

Aufgabenfokussierung auf Autoencoder und automatisches Transferlernen

Sebastian Hoch

MASTERARBEIT

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

Studiengang Informatik Master

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

30. Juni 2020

Durchgeführt bei der PSIORI GmbH

Betreuer

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

Hoch, Sebastian:

Aufgabenfokussierung auf Autoencoder und automatisches Transferlernen / Sebastian Hoch.

–

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

Hoch, Sebastian:

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

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

Vorwort

Die vorliegende Masterarbeit, habe ich als Abschlussarbeit meines Studiums der Informatik an der Hochschule Offenburg und meines Praktikums bei der PSI ORI GmbH geschrieben. Ziel war es, neue Werkzeuge zum Transferlernen bereitzustellen und zu evaluieren. 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 Fachentnisse im Bereich der Data Science konnte ich wichtige Einblicke in die Materie gewinnen.

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

Ich wünsche Ihnen viel Spaß beim Lesen dieser Arbeit.

Sebastian Hoch

Waldkirch, Juni 2020

Eidesstattliche Erklärung

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

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

Offenburg, 30. Juni 2020

Sebastian Hoch

Zusammenfassung

Aufgabenfokussierung auf Autoencoder und automatisches Transferlernen

Hohe Kosten bei der Annotation von Daten führen dazu, dass datensparsamere Wege zum Erstellen von Modellen gesucht werden. In dieser Arbeit wird ein Lösungsansatz untersucht, der ausgehend von fokussierten Repäsentationen, datensparsame Lösungen für verschiedene Aufgaben finden soll. Durch einen Multi-Task-Ansatz trägt das Finden einer Reräsentation gleichzeitig zum Lösen einer Aufgabe bei. Durch Ersetzung einer der Multi-Task können Wissentransfere 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

Inhaltsverzeichnis

1. Einleitung	5
2. Grundlagen	7
2.1. Autoencoder	7
2.2. Transferlernen	8
2.2.1. Tiefes Transferlernen	8
2.2.2. Halbüberwachtes Lernen	9
2.2.3. Multi-Task-Lernen	10
2.3. Automatisiertes maschinelles Lernen	11
2.3.1. Hyperparameter-Optimierung	11
2.3.2. Meta-Learning	12
2.3.3. Neural Architecture Search	12
2.3.4. Optimierungstechniken	12
2.4. Bibliotheken und Werkzeuge	14
2.5. Einordnung und bestehende Systeme	17
2.6. Datenverständnis	19
2.7. Datenvorbereitung	20
3. Experimente und Werkzeuge	23
3.1. Greifererkennung auf Autoencoder	23
3.2. Aufgabenfokussierung und Greifererkennung	24
3.2.1. Werkzeug: TaskFocusingOnAutoencoder	24
3.2.2. Ergebnis	27
3.3. Transferlernen und 'Greifer beladen'-Klassifikation	28
3.3.1. Werkzeug: TaskTransferOnAutoencoder	28
3.3.2. Ergebnis	29
3.4. AutoML ¹ und 'Greifer beladen'-Klassifikation	30
3.4.1. Werkzeug: AutoTaskTransferOnAutoencoder	30
3.4.2. Ergebnis	32
3.5. Datenmenge und 'Greifer beladen'-Klassifikation	32
4. Fazit	33
4.1. Zusammenfassung	33

¹automatisierte maschinelle Lernen

Inhaltsverzeichnis

4.2. Kritische Reflexion	34
4.3. Ausblick und weitere Arbeiten	34
4.3.1. Transfer auf Greiferdatensatz	34
4.3.2. Autocrane-Datensatz	34
4.3.3. Mehrfach-Aufgaben	34
4.3.4. Flexibilität der Werkzeuge	35
Abkürzungsverzeichnis	i
Tabellenverzeichnis	iii
Abbildungsverzeichnis	v
Quellcodeverzeichnis	vii
A. Anhang Basislinie Greifer	ix
B. Anhang Basislinie Baumstämme	xi
C. Anhang Greifererkennung	xiii
C.1. Greifererkennung auf Autoencoder	xiii
C.2. Mutli-Task Greifererkennung	xiii
D. Anhang Baumstämme im Greifer	xv
D.1. Transfer	xv

- Englisch->Deutsch
Fachwörter prüfen
- Quellen prüfen +
vollständigen
- Related Work ein-
ten

Todo list

Zusammenfassung auf Englisch	iv
Abstract ins Englische übersetzen	iv
Englisch->Deutsch Fachwörter prüfen	1
Quellen prüfen + Vervollständigen	1
Related Work einarbeiten	1
Convolutional Layer hier als Stand der Technik? CNN kommen von x haben sich durchgesetzt in 1,2,3,4,5	8
related work? hier / eigenes kapitel	9
related work MT-Learning	10
Bilder Einbettung auswählen	24
Grapple-MT Ergebnisse einfügen	27
Auto Ergebnisse	32
Datenmenge Ergebnisse	32
Ergebnisse Transfer Greifer	34
ref prüfen	34
nachkommastellen kürzen	xiii

1. Einleitung

Die PSIORI GmbH [**PSIORIGmbH.2020**] ist ein Projekt- und Beratungsunternehmen im Bereich der Digitalisierung und Data Science. In vielen Digitalisierungs- und Automatisierungsprojekten fallen eine Vielzahl von Aufgaben an, welche mittels künstlicher Intelligenz gelöst werden können. In der Regel wird dabei für jede Aufgabe, welche mittels künstlicher Intelligenz gelöst werden soll, eine eigene Annotation benötigt. Das Annotieren von Daten ist teuer. Zum Beispiel sollten in einem Projekt circa 80.000 Bilder mittels zehn verschiedenen Annotationen beschriftet werden, was zu Kosten von circa 240.000 Euro führt. Zusätzliche Kosten entstehen durch Personenaufwände beim Erstellen ähnlicher Modelle für Aufgaben in einer Domäne. Die Kosten solcher Data Science Projekte können also gesenkt werden, wenn weniger annotierte Daten genutzt werden müssen.

