# Evaluierung von Transferlernen mit Deep Direct Cascade Networks

Simon Tarras

June 27, 2025

# Contents

# Chapter 1

# Einleitung

## 1.1 Einführung

## 1.2 Motivation

## 1.3 Related Work

# Chapter 2

# Methodik

## 2.1 Transferlernen

Transferlernen (TF) ist das Prinzip des Lernens über einer Eselsbrücke. Es gibt mehrere Varianten, wie TF verwendet werden kann. Dies sind Task-Wechsel und Domain-Wechsel. Nur wenn keine davon genutzt wird, wird nicht von TF gesprochen. Hier wird nur der Domain-Wechsel vorgestellt werden, da nur dieser benutzt wird. Ein Domain-Wechsel ist hier der Wechsel zwischen zwei verschiedenen Datensätzen, während diegleichen Netzarten genutzt wird. Dies wird Transductive Transferlernen[JY10] genannt. Das Wissen vom ersten Datensatz wird auf den zweiten übertragen. Der erste Datensatz ist dabei die Source, der Zweite das Target. Es gibt dabei drei Stellschrauben, bei denen nicht klar ist, was besser ist: What, How, When to Transfer [JY10]. Da es sowohl eine Klassifikation als auch eine Regression ausgetestet wird, werden jeweils zwei Source- und Targetdatensätze benötigt. Für Klassifikation wird die Source der Modified National Institute of Standards and Technology [LeC+] (MNIST) Datensatzes und der Street View House Numbers (SVHN) [Net+11] der Targetdatensatz sein. Beide müssen für das Transfer ein wenig verändert werden. Der MNIST wird von 28x28 Pixel auf 32x32 erweitert, während der SVHN von farbig auf schwarz-weiß verändert wird. Dies ist notwendig, da beide Datensätze als Input denselben Shape, also diegleichen Dimensionalitäten vorweisen, haben müssen. Bei der Regression ist der Sourcedatensatz der Boston Housing Prices (Bost) [Har+97] und der Targetdatensatz der California Housing Prices (Cali) [Nug18].

Beide Datensätze müssen stark reduziert werden. Von den Acht beziehungsweise Dreizehn Spalten bleiben nur Drei übrig. Dies hat den Grund, dass nur Spalten als Sinnvoll geachtet werden, die ein passendes gegenüber haben. Der Bost-Datensatz hat allerdings ein ethnisches Problem, da dieser eine Spalte enthält, die diskriminierend ist. Diese wird entfernt. Die einzigen Spalten des Bost-Datensatzes, die übrig bleiben sind: RM, AGE, LSTAT. RM ist die durchschnittliche Zimmeranzahl pro Wohnung, AGE ist die Anzahl der Häuser, die vor 1940 bewohnt wurden und LSTAT ist der prozentuale Anteil der Bevölkerung

mit niedrigerem Status. Der Datensatz Cali behält nur die Spalten MedInc und HouseAge. MedInc ist das durchschnittliche Einkommen des Häuserblocks und HouseAge das durchschnittliche Alter. Aus den Spalten AveRooms und Households wird die durchschnittliche Anzahl von Zimmers pro Haushalt berechnet. AveRooms ist dabei die durchschnittliche Anzahl an Räumen innerhalb eines Häuserblocks, während Households die Anzahl der Haushalte innerhalb des Häuserblocks ist. Dadurch ist die berechnete Spalte zu der RM-Spalte von Bost passend. Da LSTAT und MedInc wahrscheinlich abhängig sind, da es vermutet wird, dass diejenigen Menschen, die einen niedrigeren Status vorweisen, weniger Einnahmen haben. Deshalb dürfte es über diese beiden Spalten möglich zu sein TF zu nutzen. Allerdings sind sie zueinander antiproportional, weshalb die LSTAT Spalte invertiert wird, damit es zur Proportionalität kommt. Komplexer ist die Berechnung des Alters der Häuser, da AGE nur die Anzahl der Häuser, die vor 1940 gebaut wurden, beeinhaltet, aber HouseAge das durchschnittliche Alter des Häuserblocks ist. Die Maximalanzahl der betrachteten Häuser im Bost-Datensatz ist einhundert und das Alter der Häuser vor 1940 ist 85, wenn man auf 2025 rechnet. Dadurch kann AGE auf die Art von HouseAge mit folgender Formel umgerechnet werden:

$$\frac{AGE * 85}{Maximalanzahl} \tag{2.1}$$

Dadurch sind alle Source- und Targetdatensätze zueinander kompatibel. Damit ist ausreichend geklärt, mit was TF verwendet wird.

Die nächste Frage, die geklärt werden muss ist das How to transfer. Dies wird jeweils ohne Veränderung der Weights der Netze gemacht. Es wird das neuronale Netz zuerst auf dem Sourcedatensatz trainiert und dann ohne irgendetwas zu tun auf den Targetdatensatz gewechselt, welcher auf demselbem Netz oder einem gleichen Netz wie zuvor ist. Wenn es dasselbe Netz ist, dann verändert sich nur aus welchem Datensatz der Input kommt, was bei Deep Cascade ist. Während bei dem gleichen Netz der Input immer vergrößert wird und das TF über diese Vergrößerung passiert, was bei Direct Cascade ist.

Wann TF sinnvoll ist, ist nicht klar, weshalb es mal mit früherem und späteren TF probiert wird.

## 2.2 Kaskadierung

Hier wird erklärt, was ein Kaskadennetzwerk ist und welche Besonderheiten es dabei gibt. Ein Kaskadennetzwerk ist ein Netzwerk, welches in Kaskaden, also Schrittweise, aufgebaut wird. Während bei einem klassischen Netz vorher festgelegt wird, wieviele und welche Layer dieses haben wird, ist es bei einem Kaskadennetzwerk nicht so. Ein solches Netzwerk wird erst während dem Training aufgebaut und immer erweitert. Deshalb werden, im Gegensatz zu den klassischen Netzen, nur der aktuelle neue Teil trainiert, während der Rest nicht mehr verändert wird. Die vorher gelernten Layer werden nach dem Training gefreezt. Dadurch werden die Gewichte mehr der gefreezten Layer nicht mehr verändert.

Die Kaskadennetzwerke lernen dadurch das, was zwischen den Layern gelernt wird, nicht und sind deshalb etwas schlechter als die klassischen Netzwerke bei gleich vielen Daten. Aber, weil immer nur das aktuelle gelernt wird, sind Kaskadennetzwerke im Training sehr viel schneller. Dies liegt daran, dass die Gewichte der vorherigen Layer kein sich verändertes Bild von den nachfolgenden Gewichten in jeder Epoche haben und sich nicht aktualisieren müssen.

### 2.2.1 Deep Cascade

Hier wird die Variante des Deep Cascade vorgestellt. Die Deep Cascade Netze werden iterativ während dem Training aufgebaut. Es bleibt dabei ein einziges Netz. Es wird zuerst definiert, welcher Optimizer und welcher Loss in dem Netz genutzt wird.

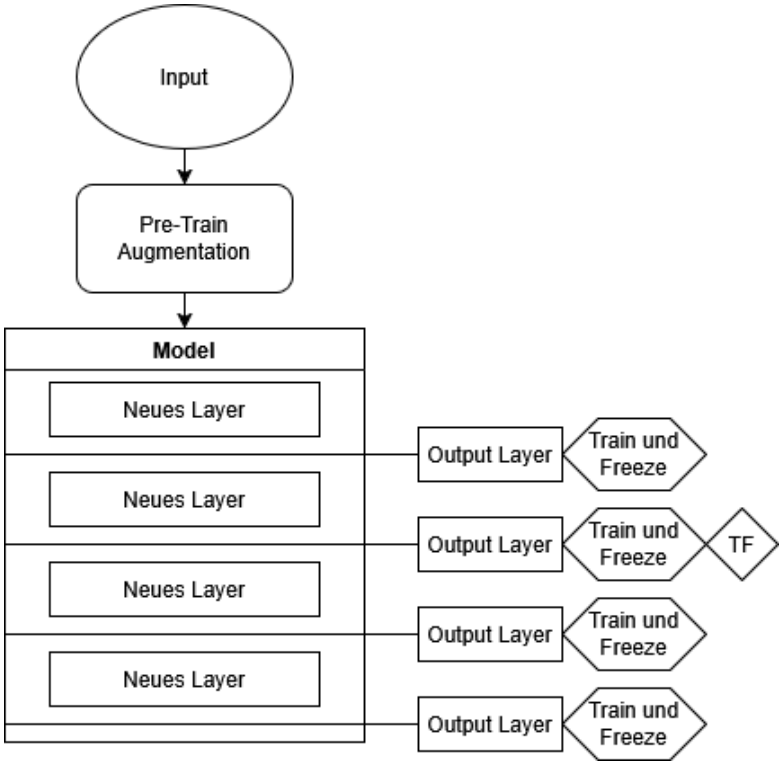


Figure 2.1: Vorstellung Deep Cascade Aufbau

Sobald dies beides gemacht wurde, wird im Netz das Erste Layer definiert. Dieses wird ergänzt durch ein Output Layer und dann trainiert. Wenn das Training beendet wird, wird das Output Layer gelöscht und ein neues Layer hinzugefügt, wie es in Figure 2.1 gezeigt wird. Zudem wird das gerade trainierte Layer gefreezt, damit dieses keine weiteren Aktualisierungen mehr bekommt.

