Environment Plugin for Eclipse *Offizielle Webseite:* http://www.pydev.org/
LaTeX	2ε	Dokumentation *Offizielle Webseite:* http://www.latex-project.org/

Tabelle 7.4: Allgemeine Software

7.2 Dokumentation

Die Dokumentation des Projekts wird mittels Latex gehandelt. Dadurch können sämtliche Dokumente des Projekts per Git verwaltet werden und einzelne Versionen der Projektmitglieder bei Bedarf einfach zusammengelegt werden.

7.3 Kommunikation

Als Kommunikationsmittel sollen Telegram, E-Mail und Telefon verwendet werden. Die Reaktion der Teammitglieder soll innert 24 Stunden, ausgenommen ist der Sonntag, erfolgen.

Kapitel 8

Qualitätsmassnahmen

Alle Arbeiten von jedem Teammitglied, werden vom jeweils Anderen gereviewt. Diese Massnahmen gilt für alle Bereiche der Arbeit.

8.1 Dokumentation

Die gesamte Dokumentation wird mit LaTeX geschrieben und in einem Git Repository versioniert.

Alle Änderungen an den Dokumenten werden vom jeweils Anderen Teammitglied gereviewt. Darüber hinaus, werden alle Dokumente mit einem Spellchecker geprüft.

8.2 Projektmanagement

Für die Aufgabenverwaltung wird ein Flipchart und Post-it Zettel verwendet. Es werden nur Tasks erfasst, die für die Koordinierung der Teamarbeit notwendig sind. Nicht erfasst, werden beispielsweise das Nachführen des Projekttagebuchs, die Erfassung neuer Tasks oder Meetings.

In Abbildung 8.1 ist zu sehen, wie das Flipchart eingeteilt wurde.

Open	FB	In Progress	Resolved
Reopened FB CS	CS		

Abbildung 8.1: Taskboard

In Abbildung 8.2 wird das Layout der einzelnen Tasks gezeigt.

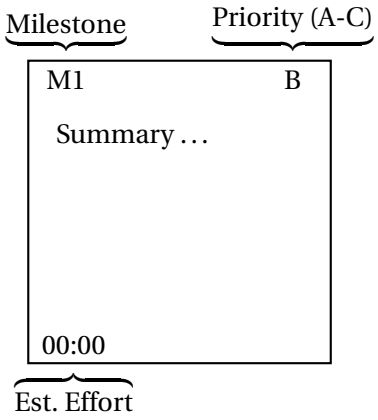


Abbildung 8.2: Task Layout

8.3 Entwicklung

8.3.1 Statische Code Analyse

Nach Möglichkeit werden Tools zur statischen Sourcecode Analyse eingesetzt. Falls möglich, wird Sonarqube[2] eingesetzt, da das Team damit am meisten Erfahrung hat.

8.3.2 Code Reviews

Für Code Reviews wird das Feature von GitLab verwendet, bei dem sich einzelne Zeilen eines Commits kommentieren lassen. Ausserdem können Kritikpunkte verbal mitgeteilt werden.

8.3.3 Code Style Guidelines

Es werden die Code Style Guidelines von Google befolgt.

Java	`https://google.github.io/styleguide/javaguide.html`
Andere	`https://github.com/google/styleguide`

Tabelle 8.1: Code Style Guidelines

8.4 Testen

8.4.1 Unit Tests

Die Testabdeckung des Codes sollte so hoch wie möglich, aber mindestens 75% sein. Die Tests müssen deterministisch sein und dürfen auf dem Buildserver niemals fehlschlagen. Ist dies trotzdem der Fall, muss der Fehler sofort behoben werden.

Für das schreiben der Tests soll der BDD[3] Stil verwendet werden.

```
1  public void shouldDoEverythingAsItOughtToBeDone() {
2    // Given
3    initialization...
4
5    // When
6    test execution...
7
8    // Then
9    assertions...
10 }
```

Listing 8.1: BDD Unit Test Stil

8.4.2 Integrationstests

Um die Genauigkeit der Lösung zu testen, muss die Lerndatenmenge zufällig in zwei Teile geteilt werden. Der eine Teil dient der Trainierung der Software und der andere wird benötigt um die Genauigkeit zu testen.[1]

Das genaue Testverfahren wird während der Versuchsplanung ausgearbeitet und ausführlich beschrieben.

Anhang A

Aufgabenstellung

Automatisches Erkennen von defekten Paketen

Aufgabenstellung für Florian Bitterlin und Cyrill Schenkel

Ausgangslage

Die Intigenia Produktions AG plant in ihrem Industrie 4.0 Projekt die Automatisierung ihrer Produktionsstrasse für die Windelproduktion. In einem der Produktionsschritte werden die einzelnen Windeln in Plastikpakete abgefüllt und diese werden verschweisst.
Beim Schweissen entsteht Ausschuss, der heute manuell erkannt und aussortiert wird. Dieser Schritt soll vollständig automatisiert werden.
Mittels Bilderkennung soll die Verpackung geprüft werden, dass keine Risse oder beschädigte Stellen sichtbar sind. Im Weiteren soll auch geprüft werden ob die Beutel richtig verschlossen (Kunststoff Beutel verschweisst) sind.
Diese Überprüfung muss sehr schnell gehen, da sonst die Produktion aufgehalten wird. Mit einem Volumen von 700-1200 Windeln pro Minute sind dies 15-40 Verpackungen und dies bei 3 Maschinen. Dies bedeutet bei den zunehmend kleineren Beutelinhalten (ca. 30Stück pro Pack), dass bei 3 Anlagen mit bis zu 120 Packungen/min zu rechnen ist. Um alle Seiten der Packung prüfen zu können, sind 6 Bilder pro Packung notwendig. Der Entscheid ob all es korrekt ist, muss innert 0,5 Sekunden passieren. Hardware, die diese Laufzeit einhalten kann, gibt es bereits. Jedoch ist nicht bekannt, ob der Erkennungsalgorithmus mit diesem Durchsatz zurechtkommen wird. Zur Erkennung müssen verschiedene Methoden des Visual Computings (Bildfilter, Konturierung, farbliche Verläufe) und des Machine Learnings evaluiert und geeignete Verfahren (bzw. Kombination der Verfahren) gewählt werden.
Zudem ist das Feststellen ob ein Produktionsfehler vorliegt nicht ganz einfach. Die farblichen Unterschiede auf der Verpackung können nur sehr schwach ausfallen. Bei Rissen in der Verpackung wird dieses Problem verschärft, da Verpackung und Inhalt beide weiss sein können. Jedoch ermöglicht die grosse Produktionsmenge und entsprechende Testdaten eine gute Basis zur Auswertung, Training und Abstimmung von Algorithmen.

Fig: Eine Art von Produktionsfehler sind nicht sauber verschweisste Verpackungen. Die blaue Markierung zeigt auf wie es eigentlich sein sollte – die roten Markierungen zeigen fehlerhafte Verschweissungen, welche von den Kunden nicht akzeptiert werden.

Aufgabenstellung

In dieser Arbeit soll ein Prototyp erstellt werden, der anhand von Bildern der Windelpakete erkennen kann, ob diese fehlerhaft sind.
Dazu kommen Verfahren der Bildverarbeitung und des Machine Learning (Klassifizierung, Supervised Learning) zum Einsatz.

Das Projekt kann in Java oder C# programmiert werden. Geeignete Libraries müssen vorgängig evaluiert werden.
Folgende Defekte soll das gewählte Verfahren entdecken:
- Schweissnaht: ist die Verpackung sauber geschlossen.
- Risse in Folie
- Evaluation des Verfahren auf statischen Bildern, Zeitdauer für Verarbeitung < 1.5s

Resultate:
- Evaluation von geeigneten SW-Bibliotheken und Verfahren
- Bericht zu den evaluierten Verfahren und SW-Bibliotheken
- Lauffähiges SW-System der evaluieren Verfahren.
- Test (inkl. Performance-Abklärungen)
- Dokumentation gemäss Vorgaben des Studiengangs Informatik.

Auftraggeber

Intigena Produktions AG, Untere Fischbachstr.2, 8932 Mettmenstetten.

Projektteam

Florian Bitterlin, fbitterl@hsr.ch
Cyrill Schenkel, cschenke@hsr.ch

Betreuung HSR

Hansjörg Huser, hhuser@hsr.ch, Tel: 055 222 49 12 (HSR Raum 8.242)

Projektabwicklung

Termine:

- Beginn der Arbeit: M**o., 19. Sept. 2016**
- Abgabetermin Kurzfassung/Poster/Mgmt-Summary zum Review: **20. Dez. 2016**
- Abgabetermin: **23. Dez. 2016, 17.00 Uhr**
- Zwischenbesprechung/Review mit Auftraggeber nach Projektplan

Arbeitsaufwand

Für die erfolgreich abgeschlossene Arbeit werden 8 ECTS angerechnet. Dies entspricht einer Arbeitsleistung von mind. 240 Stunden pro Student. (14 Wochen zu ca. 17h)

Hinweise für die Gliederung und Abwicklung des Projektes:

Gliedern Sie Ihre Arbeit in 4 bis 5 Teilschritte. Schliessen Sie jeden Teilschritt mit einem Meilenstein ab. Definieren Sie für jeden Meilenstein, welche Resultate dann vorliegen müssen!

Folgende Teilschritte bzw. Meilensteine sollten Sie in Ihrer Planung vorsehen:
Schritt 1: Projektauftrag inkl. Projektplan (mit Meilensteinen),
- Meilenstein 1: Review des Projektauftrages abgeschlossen. Projektauftrag von Auftraggeber und Dozent genehmigt
  Letzter Meilenstein: Systemtest abgeschlossen
  Termin: ca. eine Woche vor Abgabe
- Entwickeln Sie Ihre SW in einem iterativen, inkrementellen Prozess: Planen Sie möglichst früh (spätestens in der Mitte des Projektes) einen ersten lauffähigen Prototypen mit den wichtigsten und kritischsten Kernfunktionen. In die folgenden Phasen können Sie dieses Kernsystem schrittweise ausbauen und testen.
- Falls Sie in Ihrer Arbeit neue oder Ihnen unbekannte Technologien einsetzen, sollten Sie parallel zum Erarbeiten des Projektauftrages mit dem Technologiestudium beginnen.

- Setzen Sie konsequent Unit-Tests ein! Verwalten Sie ihre Software und Dokumente auf einem geeigneten Repository. Stellen Sie sicher, dass der/die Betreuer jederzeit Zugriff auf das Repository haben und dass das Projekt anhand des Repositories jederzeit wiederhergestellt werden kann.
- Achten Sie auf die Einhaltung guter Programmier- und Designprinzipien.
- Halten Sie sich im Übrigen an die Vorgaben aus dem Modul SE-Projekt.

Projektadministration

- Führen Sie ein individuelles Projekttagebuch aus dem ersichtlich wird, welche Arbeiten Sie durchgeführt haben (inkl. Zeitaufwand). Diese Angaben sollten u.a. eine individuelle Beurteilung ermöglichen.
- Dokumentieren Sie Ihre Arbeiten laufend. Legen Sie Ihre Projektdokumentation mit der aktuellen Planung und den Beschreibungen der Arbeitsresultate elektronisch in einem Projektordner ab. Dieser Projektordner sollte jederzeit einsehbar sein (z.B svn-Server oder File-Share).

Inhalt der Dokumentation

Bei der Abgabe muss jede Arbeit folgende Inhalte haben:

- **Dokumente gemäss Vorgabe: https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html (Dokument Ablaeufe_und_Regelungen_Studien-_und_Bachelorarbeiten_141027_.pdf)**
- **Aufgabenstellung**
- **Technischer Bericht**
- **Projektdokumentation**
- **Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.**
- **Zitate sind zu kennzeichnen, die Quelle ist anzugeben.**
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Projekttagebuch, Dokumentation des Projektverlaufes, Planung etc.

Fortschrittsbesprechung:

Regelmässig findet zu einem fixen Zeitpunkt eine Fortschrittsbesprechung statt.
Teilnehmer: Dozent und Studenten, bei Bedarf auch Vertreter der Auftraggeber
Termin: jeweils xxx, Raum 8.242
Traktanden

- Was wurde erreicht, was ist geplant, welche Probleme stehen an
- Review von Code/Dokumentation (Abgabe jeweils einen Tag vor dem Meeting

Falls notwendig, können weitere Besprechungen / Diskussionen einberufen werden.
Sie erstellen zu jeder Besprechung ein Kurzprotokoll, welches Sie spätestens 2 Tage nach der Sitzung per e-mail an den Betreuer senden.

Rapperswil, 19. Sept. 2016
Hansjörg Huser

Literatur

[1] David Page. *Evaluating Machine Learning Methods*. 15. Feb. 2016. URL: http://pages.cs.wisc.edu/~dpage/cs760/evaluating.pdf (besucht am 20. 09. 2016).

[2] SonarSource, Hrsg. *SonarQube*. URL: http://www.sonarqube.org/ (besucht am 20. 09. 2016).

[3] Wikipedia, Hrsg. *Behavior-driven development*. URL: https://en.wikipedia.org/wiki/Behavior-driven_development (besucht am 20. 09. 2016).

Teil II

Bericht

Kapitel 9

Ausgangslage

Bei der *Intigena Produktions AG* werden Produkte[1] in Packungen aus weicher Plastikfolie verpackt und die Verpackung zugeschweisst. Das Verpacken erfolgt automatisch. Die Packungsgrössen und Designs varieren nach Kundenwunsch. Beim Schweissen der Verpackungen kann Ausschuss entstehen. Ausschuss sind jene Packungen, die nicht vollständig verschlossen sind oder Löcher haben. Wenn Schäden nicht erkannt werden, kann dies zu teuren Rückrufaktionen führen, die so gut wie möglich verhindert werden sollten.

Bis anhin wurde solcher Ausschuss von Hand aussortiert. Mit einem Durchsatz von bis zu 120 Packungen/min pro Maschine (0.5 s/Packung) ist es jedoch undenkbar, dass die Genauigkeit der Beurteilung aufrecht erhalten werden kann, ohne die Produktion aufzuhalten. Aus diesem Grund soll die Automatisierbarkeit dieser Qualitätsprüfung untersucht werden.

Zu diesem Zweck wird in dieser Arbeit die Eignung von visuellen Sensoren geprüft. Anhand von soll mittels Software erkannt werden, ob eine allfällige Beschädigung an einer Verpackung vorliegt. Für diese Arbeit wird der Einfachheit halber angenommen, dass es sich dabei um Aufnahmen von ruhenden Packungen handelt. Für den Einsatz in der Produktionsanlage, kommen auch bewegte Aufnahmen, also ein Video-Stream in Frage.

[1] Im Rahmen dieser Arbeit sind dies Inkontinenzeinlagen für Frauen.

Kapitel 10

Problembeschreibung

Die *Intigena Produktions AG* plant die Überprüfung der Verpackungen zu automatisieren um einen Engpass bei diesem Arbeitsschritt zu vermeiden. Denn zukünftig sollen die Packungen von einer neuen Maschine direkt auf Transportpalette verladen werden. Bis anhin wurde dieser Schritt von Menschen ausgeführt. Bisher findet sich jedoch kein Anbieter für die Automatisierung der Überprüfung von Verpackungen. Gängige Industriekameras sind mit dem Erkennen der meisten auftretenden Mängel überfordert.

Deshalb soll im Rahmen dieses Projekts herausgefunden werden, ob dieses Problem mithilfe von Bildverarbeitung und Machine Learning gelöst werden kann. Die Fehlerrate bei der Aussortierung ist nicht gegeben. Ebenfalls gibt es nur grobe Schätzungen dazu, wieviele Packungen Ausschuss sind (ca. 10-20%).

10.1 Ziele

Folgende Projektziele sind nach SMART[1] definiert. Jedes Ziel hat eine Identifikationsnummer mit der Form *G–##*.

10.2 Gültigkeitsbereich der Anforderungen

Die unten aufgeführten Anforderungen betreffen den umzusetzenden Prototyp und weder das Projekt als ganzes noch die Anforderungen, an ein fertiges im Produktionsbetrieb einsetzbares Produkt, obwohl sich die Anforderungen mit denen eines solchen notwendigerweise überschneiden.

Vielmehr enthalten die unten aufgeführten Anforderungen die Funktionalität, die vonnöten ist um eine Aussage über die vorangestellten Fragestellungen zu machen.

[1] **S**pezifisch, **M**essbar, **A**kzeptiert, **R**ealistisch, **T**erminiert

G–01	Bis zum Fälligkeitsdatum des M3 (7 Wochen nach Projektbeginn) soll ein Softwareprototyp implementiert werden, der zwischen defekten und intakten Packungen unterscheiden kann.
G–02	Der Prototyp soll bis zum Fälligkeitsdatum des M4 (9 Wochen nach Projektbeginn) einen minimalen Durchsatz von 120 Packungen/min haben.
G–03	Der Prototyp soll bis zum Fälligkeitsdatum des M4 (9 Wochen nach Projektbeginn) eine minimale Genauigkeit, wie sie in (NFR–01) definiert ist, haben.

Tabelle 10.2: Projektziele

10.3 Qualitätskriterien für Anforderungen

Die unten aufgeführten und in dieser Arbeit verwendeten Qualitätskriterien für Anforderungen, wurdem [3] entnommen und sollten dem Standard IEEE 29148-2011 entsprechen.

Vollständigkeit Alle Anforderungen müssen die geforderte Funktionalität vollständig beschreiben. Unvollständige Anforderungen müssen, solange sie unvollständig bleiben mit der Markierung *tbd.*[2] versehen werden.

Notwendigkeit Jede Anforderung muss der Erfüllung mindestens eines Zieles dienen. Es muss klar sein, welches Ziel durch eine Anforderung erfüllt werden soll.

Atomizität "Jeder Anforderungssatz enthält genau eine Anforderung, das heißt, genau ein Vollverb (Prozesswort). Konjunktionen, die mehrere Anforderungen in einem Anforderungssatz ermöglichen, werden nicht verwendet." [3, S. 27]

Verfolgbarkeit Für jede Anforderung muss definiert sein, woher sie stammt. Anforderungen auf einer tieferen Ebene und Entwicklungsartefakte müssen jeweils zu einer Anforderung zurückführbar sein.

Technische Lösungsneutralität Technische Lösungsansätze dürfen durch eine Anforderung nicht im Voraus ausgeschlossen werden. "Bei der Beschreibung der notwendigen und hinreichenden Eigenschaften und Funktionalitäten des Systems muss darauf geachtet werden, dass sie keine unnötigen Einschränkungen für die Architektur und Entwicklung enthält." [3, S. 27]

Realisiserbarkeit Die Umsetzbarkeit im gegebenen Rahmen muss für jede Anforderung gegeben sein.

[2] "to be determined/to be done" [3, S. 26]

Konsistenz Anforderungen dürfen in keinem Falle im Widerspruch zu einander oder zu sich selbst stehen.

Eindeutigkeit Es darf für jede Anforderung nur eine mögliche Deutung geben, die leicht zu verstehen sein muss.

Prüfbarkeit Jede Anforderung muss durch eine Messung oder einen Test prüfbar sein.

10.3.1 Namensgebungsschema

Um die Verfolgbarkeit der Anforderungen zu gewährleisten, wird folgendes Namensgebungsschema eingeführt.

$$\underbrace{\text{FRV}}_{\text{Anforderungstyp}} - \overbrace{01}^{\text{Anforderungs ID}}$$

Abbildung 10.1: Namensgebungsschema für Anforderungen

Mit zählen wird bei 01 angefangen. Als Trennstrich wird weder "-" noch "—" sondern "–" verwendet. Folgende Anforderungstypen sind definiert.

FRV	Funktionale Anforderung (Level 0; Vision)
FRE	Funktionale Anforderung (Level 1; Epic)
FRS	Funktionale Anforderung (Level 2; User Story)
NFR	Nicht-Funktionale Anforderung

Tabelle 10.3: Anforderungstypen

10.4 Funktionale Anforderungen

In den folgenden Abschnitten werden alle funktionale Anforderungen an den Prototypen beschrieben.

10.4.1 FRV–01 • Prototyp kann Bild klassifizieren

Ursprung
G–01

Priorität
Muss

Abhängig von
—

Definition

Der Prototyp muss dazu in der Lage sein, mit einer bestimmten Genauigkeit (NFR–01), innerhalb eines vorgegebenen Zeitrahmens (NFR–02) zu entscheiden, ob auf der gegebenen eine defekte oder intakte Packung (FRS–01) abgebildet ist.

10.4.2 FRE–01 • Trainingsdaten definieren

Ursprung

FRV–01

Priorität

Muss

Abhängig von

—

Definition

ALS EIN	Projektmitarbeiter
MÖCHTE ICH	die Trainingsdaten definieren,
UM	das Modell für einen Test zu trainieren.

10.4.3 FRE–02 • Testdaten definieren

Ursprung

FRV–01

Priorität

Muss

Abhängig von

—

Definition

ALS EIN	Projektmitarbeiter
MÖCHTE ICH	die Testdaten definieren,
UM	das Modell mit den Testdaten testen zu können.

10.4.4 FRE–03 • Test durchführen

Ursprung

FRV–01

Priorität

Muss

Abhängig von

FRE–01, FRE–02

Definition

ALS EIN	Projektmitarbeiter
MÖCHTE ICH	einen Test durchführen,
UM	die Anforderungstreue des Prototypen zu prüfen.

10.4.5 FRE–04 • Report generieren

Ursprung
FRV–01

Priorität
Kann

Abhängig von
FRE–03

Definition

ALS EIN	Projektmitarbeiter
MÖCHTE ICH	einen Report eines durchgeführten Tests generieren,
UM	die Anforderungstreue des Prototypen zu prüfen.

10.4.6 FRS–01 • Foto klassifizieren

Ursprung
FRE–03

Priorität
Muss

Abhängig von
FRE–01

Definition

ALS EIN	Projektmitarbeiter
MÖCHTE ICH	ein Foto klassifizieren,
UM	um das trainierte Modell zu testen.

Defekt	Eine Packung ist defekt, wenn: • sie Löcher hat ODER • die Schweissnaht nicht richtig verschlossen ist
Intakt	Eine Packung ist intakt, wenn keine Schäden *sichtbar* sind.

