

## **Aufgabenfokussierung auf Autoencoder und automatisches Transferlernen**

Sebastian Hoch

### **MASTERARBEIT**

zur Erlangung des akademischen Grades Master of Science (M.Sc.)

Studiengang Informatik Master

Fakultät Elektrotechnik, Medizintechnik und Informatik  
Hochschule für Technik, Wirtschaft und Medien Offenburg

30 Juni 2020

Durchgeführt bei der PSIORI GmbH

Betreuer

Prof. Dr.-Ing. Janis Keuper, Hochschule Offenburg  
Dr. rer. nat. Sascha Lange, PSIORI GmbH

**Hoch, Sebastian:**

Aufgabenfokussierung auf Autoencoder und automatisches Transferlernen / Sebastian Hoch.

–

MASTERARBEIT, Offenburg: Hochschule für Technik, Wirtschaft und Medien Offenburg,  
2020. 35 Seiten.

**Hoch, Sebastian:**

Task focusing on autoencoder and automatic transfer learning / Sebastian Hoch. –

MASTER THESIS, Offenburg: Offenburg University, 2020. 35 pages.

## **Vorwort**

Die vorliegende Masterarbeit, habe ich als Abschlussarbeit meines Studiums der Informatik an der Hochschule Offenburg und meines Praktikums bei der PSIORI GmbH geschrieben. Ziel war es, neue Werkzeuge zum Transferlernen bereitzustellen und zu evaluieren. Anfang bis Mitte 2020 habe ich mich intensiv mit der Entwicklung und dem Schreiben der Masterarbeit beschäftigt.

Die Idee und die Fragestellung der Abschlussarbeit habe ich zusammen mit meinem Betreuer Dr. Sascha Lange entwickelt. Durch seine Fachentnisse im Bereich der Data Science konnte ich wichtige Einblicke in die Materie gewinnen.

Während meiner Arbeiten waren meine Betreuer, Prof. Dr.-Ing. Janis Keuper und Dr. rer. nat. Sascha Lange, und mein Kollege, Flemming Biegert immer erreichbar. Sie beantworteten meine Fragen, gaben wertvollen Input für die methodische Vorgehensweise und unterstützen mich, wann immer es notwendig war, sodass ich meine Masterarbeit erfolgreich durchführen konnte.

Ich wünsche Ihnen viel Spaß beim Lesen dieser Arbeit.

Sebastian Hoch

Waldkirch, Juni 2020

## **Eidesstattliche Erklärung**

Hiermit versichere ich eidesstattlich, dass die vorliegende Thesis von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Die Arbeit lag in gleicher oder ähnlicher Fassung noch keiner Prüfungsbehörde vor und wurde bisher nicht veröffentlicht. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Offenburg öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Offenburg, 30 Juni 2020

Sebastian Hoch

## Zusammenfassung

### ***Aufgabenfokussierung auf Autoencoder und automatisches Transferlernen***

Hohe Kosten bei der annotation von Daten führen dazu, dass datensparsamere Wege zum Erstellen von Modellen gesucht werden. In dieser Arbeit wird ein Lösungsansatz untersucht, der ausgehend von fokussierten Repäsentationen datensparsame Lösungen für verschiedene Aufgaben finden soll. Durch einen Mehrfach-Aufgaben-Ansatz trägt das Finden einer Reräsentation gleichzeitig zum Lösen einer Aufgabe bei. Durch Ersetzung einer der Mehrfach-Aufgaben können wissentransfers datensparsam auf neue Aufgabe durchgeführt werden. In der erarbeitetend und evaluierteren Lösung können Parameter automatisch gefunden werden. Bei einem Vergleich von verschiedenen Ansätzen und einem Vergleich mit verschiedenen Datenmegen ist über die Leistung der Netzwerke zu erkennen, dass der Ansatz insbesondere mit weniger Daten bessere Ergebnisse erzielt. Die gute Leistung der Ansätze motiviert zu einer Bereitstellung als Module. Die Module werden im Rahmen dier Arbeit beschrieben. Abgeschlossen wird die Arbeit mit einem Ausblick auf verbesserungen und potentielle der Ansätze.

Zusammenfassung prüfen

## Abstract

### ***Task focusing on autoencoder and automatic transfer learning***

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>5</b>
<b>2. Grundlagen</b>	<b>7</b>
2.1. Autoencoder . . . . .	7
2.2. Transferlernen . . . . .	8
2.2.1. Tiefes Transferlernen . . . . .	8
2.2.2. Halbüberwachtes Lernen . . . . .	9
2.2.3. Multi-Task-Lernen . . . . .	10
2.3. Automatisiertes maschinelles Lernen . . . . .	11
2.3.1. Hyperparameter-Optimierung . . . . .	11
2.3.2. Meta-Learning . . . . .	12
2.3.3. Neural Architecture Search . . . . .	12
2.3.4. Optimierungstechniken . . . . .	12
2.4. Bibliotheken und Werkzeuge . . . . .	14
2.5. Einordnung und bestehende Systeme . . . . .	16
2.6. Datenverständnis . . . . .	19
2.7. Datenvorbereitung . . . . .	20
<b>3. Experimente und Werkzeuge</b>	<b>23</b>
3.1. Greifererkennung auf Autoencoder . . . . .	23
3.2. Mutli-Task Experiment . . . . .	23
3.2.1. Werkzeug . . . . .	24
3.2.2. Ergebnis . . . . .	24
3.3. ConvolutionalSecondCriterionAutoenocder . . . . .	25
3.4. TransferSecondCriterionAutoenocder . . . . .	28
3.5. AutoTransferSecondCriterionAutoenocder . . . . .	29
<b>4. Transfer Lernen</b>	<b>33</b>
4.1. Transfer Experiment . . . . .	33
4.1.1. Werkzeug . . . . .	33
4.1.2. Ergebnis . . . . .	33
4.2. Auto Transfer Experiment . . . . .	33
4.2.1. Werkzeug . . . . .	34
4.2.2. Ergebnis . . . . .	34

## Inhaltsverzeichnis

---

4.3. Transfer-Experiment . . . . .	34
<b>5. Fazit</b>	<b>35</b>
5.1. Zusammenfassung . . . . .	35
5.2. Kritische Reflexion . . . . .	35
5.3. Ausblick und weitere Arbeiten . . . . .	35
5.3.1. Transfer auf Greiferdatensatz . . . . .	35
5.3.2. Autocrane-Datensatz . . . . .	35
5.3.3. Mehrfache Aufgaben . . . . .	35
5.3.4. Flexibilität der Werkzeuge . . . . .	35
<b>Abkürzungsverzeichnis</b>	<b>i</b>
<b>Tabellenverzeichnis</b>	<b>iii</b>
<b>Abbildungsverzeichnis</b>	<b>v</b>
<b>Quellcodeverzeichnis</b>	<b>vii</b>
<b>Literatur</b>	<b>ix</b>
<b>A. Anhang Basislinie Greifer</b>	<b>xiii</b>
<b>B. Anhang Basislinie Baumstämme</b>	<b>xv</b>

Quellen prüfen +  
vollständigen



# **Todo list**

Zusammenfassung prüfen . . . . .	iii
Zusammenfassung auf Englisch . . . . .	iv
Abstract ins Englische übersetzen . . . . .	iv
Quellen prüfen + Vervollständigen . . . . .	1
Metrik im Detail beschreiben . . . . .	18
Generierter Datensatz? . . . . .	20
Iou Grapple . . . . .	23
Werkezug beschreiben . . . . .	24
weight durch Verhältnis w1/w2 anpassen. . . . .	31
Worker Budget berücksichtigen / notieren . . . . .	31
TL einfügen . . . . .	33
Abb gewichtung . . . . .	34
todo bild datenmenge . . . . .	34
Adresse-Datensätze . . . . .	35
Lizenz . . . . .	35



# 1. Einleitung

Die PSIORI GmbH [PS20] ist ein Projekt- und Beratungsunternehmen im Bereich der Digitalisierung und Data Science. In vielen Digitalisierungs- und Automatisierungsprojekten fallen eine Vielzahl von Aufgaben an, welche mittels künstlicher Intelligenz gelöst werden können. In der Regel wird dabei für jede Aufgabe, welche mittels künstlicher Intelligenz gelöst werden soll, eine eigene Annotation benötigt. Das Annotieren von Daten ist teuer. Zum Beispiel sollten in einem Projekt circa 80.000 Bilder mittels zehn verschiedenen Annotationen beschriftet werden, was zu Kosten von circa 240.000 Euro führt. Zusätzliche Kosten entstehen durch Personenaufwände beim Erstellen ähnlicher Modelle für Aufgaben in einer Domäne. Die Kosten solcher Data Science Projekte können also gesenkt werden, wenn weniger annotierte Daten genutzt werden müssen.

Ziel dieser Arbeit ist es, herauszufinden, ob es ausgehen von Datenrepräsentationen einer Domäne möglich ist, den Einsatz von annotierten Daten zu reduzieren. Im Idealfall ist es möglich ausgehend von einer geeigneten Repräsentation schnell und robust neue Aufgaben in der Domäne bearbeiten zu können. Die eingesetzten Techniken sollen dabei nachhaltig bereitgestellt werden und von anderen Data Scientisten eingesetzt werden können.

Das Vorgehen des praktischen Teils der Arbeit orientiert sich an dem CRISP-DM Model [Sh00], erfolgt also iterativ. Um die Ergebnisse nachhaltig anderen Entwicklern zur Verfügung zu stellen, wird vor die Modellierungsphase eine Phase zur Werkzeugerstellung eingefügt. Die Werkzeuge unterstützen einen Data Scientisten bei der Anwendung der vorgeschlagenen Techniken. Die durchgeführten Experimente sollen insbesondere zeigen, dass die Werkzeuge und die dahinterstehende Techniken funktionieren.

Dieses Dokument gliedert sich in ein Grundlagenkapitel, in welchem die notwendigen theoretischen Grundlagen, das Umfeld sowie die genutzten Datensätze der

## 1. Einleitung

---

Arbeit vorgestellt werden. Im Hauptteil der Arbeit wird ein Multi-Task-Ansatzes vorgestellt und evaluiert. Aufbauend auf diesem Ansatz wird ein Transfer-Learning Ansatz mit einer Erweiterung des automatischen maschinellen Lernens vorgestellt und evaluiert. Abgeschlossen wird die Arbeit mit einer Bewertung der Lösung der Problemstellung und einem größeren Ausblick auf mögliche Erweiterungen und Potentiale.

## 2. Grundlagen

Im folgenden Kapitel werden die Grundlagen der Arbeit, insbesondere im Hinblick auf die gewählten Ansätze beschrieben. Begonnen wird mit den theoretischen Grundlagen, gefolgt von einer Beschreibung der bestehenden Systeme und abgeschlossen wird mit einem Abschnitt über die Daten und ihrer Vorverarbeitung.

