HAW HAMBURG

BACHELOR THESIS
Simon Martin Egge

Rekonstruktion ultrakurzer Laserpulse: Anwendung von Neuronalen Netzen zur Analyse von FROG-Traces

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Simon Martin Egge

Rekonstruktion ultrakurzer Laserpulse: Anwendung von Neuronalen Netzen zur Analyse von FROG-Traces

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus Jünemann
Zweitgutachter: Priv. Doz. Dr. habil. Tim Laarmann

Eingereicht am: 31.03.2025

Simon Martin Egge

Thema der Arbeit

Rekonstruktion ultrakurzer Laserpulse: Anwendung von Neuronalen Netzen zur Analyse von FROG-Traces

Stichworte

Neuronale Netze, Kostenfunktion, Netz mit mehreren Eingängen

Kurzzusammenfassung

Ziel dieser Bachelorarbeit ist die Analyse von FROG-Traces mittels neuronaler Netze. Es werden verschiedene Netzarchitekturen hinsichtlich ihrer Fähigkeit bewertet, das ZBP aus den gegeben FROG-Traces vorherzusagen. Das beste Netz wird zur Rekonstruktion des Amplitudenverlaufs im Zeitbereich eingesetzt. Es finden fünf verschiedene Kostenfunktionen Einsatz. Anschließend wird der Phasenverlauf im Wellenlängenbereich rekonstruiert. Zuletzt wird ein Netz mit zwei Eingängen realisiert. Dies ermöglicht die Vorhersage des Phasenverlaufs aus dem gegebenen FROG-Trace sowie dem Amplitudenverlauf im Wellenlängenbereich.

Simon Martin Egge

Title of Thesis

Reconstruction of ultrashort laser pulses: application of neural networks to analyze FROG traces

Keywords

Neural network, cost function, multiple-input Net

Abstract

The aim of this bachelorthesis is to analyse FROG traces using neural networks. Different network architectures are evaluated with respect to their ability to predict the ZBP from the given FROG traces. The best mesh is then used to reconstruct the amplitude response in the time domain. Five different cost functions are used. The phase response is then reconstructed in the wavelength domain. Finally, a mesh with two inputs is realised. This enables the prediction of the phase response from the given FROG trace and the amplitude response in the wavelength domain.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

DESY Deutsches Elektronen-Synchrotron.

FROG Frequency-resolved optical gating.

MAE Mean-Absolut-Error.

MSE Mean-Squared-Error.

ZBP Zeit-Bandbreite-Produkt.

Glossar

Adam Adam ist ein Optimierungsalgoritumus für das Training neuronaler Nezte. Adam passt die Lernrate für jeden Parameter des Netzes individuell an, basierend auf dem Mittelwert der Gradienten und dem Mittelwert der quadrierten Gradienten.

chirp Bezeichnet eine zeitliche Veränderung der Frequenz eines Pulses. Ein positiver Chirp liegt vor, wenn die Frequenz mit der Zeit ansteigt, ein negativer Chirp bei fallender Frequenz.

Epoche Eine Epoche bezeichnet den vollständigen Durchlauf des gesamten Trainingsdatensatzes.

Hyperparameter Beeinflusst das Trainingsverhlaten des Netzes und kann durch optimien über mehrere Trainingszyklen das Trainingsergebnis verbessern.

Minibatch Ein Minibatch ist ein kleiner Teil eines Datensatzes.

Pulsformer Ein Pulsformer wird verwendet, um die zeitliche Struktur eines ultrakurzern Laserpulses gezielt zu ändern.

1 Zielsetzung

Diese Arbeit beschäftigt sich mit dem Frequency-resolved optical gating (FROG)-Verfahren, einem Messverfahren zur Rekonstruktion von femtosekunden Laserpulsen. Aktuell werden die gemessenen Spektren mit der „Swamp-Software“ analysiert, die auf einem iterativen Verfahren basiert. Im Rahmen eines übergeordneten Projektes, welches einen Reinforcement-Learning-Loop zur Steuerung eines Puls-Shapers entwickelt, wird untersucht, welche neuronalen Netzarchitekturen sich besonders zur Analyse der Spektren und zur Ableitung des Amplitudenverlaufs, sowie des Phasenverlaufs eignen. Hierbei soll der Einsatz eines neuronalen Netzes als alternative Methode etabliert werden.

Es soll ein neuronales Netz ermittelt werden, welches besonders gut für die Analyse der Spektren geeignet ist. Dieses Netz soll dafür genutzt werden, den Amplitudenverlauf sowie den Phasenverlauf des Pulses zu ermitteln. Ziel ist es, ein schnelleres und rechenärmeres Verfahren zu ermitteln, dass das Analysieren dieser Spektren ermöglicht. Unterschiedliche Eigenschaften des Pulses wie das Zeit-Bandbreite-Produkt (ZBP), der Amplitudenverlauf und der Phasenverlauf sollen jeweils von separaten und speziell dafür trainierten Netzen vorhergesagt werden.

Die Auswahl verschiedener Netze wird anhand der Komplexität sowie der Anwendungsbereich der Netze getroffen. Anschließend werden die Netze auf Grundlage mathematisch generierter FROG-Traces trainiert. Das Netz soll in der Lage sein, den Amplitudenverlauf des Pulses mit ausreichender Genauigkeit vorherzusagen. Ebenfalls soll ein weiteres Netz mit ähnlichem Aufbau trainiert werden, um die Phase des Pulses vorherzusagen. Um weitere, zur Verfügung stehende, Informationen in die Evaluierung einfließen zu lassen, wird ein Netz mit mehreren Eingängen aufgebaut, in dem das vortrainierte Netz implementiert werden soll. Es soll analysiert werden, wie sehr sich die Genauigkeit der Vorhersage durch das Implementieren eines Netzes mit mehreren Eingängen verbessert.

Die Arbeit umfasst keinen physikalischen Aufbau und das Netz wird nicht mit gemessenen Daten trainiert, da das Erzeugen dieser Messdaten zu zeitintensiv ist.

Die Ergebnisse dieser Arbeit finden beim Deutsches Elektronen-Synchrotron (DESY) Anwendung. Dort wird mit femtosekunden Laserpulsen gearbeitet und ein schnelles und fehlerfreies Analysieren dieser FROG-Traces ist von großer Relevanz. Durch den Abschluss dieser Arbeit soll ein effizienteres Arbeiten mit dem FROG-Verfahren ermöglicht werden. Ziel ist es, eine schnellere und verlässlichere Einschätzung der Pulsform zu erhalten. Dadurch kann der Arbeitsprozess beschleunigt und reibungsloser gestaltet werden.

2 Theoretische Grundlagen

Die in diesem Kapitel vorgestellten Grundlagen dienen dazu, den Kontext und die methodische Basis für die folgende Untersuchung der Rekonstruktion ultrakurzer Laserpulse zu schaffen. Zunächst werden die Grundlagen des FROG Verfahrens, sowie die Berechnung und die Relevanz des ZBP erläutert. Des Weiteren wird ein Überblick über neuronale Netze veranschaulicht. Es wird der Trainingsablauf skizziert und gezeigt, weshalb sich neuronale Netze für die Auswertung von FROG-Traces eignen können. Dabei wird Fokus auf das verwendete, überwachte Lernen gelegt. Des Weiteren werden verschiedene Lernraten-Funktionen mit deren Vor- und Nachteilen vorgestellt. Darauf aufbauend wird eine Auswahl von verwendeten Kostenfunktionen und deren Funktionsweise vorgestellt. Zum Bewerten und Messen werden die verschiedene Fehlerarten, die bei neuronalen Netzen Anwendung finden, erläutert, gefolgt von den Begriffen der Unteranpassung und Überanpassung. Im letzten Teil der theoretischen Grundlagen wird die Technik des Wissenstransfers behandelt. Damit einhergehend werden verschiedene in der Arbeit verwendeten neuronale Netze behandelt.

2.1 Grundlagen Ultrakurzpulslaser

Als Ultrakurzpulslaser werden Laserpulse bezeichnet, deren zeitliche Ausdehnung nur einige Femtosekunden entspricht. Die Bedeutung dieser Impulse hat im Laufe der Zeit zugenommen, da die wissenschaftlichen Methoden fortschreitend weiterentwickelt wurden und es mittlerweile möglich ist, sehr kleine und sich schnell bewegende Objekte zu untersuchen. Mit diesen Impulsen lassen sich Ereignisse, die nur auf einen sehr kurzen Zeitraum beschenkt sind, aufzeichnen. Man stelle sich eine Stoppuhr vor, die nur in ganzen Minuten messen kann. Damit soll die Zeit gemessen werden, die ein Stein braucht, um von 20 Metern Höhe auf den Boden zu fallen. Diese Messung wird kein zufriedenstellendes Ergebnis liefern, da die Zeitskala auf der gemessen wird, viel zu groß für das

Event, was aufgezeichnet werden soll, ist. Ähnlich verhält es sich auch mit den femtosekunden Lasern. Diese sollen eine so kleine zeitliche Auflösung liefern, um damit Events aufzuzeichnen, die in der gleichen zeitlichen Größenordnung stattfinden. Mit dieser Methode wird das Vibrieren von Molekülen oder auch der Ablauf von chemischen Prozessen aufgezeichnet [17, 13]. Um diese Prozesse möglichst gut aufzeichnen zu können, muss ein bekannter Laserpuls verwendet werden. Es ist dabei wichtig festzustellen, ob der Puls einen chirp hat. Dieser beeinflusst das Ergebnis der Messung. Um herauszufinden, welche Eigenschaften dieser Puls hat, wird FROG verwendet.[9]

2.2 Frequency-Resolved Optical Gating (FROG)

FROG ist eine Messtechnik, um Laserpulse im Femtosekundenbereich zu messen. In diesen Zeitskalen gibt es keine Messgeräte, die schnell genug sind, um den zeitlichen Verlauf eines solchen Impulses aufzuzeichnen. Daher muss eine Messung im Frequenzbereich durchgeführt werden. Der Puls $E(t)$ wird durch einen Strahlteiler geleitet (siehe: Abbildung 2.1), dieser teilt den Puls in zwei identische Pulse auf. Einer der beiden erzeugten Pulse wird durch eine Verzögerungsstufe geleitet. Diese verzögert den Puls um τ. Anschließend werden beide Pulse durch eine Linse auf einen nicht linearen Kristall fokussiert. In dem nicht linearen Kristall werden die Pulse überlagert. Durch die Überlagerung der beiden Pulse werden diese autokorreliert. Es handelt sich bei diesem nichtlinearen Kristall um einen, der zu Frequenzverdopplung führt. Am Ende der Strecke wird das Spektrum der Autokorrelation aufgezeichnet. Durch Variation der Verzögerung τ entsteht eine zweidimensionale Messung. [17]

Die FROG-Messung erzeugt somit Informationen über die Spektralkomponenten in einer zeitlichen Auflösung und erstellt ein zweidimensionales Spektrum, welches in der einen Dimension die Verzögerung τ wiedergibt und in der anderen das Spektrum der Autokorrelation. Ein solches Spektrum ist in Abbildung 2.2 dargestellt.

2.2.1 Zeit-Bandbreite-Produkt

Das ZBP beschreibt den Zusammenhang zwischen der zeitlichen Ausbreitung Δt und der Bandbreite im Frequenzbereich Δf. Dieses Produkt ist minimal, wenn keine Phasenverschiebung vorliegt. Dann erreicht der Puls die minimale zeitliche Ausbreitung mit dem minimalen Frequenzspektrum. Diese Pulse sind optimal, da sie durch ihre minimale

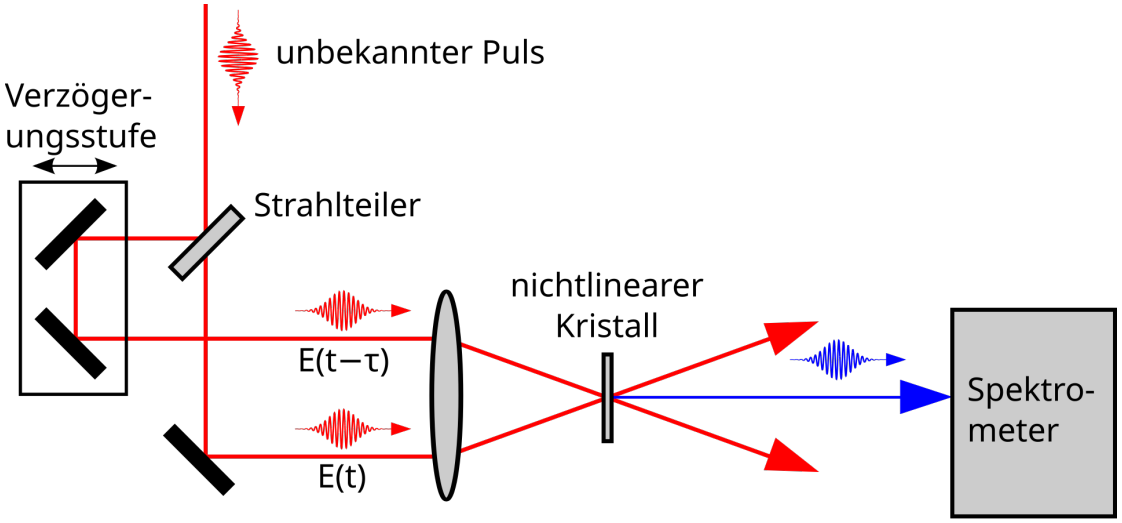


Abbildung 2.1: Aufbau einer FROG-Messstrecke [18]

Breite im Zeitbereich die bestmögliche Auflösung realisieren. Sobald eine Phasenänderung über die Zeit eingeführt wird, verbreitert sich der Puls. Diese Verbreiterung sorgt für ein höheres ZBP. Das ZBP ist einheitenlos.

2.3 Neuronales Netz

Neuronale Netze sind Computermodelle, die von der Funktionsweise des Gehirns inspiriert sind. Sie enthalten Neuronen, die miteinander verbunden sind und in Schichten aufgeteilt sind. Zwischen den Schichten sind Nichtlinearitäten eingebaut, die die Aktivierung eines biologischen Neurons nachstellen soll. Der Zusammenschluss dieser Schichten wird als Neuronales Netz bezeichnet. Durch das Anpassen der trainierbaren Parameter ist das Netz in der Lage, verschiedene Muster aus Daten zu extrahieren und aufgrund dieser Muster Vorhersagen über unbekannte Daten zu treffen. Das Bewerten dern Vorhersage wird mit einer Kostenfunktion gemacht. Anschließend wird aufgrund der berechneten Kosten mithilfe von automatischen Differenzieren die trainierbaren Parameter angepasst, um so die getroffene Vorhersage zu optimieren und den Wert der Kostenfunktion zu minimieren. Der Aufbau neuronaler Netze kann in vielfältiger Form erfolgen und wird an die jeweilige Aufgabenstellung angepasst. Somit kann ein Netz auf die gegebenen Bedingungen angepasst werden. Der strukturelle Aufbau eines Netzes besteht immer aus einer Eingabeschicht, in der die zu verarbeitenden Daten eingegeben werden, aus mehre-

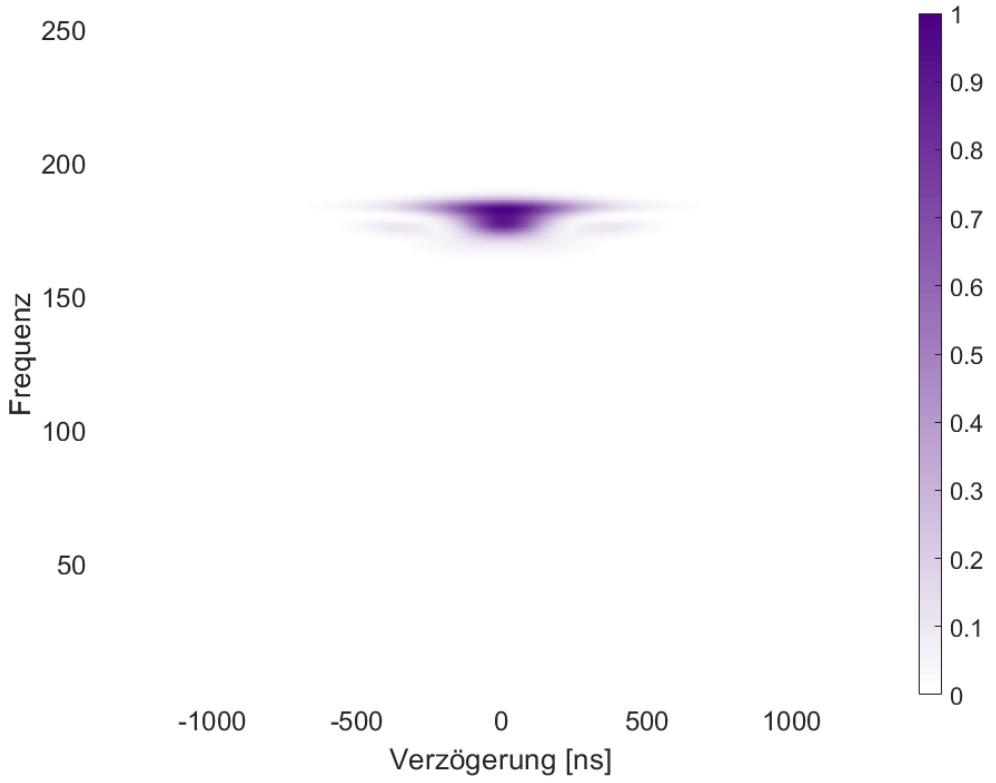


Abbildung 2.2: Beispiel Spektrum einer FROG-Simulation

ren verborgenen Schichten, die die Funktionalität des Netzes enthalten, sowie aus einer Ausgabeschicht, die die Vorhersage des Netzes ausgibt. Die Komplexität des Netzes wird durch die Anzahl, sowie die Art der verwendeten Schichten in den verborgenen Schichten bestimmt. Es ist durchaus möglich, dass ein Netz über mehrere Eingänge sowie Ausgänge verfügt.

Neuronale Netze haben bereits in einigen Anwendungsbereichen erstaunliche Ergebnisse erzielt. Ein großes Feld von neuronalen Netzen ist die Bildverarbeitung, die in der Lage ist, Objekte auf Bildern zu erkennen und zu klassifizieren.

Durch ihre modulare Aufbauweise können neuronale Netze sehr gut an die gegebenen Probleme angepasst werden. Um jedoch erfolgreich zu trainieren, wird oft eine sehr große Datenmenge benötigt. Darüber hinaus ist häufig nicht nachvollziehbar, auf welchen Merkmalen die Entscheidung des Netzes basiert. Die Vorhersage findet nicht transparent statt und kann immer fehlerbehaftet sein.

Durch ihre Fähigkeit, aus Daten zu lernen und sich an neue Informationen anzupassen, sind neuronale Netze ein zentraler Bestandteil in der Entwicklung neuer Technologien geworden. [8, 3]

2.3.1 Trainingsablauf von überwachten Lernen

Der Trainingsablauf eines neuronalen Netzes beschreibt den Prozess, in dem das Netz seine Parameter anpasst, um ein besseres Ergebnis bei der zu erledigenden Aufgabe zu erzielen. Beim Start des Trainings müssen die trainierbaren Parameter des Netzes mit einem Wert initialisiert werden. Dies kann durch zufälliges Auswählen von Werten geschehen oder es können die Parameter eines bereits trainierten Netzes geladen werden. Zum Start eines Trainingsdurchlaufs wird eine Eingabe auf das Netz gegeben. Diese Eingabe wird durch das ganze Netz durchgeführt, indem die Schichten sequenziell ausgeführt werden. Die Ausgabe des Netzes wird mit dem vorgegebenen Label verglichen und der Fehler wird mithilfe der Kostenfunktion ermittelt. Auf Grundlage dieses Fehlers wird mithilfe automatischen Differenzierens im Rückwärtsmodus ermittelt, wie die trainierbaren Parameter angepasst werden soll, um bessere Vorhersagen zu erzielen. Die Anpassung wird mithilfe eines Optimierungsalgorithmus durchgeführt. Die Stärke der Anpassung kann im Laufe des Trainings mit einer Lernraten-Funktion angepasst werden. Die Lernrate wird im Verlauf des Trainings reduziert, um so eine präzisere Feinabstimmung am Ende des Trainings zu ermöglichen. Der beschriebene Trainingsablauf wird in Abbildung 2.3 dargestellt.

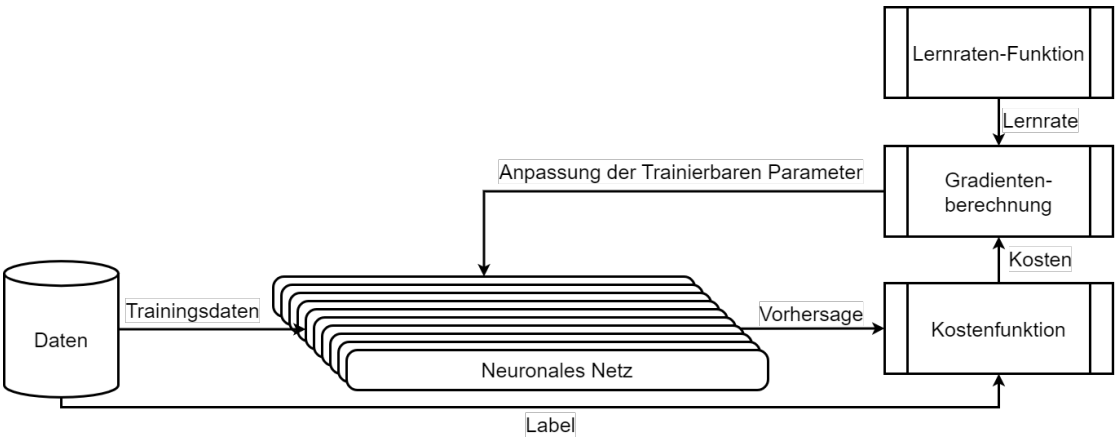


Abbildung 2.3: Trainingsablauf mit überwachten Lernen

2.3.2 Kostenfunktion

Die Kostenfunktion ist eine mathematische Funktion die den Fehler zwischen der Vorhersage des Netzes und dem Label bestimmt. Dieser Fehler kann auf viele verschiedene Arten berechnet werden und das verwendete Verfahren ist abhängig von der zu erfüllenden Aufgabe. Diese Funktion spielt eine elementare Rolle in dem effektiven Training eines neuronalen Netzes. Ziel des Trainings ist es, den Wert der Kostenfunktion zu minimieren. Der Fehler wird immer auf eine skalare Größe reduziert, an der die Qualität der Vorhersage gemessen wird.

Zur Berechnung des Fehlers benötigt die Kostenfunktion das Label und die Vorhersage des Netzes. Mit diesen beiden Werten wird ermittelt, wie gut die Vorhersage dem Label entspricht. Bei dieser Ermittlung ist es möglich, verschiedene Schwerpunkte zu legen, um ein gewünschtes Verhalten des Netzes zu erhalten. Zur effizienten Anpassung der trainierbaren Parameter ist es notwendig, die Ableitung der Kostenfunktion bestimmen zu können. In den nachfolgenden Aufführungen von Kostenfunktionen wird y als Labelvektor angegeben und $\hat{y}$ stellt die entsprechende Vorhersage dar.

Mean-Squared-Error

Diese Kostenfunktion verwendet den Mittelwert der quadratischen Abweichung. Diese Art der Fehlerermittlung ist sehr verbreitet. Die Gleichung zum Berechnen des Mean-Squared-Error (MSE) ist:

$$\text{MSE} = \frac{1}{n} \sum_{i} (y_i - \hat{y}_i)^2$$

Es wird das Quadrat der Abweichung aufsummiert, damit sich positive und negative Abweichungen nicht aufheben. Dies führt allerdings dazu, dass eventuelle Ausreißer einen sehr starken Einfluss auf den MSE haben. Wenn die Ausreißer nicht normalverteilt sind, kann es zu einer verzerrten Berechnung des MSE kommen. Diese Funktion erweist sich als sehr gut differenzierbar, was zu einem leichteren Ermitteln der Gradienten führt. Da diese Funktion keine Hyperparameter besitzt, ist sie sehr allgemeingültig und kann in vielen Bereichen eingesetzt werden. Die hohe Sensibilität kann nach Anwendungsfall einen positiven, wie auch einen negativen Einfluss haben.

Mean-Absolut-Error

Eine Kostenfunktion, die den Mean-Absolut-Error (MAE) bestimmt, weicht nur minimal von dem MSE ab. Anstatt den Fehler zu quadrieren, wird der Absolutwert der Abweichung bestimmt. Dies erfüllt dieselbe Funktionalität und sorgt dafür, dass sich positive und negative Fehler nicht gegenseitig auslöschen. Da der Fehler nicht quadriert wird, werden alle Fehler linear gewichtet.

$$\text{MAE} = \frac{1}{n} \sum_{i} |y_i - \hat{y_i}|$$

Durch die Verwendung des Absolutwertes ist die Stetigkeit der Ableitung des Fehlers nur bei einem Fehler von null nicht gegeben. Programme mit automatischem Differenzieren erkennen diesen Sonderfall und behandeln ihn intern, wodurch in der Anwendung keine Probleme entstehen. Der MAE ist ein sehr allgemeingültiger Fehler, der in vielen Fällen Anwendung finden kann. Auch hier gibt es keine Hyperparameter, die ein Einstellen benötigen. Durch seine Robustheit gegen Ausreißer wird der Einsatz von MAE ermöglicht, wo der MSE nicht angebracht ist.

Huber

Die Huber-Kostenfunktion wurde von Mathematiker Peter J. Huber im Jahr 1963 eingeführt und vereint die Vorteile von MSE und MAE. Sie ist insbesondere für robuste Regression geeignet, da sie die Auswirkung von Ausreißern reduziert. Das Verhalten der Funktion wird über einen Hyperparameter δ gesteuert. Diese Kostenfunktion verändert ihr Verhalten in Abhängigkeit des Fehlers a in Referenz zu dem Hyperparameter δ. Wenn der Betrag des Fehlers kleiner ist als δ, dann verhält sich die Kostenfunktion annähernd wie MSE. Sollte der Betrag des Fehlers größer sein als δ, dann verhält sich die Kostenfunktion annähernd wie MAE. Durch die Anwendung des quadratischen Fehlers bei kleinem a entstehen beim Ableiten dieser Funktion keine Sprünge. Der Fehler konvergiert langsamer, was zu mehr Stabilität am Ende des Trainings führt. Solange der Fehler hoch ist, wird der Absolutwert verwendet. Dadurch ist der Fehler nicht so anfällig gegenüber

Ausreißern und ist stabiler zu Beginn des Trainings.

$$a = y - \hat{y}$$
$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{für } |a| \leq \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right) & \text{sonst.} \end{cases}$$

Durch das Verwenden des quadratischen Fehlers bei kleinem a, ist die Funktion gut differenzierbar. Diese Funktion vereint das Beste von MSE und MAE. Dabei wird notwendigerweise ein Hyperparameter eingeführt, dessen Auswahl kritisch für die Funktionalität und Leistung der Funktion ist. Dieser Hyperparameter ermöglicht eine bessere Feinabstimmung, allerdings auch einen Parameter mehr, der den Optimierungsprozess des Netzes verlängert.

Gauss-Kostenfunktion

Die Gauss-Kostenfunktion ist durch die Analyse der Label entstanden. Die Amplitudenverläufe sind sehr zentriert und bestehen zum Großteil aus einem Puls in der Mitte. Diesen Umstand nutzt die Gauss-Kostenfunktion aus, in dem der absolute Fehler bestimmt wird und anschließend mit einer Gauskurve gewichtet wird. So werden Fehler, die sich in der Mitte der Vorhersage befinden, stärker gewichtet als Fehler, die sich am Rand der Vorhersage befinden. Das soll dazu führen, dass die Vorhersagengenauigkeit in der Mitte des Amplitudenverlaufes höher ist als am Rand. Es handelt sich somit um eine gewichtete MAE Kostenfunktion. Der Parameter x ist ein Vektor von äquidistanten Punkten, welche die Länge der Vorhersage des Netzes entspricht. Mit dem Parameter μ wird die Gauss-Kurve zentriert. Dieser Wert sollte nicht angepasst werden und beträgt die Hälfte der Länge von x. Der Parameter σ ist ein Hyperparameter und bestimmt die Steilheit der Gauss-Kurve. Damit kann festgelegt werden, wie Stark der Fehler auf die Mitte fokussiert werden soll. Als Richtwert kann ein Zehntel der Länge von x genommen werden.

$$\text{loss} = |y - \hat{y}| \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.1}$$

Da es sich hier um eine grundlegende Fehlerbestimmung mit dem Absolutwert handelt, ist die Differenzierbarkeit gegeben. Das Gewichten mit der Gaussverteilung führt keine Probleme der Differenzierbarkeit ein, da die Werte der Vorhersage mit den Werten der

Gauss-Kurve multipliziert werden. Diese Funktion gewichtet Fehler, die in der Mitte der Vorhersage entstehen, stärker als Fehler die am Rand entstehen. Die Gewichtung der Fehler mit einer Gaußverteilung bewirkt, dass Vorhersagen am Rand des Pulsverlaufs weniger stark optimiert werden als im Zentrum. Es führt ebenfalls dazu, dass Rauschen am Rand der Vorhersage weniger unterdrückt wird. Da die Trainingsdaten sehr mittig verteilt sind, sollte diese Funktion zu einem schnelleren Training führen und zuverlässigen Vorhersagen in der Mitte des Amplitudenverlaufes erbringen. Durch das Einführen der Gauss-Verteilung wird ein neuer Hyperparemeter σ eingeführt, mit dem die Breite der besagten Verteilung festlegen kann.

Smooth-Kostenfunktion

Diese Funktion hat das Ziel, starkes Rauschen zu unterdrücken und das Netz somit dazu zu bringen, einen glatten Amplitudenverlauf vorherzusagen. Diese Glättung soll dazu führen, dass der Fehler, der durch das Rauschen hinzugefügt wird, möglichst schnell unterbunden wird. Starkes Rauschen wird durch das Berechnen der Ableitung ermittelt. Da der Amplitudenverlauf eine sehr starke Steigung aufweist, soll diese nicht zum Fehler beitragen. Daher wird der Fehler durch das Label geteilt. So fließt die hohe Steigung des Labels nicht mit in den Fehler ein. Um eine ordentliche Vorhersage der Amplitude zu ermöglichen, ist eine Fehlerbestimmung mit dem gegebenen Label notwendig. Dazu wird der MSE verwendet und anschließend mit dem berechneten Fehler der Ableitung aufsummiert. Die Aufteilung der Zusammensetzung wird über den Hyperparameter a bestimmt.

