Duale Hochschule Baden-Württemberg Mannheim

Bachelorarbeit

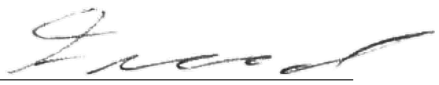
Quantisierung eines Autoencoders zur Kompression von Satellitenbildern auf einem FPGA

Studiengang Informationstechnik

Verfasser(in):	Patrick Freund
Matrikelnummer:	9820618
Firma:	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Kurs:	TINF21IT1
Firmenbetreuer:	Dipl. Ing. Karsten Westerdorff
Wissenschaftlicher Betreuer:	Dr. Alexander Dück
Bearbeitungszeitraum:	04.06.2024 - 18.09.2024

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Titel "*Quantisierung eines Autoencoders zur Kompression von Satellitenbildern auf einem FPGA*" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.



Berlin, 18.09.2024 — Patrick Freund

Ort, Datum

Zusammenfassung

Im Rahmen des ScOSA Projekts vom Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR) soll ein verteilter und leistungsfähiger On-Board-Computer entwickelt werden, der auf einem CubSat zum Einsatz kommt. Auf diesem Computer sollen neuronale Kompressionsmethoden für die Kompression von Satellitenbildern eingesetzt werden. In diesem Zusammenhang wird sich mit der Entwicklung eines Prototypen beschäftigt, bei dem ein Autoencoder auf einem Field Programmable Gate Array (FPGA) implementiert werden soll.

In dieser Bachelorarbeit wird sich mit der Anpassung des Autoencoders beschäftigt, um eine effiziente Ausführbarkeit auf der FPGA-Hardware zu ermöglichen. Eine besondere Herausforderung stellen hierbei die Fließkommaoperationen und nichtlinearen Funktionen des Kompressionsmodells dar, da diese durch die FPGA-Hardware nicht effizient berechnet werden können. Zur Bewältigung dieser Herausforderungen wurde eine vollständige Quantisierung des Encoder- und Decoder-Netzes durchgeführt. Da Quantisierungen zu einem Verlust der Rekonstruktionsleistung führen, wurden Missionsanforderungen definiert, die den maximal erlaubten Qualitätsverlust festschreiben.

Zur Realisierung der Quantisierung war eine Kombination der folgenden Ansätze erforderlich. Zum einen wurden piecewise linear approximations (dt. stückweise lineare Approximationen) (PLAs) für die nichtlinearen Funktionen erstellt und mithilfe einer Festkommaquantisierung in ganzzahlige Berechnungen umgewandelt. Zum anderen erfolgte eine Post Training Quantization (PTQ) für die Quantisierung der convolutional und deconvolutional Operationen.

Durch die Kombination der festkommaquantisierten, linearen Approximationen und der quantisierten convolutional und deconvolutional Operationen konnte eine vollständige Quantisierung des Encoder- und Decoder-Netzes des Autoencoders erfolgen, die eine effiziente Ausführbarkeit auf der FPGA-Hardware ermöglichen wird. Zusätzlich konnte ein wesentlicher Teil der Missionsanforderungen hinsichtlich des maximalen Verlusts an Rekonstruktionsqualität bereits ohne weitere Optimierung eingehalten werden. Um eine Verbesserung der Rekonstruktionsqualität herbeizuführen, wurde ein vielversprechender Optimierungsansatz vorgestellt, mit dessen Umsetzung die Einhaltung aller Missionsanforderungen ermöglicht werden könnte.

Die durchgeführte Quantisierung stellt eine gute Grundlage dar, auf die im Rahmen des ScOSA Projekts aufgebaut werden kann, um die Umsetzung eines Kompressionsmodells auf dem On-Board-Computer zu ermöglichen.

Abstract

The ScOSA project of DLR will develop a distributed high performance on-board computer for use on a CubSat. On this computer, neural compression methods are to be used for the compression of satellite images. In this context, the development of a prototype will be addressed, in which an autoencoder will be implemented on a FPGA.

This bachelor thesis deals with the adaptation of the autoencoder in order to enable efficient executability on the FPGA hardware. The floating point operations and non-linear functions of the compression model pose a particular challenge, as they cannot be efficiently computed by the FPGA hardware. To overcome these challenges, a complete quantization of the encoder and decoder network was performed. Since quantization leads to a loss of reconstruction performance, mission requirements were defined that specify the maximum allowed quality loss.