### 2.1. Autoencoder

Autoencoder [DJ87] sind Werkzeuge welche insbesondere zum Finden von Repräsentationen eingesetzt werden. Dabei komprimieren Sie die Eingabe in einen niedrigdimensionaleren Raum und rekonstruieren aus diesem Code die Eingabe. Konkret besteht ein Autoencoder aus drei Teilen, dem Encoder, dem Codelayer und dem Decoder. In Abbildung 2.1 ist das Schema eines Autoencoders abgebildet. In der einfachsten Form besteht ein Autoencoder aus einer Eingabeschicht, einer

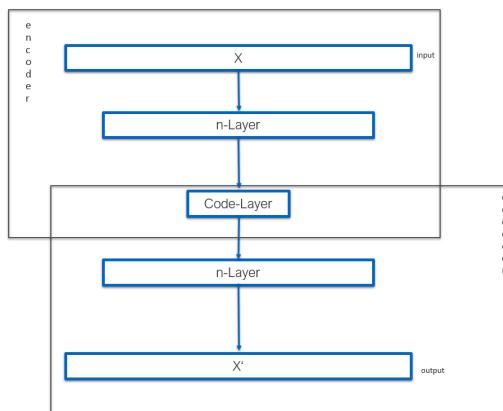


Abbildung 2.1: Schema Autoencoder

versteckten Schicht und eine Ausgabeschicht. Sie können aber auch mit mehreren Schichten, also mit 'tiefe Architekturen' genutzt werden. [HS06] Der Encoder lässt

## 2. Grundlagen

---

sich auch vereinfacht als die Funktion  $F(x) = c$  und der Decoder als die Funktion  $F(c) = x'$  darstellen wobei  $x \stackrel{!}{=} x'$  sein soll. Autoencoder gibt es in vielen verschiedenen Ausführungen. Dabei sind die meisten Typen von Autoencoder unvollständige Autoencoder. Die verborgene Schicht, das Codelayer enthält weniger Informationen als die Eingabe. Hierdurch wird die Dimensionsreduzierung erzwungen. Andere Aufgaben können Entrauschung mittels Denoising autoencoder [Vi08] oder Generierung neuer Datenpunkte mittels Variational Autoencoder [KW19] sein. Bei Contractive autoencoder [Rifai.201] nutzen einen Regularisierer in der Zielfunktion um das Modell zu zwingen eine Funktion zu lernen, die flexibler auf Variationen der Eingabewerte reagiert.

Für die Arbeit ist der Convolutional Autoencoder [Ma11] kurz CAE<sup>1</sup> interessant. Er ist ein Stacked Autoencoder welcher Faltungsschichten (Convolutional Layer) [Le99] integriert.

**Schichtenweise Vortrainieren** Im schichtenweise Vortrainieren werden einzelne Schichten eines neuronalen Netzwerkes vor dem eigentlichen Training trainiert und die Leistung des gesamten Models zu erhöhen. [Be07] Bei (symmetrischen) Autoencodern werden dabei die zueinander symmetrischen Schichten des Encoders und Decoders miteinander trainiert. Die Zielgröße ist dabei die Rekonstruktion der Eingabedaten.

## 2.2. Transferlernen

Im traditionellen maschinellem Lernen wird pro Aufgabe und Datenset ein isoliertes Modell erstellt. Das Transferlernen hat das Ziel Wissen zu teilen. Die Isolation der Modelle soll aufgehoben werden. Dabei erfolgt eine Aufteilung in Quell und Zieldomäne.

### 2.2.1. Tiefes Transferlernen

Geprägt durch die Entwicklung hin zu tiefen neuronalen Netzwerken wurden Methoden zum tiefen Transferlernen (deep transfer learning) vorgeschlagen. [Ti18]

---

<sup>1</sup>Convolutional Autoencoder

ordnet diese in vier Kategorien ein. Für eine Einordnung von klassischen Methoden des Transferlernens eignet sich die Erhebungen [Fu].

**Instanzbasiert** Instanz-basierte Ansätze ergänzen, mittels geeigneter Strategie, Gewichte in der Zieldomäne mit Gewichten aus der Quelldomäne.

**Abbildungbasiert** Abbildungs-basierte Ansätze bilden Instanzen aus der Quell- und Zieldomäne in einen neuen Datenraum ab. In dem neuen Datenraum sind die Instanzen aus den zwei Bereichen ähnlich und können für ein gemeinsames Training eines tiefen neuronales Netzwerk genutzt werden.

**Netzwerkbasiert** Der netzwerkbasierte Ansatz wiederverwendet einen Teil des in der Quelldomäne trainierten Netzwerkes in der Zieldomäne. Es werden dabei sowohl die Netzstruktur als auch die Verbindungsparameter übernommen. Die Netzwerke werden dabei in zwei Teile unterteilt. Der erste Teil ist die sprachunabhängige Merkmalstransformation und der zweite Teil ist der sprachabhängige Klassifikator. Dabei kann die sprachunabhängige Merkmalstransformation in der Zieldomäne wiederverwendet werden.

**Gegnerischbasiert** Gegenerischbasierte tiefes Transferlerning ist von dem Ansatz "erzeugende gegnerische Netzwerke"(Generative Adversarial Networks) [Ia14] inspiriert. Es werden übertragbare Repräsentationen gesucht, die sowohl auf die Quell- als auch auf die Zieldomäne anwendbar sind.

### 2.2.2. Halbüberwachtes Lernen

Halbüberwachtes Lernen ist eine Methode, die sich zwischen unüberwachtes und überwachtem Lernen einordnet. Methoden des unüberwachten Lernens arbeiten komplett ohne annotierte Daten, überwachtes Lernen arbeitet mit vollständig annotierten Daten. Das halbüberwachte Lernen arbeitet mit Teilweise annotierten Daten. Die Anzahl der nicht annotierten Daten übersteigt, in der Regel, die Menge der annotierten Daten. Diese Technik reduziert Annotationskosten durch den Einsatz weniger Daten mit Annotationen. Zu beachten ist dabei, dass sowohl die Beschrifteten als nicht beschriftete Daten aus der gleichen Verteilung entnommen werden.

## 2. Grundlagen

---

Im Gegensatz dazu sind bei dem Transferlernen die Datenverteilungen der Quell- und Zieldomäne oft unterschiedlich. [CSZ10]

### 2.2.3. Multi-Task-Lernen

Werden mehrere Aufgaben parallele gelernt spricht man von dem Multi-Task-Lernen (MTL<sup>2</sup>). Ziel ist es, Wissen durch gleichzeitiges Lernen einiger verwandter Aufgaben weiterzugeben. Es wird davon ausgegangen, dass das Lernen einer Aufgabe das Lernen der anderen Aufgaben verbessert. Im Allgemeinen wird dies durch das Lernen aller Aufgaben gemeinsam erreicht, wobei die korrelierten Informationen zwischen den einzelnen Aufgaben genutzt werden. Die Aufgaben erzeugen dabei eine niedrigdimensionale Repräsentation welche dann durch das parallele Lernen besser generalisiert. In manchen Fällen hat sich auch herausgestellt, dass Multi-Task-Lernen zum Lernen von nicht verwandten Aufgaben vorteilhaft ist. Basierend auf den Ein- und Ausgängen wird MTL in drei Fälle aufgeteilt.

**Single-Input Multi-Output** SIMO wird verwendet, um aus einer Eingabe Vorhersage von verschiedenen Arten von Ausgabezielen zu treffen. Diese Art des MTL wird auch Mehrklassen-Lernen (multi-class learning) genannt.

**Multi-Input Single-Output** MISO bedeutet es werden mehrere Eingaben zum Vorhersagen eines Ausgabeziels genutzt.

**Multi-Input Multi-Output** MIMO bedeutet es werden mehrere Eingaben zum Vorhersagen von verschiedenen Arten von Ausgabezielen genutzt.

Abbildung 2.2 zeigt die verschiedenen Ausprägungen künstlicher neuronalen Netze die auf MTL beruhen.

Induktiven Transferlernen und MTL wird darin unterschieden, dass beim induktiven Transferlernen angenommen wird, dass es eine Hauptaufgabe und eine Nebenaufgabe gibt. Die Nebenaufgabe bietet zusätzliche Informationen, um die Hauptaufgabe zu verbessern bzw. zu generalisieren. Im MTL gibt es keine solche Unterscheidung, die Aufgaben werden gleichberechtigt betrachtet. Induktiven Transfer-Lernen kann deshalb als Sonderform des MTL gesehen werden. [TW18] [PQ10]

---

<sup>2</sup>Multi-Task-Lernen

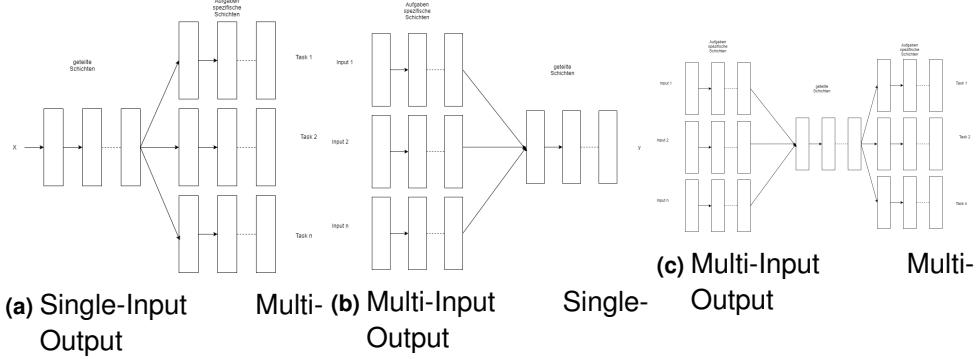


Abbildung 2.2: Multi-Task-Lernen: Ausprägungen

## 2.3. Automatisiertes maschinelles Lernen

Das automatisierte maschinelles Lernen kurz AutoML<sup>3</sup> hat das Ziel alle Aspekte des maschinellen Lernens und der Datenanalyse-Pipeline zu automatisieren. Die vollständige Automatisierung erlaubt es auch Nutzern ohne oder mit geringen Kenntnissen von ML-Techniken die Erstellung von ML-Systemen. Die volle Automatisierung ist ein langfristiges Ziel. Aktuelle Systeme sind halbautomatisch und ziehen darauf ab, Personenaufwände bei Bedarf nach und nach durch Rechenvorgänge zu reduzieren. Trotz der steigenden Rechenleistung, können AutoML-Methoden sehr rechenintensiv sein. AutoML wird in drei Methoden eingeordnet. Das Meta-Learning, Neural Architecture Search kurz NAS<sup>4</sup> und Hyperparameter-Optimierung kurz HPO<sup>5</sup>. [HKV19]