Dann wiederholt sich das Training, das löschen, das freezing und weitere hinzufügen von Layern. An einer beliebigen Stelle kann TF gemacht werden, indem, statt in der Trainingsphase den Sourcedatensatz zu nutzen, der Targetdatensatz genutzt wird.

### 2.2.2 Direct Cascade

Hier wird die Kaskadierungsvariante des Direct Cascade vorgestellt. Das Netzwerk ist hier vorher vollständig und besteht aus einem einzigen Hidden Layer und einem Output Layer. Es wird dasselbe Netz mehrfach trainiert und währenddessen wird das Wissen zwischen diesen Netzen weitergegeben.

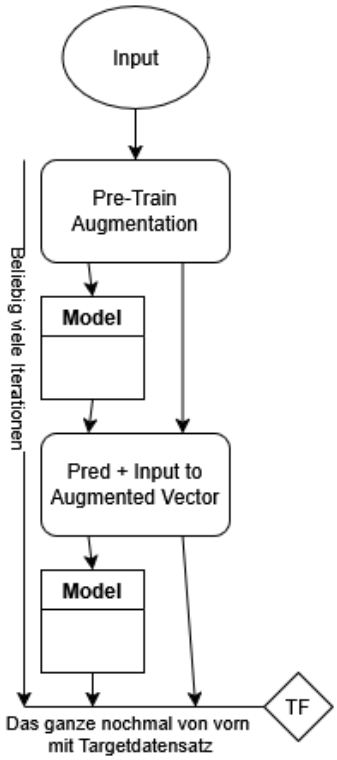


Figure 2.2: Vorstellung Direct Cascade Aufbau

Es beginnt, wie in Figure 2.2 gezeigt mit dem präpariertem Sourcedatensatz, der in die Instanz des Netzes hineingegeben wird. Damit wird diese Instanz trainiert und sobald dies beendet ist, wird einmal das fixe Netzwerk angewendet. Das Ergebnis davon ist die Prediction. Diese wird mit dem Input desselben Netzes verbunden und es entsteht der Augmented Vector. Darauf, wie dieser Augmented Vector genau entsteht, wird später nochmal eingegangen, da es bei jedem Direct Cascade Netzwerk ein wenig anders ist. Der Augmented Vector

wird in die nächste Instanz von dem Netzwerk als input hineingegeben. Die Netzinstanzen, das Training, die Prediction und Berechnung der Augmented Vectors wird beliebig häufig wiederholt. Dabei lernt das Netzwerk über den Augmented Vector das Wissen der vorherigen Netzwerke mit, da dieses als Prediction dort mit vorkommt.

TF kann nun jederzeit im Trainingsschritt gemacht werden, indem dort der Targetdatensatz statt der Sourcedatensatz als Input genommen wird. Dabei können beliebig viele Netzwerke vor und nach TF genutzt werden. Der einzige Unterschied ist der, dass der Augmented Vector für jedes Netzwerk ein wenig größer wird, da dieser sowohl das Wissen von jedem bisherigen Netzwerk als auch die Ursprungsdaten enthält.

Dabei muss hier in der Implementation bereits von Anfang an, sowohl der Sourcedatensatz als auch der Targetdatensatz in das feste Netz hineingegeben werden, um die Prediction auf dem Targetdatensatz von der Trainingsphase des Sourcedatensatzes im Augmented Vector zu integrieren, damit die Netzwerke, die auf dem Sourcedatensatz gelernt haben, Berücksichtigung finden.

## 2.3 Setup

Alle Test wurden auf einem Erazer Gaming Notebook P15601 unter Windows 10 durchgeführt. Die Neuronalen Netze laufen dabei ausschließlich auf der CPU und wurden nur trainiert, wenn der Rechner am Stromnetz angeschlossen war. Dieser Rechner hat einen intel Core i5 der neunten Generation mit 4 Kernen auf 8 logischen Prozessoren. Die Betriebsgeschwindigkeit liegt bei 2,4-5,1 GHz und die RAM-Größe liegt bei 15,8 GB bei einer Geschwindigkeit von 2667 MHz.

Es wurde mit PyCharm und der library Keras programmiert. Die Texte sind mit BibTex erstellt worden und die Plots mit der MatPlotLib library.

Dabei sind MNIST und Bost die Sourcedaten und SVHN und Cali die Targetdaten. Jeder Targetdatensatz wird händisch verkleinert, da es darum geht, nicht genügend Daten für sie allein zu haben und deshalb eine andere Methode genutzt werden muss.

## 2.4 Metrik

Es wurden drei Metriken erstellt. Die Accuracy- (ACCM), Loss- (LM) und MAE-Metrik (MAEM). MAE heißt dabei Mean absolute Error. Alle drei Metriken sind für Early Stopping und entscheiden, wieviele Epochen genutzt werden. Die Accuracy-Metrik bricht immer dann ab, wenn die Validation-Accuracy mindestens um 10% schlechter ist als die Trainingsaccuracy, da dann in dem Netzwerk Overfitting herrscht.

Die Loss- und die MAE-Metrik brechen beide dann ab, wenn der Validation-Wert der aktuellen Epoche schlechter ist als in der Epoche davor. Dies hat zur Folge, dass die Netze in lokale Minima hineinlaufen und nicht wieder herauskommen. Dabei unterliegt die Anzahl der Netze für das Direct Cascade

keiner Metrik.

## 2.5 Liste der Tests

Liste aller hier vorkommenden Netze:

1. ConvMaxPool
2. 1DConv
3. 2DConv
4. ClassOneDense
5. RegressionTwo
6. OneLayer

Davon sind ConvMaxPool und RegressionTwo Deep Cascade Netzwerke, während alle anderen Direct Cascade Netzwerke sind. Ebenso sind nur RegressionTwo und OneLayer Regressionsnetze, während der Rest Klassifikationsnetze sind.

Alle Netze werden mit dem Adam-Optimizer mit der learningrate 1e-3 gelernt. Klassifikationsnetze haben als Loss den CategoricalCrossEntropy und Softmax als Aktivierungsfunktion, während die Regressionsnetze MeanSquaredError und Linear als Aktivierungsfunktion vorweisen.

Mit allen Direct Cascade Netzwerken wurden zusätzlich Early Stopping Metriken durchgeführt mit MAEM, LM, ACCM.

Für fast alle Netze gilt, dass sie mit vielen, mittleren und wenigen Source- und Targetdaten durchgeführt wurden. Die einzige Ausnahme ist das 2DConv-Netzwerk, welches nur mit wenigen Daten durchgeführt wurde, da mit mehr Daten es technisch nicht mehr mit derselben Hardware möglich war.

Es wurde mit allen Deep Cascade Netzen ein Vergleich zwischen mit TF und ohne angefertigt.

Mit allen Netzwerken wurde der Zeitpunkt für das TF frei ausgetestet.

Alle Direct Cascade Networks haben jeweils nur ein Hidden Layer. In manchen Fällen noch mit einem Hilfslayer, um den Wechsel zwischen Filterlayern und Linearlayern zu bewerkstelligen.

Für alle Direct Cascade Networks wurde derselbe Seed für die Initialisierung der Weights genutzt.

# Chapter 3

# Allgemeine Resultate

## 3.1 Ploterklärung

In diesem Kapitel werden alle Arten der Plots einmal vorgestellt und auf alle Eigenheiten eingegangen, damit diese verstanden werden.

Es wird hier sowohl auf die Achsenbeschriftung als auch auf die Texte innerhalb der Plots eingegangen.

