# Evaluierung von Transferlernen mit Deep Direct Cascade Networks

Simon Tarras

July 13, 2025

# Contents


**1 Einleitung** **3**
- 1.1 Einführung . . . 3
- 1.2 Motivation . . . 3
- 1.3 Related Work . . . 4
- 1.4 Ausblick in diese Arbeit . . . 7

**2 Methodik** **8**
- 2.1 Transferlernen . . . 8
- 2.2 Kaskadierung . . . 10
  - 2.2.1 Deep Cascade . . . 11
  - 2.2.2 Direct Cascade . . . 11
- 2.3 Setup . . . 14
- 2.4 Metrik . . . 15
- 2.5 Liste der Tests . . . 15

**3 Konsistente Auffälligkeiten** **18**
- 3.1 Ploterklärung . . . 18
- 3.2 ConvMaxPool . . . 19
  - 3.2.1 Veränderungen bei TF . . . 20
  - 3.2.2 Overfitting auf Source-Datensatz . . . 21
- 3.3 Zeitnahme . . . 23

**4 Klassifikation** **26**
- 4.1 Größe des Target-Datensatzes . . . 27
- 4.2 Bilddimensionalität . . . 28
- 4.3 Augmentierung . . . 29
- 4.4 Mit und Ohne . . . 31
  - 4.4.1 TF . . . 31
  - 4.4.2 Kaskadierung . . . 33

**5 Regression** **35**
- 5.1 Datenaugmentation . . . 36
  - 5.1.1 Viele Daten . . . 36
  - 5.1.2 Wenig Daten . . . 37

5.2 Early Stopping . . . . . 41

**6 Zusammenfassung** **43**
6.1 Erkenntnisse . . . . . 43
6.2 Ausblick . . . . . 44
6.3 Fazit . . . . . 45

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# Chapter 1

# Einleitung

## 1.1 Einführung

Künstliche Intelligenz (KI) ist mittlerweile selbst außerhalb der Informatik einer breiten Öffentlichkeit bekannt. Die zugrunde liegende Technologie basiert in der Regel auf künstlichen neuronalen Netzen, die üblicherweise in einem einzigen Schritt vollständig konstruiert und trainiert werden. Dieser Prozess ist jedoch zeit- und rechenintensiv, weshalb alternative Verfahren wie die Cascade-Correlation-Methode entwickelt wurden [ML90]. Erste Untersuchungen zeigten, dass diese Kaskadierungsstrategie bereits bei kleineren Netzarchitekturen zufriedenstellende Ergebnisse liefert. Darauf aufbauend wurden verschiedene erweiterte Netzwerkstrukturen und Kaskadierungsverfahren entwickelt [LR92b], [TG99], [MHN18].

Ein wesentlicher Vorteil kaskadierender Netzwerke liegt in ihrer modularen Struktur, die eine flexible Anpassung an unterschiedliche Datensätze und Aufgabenstellungen ermöglicht [Mar19], [TS], [JY10]. Die vorliegende Arbeit ist im Kontext dieser Transferfähigkeit einzuordnen. Ziel ist es, die Leistungsfähigkeit verschiedener Netzwerkarchitekturen zu evaluieren und spezifische Herausforderungen im Zusammenhang mit Transferlernen (TF) und Kaskadierung zu analysieren.

## 1.2 Motivation

In vielen Anwendungsfällen, für die der Einsatz von KI grundsätzlich sinnvoll wäre, stehen lediglich sehr kleine Datensätze zur Verfügung. Diese Datenmengen sind oft so begrenzt, dass ein neuronales Netzwerk nicht ausreichend Informationen erhält, um ein robustes und generalisierbares Modell zu erlernen.

Zusätzlich stellt die lange Trainingszeit klassischer neuronaler Netzwerke eine weitere Herausforderung dar. Beide Probleme – unzureichende Datenverfügbarkeit und hoher Trainingsaufwand – sollen in dieser Arbeit adressiert und verbessert werden.

Zur Bewältigung der Problematik kleiner Datensätze wird ein auf TF basierender Ansatz verfolgt. Dabei lernt das Modell zunächst auf einer verwandten, aber besser verfügbaren Datenbasis, um dieses Vorwissen anschließend auf die eigentliche Zielaufgabe zu übertragen. Dieses Vorgehen orientiert sich an kognitiven Lernprozessen beim Menschen, bei denen bereits erlernte Konzepte genutzt werden, um neue, ähnliche Inhalte zu erschließen.

Die Netzwerkarchitektur wird derart gestaltet, dass jeweils nur ein geringer Teil des Modells gleichzeitig trainiert wird. Dadurch soll der Trainingsaufwand reduziert und eine signifikante zeitliche Effizienzsteigerung erzielt werden. Zur Umsetzung dieses Ansatzes kommen die Kaskadierungsverfahren Deep Cascade und Direct Cascade zum Einsatz.

## 1.3 Related Work

Diese Arbeit basiert auf dem CasCor-Algorithmus [ML90], der ursprünglich entwickelt wurde, um der hohen Trainingsdauer klassischer neuronaler Netze entgegenzuwirken. Aufbauend auf diesem Verfahren wird die Methode Direct Cascade implementiert, wie sie unter anderem in mehreren Papern von Ritter und Littmann [LR92a; LR92b] beschrieben ist. Als Vergleichsmodell dient das Verfahren Deep Cascade [MHN18], bei dem Netzwerke schrittweise und iterativ erweitert werden [TG99]. Dieses wurde mit TF bereits in der Dissertation von Marquez [Mar19] verwendet.

Im Rahmen dieser Arbeit wird ausschließlich der Domänenwechsel betrachtet, während der Aufgabenwechsel (Task Transfer) [TS] unberücksichtigt bleibt. Darüber hinaus werden drei zentrale Herausforderungen des TFs behandelt, wie sie in [JY10] identifiziert wurden.

Neuronale Netzwerke werden für eine Vielzahl von Aufgaben eingesetzt, die ein rechenintensives Vorgehen erfordern und dabei eine Architektur nutzen, die dem menschlichen Gehirn nachempfunden ist. Neben klassischen Anwendungen im Bereich der Computer Vision existieren auch Einsatzgebiete in der Steuerung und Planung, wie sie beispielsweise bei strategischen Spielen wie Go relevant sind [CK19].

Ein weiteres Anwendungsfeld stellt die Klassifikation dar, etwa bei der Erkennung handgeschriebener Ziffern [LeC+]. Zudem werden neuronale Netzwerke zur Approximation von Funktionen eingesetzt, typischerweise im Rahmen von Regressionsaufgaben [Spe91].

In der Praxis ergeben sich jedoch verschiedene Herausforderungen. Eine häufige Problematik ist die unzureichende Verfügbarkeit großer, qualitativ hochwertiger Datensätze für bestimmte Aufgabenstellungen [JY10]. Darüber hinaus kann das Training eines leistungsfähigen neuronalen Netzwerks äußerst zeitintensiv sein [ML90]. Generell gilt für alle neuronalen Modelle, dass ihre Vorhersagen mit einer gewissen Fehlerrate behaftet sind und somit nicht vollständig korrekt sind [RCN07].

Bei unzureichender Verfügbarkeit von Trainingsdaten wird TF eingesetzt.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

TF bezeichnet ein maschinelles Lernparadigma, bei dem bereits erworbenes Wissen aus einer Quellaufgabe (Source) genutzt wird, um die Lernleistung auf einer Zielaufgabe (Target) zu verbessern [TS]. Dabei erlernt ein neuronales Netzwerk zunächst eine Aufgabe und überträgt anschließend relevante Erkenntnisse auf eine andere, meist verwandte Aufgabe [Mar19].

Dieses Vorgehen lässt sich mit dem menschlichen Lernverhalten vergleichen, bei dem über Eselsbrücken bereits bestehendes Wissen herangezogen wird, um neues Wissen effizienter zu erwerben. Ein Netzwerk, das TF betreibt, vollzieht folglich einen kontextuellen Wechsel zwischen verschiedenen Lernaufgaben. Dabei ergeben sich insbesondere drei zentrale Forschungsfragen:

1. What to transfer
2. How to transfer
3. When to transfer

[JY10]

Die erste zentrale Forschungsfrage im TF betrifft die Beschaffenheit und Auswahl der Source- und Target-Daten zwischen denen TF angewandt werden soll.

Die zweite Fragestellung bezieht sich auf die konkrete Umsetzung des TF in der Praxis. Dabei wird untersucht, auf welche Weise neuronale Netze das Wissen aus den Source-Daten extrahieren und wie dieses Wissen im weiteren Verlauf verarbeitet und genutzt wird.

Die letzte Fragestellung bezieht sich darauf, wann der Übergang vom Lernen auf der Source- zur Target-Aufgabe erfolgen soll. In der Praxis wird dieser Übergang häufig durch Fehlermetriken gesteuert: Sobald das Modell einen bestimmten Schwellenwert unterschreitet und die Fehlerquote als ausreichend niedrig eingestuft wird, erfolgt der Transfer. Der optimale Zeitpunkt für diesen Wechsel kann jedoch experimentell bestimmt und variiert werden, um maximale Leistung zu erzielen. Es ist jedoch wichtig zu beachten, dass TF nicht zwangsläufig zu einer Leistungssteigerung führt. In bestimmten Fällen kann die Übertragung von Wissen sogar zu einer Verschlechterung der Modellleistung führen – ein Phänomen, das als Negative Transfer bezeichnet wird [JY10].

TF setzt in mindestens einem von zwei Bereichen einen Wechsel voraus: der Domain oder dem Task. Die Domain umfasst Merkmale wie die Verteilung, Repräsentation oder den strukturellen Aufbau der Daten (z.B. Formate, Dimensionen), während der Task sich auf die konkrete Lernaufgabe bezieht, wie beispielsweise Bildklassifikation oder Segmentierung [JY10].

In komplexeren Szenarien kann der Transfer nicht direkt von der Source- zur Target-Domain erfolgen. Stattdessen wird eine intermediäre, sogenannte Brückendomain eingeführt, über die der Wissenstransfer schrittweise erfolgt. Dieses Vorgehen wird als Bridge Transfer Learning bezeichnet und findet Anwendung, wenn die Unterschiede zwischen Source und Target zu groß sind oder die direkte Übertragung zu negativem Transfer führen würde [KCR17; JY10].

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Während Deep Learning grundsätzlich in der Lage ist, komplexere Aufgaben zu lösen als sogenanntes Shallow Learning, ist es gleichzeitig durch einen erheblich höheren Trainingsaufwand gekennzeichnet. Wenn dieser hohe Rechenaufwand nicht praktikabel ist, kommt Cascade Learning zum Einsatz [ML90].

Im klassischen Deep Transfer Learning wird in der Regel ein vortrainiertes Netzwerk genutzt, wobei der finale Feature-Vektor als Ausgangspunkt für den Transfer dient. Dies beruht auf der Annahme, dass in dieser Repräsentation die bedeutendsten Informationen verdichtet vorliegen – ein Ansatz, der als Feature Representation Transfer bezeichnet wird [JY10].

Im Gegensatz dazu verfolgt das Cascade Learning einen iterativen Ansatz: In jeder Trainingsiteration wird der aktuell generierte Feature-Vektor in den Transferprozess einbezogen, um herauszufinden, welcher Repräsentationsvektor tatsächlich den größten Informationsgehalt besitzt. Dieser Ansatz führt in der Regel zu einer effizienteren und weniger komplexen Netzwerkstruktur im Vergleich zum klassischen Deep Transfer Learning [Mar19].

Cascade-Netzwerke zeichnen sich dadurch aus, dass ihre Architektur nicht vor Beginn des Trainings vollständig definiert ist, sondern schrittweise während des Trainingsprozesses aufgebaut wird. Hierfür kommen sogenannte Constructive Algorithms zum Einsatz, deren wesentlicher Vorteil darin liegt, dass keine exakte Festlegung der Netzwerkgröße im Vorfeld erforderlich ist [TG99]. Das Netzwerk wächst lediglich so weit, wie es zur Lösung der jeweiligen Aufgabe notwendig ist, wodurch eine übermäßige Modellkomplexität vermieden und gleichzeitig die Trainingszeit reduziert wird [TG99; ML90].

Ein prominentes Beispiel für einen solchen Ansatz ist der Cascade Correlation Algorithmus (CasCor). Bei diesem Verfahren wird das Netzwerk schrittweise durch das Hinzufügen einzelner Neuronen (Perzeptrons) erweitert. Jedes neue Perzeptron wird so trainiert, dass es eine möglichst hohe Korrelation mit dem verbleibenden Fehler aufweist. Nach Abschluss des Trainings wird das betreffende Neuron fixiert, das heißt, seine Gewichtungen werden eingefroren und bleiben im weiteren Verlauf unverändert. Anschließend wird überprüft, ob das Netzwerk die angestrebte Fehlergrenze erreicht hat. Ist dies nicht der Fall, wird ein weiteres Perzeptron ergänzt und der Prozess wiederholt sich, bis die gewünschte Modellgüte erreicht ist [ML90].

Dieser iterative Aufbau sowie das Einfrieren bereits gelernter Gewichtungen ermöglichen ein effizientes Training, bei dem keine Backpropagation durch das gesamte Netzwerk erforderlich ist. Lediglich die Gewichte des neu hinzugefügten Neurons werden jeweils angepasst. Da jedes neue Neuron direkt mit dem Ausgang des Netzwerks verbunden ist, trägt es unmittelbar zur Fehlerkorrektur bei [ML90].

Ein verwandtes Konzept stellt das Direct Cascade Learning dar. Hierbei erfolgt keine Zwischenverarbeitung der Ausgaben bereits trainierter Neuronen. Stattdessen dient der Output eines Perzeptrons direkt als Input für das nachfolgende, wie es beispielsweise in den Architekturen Cascade QEF und Cascade LLM umgesetzt ist [LR92b; LR92a].

Obwohl TF bislang vorrangig mit klassischen tiefen Lernverfahren kombiniert wurde, existieren erste Ansätze, die TF mit Cascade Learning verknüpfen.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Ein Beispiel hierfür ist das Cascade Transfer Learning (CTL), das CasCor mit TF kombiniert [Mar19]. Zwar blieb die Leistung von CTL leicht hinter der von klassischen Fine-Tuning-Verfahren zurück, jedoch zeichnete sich CTL durch eine deutlich geringere Speicheranforderung aus [Mar19].

Neben CasCor existieren weitere Cascade-Lernalgorithmen – insbesondere solche, die dem Direct Cascade-Prinzip folgen –, die bislang noch nicht im Kontext von TF eingesetzt wurden und somit Potenzial für zukünftige Forschung bieten.

## 1.4 Ausblick in diese Arbeit

Kapitel 2 enthält eine Einführung in die grundlegenden Konzepte dieser Arbeit. Es werden die Kaskadierungsverfahren, das Prinzip des TFs, das verwendete Rechensystem, die eingesetzten Early-Stopping-Metriken sowie die für das Training verwendeten Datensätze vorgestellt. Darüber hinaus erfolgt eine Übersicht über alle durchgeführten Tests einschließlich ihrer jeweiligen Kennungen und Zielsetzungen.

Kapitel 3 widmet sich den systematisch auftretenden Phänomenen beim domänenübergreifenden TF. Im Fokus stehen hierbei der Leistungsabfall unmittelbar nach dem Transfer sowie das Overfitting auf den Source-Datensatz. Zur Analyse wird ein Deep-Cascade-Klassifikationsnetzwerk eingesetzt. Zudem werden in diesem Kapitel die Trainingszeiten aller betrachteten Netzarchitekturen verglichen und die Ergebnisse interpretiert.

Kapitel 4 behandelt ausschließlich beobachtete Effekte, die spezifisch im Kontext der Klassifikationsaufgabe auftreten. Dazu zählen die Auswirkungen unterschiedlicher Target-Datenmengen, die Erstellung sogenannter Augmented Vectors sowie der Vergleich unterschiedlicher Netzarchitekturen bei identischer Layer-Konfiguration. Ziel ist es, zu klären, warum TF im Rahmen von Kaskadierung bei Klassifikationsaufgaben nicht zuverlässig funktioniert.

Kapitel 5 fokussiert sich auf regressionsspezifische Beobachtungen. Hierfür werden sowohl Szenarien mit großen als auch mit kleinen Target-Datenmengen untersucht, wobei ein Vergleich zwischen Cascade TF, klassischer Kaskadierung und vollständig trainierten Netzwerken erfolgt. Dabei zeigt sich, dass TF in bestimmten Konstellationen zu besseren Ergebnissen führt. Da in diesem Kontext Cascade TF funktional einsetzbar ist, werden hier zusätzlich die Auswirkungen verschiedener Early-Stopping-Metriken betrachtet.

Kapitel 6 bietet eine zusammenfassende Darstellung der zentralen Erkenntnisse dieser Arbeit sowie einen Ausblick auf potenzielle weiterführende Forschungsfragen in diesem Themenfeld.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# Chapter 2

# Methodik

In diesem Kapitel werden das Konzept des Transferlernens (TF) sowie die Kaskadierungsansätze Deep Cascade und Direct Cascade erläutert. Darüber hinaus erfolgt eine detaillierte Beschreibung der für das Training verwendeten Datensätze. Es wird das Testsystem, auf dem die Experimente durchgeführt wurden, vorgestellt und die dabei eingesetzten Metriken zur Implementierung von Early Stopping eingeführt. Abschließend werden sämtliche durchgeführten Tests inklusive ihrer jeweiligen Bezeichnungen präsentiert.

