

## **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 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, Waldkirch: Hochschule für Technik, Wirtschaft und Medien Offenburg,  
2020. 45 Seiten.

**Hoch, Sebastian:**

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

MASTER THESIS, Waldkirch: Offenburg University, 2020. 45 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, Ansätze zu finden, welche Aufgaben Datensparsam lösen können. Von 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 Fachkenntnisse 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.

Waldkirch, 30. Juni 2020

Sebastian Hoch

## **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.

Waldkirch, 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 Repräsentationen, datensparsame Lösungen für verschiedene Aufgaben finden soll. Durch einen Multi-Task-Ansatz trägt das Finden einer Repräsentation gleichzeitig zum Lösen einer Aufgabe bei. Durch Ersetzung einer der Aufgaben können Wissentransfers datensparsam auf neue Aufgabe durchgeführt werden. In der erarbeiteten und evaluierten Lösung können Parameter automatisch gefunden werden. Bei einem Vergleich von verschiedenen Ansätzen und einem Vergleich mit verschiedenen Datenmengen 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 dieser Arbeit beschrieben. Abgeschlossen wird die Arbeit mit einem Ausblick auf Verbesserungen und Potenziale der Ansätze.

## Abstract

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

High costs when annotating data leads to the search for ways to get along with a reduced amount of data when creating models. This work explores a possible approach to find solutions with less data, by making use of focused representations. A multi task approach allows found representations to simultaneously help finding solutions to a task. By replacing one of the tasks, a knowledge transfer to another task, which makes efficient use of data, can be made. In this developed and evaluated approach, parameters can be found automatically. When making a comparison between different approaches and with different amounts of data, taking into account the networks score, it can be seen, that the new approach produces more optimal results – especially when working with a reduced amount of data. The high score of the approach was motivation to provide it as python modules. These modules will be described as part of this work. Finally a future prospect with possible vectors for improvement will be made.

# 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 . . . . .	9
2.2.2. Halbüberwachtes Lernen . . . . .	9
2.2.3. Multi-Task-Lernen . . . . .	10
2.3. Automatisiertes maschinelles Lernen . . . . .	12
2.3.1. Hyperparameter-Optimierung . . . . .	12
2.3.2. Meta-Learning . . . . .	13
2.3.3. Neural Architecture Search . . . . .	13
2.3.4. Optimierungstechniken . . . . .	13
2.4. Bibliotheken und Werkzeuge . . . . .	15
2.5. Einordnung und bestehende Systeme . . . . .	17
2.6. Datenverständnis . . . . .	20
2.7. Datenvorbereitung . . . . .	21
<b>3. Experimente und Werkzeuge</b>	<b>25</b>
3.1. Greifererkennung auf Repräsentation . . . . .	25
3.2. Aufgabenfokussierung und Greifererkennung . . . . .	27
3.2.1. Werkzeug: TaskFocusingOnAutoencoder . . . . .	27
3.2.2. Experiment . . . . .	30
3.3. Transferlernen und 'Greifer beladen?' -Klassifikation . . . . .	31
3.3.1. Werkzeug: TaskTransferOnAutoencoder . . . . .	31
3.3.2. Experiment . . . . .	33
3.4. AutoML und 'Greifer beladen?' . . . . .	35
3.4.1. Werkzeug: AutoTaskTransferOnAutoencoder . . . . .	35
3.4.2. Experiment . . . . .	37
3.5. Datenmenge und 'Greifer beladen?' . . . . .	37
<b>4. Fazit</b>	<b>41</b>
4.1. Zusammenfassung . . . . .	41
4.2. Kritische Reflexion . . . . .	42

## Inhaltsverzeichnis

---

4.3. Ausblick und weitere Arbeiten . . . . .	42
4.3.1. Transfer auf Greiferdatensatz . . . . .	42
4.3.2. Autocrane-Datensatz . . . . .	44
4.3.3. Erweiterung: Multi-Task-Ansatz . . . . .	44
4.3.4. Flexibilität der Werkzeuge . . . . .	45
<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. Basislinie Greifer</b>	<b>xv</b>
<b>B. Basislinie Baumstämme</b>	<b>xvii</b>
<b>C. Greifererkennung</b>	<b>xix</b>
C.1. Greifererkennung auf Autoencoder . . . . .	xix
C.2. Multi-Task Greifererkennung . . . . .	xix
<b>D. Baumstämme im Greifer</b>	<b>xxiii</b>
<b>E. AutoML</b>	<b>xxv</b>
<b>F. TTAE Greiferdetection</b>	<b>xxxi</b>

Quellen prüfen +  
vollständigen



# **Todo list**

Zusammenfassung auf Englisch . . . . .	iv
Quellen prüfen + Vervollständigen . . . . .	1



# 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 aufgabenspezifische 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 gesenkt werden, wenn weniger annotierte Daten genutzt werden müssen.

Ziel dieser Arbeit ist es, herauszufinden, ob es, von Datenrepräsentationen einer Domäne ausgehend, 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 dahinterstehenden Techniken funktionieren.

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

## 1. Einleitung

---

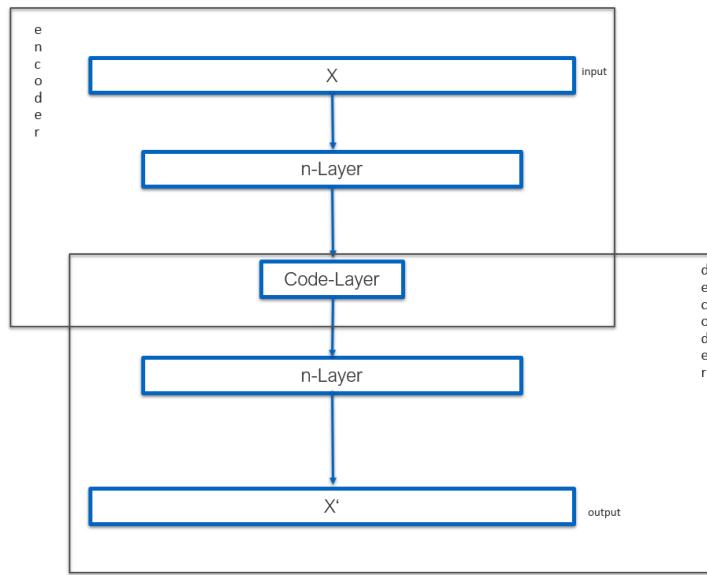
Arbeit vorgestellt werden. Der Hauptteil der Arbeit beinhaltet eine Vorstellung und Evaluierung eines Multi-Task-Ansatzes. Auf diesem Ansatz aufbauend, wird ein Transfer-Learning-Ansatz mit einer Erweiterung des automatischen maschinellen Lernens sowohl vorgestellt als auch einer Evaluierung unterzogen. Abgeschlossen wird die Arbeit mit einer Bewertung der Lösung der Problemstellung und einem umfassenden 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, den Verwandten Arbeiten, gefolgt von einer Beschreibung der bestehenden Systeme, auf welche mit einem Abschnitt über die Daten und ihrer Vorverarbeitung abgeschlossen wird.

### 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 versteckten Schicht und einer Ausgabeschicht. Sie können aber auch mit mehreren Schichten, also mit 'tiefen Architekturen' genutzt werden. [HS06] Der Encoder lässt sich auch vereinfacht als die Funktion  $F(x) = c$  und der Decoder als die Funktion  $F(c) = x'$  darstellen, wobei  $x \neq x'$  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. Contractive Autoencoder [Ri11] nutzen einen Regularisierer in der Zielfunktion, um das Modell zu zwingen eine Funktion zu lernen, die flexibler auf Variationen der Eingabewerte reagiert.



**Abbildung 2.1:** Schema Autoencoder

Für die Arbeit ist der Convolutional Autoencoder [Ma11] kurz CAE<sup>1</sup> interessant. Er ist ein Stacked Autoencoder, welcher Faltungsschichten (Convolutional Layer) integriert. Faltungsschichten kommen von [Le99] und haben sich durchgesetzt [KSH12], [Ch14] und [LBH15].

**Schichtenweise Vortrainieren** Im schichtenweise Vortrainieren [Be07] werden einzelne Schichten eines neuronalen Netzwerkes vor dem eigentlichen Training trainiert, um die Leistung des gesamten Models zu erhöhen. 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.

---

<sup>1</sup>Convolutional Autoencoder

### 2.2.1. Tiefes Transferlernen

Geprägt durch die Entwicklung hin zu tiefen neuronalen Netzwerken wurden Methoden zum Tiefen Transferlernen (deep transfer learning) vorgeschlagen. [Ta18] ordnet diese in vier Kategorien ein. Für eine Einordnung von klassischen Methoden des Transferlernens eignet sich die Erhebung [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 verwendet einen Teil des in der Quelldomäne trainierten Netzwerkes in der Zieldomäne wieder. 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, der zweite Teil ist der sprachabhängige Klassifikator. Dabei kann die sprachunabhängige Merkmalstransformation in der Zieldomäne wiederverwendet werden. Insbesondere in [Ja14], [Lo16], und [GSH18] hat sich dieser Ansatz bewährt.

**Gegnerischbasiert** Gegenerischbasiertes tiefes Transferlernen 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 unüberwachtem und überwachtem Lernen einordnen lässt. Methoden des unüberwachten Lernens arbeiten komplett ohne annotierte Daten, während überwachtes Lernen mit vollständig

annotierten Daten arbeitet. 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 auch nicht beschriftete Daten aus der gleichen Verteilung entnommen werden. 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 parallel gelernt spricht man von dem Multi-Task-Lernen kurz MTL<sup>2</sup>. Der Begriff MTL wurde insbesondere von [Ca98] geprägt. 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 zudem herausgestellt, dass Multi-Task-Lernen zum Lernen von nicht verwandten Aufgaben vorteilhaft ist. Basierend auf den Ein- und Ausgängen wird MTL von [TW18] in drei Fälle unterteilt.

**Single-Input Multi-Output** SIMO wird verwendet, um aus einer Eingabe Vorhersagen 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 Ausgabezieles 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 neuronaler Netze, die auf MTL beruhen.

---

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

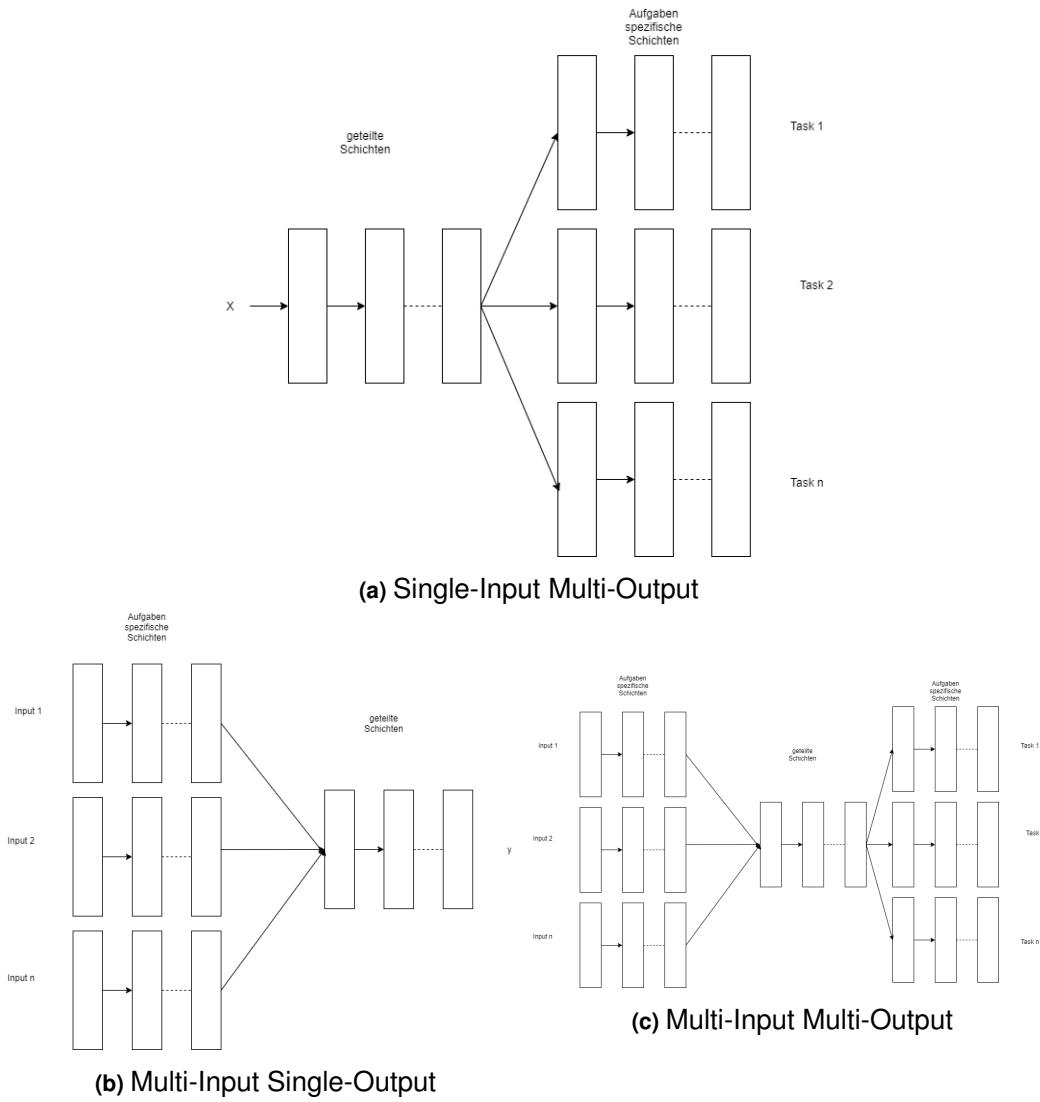


Abbildung 2.2: Multi-Task-Lernen: Ausprägungen

Induktives 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. Induktives Transfer-Lernen kann deshalb als Sonderform des MTL gesehen werden.

Die Kombination von Multi-Task-Learning tiefen Lernen für die Computervision wurde in [Yu17], [Li16a], [RVR16] und [Zh19] eingesetzt.

## 2.3. Automatisiertes maschinelles Lernen

Das automatisierte maschinelle 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 durchzuführen. Die vollständige Automatisierung ist ein langfristiges Ziel. Aktuelle Systeme sind halb-automatisch und zielen 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 Hyperparameterkonfiguration zu finden. Automatische-HPO ist die Technik des automatischen Finden der Hyperparameter, um die Leistung zu optimieren. Dabei hat sie insbesondere drei Ziele: An erster Stelle sollen Personenaufwände bei der Anwendung von maschinellem Lernen reduziert werden, zusätzlich soll es die Leistung von Algorithmen und Modellen des maschinellen Lernens verbessern. Des Weiteren sollen so in der Wissenschaft die Reproduzierbarkeit und Fairness von Studien verbessert werden, da Automatische-HPO einfacher reduzierbar ist, als manuelle-HPO. In Kapitel 2.3.4 werden einige gängige Optimierungstechniken der automatischen-HPO erläutert. [FH19]

---

<sup>3</sup>automatisierte maschinelle Lernen

<sup>4</sup>Neural Architecture Search

<sup>5</sup>Hyperparameter-Optimierung

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

**Zufallssuche** Die Zufallssuche ähnelt der Rastersuche. Sie unterscheidet sich dadurch, dass die Parameterkombinationen nicht mehr definiert werden müssen.