Ziel dieser Arbeit ist es, herauszufinden, ob es, 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 [**Shearer.2000**], 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, gefolgt von einer Beschreibung der bestehenden Systeme, auf welche mit einem Abschnitt über die Daten und ihrer Vorverarbeitung abgeschlossen wird.

2.1. Autoencoder

Autoencoder [**D.E.Rumelhart.1987**] 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 Code-layer und dem Decoder. In Abbildung 2.1 ist das Schema eines Autoencoders abgebildet. In der einfachsten Form besteht ein Autoencoder aus einer Eingabeschicht,

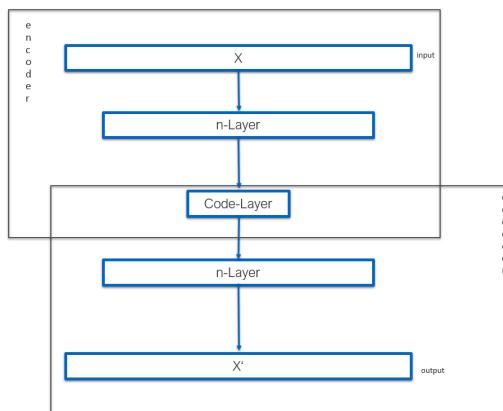


Abbildung 2.1: Schema Autoencoder

einer versteckten Schicht und einer Ausgabeschicht. Sie können aber auch mit mehreren Schichten, also mit 'tiefen Architekturen' genutzt werden. [**Hinton.2006**] Der

2. Grundlagen

Encoder lässt sich auch vereinfacht als die Funktion $F(x) = c$ und der Decoder als die Funktion $F(c) = x'$ darstellen, wobei $x \stackrel{!}{=} x'$ sein soll. Autoencoder gibt es in vielen verschiedenen Ausführungen. Dabei sind die meisten Typen von Autoencoder unvollständige Autoencoder. Die verborgene Schicht, das Codelayer enthält weniger Informationen als die Eingabe. Hierdurch wird die Dimensionsreduzierung erzwungen. Andere Aufgaben können Entrauschung mittels Denoising autoencoder [Vincent.2008] oder Generierung neuer Datenpunkte mittels Variational Autoencoder [Kingma.2019] sein. Contractive Autoencoder [Rifai.201] nutzen einen Regularisierer in der Zielfunktion, um das Modell zu zwingen eine Funktion zu lernen, die flexibler auf Variationen der Eingabewerte reagiert.

Für die Arbeit ist der Convolutional Autoencoder [Masci.2011] kurz CAE¹ interessant. Er ist ein Stacked Autoencoder, welcher Faltungsschichten (Convolutional Layer) [LeCun.1999] integriert.

Convolutional Layer
Stand der Tech-
NN kommen von
n sich durchge-
1,2,3,4,5

Schichtenweise Vortrainieren Im schichtenweise Vortrainieren werden einzelne Schichten eines neuronalen Netzwerkes vor dem eigentlichen Training trainiert, um die Leistung des gesamten Models zu erhöhen. [Bengio.2007] Bei (symmetrischen) Autoencodern werden dabei die zueinander symmetrischen Schichten des Encoders und Decoders miteinander trainiert. Die Zielgröße ist dabei die Rekonstruktion der Eingabedaten.

2.2. Transferlernen

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

2.2.1. Tiefes Transferlernen

Geprägt durch die Entwicklung hin zu tiefen neuronalen Netzwerken wurden Methoden zum Tiefen Transferlernen (deep transfer learning) vorgeschlagen. [Tirumala.2018]

¹Convolutional Autoencoder

ordnet diese in vier Kategorien ein. Für eine Einordnung von klassischen Methoden des Transferlernens eignet sich die Erhebung [**FuzhenZhuang.2019**].

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.

related work? hi
eigenes kapitel

Gegnerischbasiert Gegenerischbasiertes tiefes Transferlernen ist von dem Ansatz erzeugende gegnerische Netzwerke (Generative Adversarial Networks) [**IanJ.Goodfellow.2014**] 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

2. Grundlagen

werden. Im Gegensatz dazu sind bei dem Transferlernen die Datenverteilungen der Quell- und Zieldomäne oft unterschiedlich. [Chapelle.2010]

2.2.3. Multi-Task-Lernen

Werden mehrere Aufgaben parallel gelernt spricht man von dem Multi-Task-Lernen (MTL²). 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 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 Ausgabeziels genutzt.

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

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

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. [Thung.2018] [Pan.2010]

²Multi-Task-Lernen

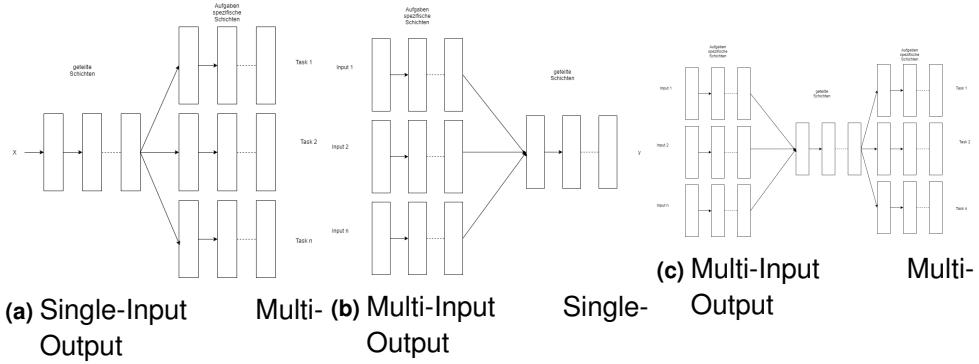


Abbildung 2.2: Multi-Task-Lernen: Ausprägungen

2.3. Automatisiertes maschinelles Lernen

Das automatisierte maschinelle Lernen kurz AutoML 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 halbautomatisch 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 NAS³ und Hyperparameter-Optimierung kurz HPO⁴. [Hutter.2019]

