technische universität
dortmund

Bachelorarbeit

End-to-End Zeitreihenklassifikation mit binären neuronalen Netzen

Hendrik Weißenfels

Januar 2022

Gutachter:
Prof. Dr. Katharina Morik
Lukas Heppe

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS-8)
http://www-ai.cs.uni-dortmund.de/index.html

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Ziel der Arbeit . . . 2
1.2 Herausforderungen . . . 2
1.3 Aufbau der Arbeit . . . 3

2 Grundlagen **5**

2.1 Zeitreihenklassifikation . . . 5
2.1.1 Klassifikationsverfahren . . . 5
2.1.2 Univariate Zeitreihe . . . 6
2.2 Neuronale Netze . . . 7
2.2.1 Neuronen . . . 7
2.2.2 Multi Layer Perceptron . . . 9
2.2.3 Lernprozess . . . 11
2.3 Convolutional Neural Networks . . . 16
2.3.1 Faltung . . . 16
2.3.2 Pooling . . . 18
2.3.3 Fully connected Layer . . . 19
2.4 Binäre neuronale Netze . . . 19
2.4.1 Komprimierung . . . 19
2.4.2 Binäre Schicht . . . 20

3 Verwandte Arbeiten **21**

3.1 The great time series classification bake off . . . 21
3.2 Deep learning for time series classification: a review . . . 23
3.3 Vergleichsmodelle . . . 25
3.3.1 Multi Layer Perceptron . . . 26
3.3.2 Fully Convolutional Neural Network . . . 27
3.3.3 Residual Neural Network . . . 28
3.4 Verbindung . . . 29

4 Betrachtete Modelle **31**
4.1 Binarisierte Vergleichsmodelle 31
4.2 Implementierte Modelle 32
4.2.1 Binäres MLP 33
4.2.2 Binäres FCN 34
4.2.3 Binäres ResNet 35
4.2.4 Binäres DenseNet 36

5 Experimente und Evaluation **39**
5.1 Experimente 39
5.2 Datensätze 41
5.3 Ergebnisse 41
5.3.1 Genauigkeitsvergleich der Modelle 42
5.3.2 Pareto-Effizienz 44
5.3.3 Pareto-Effizienz mit Gewichtungen 48
5.3.4 Vergleich mit verwandter Arbeit 51
5.3.5 Geeignete Hyperparameter 55
5.3.6 Vergleich mit state-of-the-art Ansatz 58
5.4 Bewertung der Ergebnisse 60

6 Fazit und Ausblick **61**
6.1 Ausblick 61

A Weitere Informationen **63**

Abbildungsverzeichnis **81**

Algorithmenverzeichnis **83**

Literaturverzeichnis **91**

Erklärung **91**

Kapitel 1

Einleitung

Heutzutage werden immer mehr Geräte, wie beispielsweise Smartphones und Produktionsanlagen, mit Sensoren ausgestattet, wodurch eine Fülle an Daten bereitsteht. Oftmals handelt es sich bei diesen Daten um Zeitreihen, wie beispielsweise der Messwert eines Sensors über einen Produktionsprozess. Methoden des Maschinellen Lernens können genutzt werden, um diese Daten intelligent zu verarbeiten und somit Entscheidungen, wie z.B., inwiefern ein Produktionsteil defekt ist, zu automatisieren. Diese Aufgabe nennt sich Zeitreihenklassifikation (TSC). Jedoch sind Methoden zur TSC oftmals ressourcenhungrig, wodurch teure Server und Netzwerkkommunikation notwendig sind. In manchen Anwendungsfällen müssen Daten auf den Servern gespeichert werden, wodurch Bedenken hinsichtlich des Datenschutzes entstehen können.
Eine lokale Verarbeitung direkt auf dem Gerät ist hingegen um einiges datenschutzfreundlicher, ist unabhängig von der Internetverbindung und erspart die Ausgaben teurer Netzwerkarchitekturen. Allerdings bringt diese Methode ebenfalls Herausforderungen mit sich. Für eine zeitgemäße, performante Zeitreihenklassifikation wird heutzutage von neuronalen Netzen (NNs) Gebrauch gemacht, wobei die Tendenz immer mehr in Richtung erhöhter Komplexität und Tiefe geht [1]. Tiefe neuronale Netze (DNNs) benötigen zum aktuellen Zeitpunkt noch sehr viele Ressourcen, die auf einem kleinen Device kaum bereitgestellt werden können.
Es gibt bereits mehrere Ansätze [2–12], um neuronale Netze zu komprimieren und zugleich performanter zu machen. Ein zugleich neuartiger aber auch vielversprechender Ansatz ist der Gebrauch von binären neuronale Netzen (BNNs). Dabei handelt es sich um neuronale Netze mit binären Parametern. Im Vergleich zu herkömmlichen 32-bit float DNNs sind BNNs entscheidend effizienter und kostengünstiger. Es wird während des Prozesses erheblich weniger Speicherplatz benötigt. Zudem werden die meisten rechenaufwändigen Operationen, z.B. Matrizenmultiplikationen, durch einfache bitweise Operationen ersetzt, wodurch die Laufzeit verbessert werden kann. Trotz der binarisierten Gewichte und Zustände bei BNNs werden state-of-the-art Ergebnisse bei Bilderkennung bekannter Datensätze

wie MNIST, SVHN und CIFAR-10 erreicht [13].
Das binäre neuronale Netz wird mittels des End-to-End deep learning Ansatzes trainiert. Der End-to-End Ansatz hat sich in mehreren Studien als praktikabel herausgestellt [14] und erspart dabei einige aufwendige Prozessschritte, die zeitaufwändig manuell gefertigt werden müssten.

1.1 Ziel der Arbeit

Das Ziel der Arbeit ist es, binäre neuronale Netze hinsichtlich ihrer Tauglichkeit zur Zeitreihenklassifikation zu prüfen. Die Feststellung einer möglichen Reduzierung an Speicherbedarf oder vielmehr einer Verringerung der Laufzeit würden die Entwicklung von lokal verarbeitenden Zeitreihenklassifikation beträchtlich vereinfachen oder womöglich erst realisierbar machen. Das Sammeln von Erkenntnissen über geeignete Architekturen, Aktivierungsfunktionen und andere Anpassungen zur Optimierung der Effizienz der binären neuronalen Netze sind ebenfalls erstrebenswert und Teil der Arbeit. Zudem wird im späteren Verlauf der Arbeit die Performance der jeweiligen binarisierten Architekturen mit der Performance der zugehörigen full-precision Architektur verglichen [15].

1.2 Herausforderungen

Zum Trainieren des binären neuronalen Netzes werden die Gewichtungen und die Aktivierungen binarisiert. Mit der sehr geringen 1-Bit Auflösung lässt sich einerseits sehr einfach und schnell rechnen, andererseits kommt es, verglichen mit der typischen 32-Bit Auflösung, zu einem Informationsverlust. Daher ist die Wahl der richtigen Methodik zur Binarisierung entscheidend. Zur Auswahl stehen mehrere geeignete Verfahren aus der open-source Deep Learning Bibliothek Larq [16]. Die binarisierten Parameter könnten nicht in jeder Sachlage ausreichend sein. Gegebenenfalls muss auch an anderen Stellen, wie der Eingabe eines Netzwerks, die volle Auflösung der Parameter bestehen bleiben, um die Genauigkeit des neuronalen Netzes zu gewährleisten. Zusammen mit den anfallenden Kosten für die eventuell entstehende ständige Umwandlung dieser Parameter kann es insgesamt auch zu einer verschlechterten Effizienz kommen.
Der durch die Binarisierung resultierende Informationsverlust kann durch die Wahl einer geeigneten Architektur aufgefangen werden. Die Auswahl der Architekturen ist durch die Kombinationsmöglichkeiten von Hyperparametern schier unendlich, weshalb vorab eine gut durchdachte Auswahl getroffen werden muss. Als Grundlage für diese Arbeit werden einige Architekturen aus der verwandten Arbeit von Hassan Fawaz [15] herangezogen, da diese bereits auf ihre Tauglichkeit zur Zeitreihenklassifikation geprüft wurden und somit auch ein späterer Vergleich mit den Ergebnissen der Literatur ermöglicht wird. Es lassen sich bereits folgende Angaben zu einer vorläufigen Auswahl von geeigneten Architekturen

machen. Die Architekturen Multi Layer Perceptron (MLP), Fully Convolutional Neural Networks (FCNs) und Residual Neural Network (ResNet) sind besonders interessant, da sie sich bei einer regulären Implementierung für Zeitreihenklassifikation bei univariat Zeitfolgen als performant erwiesen haben [15]. In der Arbeit von Bethge [17] wurde die neue BNN-Architektur BinaryDenseNet vorgeschlagen, die bei der dort durchgeführten Bildklassifikation alle existierenden 1-Bit-CNNs übertrifft.
Resultierend ergibt sich die folgende vorläufige Auswahl: MLP, FCN, ResNet und DenseNet.

1.3 Aufbau der Arbeit

Einleitung In der Einleitung werden Zeitreihenklassifikation und binäre neuronale Netze in einem kleinen Umfang vorgestellt. Es folgen verschiedene Beispiele zu der lokalen Verarbeitung von Zeitreihenklassifikation, wobei zudem die anfallenden Vor- und Nachteile der lokalen Verarbeitung kurz erläutert werden. Als Problemlösung bezüglich der hohen Ressourcenanforderung werden daraufhin binäre neuronale Netze in Betracht gezogen.

Grundlagen Dieses Kapitel gibt einen Überblick in die Grundlagen der Hauptkomponenten dieser Arbeit:

- **Zeitreihenklassifikation:** Es wird in einem kurzen Abschnitt ein Einblick in die Funktionsweise der Zeitreihenklassifikation geboten. Dabei wird zu einigen verbundenen Arbeiten Bezug genommen.
- **Neuronale Netze:** Daraufhin folgen Grundlagen zu der Materie von NNs, in denen bereits auf die Funktionalität von MLPs und FCNs eingegangen wird. Neben wichtigen Bestandteilen eines NNs, wie der Pooling-Operation oder verschiedener Aktivierungsfunktionen, wird der essenzielle Lernporzess ausführlich erläutert.
- **Binäre neuronale Netze:** Zuletzt werden die Besonderheiten von BNNs aufgeführt, insbesondere die gewählte Methode zur Binarisierung spielt eine wichtige Rolle. Außerdem werden die anfallenden Vor- und Nachteile von BNNs thematisiert.

Verwandte Arbeiten Dieses Kapitel stellt die zwei wichtigsten verwandten Arbeiten dieser Arbeit vor. Neben dem Review samt experimentelle Evaluation von Bagnall [18] wird insbesondere auf das Review von Hassan Fawaz [15] eingegangen, da es zu einem großen Teil die Grundlage dieser Arbeit bildet. Es werden zugleich klassische state-of-the-art Ansätze, wie COTE [19] und HIVE-COTE [20], aber auch neue Ansätze, wie der Gebrauch von DNNs vorgestellt. Die in dem Review von Hassan Fawaz [15] angefertigten DNNs werden ausführlich erläutert. Es folgt eine Argumentation zur Verbindung der verwandten Arbeiten mit dieser Arbeit.

Betrachtete Modelle In diesem Kapitel werden die verschiedenen ausgewählten Modelle vorgestellt. Die Modelle beinhalten verschiedene Variationen von Architekturen, sowie verschiedene Quantisierungsfunktion zur Binarisierung. Die Architekturen und deren zugehörige Hyperparameter werden jeweils kurz erläutert.

Experimente und Evaluation Im Anschluss werden Experimente an den Modellen durchgeführt. Die verschiedenen Modelle werden auf den Benchmarkdatensätzen trainiert. Die resultierenden Ergebnisse werden daraufhin evaluiert und mit der Performance und den Kosten der full-precision Architekturen verglichen [15], dabei wird insbesondere auf Speicherbedarf, Genauigkeit und Laufzeit geachtet. Zuletzt wird eine Einschätzung gegeben, ob sich binäre neuronale Netze zur Zeitreihenklassifikation eignen.

Fazit und Ausblick In dem letzten Abschnitt wird aus den zuvor erworbenen Erkenntnissen die Quintessenz hervorgebracht und zu einem Fazit komprimiert. Des Weiteren werden gegebenenfalls Hypothesen aufgestellt und Vorschläge zur Weiterentwicklung des Themas formuliert.

Kapitel 2

Grundlagen

2.1 Zeitreihenklassifikation

Zeitreihenklassifikation (oft auch mit TSC abgekürzt, von "Time Series Classification") stellt eine große Herausforderung auf dem Gebiet des Data-Minigs dar. Die Problematik der Zeitreihenklassifikation gewinnt immer mehr an Bedeutung, da die Anwendungsbereiche von TSCs immer häufiger den Weg in unseren Alltag finden. Dabei reichen die Anwendungsbereiche von Human Activity Recognition, zu Deutsch Aktivitätserkennung bei Menschen [21, 22], über Spracherkennung, bis hin zur Netzsicherheit [23].
Im Allgemeinen unterliegen TSCs dem klassischen Klassifikationsverfahren, das im Abschnitt 2.1.1 näher erläutert wird. Somit können TSCs wie traditionelle Klassifizierungsprobleme betrachtet werden, bei denen allerdings die Elemente der Daten einer gewissen Reihenfolge unterliegen. Obwohl die Bezeichnung Zeitreihenklassifikation den Anschein erweckt, dass die Reihenfolge durch die Zeit bestimmt wird, ist das Motiv der Reihenfolge unwesentlich. Wesentlich ist lediglich der mögliche Bezug der Elemente auf deren jeweilige Position in der Reihenfolge. Neben der anwachsenden Bedeutsamkeit der Anwendungsbereiche ist das, im Jahr 2015 veröffentlichte und frei zugängliche UCR/UEA Archiv [24] ein bedeutsamer Auslöser zur rapiden Entwicklung der Forschung rund um das Themengebiet der Zeitreihenklassifikationen. Dieses Archiv ermöglicht es TSC Algorithmen durch Benchmarks miteinander zu vergleichen. Im Kapitel 5.2 wird umfassender auf das Archiv und die in dieser Arbeit verwendeten Datensätze eingegangen.

2.1.1 Klassifikationsverfahren

Das Prinzip der Klassifizierung ist ein fundamentales Verfahren, das auch in der Natur beobachtet werden kann. Es dient unter anderem der Unterscheidung von Wichtigem und Unwichtigem. Ein einfaches Neuron in einem Nervensystem dient bereits der simplen Klassifizierung in der Bedeutsamkeit eines Reizes, durch Weiterleitung oder Ignorieren des Reizes. Im Kapitel 2.2.1 wird weiter auf dieses Teilgebiet eingegangen.

Das folgende Zitat spiegelt den Grundgedanken einer Klassifikation wider. “Das Ziel einer Klassifikation oder Clusteranalyse besteht darin, Objekte in Klassen oder Gruppen zusammenzufassen, so dass zwischen den Elementen derselben Klasse größtmögliche Ähnlichkeit und zwischen den Elementen unterschiedlicher Klassen größtmögliche Verschiedenheit erreicht wird.“ [25] Bei der Umsetzung dieser Ziele kann mit sogenannten Klassifikatoren Aussagen über die Klassifizierung getroffen werden. Ein Klassifikator ist eine Abbildung von den möglichen Eingaben auf eine Wahrscheinlichkeitsverteilung über die vorgegebenen Klassen. [18]

2.1.2 Univariate Zeitreihe

Eine Zeitreihe, bei der lediglich eine einzige Variable involviert ist, nennt sich univariate Zeitreihe. Dementsprechend variiert bei einer univariate Zeitreihe ein eindimensionaler Wert über einen Zeitraum. Neben univariaten Zeitreihen gibt es auch multivariate Zeitreihen mit M Dimensionen und somit M verschiedenen Variablen, die sich über den Zeitraum ändern. In dieser Arbeit wird jedoch der Fokus auf die univariaten Zeitreihen gelegt. Im folgenden Abschnitt wird eine univariate Zeitreihe definiert.

Definition Eine univariate Zeitreihe $X = [x_1, x_2, ..., x_n]$ besteht aus $n \in \mathbb{N}$ Werten die in einer festen Reihenfolge angeordnet sind.
Ein Datensatz $T = [(X_1, Y_1), (X_2, Y_2), ..., (X_N, Y_N)]$ ist eine Gruppe von Paaren (X_i, Y_i) mit $i \in [1, N]$. Jeder Zeitreihe X_i wird ein diskreter One-Hot-Vektor Y_i mit Länge K zugeteilt. Jedes der K Elemente stellt dabei eine Klasse des Datensatzes dar. Während alle anderen Elemente den Wert 0 haben, wird eines der Elemente mit einer 1 hervorgehoben, dessen zugehörige Klasse dann die Klasse der Zeitreihe X_i ist.

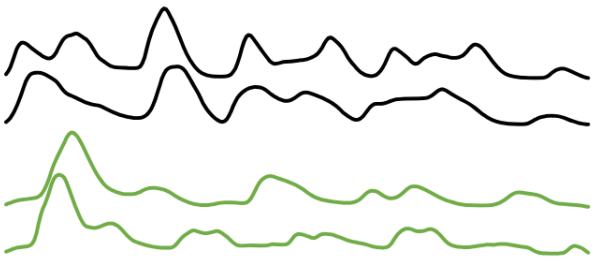

Abbildung 2.1: Abbildung von vier univariaten Zeitreihen aus der verwandten Arbeit [18].

Im Kapitel 3.1 “The great time series classification bake off“ wird ein strukturierter Überblick über das breite Spektrum an verschiedenen Verfahren zur TSC vorgestellt. Für eine zeitgemäße, performante TSC wird heutzutage immer häufiger von neuronalen Netzen Gebrauch gemacht. Auf die Grundlagen zum Themengebiet der Neuronalen Netze wird in dem folgenden Kapitel 2.2 eingegangen.

2.2 Neuronale Netze

Beim maschinellen Lernen handelt es sich um einen Teilbereich der Informatik. Häufig wird von künstlichen neuronalen Netzen (KNNs) Gebrauch gemacht, um dem Computerprogamm das sogenannte "Lernen" zu ermöglichen. Der auch in diesem Kontext etablierte Begriff "Lernen" ist auf MITCHELL [26] zurückzuführen. In diesem Kapitel wird ein Einblick in die Grundlagen neuronaler Netze gegeben.

2.2.1 Neuronen

Das Neuron bildet das Elementarteilchen eines neuronales Netzes, es fungiert als einzelne Recheneinheit, als eine sogenannte Unit. Wie so oft lag die Inspiration für dieses künstliche Neuron in der Natur. Als biologisches Vorbild des künstlichen Neurons dient die Nervenzelle bioloischer Organsimen [27]. Diese spezialisierte Zelle nimmt Reize auf, verarbeitet sie und leitet sie daraufhin weiter.
Das erste künstliche Neuron wurde 1943 von MCCULLOCH und PITTS vorgestellt [28]. Das sogenannte McCulloch-Pitts-Neuron arbeitet mit einem simplen Schwellenwert, der durch die summierte Eingabe von n binären Signalen entweder über- oder unterschritten wird und somit ein alleiniges binäres Signal weitergeleitet wird. Dieses schlichte Neuron ist bereits fähig einfache boolesche Funktionen wie NOT, AND und OR zu simulieren [28]. Aufbauend auf das McCulloch-Pitts-Neuron hat ROSENBLATT im Jahr 1958 das Perzeptron entwickelt [29]. Es wird ebenfalls eine Eingabe übergeben, die durch Verarbeitung zu einer Ausgabe führt. In diesem Fall wird dem Neuron jedoch ein Vektor x (2.1)

$$x = (x_1, x_2, ..., x_n) \tag{2.1}$$

mit $n \in \mathbb{N}$ Elementen als Eingabe übergeben. Diese Eingabe wird mit Hilfe eines Vektors w mit der gleichen Anzahl $n \in \mathbb{N}$ Gewichte verarbeitet. Dabei wird jedem Element des Eingabevektors ein spezifisches Gewicht zugeteilt [1]. Die Ausgabe des Neurons ist das Skalarprodukt des Eingebevektors und des Gewichtvektors. Daher repräsentiert das Gewicht die Relevanz des jeweiligen Eingabevektors auf die Ausgabe.
Die Ausgabe y eines Neurons lässt sich durch Gleichung (2.2) beschreiben.

$$y = \sigma(\sum_{i=0}^{n-1}(x_i \cdot w_i) + b) \tag{2.2}$$

Auf das Skalarprodukt des Eingangsvektors x und der Gewichte w wird ein Bias b addiert. Durch den Bias kann die Eingabe zusätzlich modifiziert und angepasst werden [30, 31]. Er kann als zusätzliche Gewichtung des gesamten Neurons interpretiert werden. Ein beispielsweise besonders negativer Bias würde die Überschreitung des Schwellenwertes beträchtlich erschweren. Schließlich wird die Aktivierungsfunktion σ auf den gesamten

Term g angewendet. Bei dem einfachen Perzeptron handelt es sich hierbei um die *Heaviside-Stufenfunktion* (siehe Formel 2.3), die bei Überschreitung des Schwellenwertes t den Wert 1 zurück gibt, ansonsten 0.

$$\sigma(g)_{HS} = \begin{cases} 1 & \text{falls } g > t \\ 0 & \text{sonst} \end{cases} \tag{2.3}$$

Die *Heaviside-Stufenfunktion* ist eine lineare Aktivierungsfunktion. Durch dessen Gebrauch kann das einfache Perzeptron lineare Probleme lösen [32]. Um nicht lineare Probleme, wie die XOR-Funktion, zu approximieren, muss Gebrauch von komplexeren, nicht linearen Aktivierungsfunktionen gemacht werden.

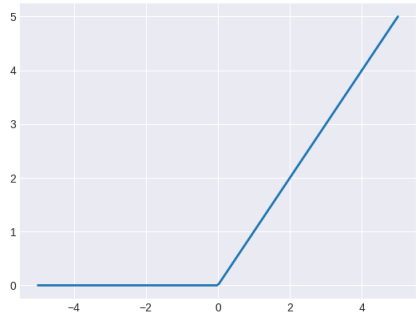

Abbildung 2.2: ReLU-Funktion

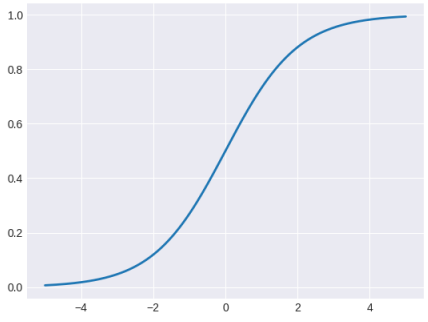

Abbildung 2.3: Sigmoid-Funktion

$$\sigma(g)_{ReLu} = max(0, g) \tag{2.4}$$

$$\sigma(g)_{Sigmoid} = \frac{1}{1 + e^{-g}} \tag{2.5}$$

Die ReLU-Aktivierungsfunktion (siehe Formel 2.4) ist für Werte größer als Null linear, daher werden viele positive Eigenschaften linearer Aktivierungsfunktionen bewahrt, obgleich es sich um eine nichtlineare Funktion handelt, da negative Werte als Null ausgegeben werden. Häufig wird mit dem Begriff *"hinge function"* auf die ReLU-Funktion verwiesen, da sie sich auf der positiven Hälfte linear und auf der negativen Hälfte nicht linear verhält [33].

Die Sigmoid-Aktivierungsfunktion hat einen S-förmigen Verlauf (s.a. 2.5). Sie ist differenzierbar, beschränkt und besitzt für $g = 0$ genau einen Wendepunkt [34]. Durch die negierte Variable g im Exponenten des Nenners geht die Ausgabe der Sigmoid-Aktivierungsfunktion gegen 0 für $\lim_{g \to -\infty}$ und gegen 1 für $\lim_{g \to \infty}$.

In dieser Arbeit wird neben der ReLU-Aktivierungsfunktion auch häufig die Softmax-Aktivierungsfunktion benutzt.

$$softmax(g)_i = \frac{exp(g_i)}{\sum_j exp(g_j))} \tag{2.6}$$

In der Gleichung 2.6 wird der Softmax-Aktivierungsfunktion ein Vektor g mit $n \in \mathbb{N}$ Elementen für n mögliche Klassifikationen übergeben [35]. Im Zähler wird der Exponent des i-ten Elements g_i des Eingangsverktors gebildet. Im Nenner wird eine Normalisierung durchgeführt, die für eine gültige Wahrscheinlichkeitsverteilung zwischen 0 und 1 sorgt.

Es ist keine Abbildung für die Softmax-Aktivierungsfunktion beigelegt, da es sich um eine multivariable Funktion handelt. Die Grafik müsste für einen Eingabevektor mit $n \in \mathbb{N}$ Elementen n Dimensionen darstellen. Für den Menschen ist es sehr schwer mehr als drei Dimensionen visualisiert zu betrachten. Die Softmax-Aktivierungsfunktion sprengt die Möglichkeiten eines einfachen Perzeptrons, sie findet jedoch im Themengebiet des nächsten Kapitels Anwendung. Im Verlauf der Arbeit wird zusätzlich noch auf binäre Aktivierungsfunktionen, wie SteSign eingegangen.

2.2.2 Multi Layer Perceptron

Um komplexere Probleme modellieren zu können, werden mehrere Perzeptrons zu einem neuronalen Netwerk vereint. Sie werden zu Schichten (s. g. Layers) gruppiert, die wiederum miteinander verbunden werden. Der Begriff *Tiefe* wird häufig zur groben Beschreibung der Anzahl an Schichten eines neuronalen Netzes verwendet [1]. Zwei Layers reichen bereits aus, um komplexe Probleme zu lösen, doch die Tendenz geht immer mehr in Richtung tieferer neuronaler Netze [1].
Ein Multi Layer Perceptron (MLP) besteht aus einer Eingabeschicht, mindestens einer versteckten Schicht und einer Ausgabeschicht. Jede dieser Schichten besteht aus mehreren Neuronen. Die Anzahl an Neuronen pro Layer und die Anzahl an versteckten Schichten, sogenannter "Hidden Layer", sind nicht vorgegeben. Durch die Verbindungen zwischen den Layern beeinflussen die Ausgaben eines Layeres die Eingaben des nachfolgenden Layers. Die Eingabe des ersten Layers bleibt hingegen unbeeinflusst, da es den Eingabeverktor x verarbeitet. Bei einem MLP handelt es sich um ein sogenanntes Feedforward Netzwerk [1]. Die Informationen fließen nur in eine Richtung, das heißt, dass die Ausgaben der Neuronen des $(l-1)$-ten Schicht ausschließlich zur Eingabe für die Neuronen in der l-ten Schicht weitergeleitet werden. Bei der Ausgabe der letzten Schicht handelt es sich um die Ausgabe des gesamten neuronalen Netzes [1,36]. In Abbildung 2.4 wird ein MLP grafisch analysiert.

Im Grunde kann man bei einem Feedforward-Netzwerk von einer Abbildung (s. g. Mapping) $f(x) = y$ sprechen. Der Eingabevektor x wird durch die Funktion f auf die Ausgabe y abgebildet.

In Kapitel 2.2.1 wurde Gleichung (2.2) eingeführt. Sie berechnet die Ausgabe für ein einzelnes Neuron. Diese Gleichung kann zur Berechnung der Ausgabe von Layern weiter-

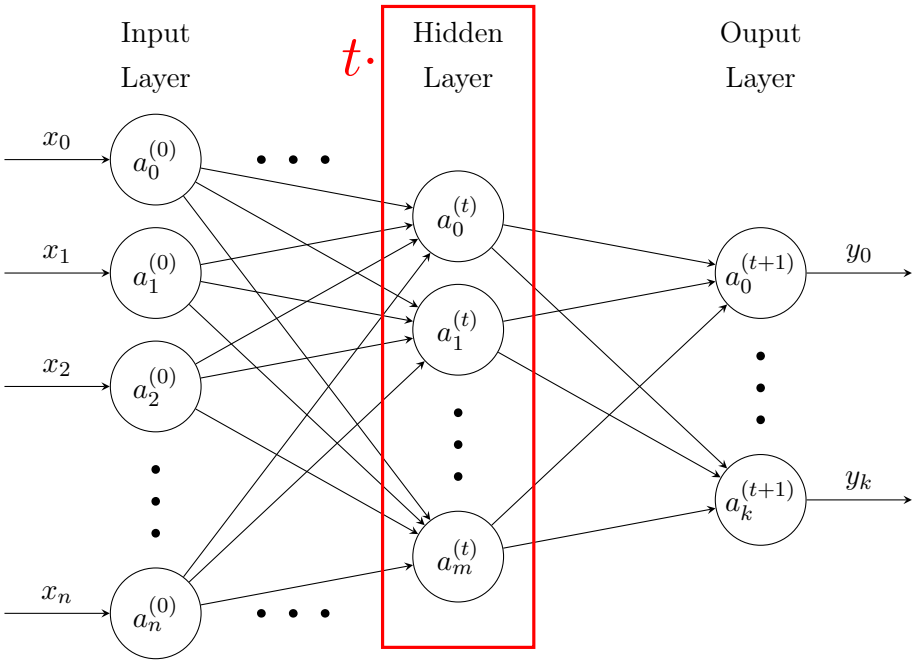

Abbildung 2.4: Beispielgraph für ein MLP. Das Input Layer mit $n \in \mathbb{N}$ Neuronen verarbeitet den Eingabevektor $x = (x_1, x_2, ..., x_n)$ und gibt seine Ausgabe an das erste Hidden Layer weiter. Das neuronale Netz hat t Hidden Layers, wobei $t \geq 1 \in \mathbb{N}$ gelten muss. Das t-te Hidden Layer mit $m \in \mathbb{N}$ Neuronen erhält die Ausgabe des vorherigen Layers als Eingabe. Nach der Verarbeitung wird die Ausgabe an die letzte Schicht weitergeleitet. Die Anzahl an Neuronen $k \in \mathbb{N}$ im Output Layer bestimmt die Anzahl der Klassen des neuronalen Netzes.

entwickelt werden. Hierfür werden sämtliche Aktivierungen (Ausgaben der Aktivierungsfunktionen) der 0-ten Schicht in den Vektor $a^{(0)} = (a_0^{(0)}, a_1^{(0)}, ..., a_n^{(0)})^T$ übertragen. Die Gewichte der 1-ten Schicht werden in eine sogenannte Gewichtsmatrix $W^{(1)}$ eingespannt. Auf das Matrix-Vektor-Produkt von Gewichtsmatrix $W^{(1)}$ und Vektor $a^{(0)}$ wird der Vektor $b^{(1)} = (b_0^{(1)}, b_1^{(1)}, ..., b_k^{(1)})^T$ addiert. Der sogenannte Biasvektor $b^{(1)}$ beinhaltet alle n Bias der n Neuronen in der 1-ten Schicht. Auf das Zwischenergebnis wird abschließend die Aktivierungsfunktion σ angewendet, um den Aktivierungsverktor $a^{(1)} = (a_1^{(1)}, a_1^{(1)}, ..., a_k^{(1)})^T$ zu erhalten. Die Gleichung 2.7 zeigt die soeben beschriebene Informationsweitergabe zwischen der 0-ten und der 1-ten Ebene.

$$\underbrace{\begin{pmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_k^{(1)} \end{pmatrix} = \sigma\left(\begin{pmatrix} w_{00} & w_{01} & \cdots & w_{0n} \\ w_{10} & w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k0} & w_{k1} & \cdots & w_{kn} \end{pmatrix} \begin{pmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_k^{(1)} \end{pmatrix}\right)}, \text{ mit } n, k \in \mathbb{N} \tag{2.7}$$

$$a^{(1)} = \sigma(W^{(1)} a^{(0)} + b^{(1)}) \tag{2.8}$$

Jede Reihe der Gewichtsmatrix entspricht den Verbindungen zwischen aller Neuronen der 0-ten Schicht und einem spezifischen Neuron in der 1-ten Schicht. Daher kann jedes Element $a_i^{(1)}$ mit $0 \leq i \leq n$ des resultierenden Vektors $a^{(1)} = (a_1^{(1)}, a_1^{(1)}, ..., a_n^{(1)})^T$ durch die ursprüngliche Gleichung (2.2) beschrieben werden.
Die Gleichung 2.7 wird meist in einer kompakteren Form referenziert (siehe Gleichung 2.8).