## 2. Grundlagen

---

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 der zur Optimierung von Zielfunktionen dient, 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 eine 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. [Fr18]

**Hyperband** Hyperband [Li17] erweitert Successive Halving [JT15]. Successive Halving kurz SH<sup>6</sup> weist einer Reihe von Hyperparameter-Konfigurationen eine einheitliche Menge an Ressourcen zu, berechnet die Leistung von allen Konfigurationen und entfernt die schlechtere Hälfte. Die Konfigurationen werden dabei zufällig gezogen. Das Ganze 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 Zufallssuche.

**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>

---

<sup>6</sup>Successive Halving

kann eine Implementierung des Werkzeuges gefunden werden. Diese Framework wurde für den praktischen Teil der Arbeit genutzt.

## 2.4. Bibliotheken und Werkzeuge

Für den praktischen Teil der Abschlussarbeit wurde insbesondere Cnvrge [cn20] genutzt. Cnvrge.io ist eine 'full-stack' Data Science Platform, welche Werkzeuge für die Erstellung, Verwaltung, Bereitstellung und Automatisierung von maschinellem Lernen bereitstellt. Cnvrge erlaubt es Arbeitsbereiche mittels Container 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 [Li19] , 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 sowie Blickwinkel geändert werden, Daten können mit zusätzlichen Informationen versehen werden und Zoomen ist möglich. Als zusätzliche Information können z.B. ein Originalbild und seine Rekonstruktion oder die Information, ob eine Vorhersage korrekt oder falsch war 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 Methode *pretrain(..)* und *fit(..)*. Mittels *pretrain(..)* werden die Schichten eines symmetrischen Autoencoder von außen nach innen wie, in [Be07] beschrieben vortrainiert. Die Zuordnung der Schichten erfolgt 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.

**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.

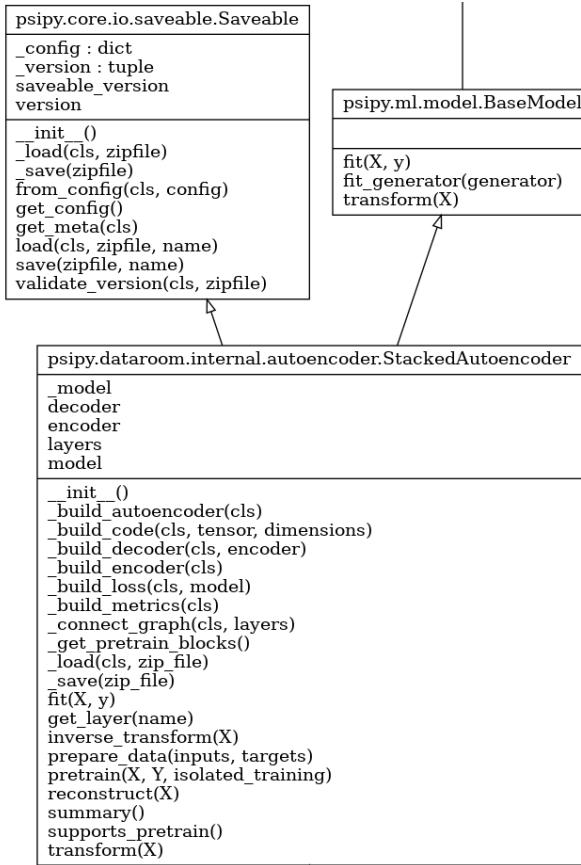


Abbildung 2.3: Klassendiagramm ConvolutionalAutoencoder

## 2.5. Einordnung und bestehende Systeme

Die Bilddaten und Aufgabenstellungen der neuronalen Netzwerke sind in die Problemstellungen des Autocrane-Projekts [PS20] von PSIORI einzuordnen. Es sind 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 mit dem Rundlaufkra, wodurch der Greifer immer im Bild ist. Für das Autocarne-Projekt sind insbesondere drei Anwendungsfälle interessant.

## 2. Grundlagen

---

psipy.ml.automation.hyperparameter_mixin.HyperparameterMixin
SUBSPACE_DELIMITER : str
hyperparameter_space
__init__(self)
get_hyperparameter(name)
get_hyperparameter_names()
get_hyperparameter_space()
get_hyperparameters()
remove_hyperparameter(name)
set_condition(condition)
set_hyperparameter(hyperparameter, subspace)
set_hyperparameter_attribute(hyperparameter_name, attribute_name, value)

Abbildung 2.4: Klassendiagramm Hyperparamettermixin

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 gefüllt werden. Der Fülltrichter muss mit Holz aus einem LKW gefüllt werden. Es ergeben sich 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.



Abbildung 2.5: Rundlaufkran (Foto: ANDRITZ)

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

PSIORI hat die Aufgabe mittels neuronalem Netzwerk gelöst. Dabei wurde auf die Technik des Single Shot MultiBox Detector (SSD<sup>7</sup>) [Li16b] zurückgegriffen. Die Vorhersagegenauigkeit dieses Modells wird als Basislinie und Vergleichswert ge-

---

<sup>7</sup>Single Shot MultiBox Detector



Abbildung 2.6: Greifer

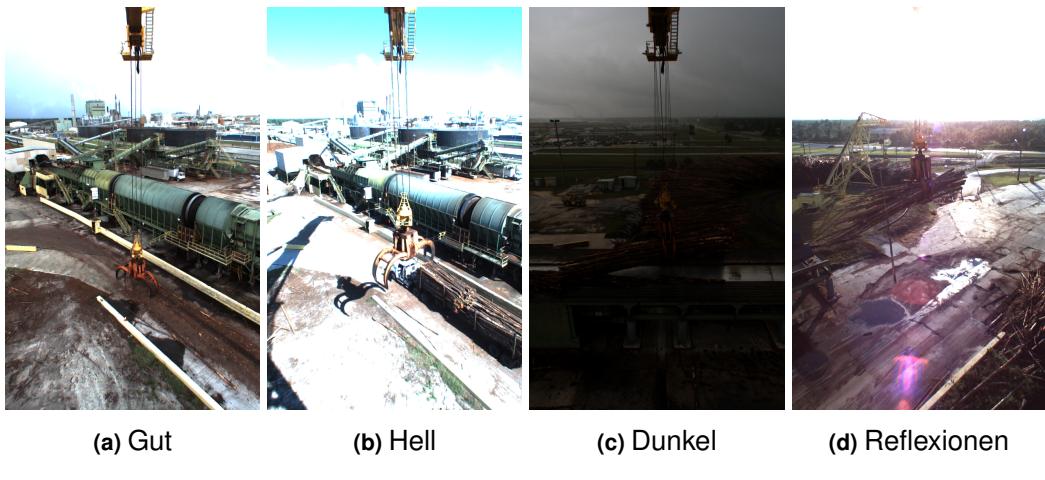
nutzt. Dabei ist die ausschlaggebende Metrik die Intersection over Union kurz IoU<sup>8</sup> mit einem Schwellenwert von 0.8. Es wird also die Fläche der Überschneidung der beiden Rahmen durch die Gesamtfläche der Rahmen geteilt und wenn ein Wert  $\geq 80\%$  erreicht wird, als korrekt vorhergesagt eingestuft. Das Modell erreicht einen Wert von 0.86. In Anhang A ist die vollständige Basislinie dargestellt.

**Greifer beladen?** Die Aufgabe 'Greifer beladen?' hat zum Ziel, zu erkennen ob sich Baumstämme im Greifer befinden oder nicht. Es handelt sich um eine Klassifikationsaufgabe. Für diese Aufgabe wurde von PSIORI ein Modell erstellt. Dieses Modell wird wie das Greifererkennungsmodell 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.

<sup>8</sup>Intersection over Union

## 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 'Greifer beladen' werden Daten mit einer passenden Beschriftung bereitgestellt.



**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.

**Generierter Greiferdatensatz** Erfahrungsgemäß lernen Autoencoder Lichtverhältnisse in Bildern relativ gut. Um für erste Versuche den Fokus von den Lichtver-

hältnissen auf die eigentliche Aufgabenstellung zu setzen wurde ein weiterer Greiferdatensatz erzeugt. Hierfür wurden 8966 Bilder mittels dem Basislinienmodell beschriftet und in 7171 Trainingsdaten und jeweils 896 Test- und Validationsaten aufgeteilt.

**Baumstammdatensatz** Der Baumstammdatensatz 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 versionierbaren 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 Baumstammdaten 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 für die 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

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

## 2. Grundlagen

---

maximalen Anzahl an Bildern, Bilder als NumPyarray zurück. Die Klassen Grapple-DataLoader und LogsDataLoader implementieren für den jeweiligen Datensatz die konkreten Methoden zum Laden der Daten.

Mittels des Moduls '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 können die Imagedatageneratoren die Bilder direkt in der benötigten Größe bereitgestellt werden. Das Modul implementiert jeweils für den Baumstamm Datensatz und den Greiferdatensatz eine Klasse zum Bereitstellen von Imagedatageneratoren.

Da die Modelle SIMO<sup>9</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 definiert. Zum Beispiel kann X ein Bild sein und Y die zugehörige Klasse. Die Multi-Task-Module 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 Image-datagenerator 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.

---

<sup>9</sup>Single-Input Multi-Output

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.



## 3. Experimente und Werkzeuge

Als großer Kostenfaktor in Data-Science-Projekten wurden die 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 (kostensparsam) Aufgaben in dieser Domäne gelöst werden können.

In den nachfolgenden Abschnitten wird zur Verdeutlichung der Ansätze zusätzlich zu den Experimenten ein Blick auf die erstellten Module geworfen.

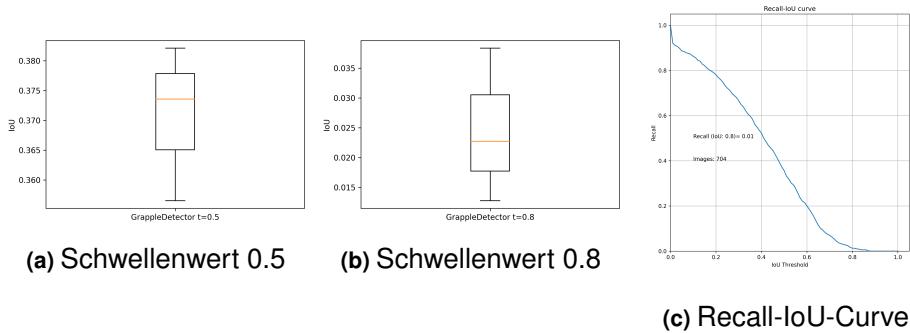
Zur Visualisierung der Einbettungen wird auf das Werkzeug Visualizer (siehe Kapitel 2.4) zurückgegriffen. Um die Daten in dem Werkzeug darzustellen, wird auf die Werte der Einbettung eine Hauptkomponentenanalyse mit 3 Komponenten durchgeführt. Die Einbettungen, in diesem Kapitel, haben die Dimension 10.

### 3.1. Greifererkennung auf Repräsentation

Als erster Ansatz wird versucht, ausgehend von einer Repräsentation, welche mittels Autoencoder erstellt wird, den Greifer zu finden. Hierzu werden die gefundenen Repräsentationen als Eingangswerte für ein neuronales Netzwerk genutzt. Das Netzwerk führt in der Ausgangsschicht eine Regression durch. In Abbildung 3.1 ist das Ergebnis abgebildet. Bei einem Schwellenwert von 0.8 wird eine Punktzahl nahe 0 erreicht. In der Recall-IoU-Kurve ist ein stetiger Abfall der Punktzahl zu sehen. Dieses schlechte Ergebnis zeigt, dass der gewählte Ansatz nicht zielführend zur Lösung des Problems ist. Als Ursache für die schlechte Leistung kann die fehlende Fokussierung der Einbettung erkannt werden. In Abbildung 3.2 ist die Einbettung des zugrundeliegenden Autoencoders abgebildet. In den beiden Abbildungen werden die Datenpunkte farblich markiert. Dabei wird die y-Position des Greifers im Bild, in der einen und die x-Position des Greifers im Bild, in der anderen Abbildung

### 3. Experimente und Werkzeuge

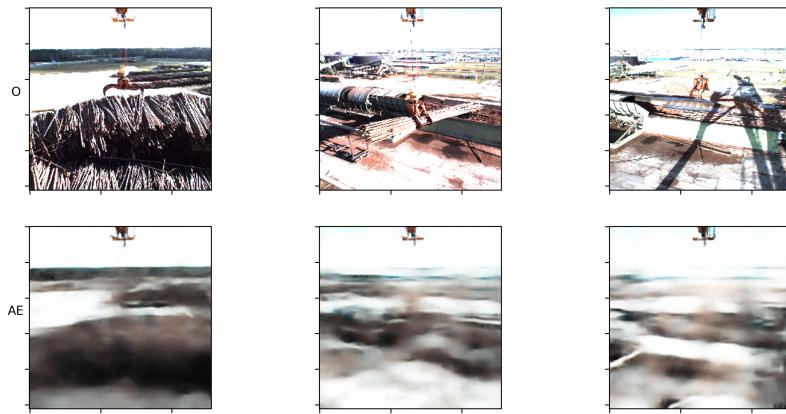
zur Zuordnung genutzt. Die Datenpunkte verteilen sich im Raum, bilden aber keine Merkmale des Greifers ab. In Abbildung 3.3 sind Bilder mit ihren Rekonstruktionen abgebildet. Es ist sehr deutlich zu erkennen, dass der sich stark verändernde Hintergrund eine Herausforderung ist. In erster Linie wurden die Lichtverhältnisse gelernt. Der Greifer verschwindet nahezu in allen Rekonstruktionen.



**Abbildung 3.1:** Ergebnis Regression auf Autoencoder



**Abbildung 3.2:** Embedding Autoencoder



**Abbildung 3.3:** Rekonstruktion Autoencoder

## 3.2. Aufgabenfokussierung und Greifererkennung

Im Ansatz 3.1 wurde die fehlende Fokussierung des Autoencoders auf den Greifer als Schwäche ausgemacht. Als neuer Ansatz wird untersucht, ob ein gleichzeitiges Lernen der Datenrepräsentation und das finden des Rahmens um den Greifer, den Autoencoder fokussiert und gleichzeitig gute Ergebnisse für die Regression gefunden werden können. Konkret wird ein MTL SIMO Ansatz untersucht.

