

Automatisches Transferlernen mittels Autoencodern

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

XX.XX.2020

Durchgeführt bei der PSIORI GmbH

Betreuer

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

Hoch, Sebastian:

Automatisches Transferlernen mittels Autoencodern / Sebastian Hoch. –
MASTERARBEIT, Offenburg: Hochschule für Technik, Wirtschaft und Medien Offenburg,
2020. 31 Seiten.

Hoch, Sebastian:

Automatic transfer learning using autoencoders / Sebastian Hoch. –
MASTER THESIS, Offenburg: Offenburg University, 2020. 31 pages.

Vorwort

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

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

Während meiner Arbeiten waren meine Betreuer, Prof. Dr.-Ing. Janis Keuper und Dr. rer. nat. Sascha Lange, und der Begleiter meines Praktikums, Flemming Biegert immer erreichbar. Sie beantworteten meine entwicklungstechnischen Fragen und gaben wertvollen Input für die methodische Vorgehensweise, 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, XX.XX.2020

Sebastian Hoch

Zusammenfassung

Automatisches Transferlernen mittels Autoencodern

Im Rahmen dieser Arbeit wurden drei Werkzeuge erstellt um Merkmalsextraktion, Transferlernen und AutoML zu kombinieren. Das erste Werkzeug gleicht einen Schwäche eines Autoencoders aus. Beim Training eines Autoencoders wird die Rekonstruktion, also der Output des Modelles und nicht direkt die Einbettung als Bewertungskriterium herangezogen. Um diese Schwäche zu kompensieren wurde der SCAE erstellt. Dieses Werkzeug ist ein Autoencoder mit weiterem Ausgang. Die Datenrepräsentation wird durch ein zweites Kriterium gestärkt. Das zweite Werkzeug nutzt die Datenrepäsentation um einen Transferlearning Task durchzuführen. Das zweite Kriterium wird durch ein neues Kriterium ersetzt. Als drittes Werkzeug wurde der TCSCAE um funktionen des AutoML erweitert. Die besten Hyperparameter werden automatisch gefunden. Die Werkzeuge wurden anhand von echten Datensätzen getestet und validiert. Dabei hat sich gezeigt, dass mit den Werkzeugen eine ähnlich gute Leistung wie auf dem herkömmlichen Weg erreicht werden kann und das durch das Transferlern sogar aufwand reduziert werden kann.

Beschreibugn der Tools weniger konkret?

Abstract

Automatic transfer learning using autoencoders

Englische Version von Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

1 Einleitung	3
1.1 Zielsetzung	3
1.2 Vorgehen	4
2 Grundlagen	5
2.1 Convolutional Autoencoder	5
2.1.1 Autoencoder	5
2.1.2 Stacked Convolutional Autoencoder	5
2.1.3 Layerwise Pretrain	5
2.2 Transfer Learning	6
2.3 AutoML	6
2.3.1 RandomSearch	6
2.3.2 HyperBand	6
2.3.3 BOHB	6
2.4 Bibliotheken und Werkzeuge	6
2.5 Einordnung und bestehende Systeme	9
2.6 Datenverständnis	11
2.7 Datenvorbereitung	13
3 Werkzeuge	17
3.1 ConvolutionalSecondCriterionAutoencoder	17
3.2 TransferSecondCriterionAutoencoder	19
3.3 AutoTransferSecondCriterionAutoencoder	21
4 Experimente	27
4.1 Versuchsaufbau	27
4.1.1 Psipy-Modul	27
4.2 Modellierung	27
4.2.1 todo: Greifer	27
4.2.2 todo: Transferlearning	27
4.2.3 todo: Holz	27
4.3 Evaluierung	27
4.3.1 todo: Greifer	27
4.3.2 todo: Holz	27

Inhaltsverzeichnis

5 Fazit	31
5.1 Zusammenfassung	31
5.2 Kritische Reflexion	31
5.3 Ausblick	31
Abkürzungsverzeichnis	i
Tabellenverzeichnis	iii
Abbildungsverzeichnis	v
Quellcodeverzeichnis	vii
Literatur	ix

Todo list

Quelle Autoencoder pretrain	8
Quelle fit Keras	8
Vorgehen auch so beschreiben / ansonsten Kapitel Einleitung anpassen	11
Im Kapitel Bestehdens System erwähnen / diesen Teil in das andere Kapitel verschieben?	11
finale Zahl setzen	13
stimmt das am Ende noch?	13
weight durch Verhältnis w1/w2 anpassen.	23
Worker Budget berücksichtingen / notieren	23
Geheimhaltung: PSIPy-Code nicht zeigen Zeichnungen sind ok, Bilder rücksprache halten, Ansosnten offen.	