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 [Kohavi.1995] 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, zusätz-

³Neural Architecture Search

⁴Hyperparameter-Optimierung

2. Grundlagen

lich 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 Optimierungs-techniken der automatischen-HPO erläutert. [Feurer.2019]

2.3.2. Meta-Learning

Unter [JoaquinVanschoren.2018] 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 [Elsken.2019] kann eine Übersicht über die Neural Architecture Search gefunden werden.

2.3.4. Optimierungstechniken

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

Rastersuche

Die Rastersuche ist eine modellunabhängige Optimierungsstrategie. Es werden Parameterkombinationen definiert und anschließend Modelle, ausgehend von den Kombinationen, erstellt und evaluiert. Diese Technik hat einige Schwächen, so müssen Parameterkombinationen definiert werden, jedes Modell muss trainiert werden und falls die optimale Konfiguration nicht enthalten ist, wird sie nie gefunden. [Michelucci.2018]

Zufallssuche

Die Zufallssuche ähnelt der Rastersuche. Sie unterscheidet sich dahingehend, dass die Parameterkombinationen nicht mehr definiert werden müssen. Es werden zufällige Stichprobenkonfigurationen aus dem (definierten) Parameterraum gezogen und evaluiert. In [BergstraJamesandYoshuaBengio..2012] 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. [Frazier.08.07.2018]

Hyperband

Hyperband [Li.2017] erweitert Successive Halving [Jamieson.27.02.2015]. Successive Halving kurz SH⁵ weist einer Reihe von Hyperparameter-Konfigurationen

⁵Successive Halving

2. Grundlagen

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 normalen Gittersuche.

BOHB

BOHB [StefanFalkner.2018] ist ein Ansatz, der Bayesian optimization und Hyperband kombiniert. Dabei wird Bayesian optimization zur Auswahl von Konfigurationen herangezogen und Hyperband bestimmt wieviele Konfigurationen, mit welchem Budget ausgeführt werden sollen. Anschließend werden die Konfigurationen mittels Successive Halving ausgeführt. Unter <https://github.com/automl/HpBandSter> kann eine Implementierung des 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 Cnvrg [[cnvrg.io.](#)] genutzt. Cnvrg.io ist eine 'full-stack' Data Science Platform, welche Werkzeuge für die Erstellung, Verwaltung, Bereitstellung und Automatisierung von maschinellem Lernen bereitstellt. Cnvrg erlaubt es Arbeitsbereiche mittels Container zu erstellen. Die Container können dabei auf Maschinen in Azure [Micorsoft.2020] zugreifen. Für die Experimente wurde ein vorgefertigter Container mit einer tesla-k80 [[Nvidia.2020](#)], fünf CPUs und 49 GB Arbeitsspeicher genutzt.

Für die Entwicklung wurden Python [[PythonSoftwareFoundation.2020](#)], Jupyter Notebooks [[ProjectJupyter.](#)] und das Framework Tensorflow [[MartinAbadi.2015](#)]

genutzt. Die wichtigsten Bibliotheken für die Arbeit sind Keras [Chollet.2015] , Numpy [Oliphant.2006] , Matplotlib [Hunter.2007] , scikit-learn [Pedregosa.2011] , ConfigSpace [Lindauer.16.08.2019] , Bayesian Optimization and Hyperband [StefanFalkner.2018].

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 [PSIORIGmbH.2019]. 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 Konstruktorgumente 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 Stacke-

2. Grundlagen

dAutoencoder 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 [Bengio.2007] beschrieben vortrainiert. Die Auswahl der Schichten erfolgt wieder in den abgeleiteten Klassen. In der *fit(..)*-Methode wird nach einigen Prüfungen die Methode *ffit(..)* [Chollet.2015] des Kerasmodels aufgerufen. In Abbildung 2.3 ist das Klassendiagramm mit den öffentlichen Methoden des ConvolutionalAutoencoder dargestellt.

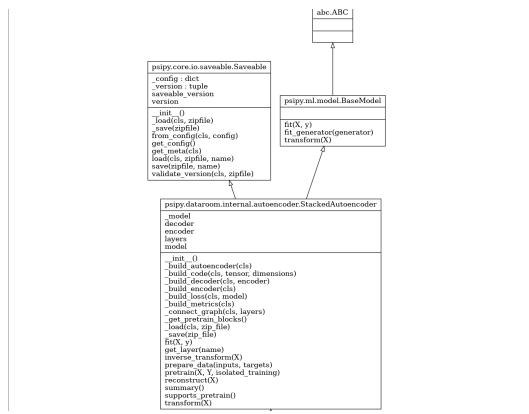


Abbildung 2.3: Klassendiagramm ConvolutionalAutoencoder

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

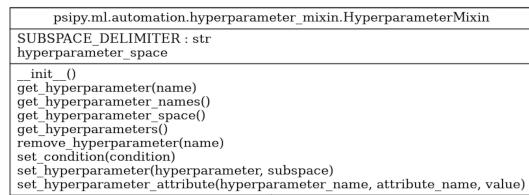


Abbildung 2.4: Klassendiagramm Hyperparamettermixin

2.5. Einordnung und bestehende Systeme

Die Bilddaten und Aufgabenstellungen der neuronalen Netzwerke sind in die Problemstellungen des Autocrane-Projekts von PSIORI einzuordnen. Es sind 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 Schlitzen 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. Die Baumstämme werden mittels LKW angeliefert und müssen nach vorgegebenen Regeln (z. B. Ausrichtung, freier Lagerplatz) als Holzstapel gelagert werden. Der Fülltrichter muss mit Holz aus den Holzstapeln befüllt werden. Der Fülltrichter muss mit Holz aus einem LKW befüllt werden. Es ergeben sich 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. [PSIORIGmbH.2020]