### 3.2.1. Werkzeug: TaskFocusingOnAutoencoder

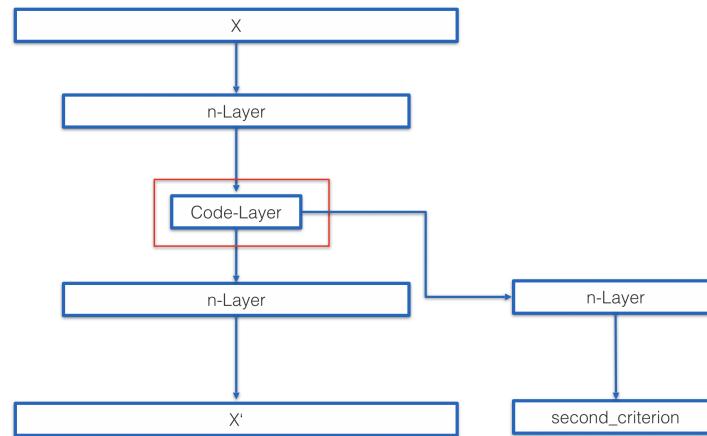
Zur Umsetzung des Ansatzes wurde ein Modul in Python erstellt. Da eine Aufgabe des Multi-Task-Ansatzes ein Autoencoder ist, wurde als Basis die Klasse ConvolutionalAutoencoder des Moduls autoencoder.py genutzt. Es wurde ein neues Modul erstellt, welches zusätzlich zu der Rekonstruktion des Autoencoders einen weiteren Ausgang bereitstellt. Der weitere 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 Ansatzes abgebildet. Die Schichten des weiteren Kriteriums werden an die Code-Schicht des Autoencoders angehängt. Es können beliebig viele Schichten genutzt

### 3. Experimente und Werkzeuge

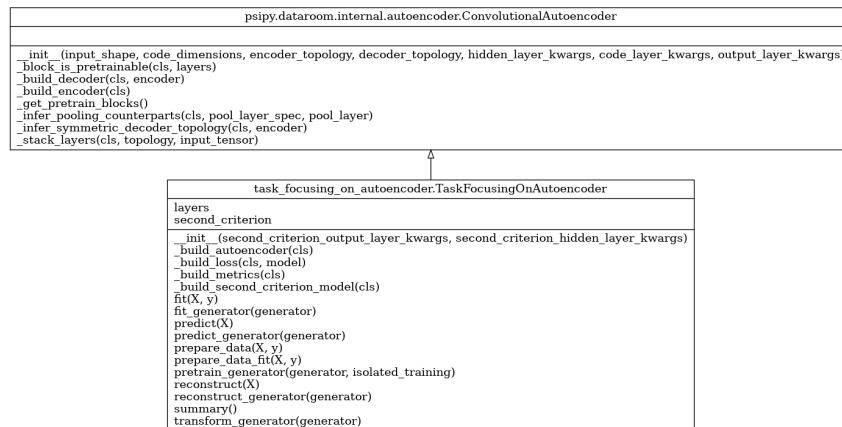
werden. Die Verlustfunktion des neuronalen Netzwerkes besteht aus der Summe der einzelnen Verlustfunktionen und einer Gewichtung. Sie lautet im Detail:

$$loss = weight1 * loss\_autoencoder + weight2 * loss\_task2 \quad (3.1)$$

Die Gewichtung der Verlustfunktionen wird per Konstruktor-Argument übergeben. In Abbildung 3.5 ist das Klassendiagramm des TaskFocusingOnAutoencoder kurz TFAE dargestellt. Über den Konstruktor können alle Argumente, welche zum Er-



**Abbildung 3.4:** Schema TaskFocusingOnAutoencoder



**Abbildung 3.5:** Klassendiagramm TaskFocusingOnAutoencoder

stellen des 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 Methoden 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 TFAE 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 TFAE erstellt wird. In den ersten 10 Zeilen wird die Architektur erstellt. Ab Zeile 12 wird eine Instanz eines TFAE mittels Argumentenliste erstellt. Zu beachten ist, dass hier keine Decoder-Architektur übergeben wird. Wenn keine Decoder-Architektur bereitgestellt wird, 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 tfae = TaskFocusingOnAutoencoder(
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 tfae.pretrain(x_train, epochs = 20, batch_size = 64)
```

**Listing 3.2:** Beispieldaufruf Pretrain in Python

### 3. Experimente und Werkzeuge

---

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 = tfae.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})  
7 )
```

**Listing 3.3:** Beispielaufruf Fit in Python

#### 3.2.2. Experiment

Mit dem TFAE<sup>1</sup> wird für die Aufgabe 'Greifererkennung' im Median für den Schwellenwert 0.5 eine Leistung von 98,77% erreicht, für den Schwellenwert von 0.8 eine Leistung von 68.30%. In Abbildung 3.6 sind die Ergebnisse im Detail dargestellt. In Anhang C.2 ist eine detaillierte Einzelaufstellung der Ergebnisse zu finden. Diese Leistung ist deutlich besser, als die erzielte Leistung mit dem Ansatz 'Greifererkennung auf Repräsentation' 3.1. Bei Betrachtung der Repräsentation in Abbildung 3.7 ist eine deutliche Anpassung an den Greifer zu erkennen. Sowohl die Farbkodierung für die y-Position als auch die x-Position des Greifers im Bild ist sehr deutlich zu erkennen. Zusätzlich lässt sich eine Trennung der einzelnen Datenpunkte in das Merkmal Greifer ist offen oder geschlossen erkennen. Es zieht sich eine sichtbare Trennung der Datenpunkte durch die Einbettung. Bei Betrachtung der Rekonstruktionen, insbesondere im direkten Vergleich zu den Rekonstruktionen des Autoencoders zeigt sich eine stärkere Ausprägung des Merkmals Greifer. In Abbildung 3.8 sind drei Bilder mit ihren jeweiligen Rekonstruktionen dargestellt. Im Anhang C.2 ist eine größere Auswahl an Rekonstruktionen dargestellt.

In Abbildung 3.7c sind in der Einbettung die Fehlvorhersagen lila eingefärbt. Die meisten Fehler sind bei geschlossenem Greifer entstanden. Werden die Einbettungen aller Versuche (C.2) betrachtet, fällt auf, dass die x-Position des Greifers im Bild die herausfordernde Größe ist.

---

<sup>1</sup>TaskFocusingOnAutoencoder

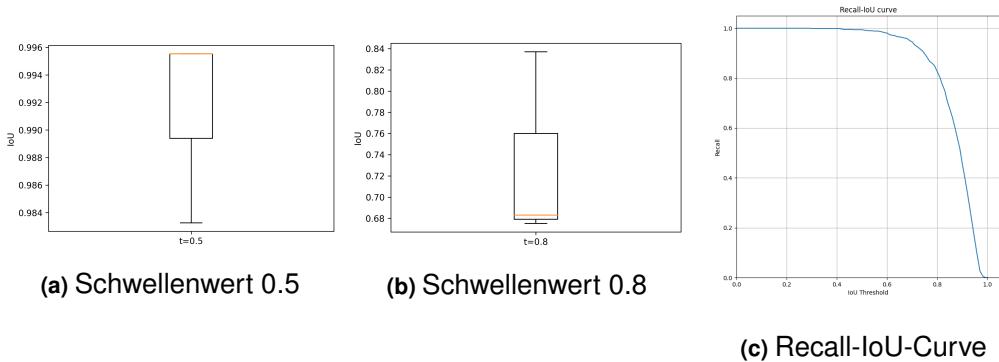


Abbildung 3.6: Ergebnis MT-Greifererkennung

### 3.3. Transferlernen und 'Greifer beladen?' -Klassifikation

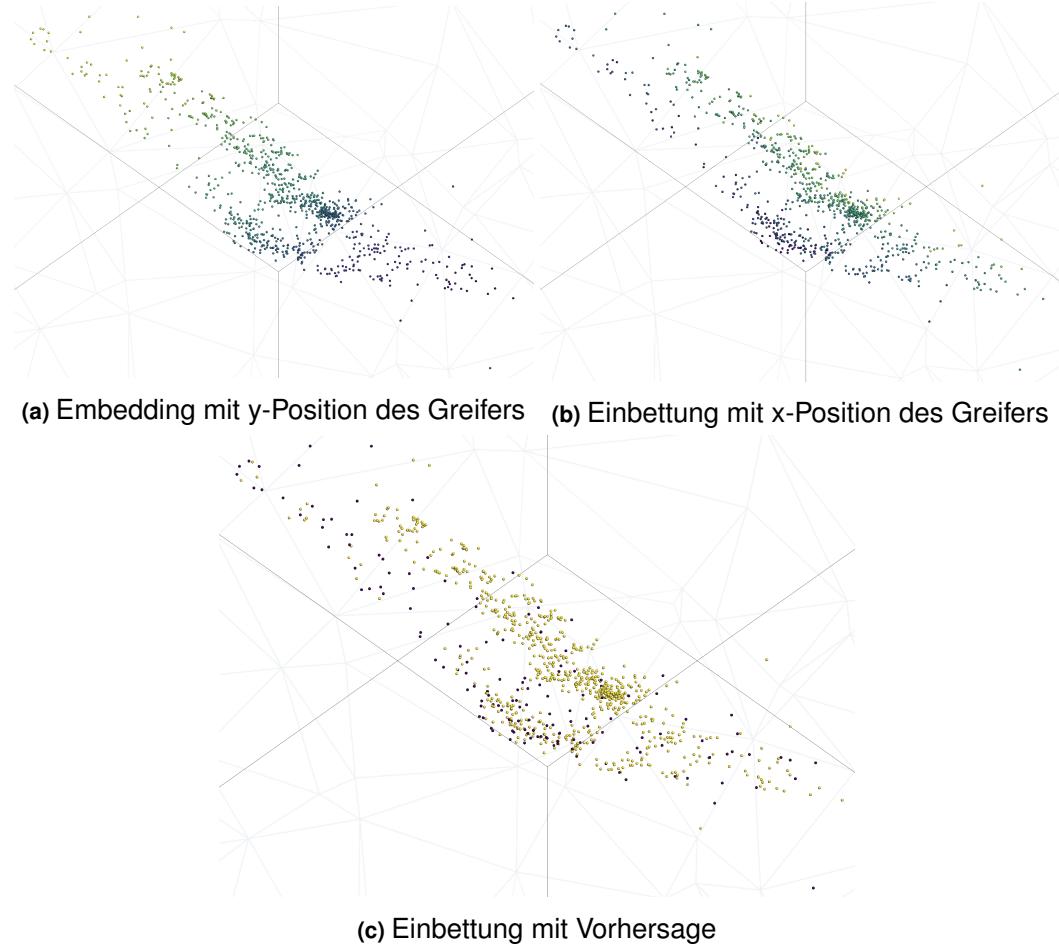
In vorangegangenen Kapitel wurde eine Repräsentation gefunden, welche die Merkmale des Greifer herausbildet. Ausgehend von dieser Repräsentation wird untersucht, ob ein Transfer auf weitere Aufgaben durchgeführt werden kann. Es wird der Ansatz des netzwerkbasierter tiefen Transferlernens genutzt um die Aufgabenstellung 'Greifer beladen?' zu lösen.

#### 3.3.1. Werkzeug: TaskTransferOnAutoencoder

Als Quelldomäne wird ein Netzwerk, welches mittels TFAE erstellt wurde genutzt. In der Zieldomäne werden die Architektur und die Gewichte des Autoencoder weiterverwendet. Der zweite Task wird durch eine neue Aufgabe ersetzt. Abbildung 3.9 zeigt das Schema des Ansatzes. Zur einfachen Anwendung des Ansatzes wurde ein Python-Modul mit der Klasse TaskTransferOnAutoencoder kurz TTAE<sup>2</sup> erstellt. Abbildung 3.10 zeigt das zugehörige Klassendiagramm. Als Basis wird ein TFAE als Konstruktorargument übergeben. Die Parameter für den Autoencoder werden aus diesem Modell kopiert. Zusätzlich müssen noch die Einstellungen für die zweite Aufgabe übergeben werden. Aus diesen beiden Teilen wird das neue Modell erstellt. Die Parameter `freeze_encoder_layers` und `freeze_decoder_layers` ermöglichen es die Merkmalstransformation unverändert zu lassen. Es wird darüber gesteuert welche Schichten nicht neu trainiert werden können. Listing 3.4 zeigt beispielhaft die Anwendung dieses Werkzeuges. Im Vergleich zu dem TFAE ist die Anwendung schon deutlich einfacher. Es gibt weniger Hyperparameter zum Modifizieren. Da