1. Einleitung

Das finden geeigneter Repäsentationen von Daten ist ein bekanntes Problem im Feld der DataScience. Dabei ist bekannt, dass Datenrepräsentation maßgeblich für die Leistungsfähigkeit von Maschinellem Lernen verantwortlich ist. Insbesondere Hochdimensionale Daten, wie Bilder, haben mit dem Fluch der Dimensionalität[] zu kämpfen. Der Einsatz von Autoencoder[] erlaubt es komprimierte Datenrepräsentationen für Bilder zu finden. Sind dabei ein unüberwachtes Lernverfahren, brauchen also keine beschrifteten Daten. In vielen Anwendungsfällen ist es teuer oder schwierig beschriftete Daten für neue Anwendungsfälle zu beschaffen. Transferlernen adressiert diese Problematik. Es hat das Ziel, gute Modelle in einer neuen Domäne basierend von Wissen in einer anderen Domäne zu erstellen.

Die PSIORI GmbH [PS20] ist ein Unternehmen, welches Projekte im Bereich der künstlichen Intelligenz für Kunden durchführt. PSIORI kann dabei auf einen mehrjährigen Erfahrungsschatz im Bereich des Maschinellem Lernen zurückgreifen und setzt dabei häufig Autoencoder und das Transferlernen zum Lösen von Aufgabenstellungen ein. Oft ist der Ansatz des Transferlernenes kostengünstiger um ein neues Modell in einer bestehenden Domäne zu erstellen.

(AutoMI auch noch Motivieren)

1.1 Zielsetzung

Ziel dieser Arbeit ist es, Werkzeuge zur Kombination der Ansatzes des Autoencoders und des Transferlernenes zu erstellen und an Hand von einem echten Datensatz zu evaluieren.

SecondCriterionAutoencoder Der SecondCriterionAutoencoder (SCAE) ist ein Werkzeug welches beim Erstellen eines Autoencoder mit weiterer Verlustfunktion unterstützt.

TransferSecondCriterionAutoencoder Der TransferSecondCriterionAutoencoder (TSCAE) ersetzt die Verlustfunktion eines SCAE.

AutoTransferSecondCriterionAutoencoder Der AutoTransferSecondCriterionAutoencoder (AutoTSCAE) erweitert den TSCAE um eine automatische Hyperparametersuche.

1.2 Vorgehen

WERKEZUGE erstellen Mnist + Emnist 1. 2.

Modelle Iterativ verbessern 1. fit 2. fitgenerator 3. 80.000 3. AutoML

[MNIST] [EMNIST]

2. Grundlagen

2.1 Convolutional Autoencoder

2.1.1 Autoencoder

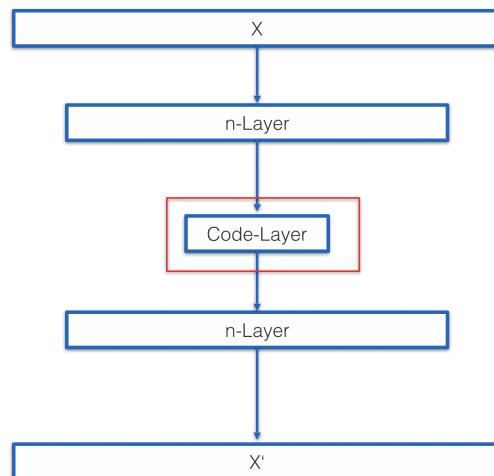


Abbildung 2.1: Schema Autoencoder

- NN

2.1.2 Stacked Convolutional Autoencoder

, Convolutions nur per Quelle erläutern

2.1.3 Layerwise Pretrain

Warum pretrain?: <https://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf> (Warum machen das andere heute nicht mehr? viele cnn haben

vanishing gradients problem über relu,... gelösst) (weitere literatur: möglicherweise Geoffrey E. Hinton)

Stacked Convolutional Auto-Encoders
Stacked Convolutional Auto-Encoders for
Hierarchical Feature Extraction
Jonathan Masci, Ueli Meier, Dan Ciresan, and Ju-
rgen Schmidhuber
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
Lugano, Switzerland
jonathan,ueli,dan,juergen@idsia.ch

- layerwise pretrain Greedy Layer-Wise Training of Deep Networks

2.2 Transfer Learning

oberste Schichten allgemeiner todo papaer verweisen

2.3 AutoML

2.3.1 RandomSearch

2.3.2 HyperBand

2.3.3 BOHB

2.4 Bibliotheken und Werkzeuge

Für den Praktischen Teil der Abschlussarbeit wurde insbesondere Cnvrgr [cn] ge-
nutzt. Cnvrgr.io ist eine "full-stack Data Science Platform" welche Werkzeuge für
die Erstellung, Verwaltung, Bereitstellung und Automatisierung von maschinellem
Lernen bereitstellt. Cnvrgr erlaubt es Arbeitsbereiche mittels Containern zu erstel-
len. Die Container können dabei auf Maschinen in Azure [Mi20] zugreifen. Für
die Experimente wurde ein vorgefertiger Container mit einer tesla-k80 [Nv20], fünf
CPUs und 49 GB Arbeitsspeicher genutzt.