### 2.3.1. Hyperparameter-Optimierung

Hyperparameter sind alle Parameter, welche vor Beginn des Trainings zur Steuerung des Trainings eingestellt werden können. Eine passende Einstellung dieser Parameter beeinflusst die Leistung eines Modells maßgeblich. In [KJ95] wurde festgestellt, dass verschiedene Hyperparameterkonfigurationen für verschiedene Datensätze am besten funktionieren. Es ist also notwendig für jede Aufgabe aufs Neue die beste Hyperparameterkonfigurationen zu finden. Automatische-HPO ist die Technik des automatischen Setzens der Hyperparameter um die Leistung zu optimieren. Dabei hat sie insbesondere drei Ziele. Es sollen Personenaufwände bei der Anwen-

<sup>3</sup>automatisiertes maschinelles Lernen

<sup>4</sup>Neural Architecture Search

<sup>5</sup>Hyperparameter-Optimierung

## 2. Grundlagen

---

dung von maschinellem Lernen zu reduziert werden. Es soll die Leistung von Algorithmen und Modellen des maschinellen Lernens verbessern. In der Wissenschaft sollen soll die Reproduzierbarkeit und Fairness von Studien verbessert werden, da Automatische-HPO einfacher reproduzierbar ist als manuelle-HPO. In dem Kapitel 2.3.4 werden einige gängige Optimierungstechniken der automatischen-HPO erläutert. [FH19]

### 2.3.2. Meta-Learning

Unter [Jo18] ist eine Übersicht über das Meta-Learning zu finden. In dieser Arbeit wird das Thema nur vollständigkeitshalber aufgelistet. Meta-Learning umfasst jede Art von Lernen, welches auf frühere Erfahrungen zurückgreift. Dabei können umso mehr Arten von Metadaten genutzt werden je ähnlicher die Aufgaben sind. Metadaten sind dabei alle Daten die frühere Lernaufgaben beschreiben. Dies können z.B Algorithmuskonfigurationen, Hyperparameter, Netzarchitekturen, Modellbewertungen und vieles mehr sein.

### 2.3.3. Neural Architecture Search

Im Deep Learning hängt die Leistung eines Modells maßgeblich von der genutzten Architektur ab. Das manuelle Suchen von Architekturen ist zeitaufwendig und fehleranfällig. Die Neural Architecture Search befasst sich damit, wie Architekturen automatisch gefunden werden können. In dieser Arbeit wurde nicht auf die Technik der Neural Architecture Search zurückgegriffen und das Thema ist nur vollständigkeitshalber aufgeführt. Unter [EMH19] kann eine Übersicht über die Neural Architecture Search gefunden werden.

### 2.3.4. Optimierungstechniken

In diesem Unterkapitel werden gängige Optimierungsmethoden des AutoML dargestellt. Die Auflistung ist nicht vollständig, deckt aber die im praktischen Teil der Arbeit zur Verfügung stehenden Methoden ab.

**Rastersuche**

Die Rastersuche ist eine modellunabhängige Optimierungsstrategie. Es werden Parameterkombinationen definiert und anschließend Modelle ausgehend von den Kombinationen erstellt und evaluiert. Diese Technik hat einige Schwächen. Es müssen Parameterkombinationen definiert werden, es muss jedes Modell trainiert werden und wenn die optimale Konfiguration nicht enthalten ist, wird sie nie gefunden. [Mi18]

**Zufallssuche**

Die Zufallssuche ähnelt der Rastersuche. Sie unterscheidet sich dahingehend, dass die Parameterkombinationen nicht mehr definiert werden müssen. Es werden zufällige Stichprobenkonfigurationen aus dem (definierten) Parameterraum gezogen und evaluiert. In [Be12] wurde gezeigt, dass insbesondere bei unterschiedlicher Wichtigkeit der Hyperparameter, bessere Ergebnisse erzielt werden.

**Bayesian optimization**

Bayesian optimization ist ein Ansatz welcher zur Optimierung von Zielfunktionen, die eine lange Zeit (Minuten oder Stunden) zur Auswertung benötigen. Im Gegensatz zur Rastersuche oder Zufallssuche ist der Ansatz modellabhängig. Der Ansatz ist iterativ und baut auf zwei Komponenten auf. Ein probabilistisches Ersatzmodell und einer Erfassungsfunktion die zur Bewertung welcher Punkt als nächstes bewertet werden soll herangezogen wird. Das Ersatzmodell wird in jeder Iteration an alle Beobachtungen angepasst. Im Gegensatz zur Blackboxfunktion, ist die Auswertung der Erfassungsfunktion billig und kann somit zur Optimierung herangezogen werden. [Fr]

**Hyperband**

Hyperband [Li17] erweitert Successive Halving [JT]. Successive Halving weiß einer Reihe von Hyperparameter-Konfigurationen ein einheitliches Menge an Ressourcen zu und berechnet die Leistung von allen Konfigurationen und entfernt die schlechtere Hälfte. Die Konfigurationen werden dabei zufällig gezogen. Das Ganze

## 2. Grundlagen

---

wird wiederholt, bis eine Konfiguration übrig bleibt. In jedem Durchlauf wird der übriggebliebenen Hälfte exponentiell mehr Ressourcen zugewiesen. SH benötigt als Eingangsparameter die Ressourcen und die Anzahl an Konfigurationen. Dabei ist es schwierig, zu entscheiden, ob wenige Konfigurationen mit mehr Ressourcen oder viele Konfigurationen mit weniger Ressourcen durchgeführt werden sollen. Hyperband erweitert SH um dieses Problem zu adressieren. Hyperband führt eine Rastersuche für die beiden Parameter durch. Es werden also mehrere SH-Durchläufe mit diversen Konfigurationen durchgeführt. Die Anzahl an Konfigurationen wird dabei immer weiter reduziert. Abgeschlossen wird eine Ausführung mit einer normalen Gittersuche.

### **BOHB**

BOHB [SAF18] ist ein Ansatz, der Bayesian optimization und Hyperband kombiniert. Dabei wird Bayesian optimization zur Auswahl von Konfigurationen herangezogen und Hyperband bestimmt wieviele Konfigurationen mit welchem Budget ausgeführt werden sollen. Anschließend werden die Konfigurationen mittels Successive Halving ausgeführt. Unter <https://github.com/automl/HpBandSter> kann eine Implementierung des Werkzeugs gefunden werden. Diese Framework wurde im praktischen Teil der Arbeit genutzt.

## 2.4. Bibliotheken und Werkzeuge

Für den praktischen Teil der Abschlussarbeit wurde insbesondere Cnvrg [[cnvrg.io.](#)] genutzt. Cnvrg.io ist eine "full-stack" Data Science Platform welche Werkzeuge für die Erstellung, Verwaltung, Bereitstellung und Automatisierung von maschinellem Lernen bereitstellt. Cnvrg erlaubt es Arbeitsbereiche mittels Containern zu erstellen. Die Container können dabei auf Maschinen in Azure [Mi20] zugreifen. Für die Experimente wurde ein vorgefertigter Container mit einer tesla-k80 [Nv20], fünf CPUs und 49 GB Arbeitsspeicher genutzt.

Für die Entwicklung wurden Python [Py20], Jupyter Notebooks [Pr] und das Framework Tensorflow [Ma15] genutzt. Die wichtigsten Bibliotheken für die Arbeit sind Keras [Ch15] , Numpy [Ol06] , Matplotlib [Hu07] , scikit-learn [Pe11] , ConfigSpace [Li] , Bayesian Optimization and Hyperband [SAF18].

Für die Visualisierung von Bildeinbettungen wurde die Software "PSIORI Visualizer" erweitert und eingesetzt. Die Software erlaubt es dreidimensionale Daten darzustellen. Dabei können Filter eingesetzt werden, der Blickwinkel geändert werden, die Daten mit zusätzlicher Information versehen werden und Zoomen ist möglich. Als zusätzliche Informationen kann z.B. ein Originalbild und seine Rekonstruktion oder die Information ob eine Vorhersage korrekt oder falsch wahr hinterlegt werden.

Kern der erstellten Softwaremodule ist das Framework Psipy [PS19]. Psipy ist ein Python-Framework für maschinelles Lernen welches von PSIORI selbst entwickelte Werkzeuge zusammenfasst und unter einer einheitlichen API zu Verfügung stellt. Diese API ist an die API des verbreiteten Frameworks scikit-learn angelehnt. Es können Modelle basierend auf scikit-learn und Tensorflow eingebunden werden. In den nachfolgenden Abschnitten werden die, für die Arbeit, wichtigsten bestehenden Module des Frameworks vorgestellt.

**saveable.py** Das Modul Saveable ist eine flexible Basisklasse die Kernfunktionalität zum Speichern und Laden von Python-Objekten bietet. Es können Modelle, welche diverse Bibliotheken nutzen auf eine einheitliche Art und Weise gespeichert werden. Um die Klasse Saveable nutzen zu können, müssen erbende Klassen ihre Konstruktorargumente an die Basisklasse übergeben. Zusätzlich ist es notwendig eine Erweiterung beim Speichern und Laden zu implementieren. Beim Speichern ist es notwendig eine Erweiterung, um alle (meist ein) Module und weitere Argumente zu implementieren. Beim Laden müssen die gespeicherten Module und Argumente geladen werden. In Listing 2.1 ist die Erweiterung zum Speichern eines Tensorflow-Models abgebildet.

```

1     ...
2     zip_file.add("model.h5", self.model)
3     ...

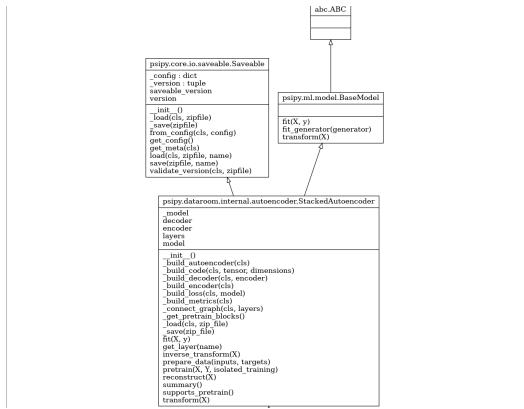
```

**Listing 2.1:** Erweiterung zum Speichern eines Tensorflow Models

**autoencoder.py** Das Modul Autoencoder enthält die drei Klassen StackedAutoencoder, FullyConnectedAutoencoder und ConvolutionalAutoencoder. Der StackedAutoencoder wird als Basisklasse für die anderen beiden Klassen genutzt. Im Konstruktor werden Methoden aufgerufen, welche in den abgeleiteten Klassen ausprogrammiert sind. Dabei wird ein Keras-Modell für einen Encoder und Decoder entsprechend von Parametern erstellt. Als weitere wichtige Methoden gibt es die

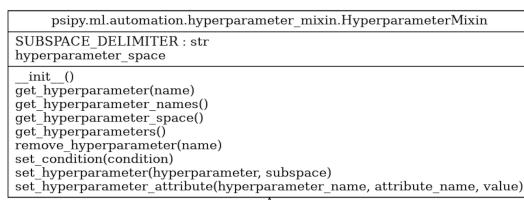
## 2. Grundlagen

Methode `pretrain(..)` und `fit(..)`. Mittels `pretrain(..)` werden die Schichten eines symmetrischen Autoencoder von außen nach innen wie in [Be07] beschrieben vortrainiert. Die Auswahl der Schichten erfolgt wieder in den abgeleiteten Klassen. In der `fit(..)`-Methode wird nach einigen Prüfungen die Methode `ffit(..)` [Ch15] des Kerasmodels aufgerufen. In Abbildung 2.3 ist das Klassendiagramm mit den öffentlichen Methoden des ConvolutionalAutoencoder dargestellt.