<sup>2</sup>TaskTransferOnAutoencoder

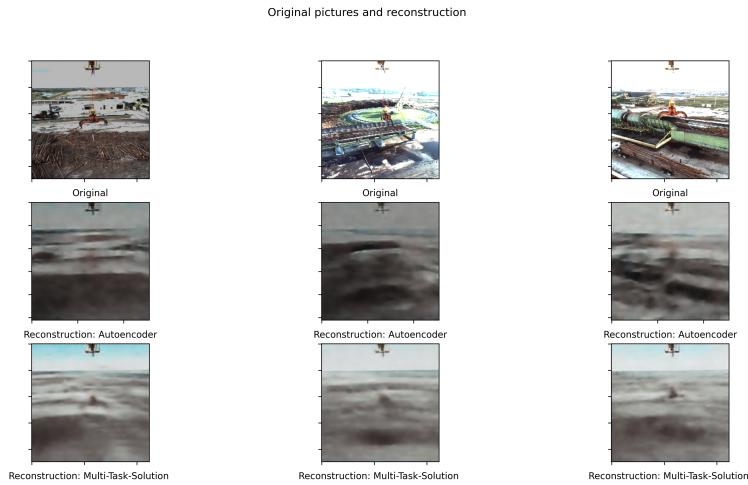
### 3. Experimente und Werkzeuge



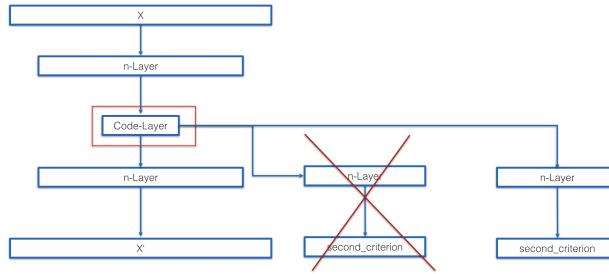
**Abbildung 3.7:** Einbettung MT

das Modell auf einem trainierten TFAE basiert, ist kein *pretrain(..)* mehr notwendig. Die Methode *fit(..)* / *fit\_generator(..)* wird auf dieselbe Weise wie in Keras 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(
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}))
```



**Abbildung 3.8:** Rekonstruktion - AE - MT



**Abbildung 3.9:** Schema TaskTransferOnAutoencoder

16 )

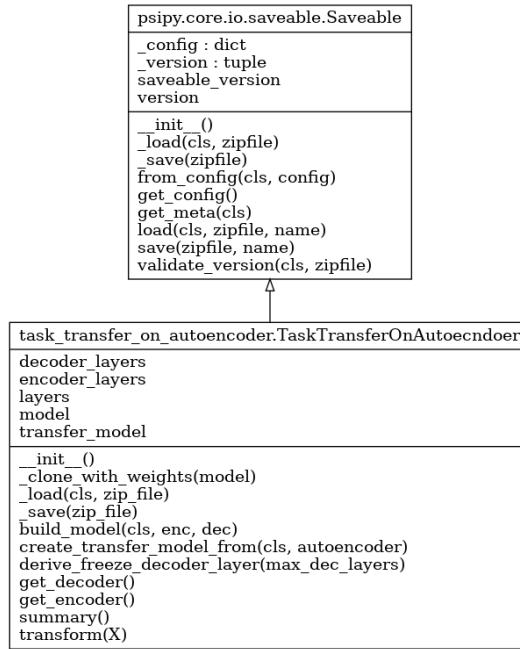
**Listing 3.4:** Beispiel TransferSecondCriterionAutoencoder in Python

#### 3.3.2. Experiment

In 3.2 wurde ein fokussiertes Embedding erstellt. Zur Überprüfung, ob ausgehend von dieser Lösung, neue Aufgaben gelöst werden können wird ein Transfer auf die Aufgabe 'Greifer beladen?' durchgeführt. Dabei kommt der gerade vorgestellte Ansatz zum Einsatz.

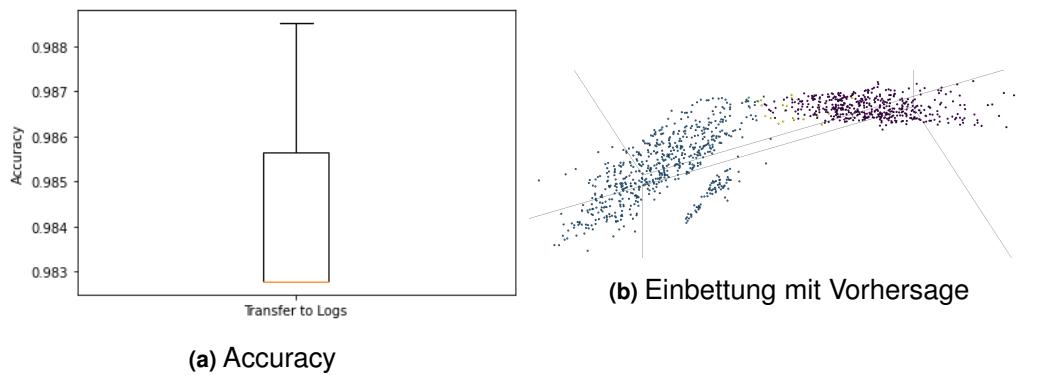
Mit dem vorgestellten Ansatz lässt sich die Aufgabe 'Greifer beladen?' lösen. Im Median von drei Versuchen wird ein Accuracy von 0.9827% erreicht, was nahezu der Leistung der Basislinie mit einer Accuracy von 0.9828% entspricht. In Abbildung 3.11 sind die Ergebnisse des Versuchs dargestellt. Die Unterabbildung 3.11b

### 3. Experimente und Werkzeuge



**Abbildung 3.10:** Klassendiagramm TaskTransferOnAutoencoder

stellt die neu gefundene Einbettung dar. Die blauen Datenpunkte entsprechen der Vorhersage True Positiv, die lila Datenpunkte der Vorhersage True Negativ, gelb entspricht False Positiv und grün False Negativ. Es ist eine deutliche Anpassung der Einbettung an die Problemstellung erkennbar, wobei der Übergang von der einen zur anderen Klasse nicht 100% korrekt ist. Der erfolgreiche Transfer zeigt, dass das die Fokussierung des Basis-Autoencoders erfolgreich ist.



**Abbildung 3.11:** Ergebnis Transfer

### 3.4. AutoML und 'Greifer beladen?'

Die starke Anpassung an die Klassifikationsaufgabe im vorangegangenen Versuch motiviert einen Blick auf die Hyperparameter zur Gewichtung der Verlustfunktion zu werfen. Ziel dieses Versuches ist es, herauszufinden welchen Einfluss die Hyperparameter auf das Ergebnis hat. Es gibt sehr viele Möglichkeiten für die Gewichtung der Verlustfunktion. Um den manuellen Aufwand für weitere Anwendungen gering zu halten, wird ein neues Modul erstellt, welches auf AutoML zur Hyperparamet- roptimierung zurückgreift.

#### 3.4.1. Werkzeug: AutoTaskTransferOnAutoencoder

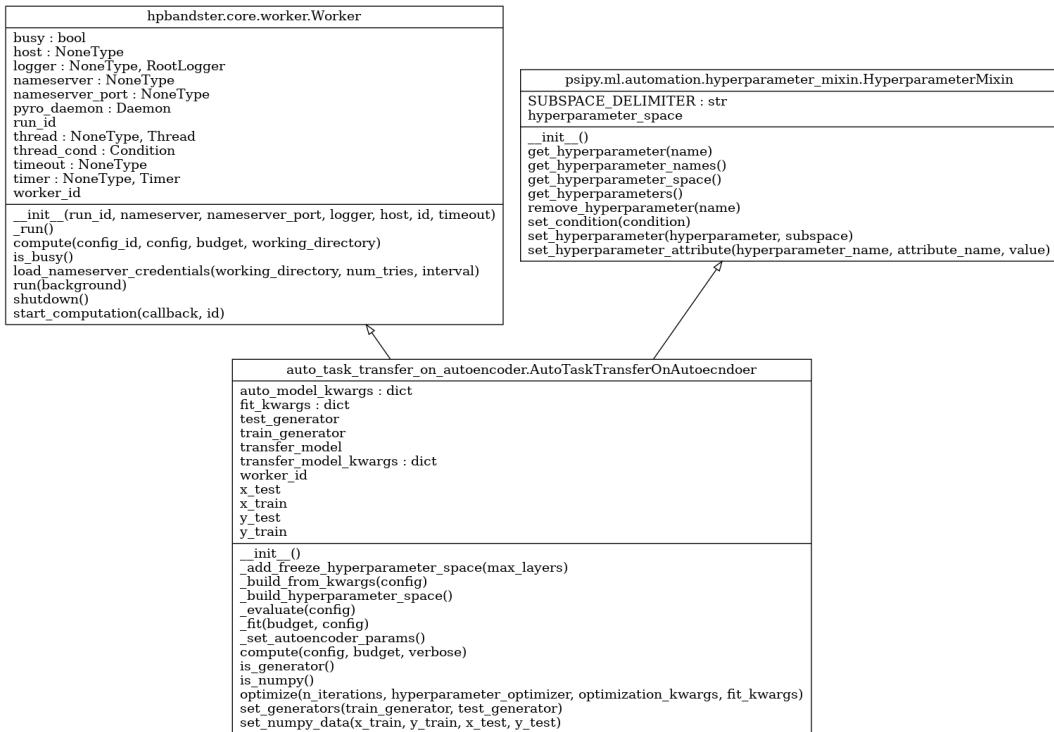
Das neue Modul integriert den Ansatz in der Klasse AutoTaskTransferOnAutoencoder, kurz AutoTTAE<sup>3</sup>. Sie integriert die Klasse TaskTransferOnAutoencoder und erweitert sie mithilfe des HpBandSter-Frameworks um AutoML-Ansätze zur HPO. Konkret erbt die Klasse von hpbandster.core.worker. Zur Speicherung und Verwaltung der Hyperparameter wird auf die Klasse HyperparameterMixin zurückgegrif- fen. Nach nach dem Instanziieren der Klasse können weitere sinnvolle Hyperpara- meter gesetzt werden. In Abbildung 3.12 ist das zugehörige Klassendiagramm ab- gebildet. In Listing 3.5 ist eine einfache Implementierung eines AutoTaskTransfe- rOnAutoencoder dargestellt. Der Konstruktor unterscheidet sich nur an einer Stelle zum Konstruktor des TaskTransferOnAutoencoder. Es wird keine Instanz des SCAE übergeben, sondern ein Pfad zu einem abgespeicherten Modell eines SCAE. Im Ge- gensatz zur bisherigen Vorgehensweise werden die Trainings und Testdaten mit der Methode *set\_generators(..)* oder *set\_numpy\_data(..)* übergeben. Ziel ist es da- bei 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 zu- sätzliche Einstellungen für die einzelnen Optimierer enthalten kann. In jeder Iterati- on der Optimierung wird ein neuer TaskTransferOnAutoencoder 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_kwargs = {'activation': 'relu'},
```

<sup>3</sup>AutoTaskTransferOnAutoencoder

### 3. Experimente und Werkzeuge



**Abbildung 3.12:** Klassendiagramm AutoTaskTransferOnAutoencoder

```

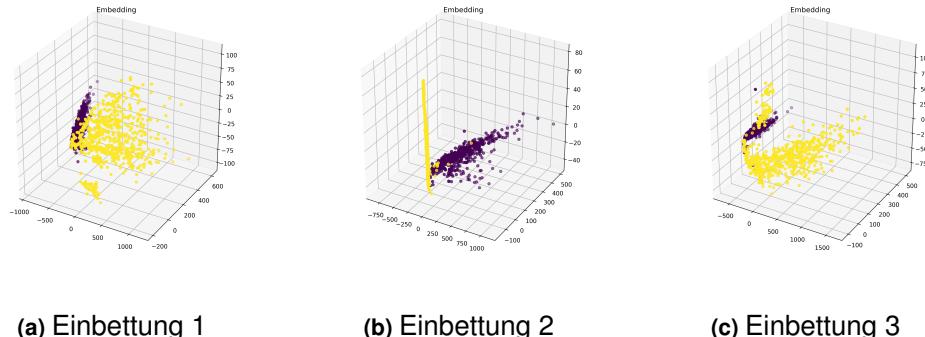
6   second_criterion_output_layer_kwarg = {'activation': 'softmax'},
7   second_criterion_metrics = {'second_criterion': 'accuracy'}
8 )
9
10 tscm.set_generators(train_datagenerator, test_datagenerator)
11
12
13 best_config, history = tscm.optimize(3
14 , 'RandomSearch'
15 , optimization_kwarg = optimization_kwarg)
  
```

**Listing 3.5:** Beispiel AutoTransferSecondCriterionAutoencoder in Python

Das Werkzeug unterstützt derzeit die Optimierer, Zufallssuche, Hyperband und BOHB. Der vom Framework bereitgestellte Parameter Budget wird zur Festlegung der Epochen eingesetzt. Um ein Overfitting zu verhindern, wird ein EarlyStopping Kriterium eingesetzt. Besondere Beachtung muss die Evaluationsmetrik zur Bewertung eines Optimierungslaufes finden. Da ein zu optimierender Hyperparameter die Gewichtung der Verlustfunktion ist, muss dies bei dem Einsatz der Evaluationsmetrik berücksichtigt werden. Die Evaluationsmetrik kann bei dem Erstellen der Klasse frei gewählt werden.

### 3.4.2. Experiment

In diesem Versuch wird die selbe Ausgangslage wie im Vorangegangenen Versuch genutzt. Der Unterschied besteht darin, dass der Hyperparameter 'Gewichtung der Verlustfunktion' automatisch gesucht wird. Zur Bewertung eines Optimierungs-durchlaufs wird der Wert  $1 - Accuracy$  des finalen Modells herangezogen. Die Ergebnisse von drei Optimierungsdurchläufen mit jeweils drei Iterationen sind in Anhang E zu finden. In den finalen Ergebnissen für die Aufgabe 'Greifer beladen?' erreichen alle drei Versuch mindestens eine Accuracy von 0.98% bei verschiede-nen Gewichtungen. Wirft man einen Blick auf die Durchläufe mit vollem Budget von 15 ist in Versuch [2,0,1] E.3 zu sehen, dass die Accuracy deutlich auf 0,79% abfällt. In Abbildung 3.13 sind die Einbettungen der drei Optimierungsdurchläufe abgebildet. Die Einzelnen Datenpunkte sind entsprechend ihrer Klasse eingefärbt. Es ist in allen drei Fällen eine Anpassung an die Klassifizierungsaufgabe erkenn-bar. In Durchlauf drei 3.13c ist die Anpassung deutlich geringer als in den anderen beiden Durchläufen. Der Versuch [2,0,1] und die Einbettung drei liefern deutliche Hinweise, dass die Gewichtung der beiden Teile der Verlustfunktion einen starken Einfluss auf die Modellqualität haben.



**Abbildung 3.13:** Einbettungen AutoML-Transfer

### 3.5. Datenmenge und 'Greifer beladen?'

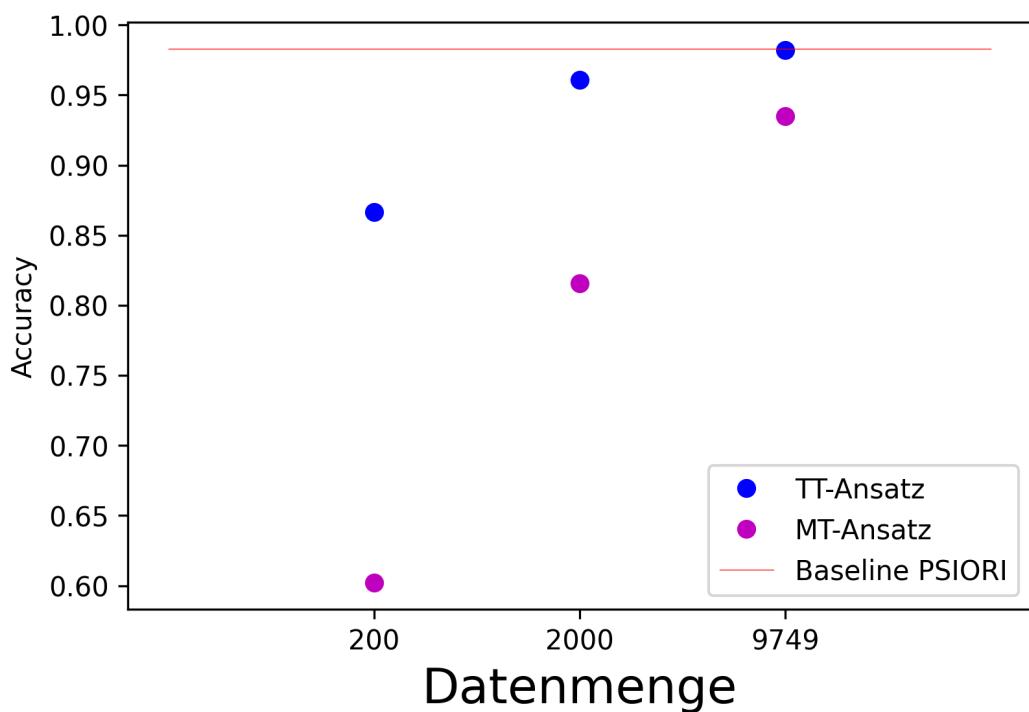
In den bisherigen Experimenten wurde die Grundsätzliche Funktionalität der Ansätze untersucht. Besonderes Kosten-Einsparungspotential besteht bei einem Einsatz von geringeren Datenmengen. In diesem Versuch wird untersucht wie sich die Lö-sung bei unterschiedlichen Datenmengen verhält. Es wird untersucht, wie sich die

### 3. Experimente und Werkzeuge

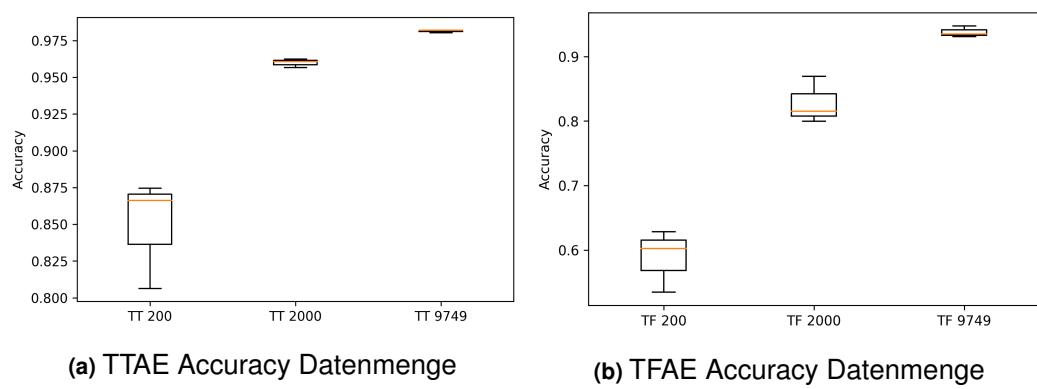
---

Leistung der Ansätze bei 200, 2000 und 9749 annotierten Daten verhält. Als Gewichtungsfaktor der Verlustfunktion werden die Hyperparameter von Versuch [2, 0, 2] E 2 genutzt.