Für die Entwicklung wurden Python [Py20], Jupyter Notebooks [**ProjectJupyter**]
und das Framework Tensorflow [Ma15] genutzt. Die wichtigsten Bibliotheken für
die Arbeit sind Keras [Ch15] , Numpy [Ol06] , Matplotlib [Hu07] , scikit-learn
[Pe11] , ConfigSpace [**Lindauer.8162019**] , Bayesian Optimization and Hyperband
[SAF18] .

Für die Visualisierung von Bildeinbettung wurde das Werkzeug "PSIORI Visualizer" erweitert und eingesetzt. Der Visualizer erlaubt es Daten in 3D darzustellen, von verschiedenen Blickwinkel und Zoomstufen zu betrachten, zu Filtern und mit zusätzlichen Informationen zu versehen. Die Abbildung 2.2 zeigt einen Screens-

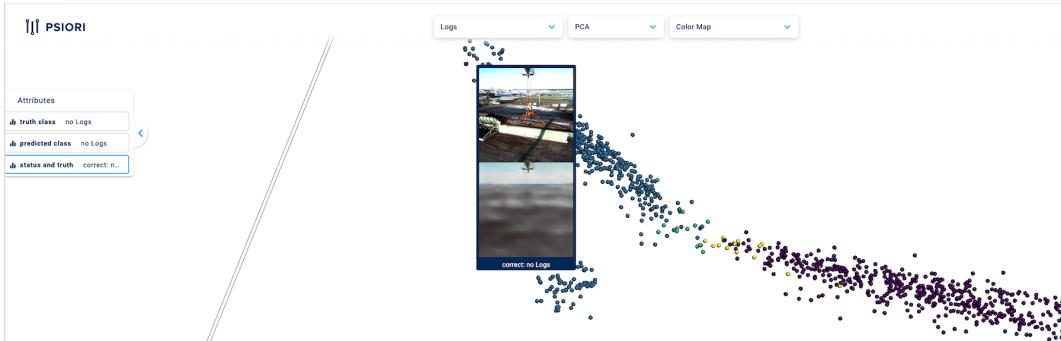


Abbildung 2.2: Beispiel PSIORI Visualizer

hot einer Visualisierung einer Einbettung. Als zusätzliche Information sind die Datenpunkte entsprechend einer Klassifikation in True-Positiv, True-Negativ, False-Negativ und False-Positiv eingefärbt. Wird über einen Datenpunkt mit der Maus geschwebt, werden zusätzliche Informationen zu dem Datenpunkt angezeigt. Diese Funktion wurde insbesondere zum Anzeigen eines Orginalbildes, ihrer Rekonstruktion mittels Autoencoder und einer Beschriftung genutzt.

Kern der erstellten Werkzeuge ist das Framework Psipy [PS19]. Psipy ist ein Python-Framework für Maschinelles Lernen welches von PSIORI selbst entwickelte Modelle zusammenfasst und eine einheitliche 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 Konstruktorgargmente 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 wichtigen Methoden gibt es die Methode *pretrain(..)* und *fit(..)*. Mittels *pretrain(..)* werden die Schichten eines symmetrischer Autoencoder von aussen nach innen wie in Greedy Layer-Wise Training of Deep Networks vortrainiert. Die Auswahl der Schichten erfolgt wieder in den abgeleiteten Klassen. In der *fit(..)*-Methode wird nach einigen Prüfungen die Methode ffit(..) [fit Keras] des Kerasmodells aufgerufen. In Abbildung 2.3 ist das Klassendiagramm mit den öffentlichen Methoden des ConvolutionalAutoencoder dargestellt.