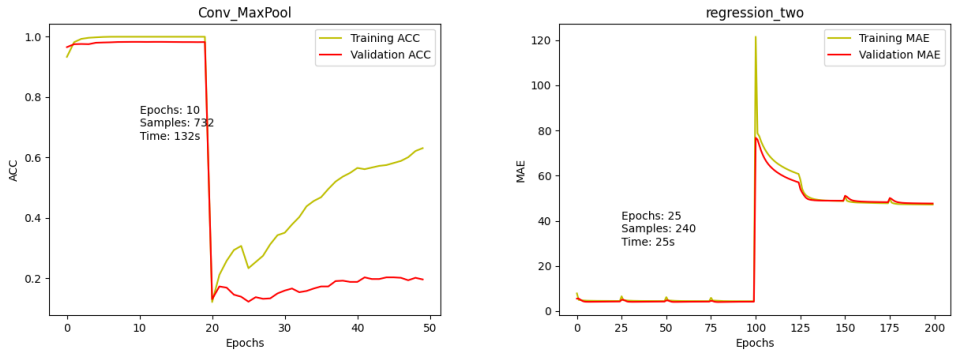


Figure 3.1: Vorstellung Plots

Dazu wird Figure 3.1 betrachtet. In beiden Teilen stehen drei Zeilen Text auf die nun einzelnd eingegangen wird. Die Erste sagt aus, wieviele Epochen pro Layer oder Netzwerk trainiert worden sind. Die Zweite beschreibt wieviele Datensamples des Trainingssets des Targetdatensatzes im Training nach TF genutzt worden sind und die dritte Zeile zeigt die gesamte Trainingsdauer in Sekunden an.

Wenn es um die Accuracy geht, was bei Klassifikation der Fall ist, dann steht ACC auf der senkrechten Achse und den Funktionsnamen dabei. Die senkrechte Achse ist dann bei 100%, wenn es bei 1 ist. In Figure 3.1 ist links ein Beispielplot

für diesen Fall.

Für die Regression, geht es um den MAE. Dies steht wiederum in den Namen der Funktionen und der senkrechten Achse. Diese Achse ist in 1000$ pro Einheit. Dabei ist es besser, je geringer der Wert ist.

## 3.2 ConvMaxPool

Anhand des ConvMaxPool-Netzwerks werden alle allgemeinen Resultate und Auffälligkeiten beschrieben. Dies ist ein Deep Cascade Classification Netzwerk und wird deshalb iterativ aufgebaut. Das Netz ist ein Convolution-Network mit Padding, sodass die Dimensionen während der Convolution-Layer sich nicht verringert. Es wird die Aktivierungsfunktion relu genutzt.

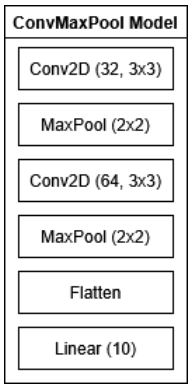


Figure 3.2: Vorstellung des ConvMaxPool Netzwerks

Alle Layer des ConvMaxPool Netzwerks sind in Figure 3.2 in korrekter Reihenfolge zu sehen. Dabei ist die erste Zahl eines Convolution Layer die Anzahl der genutzten Filter, während das folgende Tuple die Kerngröße beschreibt. Ebenfalls die Kerngröße steht bei den MaxPool Layern dort. Das Flatten- und das Linear-Layer sind der Output-Block. Das Linear-Layer benötigt zehn Nodes, da es zehn Klassen gibt. Jedes Hidden Layer wird mit zehn Epochen trainiert. Es gibt keine Early-Stopping Metrik und es wird derselbe Seed für alle Tests genutzt.

### 3.2.1 Veränderungen bei TF

Hier wird etwas sehr offensichtliches betrachtet. Dies passiert jedes Mal, wenn TF verwendet wird. Der Graph, der die Trainings- und Validationdaten nutzt, hat immer einen Einbruch in der Performanz an der Stelle an der TF gemacht wird. Dies ist in der Figure 3.3 deutlich zu sehen, bei Epoche zwanzig.

Dieser Einbruch passiert jedes Mal nach TF. Dies liegt daran, dass das Netz bisher die Targetdaten noch nie gesehen hat und bisher auf eine andere Domain mit dem Sourcedatensatz trainiert hat. Das Netz kennt nur das Wissen aus

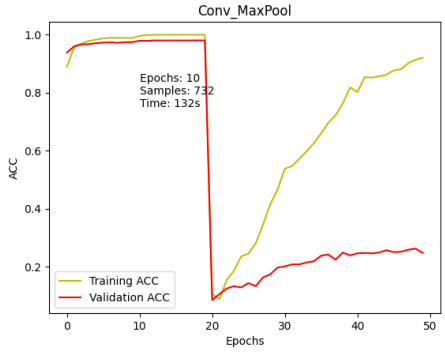


Figure 3.3: Einbruch bei TF

dem Sourcedatensatz und kann nur dieses anwenden. Wenn man aber das Testset, welches nur über die Targetdaten geht auf das ganze Netzwerk betrachtet, kommt Figure 3.4 heraus.

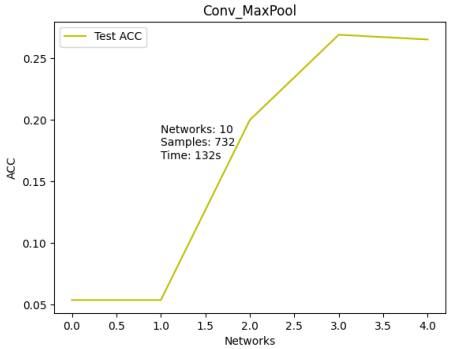


Figure 3.4: Verbesserung auf Testdaten

Der Wechsel ist hierbei bei Netzwerk 2. Es ist eindeutig zu erkennen, dass es nach TF besser wird. Dies hat den Grund, dass das Netzwerk ab diesem Zeitpunkt auf den Trainingsdaten trainiert, die zum Testdatenset passen.

### 3.2.2 Overfitting auf Sourcedatensatz

Wenn es unterschiedlich lang auf dem Sourcedatensatz trainiert wird, fällt auf, dass das Netz unterschiedlich gut auf dem Targetdatensatz ist. Da es sowieso ausgestestet werden muss, wann TF genutzt wird, wird nun das ConvMaxPool-Netzwerk genommen und nach jedem Layer TF angewandt. Das Ergebnis davon ist in Figure 3.5 zu sehen.

Auffällig ist es, dass hier die beste Performanz ohne TF ist. Bereits nach

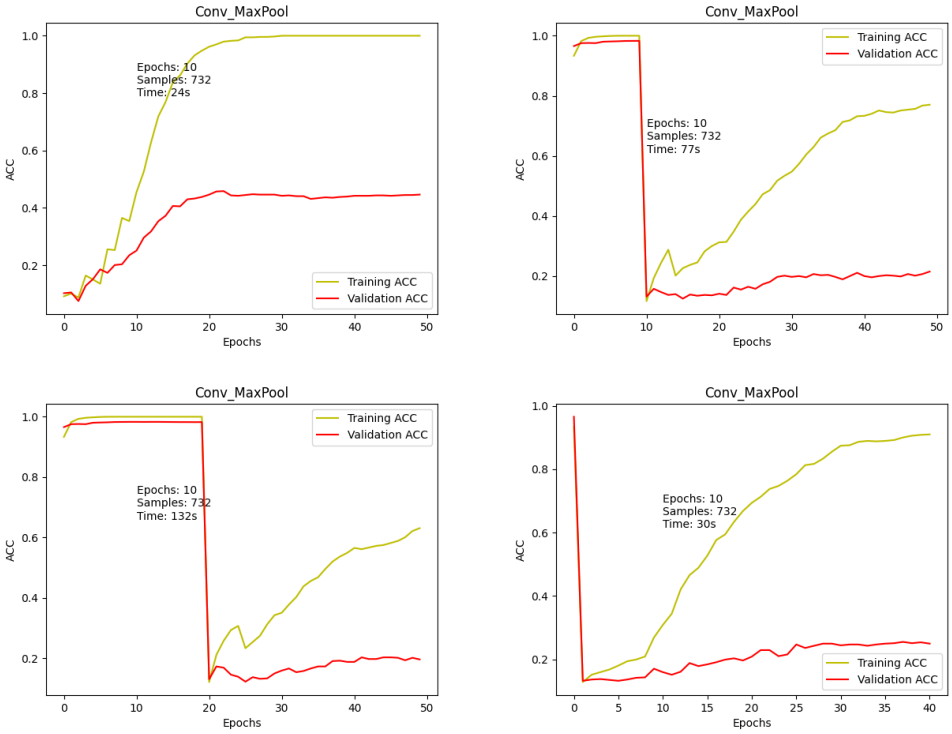


Figure 3.5: TF bei unterschiedlichen Layern

nur einer Epoche im ersten Layer, welches auf dem Sourcedatensatz trainiert wird, bricht die Accuracy ein. Dies zeigt, dass TF bei Klassifikation und Deep Cascade Netzwerken sinnfrei ist. Das Trainingsset der Trainingsdaten ist bei TF nie auch nur annähernd an den Bereich kommt, in dem es bei ohne TF ist. Daraus folgt, dass es bereits zu Overfitting auf dem Sourcedatensatz gekommen ist. Dadurch kann nicht mehr so gut auf dem Targetdatensatz gelernt werden. Dieses Overfitting passiert sogar bereits, wenn nur eine Epoche auf dem Sourcedatensatz gelernt wird, was die letzte Graphik von Figure 3.4 zeigt. Ebenso ist es offensichtlich, dass es bei jedem Graph zu Overfitting auf dem Trainingsset des Targetdatensatzes kam, da dieser um 60% höhere Accuracy als das Validationset und dem Testdatenset vorweist.