## 2.1 Transferlernen

TF basiert auf dem Prinzip der Wissensübertragung zwischen unterschiedlichen, jedoch verwandten Lernsituationen – vergleichbar mit dem Konzept einer "Eselsbrücke". Es existieren verschiedene Formen des TFs, wobei insbesondere zwischen Task Transfer und Domain Transfer (Domänenwechsel) unterschieden wird. Wird keine dieser Varianten angewendet, handelt es sich nicht um TF im engeren Sinne.

In dieser Arbeit wird ausschließlich der Domain Transfer betrachtet, da ausschließlich dieser zum Einsatz kommt. Dabei wird ein Modell, das auf einem bestimmten Datensatz (Source-Domain) trainiert wurde, zur Verbesserung der Leistung auf einem anderen, inhaltlich verwandten Datensatz (Target-Domain) genutzt – unter Beibehaltung der gleichen Modellarchitektur. Dieser Ansatz wird als Transductive Transfer Learning bezeichnet [JY10]. Ziel ist es, das in der Source-Domain erworbene Wissen auf die Target-Domain zu übertragen.

Ein zentrales Problem des TFs besteht darin, geeignete Antworten auf die drei grundlegenden Fragen zu finden: What to transfer, How to transfer und When to transfer [JY10]. Diese Aspekte sind bislang nicht abschließend geklärt und stellen weiterhin offene Forschungsfragen dar.

Da sowohl Klassifikations- als auch Regressionsmodelle evaluiert werden, kommen jeweils zwei Datensätze als Source und Target zum Einsatz:

**Klassifikation:**

1. Source-Datensatz: Modified National Institute of Standards and Technology (MNIST) [LeC+]

2. Target-Datensatz: Street View House Numbers (SVHN) [Net+11]

Beide Datensätze müssen zur strukturellen Kompatibilität für das TF angepasst werden. MNIST-Bilder (ursprünglich 28x28 Pixel, einfarbig) werden auf 32x32 Pixel skaliert. Der SVHN-Datensatz, ursprünglich in Farbe (RGB), wird in Graustufen konvertiert. Dies gewährleistet, dass beide Eingabedatensätze die gleiche Dimensionalität aufweisen und somit als Input für dasselbe neuronale Netz genutzt werden können.

**Regression:**

1. Source-Datensatz: Boston Housing Prices (BOST) [Har+97]

2. Target-Datensatz: California Housing Prices (CALI) [Nug18]

Beide Regressionsdatensätze müssen inhaltlich reduziert werden, da nur solche Merkmale berücksichtigt werden sollen, die eine semantisch und strukturell sinnvolle Entsprechung in beiden Domänen aufweisen. Im BOST-Datensatz verbleiben nur die drei Merkmale:

1. RM: durchschnittliche Anzahl von Zimmern pro Wohnung

2. AGE: Anteil der vor 1940 erbauten Häuser

3. LSTAT: Anteil der Bevölkerung mit niedrigem sozioökonomischem Status

Hinweis: Die ursprünglich im BOST-Datensatz enthaltene Variable mit ethnisch diskriminierendem Inhalt wird vollständig entfernt.

Im CALI-Datensatz verbleiben:

1. MedInc: mittleres Einkommen pro Häuserblock

2. HouseAge: durchschnittliches Alter der Gebäude

3. RoomsPerHousehold: abgeleitet durch Division von AveRooms (durchschnittliche Zimmeranzahl im Block) durch Households (Anzahl der Haushalte)

Diese Ableitung stellt die strukturelle Entsprechung zur RM-Variable im BOST-Datensatz dar.

Eine inhaltliche Verbindung besteht voraussichtlich zwischen LSTAT (BOST) und MedInc (CALI), da sozioökonomischer Status und Einkommen erfahrungsgemäß korrelieren. Da LSTAT jedoch antiproportional zur Einkommenshöhe ist, wird diese Variable invertiert, um eine proportionale Beziehung zur MedInc-Variable herzustellen – eine Voraussetzung für effektives TF.

Die Harmonisierung der Altersvariablen gestaltet sich komplexer: AGE im BOST-Datensatz beschreibt den Anteil der vor 1940 erbauten Häuser, während HouseAge im CALI-Datensatz das durchschnittliche Alter der Gebäude angibt. Um AGE in eine vergleichbare Form zu bringen, wird auf Basis einer

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

angenommenen maximalen Anzahl von 100 Häusern pro Block und einem maximalen Alter von 85 Jahren (bezogen auf das Jahr 2025) eine Umrechnung vorgenommen. Die entsprechende Formel lautet:

$$\frac{AGE * 85}{100} \tag{2.1}$$

Durch die beschriebenen Anpassungen weisen alle verwendeten Source- und Target-Datensätze strukturelle Kompatibilität zueinander auf. Damit ist die Frage "Was wird transferiert?" im Sinne des TFs hinreichend beantwortet.

Die nächste zu klärende Dimension ist das "How to Transfer". Der Transfer erfolgt in allen Fällen ohne Modifikation der Gewichtungen zuvor trainierter Netzwerkschichten. Das neuronale Netzwerk wird zunächst vollständig auf dem Source-Datensatz trainiert. Anschließend erfolgt der Übergang zum Target-Datensatz, ohne dass das Modell selbst oder dessen Gewichtungen verändert werden.

Dabei unterscheiden sich die beiden eingesetzten Kaskadierungsansätze hinsichtlich der Art des Transfers:

1. Beim Deep Cascade-Verfahren bleibt die Netzinputgröße unverändert, und lediglich die Eingabedatenquelle (Source → Target) wird gewechselt. Der Transfer vollzieht sich somit ausschließlich über die Änderung des Input-Datensatzes bei gleichbleibender Architektur und Parameterbelegung. Dabei ist zu bedenken, dass dieses Netz immer größer wird.

2. Beim Direct Cascade-Verfahren erfolgt der Transfer implizit über die kontinuierliche Erweiterung des Eingaberaums. Der sogenannte Augmented Input wird bei jedem Schritt vergrößert, sodass die auf dem Source-Datensatz erlernten Repräsentationen in den nachfolgenden Stufen weiterhin einfließen.

Die Frage "Wann ist TF sinnvoll?" (When to Transfer) kann bislang nicht eindeutig beantwortet werden. Aus diesem Grund wird in den Experimenten mit unterschiedlichen Zeitpunkten für den Transfer gearbeitet – sowohl mit einem frühen als auch mit einem späteren Übergang zum Target-Datensatz.

## 2.2 Kaskadierung

In diesem Abschnitt wird das Konzept von Kaskadennetzwerken erläutert, einschließlich ihrer charakteristischen Merkmale. Ein Kaskadennetzwerk ist eine spezielle Architekturform neuronaler Netzwerke, bei der das Netzwerk schrittweise – in sogenannten Kaskaden – aufgebaut wird. Im Gegensatz zu konventionellen neuronalen Netzwerken, bei denen die gesamte Netzwerkstruktur (Anzahl und Art der Layer sowie der Subnetzwerke) im Vorfeld vollständig definiert ist, erfolgt die Konstruktion eines Kaskadennetzwerks iterativ während des Trainingsprozesses.

Dabei wird jeweils nur der neu hinzugefügte Netzwerkabschnitt trainiert, während die zuvor trainierten Komponenten eingefroren werden. Das bedeutet,

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

dass die Gewichtungen dieser eingefrorenen Layer und Subnetzwerke nach ihrer Trainingsphase nicht mehr angepasst werden. Diese Vorgehensweise führt dazu, dass Interaktionen zwischen den verschiedenen Netzwerkstufen – insbesondere zwischen bereits trainierten und neu hinzukommenden Komponenten – nicht erlernt werden können. Daher erzielen Kaskadennetzwerke bei gleicher Datenbasis in der Regel eine geringere Modellgüte als klassische Netzwerke mit vollständig trainierbarer Struktur.

Ein wesentlicher Vorteil der Kaskadenarchitektur liegt jedoch in der Trainingsgeschwindigkeit. Da jeweils nur ein Teil des Netzwerks aktiv trainiert wird und die eingefrorenen Gewichte unverändert bleiben, entfallen die Berechnungen zur Gradientenaktualisierung für den Großteil des Netzwerks. Dies reduziert den Rechenaufwand pro Trainingsschritt erheblich und führt zu einer insgesamt schnelleren Trainingsphase.

### 2.2.1 Deep Cascade

In diesem Abschnitt wird die Variante des Deep Cascade Networks vorgestellt. Bei dieser Architektur handelt es sich um ein einzelnes neuronales Netzwerk, das schrittweise – also iterativ – während des Trainingsprozesses aufgebaut wird. Trotz der sukzessiven Erweiterung bleibt das Modell stets eine zusammenhängende Netzwerkstruktur. Zu Beginn des Trainings werden grundlegende Trainingsparameter festgelegt, insbesondere der verwendete Optimierungsalgorithmus sowie die Verlustfunktion, die für alle nachfolgenden Kaskadenstufen konsistent angewendet werden.

Nachdem sowohl der Optimierer als auch die Verlustfunktion festgelegt wurden, beginnt der schrittweise Aufbau des Deep Cascade Netzwerks mit der Definition der ersten Netzwerkschicht. Diese wird zunächst durch eine temporäre Ausgabeschicht (Output-Layer) ergänzt und anschließend trainiert. Nach Abschluss des Trainings wird der Output-Layer entfernt und die trainierte Schicht eingefroren, sodass ihre Gewichtungen im weiteren Trainingsverlauf nicht mehr angepasst werden. Danach wird eine neue Schicht dem Netzwerk hinzugefügt, wie in Abbildung 2.1 dargestellt.

Dieser Zyklus aus Training, Entfernen des Output-Layers, Einfrieren der trainierten Schicht und Hinzufügen einer weiteren Schicht wiederholt sich iterativ. Optional kann zu einem beliebigen Zeitpunkt TF initiiert werden, indem anstelle des ursprünglichen Source-Datensatzes ein Target-Datensatz für das Training verwendet wird.

### 2.2.2 Direct Cascade

In diesem Abschnitt wird die Kaskadierungsvariante des Direct Cascade Netzwerks vorgestellt. Das Netzwerk ist hierbei vollständig vorab definiert und besteht aus einem einzelnen Hidden Layer sowie einem Output Layer. Die Gesamtstruktur setzt sich aus mehreren identischen Subnetzwerken zusammen, zwischen denen während des Trainings ein Wissenstransfer stattfindet.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

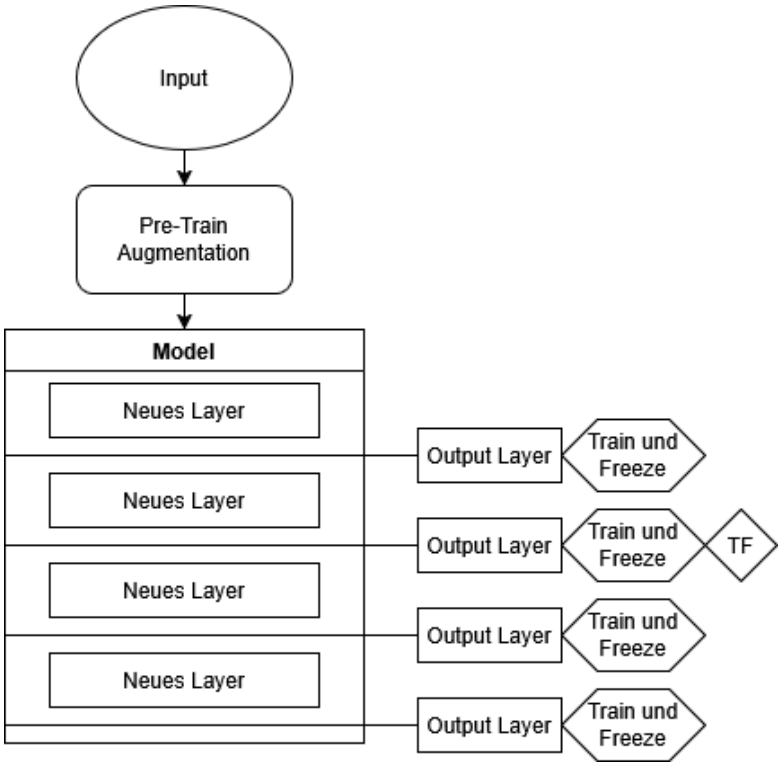


Figure 2.1: In der Darstellung wird der Aufbau und Trainingsprozess von Deep Cascade Netzwerken veranschaulicht. Das zugrunde liegende Modell besteht aus mehreren Schichten (Layern), die sukzessive – also nacheinander – trainiert werden. Zur Unterstützung des schrittweisen Trainingsprozesses werden temporäre Output-Layer extern am Netzwerk angebracht. Diese dienen ausschließlich dem Training des jeweils aktuellen Abschnitts und werden nach dessen Abschluss entfernt, da ihre Integration in die Hauptstruktur des Netzwerks den internen Informationsfluss beeinträchtigen würde. Nach dem Training werden alle bisherigen Layer eingefroren. TF kann zwischen jedem Layer vollzogen werden.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

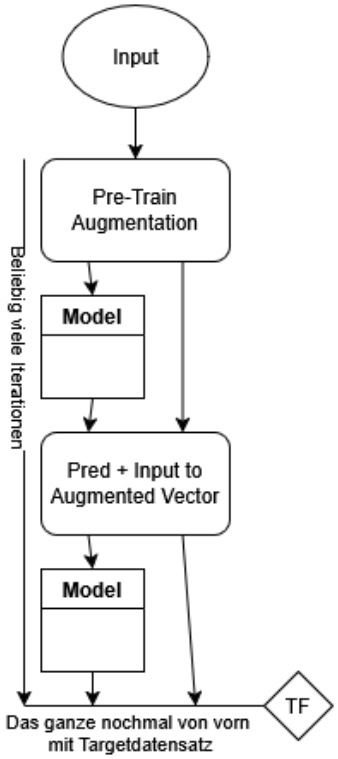


Figure 2.2: Hier wird das Direct Cascade Verfahren dargestellt. Dieses Verfahren verwendet mehrere einzelne Netzwerke (hier als Modelle bezeichnet), die jeweils nur wenige Hidden Layer aufweisen, in der Regel lediglich einen. Nach der Initialisierung und dem Training wird jedes Modell einmal ohne weiteres Training angewendet, dessen Ausgangssignal mit dem ursprünglichen Eingabesignal kombiniert wird. Diese Kombination bildet den neuen Eingabedatensatz für das nachfolgende Modell. Durch diese sukzessive Verknüpfung der Ausgaben mit den Eingaben kann das Verfahren eine Wissensweitergabe und -integration zwischen den einzelnen Modellen realisieren. Darüber kann an beliebiger Stelle TF verwendet werden.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Der Ablauf beginnt, wie in Abbildung 2.2 dargestellt, mit dem vorbereiteten Source-Datensatz, der als Eingabe in die erste Instanz des Netzwerks gegeben wird. Diese Netzwerkinstanz wird anschließend trainiert. Nach Abschluss des Trainings erfolgt eine einmalige Anwendung des fixierten Netzwerks, deren Ergebnis die Vorhersage (Prediction) darstellt. Diese Prediction wird mit dem ursprünglichen Eingabesignal desselben Netzwerks kombiniert, wodurch ein sogenannter Augmented Vector entsteht. Die genaue Bildung dieses Augmented Vectors variiert dabei leicht je nach spezifischer Implementierung des jeweiligen Direct Cascade Netzwerks und wird an späterer Stelle detaillierter erläutert.

Der Augmented Vector dient als Input für die nächste Instanz des Netzwerks. Dieser Zyklus aus Netzwerkinstanz, Training, Prediction und Augmented Vector Berechnung wird beliebig oft wiederholt. Durch die Einbindung der Prediction in den Augmented Vector kann das Netzwerk Wissen aus den zuvor trainierten Instanzen übernehmen und integrieren.

TF kann jederzeit innerhalb eines Trainingsschritts durchgeführt werden, indem anstelle des Source-Datensatzes ein Target-Datensatz als Input verwendet wird. Dabei können beliebig viele Netzwerkinstanzen vor und nach TF genutzt werden. Der einzige Unterschied besteht darin, dass der Augmented Vector mit jedem weiteren Netzwerk etwas größer wird, da er sowohl das Wissen aller bisher trainierten Netzwerke als auch die ursprünglichen Eingabedaten enthält.

Für die Implementierung bedeutet dies, dass von Beginn an sowohl der Source- als auch der Target-Datensatz in das feste Netzwerk eingespeist werden müssen. Dies ist notwendig, um die Prediction auf dem Target-Datensatz – die während der Trainingsphase mit dem Source-Datensatz generiert wurde – im Augmented Vector zu integrieren. Somit wird sichergestellt, dass das während des Trainings auf dem Source-Datensatz gelernten Wissens auch bei der Anpassung an den Target-Datensatz berücksichtigt werden.