Autoencoder

fit Keras

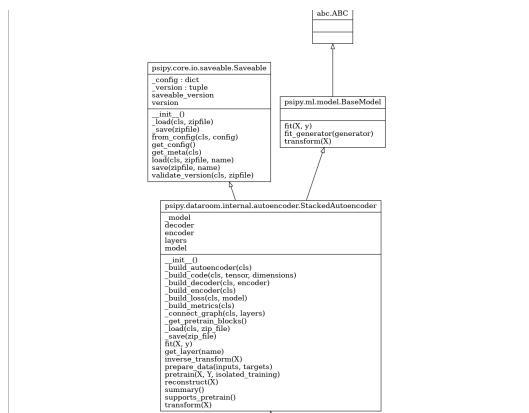


Abbildung 2.3: Klassendiagramm ConvolutionalAutoencoder

hyperparameter_mixin.py Hyperparameter_mixin wird zum Standardisierten Verwalten von Hyperparametern für AutoML-Klassen genutzt. Auf die Hyperparameter kann anschließend einheitlich zugegriffen werden. Abbildung 2.4 zeigt das zugehörige UML-Klassendiagramm mit den Methoden zum Hinzufügen, Löschen

und Laden der Hyperparameter. Da die Methoden öffentlich sind können über jede erbende Klasse die Hyperparameter eingenständig verwaltet werden.

psipy.ml.automation.hyperparameter_mixin.HyperparameterMixin
SUBSPACE_DELIMITER : str
hyperparameter_space
__init__(self, subspace)
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

2.5 Einordnung und bestehende Systeme

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



Abbildung 2.5: Rundlaufkran (Foto: ANDRITZ)

Greifer-Erkennung Bei der Aufgabenstellung Greifer-Erkennung muss in einem Bild die Position eines Rahmens um den Greifer gefunden werden. Abbildung 2.6 zeigt ein Bild mit Rahmen um den Greifer. Es handelt sich um eine klassische 'Object-Detection' Aufgabe. PSIORI hat die Aufgabe mittels neuronalem Netzwerk gelöst. Dabei wurde auf die Technik des Single Shot MultiBox Detector (SSD) [todo <https://arxiv.org/abs/1512.02325>] zurückgegriffen. Das Netz liegt als frozen_inference_graph.pb vor. Protocoll Buffer [<https://developers.google.com/protocol-buffers/>] ist ein sprachneutraler, plattformneutraler, erweiterbarer Mechanismus zur Serialisierung strukturierter Daten. In diesem Fall enthält die Datei den eingefrorenen Graph und die Model Gewichte. Um Vorhersagen treffen zu können wurde eine Klasse erstellt, welche mittels einer Tensorflowsession und dem Model vorhersagen trifft. Dabei wird liefert das Mode pro Billd einen Rahmen in welchem sich der Greifer befindet und einen Vertrauenswert. Der Vertrauenswert sagt aus wie sicher sich das Netzt mit seiner Aussage ist. Die Vorhersagegenauigkeit dieses Modells wird als Basislinie und Vergleichswert für die durchgeföhrten Versuche genutzt.

Baumstamm-Klassifikation Die Aufgabe der Baumstamm-Klassifikation hat zum Ziel, zu erkennen ob sich Baumstämme im Greifer befindet oder nicht. Es handelt sich um eine Klassifikationsaufgabe. Ein Klassifikator, für die Frage ob sich Baumstämme in dem Greifer befinden liegt als Tensorflow-MetaGraph[https://www.tensorflow.org/api_docs/python] vor. Um Vorhersagen treffen zu können wurde eine Klasse welche den MetaGraph nutzt erstellt. Die Vorhersage des Modells liefert die vorhergesagte Klasse mit einer Wahrscheinlichkeit zurück. Dieses Model wird wie das Greifererkennung Model als Basislinie und Vergleichswert für die durchgeföhrten Versuche genutzt.



Abbildung 2.6: Greifer mit Rahmen

2.6 Datenverständnis

Anlehnend dem in Kapitel 1.2 beschriebenen Vorgehen werden in diesem Kapitel die zur Verfügung stehenden Daten und deren Qualität beschrieben. Dabei ist das Kapitel entsprechend der Beschriftung der Daten in zwei Teilbereiche unterteilt.

Im Rahmen des Autocrane-Projektes 2.5 wurde eine Kamera an einem Rundlaufkran angebracht. Die Kamera ist so ausgerichtet, dass sich Aufhängung des Greifers am mittleren oberen Bildrand befindet. Bei einem Rundlaufkran kann die Auslenkung komplett um das Zentrum des Krans bewegt werden. Der Hintergrund der Bilder kann sich stark ändern. Mittels der Kamera werden kontinuierlich neue unbeschriftete Bilder aufgenommen und bei PSIORI abgelegt. Zu Beginn dieser Arbeit

Vorgehen auch
schreiben / anson
Kapitel Einleitun
passen

Im Kapitel Beste
System erwähnen
sen Teil in das a
Kapitel verschie

standen mehr 385.000 nicht beschriftete Bilder zur Verfügung. Die Bilder sind 1024 auf 648 Pixel groß und in Farbe. Sie sind in der Form (1024, 648, 3). Die einzelnen Pixel können dabei Werte zwischen 0 und 255 annehmen. Zum Erreichen der Zielstellung werden zwei Datensätze benötigt.