Die Ableitung wird durch ein einfaches numerisches Verfahren bestimmt, indem die Differenz zwischen einem Datenpunkt und dem nächsten Datenpunkt bestimmt wird. In der praktischen Anwendung muss die Funktion mit dem Skalar k abgeglichen werden, damit die beiden Summen sich ungefähr in einen ähnlichen nurmerischen Bereich befinden. Über den Hyperparameter a kann prozentual angegeben werden, welcher Teil der Funktion stärker gewichtet werden soll.

$$\text{loss} = a \cdot \sum_i (y_i - \hat{y}_i)^2 + (1-a) \cdot k \cdot \sum_i^{I-1} \frac{(\hat{y}_{(i+1)} - \hat{y}_i)^2}{y_i + \epsilon} \tag{2.2}$$

Da die Ableitung über ein einfaches Differenzverfahren bestimmt wurde, ist die Funktion gut ableitbar. Auch durch diese Funktion werden die neue Hyperparameter a und k

eingeführt, die ein genaueres Steuern des Trainingsverhaltens ermöglichen, den Optimierungsprozess allerdings verlängern. Durch den Hyperparameter a wird die Gewichtung des Ableitungsthermes gesteuert. Mit dem Hyperparameter k wird das Abgleichen der beiden Summen festgelegt.

phase-Kostenfunktion

Es ist auch möglich eine Kostenfunktion für ein Netz mit mehreren Eingängen zu definieren. Dabei kann die Kostenfunktion ebenfalls einen weiteren Wert übergeben bekommen. Die Phase ist nur dort sinnvoll, wo der Puls größer als null ist, daher wird der Fehler vor allem in diesem Bereich berechnet.Somit fließt der weitere Eingang mit in die Berechnung des Fehlers ein. Damit sich die Randwerte numerisch ebenfalls in einem ähnlichen Bereich befinden, fließen diese mit einer geringen Gewichtung in die Fehlerberechnung mit ein. Ein großer Werteunterschied in der Vorhersage könnte während des Trainings zu numerischer Instabilität führen.

Die angewendete Maske, die für die Gewichtung verwendet wird, beruht auf dem zweiten Eingang und legt den Fokus auf den Bereich, in dem der Amplitudenverlauf größer als $0,001$ ist. Der fokussierte Bereich wird mit elf multipliziert, wohingegen der außerhalb liegende Bereich nur mit dem Multiplikator von eins in die Fehlerberechnung eingeht.

$$a = \text{Amplitudenverlauf im Wellenlängenbereich} \tag{2.3}$$

$$\text{Maske}_i = \begin{cases} 11, & \text{wenn } a_i > 0{,}001 \\ 1, & \text{sonst} \end{cases} \tag{2.4}$$

$$\text{loss} = \frac{1}{n} \sum_i \left((y_i - \hat{y}_i) \cdot \text{Maske}_i \right)^2 \tag{2.5}$$

Die Maske basiert auf den Eingabedaten, wird jedoch beim Ableiten nicht als differenzierbare Funktion behandelt. Dadurch fehlt dem Netz die Information, dass Maske und Eingabe in direktem Zusammenhang stehen. Für das Netz ist es somit schwierig, die Abhängigkeit zwischen der Maske und den jeweiligen Eingangsdaten zu erkennen und zu lernen. Da sich die Maske mit jedem Eingabetupel verändert, fehlt die explizite Verbindung, um diese Beziehung effizient im Trainingsprozess zu erfassen.

Diese Kostenfunktion nutzt einen Hyperparameter, der die Schwelle zur Definition der Fehlermaske festlegt. Änderungen an dieser Schwelle haben nur einen begrenzten Einfluss

auf die Funktionalität, wodurch die Bedeutung dieses Parameters relativ gering ist. Des weiteren werden die zwei Hyperparameter verwendet, die die Multiplikatoren der Maske festlegen. Diese haben einen großen Einfluss auf das Verhalten der Kostenfunktion.

2.3.3 Lernraten-Funktionen

Die Lernraten-Funktionen sind ein Hyperparameter mit dem der Trainingsprozess des Netzes gesteuert wird. Die Lernrate nimmt auf das Anpassen der trainierbaren Parameter mithilfe des Gradienten Einfluss. Die Lernrate gibt an, wie stark die trainierbaren Parameter angepasst werden sollen. Eine hohe Lernrate sorgt dafür, dass die trainierbaren Parameter stark angepasst werden, wohingegen eine niedrige Lernrate nur für eine geringe Anpassung der trainierbaren Parameter sorgt. Die Lernraten-Funktion wird für die Optimierung des Trainings verwendet. Der Einsatz adaptiver Lernraten-Funktionen beschleunigt das Training zu Beginn durch größere Parameteranpassungen und verbessert am Ende die Genauigkeit durch feinere Anpassung. Wenn die Lernrate zu klein wird, kann es zu einem Lernstopp führen, da die trainierbaren Parameter nicht mehr angepasst werden können.

Exponetieller Abfall

Eine übliche Funktion ist das Reduzieren der Lernrate mithilfe eines exponentiellen Abfalls. Dabei wird jede Iteration die Lernrate mit einem Wert $k < 1$ multipliziert. Die Größe von k ist ein Hyperparameter mit dem angepasst werden kann, wie schnell sich die Lernrate reduziert. Der Hyperparameter sollte in Betracht auf die Länge des Trainings gewählt werden, um das Erreichen zu kleiner Werte zu vermeiden, die zu einem Lernstopp führen können.

Kosinus Abfall

Eine weitere etablierte Lernraten-Funktion ist der Kosinus Abfall. In diesem Fall wird die Lernrate mithilfe dem ersten Viertel der Kosinus-Schwingung reduziert. Diese Funktion hat zu Beginn einen längeren Verlauf mit höheren Werten, in dem das Netz noch stärkere Anpassungen vornehmen kann. Zur Mitte des Trainings sinkt die Lernrate stark ab und hat anschließend einen ebenso langen Verlauf mit kleinen Werten. Diese Funktion enthält

die beiden Hyperparameter: die Lernrate zum Start und die Lernrate zum Ende. Diese beiden Werte müssen ausgewählt werden, um den Verlauf der Funktion zu definieren.

2.3.4 Gradientenabstieg

Das Anpassen der Gewichte erfolgt durch das Ermitteln der Gradienten. Diese werden mithilfe des automatischen Differenzierens im Rückwärtsmodus bestimmt und mit der Lernrate skaliert, sowie mit dem Adam Optimierer angepasst. Durch korrektes Anpassen der trainierbaren Parameter wird die Kostenfunktion minimiert. [3]

2.3.5 Trainingsfehler, Validierungsfehler und Testfehler

Es gibt drei verschiedene Fehlerarten, die in Verbindung mit neuronalen Netzen stehen. Der Unterschied dieser Fehler besteht in ihrer Nutzung, sowie in den Daten, die zur Bestimmung verwendet werden.
Der Trainingsfehler beschreibt den Fehler, der während des Trainings errechnet wird. Dieser Fehler wird mit der gegebenen Kostenfunktion berechnet. Zur Fehlerberechnung werden die Trainingsdaten mit den entsprechenden Trainingslabel verwendet. Mit dem Fehler werden die trainierbaren Parameter über den Gradientenabstieg angepasst.
Der Validierungsfehler wird auf Grundlage des Validierungsdatensatzes bestimmt. Der Validierungsfehler dient der Überwachung der Netzperformance während des Trainings. Diese Daten werden nicht zum Trainieren verwendet, sondern zeigen, wie gut das Netz unbekannte Daten einschätzt. Dabei wird zum Bestimmen des Fehlers auch üblicherweise die angewendete Kostenfunktion genutzt.
Der Testdatensatz besteht aus einem Datensatz, der weder zum Trainieren, noch zum Validieren des Netzes verwendet wird. Er besteht aus Daten, die für das Netz vollkommen unbekannt sind. Mit der Bemessung des Testfehlers kann die Einstellung der Hyperparameter bewertet werden. Dieser Fehler wird nach Beendigung des Trainings bestimmt. Zur Berechnung dieses Fehlers kann eine neue Funktion verwendet werden, die andere Schwerpunkte legt, als die im Training verwendete Kostenfunktion.

2.3.6 Über- und Unteranpassung

Tritt Überanpassung auf, hat das Modell begonnen, die Trainingsdaten und deren zugehörige Labels auswendig zu lernen. In einem solchen Fall ist der Trainingsfehler sehr

gering, während der Validierungsfehler signifikant ansteigt. Überanpassung tritt vermehrt bei Netzen mit vielen trainierbaren Parametern auf, da diese die nötige Komplexität besitzen, um Datensätze auswendig zu lernen. Ein typisches Anzeichen für Überanpassung ist daher ein sehr niedriger Trainingsfehler bei gleichzeitig deutlich höherem Validierungsfehler.

Im Gegensatz dazu deutet das Vorliegen sowohl eines hohen Validierungsfehlers als auch eines hohen Trainingsfehlers auf Unteranpassung hin. In diesem Fall ist die Modellkapazität – also die Anzahl der trainierbaren Parameter – zu gering, wodurch das Netz nicht in der Lage ist, relevante Strukturen oder Muster in den Trainingsdaten zu erkennen. Daraus resultieren ungenaue Vorhersagen und ein insgesamt hoher Fehler.

Ein wesentlicher Faktor für das Auftreten von Überanpassung oder Unteranpassung ist die Größe des verfügbaren Datensatzes. Ein kleiner Datensatz wird tendenziell schneller von einem Netz auswendig gelernt als ein größerer, da weniger Variabilität und weniger Muster vorhanden sind, die erlernt werden müssen

2.4 Wissens-Transfer

Wissens-Transfer ist eine etablierte Methode des maschinellen Lernens, bei der ein vortrainiertes Netz, das auf einem großen Datensatz D eine Aufgabe A gelernt hat, auf einen neuen, meist kleineren Datensatz D' und eine neue, aber ähnliche Aufgabe A' angepasst wird. Voraussetzung für einen erfolgreichen Transfer ist eine hohe Ähnlichkeit zwischen den Datensätzen und Aufgaben.

Diese Technik ermöglicht es, mit reduziertem Rechenaufwand und weniger Trainingsdaten dennoch gute Ergebnisse zu erzielen. Besonders relevant ist sie, da viele leistungsfähige Netze bereits auf umfangreichen öffentlichen Datensätzen trainiert wurden und dadurch effizient weiterverwendet werden können. [11].

Im Rahmen dieser Arbeit werden verschiedene vortrainierte Netze verwendet, deren Eigenschaften im Folgenden näher beschrieben werden.

YAMNet

Das YAMNet wurde entwickelt, um verschiedene Audiosignale zu klassifizieren. Das Audiosignal wird mit 16kHz abgetastet und anschließend wird die Kurzzeit-Foruier-Transformation mit einer Fensterbreite von 25ms berechnet. Von diesem Ergebnis wird das mel-Spektrogramm aufgebaut. Mit diesem Spektrum ist das Netz in der Lage, 512 verschiedene Klassen zuzuweisen.
Das Netz ist mit Anlehnung an das MobileNet_v1 aufgebaut und verwendet die tiefenseparierbare Faltung. Diese zeichnet sich dadurch aus, dass es für jeden Kanal einen Filterkern gibt. Somit werden weniger trainierbare Parameter benötigt, was dazu führt, dass Überanpassung vermieden wird, sowie das Training beschleunigt.[16]
Die Netzarchitektur des YAMNet ist in Abbildung A.1 dargestellt. Es handelt sich um eine sehr effiziente Struktur und besitzt nur eine geringe Anzahl an Schichten, sowie an trainierbaren Parametern (vgl. Tabelle 2.1). Es wird sehr stark mit wiederkehrenden Strukturen gearbeitet.
Das Training des YAMNet wurde mit dem AudioSet durchgeführt. Dieser Datensatz wird von Google zur Verfügung gestellt und enthält 5.800 Stunden an klassifizierten Audiomaterial. Diesem Audiomaterial wurden 527 verschiedene Klassen zugewiesen. Der Datensatz wurde auf YouTube-Videos zusammengestellt, dadurch ist auch ein vielfältiges Spektrum an realistischen Hintergrundgeräuschen gegeben. [1]
Durch die geringe Komplexität ist das Netz sehr schnell und auch zur Anwendung auf Geräten mit begrenzter Hardware nutzbar. Diese geringe Komplexität kann bei komplexeren Signalen zum Nachteil werden.

MobileNet

Das MobileNet wurde zur Klassifizierung von Bildern entworfen, unter Berücksichtigung, dass das Netz auf limitierter Hardware läuft. So soll es ermöglicht werden, dass Geräte mit geringem Speicher oder geringer Rechenleistung dennoch in der Lage sind, das Netz auszuführen. Das Netz wurde zur Bild-Klassifizierung, Objekterkennung, sowie Bewältigung anderer Aufgaben entwickelt. Die erste Version wurde im April 2017 veröffentlicht, die zweite Version im März 2019. In dieser Arbeit wird ausschließlich die zweite Version behandelt. Diese verfügt über verbesserte Genauigkeit und verringerte Größe gegenüber der ersten Version. Die Entwicklung wird von Google geleitet.
Das Netz verwendet eine geringe Komplexität und ist sehr schnell in der Anwendung.

Dadurch ist der Speicherbedarf auch sehr gering.
Das Netz verwendet die tiefen-separierbare Faltung und ist durch einen Zusammenschluss einiger Funktionsblöcken aufgebaut (Abbildung A.2). Durch die tiefen-separierbare und die anschließende zweidimensionale Faltung werden die Anzahl der trainierbaren Parameter stark reduziert und die Rechenkomplexität gering gehalten.
Als Trainingsdatensatz wird der ImageNet Datensatz verwendet. Dieser enthält 14,2 Millionen Bilder mit 21.841 verschiedenen Klassen. Der Datensatz ist öffentlich zugänglich und ist für die Verbesserung des maschinellen Sehens, sowie für das Trainieren neuronaler Netze gedacht. [6]
Durch den Fokus auf die geringe Speichernutzung, sowie auf die benötigte Rechenleistung, ist das Netz sehr schnell. Allerdings ist dieses Netz zur Bildklassifizierung entwickelt worden und hat somit einen stark abweichenden Trainingsdatensatz, zu den in dieser Arbeit verwendeten Spektrogrammen. [14]

GoogLeNet

Ebenfalls zur Bild-Klassifizierung wurde das GoogLeNet entworfen. Dieses Netz gewann im Jahr 2014 die ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Ziel dieses Netzes ist es, bei hoher Effizient eine ausgezeichnete Genauigkeit zu erreichen mit einer geringen Anzahl an trainierbaren Parametern.
Es ist auf Basis der Inception-Architektur aufgebaut. Das Besondere an diesem Netz ist, dass es mit mehreren parallelen Faltungsschichten arbeitet, die unterschiedlich große Faltungskerne besitzen. Die parallele Anwendung von Faltungskernen unterschiedlicher Größe ermöglicht es dem Netz, Merkmale in verschiedenen Größenordnungen effizient zu extrahieren.
Die parallele Anwendung der Faltungsschichten ist in Abbildung A.3 dargestellt. Das Anwenden eines Funktionsblocks wird von maximalen Poolingsschichten unterbrochen. Zu Beginn der Verarbeitung wird der Bildeingang seriell behandelt. Es werden wenige Normalisierungsschichten verwendet.
Das Training des Netzes erfolgte auf dem ImageNet Datensatz. In dem ILSVRC-Wetbewerb wurden dem Netz 150.000 Bilder, sowie 1.000 Klassen zur Verfügung gestellt [6]. Diese Bilder enthielten unterschiedliche Objekte wie „Tastatur“, „Maus“, „Stift“ sowie viele Tiere. Eine weitere Version wurde auf dem Places365 Datensatz trainiert und sie ist in der Lage Orte zu klassifizieren ([12]). Die Version des Netzes wird in der Arbeit nicht behandelt.

Durch die parallele Verarbeitung der unterschiedlichen Faltungschichten kann das Netz besonders gute Muster in verschiedenen Größen extrahieren. Dies führt zu einem erhöhten Bedarf an Rechenleistung. [15]

OpenL3

Das OpenL3 (Look, Listen and Learn) wurde für die Verarbeitung von Video- und Audiodaten entwickelt. Es zielt darauf ab, den Zusammenhang zwischen Video- und Audioinformationen zu erkennen.
Für diese Aufgabe wurde das selbstüberwachte Lernen verwendet. Die Besonderheit darin ist, dass der Trainingsprozess in zwei Schritten aufgebaut wird. Zunächst wird eine Pretext-Aufgabe auf einem Datensatz ohne Label durchgeführt. Nachdem relevante Strukturen aus den Daten extrahiert wurden, erfolgt ein Wissenstransfer auf einem gelabelten Datensatz. Dieses Verfahren reduziert den Bedarf an Daten mit Label.
Der strukturelle Aufbau des Netzes (Abbildung A.4) ist stets seriell und weist wenig Besonderheiten auf. Es handelt sich um ein Netz mit wenigen Schichten sowie einer geringen Anzahl an trainierbaren Parametern. Die verwendeten Schichten zeigen keinerlei Besonderheiten auf.
Das OpenL3 wurde auf einer Teilmenge des AudioSet Datensatzes trainiert. Die Teilmenge enthielt 60 Millionen Trainingsdaten. Dabei wurden 80% der Daten zum Trainieren und jeweils 10% der Daten zum Validieren, sowie zum Testen genutzt. [2]

DenseNet

Das DenseNet (Densely Connected Convolutional Network) hat bereits in dem übergeordneten Projekt dieser Arbeit Anwendung gefunden. Das Netz wurde mit einem Fokus auf das Problem eines verschwindenden Gradienten entworfen. Es findet Anwendung in der Bildklassifizierung.
Das Problem des verschwindenden Gradienten tritt besonders bei tiefen Netzen auf, da dieser bei der automatischen Differenzierung im Rückwärtsmodus immer kleiner wird. Um dieses Problem zu umgehen, wird eine direkte Verbindung zwischen den Funktionsblöcken hergestellt. Diese sorgen dafür, das der Gradient während des automatischen Differenzierens in den oberen Schichten einen relevanten numerischen Wert besitzt.
Das Netz zeichnet sich dadurch aus, dass es besonders viele Schichten, im Vergleich zu den anderen Netzen, hat(Tabelle 2.1). All diese Schichten sind durch die zugrundeliegende

Architektur miteinander verbunden, die sogenannten „DenseBlocks“, die aus einem Zusammenschluss von Batch-Normalisierungsschichten, Relu-Aktivierungsfunktionen, sowie einer Faltungsschicht, verfügen (Abbildung A.5). Das DenseNet enthält durch die starke Wiederholung der Funktionsblöcke eine aüßerst hohe Schichtenanzahl.
Das DenseNet wurde ebenfalls auf dem ImageNet Datensatz trainiert und ist wie das GoogLeNet in der Lage 1.000 Klassen zuzuweisen.
Durch die Verwendung einer hohen Anzahl an Schichten ist das Training des Netzes speicherintensiv. Dies führt ebenfalls dazu, dass das Training zeitintensiv ist. Trotz der hohen Anzahl an verwendeten Schichten weist das DenseNet eine geringe Anzahl an trainierbaren Parametern auf, wie in Tabelle 2.1 zu erkennen ist.[5]

VGGish

Der Name VGG ist ein Akronym und bedeutet Visual Geometry Group. Dies ist eine Gruppe aus Wissenschaftlern, die sich mit verschiedenen Feldern des maschinellen Lernens beschäftigen. [4] Das verwendete VGGish baut auf den Erkentnissen des VGG-Netz auf. VGGish wurde auf mel-Spektrogrammen trainiert und gibt einen Vektor der Größe 128x1 aus. Dieser kann als Eingabe für ein Klassifikationsmodell verwendet werden und erleichtert dadurch das Training.
Eine besondere Eigenschaft von VGGish ist seine Fähigkeit, allgemeingültige und kompakte Merkmalsrepräsentationen aus Audiodateien zu erzeugen, die in vielfältigen Anwendungen eingesetzt werden können. Dabei ist das Netz mit einer geringen Tiefe, sowie einer sehr hohen Anzahl an trainierbaren Parametern ausgestattet (Tabelle 2.1). Dieses Netz hat wenige Schichten im Vergleich zu der Anzahl der trainierbaren Parameter.
Das Netz ist nach dem Prinzip aufgebaut, dass erst das eingehende Bild mit Faltungsschichten reduziert wird und schließlich alle Features mit vollverbundenen Schichten zusammengeführt werden. Diese Schichten sorgen für die große Anzahl an trainierbaren Parametern (Abbildung A.6).
Das Training des Netzes hat auf einem großen YouTube Datensatz, der Vorversion des YouTube8M Datensatzes, stattgefunden. Dieser Datensatz enthält derzeit 237.000 von Menschen klassifizierte Videoausschnitte, denen 1.000 verschiedene Label zugeordnet wurden. [19]
Durch die hohe Anzahl der trainierbaren Parameter ist es deutlich speicherintensiver als andere Modelle. Da keine Normalisierungsschichten verwendet wurden, kann die Gradientenbestimmung instabil werden. [10]

2.4.1 Vergleich der Netze

Das Verhältnis zwischen der Tiefe eines Netzes und der Anzahl der trainierbaren Parameter liefert wichtige Hinweise auf die Eigenschaften des Netzes. Tiefe Netze erfordern besondere Aufmerksamkeit hinsichtlich der Trainingsstabilität, während Modelle mit einer sehr hohen Anzahl an Parametern schneller zu Überanpassung neigen. Das DenseNet ist das größte der sechs Netze, wobei das VGGish die meisten trainierbaren Parameter besitzt. Dabei ist dieses mit der geringsten Anzahl an Schichten aufgebaut. Das OpenL3 hat ein umgekehrtes Verhältnis von Schichten und trainierbaren Parameter im Vergleich zu dem MobileNet. Das YAMNet und das GoogLeNet haben ein ähnliches Verhältnis zwischen der Anzahl der Schichten und den trainierbaren Parameter, jedoch ist das GoogLeNet fast doppelt so groß wie das YAMNet. Das MobileNet, GoogLeNet und das DenseNet sind auf Bildern trainiert worden, wohingegen die anderen Netze mit Audio-Spektren trainiert worden sind.

Tabelle 2.1: Vergleich der Validierungsfehler

Modell	Schichten	Trainierbare Parameter
YAMNet	84	3.2 Millionen
MobileNet	152	2.2 Millionen
GoogleNet	142	5.9 Millionen
OpenL3	31	4.6 Millionen
DenseNet	707	19.8 Millionen
VGGish	22	71.6 Millionen

3 Methodik

In diesem Kapitel wird die Vorgehensweise der Anwendung verschiedener neuronaler Netze zur Analyse der FROG-Traces beschrieben. Ziel ist die Vorhersage des ZBP, des Amplitudenverlaufs und des Phasenverlaufs.

Zunächst wird das Vorverarbeiten der Spektren behandelt, das notwendig ist, damit die Spektren die vorgegebenen Eingabedimensionen der vortrainierten Netze erfüllen.

Darauf aufbauend erfolgt die Bewertung verschiedener Netze anhand ihrer Vorhersage des ZBP. Als Bewertungskriterien werden hier der Validierungsfehler, die Rauschresistenz sowie die Trainingszeit betrachtet.

Das Netz, welches die besten Ergebnisse in der Vorhersage des ZBP erreicht, wird weiterverwendet, um den Amplitudenverlauf im Zeitbereich, sowie den Phasenverlauf im Wellenlängenbereich vorherzusagen. Dabei finden verschiedene Kostenfunktionen Anwendung, deren Leistung anhand des erreichten Testfehlers bewertet wird.

Im Anschluss wird zur Verbesserung der Phasenvorhersage ein Netz mit mehreren Eingängen aufgebaut, wobei der Amplitudenverlauf im Wellenlängenbereich in den Trainingsprozess integriert wird. Ebenfalls werden verschiedene Kostenfunktionen angewendet, um die bestmögliche Vorhersage zu erzielen. Die Qualität der Vorhersage wird am erreichten Testfehler bewertet.

3.1 Zuschneiden der Spektren

Ein vortrainiertes Netz hat eine vorgegebene Eingangsgröße. Diese wird durch die Architektur des Netzes vorgegeben. Das Trainieren des Netzes wird stets mit Matrizen derselben Größe durchgeführt. Wenn ein vortrainiertes Netz verwendet werden soll, dann müssen die verwendeten Daten an die Eingabegröße des Netzes angepasst werden. Die

vorhandenen Spektren müssen auf die entsprechende Größe angepasst werden. In den simulierten Spektren ist der Anteil redundanter Informationen im Vergleich zu relevanten Informationen sehr hoch. Hier eignet es sich, die Spektren auf die benötigte Größe zuzuschneiden. Da es sich hier um einen ultrakurzen Laserpuls handelt, ist der Großteil des Spektrums null (vgl. Abbildung 3.1) und enthält somit keine relevanten Informationen.

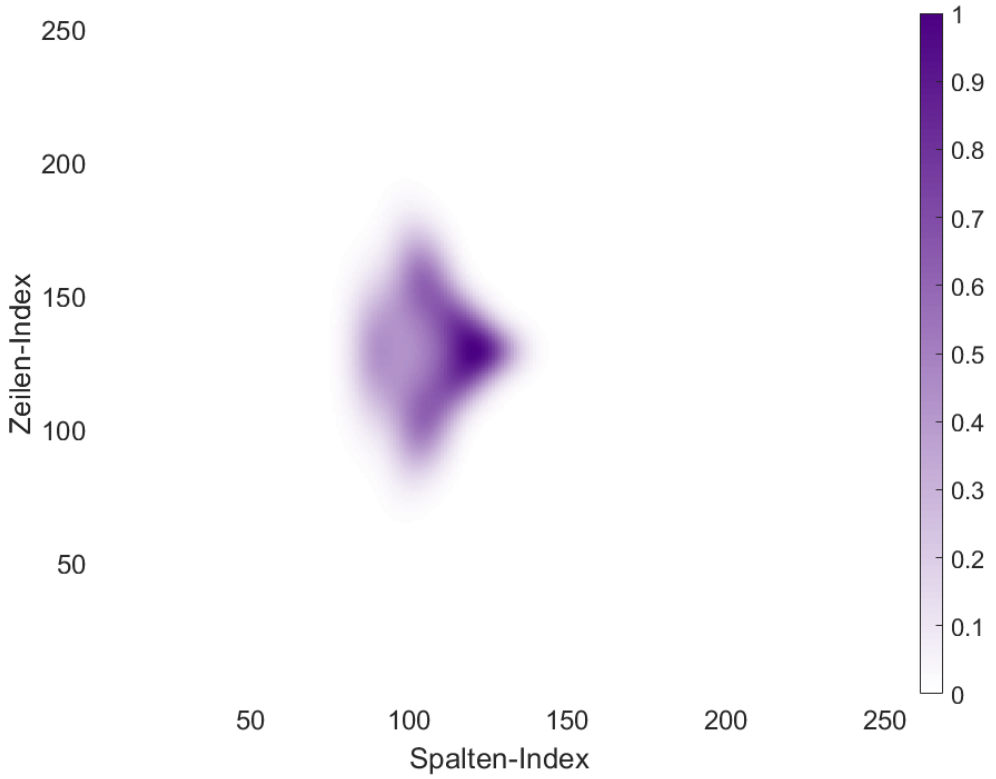


Abbildung 3.1: Ein volles Spektrum

Das Zuschneiden erleichtert das Training, da ein größerer Datenanteil relevante Informationen enthält. Nach dem Abschneiden ist der Anteil relevanter Informationen deutlich höher und das Netz kann so bessere Ergebnisse erzielen. Das Zuschneiden der Spektren wird anhand der Eingangsgröße des YAMNet erläutert. Das YAMNet hat eine vorgegebene Eingangsgröße von 96×64. Somit muss das Spektrum von 256×256 auf die Größe 96×64 zugeschnitten werden. Dies entspricht einem Ausschnitt von 9,38% (vgl. Gleichung 3.1) des gesamten Spektrums.

$$\frac{\text{Zielgöße}}{\text{Eingangsgröße}} = \frac{96 \cdot 64}{256 \cdot 256} = 0.0938 \Rightarrow 9.38\% \tag{3.1}$$

Da dies einem kleinen Bereich des Spektrums entspricht, muss dieser Ausschnitt gezielt gewählt werden.
Das Spektrum weist aufgrund des Messverfahrens eine Symmetrie entlang der horizontalen Mittelachse (Y-Achse, Index 128) auf. Somit sind die Werte unterhalb redundant zu den Informationen oberhalb der Mittelachse. Der Ausschnitt wurde so positioniert, dass die Mittelachse des Spektrums am unteren Rand des Ausschnitts liegt. Dadurch wird ausschließlich der obere Teil des Spektrums betrachtet. Der Massenschwerpunkt wurde zur Bestimmung der Platzierung des Ausschnitts entlang der X-Achse ermittelt. Dieser wird berechnet, indem alle Spalten des Spektrums aufsummiert werden. Dadurch entsteht ein Vektor mit 256 Werten. Diese werden mit ihren jeweiligen Indizes multipliziert und anschließend aufsummiert. Das Ergebnis wird durch die Gesamtmasse dividiert. In der Gleichung 3.2 repräsentiert **m** die Summe einer Spalte, **M** beschreibt die Summe aller Spalten, **i** definiert den Spaltenindex. Mit dem Anwenden dieser Gleichung wird der Mittelpunkt bestimmt, sodass sich ein Großteil der relevanten Informationen in dem gewählten Ausschnitt befinden. Dieser wird an das Netz übergeben und entspricht den vorgegebenen Eingabedimensionen.

$$i_{Schwerpunkt} = \frac{1}{M} \sum_{i=1}^{256} i \cdot m_i \tag{3.2}$$

$$M = \sum_{i=1}^{256} m_i \tag{3.3}$$

3.2 Vorhersage des Zeit-Bandbreite-Produktes

Um das Netz zu ermitteln, welches am besten den aus der Zielsetzung gestellten Anforderungen entspricht, werden verschiedene vortrainierte Netze in Betracht gezogen. Die Netze sollen das ZBP aus den vorgegebenen Spektren möglichst genau vorhersagen können. Das ZBP ist eine Größe, die die Komplexität des zeitlichen Verlaufs des Pulses repräsentiert (vgl. Unterabschnitt 2.2.1). Es ist bei idealen Pulsen gering, da sie eine minimale Ausdehnung sowohl im Zeit- als auch im Frequenzbereich aufweisen. Bei komplexeren Pulsen erhöht sich der Wert entsprechend.

Die nachfolgende Aufgabe dient als Maß für die Effektivität der verschiedenen Netze, relevante Informationen aus den Spektren zu extrahieren. Eine korrekte Vorhersage

des ZBP's indiziert, wie gut das Netz in der Lage ist, verschiedene Verläufe, wie den Amplituden- oder den Phasenverlauf des Pulses, vorherzusagen.
Zur Identifikation des optimalen Netzes wird jenes mit dem geringsten Validierungsfehler ausgewählt. Ein niedriger Validierungsfehler zeigt, dass das Netz das ZBP zuverlässig vorhersagen und relevante Informationen aus dem Spektrum extrahieren kann.
Ein entscheidender Faktor für die Netzqualität ist die Robustheit gegenüber Rauschen. Da das Netz in experimentellen Anwendungen eingesetzt werden soll, muss es in der Lage sein, relevante Informationen zuverlässig von Rauschen zu unterscheiden.
Um sich möglichst gut an ändernde Bedingungen anpassen zu können, soll das Netz schnell lernen und auch in der Lage sein, aus kleineren Trainingsdaten relevante Lernfortschritte zu extrahieren. In verschiedenen experimentellen Umgebungen können sich die auftretenden Spektren geringfügig von den Trainingsdaten unterscheiden. Daher ist es entscheidend, dass sich das Netz schnell auf veränderte Spektren anpassen lässt.
Die Anwendungszeit eines neuronalen Netzes beschreibt die Dauer, die das Netz benötigt, um eine Vorhersage auf Basis gegebener Eingabedaten zu berechnen. Eine möglichst geringe Anwendungszeit ist wünschenswert, um Verzögerungen im praktischen Einsatz zu vermeiden. Sie stellt jedoch kein primäres Auswahlkriterium für ein Modell dar, sondern wird lediglich als zusätzlicher Vorteil betrachtet.
Ein weiterer Vorteil ist, wenn das Netz zum trainieren nur einer geringe Hardware-Nutzung benötigt.
Zur Bewertung anhand der definierten Kriterien ist Voraussetzung, dass das Netz bei ausreichender Datensatzgröße weder zu Überanpassung noch zu Unteranpassung neigt. Es muss die Stabilität des Netzes während des Trainings gegeben sein.