In Abbildung 3.14 sind die Ergebnisse des Versuchs für die Aufgabe ist der Greif-er beladen dargestellt. Auf der X-Achse sind die Datenmengen, auf der y-Achse die Accuracy dargestellt. Der Versuch wurde einmal als TTAE-Ansatz hier blau und einmal als TFAE-Ansatz (lila) durchgeführt. Bei allen drei Datenmengen er-reicht der TTAE-Ansatz deutlich bessere (22%,15%,5%) Ergebnisse. Insbesondere bei niedrigen Datenmengen erzielt der TTAE-Ansatz gute Ergebnisse. Bei 2000 Bildern ist die Accuracy nur 2% schlechter als die Basislinie. Die Einzelergebnisse sind in Abbildung 3.15 dargestellt.



**Abbildung 3.14:** Vergleich-Ansätze Datenmenge



**Abbildung 3.15:** Datenmengen Leistung



## **4. Fazit**

Die Arbeit schließt mit einer Zusammenfassung der Inhalte ab, auf welche eine kritische Auseinandersetzung mit verschiedenen Ergebnissen der Abschlussarbeit folgt und durch einen ausführlichen Blick auf mögliche Erweiterungen und weitere geleistete Arbeiten komplettiert wird.

### **4.1. Zusammenfassung**

Zu Beginn der Arbeit wurde die zu bearbeitende Problemstellung erläutert, anschließend wurden die theoretischen Hintergründe, das Umfeld der Problemstellung und die Datensätze vorgestellt. Der Hauptteil der Arbeit beschäftigt sich mit aufeinander aufbauenden Ansätzen des Transferlernens. Im ersten Schritt wurde gezeigt, dass ein Ansatz, ausgehend von einer nicht fokussierten Repräsentation fehlschlägt. Darauf folgend wurde ein Multi-Task-Ansatz zum gleichzeitigen Fokussieren und Lösen einer Regressionsaufgabe vorgestellt und evaluiert. Insbesondere die gefundene Repräsentation hat dabei deutlich die aktuelle Domäne widergespiegelt. Darauf aufbauend wurde ein Ansatz des modellbasierten Transferlernen vorgestellt. Die Ergebnisse erreichen eine ähnliche Leistung wie ein Modell, welches von einem Experten erstellt wurde. Zur HPO der Aufgaben-Gewichtungsparameter wurde eine AutoML-Erweiterung eingesetzt. Dabei hat sich gezeigt, dass der Hyperparameter einen deutlichen Einfluss auf das Ergebnis hat und die automatische Suche hilfreich ist. Abschließend wurde gezeigt, dass die Transfer-Lösung insbesondere für geringe Datenmengen gute Ergebnisse liefern kann. Begleitend zu der Vorstellung der Ansätze wurde jeweils eine Python-Implementierung des Ansatzes vorgestellt.

## 4.2. Kritische Reflexion

Eine Abschlussarbeit wird immer in einem Zeitlich begrenzenden Rahmen durchgeführt. Diese zeitliche Begrenzung führt in manchen Fällen dazu, dass manche Teila-spekten der Arbeit nur angerissen oder mit begrenzenden Mittel bearbeitet werden können. In dieser Arbeit betrifft dies insbesondere das Thema AutoML, konkret den Versuch 3.4.2. Das zugehörige Python-Module AutoTTAE baut auf den anderen beiden Modulen auf und wurde somit recht spät in der Masterarbeit bearbeitet. Eine Optimierungs-Iteration dauert circa einen Tag. Diese beiden Faktoren haben dazu geführt, dass der Versuch nur mit 3 Iterationen durchgeführt wurde. Hier emp-fiehlt es sich eine Optimierung mit deutlich mehr Iterationen durchzuführen.

Auf Grund der Einfachheit (vier Ausgangsneuronen) wurde für Aufgabe 'Rahmen um Greifer' eine Regression eingesetzt. In der Regel werden für Objekterkennungs-aufgaben andere Techniken wie z.B. der bereits erwähnte SSD-Ansatz eingesetzt. Durch eine explizite Lokalisierung und Klassifizierung werden oft bessere Ergeb-nisse erzielt.

Die bisherigen Versuche wurden alle in der selben Domäne, mit Bildern und dem Greifer durchgeführt. Die gewählten Ansätze wurden für diese Umgebung gewählt. Die Aussagekraft auf andere Domänen oder Umgebungen mit weniger stark ausge-prägten Merkmalen wie ein Greifer ist schwach und sollte in weiteren Versuchen untersucht werden.

## 4.3. Ausblick und weitere Arbeiten

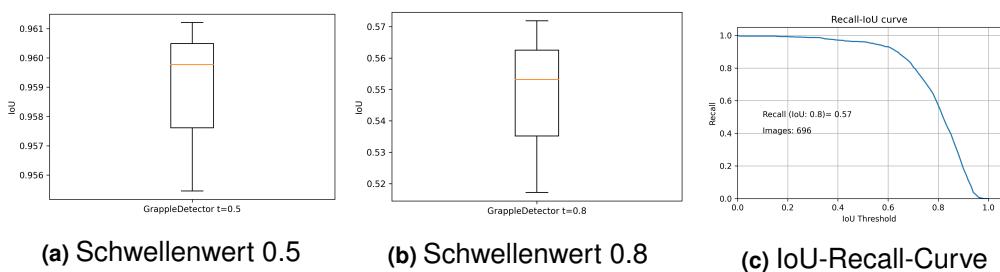
In diesem letzten Unterkapitel werden weitere (vorläufige) Ergebnisse dargestellt und ein Blick auf mögliche Erweiterungen geworfen.

### 4.3.1. 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. In Experiment 3.5 und

3.3.2 wurde gezeigt, dass der Transfer für die Aufgabe 'Greifer beladen?' funktioniert und insbesondere bei wenigen Daten stark ist. Falls es möglich ist, dass ein Transfer von der Aufgabe 'Greifer beladen?' zu der Aufgabe 'Rahmen um Greifer' möglich ist, kann sehr viel Geld gespart werden, da die Annotationen deutlich teurer sind.

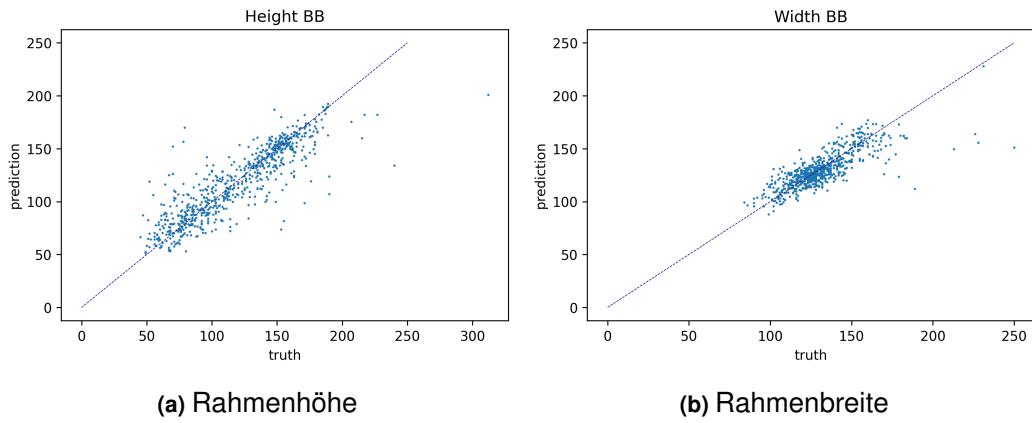
Im ersten Versuch ist der Transfer auf dem Greiferdatensatz fehlgeschlagen, durch weitere Arbeiten an dem Problem konnte eine Verbesserung erzielt werden. Zu den weiteren Arbeiten zählen, das Entfernen von Bildern bei denen der Greifer über den Rand des Bildes hinausgeht. Diese Bildern können durch eine Modifikation der Kameraposition am Kran nicht mehr vorkommen. Weitere Verbesserungen konnten durch Regularisierungstechniken erzielt werden. Im Anhang F ist die aktuelle Modellzusammenfassung zu finden. In Abbildung 4.1 sind die Ergebnisse der Aufgabe 'Rahmen um Greifer' dargestellt. Bei einem Schwellenwert von 0.5 wird eine Leistung von 96% erreicht, bei einem Schwellenwert 0.8 eine Leistung von 55%. Bei genauerer Betrachtung der Vorhersagen fällt auf, dass der Greifer gefunden wird. Die genaue Form des Greifers wird nicht immer korrekt vorhergesagt. Dieses Verhalten ist auch in Abbildung 4.2 zu sehen. In dieser Art von Abbildungen werden die vorhersagen gegen die Beschriftungen aufgetragen. Bei einer perfekten Vorhersage befinden sich alle Datenpunkte auf der Diagonale. Sowohl in der Darstellung der Rahmenhöhe, als auch der Rahmenbreite, ist zu sehen, dass die Vorhersage zur Beschriftung passt, aber sich noch verbessern kann. In Abbildung 4.3c sind die für den Schwellenwert 0.8 falsch vorhergesagten Datenpunkte in der Einbettung lila eingefärbt. Die falschen Vorhersagen verteilen sich über alle Datenpunkte. Mit Blick auf die Einfärbung der x und y Position des Greifers 4.3 zeigt sich in den Einbettungen die Position des Greifers im Bild. Widerspiegeln zu den Vorhersageergebnissen ist in der Einbettung die genaue Ausprägung des Greifers schwach repräsentiert.



**Abbildung 4.1:** IoU TTAE Greifer

## 4. Fazit

---



**Abbildung 4.2:** Beschriftung vs Vorhersage

### 4.3.2. Autocrane-Datensatz

Im Rahmen der Arbeit wurde ein Datensatz erarbeitet und Projektpartnern zur Verfügung gestellt. Eine spätere Veröffentlichung des Datensatzes unter der Adresse [psiori.com](http://psiori.com) ist geplant. Der Datensatz soll zu allgemeinen akademischen und pädagogische Zwecken genutzt werden dürfen. Alternativ kann Zugang zu dem Datensatz über die E-Mail-Adresse [info@psiori.com](mailto:info@psiori.com) angefragt werden.

### 4.3.3. Erweiterung: Multi-Task-Ansatz

Multi-Task-Lernen beschränkt sich nicht auf zwei Aufgaben. Die gezeigte Lösung kann um weitere Aufgaben erweitert werden. In Abbildung 4.4 ist der erweiterte Ansatz schematisch abgebildet. Wie in den bisherigen Ansätzen werden ausgehend von der Code-Schicht weitere Aufgaben bearbeitet. Durch die weiteren Aufgaben wird die Gewichtung der einzelnen Aufgaben noch schwerer. In Experiment 3.4.2 wurde gezeigt, dass AutoML bei der Gewichtung helfen kann. Dieser erweiterte Ansatz soll in einem Anschlussprojekt für die Praxistauglichkeit untersucht werden. Der Aufwand, das bisher entwickelte Python-Modul zu erweitern, wird sich dabei in Grenzen halten. Insbesondere müssen im Konstruktor Erweiterungen zum Erstellen der Modelle der weiteren Aufgaben getätigten werden. An anderen Stellen wie z. B. der Methode zum Trainieren des Gesamtmodells wird keine Erweiterung notwendig sein, da bereits mit Listen gearbeitet wird.

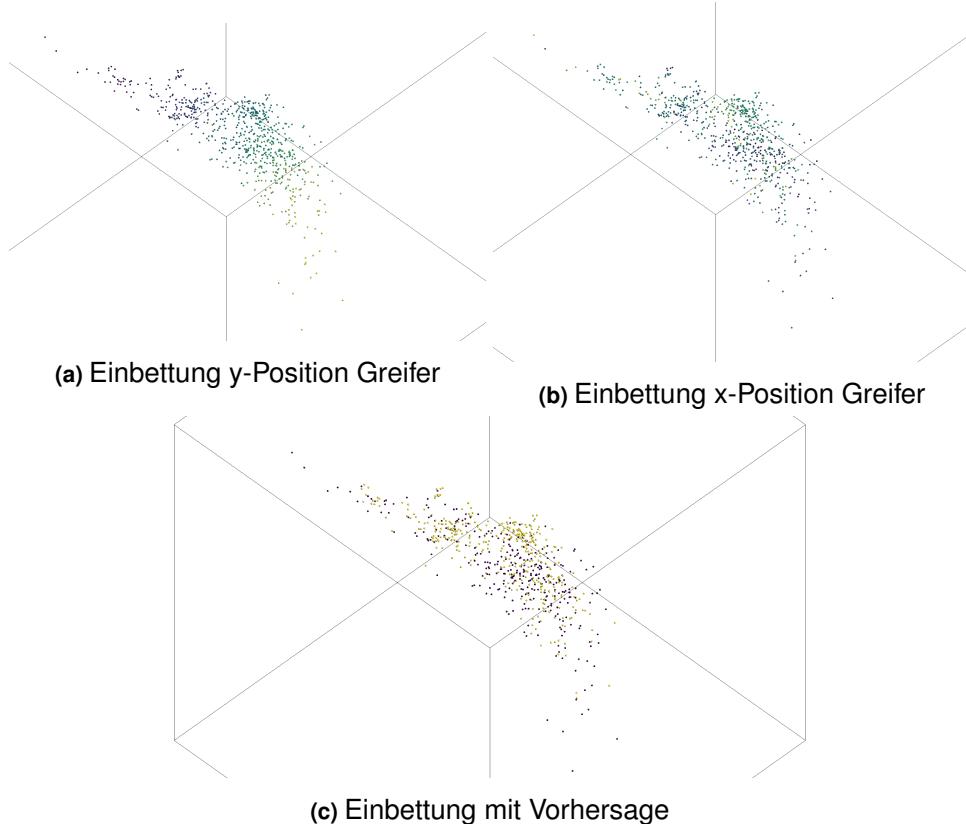


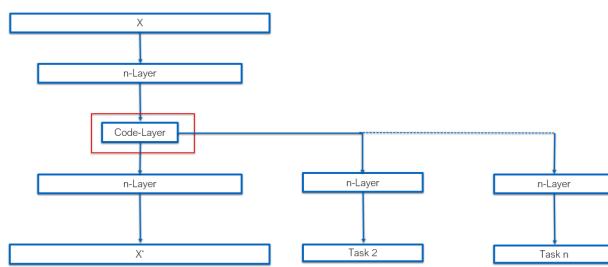
Abbildung 4.3: Einbettung TTAE Greifer

#### 4.3.4. Flexibilität der Werkzeuge

Die bisherigen Module sind starr bei der Anwendung. Es ist notwendig, ein Multi-Aufgaben-Modell zu erstellen, um anschließend den modellbasierten Transfer durchzuführen. Soll ein neuer modellbasierter Transfer durchgeführt werden, ist es zwingend erforderlich dies ausgehend des Multi-Aufgaben-Modells durchzuführen. Es ist nicht möglich einen modellbasierten Transfer, ausgehend von einem vorherigen Transfer durchzuführen. Als mögliche Erweiterung der Module ist es empfehlenswert, eine Flexibilisierung des Gesamtansatzes durchzuführen. Durch eine Anpassung bei der Modellerstellung der Module, ist es möglich, dass die Abfolge nicht mehr notwendig ist. Es ist vorstellbar, dass ein neues Modell ausgehend von einer Architekturdefinition komplett neu trainiert wird, oder ausgehend von einem Autoencoder-Modell mit null bis beliebig vielen weiteren Aufgaben trainiert werden kann. Konkret muss im Konstruktor der Module eine Zusammenführung der bisher getrennten Ansätze durchgeführt werden.