Greifer Datensatz Der Greifer Datensatz enthält Bilder, in welchen der Greifer mittels Rahmen markiert ist. Abbildung 2.7 zeigt ein Beispielhaftes Bild mit markiertem Greifer. Abbildung 2.6 zeigt ein Beispielhaftes Bild mit markiertem Greifer.

Der Datensatz besteht aus zwei Sammlungen von qualitativ unterschiedlich gut beschrifteten Bildern. Die eine Sammlung besteht aus einem bestehenden Datensatz, welcher 4.684 durch Menschen annotierten Bildern enthält. Für den zweiten Teil der Sammlung wurden mittels der bestehenden Objekterkennung 14.018 Bilder annotiert.

Baumstamm Datensatz Der Baumstamm Datensatz enthält Bilder, welche die Annotation, ob sich Baumstämme im Greifer befinden haben. Abbildung 2.7 zeigt ein Bild, in welchem der Greifer Baumstämme greift. In Abbildung 2.6 befinden sich keine Baumstämme im Greifer. Der Datensatz wurde im Zusammenarbeit mit



Abbildung 2.7: Greifer mit Baumstämmen

quality-match[<https://www.quality-match.com/imprint>] und Crowdworkern erstellt.

Weiterer Datensatz Im laufe der Arbeit wurden in Zusammenarbeit mit quality-match ein weiterer Datensatz erstellt. Dieser Datensatz enthält 80.000 beschriftete Bilder. Es wurden sowohl die Beschriftung "Logs ja / nein" als auch die Beschriftung Rahmen des Greifers erstellt. Zusätzlich wurden weitere Beschriftungen wie Helligkeit, Winkel des Greifers, ... erstellt. Diese weiteren Beschriftungen wurden in der Arbeit nicht genutzt.

finale Zahl setzen

stimmt das am Ende noch?

2.7 Datenvorbereitung

Die Datenvorbereitung dient dazu, einen finalen Datensatz zu erstellen, der die Basis für die nächste Phase der Modellierung bildet.

In dem Schritt Datenvorbereitung werden die Bilder für die Modellerstellung vorbereitet. In dieser Arbeit wurde für diesen Schritt eine Klasse Preprocessing in einem neuen Modul `data_preparation.py` erstellt. Wie in Listing

zu sehen werden die Pixel der Bilder zwischen 0 und 1 Skaliert. Die Skalierung erfolgt damit jedes Bild eine ähnliche Gewichtung

Neural networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process. As such it is good practice to normalize the pixel values so that each pixel value has a value between 0 and 1.

Die Bilddaten werden

	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

Todo list

3. Werkzeuge

Dieses Kapitel erläutert die erstellten Werkzeuge im Detail. Die Werkzeuge bauen auf einander auf und werden der Reihe nach erläutert.

3.1 ConvolutionalSecondCriterionAutoencoder

Der SCAE erweitert einen ConvolutionalAutoencoder um ein weiteres Kriterium. Es gibt also zusätzlich zu der Rekonstruktion des Autoencoders einen weiteren Ausgang. Der zweite Ausgang kann wie jeder Ausgang für eine Binärklassifikation, für eine Multiklassifikation, für eine Regression oder jede andere beliebige Aufgabe genutzt werden. In Abbildung 3.1 ist der schematische Aufbau des SCAE abgebildet. Die Schichten des zweiten Kriteriums werden an das Code-Schicht des NN angehängt. Es können beliebig viele Schichten genutzt werden. Die Verlustfunktion des NN besteht aus der Summe der einzelnen Verlustfunktionen und einer Gewichtung. Sie lautet im Detail:

$$loss = weight1 * loss_autoencoder + weight2 * loss_secondcriterion \quad (3.1)$$

Die Gewichtung der Verlustfunktionen kann dem SCAE per Konstruktor-Argument übergeben werden. Das Werkzeug ist als Python-Module implementiert. Dabei implementiert die Klasse ConvolutionalSecondCriterionAutoencoder den ConvolutionalAutoencoder aus Psipy. In Abbildung 3.2 ist das Klassendiagramm des SCAE dargestellt. Über den Konstruktor können alle Argumente welche zum Erstellen des Models notwendig sind per doppeltes Sternchen Wörterburch Argument (**kwargs) an die Klasse übergeben werden. Diese Technik erlaubt es eine mit Schlüsselwörtern versehene Argumentliste variabler Länge zu übergeben. Die Argumentlisten werden beinahe in allen Methode zum Einsatz gebracht. Sie werden insbesondere genutzt um Argumente an die zugehörigen Keras-Methoden zu übergeben. Die