Die Weitergabe der Aktivierungen zwischen den Schichten kann nun durch folgende Gleichung (2.9) für alle $l + 1$ Schichten ausgedrückt werden.

$$a^{(l)} = \begin{cases} \sigma(W^{(0)}x + b^{(0)}) & \text{falls } l = 0 \\ \sigma(W^{(l)}a^{(l-1)} + b^{(l)}) & \text{sonst} \end{cases} \tag{2.9}$$

Es handelt sich hierbei um eine Fallunterscheidung in der Gebrauch von der Gleichung 2.8 gemacht wird und diese weiterentwickelt wird. Falls es sich nicht um die Eingabeschicht handelt, wird die Gleichung 2.8 mit l anstatt 1 und $l-1$ anstatt 0 angewandt. Falls es sich jedoch um die Eingabeschicht handelt, demgemäß $l = 0$ gilt, existiert kein Layer $l - 1$. In diesem Fall kann als Eingabe kein vorheriger Aktivierungsvektor $a^{(-1)}$ übergeben werden, stattdessen wird hier der Eingabeverktor x eingegeben.

2.2.3 Lernprozess

Die Aufgabe eines neuronalen Netzes (NN) ist es, gewisse Zielwerte möglichst genau zu approximieren. Doch kein NN kann das auf Anhieb. Im Gegenteil, anfangs wirken die Approximationen des NN komplett wahllos, wobei sehr viele Fehler gemacht werden. Doch aus genau diesen Fehlern kann das NN lernen, um sich selbst zu verbessern. Das ist das Grundprinzip des Lernprozesses.

Fehlerfunktion

Die Fehlerfunktion C gibt an, wie gut oder schlecht das NN eine Klassifizierung durchgeführt hat. Sie berechnet den Fehler (auch oft Loss-Wert genannt), der die 'Distanz' zwischen dem berechneten und dem erwarteten Ergebnis bei einer Anwendung des NN auf Trainingsdaten angibt.

Eine weit verbreitete Variante der Fehlerfunktion zur Regression ist die Mean-Squared Error-Funktion (Abk. MSE) [37].

$$MSE = \frac{1}{n} \sum_{i=0}^{n-1} (t_i - y_i)^2 \tag{2.10}$$

Es werden die quadrierten Differenzen zwischen dem gewünschten Zielwert t_i und dem tatsächlich berechneten Wert a_i für alle n betrachteten Trainingsdaten mit $i < n \in \mathbb{N}$ aufsummiert. Nach der Normalisierung gibt die Gleichung 2.10 den mittleren quadrierten Fehler aller n Trainingsdaten an.

In dieser Arbeit wird die Maximum-Likelihood-Methode zur Messung der Entfernung der Wahrscheinlichkeiten bei der Klassifizierung genutzt. Die Methode greift zur Berechnung auf die Kreuzentropie zurück, die häufig für Klassifikationsprobleme angewendet wird, wenn die Ausgaben des NN als Wahrscheinlichkeiten der Zugehörigkeit zu einer Klasse interpretiert werden.
In diesem Fall handelt es sich um die bedingte Maximum-Likelihood-Methode:

$$\begin{aligned} \theta_{ML} &= argmax_\theta P(Y|X;\theta) \\ &= argmax_\theta \prod_{i=1}^{m} P(y^{(i)}|x^{(i)};\theta) \end{aligned}$$

Der Funktionsteil $P(Y|X;\theta)$ gibt die Wahrscheinlichkeit zur Vorhersage der Ausgabe y bei der Eingabe x für die Parameter θ an. X repräsentiert die gesamten Inputs und Y die gesamten Ziele. Diese Wahrscheinlichkeit ist äquivalent zum Produkt aller m Wahrscheinlichkeiten der m Inputvektoren von X. Das Produkt über so viele Wahrscheinlichkeiten kann verschiedene Probleme mit sich bringen, wie beispielsweise die Tendenz zur numerischen Unterströmungen. Daher wird der Logarithmus auf die Wahrscheinlichkeit angewendet und somit das Produkt in eine Summe umgewandelt.

$$\theta_{ML} = argmax_\theta \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)};\theta)$$

Optimierung

Das Trainieren eines neuronalen Netzes bedeutet das Lösen von Optimierungsproblemen [1]. Ein Optimierungsproblem besteht aus der Minimierung der Kostenfunktion C (auch Fehlerfunktion genannt) durch Anpassung der Parameter [1]. Die einzelne Anpassung wird als Optimierungsschritt bezeichnet.

Gradientenabstieg Zum Lösen dieser Optimierungsprobleme wird häufig das Gradientenabstiegsverfahren (Gradient Descent) als Optimierungsalgorithmus verwendet. Das Vorgehen des Algorithmus kann folgendermaßen veranschaulicht werden. Die Kostenfunktion C wird als unebene Fläche dargestellt, auf die ein Ball gelegt wird, um den niedrigsten Punkt, also das Minimum der Funktion, zu finden. Der Ball muss nun herausfinden in welche Richtung er rollen muss, um die Ausgabe der Kostenfunktion C möglichst schnell zu reduzieren. Dabei handelt es sich um die Gegenrichtung der stärksten Steigung, bzw. dem negierten Gradienten der Kostenfunktion C.

$$-\nabla C(w_j) = -\begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_j} \end{pmatrix} \quad \text{, mit } j \in \mathbb{N} \tag{2.11}$$

Bei dem Gradienten der Kostenfunktion $C(w_j)$ handelt es sich um die partielle Ableitung der Kostenfunktion C nach den Gewichten w_j. Nachdem berechnet werden kann, in welche Richtung sich die stärkste Steigung befindet, muss bestimmt werden, wie weit der Ball in diese Richtung rollen soll, bevor er stehen bleibt und die Berechnung der Richtung für die stärkste Steigung erneut durchführt. Hierbei handelt es sich um die Schrittweite, der sogenannten Lernrate des neuronalen Netzes. Die Lernrate regelt den Einfluss, den solch ein Optimierungsschritt auf die Parameter haben soll. Ein zu kleiner Wert resultiert in einer geringen Änderung pro Schritt, sodass der Gradientenabstieg sehr viele Schritte benötigt, somit auch viel Zeit, um das Minimum zu finden. Der Gegensatz wäre ein zu großer Wert, der im schlimmsten Fall sogar dafür sorgen könnte, dass das Minimum niemals erreicht wird [31, 38]. In diesem Fall würde der Ball immer eine so weite Distanz rollen, dass er bei der nächsten Überprüfung der Steigung bereits auf die andere Seite des Minimums gelangt ist. Somit würde der Ball immer wieder über das Minimum rollen, ohne es jemals zu erreichen.

Daraus ergibt sich folgendes Update für das Gewicht w_j in Schicht k.

$$\widetilde{w_j}^k = w_j^k - \eta \frac{\partial C}{\partial w_j^k} \tag{2.12}$$

Die negierte Änderung am Gewicht w_j^k ergibt sich aus der Multiplikation des Gradienten von w_j^k mit der Lernrate η. Um das aktualisierte Gewicht $\widetilde{w_j}^k$ zu erhalten, muss die negierte Änderung von dem ursprünglichen Gewicht w_j^k subtrahiert werden. Bei dem Gradientenabstiegsverfahren, auch *Gradient Descent* genannt, wird erst nach dem Durchlauf aller Trainingsbeispiele und Berechnung des Mittles der Gradienten der Optimierungsschritt durchgeführt.

Der Stochastic Gradient Descent, kurz *SDG*, verfolgt einen anderen Ansatz, da er nach jedem Trainingsbeispiel einen Optimierungschritt ausführt. Dieses Vorgehen ist um einiges rechenintensiver [1, 39], jedoch werden pro Durchlauf aller Trainingsbeispiele genauere Optimierungen vorgenommen. Um einen Kompromiss beider Varianten einzugehen, werden sogenannte Minibatches aus der Trainingsmenge gebildet. Ein Minibatch ist eine Charge an Daten, die relativ zur gesamten Trainingsmenge klein ausfällt. Erst nach dem Durchlaufen der Charge an Daten wird ein Optimierungsschritt durchgeführt. Dadurch wird der Optimierungsschritt seltener als bei *SDG* und häufiger als bei *Gradient Descent* ausgeführt, sodass es zu einem Kompromiss aus Rechenaufwand und Genauigkeit kommt.

Dieser Prozess wird in der Regel für eine gewisse Anzahl e sogenannter Epochen wiederholt. In der Praxis werden oftmals optimierte Varianten vom SGD verwendet. Unter den bekannten Varianten befinden sich beispielsweise die Adadelta-Optimierung [40] oder auch die Adam-Optimierung [40], von der in dieser Arbeit Gebrauch gemacht wird.

Backpropagation Um die optimalen Parameter durch das Optimierungsverfahren zu bestimmen, benötigt es vorerst eine möglichst effiziente Berechnung des Gradienten der Kostenfunktion. Der Backpropagation Algorithmus von Rumelhart [RHW86] erfüllt dieses Kriterium. Er nutzt den Gradientenabstieg schichtweise für alle Neuronen vom Output Layer bis hin zum Input Layer, um den errechneten Fehler zu reduzieren [1, 31, 41].

Die Kostenfunktion für ein Trainingsbeispiel kann mit Hilfe der MSE-Funktion (siehe Gleichung 2.10) folgendermaßen gebildet werden.

$$C_0 = \frac{1}{j} \sum_{j=0}^{n^{[l]}-1} (t_j - a_j^{(L)})^2 \tag{2.13}$$

Somit ergibt sich die Kostenfunktion C_0 aus der normierten Summe der quadrieren Differenz zwischen den erwarteten Werten t_j und den tatsächlichen Aktivierungen $a_j^{(L)}$ aller $n^{(L)}$ Neuronen des letzten Layers L.

Die Gleichung für die Aktivierung $a_j^{(L)}$ kann durch Betrachtung eines einzelnen Elementes des Vektors $a_i^{(1)}$ aus der Gleichung (2.7) abgelesen werden.

$$a_j^{(L)} = \sigma(z_j^{(L)}) = \sigma\left(\sum_{i=0}^{n^{(L-1)}-1} (w_{ji}^{(L)} a_i^{(L-1)}) + b_j^{(L)}\right) \tag{2.14}$$

$$\iff z_j^{(L)} = \sum_{i=0}^{n^{(L-1)}-1} (w_{ji}^{(L)} a_i^{(L-1)}) + b_j^{(L)} \tag{2.15}$$

Zur Vereinfachung kommender Gleichungen wird die Variable $z_j^{(L)}$ als Stellvertreter des Neuroneninputs eingeführt (siehe Gleichung 2.15).

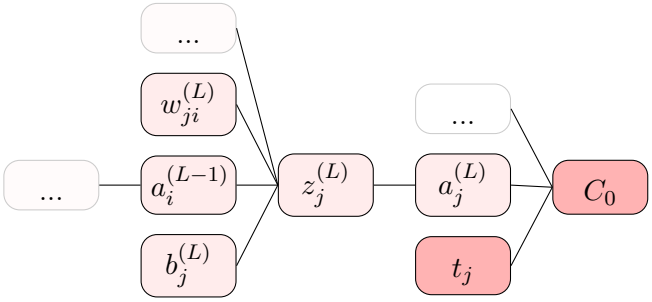

Abbildung 2.5: Dieser Baum visualisiert die Abhängigkeiten der einzelnen Variablen. In dem Beispiel wird die Abhängigkeit nur für ein fixiertes i und j vollständig modelliert, da es sonst zu unübersichtlich würde. Die Kostenfunktion C_0 subtrahiert den gewünschten Zielwert t_j von dem erhaltenden Output $a_j^{(L)}$. Der Output $a_j^{(L)}$ ist abhängig vom Input $z_j^{(L)}$ und dieser ist schlussendlich abhängig vom eigentlichen Gewicht $w_{ji}^{(L)}$. Zusätzlich ist der Input auch abhängig vom Bias $b_j^{(L)}$, sowie dem Output des vorherigen Layers $a_i^{(L-1)}$. Diese zuletzt genannte Abhängigkeit sorgt für die Rekursivität der Backpropagation. Darauf wird im weiteren Verlauf der Arbeit weiter eingegangen.

Um nun die Abhängigkeit der Veränderung der Kostenfunktion ∂C_0 bezüglich der Veränderung des Gewichtes $\partial w^{(L)}$ zu bestimmen wird die folgende partielle Ableitung (2.16) gebildet. In den jeweiligen Teilen der partielle Ableitung kann von links nach rechts erneut die in Abbildung (2.5) durchlaufenen Abhängigkeiten abgelesen werden.

$$\frac{\partial C_0}{\partial w_{ji}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{ji}^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial C_0}{\partial a_j^{(L)}} \tag{2.16}$$

Für die Abhängigkeit der Veränderung der Kostenfunktion ∂C_0 hinsichtlich der Veränderung des Bias $\partial b^{(L)}$ ergibt sich durch die ähnlichen Abhängigkeiten ein ähnlicher Term (2.17).

$$\frac{\partial C_0}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial C_0}{\partial a_j^{(L)}} \tag{2.17}$$

Die Formel zur Abhängigkeit der Veränderung der Kostenfunktion ∂C_0 bezüglich der Veränderung des Outputs des vorherigen Layers $a_i^{(L-1)}$ hat jedoch eine andere Gestalt. Die ergibt sich dadruch, dass der Output des vorherigen Layers $a_i^{(L-1)}$ nicht nur das j-ten Neuron beeinflusst, sondern sämtliche $n^{(L)}$ Neuronen des Layers l, da sie mit ihm verbunden sind.

$$\frac{\partial C_0}{\partial a_i^{(L-1)}} = \sum_{i=0}^{n^{(L)}-1} \frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial C_0}{\partial a_j^{(L)}} \tag{2.18}$$

Durch die in Abbildung 2.5 visualisierte Abhängigkeit zwischen $a_j^{(L)}$ und $a^{(L-1)}$ wird die Rekursivität der Backpropagation angestoßen, da $a^{(L-1)}$ wiederum von $a^{(L-2)}$ abhängig

ist, usw. Dadurch fängt die Backpropagation in der Ausgabeschicht an und iteriert durch das Netz schichtweise bis die Eingabeschicht erreicht wird, da dort $a^{(1)}$ nur noch abhängig vom Eingabevektor ist. Während der Iteration durch die Schichten werden die Parameter angepasst, sodass sich der berechnete Fehler reduziert.

Aus den obigen Komponenten kann somit vollständig der Gradient ∇C gebildet werden.

2.3 Convolutional Neural Networks

Aufbauend auf die Grundlagen der bereits vorgestellten Feedforward-Netzwerke stellten LeCun und Bengio [42] die Convolutional Neural Networks, kurz CNNs, vor. CNNs eignen sich besonders für Eingabedaten die Strukturen aufweisen. Beispielsweise erfüllen Bilder oder auch zeitreihenbasierte Daten dieses Kriterium, wobei letzter Datentyp eine wichtige Rolle in dieser Arbeit spielt. Während die Eingabedaten für KNNs eine feste Größe haben müssen und demzufolge gegebenenfalls skaliert werden müssen, können die Daten zur Eingabe in ein CNN auch variable Größen sein. Der Begriff Convolution bedeutet Faltung. Bei der Faltung handelt es sich um eine lineare mathematische Operation. Es handelt sich genau dann um ein CNN, wenn das NN mindestens in einer Schicht den Faltungsoperator anstatt der Matrixmultipliktation anwendet [1]. Generell bestehen die Faltungsschichten eines CNN aus drei Grundbausteinen. Zuerst wird die Faltung (Kapitel 2.3.1) durchgeführt, auf die meist eine Aktivierungsfunktion folgt. Den Abschluss des Layers bildet die Pooling-Operation (Kapitel 2.3.2). Ursprünglich wurde die Sigmoid-Aktivierungsfunktion (2.5) in CNNs verwendet. Seit der Einführung der ebenfalls bereits vorgestellten ReLu-Aktivierungsfunktion (2.4) ist es möglich geworden die CNNs mit mehr Faltungsschichten zu konstruieren.

2.3.1 Faltung

Die Faltung von Daten soll Muster in der Eingabe erkennen und anschließend extrahieren. Für diesen Zweck bewegt sich ein Filter, der auch oft als Kernel bezeichnet wird, über die Eingabe und extrahiert die Merkmale zu einer Ausgabe namens Featuremap [1]. Der Kernel ist eine quadratische mehrdimensionale Gewichtsmatrix, die meist nur einen kleinen Bruchteil der Eingabe groß ist.

$$s(t) = (x * w)(t) = \int x(a)w(t-a)\,da \tag{2.19}$$

Die Faltung ist eine lineare mathematische Operation und kann als Gleichung (2.19) notiert werden. Dabei handelt es sich um den regulären Faltungssatz auf $\mathbb{R}^n$. In der digitalen

Signalverarbeitung findet jedoch die diskrete Faltung (siehe Gleichung 2.20) durch die einfachere Handhabung des Summenzeichens mehr Verwendung.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.20}$$

Der Kernel w nimmt den Platz der Gewichtung der Eingabefunktion x ein, während a den Einfluss des Wertes auf das Ergebnis regelt. Somit ergibt sich $s(t)$ aus dem durch $w(a)$ gewichteten Mittelwert der Funktion $x(t)$ auf dem vom Kernel betrachteten Bereich [1]. Die Summe aller Gewichte w ergibt 1. Als Eingabe wird entweder die Eingabe des CNNs oder eine vorherige Featuremap übergeben. Die Eingabe ist meist mehrdimensional. Unter dem Begriff 'Stride' wird die Schrittweite des Kenels bei der Iteration über die Eingabe verstanden. Eingaben haben generell einen Rand, der durch das sogenannte 'Padding' besser auf Merkmale überprüft werden kann.

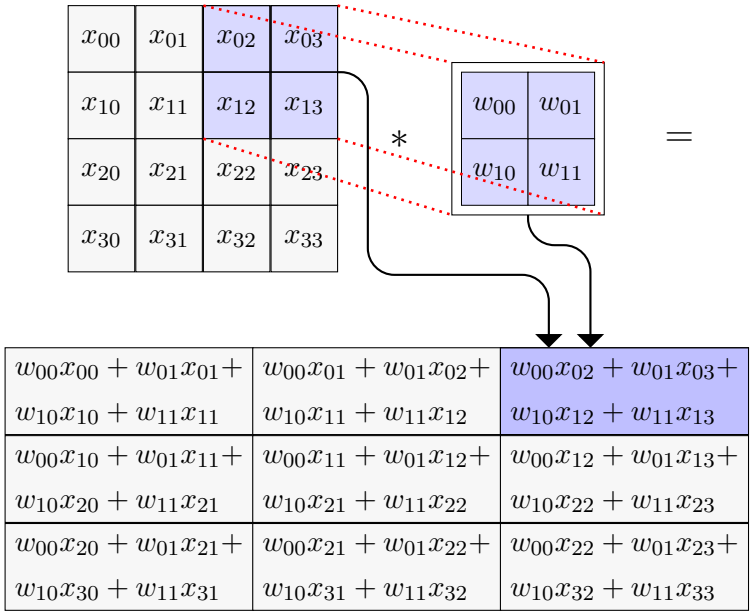

Abbildung 2.6: Beispiel einer Faltung auf einer zweidimensionale Eingabe, wie z.B. einem Bild. Der Kernel iteriert mit einer Schrittgröße von 2 und ohne Padding. Die Patches der Eingabe, hier beispielsweise x_{00}, x_{01}, x_{10}, x_{11} mit den Maßen des Kernels, werden gemäß der Gleichung (2.21) mit den Gewichtungen des Kernels verrechnet. Die einzelnen Elemente des Patches werden mit den jeweiligen Elementen des Kernels multipliziert und daraufhin aufsummiert (siehe Ausgabe/Featuremap).

$$S(t) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.21}$$

$$\iff S(t) = (I * K)(i,j) = \sum_m \sum_n K(i-m, j-n)I(m,n) \tag{2.22}$$

Die Parameter des Kernels K werden während des Lernprozesses des CNNs angepasst. Da es sich bei dem Faltungssatz um eine kommutative Operation handelt, kann er gemäß Gleichung 2.22 umgestellt werden. Die Umstellung bringt den Vorteil mit sich, dass es weniger legitime Werte für die Parameter m und n gibt und somit der Lernprozess schneller gut approximierende Werte auffinden kann.
Filter können für die einzelnen Patches der Eingabe wiederverwendet werden, somit teilen sich Neuronen einer Schicht einen Kernel. Diese Eigenschaft läuft unter dem Begriff 'Parameter Sharing'. Der dadurch resultierende Vorteil von CNNs ist der geringere Speicherbedarf für Parameter, da diese zu einem gewissen Teil geteilt werden [1]. Der gesamte Faltungs-Prozess wird in Abbildung 2.6 nochmal grafisch veranschaulicht.

2.3.2 Pooling

Nach der Faltung und der darauf folgenden Aktivierungsfunktion wird die Ausgabegröße der Featuremap durch eine Pooling-Operation reduziert. Bei der Reduktion der Auflösung spricht man vom 'Downsampling' [1]. Die Pooling-Operation sorgt für eine verbesserte Runtime und einen geringeren Speicherbedarf. Im Gegenzug kann das CNN dafür mit mehr Tiefe konstruiert werden, was den heutigen Trend bei NNs gut widerspiegelt. Wie bei der Faltung, so wird auch beim Pooling ein kleiner quadratischer Filter angewendet. Dieser wird ebenfalls über die Eingabe iteriert, jedoch unterscheidet sich die auf den betrachteten Bereich ausgeführte Rechenoperation. Neben dem Average Pooling ist das Max-Pooling eine der beliebtesten Varianten (siehe Abbildung 2.7).

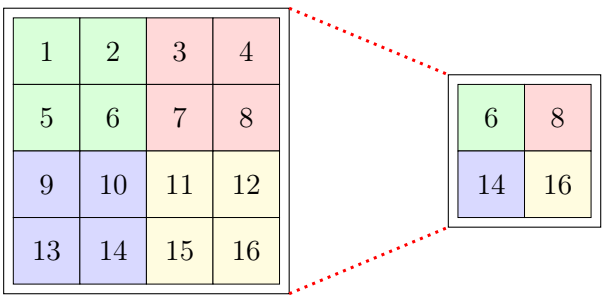

Abbildung 2.7: Beispiel einer Max-Pooling-Operation mit einer Schrittgröße von 2.

In dem vom Filter betrachteten Bereich wird das Maximum ausgewählt und in die Ausgabe übergeben. Dadurch werden die relevanten Daten im lokalen Umfeld behalten und die weniger aussagekräftigen Daten ausgefiltert.

Neben den bereits angesprochenen Nutzen hat Pooling ebenfalls den Vorteil der Invarianz der Zwischenergebnisse (Featuremaps) gegenüber kleinen Veränderungen in den Eingabedaten [43].

2.3.3 Fully connected Layer

Die Gruppierung aus Faltung, optionaler Aktivierungsfunktion und Pooling-Operation bildet zusammen eine Schicht. Die Schicht kann beliebig oft wiederholt werden, um zugleich die Komplexität des NNs und die der extrahierten Merkmale zu erhöhen. Im Gegensatz zu Feedforward-NNs sind die Layers des CNNs meist nicht vollständig vernetzt. Da bei einem CNN nur benachbarte Neuronen Verbindungen teilen und die Anzahl an Parametern bereits geringer ausfällt (s.o.), profitiert es generell von einer besseren Laufzeit und einem geringeren Speicherbedarf als ein herkömmliches Feedforward-NN.

Das Layer nach der letzten Pooling-Operation bildet jedoch als völlig vernetze Schicht die Ausnahme im CNN. Die weitergereichte Featuremap der Pooling-Operation wird in dieser Schicht 'ausgerollt', d.h. es wird jedem Element ein Neuron zugewiesen. Diese Schicht ist anschließend mit der letzten Schicht des Netzwerks, dem Outputlayer, vollständig vernetzt. Die Anzahl an Neuronen im Outputlayer entspricht der Anzahl an Klassen des NNs. Dieses Fully Connected Layer ist besonders wichtig bei Klassifizierungen und daher in dieser Arbeit nicht zu vernachlässigen.

2.4 Binäre neuronale Netze

CNNs erreichen state-of-the-art Ergebnisse in verschiedensten Anwendungsbereichen, etwa bei dem in dieser Arbeit verwendeten Klassifizierungsverfahren. Solch komplexe NN werden fast ausschließlich auf einer oder mehreren leistungsstarken GPUs trainiert, die meist einen sehr hohen Stromverbrauch aufweisen [44]. Es ist eine regelrechte Herausforderung diese NNs auf Geräten mit geringer Rechenleistung und zudem geringerem Stromverbrauch zu betreiben.

2.4.1 Komprimierung

Zur Bewältigung dieser Herausforderung ist es unumgänglich das Netzwerk performanter zu gestalten. Infolgedessen gibt es bereits einige Ansätze die Informationen in einem CNN zu komprimieren. Neben der Reduzierung von Parametern und Operationen durch kluge Netzwerk Designs setzt sich der Gebrauch von Parametern mit niedriger Auflösung durch. Reguläre NNs rechnen mit full-precision floating point Gewichten und Aktivierungen. Dabei handelt es sich um Gleitkommazahlen, die durch die Zuweisung von 32 bits Speicher eine “volle Genauigkeit“ bieten. Quantisiete Gleitkommazahlen benötigen hingegen weniger Speicher und werden in den Rechenoperationen leichter gehandhabt, jedoch weisen sie eine geringere Präzision auf. Die Quantisierung kann dabei beispielsweise auf eine 8 bit Auflösung zurückgreifen oder sogar runter bis zur Binarisierung reichen [45–48]. NNs mit binären Gewichten und Aktivierungen werden binäre neuronale Netze (BNNs) genannt und können bis einen bis zu 32 mal geringeren Speicherbedarf aufweisen und die Laufzeit bis

zu 52-fach verkürzen. Die enorme Reduktion der Laufzeit lässt sich unter anderem auf die einfacheren Rechenoperationen zurückführen. Bitweise Operationen, wie z.B. XNOR, sind für Recheneinheiten erheblich einfacher zu lösen, als die herkömmlichen arithmetischen Operationen.

2.4.2 Binäre Schicht

In einer binären Schicht werden die full-precision floating-point Werte mithilfe der Sign-Funktion [49] in binäre Werte transformiert.

$$sign(x) = \begin{cases} +1 & \text{falls } x \geq 0, \\ -1 & \text{sonst} \end{cases} \tag{2.23}$$

Um den gesamten Prozess eines NNs abzudecken, wird die Quantisierung der Werte mittels eines Straight-Through-Estimators, kurz STE [49], implementiert.

$$\text{Forward: } x_o = sign(x_i) \tag{2.24}$$

$$\text{Backward: } \frac{\partial C}{\partial x_i} = \frac{\partial C}{\partial x_o} 1_{|x_i| \leq t_{clip}} \tag{2.25}$$

Im Forwardpass (2.24) wird x_i als Input aus $\mathbb{R}$ in die Sign-Funktion (siehe Formel 2.23) übergeben und zu dem binären Output $x_o \in -1, +1$ transformiert. Während der Backpropagation (2.25) wird der Gradient für zu große Inputs x_i durch t_{clip} begrenzt [45]. Wie in den meisten Fällen wird auch in dieser Arbeit $t_{clip} = 1$ gesetzt. Die resultierenden Verläufe des STEs werden in Abbildung 2.8 dargestellt.

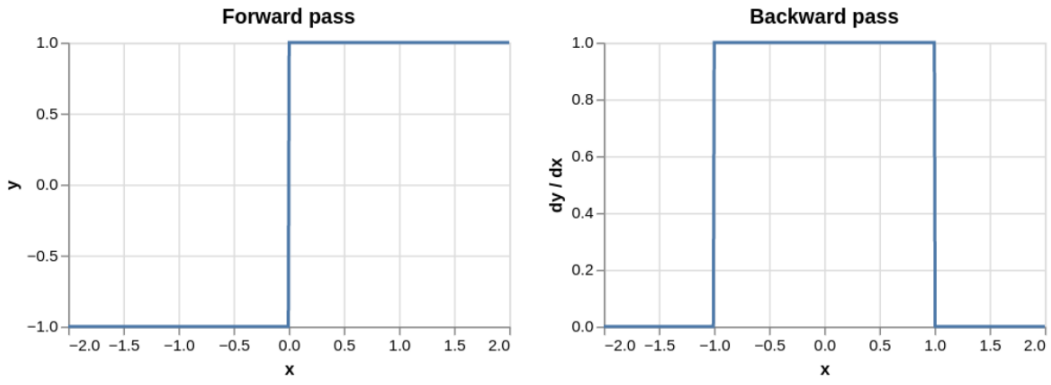

Abbildung 2.8: SteSign-Funktion im Forward Pass und im Backward Pass. (Abbildung aus [50])

Kapitel 3

Verwandte Arbeiten

In diesem Kapitel wird ein Einblick in zwei aktuelle Arbeiten gegeben, die sich ebenfalls mit dem Themengebiet der Zeitreihenklassifikation auseinandersetzen.