#### 4. Fazit

---



**Abbildung 4.4:** Multi-Task-Ansatz



# Abkürzungsverzeichnis

<b>CAE</b>	Convolutional Autoencoder .....	8
<b>MTL</b>	Multi-Task-Lernen .....	10
<b>SIMO</b>	Single-Input Multi-Output .....	22
<b>AutoML</b>	automatisierte maschinelle Lernen .....	12
<b>NAS</b>	Neural Architecture Search .....	12
<b>HPO</b>	Hyperparameter-Optimierung .....	12
<b>SH</b>	Successive Halving .....	14
<b>IoU</b>	Intersection over Union .....	19
<b>SSD</b>	Single Shot MultiBox Detector .....	18
<b>TFAE</b>	TaskFocusingOnAutoencoder .....	30
<b>TTAE</b>	TaskTransferOnAutoencoder .....	31
<b>AutoTTAE</b>	AutoTaskTransferOnAutoencoder .....	35



# **Tabellenverzeichnis**

2.1. Datenaufteilung - Train Test Validation . . . . .	21
C.1. Einzelwerte Boxplot IoU Greifererkennung auf Autoencoder . . . . .	xix
C.2. Einzelwerte Boxplot IoU MT-Greifer . . . . .	xix
D.1. Ergebnisse 'Greifer beladen?' - Datenmengen - Transfer-Ansatz und Multi-Task-Ansatz . . . . .	xxiii
F.1. Einzelwerte Boxplot IoU TT-Greifer . . . . .	xxxiii



# Abbildungsverzeichnis

2.1.	Schema Autoencoder . . . . .	8
2.2.	Multi-Task-Lernen: Ausprägungen . . . . .	11
2.3.	Klassendiagramm ConvolutionalAutoencoder . . . . .	17
2.4.	Klassendiagramm Hyperparametermixin . . . . .	18
2.5.	Rundlaufkran . . . . .	18
2.6.	Greifer . . . . .	19
2.7.	Bildqualität . . . . .	20
3.1.	Ergebnis Regression auf Autoencoder . . . . .	26
3.2.	Embedding Autoencoder . . . . .	26
3.3.	Rekonstruktion Autoencoder . . . . .	27
3.4.	Schema TaskFocusingOnAutoencoder . . . . .	28
3.5.	Klassendiagramm TaskFocusingOnAutoencoder . . . . .	28
3.6.	Ergebnis MT-Greifererkennung . . . . .	31
3.7.	Einbettung MT . . . . .	32
3.8.	Rekonstruktion - AE - MT . . . . .	33
3.9.	Schema TaskTransferOnAutoencoder . . . . .	33
3.10.	Klassendiagramm TaskTransferOnAutoencoder . . . . .	34
3.11.	Ergebnis Transfer . . . . .	34
3.12.	Klassendiagramm AutoTaskTransferOnAutoencoder . . . . .	36
3.13.	Einbettungen AutoML-Transfer . . . . .	37
3.14.	Vergleich-Ansätze Datenmenge . . . . .	38
3.15.	Datenmengen Leistung . . . . .	39
4.1.	IoU TTAE Greifer . . . . .	43
4.2.	Beschriftung vs Vorhersage . . . . .	44
4.3.	Einbettung TTAE Greifer . . . . .	45
4.4.	Ausblick Multi-Task-Ansatz . . . . .	46
A.1.	RecallIoUGrappleBaseline . . . . .	xv
B.1.	Baumstammklassifikation Basislinie ROC . . . . .	xvii
B.2.	Konfusionsmatrix Baumstammklassifikation Basislinie . . . . .	xvii
B.3.	Klassifikationsreport Baumstammklassifikation Basislinie . . . . .	xvii

## Abbildungsverzeichnis

---

C.1. Orginal - AE - MT . . . . .	xx
C.2. Einbettungen MT-Greifererkennung V1-3 . . . . .	xxi

# Listings

2.1.	Erweiterung zum Speichern eines Tensorflow Models . . . . .	16
2.2.	Aufbereitung Generatorergebnis in Python . . . . .	22
3.1.	Beispiel Erstellung ConvolutionalSecondCriterionAutoencoder in Python . . . . .	29
3.2.	Beispieldruck Pretrain in Python . . . . .	29
3.3.	Beispieldruck Fit in Python . . . . .	30
3.4.	Beispiel TransferSecondCriterionAutoencoder in Python . . . . .	32
3.5.	Beispiel AutoTransferSecondCriterionAutoencoder in Python . . . . .	35
E.1.	AutoML 1 config . . . . .	xxv
E.2.	AutoML 1 results . . . . .	xxvi
E.3.	AutoML 2 config . . . . .	xxvii
E.4.	AutoML 2 results . . . . .	xxviii
E.5.	AutoML 3 config . . . . .	xxviii
E.6.	AutoML 3 results . . . . .	xxix
F.1.	TTAE modelsummary . . . . .	xxxii



# 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>.
- [Ca98] Caruana, R.: Multitask Learning. In (Thrun, S.; Pratt, L., Hrsg.): Learning to Learn. Springer US, Boston, MA, S. 95–133, 1998, ISBN: 978-1-4615-5529-2.
- [Ch14] Christian Szegedy; Wei Liu; Yangqing Jia; Pierre Sermanet; Scott Reed; Dragomir Anguelov; Dumitru Erhan; Vincent Vanhoucke; Andrew Rabinovich: Going Deeper with Convolutions, 2014, URL: <https://arxiv.org/abs/1409.4842>.
- [Ch15] Chollet, F. et al.: Keras, 2015, URL: <https://keras.io/>.
- [cn20] cnvrg.io: cnvrg, 2020, URL: <https://cnvrg.io/>, Stand: 19.02.2020.
- [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.

- [Fr18] Frazier, P. I.: A Tutorial on Bayesian Optimization, Juli 2018, URL: <https://arxiv.org/pdf/1807.02811.pdf>.
- [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>, Stand: 01. 04. 2020.
- [GSH18] George, D.; Shen, H.; Huerta, E. A.: Classification and unsupervised clustering of LIGO data with Deep Transfer Learning. Physical Review D 97/10, 2018, ISSN: 2470-0029.
- [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.
- [Ja14] Jason Yosinski; Jeff Clune; Yoshua Bengio; Hod Lipson: How transferable are features in deep neural networks?, 2014, URL: <https://arxiv.org/abs/1411.1792>.
- [Jo18] Joaquin Vanschoren: Meta-Learning: A Survey. CoRR abs/1810.03548/, 2018.
- [JT15] Jamieson, K.; Talwalkar, A.: Non-stochastic Best Arm Identification and Hyperparameter Optimization, 2015, URL: <https://arxiv.org/pdf/1502.07943.pdf>.
- [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.
- [KSH12] Krizhevsky, A.; Sutskever, I.; Hinton, G.: ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems 25/, 2012.

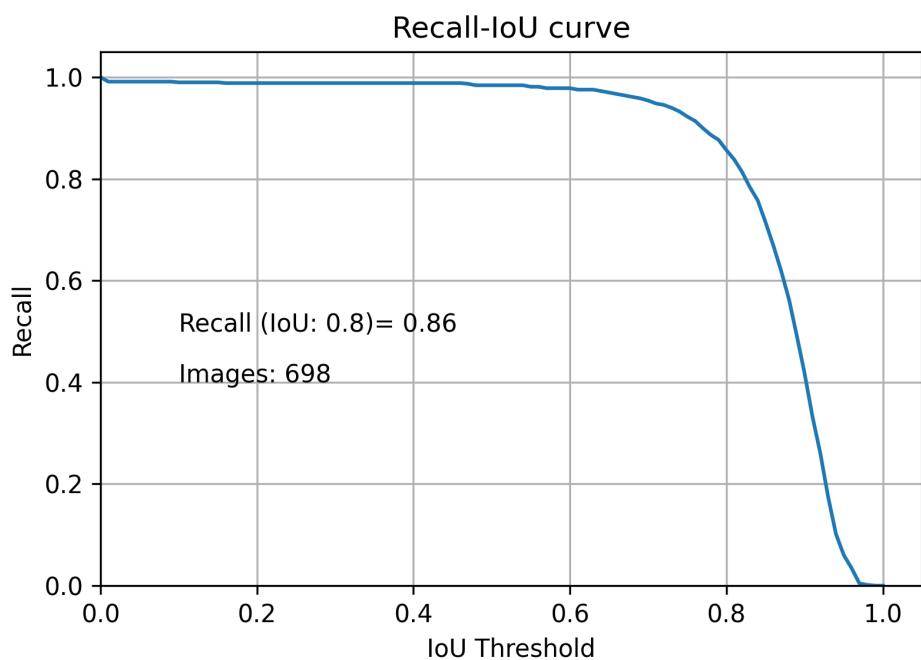
- [KW19] Kingma, D. P.; Welling, M.: An Introduction to Variational Autoencoders. *Foundations and Trends® in Machine Learning* 12/4, S. 307–392, 2019, ISSN: 1935-8245, URL: <http://dx.doi.org/10.1561/2200000056>.
- [LBH15] LeCun, Y.; Bengio, Y.; Hinton, G.: Deep learning. *Nature* 521/7553, S. 436–444, 2015, ISSN: 1476-4687.
- [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.
- [Li16a] Li, X.; Zhao, L.; Wei, L.; Yang, M.-H.; Wu, F.; Zhuang, Y.; Ling, H.; Wang, J.: DeepSaliency: Multi-Task Deep Neural Network Model for Salient Object Detection. *IEEE Transactions on Image Processing* 25/8, S. 3919–3930, 2016, ISSN: 1941-0042.
- [Li16b] 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.pdf>.
- [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>.
- [Li19] 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, 2019, URL: <https://arxiv.org/pdf/1908.06756.pdf>.
- [Lo16] Long, M.; Zhu, H.; Wang, J.; Jordan, M. I.: Unsupervised Domain Adaptation with Residual Transfer Networks. In (D. D. Lee; M. Sugiyama; U. V. Luxburg; I. Guyon; R. Garnett, Hrsg.): *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc, S. 136–144, 2016, URL: <http://papers.nips.cc/paper/6110-unsupervised-domain-adaptation-with-residual-transfer-networks.pdf>.
- [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/>, Stand: 19. 03. 2020.
- [Nv20] Nvidia: Tesla K80, 2020, URL: <https://www.nvidia.com/de-de/data-center/tesla-k80/>, Stand: 19. 02. 2020.
- [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.
- [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>, Stand: 19. 02. 2020.
- [Py20] Python Software Foundation: The Python Language Reference, 2020, URL: <https://docs.python.org/3.7/reference/>, Stand: 19. 02. 2020.
- [Ri11] Rifai, S.; Vincent, P.; Muller, X.; Glorot, X.; Bengio, Y.: Contractive Auto-Encoders: Explicit Invariance during Feature Extraction. In: Proceedings of the 28th International Conference on International

- Conference on Machine Learning. ICML'11, Omnipress, Madison, WI, USA, S. 833–840, 2011, ISBN: 9781450306195.
- [RVR16] Rajeev Ranjan; Vishal M. Patel; Rama Chellappa: HyperFace: A Deep Multi-task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition, 2016, URL: <https://arxiv.org/abs/1603.01249>.
- [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.
- [Ta18] Tan, C.; Sun, F.; Kong, T.; Zhang, W.; Yang, C.; Liu, C.: A Survey on Deep Transfer Learning. In (Kůrková, V.; Manolopoulos, Y.; Hammer, B.; Iliadis, L.; Maglogiannis, I., Hrsg.): Artificial Neural Networks and Machine Learning – ICANN 2018. Springer International Publishing, Cham, S. 270–279, 2018, ISBN: 978-3-030-01424-7.
- [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.
- [Yu17] Yuchun Fang; Zhengyan Ma; Zhaoxiang Zhang; Xu-Yao Zhang; Xiang Bai: Dynamic Multi-Task Learning with Convolutional Neural Network. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. S. 1668–1674, 2017.
- [Zh19] Zhao, Y.; Tang, F.; Dong, W.; Huang, F.; Zhang, X.: Joint face alignment and segmentation via deep multi-task learning. Multimedia Tools and Applications 78/10, S. 13131–13148, 2019.



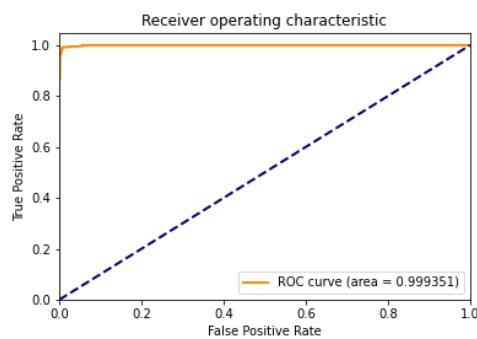
## A. Basislinie Greifer



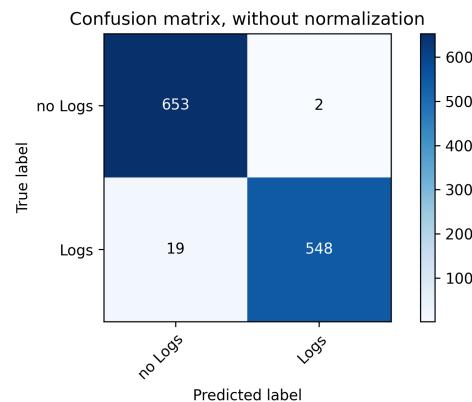
**Abbildung A.1:** Basislinie Greifererkennung