To implement the quantization, a combination of the following approaches was required. On the one hand, PLAs were created for the non-linear functions and converted into integer calculations using fixed-point quantization. On the other hand, PTQ was applied for the quantization of the convolutional and deconvolutional operations.

By combining the fixed-point quantized, linear approximations and the quantized convolutional and deconvolutional operations, a complete quantization of the encoder and decoder networks was achieved, enabling efficient execution on the FPGA hardware. Additionally, a significant portion of the mission requirements regarding the maximum loss in reconstruction quality could already be met without further optimization. To improve the reconstruction quality, a promising optimization approach was introduced, which, when implemented, could ensure compliance with all mission requirements.

The quantization performed provides a solid foundation that can be built upon within the scope of the ScOSA project to enable the implementation of a compression model on the onboard computer.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AE	Autoencoder
BPP	Bits per Pixel
CNNs	Convolutional Neural Networks
DLR	Deutschen Zentrum für Luft- und Raumfahrt e.V.
FF	FloatFunctional
FP32	Floating Point 32 Bit
FPGA	Field Programmable Gate Array
GDN	Generalized Divisive Normalization
IGDN	Inverse Generalized Divisive Normalization
INT8	Integer 8 Bit
INT16	Integer 16 Bit
INT32	Integer 32 Bit
LSB	least significant bit
MACS	Modular Aerial Camera System
MSB	most significant bit
MSE	Mean Square Error
MS-SSIM	Multiscale Structural Similarity
PLA	piecewise linear approximation (dt. stückweise lineare Approximation)
PSNR	Peak-Signal to Noise Ratio
PTQ	Post Training Quantization
QAT	Quantization Aware Training
QConfig	Quantisierungskonfiguration
QConfigs	Quantisierungskonfigurationen
QParameter	Quantisierungsparameter
RMSE	Root Mean Square Error
RNNs	Recurrent Neural Networks

RSQRT	inversen Wurzel
SSIM	Structural Similarity
SQRT	Wurzel
VAE	Variational Autoencoder

1 Einleitung

Motivation, Problemstellung, Zielsetzung

In vielen Bereichen, wie z.B. der Forstwirtschaft, der Landwirtschaft oder der Katastrophenüberwachung sind Satellitenbilder ein wichtiges Hilfsmittel zur exakten Erfassung von Oberflächen und (Wetter-) Phänomenen (vgl. [1]). Allerdings bedingen diese hochauflösenden Bilder eine hohe Datenmenge, die die begrenzten Speicher- und Übertragungskapazitäten von Raumfahrtsystemen belastet. Bildkompressionsverfahren sind daher essentiell, um die Datenmengen zu reduzieren und die begrenzten Ressourcen der Weltraumsysteme effizient zu nutzen. In den vergangenen Jahren haben zahlreiche Forschungsarbeiten gezeigt, dass neuronale Netze die traditionellen Kompressionsverfahren, wie JPEG [2] oder JPEG2000 [3] hinsichtlich der Bildkompression und -rekonstruktion deutlich übertreffen (vgl. [4]–[6]). Durch die Implementierung verlustbehafteter, neuronaler Kompressionsverfahren auf weltraumtauglicher Hardware, wie z.B. einem strahlenresistenten FPGA, kann diese beeindruckende Kompressionsperformance zur effizienten Nutzung der begrenzten Speicher- und Übertragungsressourcen eingesetzt werden. Es gibt verschiedene Projekte, die sich mit der praktischen Umsetzung und Untersuchung neuronaler Kompressionsverfahren auf realen Raumfahrtsystemen beschäftigen. Eines davon ist das ScOSA Projekt [7] des DLR, in dem ein verteilter und leistungsfähiger On-Board-Computer für einen CubSat entwickelt wird, der u.a. auch neuronale Kompressionsverfahren einsetzt.