3.1 The great time series classification bake off

Das Review samt experimenteller Evaluation [18] gibt einen strukturierten Überblick des breiten Spektrums an Ansätzen zur Zeitreihenklassifikation durch Gruppierung in 6 verschiedene Kategorien.

Vollständige Serien Bei dieser Technik werden zwei Zeitreihen auf deren gesamte Spannweite miteinander verglichen. Gleichwie bei einer regulären Klassifikation kann der Vergleich als Vektor durchgeführt werden. Der vektorielle Vergleich ist jedoch suboptimal bei einer abweichenden Lage der übereinstimmenden markanten Stellen. Aus diesem Grund wird überwiegend eine elastische Distanzmessung verwendet, die meist von der Nächste-Nachbarn-Klassifikation (1-NN) Gebrauch macht.

Phasenabhängige Intervalle Anstatt die gesamte Zeitreihe zu betrachten, gibt es Ansätze, bei denen sich der Vergleich auf Intervalle beschränkt. Durch die Wahl eines geeigneten phasenabhängigen Intervalls kann die Genauigkeit der Klassifizierung drastisch profitieren. Die Aufgabe des Algorithmus besteht darin, das optimale Intervall anhand von Attributen zu ermitteln. Dabei muss es sich nicht zwangsweise um ein einziges oder eine feste Anzahl von Intervallen handeln, eine beliebige Anzahl von Intervallen führt meist zum besten Ergebnis.
Dieses Verfahren eignet sich insbesondere für lange Zeitreihen mit Teilserien, die Störsignale beinhalten oder phasenabhängig diskriminierend sind, wie beispielsweise in Abbildung 3.1.

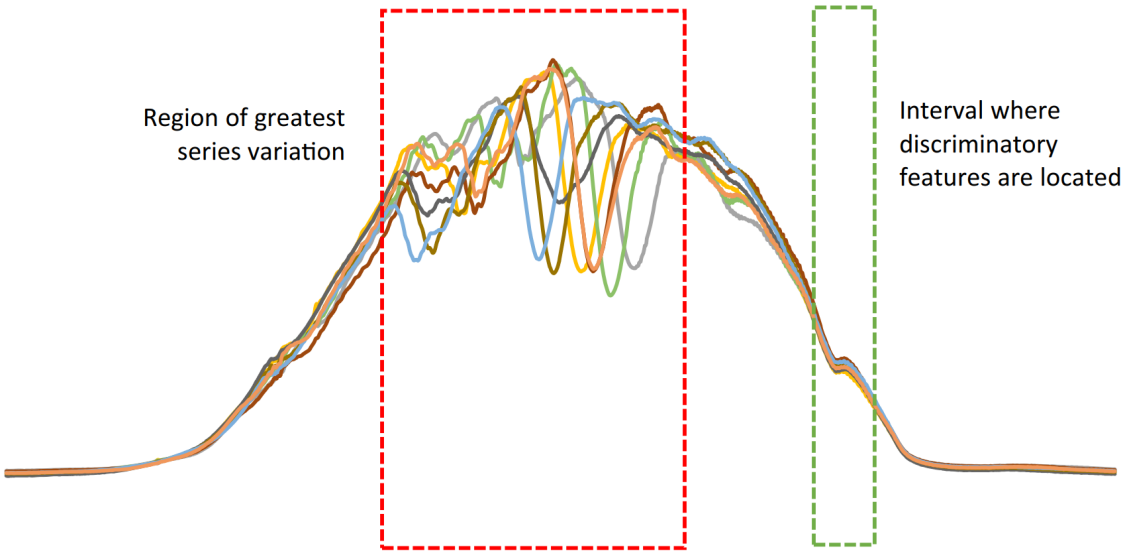

Abbildung 3.1: Graph eines beispielhaften Datensatzes bei dem dieses Verfahren vorteilhaft ist. Die Störsignale am Anfang der Zeitreihe würden die Genauigkeit der Klassifizierung negativ beeinflussen, während sich das phasenabhängig diskriminierende Interval im hinteren Bereich für eine präzise Klassifizierung eignet. (Abbildung aus Arbeit [18].)

Phasenunabhängige Shapelets Die dritte Kategorie von Ansätzen macht sich ebenfalls Teilserien zu Nutze. Diese Teilserien sind in diesem Fall jedoch phasenunabhängig, beinhalten charakteristische Muster und werden *Shapelets* genannt. Da die *Shapelets* phasenunabhängig sind, kann deren Position innerhalb der gesamten Serie vernachlässigt werden. Indessen gibt die jeweilige An- oder Abwesenheit Auskunft über mögliche Ähnlichkeit zwischen den Serien.
In Abbildung 3.2 wird dieses Klassifizierungsverfahren an einer Zeitreihe des *NonInvasiveFetalECGThorax2* Datensatzes weiter erläutert. Bei der Zeitreihe handelt es sich um die Aufnahme eines einzelnen Herzschlages. Algorithmen mit phasenabhängigen Intervallen würden in diesem Anwendungsbeispiel weniger geeignet sein, da sich das ausschlaggebende Merkmal an einer beliebigen Stelle befinden kann.

Dictionary based Klassifizierung Das soeben vorgestellte Verfahren klassifiziert mithilfe der phasenunabhängigen Shapelets die Zeitreihe anhand der besten Übereinstimmung innerhalb der Reihe. In machen Anwendungsfällen können jedoch auch die restlichen geringwertigeren Übereinstimmungen eine Rolle in der Klassifizierung spielen. In der *Dictionary based Klassifizierung* ist insbesondere die Frequenz durch das wiederholte Vorkommen des Merkmales ausschlaggebend für die Einordnung der Zeitreihe.

Kombinationen Die vorgestellten Ansätze bauen zum großen Teil aufeinander auf und erweitern sich, sie schließen sich jedoch gegenseitig nicht zwangsweise aus. Daher können

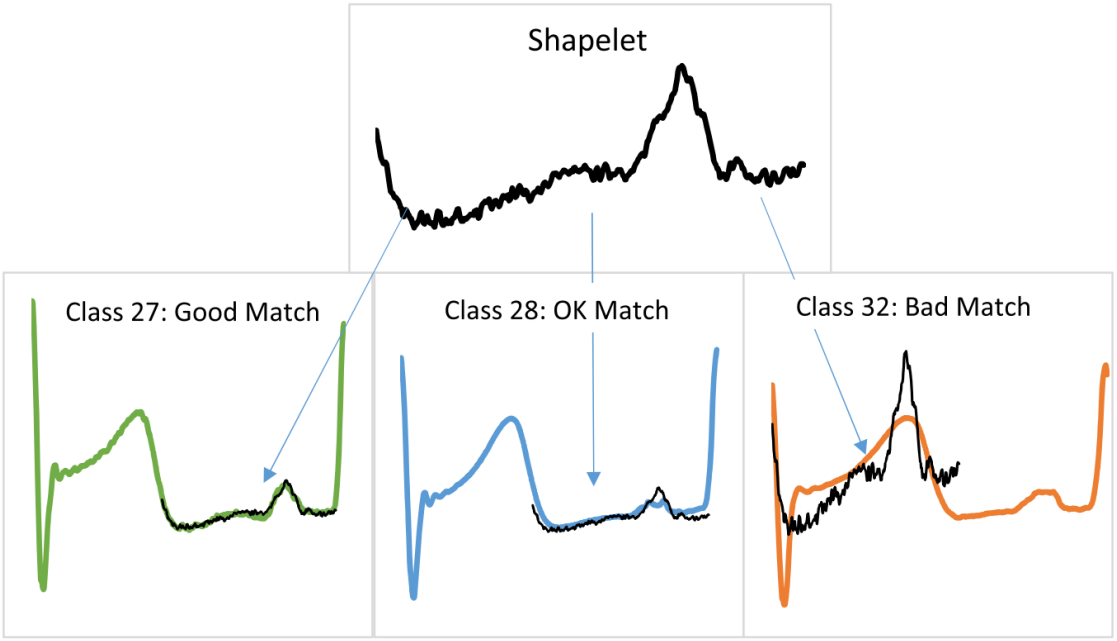

Abbildung 3.2: Ein Ausschnitt des charakteristischen Verlaufs der Klasse 27 approximiert das Shapelet gut. Dementgegen wird für die Klasse 32 eine eher schlechte Deckung festgestellt. (Abbildung aus Arbeit [18].)

auch Kombination von mehreren der bereits vorgestellten Ansätze implementiert werden. Diese Kombinationen bilden eine weitere Kategorie von Ansätzen zur Zeitreihenklassifikation. Ein weit verbreitetes Beispiel ist COTE (Collective of transformation ensembles) [51]

Modellbasierende Klassifizierung Anstatt die Zeitreihen direkt zu miteinander zu vergleichen, können stellvertretend generative Modelle genutzt werden. Für jede Zeitreihe wird ein generatives Modell erstellt und angepasst. Der Vergleich zwischen den Modellen entscheidet die Klassifizierung der zugehörigen Zeitreihen. Diese Katergorie ist jedoch nach Auffassung der Autoren für ein großen Anteil der untersuchten Datensätze nicht wettbewerbsfähig.

3.2 Deep learning for time series classification: a review

Das Ziel des Reviews vonHassan Fawaz [15] ist es state-of-the-art Performance für Zeitreihenklassifikationen unter Verwendung von Deep-Learning-Algorithmen zu erreichen. Dabei wurden 8730 Deep-Learning Modelle auf 97 Datensätze angewandt. Diese Studie ist somit die aufwändigste Studie auf dem Gebiet der Zeitreihenklassifikation und bildet das Fundament dieser Bachelorarbeit. Um einen Vergleich zwischen den verschiedenen Ansätzen zu ermöglichen, wurde das UCR/UEA Archive [24] (siehe Kapitel 5.2) und 12 multivariate

Datensätze als TSC Benchmark herangezogen.
Vorab gibt die Arbeit einen Einblick in den aktuellen Stand der Technik für Zeitreihenklassifikationen. Das Ensemble *HIVE-COTE* baut, durch die Ergänzung von zwei weiteren Klassifikatoren und einem Hierarchical Vote System auf *COTE* (steht für Collective of Transformation-Based Ensembles) auf und bildet den derzeitigen state-of-the-art Algorithmus für TSC. Unter den insgesamt 37 Klassifikatoren befinden sich renommierte Algorithemen wie *Elastic Ensemble* von Lines and Bagnall (2015) [52] oder der in 2014 von Hills vorgestellte Shaplet Transform Algorithmus (ST) [53]. Durch das Trainieren des ST Klassifikators ist *HIVE-COTE* sehr rechenintensiv.
Die meisten Ansätze für TSC nutzen keine DNNs. Auf der Grundlage wird in der Arbeit die primäre Frage gestellt, ob es einen DNN Ansatz gibt, der die state-of-the-art Performance von *HIVE-COTE* erreicht und dabei weniger komplex ist. Zusätzlich ermittelt die Arbeit, welche DNN Architekturen sich am ehesten für die Zeitreihenklassifikation eignen. Dafür werden jeweils 10 Durchläufe für 9 Deep-Learning Modelle auf 85 univariaten Zeitreihen Datensätzen (UCR/UEA) durchgeführt. Die resultierenden Ergebnisse werden in Abbildung 3.3 miteinander verglichen.

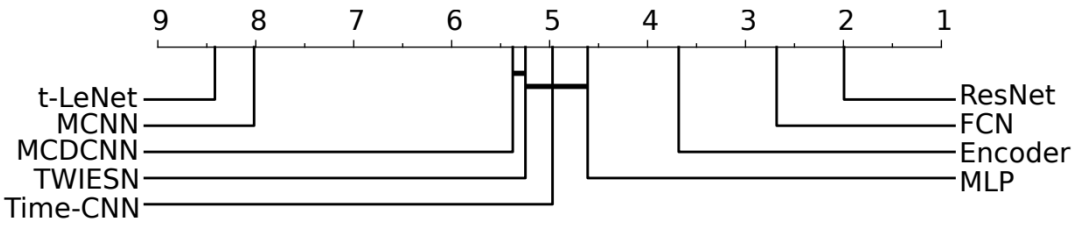

Abbildung 3.3: Paarweisen Vergleich der 9 Deep-Learning Modelle der Genauigkeit auf dem univariaten UCR/UEA Archiv. Auf der X-Achse ist der durchschnittliche Rang aufgereiht. (Abbildung aus Review [15].)

In Abbildung 3.3 ist zu erkennen, dass sich ResNet gegenüber den anderen Modellen durchsetzt. Mit einem durchschnittlichen Rang von annähernd 2 und der Überlegenheit in 50 von 85 Datensätzen setzt sich ResNet auch gegenüber FCN mit einem großen Vorsprung durch. Dieser Vorsprung wird durch die tiefe flexible Architektur von ResNet begründet. Das gute Abschneiden der CNN Architekturen kann auf den Erfolg von CNNs im zweidimensionalen Raum zurückgeführt werden. Die Ermittlung von ausschlaggebenden Merkmalen sollte für ein CNN in dem eindimensionalen Raum Zeit sogar einfacher sein.

Um die primäre Frage des Reviews zu beantworten, also ob es einen DNN Ansatz mit state-of-the-art Performance gibt, wird ResNet mit state-of-the-art Ansätzen verglichen. Die Entscheidung fiel auf ResNet aufgrund der höchsten Genauigkeit und der ähnlichen Laufzeit zu FCN. Für den Vergleich zwischen ResNet und dem aktuellen Stand der Tech-

nik werden 7 andere bekannte Klassifikatoren herangezogen. Darunter befinden sich die 4 besten der 18 von Bagnall (2017) [18] evaluierten Algorithmen: Elastic Ensemble (EE), Bag-of-SFASymbols (BOSS), Shapelet Transform (ST) und Collective of Transformation-based Ensembles (COTE). Zudem wurde mit NN-DTW-WW ein auch heutzutage sehr beliebter Ansatz hinzugefügt, in dem die Nächste-Nachbarn-Klassifikation (NN) mit warping window (WW) und DTW verbunden wird. Schließlich wird mit Proximity Forest (PF) ein äußerst moderner Ansatz und mit HIVE-COTE der aktuelle Performance Spitzenreiter dem Vergleich 3.4 hinzugefügt.

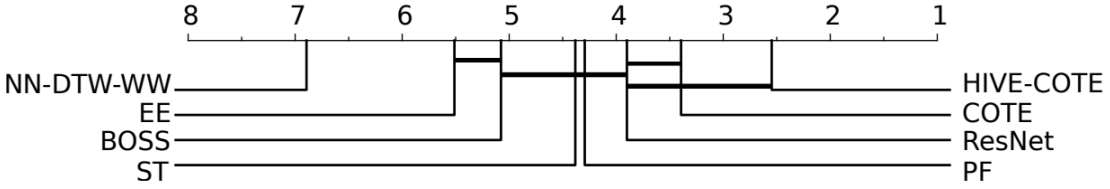

Abbildung 3.4: Paarweiser Vergleich der state-of-the-art Klassifikatoren auf dem univariaten UCR/UEA Archiv. (Abbildung aus Review [15].)

In Abbildung 3.4 wird für ResNet der Median der Genauigkeit für die Testmenge herangezogen. Zusätzlich wird erörtert, dass die Hyperparameter vom ResNet Agorithmus nicht auf die einzelnen Datensätze optimiert wurden, sondern lediglich eine Optimierung für den gesamten Benchmark stattfand. Elastic Ensemble (EE) und NN-DTW-WW schneiden am schlechtesten ab und können mit dem heutigen Standard nicht mehr mithalten. Auch in diesem Vergleich bewahrt HIVE-COTE den ersten Platz. Durch die Verwendung von 37 Algorithmen resultiert für HIVE-COTE eine enorme Laufzeit von $\mathcal{O}(N^2 \cdot T^2)$, wobei N die Anzahl an Zeitreihen des Datensatzes und T die Länge einer Zeitreihe ist. Die Laufzeit wird durch den Shapelet Transform (ST) Algorithmus bestimmt, der isoliert im Diagramm lediglich durchschnittlich abschneidet. Durch die immense Laufzeit ist HIVE-COTE in der Praxis recht ineffizient und nicht gut anwendbar. Die lineare Verfahrensweise von NN schränkt HIVE-COTE zusätzlich ein, während ResNet durch die triviale GPU-Parallelisierung von DNNs profitiert.

3.3 Vergleichsmodelle

In dem Review wurden 9 Deep-Learning Modelle auf univariaten Zeitreihen evaluiert (siehe Abbildung 3.3). Von diesen Modellen wurden 3 von Wang (2017) [54] als geeignete Architekturen zur TSC vorgeschlagen. Dabei handelt es sich um das Multi Layer Perceptron (Kapitel 3.3.1), das Fully Convolutional Neural Network (Kabitel 3.3.2) und das Residual Neural Network (Kapitel 3.3.3). Es wird ein Überblick über die 3 Modelle verschafft, da sie im Laufe der Arbeit als Vergleichsmodelle herangezogen werden.

3.3.1 Multi Layer Perceptron

Das Multi layer Perceptron (MLP) ist die herkömmlichste Form von DNNs und wurde von Wang (2017) [54] als grundliegende Architektur für TSC herangezogen. Der Aufbau des MLPs wird in Abbildung 3.5 grafisch dargestellt. Ein MLP besteht aus insgesamt 4 Schichten, die mit dem Outpot des vorherigen Layers vollständige verbunden sind. Die letzte Schicht ist ein Softmax-Klassifikator (siehe Gleichung 2.6), dessen Anzahl an Neuronen der Anzahl an Klassen des Datensatzes entspricht. Die anderen 3 Schichten sind "Hidden fully connectecd Layer" mit jeweils 500 Neuronen und einer nachfolgenden ReLU-Aktivierungsfunktion (siehe Gleichung 2.4). Vor jedem Hidden Layer wird eine Dropout Operation durchgeführt [55]. Dabei handelt es sich um eine Regularisierung des Feedforward Modells, bei der ein gewisser Prozentsatz an Neuronen deaktiviert wird. Der Prozentsatz wird mit einer Dezimalzahl angegeben und lauten für die erste Schicht 0.1, für die zweite und dritte Schicht 0.2 und in für die letzte Schicht 0.3. Neben der Verkürzung der Laufzeit bringen Dropout Operationen den zusätzlichen Vorteil, dass das sogenannte Overfitting [55] verhindert werden kann.
Unter Overfitting versteht man die Überanpassung von dem Modell, bei dem der "wahre Zusammenhang" der Trainingsdaten verschwindet und das Modell lediglich die spezifischen Trainingsdaten sehr genau abbildet. Das Phänomen tritt oft bei komplexeren Modellen mit vielen Parametern auf und sorgt für ein schlechtes Vorhersagen für bisher unbekannte Daten.
Abschließend ist die Anzahl an Parametern des MLPs abhängig von der Länge des ins Netzwerk eingegeben Zeitreihe.

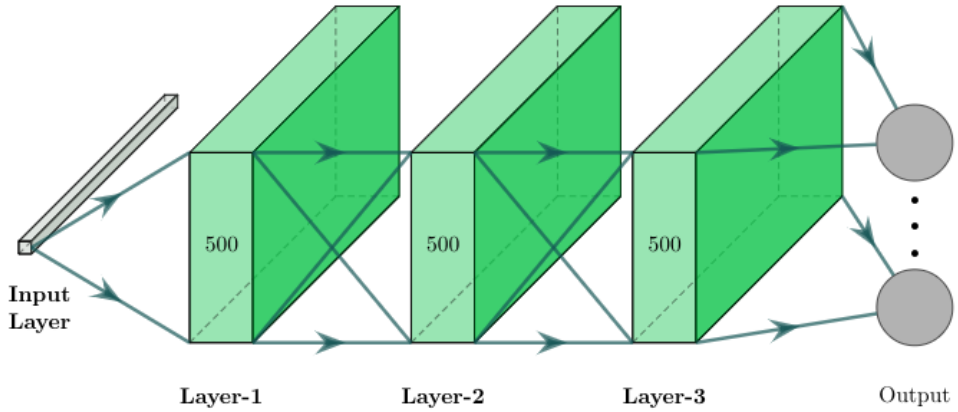

Abbildung 3.5: Grafische Darstellung der MLP Architektur für TSCs.

3.3.2 Fully Convolutional Neural Network

Die namensgebenden Faltungen werden in dieser Architektur in 3 sogenannte Faltungsblöcke aufgeteilt. In jedem Block befindet sich eine Faltung, die von einer sogenannten Batch Normalization [56] und schließlich einer ReLU-Aktivierungsfunktion (siehe Gleichung 2.4) gefolgt wird. Eine Batch Normalization wird im Allgemeinfall angewendet, um das NN zu stabilisieren. Das geschieht durch die Neuskalierung und Neuzentrierung der Aktivierungen der vorherigen Schicht unter Berücksichtigung des zugehörigen Mittelwerts und der Standardabweichung. Fully Convolutional Neural Networks (FCNs) nutzen generell keine lokalen Pooling Schichten, wodurch die Länge der Zeitreihen während der Faltungen komplett unverändert bleibt. Die hier implementierte Architektur (siehe Abbildung 3.6) hat die Besonderheit, dass das herkömmliche Fully Connected Layer mit einer Global Average Pooling (GAP) Schicht ersetzt wurde. Die Ausgabe des dritten Faltungsblocks wird in der vierten Schicht durch das GAP über die gesamte Zeitdimension gemittelt. Die Anwendung von GAP reduziert die Anzahl an Parametern im NN drastisch und sorgt für Invarianz in der Anzahl an Parametern gegenüber des Datensatzes in den ersten vier Schichten. Dem gegenüber steht die letzte Schicht, in der die Anzahl der Neuronen abhängig von der Anzahl an Klassen des Datensatzes ist. Die letzte Schicht ist ein Softmax-Klassifikator (siehe Gleichung 2.6) und vollständig verbunden mit der vorangegangen GAP-Schicht. Die Stride (siehe unter Kapitel 2.3.1) ist auf 1 gesetzt und das Padding ist 0, damit die Länge der Zeitreihe durch die Flatung unverändert bleibt. Die erste Faltung hat 128 Filter und eine Kernel Größe von 8, während die zweite Faltung mit 256 Filtern die doppelte Tiefe, jedoch mit einer Kernel Größe von 5 eine beschränktere Faltung hat. Die dritte Faltung hat wieder 128 Filter aber eine Kernel Größe von 3.
FCN hat keine Regularisierungs-Operation wie beispielsweise Dropout um das Risiko von Overfitting einzudämmen.

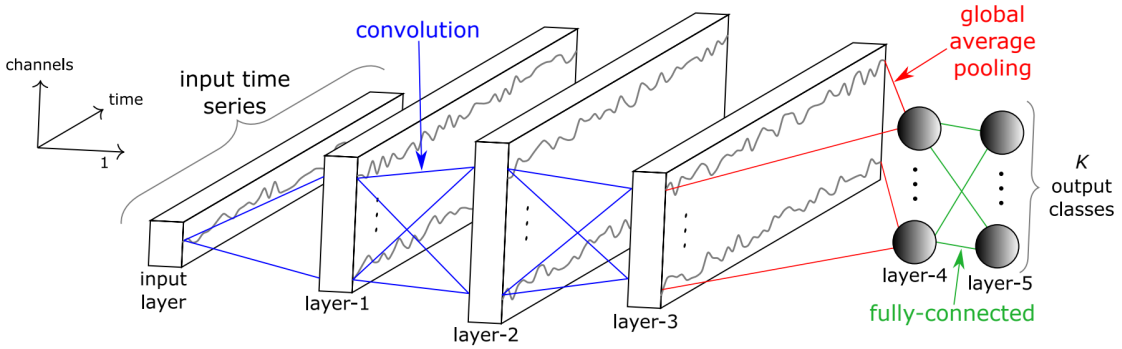

Abbildung 3.6: Grafische Darstellung der FCN Architektur für TSCs. (Abbildung aus Review [15].)

3.3.3 Residual Neural Network

Aus denen von Wang (2017) [54] vorgeschlagenen Architekturen ist ResNet [57] mit insgesamt 11 Schichten die tiefste Architektur. Wie bei der FCN Architektur (Kapitel 3.3.2) wird auch beim ResNet eine GAP Schicht vor der letzten Schicht eingesetzt. Die letzte Schicht ist in diesem Fall auch ein Softmax-Klassifikator (siehe Gleichung 2.6), dessen Anzahl an Neuronen identisch mit der Anzahl an Klassen des Datensatzes ist. Es resultiert wie beim FCN (3.3.2) eine Invarianz in der Anzahl an Parametern gegenüber der Eingabelänge, außer in der Softmax Schicht.
Die restlichen 9 Schichten sind Faltungsschichten und werden in 3 Blöcke aufgeteilt. Das Alleinstellungsmerkmal der ResNet Architektur sind die Shortcuts zwischen den Blöcken. Die genauen Verbindungpunkte der Shortcuts können in Abbildung 3.7 nachvollzogen werden. Durch diese zusätzlichen Verknüpfungen ist es dem Gradienten möglich, eine "Abkürzung" zu nehmen, wodurch das Problem der verschwindenden Gradienten [58] reduziert wird. Jeder Block besteht aus 3 Faltungsschichten, mit einer Kernel Größe von 8 für die erste Faltung, 5 für die zweite Faltung und einer Kernel Größe von 3 für die dritte Faltung. Die Schichten im ersten Block haben jeweils 64 Filter und die Schichten in der zweiten und der dritten Schicht haben jeweils 128 Filter. Nach jeder Faltung wird eine Batch Normalization [56] und daraufhin eine ReLU-Aktivierungsfunktion (siehe Gleichung 2.4) ausgeführt.

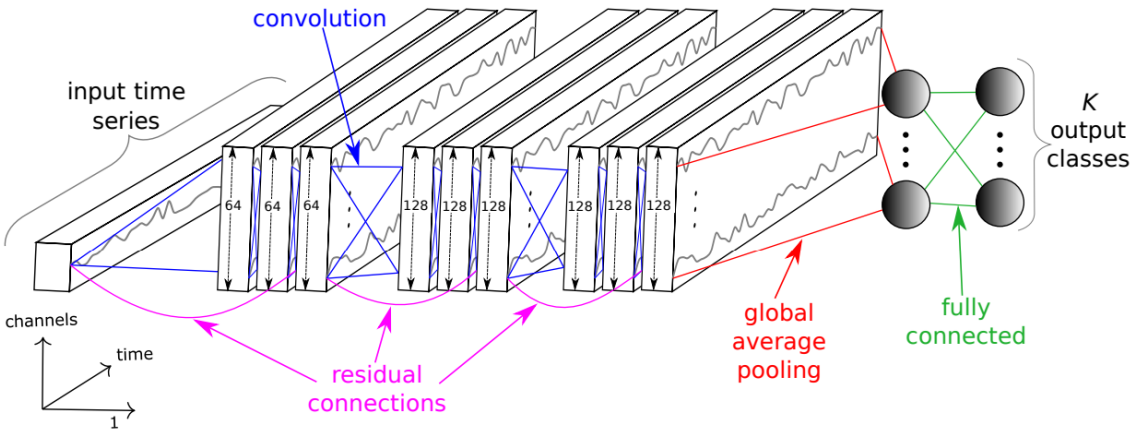

Abbildung 3.7: Grafische Darstellung der ResNet Architektur für TSCs. (Abbildung aus Review [15].)

3.4 Verbindung

Das Reviews vonHassan Fawaz [15] bildet das Fundament dieser Arbeit. Die Auswahl von geeigneten Modellen zur TSC wurde mithilfe der Ergebnisse des Reviews getroffen. Unter anderem wurden die drei Modelle MLP, FCN und ResNet gewählt, da sie sich in dem paarweisen Vergleich der Genauigkeit gegenüber sechs weiteren Deep-Learning Modellen in Abbildung 3.3 als performant erwiesen haben. Im weiteren Verlauf der Arbeit werden binäre Modelle implementiert, die vom Aufbau an die drei Architekturen im Kapitel 3.3 angelehnt sind. Des Weiteren werden diese Architekturen in der Evaluation der Arbeit als Vergleichsmodelle herangezogen. Dabei wird für einen erweiterten Vergleich von jedem Vergleichmodell auch eine binarisierte Variante implementiert (siehe Kapitel 4.1).

Neben dem strukturierten Überblick über die Ansätze zur TSC liefert das Review samt experimentelle Evaluation von Bagnall [18] auch die Daten für den Vergleich zwischen den in dieser Arbeit implementierten Modellen und dem state-of-the-art Ansatz *HIVE-COTE* (siehe Kapitel 5.3.6).

Kapitel 4

Betrachtete Modelle

Nachdem ein Überblick an Ansätzen zur Zeitreihenklassifikation gegeben wurde, werden nun die in dieser Arbeit berücksichtigten Modelle präsentiert. In dieser Arbeit wurden vier binäre Modelle implementiert und untereinander, sowie mit Literaturmodellen verglichen. Als Literaturmodelle werden die in Kapitel 3.3 bereits vorgestellten Modelle MLP (3.3.1), FCN (3.3.2) und ResNet (3.3.3) der verwandten Arbeit von Hassan Fawaz [15] betrachtet.
Zusätzlichen werden binarisierte Varianten der drei Modelle angefertigt (siehe Unterabschnitt 4.1), um einen Vergleich zwischen der Implementierung dieser Arbeit und der reinen Binarisierung von Modellen zu ermöglichen.
Die Implementierung dieser Modelle geschieht in einer Umgebung, die sich die open-source Python Bibliothek Larq zur Erstellung von BNNs zu Nutzen macht. Auf die Umgebung wird im Kapitel 5.1 genauer eingegangen.

4.1 Binarisierte Vergleichsmodelle

Eine reine Binarisierung von full-precision Modellen führt generell zu einem massiven Informationsverlust innerhalb des Modells, weswegen es zu Einbußen in der Genauigkeit kommt. Um diese Einbußen in der Genauigkeit zu ermitteln wurden binäre Varianten der full-precision Modelle MLP, FCN und ResNet angefertigt. Dafür wurden die Modelle in die Umgebung der Arbeit übertragen und fast vollständig binarisert. Der generelle Aufbau der Modelle bleibt dabei gleich, daher bleiben die in den Kapiteln 3.3.1-3.3.3 angegebenen Merkmale der Modelle gültig. Die Binarisierung wird mithilfe einer Quantisierungsfunktion durchgeführt. Als Quantisierungsfunktion wurde für alle Vergleichsmodelle der Quantisierer *SteSign* (siehe Kapitel 2.4.2) verwendet. Dieser kann in den vorgefertigten Schichten von Larq über die zwei Quantisierer Argumente an die Schicht angebracht werden. Über das eine Argument kann die Quantisierungsfunktion auf die Eingabe der Schicht angewendet werden, während das andere Argument die Anwendung der Quantisierungsfunktion auf die

Kernel Gewichtsmatrix ermöglicht. Bei allen drei Modellen werden für sämtliche Schichten beide Argumente genutzt, das bezieht selbst die Ausgabeschicht mit ein. Die einzige Ausnahme bildet jedoch die erste Schicht, in der lediglich die Quantisierungsfunktion auf die Kernel Gewichtsmatrix angewandt wird. Die Eingabe in die erste Schicht wird, damit es nicht zu einem zu frühen Informationsverlust kommt, in voller Auflösung belassen.
Im weiteren Verlauf der Arbeit wird auf die binarisierten Vergleichsmodelle meist in Kombination mit dem Präfix "Cf" (englische Abkürzung für "Vergleich") referiert.