Bei einem Regressionnetzwerk, wie dem Deep Cascade Netzwerk RegressionTwo kommt es, wie in Figure 3.6 zu sehen, nicht so schnell zu Overfitting. Weder auf dem Sourcedatensatz noch auf dem Targetdatensatz.

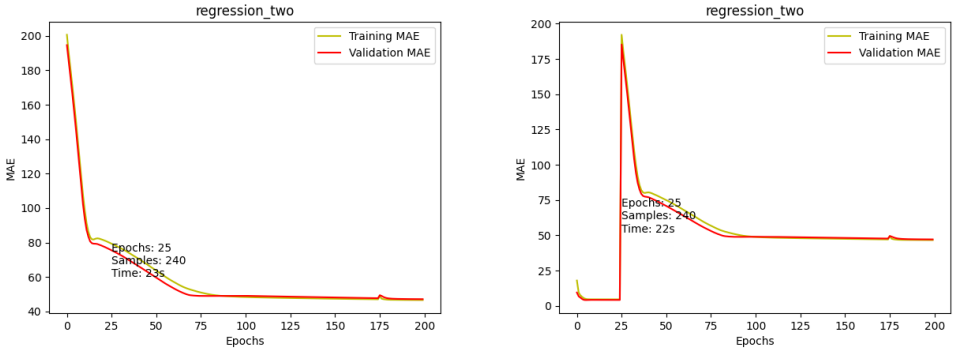


Figure 3.6: TF bei Regression

Dieses Overfitting-Problem hat nur die Klassifikation. Dies muss an der Loss-Function, die für Klassifikation benutzt wird, liegen. Also am Categorical-CrossEntropy. Dahinter ist folgende Formel:

$$CCE = -\frac{1}{M}\sum_{k=1}^{K}\sum_{m=1}^{M} y_m^k * \log(h_w(x_m, k)) \tag{3.1}$$

Dabei ist M die Anzahl der Datensamples, K die Anzahl der Klassen, $y_m^k$ das Target label, x der Input und $h_w$ die Gewichte [HW20].

## 3.3 Zeitnahme

Hier werden alle Netze bearbeitet und überprüft wieviel Zeit sie für ihr Training benötigten. Diese wird in jedem Plot angezeigt. Sie ist generell stark abhängig davon, wieviele Datensamples genutzt werden und wieviele maximale Epochen erlaubt sind. Deshalb werden hier jeweils gleich viele Datensamples

für die Klassifikationsnetze und Regressionsnetze verwendet, sowie eine gleiche Gesamtepochenanzahl.

Dazu wird jedes Mal der kleinste Targetdatensatz und der größte Sourcedatensatz genutzt, bis auf das 2DConv-Netzwerk, wenn TF verwendet wird. Solange nicht TF verwendet wird, wird nur der kleinste Targetdatensatz genommen. Jedes Klassifikationsnetz wird auf eine Gesamtanzahl von 40 Epochen trainiert. Wenn TF gemacht wird, dann nach 20 Epochen. Bei Regressionsnetzen wird auf 80 Epochen trainiert. TF wird nach 30 Epochen gemacht. Tatsächliche Graphen werden hier nicht verwendet, da es zuviele sind und die meisten an anderen Stellen in ähnlicher Form bereits vorkommen. Um diese Graphen zu kontrollieren, schaut bitte auf das GitHub Repo unter `https://github.com/Lirras/BA_EvalTF_DDCN/tree/main/Plots/ba_plots/timing`. Es wird hier keine Early-Stopping Metrik genutzt.

Dabei werden alle Klassifikationsnetze mit denselben Inputdaten gespeist, ebenso wie alle Regressionsnetze zur Vergleichbarkeit untereinander. Die jeweiligen Netzversion zwischen Cascade TF, Cascade und Complete haben dieselben Layer, sowie diegleiche Anzahl.

Hier nun die Tabelle mit allen Zeiten der Netze in einer vergleichbaren Variante:

Table 3.1: Timingvergleich in Sekunden

| **Netzwerk** | **Cascade TF** | **Cascade** | **Complete** |
|---|---|---|---|
| ConvMaxPool | 78 | 25 | 20 |
| 1DConv | 207 | 34 | 30 |
| 2DConv | 23 | 24 | 40 |
| ClassOneDense | 79 | 28 | 13 |
| RegressionTwo | 11 | 12 | 17 |
| OneLayer | 16 | 18 | 11 |

In Table 3.1 ist die Spalte Cascade TF, die Kaskadennetzwerke mit TF beeinhaltet. Die Spalte Cascade ist diejenige, in der die Netze nur auf dem Targetdatensatz gelernt haben, aber Kaskadennetzwerke sind und die Spalte Complete sind die Lernzeiten der Netze, die weder TF machen noch Kaskadennetzwerke sind, sondern dessen Layer vor dem Training bereits feststanden und komplett in einem auf dem Targetdatensatz gelernt wurden.

Auffällig ist, dass die Regressionsnetze keine große Zeitveränderung haben. Mitunter benötigt mit TF weniger Zeit als ohne, was daran liegt, dass der Targetdatensatz der Regression ein wenig größer ist als der Sourcedatensatz. Beide sind allerdings mit etwas über 200 Datensamples sehr klein.

Ebenso brauchen alle Klassifikationsnetze länger mit TF als ohne, was daran liegt, dass sie zuerst auf dem Sourcedatensatz trainieren, welcher mit 48000 Trainingsdaten recht groß ausfällt, während die Anderen direkt auf dem kleinen Targetdatensatz, der mit 732 Datensamples klein ist, trainieren.

Da Cascade TF mit dem Sourcedatensatz ist und die anderen Netze nur auf dem Targetdatensatz arbeiten sind diese meist kürzer. Allerdings benötigen die Complete Netzwerk, die ohne Kaskadierung sind, noch weniger Zeit. Dies wird an der Implementierung der Netze liegen, da bei jedem Kaskadennetzwerk das Outputlayer in jeder Kaskade mit berechnet wird, während bei den Complete Netzwerken nur ein einziges Outputlayer existiert. Des Weiteren werden keine extra Predictions und keine Berechnung der Augmented Vectors gemacht.

# Chapter 4

# Klassifikation

Hier werden kurz die Direct Cascade Netze für die Klassifikation vorgestellt. Die Besonderheit eines solchen Netzwerkes ist es, dass es immer nur ein einziges Hiddenlayer existiert und die Netze so iteriert werden, dass sie das Wissen der vorherigen mitnehmen.

Table 4.1: Direct Cascade Networks

| **Name** | **Hiddenlayer** | **Nodes/Filter** | **Aktivierung** | **Input** |
|---|---|---|---|---|
| ClassOneDense | Linear | 512 | Relu | 1 |
| 1DConv | 1DConv | 32 | Relu | 1 |
| 2DConv | 2DConv | 32 | Relu | 2 |

Dabei ist der Input die Dimension in der die Bilddaten vorliegen und nur in der ersten Zeile sind es Nodes, sonst sind es die Filter. Bei beiden Convolution-Netzen wird ein Kern der Größe 3 beziehungsweise 3x3 verwendet mit einem solchen Padding, dass die Größe der Daten dabei nicht verändert wird. Bei diesen Netzen wird als Batch jeweils 128 Datensamples genutzt.