## 2.3 Setup

Alle Experimente wurden auf einem Erazer Gaming Notebook P15601 mit Windows 10 durchgeführt. Die neuronalen Netzwerke wurden ausschließlich auf der CPU ausgeführt und im Netzbetrieb trainiert. Das System ist mit einem Intel Core i5 der 9. Generation ausgestattet, welcher über 4 physische Kerne und 8 logische Prozessoren verfügt. Die Taktrate liegt im Bereich von 2,4 bis 5,1 GHz. Der Arbeitsspeicher umfasst 15,8 GB mit einer Geschwindigkeit von 2667 MHz.

Die Implementierung erfolgte unter Verwendung von PyCharm als Entwicklungsumgebung sowie der Keras-Bibliothek für das neuronale Netzwerk-Framework. Die Dokumentation wurde mithilfe von BibTeX erstellt, und die Visualisierungen basieren auf der Matplotlib-Bibliothek.

Als Source-Datensätze wurden MNIST und Boston Housing verwendet, während SVHN und California Housing als Target-Datensätze dienen. Die Target-Datensätze wurden manuell reduziert, um bewusst eine unzureichende Datenmenge zu simulieren, sodass alternative Lernmethoden notwendig sind.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Es ist zu beachten, dass keiner der verwendeten Datensätze in normalisierter Form als Eingabe in die Modelle eingespeist wurde.

## 2.4 Metrik

Es wurden drei Evaluationsmetriken definiert: die Accuracy-Metrik (ACCM), die Loss-Metrik (LM) sowie die Mean Absolute Error-Metrik (MAEM). Diese Metriken dienen als Kriterien für das Early Stopping und bestimmen die Anzahl der Trainings-Epochen.

Das Early Stopping anhand der ACCM erfolgt, sobald die Validierungsgenauigkeit um mindestens 10% unter der Trainingsgenauigkeit liegt, was auf Overfitting im Netzwerk hinweist.

Bei der LM und MAEM wird das Training beendet, wenn der Validierungswert in der aktuellen Epoche schlechter ausfällt als in der vorherigen. Dieses Verhalten verursacht, dass das Netzwerk in einem lokalen Minimum konvergiert, aus dem es nicht mehr herausfindet.

Für die Anzahl der Netzwerke im Direct Cascade Verfahren wird keine dieser Metriken zur Steuerung des Trainings eingesetzt. Ihre Anwendung besteht darin, den Trainingsprozess vorzeitig zu beenden, noch bevor die vorgesehene maximale Anzahl an Epochen erreicht wird.

## 2.5 Liste der Tests

Von den betrachteten Netzwerken sind ConvMaxPool und RegressionTwo Deep Cascade Netzwerke, während alle weiteren Modelle als Direct Cascade Netzwerke klassifiziert werden. Zudem handelt es sich bei RegressionTwo und OneLayer um Regressionsnetzwerke, während die übrigen Modelle Klassifikationsnetzwerke darstellen.

Alle Netzwerke wurden mit dem Adam-Optimierer bei einer Lernrate von $1 * 10^{-3}$ trainiert. Für die Klassifikationsnetzwerke wurde die Verlustfunktion Categorical Cross-Entropy verwendet, kombiniert mit einer Softmax-Aktivierungsfunktion im Ausgangs-Layer. Die Regressionsnetzwerke hingegen verwenden Mean Squared Error (MSE) als Verlustfunktion und eine lineare Aktivierungsfunktion.

Zusätzlich wurden bei allen Direct Cascade Netzwerken Early-Stopping-Kriterien basierend auf den Metriken MAEM, LM und ACCM angewandt.

Für die Klassifikationsnetzwerke erfolgte das Training mit fünf unterschiedlichen Größen des Target-Datensatzes. Eine Ausnahme bildet das 2DC-Netzwerk, das aufgrund technischer Einschränkungen der verwendeten Hardware nur mit sehr kleinen Mengen von Source- und Target-Daten trainiert werden kann.

Bei den Regressionsnetzwerken wurde jeweils ein Training mit großer sowie mit kleiner Menge an Target-Daten durchgeführt.

Darüber hinaus wurden mit allen Netzwerken Vergleiche zwischen Trainingsläufen mit TF und ohne sowie zwischen Trainings ohne TF und sogenannten

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

vollständigen Netzwerken durchgeführt. Ein vollständiges Netzwerk bezeichnet hierbei ein Modell, das ohne TF und ohne Kaskadierung in einem einzigen vollständigen Training trainiert wird. Dies ist das klassische Training neuronaler Netzwerke.

Der Zeitpunkt des Einsatzes von TF wurde für alle Netzwerke experimentell frei gewählt.

Alle Direct Cascade Netzwerke verfügen jeweils über genau einen Hidden Layer. In einigen Fällen wird zusätzlich ein Hilfslayer verwendet, um den Übergang zwischen Filter-Layern (Convolutional Layer) und linearen Layern zu ermöglichen.

Für alle Experimente wurde derselbe Initialisierungs-Seed für die Gewichtungen verwendet, um Reproduzierbarkeit sicherzustellen.

Die Ergebnisse der Klassifikationstests sind in Tabelle 2.1 zusammengefasst, die Regressionstests in Tabelle 2.2. In beiden Tabellen sind Tests, die sich auf die Trainingsdauer beziehen, mit dem Suffix "Time" gekennzeichnet. Dabei steht CasTF für Kaskadierung mit TF, Cas für Kaskadierung ohne TF und Comp für Trainings ohne Kaskadierung und ohne TF - also den sogenannten vollständigen Netzen.

Die Bezeichnungen vor dem ersten Schrägstrich geben den Zeitpunkt des TF an, wobei dies mit "TF" explizit markiert ist. Fehlt dieser Eintrag, wurde kein TF durchgeführt. Anschließend folgt die Datenmenge des Target-Datensatzes sowie die Anzahl der Epochen pro Trainingsiteration. Optional enthält die Bezeichnung eine weitere Angabe, die die Gesamtanzahl der Epochen in Zehner-Schritten angibt.

| **CMP** | **COD** | **1DC** | **2DC** |
|---|---|---|---|
| TF0/732/10 | CasTFTime | CasTFTime | CasTFTime |
| TF1/732/10 | CasTime | CasTime | CasTime |
| TF2/732/10 | CompTime | CompTime | CompTime |
| TF3/732/10 | TF2/732/10 | TF2/732/10 | TF2/732/10 |
| TF4/732/10 | TF2/7k/10 | TF2/7k/10 | |
| TF5/732/10 | TF2/21k/10 | TF2/21k/10 | |
| 732/10 | TF2/36k/10 | TF2/36k/10 | |
| CasTFTime | TF2/51k/10 | TF2/51k/10 | |
| CasTime | TF10/732/10/30 | TF10/732/10/30 | |
| CompTime | 732/10/30 | 732/10/30 | |
| TF2/7k/10 | Comp/732//30 | Comp/732//30 | |
| TF2/21k/10 | ACCM/732/10 | ACCM/732/10 | |
| TF2/36k/10 | LM/732/10 | LM/732/10 | |
| TF2/51k/10 | | | |

Table 2.1: Liste aller Klassifikationstests

Ein Beispiel für eine Referenz auf einen spezifischen Testeintrag ist CMP:TF0/732/10. Diese Kennzeichnung verweist auf den Test des CMP-Netzwerks mit TF, das

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

| **Regr2** | **1Lay** |
|---|---|
| TF0/240/25 | CasTFTime |
| TF1/240/25 | CasTime |
| TF4/240/25 | CompTime |
| CasTFTime | TF11/8k/10/21 |
| CasTime | 8k/10/11 |
| CompTime | Comp/8k//8 |
| TF4/8k/10/8 | TF11/240/10/21 |
| 8k/10/8 | 240/10/11 |
| Comp/8k//8 | Comp/240//8 |
| TF4/240/10/8 | MAEM/240/10 |
| 240/10/8 | LM/240/10 |
| Comp/240//8 | TF4/206/10/8/ts |
| TF4/206/10/8/ts | 206/10/8/ts |
| 206/10/8/ts | Comp/206//8/ts |
| Comp/206//8/ts | |

Table 2.2: Liste aller Regressionstests

unmittelbar nach dem ersten Layer erfolgt. In diesem speziellen Fall wird das erste Layer lediglich über eine einzige Epoche trainiert, während alle weiteren Layer nach dem üblichen Trainingsschema optimiert werden. Das gleiche Vorgehen gilt für den Test Regr2:TF0/240/25, bei dem analog das erste Layer nur eine Epoche lang trainiert wird.

Testeinträge mit dem Suffix "ts" kennzeichnen solche Experimente, bei denen ein gezielt großer Testdatensatz verwendet wurde.

Die Tests mit unterschiedlichen TF-Layern dienen dazu, den sinnvollen Einsatz von TF zu evaluieren. Alle Tests mit dem Zusatz "Time" verfolgen das Ziel, die Trainingsdauer zu vergleichen. Die Untersuchungen mit variierenden Target-Datenmengen analysieren das Verhalten von TF bei unterschiedlichen Datenvolumina sowie die generelle Leistungsfähigkeit der Netzwerke. Tests mit einer erhöhten Anzahl an Gesamtepochen und tieferer Netzstruktur ermöglichen eine detaillierte Betrachtung des Einflusses von TF auf tiefe Netze. Die Versuche, bei denen verschiedene Metriken zum Einsatz kommen, prüfen, inwiefern diese Early-Stopping-Metriken zu einer Leistungsverbesserung führen. Schließlich dienen die Tests mit umfangreichen Testdatensätzen der Überprüfung der Generalisierungsfähigkeit der Netzwerke und der Validierung bisheriger Erkenntnisse.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# Chapter 3

# Konsistente Auffälligkeiten

In diesem Kapitel werden zentrale Eigenschaften analysiert, die sowohl bei Klassifikations- als auch bei Regressionsaufgaben auftreten und unabhängig vom zugrunde liegenden Cascade-Learning-Ansatz – sei es Deep Cascade oder Direct Cascade – beobachtet werden konnten.

Zunächst wird die Interpretation der dargestellten Plots erläutert, um ein einheitliches Verständnis der Visualisierungen zu gewährleisten. Anschließend werden zwei besonders auffällige, konsistent auftretende Phänomene diskutiert: der signifikante Leistungseinbruch unmittelbar nach Anwendung von TF sowie das ausgeprägte Overfitting auf dem Source-Datensatz.

Abschließend wird die Trainingsdauer der eingesetzten Netzwerkarchitekturen analysiert. Dabei zeigt sich, dass der Einfluss von TF auf die Trainingszeit bei kleinen Netzwerken von den in der Literatur beschriebenen Ergebnissen abweicht und und sich die zeitlichen Relationen mit zunehmender Netzwerktiefe zur Literatur passend verändern.

## 3.1 Ploterklärung

In diesem Unterkapitel werden die verschiedenen Arten der verwendeten Plots systematisch vorgestellt und deren spezifische Merkmale erläutert, um eine korrekte Interpretation zu ermöglichen.

Dabei wird sowohl auf die Achsenbeschriftungen als auch auf die innerhalb der Visualisierungen enthaltenen Texte und Annotationen eingegangen.

Zur Erläuterung wird Abbildung 3.1 herangezogen. Die Überschriften der Plots sind die vollständigen Namen der dazugehörigen Netzwerke. In beiden Diagrammen sind jeweils drei Textzeilen angegeben, auf die im Folgenden näher eingegangen wird. Die erste Zeile gibt an, wie viele Trainings-Epochen pro Layer beziehungsweise Netzwerk durchgeführt wurden. Die zweite Zeile beschreibt, wie viele Trainingsbeispiele aus dem Trainingsdatensatz des Target-Datensatzes während des Trainings unter Verwendung von TF eingesetzt wurden. Die dritte Zeile gibt die gesamte Trainingsdauer in Sekunden an.

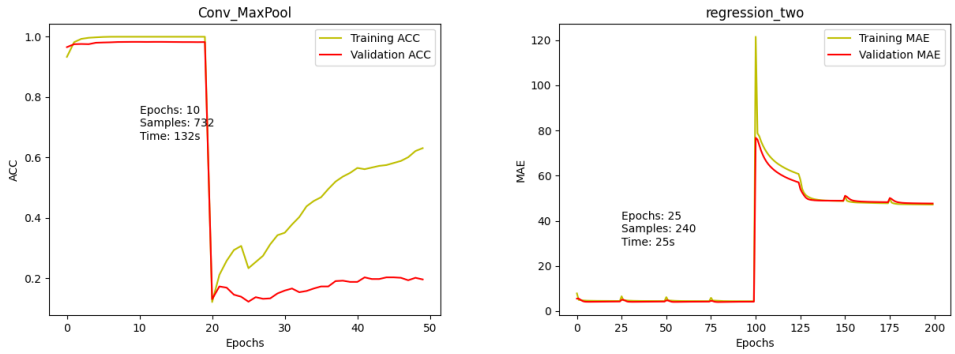


Figure 3.1: Abgebildet sind exemplarische Plots zweier Deep-Cascade-Netzwerke – links für eine Klassifikationsaufgabe, rechts für eine Regressionsaufgabe. Dargestellt werden die Tests CMP:TF2/732/10 und Regr2:TF4/240/25, die jeweils als typische Repräsentationen für den Einsatz von TF in den entsprechenden Aufgabenbereichen dienen.

Im Fall der Klassifikation, bei der die Genauigkeit (Accuracy) von Interesse ist, wird die y-Achse mit "ACC" beschriftet, was auch im Funktionsnamen reflektiert wird. Ein Wert von 1 auf der y-Achse entspricht dabei einer Accuracy von 100%. Abbildung 3.1 zeigt links ein Beispiel für diese Darstellungsweise.

Bei der Regression hingegen wird der mittlere absolute Fehler (MAE) betrachtet, welcher ebenfalls in der Bezeichnung der Funktionen sowie auf der y-Achse angegeben ist. Die y-Achse ist in Einheiten von 1000$ pro Einheit skaliert. Hier gilt: Je niedriger der MAE-Wert, desto besser das Ergebnis.

## 3.2 ConvMaxPool

Anhand des Conv_MaxPool-Netzwerks (CMP) werden sämtliche allgemeinen Ergebnisse und Besonderheiten erläutert. Dabei handelt es sich um ein Deep Cascade Klassifikationsnetzwerk, welches in iterativer Weise aufgebaut ist. Das Netzwerk ist ein Convolutional Neural Network mit Padding, sodass die räumlichen Dimensionen innerhalb der Convolutional Layer erhalten bleiben. Als Aktivierungsfunktion wird die ReLU-Funktion eingesetzt.

Alle Layer des ConvMaxPool-Netzwerks sind in Abbildung 3.2 in der korrekten Reihenfolge dargestellt. Dabei gibt die erste Zahl eines Convolutional Layers die Anzahl der verwendeten Filter an, während das anschließende Tupel die Größe der Filterkerne spezifiziert. Analog wird die Kerngröße auch bei den MaxPooling-Layern angegeben. Das Flatten- sowie das Linear-Layer bilden den Output-Block. Das Linear-Layer verfügt über zehn Neuronen, entsprechend der Anzahl der Klassen im Klassifikationsproblem. Jedes Hidden Layer wird über zehn Trainings-Epochen optimiert. Ein Early-Stopping-Verfahren kommt nicht zum Einsatz, und für alle Versuchsreihen wird derselbe Zufallsseed verwendet.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

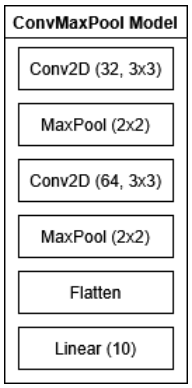


Figure 3.2: Dem CMP-Netzwerk liegen die Layer in exakt der hier von oben nach unten dargestellten Reihenfolge zugrunde.

### 3.2.1 Veränderungen bei TF

Im Folgenden wird ein sehr typisches Phänomen untersucht, das bei jeder Anwendung von TF auftritt. Der Verlauf der Performanz-Kurven für Trainings- und Validierungsdaten zeigt stets einen deutlichen Einbruch an dem Punkt, an dem das TF implementiert wird. Dieses Verhalten ist in Abbildung 3.3 insbesondere bei Epoche 20 deutlich erkennbar.

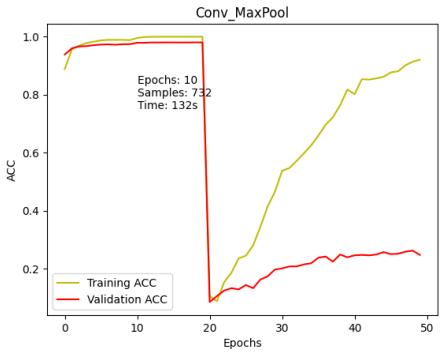


Figure 3.3: Abgebildet ist der Test CMP:TF2/732/10. Der Fokus liegt hierbei auf Epoche 20, da zu diesem Zeitpunkt das TF angewendet wurde. Infolge dessen ist ein deutlicher Einbruch der Performanz zu beobachten.