4.2 Implementierte Modelle

Da die Quantisierung von Neuronalen Netzen zu Einbußen in der Genauigkeit führt, müssen die Architekturen angepasst werden, damit sie eine Chance haben mit ihrem jeweiligen full-precision Gegenstück mitzuhalten. Neben den Einbußen in der Genauigkeit führt die Quantisierung jedoch auch zu einer drastischen Senkung des Speicherbedarfs und der Laufzeit (siehe Kapitel 2.4). Demzufolge kann das NN durch Anpassung an zusätzlicher Tiefe gewinnen, ohne dabei ineffizienter in der Praxis zu werden. Durch die zusätzliche Tiefe würde zudem der heutigen Tendenz in Richtung tieferer neuronaler Netze nachgegangen [1].

Hyperparameter Bei einem NN sind unter dem Begriff Hyperparameter (HP) die Parameter inbegriffen, die manuell gesetzt werden müssen und die Regulierung des Lernprozesses bestimmen. Um die bestmöglichen Architekturen zu ermitteln, ist die Anpassung der HP der Architekturen von großer Bedeutung. Aus diesem Grund werden in dieser Arbeit die HP der Architekturen mithilfe von Random-Search optimiert. Bei der Random-Search-Optimierung wird für den betrachteten HP zufällig ein Wert aus einer zuvor festgelegten Menge an Werten ausgewählt. Diese Auswahl an HPs wird bei jedem Datensatz für acht Variationen getätigt. Verglichen zur am häufigsten verwendeten Rastersuche [59] benötigt dieses Optimierungsverfahren lediglich einen kleinen Bruchteil der Rechenzeit mit mindestens genauso guten Ergebnissen [60]. Die betrachteten HP, sowie die zugehörigen Auswahlmengen werden für die jeweiligen Modelle in Tabellen aufgeführt (Tabellen 4.1, 4.2, 4.3 und 4.4).

Quantisierung der Modelle Die Quantisierung der folgenden vier NN wird konform zum Vorgehen bei den binarisierten Vergleichsmodellen (4.1) durchgeführt. Bei sämtlichen Schichten wird die Eingabe und die Kernel Gewichtsmatrix durch eine festgelegte Quantisierungsfunktion binarisiert, mit Ausnahme der Eingabe der ersten Schicht. Als grafischer Indikator wird die jeweilige erste Schicht des Modells in der zugehörigen Abbildung (4.1 - 4.5) bläulich hervorgehoben. Vor der Anwendung wird die Quantisierungsfunktion durch einen HP bestimmt. Für alle vier Modelle stehen dem HP dieselben drei Quantisierungsfunktionen *SteSign* (siehe Kapitel 2.4.2), *ApproxSign* und *SwishSign* zur Auswahl.

Es können neben der Eingabe in der ersten Schicht noch weitere Ausnahmen bezüglich der Quantisierung hinzukommen. Durch zwei zusätzliche HPs (siehe jeweiliges Tabellenende 4.1, 4.2, 4.3, 4.4) kann in dem jeweiligen Modell separat über die Quantisierung der Kernel Gewichtsmatrix in der ersten Schicht und der Quantisierung der Eingabe und der Kernel Gewichtsmatrix der letzten Schicht entschieden werden.

4.2.1 Binäres MLP

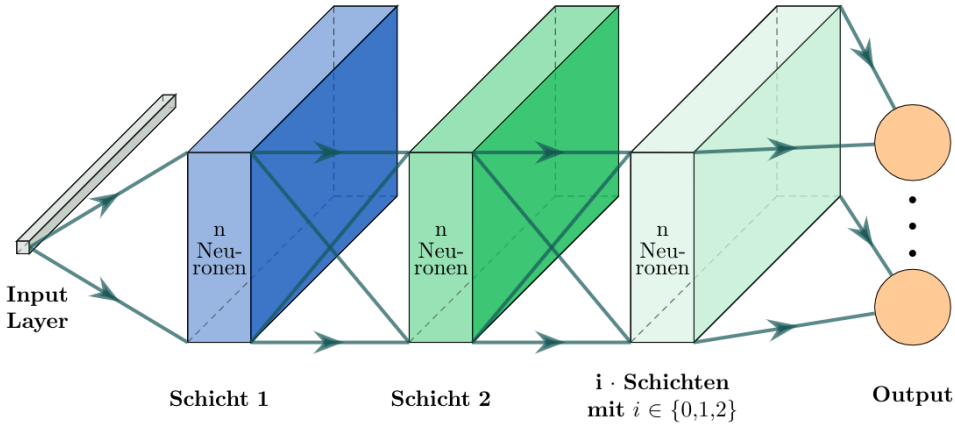

Abbildung 4.1: Grafische Darstellung der implementierten MLP Architektur für TSCs.

Der Aufbau des in dieser Arbeit implementierten binären MLPs wird in Abbildung 4.1 veranschaulicht. Das Modell besteht aus drei bis fünf vollständig vernetzten Schichten. Die letzte Schicht übernimmt die Rolle der Ausgabeschicht mit einer Softmax-Aktivierungsfunktion (siehe Gleichung 2.6), dessen Anzahl an Neuronen der Anzahl an Klassen des Datensatzes entspricht. Es müssen mindestens zwei Hidden Layer existieren, wobei durch den HP $i \in \{0, 1, 2\}$ (siehe zugehörige Tabelle 4.1) bis zu zwei weitere Hidden Layers hinzukommen können. Jedes Hidden Layer besteht aus $n \in \{128, 256, 512, 1024\}$ Neuronen. Üblicherweise folgt nach jedem Hidden Layer eine ReLU-Aktivierungsfunktion (siehe Gleichung 2.4), jedoch wird in dieser Implementierung darauf verzichtet, da durch die Quantisierungsfunktion in der Schicht bereits eine Aktivierungsfunktion durchgeführt wird. Auch in dieser Implementierung wird vor jedem Hidden Layer eine Dropout Operation durchgeführt. Die Dopout Rate wird neben der Lernrate und der Verfallsrate durch HPs bestimmt (siehe Tabelle 4.1). Die Verfallsrate (auch "Decay" genannt) sorgt innerhalb des NNs dafür, dass die nicht aktualisierten Gewichte stattdessen durch exponentielles Abfallen auf Null an Relevanz verlieren.
Wie in Kapitel 4.2 grundsätzlich beschrieben, wird durch die HPs für jedes dieser Modelle die Quantisierung der ersten und letzten Schicht optional. Da bei den ersten Auswertungen

der Ergebnisse des binären MLPs jedoch ein unerwartet hoher Speicherbedarf festgestellt wurde, wurden die beiden HPs zum Zweck der Reduzierung des Speicherbedarfs fixiert (siehe Tabellenende 4.1).

Hyperparameter des MLPs	
Funktion	mögliche Werte
Quantisierungsfunktion	SteSign, ApproxSign, SwishSign
Anzahl Neuronen pro Schicht **[n]**	128, 256, 512, 1024
Anzahl zusätzlicher Schichten **[i]**	0, 1, 2
Dropout Rate	0.0, 0.25, 0.5
Lernrate	10^{-3}
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}
Quantisierung des Kernels in erster Schicht	True
Quantisierung des Inputs und des Kernels in letzter Schicht	True

Tabelle 4.1: Aufführung der HPs und der zugehörigen Wertebereiche des MLPs.

4.2.2 Binäres FCN

Konform mit dem FCN aus Kapitel 3.3.2 werden auch bei der Implementierung des binären FCNs die namensgebende Faltungen in sogenannte Faltungsblöcke aufgeteilt. Es existieren mindestens drei Faltungblöcke, jedoch kann mithilfe des HPs $i \in \{0, 1, 2, 3, 4\}$ die Tiefe des Netzes durch bis zu vier weitere Blöcke erweitert werden. Ein Block besteht aus einer Faltungsschicht und einer darauffolgenden Batch Normalization [56]. Üblicherweise würde an dieser Stelle eine ReLU-Aktivierungsfunktion (Gleichung 2.4) folgen. In dieser binären Implementierung wird jedoch auf eine zusätzliche Aktivierungsfunktion verzichtet, da sich die Quantisierungsfunktion in der Schicht bereits eine Aktivierungsfunktion beinhaltet.
Sämtliche Faltungsschichten des Modells weisen jeweils $n \in \{32, 64, 128\}$ Filter und eine Kernel Größe von $m \in \{3, 5, 7, 9\}$ auf. Die Stride und das Padding werden auf 1 und 0 bzw. "same" gesetzt, damit die Faltung die Länge der Zeitreihe nicht abwandelt. Die Ausgabeschicht nutzt den Softmax-Klassifikator (Gleichung 2.6) und besteht aus k Neuronen, wobei k die Anzahl an Klassen des Datensatzes ist. Die Ausgabeschicht ist mit einer vorangehenden GAP-Schicht vollständig verbunden.
Wie bei dem binären MLP (4.2.1) werden auch bei dem binären FCN die Lernrate und die Verfallsrate per HP festgelegt (siehe Tabelle 4.2).
Die Abbildung (4.2) stellt die soeben beschrieben Architektur grafisch dar.

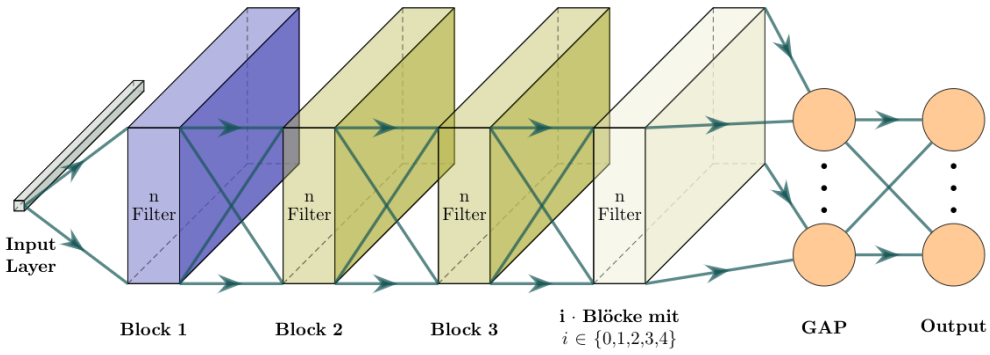

Abbildung 4.2: Grafische Darstellung der implementierten FCN Architektur für TSCs.

Hyperparameter des FCNs	
Funktion	mögliche Werte
Quantisierungsfunktion	SteSign, ApproxSign, SwishSign
Anzahl Filter pro Schicht **[n]**	32, 64, 128
Kernel Größe **[m]**	3, 5, 7, 9
Anzahl zusätzlicher Schichten **[i]**	0, 1, 2, 3, 4
Lernrate	10^{-2}, 10^{-3}, 10^{-4}
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}
Quantisierung des Kernels in erster Schicht	True, False
Quantisierung des Inputs und des Kernels in letzter Schicht	True, False

Tabelle 4.2: Aufführung der HPs und der zugehörigen Wertebereiche des FCNs.

4.2.3 Binäres ResNet

Auch die binäre Implementierung des ResNets orientiert sich überwiegend an der in Kapitel 3.3.3 vorgestellten Architektur. Der Aufbau des Modells kann durch die Abbildung 3.3.3 visuell dargestellt werden. Die beiden letzten Schichten des ResNets sind identisch mit den beiden letzten Schichten des binären FCNs (4.2.2). Es handelt sich erneut um eine GAP-Schicht, die zur Ausgabeschicht vollständig verbunden ist. Die Ausgabeschicht verwendet ebenfalls den Softmax-Klassifikator (Gleichung 2.6) und besteht aus k Neuronen, wobei k die Anzahl an Klassen des Datensatzes ist.
Das restliche Netz bildet sich aus zwei bis fünf Blöcken, abhängig von dem HP $i \in \{0, 1, 2, 3\}$ (siehe zugehörige Tabelle 4.3). Jeder Block umfasst drei Faltungsschichten mit je $n \in \{32, 64, 128, 256\}$ Filtern, die eine Filter Größe von $m \in \{3, 5, 7, 9, 11\}$ haben. Nach jeder Faltungsschicht folgt eine Batch Normalization [56] und normalerweise eine ReLU-Aktivierungsfunktion (siehe Gleichung 2.4). Auf die Aktivierungsfunktion wird in dem

binären Modell verzichtet, da die Quantisierungsfunktion bereits den gewünschten Nutzen abdeckt.

Das Alleinstellungsmerkmal einer ResNet-Architektur ist der Gebrauch von Shortcut-Verbindungen zwischen den Blöcken. In Abbildung 4.3 können die Verbindungspunkte abgelesen werden.

Deckungsgleich mit den beiden anderen Modellen werden auch beim binären ResNet die Lernrate und die Verfallsrate per HP festgelegt (siehe Tabelle 4.3).

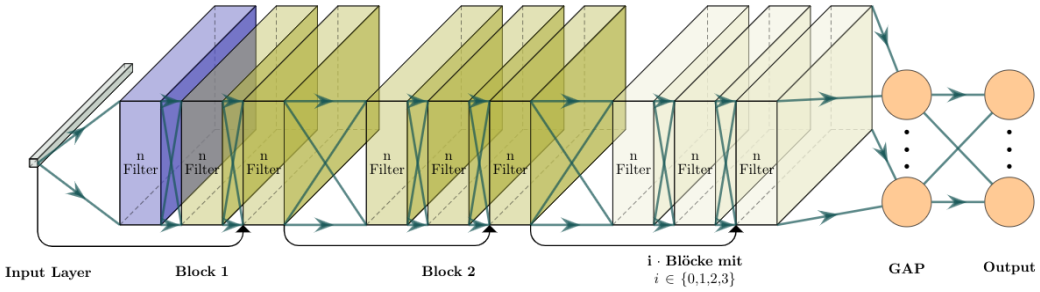

Abbildung 4.3: Grafische Darstellung der implementierten ResNet Architektur für TSCs.

Hyperparameter des ResNets	
Funktion	mögliche Werte
Quantisierungsfunktion	SteSign, ApproxSign, SwishSign
Anzahl Filter pro Schicht **[n]**	32, 64, 128, 256
Kernel Größe **[m]**	3, 5, 7, 9, 11
Anzahl zusätzlicher Blöcke **[i]**	0, 1, 2, 3
Lernrate	10^{-2}, 10^{-3}, 10^{-4}
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}
Quantisierung des Kernels in erster Schicht	True, False
Quantisierung des Inputs und des Kernels in letzter Schicht	True, False

Tabelle 4.3: Aufführung der HPs und der zugehörigen Wertebereiche des ResNets.

4.2.4 Binäres DenseNet

In der Arbeit *"Back to Simplicity: How to Train Accurate BNNs from Scratch?"* von Bethge [17] wird die neue BNN-Architektur BinaryDenseNet vorgeschlagen, die bei der dort durchgeführten Bildklassifikation alle existierenden 1-Bit-CNNs übertrifft. Es ist von großem Interesse den dortigen Erfolg auf die TSC zu übertragen. Daher wird die Dense Convolutional Network, kurz DenseNet, Architektur als BNN auch in dieser Arbeit implementiert. Die in ResNet bereits kennengelernten *Shortcuts* sind essenziell für ein genaues

BNN und verbessern die Genauigkeit durch vermehrtes Vorkommen signifikant [17]. Ein herkömmliches CNN mit L Schichten hat lediglich Verbindung zwischen den einzelnen Schichten, also L Schichten. Das DenseNet hat hingegen bei L Schichten insgesamt $\frac{L(L+1)}{2}$ Verbindungen [61].
Um das Down-Sampling (Pooling) zu vereinfachen, wird das DenseNet überwiegend in Blöcke aufgeteilt, wie bei ResNet. Im Gegensatz zum ResNet werden die *Shortcuts* jedoch nicht zwischen den Blöcken mit der Operation "add", sondern innerhalb eines Blockes mit "concatenate" verknüpft . Die Schichten innerhalb eines Blocks werden konform eines feed-forward Netzes alle miteinander verbunden (siehe Abbildung 4.4). Somit erhält jede Schicht die Featuremaps aller vorherigen Schichten innerhalb des selben Blocks als Eingabe. Durch diese zusätzlichen Verbindungen wird obendrein die Eingabe eines Blocks mit der Ausgabe des Blocks verbunden, wodurch ein freier Informationsfluss im Netz gewährleistet ist [17]. Zwischen den Blöcken befinden sich sogenannte Transition Layers, bestehend aus einer Batch Normalization [56], einer Max-Pooling-Operation (siehe 2.7) als Down-Sampling Algorithmus und einer Faltung.

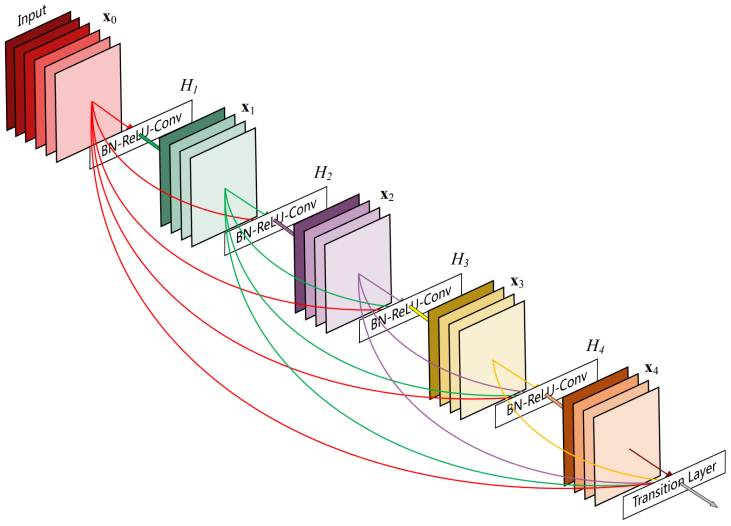

Abbildung 4.4: Grafische Darstellung eines Dense Blocks mit 5 Schichten. Jede Schicht nimmt alle vorherigen Freaturemaps als Eingabe. (Quelle [61])

In Abbildung 4.5 ist die Implementierung der binären DenseNet Architektur dieser Arbeit abgebildet. Gleichwie das binäre ResNet (4.2.3) und das binäre FCN (4.2.2) nutzt das DensNet eine GAP-Schicht und eine Softmax-Schicht (Gleichung 2.6) als letzte beiden Schichten des Netzes. Die Ausgabeschicht ist vollständig mit der vorangehenden GAP-Schicht verbunden und besteht genau aus so viele Neuronen, wie der Datensatz Klassen hat.

Die Architektur besteht aus zwei bis vier Blöcken, abhängig von dem HP $i \in \{1, 2, 3\}$, mit genau drei Schichten pro Block. Obwohl per HP (siehe Tabelle 4.4) auch mehr Schichten pro Block einfach zu konfigurieren wären, wurde dieser Wert festgelegt, damit dem zu hohen Speicherbedarf aus den ersten Auswertungen gegengewirkt wird. Um das volle Potential der $\frac{L(L+1)}{2}$ Verbindungen auszuschöpfen, wären das Testen und Evaluieren mit mehr Schichten pro Block von besonderem Interesse.

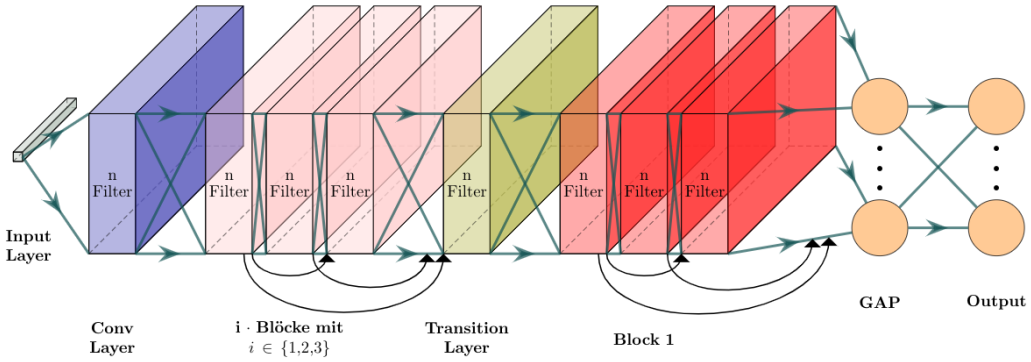

Abbildung 4.5: Grafische Darstellung der implementierten DenseNet Architektur für TSCs.

Jede Faltungsschicht hat $n \in \{32, 64\}$ Filter und eine Kernel Größe von $m \in \{3, 4, 7, 9\}$. Aufgrund des erhöhten Speicherbedarfs in den ersten Auswertungen wird die Kernel Gewichtsmatrix immer quantisiert. Zudem ist die Quantisierungsfunktion mit *SteSign* verankert. Die Lernrate und die Verfallsrate werden wie bei den anderen Modellen auch per HPs bestimmt (siehe Tabelle 4.4).

Hyperparameter des DenseNets	
Funktion	mögliche Werte
Quantisierungsfunktion	SteSign
Anzahl Filter pro Schicht **[n]**	32, 64
Kernel Größe **[m]**	3, 5, 7, 9
Anzahl zusätzlicher Blöcke **[i]**	1, 2, 3
Anzahl Schichten pro Block	3
Lernrate	10^{-3}
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}
Quantisierung des Kernels in erster Schicht	True

Tabelle 4.4: Aufführung der HPs und der zugehörigen Wertebereiche des DenseNets.

Kapitel 5

Experimente und Evaluation

In diesem Kapitel werden die Experimente der Arbeit beschrieben und ausgewertet. Es ist wird versucht folgende Fragen anhand der Experimente zu beantworten.

- **(1)** Sind BNNs geeignet zur TSC?
- **(2)** Falls ja, welche BNN-Architektur eignet sich am besten zur TSC?
- **(3)** In welchem Verhätnis stehen der Speicherbdarf und die Laufzeit gegenüber der Genauigkeit?
- **(4)** Inwiefern kann eine BNN Architektur in diesen drei Kriterien mit state-of-the-art Ansätzen wie HIVE-COTE mithalten?

Das Kapitel 5.1 gibt einen Einblick in die Umgebung in der die Experimente durchgeführt wurden und erläutert die dazu getroffenen Entscheidungen, Kapitel 5.2 gibt ein Überblick in die verwendeten Datensätze und deren Eigenschaften.

5.1 Experimente

Trainingssetup Die Experimente wurden auf dem universellen System für alle KI-Workloads namens *NVIDIA DGX A-100* [62] mit Linux als Betriebssystem ausgeführt. Um die 40 GB VRAM Speicher eines Clusters möglichst effizient zu nutzen, wurde den parallel laufenden Anwendungen immer nur ein Anteil von 4 bis 8 GB zugewiesen. Dadurch konnten mehrere Modelle gleichzeitig trainiert werden.
Als Grundgerüst der Programmierarbeit dient Python als Programmiersprache und die End-to-End-Open-Source-Plattform *TensorFlow* [63] als Framework zur Handhabung von NNs. Um eine möglichst effiziente Binarisierung zu gewährleisten, kam die open-source Deep Learning Bibliothek *Larq* [16] zum Einsatz.
Zum Tracking der Experimente wurde die Open Source-Platform *MLflow* [64] eingesetzt.

Durchführung Für die in dieser Arbeit implementierten Modelle (siehe Kapitel 4.2) wurde eine fixierte Batch Size von 32 festgelegt, da es sich dabei um einen häufig verwendeten Wert handelt. Eine Epoche bedeutete, dass ein gesamter Datensatz durch das NN einmal im Forwardpass und einmal in der Backpropagation durchlaufen wird. Für alle Modelle sind jeweils 2000 Epochen vorgesehen. Wenn sich die Genauigkeit (genau genommen die validiernde Genauigkeit) jedoch über einige Epochen hinweg nicht erhöht wird frühzeitig abgebrochen und die 2000 Epochen nicht mehr durch. Diese 2000 Epochen werden fünf mal ausgeführt, um der Auswirkung der zufälligen Initialisierungen (siehe nächsten Abschnitt) entgegenzuwirken. Dies geschieht indem der Mittelwert der fünf Wiederholungen während der Auswertung berechnet wird.
Die implementierten Modelle aus Kapitel 4.2 verfügen über HPs, die per Random-Search optimiert werden sollen (siehe Kapitel 4.2). Die Optimierung sucht für jeden Durchlauf der fünf Epochensätze eine zufällige Konfiguration der HPs aus. Für jedes Modell werden pro Datensatz acht verschiedene Konfigurationen durchlaufen.
Dieses Vorgehen wird für alle Datensätze durchlaufen. Insgesamt enthält das gesamte UCR/UEA Archiv [24] 128 Datensätze.

Auswirkung der zufälligen Initialisierungen Durch die Initialisierung eines NN werden zufällige Anfangsgewichte gesetzt. Wie es sich in anderen Arbeiten bereits herausgestellt hat, können diese Initialwerte der Gewichte die Genauigkeit des NN für TSC immens beeinflussen [15]. Dementsprechend kann es dadurch vorkommen, dass ein Modell sehr starke Einbuße in der Genauigkeit erleidet, wenn die Anfangswerte der Gewichte während der Initialisierung des Modells ungünstig ausfallen.
Um dieser Auswirkung entgegenzuwirken hilft es, die Initialisierung des Modells mehrfach durchzuführen und das Mittel der Genauigkeit zu wählen.

End-to-End Bei den in Kapitel 4 vorgestellten Modellen handelt es sich ausschließlich um diskriminierende Deep-Learning-Modelle mit End-to-End Ansätzen. Diese wurden gewählt, da sich der End-to-End Ansatz in mehreren Studien als vorteilhaft herausgestellt hat [14]. Dementsprechend werden Ansätze, die Feature-Engineering verwenden, aus dieser Arbeit ausgeschlossen.
Unter Feature-Engineering wird die vorherige Anpassung der Datensätze auf die Eigenschaften des jeweiligen Modells verstanden, mit dem Ziel, die Performance des Models zu verbessern. Die aufwendigen Prozessschritte zur Anpassung der Datensätze müssten zeitaufwändig manuell erstellt werden, spiegeln aber eine zunehmend untergeordnete Rolle. Da außerdem von solchen Anpassungen jeder Deep-Learning Algorithmus profitieren könnte, müsste zusätzlich differenziert werden, ob die Performance durch die Wahl eines geeigneten Klassifikators oder durch das Feature-Engineering beeinflusst wurde.
Letztlich verwenden die Vergleichsmodelle aus der verwandten Arbeit von Hassan Fa-

WAZ [15] ebenfalls ausschließlich diskriminierende Deep-Learning-Modelle mit End-to-End Ansätzen, sodass eine bessere Vergleichbarkeit erreicht wird.

5.2 Datensätze

Damit ein integer Vergleich zwischen den in dieser Arbeit implementierten Modellen möglich ist, werden die Modelle mit Benchmark-Datensätzen trainiert und validiert. Dabei handelt es sich um die Datensätze des öffentlich verfügbaren UCR/UEA Archives [24]. Durch den öffentlichen Zugriff gilt das UCR/UEA Archives heutzutage als der Standard für Benchmarks und Vergleiche von TSC Ansätzen. Da das UCR/UEA Archiv üblicherweise auch für die state-of-the-art Ansätze verwendet wird, stellt es eine geeignete Basis für den Vergleich zwischen den implementierten Modellen der Arbeit und bekannten state-of-the-art Ansätze dar.
Das Archiv wächst durch neu hinzukommende Datensätze kontinuierlich in der Größe. Zum Zeitpunkt der Anfertigung des verwandten Reviews *"Deep learning for time series classification: a review"* (siehe Kapitel 3.2) bestand das Archiv aus 85 univariaten TSC Datensätzen. Diese Menge an Datensätzen ist bis zur Durchführung der Experimente dieser Arbeit auf 128 angestiegen.
Die Datensätze können in sieben unterschiedliche Kategorien eingeteilt werden: ECG, Electric Devices, Image Outline, Motion Capture, Sensor Readings, Simulated Data und Spectrographs [18]. Darüber hinaus decken die verschiedenen Datensätze eine große Spanne an unterschiedlichen Eigenschaften ab. Von Datensatz zu Datensatz weichen Eigenschaften wie die unter anderem für die letzte Schicht eines NN sehr bedeutsame Anzahl an Klassen, die Länge der Zeitreihen oder auch die Größe der Traingssätze gravierend voneinander ab. Auf die möglichen Relationen zwischen dem breiten Spektrum an Eigenschaften der Datensätze und den betrachteten Modellen wird in Kapitel **??** eingegangen.
Da weitgehend alle Zeitreihen in diesem Archiv bereits z-normalisiert sind [18], entfällt diese Anpassung der Daten. Es kann aber durch vereinzelte nicht normalisierte Datensätze (z.B. *Coffee*) oder falsch durchgeführte Normalisierungen (z.B. *ECG200*) zu Anomalien in den Ergebnissen kommen [18].

5.3 Ergebnisse

In diesem Kapitel werden zunächst die Ergebnisse der Experimente vorgestellt und anhand verschiedener Verfahren evaluiert. Vergleiche zwischen implementierten Modellen dieser Arbeit (siehe Kapitel 4.2) und Ansätzen verwandter Arbeiten werden in den Unterkapiteln 5.3.4 und 5.3.6 aufgestellt.

5.3.1 Genauigkeitsvergleich der Modelle

Das signifikanteste Kriterium neuronaler Netze ist die Genauigkeit. In der folgenden Tabelle 5.1 wird ein umfangreicher Überblick über die Genauigkeiten der Modelle auf sämtlichen 128 Datensätzen geliefert. Wie bereits erwähnt wurde, haben die implementierten Modelle aus Kapitel 4.2 durch die Random-Search-Optimierung für jeden Datensatz acht HP-Variationen. Für die Tabelle 5.1 wurde für jeden Datensatz die HP-Variationen ausgewählt, die die höchste Genauigkeit aufweist.