Tabelle 10.5: Klassen von Packungen

10.4.7 FRS–02 • Fehlerrate bestimmen

Ursprung
FRE–04

Priorität
Soll

Abhängig von
FRE–03

Definition

ALS EIN	Projektmitarbeiter
MÖCHTE ICH	die Fehlerrate der Klassifizierung wissen,
UM	um die Genauigkeit der Klassifizierung zu testen.

Falls dieser Anforderung nicht entsprochen werden kann, muss die Fehlerwahrscheinlichkeit manuell ermittelt werden, was bei vielen Tests sehr aufwändig werden kann. Deshalb *soll* diese Anforderung umgesetzt werden.

Mehr Informationen können Beispielsweise bei [12] gefunden werden.

In NFR–01 werden die benötigten Formeln aufgelistet.

Siehe auch
NFR–01

10.5 Nicht-Funktionale Anforderungen

10.5.1 NFR–01 • Genauigkeit

Ursprung
G–03

Priorität
Muss

Abhängig von
FRE–03

Definition
Der Fokus liegt auf der Minimierung von FN. Angestrebt wird das zutreffen folgender Aussage.

$$FNR < 1.5\% \wedge FPR < 7.5\% \tag{10.1}$$

Laut Aussage des Kunden soll $FNR < 1-2\%$ und $FPR < 5-10\%$ sein. Es wird daher der Mittelwert der vorgegebenen Bereiche angestrebt.

$$FPR = \frac{FP}{N} \tag{10.2}$$

$$FNR = \frac{FN}{P} \tag{10.3}$$

Mehr zur binären Klassifizierung von stark unbalancierten Klassen hier:
`http://stats.stackexchange.com/questions/235808/binary-classification-with-strongly-unbalanced-classes`

[2, 12]

10.5.2 NFR–02 • Performance

Ursprung
G–02

Priorität
Muss

Abhängig von
FRE–03

Definition
Der Softwareprototyp *muss* mindestens 120 Packungen/min klassifizieren können.

Kapitel 11

Lösungskonzept

Zur Lösung der Problemstellung wurden verschiedene Varianten in Betracht gezogen. Im Laufe der konzeptionellen Phase dieser Machbarkeitsstudie wurde die Anzahl der Lösungsvarianten kontinuierlich reduziert.

Als Basis für die Umsetzung der Lösung, wurde das Inception v3 Modell und TensorFlow gewählt. Diese Wahl wird unten begründet.

Das PQC Tool soll das Tuning der Lösung vereinfachen. Seine Benutzerschnittstelle ist darauf ausgelegt, dass es einfach in Shell-Scripts verwendet werden kann.

11.1 Frameworks

Es wurden 3 Frameworks für die Umsetzung der Lösung in Betracht gezogen. Unten werden diese kurz beschrieben.

11.1.1 VLFeat

VLFeat ist eine Open Source Library deren primärer Fokus Image Understanding und Feature Extraction ist. VLFeat wurde in C geschrieben und hat ein MATLAB FFI. Im Rahmen dieser Machbarkeitsstudie wurde die MATLAB API über Octave genutzt.

11.1.2 OpenCV

OpenCV ist eine grosse Computer Vision Library, die eine Vielzahl von Machine Learning und Image Processing Algorithmen implementiert. OpenCV wurde in C und C++ geschrieben. Zusätzlich zu den C und C++ APIs gibt es Java und Python FFIs, die im Rahmen dieser Machbarkeitsstudie beide verwendet wurden.

11.1.3 TensorFlow

TensorFlow ist ein OpenSource Projekt von Google, mit welcher einfach neuronale Netze programmiert werden können. TensorFlow funktioniert mit Data Flow Graphs. Die Library kommt mit vielen vorgefertigten Opertationen. Ursprünglich wurde TensorFlow vom Google Brain Team entwickelt. Seit November 2015 kann es unter einer Apache 2.0 Open Source Lizenz bezogen werden. Es wird in vielen Google Produkten wie Google Search, Google Maps oder Google Photos verwendet. Geschrieben in C++ und Python, bietet TensorFlow auch für beide Sprachen eine API an.

11.2 Entscheidungsfindung

Mittels dem Ausschlussverfahren wurde der schliessliche Lösungsansatz ausgewählt. Zur Evaluierung jeder Lösung, wurde eine Menge von klassifizierten Daten (120 davon defekt und 50 intakt) verwendet, die in der Produktionsstätte aufgenommen wurden. Für alle durchgeführten Tests, wurden die klassifizierten Daten jeweils in Testdaten und Trainingsdaten aufgeteilt, sodass stets eine Validierungsmenge vorhanden war. Die verfügbare Datenmenge war also eher klein im Verhältnis zum Informationsgehalt eines einzelnen Datensatzes.

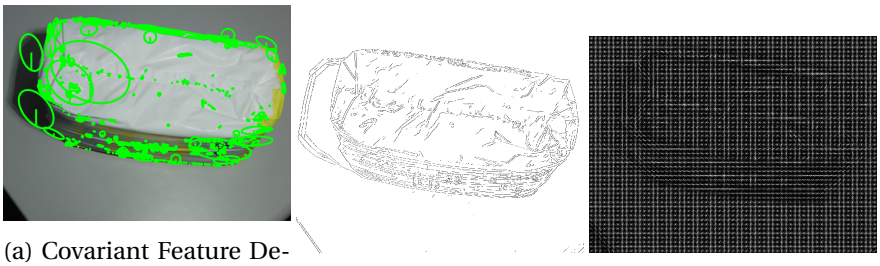
Deshalb war eine der Hauptüberlegungen, wie die relevanten Daten, die differenzierenden Merkmale, die Features aus einer extrahiert werden können. Die gewählte Vorgehensweise war daher, den Feature Vektor mittels Line- und Corner-Detection-Verfahren zu verkleinern. Hätten die Schweissnähte, Falten und Löcher auf Linien oder schliesslich Pfade abgebildet werden können, so war es die Hoffnung, daraus auf den Zustand der Packungen zurückschliessen zu können.[1] Diese Überlegung wurde mit VLFeat und OpenCV verfolgt.

Am Anfang wurde davon ausgegangen, dass die vefügbare klassifizierte Datenmenge zu klein ist, um mit TensorFlow und Deep Learning nützliche Resultate zu erzielen. Um TensorFlow ausschliessen zu können, wurden einige Tests durchgeführt, bei denen der höchste Layer von Inception v3 nachtrainiet wurde. Diese Tests brachten im Verhältnis zu den zuvor mit VLFeat und OpenCV durchgeführten Tests, viel bessere Resultate hervor.

11.2.1 Vorgehen bei der Evaluation

Die Versuche mit VLFeat wurden relativ schnell wieder abgebrochen, da das Framework nicht die benötigten Algorithmen zur Verfügung stellte. Diese hätten nachimplementiert werden müssen, was durch die Verwendung von OpenCV nicht mehr nötig war.

[1] Prof. Hansjörg Huser hat etwas ähnliches mit k-Means geschafft. Die Genauigkeit und Performance lag allerdings hinter der Lösung mit TensorFlow zurück.



(a) Covariant Feature Detector (b) Canny (c) HOG

Abbildung 11.1: VLFeat Resultate

Dennoch wurden mit VLFeat einige Algorithmen für das Extrahieren von Features getestet. Die Resultate können der Abbildung 11.1 entnommen werden. Da wird auch der Einfluss der Bildqualität sichtbar.

Die ersten Versuche mit OpenCV wurden in Java getätigt. Danach wurde nur noch Python verwendet. Die Verwendung der Python API vereinfachte Veränderungen am Code und die Visualisierung der Resultate.

Um die Parameter der eingesetzten Alogrithmen tunen zu können, wurde ein kleines UI implementiert. (Siehe Abbildung 11.2.)

Bei den Tests mit OpenCV zeigten sich weitere Probleme. Irrelevante Schätten und Beschriftungen wurden besonders gut erkannt. Dies rauszufiltern stellte sich als schwierig heraus. Ausserdem mussten die Parameter für jedes Bild anders eingestellt werden, um annähernd brauchbare Resultate zu erzielen. Kleine Unterschiede in der Position der Packungen oder der Lichtverhältnisse, führten zu komplett anderen Resultaten.

Die Tests mit TensorFlow brachten viel überzeugendere Resultate hervor. Mit Standardparameter konnte Inception v3 so umtrainiert werden, dass es einen grossen Teil der Testdaten richtig klassifizieren konnte. Die Genauigkeit entsprach jedoch bei weitem nicht den Anforderungen. Nach einer neuen Sortierung der klassifizierten Daten, verbesserten sich die Resultate leicht. Es wurde aber schnell klar, das mehr Daten von besserer Qualität benötigt wurden, um das Resultat zu verbessern. Damit und mit der Optimierung der Parameter von Inception v3, beschäftigt sich der Rest dieser Machbarkeitsstudie.

11.2.2 Entscheid

Die ersten beiden Varianten wurden in dieser Machbarkeitsstudie auch aus Zeitgründen nicht weiter verfolgt. Hansjörg Huser konnte mit diesen Methoden während des Projekts mithilfe der 2. Fotoserie doch noch bessere Resultate erzielen, jedoch kam die Lösung mit Inception v3 und TensorFlow trotzdem näher an die Anforderungen heran.

Aus den oben genannten Resultaten, ergeben sich die folgenden Entscheidungen und Schlüsse für das weitere Vorgehen:

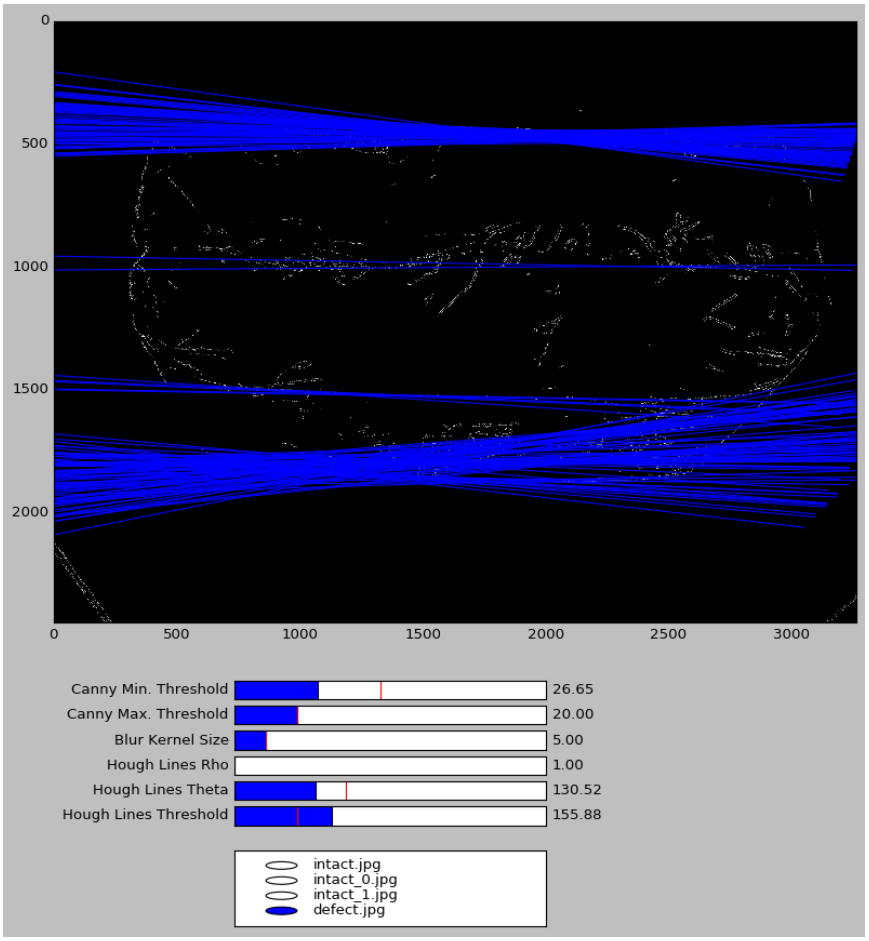


Abbildung 11.2: OpenCV: UI für Tuning von HLT und Canny Parameter

- Die Lösung wird auf der Basis von Inception v3 und TensorFlow entwickelt.
- Die Qualität und Quantität der verwendeten klassifizierten Daten muss erhöht werden, damit die Anforderungen besser erfüllt werden können.
- Der Prozess zum Aufteilen der vorhandenen klassifizierten Daten in Trainings- und Testdaten, sowie auch das Retraining und Testen des Modells werden automatisiert um das Tuning der Inception v3 Parameter zu vereinfachen.

11.3 Architektur

Die Lösung besteht aus drei Phasen. In der Phase I werden die klassifizierte Daten vorbereitet. In der Phase II wird das Modell trainiert. In der Phase III wird das trainierte Modell getestet.

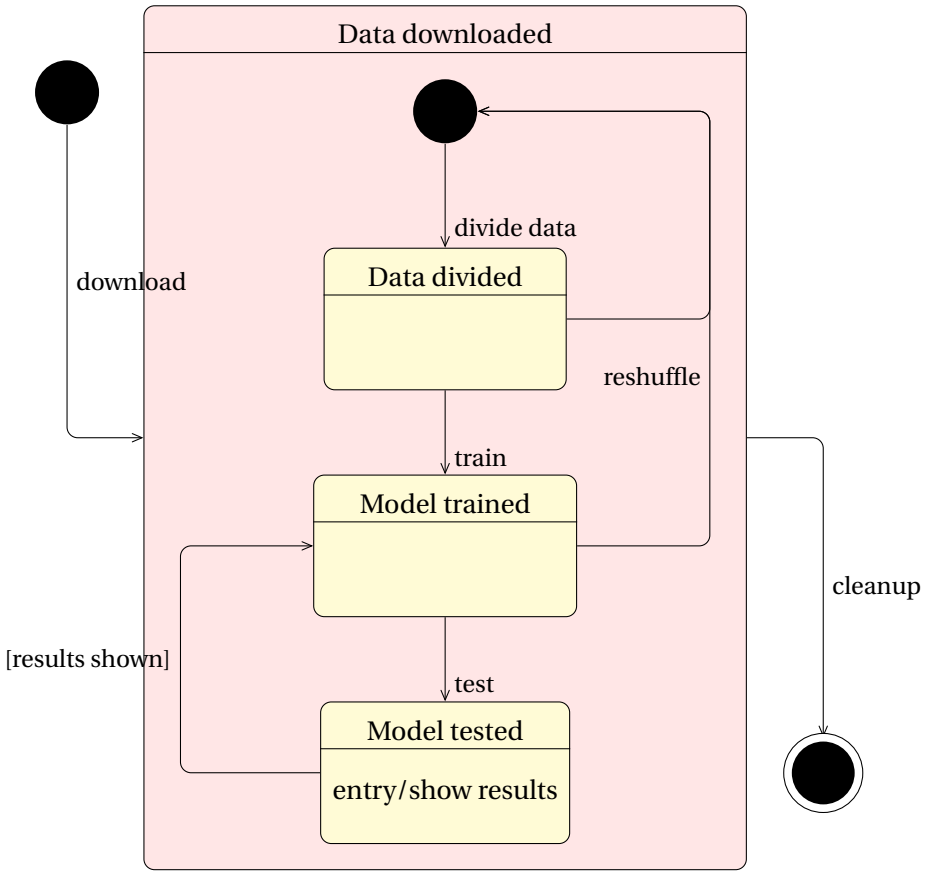


Abbildung 11.3: Überblick über die Zustände der Lösung

In Abbildung 11.3 werden die sechs Zustände der Lösung beschrieben. Die beiden Zustände *Data downloaded* und *Data divided* inklusive den Operationen *download*, *divide data* und *reshuffle* gehören der Phase I an. Dabei ist *reshuffle* ein Spezialfall. Diese Operation wird gebraucht um von den Zuständen *Data divided* und *Model trained* wieder in den Zustand *Data downloaded* zurückzukehren um die Phase I und alle folgenden Phasen erneut durchlaufen zu können. Die *reshuffle* Operation bietet im Wesentlichen die Möglichkeit einen erneuten Test unter geänderten Voraussetzungen zu machen, ohne dass dafür die klassifizierten Daten erneut heruntergeladen werden müssen. In der konkreten Implementation sind alle in Abbildung 11.3 Aufgeführten Zustände, mit der Ausnahme von *Model tested*, persistent.

Die Phase II startet mit dem Zustand *Data divided* und gelangt über die Operation *train* in ihren Endzustand *Model trained*. Um das Modell mit einer anderen Aufteilung der klassifizierten Daten in Trainings- und Testdaten erneut trainieren zu können, kann die Operation *reshuffle* verwendet werden.

Die Phase III startet mit dem Zustand *Model trained* und gelangt über die

Operation *test* in den *Model tested* Zustand. Beim Eintritt in den *Model tested* Zustand, werden die Resultate des Tests dem Benützer angezeigt (*show results*). Sobald die Resultate angezeigt wurden (*[results shown]*), kehrt die Lösung in den *Model trained* Zustand zurück.

Zu keiner speziellen Phase gehört die Operation *cleanup*. Diese führt vom Zustand *Data downloaded* zum Endzustand. Im Wesentlichen löscht diese Operation den aktuellen Zustand, inklusive der heruntergeladenen klassifizierten Daten vom Filesystem.

11.3.1 Benutzerschnittstelle

Ähnlich wie bei Git[5] sind alle Funktionalitäten der Lösung über einen einzigen Einstiegspunkt `pqc` ansteuerbar. Das erste Argument des Einstiegspunkts dient der Ansteuerung einer bestimmten Funktionalität. Weitere Argumente können der Ansteuerung von Teilfunktionalitäten oder der genaueren Spezifikation der Ausführung der zuvor bestimmten Funktionalität dienen.

```
pqc <command> [<args> | <subcommand> [args]]
```

Ab hier werden die Argumente zur Ansteuerung bestimmter Funktionalitäten *Kommando* und die Argumente zur Ansteuerung bestimmter Unterfunktionalitäten *Unterkommando* genannt.

Im Listing 11.1 ist die einfachste sinnvolle Sitzung dargestellt. Zuerst wird das Arbeitsverzeichnis mit dem `init` Kommando initialisiert. Das entspricht der Operation *download*. Darauf wird in das initialisierte Verzeichnis gewechselt. Mit dem Kommando `divide 0.3` werden die klassifizierten Daten in 70% Trainingsdaten und 30% Testdaten aufgeteilt. Der Schritt zum Training des Modells wird übersprungen und mit dem `test` Kommando implizit durchgeführt. Die Testresultate werden dem Benutzer angezeigt. Mit dem `clean` Kommando wird das Arbeitsverzeichnis wieder aufgeräumt.

```
$ pqc init pqc_wd
$ cd pqc_wd
$ pqc divide 0.3
$ pqc test
$ pqc clean
```

Listing 11.1: Beispielsitzung ohne Ausgaben

Aus dem Zustandsdiagramm werden Abhängigkeiten zwischen den einzelnen Operationen ersichtlich. Wenn die Lösung beispielsweise im Zustand *Data divided* ist und der Benützer das `test` Kommando aufruft, wird die *train* Operation vor der *test* Operation ausgeführt, da eine Abhängigkeit von der letzteren auf die erste besteht. Das selbe gilt für *reshuffle* und *divide data*, wenn die Applikation im *Data divided* oder *Model trained* Zustand ist und der Benützer das `divide` Kommando aufruft. Wenn kein Arbeitsverzeichnis initialisiert wurde,

bevor ein Kommando das vom *Data downloaded* Zustand abhängt aufgerufen wird, wird ein temporäres Arbeitsverzeichnis initialisiert.

Alle Kommandos, ihre genaue Bedeutung und alle ihre Optionen sind im Benutzerhandbuch im Anhang G beschrieben.

11.3.2 Aufteilung in Module

Sowie die Benutzerschnittstelle anhand der Aufteilung der Lösung in Phasen, Zustände und Operationen strukturiert ist, sind die Module nach diesen Kategorien strukturiert. In Abbildung 11.4 werden die Aufteilung in Module, sowie die wichtigsten öffentlichen Schnittstellen beschrieben.

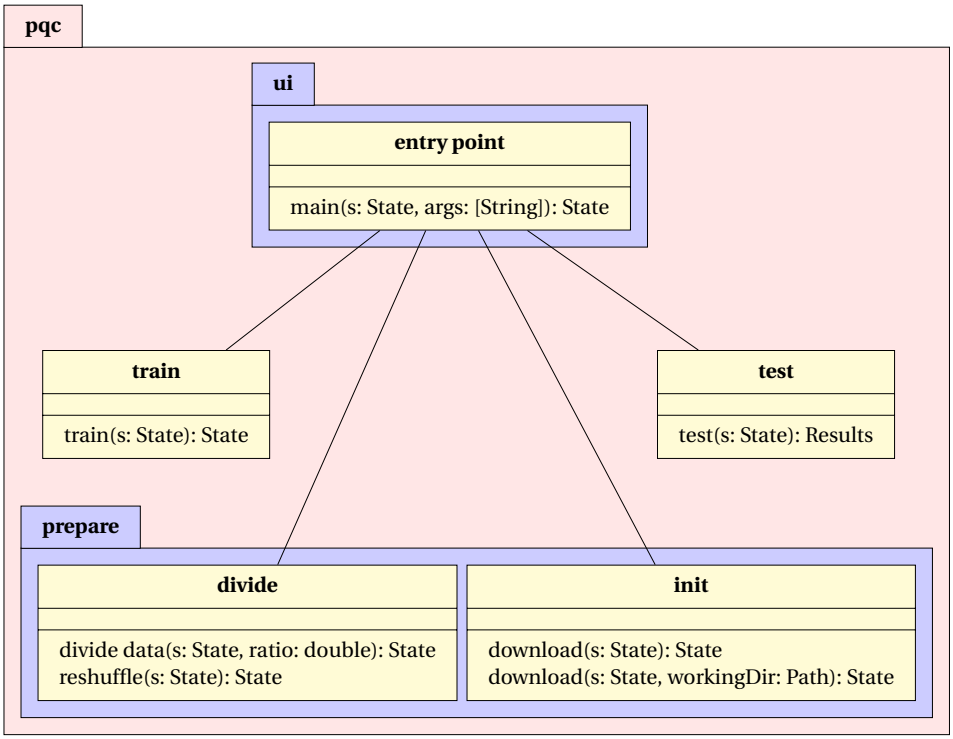


Abbildung 11.4: Aufteilung in Module

Kapitel 12

Testkonzept

Dieses Testkonzept ist seiner Struktur und seinem Inhalt nach dem im Abschnitt 8 von IEEE 829-2008 beschriebenen MTP (Master Test Plan) angelehnt.[6]

Im Folgenden wird beschrieben, wie die Einhaltung der Qualitätskriterien und der Anforderungen an das PQC Projekt gewährleistet werden soll.