**Abbildung 2.3:** Klassendiagramm ConvolutionalAutoencoder

**hyperparameter\_mixin.py** Hyperparameter\_mixin wird zum standardisierten Verwalten von Hyperparametern für AutoML-Klassen genutzt. Auf die Hyperparameter kann anschließend einheitlich zugegriffen werden. Abbildung 2.4 zeigt das zugehörige UML-Klassendiagramm mit den Methoden zum Hinzufügen, Löschen und Laden der Hyperparameter. Da die Methoden öffentlich sind, können über jede erbende Klasse die Hyperparameter eigenständig verwaltet werden.



**Abbildung 2.4:** Klassendiagramm Hyperparametermixin

## **2.5. Einordnung und bestehende Systeme**

Die Bilddaten und Aufgabenstellungen der neuronalen Netzwerke sind in die Problemstellungen des Autocrane-Projekts von PSIORI einzuordnen. Es sind also echte

Datensätze und echte Problemstellungen, wobei die gezeigten Aufgabenstellungen und Modelle nicht zwingend in dem Autocrane-Projekt zum Einsatz kommen. Das Autocrane-Projekt ist ein (laufendes) Projekt, welches das Ziel hat, einen feststehenden Rundlaufkran vollautomatischen zu steuern. In Abbildung 2.5 ist ein Rundlaufkran abgebildet. Diese Art von Kran werden in holzverarbeitenden Anlagen zum Befüllen von Fülltrichtern oder Förderbändern eingesetzt. Der Kran kann sich um 360 Grad drehen. Der Greifer kann nach oben, unten und mittels eines Schlittens entlang eines Auslegers bewegt werden. Um die Bilder aufnehmen zu können, wurde an der Kabine am Hauptstandfuß eine Kamera angebracht. Die Kamera ist auf das Ende des Auslegers und den Bereich darunter ausgerichtet. Die Kamera bewegt sich also mit dem Rundlaufkran und somit ist der Greifer immer im Bild. Für das Autocrane-Projekt sind insbesondere drei Anwendungsfälle interessant. Die Baumstämme werden mittels LKW angeliefert und müssen nach vorgegebenen Regeln (z. B. Ausrichtung, freier Lagerplatz) als Holzstapel gelagert werden. Der Fülltrichter muss mit Holz aus den Holzstapeln befüllt werden. Der Fülltrichter muss mit Holz aus einem LKW befüllt werden. Es ergeben sich also Aufgabenstellungen wie Greifer-Erkennung, Baumstamm-Erkennung, LKW-Erkennung, Strategien für das entladen und aufbewahren der Baumstämme und vieles mehr. Im Normalbetrieb werden täglich 140-200 LKW entladen. Die Ladung ist 9 - 18 Meter lang und 34 - 40 Tonnen schwer. [PS20]



Abbildung 2.5: Rundlaufkran (Foto: ANDRITZ)

**Greifererkennung** Bei der Aufgabenstellung Greifer-Erkennung muss in einem Bild die Position eines Rahmens um den Greifer gefunden werden. Abbildung 2.6a zeigt ein Bild mit Rahmens um den Greifer. Es handelt sich um eine klassische Objekterkennungs-Aufgabe.



(a) Greifer mit Rahmen

(b) Greifer mit Baumstämmen

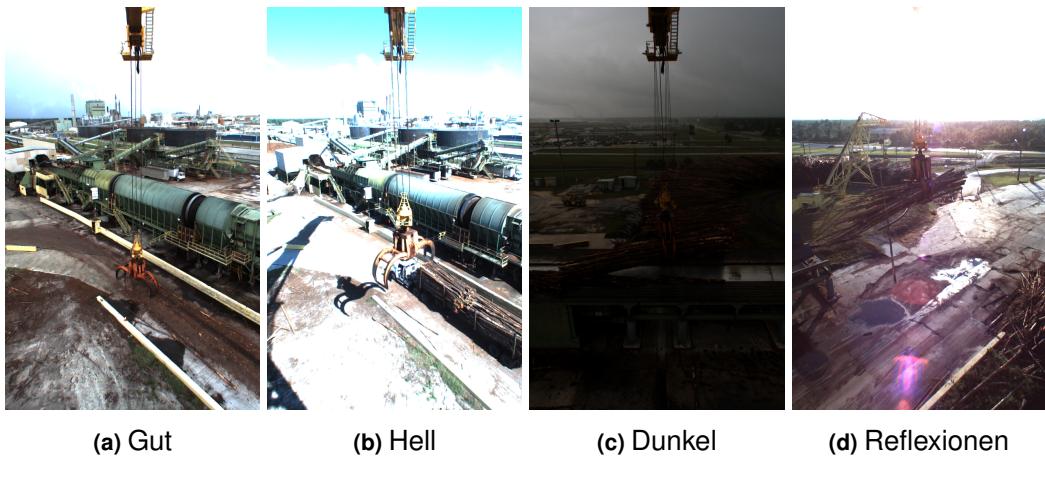
Abbildung 2.6: Greifer

PSIORI hat die Aufgabe mittels neuronalem Netzwerk gelöst. Dabei wurde auf die Technik des Single Shot MultiBox Detector (SSD) [Li16] zurückgegriffen. Die Vorhersagegenauigkeit dieses Modells wird als Basislinie und Vergleichswert genutzt. Dabei ist die ausschlaggebende Metrik die Intersection over Union mit einem Schwellenwert von 0.8. Das Modell erreicht einen Wert von 0.86. In Anhang ?? ist die vollständige Basislinie dargestellt.

**Baumstammklassifikation** Die Aufgabe der Baumstammklassifikation hat zum Ziel, zu erkennen ob sich Baumstämme im Greifer befindet oder nicht. Es handelt sich um eine Klassifikationsaufgabe. Für diese Aufgabe wurde von PSIORI ein Model erstellt. Dieses Model wird wie das Greifererkennungsmodel als Basislinie und Vergleichswert für die durchgeföhrten Versuche genutzt. Das Modell erreicht auf den Validation-Daten eine Accuracy von 0.9828%. Im Anhang B ist die vollständige Basislinie dargestellt.

## 2.6. Datenverständnis

Mittels Kamera an der Kabine können neue unbeschriftete Bilder aufgenommen und bei PSIORI abgelegt werden. Durch den Aufbau des Rundkrans und der Kameraposition befindet sich der Greifer immer im Bild. Der Hintergrund der Bilder ändert sich stark. Die geografische Lage des Rundkrans schränkt die möglichen Wetterlagen ein. Es fällt kein Schnee und es gibt wenige Regentage. In Abbildung 2.7 sind Ausprägungen der Bildqualität dargestellt. Abgesehen von Bildern in guter Qualität gibt es helle Bilder, dunkle Bilder und Bilder mit Reflexionen. Die Bilder sind 1024 auf 648 Pixel groß und in Farbe. Die einzelnen Pixel können dabei Werte zwischen 0 und 255 annehmen. Entsprechend der beiden Aufgabenstellungen Greifererkennung und Baumstammklassifikation werden Daten mit einer passenden Beschriftung benötigt.



**Abbildung 2.7:** Bildqualität

**Greiferdatensatz** Der Greifer Datensatz enthält Bilder, in welchen der Greifer mittels Rahmen markiert ist. Abbildung 2.6a zeigt ein beispielhaftes Bild mit markiertem Greifer. Die Annotationen der Position des Greifers wurde pro Bild in einer XML-Datei abgelegt. Konkret wurde die Position als x, y Koordinaten in der Form ymin, xmin, ymax und xmax abgespeichert. Über die vier Werte lässt sich problemlos ein Rahmen um den Greifer spannen. Der Datensatz besteht aus 4.684 durch Personen annotierten Bildern.

**Baumstammdatensatz** Der Baumstamm Datensatz enthält Bilder, welche die Annotation, ob sich Baumstämme im Greifer befinden oder nicht enthält. Die Bilder

sind durch zwei Ordner in Bilder mit und Bilder ohne Baumstämme aufgeteilt. Abbildung 2.6b zeigt ein Bild, in welchem der Greifer Baumstämme greift. In Abbildung 2.6a befinden sich keine Baumstämme im Greifer.

## 2.7. Datenvorbereitung

In Vorbereitung auf die Modellierungsphase wurde ein finaler Datensatz erstellt und Werkzeuge zum Laden und vorbereiten der Daten implementiert.

**Erste Iteration** Als Erstes wurden die Daten mittels Skripten in Training, Test und Validation Daten aufgeteilt. Anschließend wurden die Daten auf der Cnvrge-Plattform in einen versionierbarer Datenspeicher geladen. In Tabelle 2.1 ist die finale Datenaufteilung zu sehen. Die Greifer Daten sind in 70% Trainingsdaten und jeweils 15% Test und Validierungsdaten aufgeteilt. Die Baumstammdata sind in 80% Trainingsdaten 10% Testdaten und 10% Validierungsdaten getrennt worden. Die Trainingsdaten werden zum Trainieren der Modelle genutzt, die Testdaten zum Überprüfen der Modelle und die Validierungsdaten werden am Ende der Experimente als finale Überprüfung der Ergebnisse eingesetzt.

	Train	Test	Validation	Summe
<b>Greifer</b>	3.279	703	704	4.686
<b>Baumstämme j/n</b>	9.749	1.221	1.225	12.195

Tabelle 2.1.: Datenaufteilung - Train Test Validation

erter Datensatz? 

Für das Laden der Daten wurde ein Modul 'data\_loader.py' erstellt. Dieses Modul enthält die drei Klassen DataLoader, GrappleDataLoader, LogsDataLoader. DataLoader ist eine abstrakte Klasse, welche Methoden zum Laden der Train-, Test- und Validationdaten definiert. Sie liefern entsprechend eines Parameters, bis zur maximalen Anzahl an Bildern, Bilder als NumPyarray zurück. Die Klassen GrappleDataLoader und LogsDataLoader implementieren für den jeweiligen Datensatz die konkreten Methoden zum Laden der Daten.