Die Bewertungskriterien sind somit in der Reihenfolge der Wichtigkeit:

1. Niedriger Validierungsfehler
2. Robustheit gegenüber Rauschen
3. Schneller Lernprozess
4. Schnelle Anwendungszeit (optional)
5. Geringe Hardwarenutzung während des Lernprozesses (optional)
6. Stabiles Netz ohne Überanpassung oder Unteranpassung (Voraussetzung)

Um die Kapazität eines Netzes zu bewerten, ist die Anzahl der trainierbaren Parameter ein entscheidender Faktor. Dieser gibt an, wie viele Variablen dieses Netz besitzt und anpassen kann. Durch eine höhere Anzahl an trainierbaren Parametern steigt in der Regel auch die Trainingszeit. Die Trainingszeit hängt auch stark von der verwendeten Hardware ab. In diesem Fall werden jedoch alle Netze auf derselben Hardware trainiert, sodass ein direkter Vergleich möglich ist.

3.2.1 Aufbau des Testes

Der gesamte Programmablaufplan ist in Abbildung A.7 dargestellt. Die Trainingsoptionen sind für alle Netze gleich, um eine einheitliche Grundlage zur Bewertung zu haben. Als Optimierungsalgorithmus wird Adam verwendet. Es werden 15% der Daten zum Validieren benutzt. Die Lernrate beginnt bei 0.0005 und wird mit der Funktion eines Kosinus-Abfalls reduziert. Das Netz wird alle 30 Iterationen mit den Validierungsdaten überprüft. Zum Beschleunigen des Trainings wird eine Minibatchgröße von 64 Datenpunkten verwendet. Wenn sich der Validierungsfehler nach acht Validierungstests nicht verbessert hat, wird das Training als beendet erklärt. Sollte dieses Kriterium nicht eintreten, wird das Training nach 150 Epochen gestoppt. Das Netz mit dem niedrigsten Validierungsfehler wird anschließend abgespeichert. Die Wahl der Trainingsparameter basiert auf heuristischen Überlegungen und gängigen Methoden zur Optimierung neuronaler Netze. Zum Ausführen des Skriptes wurde die von der Hochschule zur Verfügung gestellte Hardware (Tabelle 3.1) genutzt.

Tabelle 3.1: Verwendete Hardwarekonfiguration

Komponente	**Details**
CPU	AMD Ryzen 9 3950X, 16 Kerne
GPU	NVIDIA RTX 4090, 24 GB VRAM
Arbeitsspeicher	128 GB DDR4-RAM

Der Programmablauf startet mit dem Erstellen der Ergebnistextdatei. Für jedes abgeschlossenes Training wird in dieser Textdatei eine Zeile angefügt. Innerhalb dieser Zeile werden die Parameter zur Bewertung des Netzes gespeichert. Anschließend werden drei for-Schleifen initialisiert. Mit diesen Schleifen wird über das Rausch-Niveau, über die

Anzahl der verwendeten Datenpunkte, sowie die verschiedenen verwendeten Netze iteriert. Es wird über vier verschiedene Rausch-Niveaus jeweils 0%, 0,1%, 1% sowie 3% Rauschen iteriert. Die Anzahl der verwendeten Datenpunkte verdoppelt sich mit jedem Durchlauf. Im Training werden die sechs Netze aus Abschnitt 2.4 implementiert. Zunächst werden die verwendeten Trainingsoptionen definiert. Im Anschluss erfolgt das Laden der Spektren. Falls dieselbe Anzahl an Spektren bereits im vorherigen Durchlauf verarbeitet wurden, werden diese übernommen, wodurch ein erneutes, zeitaufwendiges Einlesen entfällt. Dasselbe Verfahren wird für das Einlesen der Label verwendet. Die Spektren werden, wie in Abschnitt 3.1 beschrieben, auf die vom Netz benötigte Größe zugeschnitten. Anschließend werden die Daten und die Label in den Trainingsdatensatz, sowie in den Validierungsdatensatz aufgeteilt. Vor dem Start des Trainings muss jedes Netz individuell angepasst werden, damit das Netz eine Regressionsaufgabe bewältigen kann. Die letzten Schichten müssen bearbeitet werden, sodass das Netz über lediglich eine Ausgabegröße verfügt. Sollte das Training nicht erfolgreich sein, wird das Rausch-Niveau, die Anzahl an Datenpunkte, sowie der Modellname mit einer Fehlermeldung abgespeichert. Sollte das Training erfolgreich beendet worden sein, werden die erzielten Ergebnisse in die am Anfang erstellte Ergebnistextdatei abgespeichert. In einem Ordner wird das trainierte Netz, sowie der Trainingsverlauf abgelegt. Zu diesen beiden Dateien wird ebenfalls ein scatter-Plot abgelegt, der einige Vorhersagen aus dem Validierungsdatensatz mit den entsprechenden Label vergleicht. Mit diesem Plot lässt sich erkennen, ob das Netz irreguläre Strukturen aufweist.

3.2.2 Auswahl der Netze

Es werden sechs verschiedene neuronale Netzwerke ausgewählt, deren Leistungsfähigkeit miteinander verglichen wird. Die zur Auswahl stehenden Netze sind: YAMNet, MobileNet, GoogLeNet, OpenL3, DenseNet und VGGish. Eine ausführliche Erleuterung der Netze wurde in Abschnitt 2.4 durchgeführt.

Die Auswahl der Netze erfolgte unter anderem deshalb, weil drei Modelle aus dem Bereich der Bildverarbeitung und drei aus der Spektralanalyse stammen. Zudem sollte ein breites Spektrum an Netzwerkgrößen abgedeckt werden, insbesondere unter Berücksichtigung der Zielsetzung, die Modelle auf leistungsschwächerer Hardware anwendbar zu machen. Darüber hinaus war das DenseNet-Modell bereits in das Projekt integriert.

Diese Netze basieren auf unterschiedlichen Technologien und wurden für verschiedene Anwendungsbereiche entwickelt. Was jedoch alle vereint ist, dass sie aus Schichten bestehen und trainierbare Parameter haben. Die Auswahl der Netze erfolgte unter anderem aufgrund ihres unterschiedlichen Verhältnisses zwischen Anzahl der Schichten und der Anzahl der trainierbaren Parameter. Auch der ursprüngliche Anwendungszweck der jeweiligen Netze ist relevant, da er Rückschlüsse auf die verwendeten Trainingsdaten zulässt.

3.2.3 Validierungsfehler

Es ist für ein Netz entscheidend, auch aus kleineren Datensätzen möglichst schnell relevante Muster zu extrahieren. Der Validierungsfehler ist ein optimaler Messwert dafür, wie das Netz mit unbekannten Daten umgeht. In Abbildung 3.2 ist zu erkennen, dass mit steigender Anzahl an Datenpunkten der generelle Validierungsfehler sinkt. In diesem Graphen wird der Verlauf der Validierungsfehler für Spektren ohne Rauschen dargestellt. Es ist zu erkennen, dass es eine starke Variation des Validierungsfehlers zwischen den verschiedenen Netzen bei einer geringen Anzahl an Datenpunkten gibt, besonders zwischen dem MobileNet und dem VGGish. Mit steigender Menge der Datenpunkte nähern sich alle Kurven einem ähnlichen Validierungsfehler an. Das GoogLeNet erzielt mit 1.024 Datenpunkten einen höheren Validierungsfehler, fällt bei größeren Datensätzen allerdings wieder stark ab und erreicht somit den geringsten Validierungsfehler bei dem größten Datensatz. Das VGGish erreicht allerdings bei kleineren Datensätzen bessere Validierungsfehler. Dieser Validierungsfehler ist über alle Datensatzgrößen annähernd konstant. Es zeigt sich, dass das GoogLeNet Schwierigkeiten hat, sich bei kleineren Datensätzen anzupassen und sehr variirende Ergebnisse erziehlt. Aus dem Plot Abbildung 3.2 geht hervor, dass das VGGish einen sehr konstanten und geringen Fehler erreicht.

3.2.4 Variierendes Rauschen

Das Netz muss möglichst gut gegen Rauschen beständig sein, da dies in realen Messungen auftritt. Zur Simulation der realen Gegebenheiten wird den Spektren unterschiedlich viel Rauschen hinzugefügt. Die verschiedenen Rausch-Niveaus enthalten 0%, 0,1%, 1% sowie 3% Rauschen. Es handelt sich hierbei um additives weißes Rauschen welches durch die Daten als gegeben angenommen wird. Dabei kann der Fehler auftreten, dass durch die im Rauschen enthaltenen negativen Anteile negative Werte im Spektrum entstehen. Dieser

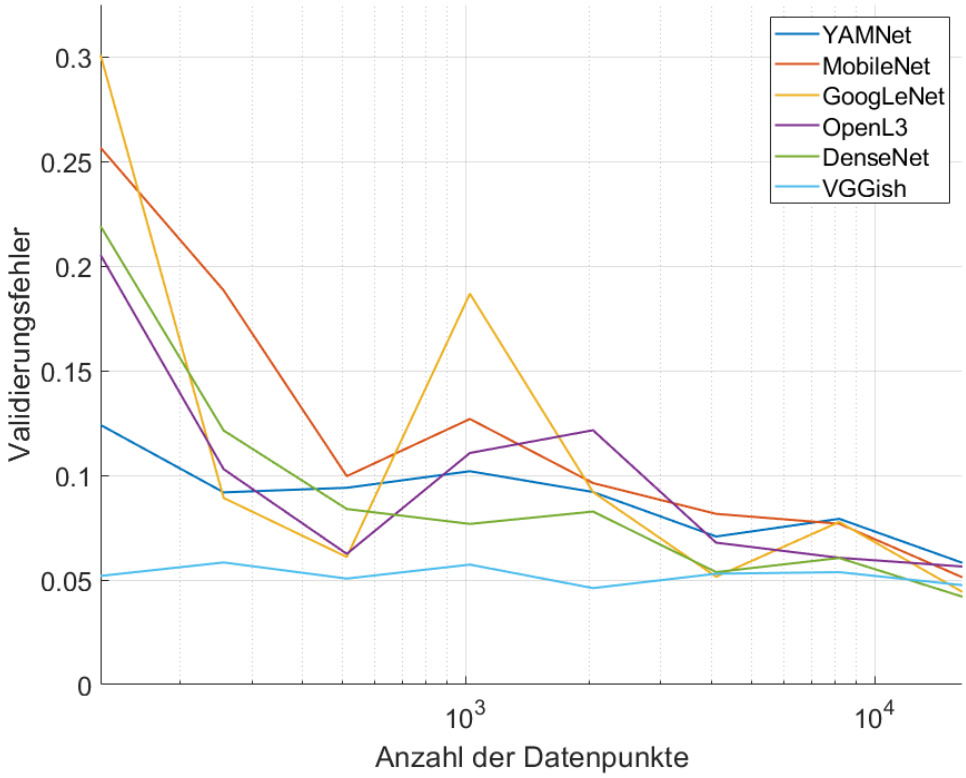


Abbildung 3.2: Erreichte minimale Validierungsfehler in Abhängigkeit von der Anzahl der Datenpunkte ohne Rauschen

Fehler nicht der Realität, wird jedoch wegen der geringen Auftrittswahrscheinlichkeit und dem geringen Einfluss auf das Training vernachlässigt.

In Abbildung 3.3 ist die Abweichung des Validierungsfehlers bei hinzugefügtem Rauschen zur Darstellung gebracht. Als Basislinie wird der erreichte Validierungsfehler ohne Rauschen abgebildet. Die Balken zeigen auf, wie stark der Validierungsfehler beim Hinzufügen von Rauschen abweicht. Zur besseren Lesbarkeit wurde die X-Achse einiger Modelle angepasst. Diese Verschiebung ist nicht in den Daten repräsentiert und dient nur der Lesbarkeit. Die Abweichung des Validierungsfehlers variiert bei wenigen Datenpunkten deutlich stärker und wird mit steigender Anzahl der Datenpunkte immer weniger. Diese ist überwiegend nach oben gerichtet, da durch das Hinzufügen von Rauschen die Vorhersage erschwert wird und somit ein höherer Validierungsfehler entsteht. Ausnahmen sind jedoch zu berücksichtigen, wie beim GoogLeNet mit 1.024 Datenpunkten ersichtlich ist. Der Validierungsfehler ist im rauschfreien Fall am höchsten, bei zunehmendem Rauschen

sinkt er hingegen. Das Rauschen hat auf das VGGish keinen starken Einfluss. Der Validierungsfehler weicht sowohl nach oben als auch nach unten ab. Das MobileNet erzielt mit Hinzufügen von Rauschen nur höhere Validierungsfehler. Abweichungen, die durch Rauschen hinzugefügt werden, sinken mit höherer Datenpunktenanzahl und beeinflussen den erreichten Validierungsfehler nur bedingt.

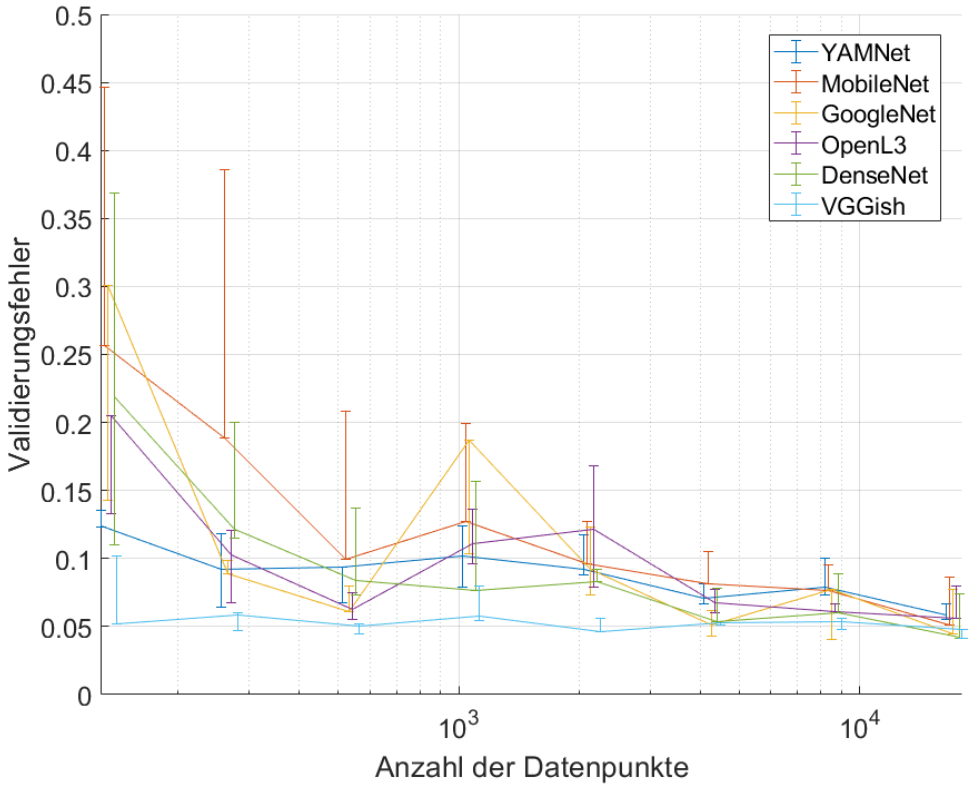


Abbildung 3.3: Abweichen des Validierungsfehlers beim Hinzufügen von Rauschen

Des weiteren ist in Tabelle 3.2 der Mittelwert des Validierungsfehlers über die Rausch-Niveaus eingetragen. Auch hier erzielt das VGGish die besten Ergebnisse über verschiedene Rausch-Niveaus. Bei dem GoogLeNet ist zu erkennen, dass hinzufügen von Rauschen nicht immer zu einer schlechteren Vorhersage führt. Dieses erreicht mit 3% Rauschen den geringsten Validierungsfehler.

Tabelle 3.2: Mittelwert der Validierungsfehler unter Einfluss von Rauschen

Modell	0% Rauschen	0,1% Rauschen	1% Rauschen	3% Rauschen
VGGish	0,0523	0,0587	0,0527	0,0569
YAMNet	0,0890	0,0837	0,0838	0,1027
OpenL3	0,0985	0,0831	0,0929	0,1006
GoogleNet	0,1129	0,0931	0,1052	0,0804
DenseNet	0,0925	0,1260	0,1039	0,1007
MobileNet	0,1222	0,1812	0,1926	0,1962

3.2.5 Trainingszeit

Die Trainingszeit stellt ein wichtiges Auswahlkriterium dar, um das am besten geeignete Netz zur Erreichung der Zielsetzung zu identifizieren. Sie gibt Aufschluss darüber, wie stark das Netz angepasst werden muss, um neue Daten effizient zu verarbeiten. Zur besseren Lesbarkeit von Abbildung 3.4 ist die y-Achse des Graphen logarithmisch dargestellt. Es wurde der Mittelwert der Trainingszeit über alle Rausch-Niveaus gebildet, damit alle Trainings Einfluss auf die gemessene Trainingszeit haben. In Abbildung 3.4 ist zu erkennen, dass sich drei Gruppen bilden. Das Training des DenseNet erweist sich als zeitintensivsten. In Bezug auf die Trainingsdauer bilden MobileNet, GoogLeNet und OpenL3 aufgrund ihrer nahezu identischen Zeiten das Mittelfeld, wohingegen das YAMNet und das VGGish die schnelleren Netze sind. Dabei ist das VGGish minimal schneller und ist in dieser Hinsicht am besten für das Erreichen der Zielstellung geeignet.

In Abbildung 3.5 wird der erreichte Validierungsfehler in Abhängigkeit von der benötigten Zeit dargestellt. Dabei werden sämtliche Rausch-Niveaus berücksichtigt. In dem Graphen ist eine Klumpenbildung zu erkennen. Eine hohe Streuung deutet darauf hin, dass das Netz den minimalen Validierungsfehler nicht konsistent erreicht. Das zeitintensivste Modell ist das DenseNet. Die überzeugendsten Ergebnisse liefert hingegen das VGGish-Netz: Es erzielt sowohl einen niedrigen Validierungsfehler als auch eine vergleichsweise geringe Trainingszeit. Zudem erreicht es diesen Fehler konsistent über verschiedene Durchläufe hinweg. Eine besonders breite Streuung der Validierungsfehler ist beim OpenL3-Modell zu beobachten, was auf eine erhöhte Varianz und eine geringere Stabilität im Lernverhalten hinweist.

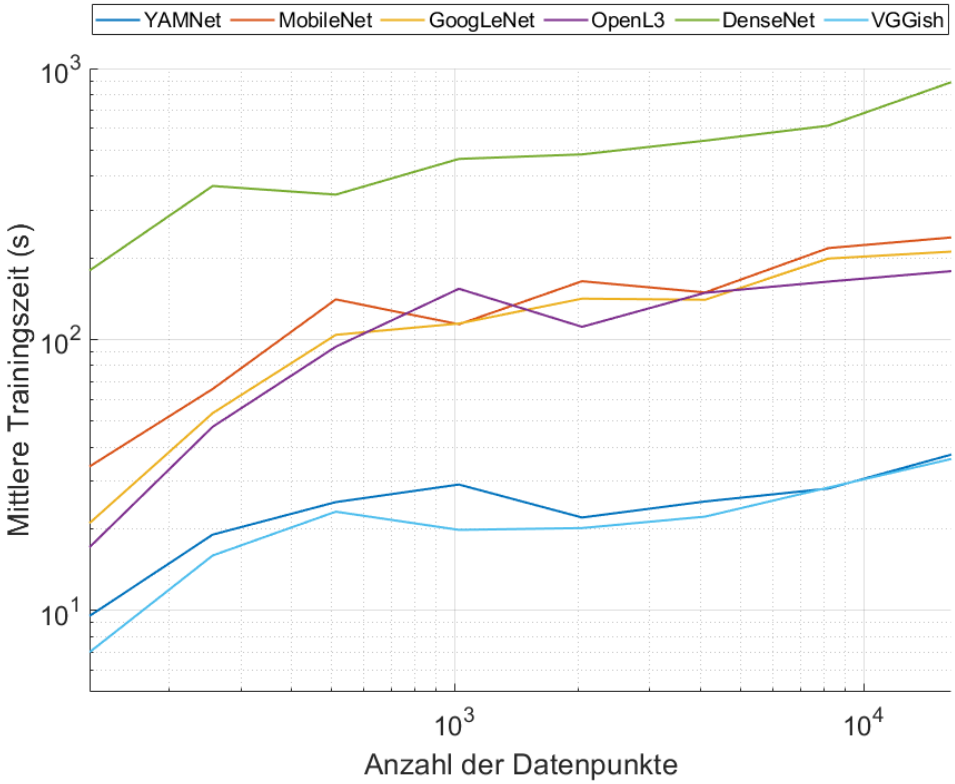


Abbildung 3.4: Trainingszeit der verschiedenen Netze bei einer unterschiedlichen Anzahl an Datenpunkten

3.2.6 Anwendungszeit

Die Anwendungszeit ist ein relevanter Faktor für die spätere Nutzung des Netzes, da eine hohe Anwendungszeit zu einer langsameren Verarbeitung führt. Dies hat zur Folge, dass sich die Analyse der Spektren verzögert.

Ausschlaggebend für die Nutzung des Modells ist die benötigte Anwendungszeit auf ein gegebenes Spektrum. Diese Zeit ist unabhängig von dem Rausch-Niveau und der Anzahl an Datenpunkten, die für das Training verwendet wurden. In diesem Fall führt das YAMNet, wohingegen das DenseNet die höchste Anwendungszeit benötigt. Somit eignet sich das YAMNet am besten zur Erfüllung der Zielsetzung.

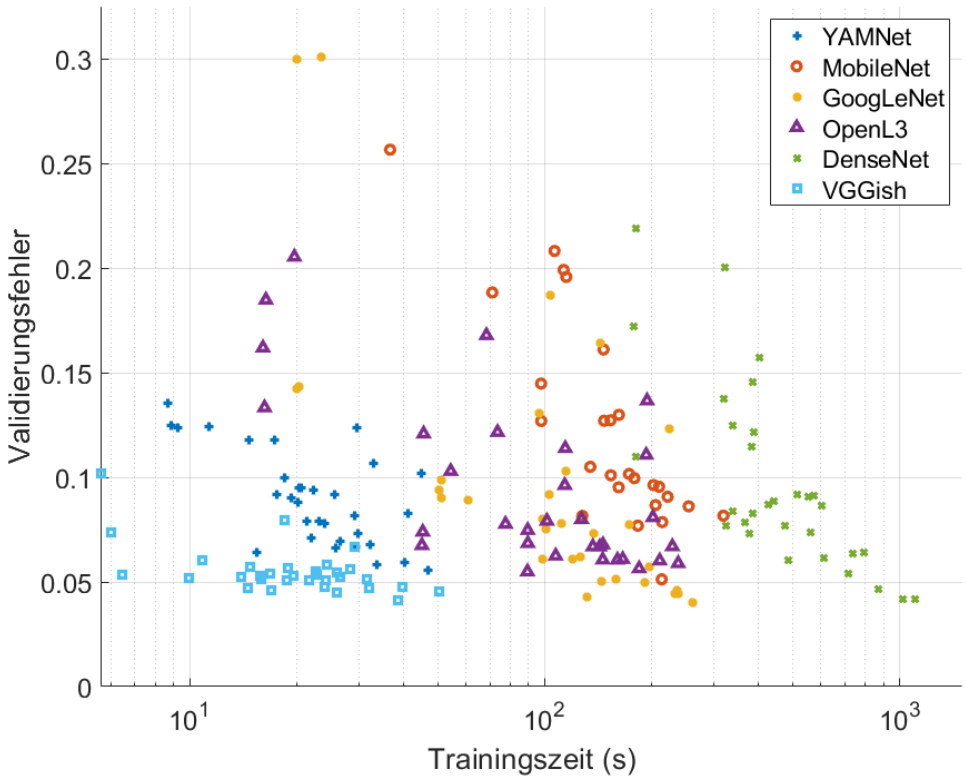


Abbildung 3.5: Erreichte Validierungsfehlers in Abhängigkeit der benötigten Trainingszeit

Tabelle 3.3: Vergleich der Anwendungszeiten der verschiedenen Netze

Model	Anwendungszeit
YAMNet	2,3 ms
VGGish	3,4 ms
MobileNet	8,6 ms
GoogLeNet	10,3 ms
OpenL3	17,5 ms
DenseNet	36,2 ms

3.2.7 Hardware-Nutzung

Die benötigte Hardware ist besonders für den Trainingsprozess entscheidend, da sowohl die Dauer des Trainings als auch dessen Umsetzbarkeit maßgeblich von der verfügbaren

Hardware abhängen. In diesem Zusammenhang ist eine geringe Hardware-Nutzung für die vorliegende Arbeit von zentraler Bedeutung, da sie sicherstellt, dass die Trainingsdurchläufe vollständig und stabil durchlaufen. Besonders kritisch für den Trainingserfolg ist die RAM-Auslastung, denn sollte der RAM aus Tabelle 3.1 nicht ausreichen, wird der Trainingsprozess von dem verwendeten Linux Betiebssystem vorzeitig abgebrochen. Im Verlauf des Trainings wird die Nutzung verschiedener Hardware-Parameter erfasst, wobei die RAM-Auslastung besonders entscheidend für den erfolgreichen Abschluss des Trainings ist.

Die unterschiedlichen Rausch-Niveaus wirken sich nicht auf die Hardwarenutzung aus, da sie weder die Größe der Daten beeinflussen noch zu veränderten Rechenoperationen führen. Dies lässt sich ebenfalls auf Grundlage der im Rahmen dieser Arbeit erhobenen, jedoch nicht dargestellten Daten feststellen. In jedem Training wird die Hardwarenutzung über den Verlauf des Trainings in einem Vektor erfasst, aus dem anschließend der Mittelwert berechnet wird. Die in Tabelle 3.4 gezeigen Werte stellen den Speicherbedaf bei einem Training von 8.192 Datenpunkten dar. Der Speicherbedarf der Trainingsdaten stellt nur einen geringen Anteil des insgesamt verbrauchten Speichers dar. Mit jeder Verdopplung der Trainingsdaten verdoppelt sich auch der Speicherbedarf. Bei 8.192 Datenpunte werden 4,30 GB an Speicherplatz für die verwendeten Daten benötigt.

Ein ähnlicher Speicherbedarf ist bei den Netzen YAMNet, OpenL3 und VGGish zu beobachten, die allesamt aus der Tonverarbeitung stammen. Deutlich höhere Speicheranforderungen weisen hingegen die Netze aus der Bildverarbeitung auf, GoogLeNet, MobileNet und DenseNet, wobei letztere den höchsten Speicherbedarf aufweist. Die VRAM-Nutzung zeigt eine ähnliche Auslastung wie der RAM. Jedoch ist dieser unabhängig von der Anzahl der Datenpunkte und wiederholt sich zyklisch über den gesamten Trainingsverlauf hinweg.

Bei dem OpenL3 gibt es eine Abweichung, da die RAM-Nutzung gering ist, die VRAM-Nutzung allerdings hoch. Es lässt sich erkennen, dass das YAMNet die geringsten Hardwareanforderungen stellt, während das DenseNet die höchsten Anforderungen aufweist, wobei der Unterschied zum MobileNet und GoogLeNet relativ gering ist. Die GPU-Auslastung während des Trainings des YAMNet ist am niedrigsten, während sie beim OpenL3 und DenseNet sehr hoch ausfällt. Die anderen Netze weisen keine signifikanten Unterschiede in der GPU-Nutzung auf.

Der Speicherbedarf der alleinigen Netze wird in Tabelle 3.4 ersichtlich. Es zeigt das VGGish-Netz den größten Speicherbedarf benötigt, was auf die in Tabelle 2.1 aufge-

führten zahlreichen trainierbaren Parameter zurückzuführen ist. Zwischen diesen beiden Tabellen besteht ein annähernd linearer Bezug.

Tabelle 3.4: Speicherbedarf der Netze

Netz	RAM Nutzung	VRAM Nutzung	Speicherbedarf des Netzes
MobileNet	38,32 GB	7,00 GB	9,84 MB
YAMNet	20,65 GB	1,11 GB	13,35 MB
OpenL3	22,81 GB	6,56 GB	18,91 MB
GoogLeNet	38,63 GB	6,30 GB	24,53 MB
DenseNet	38,23 GB	7,18 GB	84,35 MB
VGGish	22,14 GB	3,18 GB	286,58 MB

3.2.8 Über- und Unteranpassung

Aufgrund der geringen Größe des Datensatzes zu Beginn kommt es zu einer erwarteten großen Diskrepanz zwischen dem Trainingsfehler und dem Validierungsfehler (siehe Abbildung 3.6). Diese Differenz verringert sich jedoch mit zunehmender Anzahl der Datenpunkte. Folglich tritt bei keinem der Netze mit einer ausreichend großen Anzahl an Datenpunkten Überanpassung auf. Da alle Netze einen abnehmenden Validierungsfehler aufweisen, kann auch bei keinem der Netze Unteranpassung festgestellt werden.

3.2.9 Analyse der Ergebnisse

Da alle Netze ihre Stabilität unter Beweis gestellt haben und bei ausreichend großen Datensätzen weder Unteranpassung noch Überanpassung auftraten, werden sämtliche Netze zur Bewertung anhand der festgelegten Bewertungskriterien herangezogen. Zu Beginn von Abschnitt 3.2 wurden die Bewertungskriterien in der Reihenfolge ihrer Bedeutung definiert. Der wichtigste Faktor ist der niedrige Validierungsfehler, gefolgt von der Rauschresistenz und der Lernprozessgeschwindigkeit. Die schnelle Anwendungszeit sowie die geringe Hardware-Nutzung sind optionale Kriterien in der Bewertung. Bei dem Validierungsfehler in Unterabschnitt 3.2.3 erreicht das VGGish Netz die besten Ergebnisse. Ebenso ist es das Netz, welches am beständigsten gegen Rauschen ist (siehe Unterabschnitt 3.2.4). Beim Bewerten der Lerngeschwindigkeit erzielten sowohl das YAMNet und das VGGish die besten Ergebnisse (siehe Unterabschnitt 3.2.5). Hinsichtlich der

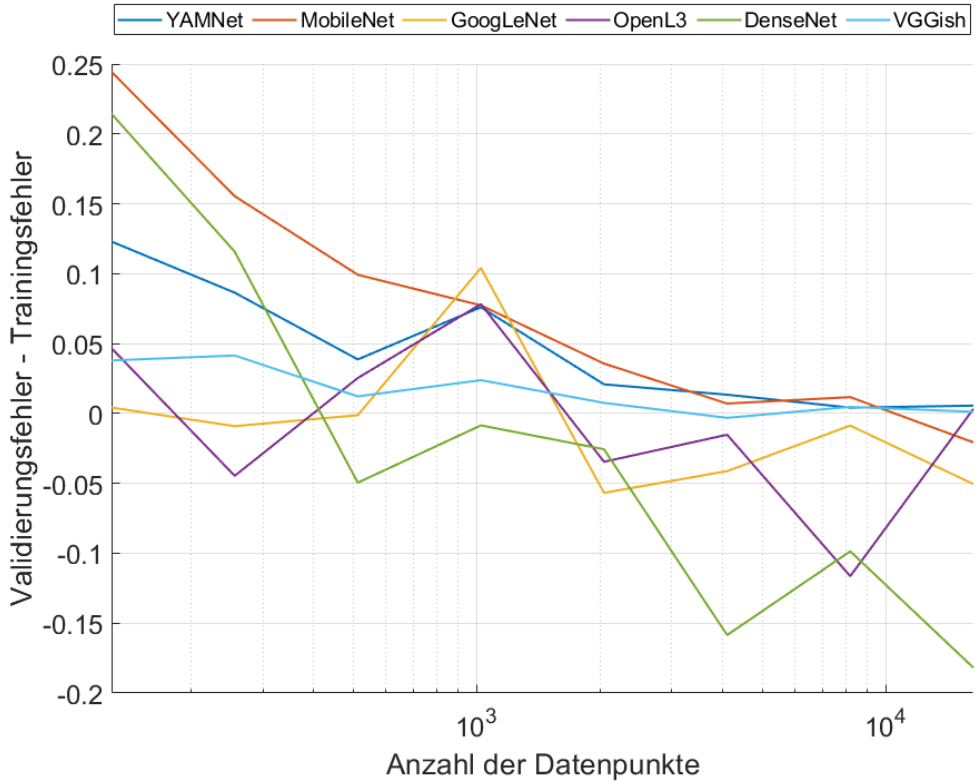


Abbildung 3.6: Differenz zwischen dem Validierungsfehler und dem Trainingsfehler

optionalen Bewertungskriterien, wie der Anwendungszeit (siehe Unterabschnitt 3.2.6), zeigen sowohl das YAMNet als auch das VGGish sehr gute Ergebnisse. In Bezug auf die Hardware-Nutzung befinden sich beide Netze zudem auf einem vergleichbaren Niveau wie das OpenL3 (siehe Unterabschnitt 3.2.7). Das MobileNet erfüllt die definierten Anforderungen am wenigsten, da es einen vergleichsweise hohen Validierungsfehler aufweist, die stärkste Abweichung bei Rauscheinflüssen zeigt und sich hinsichtlich der Trainingszeit lediglich im mittleren Bereich einordnet. Die Auswertung sämtlicher Bewertungskriterien zeigt, dass das VGGish-Netz am besten für die Analyse der Spektren geeignet ist. Insbesondere die niedrigen Validierungsfehler bereits bei kleinen Datensätzen, sowie die hohe Robustheit gegenüber Rauschen und die schnelle Trainingszeit sprechen für seine Eignung. Darüber hinaus ermöglicht die vergleichsweise geringe Hardware-Anforderung in Kombination mit kurzen Anwendungszeiten eine effiziente und schnelle Vorhersage.