Abbildung 2.5: Rundlaufkran (Foto: ANDRITZ)

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



(a) Greifer mit Rahmen

(b) Greifer mit Baumstämmen

Abbildung 2.6: Greifer

PSIORI hat die Aufgabe mittels neuronalem Netzwerk gelöst. Dabei wurde auf die Technik des Single Shot MultiBox Detector (SSD) [**Liu.08.12.2015**] zurückgegriffen. Die Vorhersagegenauigkeit dieses Modells wird als Basislinie und Vergleichswert genutzt. Dabei ist die ausschlaggebende Metrik die Intersection over Union kurz IoU⁶ 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.

⁶Intersection over Union

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 durchgeführten Versuche genutzt. Das Modell erreicht auf den Validation-Daten eine Accuracy von 0.9828%. Im Anhang B ist die vollständige Basislinie dargestellt.

2.6. Datenverständnis

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



Abbildung 2.7: Bildqualität

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

Generierter Greiferdatensatz Erfahrungsgemäß lernen Autoencoder Lichtverhältnisse in Bildern relativ gut. Um für erste Versuche den Fokus von den Lichtverhältnissen 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 Cnvrgr-Plattform in einen versionierbaren Datenspeicher geladen. In Tabelle D.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
Greifer	3.279	703	704	4.686
Baumstämme j/n	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 maximalen Anzahl an Bildern, Bilder als Numparray zurück. Die Klassen GrappleDataLoader 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 [Chollet.2015] adressiert. Sie erlauben es Bilder stapelweise bereitzustellen. Zusätzlich werden bei den Imagedatageneratoren direkt die Zielgrößen bereitgestellt. Das Modul implementiert jeweils für den Baumstamm Datensatz und den Greiferdatensatz eine Klasse zum Bereitstellen von ImageDatageneratoren.

Da die Modelle SIMO⁷-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 Werkzeuge mit zweitem Kriterium benötigen Generatoren, die Einträge der Stapel der Form $X, [X, Y]$ erzeugen. Die Zielgröße hat sich zu einer Liste mit zwei Größen verändert. Die erste Zielgröße entspricht den Eingangsdaten, sie werden für den Decoder-Ausgang genutzt. Die zweite Zielgröße wird für den zweiten Ausgang

⁷Single-Input Multi-Output

2. Grundlagen

genutzt. Sie entspricht der Zielgröße eines normalen ImageDataGeneratoren. Das Ganze wird mit Hilfe der Klasse tensorflow.keras.utils.Sequence erreicht. Ihr wird im Konstruktor ein ImageDataGenerator übergeben. Bei der Bereitstellung eines Elementes wird der Rückgabewert des Generators angepasst. Die entscheidenden Codezeilen sind in Listing 2.2 dargestellt.

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

Listing 2.2: Aufbereitung Generatorergebnis in Python

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

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

3. Experimente und Werkzeuge

Als großer Kostenfaktor in DataScience-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.

3.1. Greifererkennung auf Autoencoder

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 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 dass der sich stark verändernde

3. Experimente und Werkzeuge

Hintergrund eine Herausforderung ist. In erster Linie wurden die Lichtverhältnisse gelernt. Der Greifer verschwindet nahezu in allen Rekonstruktionen.

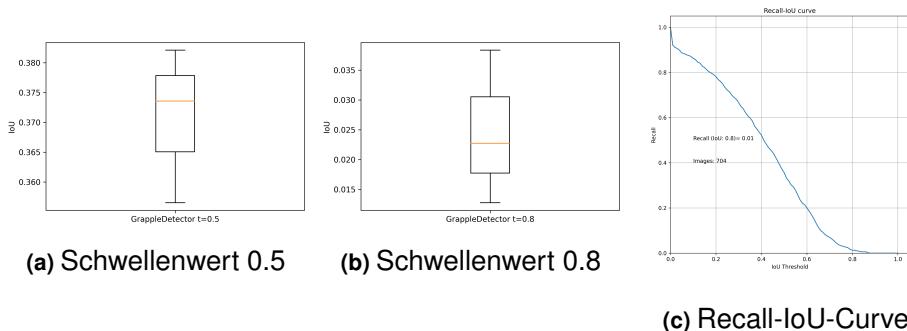
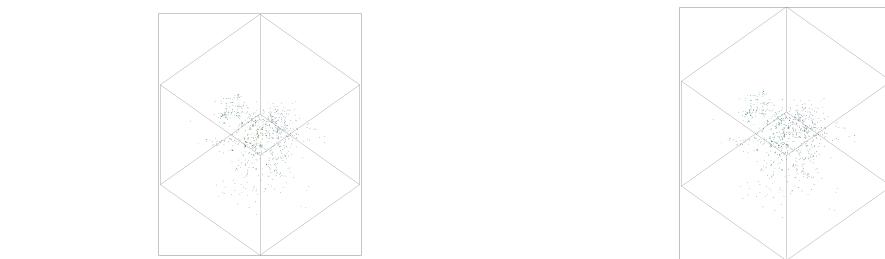