Im Rahmen des ScOSA Projekts erfolgt die Implementierung eines neuronalen Kompressionsmodells auf einem inferenzbeschleunigenden, weltraumtauglichen FPGA. Um eine effiziente Ausführung solcher neuronaler Kompressionsmodelle zu ermöglichen, müssen mehrere Herausforderungen bewältigt werden. Zum einen werden Kompressionsmodelle typischerweise auf Floating Point 32 Bit (FP32) trainiert, während FPGA-Hardware in der Regel über keine Hardwareelemente zur effizienten Verarbeitung von Fließkommazahlen verfügt. Stattdessen sind sie auf Berechnungen mit Festkomma- und Ganzzahlen optimiert. Zum anderen sind die nichtlinearen Funktionen in den neuronalen Netzen der Kompressionsmodelle problematisch, da deren Berechnung iterativ erfolgen muss. In den parallelisierten Pipelinestrukturen, die in der FPGA-Hardware zur Inferenzbeschleunigung verwendet werden, führt dies zu ungewissen Verzögerungen in den Ausführungszeiten und zu einem verminderten

Datendurchsatz. Die nichtlinearen Funktionen haben folglich einen negativen Einfluss auf den Datendurchsatz und die Inferenzgeschwindigkeit. (vgl. [8])

Für einen ersten Prototypen soll im Rahmen des Projekts der faktorisierte Variational Autoencoder (VAE) aus [9] auf dem FPGA Xilinx Zynq 7020 implementiert werden. Das Ziel dieser Arbeit ist es, eine effiziente Ausführbarkeit dieses Autoencoders auf der Hardware zu ermöglichen. Eine potenzielle Lösung der diesbezüglichen Herausforderungen ist die vollständige Quantisierung des Autoencoders bzw. seines rechenintensiven Encoder- und Decoder-Netzes. Zur Erstellung einer vollständigen Quantisierung, die beide Herausforderungen bewältigen kann, ist die Kombination mehrerer Ansätze erforderlich. Zum einen muss eine lineare Approximation und Quantisierung der nichtlinearen Funktionen des Encoders und Decoders erfolgen. Zum anderen ist eine Quantisierung sämtlicher convolutional und deconvolutional Operationen in den Netzschichten erforderlich. Zur Umsetzung der linearen Approximation und der Quantisierungen werden in dieser Arbeit geeignete Methodiken vorgestellt. Durch die Kombination dieser Ansätze kann eine vollständige Quantisierung realisiert werden, die die genannten Herausforderungen löst und somit eine effiziente Ausführbarkeit des Autoencoders ermöglicht. Da Approximationen und Quantisierungen zu Verlusten bei der Bildkompression und -rekonstruktion führen, wurden Missionsanforderungen hinsichtlich des maximalen Qualitätsverlustes im Vergleich zum Originalmodell definiert. Diese liegen bei 8 dB und 6 % in den Metriken Peak-Signal to Noise Ratio (PSNR) und Multiscale Structural Similarity (MS-SSIM).

Vorgehensweise

Um das Ziel dieser Arbeit systematisch zu erarbeiten und strukturiert darzustellen, wurde folgende Vorgehensweise gewählt. Zunächst werden in Kapitel 2 die für diese Arbeit relevanten theoretischen Grundlagen geschaffen. Daraufhin folgt eine detaillierte Beschreibung des Encoder- und Decoder-Netzes des verwendeten neuronalen Kompressionsmodells sowie der verwendeten Datensätze. Den Hauptteil bilden Kapitel 4 bis Kapitel 6. In Kapitel 4 werden zunächst die Methodiken zur Approximation und Quantisierung des Encoder- und Decoder-Netzes beschrieben. Diese werden in Kapitel 5 angewendet und deren Kompressions- und Rekonstruktionsergebnisse anhand verschiedener Metriken evaluiert. Anschließend wird in Kapitel 6 ein Optimierungsansatz vorgestellt, mit dem eine Verbesserung der Quantisierung erreicht werden könnte. Abschließend werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf mögliche weitere Forschungsarbeiten gegeben.