## 4.1 Größe des Targetdatensatzes

Hier wird auf die Änderungen der Performanz der Netze eingegangen, die dadurch entstehen, wenn der Targetdatensatz unterschiedliche Größen hat. Die Vermutung ist, dass es schlechter wird, je weniger Daten vorhanden sind.

Um Vergleiche zu haben, werden wieder insgesamt 40 Epochen trainiert und nach zwanzig TF vollzogen. Dies wird mit den Netzwerken ConvMaxPool, 1DConv und ClassOneDense vollzogen. Hier wird auch auf den Testdatensatz eingegangen.

In Figure 4.1 sieht man die Testdatenläufe bezüglich der Menge der Trainingsdaten. Dabei ist die erste Zahl in der Legende die Menge und die Zweite die Dauer. Dabei bezieht die Menge sich nur auf den Targetdatensatz. Es wird

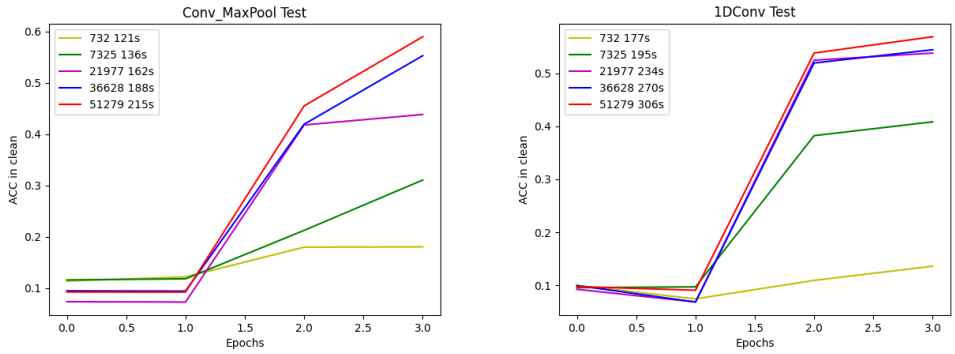


Figure 4.1: Vergleich zwischen Deep und Direct Cascade

deutlich, dass es länger dauert, je mehr Daten vorhanden sind. Ebenso bestätigt sich die Vermutung, dass es auch bei Kaskadennetzwerken mit TF besser wird, je mehr Daten vorhanden sind. Ebenfalls zeigt sich, dass Deep Cascade etwas besser ist als Direct Cascade. Dies dürfte daran liegen, dass Direct Cascade Netzwerke sind, die nur ein Hidden Layer haben, während es bei Deep Cascade mehrere sind.

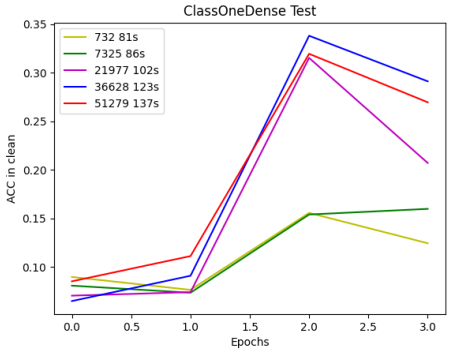


Figure 4.2: Targetgrößenänderung bei nur Linearlayern

In Figure 4.2 ist das Deep Cascade Netzwerk, welches als Hidden Layer ausschließlich Linearlayer hat. Auffällig ist, dass dieses bei egal wievielen Datensamples immer schlechter abschneidet als die anderen beiden Netzwerke, die Convolutionlayer besitzen.

Generell ist es bei allen Testplots so, dass sie nie eine annehmbar hohe Accuracy besitzen und das auch dann nicht, wenn es genügend Daten gibt, damit auf dem Targetdatensatz direkt gelernt werden könnte. Wenn dies mit dem Deep Cascade Netzwerk gemacht wird, kommt dabei eine Accuracy von etwa 70% heraus, was deutlich besser als jedes TF Netzwerk ist.

## 4.2 Bilddimensionalität

Bei den beiden Convolution Direct Cascade Netzwerken ist der einzige Unterschied, dass sie die Bilder in ein- beziehungsweise in zweidimensionaler Form sehen. Dabei fällt aber auf, dass es im zweidimensionalen Fall etwas besser ist. Dies liegt daran, dass das eindimensionale Netz in dem Filterlayer nur die Daten direkt rechts und links mit einbezieht. Das zweidimensionale Netzwerk hingegen nutzt bei der Operation jenes Layers nicht nur die direkt rechts und links, sondern auch die Daten, die oben und unten angrenzend sind, sowie die Daten, die in jede Richtung schräg vorkommen. Die Verbesserung ist aber nur minimal, wie in Figure 4.3 zu sehen.

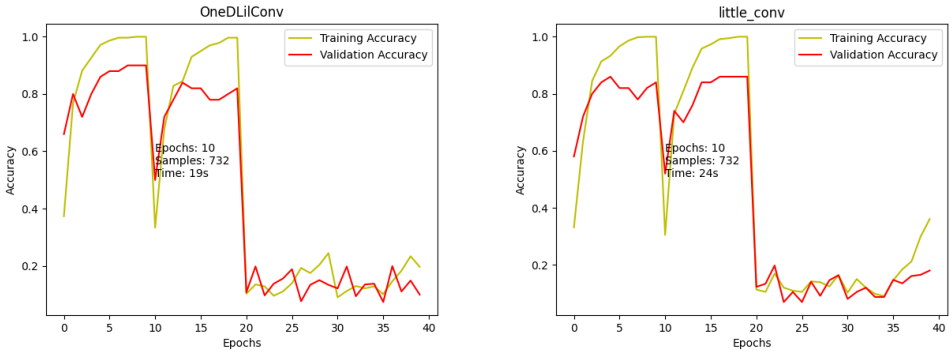


Figure 4.3: Vergleich 1D mit 2D

Da das zweidimensionale Netzwerk mit nicht so vielen Daten genutzt werden kann, hat es hier eine sehr viel kürzere Zeit. Es kann deshalb nicht genutzt werden, da die Berechnung des Augmented Vectors zu Speicherplatzproblemen im Arbeitsspeicher führt.

Weil diese Veränderung nur minimal ist, reicht es nur das eindimensionale Netz in den meisten Fällen zu betrachten, weshalb die technischen Probleme beim zweidimensionalen Netzwerk nicht so hinderlich für die Evaluierung von TF sind.

## 4.3 Augmentierung

Hier wird auf die Erstellung der Augmented Vectors der Direct Cascade Netzwerk für die Klassifikation eingegangen. Jedes der drei Netze hat eine eigene Berechnung davon. Es wird zuerst das ClassOneDense-, dann das 1DConv- und zum Schluss das 2DConv-Netzwerk betrachtet.

Bei allen Netzwerken wird der Input des Netzwerkes und die Prediction gebraucht, um diesen Vector zu erstellen. Der Input ist entweder der Datensatz selbst oder der vorherige Augmented Vector. Nur beim ersten Mal ist es der Datensatz, da danach der Augmented Vector existiert. Der Augmented Vector

wächst dabei bei jeder Iteration von einem neuen Netzwerk an, da darüber das Wissen aller vorherigen auf das neue übertragen wird. Die Prediction ist das Ergebnis, welches aus der Inferenz des fertig trainierten Netzwerkes kommt. Im folgenden bedeuten die Buchstaben N, W, H und C Datensamples, Bildbreite, Bildhöhe und Channel.

Das ClassOneDense-Netzwerk hat als Input den Datensatz mit folgendem Shape: (N, W*H). Die Prediction hat immer den Shape (N, 10). Diese beiden Sachen werden in der zweiten Dimension konkateniert. Dies ergibt die Formel 4.1 und damit den Augmented Vector.

$$AugVec(Input(N, W * H), Prediction(N, 10)) = (N, (W * H).10) \tag{4.1}$$

Das 1DConv-Netzwerk hat als Input hingegen den Datensatz in folgendem Shape: (N, W*H, C). Da der Channel nur eindimensional ist, wird dieser zuerst entfernt und dann die Berechnung nach der Formel 4.1 durchgeführt. Zum Schluss wird die Channeldimension wieder hinzugefügt. Beide bisher behandelten Netzwerke haben somit einen um N*10 Einträge linear wachsenden Augmented Vector.