Dieser Leistungseinbruch tritt konsistent nach der Anwendung von TF auf. Dies ist darauf zurückzuführen, dass das Netzwerk bislang ausschließlich auf dem Source-Datensatz trainiert wurde und somit keine Kenntnis der Target-Daten besitzt. Das Modell verfügt demnach nur über das Wissen aus der Source-Domäne und kann dieses initial nur auf die neue Domäne übertragen. Betrachtet man jedoch das Testergebnis, das ausschließlich auf dem Target-Datensatz

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

basiert, so ergibt sich das in Abbildung 3.4 dargestellte Bild.

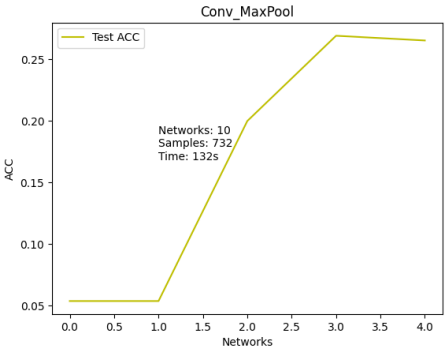


Figure 3.4: Diese Abbildung zeigt den Test CMP:TF2/732/10. Dieser basiert auf Target-Daten, die dem Netzwerk während des Trainings nicht zugänglich waren. Nach Abschluss des Trainings jedes Layers erfolgt eine Evaluation anhand der Testdaten. Dementsprechend ist der Punkt, an dem TF durchgeführt wird, bei zwei Netzwerken erkennbar. Zu diesem Zeitpunkt verbessert sich die Performanz deutlich.

Der Wechsel erfolgt hierbei beim zweiten Netzwerk. Es ist deutlich erkennbar, dass sich die Performanz nach Anwendung des TFs verbessert. Dies liegt darin begründet, dass das Netzwerk ab diesem Zeitpunkt auf Trainingsdaten trainiert wird, die mit dem Testdatensatz übereinstimmen, da dieser ausschließlich Daten aus dem Target-Datensatz enthält.

### 3.2.2 Overfitting auf Source-Datensatz

Bei variierender Trainingsdauer auf dem Source-Datensatz zeigt sich, dass die Leistungsfähigkeit des Netzwerks auf dem Target-Datensatz unterschiedlich ausfällt. Da der optimale Zeitpunkt für den Einsatz von TF ohnehin ermittelt werden muss, wird im Folgenden das CMP-Netzwerk verwendet, wobei nach jedem Layer TF angewandt wird. Die daraus resultierenden Ergebnisse sind in Abbildung 3.5 dargestellt.

Es fällt auf, dass die beste Performanz ohne den Einsatz von TF erzielt wird. Bereits nach nur einer Trainings-Epoche im ersten Layer, das auf dem Source-Datensatz trainiert wird, ist ein deutlicher Einbruch der Accuracy zu beobachten. Dies legt nahe, dass TF im Kontext von Klassifikationsaufgaben mit Deep Cascade Netzwerken wenig sinnvoll ist. Das Trainingsset, das bei TF verwendet wird, erreicht nie annähernd die Leistungsbereiche, die ohne TF erzielt werden. Dabei nimmt die Accuracy auf den Trainingsdaten mit zunehmender Verzögerung des Einsatzes von TF kontinuierlich ab. Daraus lässt sich ableiten, dass bereits ein Overfitting auf den Source-Datensatz stattgefunden hat, welches das Lernen auf dem Target-Datensatz erheblich erschwert. Dieses Overfitting tritt dabei bereits nach einem einzigen Layer auf dem Source-Datensatz auf, wie die Grafik unten links in Abbildung 3.5 verdeutlicht.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

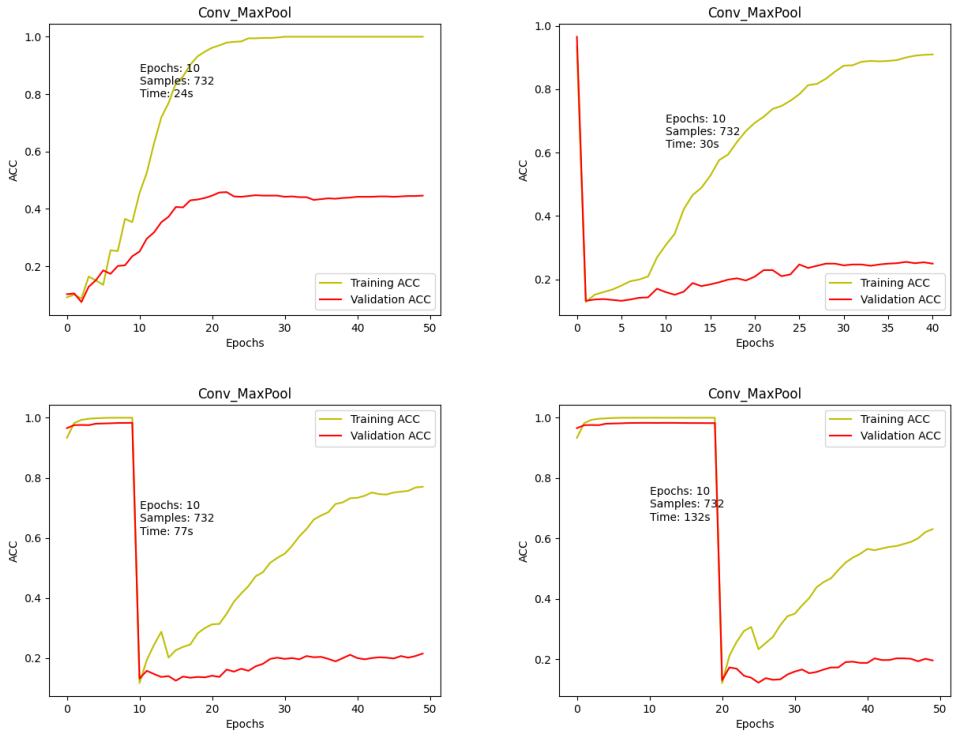


Figure 3.5: In der Abbildung sind von links oben nach rechts unten die Plots folgender Tests dargestellt: CMP:732/10, CMP:TF0/732/10, CMP:TF1/732/10 sowie CMP:TF2/732/10. Bei allen dargestellten Plots zeigt sich ein ähnliches Problem, nämlich eine zunehmende Diskrepanz zwischen Trainings-Accuracy und Validierungs-Accuracy über den Trainingsverlauf hinweg.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Darüber hinaus ist in allen dargestellten Plots erkennbar, dass das Overfitting auf das Trainingsset des Target-Datensatzes vorhanden ist, da die Trainings-Accuracy dort um etwa 60% höher liegt als bei der Validierungs- und bei der Test-Accuracy.

Im Gegensatz dazu zeigt ein Regressionsnetzwerk, wie das Deep Cascade Netzwerk RegressionTwo, wie in Abbildung 3.6 dargestellt, kein schnelles Auftreten von Overfitting. Weder auf dem Source-Datensatz noch auf dem Target-Datensatz lässt sich ein entsprechendes Verhalten beobachten.

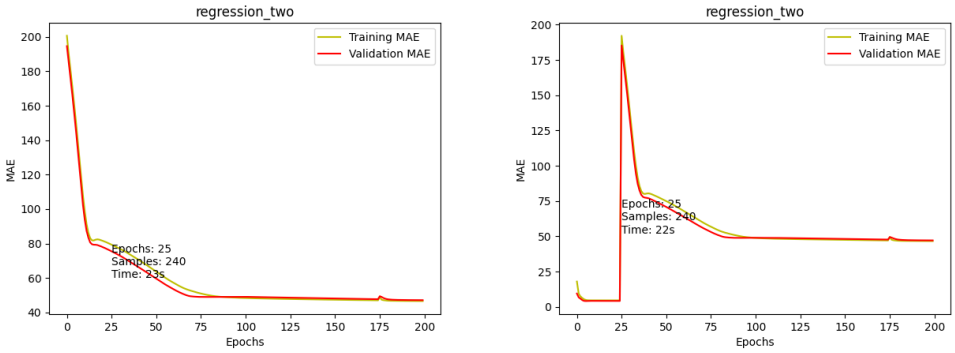


Figure 3.6: In der Abbildung sind links die Ergebnisse des Tests Regr2:240/25 und rechts die des Tests Regr2:TF1/240/25 dargestellt. Der MAE zwischen Trainings- und Validationsset weist dabei kaum Unterschiede auf, was auf ein fehlendes Overfitting hinweist.

Das Overfitting-Problem tritt ausschließlich bei der Klassifikation auf. Dies lässt darauf schließen, dass die Ursache in der verwendeten Loss-Funktion oder Aktivierungsfunktion für die Klassifikationsaufgaben liegt.

## 3.3 Zeitnahme

In diesem Abschnitt werden alle Netzwerke hinsichtlich ihrer Trainingsdauer untersucht. Die Trainingszeit wird in jedem Plot angegeben und ist im Wesentlichen abhängig von der Anzahl der verwendeten Trainingsdaten sowie der maximal erlaubten Epochenanzahl. Daher werden für die Klassifikations- und Regressionsnetzwerke jeweils gleiche Mengen an Trainingsdaten und identische Gesamtepochenzahlen verwendet.

Beim Einsatz von TF kommen stets der kleinste Target-Datensatz sowie der größte Source-Datensatz zum Einsatz. Ohne TF wird ausschließlich der kleinste Target-Datensatz verwendet. Jedes Klassifikationsnetzwerk wird über insgesamt 40 Epochen trainiert, wobei bei Anwendung von TF nach 20 Epochen gewechselt wird. Regressionsnetzwerke werden über 80 Epochen trainiert, mit einem TF-Wechsel nach 30 Epochen.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Die entsprechenden Graphen werden an dieser Stelle nicht dargestellt, da deren Anzahl zu groß ist und vergleichbare Darstellungen bereits an anderen Stellen verfügbar sind. Für die Einsichtnahme und Überprüfung dieser Ergebnisse verweisen wir auf das zugehörige GitHub-Repository unter `https://github.com/Lirras/BA_EvalTF_DDCN/tree/main/Plots/ba_plots/timing`.

Es wird keine Early-Stopping-Metrik verwendet. Alle Klassifikationsnetzwerke erhalten identische Eingabedaten, ebenso wie alle Regressionsnetzwerke, um eine bessere Vergleichbarkeit innerhalb der jeweiligen Gruppe zu gewährleisten. Die verschiedenen Netzversionen – Cascade TF, Cascade und Complete – besitzen dabei dieselben Layer in gleicher Anzahl.

Im Folgenden wird eine Tabelle präsentiert, die die Trainingszeiten aller Netzwerke in einer vergleichbaren Form zusammenfasst:

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

Table 3.1: Dies stellt den Vergleich der Trainingsdauer zwischen den jeweiligen Netzwerken und deren Varianten dar. Die Zeitangaben erfolgen in Sekunden.

In Tabelle 3.1 sind die Trainingszeiten der unterschiedlichen Netzwerkvarianten zusammengefasst. Die Spalte "Cascade TF" umfasst Kaskadennetzwerke mit TF. Die Spalte "Cascade" enthält Kaskadennetzwerke, die ausschließlich auf dem Target-Datensatz trainiert wurden. Die Spalte "Complete" schließlich zeigt die Trainingszeiten von Netzwerken, die weder TF verwenden noch kaskadiert sind, deren Layer vor dem Training festgelegt wurden und die vollständig auf dem Target-Datensatz trainiert wurden.

Auffällig ist, dass die Regressionsnetzwerke keine signifikanten Unterschiede in der Trainingsdauer aufweisen. Teilweise benötigt die Variante mit TF sogar weniger Zeit als ohne, was darauf zurückzuführen ist, dass der Target-Datensatz bei der Regression etwas größer als der Source-Datensatz ist. Beide Datensätze umfassen jedoch mit etwas über 200 Trainingsbeispielen nur eine geringe Stichprobengröße.

Im Gegensatz dazu benötigen alle Klassifikationsnetzwerke mit TF eine längere Trainingszeit als ohne TF. Dies liegt daran, dass sie zunächst auf dem Source-Datensatz trainieren, welcher mit etwa 48.000 Trainingsbeispielen umfangreich ist, während die anderen Netzwerke direkt auf dem kleinen Target-Datensatz mit 732 Samples trainieren.

Da die Cascade-TF-Netzwerke auf dem Source-Datensatz trainieren und die anderen Netzwerke ausschließlich auf dem Target-Datensatz, sind letztere in der Regel schneller. Innerhalb dieser Gruppe weisen die Complete-Netzwerke, die nicht kaskadiert sind, die kürzesten Trainingszeiten auf. Dies ist darauf

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

zurückzuführen, dass bei jedem Kaskadennetzwerk das Output-Layer in jeder Stufe berechnet wird, während die Complete-Netzwerke nur ein einziges Output-Layer besitzen. Zudem entfallen hier zusätzliche Predictions sowie die Berechnung von Augmented Vectors.

Nicht zuletzt sind die verwendeten Netzwerke nur moderat komplex und der betrachtete Target-Datensatz ist relativ klein, sodass die Trainingszeit weniger vom eigentlichen Training, sondern vielmehr von den damit verbundenen Berechnungen dominiert wird. Gleichzeitig erfolgt das Training in diesem Fall mit lediglich 40 Epochen, was im Vergleich als relativ geringe Anzahl einzustufen ist.

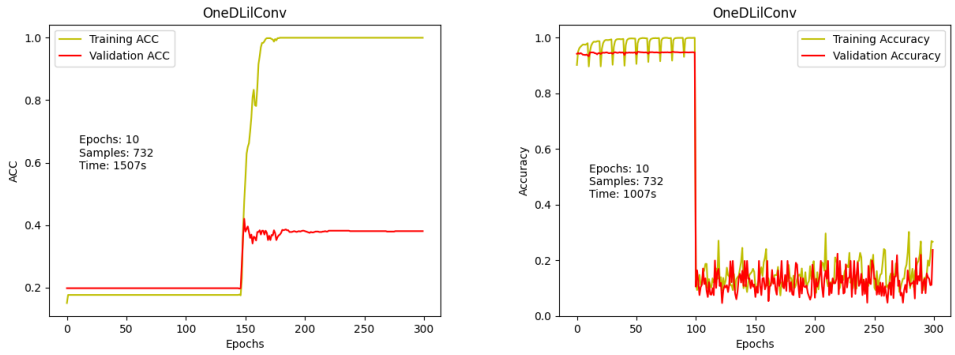


Figure 3.7: In diesem Fall erfolgt das Training über eine hohe Anzahl an Epochen unter Einsatz tieferer und komplexerer Netzarchitekturen, wodurch sich das Verhältnis der Trainingsdauer umkehrt. Veranschaulicht wird dies anhand der Tests 1DC:Comp/732//30 sowie 1DC:TF10/732/10/30.

Wird mit einer höheren Anzahl an Epochen – beispielsweise 300 – und tieferen Netzarchitekturen trainiert, verschieben sich die zeitlichen Verhältnisse dahingehend, dass ein vollständig trainiertes Netzwerk mehr Zeit benötigt als ein kaskadiertes Transfer-Learning-Netzwerk. Diese Entwicklung ist in Abbildung 3.7 dargestellt.

Die Ursache hierfür liegt darin, dass ab einer bestimmten Netzwerktiefe und Epochenanzahl die Trainingsdauer nicht mehr primär durch strukturelle Aspekte – wie das Hinzufügen oder Entfernen von Layern oder die Berechnungszeit für den Augmented Vector – bestimmt wird, sondern maßgeblich vom eigentlichen Trainingsprozess.

Kaskadierte Netzwerke weisen in diesem Zusammenhang einen zeitlichen Vorteil auf, da sie jeweils nur die neu hinzugefügten Netzwerkbereiche trainieren müssen. Im Gegensatz dazu muss ein vollständiges Netzwerk sämtliche enthaltenen Layer kontinuierlich berücksichtigen, wobei diese während des Trainings einem fortlaufenden Änderungsprozess unterliegen. Dies führt zu häufigeren Gewichtsanpassungen und somit zu einem insgesamt höheren Rechenaufwand.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# Chapter 4

# Klassifikation

In diesem Kapitel werden die Eigenschaften und Erkenntnisse im Bereich der Klassifikation detailliert untersucht. Zunächst werden die Direct Cascade Netzwerke vorgestellt und gemeinsam mit dem Deep Cascade Netzwerk analysiert. Dabei wird untersucht, wie sich die Menge der verfügbaren Target-Daten auf das Trainingsergebnis auswirkt. Zudem wird erläutert, weshalb das 2DC-Netzwerk trotz potenziell besserer Ergebnisse nicht weiter betrachtet wird. Ein weiterer Schwerpunkt liegt auf der detaillierten Analyse der Augmented Vectors bei den Direct Cascade Netzwerken. Abschließend werden die Unterschiede und Auswirkungen beim Wechsel zwischen Cascade TF, Cascade und Complete Netzwerken diskutiert.