Abbildung 3.1: Ergebnis Regression auf Autoencoder



(a) Embedding mit y-Position des Greifers (b) Embedding mit x-Position des Greifers

Abbildung 3.2: Embedding 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 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

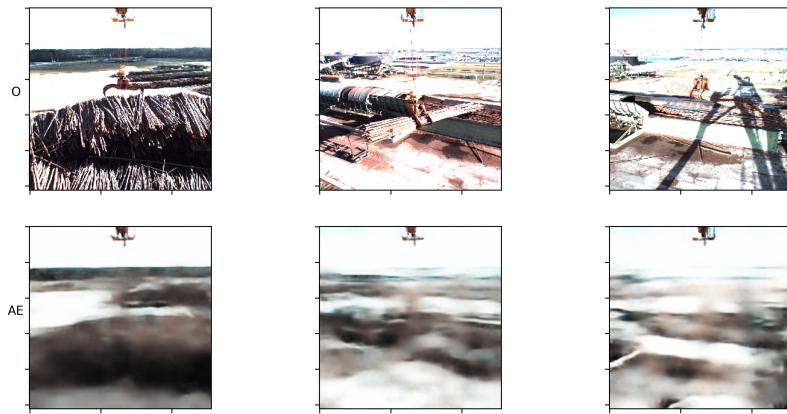


Abbildung 3.3: Rekonstruktion Autoencoder

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 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 Erstellen 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(..)`

3. Experimente und Werkzeuge

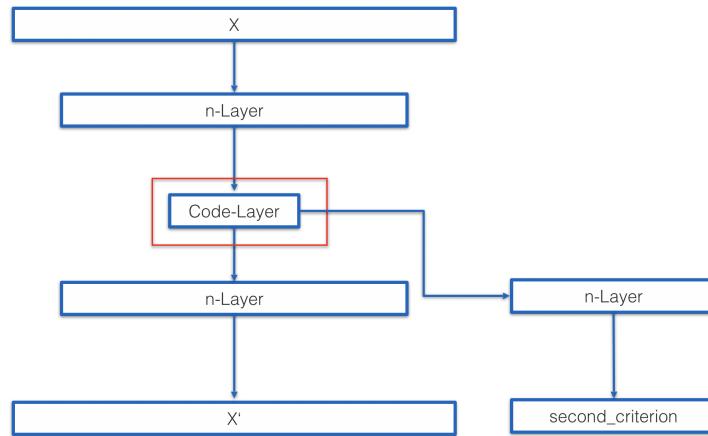


Abbildung 3.4: Schema TaskFocusingOnAutoencoder

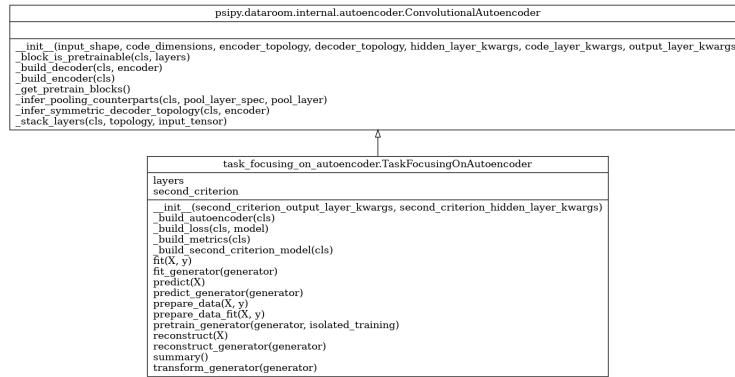


Abbildung 3.5: Klassendiagramm TaskFocusingOnAutoencoder

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: Beispelaufruf Pretrain in Python

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

```

1 history = 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: Beispelaufruf Fit in Python

3.2.2. Ergebnis

In Abbildung xyz

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

Grapple-MT Erg
einfügen

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 das Netzwerk, welches in Experiment 3.2 erstellt wurde genutzt. In der Zieldomäne werden die Architektur und die Gewichte des Autoencoder weiterverwendet. Der zweite Task wird durch die Aufgabe der Klassifikation ob sich Baumstämme im Greifer befinden ersetzt. Abbildung 3.6 zeigt das Schema des Ansatzes. Zur einfachen Anwendung des Ansatzes wurde ein Python-Modul mit der

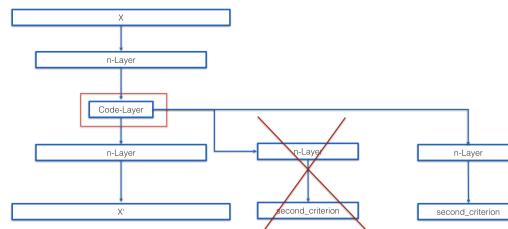


Abbildung 3.6: Schema TaskTransferOnAutoencoder

Klasse TaskTransferOnAutoencoder kurz TTAE¹ erstellt. Abbildung 3.7 zeigt das zugehörige Klassendiagramm. Als Basis wird ein TFAE² 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 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.

¹TaskTransferOnAutoencoder

²TaskFocusingOnAutoencoder

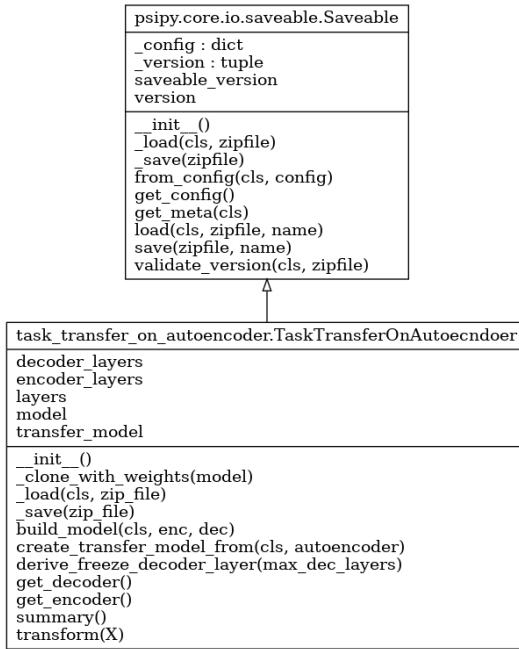


Abbildung 3.7: Klassendiagramm TaskTransferOnAutoencoder