Mittels dem Modul 'data\_preparation.py' und der Klasse Preprocessing werden die Bilder auf die passende Größe verkleinert oder vergrößert und auf den Wertebereich zwischen 0 und 1 normalisiert.

**Zweite Iteration** In der zweiten Iteration wurde ein neues Modul namens `data_generator_provider.py` erstellt. Da Bilder als Numpyarray direkt in den Speicher geladen werden können nicht beliebig viele Bilder genutzt werden. Dieses Problem wird von den Keras ImageDataGeneratoren [Ch15] adressiert. Sie erlauben es Bilder stapelweise bereitzustellen. Zusätzlich werden bei den Imagedatageneratoren direkt die Zielgrößen bereitgestellt. Das Modul implementiert jeweils für den Baumstamm Datensatz und den Greiferdatensatz eine Klasse zum Bereitstellen von ImageDatageneratoren.

Da die Modelle SIMO<sup>6</sup>-Modelle sind erfolgt zusätzlich noch eine Aufbereitung der Bereitstellung der Daten. Standardmäßig stellen die Generatoren Stapel mit Einträgen der Form  $X, Y$  bereit. Wobei X die Eingangsdaten sind und Y die Zielgröße. Zum Beispiel kann X ein Bild sein und Y die zugehörige Klasse. Die Werkzeuge mit zweitem Kriterium benötigen Generatoren die Einträge der Stapel der Form  $X, [X, Y]$  erzeugen. Die Zielgröße hat sich zu einer Liste mit zwei Größen verändert. Die erste Zielgröße entspricht den Eingangsdaten, sie werden für den Decoder-Ausgang genutzt. Die zweite Zielgröße wird für den zweiten Ausgang genutzt. Sie entspricht der Zielgröße eines normalen ImageDataGeneratoren. Das Ganze wird mit Hilfe der Klasse `tensorflow.keras.utils.Sequence` erreicht. Ihr wird im Konstruktor ein `ImageDataGenerator` übergeben. Bei der Bereitstellung eines Elementes wird der Rückgabewert des Generators angepasst. Die entscheidenden Codezeilen sind in Listing 2.2 dargestellt.

```
1 res = self.generator.next()
2 return res[0], [res[0], res[1]]
```

**Listing 2.2:** Aufbereitung Generatorergebnis in Python

Für das Vortrainieren werden Generatoren bereitgestellt, welche Stapel mit Einträgen der Form  $X, X$  erzeugen. Hierbei wird auf Standardfunktionalität der Klasse `ImageDataGenerator` zurückgegriffen.

Die ImageDataGeneratoren bieten Standardfunktionalität zur Bildverstärkung. Alle Generatoren normalisieren die Werte der Bilder zwischen 0 und 1. Die Generatoren für die Trainingsdaten führen noch zufällige Veränderungen der Helligkeit und Kanalverschiebungen durch. Bei den Baumstamm Daten werden die Bilder zusätzlich horizontal umgedreht.

---

<sup>6</sup>Single-Input Multi-Output



## 3. Experimente und Werkzeuge

Als Kostenfaktor in DataScience-Projekten wurden die hohen Kosten zum Annotieren von Daten ermittelt. Im Autocrane-Projekt werden viele Aufgaben in einer Domäne durchgeführt. Als Frage stellt sich, ob eine geeignete Repräsentation der Domäne gefunden und ausgehend von dieser Repräsentation datensparsam (kosten-sparsam) aufgaben in dieser Domäne gelöst werden können.

### 3.1. Greifererkennung auf Autoencoder

Autoencoder sind ein Ansatz um unüberwacht, also ohne konkrete Aufgabe / Annotation, Repräsentationen zu finden. Ausgehend von dieser Repräsentation soll die Aufgabe Greifererkennung gelöst werden. Hierzu wird auf die Datenrepräsentation des Autoencoders eine Regression ausgeführt. In Abbildung 3.1 sind die Ergebnisse von drei Durchläufen abgebildet. Dieses Lösung erreicht deutlich schlechtere Ergebnisse als die Basislinie mit einer IoU von .

Iou Grapple

In Abbildung ?? ist das Embedding des zugrundeliegenden Autoencoders abgebildet. Die Datenpunkte verteilen sich gut im Raum, bilden aber keine Merkmale des Greifers ab. In Abbildung ?? sind Bilder mit ihren Rekonstruktionen abgebildet. Es ist sehr deutlich zu erkennen, dass sich der stark ändernde Hintergrund herausfordernd ist und in erster Linie die Lichtverhältnisse gelernt wurden.

### 3.2. Mutli-Task Experiment

Im Ansatz 3.1 wurde als schwäche des Ansatzes die fehlende Fokusierung des Auto-encoders auf den Greifer ausgemacht. Multi-Task-Lernen soll durch gleichzeitiges

### 3. Experimente und Werkzeuge

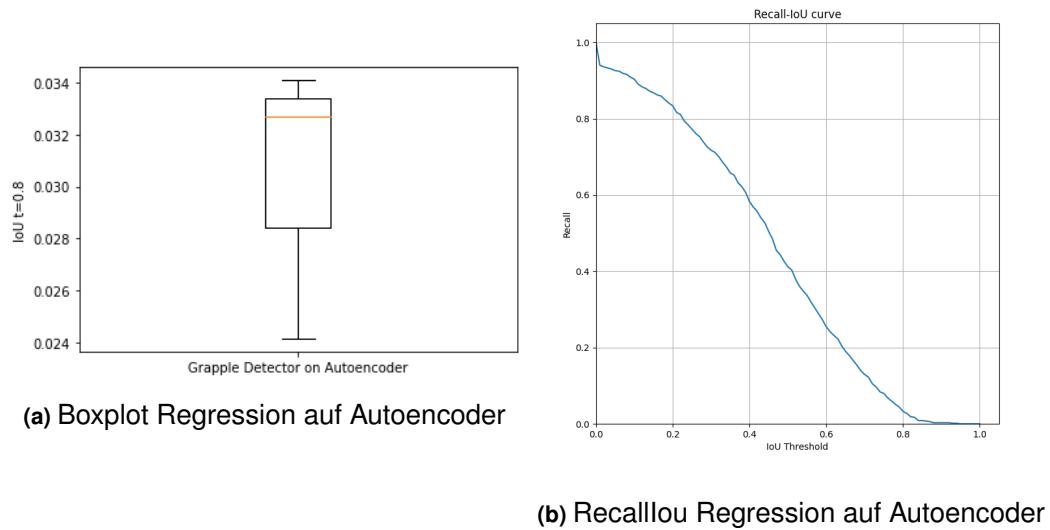


Abbildung 3.1: Ergebniss Regression auf Autoencoder

bearbeiten der Aufgaben Repräsentation finden und Greifererkennung die schwäche des fehlenden Fokus kompensieren.

#### 3.2.1. Werkzeug

Zur Umsetzung der SIMO Multi-Task-Idee wurde ein Modul in Python erstellt. Das Werkzeug erweitert ...

#### 3.2.2. Ergebnis

In Abbildung ?? bla blub

In der Repräsentation werden die Merkmale des Greifers deutlich abgebildet. .... Vergleicht man die Rekonstruktionen des ersten Ansatzes mit denen des TaskFocusin-gAE lässt sich eine Fokussierung auf den Greifer erkennen. Besonders in Bild ?? zu sehen.

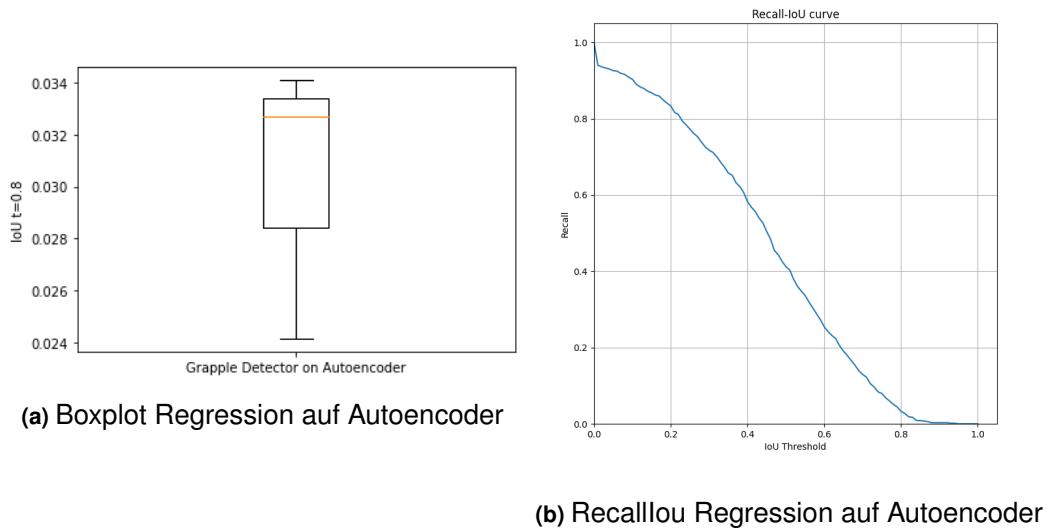


Abbildung 3.2: Embedding

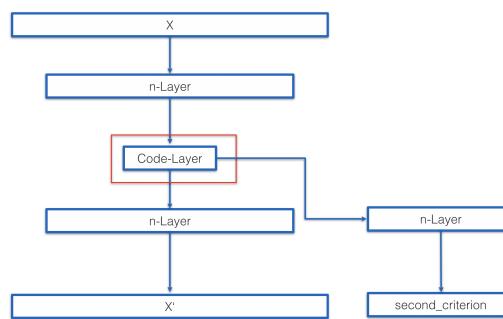


Abbildung 3.3: Schema SecondCriterionAutoencoder

### 3.3. ConvolutionalSecondCriterionAutoencoder

Der SCAE erweitert einen ConvolutionalAutoencoder um ein weiteres Kriterium. Es gibt also zusätzlich zu der Rekonstruktion des Autoencoders einen weiteren Ausgang. Der zweite Ausgang kann wie jeder Ausgang für eine Binärklassifikation, für eine Multiklassifikation, für eine Regression oder jede andere beliebige Aufgabe genutzt werden. In Abbildung 3.4 ist der schematische Aufbau des SCAE abgebildet. Die Schichten des zweiten Kriteriums werden an die Code-Schicht des NN angehängt. Es können beliebig viele Schichten genutzt werden. Die Verlustfunktion des NN besteht aus der Summe der einzelnen Verlustfunktionen und einer Gewichtung. Sie lautet im Detail:

$$loss = weight1 * loss\_autoencoder + weight2 * loss\_secondcriterion \quad (3.1)$$