Die Direct Cascade Netzwerke für die Klassifikation zeichnen sich dadurch aus, dass jeweils nur ein einzelnes Hidden Layer verwendet wird und die Netzwerke iterativ aufgebaut sind, wobei das Wissen der vorherigen Iterationen übernommen wird.

| **Name** | **Hiddenlayer** | **N/F** | **Aktivierung** | **Inputdim** |
|---|---|---|---|---|
| COD | Linear | 512 | Relu | 1 |
| 1DC | 1DConv | 32 | Relu | 1 |
| 2DC | 2DConv | 32 | Relu | 2 |

Table 4.1: In dieser Übersicht sind alle Direct Cascade Netzwerke für die Klassifikation mit den wesentlichen internen Parametern dargestellt. "Hidden Layer" bezeichnet die Art des verwendeten versteckten Layers. "N/F" gibt die Anzahl der Neuronen (Nodes) beziehungsweise Filter im jeweiligen Layer an. Unter "Aktivierung" ist die im Hidden Layer eingesetzte Aktivierungsfunktion zu verstehen, und "Inputdim" beschreibt die Dimensionalität der Eingabedaten.

In Tabelle 4.1 sind sämtliche Direct Cascade Klassifikationsnetzwerke aufgeführt. Diese umfassen die Modelle Classification_one_Dense (COD), OneDLilConv (1DC) sowie little_conv (2DC).

Der Input entspricht der Dimensionalität, in der die Bilddaten vorliegen. Lediglich in der ersten Zeile wird die Anzahl der Neuronen (Nodes) angegeben,

während in den übrigen Zeilen die Anzahl der Filter angegeben ist. Bei beiden Convolutional-Netzwerken wird ein Kernel der Größe 3 bzw. 3x3 verwendet, wobei durch entsprechendes Padding die räumlichen Dimensionen der Daten erhalten bleiben. Für das Training wird in diesen Netzwerken eine Batch-Größe von jeweils 128 Datensamples verwendet.

## 4.1 Größe des Target-Datensatzes

In diesem Abschnitt wird die Auswirkung unterschiedlicher Größen des Target-Datensatzes auf die Performanz der Netzwerke untersucht. Es wird die Hypothese überprüft, dass die Leistung mit abnehmender Datenmenge schlechter wird.

Für eine vergleichbare Basis werden alle Netzwerke jeweils über insgesamt 40 Epochen trainiert, wobei nach 20 Epochen ein TF durchgeführt wird. Diese Untersuchung erfolgt anhand der Netzwerke CMP, 1DC und COD. Zusätzlich wird auch die Performanz auf dem Testdatensatz betrachtet.

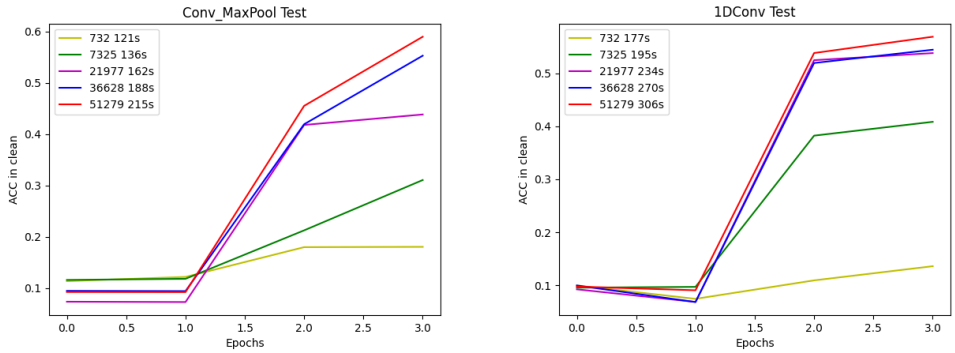


Figure 4.1: Abgebildet ist die Veränderung der Accuracy bei Convolutional-Netzwerken in Abhängigkeit von der Größe des Target-Datensatzes. Die Plots zeigen jeweils die Ergebnisse auf dem Testdatensatz des Target-Datensatzes, wobei links das CMP-Netzwerk und rechts das 1DC-Netzwerk dargestellt sind. Die dargestellten Tests umfassen für CMP die Varianten CMP:TF2/732/10, CMP:TF2/7k/10, CMP:TF2/21k/10, CMP:TF2/36k/10 und CMP:TF2/51k/10, sowie für 1DC die Varianten 1DC:TF2/732/10, 1DC:TF2/7k/10, 1DC:TF2/21k/10, 1DC:TF2/36k/10 und 1DC:TF2/51k/10, jeweils dargestellt in den Farben Gelb, Grün, Lila, Blau und Rot.

In Abbildung 4.1 sind die Testläufe in Abhängigkeit von der Größe des Trainingsdatensatzes dargestellt. Die erste Zahl in der Legende bezeichnet die Datenmenge, die zweite Zahl die Trainingsdauer. Die Datenmenge bezieht sich ausschließlich auf den Target-Datensatz. Es zeigt sich, dass die Trainingsdauer mit zunehmender Datenmenge ansteigt. Zudem bestätigt sich die Annahme, dass die Performanz bei Kaskadennetzwerken mit TF mit wachsender Datenmenge verbessert wird. Darüber hinaus weist das Deep Cascade Netzwerk eine leicht

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

bessere Performanz auf als das Direct Cascade Netzwerk, was vermutlich darauf zurückzuführen ist, dass Direct Cascade Netzwerke lediglich ein Hidden Layer besitzen, während Deep Cascade Netzwerke aus mehreren Hidden Layern bestehen und daher in der Lage sind, komplexere Problemstellungen effizienter zu erlernen.

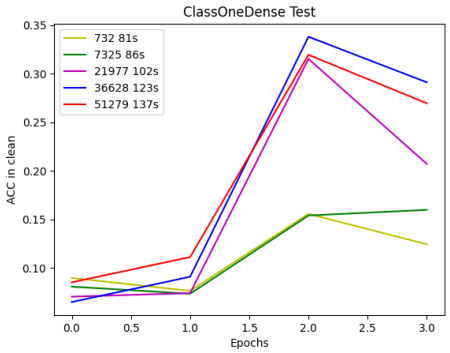


Figure 4.2: Abgebildet ist die Veränderung der Test-Accuracy eines linearen Netzwerks in Abhängigkeit von der Größe des Target-Datensatzes. Die dargestellten Tests umfassen die Varianten COD:TF2/732/10, COD:TF2/7k/10, COD:TF2/21k/10, COD:TF2/36k/10 und COD:TF2/51k/10, dargestellt in den Farben Gelb, Grün, Lila, Blau und Rot.

Abbildung 4.2 zeigt die Ergebnisse des Direct Cascade Netzwerks, das ausschließlich lineare Hidden Layer verwendet. Auffällig ist, dass dieses Netzwerk unabhängig von der Anzahl der Trainingsdaten stets eine schlechtere Performanz aufweist als die beiden anderen Netzwerke, welche Convolutional Layer enthalten. Dies ist darauf zurückzuführen, dass das lineare Netzwerk die relevanten Merkmale für die Bilderkennung nicht adäquat extrahieren kann, da die Daten eine komplexe und nicht-lineare Struktur aufweisen.

Generell zeigt sich in allen Testplots, dass die erzielte Accuracy der Netzwerke mit TF selbst bei ausreichender Datenmenge, die ein direktes Training auf dem Target-Datensatz ermöglichen würde, nie eine zufriedenstellend hohe Accuracy erreicht. Das direkte Training auf dem Target-Datensatz mit dem Deep Cascade Netzwerk mit sehr vielen Trainingsbeispielen hingegen erzielt eine Accuracy von etwa 70%, was deutlich über den Ergebnissen der TF Netzwerke liegt.

## 4.2 Bilddimensionalität

Die beiden Convolutional Direct Cascade Netzwerke unterscheiden sich ausschließlich darin, ob die Eingabedaten in ein- oder zweidimensionaler Form verarbeitet werden. Es lässt sich jedoch erkennen, dass die zweidimensionale Variante eine leicht bessere Performanz erzielt.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Der Grund dafür liegt in der Art der Filteroperationen: Während das eindimensionale Netzwerk in den Convolutional Layern lediglich benachbarte Datenpunkte entlang einer Achse berücksichtigt, kann das zweidimensionale Netzwerk zusätzlich auf Informationen aus vertikalen sowie diagonalen Nachbarschaften zugreifen. Dadurch wird eine umfassendere Erfassung lokaler Bildstrukturen ermöglicht.

Der Leistungsunterschied bleibt jedoch gering, wie in Abbildung 4.3 dargestellt.

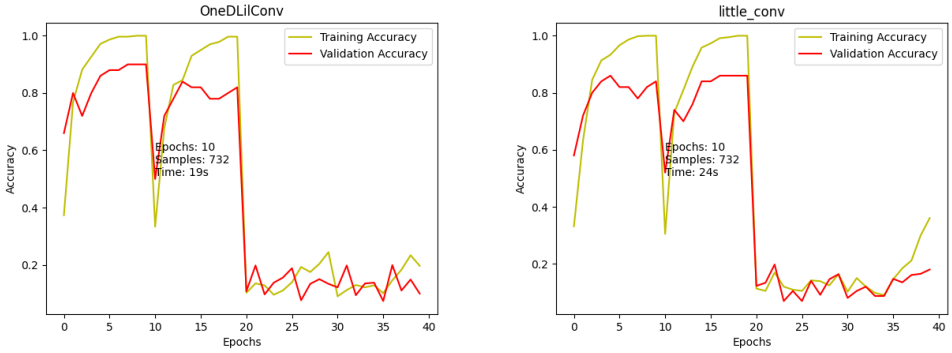


Figure 4.3: In der Abbildung sind links die Ergebnisse des Tests 1DC:TF2/732/10 und rechts jene von 2DC:TF2/732/10 dargestellt. Ziel dieser Gegenüberstellung ist es, einen möglichst geringen Unterschied in der Performanz zwischen beiden Netzwerkarchitekturen nachzuweisen, um die weitere Analyse auf das 1DC-Netzwerk beschränken zu können.

Da das zweidimensionale Netzwerk aufgrund technischer Einschränkungen nur mit einer begrenzten Datenmenge eingesetzt werden kann, sollte es in diesem Fall eine deutlich kürzere Trainingszeit aufweisen. Die längere Dauer resultiert daraus, dass die Berechnung des Augmented Vectors einen erheblichen Zeitaufwand erfordert und gleichzeitig die Hauptursache für die Einschränkungen darstellt, da während der Berechnung Speicherplatzengpässe im Arbeitsspeicher auftreten.

Da der Unterschied in der Performanz zwischen ein- und zweidimensionalem Netzwerk lediglich marginal ist, genügt es in den meisten Fällen, das eindimensionale Netzwerk für die Analyse heranzuziehen. Die damit verbundenen technischen Limitationen des zweidimensionalen Netzwerks stellen somit kein wesentliches Hindernis für die Evaluation des TFs dar.

## 4.3 Augmentierung

In diesem Abschnitt wird die Erstellung der Augmented Vectors für die Direct Cascade Netzwerke im Klassifikationskontext beschrieben. Jedes der drei betrachteten Netzwerke – COD, 1DC und 2DC – verwendet dabei eine eigene

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Methode zur Generierung dieser Vektoren. Die Darstellung erfolgt in der genannten Reihenfolge: zunächst das COD-, anschließend das 1DC- und abschließend das 2DC-Netzwerk.

Allen Netzwerken gemeinsam ist, dass sowohl der Input des Netzwerks als auch dessen Prediction zur Berechnung des Augmented Vectors herangezogen werden. Als Input dient entweder der ursprüngliche Datensatz oder – ab der zweiten Iteration – der jeweils zuletzt erzeugte Augmented Vector. Nur bei der ersten Iteration wird der originale Datensatz verwendet, da zu diesem Zeitpunkt noch kein Augmented Vector existiert.

Mit jeder weiteren Iteration wächst der Augmented Vector, da er Informationen aus allen vorhergehenden Netzwerken integriert und so das bisherige Wissen an das nächste Netzwerk weitergibt. Die Prediction entspricht dabei der Ausgabe des trainierten Netzwerks auf Basis der jeweils aktuellen Eingabe.

Im Folgenden bezeichnen die Variablen N, W, H und C die Anzahl der Datensamples, die Bildbreite, die Bildhöhe sowie die Anzahl der Kanäle (Channels).

Für das COD-Netzwerk gilt: Der Input liegt in der Form (N,WxH) vor, d.h., die Bilddaten wurden vorab in ein Vektorformat überführt. Die Prediction hat die Form (N,10), entsprechend der zehn Klassenzugehörigkeiten. Beide Matrizen werden entlang der zweiten Dimension miteinander konkateniert, was zur Definition des Augmented Vectors gemäß Gleichung 4.1 führt.

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

Beim 1DC-Netzwerk liegt der Input in der Form (N,WxH,C) vor. Da der Kanal in diesem Fall eindimensional ist, wird die Channel-Dimension zunächst entfernt. Anschließend erfolgt die Berechnung des Augmented Vectors gemäß Gleichung 4.1. Nach der Berechnung wird die Channel-Dimension wieder ergänzt, um die ursprüngliche Struktur beizubehalten. In beiden bisher betrachteten Netzwerken (COD und 1DC) wächst der Augmented Vector mit jeder Iteration linear um $N * 10$ Einträge.

Das 2DC-Netzwerk hingegen arbeitet mit einem komplexeren Eingabeformat: (N,W,H,C). Die Prediction liegt auch hier in der Form (N,10) vor. Zur Fusion von Input und Prediction werden für jedes Sample N zehn zusätzliche Arrays erzeugt, jeweils in der Form (W,H,C). Diese Arrays enthalten in jedem Eintrag die Wahrscheinlichkeitswerte der Prediction für jede der zehn Klassen in Bezug auf N. Anschließend erfolgt eine Konkatenation entlang der Channel-Dimension, wodurch für jedes N sowohl die Eingangsdaten als auch die Vorhersagewahrscheinlichkeiten aller zehn Klassen kanalweise zusammengeführt werden. Dieses Verfahren führt zur Definition des Augmented Vectors gemäß Gleichung 4.2.

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

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

Der sogenannte ConVec bezeichnet den Vektor, in dem die einzelnen Werte der Prediction – entsprechend den Klassen eins bis zehn – jeweils in Form eines Arrays der Dimension (W,H,C) abgelegt sind. Bei jeder Iteration des Netzwerks führt dieses Verfahren jedoch zu einem speicherintensiven Wachstum des Augmented Vectors. Die Skalierung des Speicherbedarfs erfolgt gemäß der in Gleichung 4.3 dargestellten Beziehung.

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

Daraus ergibt sich, dass der Arbeitsspeicherbedarf bei jeder Iteration mit einer Steigerungsrate wächst, die dem Zehnfachen der Größe des jeweiligen Datensatzes entspricht. Bei Datensamples, die bereits im Ausgangszustand eine Größe von 8192 Bytes pro Datenpunkt aufweisen, führt dieses Vorgehen zu einem sehr stark linear ansteigenden Speicherverbrauch. Daher ist die hier beschriebene Methode zur Erstellung des Augmented Vectors aus praktischer Sicht nicht sinnvoll einsetzbar. Aus diesem Grund wird das 2DC-Netzwerk im weiteren Verlauf der Arbeit nicht mehr berücksichtigt.

## 4.4 Mit und Ohne

### 4.4.1 TF

In diesem Abschnitt werden die Direct Cascade Netzwerke jeweils einmal mit und einmal ohne TF evaluiert. Im Vergleich zu den bisherigen Experimenten erfolgt das Training mit einer deutlich erhöhten Anzahl an Netzwerkiterationen innerhalb der Kaskade, wobei die Anzahl der Epochen pro Einzelnetzwerk konstant bleibt. Für alle Tests wird ausschließlich eine geringe Menge an Target-Daten verwendet.

Wie in Abbildung 4.4 ersichtlich, besteht kein signifikanter Unterschied in der Accuracy zwischen den Modellen mit und ohne TF in den Convolutional Layern. In beiden Fällen bleibt die Leistung auf einem sehr niedrigen Niveau, was sich ebenfalls in den Testergebnissen widerspiegelt.

Das beobachtete Rauschen in den Darstellungen resultiert daraus, dass nach jeweils zehn Epochen ein neues Netzwerk initialisiert und trainiert wird. Obwohl das Wissen der zuvor trainierten Netzwerke über die Eingabedaten verfügbar ist, wird dieses nicht unmittelbar durch optimale Gewichtungen transferiert.

Abbildung 4.5 zeigt ein vergleichbares Ergebnis, basierend auf linearen Schichten (Layer). Für dieses Verhalten kommen zwei Ursachen in Betracht: Zum einen könnte das Kaskadieren der Netzwerke nicht wie vorgesehen funktionieren, zum anderen könnte eine unzureichende Menge an Target-Daten vorliegen. Letzteres wurde bereits in vorangegangenen Untersuchungen überprüft und führte zwar zu einer Verbesserung, jedoch blieben die Resultate nur moderat.