2 Theoretischer Hintergrund

2.1 Bildkompression mit neuronalen Netzen

Die Kompression ist ein zentraler Prozess in der digitalen Welt. Das Ziel der Kompression ist die Reduzierung der Datenmenge, die zur Darstellung von Informationen verwendet wird. Infolgedessen zielt die Bildkompression darauf ab, die Datenmenge zu reduzieren, die zur Kodierung eines Bildes benötigt wird. In den meisten Fällen werden für die Darstellung der Informationen eines Bildes jedoch mehr Daten verwendet, als tatsächlich notwendig sind. Diese überflüssigen Daten werden in verschiedene Redundanzkategorien eingeteilt, wie z.B. die Kodierungs-, die Interpixel- oder die psychovisuellen Redundanzen. Die Reduzierung dieser Redundanzen wird als Kompression bezeichnet. Wenn ein Kompressionsverfahren ausschließlich die redundanten Daten entfernt, die vollständig rekonstruiert werden können, ist das Verfahren verlustfrei. Werden hingegen zusätzlich weniger essenzielle Informationen entfernt, die nicht vollständig wiederhergestellt werden können, handelt es sich um ein verlustbehaftetes Kompressionsverfahren. Damit lassen sich stärkere Reduktionen der Datenmengen erreichen, es treten dafür aber Fehler (Distortions) bei der Bildrekonstruktion auf. (vgl. [10, S.309-314, 343, 362, 374–375])
In den vergangenen Jahren konnte in verschiedensten Forschungsarbeiten gezeigt werden, dass neuronale Netzwerke im Bereich der verlustbehafteten Bildkompression bessere Kompressions- und Rekonstruktionsergebnisse erzielen können als die etablierten Standardcodecs. Architekturen, die für diese Aufgabe verwendet werden, sind u.a. die Convolutional Neural Networks (CNNs), die Autoencoder (AE), die Variational Autoencoder (VAE), die Recurrent Neural Networks (RNNs) und einige weitere. (vgl. [4]–[6])
Das Ziel dieses Grundlagenabschnitts ist die theoretische Erläuterung der Bestandteile und Schichten des Encoder- und Decoder-Netzes des betrachteten Autoencoders. Auf die zusammenhängenden Architekturen der beiden Netze wird in Abschnitt 3.1 eingegangen. Zunächst werden die convolutional und deconvolutional Operationen im Bereich der Bildverarbeitung vorgestellt, die im weiteren Verlauf der Arbeit auch als conv- und deconv-Operationen bezeichnet werden. Anschließend wird die Generalized Divisive Normalization (GDN) vorgestellt, die als Aktivierungsfunktion im betrachteten Autoencoder verwendet wird. Zuletzt erfolgt im Rahmen dieses Abschnitts eine Darstellung von Metriken,

die zur Bewertung der Bildkompression und -rekonstruktion des Autoencoders eingesetzt werden.

2.1.1 Convolutional Operationen

Bei einer convolutional Operation wird auf eine Eingangsmatrix mit den Dimensionen $H \times W \times C$ (Höhe, Breite, Kanäle) ein Filter mit den Dimensionen $f_H \times f_W \times C$ angewendet. Dazu wird der Filter schrittweise über die Eingangsmatrix bewegt, wobei in jedem Schritt das elementweise Produkt zwischen dem Filter und der Matrix berechnet und kumuliert wird. Die Schrittweite des Filters wird als Stride bezeichnet. Je höher der gewählte Stride ist, desto kleiner wird die Ausgangsmatrix, auch Feature-Map genannt. Um der Verkleinerung entgegenzuwirken und gleichzeitig die Randinformationen der Eingangsmatrix zu berücksichtigen, kann ein (Zero)-Padding mit P Pixeln durchgeführt werden. Dies bedeutet, dass die Eingangsmatrix mit P Nullen umrandet wird. In Abb. 2.1 sind die ersten zwei Schritte einer convolutional Operation mit einer $3 \times 3 \times 1$ Eingangsmatrix, einem $3 \times 3 \times 1$ Filter, einem Stride von 2 und einem Padding von 1 dargestellt. In der Abbildung stellt das mittig liegende 3×3 Feld die Eingangsmatrix dar, die mit 0en umrandet ist. Der Filter wird durch den dunkelblauen 3×3 Bereich angedeutet. Die Ausgangsmatrix bzw. Feature-Map ist in gelb dargestellt. Das orange Segment der Feature-Map ist das Element, das durch Anwendung des Filters auf die Elemente der gepaddeten Eingangsmatrix berechnet wird. Es ist zu erkennen, dass die Anwendung der convolutional Operation zu einer Verkleinerung der Feature-Map führt. (vgl. [11])

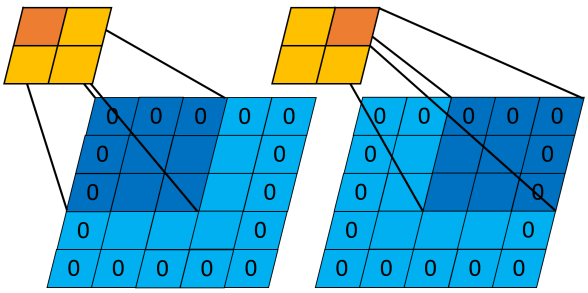


Abbildung 2.1: Darstellung der ersten beiden Schritte einer convolutional Operation mit Padding 1 und Stride 2 (vgl. [11])