12.1 Gültigkeitsbereich

Die Gültigkeit dieses Testkonzepts beschränkt sich auf das im Rahmen des PQC Projekts erstellte pqc-Tool, sowie es im Kapitel 10 und Kapitel 11, insbesondere im Abschnitt 11.3 spezifiziert ist.

Getestet wird einerseits, ob sich das pqc-Tool so verhaltet, wie das im Abschnitt 11.3 (Architektur) beschrieben ist und andererseits, ob die Lösung allen Nichtfunktionalen und Funktionalen Anforderungen gerecht wird und damit alle Projektziele erfüllt sind.

Die Abhängigkeiten des Projekts werden nicht getestet, da angenommen wird, dass diese bereits ausreichend getestet sind.

Dieser Testplan gilt für die gesamte Dauer des Projekts. Begrenzungen für die Gültigkeit der einzelnen Testmethoden werden weiter unten geregelt. (Siehe Unterabschnitt 12.3.3.)

Die Entwicklungsmethode folgt einem eher iterativen Modell. Wobei allerdings die grundlegenden Anforderungen alle am Anfang des Projekts festgelegt werden und danach nur leicht angepasst werden. Näheres dazu kann dem Projektplan entnommen werden.

Das heisst, dass Tests während des Projekts immer wieder durchgeführt und angepasst werden müssen. Während der Versuchsdurchführungsphase, wird Adhoc Testing als Testmethode verwendet. Unit-Tests kommen nicht in Frage, da die Lösung sehr Eingabe-/Ausgabeorientiert ist. Zum Ende der Versuchsdurchführungsphase wird ein Systemtest durchgeführt.

Einige der im IEEE 829-2008 spezifizierten Inhalte eines MTPs werden in diesem Dokument stark verkürzt geführt, aufgrund des kleinen Projektumfangs.

12.2 Systemübersicht und Schlüsselfunktionalitäten

Das System und alle Anforderungen an dasselbe sind im Kapitel 10 und Kapitel 11 ausführlich spezifiziert. Für das Testkonzept sind das Abbildung 11.3 (Zustandsdiagramm), sowie alle funktionalen[1] und nichtfunktionalen[2] Anforderungen von besonderer Bedeutung.

12.3 Testübersicht

Während der Versuchsdurchführungsphase werden hauptsächlich Ad-Hoc Tests durchgeführt, in der Auswertungsphase kommen dann formalere Testmethoden zum Einsatz.

12.3.1 Organisation

Die Projektmitglieder sind gleichermassen für das Testing verantwortlich. Einerseits testen sie jeweils ihre eigenen Implementationen und andererseits wird jede Codeänderung von jemandem gereviewt, der nicht an dieser beteiligt war.

Die Einhaltung der Anforderungen wird während der Versuchsdurchführungsphase an den wöchentlichen Projektsitzungen geprüft.

Zum Ende des Projekts führt ein Projektmitglied die Systemtests durch und schreibt ein Testprotokoll. Das andere Teammitglied reviewt das Testprotokoll.

Zusätzlich wird noch eine Auswertung der Kennzahlen, die vom pqc-Tool erzeugt werden gemacht und diese dann mit den Anforderungen verglichen. Die Differenz wird im Projektbericht festgehalten.

12.3.2 Test Zeitplan

Bei *M2* wird das Kapitel 10 und damit alle funktionalen und nichtfunktionalen Anforderungen abgenommen.

Bei *M3* wird der aktuelle Stand der Implementation den Anforderungen gegenübergestellt und die Differenz zwischen Zustand und Ziel bestimmt. Dies geschieht in informeller Weise.

Bei *M4* wird die Implementation nochmals informell mit den Anforderungen verglichen und ggf. noch Änderungen daran vorgenommen, falls dies notwendig ist.

Bei *M5* muss das Protokoll der Systemtests in überarbeiteter Form vorliegen. Weiter muss die Auswertung der Kennzahlen und der Vergleich mit den Anforderungen fertiggestellt sein.

[1] Siehe Abschnitt 10.4.
[2] Siehe Abschnitt 10.5.

12.3.3 Tools, Techniken, Methoden und Metriken

Für die Architekturtests wird das Testprotokoll in I verwendet.

Der erste Teil der Systemtests, die Optimierung der Parameter, muss so dokumentiert werden, dass die Herleitung der Resultate nachvollziehbar wird. Dazu sollen auch alle Metriken und Entscheidungen gut dargestellt, erklärt und begründet werden.

Beim zweiten Teil der Systemtests, der Gegenüberstellung von Anforderungen und Endresultate der Versuche, muss die Differenz zwischen diesen klar ersichtlich sein.

12.4 Testaufgaben

Die unten beschriebenen Aufgaben dienen der Einhaltung der vorausgesetzten Anforderungen und Qualitätskriterien.

12.4.1 Ad-Hoc Tests

Währen der Versuchsdurchführungsphase werden nach Bedarf Ad-Hoc Tests durchgeführt. Deren Inhalt und Ausmass folgen dem ermessen der Projektmitglieder.

12.4.2 Architekturtests

Für die Architekturtests wird das Protokoll in I ausgefüllt. Dabei soll sichergestellt werden, dass alle Kriterien aus Abschnitt 11.3 eingehalten werden.

12.4.3 Systemtest

Der Systemtest wird im Rahmen von Kapitel 14 durchgeführt. Dabei muss dokumentiert werden, welche Parameter verwendet werden, wieso diese verwendet werden, welche Kennzahlen mit diesen zustande kommen, was diese bedeuten und ein Vergleich derselben mit den Anforderungen aus Kapitel 10.

Dabei muss klar ersichtlich sein, wie gut jede Anforderung erfüllt ist. Zudem sollen Vorschläge gemacht werden, wie die Resultate zustande kommen und wie sie verbessert werden können.

Kapitel 13

Umsetzung

Die Umsetzung erfolgte in drei Phasen. In der ersten Phase wurden die notwendigen Trainingsdaten beschafft. In der zweiten Phase wurde das PQC Tool entwickelt. In der letzten Phase wurden mithilfe von PQC und verschiedenen Shell-Scripts Daten Erhoben, die in der Ergebnisdiskussion (Siehe Kapitel 14) benötigt wurden.

In den folgenden Abschnitten werden alle Phasen kurz beschrieben. Die Beschreibung der zweiten Phase ist in der Form einer allgemeinen Beschreibung bestimmter Aspekte des verwendeten CNNs (Inception) gehalten.

13.1 Trainingsdaten

Im Rahmen dieser Arbeit wurden zwei Fotoserien von Packungen gemacht. Die 1. Fotoserie enthält Aufnahmen von 136 defekten und 30 intakten Packungen. Die 2. Fotoserie enthält Aufnahmen von 341 defekten und 136 intakten Packungen. Die Fotoserien unterscheiden sich in mehreren Attributen voneinander.

Die 1. Fotoserie wurde unter teilweise stark unterschiedlichen Lichtverhältnissen, auf einem hauptsächlich aber nicht ausschliesslich weissen Hintergrund aufgenommen. Die Auflösung der Aufnahmen in der ersten Serie ist 3264×2448 Pixels.

Die Aufnahmen der zweiten Serie wurden alle unter ähnlichen Lichtverhältnissen, auf einem schwarzen Hintergrund aufgenommen. Die Auflösung der Aufnahmen in der zweiten Serie ist 2272×1704 Pixels.

Bei der ersten Serie konnten folgende Probleme festgestellt werden.[1] (1) Die Menge von Trainingsdaten ist insgesamt zu klein. (2) Die Menge von intakten Aufnahmen ist insbesondere zu klein. (3) Durch den unregelmässigen Hintergrund haben die Aufnahmen viel Noise. (4) Dasselbe gilt für die Schatten der

[1] Da das verwendete Modell (Inception) als CNN dem Visuellen Kortex nachempfunden ist, wurde davon ausgegangen, dass diejenigen Faktoren, die die Klassifizierung der Aufnahmen durch Menschen vereinfachen (z.B. höherer Kontrast von den Objekten zum Hintergrund), das selbe für Inception leisten.[8]

(a) Defekt (b) Defekt

(c) Defekt (d) Defekt

Abbildung 13.1: Defekte Packungen (1. Fotoserie)

Packungen. (5) Die Lichtverhältnisse zwischen den Aufnahmen waren zu unterschiedlich.

Problem 1 wurde bei der zweiten Serie so angegangen, dass alle Packungen mehrfach fotographiert wurden. Die Annahme war, dass sich mehrere Aufnahmen einer einzelnen Packung, durch deren Handhabung visuell genügend unterscheiden würden, sodass insgesamt weniger Packungen benötigt wurden.

Problem 2 wurde einerseits mit der oben genannten Vorgehensweise und andrerseits durch eine grössere Anzahl von Packungen gelöst.

Problem 3 und 4 wurden gelöst, indem ein neutraler mattschwarzer Hintergrund aus Fotokarton verwendet wurde. Dieser Hintergrund verringerte die Lichtreflektionen und die Schatten. Ausserdem konnte der Kontrast zwischen Packung und Hintergrund erhöht werden, was die Erkennung des Objekts vereinfacht.

Problem 5 wurde ansatzweise gelöst, indem die Aufnahmen an einem Ort aufgenommen wurden, an dem während der Aufnahmen ähnliche natürliche Lichtverhältnisse (Sonneneinstrahlung) herrschten. Eine geeignete Lichtanlage könnte unter Umständen noch bessere Resultate liefern.

Einen Mangel, welchen beide Fotoserien gemeinsam haben, ist die unkla-

(a) Intakt (b) Intakt

(c) Intakt (d) Intakt

Abbildung 13.2: Intakte Packungen (1. Fotoserie)

re Aufteilung der Aufnahmen in die beiden Kategorien *defekt* und *intakt*. Man vergleiche dazu Abbildung 13.1a und Abbildung 13.1d mit Abbildung 13.2c und Abbildung 13.2d. Alle vier dieser Aufnahmen sind aus der ersten Fotoserie. Die ersten beiden wurden jedoch als defekt eingestuft, während die anderen beiden als intakt gelten. Der Unterschied ist minimal und das obwohl die Aufteilung erneut gemacht wurde, nachdem festgestellt wurde, dass die Ursprüngliche Einteilung noch unklarer war. Dasselbe Phänomen lässt sich bei der zweiten Serie beobachten, wo Abbildung 13.3b und Abbildung 13.4a ähnlicher zueinander sind als viele andere Paare in der Menge der als defekt eingestuften Packungen. Ein Beispiel dafür sind Abbildung 13.3a und Abbildung 13.3c.

Die Überlegenheit der zweiten Serie über die erste lässt sich anhand der unterschiedlichen Metriken der mit ihnen trainierten Modellen feststellen. In Abbildung 13.5 werden die Metriken von zwei solchen Modelle miteinander verglichen. Alle Metriken haben sich stark verbessert. Es ist gut möglich, dass mit einer besseren Klassifizierung der Aufnahmen und noch geeigneteren Aufnahmen noch bessere Resultate erzielt werden können.

(a) Offensichtlicher Defekt (b) Subtiler Defekt

(c) Offensichtlicher Defekt (d) Offensichtlicher Defekt

Abbildung 13.3: Defekte Packungen (2. Fotoserie)

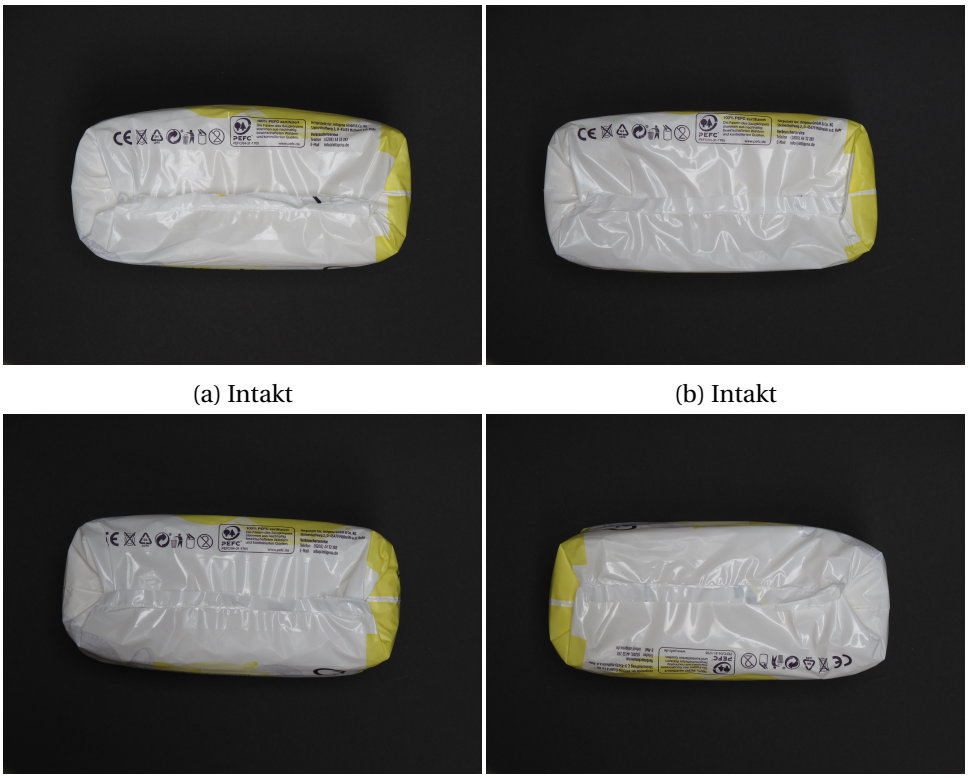
13.2 Inception

Inception ist eine CNN Architektur für Bilderkennung und -klassifizierung. Eine Ausprägung von Inception, genannt GoogLeNet, wurde bei der ILSVRC14 eingereicht und gewann in mehreren Kategorien. [2] Die Inception Architektur wird in [7] ausführlich beschrieben.

In PQC wird Inception v3 verwendet. Der Code der das Modell herunterlädt und den Obersten Layer neu trainiert ist von TensorFlow. Alle Layers von Inception von Anfang an zu trainieren kann Tage oder Wochen dauern und war deshalb nicht möglich.[10]

Google hat im August dieses Jahres (2016) die nächste Version von Inception (Inception-ResNet-v2) veröffentlicht, die noch mehr Layers als Inception v3 hat.[1] Mit diesem Modell, könnten die erzielten Resultate möglicherweise noch verbessert werden, sofern das die Qualität der Trainingsdaten zulässt.

[2] GoogLeNet gewann insbesondere in der Kategorie "Task 1b: Object detection with additional training data".[7]



(a) Intakt (b) Intakt

(c) Intakt (d) Intakt

Abbildung 13.4: Intakte Packungen (2. Fotoserie)

13.2.1 Neutrainieren der Höchsten Schicht

Das Inception Modell hat viele Schichten. Je höher eine Schicht desto höher ist der Abstraktionsgrad ihres Outputs. Wenn zum Beispiel ein niedrigerer Layer Kanten und Ähnliches erkennt, könnte ein höherer Layer abstraktere Konzepte, wie den Unterschied zwischen Hintergrund und Objekt davon ableiten. [3] Bei Inception findet die effektive Klassifizerung von Aufnahmen in der höchsten Schicht statt. Deshalb muss bloss diese Schicht neu trainiert werden, um dem Modell den Unterschied zwischen defekten und intakten Packungen beizubringen. Dadurch, dass nur die unterste Schicht des Modells trainiert werden muss, ist der Prozess des Trainierens verhältnismässig schnell (i.d.R. ca. 2–5min, das hängt aber von der Anzahl Trainingsschritten ab).

Die Inputdaten für die letzte Schicht des Modells werden Bottlenecks ge-

[3] "As these "Inception modules" are stacked on top of each other, their output correlation statistics are bound to vary: as features of higher abstraction are captured by higher layers, their spatial concentration is expected to decrease. [...] Furthermore, the design follows the practical intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from the different scales simultaneously." [7, S. 3–4]

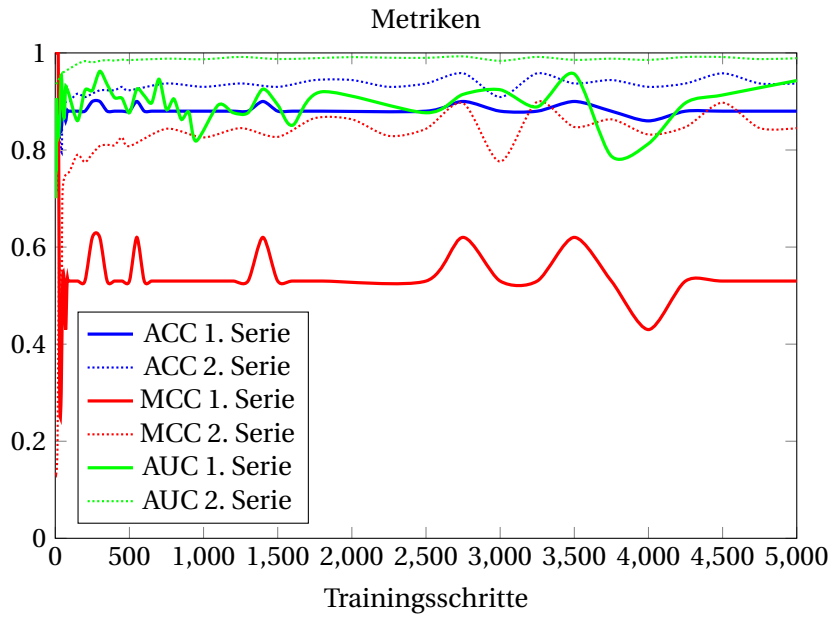


Abbildung 13.5: Vergleich der Metriken nach $x \in]0;5000]$ Trainingsschritten

nannt.[4] Der für das Retraining verwendete Code berechnet die Bottlenecks im Voraus. Sodass diese während dem Trainieren nicht bei jedem Schritt neu berechnet werden müssen.

Das Ziel des Trainings ist es die Kreuzentropie zu verkleinern. Bei jedem Trainingsschritt wird die Kreuzentropie mit der Softmax Funktion gemessen und mit dem Gradientenverfahren minimiert. (Siehe Listing 13.1.)

13.2.2 Testdaten

Die Testdaten haben die gleiche Struktur wie die Trainingsdaten. PQC teilt die klassifizierten Daten vor dem Trainieren zufällig (Gleichverteilt) in zwei Mengen: Trainingsdaten und Testdaten. Die Testdaten werden als Validierungsmenge benutzt um Metriken zur Performance eines trainierten Modells zu berechnen.

PQC verwendet standardmässig 30% der Trainingsdaten als Testdaten. Diese Daten werden nicht zum trainieren verwendet. Um die höchste Schicht von Inception v3 neu zu trainieren, wird das retrain.py Modul von TensorFlow verwendet. Dieses Modul verwendet weitere 10% der verbleibenden Trainingsdaten als Testdaten und daher nicht zur Trainierung des Modells. Dieser Faktor kann über einen Parameter eingestellt werden. Wird dieser Faktor zu klein eingestellt, resultiert das in einem ZeroDivisionError. Um dieses Problem zu lösen, müsste das retrain.py Modul kopiert und modifiziert werden. Es wurde entschie-

[4] Die zweithöchste Schicht wird als Bottleneck Schicht bezeichnet.[4]

```
1  final_tensor = tf.nn.softmax(logits, name=final_tensor_name)
2  tf.histogram_summary(final_tensor_name + '/activations', final_tensor)
3
4  with tf.name_scope('cross_entropy'):
5    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
6      logits, ground_truth_input)
7    with tf.name_scope('total'):
8      cross_entropy_mean = tf.reduce_mean(cross_entropy)
9    tf.scalar_summary('cross entropy', cross_entropy_mean)
10
11 with tf.name_scope('train'):
12   train_step = tf.train.GradientDescentOptimizer(FLAGS.learning_rate).minimize(
13       cross_entropy_mean)
```

Listing 13.1: Trainingsschritt: Minimierung der Kreuzentropie

den, dies nicht zu tun, um das Upgraden auf neue TensorFlow Versionen zu vereinfachen. Für den produktiven Einsatz wird allerdings empfohlen dies zu tun. Die 10% fallen i.d.R nicht stark ins Gewicht. Beim Standard Testdaten Faktor (30%) sind es lediglich 7% aller Trainingsdaten[5], die von retrain.py als Validierungsmenge gebraucht werden.

13.3 Datenerhebung für die Ergebnisdiskussion

Die Benutzerschnittstelle (Siehe Unterabschnitt 11.3.1) wurde so spezifiziert, dass es einfach ist PQC in Shell-Scripts zu verwenden. Zur Erhebung der Daten für die Ergebnisdiskussion (Siehe Kapitel 14) wurden mehrere Shell-Scripts geschrieben.

Im Folgenden wird eines dieser Shell-Scripts näher betrachtet.