## B. 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



## C. Greifererkennung

### C.1. Greifererkennung auf Autoencoder

Versuch	IoU t=0.5	IoU t=0.8
1	0.3565	0.0127
2	0.3735	0.0227
3	0.3821	0.0383

Tabelle C.1.: Einzelwerte Boxplot IoU Greifererkennung auf Autoencoder

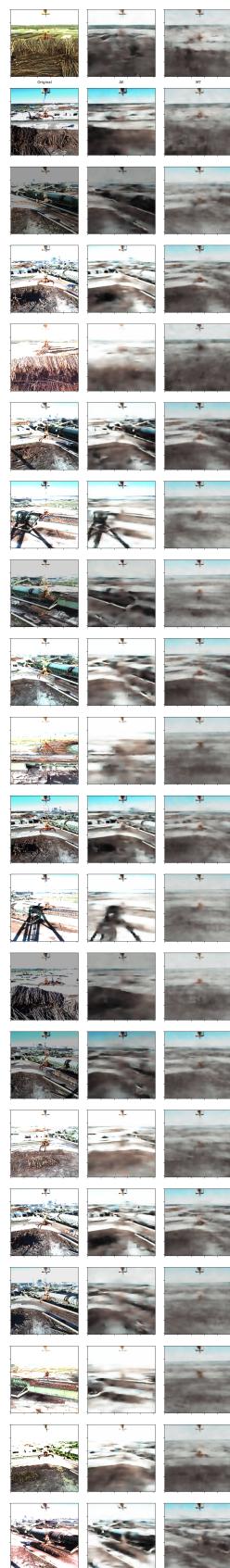
### C.2. Mutli-Task Greifererkennung

Nr.	IoU t=0.5	IoU t=0.8
1	0.9832	0.6830
2	0.9877	0.6752
3	0.9955	0.8370

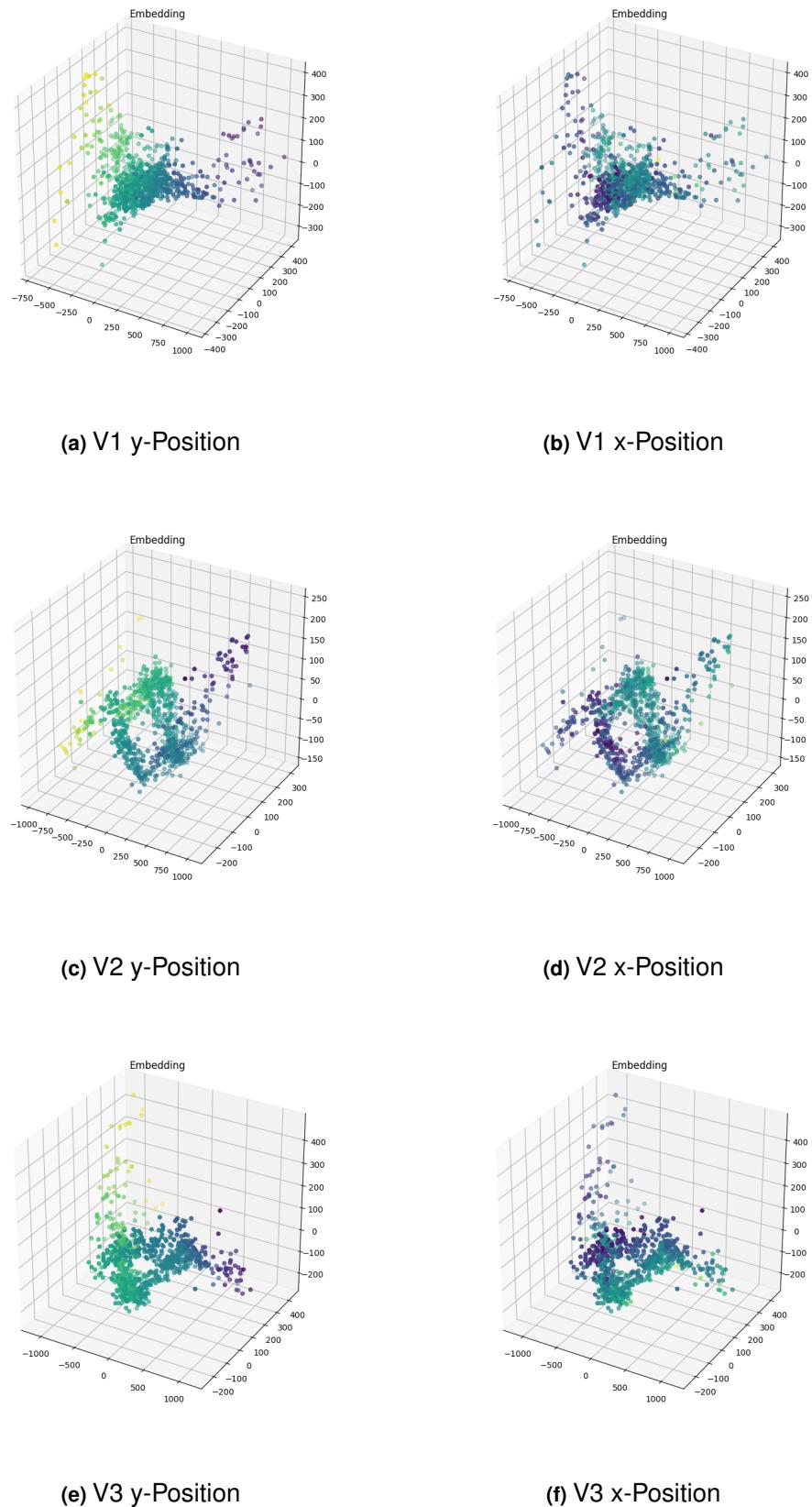
Tabelle C.2.: Einzelwerte Boxplot IoU MT-Greifer

## C. Greifererkennung

---



**Abbildung C.1:** Orginal - AE - MT



**Abbildung C.2:** Einbettungen MT-Greifererkennung V1-3



## D. Baumstämme im Greifer

<b>Transfer Logs 9749</b>	0.9885	0.9827	0.9827
<b>Transfer Logs 200 gewichtet</b>	0.8662	0.8744	0.8063
<b>Transfer Logs 2000 gewichtet</b>	0.9622	0.9565	0.9606
<b>Transfer Logs 9749 gewichtet</b>	0.9803	0.9819	0.9819
<b>MT Logs 200 gewichtet</b>	0.5348	0.6021	0.6283
<b>MT Logs 2000 gewichtet</b>	0.8695	0.8154	0.7998
<b>MT Logs 9749 gewichtet</b>	0.9310	0.9350	0.9474

**Tabelle D.1.:** Ergebnisse 'Greifer beladen?' - Datenmengen - Transfer-Ansatz und Multi-Task-Ansatz



## E. AutoML

```
1 [[0, 0, 0], {"autoencoder_loss_weight": 0.00021209993033661237, "second_criterion_loss_weight": 0.5197943508769243}, {"model_based_pick": false}]  
2 [[0, 0, 1], {"autoencoder_loss_weight": 0.16794720950931571, "second_criterion_loss_weight": 0.005770966645990509}, {"model_based_pick": false}]  
3 [[0, 0, 2], {"autoencoder_loss_weight": 0.00016039854883165636, "second_criterion_loss_weight": 0.6168909156684319}, {"model_based_pick": false}]  
4 [[0, 0, 3], {"autoencoder_loss_weight": 0.0007173242095539351, "second_criterion_loss_weight": 0.13992145854902419}, {"model_based_pick": false}]  
5 [[0, 0, 4], {"autoencoder_loss_weight": 0.0122685335169901, "second_criterion_loss_weight": 0.09488923315549726}, {"model_based_pick": false}]  
6 [[0, 0, 5], {"autoencoder_loss_weight": 0.002896501123937137, "second_criterion_loss_weight": 0.000156546938035246}, {"model_based_pick": false}]  
7 [[0, 0, 6], {"autoencoder_loss_weight": 0.004934828714918127, "second_criterion_loss_weight": 0.11152082378659478}, {"model_based_pick": true}]  
8 [[0, 0, 7], {"autoencoder_loss_weight": 0.0089605998167455, "second_criterion_loss_weight": 0.10280436645416356}, {"model_based_pick": true}]  
9 [[0, 0, 8], {"autoencoder_loss_weight": 0.009490150278196476, "second_criterion_loss_weight": 0.006099547357847074}, {"model_based_pick": false}]  
10 [[1, 0, 0], {"autoencoder_loss_weight": 0.0003820501879486534, "second_criterion_loss_weight": 0.0004275678066579015}, {"model_based_pick": false}]  
11 [[1, 0, 1], {"autoencoder_loss_weight": 0.00018995165142525513, "second_criterion_loss_weight": 0.026726841198269072}, {"model_based_pick": false}]  
12 [[1, 0, 2], {"autoencoder_loss_weight": 0.007596927772956694, "second_criterion_loss_weight": 0.10637389751757345}, {"model_based_pick": true}]  
13 [[2, 0, 0], {"autoencoder_loss_weight": 0.04600764999821077, "second_criterion_loss_weight": 0.09738893931549333}, {"model_based_pick": true}]
```

## E. AutoML

---

```

14 [[2, 0, 1], {"autoencoder_loss_weight": 0.13611398334624183, "second_criterion_loss_weight": 0.10322388535617699}, {"model_based_pick": true}]
15 [[2, 0, 2], {"autoencoder_loss_weight": 0.01933437030411625, "second_criterion_loss_weight": 0.00012966387883219392}, {"model_based_pick": false}]

```

**Listing E.1:** AutoML 1 config

```

1 [[0, 0, 0], 1.6666666666666665, {"submitted": 1590653569.8985553, "started": 1590653569.8986335, "finished": 1590659310.3605034}, {"loss": 0.018047571182250977}, null]
2 [[0, 0, 1], 1.6666666666666665, {"submitted": 1590659310.3621042, "started": 1590659310.3621826, "finished": 1590665054.9218247}, {"loss": 0.02461034059524536}, null]
3 [[0, 0, 2], 1.6666666666666665, {"submitted": 1590665054.9242227, "started": 1590665054.9242227, "finished": 1590670791.651544}, {"loss": 0.02379000186920166}, null]
4 [[0, 0, 3], 1.6666666666666665, {"submitted": 1590670791.6537735, "started": 1590670791.6538463, "finished": 1590676519.5094314}, {"loss": 0.016406893730163574}, null]
5 [[0, 0, 4], 1.6666666666666665, {"submitted": 1590676519.5113797, "started": 1590676519.51145, "finished": 1590682255.8363159}, {"loss": 0.01394587755203247}, null]
6 [[0, 0, 5], 1.6666666666666665, {"submitted": 1590682255.8384755, "started": 1590682255.8385518, "finished": 1590687991.987465}, {"loss": 0.02461034059524536}, null]
7 [[0, 0, 6], 1.6666666666666665, {"submitted": 1590687992.0251045, "started": 1590687992.0251799, "finished": 1590693714.5058358}, {"loss": 0.014766216278076172}, null]
8 [[0, 0, 7], 1.6666666666666665, {"submitted": 1590693714.5408902, "started": 1590693714.5409708, "finished": 1590699432.0042362}, {"loss": 0.02461034059524536}, null]
9 [[0, 0, 8], 1.6666666666666665, {"submitted": 1590699432.0069268, "started": 1590699432.0070055, "finished": 1590705158.9679961}, {"loss": 0.02050858736038208}, null]
10 [[0, 0, 3], 5.0, {"submitted": 1590705158.9700077, "started": 1590705158.9700935, "finished": 1590713438.938362}, {"loss": 0.017227232456207275}, null]
11 [[0, 0, 4], 5.0, {"submitted": 1590713438.9397898, "started": 1590713438.939883, "finished": 1590723703.5933702}, {"loss": 0.014766216278076172}, null]
12 [[0, 0, 6], 5.0, {"submitted": 1590723703.5943506, "started": 1590723703.59448, "finished": 1590733113.6197166}, {"loss": 0.015586555004119873}, null]
13 [[0, 0, 4], 15.0, {"submitted": 1590733113.6213455, "started": 1590733113.6214201, "finished": 1590741998.865016}, {"loss": 0.014766216278076172}, null]
14 [[1, 0, 0], 5.0, {"submitted": 1590741998.8670812, "started": 1590741998.8671556, "finished": 1590763435.4490135}, {"loss": 0.018867909908294678}, null]
15 [[1, 0, 1], 5.0, {"submitted": 1590763435.4528549, "started": 1590763435.4529276, "finished": 1590775199.278057}, {"loss": 0.016406893730163574}, null]
16 [[1, 0, 2], 5.0, {"submitted": 1590775199.3114965, "started": 1590775199.3115723, "finished": 1590784880.9130924}, {"loss": 0.014766216278076172}, null]
17 [[1, 0, 2], 15.0, {"submitted": 1590784880.915058, "started": 1590784880.9151316, "finished": 1590794561.5784929}, {"loss": 0.01312553882598877}, null]
18 [[2, 0, 0], 15.0, {"submitted": 1590794561.6126454, "started": 1590794561.612723, "finished": 1590805377.7481515}, {"loss": 0.018867909908294678}, null]

```

```

19 [[2, 0, 1], 15.0, {"submitted": 1590805377.7819529, "started": 1590805377.7820294,
  "finished": 1590814221.8007407}, {"loss": 0.018867909908294678}, null]
20 [[2, 0, 2], 15.0, {"submitted": 1590814221.8026123, "started": 1590814221.8026924,
  "finished": 1590899290.2845824}, {"loss": 0.17719441652297974}, null]

```

**Listing E.2:** AutoML 1 results

```

1 [[0, 0, 0], {"autoencoder_loss_weight": 0.04440200785018703, "
  second_criterion_loss_weight": 0.002226288738906894}, {"model_based_pick": false}]
2 [[0, 0, 1], {"autoencoder_loss_weight": 0.00023692486074275592, "
  second_criterion_loss_weight": 1.4874627184862477}, {"model_based_pick": false}]
3 [[0, 0, 2], {"autoencoder_loss_weight": 0.2597034828956662, "
  second_criterion_loss_weight": 0.014542958251164593}, {"model_based_pick": false}]
4 [[0, 0, 3], {"autoencoder_loss_weight": 0.1681258608108659, "
  second_criterion_loss_weight": 0.5033954523563141}, {"model_based_pick": false}]
5 [[0, 0, 4], {"autoencoder_loss_weight": 0.0007048770668933648, "
  second_criterion_loss_weight": 0.0008535127114456933}, {"model_based_pick": false}]
6 [[0, 0, 5], {"autoencoder_loss_weight": 0.9711011168475416, "
  second_criterion_loss_weight": 1.2221575976656935}, {"model_based_pick": false}]
7 [[0, 0, 6], {"autoencoder_loss_weight": 0.0004454721297698147, "
  second_criterion_loss_weight": 0.0001364568192031105}, {"model_based_pick": true}]
8 [[0, 0, 7], {"autoencoder_loss_weight": 0.007903675440121686, "
  second_criterion_loss_weight": 0.0010001766933095053}, {"model_based_pick": true}]
9 [[0, 0, 8], {"autoencoder_loss_weight": 0.002080542429566222, "
  second_criterion_loss_weight": 0.0008621342506429814}, {"model_based_pick": true}]
10 [[1, 0, 0], {"autoencoder_loss_weight": 0.004953542798215495, "
  second_criterion_loss_weight": 0.0010917658397937785}, {"model_based_pick": true}]
11 [[1, 0, 1], {"autoencoder_loss_weight": 0.00428770904645574, "
  second_criterion_loss_weight": 0.00100020038631006866}, {"model_based_pick": true}]
12 [[1, 0, 2], {"autoencoder_loss_weight": 0.0007725825991106794, "
  second_criterion_loss_weight": 1.9737880400207493}, {"model_based_pick": false}]
13 [[2, 0, 0], {"autoencoder_loss_weight": 0.012207234825135407, "
  second_criterion_loss_weight": 0.004023934485903076}, {"model_based_pick": true}]
14 [[2, 0, 1], {"autoencoder_loss_weight": 1.244768067174401, "
  second_criterion_loss_weight": 0.00019176567892909577}, {"model_based_pick": false}]
15 [[2, 0, 2], {"autoencoder_loss_weight": 0.003489630114199584, "
  second_criterion_loss_weight": 0.0071063762928800156}, {"model_based_pick": true}]