Modelle	**MLP**	**FCN**	**ResNet**	**DenseNet**	**CfMLP**	**CfFCN**	**CfResNet**
Datensätze							
ACSF1	87.24	94.86	97.00	93.72	90.00	90.00	90.00
Adiac	97.33	98.31	98.19	97.78	97.30	97.30	97.30
AllGestureWiimoteX	-	-	90.00	-	-	-	-
AllGestureWiimoteY	-	-	90.00	-	-	-	-
AllGestureWiimoteZ	-	-	90.00	-	-	-	-
ArrowHead	64.61	69.18	72.42	73.94	64.04	64.04	66.67
BME	81.56	91.38	88.00	96.49	66.67	64.44	66.67
Beef	79.60	76.13	77.60	84.00	75.20	77.60	72.80
BeetleFly	70.00	68.00	65.00	72.00	50.00	50.00	50.00
BirdChicken	61.00	89.00	60.00	86.00	50.00	50.00	50.00
CBF	77.16	92.63	94.48	96.30	59.97	66.67	64.41
Car	73.58	74.92	69.50	82.83	75.00	72.33	72.33
Chinatown	70.09	97.67	97.38	94.17	36.44	54.52	63.56
ChlorineConcentration	68.78	74.71	74.93	71.10	67.53	68.40	63.51
CinCECGTorso	67.82	73.14	73.97	77.90	72.48	75.00	72.49
Coffee	57.14	78.57	78.57	66.43	52.14	52.14	47.86
Computers	52.96	74.48	69.92	71.04	50.00	50.00	50.00
CricketX	87.71	93.77	93.94	95.01	91.67	90.29	91.67
CricketY	88.09	93.37	93.38	94.79	91.67	91.67	91.67
CricketZ	88.63	93.55	93.87	95.48	91.67	90.25	91.67
Crop	95.05	97.27	97.64	97.45	95.83	95.83	95.83
DiatomSizeReduction	75.02	76.24	77.88	87.52	71.08	66.93	73.07
DistalPhalanxOutlineAgeGroup	72.42	83.02	80.34	81.87	65.37	65.37	66.24
DistalPhalanxOutlineCorrect	57.75	73.55	73.12	73.12	58.33	55.00	55.00
DistalPhalanxTW	81.03	89.57	89.21	89.90	78.06	79.38	79.23
DodgerLoopDay	-	-	79.64	-	-	-	-
DodgerLoopGame	-	-	50.43	-	-	-	-
DodgerLoopWeekend	-	-	35.65	-	-	-	-
ECG200	77.60	82.00	78.80	84.20	64.00	64.00	58.40
ECG5000	96.55	97.27	97.01	97.44	82.01	81.34	80.67
ECGFiveDays	72.85	77.70	86.92	73.80	49.83	49.94	49.94
EOGHorizontalSignal	87.54	92.60	90.16	92.33	91.67	91.67	91.67
EOGVerticalSignal	87.71	91.96	89.78	91.78	91.67	91.67	91.67
Earthquakes	64.60	74.68	74.82	74.24	-	74.82	74.82
ElectricDevices	80.47	92.10	92.06	91.87	85.71	85.71	85.71
EthanolLevel	75.00	72.30	64.51	73.65	75.00	70.00	69.96
FaceAll	91.57	96.71	97.01	96.52	92.86	92.86	92.86
FaceFour	78.12	83.41	91.82	94.20	72.95	72.95	72.95
FacesUCR	92.22	97.05	97.84	97.73	91.84	92.86	92.86
FiftyWords	97.81	98.46	98.36	98.98	97.70	97.61	98.00
Fish	83.67	98.07	84.11	87.98	83.58	81.60	85.71

Fortsetzung auf der nächsten Seite

Modelle	MLP	FCN	ResNet	DenseNet	CfMLP	CfFCN	CfResNet
Datensätze							
FordA	51.59	90.12	93.41	94.09	50.32	50.32	49.68
FordB	50.59	74.94	81.21	80.49	49.90	50.10	49.90
FreezerRegularTrain	71.93	89.02	68.42	66.27	50.00	50.00	50.00
FreezerSmallTrain	68.84	68.23	62.93	66.72	50.00	50.00	50.00
Fungi	95.88	94.40	95.96	94.26	94.44	93.46	94.44
GestureMidAirD1	-	-	95.44	-	-	-	-
GestureMidAirD2	-	-	95.44	-	-	-	-
GestureMidAirD3	-	-	95.44	-	-	-	-
GesturePebbleZ1	-	-	83.33	-	-	-	-
GesturePebbleZ2	-	-	78.78	-	-	-	-
GunPoint	68.93	98.27	85.07	80.27	49.60	50.13	50.40
GunPointAgeSpan	75.63	98.29	88.61	88.92	50.13	50.13	50.13
GunPointMaleVersusFemale	84.94	97.66	97.09	95.32	52.53	49.49	48.48
GunPointOldVersusYoung	70.73	93.21	76.00	84.76	50.48	51.43	51.43
Ham	54.86	62.29	68.76	62.29	49.71	50.29	50.29
HandOutlines	64.22	67.78	64.19	87.08	52.81	58.43	64.05
Haptics	74.29	78.70	75.55	76.03	77.66	77.66	77.74
Herring	59.38	59.38	59.69	53.12	48.12	51.88	48.12
HouseTwenty	66.05	73.11	91.76	90.76	48.40	54.79	51.60
InlineSkate	80.23	85.27	83.13	84.62	85.71	78.09	79.95
InsectEPGRegularTrain	71.94	74.56	86.13	95.02	66.32	62.86	65.62
InsectEPGSmallTrain	72.77	73.92	75.90	76.81	62.52	65.97	64.07
InsectWingbeatSound	90.91	90.87	89.89	90.83	90.91	90.91	90.91
ItalyPowerDemand	90.36	95.90	95.55	94.13	50.09	50.03	49.91
LargeKitchenAppliances	63.43	82.24	88.02	84.85	66.67	64.44	66.67
Lightning2	60.66	69.51	74.75	71.48	49.18	54.10	52.46
Lightning7	85.24	86.77	88.22	93.15	85.71	80.90	83.56
Mallat	85.62	93.90	83.29	88.87	87.50	83.73	87.50
Meat	67.56	67.56	60.00	68.33	64.44	64.44	64.44
MedicalImages	90.54	93.91	93.92	93.25	90.23	88.43	90.17
MelbournePedestrian	-	-	90.00	-	-	-	-
MiddlePhalanxOutlineAgeGroup	61.04	71.52	70.69	70.82	58.35	58.35	58.35
MiddlePhalanxOutlineCorrect	57.04	75.40	72.99	71.41	51.41	48.59	54.23
MiddlePhalanxTW	84.00	87.58	86.77	87.64	83.33	77.45	81.82
MixedShapesRegularTrain	88.07	96.85	95.93	94.50	80.00	78.16	74.89
MixedShapesSmallTrain	86.02	86.77	80.89	88.81	80.00	72.33	78.16
MoteStrain	74.15	92.44	91.18	87.14	52.35	47.65	52.35
NonInvasiveFetalECGThorax1	97.62	98.92	99.20	97.90	97.62	97.62	97.62
NonInvasiveFetalECGThorax2	97.62	99.13	98.48	98.00	97.62	97.62	97.62
OSULeaf	78.62	96.39	90.22	93.13	81.27	79.15	81.27
OliveOil	74.00	70.00	74.33	72.17	70.67	73.00	74.00
PLAID	-	-	90.91	-	-	-	-
PhalangesOutlinesCorrect	61.38	76.41	72.26	73.99	52.26	56.78	47.74
Phoneme	95.74	97.47	97.24	97.39	97.44	97.44	97.44
PickupGestureWiimoteZ	-	-	85.20	-	-	-	-
PigAirwayPressure	96.60	98.20	97.66	98.10	98.08	98.08	98.08
PigArtPressure	96.57	99.71	99.03	99.38	98.08	98.08	98.08
PigCVP	96.54	98.16	99.06	98.07	98.08	98.08	98.08
Plane	96.30	99.89	99.73	99.59	83.40	83.73	83.73
PowerCons	84.78	87.78	89.56	96.33	50.00	50.00	50.00
ProximalPhalanxOutlineAgeGroup	66.50	89.46	87.22	88.46	66.50	65.40	60.00
ProximalPhalanxOutlineCorrect	63.44	84.74	81.24	78.90	68.38	68.38	68.38
ProximalPhalanxTW	83.35	92.76	92.07	92.24	80.20	82.18	83.33

Fortsetzung auf der nächsten Seite

Modelle	**MLP**	**FCN**	**ResNet**	**DenseNet**	**CfMLP**	**CfFCN**	**CfResNet**
Datensätze							
RefrigerationDevices	60.85	70.33	68.00	68.80	66.67	62.22	62.22
Rock	74.90	71.50	65.85	73.20	69.00	75.00	75.00
ScreenType	63.20	73.47	66.51	63.59	66.67	62.22	57.78
SemgHandGenderCh2	55.80	77.73	85.63	72.90	47.00	41.00	53.00
SemgHandMovementCh2	76.84	83.52	85.67	83.20	83.33	83.33	83.33
SemgHandSubjectCh2	75.64	85.85	90.66	85.50	80.00	77.60	72.80
ShakeGestureWiimoteZ	-	-	82.00	-	-	-	-
ShapeletSim	54.78	75.33	71.89	74.11	50.00	50.00	50.00
ShapesAll	97.81	99.20	99.32	98.93	98.33	98.33	98.33
SmallKitchenAppliances	64.21	85.67	84.85	87.48	66.67	66.67	66.67
SmoothSubspace	75.56	96.80	97.51	97.16	66.67	64.44	66.67
SonyAIBORobotSurface1	63.36	90.82	92.21	92.31	42.93	45.76	48.59
SonyAIBORobotSurface2	76.83	82.50	85.12	83.06	57.02	52.34	47.66
StarLightCurves	84.53	96.42	93.43	94.00	70.79	67.70	67.70
Strawberry	62.59	95.73	94.92	94.11	58.59	52.86	47.14
SwedishLeaf	91.59	98.72	98.80	98.34	93.33	93.33	93.33
Symbols	87.93	83.86	80.40	95.19	78.78	81.16	78.88
SyntheticControl	92.69	99.52	99.73	99.53	81.11	81.11	83.33
ToeSegmentation1	55.61	87.37	90.18	84.82	51.58	50.00	52.63
ToeSegmentation2	59.54	82.92	87.54	70.00	43.69	31.08	56.31
Trace	77.90	99.40	83.40	99.90	75.00	69.80	71.90
TwoLeadECG	56.44	98.31	98.12	94.43	50.01	49.99	49.99
TwoPatterns	88.04	95.73	100.00	100.00	75.00	69.93	75.00
UMD	76.30	96.06	86.85	92.69	64.44	64.44	66.67
UWaveGestureLibraryAll	96.07	93.98	93.16	94.34	87.50	83.77	87.50
UWaveGestureLibraryX	88.41	93.23	93.61	94.36	87.50	87.50	87.50
UWaveGestureLibraryY	86.95	91.30	90.38	92.24	87.50	87.50	85.63
UWaveGestureLibraryZ	87.06	92.35	92.46	93.12	87.50	87.50	85.64
Wafer	90.76	99.77	99.69	99.69	89.21	89.21	89.21
Wine	53.33	54.81	54.07	55.31	50.00	50.00	50.00
WordSynonyms	95.20	96.41	95.89	96.77	95.55	96.00	96.00
Worms	74.81	87.69	86.65	83.77	79.43	79.43	76.47
WormsTwoClass	58.70	71.17	69.35	68.77	51.43	48.57	47.62
Yoga	54.37	69.35	59.98	70.78	50.71	52.14	50.71
Durchschnitt	**76.60**	**86.23**	**84.66**	**86.12**	**70.99**	**70.50**	**70.93**

Tabelle 5.1: Genauigkeiten der Modelle.

5.3.2 Pareto-Effizienz

In diesem Teil der Evaluation der Ergebnisse werden die implementierten Modelle anhand der drei wichtigsten Leistungsmerkmale binärer neuronaler Netze miteinander verglichen. Bei diesen Leistungsmerkmalen handelt es sich um die Genauigkeit, den Speicherbedarf und die Laufzeit der Modelle. Es ist erstrebenswert die Genauigkeit eines Modells zu erhöhen, während der Speicherbedarf, sowie die Laufzeit gesenkt werden. In der Regel konkurrieren diese drei Merkmale jedoch miteinander, da beispielsweise oftmals eine höhere Genauigkeit im Gegenzug einen erhöhten Speicherbedarf oder eine erhöhte Laufzeit bedeutet. Nur durch einen geeignete Kompromiss aus diesen drei Leistungsmerkmalen findet sich das für den Anwendungsfall optimale Modell. Je nach Anwendungsfall kann die Relevanz der ein-

zelnen Merkmale variieren, daher kann es vorteilhaft sein, eine Gewichtung zwischen den Merkmalen vorzunehmen, dies geschieht in Kapitel 5.3.3.
Um das optimale Modell bei mehreren relevanten Leistungsmerkmalen zu finden, eignet sich der Nachweis des *Pareto-Optimums*. Es liegt ein Pareto-Optimum vor, wenn es nicht möglich ist, ein Leistungsmerkmal zu verbessern, ohne dadurch zugleich andere Leistungsmerkmale zu verschlechtern. Bei der Menge aller Pareto-Optima wird von der sogenannten *Pareto-Front* gesprochen. Zur Veranschaulichung des Verfahrens der Pareto-Effizienz wird es im folgenden Abschnitt anhand eines konkreten Beispiels erläutert.

In dem Streudiagramm 5.1 stellen die Punkte beziehungsweise Kreuze Architekturen da. Auf der X-Achse und der Y-Achse sind die Leistungsmerkmale Laufzeit und Genauigkeit vorzufinden. Wie bereits im letzten Absatz erwähnt wurde, ist es erstrebenswert die Genauigkeit zu maximieren, während die Laufzeit minimiert werden soll. Für jeden Punkt existiert ein Bereich, in dem der Punkt alle anderen Punkte in dem Bereich dominiert. Der Bereich wird vom Punkt aus in die entgegengesetzte Richtung der Optimierung, parallel entlang der Achsen, abgegrenzt. Ein Punkt "dominiert" einen anderen Punkt, wenn jedes der Leistungsmerkmale besser ist. Im Falle des braunen Punktes in dem Streudiagramm befindet sich der violette Punkt im exemplarisch bräunlich gefärbten Bereich, somit wird der violette Punkt vom braunen Punkt dominiert. Der braune und der orange Punkt dominieren sich nicht gegenseitig, somit sind sie beide Punkte Pareto-effizienz und gehören zur rot eingezeichneten Pareto-Front. Alle Punkte unter der Pareto-Front werden dominiert, indes Punkte oberhalb der Pareto-Front nicht realisierbar sind.

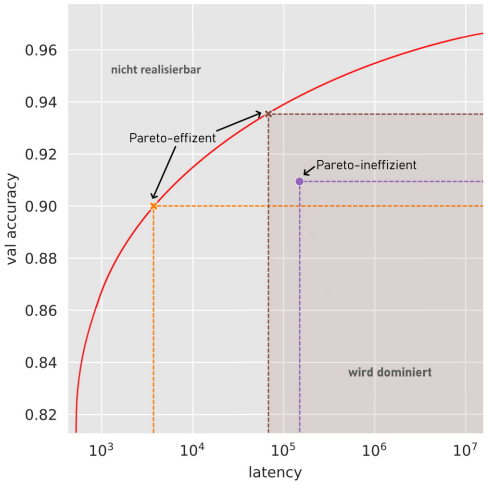

Abbildung 5.1: Beispielhafte Darstellung zur Erläuterung der Pareto-Effizienz

Das Verfahren der Pareto-Effizienz kann auch für die bereits erwähnten drei Leistungsmerkmale angewendet werden, wodurch sich das Prinzip in das dreidimensionale Koordinatensystem verschiebt. Mithilfe dessen wurde für jedes der 128 univariaten Datensätzen des UCR/UEA Archives [24] die Pareto-Optima bestimmt. Zudem wird durch die Überführung der Modelle in ein Koordinatensystem darüber hinaus ein grafischer Vergleich zwischen den Modellen ermöglicht. Zur besseren Lesbarkeit wird das dreidimensionale Koordinatensystem in zwei zweidimensionale Koordinatensysteme aufgeteilt und der Speicherbedarf,

sowie die Laufzeit logarithmisch dargestellt. Die jeweils zugehörigen Punkte können über die Genauigkeitswerte auf der Y-Achse zugeordnet werden.

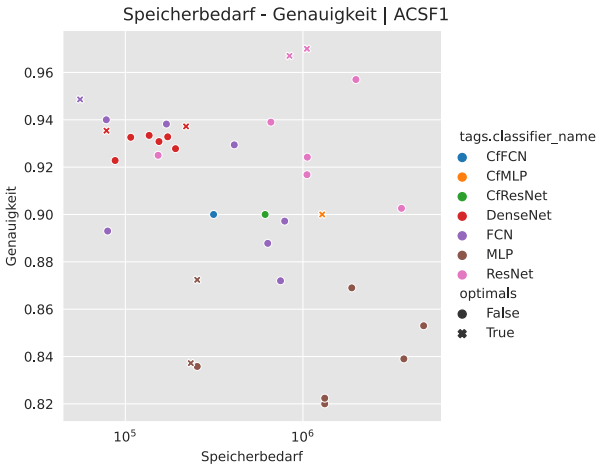

Abbildung 5.2: Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und Pareto-Effizienz für den Datensatz ACSF1.

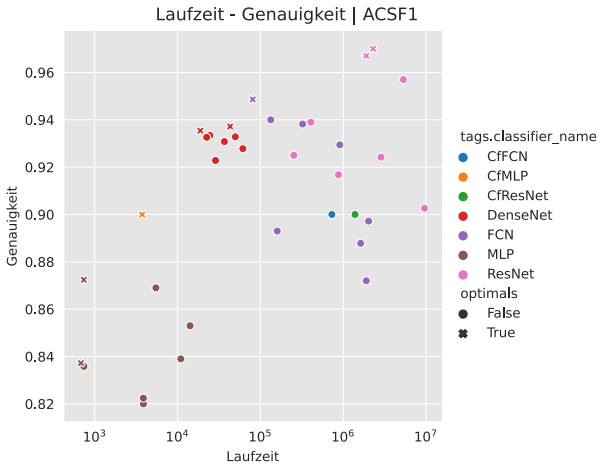

Abbildung 5.3: Streudiagramm mit Genauigkeit in Prozent und Laufzeit in μs auf den Achsen und Pareto-Effizienz für den Datensatz ACSF1.

In den Streudiagrammen 5.2 und 5.3 wurden die Ansätze auf dem ACSF1 Datensatz trainiert. In der Legende werden den Ansätzen verschiedene Farben zugeordnet. Die Ansätze MLP, FCN, ResNet und DenseNet haben durch die Verwendung der Random-Search-Optimierung 4.2 jeweils acht unterschiedliche Architekturen. Die regulären Punkte werden von den als Kreuz gekennzeichneten Pareto-Optima dominiert.
Bereits der erste Datensatz lässt auf mutmaßliche Eigenschaften der Ansätze deuten, die sich im späteren Verlauf auch erweisen (siehe Kapitel 5.3.4). Wie in dem Streudiagramm 5.3 zu erkennen ist, tendieren die MLP-Architekturen zu einer verhältnismäßig geringen Laufzeit, jedoch spiegelt sich das in einer unterdurchschnittlichen Genauigkeit wieder. Die geringe Laufzeit einer MLP-Architektur spiegelt sich unter anderem in der CfMLP-Architektur (orange markiert) als Pareto-Optimum wieder.

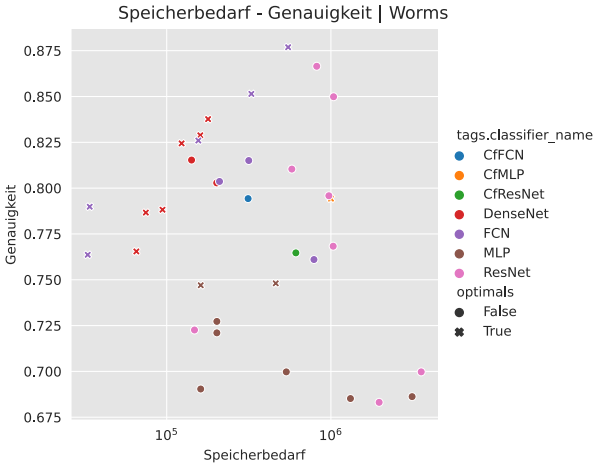

Abbildung 5.4: Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und Pareto-Effizienz für den Datensatz Worms.

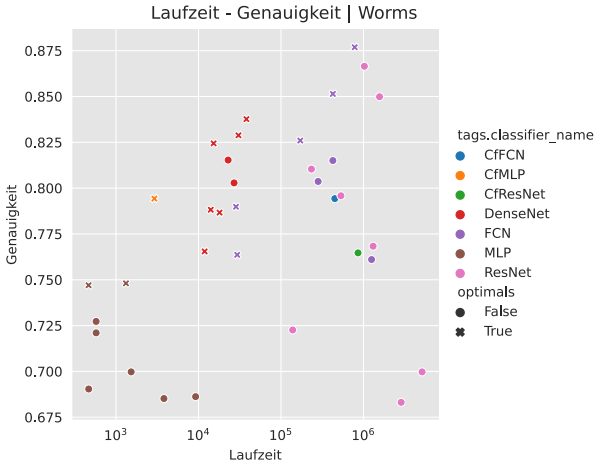

Abbildung 5.5: Streudiagramm mit Genauigkeit in Prozent und Laufzeit in μs auf den Achsen und Pareto-Effizienz für den Datensatz Worms.

Die ResNet-Architekturen bilden im Vergleich zu den MLP-Architekturen das Gegenstück, da sie hingegen eine im Durchschnitt sehr hohe Genauigkeit, jedoch einen auch einen hohen Ressourcenverbrauch aufweisen. Das obere Mittelfeld teilen sich die DenseNet- und FCN-Architekturen, wobei DenseNet etwas ressourcenschonender und präziser ist. Die in Kapitel 4.2 erwähnten Einschränkungen im Wertebereich der HPs der DenseNet-Architekturen macht sich in dem geringeren Streukreis der roten Punkte bemerkbar.

Ebenfalls für die Werte in den Streudiagrammen 5.4 und 5.5 trifft diese Begutachtung zu. Die Menge an Pareto-Optima (gekennzeichnet durch Kreuze) besteht zum großen Teil aus Architekturen mit geringem Ressourcenverbrauch/Kosten, jedoch einer verhältnismäßig niedrigen Genauigkeit. Die erneute Pareto-Effizienz des CfMLPs (orange markiert) bekräftigt diese Beobachtung. Obgleich die Relevanz der Leistungsmerkmale je nach Anwendungsfall variiert kann, ist die Genauigkeit eines Ansatzes in der Regel der wichtigste Faktor. Um der Genauigkeit zu einer Priorisierung zu verschaffen, wird im Kapitel 5.3.3 eine Gewichtung im Pareto-Effizienz-Verfahren vorgenommen.

Ausgang der Pareto-Effizienz Um einen Vergleich der Ansätze basierend auf alle 128 Datensätze zu ermöglichen, wird für jeden Datensatz notiert, ob die jeweiligen Ansätze mit einer ihrer verschiedenen Architekturen in der Pareto-Front liegen. Damit die unterschiedlich großen Wertebereiche der HPs der implementierten Modelle keine negativen Einwirkungen in dem Vergleich haben, wird für jeden Ansatz immer nur die beste Architektur gewählt. Zudem entsteht durch die Auswahl nur einer Architektur kein Nachteil für die Vergleichsansätze CfMLP, CfFCN und CfResNet, die jeweils lediglich nur eine Architektur haben. Somit fließt das vermehrte Vorkommen von unterschiedlichen Architekturen eines

Ansatzes in der Pareto-Front eines Datensatzes nicht in die resultierende Abbildung 5.6 mit ein.

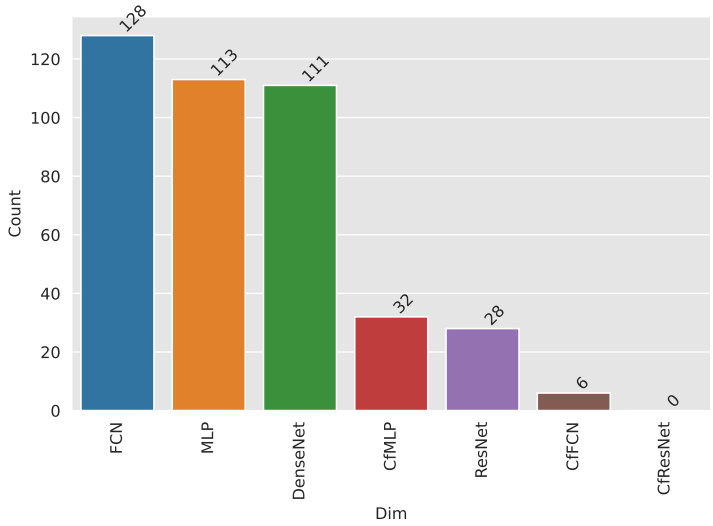

Abbildung 5.6: Balkendiagramm des Pareto-Effizienz-Vergleichs zwischen den Modellen, basierend auf alle 128 Datensätze.

Das Balkendiagramm 5.6 führt den FCN-Ansatz als geeignetsten auf. Mit einer Anzahl von 128 notierten Pareto-Optima ist für jeden der 128 Datensätze mindestens eine der FCN-Architekturen Pareto-effizient. Die Ansätze MLP und DenseNet schneiden mit großem Abstand vor den restlichen Modellen in etwa gleich ab. Dieses Ergebnis lässt sich durch die zuvor aufgestellten Deutungen der Streudiagramme 5.2 und 5.3 erklären. Während sich DenseNet durch die durchwegs überdurchschnittliche Genauigkeit und die moderaten Kosten (Speicherbedarf, Laufzeit) durchsetzt, punktet MLP überwiegend durch die verhältnismäßig sehr geringe Laufzeiten und weniger durch erhöhte Genauigkeiten. Letztere Eigenschaften treffen ebenfalls für das Vergleichmodell CfMLP zu, während die anderen Vergleichmodelle keine dermaßen geringe Laufzeit aufweisen und somit schwächer abschneiden. Im Gegensatz zu den anderen implementierten Ansätzen (siehe Kapitel 4.2) schneidet ResNet selbst schwächer als das Vergleichmodell CfMLP ab. Dies lässt sich trotz der verhältnismäßig hohen Genauigkeit auf die erhöhten Kosten zurückführen.

5.3.3 Pareto-Effizienz mit Gewichtungen

Da die Genauigkeit verglichen zu den Kosten in den meisten Anwenungsfällen der TSC eine erhöhte Relevanz hat, wird in diesem Kapitel eine Gewichtung für das Pareto-Effizienz-Verfahren eingeführt. Mit den drei Leistungsmerkmalen Genauigkeit, Speicherbedarf und Laufzeit wurde das Verfahren bisher für einen dreidimensionalen Raum durchgeführt. Das führt dazu, dass eine Architektur auch z.B. dann als Pareto-effizient gewertet wird, wenn

sie mit besonders wenig Speicherbedarf oder Laufzeit (Inputs) nur eine unbrauchbare Genauigkeit (Output) erreicht. Um dieses Problem auszuschließen, werden hier die beiden Inputs (oder "Kosten") mit dem erreichten Output in Beziehung gesetzt beziehungsweise gewichtet. Mit dem Ansatz, dass eine Genauigkeit von 90% doppelt so gut ist wie eine Genauigkeit von 80%, wird für die Gewichtung die verbleibende Ungenauigkeit, also 10% beziehungsweise 20% verwendet, und da diese Ungenauigkeit für eine bessere Effizienz ebenso minimiert werden soll, wie Speicherbedarf oder Laufzeit, werden für die Bestimmung des Pareto-Optimums jeweils die Produkte von Speicherplatz und Ungenauigkeit, sowie von Laufzeit und Ungenauigkeit verwendet. Der Bezug auf die (Un-)Genauigkeit wird hier in Anlehnung an den Effizienzvergleich in einer Data Envelopment Analysis (DEA) verwandt. Bei diesem Verfahren aus dem Bereich des Operations Research werden Unternehmen oder dergleichen anhand ihrer Outputs (Menge an Produkten etc., hier Genauigkeit) und der dazu benötigten Inputs (Material, Arbeitszeit, Kosten etc., hier Speicherbedarf und Laufzeit) verglichen, ohne dass dafür ein mathematischer Zusammenhang zwischen den Inputs und den Outputs notwendig wäre [65]. Auf unsere Anwendung übertragen ergibt sich hier die Betrachtung, die erreichte Genauigkeit der Laufzeit und dem Speicherplatz gegenüberzustellen.
Zudem weiteren Gewichtung werden der Speicherbedarf und die Laufzeit nicht nur wie bisher in den Abbildungen logarithmisch dargestellt, sondern vor der Eingabe auf mit der Basis 10 logarithmiert.

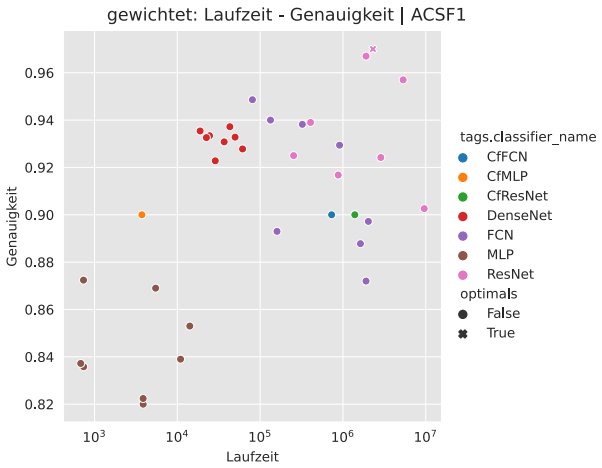

Abbildung 5.7: Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz ACSF1.

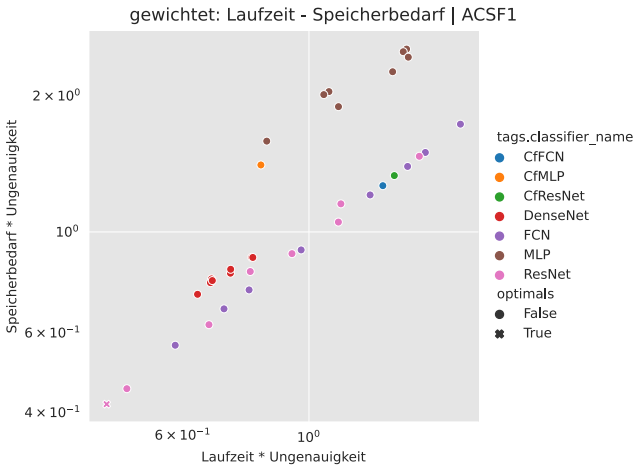

Abbildung 5.8: Streudiagramm mit $Speicherbedarf \cdot Ungenauigkeit$ und $Laufzeit \cdot Ungenauigkeit$ auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz ACSF1.