3.3 Vorhersage der Amplitude

Ziel ist es, auf Basis des gemessenen Spektrums den zeitlichen Amplitudenverlauf zu rekonstruieren.

Zunächst wird durch Interpolation der Label eine mögliche Optimierung der Vorhersage bewertet. Anschließend erfolgt eine Evaluation unterschiedlicher Kostenfunktionen, darunter MSE, MAE, Huber, Gauss und Smooth. Diese unterschiedlichen Kostenfunktionen setzen jeweils einen eigenen Schwerpunkt auf verschiedene Aspekte der Vorhersage. Nach dem Training des Netzes wird die Leistung anhand des Testfehlers, sowie des Trainingsverlauf und der Trainingszeit bewertet. Basierend auf den Ergebnissen aus Abschnitt 3.2 wurde festgestellt, dass sich VGGish am besten zur Analyse der Spektren eignet. Da es sich hierbei um die Vorhersage eines Verlaufs handelt, wird die Ausgabeschicht des Netzes angepasst, sodass diese die korrekte Dimensionsgröße aufweist.

3.3.1 Interpolieren der Label

Die Label bestehen aus 256 äquidistant verteilten Datenpunkten. Diese gleichmäßige Verteilung bringt den Nachteil mit sich, dass die Auflösung über das gesamte Label hinweg konstant ist. Ein wesentlicher Nachteil der gleichmäßigen Verteilung besteht darin, dass der Impuls lediglich im Zentrum des Label lokalisiert ist, während sich der Großteil der Datenpunkte in Bereichen nahe null befindet. Dadurch steht dem neuronalen Netz nur eine begrenzte Anzahl relevanter Datenpunkte zur Verfügung, um den Impuls präzise zu rekonstruieren. Zusätzlich kann die konstante Auflösung insbesondere bei steilen Signalverläufen zu deutlichen Unterschieden zwischen benachbarten Datenpunkten führen (Abbildung 3.7).

Vor dem Training werden die Label interpoliert. Die Anzahl der Datenpunkte pro Label ist variabel und lässt sich vor dem Training flexibel anpassen, wodurch sie als Hyperparameter betrachtet wird. In dem nachfolgenden Beispiel werden die Anzahl von 256 auf 512 Datenpunkte erhöht. Es ist zu beachten, dass eine höhere Anzahl an Datenpunkten zu einer verlängerten Trainingszeit führen kann. Die Interpolation der Datenpunkte erfolgt nicht äquidistant. Ziel ist es, die Abstände der Datenpunkte am Rand des Pulses größer zu wählen als im Zentrum, da sich in den Randbereichen nur wenige informationsrelevante Anteile befinden. Der Impuls selbst konzentriert sich hauptsächlich in der Mitte des Labels, sodass dort eine höhere Auflösung erforderlich ist. Die Verteilung der

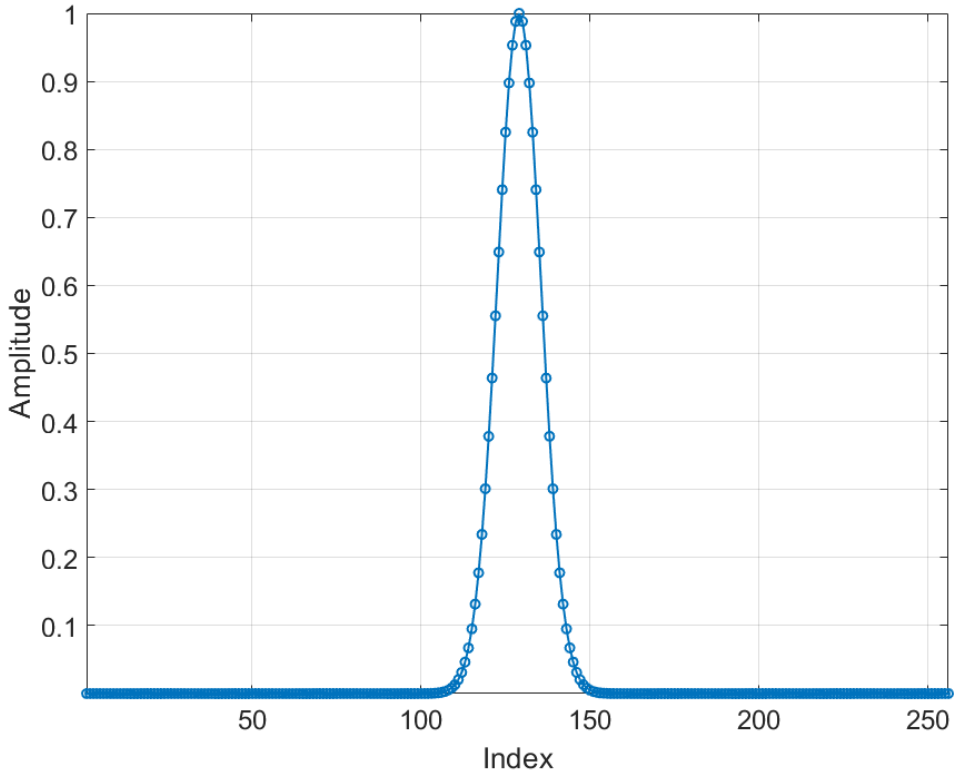


Abbildung 3.7: Ursprünglich Aufteilung der Datenpunkte.

Datenpunkte mithilfe eines Polynoms dritten Grades realisiert. Dies führt zu einer Erhöhung der Anzahl der Hyperparameter, da zusätzlich die Polynomkoeffizienten bestimmt werden müssen. Dabei gilt die Vorgabe, dass das Polynom keinen Wendepunkt enthalten darf, um eine stetig steigende Funktion zu gewährleisten. Andernfalls würden Indizes mehrfach vergeben, was zu Fehlern führt. In der folgenden Anwendug wird das Polynom aus Gleichung 3.5 verwendet. Die verwendeten Polynomkoeffizienten wurden durch empirisches Ausprobieren und zielgerichtete Überlegungen ausgewählt.

$$x \in \mathbb{R}^{512}, \quad x_i \in [-1, 1] \quad \text{für } i = 1, \ldots, 512, \quad x \text{ äquidistant} \tag{3.4}$$

$$y = x^3 + 0.2x \tag{3.5}$$

$$Y = \left(\frac{y - \min(y)}{\max(y) - \min(y)} \right) \cdot (256 - 1) + 1 \tag{3.6}$$

Ausgehend von einem äquidistanten Vektor mit 256 Punkten soll durch Interpolation ein neuer Vektor mit 512 Punkten generiert werden, dessen Verteilung gezielt verzerrt ist, numerisch sich allerdings auf den Bereich eins bis 256 beschrenkt. Zunächst wird ein Vektor x mit 512 Punkten erstellt, welcher äquidistant verteilte Werte in dem Bereich von minus eins bis eins aufweißt. Mit diesem Vektor wird die neue Achse y mit der Gleichung 3.5 bestimmt. Anschließend wird die Achse auf den Bereich von eins bis 256 skalliert um so den gleichen Wertebereich wie die ursprüngliche äquidistante Index-Achse zu besitzen.

Mit der neuen Achse Y wird die Interpolation am Label durchgeführt. Zur Durchführung der Interpolation wird die MATLAB-Funktion „`interp1`“ mit der Interpolationsmethode „`spline`“ verwendet.

In Abbildung 3.8 ist der Vergleich zwischen dem Puls mit und ohne Interpolation dargestellt. Es ist zu erkennen, dass sich mehr Datenpunkte in der Mitte des Pulses Befinden und das der Abstand der Interpolierten Datenpunkte am Rand zunimmt. Das führt dazu, dass das Netz mehr Freiheitsgrade hat, den Verlauf des Impulses zu rekonstruieren. Dadurch, dass der Puls nun deutlich mehr Datenpunkte enthält, hat die Kostenfunktion bei einem schlecht dargestellten Puls einen höheren Wert und somit kann ein besseres Ergebnis erzielt werden.

Dem Netzwerk werden während des Trainings die Vektoren mit den Datenpunkten übergeben, ohne Informationen über deren Abstände zueinander. Dies hat zur Folge, dass das Netzwerk den Puls mit größerer Breite und abgeschwächten Differenzen zwischen den Punkten interpretiert. Es ist ersichtlich, dass zwischen den einzelnen Datenpunkten deutlich geringere Abstände bestehen. Zudem zeigt sich, dass der Randbereich reduziert wurde, wodurch ein größerer Teil des Labels zur Rekonstruktion des Pulses genutzt werden kann.

Um die Wirksamkeit der Methode nicht ausschließlich grafisch zu belegen, sind die erzielten Verbesserungen zusätzlich in Tabelle 3.5 numerisch dargestellt. Zwar erhöht sich die absolute Anzahl an Datenpunkten, deren Wert unter 1% des Maximalwertes liegt, jedoch sinkt deren prozentualer Anteil um 30,08 Prozentpunkte. Dies weist auf eine signifikante Erhöhung von relevanten zu trivialen Datenpunkten hin.

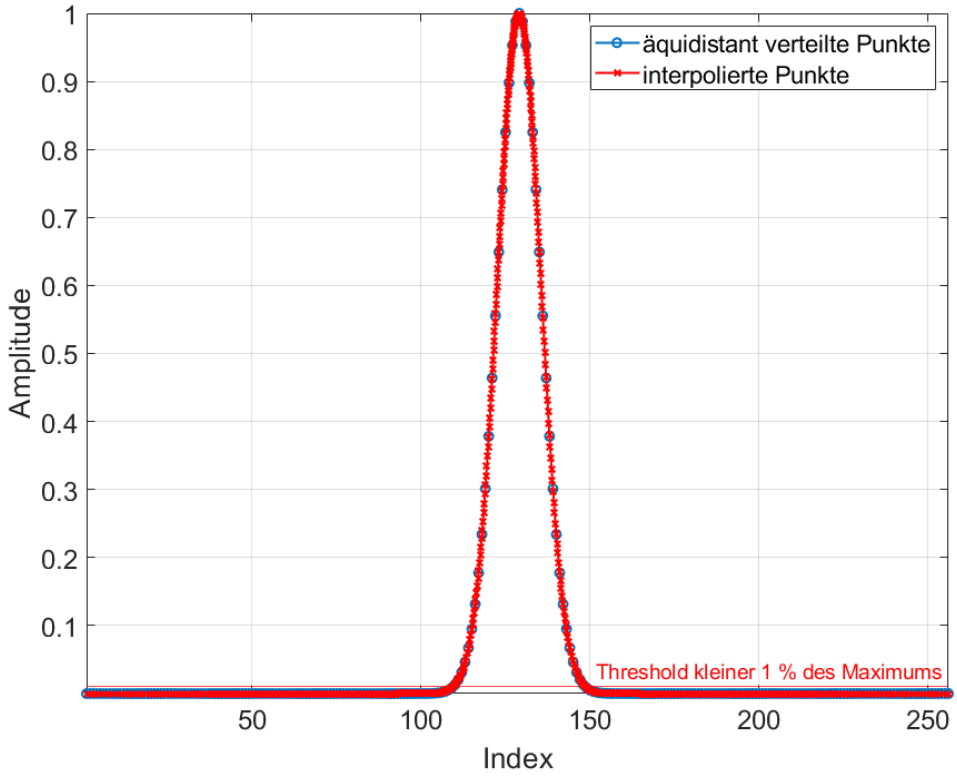


Abbildung 3.8: Vergleich des Originalpulses und des interpolierten Pulses

Tabelle 3.5: Anteil der Datenpunkte unterhalb von 1% des Maximalwertes

Punkte	**Punkte** $< 1\%$	**Prozentualer Anteil**
256	217	84,77%
512	280	54,69%

Trainingsablauf

Es wird der in Abbildung A.8 dargestellte Trainingsablauf in einer etwas angepassten Art verwendet. Anstatt über verschiedene Kostenfuktionen zu iterieren wird über die verschiedenen Interpolations-Auflösungen iteriert. Es wird der komplette zur Verfügung stehende Datensatz mit allen Rausch-Niveaus verwendet. Dieser wurde zu 85% zum Trainieren und zu 15% zum validieren verwendet. Der Validierungsfehler wird alle 15 Itera-

tionen bestimmt. Das Training erstreckt sich über eine Epoche mit einer Minibatchgröße von 256. Die Lernrate beginnt bei 0,005 und fällt exponentiell ab.

Trainingsverlauf

Um die Effektivität der Interpolation zu prüfen, werden vier Trainingsdurchläufe mit verschieden Anzahl an Label-Datenpunkten gestartet. Die Trainingsdurchläufe werden mit einer Auflösung von 256 Label-Datenpunkte ohne Interpolation, 256, 512 sowie 1024 Label-Datenpunkte mit Interpolation durchgeführt. Es wird eine Epoche mit einem Datensatz ohne Rauschen trainiert. Als Kostenfunktion kommt der MSE zum Einsatz, da dieser bereits etabliert ist und eine schnelle Konvergenz innerhalb einer Epoche ermöglicht.

In Abbildung 3.9 ist der Trainingsverlauf für die verschiedenen Interpolationsstufen dargestellt. Der unterschiedliche Fehlerverlauf zwischen den Trainingsdurchläufen mit und ohne Interpolation zeigt, dass die Interpolation grundsätzlich einen Einfluss auf das Training hat, während die Höhe der Interpolation einen deutlich geringeren Einfluss ausübt. Ein möglicher Grund für den erhöhten Fehlerwert bei interpolierten Daten könnte darin liegen, dass durch die Interpolation weniger Datenpunkte mit dem Wert null vorhanden sind, deren Vorhersage vergleichsweise einfach ist. Stattdessen treten vermehrt komplexere Werte auf, deren Vorhersage schwieriger ist und somit zu einem höheren Fehler führen kann.

Die in Tabelle 3.6 dargestellten Trainingszeiten zeigen ein ähnliches Verhalten. Während der generelle Einsatz von Interpolation einen merklichen Einfluss auf die Trainingszeit hat, wirkt sich die Höhe der Interpolation nur geringfügig aus. Insgesamt führt die Interpolation zu einer Erhöhung der Trainingszeit um 47,90%, was als signifikanter Anstieg zu bewerten ist.

Tabelle 3.6: Trainingszeit bei verschieden hoher Interpolation

Anzahl an Label-Datenpunkten	Trainingszeit
256 ohne Interpolation	234,58 s
256 mit Interpolation	346,95 s
512 mit Interpolation	326,97 s
1024 mit Interpolation	345,70 s

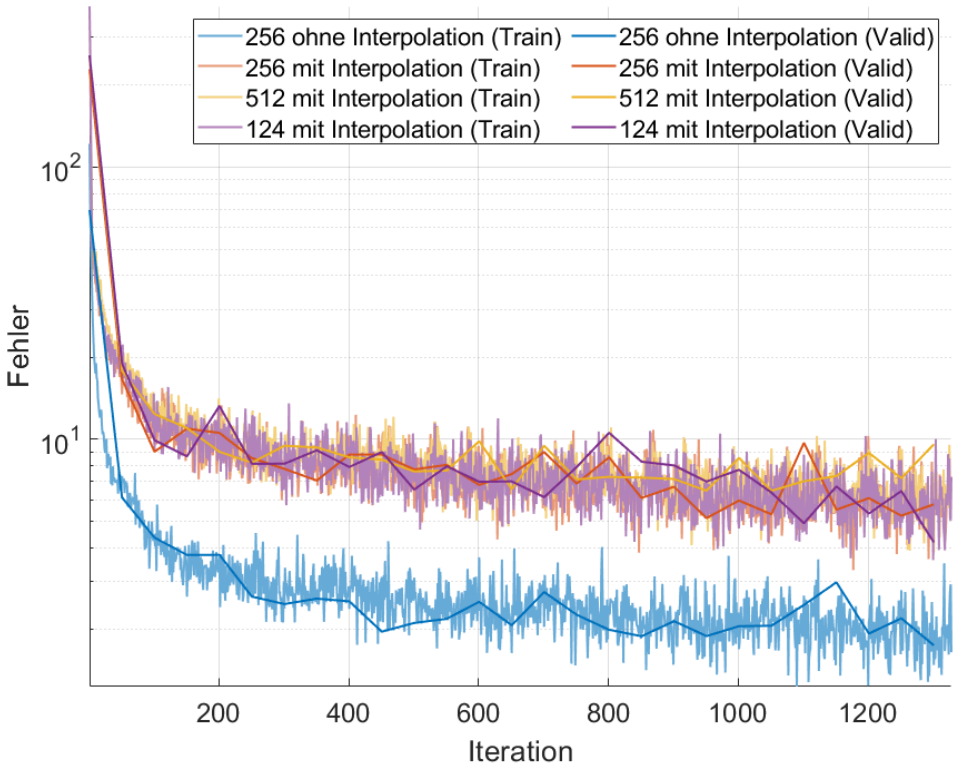


Abbildung 3.9: Trainingsverlauf der verschiedenen Interpolationsstufen

Testfehler

Der Testfehler dient als Maßstab zur Bewertung der Effektivität der durchgeführten Interpolation. Entscheidend ist dabei auch das persönliche Einschätzen einiger Vorhersagen, um Vor- und Nachteile verschiedener Interpolationsstufen zu bewerten. Der Testfehler soll ausschließlich die Genauigkeit der Puls-Vorhersage widerspiegeln und nicht durch Abweichungen beeinflusst werden, die durch Rauschen in irrelevanten Bereichen entstehen. Aus diesem Grund wird der Fehler nur in den Bereichen des Pulses betrachtet, in denen das zugehörige Label einen Wert von mehr als 0,001 aufweist. Werte unterhalb dieser Schwelle werden ignoriert, da sie keine relevanten Informationen enthalten und nicht zur Pulsform beitragen. Anschließend wird der Betrag der Differenz zwischen Vorhersage und Label berechnet. Zur Bestimmung einer einzelnen Kennzahl wird anschließend der Mittelwert aller Fehler berechnet die größer als null sind, damit ausgeschlossenen Werte keinen Einfluss auf die Mittelwertbildung haben. Dieser Fehler wird durch die Anzahl

an Label geteilt um den Testfehler pro Label zu erhalten. Dieser ist in Tabelle 3.7 dargestellt. Aus dem Testfehler Tabelle 3.7 ergibt sich, ähnlich wie beim Trainingsverlauf, dass eine Vorhersage ohne Interpolation den niedrigsten Fehler erzeugt. Der Mittelwert des Fehlers mit Interpolation ist um 17,04% höher als der Fehler ohne Interpolation.

Tabelle 3.7: Testfehler bei verschieden hoher Interpolation

Label-Datenpunkte	Testfehler
256 ohne Interpolation	0.1440
256 mit Interpolation	0.1684
512 mit Interpolation	0.1677
1024 mit Interpolation	0.1695

Zu beachten ist der Einfluss des ZBP des Testpulses auf den Testfehler. In Abbildung 3.10 ist der Testfehler in Abhängigkeit vom ZBP dargestellt. Der Graph zeigt eine Regressionsgerade, die durch die berechneten Testfehler in Abhängigkeit vom ZBP gelegt wurde. Es ist mit einer positiven Steigung zu rechnen, da die Vorhersage mit zunehmendem ZBP an Komplexität gewinnt. Die Regressionslinien zeigen dabei ein ähnliches Verlaufsmuster. Bei einer Interpolation mit einer Auflösung von 512 Punkten weist die Linie eine geringere Steigung auf, was darauf hindeutet, dass mit dieser Anzahl an Label-Datenpunkten die Vorhersage eines Spektrums mit höherem ZBP besser gelingt.

Vorhersagen auf dem Testdatensatz

In Unterabschnitt A.2.3 werden Pulse mit unterschiedlichem ZBP sowie variierender Auflösung der Labels dargestellt. Für jede Labelauflösung werden vier Vorhersagen mit jeweils unterschiedlichem ZBP gezeigt. Auf der X-Achse ist die zeitliche Verzögerung des Pulses aufgetragen, auf der Y-Achse die Amplitudenhöhe. Der Verlauf des Labels ist rot, der der Vorhersage blau dargestellt.

Die Vorhersage des einfachsten Pulses mit einem ZBP von 0,25 verläuft fast immer fehlerfrei. Kleine Wellen am Fuße des Pulses sind in allen Vorhersagen zu erkennen, die nicht dem Label entsprechen. Bei der Vorhersage mit einem ZBP von 2,8131 treten in sämtlichen Vorhersagen Fehler auf, wobei diese ähnliche Muster aufweisen. Bei dem Puls mit dem nächsthöheren ZBP weichen die Pulse stärker voneinander ab. Die Vorhersage mit 1024 Punkten weißt die Größte Ähnlichkeit zum Label auf. Mit abnehmender Auflösung

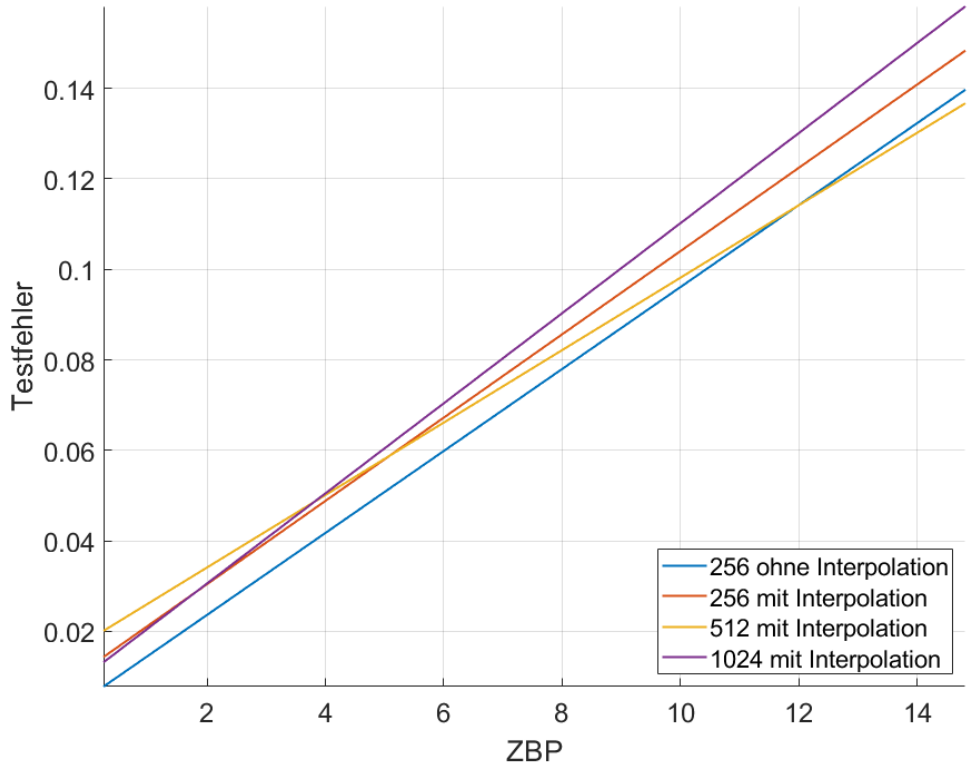


Abbildung 3.10: Regressionsgerade Testfehler in Abhängigkeit des ZBP

sinkt auch die Genauigkeit der Vorhersage. Bei der schwierigsten Vorhersage ähneln sich die Vorhersagen mit 1.024 Punkten und die ohne Interpolation am meisten. Die Vorhersagen mit 256 Punkten sowie 512 Punkten sind allerdings genauer in der Vorhersage.

Fazit

Die detaillierte Analyse des Einflusses der Hochskalierung der Labels zeigt, dass diese keinen positiven Effekt auf die Leistungsfähigkeit des Netzes hat. Sowohl der Anstieg der Trainingsdauer als auch der höhere Testfehler deuten darauf hin, dass durch die Interpolation keine Leistungsverbesserung erzielt wird. Auch eine qualitative Bewertung der individuellen Vorhersagen ergibt keinen erkennbaren Mehrwert. Aus diesem Grund wird im weiteren Verlauf auf die Verwendung der Interpolation verzichtet.

3.3.2 Auswahl und Vergleich verschiedener Kostenfunktionen

Die verwendeten Kostenfunktionen wurden in Unterabschnitt 2.3.2 behandelt. Sie haben alle unterschiedlichen Fokus. Der MSE (Unterunterabschnitt 2.3.2) ist eine standard Kostenfunktion, die besonders sensibel gegenüber Ausreißern ist, wohingegen der MAE (Unterunterabschnitt 2.3.2) besondere Stabilität aufweißt. Huber (Unterunterabschnitt 2.3.2) vereint mit einem Hyperparameter das Beste von MSE und MAE. Die Kostenfunktion Gauss (Unterunterabschnitt 2.3.2) wurde selber entwickelt und legt besonderen Fokus auf die Mitte der Vorhersage. Die letzte verwendete Kostenfunktion ist die Smooth (Unterunterabschnitt 2.3.2) mit dem Fokus darauf, dass die Vorhersage möglichst kein Rauschen enthält und eine glatte Kurve wiedergibt. Die Kostenfunktionen MSE, MAE und Huber wurden aufgrund ihrer etablierten Anwendung in Regressionsaufgaben ausgewählt, während die Gauss- und Smooth-Kostenfunktionen aufgrund ihres direkten Bezugs zu den Label berücksichtigt wurden. Die Wahl der verwendete Kostenfunktion hat einen erheblichen Einfluss auf den Trainingsprozess. Eine geeigneten Kostenfunktion beschleunigt das Training, sowie führt es zu besseren Ergebnissen. Nach der Bestimmung der Kosten werden die Gradienten mittels automatischem Differenzieren im Rückwärtsmodus berechnet. Durch eine gezielte Zusammensetzung der Kostenfunktion kann somit maßgeblich beeinflusst werden, wie die trainierbaren Parameter angepasst werden.

3.3.3 Trainingsablauf

Für das Training werden sämtliche 400.000 verfügbare Spektren verwendet, dadurch sind unterschiedliche Rausch-Niveaus enthalten. Zu jedem verwendete Label gehören vier Spektren mit unterschiedlichem Rausch-Niveau.

Der Trainingsablauf ist in Abbildung A.8 dargestellt. Es handelt sich dabei um eine individuelle programmierte Trainingsschleife, um die eine maximale Kontrolle über die Daten sowie größtmögliche Flexibilität bei der Anpassung des Trainingsprozesses bietet. Das Training beginnt mit dem Erstellen der Ergebnisdatei, welche zum Abspeichern der Trainingsergebnisse genutzt wird. Der Trainingsablauf durchläuft alle fünf verwendeten Kostenfunktionen. Bei jedem Durchlauf werden die Trainingsoptionen festgelegt und der fileDatastore erstellt, da dies nicht besonders Zeitintensiv ist. Zur Validierung werden 15% der Daten verwendet, die übrigen 85% dienen dem Training. Das VGGish wird mit den vortrainierten Parametern geladen. Anschließend werden die Minibatchqueues zur

Datenbereitstellung erstellt. Zu Beginn jeder Epoche werden die Trainingsdaten durchmischt, ein wichtiger Schritt, um zu vermeiden, dass Spektren mit identischem Label, aber unterschiedlichem Rauschniveau, in direkter Abfolge verarbeitet werden. Daraufhin wird die exponentiell fallende Lernrate mit dem Wert $k = 0,999$ angepasst. Es werden 256 Daten aus der Trainingsminibatschqueue geladen und mit der entsprechenden Kostenfunktion verarbeitet. Mittels der bestimmten Kosten wird der Gradientenabstieg bestimmt und die trainierbaren Parameter mit Adam angepasst. Der Validierungsfehler wird alle 50 Iterationen berechnet. Der gesamte Trainingsablauf erstreckt sich über fünf Epochen.

3.3.4 Ergebnisse

Im nachfolgenden Abschnitt werden die erzielten Resultate aus dem Training dargestellt und analysiert. Dabei stehen die Aspekte der Trainingszeit, des Trainingsverlaufs sowie der erzielte Testfehler in Abhängigkeit von den unterschiedlichen Kostenfunktionen im Fokus.

Trainingszeit

Die in der Kostenfunktion durchgeführten Berechnungen können die Trainingszeit maßgeblich beeinflussen. Insbesondere komplexe Rechenoperationen oder zeitintensive Prozesse innerhalb der Kostenfunktion können zu einer relevanten Verzögerung des Trainings führen. Die Trainingszeit soll Aufschluss darüber geben, ob die einzelnen Kostenfunktionen viel Zeit im Training beanspruchen. Da die Abweichung der Trainingszeit zwischen den verschiedenen Kostenfunktionen nur sehr gering ist (vgl. Tabelle 3.8). Die maximale Abweichung beträgt elf Minuten, was 5,26% des Mittelwerts entspricht. Somit lässt sich der Einfluss der Kostenfunktionen auf die Trainingszeit vernachlässigen.