```

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}))
16
  
```

Listing 3.4: Beispiel TransferSecondCriterionAutoencoder in Python

3.3.2. Ergebnis

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 3.8 sind die Ergebnisse des Versuchs dargestellt. Die Unterabbildung 3.8b

3. Experimente und Werkzeuge

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 noch nicht 100% korrekt ist.

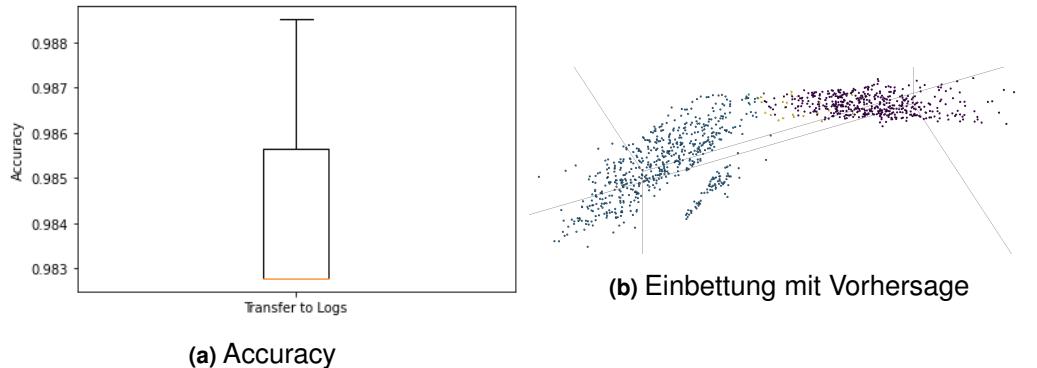


Abbildung 3.8: Ergebnis Transfer

3.4. AutoML und 'Greifer beladen'-Klassifikation

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 gering zu halten, wird ein neues Modul erstellt, welches auf AutoML zur Hyperparameteroptimierung zurückgreift.

3.4.1. Werkzeug: AutoTaskTransferOnAutoencoder

Das neue Modul integriert den Ansatz in der Klasse AutoTaskTransferOnAutoencoder, kurz AutoTTAE³. 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 zurückgegriffen. Nach dem Instanziieren der Klasse können weitere sinnvolle Hyperpara-

³ AutoTaskTransferOnAutoencoder

meter gesetzt werden. In Abbildung 3.9 ist das zugehörige Klassendiagramm abgebildet. In Listing 3.5 ist eine einfache Implementierung eines AutoTaskTransfe-

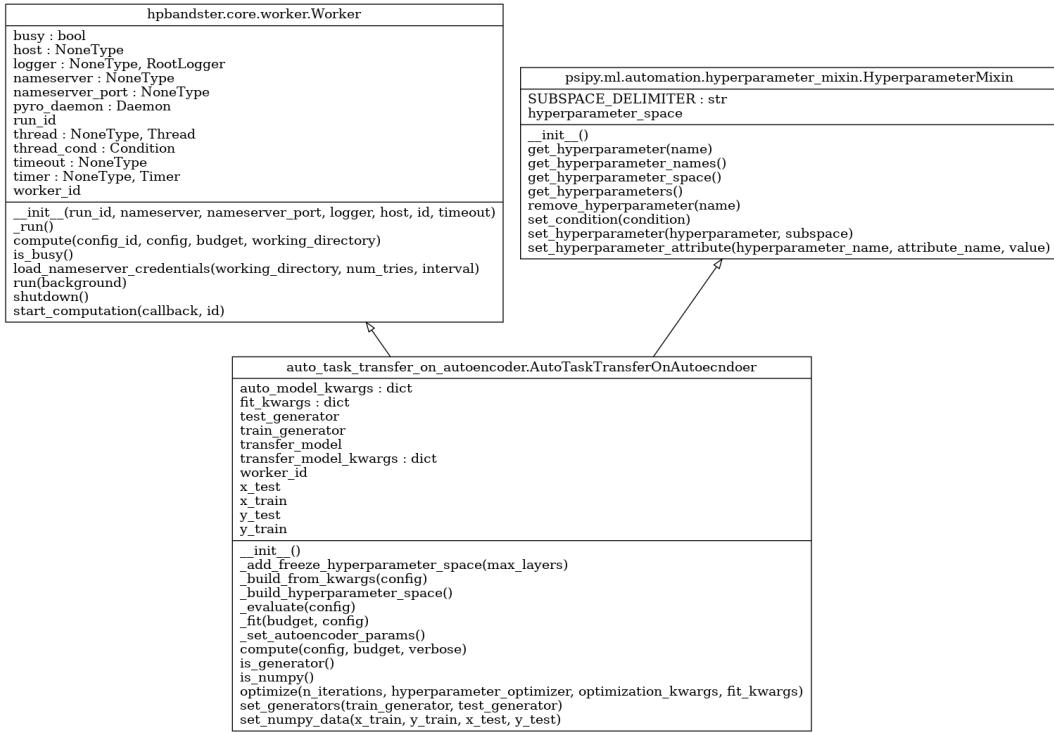


Abbildung 3.9: Klassendiagramm AutoTaskTransferOnAutoencoder

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 Gegensatz zur bisherigen Vorgehensweise werden die Trainings und Testdaten mit der Methode `set_generators(..)` oder `set_numpydata(..)` übergeben. Ziel ist es dabei die Komplexität der Methode `optimize(..)` zu reduzieren. Durch den Aufruf von `optimize(..)`, wird der Optimierungsvorgang gestartet. Die Parameter sind dabei die Anzahl an Iterationen, die Optimierungsstrategie und ein Wörterbuch, welches zusätzliche Einstellungen für die einzelnen Optimierer enthalten kann. In jeder Iteration der Optimierung wird ein neuer 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_kwarg = {'activation': 'relu'},
6 second_criterion_output_layer_kwarg = {'activation': 'softmax'},
7 second_criterion_metrics = {'second_criterion': 'accuracy'}
8 )