Das 2DConv-Netzwerk hat einen komplexeren Input mit dem Shape: (N, W, H, C). Dies muss verbunden werden mit der Prediction die in der Form (N, 10) vorliegt. Dafür wird für jedes N zehn Arrays gebaut, die alle die Form (W, H, C) haben. Diese haben von eins bis zehn den Inhalt der Prediction. Dies wird dann auf der Channeldimension konkateniert. Daraus resultiert die Formel 4.2.

$$\begin{aligned} AugVec(Input(N, W, H, C), Prediction(N, 10)) = Input(N, W, H, C.ConVec) \\ ConVec(W, H, C)[0-9] = Prediction(10)[0-9] \end{aligned} \tag{4.2}$$

Dabei ist der ConVec der Vector in dem die Predictionwerte von eins bis zehn jeweils in der Form (W, H, C) enthalten sind. Dies skaliert bei den Netziterationen im Speicherplatz aber in der Form, wie es in Formel 4.3 zu sehen ist.

$$AugVecNew = N * W * H * C_{old} + N * W * H * 10 \tag{4.3}$$

Daraus ergibt sich, dass der Arbeitsspeicherplatz mit einer Steigung von dem Zehnfachen des Datensatzes zunimmt. Bei Daten, die in einem Sample bereits 8192 Bytes benötigen, ist klar, dass diese Variante des Bauens des Augmented Vectors nicht durchgeführt werden sollte, da dieser zu schnell zu groß wird. Aus diesem Grund wird das 2DConv-Netzwerk im Folgenden nicht mehr verwendet.

## 4.4 Mit und Ohne

### 4.4.1 TF

Hier werden die Netze jeweils einmal mit und einmal ohne Transferlernen ausgetestet. Es werden nur die Direct Cascade Netzwerke betrachtet und sie wer-

den mit deutlich mehr Netziterationen trainiert als bisher. Die Epochenanzahl pro Netzwerk bleibt aber gleich. Dabei werden jeweils nur wenig Targetdaten verwendet.

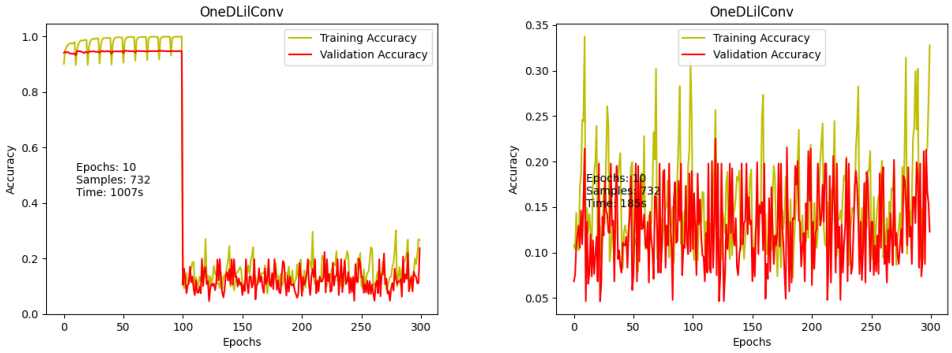


Figure 4.4: 1DConv TF Vergleich

Wie in Figure 4.4 zu sehen gibt es keinen Unterschied zwischen der Accuracy mit TF zu der ohne bei Convolutionlayern. In beiden Fällen ist diese extrem schlecht. Dies zeigt sich auch auf den Testdaten.

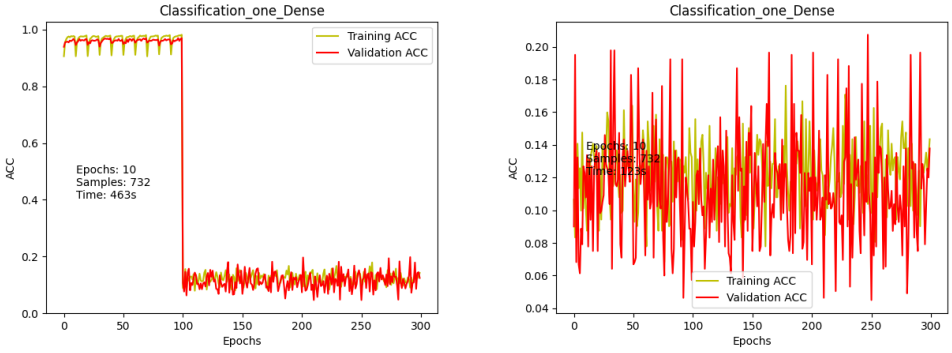


Figure 4.5: ClassOneDense TF Vergleich

In Figure 4.5 zeigt sich dasselbe Bild nur auf Basis von Linearlayern. Dies kann zwei Gründe haben: Entweder funktioniert das Kaskadieren nicht oder es sind nicht genügend Targetdaten vorhanden. Letzteres wurde oben ausgetestet und lieferte zwar bessere, aber trotzdem nur mäßige Ergebnisse. Das Rauschen in den Plots kommt hier daher, dass alle zehn Epochen ein neues Netzwerk angefangen wird zu lernen. Dies hat zwar das Wissen aller vorherigen Netze im Input, aber nicht in der Art, dass die Gewichte direkt gleich gut sind.

### 4.4.2 Kaskadierung

Also wird hier ausgetestet was passiert, wenn nicht kaskadiert wird. Da dies nur dann gut geht, wenn TF nicht verwendet wird, wird alles direkt auf dem Targetdatensatz gelernt. Die Netze werden so verändert, dass sie insgesamt gleich viele Hiddenlayer wie alle kleinen Netze, die im Direct Cascade Verfahren vorkommen, zusammen haben. Es werden dabei auch gleich viele Epochen insgesamt benutzt wie eben gerade.

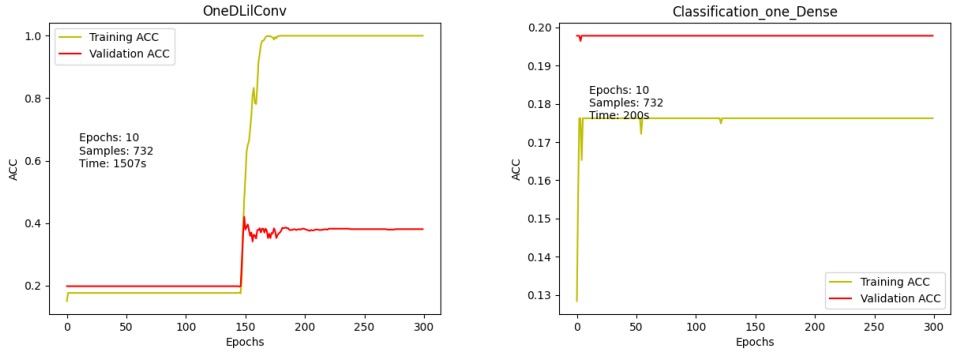


Figure 4.6: Netze ohne Kaskadierung

Was in der Figure 4.6 auffällt ist, dass es bei den meisten Epochen zu keinem Lerneffekt kommt. Ebenso kommt Overfitting vor, was bei so wenigen Daten zu erwarten ist. Obwohl es hier nicht TF angewendet wird, gibt es in der Mitte des einen Plot einen plötzlichen Anstieg der Accuracy. Der Start dieser Verbesserung kam davon, dass der Validationwert minimal schlechter wurde, während der Trainingswert minimal besser wurde. Beide Veränderungen waren im Bereich von Zehnteln der Prozente. Deswegen scheint es so, dass es zu einem lokalen Maximum während des Trainings gekommen ist. Bei dem anderen Netz blieb der Wert auf dem des lokalen Maximums stehen, denn es sind die exakt gleichen Werte. Trotzdem wird durch Figure 4.6 klar, dass es bei so wenigen Daten eine maximale Accuracy von 40% zu erwarten ist. Dies ist das globale Maximum, da es den maximal möglichen Wert auf den Trainingsdaten vorweist. An diese kommt weder die Version des nur Kaskadierens noch die des Kaskadierens mit TF auch nur ansatzweise heran. Diese haben einen maximalen Wert von 20% und sind somit nur halb so gut.

Daraus folgt, dass es bereits an der Kaskadierung liegt, dass Klassifikation sinnlos mit TF in dem Direct Cascade Verfahren ist.

# Chapter 5

# Regression

Hier werden die beiden Regressionsnetze vorgestellt. Beide haben als Input Tabellen mit drei Spalten. Welche das genau sind, wurde oben bereits erklärt. Sie haben ebenfalls beide den Adam Optimizer mit der Mean Squared Error-Lossfunction. Als Outputlayer wird für Regression typisch ein einzelnes Linearlayer mit einer Node und der Linear Activation Function genutzt.