Trainings- und Validierungsverlauf

In Abbildung 3.11 ist zu erkennen, dass das Netz mit allen Kostenfunktionen in der Lage ist den Fehler zu reduzieren. Die absolute Höhe des erreichten Fehlers ist jedoch nicht unmittelbar vergleichbar und erlaubt keine Aussage über die Vorhersagegenauigkeit der Netze, da die verschiedenen Kostenfunktionen Fehler in unterschiedlicher Größenordnung berechnen. Der unterschiedliche Verlauf zu betrachten, der widerspiegelt, welchen

Tabelle 3.8: Trainingszeit der verschiedenen Kostenfunktionen

Kostenfunktion	Trainingszeit
MSE	127,50 min
MAE	136,60 min
Gauss	135,21 min
Huber	138,83 min
Smooth	132,87 min

Einfluss die Kostenfunktionen auf das Training des Netzes haben. Es sind Ähnlichkeiten zwischen MSE und Smooth zu erkennen, beide dieser Funktionen sorgen für ein sehr starkes Variieren des Trainings- sowie Validierungsfehlers. Dies liegt daran, dass der Fehler in beiden Fällen quadriert wird und somit sensibel gegen Ausreißer ist. Diese kommen im Training öfter vor und sorgen so für starke Fluktuation im Fehler. Gauss und MAE hingegen haben deutlich weniger Fluktuation im Fehler. Huber startet hingegen mit geringer Fluktuation, die zum Ende des Trainings mehr wird. Der Abfall aller Fehlerfunktionen ist allerdings recht ähnlich, somit sorgt keine der verwendeten Funktionen für ein beschleunigtes Konvergieren auf ein Minimum. Zudem ist erkennbar, dass gegen Ende des Trainings bei allen Funktionen nur noch geringe Verbesserungen auftreten, was darauf hindeutet, dass jeweils ein Minimum erreicht wurde.

Testfehler

Der Testfehler ist das wichtigste Bewertungskriterium, da der Trainingsfehler, sowie Validierungsfehler von der Kostenfunktion abhängen und so keinen direkten Vergleich ermöglichen. Der verfügbare Datensatz enthält 20.000 Spektren, diese wurden nach dem ZBP sortiert und anschließend eine lineare Auswahl getroffen. Zum Bestimmen des Testfehlers wurden 1000 Spektren angewendet. So sind in dem Testdatensatz eine breite Anzahl von Spektren mit unterschiedlichem ZBP. Zur Fehlerbestimmung wurde der Mittelwert der absoluten Abweichung berechnet und anschließend über alle 1.000 Vorhersagen der Mittelwert gebildet, um so den Testfehler pro Vorhersage zu erhalten. Wie in Tabelle 3.9 zu sehen ist, weichen die ermittelten Testfehler nur gering voneinander ab. Das zeigt, dass das Training mit allen Kostenfunktionen erfolgreich war. Den geringsten Testfehler erreicht die Gauss-Kostenfunktion mit einem Testfehler von $49,7 \cdot 10^{-3}$. Die Abweichung zwischen dem niedrigsten und dem höchsten Testfehler beträgt 10,06%.

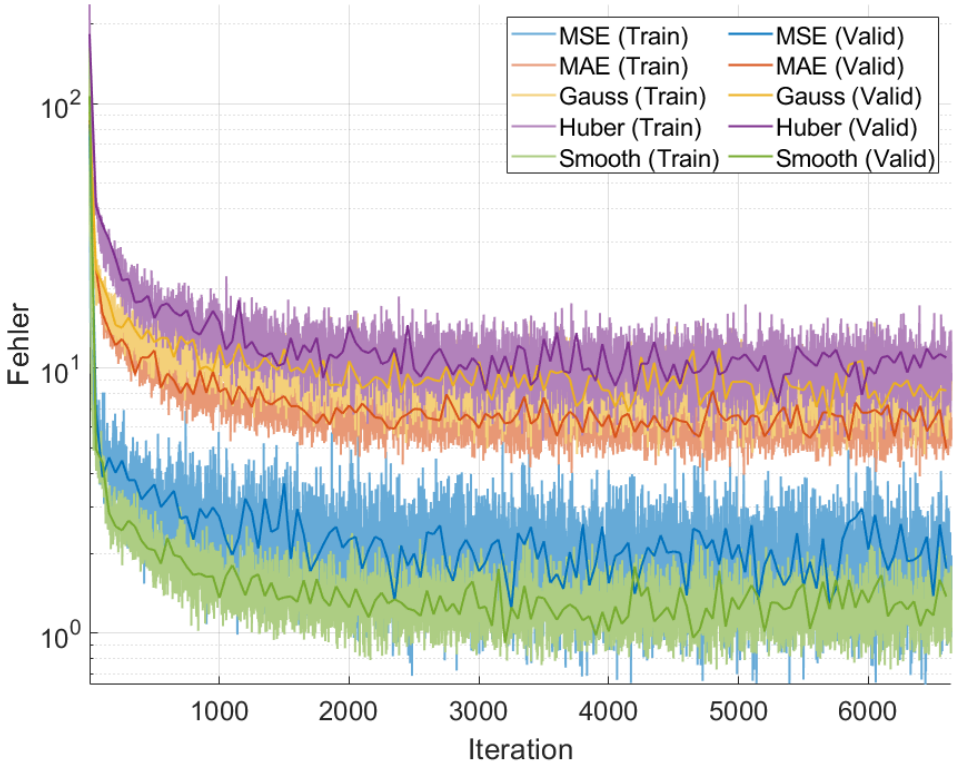


Abbildung 3.11: Trainings- Validierungsferlauf

Tabelle 3.9: Erreichter Testfehler

Kostenfunktion	Testfehler 10^{-3}
MSE	52,2
MAE	54,7
Gauss	49,7
Huber	51,1
Smooth	50,2

Vergleich der Vorhersagen der verschiedenen Kostenfunktionen

In dem Unterabschnitt A.2.4 werden einige Vorhersagen der Netze gezeigt, welche unter Verwendung verschiedener Kostenfunktionen trainiert wurden. Die Zeitliche Verzögerung ist auf der X-Achse aufgetragen, auf der Y-Achse die Amplitude des Pulses. In rot ist

das vorgegeben Label und in blau die Vorhersage des Netzes aufgetragen. Die Vorhersage aller Netze unterscheidet sich bei den beiden Pulsen mit dem geringsten ZBP nur geringfügig. Erst bei dem ZBP von 5,0035 sind signifikante Unterschiede erkennbar. Auffällig dabei ist, dass die mit der Gauss sowie MSE Kostenfunktion trainierten Netze zu verrauschteren Vorhersagen neigen. Im Gegensatz dazu erzeugen die übrigen Kostenfunktionen überwiegend glatte Pulsvorhersagen. Bei der Vorhersage mit dem höchsten ZBP zeigen sich deutliche Abweichung zwischen den Netzen. Es ist jedoch zu beachten, dass alle Netze sehr gut die Breite des Pulses erfassen, jedoch die Form nicht präzise rekonstruieren. Die größte Ähnlichkeit zum Label weist die Vorhersage unter Verwendung der Smooth-Kostenfunktion auf. Das Netz, welches mit der Gauss-Kostenfunktion trainiert wurde ist in Abbildung 3.12 dargestellt und enthält ebenfalls wie MSE Rauschen. Die Vorhersage des Pules mit dem geringsten weißt keinen erkennbaren Unterschied zu dem Label auf. Das lokale Minimum des Pulses mit dem von 2,4528 ist sehr gut getroffen. Mit steigendem erzeugt das Netz auch mehr Rauschen, teilweise auch in Bereichen wo der Puls null sein sollte. Die Vorhersage des Pulses mit dem Höchsten ZBP fällt insgesamt weniger überzeugend aus.

3.3.5 Diskussion und Bewertung der Ergebnisse

Aus den vorherigen Sektionen geht hervor, dass hinsichtlich der benötigten Trainingszeit keine relevanten Unterschiede zwischen den untersuchten Kostenfunktionen bestehen (Unterunterabschnitt 3.3.4). Ebenfalls sind bei dem Training- und Validierungsfehlerverlaufes keine signifikanten Differenzen zu erkennen (Unterunterabschnitt 3.3.4). Bei dem erreichten Testfehler aus Unterunterabschnitt 3.3.4 erzielt die Gauss Kostenfunktion die besten Ergebnisse. Bezogen auf die bewertung der individuellen Vorhersagen in Unterunterabschnitt 3.3.4 überzeugt hingegen die Smooth Kostenfunktion durch einen besonders glatten Verlauf und der besten Vorhersage bei dem Höchsten ZBP. Da es sich hierbei jedoch nur um einen kleinen Ausschnitt der insgesamt 1.000 im Test verwendeten Pulse handelt, ist die Gauss-Kostenfunktion insgesamt zu bevorzugen, da sie über den gesamten Testdatensatz hinweg die besseren Resultate liefert.

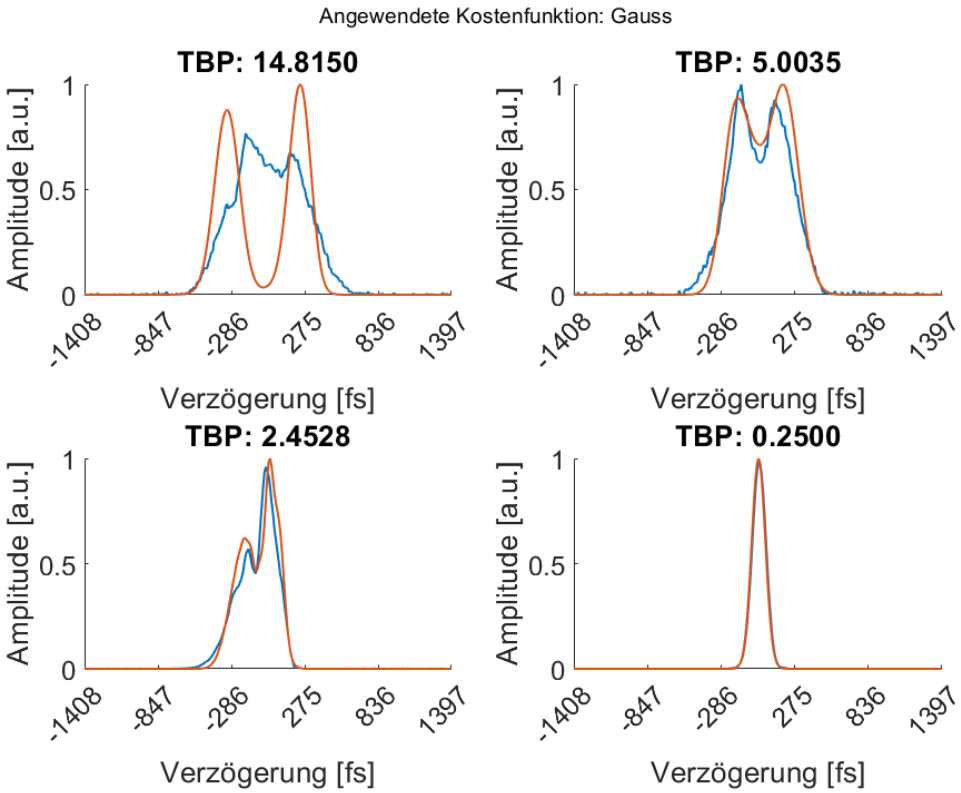


Abbildung 3.12: Vorhersage des mit der Gauss-Kostenfunktion trainierten Netzes mit verschiedenen Zeit-Bandbreite-Produkten. Die rote Linie stellt das vorgegebene Label dar und die blaue Linie stellt die Vorhersage des Netzes dar

3.4 Vorhersage der Phase

Es soll durch die Analyse des Spektrums der Phasenverlauf rekonstruiert werden. Die Ermittlung des Phasenverlaufs ist ein essenzieller Bestandteil, um den Pulsformer, des übergeordneten Projekts, präzise einstellen zu können.

Die Phase wird im Wellenlängenbereich vorhergesagt, da sie einen stetigeren Verlauf besitzt, welcher für ein neuronales Netz leichter vorherzusagen ist. Es wird sich bei der Vorhersage auf die Form des Phasenverlaufs bezogen, alle Phasenverläufe sind zur Stabilität des Trainings auf den Bereich von null bis eins normiert. Die Phase ist ausschließlich in dem Bereich definiert, in dem auch der Amplitudenverlauf existiert, da sie außerhalb dieses Bereichs nicht eindeutig bestimmt werden kann.

3.4.1 Auswahl der Kostenfunktionen

Auch hier finden wieder alle, in Unterabschnitt 2.3.2 behandelten Kostenfunktion Anwendung. Da diese bereits definiert sind, wird die Leistung der verschiedenen Kostenfunktionen analysiert. Die generellen Kostenfunktionen MSE, MAE und Huber haben dieselben Vor- und Nachteile wie bereits behandelt. Wohingegen Gauss und Smooth auf die Vorhersage der Amplitudenverläufe angepasst wurden. Es wird beobachtet, wie deren Konfiguration Einfluss auf die Vorhersage hat.

3.4.2 Trainingsablauf

Zur Erhöhung der Trainingsstabilität wird ein möglichst einfacher Datensatz verwendet. Daher kommt der rauschfreie Datensatz mit 100.000 Spektren und den dazugehörigen Label zum Einsatz. Der Trainingsablauf verhält sich identisch, wie in Unterabschnitt 3.3.3 beschrieben, allerdings wird der Phasenverlauf und nicht der Amplitudenverlauf als Label übergeben. Die gegebenen Kostenfunktionen aus Unterabschnitt 2.3.2 werden hier auch angewendet, um ein möglichst breites Spektrum von Ergebnissen zu erzielen. Als Lernratenfunktion wird der Kosinusabfall gewählt, da dieser über einen längeren Zeitraum eine höhere Lernrate aufrechterhält und somit eine stärkere Anpassung des Netzes ermöglicht. Zu Beginn des Trainings startet die Lernrate bei $0,005$ und sinkt zum Ende des Trainings auf den Wert 10^{-9}. Als Minibatchgröße werden 256 Datenpaare geladen.

3.4.3 Ergebnisse

Im Folgenden werden die Ergebnisse der Phasenvorhersage unter Betrachtung der verschiedenen Kostenfunktion dargestellt und hinsichtlich ihrer Ergebnisse analysiert.

Trainings- und Validierungsverlauf

Die Vorhersage der Phase erweist sich als deutlich komplexer als die Vorhersage des Amplitudenverlaufes. Wie in Abbildung 3.13 dargestellt ist, ist der Trainingsverlauf für alle Kostenfunktionen sehr ähnlich. Es beginnt mit einem starken Abfall und anschließend ist kaum ein Reduzieren des Fehlers zu erkennen.

Die Kostenfunktion von Huber und MAE zeigen über den Trainingsverlauf eine leichte Verbesserung des Fehlers, wohingegen Smooth, Gauss und MSE keinerlei Verbesserung aufzeigen. Das deutet klar darauf hin, dass kein Lernerfolg stattfindet.

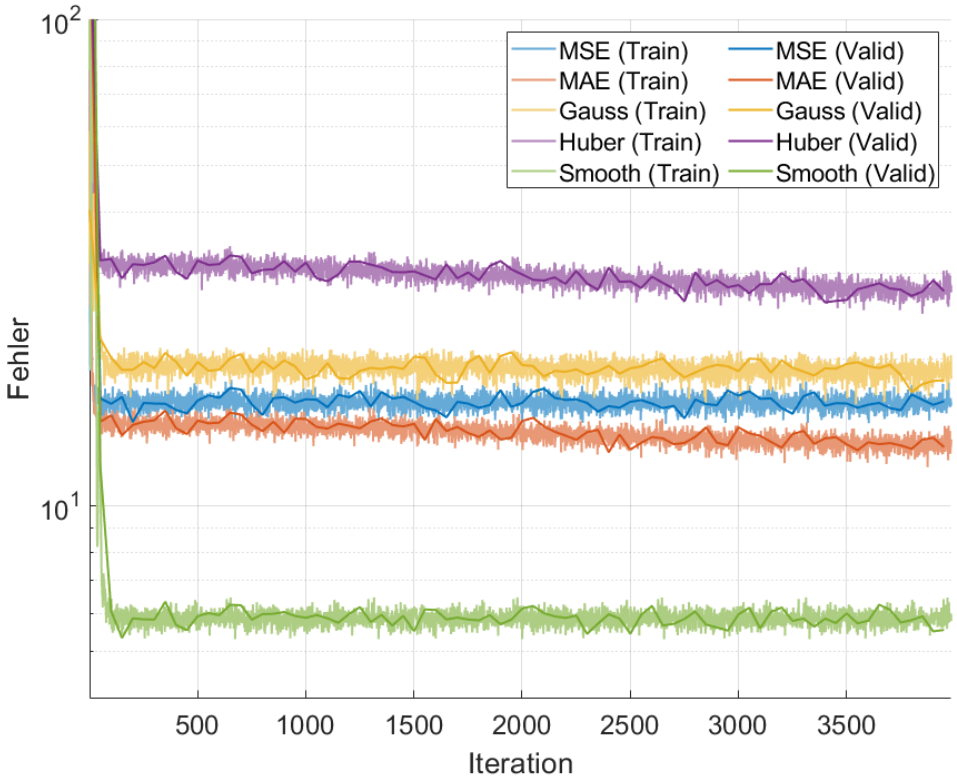


Abbildung 3.13: Trainingsverlauf der verschiedenen Kostenfunktionen

Gradientenverlauf

Da der Trainingsverlauf für alle Kostenfunktionen ein so lineares Verhalten aufweist, wird der Gradientenverlauf über die Trainingszeit beobachtet. Es handelt sich hier um den mittleren Gradienten des gesamten Netzes. Dieser gibt Aufschluss darüber, wie groß die Anpassungen sind, die am Netz vorgenommen werden. Ein niedriger Gradientenverlauf weist darauf hin, dass der Gradient während des automatischen Differenzierens sehr klein wird und infolgedessen verschwindet. Tritt dieses Problem auf, sind aufgrund der minimalen trainierbaren Parameteraktualisierung nur noch eingeschränkte Anpassungen des Netzes möglich. Eine weitere potenzielle Ursache für das verschwindenden des Gradienten

besteht darin, dass das Netz nicht in der Lage ist, relevante Muster oder Strukturen aus den Daten herausarbeiten kann und nicht genug Informationen besitzt, um eine präzise Vorhersage zu treffen. Da sich der Amplitudenverlauf und der Phasenverlauf numerisch in demselben Bereich befinden, sind numerische Probleme des Trainings ausgeschlossen.

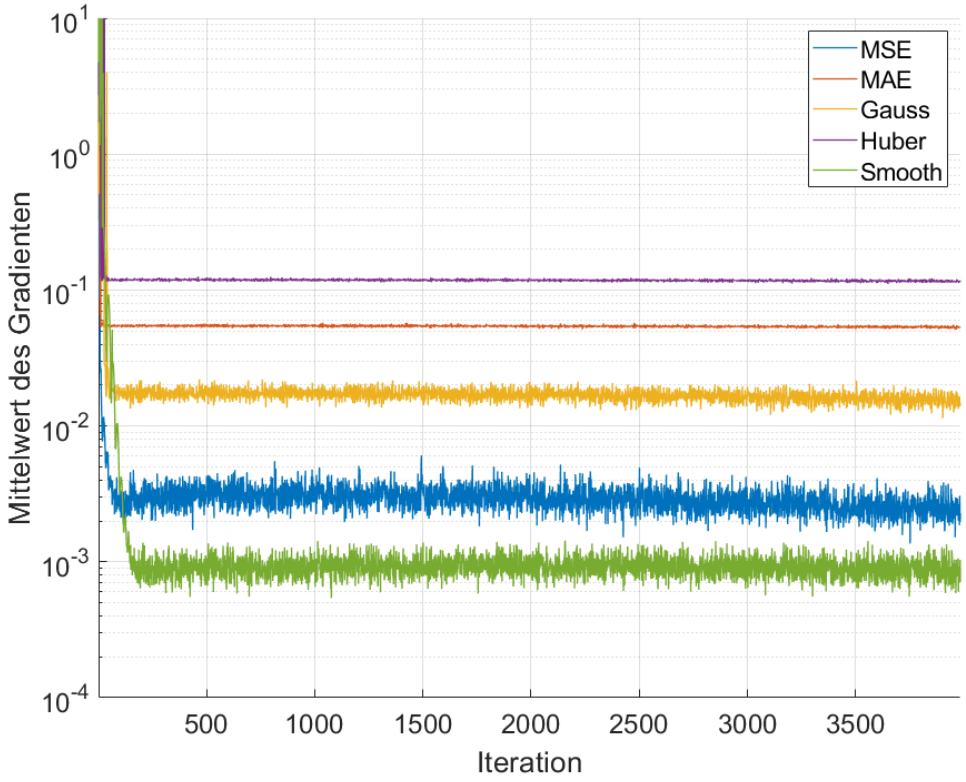


Abbildung 3.14: Gradientenverlauf der verschiedenen Kostenfunktionen

3.4.4 Testfehler

Der Testfehler wird auf gleiche Weise wie in Unterunterabschnitt 3.3.4 beschrieben, bestimmt.

Der Testfehler ist in diesem Fall nicht sehr aussagekräftig, da bei allen Kostenfunktionen das Training nicht erfolgreich war. Die erreichten Testfehler sind in Tabelle 3.10 dargestellt und weisen starke Unterschiede auf, wobei der Smooth den geringsten Fehler

erzielt. Die Vorhersagen sind in Unterabschnitt A.2.5 dargestellt. Keine der Vorhersagen entspricht dem vorgegebenen Label. Zwei verschiedene Verhaltensmuster sind zu erkennen, entweder besteht die Vorhersage des Netzes ausschließlich Nullen oder sie enthält Rausche, was unabhängig von der Eingabe ist. Eine Ausnahme bildet das Netz, das mit der Smooth-Kostenfunktion trainiert wurde. Dieses liefert für jedes eingegebene Spektrum eine identische Vorhersage mit einem kleinen Hügel, wodurch ein geringerer Fehler berechnet wird. Von einer gelungenen Vorhersage kann jedoch auch in diesem Fall nicht gesprochen werden.

Tabelle 3.10: Erreichte Testfehler der verschiedenen Kostenfunktionen

Kostenfunktion	Testfehler
MSE	6480,5986
MAE	47,9920
Gauss	27,3578
Huber	0,0841
Smooth	0,0798

3.4.5 Diskussion und Bewertung der Ergebnisse

Obwohl verschiedene Kostenfunktionen verwendet wurden, zeigen alle trainierten Modelle keinen erfolgreichen Trainingsverlauf. Die Netze generalisieren so stark, dass sie keine relevanten Zusammenhänge zwischen Eingabe und Phase erkennen. Infolgedessen bleiben die Vorhersagen unbrauchbar. Der Gradientenverlauf deutet darauf hin, dass bereits zu Beginn des Trainings keine relevanten Strukturen und Muster aus den gegebenen Daten herausgearbeitet werden können, die eine Verbesserung der Vorhersage ermöglichen. Somit müssen weitere Informationen über den Puls zur Verfügung gestellt werden, um eine verbesserte Vorhersage zu ermöglichen.

3.5 Vorhersage der Phase bei bekannter Amplitude

Wie aus Abschnitt 3.4 hervorgeht, stehen nicht genügend Informationen zur Verfügung, um den Phasenverlauf im Wellenlängenbereich allein auf Basis des Spektrums zuverlässig zu rekonstruieren. Um die Vorhersage der Phase zu ermöglichen, wird der Amplitudenverlauf im Wellenlängenbereich ebenfalls als Informationen in das Netz eingebunden. Dieser

Amplitudenverlauf im Wellenlängenbereich kann durch das gegebene Spektrum mathematisch bestimmt werden, jedoch ist die Rekonstruktion nicht Teil dieser Arbeit und wird in weiterführender Literatur wie in [7] behandelt. Der Amplitudenverlauf im Wellenlängenbereich wird in dieser Arbeit als gegeben betrachtet. Der Begriff *Amplitudenverlauf* bezieht sich im folgenden Abschnitt ausschließlich auf den Verlauf im Wellenlängenbereich. Damit das Netz in der Lage ist mehrere Eingangsinformationen verarbeiten zu können, muss der Aufbau des Netzes überarbeitet werden.

3.5.1 Aufbau eines Netzes mit zwei Eingängen

Zur Einbindung sowohl des Spektrums als auch des Amplitudenverlaufs wird ein Netz mit zwei separaten Eingängen erstellt. Zur Verarbeitung des Spektrums wird weiterhin das VGGish verwendet. Da das YAMNet in Abschnitt 3.2 ebenfalls sehr gute Ergebnisse erbracht hat und dabei sehr wenig Hardware-Ressourcen genutzt hat, wird es zur Analyse des Amplitudenverlaufs verwendet. Die Abbildung 3.15 stellt den Aufbau des Netzes dar. Da die Eingangsgröße des YAMNet nicht mit der Diemension des gegebenen Amplitudenverlaufs übereinstimmt, muss dieser für die weiter Verarbeitung angepasst werden. Dies erfolgt in einer individuellen Umformungsschicht, welche nachfolgend in Unterabschnitt 3.5.2 näher behandelt wird. In einer Verkettungschicht werden die beiden gleichgroßen Ausgänge der Netze zusammengeführt. Nach dieser werden drei Funktionsblöcke angewendet. Diese Funktionsblöcke beinhalten jeweils eine vollverbundenen Schichten, gefolgt von einer Dropout Schicht, ergenzt durch eine Bacht-Normalisierungsschicht und abschließend eine Relu-Aktivierungsfunktionen. Dieser Aufbau orientiert sich an dem Funktionsblock des YAMNet und wird durch Dropoutschichten ergänzt, um mehr Variation in der Vorhersage zu erzeugen und einer Generalisierung des Netzes vorzubeugen. Die Größe der vollverbundenen Schichten startet nach der Verkettungsschicht mit 512 Neuronen, verdoppelt sich anschließend auf 1.024 und halbiert sich mit jeder Schicht zum Ausgang hin. Die Ausgangsschicht hat die vorgegebene Größe von 256, um die Vorhersage der Phase auszugeben. Da bereits in Abschnitt 3.4 ermittelt wurde, dass ein verschwindender Gradient ein Problem darstellen könnte, wird vor der Ausgangsschicht noch eine Addierungsschicht eingebaut. Dies ermöglicht eine effizientere Rückführung des Gradienten, wie sie auch in MobileNet eingesetzt wird, um dem Problem des verschwindenden Gradienten entgegenzuwirken.

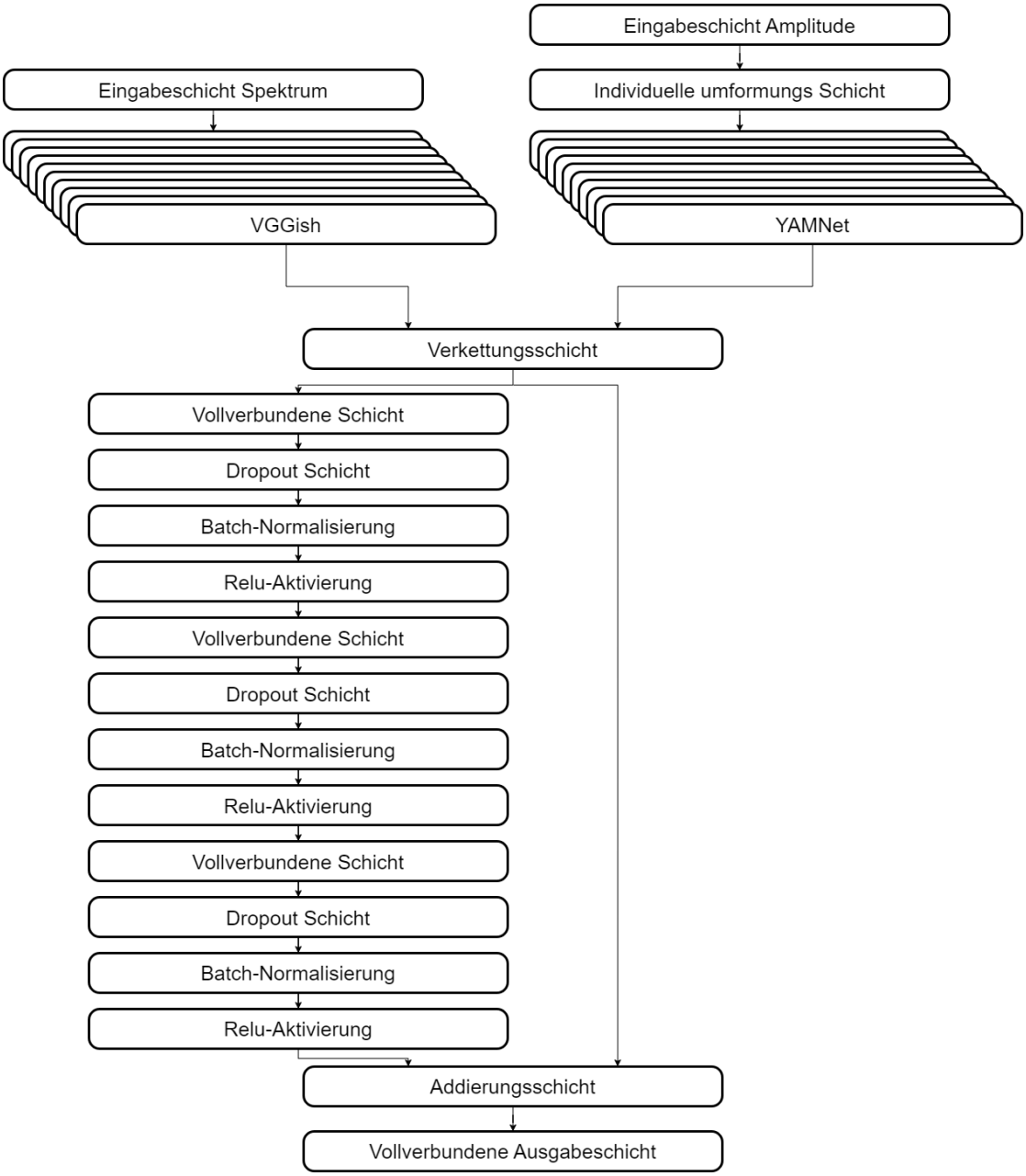


Abbildung 3.15: Aufbau des Netzes mit zwei Eingängen

3.5.2 Aufbereiten der Amplitude für die Eingangsgröße des YAMNet

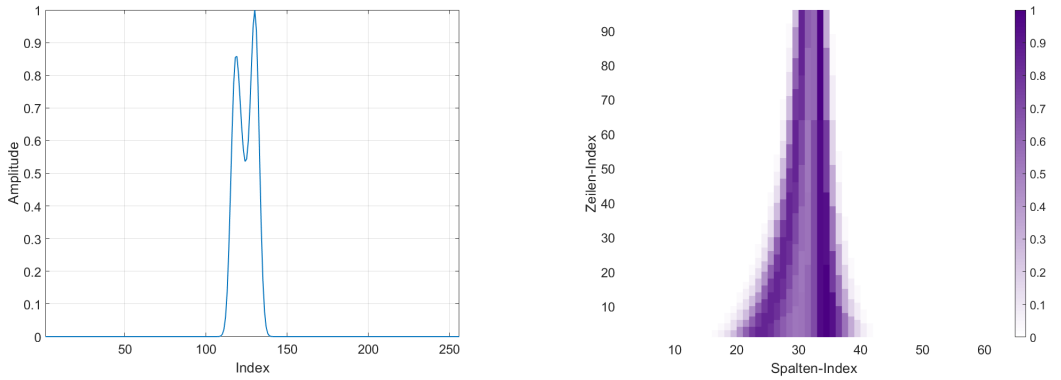
Das YAMNet benötigt eine Eingabematrix der Größe 96x64x1, somit muss der Amplitudenverlauf mit der ursprünglichen Diemension von 256x1 aufbereitet werden. Um

möglichst viele Informationen des Amplitudenverlaufs in das YAMNet zu geben, wird ein möglichst großer Teil mit der vollen Auflösung des Amplitudenverlaufs dargestellt. Es soll ebenfalls die gesamte Breite des Amplitudenverlaufs dargestellt werden. Zur Umsetzung diesen Ansatzes, wird die Matrix zeilenweise aus dem Amplitudenverlauf aufgebaut. Zu Beginn wird die Zeile mit der größtmöglichen Auflösung des Amplitudenverlaufes implementiert. In jeder neuen Zeile wird ein breiterer zentrierter Ausschnitt des Amplitudenverlaufs dargestellt. In der letzten Spalte ist der gesamte Amplitudenverlauf enthalten, allerdings in verringerter Auflösung. Die größtmögliche Auflösung wird zu Beginn des Prozesses in der Matrix erreicht. So wird aus dem eindimensionalen Amplitudenverlauf eine zweidimensionale Matrix erschaffen. Ein Beispiel des Prozesses wird in Abbildung 3.16 dargestellt. In der untersten Zeile von Abbildung 3.16b ist die höchstmögliche Auflösung des Amplitudenverlaufs zu erkennen. Die Auflösung reduziert sich in den darüberliegenden Zeilen, bis in der obersten Zeile der vollständige Amplitudenverlauf abgebildet wird. Auf diese Weise erhält das Netzwerk eine Referenz darüber, an welcher Position sich der Amplitudenverlauf befindet.