### 3. Experimente und Werkzeuge

Die Gewichtung der Verlustfunktionen kann dem SCAE per Konstruktor-Argument übergeben werden. Das Werkzeug ist als Python-Module implementiert. Dabei implementiert die Klasse ConvolutionalSecondCriterionAutoencoder den ConvolutionalAutoencoder aus Psipy. In Abbildung 3.5 ist das Klassendiagramm des SCAE dargestellt. Über den Konstruktor können alle Argumente welche zum Erstellen des

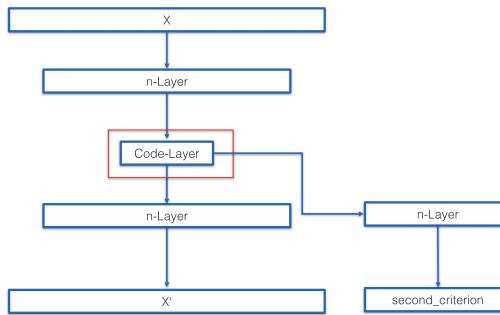


Abbildung 3.4: Schema SecondCriterionAutoencoder

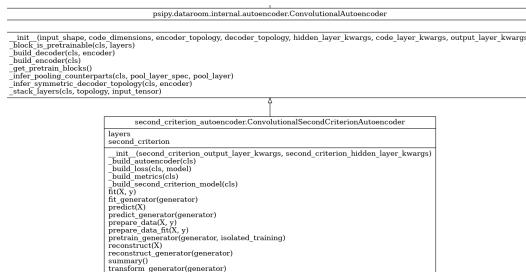


Abbildung 3.5: Klassendiagramm ConvolutionalSecondCriterionAutoencoder

Models notwendig sind per doppeltes Sternchen Wörterbuch Argument (\*\*kwargs) an die Klasse übergeben werden. Diese Technik erlaubt es eine mit Schlüsselwörtern versehene Argumentliste variabler Länge zu übergeben. Die Argumentlisten werden beinahe in allen Methode zum Einsatz gebracht. Sie werden insbesondere genutzt, um Argumente an die zugehörigen Keras-Methoden zu übergeben. Die Namensgebung der Methode orientiert sich dabei an Keras. So wird z. B. in dem Methodenaufruf *fit(..)* unter anderem auch die Keras-Methode *fit(..)* aufgerufen. Um einen SCAE zu trainieren, ist es notwendig eine Instanz zu erzeugen, die Methode *pretrain(..)* aufzurufen und ihn anschließend mit der Methode *fit(..)* zu trainieren. In dem Methodenaufruf *pretrain(..)* wird das Modell erstellt und schichtenweise vortrainiert. Das eigentliche Training erfolgt in der Methode *fit(..)*. Alternativ können auch die zugehörigen Generatorenklassen aufgerufen werden.

In Listing 3.1 ist beispielhaft dargestellt, wie ein SCAE erstellt wird. In den ersten 10 Zeilen wird die Architektur erstellt. Ab Zeile 12 wird eine Instanz eines CSCA mittels Argumentenliste erstellt. Zu beachten ist, dass hier keine Decoder-Architektur übergeben wird. Wenn keine Decoder-Architektur bereitgestellt wird sie beim Erstellen des eigentlichen Modells aus der Encoder-Architektur abgeleitet.

```

1  encoder_topology = [("Conv2D", {"filters": 8, "kernel_size": (3, 3)}),
2  ("Conv2D", {"filters": 8, "kernel_size": (3, 3)}),
3  ('MaxPooling2D', {"pool_size": (2, 2)}),
4  ("Conv2D", {"filters": 16, "kernel_size": (3, 3)}),
5  ("MaxPooling2D", {"pool_size": (2, 2)}),
6  ("Conv2D", {"filters": 16, "kernel_size": (3, 3)}),
7  ("Flatten", {}),
8  ("Dense", {"units": 16})]

9

10 second_criterion_topology = [{"Dense", {"units": num_classes}}]

11

12 cscsa = ConvolutionalSecondCriterionAutoencoder(
13     input_shape=(28, 28, 1),
14     code_dimensions=3,
15     encoder_topology=encoder_topology,
16     second_criterion_topology=second_criterion_topology,
17     hidden_layer_kwargs = {'activation': 'relu'},
18     output_layer_kwargs = {'activation': 'sigmoid'},
19     second_criterion_hidden_layer_kwargs = {'activation': 'relu'},
20     second_criterion_output_layer_kwargs = {'activation': 'softmax'},
21     second_criterion_loss = 'categorical_crossentropy',
22     loss_weights=[8., 1.],
23     second_criterion_metrics = {'second_criterion': 'accuracy'},
24     code_layer_kwargs=dict())

```

**Listing 3.1:** Beispiel Erstellung ConvolutionalSecondCriterionAutoencoder in Python

Listing 3.2 zeigt den Aufruf der Methode Pretrain. Der Aufruf führt zu einem Schichtenweise-Trainieren des Netzwerkes mit den Daten x\_train bei 20 Epochen und einer Stapelgröße von 64.

```
1 cscsa.pretrain(x_train, epochs = 20, batch_size = 64)
```

**Listing 3.2:** Beispieldaufruf Pretrain in Python

Der Methodenaufruf *fit(..)* funktioniert wie der *fit(..)*-Aufruf in Keras. In Zeile drei des Listing 3.2 ist zu erkennen, dass die Zielgrößen der verschiedenen Ausgänge einfach als Python-Wörterbuch übergeben werden können.

```

1 history = cscsa.fit(
2     x_train,
3     {"decoder": x_train, "second_criterion": y_train},
4     epochs=200,
5     batch_size = 64,
6     validation_data=(x_test, {"decoder": x_test, "second_criterion": y_test}))

```

**Listing 3.3:** Beispieldaufruf Fit in Python

### 3.4. TransferSecondCriterionAutoencoder

Ein TransferSecondCriterionAutoencoder basiert auf einem SCAE. Das zweite Kriterium wird durch ein neues Kriterium ersetzt. Zum Beispiel kann ein SCAE eine Objektkennung durchführen. Bei einem TransferSecondCriterionAutoencoder wird die Objektkennung durch eine Klassifizierungsaufgabe ersetzt. Die Architektur und die Gewichte des Autoencoders werden weiterverwendet. Abbildung 3.6 zeigt den Aufbau des TSCAE. Im Klassendiagramm 3.7 für den TSCAE ist zu sehen. Das

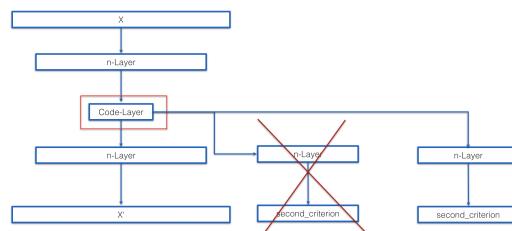


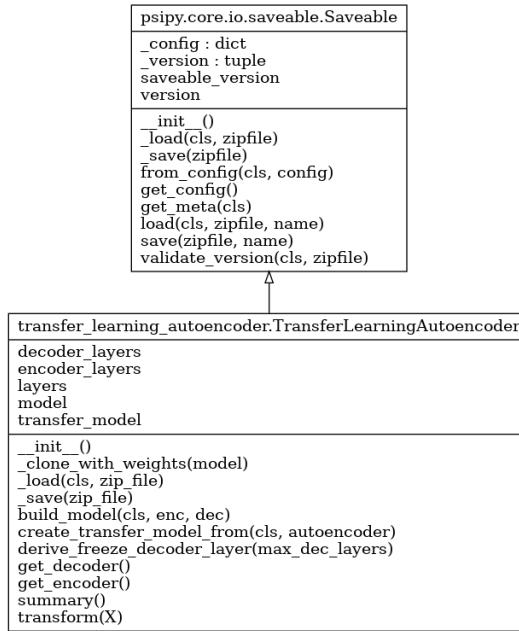
Abbildung 3.6: Schema TransferSecondCriterionAutoencoder

Besondere ist, dass ein SCAE als Konstruktorargument übergeben wird. Die Einstellungen für den ConvolutionalAutoencoder werden aus diesem Model kopiert. Zusätzlich müssen nur noch die Einstellungen für das zweite Kriterium übergeben werden. Aus diesen wird das Model für das zweite Kriterium erstellt. Neu hinzugekommen ist, ein Parameter(freeze\_encoder\_layers) über den eingestellt werden kann, ob und welche Schichten nicht neu trainiert werden können. Es können direkt Schichten ausgewählt werden oder es kann über einen Ganzahlenwert beginnend über die erste Schicht Schichten eingefroren. Werden nur Werte für den Encoder übergeben, werden Werte für den Decoder (freeze\_decoder\_layers) daraus abgeleitet. Listing 3.4 zeigt beispielhaft die Anwendung dieses Werkzeuges. Im Vergleich zu dem SCAE ist die Anwendung schon deutlich einfacher. Es gibt weniger Hyperparameter zum Beachten. Da das Modell auf einem trainierten SCAE basiert ist kein *pretrain(..)* mehr notwendig. Die Methode *fit(..)* wird auf dieselbe Weise wie bei SCAE angewendet.

```

1 tscm = TransferLearningConvolutionalSecondCriterionAutoencoder(csc_autoencoder,
2 second_criterion_topology=second_criterion_topology,
3 second_criterion_loss = 'binary_crossentropy',
4 second_criterion_hidden_layer_kwargs = {'activation': 'relu'},
5 second_criterion_output_layer_kwargs = {'activation': 'sigmoid'},
6 loss_weights=[1, 0.01],
7 freeze_encoder_layers = 2
8 ,freeze_decoder_layers =[0,1])
9
10 history = tscm.fit(

```



**Abbildung 3.7:** Klassendiagramm TransferSecondCriterionAutoencoder

```

11     x_train,
12     {"decoder": x_train, "second_criterion": y_train},
13     epochs=1,
14     batch_size = 128,
15     validation_data=(x_test,{ "decoder": x_test, "second_criterion": y_test}))
16   )
  
```

**Listing 3.4:** Beispiel TransferSecondCriterionAutoencoder in Python

### 3.5. AutoTransferSecondCriterionAutoencoder

Der AutoTransferSecondCriterionAutoencoder ist ein TransferSecondCriterionAutoencoder welcher mithilfe von HpBandSter AutoML zur Hyperparameteroptimierung einsetzt. Konkret erbt die Klasse von hpbandster.core.worker. Zur Speicherung und Verwaltung der Hyperparameter wird auf die Klasse HyperparameterMixin zurückgegriffen. Bei der Instanziierung der Klasse werden sinnvolle Standardwerte für Hyperparameter gesetzt. Die Werte können aber später noch angepasst werden. In Abbildung 3.8 ist das zugehörige Klassendiagramm abgebildet. In Listing 3.5 ist eine einfache Implementierung eines AutoTransferSecondCriterionAutoencoder dargestellt. Der Konstruktor unterscheidet sich nur an einer Stelle zum Konstruktor des TransferSecondCriterionAutoencoder. Es wird keine Instanz des SCAE übergeben, sondern ein Pfad zu einem abgespeicherten Modell eines SCAE. Im Ge-