Hier werden die Parameter definiert, die getestet werden sollen. In diesem Falle ist es die Anzahl der Testschritte (STEPS).

```
3  STEPS="$(seq 1 1 10) $(seq 20 10 100) $(seq 150 50 1000) $(seq 1100 100
       2000) $(seq 2250 250 5000)"
4  DATA="data.zip"
```

Listing 13.2: `test_steps.sh`: Zeilen 3–4

Als nächstes werden die klassifizierten Daten in Trainings- und Testdaten geteilt und die Bottleneckfiles generiert. Für alle Testdurchläufe soll die selbe Aufteilung verwendet werden.

```
8  setup ()
9  {
10     echo -n ">> Generate prototype ..."
11     if [ -f $PROTOTYPE/$PROTOTYPE.tgz ]
12     then
```

[5] $(100\% - 30\%) * 10\% = 7\%$

```
13          echo -e "\r>> Generate prototype [SKIPPED]"
14          return
15      fi
16      mkdir -p $PROTOTYPE
17      cp $DATA $PROTOTYPE
18      cd $PROTOTYPE
19      echo -n -e "\r>> Generate prototype {INITIALIZING...}"
20      pqc init &> /dev/null
21      echo -n -e "\r>> Generate prototype {DIVIDING...}"
22      pqc divide 0.3
23      echo -n -e "\r>> Generate prototype {TRAINING...}"
24      pqc train --steps 1 &> ../prototype.train.log
25      echo -n -e "\r>> Generate prototype {PACKING...}"
26      tar czf $PROTOTYPE.tgz bottlenecks data test inception .pqc
27      cd ..
28      echo -e "\r>> Generate prototype [DONE]"
29  }
```

Listing 13.3: `test_steps.sh`: Zeilen 8–29

Im nächsten Schritt wird das Modell mit allen zuvor definierten Trainingsschrittzahlen trainiert und validiert. Die Ausgaben vom Training und der Validierung werden in Files gespeichert. Erstere für das debugging und letztere für die spätere Aggregierung der Daten.

```
33  for I in $STEPS
34  do
35      echo -n ">> With $I steps ..."
36      if [ -d $I ]
37      then
38          echo -e "\r>> With $I steps [SKIPPED]"
39          continue
40      fi
41      mkdir $I
42      cp $PROTOTYPE/$PROTOTYPE.tgz $I
43      cd $I
44      tar xzf $PROTOTYPE.tgz
45      rm -rf $PROTOTYPE.tgz
46      echo -n -e "\r>> With $I steps {TRAINING...}"
47      pqc train --steps $I &> ../$I.train.log
48      echo -n -e "\r>> With $I steps {VALIDATING...}"
49      pqc test |& sed "/>>/d" > ../$I.log
50      rm -rf data test inception bottlenecks
51      cd ..
52      echo -e "\r>> With $I steps [DONE]";
53  done
```

Listing 13.4: `test_steps.sh`: Zeilen 33–53

Die beiden Prozeduren extract und extractP dienen dem Extrahieren von Validierungsresultaten aus den Test Logs.

```
55  extract() {
56      KEY=$1
57      VALUE=$(grep "$2" ${1}.log | cut -d: -f 2 | cut -d% -f 1 | cut -ds -f1)
```

```
58      if [ -n "$VALUE" ]
59      then
60          echo -n "(${KEY},${VALUE})"
61      fi
62  }
```

Listing 13.5: `test_steps.sh`: Zeilen 55–62

Bei extractP werden im Gegensatz zu extract die Werte durch 100 dividiert, da es sich bei diesen Werten um Prozentwerte handelt.

```
64  extractP() {
65      KEY=$1
66      VALUE=$(grep "$2" ${1}.log | cut -d: -f 2 | cut -d% -f 1 | cut -ds -f1)
67      if [ -n "$VALUE" ]
68      then
69          VALUE=$(echo "2k0 0 $VALUE 100 /p" | dc)
70          # 0 0 is a workaround for mcc sometimes being sub-zero
71          echo -n "(${KEY}, ${VALUE})"
72      fi
73  }
```

Listing 13.6: `test_steps.sh`: Zeilen 64–73

Als nächstes werden alle Files für die Reports erstellt oder geleert.

```
76  echo > avgt.data
77  echo > acc.data
78  echo > fpr.data
79  echo > fnr.data
80  echo > mcc.data
81  echo > auc.data
```

Listing 13.7: `test_steps.sh`: Zeilen 76–81

Dann werden für jede Trainingsschrittzahl die Testresultate in die Reports geschrieben.

```
82  for I in $STEPS
83  do
84      extract $I "Average" >> avgt.data
85      extractP $I "Accuracy" >> acc.data
86      extractP $I "False pos" >> fpr.data
87      extractP $I "False neg" >> fnr.data
88      extractP $I "Matthews" >> mcc.data
89      extract $I "Area under" >> auc.data;
90  done
```

Listing 13.8: `test_steps.sh`: Zeilen 82–90

Die Reports können direkt mit pgfplots verwendet werden um Diagramme zu erstellen. Ein Solches Diagramm ist zum Beispiel in Abbildung 13.5 zu sehen.

Kapitel 14

Ergebnisdiskussion

14.1 Auswertung

PQC erlaubt es anhand von drei verschiedenen Parametern Einfluss auf die Tests zu nehmen. Nachfolgend werden diese kurz erläutert und es wird erklärt, welche Bestrebungen unternommen wurden, diese zu optimieren, bzw. weshalb die bestimmten Standardwerte festgesetzt wurden.

14.1.1 Aufteilungsverhältnis

PQC bietet die Möglichkeit das Aufteilungsverhältnis zwischen Trainings- und Testdaten vor dem Trainieren des Modells zu bestimmen. Um die Auswirkung der Aufteilung auf die Qualität des Modells aufzeigen zu können, wurde eine Reihe von Tests mit unterschiedlichen Verhältnissen durchgeführt. Dabei wurden Tests mit einem Anteil von 10% Testdaten an den klassifizierte Daten bis zu einem Anteil von 80% Testdaten in 10% Schritten gemacht. Ein grösseres Verhältnis als 80% Testdaten ist nicht möglich, da dadurch in den vorhandenen klassifizierten Daten mit ca. 470 Aufnahmen die Datenmenge der Trainingsdaten zu gering wird und zu wenig Aufnahmen für das Trainieren des Modells zur Verfügung stehen. Die minimale Anzahl Datensätze pro Klasse, welche als Trainingsdaten gebraucht werden ist 20. Für die Anzahl Trainingsschritte und den Threshold wurden jeweils die Standardwerte verwendet, (1000 Trainingschritte, 0.8 Threshold).

Resultate

Da die Testdaten nach dem Zufallsprinzip (gleichverteilt) aus den Trainingsdaten ausgewählt werden, können die Modelle von sehr unterschiedlicher Qualität sein.

Aus diesem Grund und vor allem durch den bereits erklärten Noise in den Trainingsdaten kann dies zu sehr starken Schwankungen in den erreichten Werten bei den Resultaten führen.

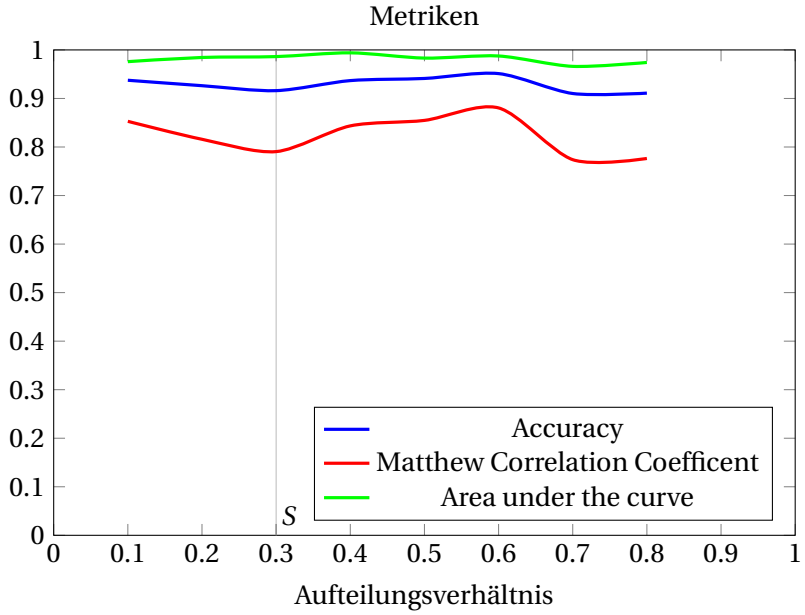


Abbildung 14.1: Metriken Aufteilungsverhältnis

Eine Aussage welche aufgrund der Resultate aus den Tests jedoch getroffen werden kann, ist die, dass ein zu grosses Verhältnis von Testdaten oder Trainingsdaten zu Problemen führen kann. Bei einem zu grossen Anteil von Testdaten an der verwendeten Fotoserie, kommt es ab einem gewissen Punkt zum Problem, dass nicht mehr genügend Aufnahme zum Trainieren vorhanden sind. Das trainierte Modell weist dadurch nicht die benötigte Qualität auf oder aber sogar TensorFlow selbst reklamiert, da zu wenig Trainingsdaten vorhanden sind.

Bei einem zu grossen Anteil an Trainingsdaten kann hingegen ein anderes Problem auftreten. Durch den hohen Anteil an Trainingsdaten ist die Qualität des Modells möglicherweise besser, aufgrund der geringen Testdatenmenge sinkt jedoch ebensfalls die Aussagekraft der Auswertungen mit PQC.

Schlussfolgerung

Die Schlussfolgerung um den Standardwert für das Aufteilungsverhältnis festzulegen wird in diesem Fall nicht aus den Resultaten der oben beschriebenen Versuchsreihe gezogen, sondern aus den während der Arbeit mit TensorFlow und PQC gesammelten Erfahrungen.

Gemäss den folgenden Überlegungen wurde ein Verhältnis von 30% Testdaten zu 70% Trainingsdaten gewählt. Es soll ein Modell welches eine gewisse Qualität aufweist erreicht werden, welches auch möglichst die in NFR–02 beschriebenen Ziele erreicht. Damit die erzielten Werte in der Auswertung auch eine gewisse Signifikanz erhalten, müssen jedoch auch genügend Testdaten zur

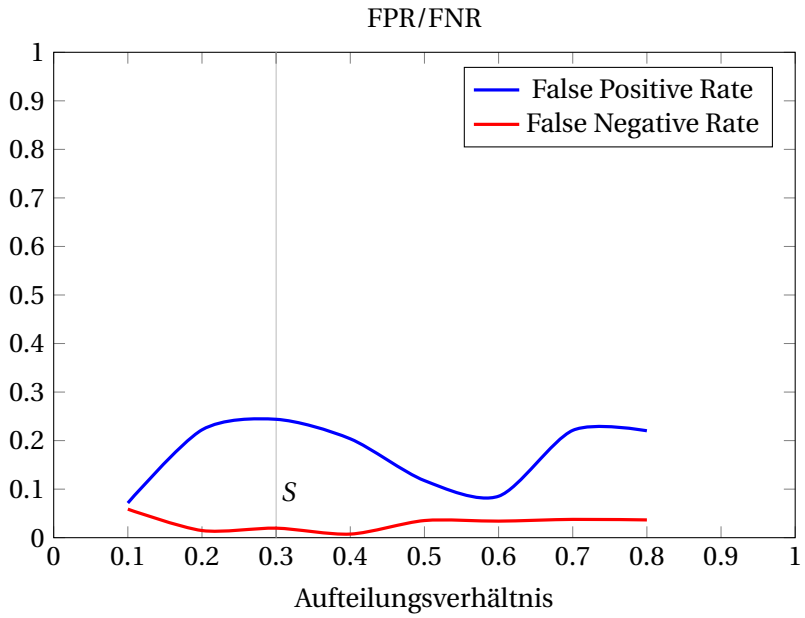


Abbildung 14.2: FPR/FNR Aufteilungsverhältnis

Verfügung stehen. Aus dem festgelegten Wert folgt also eine Art Tradeoff zwischen Aussagekräftigen Testresultaten und einem Modell von guter Qualität.

14.1.2 Trainingsschritte

Mit Hilfe von PQC wurden Tests durchgeführt, welche die Auswirkung der Anzahl Trainingsschritte auf die Qualität des Modells aufzeigen sollten. Es wurden Tests mit 5 bis 50 Trainingsschritten im Abstand von 5 Schritten und ab 50 bis 5000 Trainingschritten im Abstand von 50 Schritten durchgeführt. Für die Datenaufteilung und den Threshold wurden jeweils die Standardwerte verwendet, (30% Testdaten, 70% Trainingsdaten, 0.8 Threshold). Alle Tests wurden jeweils mit denselben klassifizierten Daten durchgeführt.

Resultate

In Abbildung 14.3 ist erkennbar, dass sowohl ACC als auch MCC sich mit der Anzahl Trainingsschritten verbessern. Zu Beginn kann ein starker Anstieg der Metriken beobachtet werden, welcher ab einer Anzahl von ca. 500 Trainingschritten jedoch abflacht und ab ca. 1000 Schritten nur noch durch sogenannten Noise beim Trainieren Schwankungen aufweisst.

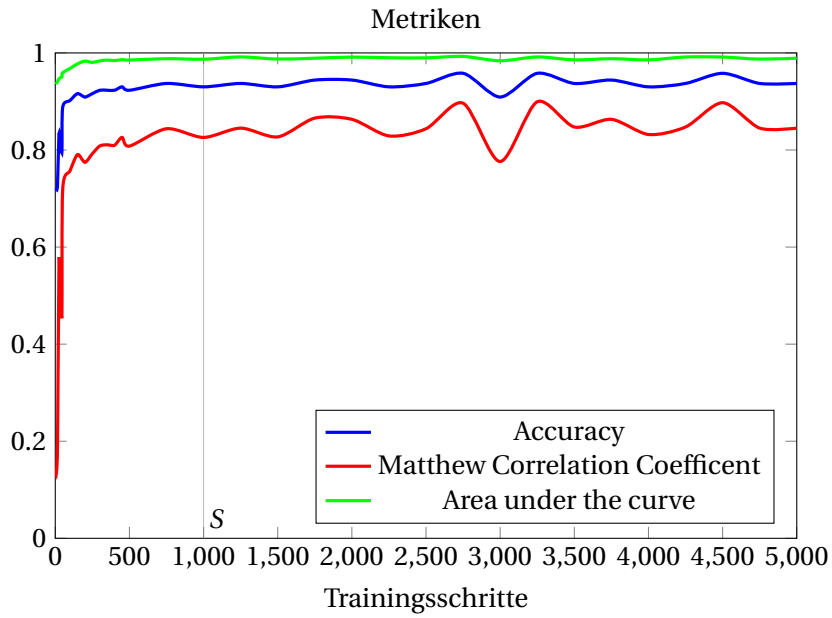


Abbildung 14.3: Metriken Trainingsschritte

Das Diagramm in Abbildung 14.4 zeigt die Auswirkungen der Anzahl Trainingschritte auf FPR und FNR. Mit sehr wenigen Trainingsschritten werden die Packungen von PQC klar als Defekt klassifiziert. Dies zeigt sich anhand der FPR welche bei 5 Trainingsschritten einen Wert von beinahe 100% aufweist. Mit der Erhöhung der Anzahl Trainingschritte flacht dieser Wert jedoch sehr schnell ab und steigt ab ca. 500 Schritten auch durch Schwankungen nicht mehr über 20%.

Es wird vermutet, dass das vor allem dem Noise in den Trainingsdaten geschuldet ist. Die FNR ist zu Beginn bei 0% und zeigt auch mit der Erhöhung der Trainingsschritte keine signifikante Veränderung und bleibt stets unter einem Wert von 5%. Auch hier wird die Ursache der Schwankungen beim Noise vermutet.

Schlussfolgerung

Die Tests haben gezeigt, dass die Anzahl der Trainingsschritte einen starken Einfluss auf die Qualität des Modells haben kann. Es hat sich aber ebenso gezeigt, dass ab einer gewissen Anzahl ein Maximum der Verbesserbarkeit erreicht ist. So zeigte auch ein Modell, welches mit 100'000 Trainingschritten trainiert wurde keinerlei Verbesserung in der Qualität gegenüber einem Modell welches mit lediglich 4000 Schritten trainiert wurde.

Die Anzahl ist klein genug, dass die Traingszeit in einem kleineren Rahmen bleibt, jedoch gross genug, dass das umtrainierte Modell bereits an die in NFR–02 bestimmten Ziele herankommt.

Als Standardwert für die Anzahl Trainingschritte welche PQC verwendet um

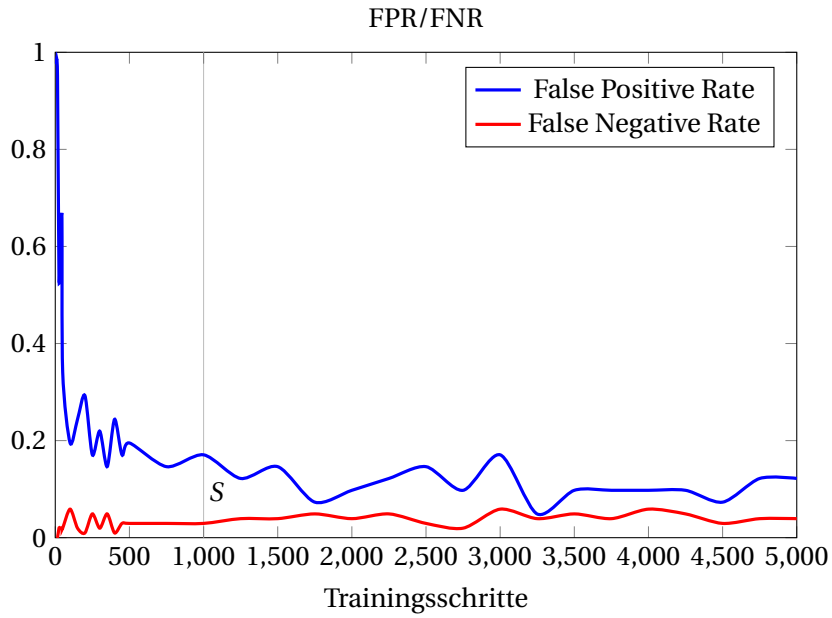


Abbildung 14.4: FPR/FNR Trainingsschritte

das Modell neu zu trainieren, wird 1000 gesetzt. Siehe die entsprechende Markierung in Abbildung 14.3 und Abbildung 14.4. Um die Qualität des Modells weiter zu verbessern muss daher bei anderen Faktoren angesetzt werden, z.B. bei der Bildqualität. Die Möglichkeiten bei der Anzahl Trainingsschritten sind im beschriebenen Rahmen ausgeschöpft worden.

14.1.3 Threshold

Eine Anforderung welche zum Beginn der Arbeit gestellt wurde, war es, Packungen tendenziell eher als defekt anstatt intakt einzustufen. Es sollen möglichst keine defekten nach der Überprüfung durch PQC mehr in den Umlauf gelangen. Es ist einfacher aussortierte Packungen wieder in den Produktionsablauf einzuführen als defekte Packungen vor dem Versand an den Kunden doch noch heraus zu fischen.

Das trainierte Modell bewertet jede Packung aufgrund der trainierten Eigenschaften mit welcher Wahrscheinlichkeit eine Packung als intakt oder defekt einzustufen ist. Der Threshold dient bei der Auswertung als Schwellenwert, den die Einstufung als intakt überschreiten muss, damit eine Packung auch als eine intakte Packung gewertet wird.

Mit Hilfe von PQC wurde eine Testserie durchgeführt, bei welcher die Auswirkung, den die Verschiebung des Threshold auf die Metriken hat, festgestellt werden sollte. Dabei wurden Tests mit einem Threshold von 1% bis zu einem Threshold von 99% in 1% Schritten gemacht. Für die Datenaufteilung und den

die Anzahl Trainingsschritte wurden jeweils die Standardwerte verwendet, (30% Testdaten, 70% Trainingsdaten, 1000 Trainingsschritte). Alle Tests wurden jeweils mit den selben klassifizierte Daten durchgeführt.

Resultate

In Abbildung 14.5 ist klar erkennbar, welchen Einfluss der Threshold auf die Werte der Metrikresultate hat. ACC wie auch MCC steigen zuerst stark an, da auch defekte Packungen zu einem gewissen Anteil als intakt eingestuft werden können. Zu Beginn werden praktisch alle Packungen zuerst einmal als intakt eingestuft, da der Threshold schlicht und einfach eine zu tiefe Grenze darstellt, die eine Packung erreichen muss um als intakt gezählt zu werden.

Bis zu einem Wert von 50% flacht sich die Kurve immer stärker ab und erreicht ab 60% bis 80% ein relativ konstantes Hoch bei ACC wie auch MCC. Ab einem Wert von über 80% ist dann wieder ein starkes Gefälle zu beobachten. Dies erklärt sich daher, dass der Threshold zu hoch angesetzt wird. Immer mehr Packungen können nicht mehr als intakt eingestuft werden, da sie von TensorFlow mit einem zu geringen Wert als solche klassifiziert werden.

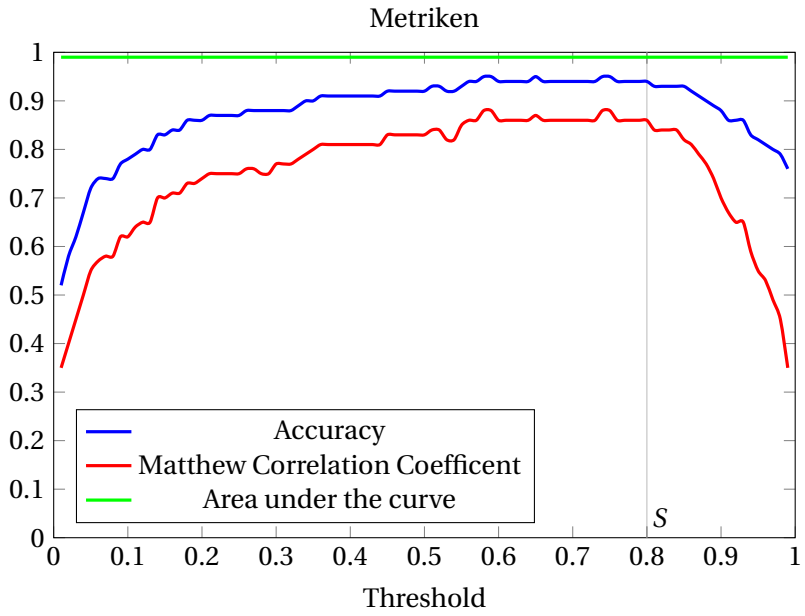


Abbildung 14.5: Metriken Threshold

Abbildung 14.5 zeigt sehr schön die Auswirkungen des Thresholds auf die klassifizierung in intakt und defekt. Zu beginn ist der Threshold sehr niedrig, fast alle Packungen werden als intakt gewertet, weshalb die FNR sehr hoch ist und FPR schwindend niedrig ist. Bei ca. 55% treffen sich die beiden Kurven, es ist in etwa ausgeglichen zwischen Falschzuweisungen in intakt und defekt.

Durch eine weitere steigerung des Threshold steigt auch der Wert von FPR. Aufgrund einer sehr grossesn Schnittmenge an Eigenschaften in den Aufnahmen zwischen den beiden Klassen kann keine eindeutige Zuweisung von eigentlich intakten Packungenn erfolgen. Sie weisen schlicht zu wenig eindeutige Eigenschaften auf, um sie entsprechend zuordnen zu können. Der gleiche entgegengesetzte Effekt kann auch bei FNR beobachtet werden.