3 Werkzeuge

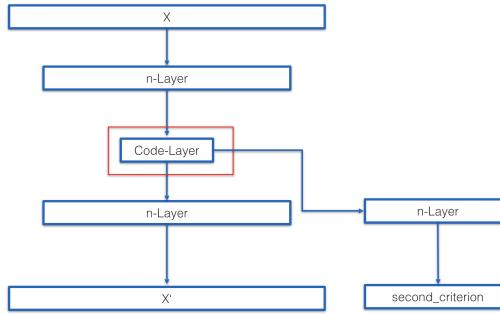


Abbildung 3.1: Schema SecondCriterionAutoencoder

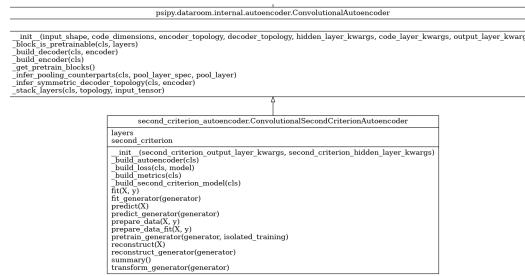


Abbildung 3.2: Klassendiagramm ConvolutionalSecondCriterionAutoencoder

Namensgebung der Methode orientiert sich dabei an Keras. So wird z.B. in dem Methodenaufruf *fit(..)* unter anderem auch die Keras-Methode *fit(..)* aufgerufen. Um einen SCAE zu trainieren ist es notwendig einen 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 beispielhafte dargestellt wie ein SCAE erstellt wird. In den ersten 10 Zeilen wird die Architektur erstellt. Ab Zeile 12 wird eine Instanz eines CSCA mittels Argumentenliste erstellt. Zu beachten ist, dass hier keine Decoder-Architektur übergeben wird. Wenn keine Decoder-Architektur bereitgestellt wird wird sie beim Erstellen des eigentlichen Modells aus der Encoder-Architektur abgeleitet.

```

1  encoder_topology = [("Conv2D", {"filters": 8, "kernel_size": (3, 3)}),
2  ("Conv2D", {"filters": 8, "kernel_size": (3, 3)}),
3  ('MaxPooling2D', {"pool_size": (2, 2)}),
4  ("Conv2D", {"filters": 16, "kernel_size": (3, 3)}),
5  ("MaxPooling2D", {"pool_size": (2, 2)}),
6  ("Conv2D", {"filters": 16, "kernel_size": (3, 3)}),
7  ("Flatten", {}),
8  ("Dense", {"units": 16})]
9
  
```

```

10 second_criterion_topology = [("Dense", {"units": num_classes})]
11
12 cscsa = ConvolutionalSecondCriterionAutoencoder(
13     input_shape=(28, 28, 1),
14     code_dimensions=3,
15     encoder_topology=encoder_topology,
16     second_criterion_topology=second_criterion_topology,
17     hidden_layer_kwargs = {'activation': 'relu'},
18     output_layer_kwargs = {'activation': 'sigmoid'},
19     second_criterion_hidden_layer_kwargs = {'activation': 'relu'},
20     second_criterion_output_layer_kwargs = {'activation': 'softmax'},
21     second_criterion_loss = 'categorical_crossentropy',
22     loss_weights=[8., 1.],
23     second_criterion_metrics = {'second_criterion': 'accuracy'},
24     code_layer_kwargs=dict())

```

Listing 3.1: Beispiel Erstellung ConvolutionalSecondCriterionAutoencoder in Python

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

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

Listing 3.2: Beispieldaufruf Pretrain in Python

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

```

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

```

Listing 3.3: Beispieldaufruf Fit in Python

3.2 TransferSecondCriterionAutoencoder

Ein TransferSecondCriterionAutoencoder basiert auf einem SCAE. Das zweite Kriterium wird durch ein neues Kriterium ersetzt. Zum Beispiel kann ein SCAE eine Objekterkennung durchführen. Bei einem TransferSecondCriterionAutoencoder wird die Objekterkennung durch eine Klassifikationsaufgabe ersetzt. Abbildung 3.3 zeigt den Aufbau des TSCAE. Im Klassendiagramm 3.4 für den TSCAE ist zu sehen. Das besondere ist, dass ein SCAE als Konstruktorargument übergeben