Diese Funktionalität muss in dem Netz verarbeitet werden, damit die Kostenfunktion den Amplitudenverlauf in der ursprünglichen Form verarbeiten kann. Um die Funktionalität in das Netz mit einzubauen, wird eine individuelle Umformungsschicht mit dem Namen „customReshapeLayer“ erstellt. Diese Schicht wird zwischen die Eingangsschicht, in die der Amplitudenverlauf eingeführt wird, und die Eingangsschicht des YAMNet platziert. Sie enthält keine trainierbaren Parameter, somit enthält diese Schicht keine Funktionalität bei dem Gradientenbestimmen.

Trainingsablauf

Der Trainingsablauf unterscheidet sich nur kaum von den in Unterabschnitt 3.3.3 beschriebenen. Der verwendete Datensatz enthält 100.000 Spektren, Amplitudenverläufe sowie Phasenverlaufe ohne Rauschen. Es werden 85% der Daten zum Trainineren und 15% zum Validieren genutzt. Die Minibatchqueue gibt jeweils ein Datentupel aus Spektrum, Amplitudenverlauf und Phasenverlauf aus. Es wird eine Minibatchgröße von 128 verwendet, um präziseres Trainingergebnis durch genaueres Anpassen der Gradienten zu ermöglichen. Als Lernratenfunktion wird der Kosinusabfall mit dem Startwert von $0,005$ und dem Endwert von 10^{-9} genutzt. Der verwendete Optimierungsalgorithmus ist Adam. Das Training erstreckt sich über 20 Epochen.



(a) Eindimensionale Darstellung des Amplitudenverlaufs

(b) Zweidimensionale Darstellung als Eingang für das YAMNet

Abbildung 3.16: Umwandlung des Amplitudenverlaufs in eine zweidimensionale Darstellung

Kostenfunktionen

Das Netz wird mit drei verschiedenen Kostenfunktionen trainiert. Es wird eine Kostenfunktion definiert, die sich auf den übergebenen Amplitudenverlauf bezieht. Diese ist in Unterunterabschnitt 2.3.2 beschrieben und gewichtet Fehler im Existenzbereich des Amplitudenverlaufs deutlich stärker als außerhalb. Auf diese Weise soll erreicht werden, dass sich das Netz bei der Vorhersage vorrangig auf den Bereich konzentriert, in dem eine Phase existiert. Ziel ist es, die relevanten Regionen stärker zu berücksichtigen und somit die Vorhersagegenauigkeit zu verbessern. Zu Referenzzwecken werden ebenfalls der MSE und der MAE als Kostenfunktion verwendet.

3.5.3 Ergebnisse

Nachfolgend wird der Trainings- und Validierungsverlauf sowie der Testfehler und einige Vorhersagen betrachtet.

Trainings- und Validierungsverlauf

In Abbildung 3.17 ist der Trainings- und Validierungsfehlerverlauf aufgetragen. Es ist bei allen Kostenfunktion ein Abfall des Fehlers zu erkenne, das bedeutet, dass mit allen Kostenfunktionen etwas gelernt wird. Das Training mit der gewichteten Kostenfunktion

beginnt bereits sehr früh, eine niedrige Steigung aufzuweisen. Die Kostenfunktion MAE zeigt bis zum Ende des Trainings noch Verbesserungen, allerdings fällt der Fehler nur noch sehr langsam ab. Die MSE Kostenfunktion weist den besten Trainingsverlauf auf, da sie über viele Iterationen noch Verbesserungen erreicht und gegen Trainingsende einen geringen Verlust aufweißt.

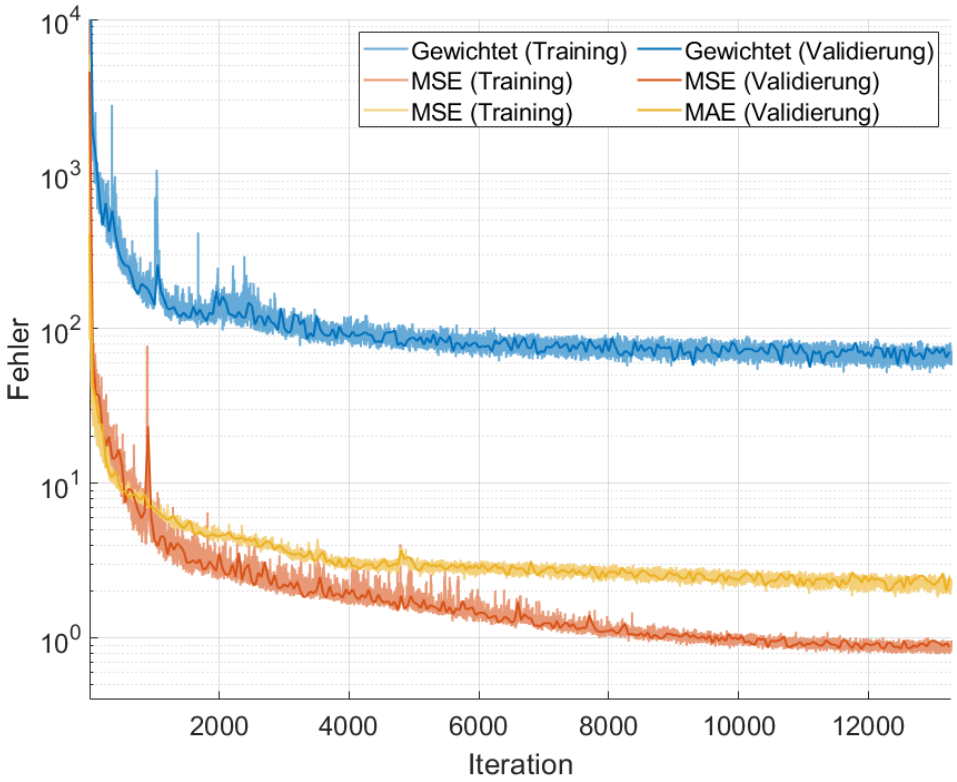


Abbildung 3.17: Verlauf des Trainings- und Validierungsfehlers beim Netz mit mehreren Eingängen und unterschiedlichen Kostenfunktionen

Testfehler

Hier wird ebenfalls die selbe Testfehlerermittlung wie in Unterunterabschnitt 3.3.4 beschrieben angewendet. Aus Tabelle 3.11 geht hervor, dass sowohl die gewichtete Kostenfunktion, als auch MAE-Kostenfunktion erfolgreiche Ergenisse erziehlen, allerdings tritt bei der MSE-Kostenfunktion eine unerwartete Anomalie auf. Die Vorhersage des Netzes, welches mit dem MSE trainiert wurde, liefert sehr unzutreffende Ergebnisse. Eine

mögliche Ursache für so ein Verhalten kann Überanpassung sein, jedoch würde der Validierungsfehler dies bereits im Training andeuten. Da der Validierungsfehler während des Trainings keine große Abweichung zu dem Trainingsfehler aufweist, kann es sich hier nicht um Überanpassung handeln. Weitere Gründe für ein solches Verhalten können auf einen fehlerhaften Testdatensatz zurückgeführt werden. Da der verwendete Testdatensatz allerdings in dem gleichen numerischen Bereich befindet und die anderen trainierten Netze aussagekräftige Ergebnisse liefern, wird diese Option ebenfalls ausgeschlossen. Zuletzt können noch mögliche Fehler beim Speichern der Ergebnisse die Ursache für eine solche Vorhersage sein. Diese Vorhersage hat Ähnlichkeiten mit einer Vorhersage eines Netzes, welches nicht trainiert wurde. Es handelt sich um reines Rauschen ohne Erkennung von Mustern, wie in Abbildung A.23 zu erkennen ist. Es könnte sein, dass beim Abspeichern des Netzes die erste Version des Netzes abgespeichert und nicht die trainierte Version. Dies ist jedoch ebenfalls nicht möglich, da alle Netze auf die gleiche Weise abgespeichert wurden und dieser Fehler bei anderen Netzen nicht aufgetreten ist. Anwenden von Daten die in dem Trainingszyklus verwendet wurden zeigen eine verlässliche Vorhersage (siehe Abbildung A.22), somit kann auch sichergestellt werden, dass die trainierte Version des Netzes abgespeichert wurde. Die Ursache dieses Fehlverhaltens kann nicht eindeutig ermittelt werden.

Die Vorhersage des Netzes, welches mit dem MAE trainiert wurde, erreicht einen geringeren Trainingsfehler. In der Analyse der Vorhersagen in Abbildung A.25 ist zu erkennen, dass das Netz eine sehr gute Vorhersage bei einem kleinen ZBP erreicht. Es ist sehr gut in der Lage, den Bereich einzuschätzen, in dem sich die Phase befindet. Bei einem hohen ZBP weicht diese Vorhersage allerdings ab. Das mit der gewichteten Kostenfunktion trainierte Netz erreicht keine so klaren Bereiche und weist mehr Schwingungen auf. Bei der Vorhersage des Pulses mit dem geringsten ZBP zeigt sich eine signifikante Abweichung außerhalb des relevanten Bereichs des Phasenverlaufs.

Tabelle 3.11: Testfehler verschiedener Kostenfunktionen bei einem Netz mit mehreren Eingängen

Kostenfunktion	Testfehler
Gewichtet	0,0887
MSE	1831,7975
MAE	0,0677

3.5.4 Bewertung der Ergebnisse

Da die Vorhersage der Phase, ausschließlich mit Verwenden des Spektrums, nicht erfolgreich war, wie in Abschnitt 3.4 ermittelt, wurde die Zielsetzung erreicht, indem durch Einbeziehen des Amplitudenverlaufs eine verbesserte Vorhersage erzielt werden konnte.

4 Fazit

Das Ziel dieser Arbeit war es, FROG-Traces mithilfe von neuronalen Netzen zu analysieren und daraus den Amplituden- sowie den Phasenverlauf ultrakurzer Laserpulse zu rekonstruieren. Zu Beginn dieser Arbeit wurde ermittelt, welches vortrainierte Netz für die Analyse der Spektren am besten geeignet ist. Anschließend wurde das ermittelte Netz verwendet, um den Amplituden-, sowie Phasenverlauf vorherzusagen. Um die Vorhersage des Phasenverlaufs zu verbessern, wurde ein Netz mit mehreren Eingängen implementiert, welches den Amplitudenverlauf im Wellenlängenbereich mit in das Training aufnimmt.

Zur Bewertung der Netzarchitekturen wurden Testverfahren erarbeitet, welche die Vorhersagegenauigkeit, die Rauschresistenz, sowie die Trainingsgeschwindigkeit berücksichtigten. Anhand dieser Tests wurde ermittelt, dass das VGGish Netz am besten in der Lage ist, Informationen aus den gegebenen Spektren zu extrahieren. Es weist bereits bei kleinen Datensätzen eine hohe Genauigkeit auf, zeigt geringe Sensibilität gegenüber Rauschen, überzeugt weiterhin mit einer schnellen Trainingszeit.

Nachdem diese Erkenntnis gewonnen wurde, hat das Netz Anwendung zur Vorhersage des Amplitudenverlaufes im Zeitbereich gefunden. Zunächst wurde eine Interpolation der Label getestet, welche allerdings keine Erfolge erzielte. Anschließend wurden individuelle Kostenfunktionen implementiert und deren Einfluss auf den Lernerfolg bewertet. Besonders die Gauss-Kostenfunktion lieferte die präzisesten Ergebnisse.

Die Vorhersage des Phasenverlaufes im Wellenlängenbereich stellte sich als deutlich schwieriger heraus. Das Training des Netzes mit verschiedenen Kostenfunktionen stellte sich als nicht erfolgreich heraus. Keine der verwendeten Kostenfunktionen hat zu zufriedenstellenden Ergebnissen geführt.

Durch die zusätzliche Integration des Amplitudenverlaufs im Trainingsprozess wird ein neuronales Netz mit mehreren Eingängen realisiert. Diese Erweiterung führt zu einer signifikanten Verbesserung der Vorhersagefähigkeit des Phasenverlaufs.

Die aufgeführten Untersuchungen zeigen, dass neuronale Netze in der Lage sind, FROG-Traces zu analysieren und den Amplituden- sowie den Phasenverlauf vorherzusagen. Die Vorhersage des Amplitudenverlaufes erreicht eine höhere Genauigkeit, als die Vorhersage des Phasenverlaufes. Diese ist nur durch zusätzliches Einbinden des Amplitudenverlaufes im Wellenlängenbereich erfolgreich.

5 Ausblick

Die Phasenvorhersage zeigte im Vergleich zur Vorhersage der Amplitude eine deutlich geringere Genauigkeit auf. Zukünftige Arbeiten sollten spezifisch angepasste Kostenfunktionen implementieren, die Ambiguitäten der Phase gezielt berücksichtigen. Dadurch sollte ein stabilerer Trainingsverlauf sowie ein Optimieren der Vorhersagen ermöglicht werden.

Zur Phasenvorhersage basierend ausschließlich auf dem Spektrum sollten auch Netzarchitekturen untersucht werden, die einen stabilen Gradientenverlauf aufweisen und mit mehr Komplexität ausgestattet sind um dem Unteranpassen entgegenzuwirken.

Der bisher als gegeben betrachtet Amplitudenverlauf im Wellenlängenbereich könnte zukünftig mathematisch aus den Spektren ermittelt werden. Dadurch ließe sich die Phasenvorhersage nur aufgrund Lage des Spektrums realisieren. Dies würde die Anwendung auf reale Messdaten vereinfachen.

Da die in dieser Arbeit trainieren Netze alle denselben Eingang nutzen, erscheint ein zusammenschluss zu einem Nezt mit mehreren Ausgabeparametern sinnvoll. Ein solches Netz könnte gleichzeitig das ZBP, den Amplitudenverlauf im Zeitbereich und den Phasenverlauf im Wellenlängenbereich vorhersagen.

Die in dieser Arbeit erzielten Ergebnisse sollten in das übergeordnete Projekt überführt und für die Steuerung des Pulsshapsers nutzbar gemacht werden. Insbesondere das VGGish sollte in dem verstärkenden Trainingsloop implementiert werden. Dazu müssen die nötigen Anpassungen vorgenommen werden, um das Netz in einem verstärkenden Trainingsloop einsetzen zu können.

Darauffolgend sollte das Netz auf reale Messdaten angewendet werden, um die Übertragbarkeit der Ergebnisse zu prüfen. Dazu sind gezielte Anpassungen an real vorliegende Bedingungen notwendig. Die in dieser Arbeit beobachtete schnelle Anpassungsfähigkeit des VGGish bietet hierfür eine vielversprechende Grundlage.

Literaturverzeichnis

[1] AudioSet: *AudioSet*. 2025. – URL https://research.google.com/audioset/. – Zugriffsdatum: 19.03.2025

[2] Cramer, J. ; Wu, H-.H. ; Salamon, J. ; Bello, J. P.: Look, Listen and Learn More: Design Choices for Deep Audio Embeddings. In: *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*. Brighton, UK, May 2019

[3] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – http://www.deeplearningbook.org

[4] Group, Visual G.: *Visual Geometry Group*. 2025. – URL https://www.robots.ox.ac.uk/~vgg/. – Zugriffsdatum: 24.02.2025

[5] Huang, Gao ; Liu, Zhuang ; Maaten, Laurens van der ; Weinberger, Kilian Q.: *Densely Connected Convolutional Networks*. 2018. – URL https://arxiv.org/abs/1608.06993

[6] imageNet: *imageNet*. 2025. – URL https://www.image-net.org/index.php. – Zugriffsdatum: 19.03.2025

[7] Jafari, Rana ; Jones, Travis ; Trebino, Rick: 100frequency-resolved optical gating. In: *Optics Express* 27 (2019), Januar, Nr. 3, S. 2112. – URL http://dx.doi.org/10.1364/OE.27.002112. – ISSN 1094-4087

[8] Jünemann, Klaus: *Maschinelles Lernen und Neuronale Netze*. HAW Hamburg, 2022

[9] Keller, U.: *Ultrafast Lasers: A Comprehensive Introduction to Fundamental Principles with Practical Applications*. Springer International Publishing, 2022 (Graduate Texts in Physics). – URL https://books.google.de/books?id=jjuEzgEACAAJ. – ISBN 9783030825317

[10] MathWorks: *Matlab Eintrag von VGGish*. 2025. – URL https://de.mathworks.com/help/deeplearning/ref/vggish.html. – Zugriffsdatum: 24.02.2025

[11] Pan, Sinno J. ; Yang, Qiang: A Survey on Transfer Learning. In: *IEEE Transactions on Knowledge and Data Engineering* (2010)

[12] Places365: *Places365*. 2025. – URL http://places2.csail.mit.edu/index.html. – Zugriffsdatum: 19.03.2025

[13] Richman, Rick Trebino Kenneth W. DeLong David N. Fittinghoff John N. Sweetser Marco A. Krumbügel Bruce A. ; Kane, Daniel J.: Measuring ultrashort laser pulses in the time-frequency domain using frequency-resolved optical gating. In: *Review of Scientific Instruments* 68 (1997), Nr. 9

[14] Sandler, Mark ; Howard, Andrew ; Zhu, Menglong ; Zhmoginov, Andrey ; Chen, Liang-Chieh: *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. – URL https://arxiv.org/abs/1801.04381

[15] Szegedy, Christian ; Liu, Wei ; Jia, Yangqing ; Sermanet, Pierre ; Reed, Scott ; Anguelov, Dragomir ; Erhan, Dumitru ; Vanhoucke, Vincent ; Rabinovich, Andrew: *Going Deeper with Convolutions*. 2014. – URL https://arxiv.org/abs/1409.4842

[16] TensorFlow Team, plakal and Manoj Plakal: *YAMNet: An Audio Event Classification Model*. 2024. – URL https://github.com/tensorflow/models/tree/master/research/audioset/yamnet. – Accessed: 30.01.2025

[17] Trebino, R.: *Frequency-Resolved Optical Gating: The Measurement of Ultrashort Laser Pulses*. Springer Science & Business Media, 2000. – URL https://books.google.de/books/about/Frequency_Resolved_Optical_Gating_The_Me.html?id=yfLIg6E69D8C&redir_esc=y. – ISBN 978-1402070662

[18] Wikipedia: *Aufbau der FROG-Messstrecke*. 2021. – URL https://de.m.wikipedia.org/wiki/Datei:Frequency_resolved_optical_gating_20210207.svg. – Accessed: 24.03.2025

[19] YouTube8M: *YouTube8M*. 2025. – URL https://research.google.com/youtube8m/. – Zugriffsdatum: 19.03.2025