Ab einem Wert von über 80% steigt FPR sehr stark an. Dies ist auf die gleichen Erklärung zurück führbar wie auch schon die starke Veränderung in Abbildung 14.5 ab dem selben Wert erkennbar ist. Zu wenige intakte erreichen eine so gute Bewertung durch TensorFlow, dass sie die Grenze des Threshold überspringen können.

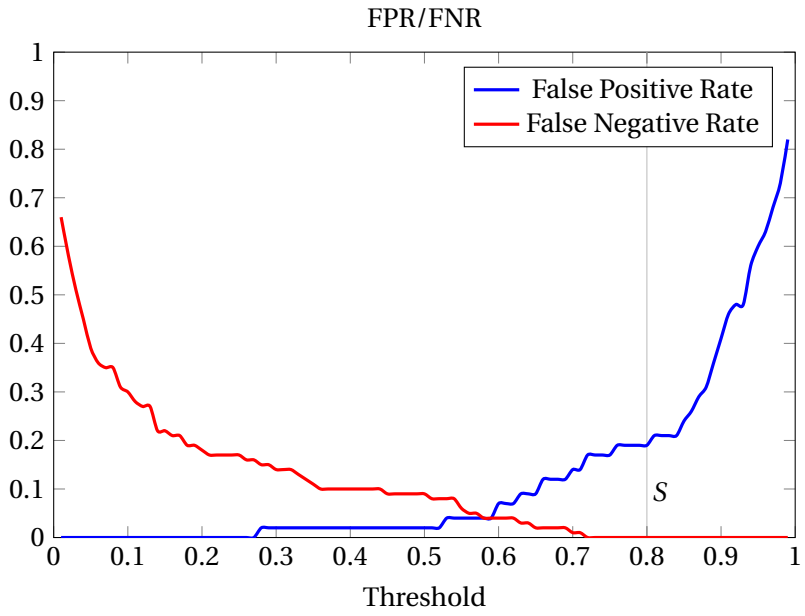


Abbildung 14.6: FPR/FNR Threshold

Schlussfolgerung

Mit einem zu niedrigem Threshold werden zu viele defekte Packungen, aufgrund des Problems mit dem Noise, als intakt eingestuft. Sobald der Threshold jedoch zu hoch angesetzt wird, können zu viele der Packungen aufgrund der Klassifizierung nicht mehr als intakt eingestuft werden, obwohl sie eigentlich als eher intakt klassifiziert wurden.

Auch der Threshold-Standardwerts wird wieder über ein Tradeoff zwischen FNR und FPR sowie den restlichen Metriken festgelegt. Hauptaugenmerk wird dabei auf Abbildung 14.6 gelegt. FNR soll möglichst niedrig ausfallen, gleichzeitig darf FPR jedoch auch nicht zu gross werden. Die Metriken in Abbildung 14.5 wurden zwar für die Festlegung ebenfalls beachtet, da der Threshold jedoch der

Parameter ist, mit welchem am direktesten auf die Werte von FPR und FNR eingewirkt werden kann, fallen sie nicht so stark ins Gewicht. Ziel ist es die geforderten Ziele in NFR–02 zu erreichen. Somit wird als Standardwert für den Threshold in PQC ein Wert von 0.8 gesetzt. Siehe die entsprechende Markierung in Abbildung 14.5 und Abbildung 14.6.

14.2 Zielerreichung

14.2.1 FRV-01

Das Ziel von FRV–01 wurde erreicht. NFR–02 wie auch FRS–01 konnten mit dem Prototypen PQC umgesetzt werden. Auch wenn für NFR–01 nur das geforderte Minimalziel erreicht wurde, kann FRV–01 unter den gegebenen Bedingungen als erfüllt betrachtet werden.

14.2.2 FRE-01

FRE–01 wurde in dem Sinne umgesetzt, dass PQC in der Lage ist, auf Filesystemebene zu unterscheiden, was als Trainingsdaten definiert wurde, solange die vorgegebene Dateistruktur eingehalten wird.

Ein Projektmitarbeiter ist in der Lage von Hand Aufnahmen von Packungen im Ordner *data* entsprechend ihrer Klasse in die Ordner intakt und defekt einzuordnen und so die Trainingsdaten zu definieren. Zusätzlich wurde die Möglichkeit geschaffen mit Hilfe von *divide* und *reshuffle* die vorhandenen Aufnahme automatisch und zufällig in Lern- und Testdaten aufteilen zu lassen.

14.2.3 FRE-02

FRE–02 wurde im selben Sinn umgesetzt wie auch FRE–01. PQC in der Lage ist, auf Filesystemebene zu unterscheiden, was als Testdaten definiert wurde, solange die vorgegebene Dateistruktur eingehalten wird.

Ein Projektmitarbeiter ist auch hier in der Lage von Hand Aufnahmen von Packungen im Ordner *test* entsprechend ihrer Klasse in die Ordner intakt und defekt einzuordnen und so die Testdaten zu definieren. Zusätzlich besteht auch hier die Möglichkeit mit Hilfe von *divide* und *reshuffle* die vorhandenen Aufnahmen automatisch und zufällig in Lern- und Testdaten aufteilen zu lassen.

14.2.4 FRE-03

Wie in FRE–03 gefordert ist es mit PQC möglich, tests gegen das trainierte Modell laufen zu lassen. Durch den Aufbau von PQC ist es möglich Tests auch mehrfach zu wiederholen oder auch die Testdaten von Hand anzupassen. Durch den in FRE–04 geforderten Report ist es möglich die Anforderungstreue des Modells zu prüfen.

14.2.5 FRE-04

PQC gibt nach jedem Testdurchlauf die erreichten Metriken aus. Durch PQC werden automatisch ACC, MCC, AUC sowie die Durchschnittliche Zeit welche für die klassifizierung einer Packung benötigt wurde und auch die Werte der Confusion Matrix [12] berechnet. FRE–04 ist daher als erfüllt zu betrachten.

14.2.6 FRS-01

Es ist mit PQC nicht direkt möglich nur eine klassifizieren zu können. Da das definierte Ziel in FRS–01 jedoch ist, das trainierte Modell testen zu wollen, kann dieses Requirement zumindest teilweise als erfüllt betrachtet werden.

Um ein Modell ausführlich zu überprüfen, muss es mit jeweils einer grösseren Menge von Aufnahmen getestet werden, um eine sinnvolle Auswertung machen zu können. Dafür werden jeweils alle als Testdaten definierten Aufnahmen bei einem Testdurchlauf durch PQC klassifiziert und aus den Resultaten und der automatisch erfolgten Auswertung kann die Qualität des trainierten Modells bestummen werden. Das Feature nur ein einzelnes Bild von PQC klassifizieren zu lassen, ohne berechnen von Metriken, kann für eine weiterführende Arbeit in betracht gezogen werden.

14.2.7 FRS-02

PQC berechnet nach einem Testdurchlauf jeweils automatisch verschiedene Metriken. Unter anderem auch in Confusion Matrix definierte Fehlerraten[12]. FNR und FPR sind dabei direkt ablesbar und FRS–02 gilt damit als erfüllt.

14.2.8 NFR-01

NFR–01 galt während der Optimierungsphase des Projekts das Hauptaugenmerk. Es wurde versucht mit mehreren Parameteranpassungen den Klassifizierer zu optimieren. Mit der Zeit kristallisierte sich jedoch heraus, dass nicht unendlich über die vorhandenen Parameter optimiert werden kann. Zu einem erheblichen Teil sind die erreichten Zahlen auch von der Qualität der verwendeten Trainingsdaten abhängig.

PQC konnte jedoch so weit optimiert werden, dass das angestrebte Minimalziel von einer FNR unter 1.5% erreicht wurde. Es stellte sich jedoch als äusserst schwierig heraus gleichzeitig auch FPR unter dem angestrebten Ziel von 7.5% zu halten. Dies liegt unter anderem auch an der grossen Schnittmenge zwischen intakten und defekten, also nicht klar zuweisbaren, Packungen.

14.2.9 NFR-02

NFR–02 fordert von PQC 120 Packungen pro Minute klassifizieren zu können oder anderst ausgedrückt nicht mehr als 0.5 Sekunden pro Packung für die klas-

sifizierung zu benötigen.

Auf einem von der HSR zur Vergügung gestellten Arbeitsplatzrechner mit einer Intel Xeon E3-1245 CPU mit 3.4GHz und 16GB Arbeitsspeicher unter einem Debian Sid Betriebssystem wurden durchschnittliche Zeiten von unter 0.2 Sekunden erreicht.

Auch in einer Virtuellen Maschine (Ubuntu 16.04 mit 8 Cores und 8GB Arbeitsspeicher) welche auf einem gleichen Rechner wie oben beschrieben, unter Windows 7 betrieben wurde, konnten Zeiten von unter 0.3 Sekunden erreicht werden. Selbst wenn die beschriebene VM nur auf einem Laptop (Lenovo T440s mit i7U CPU) und mit 4 Cores betrieben wird, werden Zahlen deutlich unter 0.4 Sekunden erreicht.

Ob das in NFR–02 geforderte Ziel erreicht wird, hängt sehr stark von den Spezifikationen des verwendeten Testsystems ab. Es hat sich jedoch gezeigt das kein besonders hochgezüchtetes System benötigt wird um das gefordete Ziel zu erreichen.

14.3 Bewertung

Das Ergebnis der Arbeit ist zufriedenstellend. Ein Problem, welches sich stellte, waren die bereits beschriebenen unklaren Definitionen, bzw. die stark verschwommene Trennungslinie zwischen den Klassen. Im Rahmen dieser Machbarkeitsstudie wurde gezeigt, wie die Problemstellung mit der Hilfe von state of the art Bildklassifizierung mittels Deep Learning gelöst werden kann.

14.3.1 Standardwerte

Aufgrund der vorgenommenen Tests (Siehe Abschnitt 14.1) konnten für die verschiedenen Parameter sinnvolle Standardwerte definiert werden. Festzuhalten ist jedoch, dass diese nicht in Stein gemeiselt sind. Die Werte wurden jeweils mit dem Fokus, die Resultate für die Trainingsdaten aus der 2. Fotoserie zu optimieren, bestimmt. Sie sind daher eher als Richtwerte zu verstehen. Die Qualität sowie auch andere Eigenschaften der Aufnahme, wie Schatten oder Stärke der Beleuchtung und Reflektion, können einen grossen Einfluss auf die Resultate haben. Die optimalen Werte müssen daher für jede neue Fotoserie, sowie auch für jedes neue Packungsdesign oder Änderung der Aufnahmebedingungen neu evaluiert und angepasst werden.

14.3.2 Anforderungen und Ziele

Die Anforderungen konnten grösstenteils erfüllt werden. So konnte zum Beispiel NFR–02 bereits mit einer schnelleren Standard-CPU erreicht, bzw. deutlich unterboten werden. Andere Anforderungen konnten leider nicht vollständig umgesetzt werden. Alle funktionale Anforderungen konnten jedoch beinahe

vollständig umgesetzt werden. Der in FRS–01 fehlende Teil, könnte zum Beispiel ohne viel Aufwand als Feature in PQC ergänzt werden.

NFR–01 kann wohl als für den Prototyp wichtigste Anforderung betrachtet werden. Sie kann leider nicht mit genügend Sicherheit als vollständig umgesetzt bezeichnet werden. Die erzielten Resultate sind jedoch, mit Blick auf die wenige zur Verfügung stehende Zeit zur Optimierung, zufriedenstellend. Der Prototyp kommt sehr nahe an NFR–01 heran. Es wurde eine starke Basis für die zukünftige Erreichung dieser Anforderung gelegt.

Zu Beginn dieser Semesterarbeit wurden drei Ziele definiert. (G–01, G–02 und G–03) G–01 wurde durch die Implementierung von PQC erreicht. G–02 wurde ebenfalls klar erreicht, wie das in Unterabschnitt 14.2.9 dargelegt wird. G–03 wurde nicht vollständig erreicht, jedoch wurde eine starke Annäherung erreicht, wie das in Unterabschnitt 14.2.8 aufgezeigt wird.

Kapitel 15

Zusammenfassung und Ausblick

Zu Beginn dieser Semesterarbeit stand die Frage, können aufgrund von Kameraaufnahmen mittels Software defekte von intakten Packungen unterschieden werden. Diese Frage muss, das wurde anhand eines Beispiels bewiesen, klar mit Ja beantwortet werden. Dieses Beispiel, PQC, wurde im Rahmen dieser Arbeit entwickelt, um diesen Beweis zu erbringen. Dazu wurden mit TensorFlow und Inception v3 Technologien eingesetzt, die dem aktuellen Stand der Technik auf dem Gebiet der Bildklassifizierung entsprechen.

Obwohl nicht jede Anforderung im Detail erfüllt werden konnten, so wurden doch sehr gute Resultate, vor allem bei der Klassifizierungsgeschwindigkeit erzielt. Zudem wurde die Basis für zukünftige Arbeiten zu dieser Problemstellung gelegt.

Es konnte festgestellt werden, dass die Qualität der Trainingsdaten, sowie die Schärfe der Klassen eine sehr grosse Rolle bei der Optimierung der Qualität des Klassifizierers spielen. Ausserdem konnte aufgezeigt werden, wie verschiedene Parameter für die Trainierung und die Benutzung des Modells optimiert werden können.

Für zukünftige Arbeiten zu dieser Problemstellung stellen sich also die folgenden Aufgaben:

1. Eine schärfere Definition der Klassen erstellen. Dies muss umbedingt in der Zusammenarbeit mit dem Auftraggeber geschehen.
2. Prüfen, ob mit höherern oder niedrigeren Auflösungen bessere Resultate erzielt werden können.
3. Die Aufnahmebedingungen für die Trainingsdaten optimieren.
4. Prüfen, wie und ob das Modell während seinem Einsatz weiter optimiert werden kann.
5. Ein Konzept für die Konkrete Umsetzung erstellen. Dabei müssen Faktoren wie Bewegungsunschärfe, Hintergrundgeräusche und Weitere berücksichtigt werden.

Anhang B

Reflektion

B.1 Florian Bitterlin

Insgesamt ist die Studienarbeit sehr gut verlaufen. Auch in die Zusammenarbeit mit CS hat sehr gut funktioniert. Wir konnten uns trotz des sehr unterschiedlichen Stundenplans gut über die gewählten Kommunikationswege koordinieren und verständigen.

Wir entschieden uns für eine analoge Aufgabenverwaltung. Dadurch waren wir an das Arbeitszimmer gebunden. Da wir jedoch meistens sowieso im Arbeitszimmer arbeiteten, entwickelte sich dies jedoch nicht zu einem grossen Problem. Dies zwang uns auch gemeinsam zu planen, wodurch beide immer im Blick hatten, welche Arbeiten der jeweils andere am erledigen war. Unseren Zeitplan konnten wir relativ gut einhalten. Lediglich bei der Implementation wäre etwas mehr Zeit nötig gewesen.

Wir erhielten leider erst kurz vor dem Beginn der Arbeit die Möglichkeit uns in die Aufgabenstellung einzulesen. Dadurch waren wir nicht in der Lage uns genügend auf das Projekt vorzubereiten. Da wir die ersten Wochen dazu verwendeten das Projekt zu planen und die Anforderungen formulierten, waren wir jedoch in der Lage Unklarheiten zu beseitigen und uns einen Überblick zu verschaffen.

Machine Learning und Image Processing war für uns beide ein eher neues Gebiet, in das wir uns zuerst einarbeiten mussten. Der Zeitplan erlaubte es uns jedoch nicht genug Zeit darauf zu verwenden und wir konnten den recht komplexen Themenbereich nur oberflächlich erforschen. Dies erschwerte uns die Recherchearbeit um die Technologien für die Umsetzung zu finden und führte auch zu späteren Zeitpunkten zu Unklarheiten.

In der Zusammenarbeit mit dem Industriepartner hätten ich mir gerne eine direktere Kommunkation gewünscht. Insbesondere bei der Beschaffung der Packungen für die Erstellung der Bilder gab es Unklarheiten. Insgesamt war die Kommunkation jedoch stets freundlich und es konnte sehr gut über die gegebene Aufgabenstellung und die Umsetzung diskutiert werden.

B.2 Cyrill Schenkel

Die Zusammenarbeit mit FB ist sehr gut verlaufen. Zum Beispiel gab es keine grösseren Kommunikationsschwierigkeiten. Die gewählten Hilfsmittel waren angemessen, obwohl die eher rudimentäre Aufgabenverwaltung auch ihre negative Kehrseiten hatte, zum Beispiel die örtliche Gebundenheit an den Arbeitsraum.

Da wir genauere Informationen erst zum 18. September 2016 erhalten haben, konnten wir uns davor inhaltlich in keinster Weise auf das Projekt vorbereiten. Wir konnten uns aber in der ersten Woche ein klareres Bild darüber machen, was die genaue Problemstellung hätte sein können und wie wir das Projekt angehen wollten. Die erste Woche auf die Ausarbeitung des Projektplans zu verwenden war eine gute Idee.

Es war teilweise eher schwierig Lösungsansätze und Lösungen zu finden, da wir uns zuvor nur sehr oberflächlich mit Machine Learning und Image Processing auseinandergesetzt haben. In der Zeit, die uns zur Verfügung stand, konnten wir uns nicht eingehend mit diesen Themenbereichen beschäftigen. Das hat während der Arbeit zu Unsicherheiten geführt.

Meine persönlichen Schlüsse für spätere Arbeiten (z.B. die BA) sind:

- Dokumentekonzept im Voraus erstellen.
- Projektplan im Voraus erstellen.
- Im Voraus in Themen einarbeiten, die mir nicht vertraut sind.

Es war eine gute Entscheidung die Anforderungen am Anfang des Projekts klar zu definieren. Das hat die Auswertung stark vereinfacht.

Der Zeitplan war insgesamt sehr gut. Die Zeit für die Implementation war allerdings eher knapp berechnet und wir hätten früher mit der Auswertung beginnen sollen. Dann hätten wir mehr Zeit für die Überarbeitung des Dokuments gehabt.

Anhang C

Zeitabrechnung

Die Zeiterfassung geschah in der Form eines Projekttagebuchs. So erfasste jedes Projektmitglied inidviduell in einem eigenen Dokument seine Arbeiten, welche es pro Tag erledigt hat und wieviel Zeit dafür aufgewendet wurde, siehe Anhang D und E. Dadurch ist einsehbar, wer an welchen Teilen der Arbeit beschäftigt war und wieviel Zeit dafür in Anspruch genommen wurde.

Aufgrund des Charakters der Arbeit bestand ein grosser Teil der Arbeit aus Recherche zum Thema, zu den möglichen einsetzbaren Technologien und dem Studium der Funktionsweise von TensorFlow selbst. Besonders in der ersten Phase der Versuchsdurchführung wurde deshalb ein grosser Teil der eigentlichen Arbeitszeit nicht erfasst, da dies zu Beginn nicht als Leistung angesehen wurde. Da dies jedoch trotzdem einen erheblichen Aufwand bedeutete fehlt dies nun teilweise in den vermerken in den Projekttagebücher.

Anhang D

Projekttagebuch Florian Bitterlin

D.1 Woche 1 (19. September - 25. September): Projektplan

D.1.1 Montag

Titel	Beschreibung	Aufwand
Kickoff Meeting	—	1h
Einrichtung Arbeitsplatz	Ich habe den Computer einerichtet sowie alle Software, die ich für die Arbeit brauche installiert und konfiguriert.	2h
Kickoff Protokoll	Ich habe das Protokoll für das Kickoff Meeting gereviewt und zusammen mit den Meetingsvorschlägen an Herrn Huser verschickt.	30m

D.1.2 Dienstag

Titel	Beschreibung	Aufwand
Arbeitsplannung	Taskmanagement definiert und Tasktafel vorbereitet	2h
Infrastruktur schreiben	Kapitel Infrastruktur im Projektplan geschrieben und angebrachte Änderungsvorschläge eingepflegt.	1.5h
Zeitliche Plannung	Review des Kapitels	30m

D.1.3 Mittwoch

Titel	Beschreibung	Aufwand
Arbeitspakete	Arbeitspakete für den Projektplan besprochen	30m

Projektübersicht schreiben	Kapitel Projektübersicht im Projektplan geschrieben und angebrachte Änderungsvorschläge eingepflegt.	2.5h
Qualitätsmassnahmen	Review des Kapitels	30m
Projektorganisation	Review des Kapitels	30m

D.1.4 Donnerstag

Titel	Beschreibung	Aufwand
Arbeitspakete	Kapitel Arbeitspakete gereviewt	30m
Einleitung schreiben	Kapitel Einleitung im Projektplan geschrieben.	30m
Projektübersicht	Angebrachte Änderungsvorschläge eingepflegt.	30m

D.1.5 Freitag

Titel	Beschreibung	Aufwand
Risikomanagement diskutiert	Diskussion welche Risiken im Projekt bestehen	30m
Risikomanagement schreiben	Kapitel Risikomanagement im Projektplan geschrieben.	1.5h

D.1.6 Sonntag

Titel	Beschreibung	Aufwand
Risikomanagement schreiben	Kapitel Risikomanagement im Projektplan geschrieben.	1h

D.2 Woche 2 (26. September - 2. Oktober): Versuchsplanung

D.2.1 Montag

Titel	Beschreibung	Aufwand
Review Projektplan	Review finalisierter Projektplan	30m
Installation Texlipse	Installation des neu verwendeten Latexeditors	1h

D.2.2 Dienstag

Titel	Beschreibung	Aufwand

Review Projektbericht	Review aufgesetzter Projektbericht	30m
Weekly	Meeting mit Herr Huser und Herr Jucker	1h

D.2.3 Mittwoch

Titel	Beschreibung	Aufwand
Motivation schreiben	Motivation in Bericht schreiben	1.5h
Build für Dokumente Aufsetzen	Begonnen Build für Dokumente aufzusetzen	2h

D.2.4 Donnerstag

Titel	Beschreibung	Aufwand
Webredirect auf Buildserver	Webredirect auf Buildserver eingerichtet um einfacher Zugang zu haben: [redacted]	1h

D.3 Woche 3 (3. Oktober - 9. Oktober): Versuchsplanung

D.3.1 Montag

Titel	Beschreibung	Aufwand
Besuch Fabrik	Besuch der Produktionsstätte in Mettmenstetten und anfertigen der Lerndaten.	3h

D.3.2 Dienstag

Titel	Beschreibung	Aufwand
Review Requirements	Review der Functional und Non-Functional Requirements	2h
Recherche	Recherche zu ML Libraries	2h