Wie anhand der ResNet-Architektur in dem Streudiagramm 5.7 zu sehen ist, werden mit dem gewichteten Verfahren Architekturen mit hoher Genauigkeit präferiert. Das aus dem

Multiplizieren der Eingaben mit der Ungenauigkeit entstehende Verfahren kann anhand des Streudiagramms 5.8 nachvollzogen werden. Die Ermittlung der Pareto-Optima findet in Diagramm 5.8 in Richtung der Minimierung der beiden Kosten statt und bestimmt durch die veränderte Anordnung der Punkte insgesamt weniger Pareto-Optima.

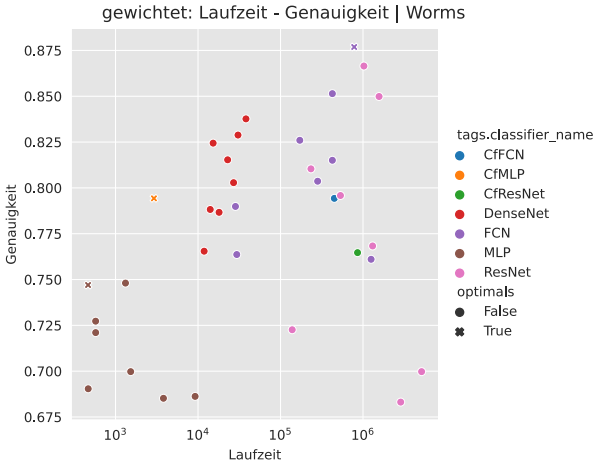

Abbildung 5.9: Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz Worms.

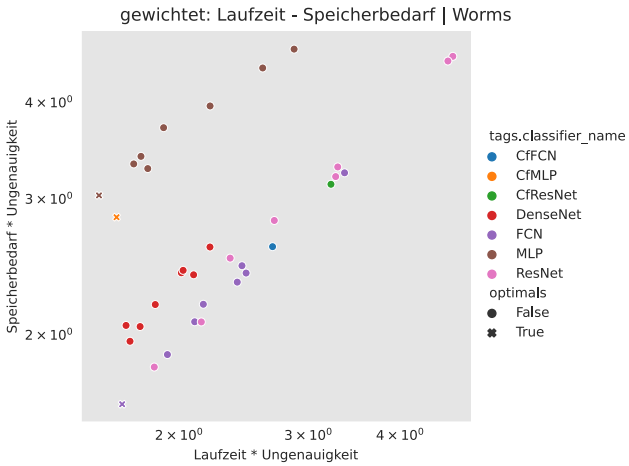

Abbildung 5.10: Streudiagramm mit $Speicherbedarf \cdot Ungenauigkeit$ und $Laufzeit \cdot Ungenauigkeit$auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz Worms.

Die beiden Streudiagramme 5.9 und 5.10 des Datensatzes Worms zeigen jedoch beispielhaft, dass trotz der stärkeren Gewichtung der Genauigkeit auch ressourcenschonende Ansätze wie MLP und CfMLP Pareto-optimal sein können.

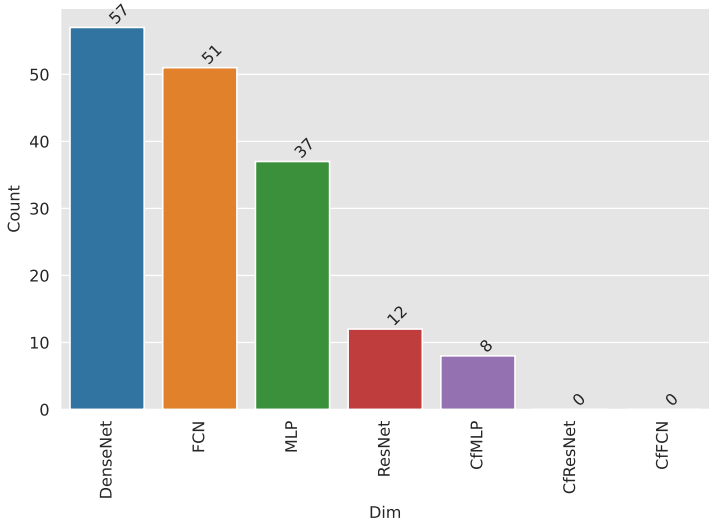

Abbildung 5.11: Balkendiagramm des gewichteten Pareto-Effizienz-Vergleichs zwischen den Modellen, basierend auf alle 128 Datensätze.

Im erneuten Vergleich der Ansätze (siehe Balkendiagramm 5.11 basierend auf alle 128 Datensätze hat sich die Verteilung zugunsten der präziseren und zugleich komplexeren Ansätze verschoben. Mit einer Pareto-Effizienz in 57 der 128 Datensätze überragt das komplexeste Modell DenseNet die anderen Ansätze.

5.3.4 Vergleich mit verwandter Arbeit

Wie bereits in Kapitel 3.4 erwähnt wurde, baut diese Arbeit auf dem Review von Hassan Fawaz [15] auf. In diesem Kapitel wird ein Vergleich zwischen den full-precision Modellen des Reviews [15] und der in dieser Arbeit implementierten Modellen aufgestellt. Zu diesem Zweck wurden die full-precision Modellen des Reviews [15] unter den selben Voraussetzungen und dem selben System (*NVIDIA DGX A-100* [62]) wie die implementierten Modelle aus Kapitel 4.2 trainiert und getrackt. Da in dem Review keine DenseNet-Archtiektur implementiert wurde, wird diese nicht in den Vergleich mit einbezogen. Erneut werden die drei Leistungsmerkmale Genauigkeit, Speicherbedarf und Laufzeit herangezogen. Damit bei dem Vergleich die Leistungsmerkmale von der selben Architektur und nicht von mehreren Variation mit anderen HPs stammen, wird vorab für jeden Ansatz die jeweilig günstigste Architektur ermittelt. Zu diesem Zweck wird erneut das gewichtete Pareto-Effizienz-Verfahren aus Kapitel 5.3.3 angewendet. Im Gegensatz zur Optimierung in dem Review [15] wird das Modell nicht auf allen Datensätzen gleichzeitig, sondern explizit für jeden Datensatz einzeln optimiert. Um für jeden Ansatz die günstigste Architektur zu ermitteln, wird somit bei jedem Datensatz für jeden Ansatz isoliert das gewichtete Pareto-Effizienz-Verfahren durchgeführt.

In den folgenden Tabellen (5.2 - 5.7) werden die ausgewählten Architekturen der in dieser Arbeit implementierten Ansätze in Relation mit den full-precision Modellen des Reviews [15] gesetzt. Die Werte der jeweiligen Architektur werden in der ersten Reihe der jeweiligen Tabelle in Prozent für die Genauigkeit, in Mikrosekunden (μs) für die Laufzeit und in byte für den Speicherbedarf angegeben. Zur Übersichtlichkeit handelt es sich bei den Werten um die Durchschnittswerte aller Datensätze (im Anhang befinden sich die zugehörigen Tabellen mit den Verhältnissen bei den einzelnen Datensätzen). In der zweiten Reihe werden die Leistungsmerkmale der jeweiligen Architektur mit dem gegenüberstehenden full-precision Modell in Prozenten verglichen, wobei 100% dem Wert des full-precision Modells entspricht. In gleicher Weise wird in der letzten Zeile das Verhältnis als Bruch angegeben. Im Nenner des Bruches kann man den Faktor der Verkleinerung ablesen.

MLP:

	Genauigkeit	Laufzeit	Speicher
Werte	76.31	443.26	17278.51
Prozent	108.19 %	18.84 %	0.56 %
Bruch	0.92^{-1}	5.31^{-1}	178.98^{-1}

Tabelle 5.2: Verhältnis des MLPs zum full-precision-CfMLP.

CfMLP:

	Genauigkeit	Laufzeit	Speicher
Werte	70.99	2480.25	102504.15
Prozent	100.64 %	105.44 %	3.31 %
Bruch	0.99^{-1}	0.95^{-1}	30.17^{-1}

Tabelle 5.3: Verhältnis des CfMLPs zum full-precision-CfMLP.

FCN:

	Genauigkeit	Laufzeit	Speicher
Werte	85.34	139474.87	25353.59
Prozent	108.59 %	52.38 %	2.38 %
Bruch	0.92^{-1}	1.91^{-1}	41.94^{-1}

Tabelle 5.4: Verhältnis des FCNs zum full-precision-CfFCN.

CfFCN:

	Genauigkeit	Laufzeit	Speicher
Werte	70.5	281675.74	39214.69
Prozent	89.7 %	105.78 %	3.69 %
Bruch	1.11^{-1}	0.95^{-1}	27.12^{-1}

Tabelle 5.5: Verhältnis des CfFCNs zum full-precision-CfMLP.

ResNet:

	Genauigkeit	Laufzeit	Speicher
Werte	84.46	317892.89	95250.24
Prozent	104.72 %	62.52 %	4.71 %
Bruch	0.95^{-1}	1.6^{-1}	21.21^{-1}

Tabelle 5.6: Verhältnis des ResNets zum full-precision-CfMLP.

CfResNet:

	Genauigkeit	Laufzeit	Speicher
Werte	70.93	542705.93	76534.69
Prozent	87.94 %	106.73 %	3.79 %
Bruch	1.14^{-1}	0.94^{-1}	26.4^{-1}

Tabelle 5.7: Verhältnis des CfResNets zum full-precision-CfMLP.

Gemäß der Tabellen 5.5 und 5.7 nimmt von den full-precision Vergleichsmodellen zu den binarisierten Vergleichsmodellen die Genauigkeit ab, während die Laufzeit in etwa gleich bleibt und der Speicherbedarf drastisch sinkt. Zwischen dem full-precision-CfMLP und dem CfMLP stimmen die Genauigkeiten jedoch annähernd überein (siehe Tabelle 5.3), das mag am schlechten Abschneiden der MLP-Architektur im Review [15] liegen. Der Speicherbedarf reduziert sich bei den Vergleichsmodellen um einen Faktor von knapp unter 32. Durch die Binarisierung des Modells werden die *32-bit-floating-Point* Gewichte und Aktivierungen zu binären Zahlen umgewandelt und reservieren daher nur noch ein Zweiunddreißigstel der bits. Dass der Faktor etwas unter 32 liegt, kann durch das Belassen der vollen Auflösung der Eingabe in die erste Schicht begründen werden (siehe Kapitel 4.1).

Die Relationen zwischen den gewählten Architekturen der in dieser Arbeit implementierten Ansätze (MLP, FCN, ResNet) und den full-precision Vergleichsmodellen weisen alle eine Verbesserung in der Genauigkeit auf und zugleich eine zum Teil drastische Senkungen der Kosten (siehe Tabellen 5.2, 5.4, 5.6).
Im Vergleich zum (binären) CfResNet hat das ResNet einen erhöhten Speicherbedarf, was

darauf schließen lässt, dass die ResNet-Architektur durch ausgewählte HPs an Tiefe beziehungsweise Komplexität gewonnen hat. Der Speicherbedarf der MLP-Architektur wurde um einen erheblichen Faktor von 178.98 verkleinert. Im Vergleich zum bereits stark reduzierten Speicherbedarfs des CfMLPs benötigt die MLP-Architektur ungefähr nur ein Sechstel, während auch die Laufzeit gesenkt wird und die Genauigkeit steigt. Eine mögliche Erklärung für diese extremen Werte mag die Wahl geeigneter HPs sein. Beispielsweise ist die Anzahl an Neuronen pro Schicht für die CfMLP-Architektur auf genau 500 festgelegt, demgegenüber ist bei dem implementierten MLP-Modell die Anzahl an Neuronen durch den HP $n \in \{128, 256, 512, 1024\}$ variabel (siehe Kapitel 4.2.1). Der Wert 128 ergibt sich als geeignetsten für den HP n, wodurch sich der Speicherbedarf des MLPs senken würde. In Kapitel 5.3.5 wird auf weitere per Random-Search-Optimierung ermittelte geeignete HPs eingegangen.

In der folgenden Tabelle 5.8 wird die Genauigkeit zwischen den obig ausgewählten Architekturen der implementierten Modelle aus Kapitel 4.2 und den full-precision-Vergleichsmodellen aus dem Review von Hassan Fawaz [15] anhand der Differenz für alle Datensätze tabellarisch aufgeführt. Ein positiver Wert entspricht einer Verbesserung des implementierten Modells gegenüber des full-precision-Vergleichsmodells.

	MLP	FCN	ResNet	CfMLP	CfFCN	CfResNet
tags.dataset						
ACSF1	31.44	5.06	5.40	34.20	0.20	-1.60
Adiac	58.20	14.17	14.86	58.17	13.15	13.97
ArrowHead	-13.79	-15.16	-11.35	-14.36	-20.30	-17.10
BME	-8.98	7.78	-11.87	-23.87	-19.16	-33.20
Beef	8.27	8.13	2.27	3.87	9.60	-2.53
BeetleFly	-27.00	-23.00	-20.00	-38.00	-41.00	-35.00
BirdChicken	-13.00	-5.00	-29.00	-24.00	-44.00	-38.00
CBF	-11.74	-6.75	-5.10	-26.94	-32.71	-35.16
Car	-5.58	-16.42	-24.08	-3.33	-19.00	-19.33
Chinatown	-17.08	-0.29	-0.47	-50.73	-43.44	-34.29
ChlorineConcentration	-11.22	-7.13	-10.34	-12.46	-13.25	-21.77
CinCECGTorso	-15.94	-9.74	-9.81	-11.28	-7.88	-11.30
Coffee	-42.14	-21.43	-21.43	-47.14	-47.86	-52.14
Computers	-2.88	-7.44	-10.64	-5.84	-31.92	-30.56
CricketX	28.19	14.21	13.82	32.54	10.85	11.77
CricketY	28.25	13.08	11.39	31.82	12.38	10.64
CricketZ	25.70	12.52	12.82	28.74	9.27	10.79
Crop	33.28	23.45	23.26	34.06	22.02	21.54
DiatomSizeReduction	-17.24	41.60	47.81	-19.84	32.29	43.01
DistalPhalanxOutlineAgeGroup	7.67	9.93	6.86	0.62	-6.43	-5.56
DistalPhalanxOutlineCorrect	-16.96	-5.00	-3.91	-14.35	-21.01	-22.03
DistalPhalanxTW	20.02	19.98	22.66	17.05	9.88	12.90
ECG200	-13.80	-6.80	-10.00	-27.40	-24.80	-29.00
ECG5000	3.58	2.83	2.97	-10.96	-12.66	-12.84
ECGFiveDays	-24.44	-26.78	-9.71	-47.46	-48.59	-46.69

Continued on next page

tags.dataset	MLP	FCN	ResNet	CfMLP	CfFCN	CfResNet
EOGHorizontalSignal	44.24	34.41	30.22	48.46	35.20	31.72
EOGVerticalSignal	45.91	45.67	44.03	49.90	47.03	47.14
Earthquakes	-8.06	2.01	3.45	-	2.30	3.60
ElectricDevices	21.10	21.45	19.09	26.46	15.07	12.93
EthanolLevel	36.40	18.02	-13.44	36.40	21.60	-5.88
FaceAll	12.18	2.52	10.05	13.47	-0.89	6.18
FaceFour	-6.76	-9.72	-3.64	-10.68	-20.00	-22.50
FacesUCR	9.07	2.71	2.41	8.70	-1.48	-2.57
FiftyWords	26.99	33.50	24.33	26.89	33.04	23.98
Fish	-1.13	1.96	-13.94	-1.22	-14.51	-12.34
FordA	-30.67	-1.29	-0.29	-31.30	-41.09	-44.02
FordB	-20.07	-2.77	-1.46	-20.77	-27.14	-31.41
FreezerRegularTrain	-18.67	-10.66	-32.03	-40.60	-49.68	-49.85
FreezerSmallTrain	-0.55	-4.82	-20.29	-18.58	-18.31	-33.22
Fungi	9.21	92.57	78.22	8.10	91.64	76.70
GunPoint	-23.87	-1.73	-14.00	-43.20	-49.87	-48.67
GunPointAgeSpan	-17.78	-1.27	-11.08	-43.29	-49.43	-49.56
GunPointMaleVersusFemale	-13.10	-2.03	-2.15	-45.51	-50.19	-50.76
GunPointOldVersusYoung	-23.37	-5.65	-22.92	-43.62	-47.43	-47.49
Ham	-15.05	-15.05	-7.05	-20.19	-20.38	-25.52
HandOutlines	-27.19	-15.08	-	-38.59	-21.46	-27.30
Haptics	31.75	28.23	24.57	35.13	28.70	26.77
Herring	10.31	-5.00	-0.62	-0.94	-12.50	-11.88
HouseTwenty	-9.92	-25.04	-	-25.04	-43.36	-46.72
InlineSkate	46.77	51.34	-	52.26	44.92	42.24
InsectEPGRegularTrain	7.18	-27.74	-13.63	1.74	-37.05	-34.14
InsectEPGSmallTrain	10.04	45.30	38.71	-0.21	44.12	26.88
InsectWingbeatSound	30.48	50.31	39.96	30.48	51.72	40.98
ItalyPowerDemand	-4.98	-0.39	-0.60	-45.25	-46.26	-46.24
LargeKitchenAppliances	16.44	-8.05	-2.12	19.68	-25.85	-23.47
Lightning2	-7.54	-3.93	-3.28	-19.02	-19.34	-25.57
Lightning7	23.60	3.41	5.48	24.07	-1.57	0.82
Mallat	-6.66	-2.81	-14.08	-4.77	-12.98	-9.86
Meat	-21.78	-20.33	-39.00	-24.89	-15.89	-34.56
MedicalImages	17.94	16.07	16.94	18.36	10.59	13.20
MiddlePhalanxOutlineAgeGroup	6.23	16.28	16.15	6.15	4.85	3.81
MiddlePhalanxOutlineCorrect	-18.42	-6.53	-9.76	-24.05	-30.86	-28.38
MiddlePhalanxTW	30.37	35.95	37.29	29.70	27.32	32.34
MixedShapesRegularTrain	-2.60	1.36	-1.36	-10.67	-17.33	-22.40
MixedShapesSmallTrain	1.92	-3.48	-10.76	-4.11	-17.01	-13.50
MoteStrain	-11.36	-1.13	-1.21	-33.16	-45.93	-40.05
NonInvasiveFetalECGThorax1	6.14	3.09	5.06	6.14	1.79	3.48
NonInvasiveFetalECGThorax2	5.78	3.83	4.10	5.78	2.31	3.26
OSULeaf	22.67	-1.46	-7.80	25.32	-18.71	-16.75
OliveOil	8.67	-8.33	-13.00	5.33	1.00	-10.67
PhalangesOutlinesCorrect	-14.20	-7.16	-14.43	-23.31	-24.99	-36.78
Phoneme	86.32	64.63	63.89	88.02	64.64	64.09
PigAirwayPressure	89.99	80.87	-	91.54	80.87	57.50
PigArtPressure	86.09	1.06	-	87.60	-0.58	-1.06
PigCVP	88.94	14.99	-	90.48	15.00	6.25
Plane	-1.41	-0.11	-0.27	-14.31	-16.27	-16.27
PowerCons	-12.89	-0.89	1.67	-47.67	-36.33	-37.89
ProximalPhalanxOutlineAgeGroup	-18.37	5.37	0.52	-18.37	-17.14	-24.68

Continued on next page

tags.dataset	MLP	FCN	ResNet	CfMLP	CfFCN	CfResNet
ProximalPhalanxOutlineCorrect	-9.55	-5.91	-11.27	-4.60	-22.27	-23.57
ProximalPhalanxTW	6.67	15.50	14.80	3.51	6.08	6.07
RefrigerationDevices	23.15	17.08	14.99	28.96	12.57	9.21
Rock	-10.30	7.30	9.00	-16.20	11.80	19.80
ScreenType	22.99	10.42	4.96	26.45	0.04	-3.77
SemgHandGenderCh2	-26.43	-3.83	3.27	-35.23	-40.57	-29.37
SemgHandMovementCh2	33.33	35.79	41.76	39.82	35.73	39.42
SemgHandSubjectCh2	-6.10	9.72	16.79	-1.73	3.38	-1.07
ShapeletSim	1.00	4.78	-6.33	-1.33	-20.56	-28.22
ShapesAll	20.17	9.80	6.69	20.70	8.93	5.70
SmallKitchenAppliances	26.19	7.96	6.72	28.64	-11.04	-11.47
SmoothSubspace	-22.44	-0.67	-0.49	-31.33	-33.02	-31.33
SonyAIBORobotSurface1	-5.86	-5.02	-3.86	-26.29	-50.08	-47.49
SonyAIBORobotSurface2	-9.25	-20.21	-12.42	-26.11	-45.69	-49.88
StarLightCurves	-10.46	-0.08	-3.80	-24.21	-28.80	-29.53
Strawberry	-33.30	-1.78	-3.08	-37.30	-44.65	-50.86
SwedishLeaf	6.49	1.90	2.55	8.79	-3.40	-2.92
Symbols	4.12	-11.96	-8.90	-4.85	-14.38	-10.43
SyntheticControl	-4.64	0.66	0.07	-16.22	-17.76	-16.33
ToeSegmentation1	-3.60	-8.77	-5.53	-7.28	-46.14	-43.07
ToeSegmentation2	-14.92	-6.00	-1.85	-30.77	-57.85	-33.08
Trace	-2.70	-0.60	-16.60	-5.60	-30.20	-28.10
TwoLeadECG	-18.86	-1.63	-1.88	-25.29	-49.96	-50.01
TwoPatterns	-6.77	8.68	0.00	-19.81	-17.12	-25.00
UMD	-18.56	-2.69	-13.89	-30.42	-34.31	-32.36
UWaveGestureLibraryAll	0.69	12.19	7.07	-7.87	1.97	1.42
UWaveGestureLibraryX	11.62	17.32	15.49	10.71	12.12	9.38
UWaveGestureLibraryY	17.06	26.30	23.80	17.61	23.25	19.05
UWaveGestureLibraryZ	17.36	19.14	17.60	17.80	14.83	10.78
Wafer	-8.83	0.05	-0.12	-10.37	-10.51	-10.59
Wine	-0.74	-11.11	-22.22	-4.07	-11.11	-22.22
WordSynonyms	35.30	39.58	34.22	35.68	39.89	34.34
Worms	28.99	9.51	10.55	33.71	1.25	0.36
WormsTwoClass	-2.08	-5.45	-8.31	-9.35	-25.71	-27.19
Yoga	-31.61	-14.37	-26.98	-34.91	-31.58	-35.96
Durchschnitt	**4.49**	**5.84**	**2.44**	**-0.83**	**-9.00**	**-10.80**

Tabelle 5.8: Vergleich der Genauigkeit zwischen den implementierten Modellen und den full-precision-Vergleichsmodellen aus dem Review von Hassan Fawaz [15].

Anhand der Tabelle 5.8 wird erneut deutlich, dass die implementierten Modelle aus Kapitel 4.2 an Präzision dazu gewinnen, während die binarisierten Vergleichsmodelle an Genauigkeit einbüßen.

5.3.5 Geeignete Hyperparameter

In Kapitel 4.2 wurden die Hyperparameter und die zugehörigen Wertebereiche für die in dieser Arbeit implementierten Ansätze mithilfe der Tabellen 4.1 - 4.4 vorgestellt. Diese Tabellen werden in diesem Abschnitt der Evaluation wieder aufgegriffen und ergänzt (siehe Tabellen 5.9 - 5.12). Durch das vorherige Kapitel 5.3.4 wurde das gewichtete Pareto-

Effizienz-Verfahren bereits bei jedem Datensatz für jeden Ansatz isoliert durchgeführt. Es handelt sich um die isolierte Durchführung des Verfahrens, da zur Ermittlung geeigneter HPs lediglich der Vergleich innerhalb eines Ansatzes, zwischen den unterschiedlichen HP-Variationen, sinnig ist.
Für jeden HP wird 1024 mal ein Wert per Random-Search-Optimierung ausgewählt, da es für alle 128 Datensätze jeweils 8 Variationen gibt. Da die Random-Search-Optimierung zufällig vorgeht, kann nicht garantiert werden, dass die Werte in einem Wertebereich gleich oft erprobt werden. Vereinzelt kann es sogar vorkommen, dass Werte so selten getestet werden, dass die Grundlage für eine Beurteilung der Resultate nicht genügt. Aus diesem Grund werden Werte die weniger als 128 mal vorkommen in den Tabellen ausgegraut. In der neu hinzugefügten dritten Spalte der Tabellen wird für jeden Wert prozentual angegeben, wie häufig die Architektur mit diesem Wert als HP Pareto-effizient ist (sei ein Wert 512 mal für Variationen ausgewählt worden und 64 dieser Variationen lägen in der Pareto-Front, dann entspräche das einer Häufigkeit von 12.5%).
Die Werte mit der höchsten Häufigkeit innerhalb des Wertbereichs werden in den Tabellen rot hervorgehoben, bei diesen Werten handelt es sich um die optimalen HPs des jeweiligen Ansatzes.
Diese Betrachtung der HPs ergibt einen überschaubaren Überblick über geeignete HPs, jedoch werden die Ansätze eigentlich spezifisch für jeden Datensatz und nicht das gesamte Archiv optimiert. Eine Evaluation über geeignete HPs für die einzelnen Datensätze wäre unanschaulich und bei 8 HP-Variationen pro Datensatz kaum aussagend. Im Weiteren geht diese Evaluation der HPs eher weniger auf die Tauglichkeit der tatsächlichen Kombination der geeigneten Werte (rot markiert) ein, da diese Kombination nicht zwangsweise getestet wurde oder der Test gravierenden Einfluss auf das Ergebnis hatte. Daher kann es möglich sein, dass die Werte in dieser Konstellation gegenseitig ihr Potential mindern. Nichtsdestotrotz lassen sich anhand dieser Auswertung einige Erkenntnisse gewinnen.

Auswertung der Hyperparameter des MLPs		
Funktion	mögliche Werte	Häufigkeit Pareto-Opimum
Quantisierungsfunktion	SteSign, ApproxSign, SwishSign	20.72%, 3.69%, 2.73%
Anzahl Neuronen pro Schicht **[n]**	128, 256, 512, 1024	22.09%, 2.85%, 0%, 0.75%
Anzahl zusätzlicher Schichten **[i]**	0, 1, 2	25.71%, 3.45%, 2.94%
Dropout Rate	0.0, 0.25, 0.5	39.47%, 0%, 2.01%
Lernrate	10^{-3}	11.71%
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}	27.76%, 1.59%, 3.01%,
Quantisierung in erster Schicht	True	11.71%
Quantisierung in letzter Schicht	True	11.71%

Tabelle 5.9: Evalutation der Werte in den Wertebereichen der HPs des MLPs.

Auswertung der Hyperparameter des FCNs		
Funktion	mögliche Werte	Häufigkeit Pareto-Opimum
Quantisierungsfunktion	SteSign, ApproxSign, SwishSign	1.97%, 21.64%, 10.75%
Anzahl Filter pro Schicht **[n]**	32, 64, 128	27.07%, 0%, 5.93%
Kernel Größe **[m]**	3, 5, 7, 9	4.26%, 24.08%, 22.3%, 4.09%
Anzahl zusätzlicher Schichten **[i]**	0, 1, 2, 3, 4	3.83%, 42.19%, 21.62%, 4.06%, 9.52%
Lernrate	10^{-2}, 10^{-3}, 10^{-4}	22.45%, 11.67%, 6.48%
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}	19.3%, 8.28%, 3.33%
Quantisierung in erster Schicht	True, False	4.48%, 18.4%
Quantisierung in letzter Schicht	True, False	2.5%, 13.08%

Tabelle 5.10: Evalutation der Werte in den Wertebereichen der HPs des FCNs.

Auswertung der Hyperparameter des ResNets		
Funktion	mögliche Werte	Häufigkeit Pareto-Opimum
Quantisierungsfunktion	SteSign, ApproxSign, SwishSign	17.19%, 11.33%, 11.33%
Anzahl Filter pro Schicht **[n]**	32, 64, 128, 256	14.84%, 0%, 14.58%, 0%
Kernel Größe **[m]**	3, 5, 7, 9, 11	13.28%, 14.84%, 6.25%, 21.87%
Anzahl zusätzlicher Blöcke **[i]**	0, 1, 2, 3	21.87%, 0%, 13.28%, 6.05%
Lernrate	10^{-2}, 10^{-3}, 10^{-4}	13.67%, 12.89%, 10.94%
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}	10.94%, 16.09%, 0%
Quantisierung in erster Schicht	True, False	16.21%, 9.37%
Quantisierung in letzter Schicht	True, False	0%, 12.79%

Tabelle 5.11: Evalutation der Werte in den Wertebereichen der HPs des ResNets.

Auswertung der Hyperparameter des DenseNets		
Funktion	mögliche Werte	Häufigkeit Pareto-Opimum
Quantisierungsfunktion	SteSign	12.6%
Anzahl Filter pro Schicht **[n]**	32, 64	13.62%, 5.47%
Kernel Größe **[m]**	3, 5, 7, 9	6.84%, 7.42%, 25.78%, 32.81%
Anzahl zusätzlicher Blöcke **[i]**	1, 2, 3	9.37%, 7.03%, 15%
Anzahl Schichten pro Block	3	12.6%
Lernrate	10^{-3}	12.6%
Verfallsrate (Decay)	0.0, 10^{-3}, 10^{-4}	8.07%, 12.7%, 25.78%
Quantisierung in erster Schicht	True	12.6%

Tabelle 5.12: Evalutation der Werte in den Wertebereichen der HPs des DenseNets.

Generell kann bei allen vier Ansätzen eine Tendenz zu den niedrigeren HP-Werten beobachtet werden. Das kann durch den Einfluss der Kosten in die Pareto-Effizienz begründet werden, da größere HP-Werte generell eine Erhöhung der Kosten mit sich tragen. Dies könnte zudem eine Erklärung bezüglich des sehr geringen Speicherbedarfs der Modelle

(siehe Tabellen 5.2, 5.4) liefern. Das DenseNet scheint hingegen leicht in Richtung eines tieferen Netzes zu tendieren, jedoch lässt sich das durch die relativ stark eingeschränkten Wertebereiche nicht mit Sicherheit sagen.
Besonders auffallend ist der optimale Wert für die Kernel Größe des ResNets und des DenseNets. Bei beiden Ansätzen eignen sich für die Kernel Größe m die größten Werte des jeweiligen Wertebereichs.