```

3. Experimente und Werkzeuge

```
9  
10 tscm.set_generators(train_datagenerator,test_datagenerator)  
11  
12  
13 best_config, history = tscm.optimize(3  
14 , 'RandomSearch'  
15 ,optimization_kwarg s = optimization_kwarg s)
```

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.

ergebnisse

3.4.2. Ergebnis

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

enige Ergebnisse

3.5. Datenmenge und 'Greifer beladen'-Klassifikation

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

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

4. Fazit

Zum Abschluss der Arbeit werden die Inhalte zusammengefasst, es folgt eine kritische Auseinandersetzung mit verschiedenen Ergebnissen der Abschlussarbeit und es wird ein ausführlicher Blick auf mögliche Erweiterungen und weitere geleistete Arbeiten gelegt.

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. Daraufhin wurde ein Multi-Task-Ansatz zum gleichzeitigen Fokussieren und Lösen einer Regressionsaufgabe vorgestellt und evaluiert. Insbesondere die gefundene Repräsentation hat dabei deutliche die aktuelle Domäne widergespiegelt. Darauf aufbauend wurde ein Ansatz des modellbasierten Transferlernens 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

andere Domäne testen Greifer ist sehr klar Daten

4.3. Ausblick und weitere Arbeiten

In diesem letzten Unterkapitel werden weitere (vorläufige) Ergebnisse dargestellt und da die untersuchten Methoden

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.

sse Transfer

4.3.2. Autocrane-Datensatz

Im Rahmen der Arbeit wurde ein Datensatz erarbeitet und Projektpartner zur Verfügung gestellt. Eine spätere Veröffentlichung des Datensatzes unter der Adresse psiori.com ist geplant. Der Datensatz soll zur allgemeinen akademischen und pädagogische Zwecke genutzt werden dürfen. Alternativ kann Zugang zu dem Datensatz über die E-Mail-Adresse info@psiori.com erfragt werden.

4.3.3. Mehrfach-Aufgaben

Mehrfach-Aufgaben-Lernen beschränkt sich nicht auf zwei Aufgaben. Die gezeigte Lösung kann um weitere Aufgaben erweitert werden. In Abbildung 4.1 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 ?? wurde gezeigt, dass BOHB bei der Gewichtung helfen kann. Dieser

erweiterte Ansatz soll in einem Anschlussprojekt für die Praxistauglichkeit untersucht werden. Der Aufwand, das bisher entwickelte Python-Module zu erweitern wird sich dabei in Grenzen halten. Insbesondere müssen im Konstruktor Erweiterungen zum Erstellen der Modelle der weiteren Aufgaben getätigt werden. An anderen Stellen wie z. B. die Methode zum Trainieren des Gesamtmodells wird keine Erweiterung notwendig sein. Es wird schon mit Listen gearbeitet.

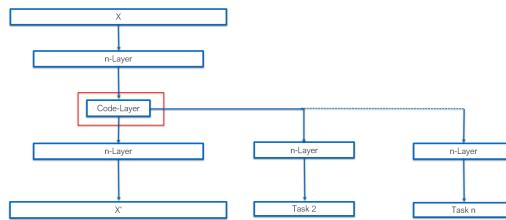


Abbildung 4.1: Multi-Task-Ansatz

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.

Abkürzungsverzeichnis

AutoML	automatisierte maschinelle Lernen	v
TFAE	TaskFocusingOnAutoencoder	28
TTAE	TaskTransferOnAutoencoder	28
AutoTTAE	AutoTaskTransferOnAutoencoder	30
NAS	Neural Architecture Search	11
HPO	Hyperparameter-Optimierung	11
CAE	Convolutional Autoencoder	8
MTL	Multi-Task-Lernen	10
SIMO	Single-Input Multi-Output	21
SH	Successive Halving	13
IoU	Intersection over Union	18

Tabellenverzeichnis

2.1. Datenaufteilung - Train Test Validation	21
C.1. Einzelwerte Boxplot IoU Greifererkennung auf Autoencoder	xiii
C.2. Einzelwerte Boxplot IoU MT-Greifer	xiii
D.1. Datenaufteilung - Train Test Validation	xv

Abbildungsverzeichnis

2.1.	Schema Autoencoder	7
2.2.	Multi-Task-Lernen: Ausprägungen	11
2.3.	Klassendiagramm ConvolutionalAutoencoder	16
2.4.	Klassendiagramm Hyperparametermixin	16
2.5.	Rundlaufkran	17
2.6.	Greifer	18
2.7.	Bildqualität	19
3.1.	Ergebnis Regression auf Autoencoder	24
3.2.	Embedding Autoencoder	24
3.3.	Rekonstruktion Autoencoder	25
3.4.	Schema TaskFocusingOnAutoencoder	26
3.5.	Klassendiagramm TaskFocusingOnAutoencoder	26
3.6.	Schema TaskTransferOnAutoencoder	28
3.7.	Klassendiagramm TaskTransferOnAutoencoder	29
3.8.	Ergebnis Transfer	30
3.9.	Klassendiagramm AutoTaskTransferOnAutoencoder	31
4.1.	Ausblick Multi-Task-Ansatz	35
A.1.	RecallIoUGrappleBaseline	ix
B.1.	Baumstammklassifikation Basislinie ROC	xi
B.2.	Konfusionsmatrix Baumstammklassifikation Basislinie	xi
B.3.	Klassifikationsreport Baumstammklassifikation Basislinie	xi