D.3.3 Mittwoch

Titel	Beschreibung	Aufwand
Recherche	Lesen der entsprechenden Kapitel in ML Paradigms	2h
Recherche	Recherche zu ML Libraries Online	2h

D.3.4 Donnerstag

Titel	Beschreibung	Aufwand
Anpassung Server	Ändern der Logwatch config und Software update	1h

D.3.5 Freitag

Titel	Beschreibung	Aufwand
Recherche	Recherche zu ML Libraries Online	2h

D.4 Woche 4 (10. Oktober - 16. Oktober): Versuchsplanung

D.4.1 Montag

Titel	Beschreibung	Aufwand
Weekly Meeting	Besprechen des Besuchs in der Fabrik, Probleme mit den Lerndaten und mögliche Lösungsansätze.	30m
Aufsetzen IDE für OpenCV	OpenCV integriert und Testprojekt aufgesetzt	30m
Feature Extraction mit OpenCV	Versuch der Kantenerkennung auf Paketen mit Hilfe von OpenCV	2h

D.4.2 Dienstag

Titel	Beschreibung	Aufwand
Meeting	Besprechen des Fortschritts mit CS und weitere Möglichkeiten mit anderen/besseren Fotos die Feature Extraction zu verbessern	30m
Feature Extraction mit OpenCV	Versuch der Kantenerkennung auf Paketen mit Hilfe von OpenCV	2h

D.4.3 Mittwoch

Titel	Beschreibung	Aufwand
Meeting	Besprechen des weiteren Vorgehens mit CS und des bereits erreichten. Aufteilung von Dokumentationsarbeiten um Entscheide festzuhalten.	30m

Aufsetzen Python-Umgebung mit OpenCV	Aufsetzen von Python auf dem Arbeitsplatz-PC um die Versuche von CS zu testen und zu erweitern.	2h
Dokumentation Entscheid CV-Framework	Die Problemstellung und die Entscheidung betreffend OpenCV für die Feature Extraction wurde im Bericht dokumentiert	2h

D.4.4 Donnerstag

Titel	Beschreibung	Aufwand
Sortieren Lerndaten	Ich habe die ca 170 Bilder aus der Fabrik neu sortiert, dass heisst neu klassifiziert nach defekt und intakt, um die Schnittmenge, welche unsere Meinung nach besteht, zu verkleinern und so eine klarere Linie zu haben, was defekt und was intakt ist.	45m
Aufsetzen TensorFlow	Ich habe eine Linux-VM aufgesetzt und darauf Python und Tensorflow installiert.	2h
Testen mit TensorFlow	Ich habe das Tutorial TensorFlow for Poets durchgearbeitet und danach das Modell mit den neu klassifierten Paketbildern trainiert und getestet.	1.5h

D.5 Woche 5 (17. Oktober - 23. Oktober): Versuchsdurchführung I

D.5.1 Montag

Titel	Beschreibung	Aufwand
Meeting	Besprechen des weiteren Vorgehens mit CS und des bereits erreichten. Beschluss, dass mit weiteren grösseren Versuchen gewartet wird, bis wir neue Pakete erhalten um eine grössere Menge an Lerndaten in besserer Qualität zu erstellen	30m

D.5.2 Dienstag

Titel	Beschreibung	Aufwand
Weekly Meeting	Vorbereitung und Besprechung des weiteren Vorgehens mit HH.	1:30h

D.6 Woche 6 (24. Oktober - 30. Oktober): Versuchsdurchführung I

D.6.1 Dienstag

Titel	Beschreibung	Aufwand
Pakete besorgen	Pakete zum erstellen der neuen Lerndaten bei Herrn Jucker abgeholt	30m

D.7 Woche 7 (31. Oktober - 6. November): Versuchsdurchführung I

D.7.1 Dienstag

Titel	Beschreibung	Aufwand
Fotoshooting	Erstellen der neuen Lerndaten	3h
Meeting	Besprechen des weiteren Vorgehens mit CS	30min

D.7.2 Mittwoch

Titel	Beschreibung	Aufwand
Parallelprogramming	Parallelprogramming am Script um den TF-Prozess zu automatisieren	1.5h

D.7.3 Donnerstag

Titel	Beschreibung	Aufwand
Meeting	Zuerst haben CS und ich das weitere Vorgehen geplant. Danach hatten wir mit Herrn Huser noch ein kurzes Meeting.	1.5h

D.7.4 Freitag

Titel	Beschreibung	Aufwand
Dokumentation ML und CV Frameworks	Ich habe das Kapitel ML und CV Frameworks umgeschrieben um die neue Ausgangslage seit den Versuchen mit Tensorflow abzudecken.	1.5h

D.8 Woche 8 (7. November - 13. November): Versuchsdurchführung II

D.8.1 Dienstag

Titel	Beschreibung	Aufwand
Meeting	Weekly mit HH	30m
Meeting Protokoll	Meeting Protokoll fürs Weekly vom 3.11.16	30min
Meeting Protokoll	Meeting Protokoll fürs Weekly vom 8.11.16	30min

D.8.2 Mittwoch

Titel	Beschreibung	Aufwand
Codestudium	Codestudium von pqc	1.5h
Installation pqc	Installation von pqc in der VM	1h
TensorBoard Dokumentation	Dokumentation von Nutzen und Verwendung von TensorBoard	1h

D.8.3 Donnerstag

Titel	Beschreibung	Aufwand
Test-Modul implementiert	Test-Modul implementiert inkl. einfacher Performance-Metrik	2.5h

D.8.4 Freitag

Titel	Beschreibung	Aufwand
Test-Modul implementiert	Test-Modul refactoring	1h

D.9 Woche 9 (14. November - 20. November): Versuchsdurchführung II

D.9.1 Montag

Titel	Beschreibung	Aufwand
Meeting	Taskplanung für Woche 9	30m
Review pqc setup	Review des pqc setups und der einzelnen Module	1.5h
Metrik-Modul	Extrahieren aus dem Test-Modul und Implementation/Refactoring des Metrik-Moduls	2h

D.9.2 Dienstag

Titel	Beschreibung	Aufwand
Meeting	Weekly mit HH inkl. Protokoll gegenlesen	1h
Bug Fix in metrics	Bug Fix und Code dokumentation in metrics und test	1h

D.9.3 Mittwoch

Titel	Beschreibung	Aufwand
Logger implementieren	Logger implementiert und alle Ausgaben auf diesen umgeleitet	4h

D.9.4 Donnerstag

Titel	Beschreibung	Aufwand
AUC implementieren	Area under the curve implementieren	2.5h

D.9.5 Freitag

Titel	Beschreibung	Aufwand
AUC review	Area under the curve review der eigenen Implementation	0.5h

D.10 Woche 10 (21. November - 27. November): Auswertung

D.10.1 Montag

Titel	Beschreibung	Aufwand
Meeting	Taskplanung für Woche 10	30m

D.10.2 Dienstag

Titel	Beschreibung	Aufwand
Meeting	Weekly Woche 10 mit HH	45m
Weekly Protokoll	Weekly Protokoll geschrieben	45m

D.10.3 Mittwoch

Titel	Beschreibung	Aufwand

Logger implementieren	Print-Funktion überladen um Ausgaben im Retrain-Modul im Quiet-Mode zu unterbinden	2h
Anzahl Trainingsschritte als Parameter	Parameter hinzugefügt um die Anzahl Trainingsschritte zum trainieren des Modells zu steuern	1h
Benutzerhandbuch	Begonnen das Benutzerhandbuch zu schreiben	30m

D.10.4 Donnerstag

Titel	Beschreibung	Aufwand
Benutzerhandbuch	Benutzerhandbuch und Hilfetext von pqc abgeglichen	1h

D.10.5 Freitag

Titel	Beschreibung	Aufwand
Benutzerhandbuch	Benutzerhandbuch erweitert	1h

D.11 Woche 11 (28. November - 04. Dezember): Auswertung

D.11.1 Montag

Titel	Beschreibung	Aufwand
Benutzerhandbuch	Benutzerhandbuch geschrieben	2h

D.11.2 Dienstag

Titel	Beschreibung	Aufwand
Meeting	Taskplanung für Rest von M5	1h
Formulare	Formulare und fehlende Kapitel in Bericht eingefügt	1h

D.11.3 Mitwwoch

Titel	Beschreibung	Aufwand
Ergebiss Auswertung	Generierung von Testmetriken für die Ergebnisdiskussion	5h

D.11.4 Mitwwoch

Titel	Beschreibung	Aufwand
Ergebiss Auswertung	Generierung von Testmetriken für die Ergebnisdiskussion	2h
Review	Review von Testkonzept und Installationsanleitung	2h

D.12 Woche 12 (05. Dezember - 11. Dezember): Auswertung

D.12.1 Montag

Titel	Beschreibung	Aufwand
Ergebnisdiskussion	Ergebnisdiskussion geschrieben	2h
Meeting	Diskussion Auswertung mit CS	1h

D.12.2 Dienstag

Titel	Beschreibung	Aufwand
Ergebnisdiskussion	Diagramme eingefügt	1.5h
Meeting	Weekly Woche 12 mit HH	45m

D.12.3 Mittwoch

Titel	Beschreibung	Aufwand
Ergebnisdiskussion	Kaptiel Trainingschritte	4h

D.12.4 Donnerstag

Titel	Beschreibung	Aufwand
Ergebnisdiskussion	Kaptiel Ratio und Threshold und weitere Auswertungen	6h

D.12.5 Freitag

Titel	Beschreibung	Aufwand
Ergebnisdiskussion	Kaptiel Zielerreichung und Bewertung schreiben	4h

D.12.6 Samstag

Titel	Beschreibung	Aufwand
Umsetzung	Review des Kapitels	30m

D.13 Woche 13 (12. Dezember - 18. Dezember): Abgabe I

D.13.1 Montag

Titel	Beschreibung	Aufwand
Review Umsetzung	Erneutes Review vom Kapitel Umsetzung	3h
Zeitabrechnung vorbereiten	Zeitabrechnung auswerten und vorbereiten für Dokumentation	3h

D.13.2 Dienstag

Titel	Beschreibung	Aufwand
Meeting	Meeting Weekly mit HH	45m
Zeitabrechnung abschliessen	Zeitabrechnung im Bericht einbinden und dokumentieren	2h

D.13.3 Mittwoch

Titel	Beschreibung	Aufwand
Review	Review Abstract	30m
Dokumentation	Management Summary verfasst	2h
Dokumentation	Persönliche Reflektion verfasst	1.5h

D.13.4 Donnerstag

D.14 Woche 14 (19. Dezember - 23. Dezember): Abgabe II

D.14.1 Montag

Titel	Beschreibung	Aufwand
Review	Review der Glossar-Anpassungen	1h
Dokumentation	Korrektur von Rechtschreibfehlern	1h

D.14.2 Dienstag

Titel	Beschreibung	Aufwand
Review	Anpassungen in der Dokumentation betreffend dem Review der Glossaranpassungen	1h
Meeting	Meeting Weekly mit HH	30m
Administration	Formulare gedruckt, unterschreiben lassen und wieder einscannen	30m

Administration	Vorbereitung der Abgabe über Moodle	30m
Administration	Einreichung des Abstract über DAB-Tool	1h

D.14.3 Mittwoch

Titel	**Beschreibung**	**Aufwand**
Weekly Protokoll	Schreiben der Protokolle für Weekly 13 und 14	30m
Administration	Dokumentation drucken und CD brennen für die Abgabe	1h
Administration	Abgabe der Arbeit auf Moodle	30m

Anhang E

Projekttagebuch Cyrill Schenkel

E.1 Woche 1 (19. September – 25. September): Projektplan

E.1.1 Montag

Titel	Beschreibung	Aufwand
Kickoff Meeting	—	1:00h
Einrichtung Arbeitsplatz	Ich habe das Betriebssystem aufgesetzt, dass ich verwenden werde, sowie alle Software, die ich für die Arbeit brauche installiert und konfiguriert.	2:00h
Kickoff Protokoll	Ich habe das Protokoll fürs Kickoff Meeting verfasst.	0:30h
Projektplan	Ich habe den Projektplan erstellt, die Outline davon geschrieben und die ersten Projektphasen im Zeitplan beschrieben.	1:00h

E.1.2 Dienstag

Titel	Beschreibung	Aufwand
Zeitliche Planung schreiben	Ich habe die Zeitplanung im Projektplan fertiggestellt.	1:35h
Meeting	FB und ich haben unser Task Management diskutiert, unser Taskboard gestaltet und die ersten Tasks diskutiert und erstellt.	2:00h
Qualitätsmassnahmen schreiben	Ich habe die Qualitätsmassnahmen im Projektplan beschrieben. Inklusive den vorhin besprochenen zur Koordination der Teamarbeit.	2:35h

Projektorganisation beschreiben	Ich habe ein Organigramm erstellt und die Rollen der verschiedenen Projektmitglieder genannt.	1:00h
Review vom Kapitel Infrastruktur	Ich habe das Kapitel Infrastruktur von FB gereviewt und Kritikpunkte angebracht, sowie einige Fragen geklärt.	0:25h
Cleanup Infrastruktur	Ich habe das Kapitel Infrastruktur von FB erneut gereviewt und einige Layoutprobleme behoben.	0:30h

E.1.3 Mittwoch

Titel	Beschreibung	Aufwand
Projektorganisation erweitert	Ich habe die E-Mail Adressen dem Kapitel Projektorganisation hizugefügt.	0:10h
Review vom Kapitel Infrastruktur	Ich habe das Kapitel Infrastruktur ein letztes Mal gereviewt.	0:05h
Arbeitspakete definieren	Ich habe alle Arbeitspakete, die zu diesem Zeitpunkt feststehen aufgelistet. Dann haben FB und ich diskutiert, ob ich etwas vergessen habe. Danach habe ich damit begonnen, jedes Arbeitspaket zu beschreiben.	1:20h

E.1.4 Donnerstag

Titel	Beschreibung	Aufwand
Arbeitspakete beschrieben	Ich habe Beschreibungen zu allen Arbeitspaketen geschrieben, so wie ich es zuvor mit FB abgemacht habe.	2:00h
Review vom Kapitel Projektübersicht	Ich habe das Kapitel von FB gereviewt.	0:15h

E.1.5 Freitag

Titel	Beschreibung	Aufwand
Review vom Kapitel Einleitung	Ich habe das Kapitel Einleitung von FB gereviewt und Änderungen vorgenommen.	0:10h
Dokument finalisieren	Ich habe damit begonnen das Dokument zu finalisieren. Ich habe dabei einige Layoutfehler behoben, Grafiken verschoben und einzelne kleine Textfragmente verändert bzw. geschrieben.	0:35h

E.2 Woche 2 (26. September – 2. Oktober): Versuchsplanung I

E.2.1 Montag

Titel	Beschreibung	Aufwand
Dokument finalisieren	Ich habe die Titelseite ersetzt und mehrere Layoutprobleme behoben.	0:40h
Tasks schreiben	Ich habe die Tasks für M2 geschrieben.	0:30h
Fragen für Meeting formulieren	Ich habe unsere Fragen für das Meeting am Dienstag aufgeschrieben, damit sich Herr Huser besser darauf vorbereiten kann.	0:45h

E.2.2 Dienstag

Titel	Beschreibung	Aufwand
Ausgangslage beschreiben	Ich habe die Ausgangslage des Projekts für den Projektbericht beschrieben.	1:45h
Weekly Meeting	Ich habe am Meeting und der Nachbesprechung mit FB teilgenommen.	1:00h
Funktionale Anforderungen beschreiben	Ich habe erste Anforderungen definiert, die Qualitätskriterien für dieselben ausformuliert und den Gültigkeitsbereich derselben festgelegt.	3:30h

E.2.3 Mittwoch

Titel	Beschreibung	Aufwand
Motivation Reviewen	Ich habe gereviewt, was FB zur Motivation geschrieben hat.	0:15h
Funktionale Anforderungen beschreiben	Heute hab ich FRS–01 beschrieben.	0:15h

E.2.4 Donnerstag

Titel	Beschreibung	Aufwand
Funktionale Anforderungen beschreiben	Ich habe die Projektziele beschrieben und mehr Anforderungen hinzugefügt, sowie die bestehenden Anforderungen um Abhängigkeiten und Prioritäten ergänzt.	1:15h

Build für Dokumente konfigurieren	Da wir das Dokument *Security Guide Virtuelle Server* beide nicht gefunden haben, hab ich den IT-Services ein Mail geschickt und nach dem entsprechenden Pfad gefragt.	0:35h

E.2.5 Freitag

Titel	Beschreibung	Aufwand
NFR beschreiben	Ich habe zuerst ein E-Mail an die Herren Huser und Jucker geschickt mit Fragen zu den Genauigkeitsbedingungen. Danach hab ich nachgelesen, wie die Genauigkeit des Klassifizierers gemessen werden könnte.	2:16h

E.3 Woche 3 (3. Oktober – 9. Oktober): Versuchsplanung II

E.3.1 Montag

Titel	Beschreibung	Aufwand
Besuch der Produktionsanlage	FB und ich haben die Produktionsanlage besucht. Bei dieser Gelegenheit haben wir gerade über 170 Bilder von Defekten und Intakten Packungen gemacht.	3:00h
NFR beschreiben	Da ich die Antwort auf mein E-Mail erhalten habe, konnte ich nun die Genauigkeitsanforderung ausformulieren.	1:15h
Build für Dokumentation konfigurieren	Um das kompilieren der Dokument zu automatisieren, habe ich ein Build-Script geschrieben. In der Zukunft sind keine Änderungen an diesem Script vonnöten.	2:00h

E.3.2 Dienstag

Titel	Beschreibung	Aufwand
Build für Dokumente konfigurieren	Ich habe den Build für die Dokumente im Jenkins konfiguriert. Dabei sind Probleme aufgetreten, die ich auf die Benutzung von Docker zurückführte. Diese Probleme habe ich gelöst, indem ich eine neue Build Node definiert habe.	1:05h

Liste von Libraries erstellen	Ich habe mit der Liste begonnen. Dabei habe ich Libraries und Frameworks aussortiert, die unseren Anforderungen nicht genügten.	3:15h
Recherche	Ich habe damit begonnen nach Lösungen für unsere Problemstellung zu suchen. Dabei bin ich zum Schluss gelangt, dass wir versuchen müssen mit Feature Extraction Techniken möglichst kleine Feature-Vektoren aus den Bildern zu extrahieren, damit wir trotz der sehr kleinen Menge Daten ein Resultat erzielen können. Wir haben Bücher von der Bibliothek gesucht und mehrer E-Books heruntergeladen und die Recherchearbeit aufgeteilt.	3:00h

E.4 Woche 4 (10. Oktober – 16. Oktober): Versuchsplanung III

E.4.1 Montag

Titel	Beschreibung	Aufwand
Weekly Meeting	Besprechung des weiteren Vorgehens.	0:30h
Recherche	Kapitel im Buch *Machine Learning Paradigms* gelesen.	1:30h
Ansatz I	Ich hab verschiedene Strategien zur Extraktion von Features aus den Bildern mit der Hilfe von VLFeat getestet.	3:00h

E.4.2 Dienstag

Titel	Beschreibung	Aufwand
Ansatz II	Ich hab verschiedene Techniken zur Linienerkennung mit OpenCV ausprobiert. Die Tests waren nicht wirklich erfolgreich. Aber ich konnte ein kleines Tool erstellen, mit dem man verschiedene Parameter einfach verändern konnte.	7:15h
Weiteres Vorgehen vorgeschlagen	Ich habe ein kurzes Dokument mit meinen Vorschlägen zum weiteren Vorgehen erstellt.	1:00h

E.4.3 Mittwoch

Titel	Beschreibung	Aufwand
Bisheriges Vorgehen Dokumentieren	Ich habe damit begonnen das bisherige Vorgehen zu dokumentieren.	0:30h
Ansatz III	Beim dokumentieren ist mir aufgefallen, dass wir bis anhin einfach angenommen haben, dass Deep Learning mit unserer kleinen Datenmenge zu nichts führt. Deshalb, hab ich mich dazu entschieden einen kurzen Test mit TF (Tensor Flow) zu machen, um sicherzustellen, dass das nicht funktioniert. Überraschenderweise hat es viel besser funktioniert als wir angenommen haben. Mit einem retraining von Imagine konnten wir verhältnismässig gute Resultate erzielen.	1:30h

E.4.4 Donnerstag

Titel	Beschreibung	Aufwand
Weiter Tests mit TF	Am Tag zuvor haben wir festgestellt, dass wir mit TF relativ gute Resultate erreichen können. Heute habe ich noch weitere Tests gemacht um sicher zu gehen, dass es sich dabei nicht um einen Zufall handelte. Das Verhalten schien weitestgehend unseren Vorstellungen zu entsprechen. 9 von 10 Tests waren erfolgreich. Es gilt nun das Resultat zu verfeinern und die Performance zu verbessern.	1:30h

E.5 Woche 5 (17. Oktober – 23. Oktober): Versuchsdurchführung I

E.5.1 Dienstag

Titel	Beschreibung	Aufwand
Weekly Meeting	Vorbereitung und Besprechung des weiteren Vorgehens.	1:30h
Meetingprotokoll	Protokoll des heutigen Meetings verfasst.	0:30h

E.6 Woche 6 (24. Oktober – 30. Oktober)

In dieser Woche bin ich nicht dazu gekommen das Tagebuch nachzuführen.

E.7 Woche 7 (31. Oktober – 6. November)

E.7.1 Montag

Nicht aufgeschrieben.