### 3. Experimente und Werkzeuge



**Abbildung 3.8:** Klassendiagramm AutoTransferSecondCriterionAutoencoder

gensatz zur bisherigen Vorgehensweise werden die Trainings und Testdaten mit der Methode `set_generators(..)` oder `set_numpydata(..)` übergeben. Ziel ist es dabei die Komplexität der Methode `optimize(..)` zu reduzieren. Durch den Aufruf von `optimize(..)` wird der Optimierungsvorgang gestartet. Die Parameter sind dabei die Anzahl an Iterationen, die Optimierungsstrategie und ein Wörterbuch, welches zusätzliche Einstellungen für die einzelnen Optimierer enthalten kann. In jeder Iteration der Optimierung wird ein neuer TransferSecondCriterionAutoencoder erstellt und mittels der ausgewählten Hyperparameter und übergebenen Daten trainiert.

```

1 tscm = AutoTransferConvolutionalSecondCriterionAutoencoder(max_deep_freeze=2,
2     path_to_model = path_to_base_model,
3     second_criterion_topology=second_criterion_topology,
4     second_criterion_loss = 'categorical_crossentropy',
5     second_criterion_hidden_layer_kwarg = {'activation': 'relu'},
6     second_criterion_output_layer_kwarg = {'activation': 'softmax'},
7     second_criterion_metrics = {'second_criterion': 'accuracy'}
8 )
9
10 tscm.set_generators(train_datagenerator, test_datagenerator)
11
12
13 best_config, history = tscm.optimize(3
14     , 'RandomSearch'
15     , optimization_kwarg = optimization_kwarg)

```

**Listing 3.5:** Beispiel AutoTransferSecondCriterionAutoencoder in Python

Das Werkzeug unterstützt derzeit die Optimierer, Randomsearch, Hyperband und BOHB. In Tabelle 3.1 sind die Hyperparameter und ihre mögliche Werte dargestellt. Der Wertebereich für die Stapelgröße wurde aus der Autocrane-Problemstellung abgeleitet. Sie kann maximal, so groß werden, dass kein Speicherplatz-Fehler auf-

<b>Hyperparameter</b>	<b>Werte</b>	<b>Datentyp</b>
<b>Optimierer</b>	Adam, sgd, rmsprop	Kategorial
<b>Batch_size</b>	32 - 1024	Ganzzahl
<b>Epochen</b>	10 - 10.000	Ganzzahl
<b>autoencoder_loss_weight</b>	0.01 - 1	Fließkommazahl
<b>second_criterion_loss_weight</b>	0.01 - 1	Fließkommazahl
<b>freeze_encoder_layers</b>	0 - Parameter	Ganzzahl
<b>freeze_decoder_layers</b>	0 - Parameter	Ganzzahl

**Tabelle 3.1.:** Standard Hyperparameter für AutoML-Suche

treten kann. Die maximale Anzahl der Epochen wurde bewusst groß gewählt. Es sollen weitere Problemstellungen ohne viel Konfigurationsaufwand gelöst werden können. Absurd lange Laufzeiten können durch ein EarlyStopping Kriterium verhindert werden. Die Maximalanzahl der möglicherweise einzufrierenden Schichten wird über den Anwender des Werkzeuges definiert, sie sind anwendungsspezifisch.

weight durch Verwaltung von w1/w2 anpassen.

Worker Budget berücksichtigen / notieren



## 4. Transfer Lernen

In Kapitel 3 wurde gezeigt, dass ein Autoencoder auf das Merkmal Greifer fokussiert werden kann. In diesem Kapitel wird ausgehend von der Repräsentation die Frage beantwortet ob die Erkenntnisse auf andere Aufgaben transferiert werden können

### 4.1. Transfer Experiment

Um einen Transfer auf neue Aufgaben durchführen zu können wurde ein neues Werkzeug TransferTaskFocusAE erstellt. Dieses Werkzeug basiert auf dem TaskFocusAE und ersetzt den fokussierenden Task.

#### 4.1.1. Werkzeug

Konkret wird der 2 Task ersetzt die Architektur und gelernten gewichte bleiben gleich . ....

TL einfügen

#### 4.1.2. Ergebnis

In Abbildung ist das Ergebnis eines TransferTFAE zu sehen, es wird beinahe der selbe Score erreicht.

### 4.2. Auto Transfer Experiment

Ein Hyperparameter des TransferTFAE ist die Gewichtung der Verlustfunktion. Es soll herausgefunden werden, welchen Einfluss der Hyperparameter auf die Ergebnisse hat.

## 4. Transfer Lernen

---

nisse hat. Da es sehr viele Möglichkeiten für die Gewichtung der Verlustfunktion gibt wird hierzu ein neues Modul erstellt. Dieses Modul greift auf den AutoML-Ansatz der automatischen Hyperparameteroptimierung zu.

### 4.2.1. Werkzeug

### 4.2.2. Ergebnis

Die Gewichtung der beiden Teile der Verlustfunktion hat einen starken Einfluss auf die Modellqualität. In Abbildung x,y,z ist zu sehen, dass Bei einer Gewichtung von abc das Beste Ergebnis erreicht wird. Besonders interessant ist, dass es (Kurve erläutern.)

## 4.3. Transfer-Experiment

In den vorangegangen Experimenten wurde gezeigt, dass ein Transfer möglich ist und gute Ergebnisse erzielt. In diesem Experiment soll herausgefunden werden, in welchem Maß der Transfer eine Steigerung der Leistung erbringt. Hierzu werden die genutzten Datenmengen auf 200, 2000 und 9749 annotierte Bilder begrenzt.

In Abbildung sind die Ergebnisse mit verschiedenen Datenmengen dargestellt. Es sticht hervor...

## **5. Fazit**

### **5.1. Zusammenfassung**

### **5.2. Kritische Reflexion**

### **5.3. Ausblick und weitere Arbeiten**

#### **5.3.1. Transfer auf Greiferdatensatz**

Die Annotationskosten sind je nach Art von Annotation und Aufwand unterschiedlich. Eine einfache Zuordnung eines Bildes zu einer von zwei Klassen liegt dabei im einstelligen Cent-Bereich. Die Markierung eines Objektes in einem Bild mittels Rahmen verursacht Kosten im zweistelligen Cent-Bereich und die Markierung von Winkeln in einem Bild kann mehr als einen Euro kosten.

#### **5.3.2. Autocrane-Datensatz**

Im Rahmen der Arbeit wurden ~12.000 Bilder mit der Annotation "befinden sich Baumstämme im Greifer oder nicht?" und 4.500 Bilder mit der Annotation "Rahmen um den Greifer" genutzt. Diese Beiden Datensätze sind unter der Adresse [unter einer Adresse-Datensatz](#) veröffentlicht und können entsprechend der Lizenz genutzt werden.

#### **5.3.3. Mehrfache Aufgaben**

#### **5.3.4. Flexibilität der Werkzeuge**





# Abkürzungsverzeichnis

<b>AutoML</b>	automatisiertes maschinelles Lernen .....	11
<b>NAS</b>	Neural Architecture Search .....	11
<b>HPO</b>	Hyperparameter-Optimierung .....	11
<b>CAE</b>	Convolutional Autoencoder .....	8
<b>MTL</b>	Multi-Task-Lernen .....	10
<b>SIMO</b>	Single-Input Multi-Output .....	21



# **Tabellenverzeichnis**

2.1. Datenaufteilung - Train Test Validation . . . . .	20
3.1. Standard Hyperparameter für AutoML-Suche . . . . .	31



# Abbildungsverzeichnis

2.1.	Schema Autoencoder . . . . .	7
2.2.	Multi-Task-Lernen: Ausprägungen . . . . .	11
2.3.	Klassendiagramm ConvolutionalAutoencoder . . . . .	16
2.4.	Klassendiagramm Hyperparametermixin . . . . .	16
2.5.	Rundlaufkran . . . . .	17
2.6.	Greifer . . . . .	18
2.7.	Bildqualität . . . . .	19
3.1.	Ergebniss Regression auf Autoencoder . . . . .	24
3.2.	Embedding . . . . .	25
3.3.	Schema SecondCriterionAutoenocder . . . . .	25
3.4.	Schema SecondCriterionAutoenocder . . . . .	26
3.5.	Klassendiagramm ConvolutionalSecondCriterionAutoencoder . . . . .	26
3.6.	Schema TransferSecondCriterionAutoenocder . . . . .	28
3.7.	Klassendiagramm TransferSecondCriterionAutoenocder . . . . .	29
3.8.	Klassendiagramm AutoTransferSecondCriterionAutoenocder . . . . .	30
A.1.	RecallIoUGrappleBaseline . . . . .	xiii
B.1.	Baumstammklassifikation Basislinie ROC . . . . .	xv
B.2.	Konfusionsmatrix Baumstammklassifikation Basislinie . . . . .	xv
B.3.	Klassifikationsreport Baumstammklassifikation Basislinie . . . . .	xv



# Listings

2.1.	Erweiterung zum Speichern eines Tensorflow Models . . . . .	15
2.2.	Aufbereitung Generatorergebnis in Python . . . . .	21
3.1.	Beispiel Erstellung ConvolutionalSecondCriterionAutoencoder in Python . . . . .	27
3.2.	Beispieldruck Pretrain in Python . . . . .	27
3.3.	Beispieldruck Fit in Python . . . . .	27
3.4.	Beispiel TransferSecondCriterionAutoencoder in Python . . . . .	28
3.5.	Beispiel AutoTransferSecondCriterionAutoencoder in Python . . . . .	30