Listings

2.1.	Erweiterung zum Speichern eines Tensorflow Models	15
2.2.	Aufbereitung Generatorergebnis in Python	22
3.1.	Beispiel Erstellung ConvolutionalSecondCriterionAutoencoder in Python	26
3.2.	Beispieldruck Pretrain in Python	27
3.3.	Beispieldruck Fit in Python	27
3.4.	Beispiel TransferSecondCriterionAutoencoder in Python	29
3.5.	Beispiel AutoTransferSecondCriterionAutoencoder in Python	31

A. Anhang Basislinie Greifer

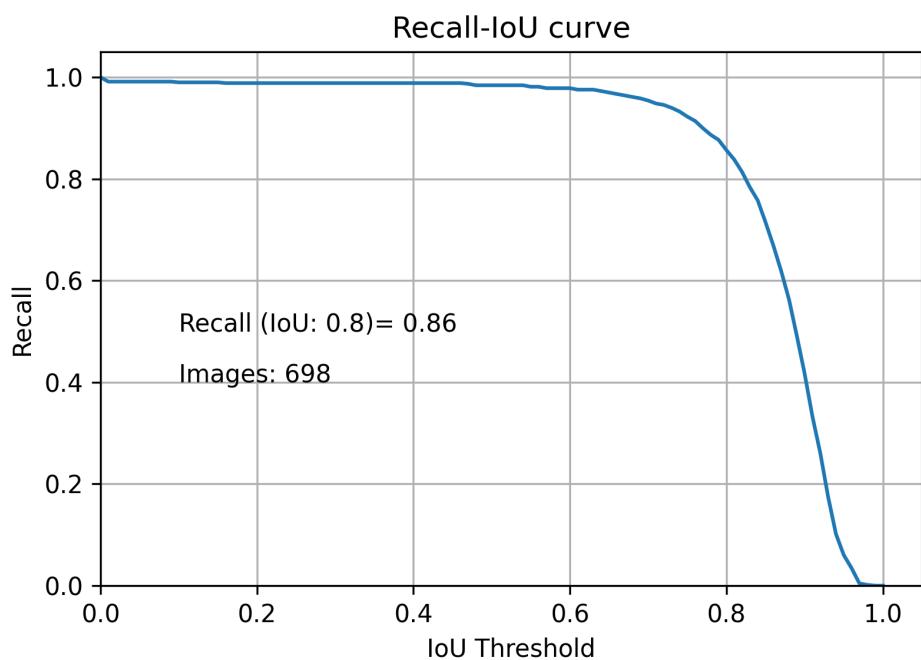


Abbildung A.1: Basislinie Greifererkennung

x

B. Anhang Basislinie Baumstämme

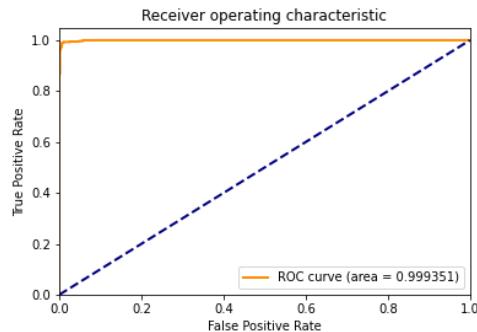


Abbildung B.1: receiver operating characteristic

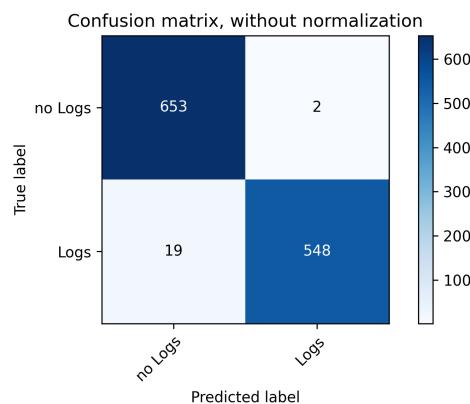


Abbildung B.2: Konfusionsmatrix

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

Abbildung B.3: Klassifikationsreport

C. Anhang Greifererkennung

C.1. Greifererkennung auf Autoencoder

Versuch	IoU t=0.5	IoU t=0.8
1	0.3565340909090909	0.01278409090909091
2	0.37357954545454547	0.022727272727272728
3	0.3821022727272727	0.03835227272727273

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

C.2. Mutli-Task Greifererkennung

Nr.	IoU t=0.5	IoU t=0.8
1	0.8638392857142857	0.22544642857142858
2	0.9899553571428571	0.7555803571428571
3	0.8571428571428571	0.21316964285714285
4	0.9966517857142857	0.8258928571428571
5	0.9944196428571429	0.8258928571428571
6	0.8816964285714286	0.29017857142857145
7	0.9866071428571429	0.6662946428571429

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

D. Anhang Baumstämme im Greifer

D.1. Transfer

Transfer Logs 9749	0.9885	0.9827	0.9827
Transfer Logs 200 gewichtet			
Transfer Logs 2000 gewichteter			
Transfer Logs 9749 gewichtet			

Tabelle D.1.: Datenaufteilung - Train Test Validation