E.7.2 Dienstag

Titel	Beschreibung	Aufwand
Trainingsdaten erfassen	Heute haben wir über 470 Fotos von Packungen gemacht. Die brauchen wir um das Modell zu trainieren	3:00h
Prototyp erstellen	Mein Ziel war es ein kurzes Script zu schreiben, das die Trainingsdaten runterlädt, in Trainings- und Testdaten unterteilt, ein Modell trainiert und das Modell mit den Testdaten überprüft. Alles ausser das letzte Ziel habe ich erreicht. Ein Problem bei der Umsetzung war, dass ich Code, der für eine neuere TF Version gedacht war, als ich installiert hatte verwendet habe.	5:30h

E.7.3 Mittwoch

Titel	Beschreibung	Aufwand
Code Refactorings	Ich hab heute noch einige kleine Änderungen am Code von gestern vorgenommen.	1:00h

E.7.4 Donnerstag

Titel	Beschreibung	Aufwand
Meeting	Zuerst haben FB und ich das weitere Vorgehen geplant. Danach hatten wir noch ein kurzes Meeting mit Herrn Huser.	1:45h

E.7.5 Freitag

Titel	Beschreibung	Aufwand

Architektur	Heute hab ich damit begonnen, die Architektur zu spezifizieren.	3:15h

E.8 Woche 8 (7. November – 13. November)

E.8.1 Montag

Titel	Beschreibung	Aufwand
Architektur	Heute hab ich die Spezifikation der Architektur abgeschlossen.	2:35h
Review	Ich habe FBs Überarbeitung des Abschnitts Entscheidungsfindung gereviewt.	0:30h
API Dokumentation	Ich habe damit begonnen, die API des Prototypen zu dokumentieren.	0:45h

E.8.2 Dienstag

Titel	Beschreibung	Aufwand
Meeting	Weekly Meeting mit HH und FB.	0:30h
API Dokumentation	Ich habe die API Dokumentation alles bisherigen Codes abgeschlossen.	1:30h

E.8.3 Donnerstag

Titel	Beschreibung	Aufwand
Caching	Ich habe damit begonnen das “Caching” zu implementieren.	1:30h

E.8.4 Freitag

Titel	Beschreibung	Aufwand
Caching	Ich habe die Implementation des “Caching” abgeschlossen.	2:30h

E.9 Woche 9 (14. November – 20. November)

E.9.1 Montag

Titel	Beschreibung	Aufwand
Reshuffle	Ich habe die “reshuffle” Funktionalität implementiert.	2:35h

Clean	Ich habe die “clean” Funktionalität implementiert.	0:35h
Threshold	Ich habe die “threshold” Option implementiert.	1:15h

E.9.2 Dienstag

Titel	Beschreibung	Aufwand
API Dokumentation	Heute habe ich die API Dokumentation vervollständigt.	3:15h
Weekly	Weekly mit HH und FB.	0:45h
Protokoll	Ich habe das Weekly Protokoll geschrieben.	0:10h

E.9.3 Mittwoch

Titel	Beschreibung	Aufwand
Verschiedene Reviews	Heute habe ich verschiedene Änderungen von FB gereviewt und mit ihm korrigiert.	3:00h
CLI Hilfetext	Heute habe ich den Hilfetext für das CLI von PQC geschrieben.	1:15h

E.10 Woche 10 (21. November – 27. November)

E.10.1 Dienstag

Titel	Beschreibung	Aufwand
Bugfix	Heute habe ich den Crash gefixt. Die Lösung lag darin TF vor dem Testen zu resetten.	1:30h
Kennzahlen Review	Heute habe ich die Berechnung der Kennzahlen gereviewt.	0:45h
Weekly	Heute habe ich am Meeting mit HH und FB teilgenommen.	0:45h
API Dokumentation	Heute habe ich die API Dokumentation dem Bericht angehängt.	2:00h

E.10.2 Donnerstag

Titel	Beschreibung	Aufwand
API Dokumentation	Heute habe ich Probleme mit dem Anhängen der API Dokumentation an den Bericht behoben.	1:00h

E.11 Woche 11 (28. November – 4. Dezember)

E.11.1 Dienstag

Titel	Beschreibung	Aufwand
Testprotokoll	Heute habe ich damit begonnen das Testkonzept zu schreiben.	4:30h

E.11.2 Mittwoch

Titel	Beschreibung	Aufwand
Testprotokoll	Heute habe ich am Testkonzept weitergeschrieben und mit dem Testprotokoll begonnen.	2:05h

E.11.3 Donnerstag

Titel	Beschreibung	Aufwand
Testprotokoll	Heute habe ich das Testprotokoll und das Testkonzept abgeschlossen.	0:35h
Review Logger	Heute habe ich die Integration eines Loggers ins Projekt gereviewt.	1:00h
Review Benutzerhandbuch	Heute habe ich das Benutzerhandbuch gereviewt.	1:10h

E.12 Woche 12 (5. Dezember – 11. Dezember)

E.12.1 Montag

Titel	Beschreibung	Aufwand
Review Benutzerhandbuch	Heute habe ich mein Review des Benutzerhandbuches fortgesetzt.	0:45h
Review Formulare	Heute habe ich die von FB eingefügten Formulare gereviewt.	0:30h
Review Benutzerhandbuch	Heute habe ich das Benutzerhandbuch gereviewt.	1:10h
Dokument umstrukturieren	Heute habe ich damit begonnen die Struktur der Dokumente zu erneuern.	1:00h

E.12.2 Dienstag

Titel	Beschreibung	Aufwand

Dokument umstrukturiert	Heute habe ich den Projektplan und den Bericht zu einem kombinierten Dokument zusammengefasst.	1:50h
Weekly	Heute habe ich am Meeting mit FB und HH teilgenommen.	1:00h
Protokoll	Meeting Protokoll verfasst.	0:10h
Kapitel: Umsetzung	Ich habe mit dem Kapitel zur Umsetzung begonnen.	1:30h

E.12.3 Mittwoch

Titel	Beschreibung	Aufwand
Kapitel: Umsetzung	Ich habe am Kapitel zur Umsetzung weitergeschrieben.	1:30h

E.12.4 Donnerstag

Titel	Beschreibung	Aufwand
Kapitel: Umsetzung	Ich habe am Kapitel zur Umsetzung weitergeschrieben.	7:15h

E.12.5 Freitag

Titel	Beschreibung	Aufwand
Kapitel: Umsetzung	Ich habe das Kapitel zur Umsetzung abgeschlossen.	6:30h
Kapitel: Zusammenfassung und Ausblick	Ich habe das Kapitel: Zusammenfassung und Ausblick geschrieben.	1:40h

E.13 Woche 13 (12. Dezember – 18. Dezember)

E.13.1 Montag

Titel	Beschreibung	Aufwand
Kleine Korrekturen	Heute habe ich einige kleine Rechtschreibfehler korrigiert.	2:00h

E.13.2 Dienstag

Titel	Beschreibung	Aufwand
Abstract	Ich habe das Abstract geschrieben.	2:10h

Reflektion schreiben	Ich habe meine Reflektion geschrieben.	2:00h

E.13.3 Mittwoch

Titel	Beschreibung	Aufwand
Verschiedene Korrekturen	Ich habe verschiedene Fehler im Bericht korrigiert.	2:00h

E.14 Wochen 13 und 14

Aus Zeitgründen und aufgrund eines Datenverlusts konnte dieses Tagebuch ab hier nicht weitergeführt werden.

Anhang F

Sitzungsprotokolle

Protokoll: Kickoff

Florian Bitterlin
Cyrill Schenkel

19. September 2016

1 Fragen zur Aufgabenstellung

1.1 Sind die Lerndaten/Fotos bereits verfügbar?

Nein, aber sie sollten so schnell wie möglich verfügbar gemacht werden. Herr Huser kümmert sich darum.

1.2 Kann der Hintergrund der Fotos als neutral angenommen werden?

Ja. Sonst könnte man die Fotos auch zuschneiden.

1.3 Ist 4 Wochen eine gute Dauer für die Evaluierungsphase?

Grundsätzlich ja. Es sollte festgelegt werden, was wir in der Hälfte der Arbeit erreicht sein sollte.

2 Beschlüsse

Meetingstermine
: Sind noch nicht klar. Muss zu einem späteren Zeitpunkt endgültig geregelt werden.

Server
: Herr Huser hat einen Buildserver beantragt.

Abgabetermin des Projektplans
: 26. September 2016

Protokoll: Woche 2

Florian Bitterlin
Cyrill Schenkel

27. September 2016

1 Offene Fragen

1.1 Wann können wir die Lerndaten bekommen?

Während Besuch in der Fabrik Beispiele (Negativ & Positiv) besprechen vor Ort.

1.2 In welcher Form können wir die Lerndaten bekommen?

Sehr wahrscheinlich selbst fotografieren von Beispielen oder während Besuch in der Fabrik.

1.3 Wie hoch ist die Fehlerrate beim aktuellen System?

Es ist vermutlich keine Aussage möglich, bzw. wird nicht gemessen. Kontrolle erfolgt zur Zeit beim Verpacken der Packungen in Kisten oder auf Palletts. Je schneller die Produktion umso schlechter ist also die Kontrolle.

2 Beschlüsse

Einschränkungen
: Konzentration nur auf die Sicht von oben. Falls genügend Zeit können auch weitere Fehler betrachtet werden.

Server
: Der Build-Server wurde am 27. September zur Verfügung gestellt.

Meetingstermine
: Grundsätzlich zu beginn jeweils am Dienstag um 13.30h.

Besuch der Fabrik
: Montag, den 3. Oktober, 9.15h in Mettmenstetten

Protokoll: Woche 5

Florian Bitterlin
Cyrill Schenkel

18. Oktober 2016

1 Drei Ansätze

FB und CS haben die drei Ansätze (VLFeat, OpenCV und TensorFlow), die sie getestet haben erläutert und die Resultate HH gezeigt. Es wurden alle drei Ansätze diskutiert. Mit Canny und Hough konnten keine brauchbaren Resultate erzielt werden, was zumindest teilweise an den schlechten und unterschiedlichen Qualitäten der Bilder liegt. Mit TensorFlow (ab hier TF) hingegen konnten sehr einfach relativ gute Resultate erzielt werden. Mit TF müssen allerdings noch mehr Tests gemacht werden um die bisherigen Erfahrungen zu untermauern.

2 Weiters Vorgehen

Am Donnerstag werden die neuen Packungen verfügbar sein. Dann wird es darum gehen neue Bilder zu machen und herauszufinden, unter welchen Bedingungen die besten Resultate erzielt werden können. Je nach dem sollten die Verfahren, die bisher nicht sehr erfolgreich waren, mit den neuen Bildern nochmals ausprobiert werden.

Die TF Lösung muss weiter verfeinert werden. Die Genauigkeit muss erhöht und die Klassifizierung verschnellert werden.

Protokoll: Woche 7

Florian Bitterlin
Cyrill Schenkel

3. November 2016

1 Bisherige Arbeiten

FB und CS haben die neu erstellten Lerndaten und deren Erstellungsprozess vorgestellt. Dank dem neuen, von der Intigena zur Verfügung gestellten, Material konnten mehrere Hundert Bilder besserer Qualität erstellt werden.

Mit HH wurde nochmals besprochen, warum wir uns für TensorFlow entschieden haben. Ausserdem wurde besprochen, wie sich der bestehende Prozess des Trainierens und Testens automatisieren lässt.

2 Weiters Vorgehen

Der Entscheid zu TensorFlow und zur Ablehnung von OpenCV und VLfeat wird ausführlich im Bericht festgehalten. Der Prozess zum Testen, Trainieren und Aufteilen der Lerndaten in Test- und Trainingsdaten wird automatisiert und ebenfalls im Bericht dokumentiert.

Protokoll: Woche 8

Florian Bitterlin
Cyrill Schenkel

8. November 2016

1 Bisherige Arbeiten

Die Dokumentation wurde auf den neusten Stand gebracht. Die Entscheidungsfindung für TensorFlow wurde dokumentiert, ebenso die Architektur des Skripts für die Automatiersung des TensorFlow Prozesses.

Mit HH wurde nochmals besprochen, welche genauen Kennzahlen für unsere Tests wichtig sind und welche umgesetzt werden sollten. Unteranderem sollten FPR und NPR sowie auch die AUC ermittelt werden.

2 Weiters Vorgehen

Unser Code soll soweit erweitert werden, dass Performancetests für das Klassifizieren von Paketen möglich sind. Desweiteren sollen Unittests erstellt werden. TensorBoard und seinen Einsatz in unserem Projekt soll dokumentiert werden.

Protokoll: Woche 9

Florian Bitterlin
Cyrill Schenkel

15. November 2016

1 Bisherige Arbeiten

Alle Metrics bis auf AUC wurden implementiert. Das Tool ist jetzt benützbar und mit den ersten Tests werden die Performance und Genauigkeitsziele eingehalten.

2 Weiters Vorgehen

- Die Entwicklungsphase wird während dem Rest der Woche oder bis zum nächsten Meeting abgeschlossen. Dazu gehört auch die API Dokumentation und das User Manual.
- FB schickt HH die URL des Git Repositories.

Protokoll: Woche 10

Florian Bitterlin
Cyrill Schenkel

22. November 2016

1 Bisherige Arbeiten

Alle Metrics inkl. AUC wurden implementiert. Das Tool ist benutzbar, Performance und Genauigkeitsziele werden eingehalten. Die API-Dokumentation wurde vervollständigt und bestehende Bugs gefixt.

2 Weiters Vorgehen

- Mit der Auswertung der Resultate aus der Entwicklungsphase wird begonnen. Die verschiedensten Dokumente wie Installationsanleitung, Benutzerhandbuch, etc. werden geschrieben und der Bericht vervollständigt.
- Das Weekly vom 29.11. wird aufgrund von Abwesenheit von HH nicht durchgeführt.

Protokoll: Woche 12

Florian Bitterlin
Cyrill Schenkel

6. Dezember 2016

- Herr Huser gibt Feedback zum Stand der Dokumentation von diesem Freitag.
- Das Kapitel zur Umsetzung richtet sich an jene, die an Folgeprojekten arbeiten.
- Die Formulare werden Ausgedruckt und Unterschrieben, dann eingescannt und wieder ins Dokument eingefügt.
- Als Beilagen gelten: der Sourcecode und der Bericht.

Protokoll: Woche 13

Florian Bitterlin
Cyrill Schenkel

13. Dezember 2016

- FB und CS geben Feedback zum Stand der Dokumentation.
- Die Arbeit wird kurz mündlich zusammen reflektiert.
- Herr Huser zeigt sich zufrieden mit dem Stand der Dokumentation und dem Resultat der Arbeit.
- Das Abstract wird vor der Abgabe über das DAB-Tool an Herr Huser gesendet. Herr Huser gibt vor der Abgabefrist am 20.12.16 ein Feedback.

Protokoll: Woche 14

Florian Bitterlin
Cyrill Schenkel

20. Dezember 2016

- Die Dokumentation konnte fertiggestellt werden. Es folgt noch die finalisierung des Dokuments.
- Die benötigten Forumlare wurden von den Beteiligten Unterschrieben. Sie werden für die Abgabe noch eingescannt.
- Die Änderungswünsche von Herr Huser wurden im Abstract eingepflegt.
- Die Titelseite-Vorgabe der Schule ist für Herr Huser nicht zwinged. Die Titelseite wird daher gestalterisch angepasst.

Anhang G

Benutzerhandbuch

pqc- der Packung-Qualitäts-Checker

G.1 Beschreibung

PQC hilft beim trainieren eines binären Klassifizierers um Packungen als Defekt oder Intakt zu bestimmen.

G.2 Kurzfassung

```
pqc [-h | --help] <command> [<args>]
```

Listing G.1: pqc Kurzfassung

G.3 Kommandos

init Initialisiert das aktuelle Verzeichnis als Arbeitsverzeichnis für pqc und ladet die Trainingsdaten. Falls in dem Verzeichnis bereits eine Datei mit dem Namen data.zip vorhanden ist, wird dieses verwendet.

divide Teilt die klassifizierten Daten auf in Trainings- und Testdaten. Der Anteil der klassifizierten Daten, welche als Testdaten verwendet werden sollen, wird als Argument als Fliesskommazahl erwartet. Bsp.: `pqc divide 0.3`

reshuffle Verteilt die vorhandenen klassifizierten Daten in Trainings- und Testdaten neu, um ein umtrainieren mit einer anderen Aufteilung zu ermöglichen.

train Ladet das Inception-Modell von Google herunter und trainiert es mithilfe der vorhandenen Testdaten auf den gegebenen Anwendungsfall um. Sollte pqc im aktuellen Arbeitsverzeichnis noch nicht in Trainings- und Testdaten aufgeteilt sein, werden vor dem Trainieren noch die klassifizierten

Daten aufgeteilt. Die Anzahl der Trainingsschritte kann als Argument mitgegeben werden, wobei weniger als 500 oder mehr 4000 Schritte nicht als Sinnvoll erachtet werden. Bsp.: `pqc train -s 1000`

test Testet das umtrainierte Modell mithilfe der vorhandenen Testdaten und gibt die erreichten Metriken der Tests aus. Sollte pqc im aktuellen Arbeitsverzeichnis noch nicht trainiert sein, wird vor dem Testen noch das Modell trainiert. Der Threshold bestimmt, zu wieviel Prozent ein Packung als Intakt klassifiziert werden muss, damit es als wirklich Intakt gezählt werden kann. Der Threshold wird als Fliesskommazahl angegeben. Bsp.: `pqc test -p 0.8`

clean Räumt das Arbeitverzeichnis von pqc auf und löscht alle vorhandenen Daten bis auf das Archiv der klassifizierten Daten. *data.zip*.

G.3.1 Syntax

```
$ pqc init [-t|--tmpdir] [-v|--verbose] [-q|--quiet]
$ pqc divide <ratio> [-v|--verbose] [-q|--quiet]
$ pqc reshuffle [-v|--verbose] [-q|--quiet]
$ pqc train [-s|--steps <steps>] [-v|--verbose] [-q|--quiet]
$ pqc test [-p|--threshold <threshold>] [-v|--verbose] [-q|--quiet]
$ pqc clean [-v|--verbose] [-q|--quiet]
```

Listing G.2: pqc command syntax

G.3.2 Parameter

ratio Anteil der Aufnahmen welche beim Aufteilen der klassifizierten Daten in Trainings- und Testdaten als Testdaten eingestuft werden sollen. Erwartet wird eine Fliesskommazahl im Wertebereich von 0.0 bis 1.0.

steps Anzahl der Trainingsschritte die beim umtrainieren des Modells gemacht werden sollen. Erwartet wird eine Ganzzahl.

threshold Wert welchen die prognostizierte Wahrscheinlichkeit, dass eine Packung Intakt ist, überschreiten muss, damit die Packung tatsächlich als intakt eingstuft wird. Erwartet wird eine Fliesskommazahl im Wertebereich von 0.0 bis 1.0.

-h, --help Zeigt den Hilfetext von pqc in der Kommandozeile an.

-q, --quiet Reduziert die Ausgabe von pqc auf eine absolutes Minimum.

-t, --tmpdir Initialisiert pqc in einem temporärem Verzeichnis anstatt im aktuellem Verzeichnis.

-v, --verbose Erweitert die Ausgabe von pqc auf detailiertere Angaben und Auswertungen.

G.4 Installationsanleitung

1. Source Code Repository klonen.

   ```
   git clone [redacted]/Studienarbeit/tf_classifier.git pqc
   ```

2. In das Root-Verzeichnis des Sourcetrees wechseln.

   ```
   cd pqc
   ```

3. Die Abhängigkeiten des Programms installieren.

   (a) Submodule installieren

   ```
   git submodule init
   ```

   ```
   git submodule update
   ```

   (b) Pip3 installieren.
   Unter Debian/Ubuntu mit:

   ```
   apt install python3-pip
   ```

   (c) TensorFlow installieren.[9]
   Die installierte TensorFlow Version muss genau 0.11.0rc2 sein.

   (d) Sonstige Abhängigkeiten installieren.

   ```
   pip3 install --user -r requirements.txt
   ```

4. Das Programm selbst installieren.

   ```
   pip3 install --user --upgrade .
   ```

5. Installation testen.

   Folgendes Kommando:

   ```
   pqc
   ```

   sollte die installierte Version von *pqc* ausgeben.

G.5 TensorBoard

TensorBoard ist eine Sammlung von Werkzeugen zur Visualisierung des Lerprozesses von TensorFlow Modellen. Die Visualisierungen können das Debuggen und Optimieren von Modellen vereinfachen.

TensorBoard kann benutzt werden, um den TensorFlow-Graphen und quantitative Ausführungsmetriken zu visualisieren. Sowie auch um zusätzliche Daten, wie die verwendeten Trainingsdaten, anzuzeigen.

Um TensorBoard zu starten, geben Sie den folgenden Befehl ein (alternativ: `python -m tensorflow.tensorboard`):

```
$ tensorboard --logdir=<path>
```

Listing G.3: TensorBoard Aufruf

Die `--logdir` Option zeigt dabei das Verzeichnis mit den Eventlogs von TensorFlow an. Unterverzeichnisse im Log-Verzeichnis werden von TensorBoard als einzelne Runs interpretiert und als solche visualisiert.

Sobald TensorBoard gestartet ist, kann im Browser `localhost:6006` aufgerufen werden, um TensorBoard zu verwenden. Es wird empfohlen Chrome zu benutzen, da bei anderen Browsern mit Anzeige-Fehlern gerechnet werden muss.

Das Standard `--logdir` von `pqc` ist `<working-dir>/logdir`. Wobei `<working-dir>` das Arbeitsverzeichnis ist, das mit `pqc init` initialisiert wurde. [1]

G.6 Autoren

PQC wurde von Florian Bitterlin und Cyrill Schenkel unter Betreuung von Hansjörg Huser und mit der Intigena Produktions (Schweiz) AG als Industriepartner im Rahmen einer Studienarbeit an der Hochschule für Technik in Rapperswil (HSR) entwickelt.

[1] "TensorBoard: Visualizing Learning" [11]

Anhang H

Programmiererhandbuch

PQC Documentation

Release 0.0.1

Florian Bitterlin, Cyrill Schenkel

Dec 14, 2016

CHAPTER

ONE

SUBPACKAGES

1.1 pqc.prepare package

1.1.1 Submodules

pqc.prepare.data_archive module

pqc.prepare.data_archive.**unpack_data**(*working_dir*)
Unpack the data archive.

Parameters **working_dir** – Path to the working directory.

See: :param working_dir: Path to the working directory. See:

pqc.defaults.DATA_ARCHIVE

pqc.prepare.divide module

pqc.prepare.divide.**divide_data**(*working_dir*, *test_data_ratio*)
Divide the data into test and training data.

Parameters

- **working_dir** – Path to the working directory.
- **test_data_ratio** – The ratio of test to training data.

pqc.prepare.divide.**reshuffle**(*working_dir*, *test_data_ratio*)
Reshuffle the test and training data.

Divide the data into test and training data, when it was already divided before.