# Literatur

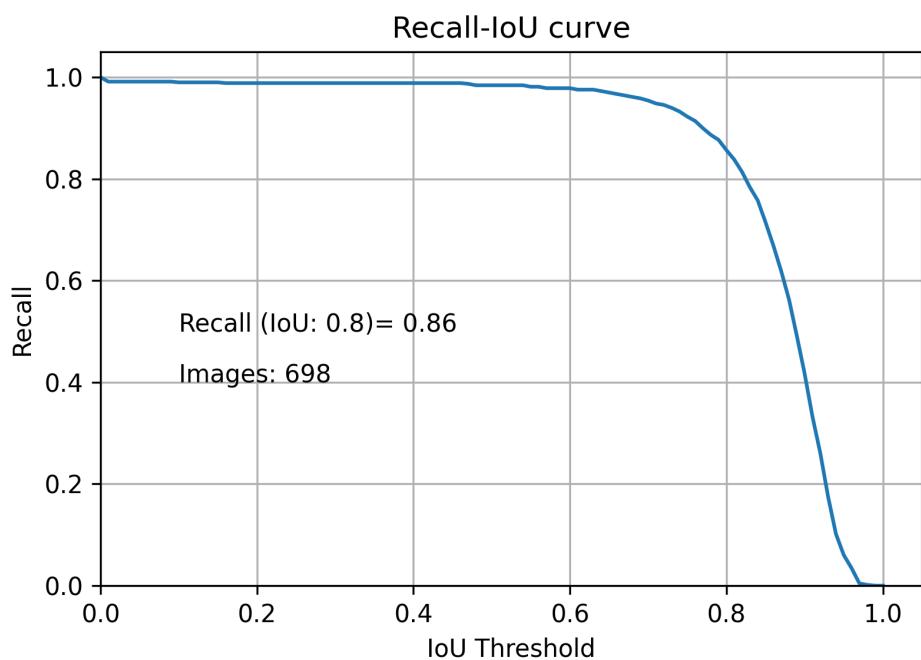
- [Be07] Bengio, Y.; Lamblin, P.; Popovici, D.; Larochelle, H.; Montreal, U.: Greedy layer-wise training of deep networks. In: Bd. 19, 2007.
- [Be12] Bergstra, James, and Yoshua Bengio.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13/, S. 281–305, 2012, URL: <http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf>.
- [Ch15] Chollet, F. et al.: Keras, 2015, URL: <https://keras.io/>.
- [CSZ10] Chapelle, O.; Schlkopf, B.; Zien, A.: Semi-Supervised Learning. The MIT Press, 2010, ISBN: 0262514125.
- [DJ87] D. E. Rumelhart; J. L. McClelland: Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations. S. 318–362, 1987.
- [EMH19] Elsken, T.; Metzen, J. H.; Hutter, F.: Neural Architecture Search: 3. In (Hutter, F.; Kotthoff, L.; Vanschoren, J., Hrsg.): Automatic Machine Learning: Methods, Systems, Challenges. Springer, S. 69–86, 2019.
- [FH19] Feurer, M.; Hutter, F.: Hyperparameter Optimization: 1. In (Hutter, F.; Kotthoff, L.; Vanschoren, J., Hrsg.): Automatic Machine Learning: Methods, Systems, Challenges. Springer, S. 3–38, 2019.
- [Fr] Frazier, P. I.: A Tutorial on Bayesian Optimization, URL: <https://arxiv.org/pdf/1807.02811>.
- [Fu] Fuzhen Zhuang; Zhiyuan Qi; Keyu Duan; Dongbo Xi; Yongchun Zhu; Hengshu Zhu; Hui Xiong; Qing He: A Comprehensive Survey on Transfer Learning, URL: <https://arxiv.org/abs/1911.02685v2>.
- [HKV19] Hutter, F.; Kotthoff, L.; Vanschoren, J., Hrsg.: Automatic Machine Learning: Methods, Systems, Challenges. Springer, 2019.

- [HS06] Hinton, G. E.; Salakhutdinov, R. R.: Reducing the Dimensionality of Data with Neural Networks. *Science* (New York, N.Y.) 313/, S. 504–507, 2006.
- [Hu07] Hunter: Matplotlib: A 2D graphics environment, 2007.
- [Ia14] Ian J. Goodfellow; Jean Pouget-Abadie; Mehdi Mirza; Bing Xu; David Warde-Farley; Sherjil Ozair; Aaron Courville; Yoshua Bengio: Generative Adversarial Networks, 2014.
- [Jo18] Joaquin Vanschoren: Meta-Learning: A Survey. CoRR abs/1810.03548/, 2018.
- [JT] Jamieson, K.; Talwalkar, A.: Non-stochastic Best Arm Identification and Hyperparameter Optimization, URL: <https://arxiv.org/pdf/1502.07943>.
- [KJ95] Kohavi, R.; John, G. H.: Automatic Parameter Selection by Minimizing Estimated Error. In: In Proceedings of the Twelfth International Conference on Machine Learning. Morgan Kaufmann, S. 304–312, 1995.
- [KW19] Kingma, D. P.; Welling, M.: An Introduction to Variational Autoencoders. Foundations and Trends® in Machine Learning 12/4, S. 307–392, 2019, ISSN: 1935-8245, URL: <http://dx.doi.org/10.1561/2200000056>.
- [Le99] LeCun, Y.; Haffner, P.; Bottou, L.; Bengio, Y.: Object Recognition with Gradient-Based Learning. In: Shape, Contour and Grouping in Computer Vision. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 319–345, 1999, ISBN: 978-3-540-46805-9.
- [Li] Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Marben, J.; Müller, P.; Hutter, F.: BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters, URL: <https://arxiv.org/pdf/1908.06756>.
- [Li16] Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; Berg, A. C.: SSD: Single Shot MultiBox Detector, 2016, URL: <https://arxiv.org/pdf/1512.02325>.
- [Li17] Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; Talwalkar, A.: HYPERBAND: BANDIT-BASED CONFIGURATION EVALUATION FOR HYPERPARAMETER OPTIMIZATION. ICLR/, 2017, URL: <https://openreview.net/pdf?id=ry18Ww5ee>.

- [Ma11] Masci, J.; Meier, U.; Cireşan, D.; Schmidhuber, J.: Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In (Honkela, T.; Duch, W.; Girolami, M.; Kaski, S., Hrsg.): Artificial Neural Networks and Machine Learning – ICANN 2011. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 52–59, 2011, ISBN: 978-3-642-21735-7.
- [Ma15] Martin Abadi; Ashish Agarwal; Paul Barham; Eugene Brevdo; Zhifeng Chen; Craig Citro; Corrado, G. S.; Andy Davis; Jeffrey Dean; Matthieu Devin; Sanjay Ghemawat; Ian Goodfellow; Andrew Harp; Geoffrey Irving; Michael Isard; Jia, Y.; Rafal Jozefowicz; Lukasz Kaiser; Manjunath Kudlur; Josh Levenberg; Dandelion Mané; Rajat Monga; Sherry Moore; Derek Murray; Chris Olah; Mike Schuster; Jonathon Shlens; Benoit Steiner; Ilya Sutskever; Kunal Talwar; Paul Tucker; Vincent Vanhoucke; Vijay Vasudevan; Fernanda Viégas; Oriol Vinyals; Pete Warden; Martin Wattenberg; Martin Wicke; Yuan Yu; Xiaoqiang Zheng: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015, URL: <https://www.tensorflow.org/>.
- [Mi18] Michelucci, U.: Hyperparameter Tuning. In: Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks. Apress, Berkeley, CA, S. 271–322, 2018, ISBN: 978-1-4842-3790-8.
- [Mi20] Microsoft: Microsoft Azure, 2020, URL: <https://azure.microsoft.com/de-de/>.
- [Nv20] Nvidia: Tesla K80, 2020, URL: <https://www.nvidia.com/de-de/data-center/tesla-k80/>.
- [Ol06] Oliphant, T.: NumPy: A guide to NumPy, hrsg. von Trelgol Publishing USA, 2006, URL: <http://www.numpy.org/>.
- [Pe11] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12/, S. 2825–2830, 2011.
- [PQ10] Pan, S. J.; Qiang, Y.: A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering 22/10, S. 1345–1359, 2010.
- [Pr] Project Jupyter: jupyter, URL: <https://jupyter.org/>.
- [PS19] PSIORI GmbH: Autocrane wiki, hrsg. von PSIORI GmbH, 2019, URL: <https://psiori.atlassian.net/wiki/spaces/2A/overview>.

- [PS20] PSIORI GmbH: PSIORI, 2020, URL: <https://www.psiori.com/de>.
- [Py20] Python Software Foundation: The Python Language Reference, 2020, URL: <https://docs.python.org/3.7/reference/>.
- [SAF18] Stefan Falkner; Aaron Klein; Frank Hutter: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. arXiv:1807.01774/, 2018, URL: <https://openreview.net/pdf?id=ry18Ww5ee>.
- [Sh00] Shearer, C.: The CRISP-DM Model: The New Blueprint for Data Mining. Journal of Data Warehousing 5/4, 2000.
- [Ti18] Tirumala, S. S.: A Deep Autoencoder-Based Knowledge Transfer Approach. In (Chaki, N.; Cortesi, A.; Devarakonda, N., Hrsg.): Proceedings of International Conference on Computational Intelligence and Data Engineering. Springer Singapore, Singapore, S. 277–284, 2018, ISBN: 978-981-10-6319-0.
- [TW18] Thung, K.; Wee, C.-Y.: A brief review on multi-task learning. Multimedia Tools and Applications 77/, 2018.
- [Vi08] Vincent, P.; Larochelle, H.; Bengio, Y.; Manzagol, P.-A.: Extracting and composing robust features with denoising autoencoders. In. S. 1096–1103, 2008.

## A. Anhang Basislinie Greifer



**Abbildung A.1:** Basislinie Greifererkennung



## B. Anhang Basislinie Baumstämme

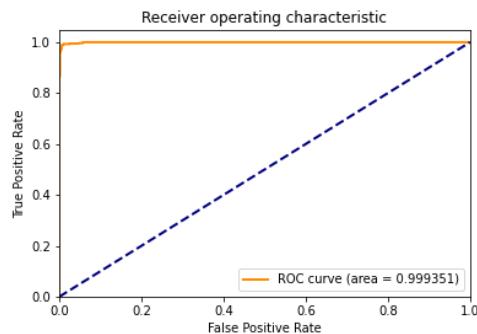


Abbildung B.1: receiver operating characteristic

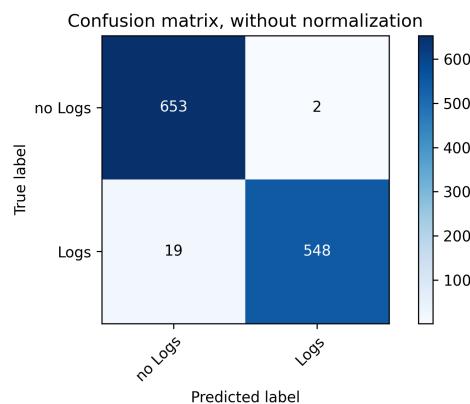


Abbildung B.2: Konfusionsmatrix

	precision	recall	f1-score	support
no Logs	0.97	1.00	0.98	655
Logs	1.00	0.97	0.98	567
accuracy			0.98	1222
macro avg	0.98	0.98	0.98	1222
weighted avg	0.98	0.98	0.98	1222

Abbildung B.3: Klassifikationsreport