5.3.6 Vergleich mit state-of-the-art Ansatz

In diesem Abschnitt wird die beste Architektur der in Kapitel 4.2 vorgestellten Ansätze mit dem derzeitigem state-of-the-art Alogrithmus *HIVE-COTE* verglichen. Anhand der Abbildung 5.11 wird DenseNet als bester Ansatz für diesen Vergleich herangezogen. Erneut wurde das gewichtete Pareto-Effizienz-Verfahren aus Kapitel 5.3.3 angewandt, um für jeden Datensatz die optimale DenseNet-Architektur zu wählen.
Um den Vergleich seitens des state-of-the-art Ansatzes zu ermöglichen, wurden die im "UEA UCR Time Series Classification Repository" [18] zur Verfügung gestellten Ergebnisse des auf dem UCR/UEA Archiv [24] Trainierten *HIVE-COTE* Ensembles ausgewertet. In Tabelle 5.12 werden die Präzisionen zwischen DenseNet und *HIVE-COTE* durch das Bilden der Differenz verglichen. Ein negativer Wert entspricht einer der Überlegenheit von *HIVE-COTE*. Die durchschnittliche Differenz über alle Datensätze beträgt **-2,11**, somit schneidet der state-of-the-art Ansatz *HIVE-COTE* insgesamt besser ab als die DenseNet-Architektur dieser Arbeit.
Im Hinblick auf die Kosten schneidet das Ensemble *HIVE-COTE* jedoch um einiges schlechter ab. Wie in Kapitel 3.2 bereits aufgeführt wurde, besteht das Ensemble aus 37 Klassifikatoren, darunter befindet sich auch der sehr zeitaufwendige Shapelet Transform Algorithmus. Infolgedessen beläuft sich die Zeitkomplexität von *HIVE-COTE* auf $\mathcal{O}(N^2 \cdot T^2)$, wobei N die Anzahl an Zeitreihen des jeweiligen Datensatzes und T die Länge einer Zeitreihe darstellt [15]. Die Zeitkomplexität eines DenseNets beträgt hingegen lediglich $0(L^2)$, mit L als Tiefe des Netzes [66]. Somit kann die Aussage getroffen werden, dass die Laufzeit des DenseNets erheblich geringer sein wird, als die von *HIVE-COTE*. Bezüglich des Speicherbedarfs kann die Annahme getroffen werden, dass *HIVE-COTE* aufgrund der 37 Klassifikatoren auch in dieser Hinsicht kostspieliger sein wird.

Datensatz	Genauigkeit
ACSF1	8.54
Adiac	18.16
ArrowHead	-13.66
BME	-1.73
Beef	10.44
BeetleFly	-24.33
BirdChicken	-8
CBF	-3.52
Car	-4.06
Chinatown	-2.11
ChlorineConcentration	-3.89
CinCECGTorso	-21.47
Coffee	-33.57
Computers	-10.07
CricketX	13.39
CricketY	13.80
CricketZ	12.09
Crop	20.63
DiatomSizeReduction	-3.91
DistalPhalanxOutlineAgeGroup	-0.53
DistalPhalanxOutlineCorrect	-10.62
DistalPhalanxTW	20.29
ECG200	-1.66
ECG5000	2.88
ECGFiveDays	-25.58
EOGHorizontalSignal	11.99
EOGVerticalSignal	15.48
Earthquakes	-1.22
ElectricDevices	3.9
EthanolLevel	-11.25
FaceAll	-1.45
FaceFour	-3.11
FacesUCR	1.59
FiftyWords	21.81
Fish	-9.96
FordA	-0.32
FordB	-12.7
FreezerRegularTrain	-33.67
FreezerSmallTrain	-31.65
GunPoint	-19.56
GunPointAgeSpan	-10.74
GunPointMaleVersusFemale	-4.67
GunPointOldVersusYoung	-15.24
Ham	-22.29
HandOutlines	-4.58
Haptics	22.09
Herring	-8.39
HouseTwenty	-7.11
InlineSkate	32.91
InsectEPGRegularTrain	-4.98
InsectEPGSmallTrain	-23.19
InsectWingbeatSound	26.8
ItalyPowerDemand	-1.69
LargeKitchenAppliances	-7.16
Lightning2	-5.85
Lightning7	17.4

Datensatz	Genauigkeit
Mallat	-7.85
Meat	-30.28
MedicalImages	19.21
MiddlePhalanxOutlineAgeGroup	0.39
MiddlePhalanxOutlineCorrect	-11.47
MiddlePhalanxTW	29.26
MixedShapesRegularTrain	-2.09
MixedShapesSmallTrain	-5.66
MoteStrain	-6.51
NonInvasiveFetalECGThorax1	5.12
NonInvasiveFetalECGThorax2	2.93
OSULeaf	-4.35
OliveOil	-16.17
PhalangesOutlinesCorrect	-8.66
Phoneme	60.27
PigAirwayPressure	2.32
PigArtPressure	2.7
PigCVP	2.76
Plane	-0.41
PowerCons	-2.91
ProximalPhalanxOutlineAgeGroup	2.86
ProximalPhalanxOutlineCorrect	-9.62
ProximalPhalanxTW	10.63
RefrigerationDevices	-11.24
Rock	-12.33
ScreenType	-8.83
SemgHandGenderCh2	-24.01
SemgHandMovementCh2	-5.73
SemgHandSubjectCh2	-9.56
ShapeletSim	-25.89
ShapesAll	5.8
SmallKitchenAppliances	4.65
SmoothSubspace	-1.46
SonyAIBORobotSurface1	9.68
SonyAIBORobotSurface2	-11.25
StarLightCurves	-3.96
Strawberry	-3.4
SwedishLeaf	3.4
Symbols	-1.66
SyntheticControl	0.11
ToeSegmentation1	-11.14
ToeSegmentation2	-26.82
Trace	-0.1
TwoLeadECG	-5.19
TwoPatterns	0.05
UMD	-4.05
UWaveGestureLibraryAll	-2.32
UWaveGestureLibraryX	11.01
UWaveGestureLibraryY	16.77
UWaveGestureLibraryZ	15.6
Wafer	-0.31
Wine	-38.83
WordSynonyms	27.45
Worms	10.8
WormsTwoClass	-10.39
Yoga	-20.46

Abbildung 5.12: Vergleich der Genauigkeit zwischen DenseNet und *HIVE-COTE*.

5.4 Bewertung der Ergebnisse

Die Ergebnisse für die implementierten Modelle (MLP, FCN, ResNet, DenseNet) sind insgesamt sehr zufriedenstellend und auch die binarisierten Vergleichsmodelle zeigen erstaunlich gute Ergebnisse. Während dabei die Laufzeit durch die Binarisierung nur wenig verändert wird und der Speicherbedarf sogar sehr stark reduziert wird (Faktor fast 32), sinkt die Genauigkeit generell um ca 10%.
Bei der Betrachtung der ungewichteten Pareto-Effizienz erhalten insbesondere MLP und FCN gute Bewertungen – auch im Vergleich zu DenseNet. Vor allem bei MLP ist diese Bewertung jedoch auf eine sehr geringe Laufzeit zurückzuführen, während die Genauigkeit nur vergleichsweise gering ist. Bei der mit der erreichten Genauigkeit gewichteten Bestimmung der Pareto-Effizienz der Ansätze MLP, FCN, ResNet, DenseNet, CfMLP, CfFCN und CfResNet wurde DenseNet als bester Kompromiss aus Genauigkeit, Speicherbedarf und Laufzeit identifiziert. ResNet wies in diesem Vergleich einen zu hohen Bedarf an Laufzeit und Speicherplatz auf, so dass es trotz überdurchschnittlicher Genauigkeit keine gute Bewertung erreichte.

Mit dem Maßstab der gewichteten Pareto-Effizienz separat für jeden einzelnen Ansatz wurde für jeden Datensatz die jeweils am besten geeignete Architektur/HP-Variation ermittelt und in den drei Leistungsmerkmalen Genauigkeit, Laufzeit und Speicherbedarf mit dem entsprechenden full-precision Vergleichsmodell aus dem Review von Hassan Fawaz [15] verglichen. Dabei zeigte sich eine scheinbare Überlegenheit unserer Architekturen. Trotz Binarisierung erreichen sie eine höhere Genauigkeit, eine Senkung der Laufzeit und einen sehr stark verringerten Speicherplatzbedarf.

Bei der Auswertung der Eignung verschiedener Werte für die HPs eines Ansatzes zeigte sich, dass die Modelle überwiegend zu kleineren HPs tendieren und dies auch meist zu kleineren Architekturen führt. Vermutlich ist dies der Grund für den überraschend geringen Speicherbedarf unserer Modelle, der beim MLP sogar nur noch ein 178-stel im Vergleich zum full-precision-CFMLP beträgt.
Um die Menge an Informationen zu behalten, kann es sinnvoll sein, z.B. bei der Eingabe in die erste Schicht (also Eingabe ins Netz) die volle Auflösung aufrecht zu behalten. Die Modelle bei denen diese Binarisierung unterlassen wurde, sind auch aus der HP-Analyse heraus öfter Pareto-effizient gewesen.

DenseNet zeigte auf dem UCR/UEA Archiv eine nur um ca. 2% schlechtere Genauigkeit als der state-of-the-art Ansatz *HIVE-COTE*, obwohl *HIVE-COTE* eine sehr viel höhere Laufzeit (Zeitkomplexität) und vermutlich nicht nur durch die full-precision-Arbeitsweise einen auch sehr viel größeren Speicherbedarf hat.

Kapitel 6

Fazit und Ausblick

In der vorliegenden Arbeit wurden binäre neuronale Netze auf ihre Tauglichkeit zur Zeitreihenklassifikation geprüft. Sie erwiesen sich dabei prinzipiell als sehr gut geeignet.

Dabei zeigte sich, dass eine einfache Binarisierung zwar in einer starken Reduzierung des Speicherbedarfs bei kaum veränderter Laufzeit resultiert, damit allerdings ein recht großer Informationsverlust einhergeht.

Werden die Modelle durch optimierte Wahl von Hyperparametern angepasst, dann erhöht sich die erreichte Genauigkeit, während sich neben dem erheblich kleineren Speicherbedarf auch die Laufzeit um die Hälfte oder sogar um den Faktor 5 verringert. Mit so wesentlich weniger Ressourcenbedarf vereinfacht sich die Entwicklung von lokalverarbeitenden Zeitreihenklassifikationen beträchtlich und macht diese zum Teil erst realisierbar.

6.1 Ausblick

Die Ursache dafür, dass der Resourcenbedarf bei sehr guter Genauigkeit so sehr abnimmt, sollte genauer untersucht werden. Während normalerweise Kompromisse eingegangen werden müssen, traten in diesem Fall ausschließlich Verbesserungen auf.

Insbesondere die extrem starke, 178-fache Verkleinerung des Speicherbedarfs des MLPs ist fragwürdig. Ein möglicher Erklärungsansatz wäre, dass unter anderem der erfolgreichste HP-Wert von 128 Neuronen pro Schicht den Speicherbedarf um einen ähnlichen Faktor verringert, während das full-precision VergleichsMLP immer konstante 500 Neuronen aufweist.

Die relativ schlechte Bewertung für ResNet im Vergleich zu MLP, FCN und DenseNet überrascht ebenfalls, da ResNet eigentlich eine weiterentwickelte Form eines FCNs dar-

stellt und in der Vergleichsarbeit DL4TSC [15] mit Abstand der Spitzenreiter war. Die Ursache für die unterschiedlichen Bewertungen könnte darin liegen, dass in der Vergleichsarbeit die bei ResNet relativ hohen Kosten nicht betrachtet wurden.

Bei weiteren Untersuchungen sollte der Wertebereich für die HPs von DenseNet etwas ausgeweitet werden. DenseNet sollte auch mit mehr als 3 Schichten pro Block getestet werden, da die Anzahl Shortcuts innerhalb eines Blocks exponentiell zur Anzahl an Schichten in einem Block steigen. Zudem sind diese Shortcuts die Besonderheit des DenseNets. Außerdem könnte es, da die Kosten so gering sind, sinnvoll sein, die Modelle noch tiefer zu machen, um damit gegebenenfalls noch bessere Genauigkeiten zu erreichen.

Anhang A

Weitere Informationen

tags.dataset	acc_MLP	runtime_MLP	memory_MLP
ACSF1	156.34	19.71	0.64
Adiac	248.74	13.67	0.48
ArrowHead	82.41	15.60	0.39
BME	90.08	12.16	0.34
Beef	111.59	59.41	1.48
BeetleFly	69.32	17.69	0.46
BirdChicken	82.43	56.14	1.49
CBF	86.49	12.44	0.34
Car	92.87	19.05	0.47
Chinatown	80.40	9.71	0.29
ChlorineConcentration	85.98	13.65	0.36
CinCECGTorso	80.97	24.20	0.60
Coffee	57.55	14.95	0.40
Computers	94.84	19.47	0.50
CricketX	147.68	15.41	0.41
CricketY	147.20	11.90	0.41
CricketZ	140.85	17.24	0.60
Crop	153.87	8.85	0.32
DiatomSizeReduction	81.04	12.26	0.42
DistalPhalanxOutlineAgeGroup	111.85	8.95	0.32
DistalPhalanxOutlineCorrect	76.67	8.89	0.32
DistalPhalanxTW	132.82	9.04	0.32
ECG200	84.90	8.88	0.33
ECG5000	103.85	9.91	0.35
ECGFiveDays	74.88	58.85	1.45
EOGHorizontalSignal	202.40	17.57	0.57
EOGVerticalSignal	209.91	17.61	0.57
Earthquakes	88.91	13.65	0.46
ElectricDevices	135.61	9.28	0.33
EthanolLevel	194.30	19.14	0.61
FaceAll	115.34	9.89	0.35
FaceFour	91.92	12.25	0.42
FacesUCR	110.91	9.88	0.35
FiftyWords	138.12	11.79	0.41
Fish	98.67	12.84	0.45
FordA	62.43	13.56	0.45

Continued on next page

tags.dataset	acc_MLP	runtime_MLP	memory_MLP
FordB	71.59	18.60	0.62
FreezerRegularTrain	79.40	11.80	0.40
FreezerSmallTrain	99.20	11.72	0.40
Fungi	110.66	10.85	0.38
GunPoint	74.28	9.99	0.35
GunPointAgeSpan	80.96	10.00	0.35
GunPointMaleVersusFemale	86.64	9.96	0.35
GunPointOldVersusYoung	75.17	13.19	0.35
Ham	78.47	16.80	0.44
HandOutlines	70.25	26.12	0.66
Haptics	174.66	21.37	0.55
Herring	121.02	17.71	0.46
HouseTwenty	86.50	26.46	0.63
InlineSkate	239.81	25.71	0.62
InsectEPGRegularTrain	111.11	19.51	0.48
InsectEPGSmallTrain	116.01	18.05	0.48
InsectWingbeatSound	150.45	13.59	0.39
ItalyPowerDemand	94.78	9.63	0.29
LargeKitchenAppliances	135.00	26.95	0.65
Lightning2	88.94	14.60	0.48
Lightning7	138.29	12.00	0.41
Mallat	92.78	16.67	0.54
Meat	75.62	13.18	0.44
MedicalImages	124.97	9.38	0.33
MiddlePhalanxOutlineAgeGroup	111.94	8.98	0.32
MiddlePhalanxOutlineCorrect	75.59	8.88	0.32
MiddlePhalanxTW	156.62	9.01	0.32
MixedShapesRegularTrain	97.13	16.75	0.54
MixedShapesSmallTrain	102.28	16.88	0.54
MoteStrain	86.72	41.80	1.44
NonInvasiveFetalECGThorax1	106.71	15.56	0.51
NonInvasiveFetalECGThorax2	106.29	45.39	1.51
OSULeaf	140.52	13.01	0.44
OliveOil	113.27	14.12	0.47
PhalangesOutlinesCorrect	81.22	9.01	0.32
Phoneme	1016.41	20.38	0.68
PigAirwayPressure	1476.36	19.76	0.63
PigArtPressure	921.44	19.61	0.63
PigCVP	1270.86	21.99	0.72
Plane	98.55	13.87	0.35
PowerCons	86.80	12.82	0.35
ProximalPhalanxOutlineAgeGroup	78.35	11.33	0.32
ProximalPhalanxOutlineCorrect	86.91	19.54	0.56
ProximalPhalanxTW	108.69	11.06	0.32
RefrigerationDevices	161.39	22.81	0.65
Rock	87.91	66.38	1.56
ScreenType	157.16	24.69	0.65
SemgHandGenderCh2	67.86	22.49	0.59
SemgHandMovementCh2	176.59	21.78	0.59
SemgHandSubjectCh2	92.54	23.52	0.59
ShapeletSim	101.95	21.84	0.62
ShapesAll	125.98	14.26	0.48
SmallKitchenAppliances	168.86	19.47	0.65

Continued on next page

	acc_MLP	runtime_MLP	memory_MLP
tags.dataset			
SmoothSubspace	77.10	7.89	0.29
SonyAIBORobotSurface1	91.54	8.81	0.32
SonyAIBORobotSurface2	88.87	8.63	0.31
StarLightCurves	88.99	16.74	0.54
Strawberry	65.28	16.82	0.59
SwedishLeaf	107.68	9.84	0.35
Symbols	104.93	12.79	0.43
SyntheticControl	95.23	8.69	0.31
ToeSegmentation1	93.89	11.52	0.39
ToeSegmentation2	79.96	43.47	1.47
Trace	96.65	11.52	0.39
TwoLeadECG	74.95	8.95	0.32
TwoPatterns	92.86	9.69	0.34
UMD	80.43	9.97	0.35
UWaveGestureLibraryAll	100.73	16.35	0.53
UWaveGestureLibraryX	115.13	12.00	0.41
UWaveGestureLibraryY	124.40	11.98	0.41
UWaveGestureLibraryZ	124.91	11.94	0.41
Wafer	91.14	9.98	0.35
Wine	98.63	11.09	0.38
WordSynonyms	158.95	11.75	0.40
Worms	163.41	16.01	0.53
WormsTwoClass	96.58	16.33	0.53
Yoga	63.08	12.98	0.44
Durchschnitt	108.19	18.84	0.56

Tabelle A.1: Prozentuales Verhältnis von Genauigkeit, Laufzeit und Speicher zwischen MLP und full-precision-CfMLP (100% entspricht dem Wert des full-precision-CfMLPs).

	acc_FCN	runtime_FCN	memory_FCN
tags.dataset			
ACSF1	105.63	11.10	0.65
Adiac	116.84	18.32	1.24
ArrowHead	82.02	20.15	0.83
BME	109.30	22.08	0.93
Beef	111.96	24.16	0.93
BeetleFly	74.73	126.01	4.45
BirdChicken	94.68	228.28	7.46
CBF	93.21	230.53	6.37
Car	82.03	19.43	0.62
Chinatown	99.70	229.17	6.33
ChlorineConcentration	91.26	54.50	1.74
CinCECGTorso	88.25	8.51	0.38
Coffee	78.57	227.74	6.33
Computers	90.92	7.38	0.36
CricketX	117.88	8.21	0.49
CricketY	116.49	8.26	0.49
CricketZ	115.46	8.19	0.49
Crop	131.77	6.39	0.63
DiatomSizeReduction	220.09	174.71	6.42
DistalPhalanxOutlineAgeGroup	113.83	6.37	0.38
DistalPhalanxOutlineCorrect	93.42	6.36	0.36

Continued on next page

tags.dataset	acc_FCN	runtime_FCN	memory_FCN
DistalPhalanxTW	128.74	6.37	0.41
ECG200	92.34	6.39	0.36
ECG5000	103.01	6.40	0.40
ECGFiveDays	72.82	6.37	0.36
EOGHorizontalSignal	160.94	6.42	0.49
EOGVerticalSignal	202.30	6.39	0.49
Earthquakes	102.78	6.58	0.35
ElectricDevices	130.37	6.27	0.42
EthanolLevel	137.23	6.59	0.38
FaceAll	102.69	6.37	0.51
FaceFour	89.55	63.08	2.42
FacesUCR	102.88	8.52	0.51
FiftyWords	151.87	8.26	0.94
Fish	102.04	230.34	6.56
FordA	98.59	8.17	0.36
FordB	96.42	6.37	0.36
FreezerRegularTrain	89.31	174.07	6.33
FreezerSmallTrain	92.94	6.61	0.35
Fungi	5164.38	95.34	4.46
GunPoint	98.27	175.51	6.33
GunPointAgeSpan	98.73	174.98	6.33
GunPointMaleVersusFemale	97.97	6.32	0.36
GunPointOldVersusYoung	94.28	174.92	6.33
Ham	78.71	6.36	0.36
HandOutlines	81.12	6.60	0.35
Haptics	157.67	6.62	0.39
Herring	92.23	38.39	1.69
HouseTwenty	74.49	6.39	0.36
InlineSkate	254.81	6.33	0.42
InsectEPGRegularTrain	72.24	6.65	0.36
InsectEPGSmallTrain	307.35	6.60	0.36
InsectWingbeatSound	228.37	6.39	0.47
ItalyPowerDemand	99.60	6.45	0.36
LargeKitchenAppliances	91.08	6.37	0.38
Lightning2	94.64	6.45	0.36
Lightning7	104.13	6.39	0.42
Mallat	97.09	175.06	6.60
Meat	74.69	6.38	0.38
MedicalImages	120.65	6.38	0.46
MiddlePhalanxOutlineAgeGroup	130.42	6.60	0.36
MiddlePhalanxOutlineCorrect	91.78	6.30	0.36
MiddlePhalanxTW	171.72	6.42	0.41
MixedShapesRegularTrain	101.43	175.34	6.47
MixedShapesSmallTrain	96.10	8.23	0.40
MoteStrain	98.79	238.90	6.33
NonInvasiveFetalECGThorax1	103.23	49.03	3.57
NonInvasiveFetalECGThorax2	104.01	49.07	3.57
OSULeaf	98.51	174.26	6.51
OliveOil	88.43	6.57	0.38
PhalangesOutlinesCorrect	91.25	6.38	0.36
Phoneme	297.08	6.44	0.80
PigAirwayPressure	569.83	6.61	0.95
PigArtPressure	101.07	175.18	8.56

Continued on next page

tags.dataset	acc_FCN	runtime_FCN	memory_FCN
PigCVP	118.05	63.11	4.65
Plane	99.89	169.30	6.56
PowerCons	98.97	6.36	0.36
ProximalPhalanxOutlineAgeGroup	106.50	6.35	0.38
ProximalPhalanxOutlineCorrect	93.48	174.20	6.33
ProximalPhalanxTW	120.36	6.40	0.41
RefrigerationDevices	134.41	6.36	0.38
Rock	111.55	6.57	0.38
ScreenType	116.75	6.39	0.38
SemgHandGenderCh2	95.30	6.33	0.36
SemgHandMovementCh2	175.19	63.01	2.52
SemgHandSubjectCh2	113.09	6.35	0.40
ShapeletSim	106.77	174.73	6.33
ShapesAll	110.97	227.92	8.90
SmallKitchenAppliances	110.25	7.58	0.38
SmoothSubspace	99.32	8.91	0.38
SonyAIBORobotSurface1	94.76	220.37	6.33
SonyAIBORobotSurface2	79.38	8.18	0.36
StarLightCurves	99.92	174.85	6.37
Strawberry	98.17	6.37	0.36
SwedishLeaf	101.96	6.42	0.52
Symbols	87.48	62.78	2.52
SyntheticControl	100.66	6.38	0.41
ToeSegmentation1	90.88	174.91	6.33
ToeSegmentation2	93.25	175.53	6.33
Trace	99.40	174.57	6.42
TwoLeadECG	98.37	175.04	6.33
TwoPatterns	109.97	280.31	9.29
UMD	97.28	174.63	6.37
UWaveGestureLibraryAll	114.90	174.39	6.60
UWaveGestureLibraryX	122.97	6.37	0.44
UWaveGestureLibraryY	140.93	6.31	0.44
UWaveGestureLibraryZ	126.33	6.36	0.44
Wafer	100.05	95.02	3.72
Wine	81.82	6.63	0.35
WordSynonyms	170.53	6.31	0.64
Worms	112.16	175.27	6.47
WormsTwoClass	92.66	6.35	0.36
Yoga	82.84	174.96	6.33
Durchschnitt	108.59	52.38	2.38

Tabelle A.2: Prozentuales Verhältnis von Genauigkeit, Laufzeit und Speicher zwischen FCN und full-precision-CfFCN (100% entspricht dem Wert des full-precision-CfFCNs).

tags.dataset	acc_ResNet	runtime_ResNet	memory_ResNet
ACSF1	105.90	165.62	6.51
Adiac	117.84	131.93	5.84
ArrowHead	86.45	199.79	6.38
BME	88.12	203.61	6.38
Beef	103.01	197.96	6.43
BeetleFly	76.47	62.07	5.72

Continued on next page

tags.dataset	acc_ResNet	runtime_ResNet	memory_ResNet
BirdChicken	67.05	63.10	5.72
CBF	94.88	197.98	6.38
Car	73.73	27.19	3.48
Chinatown	99.52	205.51	6.36
ChlorineConcentration	87.87	176.02	6.35
CinCECGTorso	88.29	27.04	3.48
Coffee	78.57	182.73	6.36
Computers	86.79	27.12	3.28
CricketX	117.30	62.40	6.72
CricketY	114.06	16.32	0.96
CricketZ	115.85	27.26	4.28
Crop	131.31	27.94	5.48
DiatomSizeReduction	259.02	184.11	6.40
DistalPhalanxOutlineAgeGroup	109.55	27.70	3.38
DistalPhalanxOutlineCorrect	94.92	119.93	5.00
DistalPhalanxTW	134.16	119.70	5.10
ECG200	88.56	27.70	3.28
ECG5000	103.18	16.23	0.91
ECGFiveDays	89.95	183.01	6.36
EOGHorizontalSignal	150.41	28.41	4.28
EOGVerticalSignal	198.88	27.29	4.28
Earthquakes	104.85	27.52	3.28
ElectricDevices	126.23	16.23	0.93
EthanolLevel	82.28	17.87	0.91
FaceAll	111.60	65.25	6.92
FaceFour	96.19	202.16	6.40
FacesUCR	102.53	64.55	6.92
FiftyWords	132.88	135.18	6.15
Fish	85.78	189.37	6.48
FordA	99.69	62.19	5.72
FordB	98.21	16.26	0.90
FreezerRegularTrain	67.92	27.76	3.28
FreezerSmallTrain	75.62	28.66	3.28
Fungi	540.88	65.61	7.32
GunPoint	85.87	29.13	3.28
GunPointAgeSpan	88.89	27.88	3.28
GunPointMaleVersusFemale	97.83	27.65	3.28
GunPointOldVersusYoung	76.83	27.28	3.28
Ham	90.70	65.64	5.72
Haptics	148.20	182.34	6.43
Herring	98.96	27.15	3.28
InsectEPGRegularTrain	86.34	182.65	6.38
InsectEPGSmallTrain	204.10	182.08	6.38
InsectWingbeatSound	180.03	62.34	6.62
ItalyPowerDemand	99.37	210.27	6.36
LargeKitchenAppliances	97.65	27.26	3.38
Lightning2	95.80	27.13	3.28
Lightning7	106.62	27.40	3.78
Mallat	85.54	27.02	3.88
Meat	60.61	27.54	3.38
MedicalImages	122.01	63.93	6.52
MiddlePhalanxOutlineAgeGroup	129.60	128.79	5.03
MiddlePhalanxOutlineCorrect	88.19	35.50	3.28

Continued on next page

tags.dataset	acc_ResNet	runtime_ResNet	memory_ResNet
MiddlePhalanxTW	175.37	17.17	0.92
MixedShapesRegularTrain	98.61	120.38	5.07
MixedShapesSmallTrain	88.26	27.00	3.58
MoteStrain	98.69	184.30	6.36
NonInvasiveFetalECGThorax1	105.38	120.79	5.96
NonInvasiveFetalECGThorax2	104.34	27.23	7.27
OSULeaf	92.05	63.58	6.12
OliveOil	84.65	16.35	0.91
PhalangesOutlinesCorrect	82.93	16.38	0.90
Phoneme	291.62	16.30	1.13
Plane	99.73	151.85	6.44
PowerCons	101.90	28.58	3.28
ProximalPhalanxOutlineAgeGroup	100.61	16.38	0.90
ProximalPhalanxOutlineCorrect	87.74	119.54	5.00
ProximalPhalanxTW	119.15	181.81	6.45
RefrigerationDevices	128.27	16.20	0.90
Rock	116.30	16.56	0.91
ScreenType	108.06	123.87	5.03
SemgHandGenderCh2	103.97	26.93	3.28
SemgHandMovementCh2	195.09	31.27	3.68
SemgHandSubjectCh2	122.73	31.40	3.58
ShapeletSim	91.90	207.43	6.36
ShapesAll	107.22	134.13	6.39
SmallKitchenAppliances	108.60	152.70	6.35
SmoothSubspace	99.50	66.73	5.82
SonyAIBORobotSurface1	95.98	182.52	6.36
SonyAIBORobotSurface2	87.26	63.56	5.72
StarLightCurves	96.09	29.33	3.38
Strawberry	96.86	153.94	6.32
SwedishLeaf	102.65	119.92	5.32
Symbols	90.03	181.86	6.45
SyntheticControl	100.07	68.85	6.12
ToeSegmentation1	94.23	181.52	6.36
ToeSegmentation2	97.93	184.07	6.36
Trace	83.40	27.59	3.48
TwoLeadECG	98.12	183.01	6.36
TwoPatterns	100.00	62.22	5.92
UMD	85.97	28.01	3.38
UWaveGestureLibraryAll	108.22	152.46	6.47
UWaveGestureLibraryX	119.82	152.90	6.47
UWaveGestureLibraryY	135.74	119.75	5.15
UWaveGestureLibraryZ	123.51	152.03	6.47
Wafer	99.88	120.42	5.00
Wine	69.23	16.15	0.90
WordSynonyms	155.50	62.33	8.01
Worms	113.86	119.75	5.07
WormsTwoClass	88.89	27.81	3.28
Yoga	68.87	28.07	3.28
Durchschnitt	104.72	62.52	4.71

Tabelle A.3: Prozentuales Verhältnis von Genauigkeit, Laufzeit und Speicher zwischen ResNet und full-precision-CfResNet (100% entspricht dem Wert des full-precision-CfResNets).