Diese Beobachtungen legen nahe, dass insbesondere beim Kaskadieren der Netzwerke Schwierigkeiten bestehen.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

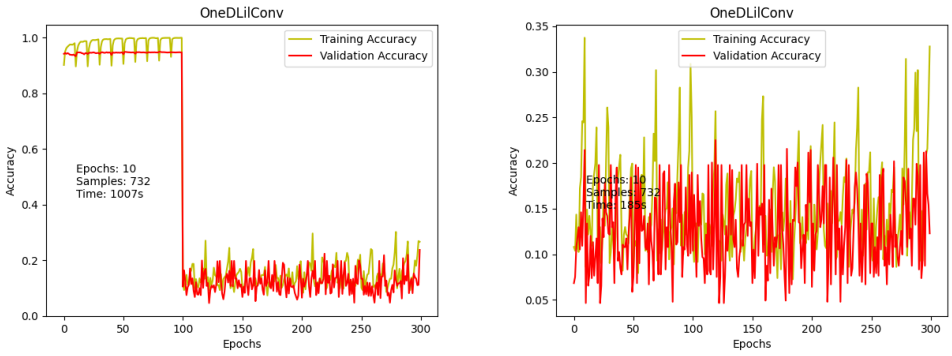


Figure 4.4: In der Abbildung sind links die Ergebnisse des Tests 1DC:TF10/732/10/30 und rechts jene des Tests 1DC:732/10/30 dargestellt. Die Variante mit TF weist eine deutlich längere Trainingszeit auf, was auf die Größe des verwendeten Source-Datensatzes zurückzuführen ist. In beiden Fällen zeigt sich jedoch, dass die Netzwerke unabhängig vom Einsatz von TF eine insgesamt geringe Performanz aufweisen.

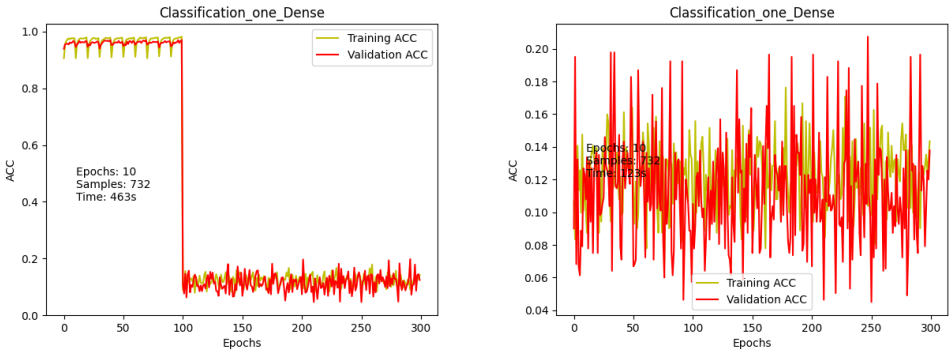


Figure 4.5: Links ist das Testergebnis des Experiments COD:TF10/732/10/30 dargestellt, bei dem TF eingesetzt wurde, rechts jenes des Experiments COD:732/10/30 ohne TF. Auch in diesem weiteren Fall, bei dem die Architektur mit Linearen Layern als Hidden Layer verwendet wurde, zeigen sich unabhängig vom Einsatz von TF ebenfalls geringe Leistungswerte.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

### 4.4.2 Kaskadierung

Im Folgenden wird untersucht, welche Auswirkungen auftreten, wenn auf das Kaskadieren verzichtet wird. Da ein zufriedenstellendes Ergebnis nur ohne Verwendung von TF zu erwarten ist, erfolgt das Training direkt auf dem Target-Datensatz. Die Architektur der vollständigen Netzwerke wird so angepasst, dass die Gesamtanzahl der Hidden Layer der Summe der Hidden Layer aller einzelnen Netzwerke im Direct-Cascade-Verfahren entspricht. Dabei wird die Gesamtanzahl der Trainings-Epochen beibehalten, um eine vergleichbare Trainingsdauer sicherzustellen.

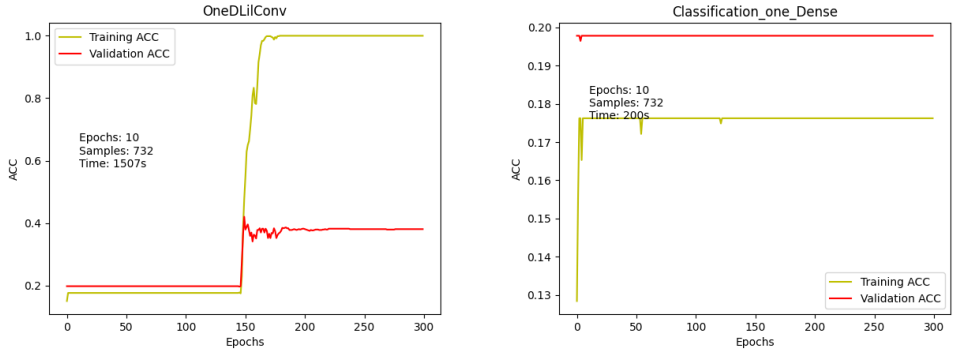


Figure 4.6: Die dargestellten Ergebnisse zeigen Testläufe ohne Kaskadierung. Konkret sind links die Resultate für das 1DC-Netzwerk (1DC:Comp/732//30) und rechts für das COD-Netzwerk (COD:Comp/732//30) dargestellt. Es ist ersichtlich, dass eines der Modelle in einem lokalen Maximum stecken bleibt. Zudem lässt sich aus den Ergebnissen ableiten, welcher maximale Accuracy-Wert unter der gegebenen, begrenzten Datenmenge realistischerweise erreicht werden kann. Dieser Wert wird ausschließlich durch den Einsatz der vollständigen, hier beschriebenen Netzwerkarchitekturen erzielt.

In Abbildung 4.6 fällt auf, dass während der meisten Epochen kein Lerneffekt eintritt. Zudem ist Overfitting erkennbar – ein zu erwartendes Verhalten angesichts der geringen Menge an Trainingsdaten. Obwohl in diesem Experiment kein TF eingesetzt wurde, zeigt einer der beiden Plots in der Mitte einen plötzlichen Anstieg der Accuracy. Dieser Anstieg wurde durch eine minimale Verbesserung des Trainingswerts bei gleichzeitig minimaler Verschlechterung des Validierungswerts ausgelöst; beide Änderungen lagen im Bereich von Zehntelprozenten. Dies deutet auf das Erreichen eines lokalen Maximums im Trainingsverlauf hin.

Im Vergleich dazu bleibt das andere Netzwerk dauerhaft auf dem Niveau dieses lokalen Maximums – beide Resultate zeigen exakt identische Werte. Aus Abbildung 4.6 lässt sich zudem ableiten, dass unter den gegebenen Bedingungen eine maximale Accuracy von etwa 40% erreichbar ist. Dieser Wert stellt das globale Maximum dar, da er die bestmögliche Performanz auf den Train-

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

ingsdaten widerspiegelt.

Weder das reine Kaskadieren noch die Kombination aus Kaskadierung und TF erreichen vergleichbare Ergebnisse – beide Varianten erreichen lediglich eine maximale Accuracy von etwa 20% und damit nur etwa die Hälfte der möglichen Leistung.

Diese Beobachtungen legen nahe, dass die Ursache für die stark eingeschränkte Klassifikationsleistung im Direct-Cascade-Verfahren mit TF bereits in der Art der Kaskadierung selbst zu suchen ist. Mögliche Gründe hierfür könnten in der wiederholten Anwendung der Categorical-Crossentropy-Verlustfunktion liegen, die sich gegenseitig negativ beeinflussen könnte. Alternativ könnte auch die Softmax-Aktivierungsfunktion oder das konkrete Vorgehen beim Kaskadieren die Ursache darstellen.

Letztlich erweist sich der Einsatz von TF in Kombination mit dem Direct-Cascade-Verfahren unter Verwendung der in dieser Arbeit genutzten Datensätze und der beschriebenen Augmented Vectors als nicht zielführend, da die erzielten Accuracy-Werte selbst bei umfangreichen Trainingsdaten 60% nicht überschreiten und im Vergleich zu einem vollständig trainierten Netzwerk signifikant schlechter ausfallen.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# Chapter 5

# Regression

In diesem Kapitel werden die spezifischen Erkenntnisse zur Regressionsaufgabe dargestellt und analysiert. Zunächst erfolgt die Einführung der beiden Netzarchitekturen – Deep Cascade und Direct Cascade –, um darauf aufbauend einen Vergleich zwischen den Varianten Cascade mit Transferlernen (TF), Cascade ohne TF sowie der jeweiligen vollständigen Versionen der Netzwerke unter Bedingungen mit vielen bzw. wenigen Target-Daten vorzunehmen. Dabei zeigen sich deutliche Unterschiede zwischen Deep und Direct Cascade, wobei insgesamt eine prinzipielle Funktionsfähigkeit beider Ansätze bestätigt werden kann. Abschließend werden Early-Stopping-Metriken angewendet und deren Ergebnisse diskutiert.

Die beiden Regressionsnetzwerke werden im Folgenden im Detail vorgestellt. Beide Modelle erhalten als Eingabe tabellarische Daten mit drei Merkmalen, deren genaue Beschreibung bereits im Kapitel 2 erfolgte. Für das Training kommt in beiden Fällen der Adam-Optimierer in Kombination mit der Mean-Squared-Error-Verlustfunktion zum Einsatz. Der Output-Layer entspricht einem typischen Regressions-Setup: ein einzelner linearer Layer mit einer Neuroneneinheit und linearer Aktivierungsfunktion.

In Abbildung 5.1 ist die vollständige Architektur des regression_two-Netzwerks (Regr2) dargestellt. Es handelt sich dabei um ein Deep-Cascade-Netzwerk, bei dem die Layer schrittweise – also Layer für Layer – trainiert werden. Die Zahl hinter den Linear-Layern gibt jeweils die Anzahl der enthaltenen Neuronen an, während die Angabe hinter den Dropout-Layern den Anteil (bezogen auf den Wert 1) der während des Trainings pro Epoche deaktivierten Neuronen beschreibt.

Das sogenannte Regression_one_Layer-Netzwerk (1Lay) stellt das Direct-Cascade-Regressionsmodell dar. Es besteht aus einem einzigen Hidden Layer in Form einer Linear-Schicht mit 128 Neuronen und verwendet die ReLU-Aktivierungsfunktion. Dieses Netzwerk wird iterativ eingesetzt, wobei zwischen den einzelnen Trainingsdurchläufen Wissen in Form eines Augmented Vectors übertragen wird. Dieser Vektor entsteht, indem die Prediction des jeweils vorherigen Netzwerks als zusätzliche Spalte zur ursprünglichen Input-Tabelle hinzugefügt wird. Die

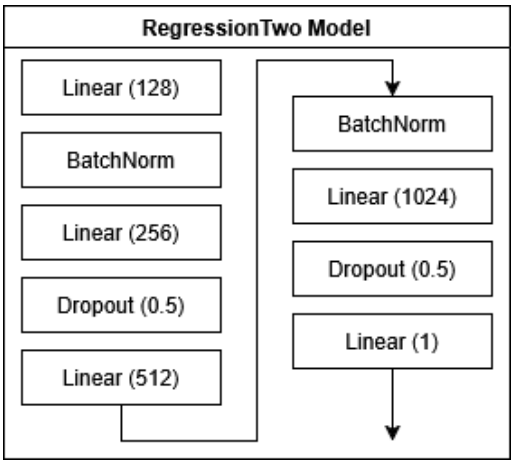


Figure 5.1: In dieser Abbildung ist die Architektur des Regr2-Netzwerks im Detail dargestellt. Die Anordnung der Layer erfolgt von oben nach unten entlang der angegebenen Pfeilrichtung. Bei den Linear-Layern ist die jeweilige Anzahl der enthaltenen Neuronen angegeben; bei den Dropout-Layern gibt der Wert den Prozentsatz der Neuronen an, die während des Trainings zufällig deaktiviert werden.

erweiterte Eingabematrix dient anschließend als Input für das nachfolgende Netzwerk.

## 5.1 Datenaugmentation

Der Source-Datensatz BOST umfasst lediglich 506 Datenpunkte und ist damit als sehr klein einzustufen. Für die folgenden Experimente werden davon 51 Samples als Testmenge, 91 als Validierungsmenge und 364 als Trainingsmenge verwendet.

Im Gegensatz dazu ist der Target-Datensatz CALI mit etwa 26.000 Einträgen deutlich umfangreicher und wird je nach Versuchskonfiguration entsprechend reduziert. Das Training erfolgt mit einer Batchgröße von 16 Samples.

### 5.1.1 Viele Daten

In diesem Abschnitt werden sämtliche Vergleichsexperimente unter Verwendung des vollständigen Target-Datensatzes durchgeführt. Dabei umfasst die Trainingsmenge etwa 8.000 Datenpunkte, während die Testmenge rund 4.000 Samples beinhaltet.

Verglichen werden drei Netzwerkvarianten: vollständige Netzwerke, Cascade-Netzwerke ohne TF sowie Cascade-Netzwerke mit TF. Unter einem vollständgien Netzwerk wird ein Modell verstanden, dessen gesamte Architektur – inklusive aller Layer – im Voraus definiert ist und das in einem einzigen Trainingslauf optimiert wird. Dies entspricht dem üblichen Vorgehen beim Training klassischer

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

neuronaler Netzwerke.

Es erfolgt ein direkter Vergleich zwischen dem Deep-Cascade-Ansatz, dem Direct-Cascade-Verfahren und dem vollständigen Netzwerkmodell.

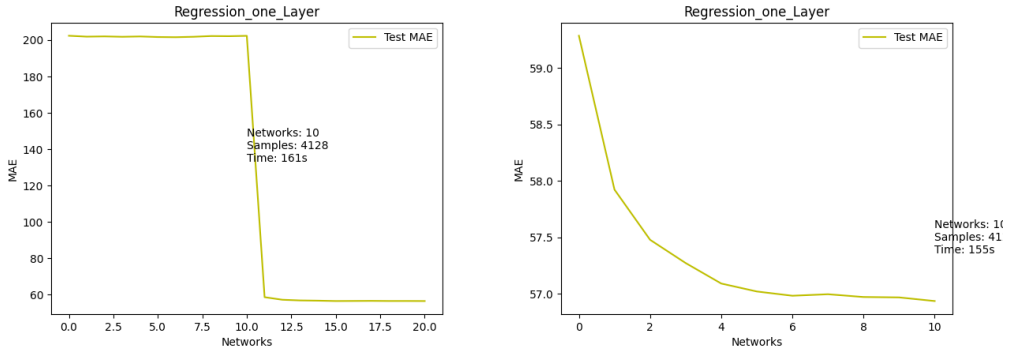


Figure 5.2: In der Abbildung sind die Ergebnisse der Testläufe 1Lay:TF11/8k/10/21 und 1Lay:8k/10/11 dargestellt. Beide Modelle erreichen am Ende eine vergleichbare Leistung, unterscheiden sich jedoch hinsichtlich der Anzahl an Trainingsdurchläufen bzw. der benötigten Zeit bis zum Erreichen dieses Leistungsniveaus.

Abbildung 5.2 zeigt die Ergebnisse des Direct Cascade-Verfahrens. Die linke Seite stellt die Variante mit TF dar. Es fällt auf, dass bei vielen Target-Daten das direkte Training auf den Target-Daten zu besseren Ergebnissen führt, da das vom Source-Datensatz übertragene Wissen oftmals weniger gut auf die Target-Daten übertragbar ist. Dieser Effekt tritt jedoch nur bei ausreichender Verfügbarkeit von Target-Daten auf. Im Vergleich dazu erzielen auch die Deep Cascade-Modelle vergleichbare Ergebnisse. Ein vollständig trainiertes Netzwerk, wie in Abbildung 5.3 dargestellt, erreicht eine ähnliche Performanz wie die beiden Kaskadenversionen.

Dies ist darauf zurückzuführen, dass die lineare Aktivierungsfunktion sowie der MSE Loss im Kontext der Kaskadierung keine negativen Effekte verursachen. Dementsprechend erzielt die Regression mit Kaskadennetzwerken eine deutlich bessere Leistung als die Klassifikation, da sie nahe an die Ergebnisse eines vollständig trainierten Netzwerks heranreicht – ein Effekt, der bei Klassifikationsaufgaben bisher nicht beobachtet wurde.

Da das Netzwerk vergleichbare Ergebnisse mit TF erzielt, kann davon ausgegangen werden, dass der Einsatz von TF in diesem Fall erfolgreich war.

### 5.1.2 Wenig Daten

In diesem Unterkapitel wird die Größe des Target-Datensatzes signifikant reduziert, sodass lediglich noch 240 Stichproben zur Verfügung stehen. Die gleichen Tests wie im vorherigen Abschnitt werden erneut durchgeführt.