3 Werkzeuge

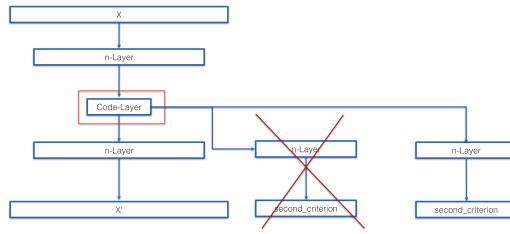


Abbildung 3.3: Schema TransferSecondCriterionAutoencoder

wird. Die Einstellungen für den ConvolutionalAutoencoder werden aus diesem Modell kopiert. Zusätzlich müssen nur noch die Einstellungen für das zweite Kriterium übergeben werden. Aus diesen wird das Model für das zweite Kriterium erstellt. Neu hinzugekommen ist, ein Parameter(freeze_encoder_layers) über den eingestellt werden kann, ob und welche Schichten nicht neu trainiert werden können. Es können direkt Schichten ausgewählt werden oder es kann über einen Ganzahlenwert beginnend über die erste Schicht Schichten eingefroren. Werden nur Werte für den Encoder übergeben werden Werte für den Decoder (freeze_decoder_layers) daraus abgeleitet. Listing 3.4 zeigt einen Beispielhaft die Anwendung dieses Werkzeuges.

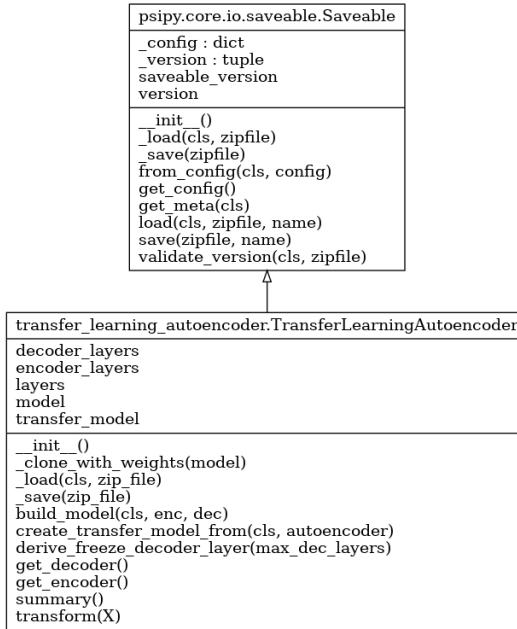


Abbildung 3.4: Klassendiagramm TransferSecondCriterionAutoencoder

Im Vergleich zu dem SCAE ist die Anwendung schon deutlich einfacher. Es gibt weniger Hyperparameter zum Beachten. Da das Modell auf einem trainierten SCAE basiert ist ist kein `pretrain(..)` mehr notwendig. Die Methode `fit(..)` wird auf die selbe Weise wie bei SCAE angewendet.

```

1  tscm = TransferLearningConvolutionalSecondCriterionAutoencoder(csc_autoencoder,
2    second_criterion_topology=second_criterion_topology,
3    second_criterion_loss = 'binary_crossentropy',
4    second_criterion_hidden_layer_kwarg = {'activation': 'relu'},
5    second_criterion_output_layer_kwarg = {'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 AutoTransferSecondCriterionAutoencoder

Der AutoTransferSecondCriterionAutoencoder ist ein TransferSecondCriterionAutoencoder welcher mit Hilfe von HpBandSter AutoML zur Hyperparameteroptimierung einsetzt. Konkret ist erbt die Klasse von hpbandster.core.worker. Zur Speicherung und Verwaltung der Hyperparameter wird auf die Klasse Hyperparameter-Mixin zurückgegriffen. Bei der Instanzierung der Klasse werden sinnvolle Standardwerte für Hyperparameter gesetzt. Die Werte können aber später noch angepasst werden. In Abbildung 3.5 ist das zugehörige Klassendiagramm abgebildet. In Lis-



Abbildung 3.5: Klassendiagramm AutoTransferSecondCriterionAutoencoder

ting 3.5 ist eine einfache Implementierung eines AutoTransferSecondCriterionAutoencoder dargestellt. Der Konstruktor unterscheidet sich nur an einer Stelle zum Konstruktor des TransferSecondCriterionAutoencoder. Es wird keine Instanz des SCAE übergeben sondern ein Pfad zu einem abgespeicherten Modell eines SCAE. Im 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 TransferSecondCriterionAutoencoder erstellt und mittels der ausgewählten Hyperparameter und übergebenen Daten trainiert.