A Anhang

A.1 Theoretische Grundlagen

A.1.1 Aufbau der Verwendeten Netze

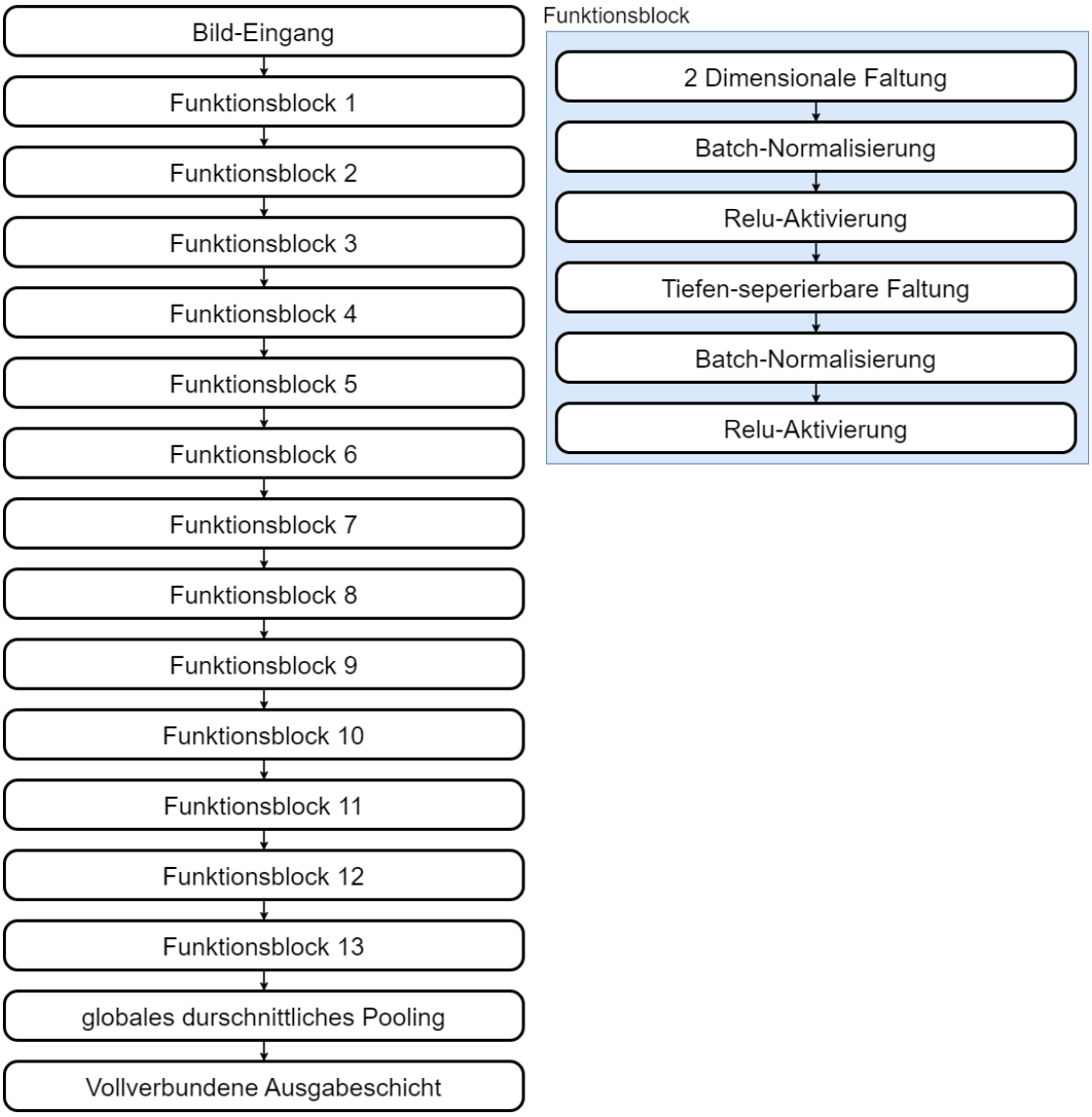


Abbildung A.1: Aufbau des YAMNet

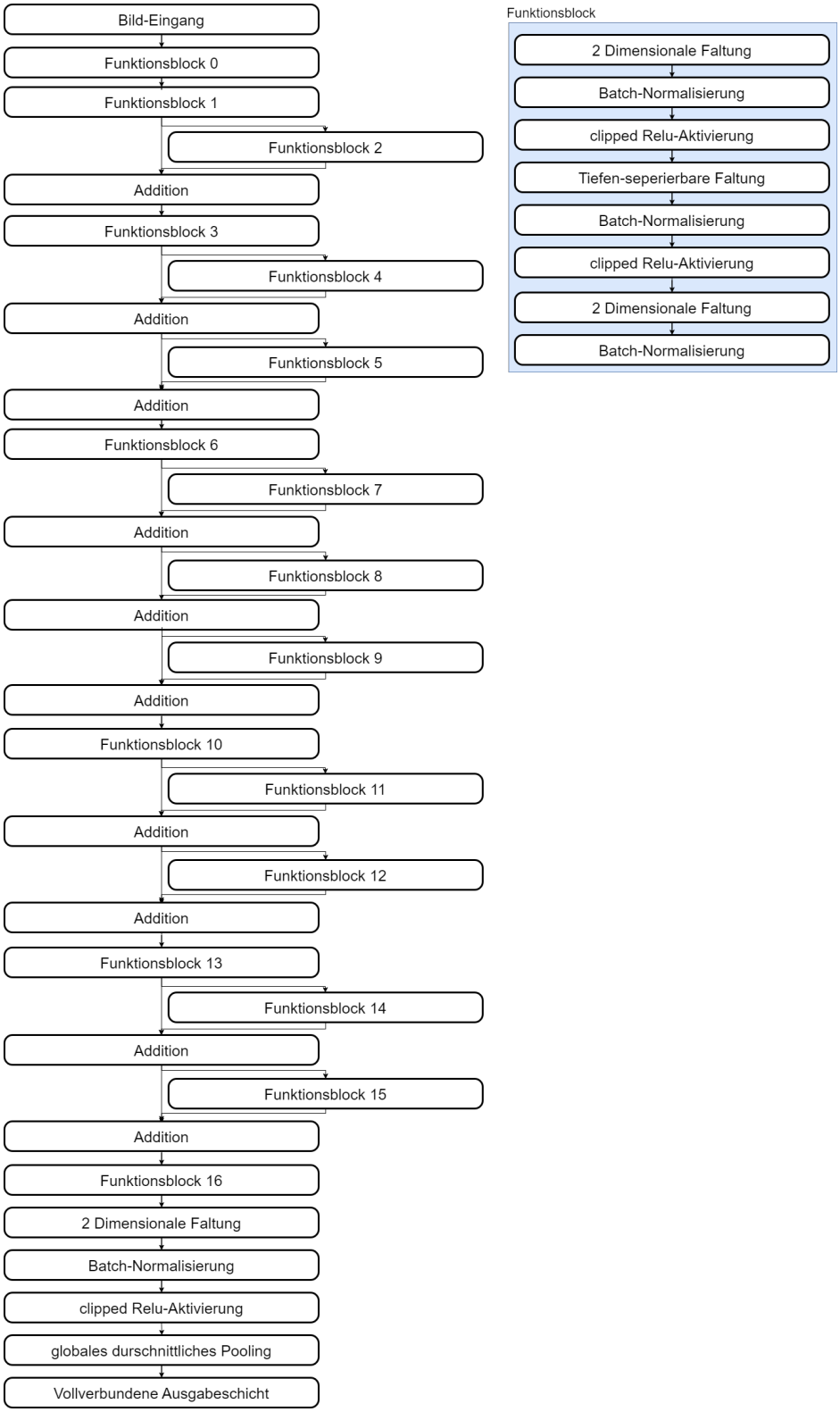


Abbildung A.2: Aufbau des MobileNet

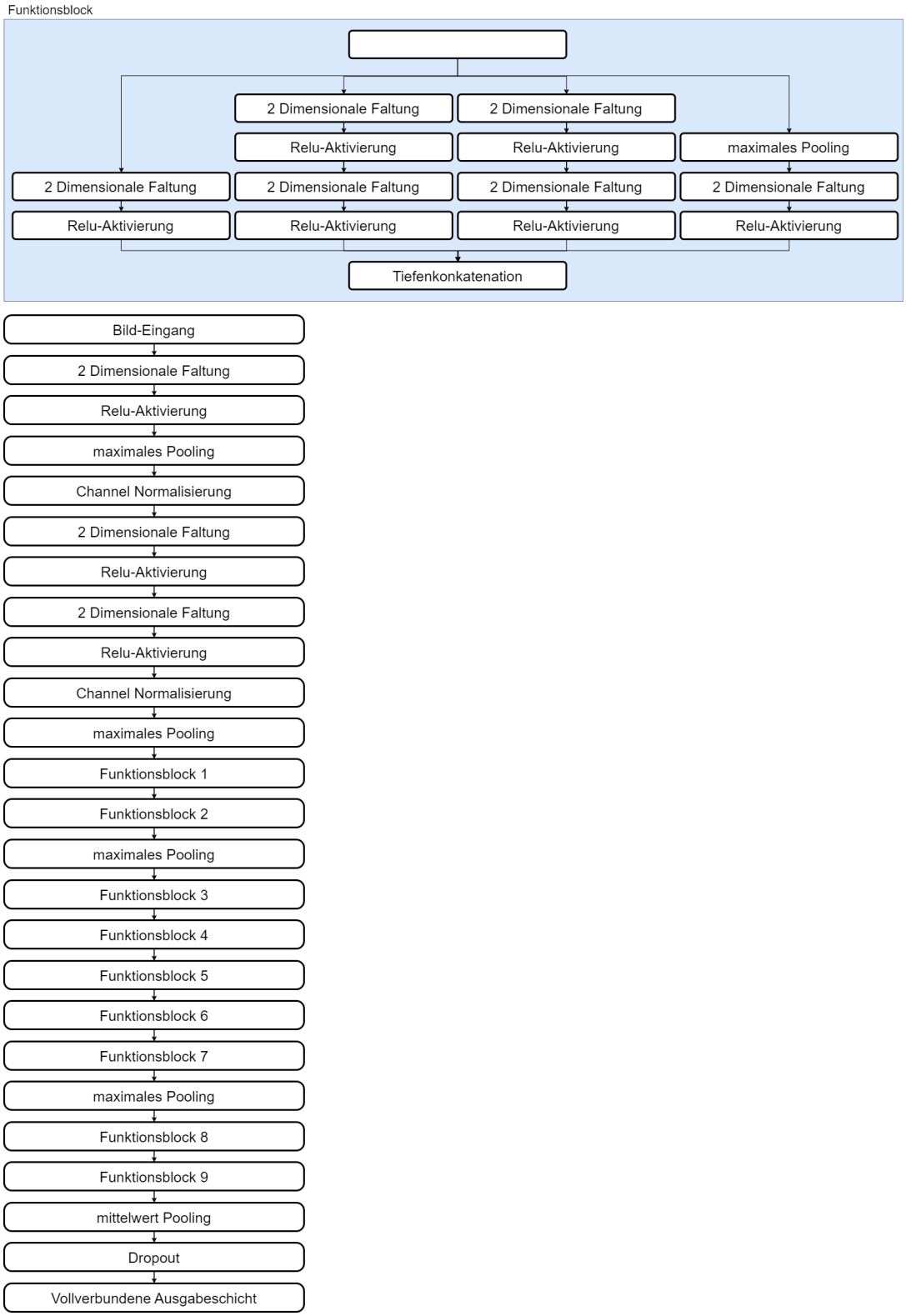


Abbildung A.3: Aufbau des GoogLeNet

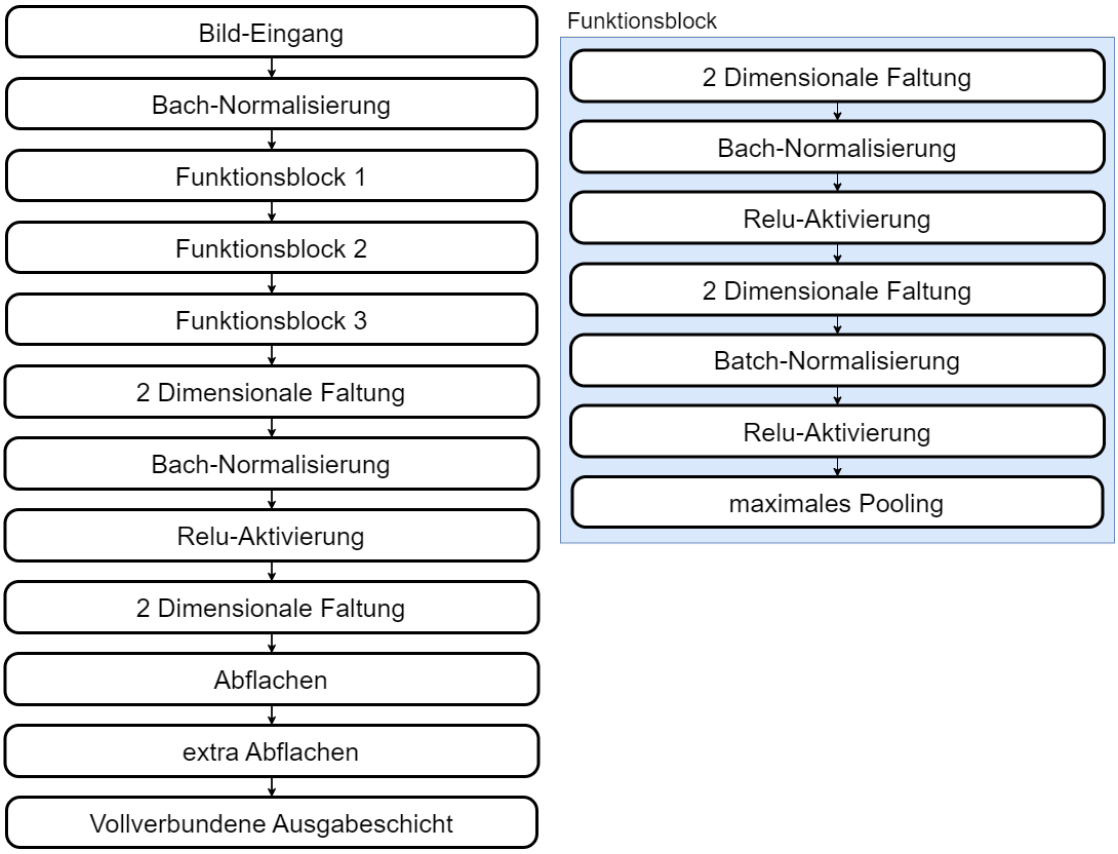


Abbildung A.4: Aufbau des OpenL3

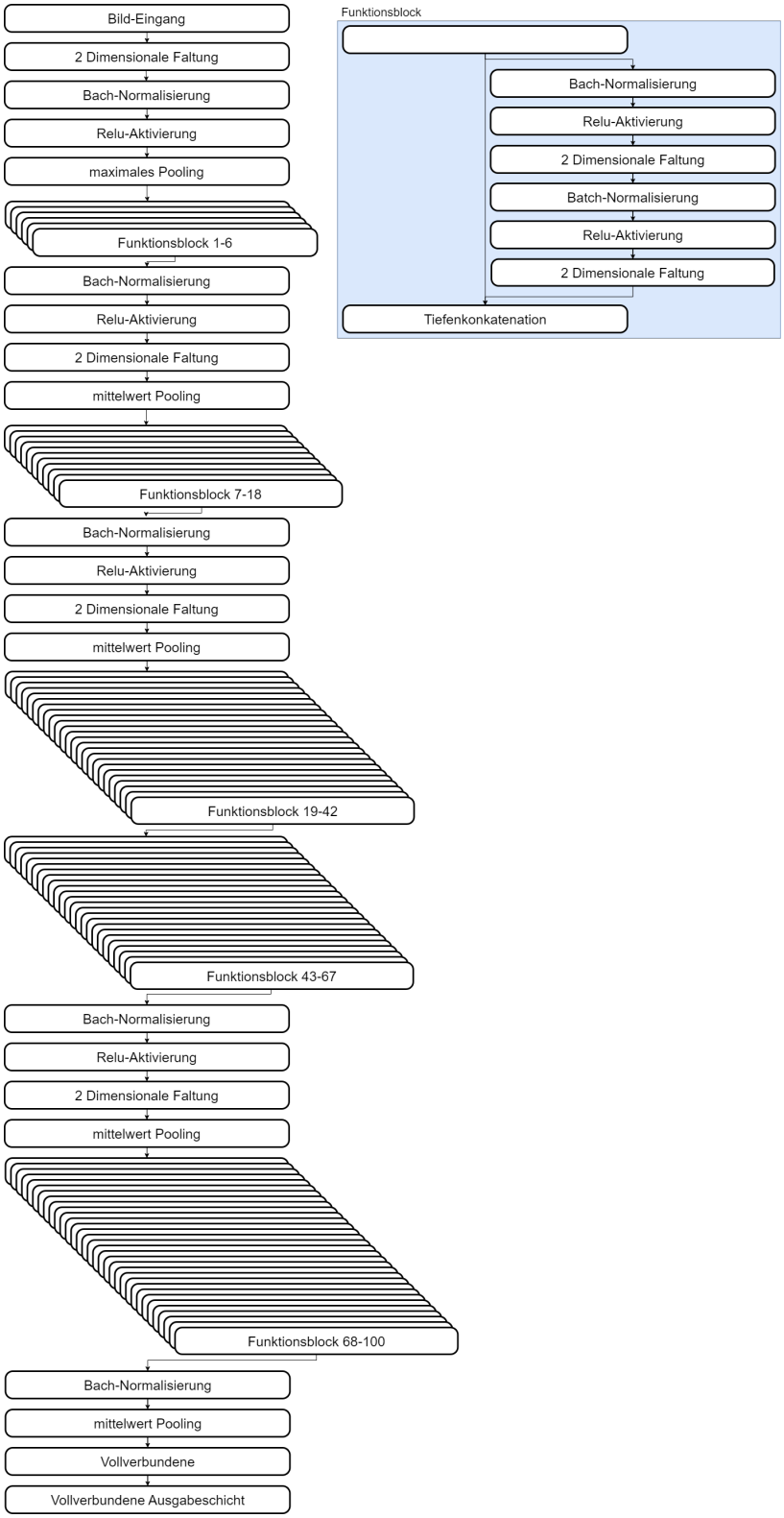
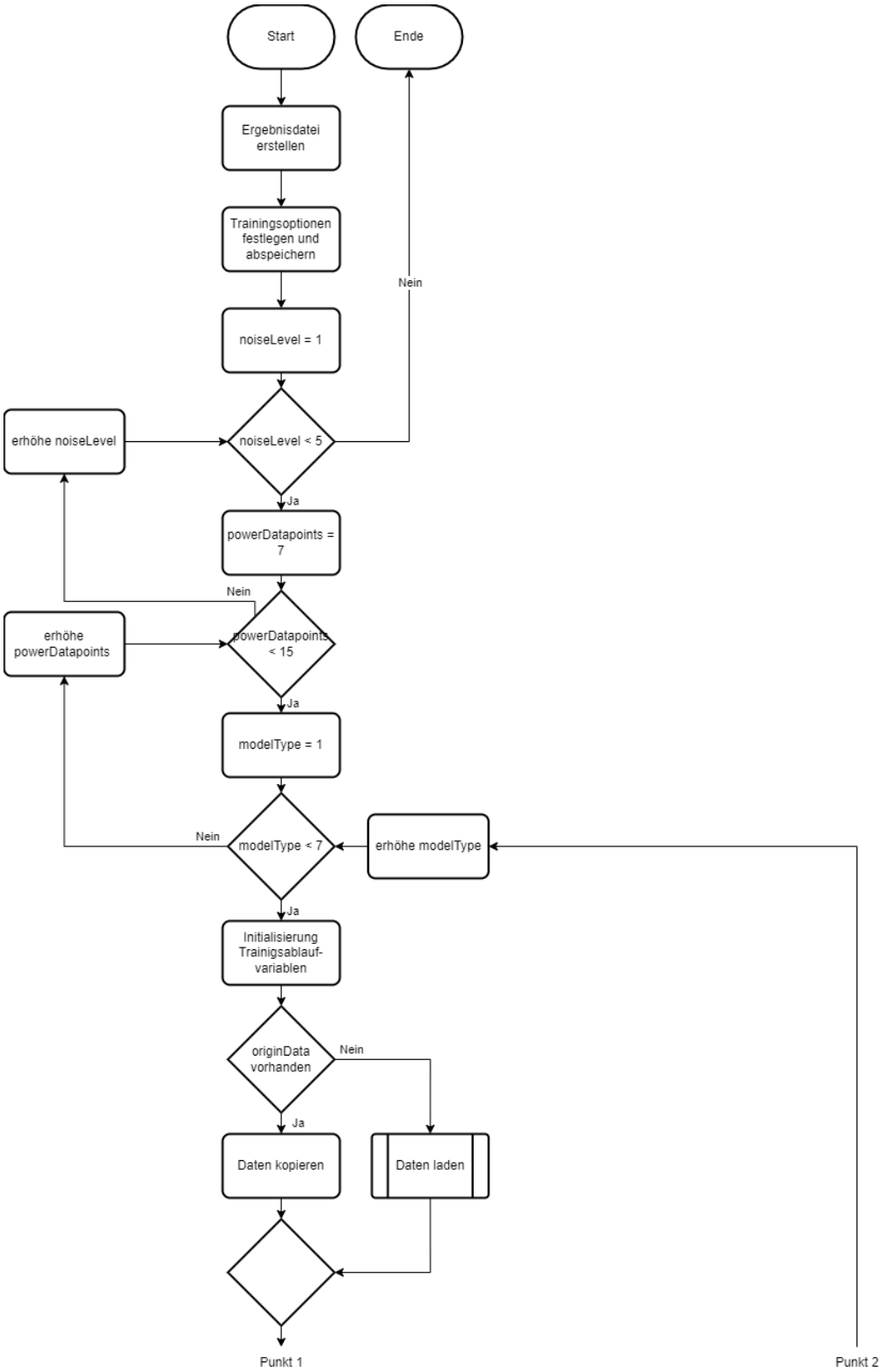


Abbildung A.5: Aufbau des DenseNet

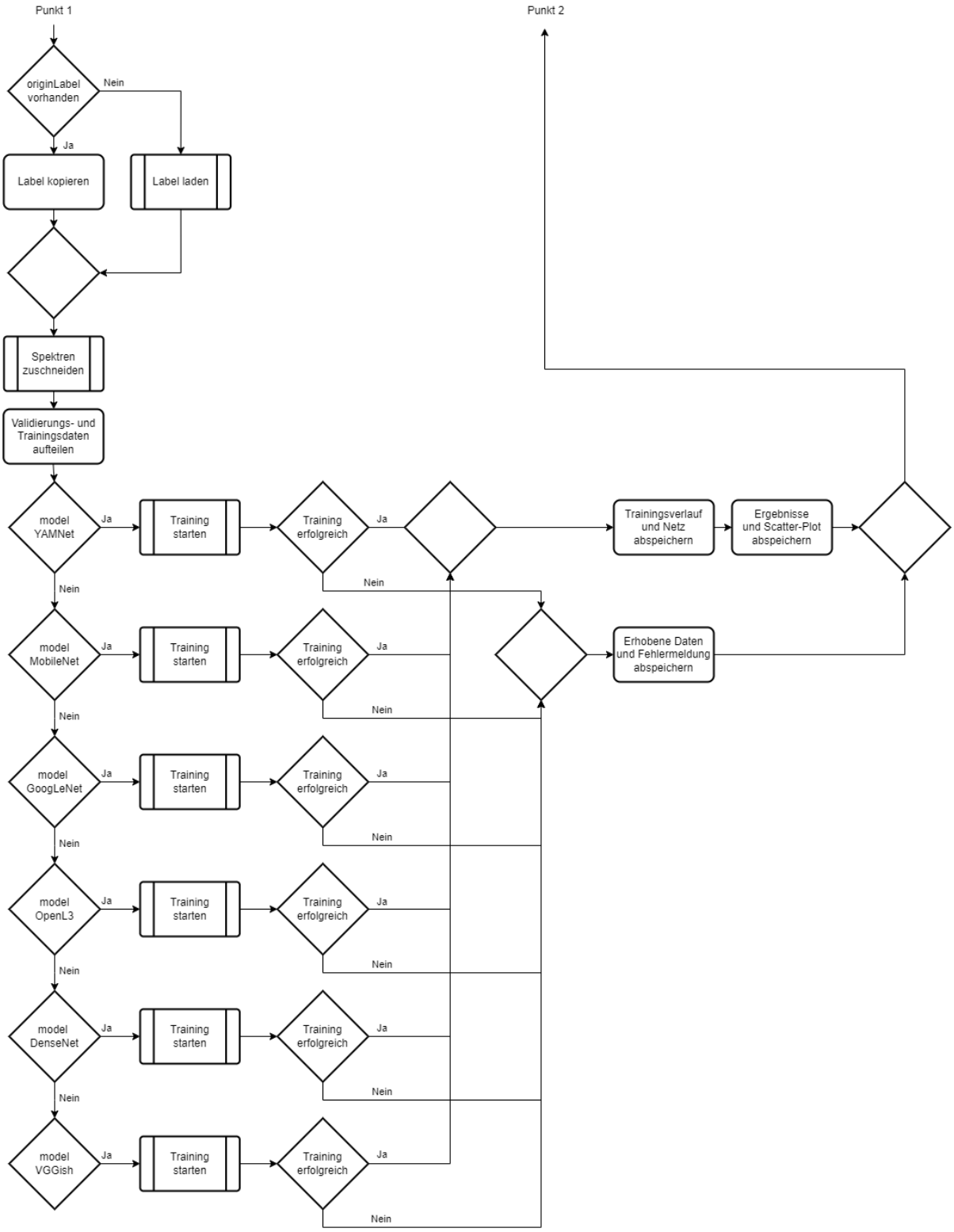
Abbildung A.6: Aufbau des VGGish

A.2 Methodik

A.2.1 Vorhersage des ZBP



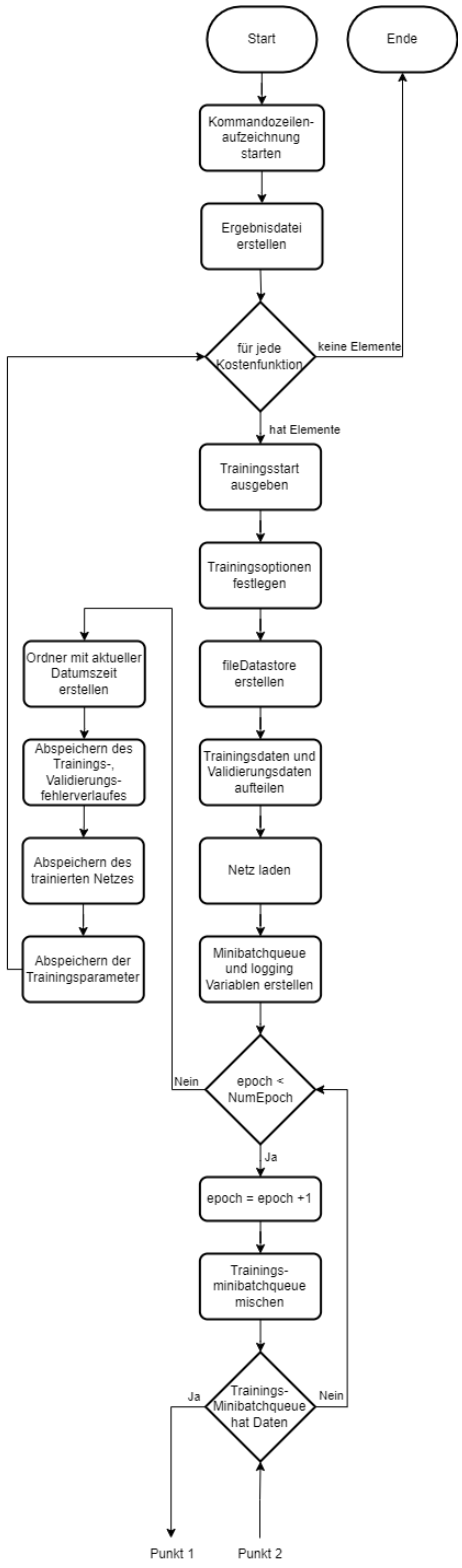
(a) Trainingsablauf Teil 1



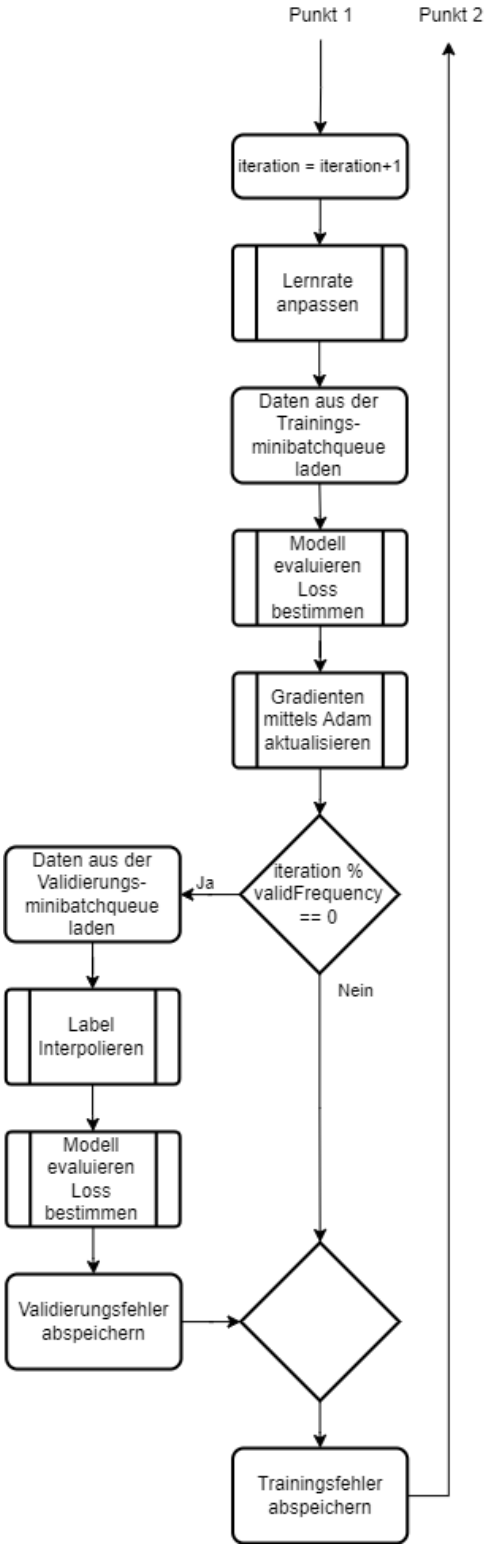
(b) Trainingsablauf Teil 2

Abbildung A.7: Darstellung des Trainingsablaufs zur Vorhersage des ZBP

A.2.2 Individueller Trainingsablauf



(a) Trainingsablauf Teil 1



(b) Trainingsablauf Teil 2

Abbildung A.8: Darstellung des individuellen Trainingsablaufs

A.2.3 Interpolation der Amplitudenverläufe

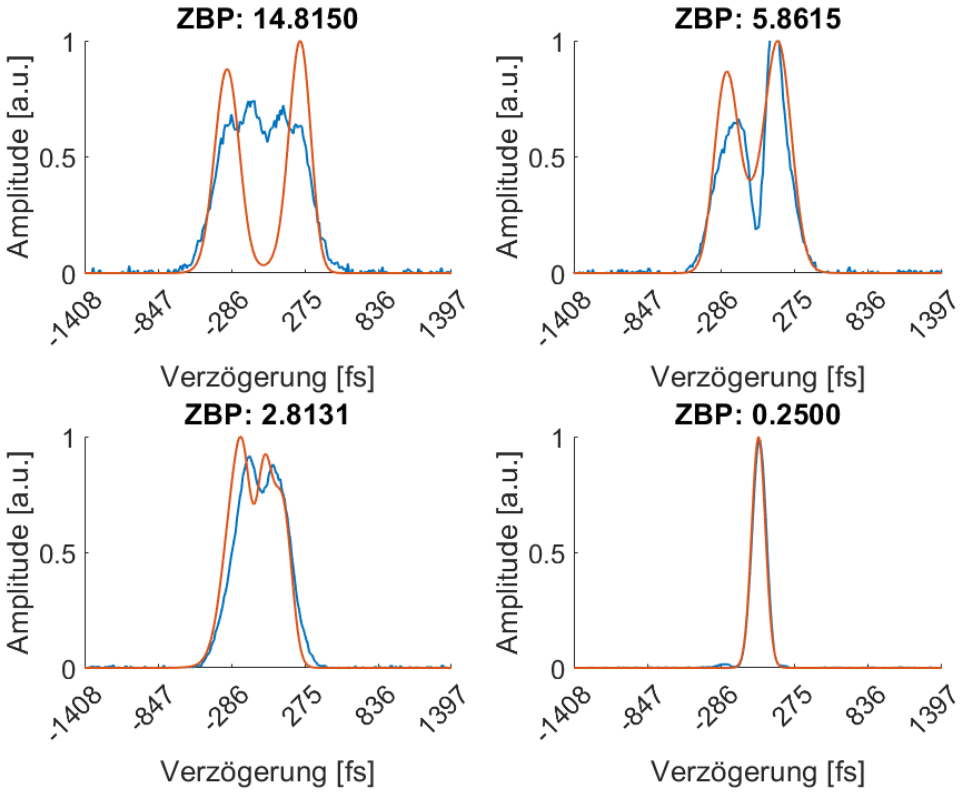


Abbildung A.9: 256 Label-Datenpunkte ohne Interpolation

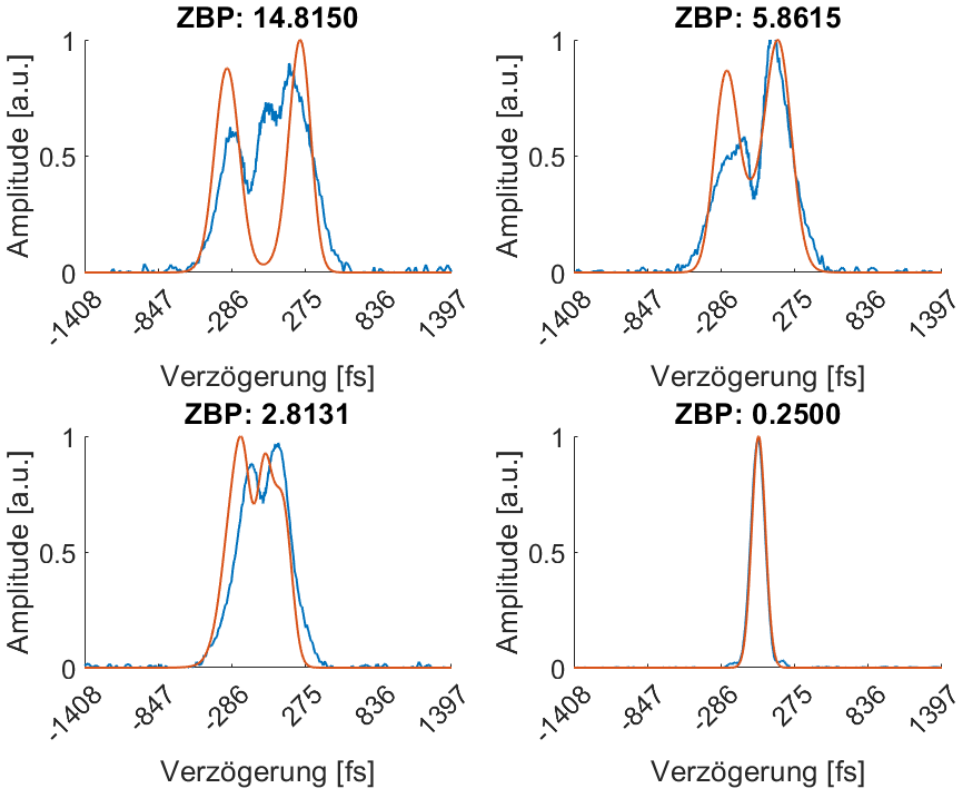


Abbildung A.10: 256 Label-Datenpunkte mit Interpolation

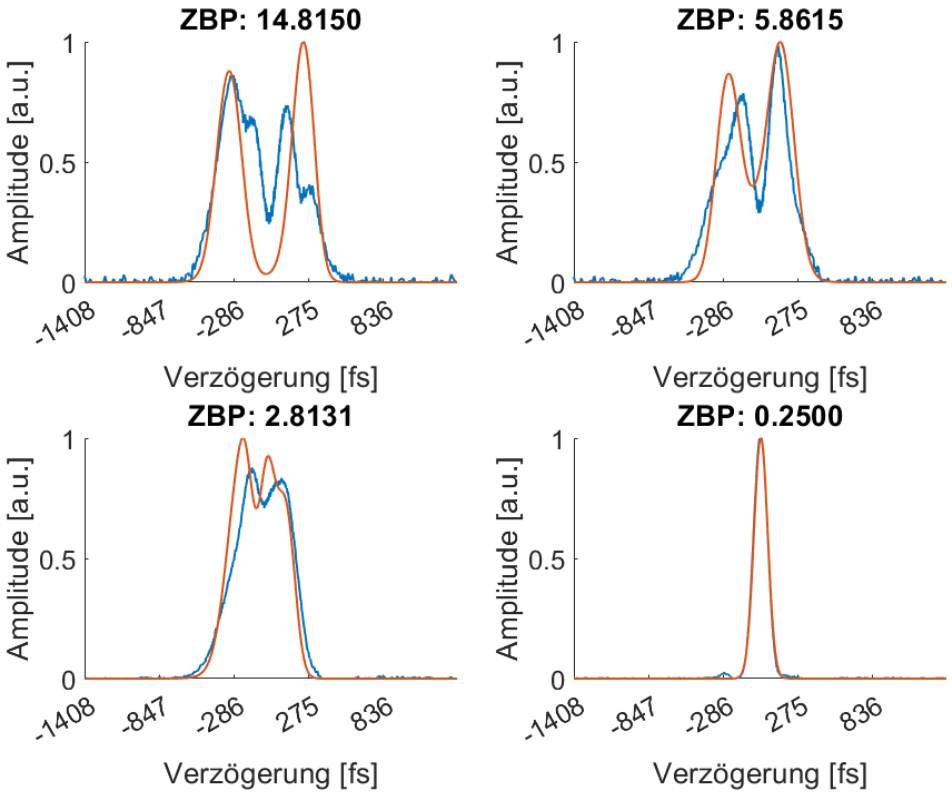


Abbildung A.11: 512 Label-Datenpunkte mit Interpolation

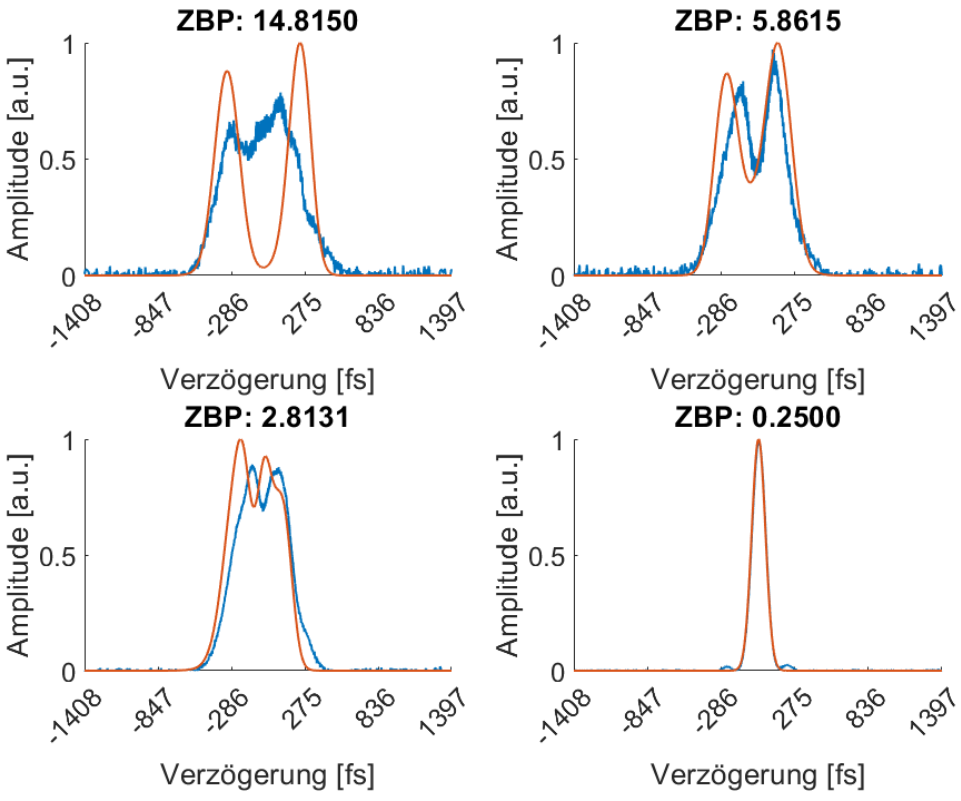


Abbildung A.12: 1024 Label-Datenpunkte mit Interpolation

A.2.4 Vorhersage der Amplitude

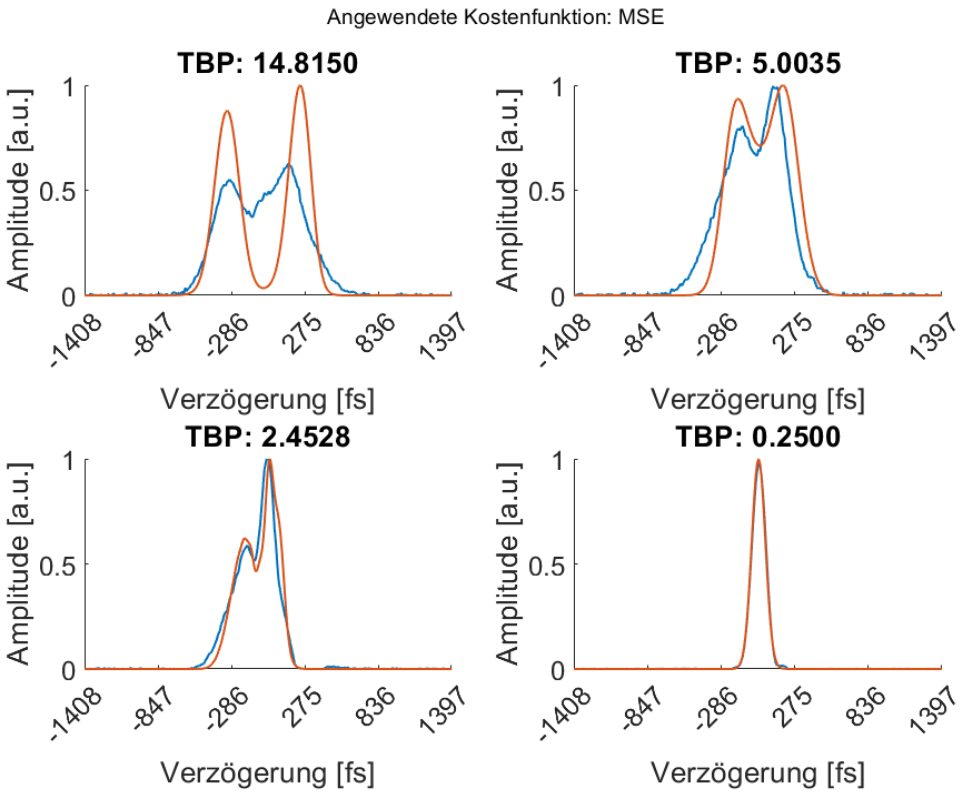


Abbildung A.13: Vorhersagen des Amplitudenverlaufs des mit MSE trainierten Netzes