Parameters

- **working_dir** – Path to the working directory.
- **test_data_ratio** – The ratio of test to training data.

pqc.prepare.init module

pqc.prepare.init.**create_working_dir**(*parent=None*)

Parameters **parent** – (Default value = WORKING_DIR_LOCATION)

pqc.prepare.init.**download_data**(*working_dir*, *download_url*)
Download training data from ‘download_url’.

Parameters

- **working_dir** – Path to the working directory.

- **download_url** – URL of the training data.

pqc.prepare.init.**init**(*working_dir=None*, *download_url='https://.../data.zip'*)
Initialize 'working_dir'.

Kwargs: working_dir: Path to the working directory. download_url: URL of the training data.

Parameters

- **working_dir** – (Default value = None)
- **download_url** – (Default value = DOWNLOAD_URL)

Returns Path to working directory

1.1.2 Module contents

1.2 pqc.test package

1.2.1 Submodules

pqc.test.metrics module

pqc.test.metrics.**calc_metrics**(*test_results*, *threshold*)
Calculate and output metrics from test results.

Parameters

- **test_results** – Test results of type TestData.
- **threshold** – Threshold of prediction to count as intact.

class pqc.test.metrics.**metric_calculator**(*test_data*)
Bases: object

accuracy()

area_under_the_curve()

average_time()

false_negative_rate()

false_positive_rate()

matthews_correlation_coefficient()

print_metric()

true_negative_rate()

true_positive_rate()

pqc.test.test module

class pqc.test.test.**test_data**(*path*, *label*)
Bases: object

pqc.test.test.**test_model**(*working_dir*, *threshold=0.8*)
Test images against the trained model.

Testing is based on the tutorial TensorFlow for poets https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/ (visited on 2016-12-02)

Parameters

- **working_dir** – Path to working directory.
- **threshold** – Threshold of prediction_string to count as intact. (Default value = THRESHOLD)

1.2.2 Module contents

1.3 pqc.ui package

1.3.1 Submodules

pqc.ui.entry_point module

pqc.ui.entry_point.**main**()
Entry point of the application.

1.3.2 Module contents

CHAPTER

TWO

SUBMODULES

2.1 pqc.clean module

pqc.clean.**clean**(*working_dir*)
Delete all generated files.

The ‘pqc.defaults.STATE_FILE’ is also deleted from the ‘working_dir’. The ‘pqc.defaults.DATA_ARCHIVE’ isn’t deleted, so that it doesn’t need to be downloaded again.

Parameters **working_dir** – Path to the working directory.

2.2 pqc.defaults module

exception pqc.defaults.**NotInitialized**
Bases: BaseException

pqc.defaults.**check_dependencies**(*working_dir*, **dependencies*)
Check if all ‘dependencies’ exist in the ‘working_dir’.

Parameters

- **working_dir** – Path to the working directory.
- **dependencies** – Files to check.
- ***dependencies** –

Returns when all ‘dependencies’ are contained in ‘working_dir’

Return type True

pqc.defaults.**data_archive**(*working_dir*)
Build the data archive path for the given ‘working_dir’.

Parameters **working_dir** – Path to the working directory.

pqc.defaults.**data_dir**(*working_dir*)
Build the data directory path for the given ‘working_dir’.

Parameters **working_dir** – Path to the working directory.

class pqc.defaults.**label**
Bases: object

Class labels.

all = [’intakt’, ‘defekt’]

defect = ‘defekt’

intact = ‘intakt’

pqc.defaults.**state_file**(*working_dir*)
Build the state file path for the given ‘working_dir’.

Parameters **working_dir** – Path to the working directory.

pqc.defaults.**test_dir**(*working_dir*)
Build the test data directory path for the given ‘working_dir’.

Parameters **working_dir** – Path to the working directory.

2.3 pqc.fs_utils module

pqc.fs_utils.**delete_if_exists**(*path*)
Delete the file or directory at the given ‘path’.

If ‘path’ points to a directory, that directory is deleted recursively. If there is no file or directory at ‘path’ nothing happens.

Parameters **path** – The path of the resource to delete.

pqc.fs_utils.**guess_working_dir**()
Guess the working directory based on ‘cwd’.

Returns The path to ‘cwd’ or the parent directory of ‘cwd’ nearest to ‘cwd’ which is a working directory.

pqc.fs_utils.**in_wd**(*working_dir*, *files*)
Make the paths in ‘files’ absolute.

Parameters

- **working_dir** – Path to the working directory.
- **files** – An iterator over paths relative to ‘working_dir’

Returns An iterator over the absolute paths of the paths in ‘files’

pqc.fs_utils.**list_data**(*working_dir*, *label*)
List data with given ‘label’ in ‘working_dir’.

Parameters

- **working_dir** – Path to the working directory.
- **label** – Class label.

Returns The list of all data with the given ‘label’.

2.4 pqc.state module

pqc.state.**get_ratio**(*working_dir*)
Retrieve the test and training data ratio.

Parameters **working_dir** – Path to the working directory.

Returns The test and training data ratio.

pqc.state.**get_steps**(*working_dir*)
Retreive the amount of training steps.

Parameters **working_dir** – Path to the working directory.

Returns The amount of training steps.

See: pqc.state.set_steps

pqc.state.**get_threshold**(*working_dir*)
Retreive the intact threshold.

Parameters **working_dir** – Path to the working directory.

Returns The intact threshold.

See: pqc.state.set_threshold

pqc.state.**init_state**(*working_dir*)
Initialize the state.

Create a ‘pqc.defaults.STATE_FILE’ and set all fields to their default value.

Parameters **working_dir** – Path to the working directory.

pqc.state.**is_divided**(*working_dir*)
Check if the data in ‘working_dir’ is divided.

Parameters **working_dir** – Path to the working directory.

pqc.state.**is_initialized**(*working_dir*)
Check if ‘working_dir’ is initialized.

Parameters **working_dir** – Path to the working directory.

pqc.state.**is_trained**(*working_dir*)
Check if the model in ‘working_dir’ is trained..

Parameters **working_dir** – Path to the working directory.

pqc.state.**set_ratio**(*working_dir*, *ratio*)
Set the test and training data ratio to the given ‘ratio’.

Parameters

- **working_dir** – Path to the working directory.
- **ratio** – The test and training data ratio.

pqc.state.**set_steps**(*working_dir*, *steps*)
Set the amount of training steps to retrain the model.

Parameters

- **working_dir** – Path to the working directory.
- **steps** – The amount of training steps.

pqc.state.**set_threshold**(*working_dir*, *threshold*)
Set the intact threshold.

Only packages which are assigned a prediction rating greater than the ‘threshold’ are counted as ‘pqc.defaults.label.intact’.

Parameters

- **working_dir** – Path to the working directory.
- **threshold** – The intact threshold.

2.5 pqc.train module

pqc.train.**train_model**(*working_dir*, *steps=1000*)
Train the model.

Parameters

- **working_dir** – Path to the working directory.

- **steps** – (Default value = TRAINING_STEPS)

2.6 Module contents

p

U

Anhang I

Testprotokoll

I.1 Voraussetzungen

Bevor mit der Testdurchführung begonnen wird, müssen folgende Bedingungen erfüllt sein.

- Die neuste pqc Version muss installiert sein.
- Ein leeres Verzeichnis muss erstellt worden sein. (ab hier: <**dir**>)
- Das aktuelle Arbeitsverzeichnis muss <**dir**> sein.

I.2 Tests

I.2.1 T1 • Initialisierung

Ausgangslage

```
$ tree
.

0 directories, 0 files
```

Eingabe

```
pqc init
```

Endzustand (Soll)

```
$ pqc init
>> Downloading data.zip 244.1 MiB
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
```

```
‘-- data.zip

3 directories, 1 file
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.2 T2 • Daten in Trainings- und Testdaten aufteilen

Ausgangslage

Der Soll-Endzustand von T1.

Eingabe

```
$ pqc divide 0.3
```

Endzustand (Soll)

```
 $ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   ‘-- intakt
|-- data.zip
‘-- test
    |-- defekt
    ‘-- intakt

6 directories, 1 file
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.3 T3 • Trainieren

Ausgangslage

Der Soll-Endzustand von T2.

Eingabe

```
pqc train -s 500
```

Endzustand (Soll)

```
$ pqc train -s 500
>> Downloading [...] 100.0%
>> Training model
Looking for images in 'intakt'
Looking for images in 'defekt'
[...]

$ tree -L 2 -n --charset=ASCII
.
|-- bottlenecks
|   |-- defekt
|   `-- intakt
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   [(*...*)]
|-- logdir
|   |-- train
|   `-- validation
|-- retrained_graph.pb
|-- retrained_labels.txt
`-- test
    |-- defekt
    `-- intakt

13 directories, 9 files
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.4 T4 • Testen

Ausgangslage

Der Soll-Endzustand von T3.

Eingabe

```
pqc test -p 0.75
```

Endzustand (Soll)

Die genauen ausgegeben Werte, können abweichen.

```
$ pqc test -p 0.75
>> Testing images (143/143)
[...]
```

```
Average time/image: 0.20287s
Positive (defekt): 102, Negative (intakt): 41
TP 100, TN 35, FP 6, FN 2
Accuracy: 94.41%
False positive rate: 14.63%
False negative rate: 1.96%
Matthews correlation coefficient: 86.13%
Area under the curve: 0.98685
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.5 T5 • Zustand

Ausgangslage

Der Soll-Endzustand von T4.

Eingabe

```
cat .pqc
```

Endzustand (Soll)

```
$ cat .pqc
{"threshold": 0.75, "steps": 500, "ratio": 0.3}
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.6 T6 • Clean

Ausgangslage

Der Soll-Endzustand von T5.

Eingabe

```
pqc clean
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
‘-- data.zip

0 directories, 1 file
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.7 T7 • Initialisierung nach Clean

Ausgangslage

Der Soll-Endzustand von T6.

Eingabe

```
pqc init
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
`-- data.zip

3 directories, 1 file
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.8 T8 • Direkt trainieren

Ausgangslage

<dir> ist initialisiert.

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
`-- data.zip

3 directories, 1 file
```

Eingabe

```
pqc train
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
|-- bottlenecks
|   |-- defekt
|   `-- intakt
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   [...]
|-- logdir
|   |-- train
|   `-- validation
|-- retrained_graph.pb
|-- retrained_labels.txt
`-- test
    |-- defekt
    `-- intakt

13 directories, 9 files
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

I.2.9 T9 • Reshuffle

Ausgangslage

Der Soll-Endzustand von T8.

Eingabe

```
pqc reshuffle
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   |-- classify_image_graph_def.pb
|   |-- cropped_panda.jpg
|   |-- imagenet_2012_challenge_label_map_proto.pbtxt
|   |-- imagenet_synset_to_human_label_map.txt
|   |-- inception-2015-12-05.tgz
```

```
|   `-- LICENSE
|-- logdir
|   |-- train
|   `-- validation
`-- test
    |-- defekt
    `-- intakt

10 directories, 7 files
```

Endzustand (Ist)

[Wird vom Tester/der Testerin eingetragen.]

Anhang J

Testresultate

J.1 Voraussetzungen

Bevor mit der Testdurchführung begonnen wird, müssen folgende Bedingungen erfüllt sein.

- Die neuste pqc Version muss installiert sein.
- Ein leeres Verzeichnis muss erstellt worden sein. (ab hier: <**dir**>)
- Das aktuelle Arbeitsverzeichnis muss <**dir**> sein.

J.2 Tests

J.2.1 T1 • Initialisierung

Ausgangslage

```
$ tree
.

0 directories, 0 files
```

Eingabe

```
pqc init
```

Endzustand (Soll)

```
$ pqc init
>> Downloading data.zip 244.1 MiB
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
```

```
‘-- data.zip

3 directories, 1 file
```

Endzustand (Ist)

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   ‘-- intakt
‘-- data.zip

3 directories, 1 file
```

J.2.2 T2 • Daten in Trainings- und Testdaten aufteilen

Ausgangslage

Der Soll-Endzustand von T1.

Eingabe

```
$ pqc divide 0.3
```

Endzustand (Soll)

```
 $ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   ‘-- intakt
|-- data.zip
‘-- test
    |-- defekt
    ‘-- intakt

6 directories, 1 file
```

Endzustand (Ist)

```
 $ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   ‘-- intakt
|-- data.zip
‘-- test
    |-- defekt
    ‘-- intakt

6 directories, 1 file
```

J.2.3 T3 • Trainieren

Ausgangslage

Der Soll-Endzustand von T2.

Eingabe

```
pqc train -s 500
```

Endzustand (Soll)

```
$ pqc train -s 500
>> Downloading [...] 100.0%
>> Training model
Looking for images in 'intakt'
Looking for images in 'defekt'
[...]

$ tree -L 2 -n --charset=ASCII
.
|-- bottlenecks
|   |-- defekt
|   `-- intakt
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   [(*...*)]
|-- logdir
|   |-- train
|   `-- validation
|-- retrained_graph.pb
|-- retrained_labels.txt
`-- test
    |-- defekt
    `-- intakt

13 directories, 9 files
```

Endzustand (Ist)

```
$ tree -L 2 -n --charset=ASCII
.
|-- bottlenecks
|   |-- defekt
|   `-- intakt
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
```

```
|   [(*...*)]
|-- logdir
|   |-- train
|   `-- validation
|-- retrained_graph.pb
|-- retrained_labels.txt
`-- test
    |-- defekt
    `-- intakt

13 directories, 9 files
```

J.2.4 T4 • Testen

Ausgangslage

Der Soll-Endzustand von T3.

Eingabe

```
pqc test -p 0.75
```

Endzustand (Soll)

Die genauen ausgegeben Werte, können abweichen.

```
$ pqc test -p 0.75
>> Testing images (143/143)
[...]
Average time/image: 0.20287s
Positive (defekt): 102, Negative (intakt): 41
TP 100, TN 35, FP 6, FN 2
Accuracy: 94.41%
False positive rate: 14.63%
False negative rate: 1.96%
Matthews correlation coefficient: 86.13%
Area under the curve: 0.98685
```

Endzustand (Ist)

```
$ pqc test -p 0.75
>> Testing images (143/143)
[...]
Average time/image: 0.31641s
Positive (defekt): 102, Negative (intakt): 41
TP 99, TN 34, FP 7, FN 3
Accuracy: 93.01%
False positive rate: 17.07%
False negative rate: 2.94%
Matthews correlation coefficient: 82.59%
Area under the curve: 0.98898
```

J.2.5 T5 • Zustand

Ausgangslage

Der Soll-Endzustand von T4.

Eingabe

```
cat .pqc
```

Endzustand (Soll)

```
$ cat .pqc
{"threshold": 0.75, "steps": 500, "ratio": 0.3}
```

Endzustand (Ist)

```
$ cat .pqc
{"threshold": 0.75, "steps": 500, "ratio": 0.3}
```

J.2.6 T6 • Clean

Ausgangslage

Der Soll-Endzustand von T5.

Eingabe

```
pqc clean
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
‘-- data.zip

0 directories, 1 file
```

Endzustand (Ist)

```
$ tree -L 2 -n --charset=ASCII
.
‘-- data.zip

0 directories, 1 file
```

J.2.7 T7 • Initialisierung nach Clean

Ausgangslage

Der Soll-Endzustand von T6.

Eingabe

```
pqc init
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
`-- data.zip

3 directories, 1 file
```

Endzustand (Ist)

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
`-- data.zip

3 directories, 1 file
```

J.2.8 T8 • Direkt trainieren

Ausgangslage

<dir> ist initialisiert.

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
`-- data.zip

3 directories, 1 file
```

Eingabe

```
pqc train
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
|-- bottlenecks
|   |-- defekt
|   `-- intakt
```

```
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   [...]
|-- logdir
|   |-- train
|   `-- validation
|-- retrained_graph.pb
|-- retrained_labels.txt
`-- test
    |-- defekt
    `-- intakt

13 directories, 9 files
```

Endzustand (Ist)

```
$ tree -L 2 -n --charset=ASCII
.
|-- bottlenecks
|   |-- defekt
|   `-- intakt
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   [...]
|-- logdir
|   |-- train
|   `-- validation
|-- retrained_graph.pb
|-- retrained_labels.txt
`-- test
    |-- defekt
    `-- intakt

13 directories, 9 files
```

J.2.9 T9 • Reshuffle

Ausgangslage

Der Soll-Endzustand von T8.

Eingabe

```
pqc reshuffle
```

Endzustand (Soll)

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   |-- classify_image_graph_def.pb
|   |-- cropped_panda.jpg
|   |-- imagenet_2012_challenge_label_map_proto.pbtxt
|   |-- imagenet_synset_to_human_label_map.txt
|   |-- inception-2015-12-05.tgz
|   `-- LICENSE
|-- logdir
|   |-- train
|   `-- validation
`-- test
    |-- defekt
    `-- intakt

10 directories, 7 files
```

Endzustand (Ist)

```
$ tree -L 2 -n --charset=ASCII
.
|-- data
|   |-- defekt
|   `-- intakt
|-- data.zip
|-- inception
|   |-- classify_image_graph_def.pb
|   |-- cropped_panda.jpg
|   |-- imagenet_2012_challenge_label_map_proto.pbtxt
|   |-- imagenet_synset_to_human_label_map.txt
|   |-- inception-2015-12-05.tgz
|   `-- LICENSE
|-- logdir
|   |-- train
|   `-- validation
`-- test
    |-- defekt
    `-- intakt

10 directories, 7 files
```

Glossar

1. Fotoserie Während dieser Machbarkeitsstudie wurden zwei Fotoserien aufgenommen. Die erste dieser Serien wurde bei einem Besuch in der Produktionsstätte angefertigt. 49–51

2. Fotoserie Während dieser Machbarkeitsstudie wurden zwei Fotoserien aufgenommen. Die zweite dieser Serien wurde unter Laborbedingungen angefertigt. Die Qualität der Aufnahme in der zweiten Serie ist daher viel besser als die der Ersten. 41, 49–53, 67

ACC Genauigkeit (accuracy). 54, 60, 63, 66

API Application Programmer Interface. 39–41

AUC Area Under The Curve. 54, 66

Aufnahme Als Aufnahme wird ein Foto bezeichnet. Eine Aufnahme ist üblicherweise Teil einer Fotoserie. 31, 49–51, 53, 58, 59, 64–67, 109

CNN Convolutional Neural Network. 49, 52

FFI Foreing Function Interface. 39

FN False Negatives. 37, 38

FNR False Negative Rate oder Miss Rate. 37, 38, 61, 63–66

Fotoserie Als Fotoserie wird eine Menge von klassifizierten Daten bezeichnet. 49–51, 59, 67

FP False Positive. 38

FPR False Positive Rate oder Fall-out. 37, 38, 61, 63–66

ILSVRC14 ImageNet Large-Scale Visual Recognition Challange 2014. 52

klassifizierte Daten Als klassifizierte Daten werden beliebige Mengen von Aufnahmen bezeichnet, die als defekt und intakt klassifiziert sind. 40–44, 54, 55, 58, 60, 63, 108, 109

MCC Matthews Correlation Coefficient. 54, 60, 63, 66

N Negatives. 38

P Positives. 38

Packung Als Packung wird eine Verpackung samt Inhalt bezeichnet. Wird also die Packung als defekt bezeichnet kann das heissen, dass die Verpackung oder der Inhalt defekt ist. Im Rahmen dieser Arbeit wird allerdings nur auf die Verpackung eingegangen. Deshalb ist mit einer defekten Packung stets eine defekte Verpackung gemeint. iii, iv, 31–33, 35, 36, 38, 40, 41, 49–53, 61–65, 69, 70, 108, 109

pqc Packet Quality Checker. 39, 44, 46, 47, 49, 52, 54, 55, 58–62, 65, 66, 68, 69, 108–111, 125, 132

TensorFlow TensorFlow. iii, 39–41, 52, 54, 55, 59, 63, 64, 69, 72, 110, 111

Testdaten Als Testdaten wird eine Menge von klassifizierten Daten bezeichnet, die zum Validieren eines Modells verwendet wird.. 35, 40–44, 54, 55, 58–60, 63, 65, 66, 108, 109

Trainingsdaten Als Trainingsdaten wird eine Menge von klassifizierten Daten bezeichnet, die zum trainieren eines Modells verwendet wird. 35, 40, 42–44, 49, 52, 54, 55, 58–61, 63, 65–67, 69, 108–110

Literatur

[1] Alex Alemi. *Improving Inception and Image Classification in TensorFlow*. Aug. 2016. URL: https://research.googleblog.com/2016/08/improving-inception-and-image.html (besucht am 08. 12. 2016).

[2] Jason Brownlee. *Classification Accuracy is Not Enough: More Performance Measures You Can Use*. Hrsg. von Machine Learning Mastery. 21. März 2014. URL: http://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/ (besucht am 30. 09. 2016).

[3] Chris Rupp & die SOPHISTen. *Requirements-Engineering und -Management*. 6. Aufl. Carl Hanser Verlag, 2014.

[4] Google Developers, Hrsg. *TensorFlow For Poets*. URL: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/ (besucht am 09. 12. 2016).

[5] *Git*. URL: https://git-scm.com/ (besucht am 07. 11. 2016).

[6] "IEEE Standard for Software and System Test Documentation". In: *IEEE Std 829-2008* (Juli 2008), S. 1–150. DOI: 10.1109/IEEESTD.2008.4578383.

[7] Stanford Vision Lab, Hrsg. *Results of ILSVRC2014*. 2014. URL: http://www.image-net.org/challenges/LSVRC/2014/results (besucht am 08. 12. 2016).

[8] Christian Szegedy u. a. "Going Deeper With Convolutions". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juni 2015. URL: http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf.

[9] TensorFlow, Hrsg. *Download and Setup*. URL: https://www.tensorflow.org/versions/r0.11/get_started/os_setup.html#pip-installation (besucht am 22. 11. 2016).

[10] TensorFlow, Hrsg. *Inception in TensorFlow*. URL: https://github.com/tensorflow/models/tree/master/inception#how-to-train-from-scratch (besucht am 08. 12. 2016).

[11] TensorFlow, Hrsg. *TensorBoard: Visualizing Learning*. URL: https://www.tensorflow.org/versions/r0.11/how_tos/summaries_and_tensorboard/index.html (besucht am 09. 11. 2016).

[12] Wikipedia, Hrsg. *Confusion Matrix*. 21. Sep. 2016. URL: https://en.wikipedia.org/wiki/Confusion_matrix (besucht am 29. 09. 2016).

Abbildungsverzeichnis

Listings

Tabellenverzeichnis