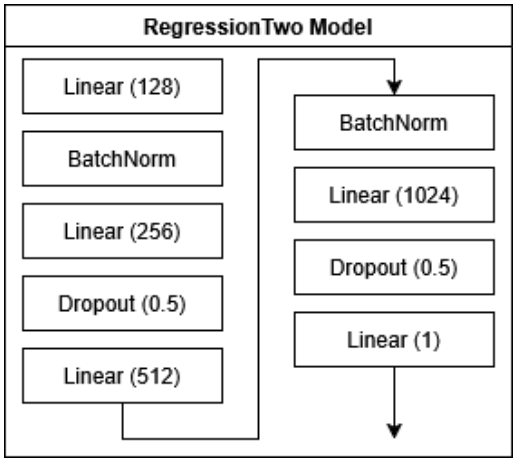


Figure 5.1: Vorstellung RegressionTwo Netzwerk

In Figure 5.1 ist das RegressionTwo-Netzwerk mit allen seinen Layern. Dies ist ein Deep Cascade Netzwerk. Es wird also Layer für Layer trainiert. Dabei ist die Zahl hinter Linear die Anzahl der Nodes und die Zahl hinter Dropout die Prozente bezüglich dem Wert eins, die während des Trainings pro Epoche wegfallen.

Das OneLayer-Netzwerk ist das Direct Cascade Regressionsnetz. Dieses hat nur ein Hiddenlayer mit einem Linearlayer mit 128 Nodes. Die Aktivierungsfunktion in diesem Layer ist Relu. Es wird iterativ genutzt und zwischen den

Netzen wissen mittels eines Augmented Vectors als neuen Input übertragen. Dieser wird mit der Prediction des vorherigen Netzes berechnet, indem diese als neue Spalte in der Inputtabelle des bisherigen Inputs hinzugefügt wird. Dies ist der Augmented Vector, der als neuer Input für das nächste Netz dient.

## 5.1 Datenaugmentation

Der Sourcedatensatz Bost hat nur 506 Datensamples insgesamt und ist somit sehr klein. Davon werden im folgenden 51 Samples als Testset, 91 als Validationset und 364 als Trainingsset genutzt.

Der Targetdatensatz Cali ist hingegen mit etwa 26 Tausend Samples sehr groß und wird nach Bedarf verkleinert. Es werden jeweils als Batch 16 Samples genutzt.

### 5.1.1 Viele Daten

Hier werden alle Vergleiche bei vollem Targetdatensatz verwendet. Dadurch umfassen die Trainingsdaten etwa 8000 Samples und die Testdaten etwa 4000. Diese Vergleiche beinhalten Komplette, TF Cascade und Cascade Netzwerke. Dabei wird als komplettes Netzwerk ein Netzwerk, welches alle Layer vorher definiert hat und dieses eine Netzwerk mit einem einzigen Trainingsaufruf alles trainiert, was der normale Fall eines neuronalen Netzwerks ist.

Es wird sowohl der Vergleich zwischen Deep Cascade, Direct Cascade und dem Kompletten Netzwerk gemacht.

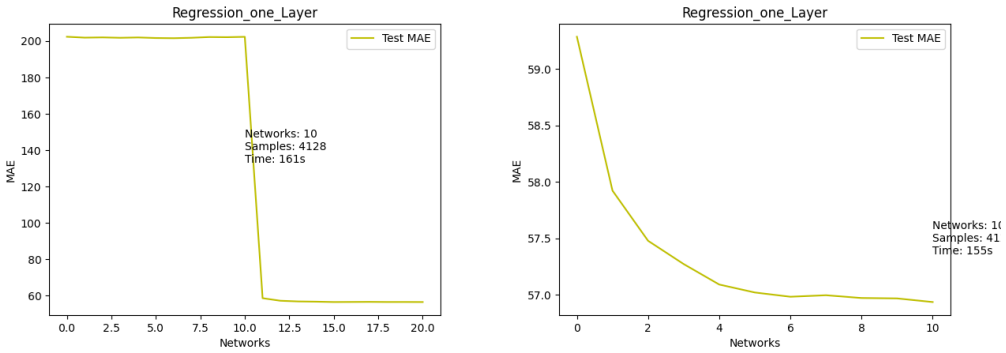


Figure 5.2: Vergleich im OneLayer Netzwerk

In Figure 5.2 ist das Ergebnis des Direct Cascade. Auf der linken Seite ist die Versioin mit TF. Dabei fällt auf, dass es bei vielen Targetdaten besser ist auf dem Targetdatensatz direkt zu lernen, denn das Wissen, welches vom Sourcedatensatz übertragen wird, ist nicht so passend, wie das von den Targetdaten. Dies passiert aber nur, wenn es genügend Targetdaten gibt. Dabei kommt auch beim Deep Cascade ähnliche Werte heraus. Ein komplettes Netzwerk, wie es

für Figure 5.3 genutzt wurde, hat etwa dieselbe Performanz, wie die beiden Kaskadenversionen.

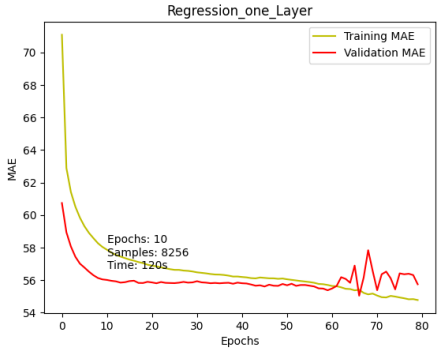


Figure 5.3: Komplettes OneLayer

Dies liegt daran, dass die lineare Aktivierungsfunktion und der Mean Squared Error Loss für die Kaskadierung nicht störend sind. Also funktioniert Regression deutlich besser mit Kaskadennetzwerken als die Klassifikation, da sie an das Ergebnis des kompletten Netzes herankommt, was bei Klassifikation nie passiert ist.

### 5.1.2 Wenig Daten

In diesem Unterkapitel wird der Targetdatensatz deutlich verkleinert und hat dann nur noch 240 Datensamples. Es werden dieselben Tests wie im vorherigen Unterkapitel durchgeführt.

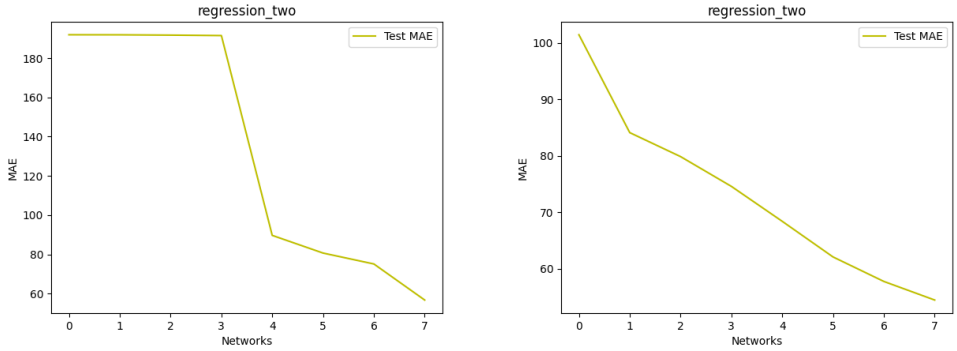


Figure 5.4: Vergleich RegressionTwo Netzwerk

In Figure 5.4 sind die Ergebniss des Deep Cascade Netzwerks. Es ist ohne TF tatsächlich besser als mit. Dies liegt daran, dass die Gewichte der ersten

Hälfte des Netzes nur auf dem Sourcedatensatz passend gelernt haben.

Es gibt aber deutliche Unterschiede zu Direct Cascade, weshalb beide hier gezeigt werden.