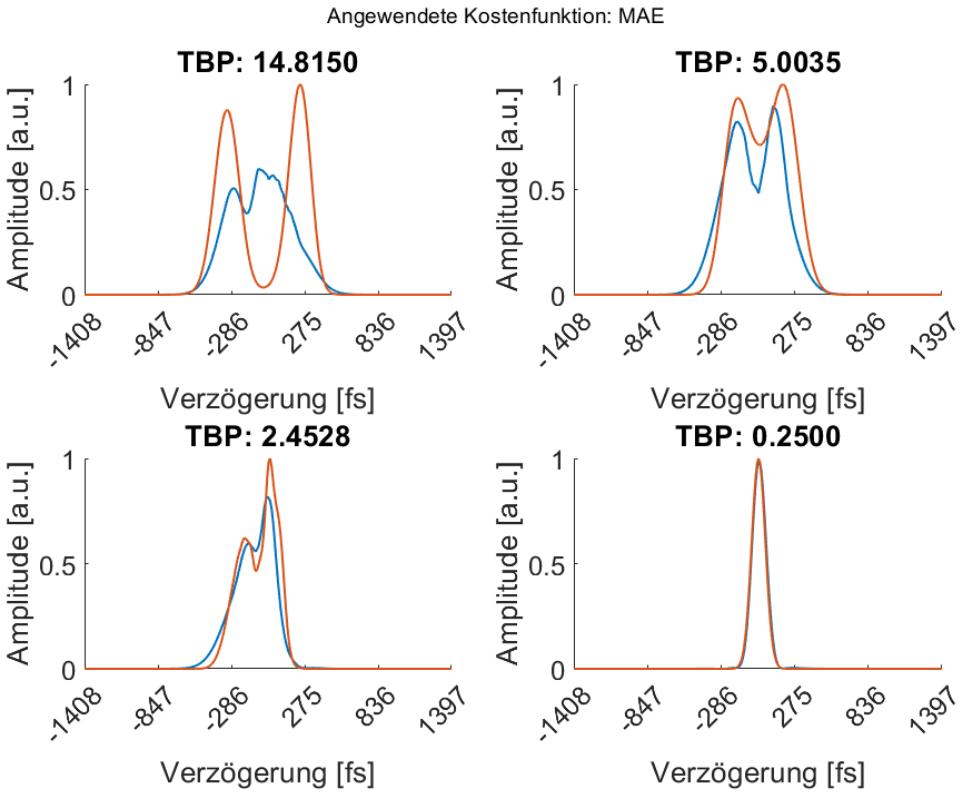


Abbildung A.14: Vorhersagen des Amplitudenverlaufs des mit MAE trainierten Netzes

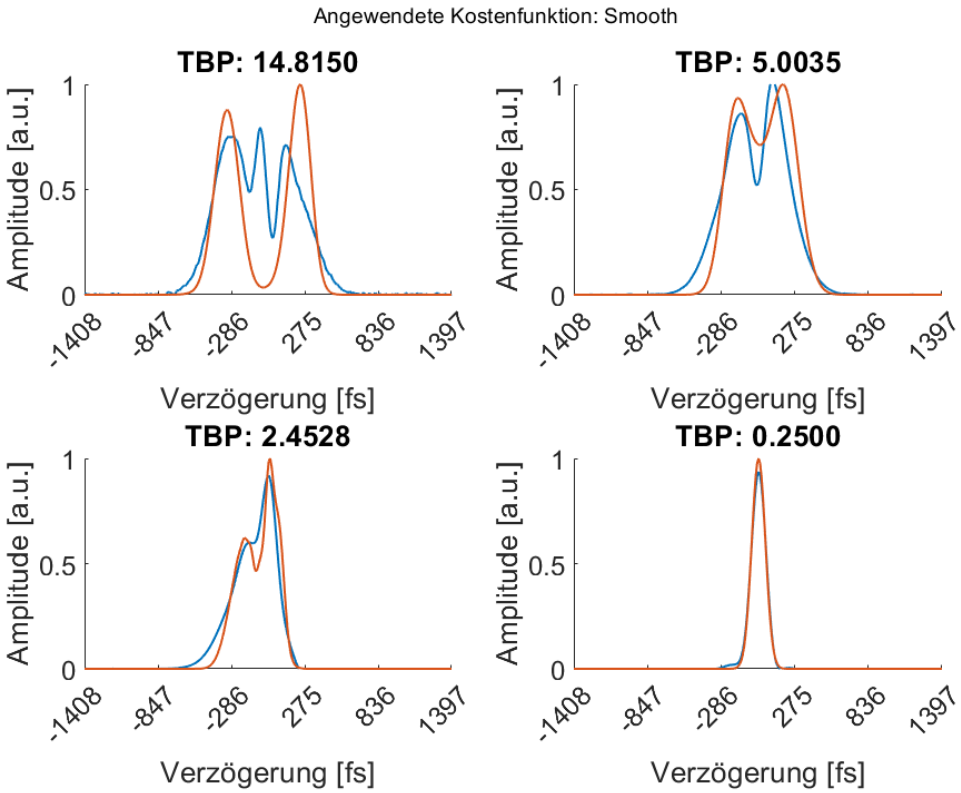


Abbildung A.15: Vorhersagen des Amplitudenverlaufs des mit Smooth trainierten Netzes

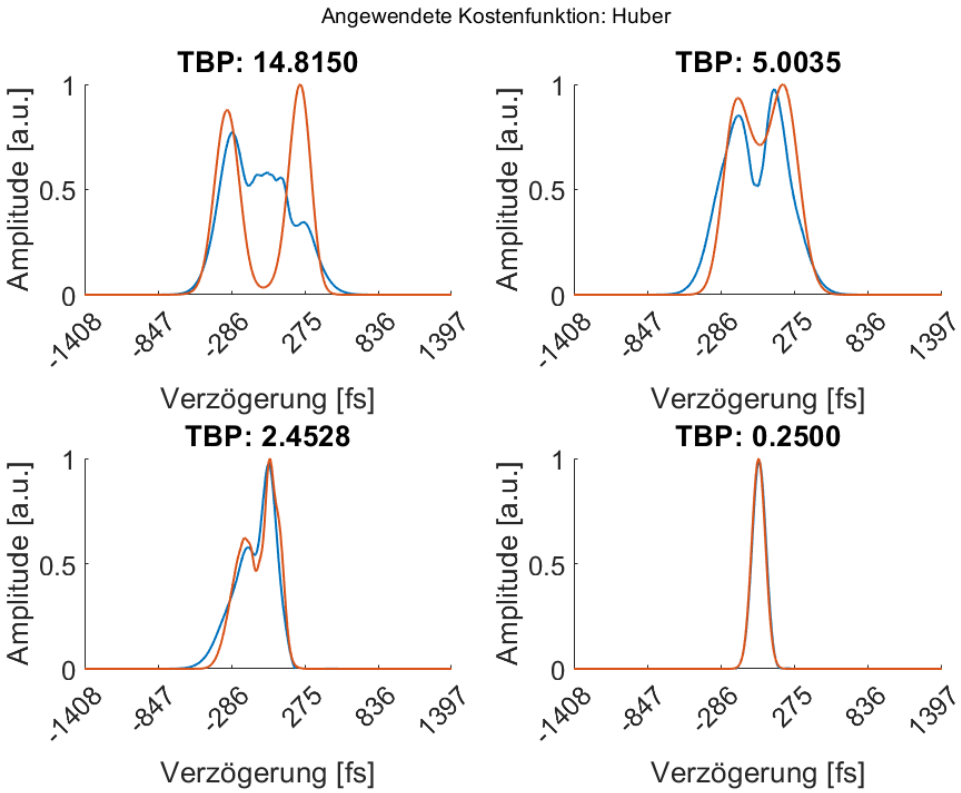


Abbildung A.16: Vorhersagen des Amplitudenverlaufs des mit Huber trainierten Netzes

A.2.5 Vorhersage der Phase

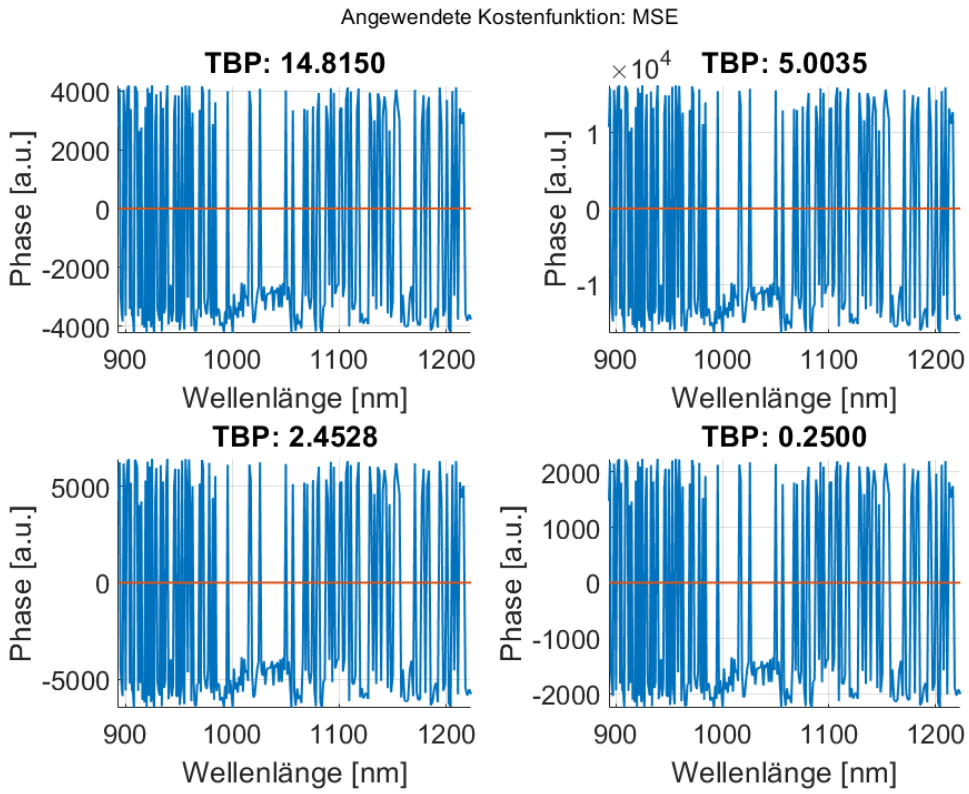


Abbildung A.17: Vorhersagen des Phasenverlaufs des mit MSE trainierten Netzes

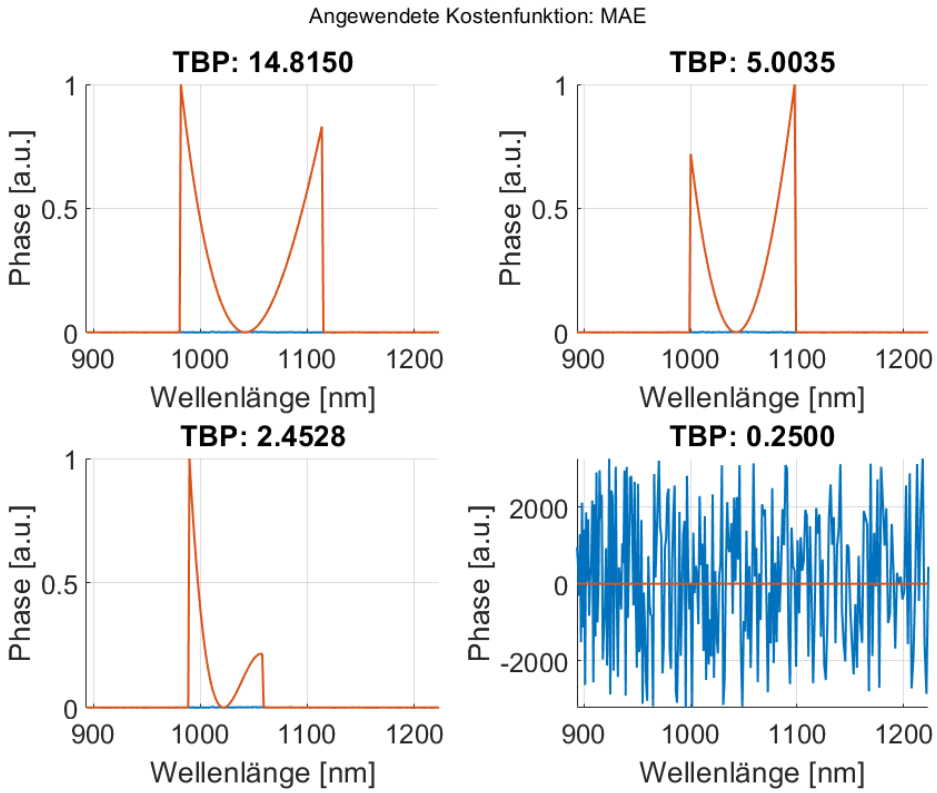


Abbildung A.18: Vorhersagen des Phasenverlaufs des mit MAE trainierten Netzes

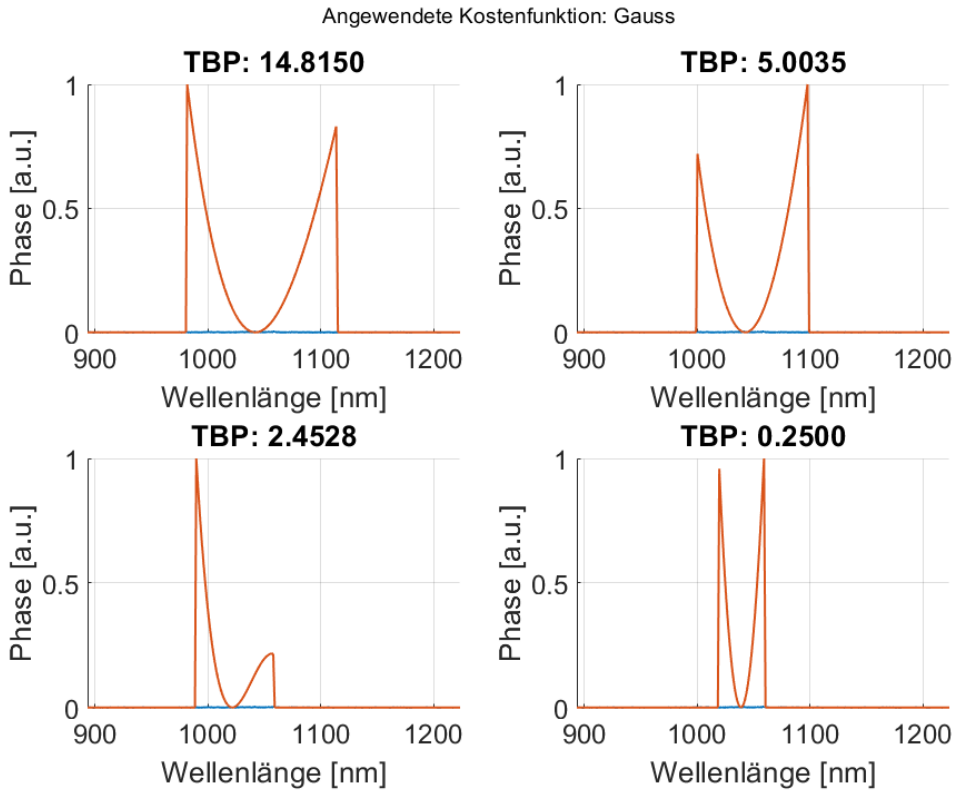


Abbildung A.19: Vorhersagen des Phasenverlaufs des mit Gauss trainierten Netzes

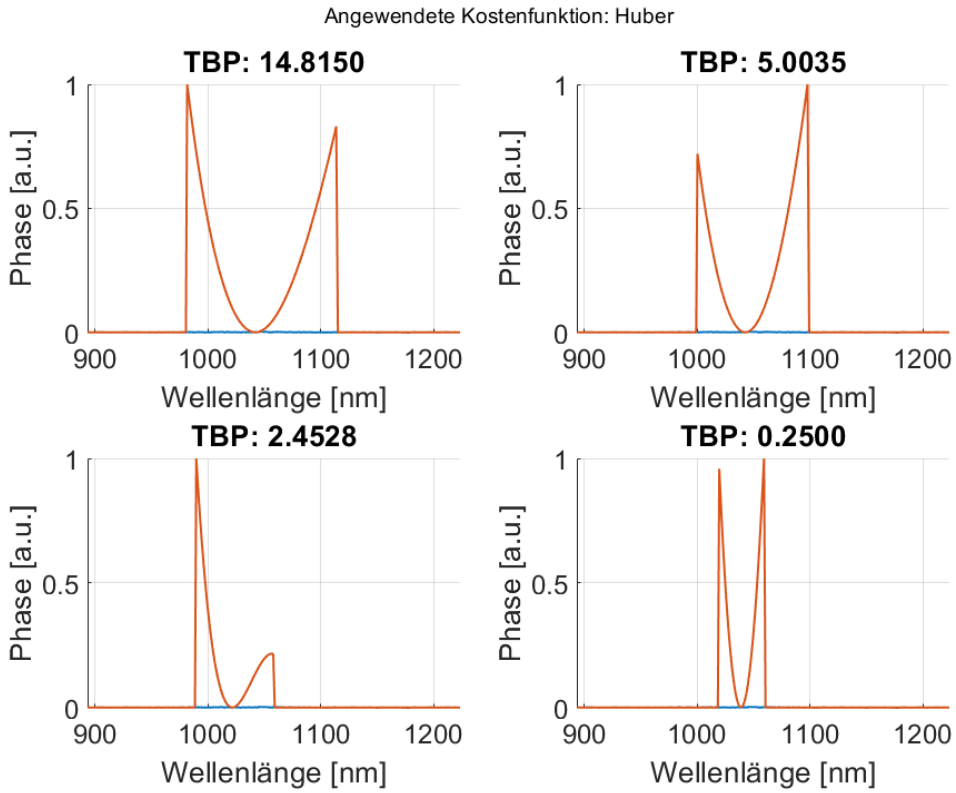


Abbildung A.20: Vorhersagen des Phasenverlaufs des mit Huber trainierten Netzes

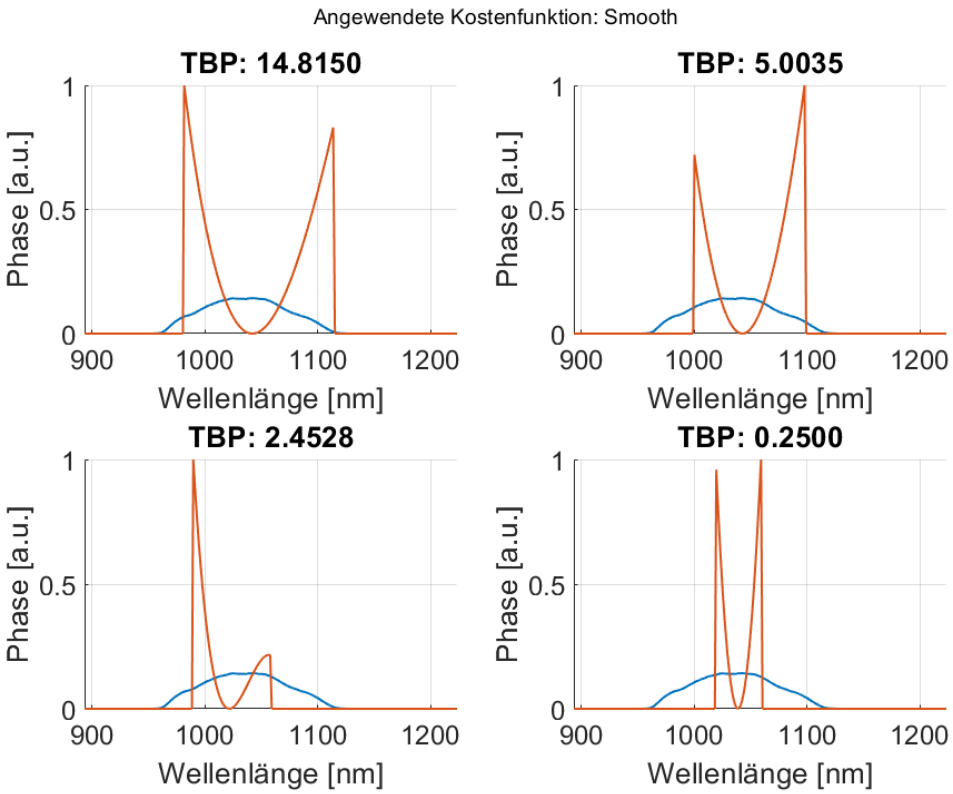


Abbildung A.21: Vorhersagen des Phasenverlaufs des mit Smooth trainierten Netzes

A.2.6 Vorhersage der Phase bei bekannter Amplitude

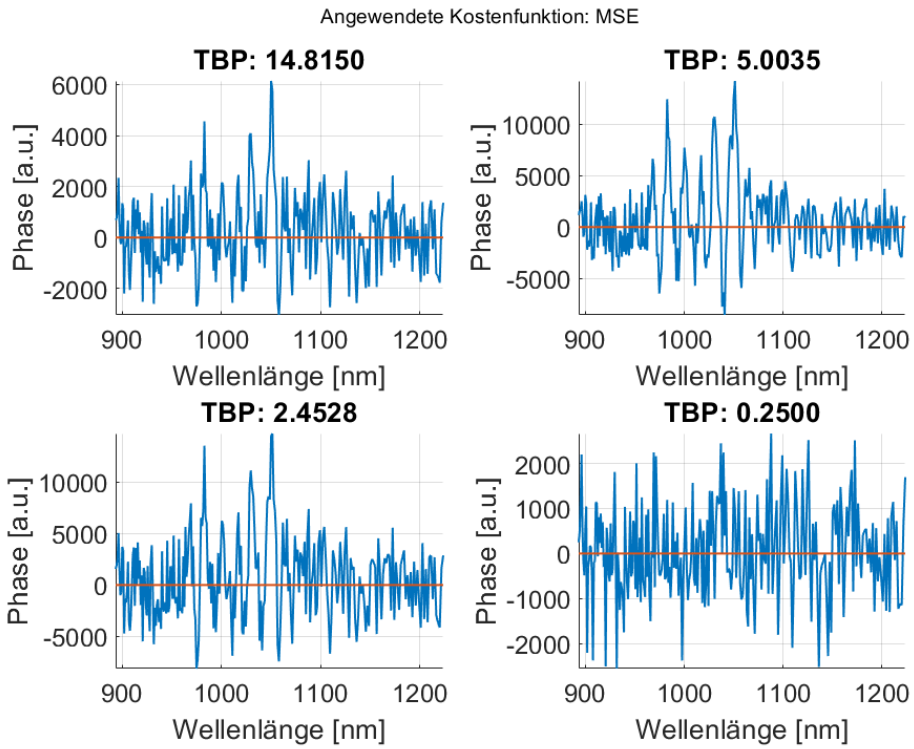


Abbildung A.22: Vorhersage der Phase bei bekannter Amplitude des mit der MSE trainierten Netzes

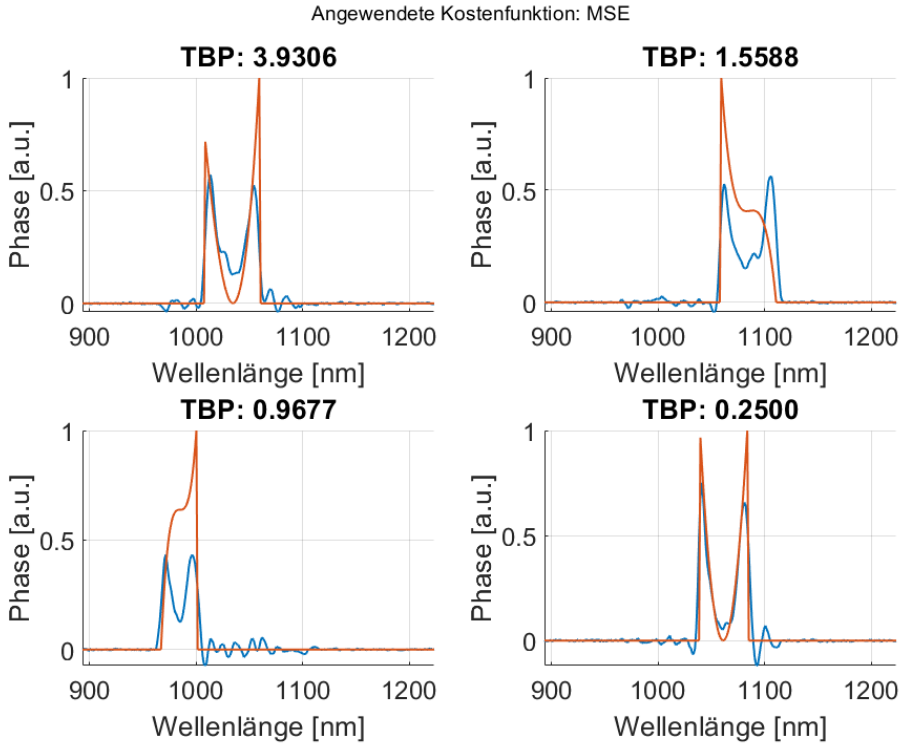


Abbildung A.23: Vorhersage der Phase bei bekannter Amplitude des mit der MSE trainierten Netzes auf dem zum training verwendeten Datensatz

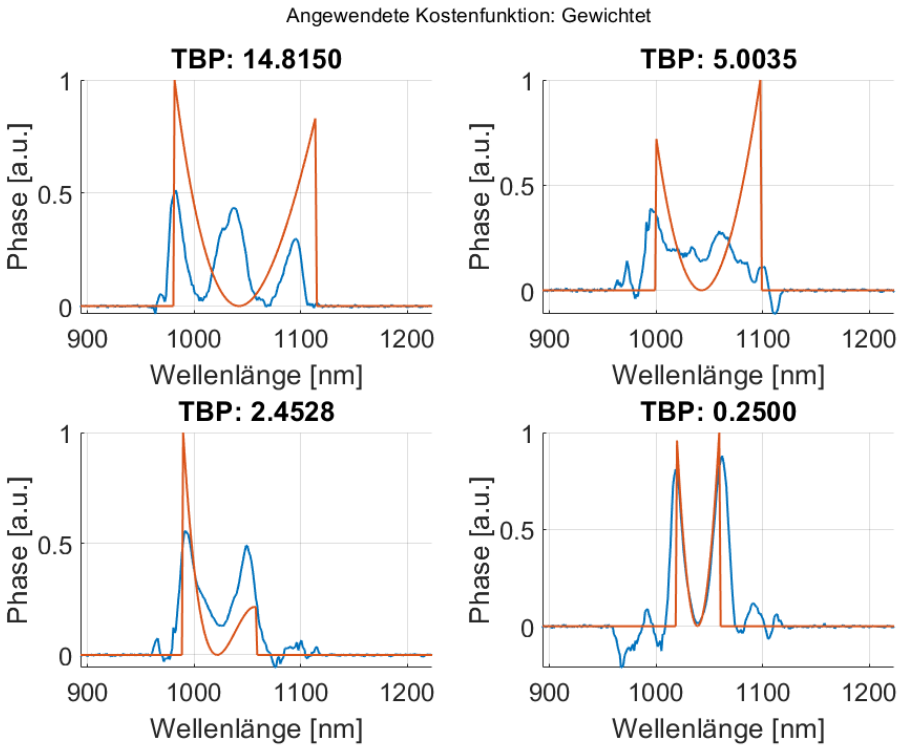


Abbildung A.24: Vorhersage der Phase bei bekannter Amplitude des mit der MSE trainierten Netzes

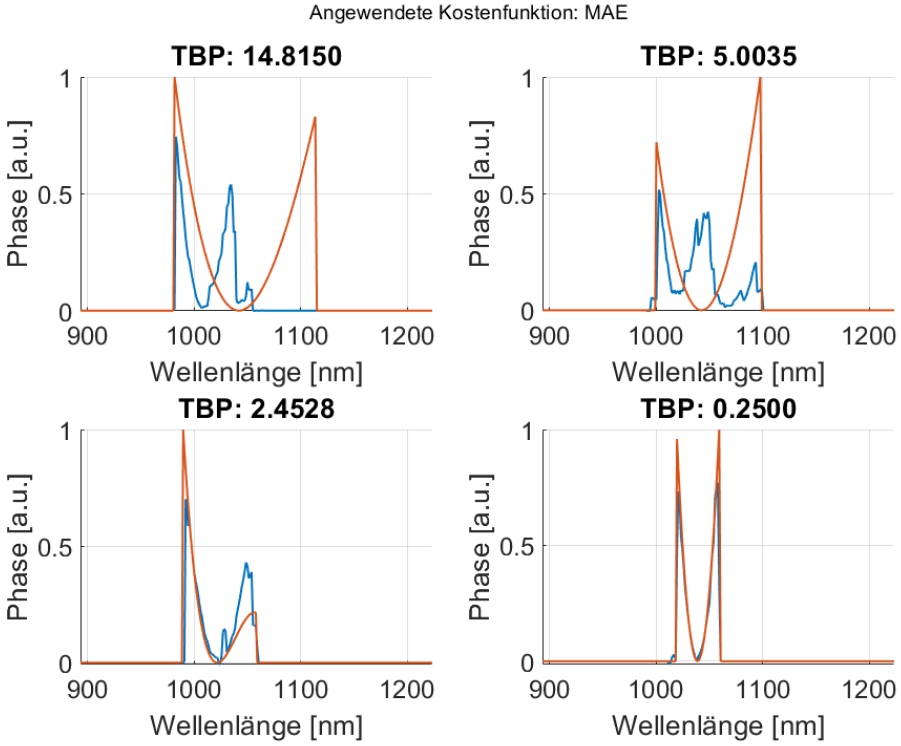


Abbildung A.25: Vorhersage der Phase bei bekannter Amplitude des mit der MSE trainierten Netzes

A.3 Verzeichnis

Auf der beiliegenden CD sind die Programmcodes sowie die in dieser Arbeit behandelten Ergebnisse in der folgenden Ordnerstruktur abgelegt:

```
CD
├── 1_Vorverarbeiten der Daten
├── 2_Trainingsabläufe
│   ├── Kostenfunktionen
│   └── Lernartenfunktionen
├── 3_Analyse der Trainingserfolge
├── 4_Ergebnisse
├── README.txt
└── Thesis_Simon_Martin_Egge.pdf
```

In dem Ordner „1_Vorverarbeiten der Daten“ enthält Funktionen und Skripte abgelegt, die Daten für das Training aufbereiten oder laden. Der zweite Ordner enthält zwei weitere Ordner, in denen jeweils die erstellten Kostenfunktionen sowie die verwendeten Lernratenfunktionen abgelegt sind. Ebenso sind in dem Ordner „2_Trainingsabläufe“ die Skripte, welche zum Trainieren der Netze verwendet wurden, abgelegt. Zu Analyse der vom Netz erzielten Ergebnisse wurden die verwendeten Skripte in dem dritten Ordner abgelegt. In dem letzten Ordner sind die erzielten Ergebnisse in Matlab-File Format abgelegt. Ebenfalls sind auf der CD ein README.txt und diese Bachelorarbeit enthalten.

A.4 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
MATLAB	Programmierumgebung und Anwendung des Codes
ChatGPT	Sprachliche Aufarbeitung des Textes
draw.io	Erstellen von Zeichnungen

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original