```

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

```

Listing 3.5: Beispiel AutoTransferSecondCriterionAutoencoder in Python

Das Werkzeug unterstützt derzeit die Optimierer, Randomsearch, Hyperband und BOHB. In Tabelle 3.1 sind die Hyperparameter und ihre möglichen Werte dargestellt. Der Wertebereich für die Batchsize wurde aus der Autocrane-Problemstellung

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

Tabelle 3.1: Standard Hyperparameter für AutoML-Suche

abgeleitet. Sie kann maximal so groß werden, dass kein Speicherplatz-Fehler auftreten kann. Die maximale Anzahl der Epochen wurde bewusst groß gewählt. Es sollen weitere Problemstellungen ohne viel Konfigurationsaufwand gelöst werden können. Absurd lange Laufzeiten können durch ein EarlyStopping Kriterium verhindert werden. Die Maximalanzahl der möglicherweise einzufrierenden Schichten wird über den Anwender des Werkzeuges definiert, sie sind Anwendungsspezifisch.

weight durch Verwaltung von w1/w2 anpassen.

Worker Budget berücksichtigen / notieren

Todo list

4. Experimente

4.1 Versuchsaufbau

4.1.1 Psipy-Modul

Zweites Kriterium Autoencoder Warum pretrian?: <https://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf> (Warum machen das andere heute nicht mehr? viele cnn haben vanishing gradients problem über relu,... gelösst) (weitere literatur: möglicherweise Geoffrey E. Hinton)

Transfer-Lernen Autoencoder

Auto-Transfer-Lernen Autoencoder

4.2 Modellierung

4.2.1 todo: Greifer

4.2.2 todo: Transferlearning

4.2.3 todo: Holz

4.3 Evaluierung

4.3.1 todo: Greifer

4.3.2 todo: Holz

Todo list

5. Fazit

5.1 Zusammenfassung

5.2 Kritische Reflexion

5.3 Ausblick

insbesondere die möglichen Addons aufführen

Abkürzungsverzeichnis

Tabellenverzeichnis

2.1	Datenaufteilung - Train Test Validation	13
3.1	Standard Hyperparameter für AutoML-Suche	22

Abbildungsverzeichnis

2.1	Schema Autoencoder	5
2.2	Beispiel PSIORI Visualizer	7
2.3	Klassendiagramm ConvolutionalAutoencoder	8
2.4	Klassendiagramm Hyperparametermixin	9
2.5	Rund-Kran	10
2.6	Bsp. Bild: Greifer mit Rahmen	11
2.7	Bsp. Bild: Greifer mit Baumstämmen	12
3.1	Schema SecondCriterionAutoencoder	18
3.2	Klassendiagramm ConvolutionalSecondCriterionAutoencoder	18
3.3	Schema TransferSecondCriterionAutoencoder	20
3.4	Klassendiagramm TransferSecondCriterionAutoencoder	20
3.5	Klassendiagramm AutoTransferSecondCriterionAutoencoder	21

Listings

2.1	Erweiterung zum Speichern eines Tensorflow Models	8
3.1	Beispiel Erstellung ConvolutionalSecondCriterionAutoencoder in Python	18
3.2	Beispieldaufruf Pretrain in Python	19
3.3	Beispieldaufruf Fit in Python	19
3.4	Beispiel TransferSecondCriterionAutoencoder in Python	21
3.5	Beispiel AutoTransferSecondCriterionAutoencoder in Python	22

Literatur

- [Ch15] Chollet, F. et al.: Keras, 2015.
- [cn] cnvrg.io: cnvrg.io, URL: <https://cnvrg.io/>.
- [Hu07] Hunter: Matplotlib: A 2D graphics environment, 2007.
- [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/>.
- [Mi20] Micorsoft: Microsoft Azure, 2020, URL: <https://azure.microsoft.com/de-de/>.
- [Nv20] Nvidia: Tesla K80, 2020, URL: <https://www.nvidia.com/de-de/data-center/tesla-k80/>.
- [Ol06] Oliphant, T.: NumPy: A guide to NumPy, hrsg. von Trelgol Publishing USA, 2006, URL: <http://www.numpy.org/>.
- [Pe11] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12/, S. 2825–2830, 2011.
- [PS19] PSIORI GmbH: psipy documentation, hrsg. von PSIORI GmbH, 2019, URL: <https://psipy.azurewebsites.net/source/psipy.html>.

Literatur

- [PS20] PSIORI GmbH: PSIORI, 2020, URL: <https://www.psiori.com/de>.
- [Py20] Python Software Foundation: The Python Language Reference, 2020, URL: <https://docs.python.org/3.7/reference/>.
- [SAF18] Stefan Falkner; Aaron Klein; Frank Hutter: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. arXiv:1807.01774/, 2018, URL: <https://ml.informatik.uni-freiburg.de/papers/18-ICML-BOHB.pdf>.