```

**Listing E.3:** AutoML 2 config

## E. AutoML

---

```

1 [[0, 0, 0], 1.6666666666666665, {"submitted": 1590654862.1828277, "started":  
1590654862.1829093, "finished": 1590660613.4056783}, {"loss":  
0.017227232456207275}, null]  
2 [[0, 0, 1], 1.6666666666666665, {"submitted": 1590660613.4072785, "started":  
1590660613.4073539, "finished": 1590666341.1833882}, {"loss":  
0.021328985691070557}, null]  
3 [[0, 0, 2], 1.6666666666666665, {"submitted": 1590666341.184982, "started":  
1590666341.1850567, "finished": 1590672090.216009}, {"loss":  
0.037735819816589355}, null]  
4 [[0, 0, 3], 1.6666666666666665, {"submitted": 1590672090.218086, "started":  
1590672090.218161, "finished": 1590677828.6221488}, {"loss":  
0.02050858736038208}, null]  
5 [[0, 0, 4], 1.6666666666666665, {"submitted": 1590677828.6244273, "started":  
1590677828.6244752, "finished": 1590683537.040007}, {"loss":  
0.01394587755203247}, null]  
6 [[0, 0, 5], 1.6666666666666665, {"submitted": 1590683537.0422094, "started":  
1590683537.0422845, "finished": 1590689258.6220567}, {"loss":  
0.022149324417114258}, null]  
7 [[0, 0, 6], 1.6666666666666665, {"submitted": 1590689258.6579773, "started":  
1590689258.6580539, "finished": 1590694962.9890494}, {"loss":  
0.02379000186920166}, null]  
8 [[0, 0, 7], 1.6666666666666665, {"submitted": 1590694963.0245247, "started":  
1590694963.0245974, "finished": 1590700690.642876}, {"loss":  
0.015586555004119873}, null]  
9 [[0, 0, 8], 1.6666666666666665, {"submitted": 1590700690.678872, "started":  
1590700690.6789494, "finished": 1590706433.3123708}, {"loss":  
0.018047571182250977}, null]  
10 [[0, 0, 0], 5.0, {"submitted": 1590706433.3147688, "started": 1590706433.314844,  
"finished": 1590721799.4320598}, {"loss": 0.014766216278076172}, null]  
11 [[0, 0, 4], 5.0, {"submitted": 1590721799.433411, "started": 1590721799.4334853,  
"finished": 1590731775.1398897}, {"loss": 0.015586555004119873}, null]  
12 [[0, 0, 0], 15.0, {"submitted": 1590743488.4551537, "started": 1590743488.4552212,  
"finished": 1590755746.2579317}, {"loss": 0.017227232456207275}, null]  
13 [[1, 0, 0], 5.0, {"submitted": 1590755746.2932274, "started": 1590755746.2933583,  
"finished": 1590766718.065557}, {"loss": 0.017227232456207275}, null]  
14 [[1, 0, 1], 5.0, {"submitted": 1590766718.1017776, "started": 1590766718.101873,  
"finished": 1590782121.1684978}, {"loss": 0.014766216278076172}, null]  
15 [[1, 0, 2], 5.0, {"submitted": 1590782121.1704419, "started": 1590782121.170519,  
"finished": 1590790378.6628041}, {"loss": 0.01312553882598877}, null]  
16 [[1, 0, 2], 15.0, {"submitted": 1590790378.6646698, "started": 1590790378.6647408,  
"finished": 1590800933.9118834}, {"loss": 0.016406893730163574}, null]  
17 [[2, 0, 0], 15.0, {"submitted": 1590800933.9486916, "started": 1590800933.9487896,  
"finished": 1590812621.3249633}, {"loss": 0.016406893730163574}, null]  
18 [[2, 0, 1], 15.0, {"submitted": 1590812621.326876, "started": 1590812621.3269536,  
"finished": 1590897713.940387}, {"loss": 0.21985232830047607}, null]  
19 [[2, 0, 2], 15.0, {"submitted": 1590897713.9766533, "started": 1590897713.9767263,  
"finished": 1590908241.4061368}, {"loss": 0.015586555004119873}, null]

```

**Listing E.4:** AutoML 2 results

```

1 [[0, 0, 0], {"autoencoder_loss_weight": 0.005442239272699528,  
second_criterion_loss_weight": 0.05144872129916094}, {"model_based_pick": false  
}]

```

```

2 [[0, 0, 1], {"autoencoder_loss_weight": 0.7118297179664541, "second_criterion_loss_weight": 0.036180931992997864}, {"model_based_pick": false}]
3 [[0, 0, 2], {"autoencoder_loss_weight": 1.2062951967751687, "second_criterion_loss_weight": 1.3618690697161089}, {"model_based_pick": false}]
4 [[0, 0, 3], {"autoencoder_loss_weight": 0.010769922518937786, "second_criterion_loss_weight": 1.9130617913158043}, {"model_based_pick": false}]
5 [[0, 0, 4], {"autoencoder_loss_weight": 0.07315592439637968, "second_criterion_loss_weight": 0.32962956218197215}, {"model_based_pick": false}]
6 [[0, 0, 5], {"autoencoder_loss_weight": 0.003078088936255799, "second_criterion_loss_weight": 0.0002272944019336546}, {"model_based_pick": false}]
7 [[0, 0, 6], {"autoencoder_loss_weight": 0.004209808437643514, "second_criterion_loss_weight": 0.0724359020040274}, {"model_based_pick": false}]
8 [[0, 0, 7], {"autoencoder_loss_weight": 0.007500748061006794, "second_criterion_loss_weight": 1.3886761517038704}, {"model_based_pick": true}]
9 [[0, 0, 8], {"autoencoder_loss_weight": 0.00015705716256490278, "second_criterion_loss_weight": 0.00018305437951322067}, {"model_based_pick": false}]
10 [[1, 0, 0], {"autoencoder_loss_weight": 0.0019517838732991408, "second_criterion_loss_weight": 0.0002514371239735307}, {"model_based_pick": false}]
11 [[1, 0, 1], {"autoencoder_loss_weight": 0.014593829273141129, "second_criterion_loss_weight": 1.764666824662084}, {"model_based_pick": true}]
12 [[1, 0, 2], {"autoencoder_loss_weight": 0.012610868971815068, "second_criterion_loss_weight": 1.582692260108973}, {"model_based_pick": true}]
13 [[2, 0, 0], {"autoencoder_loss_weight": 0.004140498484276515, "second_criterion_loss_weight": 0.0006219336945559528}, {"model_based_pick": false}]
14 [[2, 0, 1], {"autoencoder_loss_weight": 1.1416047492940438, "second_criterion_loss_weight": 0.04940654085560021}, {"model_based_pick": true}]
15 [[2, 0, 2], {"autoencoder_loss_weight": 1.5266154213065302, "second_criterion_loss_weight": 0.4943894587693296}, {"model_based_pick": true}]

```

**Listing E.5:** AutoML 3 config

```

1 [[0, 0, 0], 1.6666666666666665, {"submitted": 1590765002.0917218, "started": 1590765002.0918014, "finished": 1590770737.3521612}, {"loss": 0.01968824863433838}, null]
2 [[0, 0, 1], 1.6666666666666665, {"submitted": 1590770737.3538744, "started": 1590770737.3539586, "finished": 1590776450.4094603}, {"loss": 0.02050858736038208}, null]
3 [[0, 0, 2], 1.6666666666666665, {"submitted": 1590776450.411038, "started": 1590776450.4111094, "finished": 1590782165.051875}, {"loss": 0.027071356773376465}, null]
4 [[0, 0, 3], 1.6666666666666665, {"submitted": 1590782165.0538504, "started": 1590782165.0539207, "finished": 1590787881.6047473}, {"loss": 0.018867909908294678}, null]

```

## E. AutoML

---

```
5 [[0, 0, 4], 1.6666666666666665, {"submitted": 1590787881.607019, "started":  
1590787881.607097, "finished": 1590793590.7224588}, {"loss":  
0.014766216278076172}, null]  
6 [[0, 0, 5], 1.6666666666666665, {"submitted": 1590793590.7243662, "started":  
1590793590.724443, "finished": 1590799308.1724072}, {"loss":  
0.02296966314315796}, null]  
7 [[0, 0, 6], 1.6666666666666665, {"submitted": 1590799308.1745405, "started":  
1590799308.1746123, "finished": 1590805025.53138}, {"loss":  
0.014766216278076172}, null]  
8 [[0, 0, 7], 1.6666666666666665, {"submitted": 1590805025.565349, "started":  
1590805025.5654283, "finished": 1590810738.595393}, {"loss":  
0.01968824863433838}, null]  
9 [[0, 0, 8], 1.6666666666666665, {"submitted": 1590810738.5975933, "started":  
1590810738.5976677, "finished": 1590816446.2123072}, {"loss":  
0.025430679321289062}, null]  
10 [[0, 0, 4], 5.0, {"submitted": 1590826392.4400935, "started": 1590826392.4401693,  
finished": 1590836353.2363422}, {"loss": 0.014766216278076172}, null]  
11 [[0, 0, 6], 5.0, {"submitted": 1590836353.2377238, "started": 1590836353.2379396,  
finished": 1590846020.6962156}, {"loss": 0.016406893730163574}, null]  
12 [[0, 0, 3], 15.0, {"submitted": 1590846020.69776, "started": 1590846020.6978347,  
finished": 1590855412.901872}, {"loss": 0.01312553882598877}, null]  
13 [[1, 0, 1], 5.0, {"submitted": 1590866534.0310533, "started": 1590866534.0311358,  
finished": 1590877328.5749404}, {"loss": 0.016406893730163574}, null]  
14 [[1, 0, 2], 5.0, {"submitted": 1590877328.6073856, "started": 1590877328.6074634,  
finished": 1590888962.6976082}, {"loss": 0.017227232456207275}, null]  
15 [[1, 0, 1], 15.0, {"submitted": 1590888962.7008479, "started": 1590888962.7009232,  
finished": 1590898327.5878718}, {"loss": 0.018047571182250977}, null]  
16 [[2, 0, 0], 15.0, {"submitted": 1590898327.590046, "started": 1590898327.5901172,  
finished": 1590907422.1790972}, {"loss": 0.017227232456207275}, null]  
17 [[2, 0, 1], 15.0, {"submitted": 1590907422.2143378, "started": 1590907422.2144222,  
finished": 1590925733.3175535}, {"loss": 0.01394587755203247}, null]  
18 [[2, 0, 2], 15.0, {"submitted": 1590925733.350404, "started": 1590925733.3504772,  
finished": 1590937086.457241}, {"loss": 0.01312553882598877}, null]
```

**Listing E.6:** AutoML 3 results

## F. TTAE Greiferdetection

```
1 Model: "encoder"
2 -----
3 Layer (type)          Output Shape         Param #
4 -----
5 encoder_input (InputLayer) [(None, 256, 192, 3)] 0
6 -----
7 dropout (Dropout)      (None, 256, 192, 3) 0
8 -----
9 conv2d (Conv2D)        (None, 256, 192, 8)   224
10 -----
11 conv2d_1 (Conv2D)     (None, 256, 192, 8)   584
12 -----
13 conv2d_2 (Conv2D)     (None, 128, 96, 8)    584
14 -----
15 batch_normalization (BatchNo (None, 128, 96, 8)) 32
16 -----
17 conv2d_3 (Conv2D)     (None, 128, 96, 16)   1168
18 -----
19 conv2d_4 (Conv2D)     (None, 64, 48, 16)    2320
20 -----
21 batch_normalization_1 (Batch (None, 64, 48, 16)) 64
22 -----
23 conv2d_5 (Conv2D)     (None, 64, 48, 32)   4640
24 -----
25 conv2d_6 (Conv2D)     (None, 64, 48, 32)   9248
26 -----
27 conv2d_7 (Conv2D)     (None, 32, 24, 32)   9248
28 -----
29 batch_normalization_2 (Batch (None, 32, 24, 32)) 128
30 -----
31 conv2d_8 (Conv2D)     (None, 32, 24, 64)   18496
32 -----
33 conv2d_9 (Conv2D)     (None, 16, 12, 64)   36928
34 -----
35 batch_normalization_3 (Batch (None, 16, 12, 64)) 256
36 -----
37 conv2d_10 (Conv2D)    (None, 16, 12, 64)  36928
38 -----
39 conv2d_11 (Conv2D)    (None, 8, 6, 64)    36928
40 -----
41 flatten (Flatten)    (None, 3072)       0
```

## F. TTAE Greiferdetection

```
42
43 dense (Dense)           (None, 512)      1573376
44 -----
45 dense_1 (Dense)         (None, 128)       65664
46 -----
47 dense_2 (Dense)         (None, 32)        4128
48 -----
49 code (Model)            (None, 15)        495
50 =====
51 Total params: 1,801,439
52 Trainable params: 1,801,199
53 Non-trainable params: 240
54 -----
55 Model: "decoder"
56 -----
57 Layer (type)            Output Shape       Param #
58 =====
59 decoder_input (InputLayer) [(None, 15)]    0
60 -----
61 dense_3 (Dense)          (None, 32)        512
62 -----
63 dense_4 (Dense)          (None, 128)       4224
64 -----
65 dense_5 (Dense)          (None, 512)       66048
66 -----
67 dense_6 (Dense)          (None, 3072)      1575936
68 -----
69 reshape (Reshape)        (None, 8, 6, 64)   0
70 -----
71 conv2d_transpose (Conv2DTr (None, 16, 12, 64) 36928
72 -----
73 batch_normalization_4 (Batch (None, 16, 12, 64) 256
74 -----
75 conv2d_transpose_1 (Conv2DTr (None, 16, 12, 64) 36928
76 -----
77 conv2d_transpose_2 (Conv2DTr (None, 32, 24, 64) 36928
78 -----
79 batch_normalization_5 (Batch (None, 32, 24, 64) 256
80 -----
81 conv2d_transpose_3 (Conv2DTr (None, 32, 24, 32) 18464
82 -----
83 conv2d_transpose_4 (Conv2DTr (None, 64, 48, 32) 9248
84 -----
85 conv2d_transpose_5 (Conv2DTr (None, 64, 48, 32) 9248
86 -----
87 batch_normalization_6 (Batch (None, 64, 48, 32) 128
88 -----
89 conv2d_transpose_6 (Conv2DTr (None, 64, 48, 16) 4624
90 -----
91 conv2d_transpose_7 (Conv2DTr (None, 128, 96, 16) 2320
92 -----
93 batch_normalization_7 (Batch (None, 128, 96, 16) 64
94 -----
```

```

95 conv2d_transpose_8 (Conv2DTr (None, 128, 96, 8) 1160
96 -----
97 conv2d_transpose_9 (Conv2DTr (None, 256, 192, 8) 584
98 -----
99 conv2d_transpose_10 (Conv2DT (None, 256, 192, 8) 584
100 -----
101 conv2d_12 (Conv2D)      (None, 256, 192, 3) 219
102 =====
103 Total params: 1,804,659
104 Trainable params: 1,804,307
105 Non-trainable params: 352
106 -----
107 Model: "second_criterion"
108 -----
109 Layer (type)          Output Shape         Param #
110 =====
111 second_criterion_input (Input [(None, 15)]) 0
112 -----
113 dense (Dense)         (None, 24)           384
114 -----
115 dense_1 (Dense)       (None, 8)            200
116 -----
117 dense_2 (Dense)       (None, 4)            36
118 -----
119 Total params: 620
120 Trainable params: 620
121 Non-trainable params: 0
122 -----

```

**Listing F.1:** TTAE modelsummary

Nr.	IoU t=0.5	IoU t=0.8
1	0.9597	0.5172
2	0.9554	0.5531
3	0.9612	0.5718

**Tabelle F.1.:** Einzelwerte Boxplot IoU TT-Greifer