Abbildung 5.4 zeigt die Ergebnisse des Deep Cascade Netzwerks. Tatsächlich erzielt das Netzwerk ohne TF eine bessere Performanz als mit TF. Dies ist

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

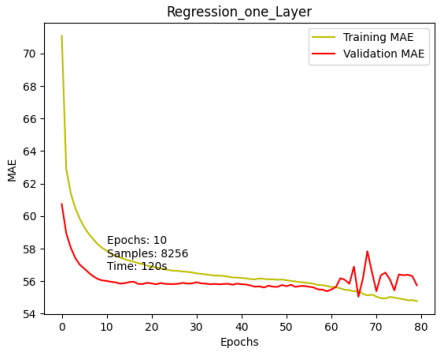


Figure 5.3: Bei dem vorliegenden Modell handelt es sich um den Test 1Lay:Comp/8k//8. Dieser wurde unter Verwendung einer großen Menge an Target-Daten direkt trainiert, weshalb die erzielten Ergebnisse im weiteren Verlauf als Referenz für die bestmögliche Leistung herangezogen werden.

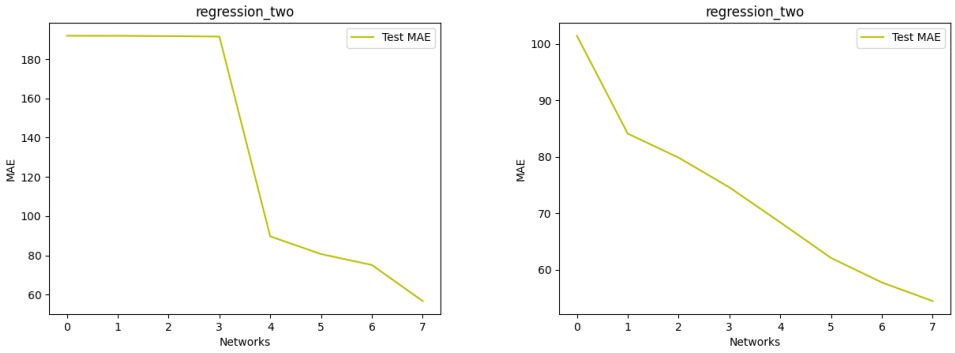


Figure 5.4: Abgebildet ist der Vergleich des Regr2-Netzwerks mit und ohne TF bei geringer Verfügbarkeit von Target-Daten. Die Tests entsprechen links Regr2:TF4/240/10/8 und rechts Regr2:240/10/8. Dabei zeigt sich, dass das Netzwerk ohne TF eine bessere Performanz erzielt als mit.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

darauf zurückzuführen, dass die Gewichte der ersten Netzwerkhälfte primär auf dem Source-Datensatz optimiert wurden und somit weniger gut auf den Target-Datensatz übertragbar sind. Eine effektive Anpassung auf dem Target-Datensatz ist ebenfalls nicht möglich, da dieser eine unzureichende Anzahl an Stichproben enthält. Deep Cascade mit aktiviertem TF ist jedoch in einem Leistungsbereich, der eine praktische Nutzung weiterhin rechtfertigt.

Da sich die Ergebnisse deutlich von denen des Direct Cascade unterscheiden, wird im Folgenden die andere Netzwerkarchitektur betrachtet.

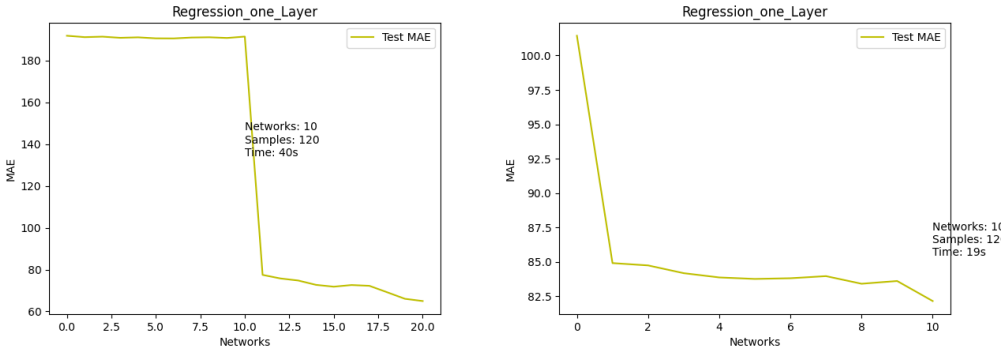


Figure 5.5: Die Abbildung zeigt den Vergleich des 1Lay-Netzwerks im Szenario mit geringer Anzahl an Target-Daten, jeweils mit und ohne TF. Die zugrunde liegenden Tests lauten: links 1Lay:TF11/240/10/21 und rechts 1Lay:240/10/11. Auffällig ist, dass in diesem Fall das Netzwerk mit TF eine bessere Leistung erzielt als ohne.

Abbildung 5.5 zeigt die Ergebnisse des Direct Cascade Netzwerks. Es wird deutlich, dass diese Kaskadierungsvariante ohne TF eine signifikant schlechtere Leistung erzielt als mit TF. Dies weist auf einen positiven Transfer hin: Die Predicitons, die mithilfe des Source-Datensatzes auf bisher unberücksichtigte Target-Daten generiert werden, tragen zur Verbesserung der Gesamtergebnisse bei. Dadurch steht zu Beginn des Trainings auf dem Target-Datensatz eine größere Menge an Daten pro Stichprobe zur Verfügung. Diese zusätzlichen Informationen wirken sich nicht negativ auf die Modellleistung aus, sodass das Training mit TF insgesamt bessere Resultate liefert. Das Direct Cascade Netzwerk zeigt mit aktiviertem TF eine mit dem vollständigen Netzwerk etwas schlechtere, aber vergleichbare Performanz und erfüllt somit die Anforderungen für eine praxisnahe Anwendung.

Das Deep Cascade Netzwerk erreicht hingegen unabhängig vom Einsatz von TF eine leicht bessere Performanz. Dieser Unterschied könnte jedoch auch auf die abweichende Netzwerkarchitektur zurückzuführen sein, da die Layer-Strukturen nicht vollständig identisch sind. Dies könnte jedoch darauf zurückzuführen sein, dass die Gewichtungen der initialen Layer das Lernen auf dem Target-Datensatz nur geringfügig beeinflussen und die relevanten Merkmalsrepräsentationen primär in den nachfolgenden Hidden Layers erlernt werden können.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

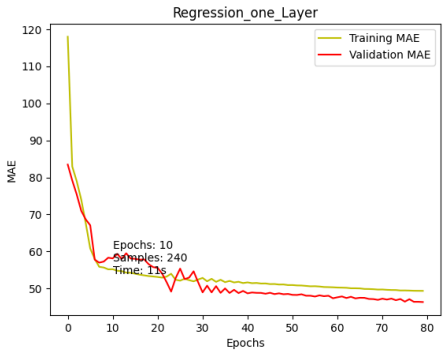


Figure 5.6: Dem dargestellten Plot liegt der Test 1Lay:Comp/240//8 zugrunde. Trotz der geringen Anzahl an verfügbaren Daten erzielt dieses Modell bessere Ergebnisse als alle anderen getesteten Varianten – obwohl es weder Kaskadierung noch TF verwendet.

Die besten Ergebnisse liefert schließlich die Variante ohne Kaskadierung und ohne TF, bei der ein vollständiges Netzwerk direkt auf dem Ziel-Datensatz trainiert wurde, wie in Abbildung 5.6 dargestellt.

Der MAE-Wert auf den Testdaten liegt in diesem Fall bei etwa 53.000 US-Dollar, während er bei den Kaskadennetzwerken üblicherweise im Bereich von 60.000 bis 80.000 US-Dollar anzusiedeln ist.

Ein neuronales Netzwerk ohne Kaskadierung, jedoch mit einer hohen Anzahl an Hidden Layers und einem einzigen Trainingsdurchlauf über 80 Epochen, zeigt selbst bei geringer Datenmenge eine bessere Leistung als Netzwerke mit Kaskadierung. Es erreicht zudem nahezu den Fehlerwert, der bei einem deutlich größeren Trainingsdatensatz erzielt wird.

Im Gegensatz zu den anderen Netzarchitekturen besitzt dieses Modell die Fähigkeit, zwischen den Layern zu lernen und unvollständig verarbeitete Eingabedaten für die weitere Verarbeitung heranzuziehen. Dies dürfte maßgeblich zur verbesserten Modellperformanz beitragen.

Wird die Evaluation mit einem explizit großen Testdatensubset durchgeführt, so verschlechtert sich der absolute MAE-Wert bei allen Tests geringfügig. Die relativen Leistungsunterschiede zwischen den verschiedenen Netzarchitekturen bleiben jedoch unverändert.

Die Anwendung von TF ist sowohl in Kombination mit Deep Cascade als auch mit Direct Cascade möglich, wobei in beiden Fällen eine akzeptable Performanz erzielt werden kann. Für den praktischen Einsatz erscheinen daher beide Varianten grundsätzlich geeignet. Es ist jedoch empfehlenswert, weitere Untersuchungen mit deutlich kleineren Datensätzen durchzuführen, da bisher primär die Kaskadennetze einen Qualitätsverlust zeigten, wenn von großen auf kleinere Datenmengen umgestellt wurde.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

## 5.2 Early Stopping

In diesem Abschnitt werden Regressionsnetzwerke unter Einsatz von Early Stopping evaluiert, wobei erläutert wird, warum diese Methode für Klassifikationsnetzwerke nicht zielführend ist. Der Fokus liegt hierbei ausschließlich auf Direct Cascade Netzarchitekturen. Early Stopping stellt eine Methode dar, die eingesetzt wird, um Overfitting des Modells auf die Trainingsdaten entgegenzuwirken.

Für sowohl die Regressions- als auch die Klassifikationsmodelle wurde LM als Stoppkriterium verwendet. Zusätzlich kamen bei der Regression die MAEM und bei der Klassifikation die ACCM als Metriken zum Einsatz.

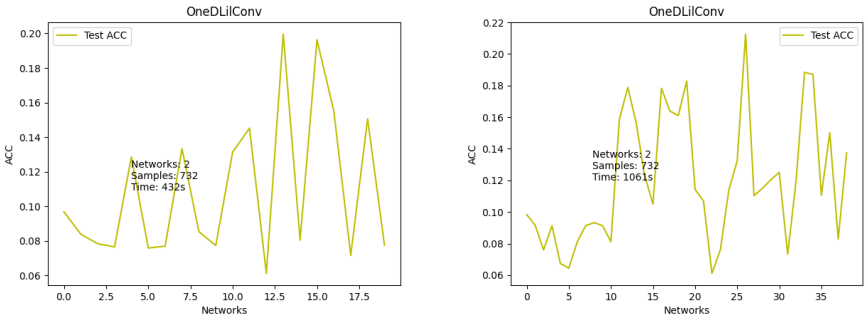


Figure 5.7: Hier werden die Testergebnisse für das 1DC-Netzwerk unter Verwendung von Early Stopping dargestellt. Deutlich erkennbar ist, dass sich die Modellleistung durch Early Stopping nicht verbessert. Die exakten Testbezeichnungen lauten: links 1DC:LM/732/10 und rechts 1DC:ACCM/732/10

Bei den Klassifikationsnetzen führt das Early Stopping auf Basis von ACCM nur gelegentlich zu einem vorzeitigen Abbruch des Trainings, ohne jedoch eine Verbesserung der Modellleistung zu bewirken. Unter Verwendung von LM tritt dieser Abbruch häufiger auf, was zu einer schnelleren Trainingskonvergenz führt. Dennoch bleibt die Leistung der kaskadierten Klassifikationsnetzwerke derart unzureichend, dass ein praktischer Einsatz ausgeschlossen ist. Die Ineffektivität von sowohl LM als auch ACCM in diesem Kontext wird anschaulich in Abbildung 5.7 dargestellt. Hervorzuheben ist, dass ACCM die einzige der betrachteten Metriken ist, bei der ein Maximum angestrebt wird.

Daher erfolgt im Folgenden eine vertiefte Analyse des Regressionsnetzwerks 1Lay. Dabei zielen die verwendeten Metriken, LM und MAEM, auf eine Minimierung des Fehlermaßes ab.

Die Anwendung der Early-Stopping-Metriken LM und MAEM liefert zwar teilweise akzeptable Ergebnisse, jedoch sind diese deutlich schlechter als die Resultate eines Trainings ohne Early Stopping, wie aus den Werten in Abbildung 5.8 ersichtlich ist.

Die Qualität der erzielten Ergebnisse entspricht in etwa der, als hätte das 1Lay-Netzwerk in der Kaskadenversion mit einer geringen Anzahl an Target-Daten direkt auf diese trainiert. Die unzureichende Leistung der Early-Stopping-

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

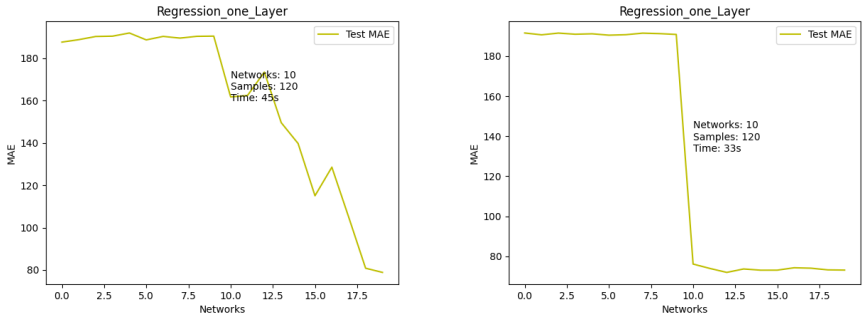


Figure 5.8: Hier sind die Testergebnisse des 1Lay-Netzwerks unter Verwendung von Early Stopping veranschaulicht. Es zeigt sich, dass die Modellperformanz mit Early Stopping schlechter ausfällt als ohne. Obwohl es in einigen Fällen zu vorzeitigen Trainingsabbrüchen kam, wurde dadurch Overfitting nicht entscheidend verhindert. Die eingesetzten Metriken erweisen sich somit als ineffektiv. Die genauen Bezeichnungen der Testläufe lauten: links 1Lay:LM/120/10 und rechts 1Lay:MAEM/120/10.

Metriken ist darauf zurückzuführen, dass sie keine temporären Verschlechterungen im Validierungsdatensatz tolerieren und das Training bei der ersten beobachteten Verschlechterung abbrechen. Dadurch wird häufig nicht das tatsächliche Minimum des Fehlermaßes über den augmentierten Vektor weitergegeben, sondern lediglich ein leicht abweichender Wert.

Zudem sind diese Metriken nicht in der Lage, ein globales Minimum zu identifizieren, wenn das Trainingsverfahren in einem lokalen Minimum verharrt. In einem solchen Fall führt das Early Stopping dazu, dass das Modell im lokalen Minimum verbleibt, anstatt weiter zu optimieren.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# Chapter 6

# Zusammenfassung

## 6.1 Erkenntnisse

**Bei Klassifikation und Regression:**
Es zeigt sich ein relativ schneller Overfitting-Effekt auf dem Source-Datensatz.

An der Stelle des TFs bricht die Performanz des Netzes deutlich ein. Anschließend kommt es zwar zu einer teilweisen Erholung, jedoch erreicht die Leistung nicht mehr das vorherige Niveau.

In der vorliegenden Implementierung sind die herkömmlichen Netzwerke bei kleinen Datensätzen und flacher Netzstruktur schneller als die Direct Cascade Netzwerke, welche wiederum schneller trainieren als die Deep Cascade Architekturen. Die Performanz variiert jedoch signifikant zwischen den einzelnen Netzwerktypen.

Bei zunehmender Netzwerktiefe und einer hohen Anzahl an Trainingsepochen kehrt sich das Zeitverhältnis zwischen den Kaskadenversionen des Netzwerks und dem vollständigen Netzwerk um, sodass Letzteres im Vergleich deutlich langsamer ist als beide Kaskadenvarianten.

**Klassifikation:**
Bei der Klassifikation ist der Accuracy-Wert primär von der Größe des Target-Datensatzes abhängig: Je geringer die Datenmenge, desto schlechter fällt die Klassifikationsgenauigkeit aus.

Die Klassifikationsleistung ist derart unzureichend, dass ein Einsatz in diesem Kontext nicht sinnvoll erscheint, insbesondere wenn die Accuracy nur bei etwa 20% liegt.

Bei TF zeigt sich eine leicht schlechtere Performanz im Vergleich zum Training ohne TF.

Ein Cascade-Netzwerk erzielt eine schlechtere Leistung als ein vollständig trainiertes Netzwerk ohne Kaskadierung.

Die Verarbeitung mehrdimensionaler Augmented Vectors kann zu Problemen mit dem Arbeitsspeicher führen.

Die eindimensionale Klassifikation ist geringfügig schlechter als die zwei-

dimensionale; dieser Unterschied ist jedoch minimal und kann vernachlässigt werden.

**Regression:**
Bei der Regression führt der Einsatz von TF bei wenigen Trainingsdaten zu besseren Ergebnissen als ein Training des Netzwerks ausschließlich auf dem Target-Datensatz von Grund auf — dies gilt jedoch nur für Direct Cascade Netzwerke.

Die Leistungsfähigkeit der Regressionsnetzwerke mit eingesetztem TF ist ausreichend, um eine praktische Anwendbarkeit zu gewährleisten.

Der Performanz-Abfall bei der Regression fällt insgesamt deutlich geringer aus als bei der Klassifikation.