tags.dataset	acc_CfMLP	runtime_CfMLP	memory_CfMLP
ACSF1	161.29	99.98	3.24
Adiac	248.65	100.69	3.37
AllGestureWiimoteX	-	-	3.32
AllGestureWiimoteY	-	-	3.32
AllGestureWiimoteZ	-	-	3.32
ArrowHead	81.68	101.09	3.36
BME	73.64	100.46	3.38
Beef	105.42	99.33	3.32
BeetleFly	56.82	100.15	3.32
BirdChicken	67.57	100.28	3.32
CBF	69.00	100.23	3.38
Car	95.74	101.15	3.31
Chinatown	41.81	102.24	3.41
ChlorineConcentration	84.42	103.70	3.37
CinCECGTorso	86.54	109.09	3.24
Coffee	52.52	100.37	3.35
Computers	89.54	104.25	3.29
CricketX	155.03	98.18	3.35
CricketY	153.17	100.57	3.35
CricketZ	145.68	100.12	3.35
Crop	155.14	101.33	3.40
DiatomSizeReduction	78.18	100.46	3.34
DistalPhalanxOutlineAgeGroup	100.96	100.41	3.39
DistalPhalanxOutlineCorrect	80.26	99.67	3.39
DistalPhalanxTW	127.95	100.24	3.39
DodgerLoopDay	-	-	3.35
DodgerLoopGame	-	-	3.35
DodgerLoopWeekend	-	-	3.35
ECG200	70.02	111.22	3.39
ECG5000	88.21	122.46	3.38
ECGFiveDays	51.22	108.99	3.38
EOGHorizontalSignal	212.17	114.87	3.25
EOGVerticalSignal	219.47	112.90	3.25
ElectricDevices	144.65	116.63	3.39
EthanolLevel	194.30	113.06	3.23
FaceAll	116.97	116.02	3.38
FaceFour	87.23	107.91	3.34
FacesUCR	110.46	110.31	3.38
FiftyWords	137.97	111.34	3.35
Fish	98.56	112.70	3.32
FordA	61.65	109.02	3.32
FordB	70.61	115.55	3.32
FreezerRegularTrain	55.19	112.04	3.35
FreezerSmallTrain	72.91	116.81	3.35
Fungi	109.38	118.72	3.37
GestureMidAirD1	-	-	3.34
GestureMidAirD2	-	-	3.34
GestureMidAirD3	-	-	3.34
GesturePebbleZ1	-	-	3.32
GesturePebbleZ2	-	-	3.32
GunPoint	53.45	114.40	3.38
GunPointAgeSpan	53.66	115.09	3.38

Continued on next page

tags.dataset	acc_CfMLP	runtime_CfMLP	memory_CfMLP
GunPointMaleVersusFemale	53.58	115.91	3.38
GunPointOldVersusYoung	53.64	108.63	3.38
Ham	71.12	109.47	3.33
HandOutlines	57.78	113.89	3.20
Haptics	182.60	111.11	3.26
Herring	98.09	108.64	3.32
HouseTwenty	65.90	109.18	3.22
InlineSkate	256.21	112.59	3.23
InsectEPGRegularTrain	102.69	125.75	3.31
InsectEPGSmallTrain	99.66	110.34	3.31
InsectWingbeatSound	150.45	101.43	3.36
ItalyPowerDemand	52.54	100.80	3.41
LargeKitchenAppliances	141.88	100.67	3.29
Lightning2	72.12	100.31	3.30
Lightning7	139.05	102.58	3.34
Mallat	94.83	101.15	3.27
Meat	72.14	100.61	3.33
MedicalImages	125.55	100.23	3.39
MelbournePedestrian	-	-	3.41
MiddlePhalanxOutlineAgeGroup	111.77	100.63	3.39
MiddlePhalanxOutlineCorrect	68.12	100.16	3.39
MiddlePhalanxTW	155.37	-	3.39
MixedShapesRegularTrain	88.23	100.74	3.27
MixedShapesSmallTrain	95.12	100.85	3.27
MoteStrain	61.22	100.95	3.39
NonInvasiveFetalECGThorax1	106.71	100.43	3.29
NonInvasiveFetalECGThorax2	106.30	100.50	3.29
OSULeaf	145.25	100.83	3.33
OliveOil	108.16	100.11	3.31
PLAID	-	-	3.25
PhalangesOutlinesCorrect	69.15	100.71	3.39
Phoneme	1034.37	100.53	3.27
PickupGestureWiimoteZ	-	-	3.34
PigAirwayPressure	1500.00	101.41	3.22
PigArtPressure	935.78	101.13	3.22
PigCVP	1291.14	100.21	3.22
Plane	85.35	100.21	3.38
PowerCons	51.19	101.13	3.38
ProximalPhalanxOutlineAgeGroup	78.35	100.42	3.39
ProximalPhalanxOutlineCorrect	93.69	100.37	3.39
ProximalPhalanxTW	104.58	101.29	3.39
RefrigerationDevices	176.80	100.30	3.29
Rock	80.99	100.01	3.20
ScreenType	165.78	100.62	3.29
SemgHandGenderCh2	57.15	100.14	3.24
SemgHandMovementCh2	191.52	100.28	3.24
SemgHandSubjectCh2	97.88	100.30	3.24
ShakeGestureWiimoteZ	-	-	3.33
ShapeletSim	97.40	100.29	3.32
ShapesAll	126.66	100.40	3.32
SmallKitchenAppliances	175.32	101.17	3.29
SmoothSubspace	68.03	101.49	3.41
SonyAIBORobotSurface1	62.02	100.38	3.40

Continued on next page

	acc_CfMLP	runtime_CfMLP	memory_CfMLP
tags.dataset			
SonyAIBORobotSurface2	68.59	100.36	3.40
StarLightCurves	74.52	100.13	3.27
Strawberry	61.10	101.27	3.36
SwedishLeaf	110.40	101.11	3.38
Symbols	94.20	101.49	3.33
SyntheticControl	83.33	100.08	3.40
ToeSegmentation1	87.63	100.49	3.35
ToeSegmentation2	58.68	102.21	3.34
Trace	93.05	102.25	3.35
TwoLeadECG	66.42	101.20	3.39
TwoPatterns	79.11	100.94	3.38
UMD	67.94	101.02	3.38
UWaveGestureLibraryAll	91.75	100.15	3.27
UWaveGestureLibraryX	113.95	102.32	3.35
UWaveGestureLibraryY	125.19	100.62	3.35
UWaveGestureLibraryZ	125.53	101.48	3.35
Wafer	89.58	100.67	3.38
Wine	92.47	100.18	3.36
WordSynonyms	159.59	101.19	3.35
Worms	173.75	100.64	3.28
WormsTwoClass	84.62	101.35	3.28
Yoga	59.23	101.12	3.33
Durchschnitt	100.64	105.44	3.31

Tabelle A.4: Prozentuales Verhältnis von Genauigkeit, Laufzeit und Speicher zwischen CfMLP und full-precision-CfMLP (100% entspricht dem Wert des full-precision-CfMLPs).

CfFCN:

	acc_CfFCN	runtime_CfFCN	memory_CfFCN
tags.dataset			
ACSF1	100.22	100.47	3.69
Adiac	115.63	99.31	3.69
AllGestureWiimoteX	-	-	3.69
AllGestureWiimoteY	-	-	3.69
AllGestureWiimoteZ	-	-	3.69
ArrowHead	75.93	100.48	3.69
BME	77.09	100.48	3.69
Beef	114.12	100.49	3.69
BeetleFly	54.95	100.77	3.69
BirdChicken	53.19	100.03	3.69
CBF	67.08	100.36	3.69
Car	79.20	101.77	3.69
Chinatown	55.65	98.48	3.69
ChlorineConcentration	83.77	100.42	3.69
CinCECGTorso	90.49	100.43	3.69
Coffee	52.14	100.23	3.69
Computers	61.04	100.93	3.69
CricketX	113.66	100.33	3.69
CricketY	115.62	109.00	3.69
CricketZ	111.45	100.26	3.69
Crop	129.83	100.98	3.69

Continued on next page

tags.dataset	acc_CfFCN	runtime_CfFCN	memory_CfFCN
DiatomSizeReduction	193.21	103.88	3.69
DistalPhalanxOutlineAgeGroup	91.05	100.69	3.69
DistalPhalanxOutlineCorrect	72.35	100.28	3.69
DistalPhalanxTW	114.22	100.96	3.69
DodgerLoopDay	-	-	3.69
DodgerLoopGame	-	-	3.69
DodgerLoopWeekend	-	-	3.69
ECG200	72.07	101.24	3.69
ECG5000	86.53	100.50	3.69
ECGFiveDays	50.68	100.65	3.69
EOGHorizontalSignal	162.35	110.02	3.69
EOGVerticalSignal	205.34	104.78	3.69
Earthquakes	103.17	100.77	3.69
ElectricDevices	121.33	98.32	3.69
EthanolLevel	144.63	103.46	3.69
FaceAll	99.05	109.86	3.69
FaceFour	78.48	106.61	3.69
FacesUCR	98.43	105.83	3.69
FiftyWords	151.16	106.31	3.69
Fish	84.90	104.54	3.69
FordA	55.05	103.00	3.69
FordB	64.87	107.13	3.69
FreezerRegularTrain	50.16	103.49	3.69
FreezerSmallTrain	73.20	102.93	3.69
Fungi	5113.07	105.28	3.69
GestureMidAirD1	-	-	3.69
GestureMidAirD2	-	-	3.69
GestureMidAirD3	-	-	3.69
GesturePebbleZ1	-	-	3.69
GesturePebbleZ2	-	-	3.69
GunPoint	50.13	102.42	3.69
GunPointAgeSpan	50.35	105.95	3.69
GunPointMaleVersusFemale	49.65	106.92	3.69
GunPointOldVersusYoung	52.02	104.87	3.69
Ham	71.16	101.71	3.69
HandOutlines	73.14	102.11	3.69
Haptics	158.62	101.96	3.69
Herring	80.58	104.65	3.69
HouseTwenty	55.82	101.95	3.69
InlineSkate	235.46	102.68	3.69
InsectEPGRegularTrain	62.92	104.77	3.69
InsectEPGSmallTrain	301.96	102.59	3.69
InsectWingbeatSound	231.96	103.12	3.69
ItalyPowerDemand	51.96	108.49	3.69
LargeKitchenAppliances	71.37	102.49	3.69
Lightning2	73.66	105.38	3.69
Lightning7	98.10	102.90	3.69
Mallat	86.58	102.77	3.69
Meat	80.22	103.72	3.69
MedicalImages	113.60	102.46	3.69
MelbournePedestrian	-	-	3.69
MiddlePhalanxOutlineAgeGroup	109.06	101.96	3.69
MiddlePhalanxOutlineCorrect	61.16	103.99	3.69

Continued on next page

tags.dataset	acc_CfFCN	runtime_CfFCN	memory_CfFCN
MiddlePhalanxTW	154.49	103.17	3.69
MixedShapesRegularTrain	81.85	103.07	3.69
MixedShapesSmallTrain	80.96	104.45	3.69
MoteStrain	50.92	104.87	3.69
NonInvasiveFetalECGThorax1	101.87	103.74	3.69
NonInvasiveFetalECGThorax2	102.42	109.45	3.69
OSULeaf	80.88	103.52	3.69
OliveOil	101.39	103.95	3.69
PLAID	-	-	3.69
PhalangesOutlinesCorrect	69.44	108.20	3.69
Phoneme	297.10	102.22	3.69
PickupGestureWiimoteZ	-	-	3.69
PigAirwayPressure	569.83	101.61	3.69
PigArtPressure	99.42	102.84	3.69
PigCVP	118.06	103.28	3.69
Plane	83.73	101.48	3.69
PowerCons	57.92	103.66	3.69
ProximalPhalanxOutlineAgeGroup	79.24	112.35	3.69
ProximalPhalanxOutlineCorrect	75.44	102.45	3.69
ProximalPhalanxTW	107.99	112.06	3.69
RefrigerationDevices	125.31	104.87	3.69
Rock	118.67	102.47	3.69
ScreenType	100.06	103.31	3.69
SemgHandGenderCh2	50.27	103.03	3.69
SemgHandMovementCh2	175.07	102.35	3.69
SemgHandSubjectCh2	104.55	103.48	3.69
ShakeGestureWiimoteZ	-	-	3.69
ShapeletSim	70.87	103.31	3.69
ShapesAll	109.99	104.92	3.69
SmallKitchenAppliances	85.79	103.73	3.69
SmoothSubspace	66.12	111.00	3.69
SonyAIBORobotSurface1	47.74	107.08	3.69
SonyAIBORobotSurface2	53.39	107.58	3.69
StarLightCurves	70.15	105.40	3.69
Strawberry	54.21	106.75	3.69
SwedishLeaf	96.48	106.36	3.69
Symbols	84.95	105.04	3.69
SyntheticControl	82.04	109.44	3.69
ToeSegmentation1	52.01	106.24	3.69
ToeSegmentation2	34.95	106.27	3.69
Trace	69.80	108.45	3.69
TwoLeadECG	50.02	107.86	3.69
TwoPatterns	80.33	110.31	3.69
UMD	65.26	107.75	3.69
UWaveGestureLibraryAll	102.41	105.39	3.69
UWaveGestureLibraryX	116.08	101.58	3.69
UWaveGestureLibraryY	136.19	99.79	3.69
UWaveGestureLibraryZ	120.41	100.26	3.69
Wafer	89.46	100.11	3.69
Wine	81.82	100.17	3.69
WordSynonyms	171.08	100.85	3.69
Worms	101.59	100.57	3.69
WormsTwoClass	65.38	100.20	3.69

Continued on next page

	acc_CfFCN	runtime_CfFCN	memory_CfFCN
tags.dataset			
Yoga	62.28	99.99	3.69
Durchschnitt	89.70	105.78	3.69

Tabelle A.5: Prozentuales Verhältnis von Genauigkeit, Laufzeit und Speicher zwischen CfFCN und full-precision-CfFCN (100% entspricht dem Wert des full-precision-CfFCNs).

	acc_CfResNet	runtime_CfResNet	memory_CfResNet
tags.dataset			
ACSF1	98.25	100.18	3.79
Adiac	116.77	99.58	3.79
AllGestureWiimoteX	-	-	3.79
AllGestureWiimoteY	-	-	3.79
AllGestureWiimoteZ	-	-	3.79
ArrowHead	79.58	100.61	3.79
BME	66.76	100.42	3.79
Beef	96.64	100.13	3.79
BeetleFly	58.82	100.51	3.79
BirdChicken	56.82	100.29	3.79
CBF	64.69	100.63	3.79
Car	78.91	100.18	3.79
Chinatown	64.96	100.60	3.79
ChlorineConcentration	74.48	100.84	3.79
CinCECGTorso	86.52	100.42	3.79
Coffee	47.86	100.57	3.79
Computers	62.07	100.71	3.79
CricketX	114.73	100.45	3.79
CricketY	113.13	100.10	3.79
CricketZ	113.35	100.81	3.79
Crop	128.99	100.68	3.79
DiatomSizeReduction	243.04	100.53	3.79
DistalPhalanxOutlineAgeGroup	92.25	100.46	3.79
DistalPhalanxOutlineCorrect	71.40	100.25	3.79
DistalPhalanxTW	119.45	100.82	3.79
DodgerLoopDay	-	-	3.79
DodgerLoopGame	-	-	3.79
DodgerLoopWeekend	-	-	3.79
ECG200	66.82	112.05	3.79
ECG5000	86.27	112.08	3.79
ECGFiveDays	51.68	113.46	3.79
EOGHorizontalSignal	152.92	101.73	3.79
EOGVerticalSignal	205.85	101.44	3.79
Earthquakes	105.05	102.96	3.79
ElectricDevices	117.76	108.18	3.79
EthanolLevel	92.25	100.93	3.79
FaceAll	107.13	100.30	3.79
FaceFour	76.43	110.47	3.79
FacesUCR	97.31	108.42	3.79
FiftyWords	132.39	107.09	3.79
Fish	87.41	106.05	3.79
FordA	53.02	107.57	3.79
FordB	61.37	110.94	3.79
FreezerRegularTrain	50.08	107.03	3.79

Continued on next page

tags.dataset	acc_CfResNet	runtime_CfResNet	memory_CfResNet
FreezerSmallTrain	60.08	111.12	3.79
Fungi	532.32	111.93	3.79
GestureMidAirD1	-	-	3.79
GestureMidAirD2	-	-	3.79
GestureMidAirD3	-	-	3.79
GesturePebbleZ1	-	-	3.79
GesturePebbleZ2	-	-	3.79
GunPoint	50.87	107.60	3.79
GunPointAgeSpan	50.29	104.47	3.79
GunPointMaleVersusFemale	48.85	102.64	3.79
GunPointOldVersusYoung	51.99	113.20	3.79
Ham	66.33	111.40	3.79
HandOutlines	70.12	-	3.79
Haptics	152.51	111.35	3.79
Herring	80.21	115.83	3.79
HouseTwenty	52.48	115.38	3.79
InlineSkate	212.01	113.30	3.79
InsectEPGRegularTrain	65.78	117.67	3.79
InsectEPGSmallTrain	172.28	112.01	3.79
InsectWingbeatSound	182.08	113.69	3.79
ItalyPowerDemand	51.91	112.91	3.79
LargeKitchenAppliances	73.96	114.92	3.79
Lightning2	67.23	112.02	3.79
Lightning7	100.99	112.06	3.79
Mallat	89.87	114.53	3.79
Meat	65.10	111.18	3.79
MedicalImages	117.15	111.94	3.79
MelbournePedestrian	-	-	3.79
MiddlePhalanxOutlineAgeGroup	106.98	109.68	3.79
MiddlePhalanxOutlineCorrect	65.64	107.17	3.79
MiddlePhalanxTW	165.35	110.28	3.79
MixedShapesRegularTrain	76.98	111.61	3.79
MixedShapesSmallTrain	85.27	111.81	3.79
MoteStrain	56.66	110.10	3.79
NonInvasiveFetalECGThorax1	103.70	109.89	3.79
NonInvasiveFetalECGThorax2	103.45	110.89	3.79
OSULeaf	82.91	109.32	3.79
OliveOil	87.40	112.52	3.79
PLAID	-	-	3.79
PhalangesOutlinesCorrect	56.48	111.61	3.79
Phoneme	292.22	113.37	3.79
PickupGestureWiimoteZ	-	-	3.79
PigAirwayPressure	241.71	109.57	3.79
PigArtPressure	98.93	108.72	3.79
PigCVP	106.81	102.56	3.79
Plane	83.73	111.50	3.79
PowerCons	56.89	111.06	3.79
ProximalPhalanxOutlineAgeGroup	70.85	109.82	3.79
ProximalPhalanxOutlineCorrect	74.36	117.21	3.79
ProximalPhalanxTW	107.85	115.95	3.79
RefrigerationDevices	117.37	115.83	3.79
Rock	135.87	110.68	3.79
ScreenType	93.88	111.70	3.79

Continued on next page

tags.dataset	acc_CfResNet	runtime_CfResNet	memory_CfResNet
SemgHandGenderCh2	64.35	111.75	3.79
SemgHandMovementCh2	189.78	112.42	3.79
SemgHandSubjectCh2	98.56	112.00	3.79
ShakeGestureWiimoteZ	-	-	3.79
ShapeletSim	63.92	116.49	3.79
ShapesAll	106.15	114.42	3.79
SmallKitchenAppliances	85.32	111.30	3.79
SmoothSubspace	68.03	112.95	3.79
SonyAIBORobotSurface1	50.57	108.40	3.79
SonyAIBORobotSurface2	48.86	111.77	3.79
StarLightCurves	69.63	115.27	3.79
Strawberry	48.10	112.62	3.79
SwedishLeaf	96.96	106.06	3.79
Symbols	88.32	106.45	3.79
SyntheticControl	83.61	103.47	3.79
ToeSegmentation1	55.00	103.99	3.79
ToeSegmentation2	62.99	100.82	3.79
Trace	71.90	100.51	3.79
TwoLeadECG	49.99	100.73	3.79
TwoPatterns	75.00	100.65	3.79
UMD	67.32	100.43	3.79
UWaveGestureLibraryAll	101.65	100.15	3.79
UWaveGestureLibraryX	112.00	100.55	3.79
UWaveGestureLibraryY	128.61	100.31	3.79
UWaveGestureLibraryZ	114.40	100.16	3.79
Wafer	89.39	100.20	3.79
Wine	69.23	100.35	3.79
WordSynonyms	155.69	100.44	3.79
Worms	100.48	100.33	3.79
WormsTwoClass	63.66	100.41	3.79
Yoga	58.51	100.46	3.79
Durchschnitt	87.94	106.73	3.79

Tabelle A.6: Prozentuales Verhältnis von Genauigkeit, Laufzeit und Speicher zwischen CfResNet und full-precision-CfResNet (100% entspricht dem Wert des full-precision-CfFResNets).

Abbildungsverzeichnis

2.1 Abbildung von vier univariaten Zeitreihen aus der verwandten Arbeit [18]. . 6

2.2 ReLU-Funktion 8

2.3 Sigmoid-Funktion 8

2.4 Beispielgraph für ein MLP. Das Input Layer mit $n \in \mathbb{N}$ Neuronen verarbeitet den Eingabevektor $x = (x_1, x_2, ..., x_n)$ und gibt seine Ausgabe an das erste Hidden Layer weiter. Das neuronale Netz hat t Hidden Layers, wobei $t \geq 1 \in \mathbb{N}$ gelten muss. Das t-te Hidden Layer mit $m \in \mathbb{N}$ Neuronen erhält die Ausgabe des vorherigen Layers als Eingabe. Nach der Verarbeitung wird die Ausgabe an die letzte Schicht weitergeleitet. Die Anzahl an Neuronen $k \in \mathbb{N}$ im Output Layer bestimmt die Anzahl der Klassen des neuronalen Netzes. . 10

2.5 Dieser Baum visualisiert die Abhängigkeiten der einzelnen Variablen. In dem Beispiel wird die Abhängigkeit nur für ein fixiertes i und j vollständig modelliert, da es sonst zu unübersichtlich würde. Die Kostenfunktion C_0 subtrahiert den gewünschten Zielwert t_j von dem erhaltenden Output $a_j^{(L)}$. Der Output $a_j^{(L)}$ ist abhängig vom Input $z_j^{(L)}$ und dieser ist schlussendlich abhängig vom eigentlichen Gewicht $w_{ji}^{(L)}$. Zusätzlich ist der Input auch abhängig vom Bias $b_j^{(L)}$, sowie dem Output des vorherigen Layers $a_i^{(L-1)}$. Diese zuletzt genannte Abhängigkeit sorgt für die Rekursivität der Backpropagation. Darauf wird im weiteren Verlauf der Arbeit weiter eingegangen. 15

2.6 Beispiel einer Faltung auf einer zweidimensionale Eingabe, wie z.B. einem Bild. Der Kernel iteriert mit einer Schrittgröße von 2 und ohne Padding. Die Patches der Eingabe, hier beispielsweise x_{00}, x_{01}, x_{10}, x_{11} mit den Maßen des Kernels, werden gemäß der Gleichung (2.21) mit den Gewichtungen des Kernels verrechnet. Die einzelnen Elemente des Patches werden mit den jeweiligen Elementen des Kernels multipliziert und daraufhin aufsummiert (siehe Ausgabe/Featuremap). 17

2.7 Beispiel einer Max-Pooling-Operation mit einer Schrittgröße von 2. 18

2.8 SteSign-Funktion im Forward Pass und im Backward Pass. (Abbildung aus [50]) 20

3.1 Graph eines beispielhaften Datensatzes bei dem dieses Verfahren vorteilhaft ist. Die Störsignale am Anfang der Zeitreihe würden die Genauigkeit der Klassifizierung negativ beeinflussen, während sich das phasenabhängig diskriminierende Interval im hinteren Bereich für eine präzise Klassifizierung eignet. (Abbildung aus Arbeit [18].) 22
3.2 Ein Ausschnitt des charakteristischen Verlaufs der Klasse 27 approximiert das Shapelet gut. Dementgegen wird für die Klasse 32 eine eher schlechte Deckung festgestellt. (Abbildung aus Arbeit [18].) 23
3.3 Paarweisen Vergleich der 9 Deep-Learning Modelle der Genauigkeit auf dem univariaten UCR/UEA Archiv. Auf der X-Achse ist der durchschnittliche Rang aufgereiht. (Abbildung aus Review [15].) 24
3.4 Paarweiser Vergleich der state-of-the-art Klassifikatoren auf dem univariaten UCR/UEA Archiv. (Abbildung aus Review [15].) 25
3.5 Grafische Darstellung der MLP Architektur für TSCs. 26
3.6 Grafische Darstellung der FCN Architektur für TSCs. (Abbildung aus Review [15].) 27
3.7 Grafische Darstellung der ResNet Architektur für TSCs. (Abbildung aus Review [15].) 28

4.1 Grafische Darstellung der implementierten MLP Architektur für TSCs. . . . 33
4.2 Grafische Darstellung der implementierten FCN Architektur für TSCs. . . . 35
4.3 Grafische Darstellung der implementierten ResNet Architektur für TSCs. . . 36
4.4 Grafische Darstellung eines Dense Blocks mit 5 Schichten. Jede Schicht nimmt alle vorherigen Featuremaps als Eingabe. (Quelle [61]) 37
4.5 Grafische Darstellung der implementierten DenseNet Architektur für TSCs. 38

5.1 Beispielhafte Darstellung zur Erläuterung der Pareto-Effizienz 45
5.2 Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und Pareto-Effizienz für den Datensatz ACSF1. 46
5.3 Streudiagramm mit Genauigkeit in Prozent und Laufzeit in μs auf den Achsen und Pareto-Effizienz für den Datensatz ACSF1. 46
5.4 Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und Pareto-Effizienz für den Datensatz Worms. 47
5.5 Streudiagramm mit Genauigkeit in Prozent und Laufzeit in μs auf den Achsen und Pareto-Effizienz für den Datensatz Worms. 47
5.6 Balkendiagramm des Pareto-Effizienz-Vergleichs zwischen den Modellen, basierend auf alle 128 Datensätze. 48
5.7 Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz ACSF1. . . 49

5.8 Streudiagramm mit $Speicherbedarf{\cdot}Ungenauigkeit$ und $Laufzeit{\cdot}Ungenauigkeit$ auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz ACSF1. 49
5.9 Streudiagramm mit Genauigkeit in Prozent und Speicherbedarf in byte auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz Worms. . . 50
5.10 Streudiagramm mit $Speicherbedarf{\cdot}Ungenauigkeit$ und $Laufzeit{\cdot}Ungenauigkeit$auf den Achsen und gewichteter Pareto-Effizienz für den Datensatz Worms. . . 50
5.11 Balkendiagramm des gewichteten Pareto-Effizienz-Vergleichs zwischen den Modellen, basierend auf alle 128 Datensätze. 50
5.12 Vergleich der Genauigkeit zwischen DenseNet und *HIVE-COTE*. 59

Algorithmenverzeichnis

Literaturverzeichnis

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Yu Cheng, Felix X. Yu, Rogério Schmidt Feris, Sanjiv Kumar, Alok N. Choudhary, and Shih-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2857–2865. IEEE Computer Society, 2015.

[3] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[4] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143, 2015.

[5] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014.

[6] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4820–4828. IEEE Computer Society, 2016.

[7] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q.

Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 1269–1277, 2014.

[8] Zheng Xu, Yen-Chang Hsu, and Jiawei Huang. Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018.

[9] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[11] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520. IEEE Computer Society, 2018.

[12] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4107–4115, 2016.

[14] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin.

In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 173–182. JMLR.org, 2016.

[15] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, Mar 2019.

[16] Tom Bannink, Arash Bakhtiari, Adam Hillier, Lukas Geiger, Tim de Bruin, Leon Overweel, Jelmer Neeven, and Koen Helwegen. Larq compute engine: Design, benchmark, and deploy state-of-the-art binarized neural networks. *CoRR*, abs/2011.09398, 2020.

[17] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Back to simplicity: How to train accurate bnns from scratch?, 2019.

[18] Anthony J. Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 31(3):606–660, 2017.

[19] Anthony J. Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with COTE: the collective of transformation-based ensembles. *IEEE Trans. Knowl. Data Eng.*, 27(9):2522–2535, 2015.

[20] Jason Lines, Sarah Taylor, and Anthony J. Bagnall. HIVE-COTE: the hierarchical vote collective of transformation-based ensembles for time series classification. In Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 1041–1046. IEEE Computer Society, 2016.

[21] Henry Friday Nweke, Ying Wah Teh, Mohammed Ali Al-garadi, and Uzoma Rita Alo. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Expert Systems with Applications*, 105:233–261, 2018.

[22] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, Mar 2019.

[23] Gian Antonio Susto, Angelo Cenedese, and Matteo Terzi. *Time-Series Classification Methods: Review and Applications to Power Systems Data*, pages 179–220. 01 2018.

[24] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive, 2019.

[25] *Klassifikationsverfahren*, pages 173–206. Gabler, Wiesbaden, 2008.

[26] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[27] B. YEGNANARAYANA. *ARTIFICIAL NEURAL NETWORKS*. PHI Learning, 2009.

[28] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[29] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[30] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[31] Sandro Skansi. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition, 2018.

[32] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[34] Kim Ryong, Park Kyung-Hye, and Young-Kuk Kim. Improving similarity measurement of user's rating value using sigmoid function in personalization system. In *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*, IMCOM '16, New York, NY, USA, 2016. Association for Computing Machinery.

[35] John Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990.

[36] Murat Hüsnü Sazli. A brief review of feed-forward neural networks. 2006.

[37] Russell D. Reed and Robert J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, 1999.

[38] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2nd edition, 2019.

[39] Michael A. Nielsen. Neural networks and deep learning, 2018.

[40] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019.

[41] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *Machine Learning Definition and Basics*, pages 1–17. 05 2019.

[42] Yann Lecun and Yoshua Bengio. *Convolutional Networks for Images, Speech and Time Series*, pages 255–258. The MIT Press, 1995.

[43] Liang Zheng, Yali Zhao, Shengjin Wang, Jingdong Wang, and Qi Tian. Good practice in cnn feature transfer, 2016.

[44] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1337–1345, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[45] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[46] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network, 2017.

[47] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016.

[48] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm, 2018.

[49] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

[50] Larq quantizers - stesign. `https://docs.larq.dev/larq/api/quantizers/#stesign`.

[51] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[52] Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29, 06 2014.

[53] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28, 05 2013.

[54] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585, 2017.

[55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[56] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the Institute of Radio Engineers*, 86(11):2278–2323, 1998.

[60] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[61] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[62] Nvidia dgx a-100. `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/A100-PCIE-Prduct-Brief.pdf`.

[63] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[64] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, and Corey Zumar. Developments in mlflow: A system to accelerate the machine learning lifecycle. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, DEEM'20, New York, NY, USA, 2020. Association for Computing Machinery.

[65] A. Charnes, W. W. Cooper, and E. Rhodes. Measuring the efficiency of decision making units. *European Journal of Operational Research*, 2(6):429–444, 1978.

[66] Hanzhang Hu, Debadeepta Dey, Allison Del Giorno, Martial Hebert, and J. Andrew Bagnell. Log-densenet: How to sparsify a densenet, 2017.