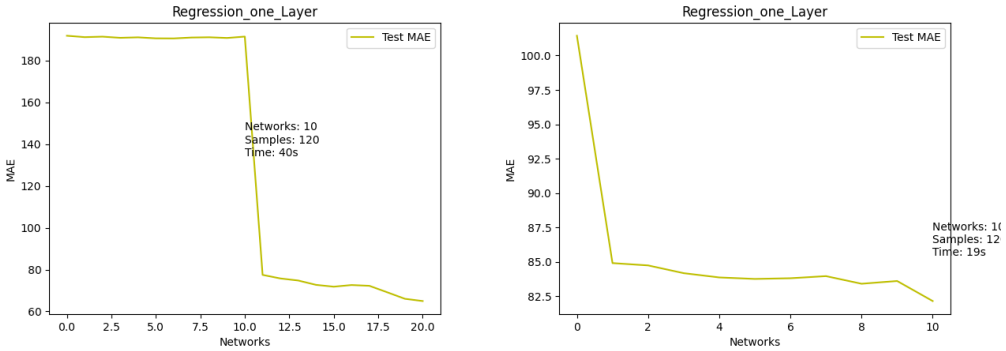


Figure 5.5: Vergleich OneLayer Netzwerk

Figure 5.5 bezieht sich auf das Direct Cascade Netzwerk. Es wird deutlich, dass in dieser Kaskadierungsvariante das Netz deutlich schlechter ohne TF ist als mit. Die liegt daran, dass das TF über den Augmented Vector als veränderten Input funktioniert und nicht über Gewichte, die von dem Sourcedatensatz fertig gelernt worden sind. Allerdings ist das Deep Cascade Netzwerk trotzdem besser. Dies kann aber auch an dem dahinter liegenden Netz liegen, da sie nicht nur die gleichen Layer haben. Noch besser ist die Variante, die weder Kaskadierung noch TF nutzt, wie in Figure 5.6 gezeigt.

Der MAE-Wert der Testdaten beläuft sich hier auf 53 Tausend Dollar, während dieser sonst bei so wenig Datensamples bei 60 bis 80 liegt.

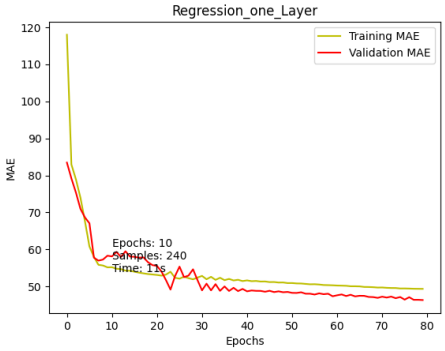


Figure 5.6: Komplettes OneLayer

## 5.2 Early Stopping

Hier werden die Regressionsnetze mit Early Stopping verwendet und auch erklärt, warum das bei Klassifikation sinnlos ist. Dabei werden nur die Direct Cascade Netze betrachtet.

Sowohl für die Regressionsnetze als auch für die Klassifikationsnetze wurde LM verwendet. Zudem für Regression noch MAEM und für Klassifikation ACCM.

Bei der Klassifikation kommt es nur manchmal zu einem Abbruch der Epochen über das ACCM, aber es wird dadurch nicht besser. Mit LM kommt dieser Abbruch öfter vor und das Training geht somit zwar schneller, jedoch bleibt Klassifikation mit Kaskadierung so schlecht, dass es nicht genutzt werden kann. Dass weder LM noch ACCM funktioniert sieht man deutlich in Figure 5.7. ACCM ist die einzige der hier vorkommenden Metriken, dessen Ziel ein Maximum ist.

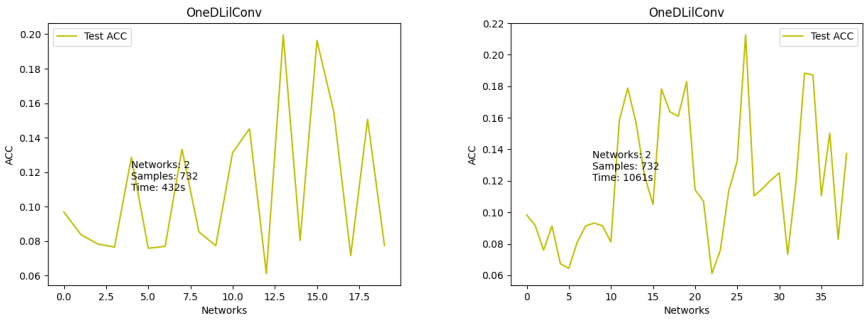


Figure 5.7: LM und ACCM mit 1DConv

Deshalb wird sich hier eingehender mit dem Regressionsnetz OneLayer befasst. Die Metriken LM und MAEM suchen dabei ein Minimum.

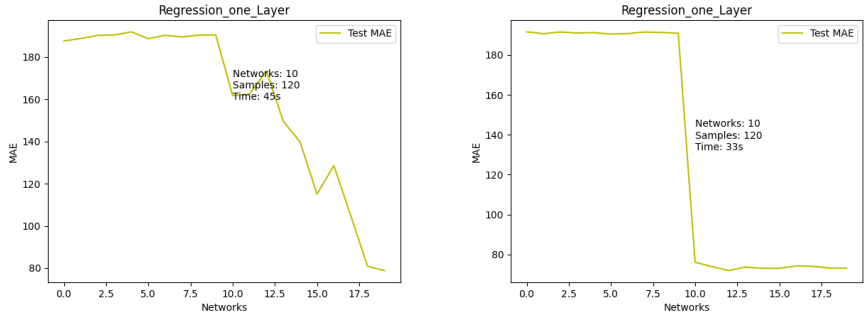


Figure 5.8: LM und MAEM mit OneLayer

Dieses liefert mit den beiden Early-Stopping Metriken LM und MAEM halb-

wegs brauchbare Ergebnisse, jedoch sind diese deutlich schlechter als ein Training ohne diese, wie an den Werten von Figure 5.8 abgelesen werden kann.

Diese Werte sind so schlecht als hätte man das OneLayer Netzwerk mit wenigen Targetdaten direkt auf diesen Datensatz lernen lassen. Das diese Early-Stopping Metriken so schleht sind, liegt daran, dass sie keine Verschlechterung im Validationset des Datensatzes dulden und ab der ersten das Netz der aktuellen Netziteration beenden. Dadurch ist selten das tatsächliche Minimum das, was über den Augmented Vector weitergegeben wird, sondern nur ein leicht abweichendes. Dazu kommt, dass diese Metriken nicht das globale Minimum finden können, wenn sie auf ein lokales treffen, denn sie werden Versuchen in diesem zu verbleiben.

# Chapter 6

# Diskussion

## 6.1 Erkenntnisse

Allgemeines:

Das ist alles ein Plot eines DeepCascade Networks, welches vorgestellt wird: Es kommt relativ schnell zu Overfitting auf dem Sourcedatensatz.

Bei der Stelle des TFs bricht die Performanz des Netzes ein, nur bei deep Cascade erholt sich das ein wenig.

Die normalen Netzwerke sind im kleinen schneller als die Direct Cascade und diese wiederum schneller als die Deep Cascade Netze. Sie sind in der Performanz aber unterschiedlich gut.

Klassifikation:

Bei der Klassifikation ist der endgültige Accuracy-Wert hauptsächlich von der Datenmenge des Targetdatensatzes abhängig. Je weniger Daten, desto schlechter ist der Wert.

Klassifikation läuft so schlecht, dass man es nicht tun sollte, da es nichts bringt.

Bei TF ist die Performanz des Netzes etwas schlechter als ohne. Bei Cascade Networks ist es ebenfalls schlechter.

Die Erstellung bei mehrdimensionalen Augmented Vectors kann zur Arbeitsspeicherplatzproblemen führen.

Eindimensionale Klassifikation ist minimal schlechter als zweidimensionale. Dies ist aber so gering, dass es Vernachlässigbar ist.

Regression:

Die Regression ist mit TF bei wenigen Daten besser als, wenn sie auf den Targetdatensatz von Scratch lernt. Aber nur, wenn es sich um Direct Cascade handelt.

In der Regression ist der Abfall der Performanz bei weitem weniger groß.

Die simplen hier aufgeführten Early-Stopping Metriken verschlechtern das Ergebnis.

## 6.2 Ausblick

Hier wird ausgeführt, was alles noch in die Richtung gemacht werden kann.

## 6.3 Fazit

Hier werden die abschließenden Worte stehen.

# Bibliography


[Har+97] David Harrison et al. *Corrected Version of Boston Housing Data*. StatLib Library from Carnegie Mellon University. 1997.

[HW20] Yaoshiang Ho and Samuel Wookey. "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling". In: *IEEE Access* 8 (2020), pp. 4806–4813. DOI: 10.1109/ACCESS.2019.2962617.

[JY10] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010).

[LeC+] Yann LeCun et al. *Learning Algorithms for Classification: A Comparison on handwritten digit recognition*.

[Net+11] Yuval Netzer et al. "Reading Digits in Natural Images with Unsupervised FEature Learning". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. Google Inc., Mountain View, CA and Stanford University, Stanford, CA, 2011.

[Nug18] Cam Nugent. *California Housing Prices*. online at www.kaggle.com. 2018.