Die hier verwendeten einfachen Early-Stopping-Metriken verschlechtern die Ergebnisse, da sie dazu neigen, in lokalen Minima stecken zu bleiben.

## 6.2 Ausblick

Die Klassifikation sollte nicht weiter im Rahmen von Deep- oder Direct-Cascade-Architekturen untersucht werden, da die Performanz unabhängig von der Datenmenge konstant schlechter ist als bei einem vollständig trainierten Netzwerk. Eine potenzielle Verbesserung könnte nur durch die Verwendung alternativer Loss-Funktionen anstelle von Categorical Cross-Entropy oder durch eine modifizierte Konstruktion der Augmented Vectors zwischen den Netzwerkmodulen erzielt werden. Ebenso wäre die Erprobung anderer Kaskadierungsverfahren denkbar, wobei deren Erfolgsaussichten jedoch als gering eingeschätzt werden.

Für die Regressionsnetzwerke besteht die Möglichkeit, alternative Loss-Funktionen anstelle des MSEs einzusetzen. Darüber hinaus sollten auch Target-Datensätze mit weniger als einhundert Instanzen als Eingabe untersucht werden, um die Leistungsfähigkeit der Modelle unter noch limitierteren Bedingungen zu evaluieren.

In beiden Anwendungsfällen könnten zudem andere Early-Stopping-Kriterien angewandt werden, die auch geringfügige Verschlechterungen im Validierungsmaß tolerieren. Zusätzlich sind Metriken denkbar, die die Anzahl der Netziterationen adaptiv steuern. Des Weiteren können Trainingsverfahren ohne feste Maximalanzahl an Epochen implementiert werden, wobei hier das Risiko besteht, dass das Training unendlich lange andauert.

Darüber hinaus kann die Verwendung alternativer Optimierungsalgorithmen anstelle von Adam evaluiert werden, ebenso wie der Einsatz unterschiedlicher Source- und Target-Datensätze. Ebenso ist eine veränderte Vorverarbeitung der bestehenden Datensätze für die Netzwerke denkbar.

Weiterhin sollte untersucht werden, wie sich TF bei einem Wechsel der Aufgabenstellung verhält, ebenso wie bei einem kombinierten Task- und Domainwechsel.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

## 6.3 Fazit

Die Klassifikation ist in Kombination mit Direct Cascade und TF unter Verwendung der hier eingesetzten Augmented Vectors nicht praktikabel. Deep Cascade Architekturen erzielen zwar eine bessere Klassifikationsleistung, sind jedoch ebenfalls nicht sinnvoll einsetzbar.

Im Regressionskontext ist der Einsatz von Direct Cascade-Netzwerken mit aktiviertem TF grundsätzlich möglich, da die resultierende Performanz lediglich geringfügig unter der vollständiger Netzwerke liegt und vergleichbar mit derjenigen von Deep Cascade-Netzwerken unter Verwendung von TF ist.

Das Verfahren funktioniert auch mit Deep Cascade Netzwerken und zeigt selbst in diesem Ausnahmefall eine mit dem vollständigen Netzwerk vergleichbaren Performanz.

Vollständig trainierte Netzwerke mit der gleichen Gesamtanzahl an Hidden Layer sind insbesondere bei sehr kleinen Datensätzen und flacher Netzstruktur schneller als Deep Cascade Modelle, welche wiederum schneller als Direct Cascade Architekturen sind.

Bei tiefen Netzwerkarchitekturen und einer großen Anzahl an Trainingsepochen verändert sich das Bild dahingehend, dass vollständige Netzwerke deutlich langsamer sind als sämtliche Kaskadierungsvarianten. Dieser Effekt tritt unabhängig davon auf, ob TF verwendet wird oder nicht.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# List of Figures

2.1 In der Darstellung wird der Aufbau und Trainingsprozess von Deep Cascade Netzwerken veranschaulicht. Das zugrunde liegende Modell besteht aus mehreren Schichten (Layern), die sukzessive – also nacheinander – trainiert werden. Zur Unterstützung des schrittweisen Trainingsprozesses werden temporäre Output-Layer extern am Netzwerk angebracht. Diese dienen ausschließlich dem Training des jeweils aktuellen Abschnitts und werden nach dessen Abschluss entfernt, da ihre Integration in die Hauptstruktur des Netzwerks den internen Informationsfluss beeinträchtigen würde. Nach dem Training werden alle bisherigen Layer eingefroren. TF kann zwischen jedem Layer vollzogen werden. . . . . 12

2.2 Hier wird das Direct Cascade Verfahren dargestellt. Dieses Verfahren verwendet mehrere einzelne Netzwerke (hier als Modelle bezeichnet), die jeweils nur wenige Hidden Layer aufweisen, in der Regel lediglich einen. Nach der Initialisierung und dem Training wird jedes Modell einmal ohne weiteres Training angewendet, dessen Ausgangssignal mit dem ursprünglichen Eingabesignal kombiniert wird. Diese Kombination bildet den neuen Eingabedatensatz für das nachfolgende Modell. Durch diese sukzessive Verknüpfung der Ausgaben mit den Eingaben kann das Verfahren eine Wissensweitergabe und -integration zwischen den einzelnen Modellen realisieren. Darüber kann an beliebiger Stelle TF verwendet werden. . . . . 13

3.1 Abgebildet sind exemplarische Plots zweier Deep-Cascade-Netzwerke – links für eine Klassifikationsaufgabe, rechts für eine Regressionsaufgabe. Dargestellt werden die Tests CMP:TF2/732/10 und Regr2:TF4/240/25, die jeweils als typische Repräsentationen für den Einsatz von TF in den entsprechenden Aufgabenbereichen dienen. . . . . 19

3.2 Dem CMP-Netzwerk liegen die Layer in exakt der hier von oben nach unten dargestellten Reihenfolge zugrunde. . . . . 20

3.3 Abgebildet ist der Test CMP:TF2/732/10. Der Fokus liegt hierbei auf Epoche 20, da zu diesem Zeitpunkt das TF angewendet wurde. Infolge dessen ist ein deutlicher Einbruch der Performanz zu beobachten. . . . 20

3.4 Diese Abbildung zeigt den Test CMP:TF2/732/10. Dieser basiert auf Target-Daten, die dem Netzwerk während des Trainings nicht zugänglich waren. Nach Abschluss des Trainings jedes Layers erfolgt eine Evaluation anhand der Testdaten. Dementsprechend ist der Punkt, an dem TF durchgeführt wird, bei zwei Netzwerken erkennbar. Zu diesem Zeitpunkt verbessert sich die Performanz deutlich. . . . . 21
3.5 In der Abbildung sind von links oben nach rechts unten die Plots folgender Tests dargestellt: CMP:732/10, CMP:TF0/732/10, CMP:TF1/732/10 sowie CMP:TF2/732/10. Bei allen dargestellten Plots zeigt sich ein ähnliches Problem, nämlich eine zunehmende Diskrepanz zwischen Trainings-Accuracy und Validierungs-Accuracy über den Trainingsverlauf hinweg. . . . . 22
3.6 In der Abbildung sind links die Ergebnisse des Tests Regr2:240/25 und rechts die des Tests Regr2:TF1/240/25 dargestellt. Der MAE zwischen Trainings- und Validationsset weist dabei kaum Unterschiede auf, was auf ein fehlendes Overfitting hinweist. . . . . 23
3.7 In diesem Fall erfolgt das Training über eine hohe Anzahl an Epochen unter Einsatz tieferer und komplexerer Netzarchitekturen, wodurch sich das Verhältnis der Trainingsdauer umkehrt. Veranschaulicht wird dies anhand der Tests 1DC:Comp/732//30 sowie 1DC:TF10/732/10/30. 25

4.1 Abgebildet ist die Veränderung der Accuracy bei Convolutional-Netzwerken in Abhängigkeit von der Größe des Target-Datensatzes. Die Plots zeigen jeweils die Ergebnisse auf dem Testdatensatz des Target-Datensatzes, wobei links das CMP-Netzwerk und rechts das 1DC-Netzwerk dargestellt sind. Die dargestellten Tests umfassen für CMP die Varianten CMP:TF2/732/10, CMP:TF2/7k/10, CMP:TF2/21k/10, CMP:TF2/36k/10 und CMP:TF2/51k/10, sowie für 1DC die Varianten 1DC:TF2/732/10, 1DC:TF2/7k/10, 1DC:TF2/21k/10, 1DC:TF2/36k/10 und 1DC:TF2/51k/10, jeweils dargestellt in den Farben Gelb, Grün, Lila, Blau und Rot. . . . . 27
4.2 Abgebildet ist die Veränderung der Test-Accuracy eines linearen Netzwerks in Abhängigkeit von der Größe des Target-Datensatzes. Die dargestellten Tests umfassen die Varianten COD:TF2/732/10, COD:TF2/7k/10, COD:TF2/21k/10, COD:TF2/36k/10 und COD:TF2/51k/10, dargestellt in den Farben Gelb, Grün, Lila, Blau und Rot. . . . . 28
4.3 In der Abbildung sind links die Ergebnisse des Tests 1DC:TF2/732/10 und rechts jene von 2DC:TF2/732/10 dargestellt. Ziel dieser Gegenüberstellung ist es, einen möglichst geringen Unterschied in der Performanz zwischen beiden Netzwerkarchitekturen nachzuweisen, um die weitere Analyse auf das 1DC-Netzwerk beschränken zu können. . . . . 29
4.4 In der Abbildung sind links die Ergebnisse des Tests 1DC:TF10/732/10/30 und rechts jene des Tests 1DC:732/10/30 dargestellt. Die Variante mit TF weist eine deutlich längere Trainingszeit auf, was auf die Größe des verwendeten Source-Datensatzes zurückzuführen ist. In beiden Fällen zeigt sich jedoch, dass die Netzwerke unabhängig vom Einsatz von TF eine insgesamt geringe Performanz aufweisen. . . . . 32

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

4.5 Links ist das Testergebnis des Experiments COD:TF10/732/10/30 dargestellt, bei dem TF eingesetzt wurde, rechts jenes des Experiments COD:732/10/30 ohne TF. Auch in diesem weiteren Fall, bei dem die Architektur mit Linearen Layern als Hidden Layer verwendet wurde, zeigen sich unabhängig vom Einsatz von TF ebenfalls geringe Leistungswerte. . . . 32
4.6 Die dargestellten Ergebnisse zeigen Testläufe ohne Kaskadierung. Konkret sind links die Resultate für das 1DC-Netzwerk (1DC:Comp/732//30) und rechts für das COD-Netzwerk (COD:Comp/732//30) dargestellt. Es ist ersichtlich, dass eines der Modelle in einem lokalen Maximum stecken bleibt. Zudem lässt sich aus den Ergebnissen ableiten, welcher maximale Accuracy-Wert unter der gegebenen, begrenzten Datenmenge realistischerweise erreicht werden kann. Dieser Wert wird ausschließlich durch den Einsatz der vollständigen, hier beschriebenen Netzwerkarchitekturen erzielt. . . . . 33

5.1 In dieser Abbildung ist die Architektur des Regr2-Netzwerks im Detail dargestellt. Die Anordnung der Layer erfolgt von oben nach unten entlang der angegebenen Pfeilrichtung. Bei den Linear-Layern ist die jeweilige Anzahl der enthaltenen Neuronen angegeben; bei den Dropout-Layern gibt der Wert den Prozentsatz der Neuronen an, die während des Trainings zufällig deaktiviert werden. . . . . 36
5.2 In der Abbildung sind die Ergebnisse der Testläufe 1Lay:TF11/8k/10/21 und 1Lay:8k/10/11 dargestellt. Beide Modelle erreichen am Ende eine vergleichbare Leistung, unterscheiden sich jedoch hinsichtlich der Anzahl an Trainingsdurchläufen bzw. der benötigten Zeit bis zum Erreichen dieses Leistungsniveaus. . . . . 37
5.3 Bei dem vorliegenden Modell handelt es sich um den Test 1Lay:Comp/8k//8. Dieser wurde unter Verwendung einer großen Menge an Target-Daten direkt trainiert, weshalb die erzielten Ergebnisse im weiteren Verlauf als Referenz für die bestmögliche Leistung herangezogen werden. . . . 38
5.4 Abgebildet ist der Vergleich des Regr2-Netzwerks mit und ohne TF bei geringer Verfügbarkeit von Target-Daten. Die Tests entsprechen links Regr2:TF4/240/10/8 und rechts Regr2:240/10/8. Dabei zeigt sich, dass das Netzwerk ohne TF eine bessere Performanz erzielt als mit. 38
5.5 Die Abbildung zeigt den Vergleich des 1Lay-Netzwerks im Szenario mit geringer Anzahl an Target-Daten, jeweils mit und ohne TF. Die zugrunde liegenden Tests lauten: links 1Lay:TF11/240/10/21 und rechts 1Lay:240/10/11. Auffällig ist, dass in diesem Fall das Netzwerk mit TF eine bessere Leistung erzielt als ohne. . . . . 39
5.6 Dem dargestellten Plot liegt der Test 1Lay:Comp/240//8 zugrunde. Trotz der geringen Anzahl an verfügbaren Daten erzielt dieses Modell bessere Ergebnisse als alle anderen getesteten Varianten – obwohl es weder Kaskadierung noch TF verwendet. . . . . 40

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

5.7 Hier werden die Testergebnisse für das 1DC-Netzwerk unter Verwendung von Early Stopping dargestellt. Deutlich erkennbar ist, dass sich die Modellleistung durch Early Stopping nicht verbessert. Die exakten Testbezeichnungen lauten: links 1DC:LM/732/10 und rechts 1DC:ACCM/732/10 . . . . . 41
5.8 Hier sind die Testergebnisse des 1Lay-Netzwerks unter Verwendung von Early Stopping veranschaulicht. Es zeigt sich, dass die Modellperformanz mit Early Stopping schlechter ausfällt als ohne. Obwohl es in einigen Fällen zu vorzeitigen Trainingsabbrüchen kam, wurde dadurch Overfitting nicht entscheidend verhindert. Die eingesetzten Metriken erweisen sich somit als ineffektiv. Die genauen Bezeichnungen der Testläufe lauten: links 1Lay:LM/120/10 und rechts 1Lay:MAEM/120/10. 42

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.

# List of Tables

2.1 Liste aller Klassifikationstests . . . . . 16
2.2 Liste aller Regressionstests . . . . . 17

3.1 Dies stellt den Vergleich der Trainingsdauer zwischen den jeweiligen Netzwerken und deren Varianten dar. Die Zeitangaben erfolgen in Sekunden. . . . . 24

4.1 In dieser Übersicht sind alle Direct Cascade Netzwerke für die Klassifikation mit den wesentlichen internen Parametern dargestellt. "Hidden Layer" bezeichnet die Art des verwendeten versteckten Layers. "N/F" gibt die Anzahl der Neuronen (Nodes) beziehungsweise Filter im jeweiligen Layer an. Unter "Aktivierung" ist die im Hidden Layer eingesetzte Aktivierungsfunktion zu verstehen, und "Inputdim" beschreibt die Dimensionalität der Eingabedaten. . . . . 26

# Bibliography


[CK19] Radoslaw M. Cichy and Daniel Kaiser. "Deep Neural Networks as Scientific Models". In: *Trends in Cognitive Sciences* 23.4 (2019).

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

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

[KCR17] Hak Gu Kim, Yeoreum Choi, and Yong Man Ro. "Modality-bridge Transfer Learning for Medical Image Classification". In: *10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics*. 2017.

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

[LR92a] Enno Littmann and Helge Ritter. "Cascade LLM Networks". In: *Artificial Neural Networks 2*. Ed. by I. Aleksander and J. Taylor. 1992.

[LR92b] Enno Littmann and Helge Ritter. "Cascade Network Architectures". In: *Intern. Joint Conference On Neural Networks*. 1992.

[Mar19] Enrique S. Marquez. "Deep Cascade Learning". PhD thesis. Faculty of Engineering, Physical Sciences Electronics, and Computer Science, 2019.

[MHN18] Enrique S. Marquez, Jonathon S. Hare, and Mahesan Niranjan. "Deep Cascade Learning". In: *IEEE Transactions on neural networks and learning systems* 29.11 (2018).

[ML90] Enrique S. Marquez and Christian Lebiere. *The Cascade-Correlation Learning Architecture*. Tech. rep. School of Computer Science, Carnegie-Mellon University, 1990.

[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.

[RCN07] Miguel Rocha, Paulo Cortez, and José Neves. "Evolution of neural networks for classification and regression". In: *Neurocomputing* 70.16-18 (2007).

[Spe91] Donald F. Specht. "A General Regression Neural Network". In: *IEEE Transactions on neural networks* 2.6 (1991).

[TG99] Nicholas K. Treadgold and Tamáis D. Gedeon. "Exploring Constructive Cascade Networks". In: *IEEE Transactions on neural networks* 10.6 (1999).

[TS] Lisa Torrey and jude Shavlik. "Transfer Learning". In: chap. 11, pp. 242–244.

*Der hinterliegende Code, die Plots und die Textausarbeitung ist zu finden unter:* `https://github.com/Lirras/BA_EvalTF_DDCN`.