Convolutional Operationen werden in den meisten Fällen in eigenständigen Netzschichten implementiert. Diese werden im Folgenden als conv2D-Schichten bezeichnet. Der Grund für die Verwendung von convolutional Operationen in neuronalen Netzen besteht darin,

dass sie die Bilddaten und -informationen reduzieren, indem sie relevante Merkmale, wie z.B. Kanten, Ecken oder Muster, extrahieren. Durch mehrere aufeinanderfolgende convolutional Operationen und nichtlineare Aktivierungsfunktionen kann mit zunehmender Tiefe des Netzes eine immer stärkere Datenreduktion erreicht werden, während gleichzeitig immer komplexere Merkmale extrahiert werden können. Durch das Trainieren der Gewichte der einzelnen Filter lernt das Netz genau die Merkmale aus den Bildern zu entnehmen, die zu einer bestmöglichen Lösung der vorliegenden Aufgabe führen. (vgl. [11])

2.1.2 Deconvolutional Operationen

Bei einer deconvolutional Operation wird eine Feature-Map als Eingangsmatrix verwendet, die mithilfe eines Filters in eine größere Ausgangsmatrix projiziert wird. Hierzu werden um und zwischen den Elementen der Eingangsmatrix Nullen eingefügt. Die Anzahl und Position der Nullen sind dabei abhängig vom gewählten Stride und Padding, wie in [11] aufgezeigt wird. Anschließend werden die Werte der Ausgangsmatrix äquivalent zur convolutional Operation berechnet. In Abb. 2.2 sind die ersten beiden Schritte einer deconvolutional Operation mit einer $3\times3\times1$ Eingangsmatrix, einem $3\times3\times1$ Filter, ohne Stride und Padding dargestellt. Auch in dieser Darstellung wird die Eingangsmatrix durch den mittig liegenden, blauen Bereich illustriert, während der Filter durch das dunklere 3×3 Segment angedeutet wird. Die Nullen um die Eingangsmatrix dienen nicht als Padding, sondern sind notwendig für die Durchführung der deconvolutional Operation. Die Ausgangsmatrix ist in gelb dargestellt. Das orangefarbene Segment stellt das Element der Ausgabematrix dar, das sich aus der Anwendung des Filters auf die Elemente der Eingabematrix an der dargestellten Stelle ergibt. Hierbei ist deutlich zu erkennen, dass die Anwendung der Operation zu einer größeren $5\times5\times1$ Ausgangsmatrix führt. (vgl. [11], [12])

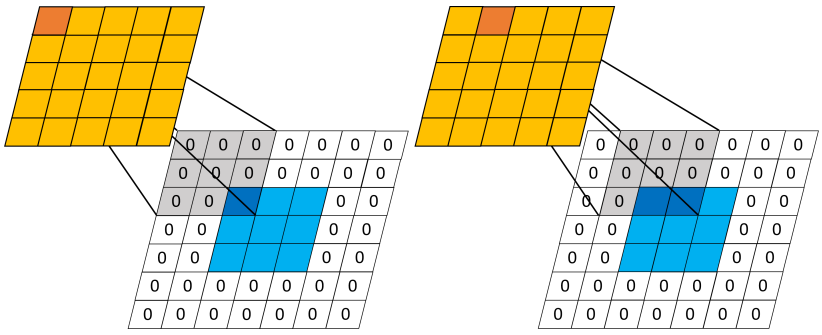


Abbildung 2.2: Darstellung der ersten beiden Schritte einer deconvolutional Operation ohne Padding und Stride

Während convolutional Operationen die Bilddaten und -informationen reduzieren, zielen deconvolutional Operationen darauf ab, die ursprüngliche Struktur eines Bildes wiederherzustellen bzw. eine intelligente Vergrößerung (Upsampling) der Eingabe zu erzeugen. Aus diesem Grund werden sie häufig in neuronalen Netzen zur Vergrößerung oder Rekonstruktion von Bildern eingesetzt. Ein typisches Anwendungsgebiet ist das Decoder-Netz eines Autoencoders, in dem aus einer komprimierten, latenten Darstellung das ursprüngliche Bild bestmöglich rekonstruiert werden muss. Die Netzschichten für die deconvolutional Operationen werden im Folgenden als convTranspose2D-Schichten bezeichnet. (vgl. [11], [12])

2.1.3 Generalised Divisive Normalisation

Allgemeine Beschreibung

Bei der GDN von Ballé et al. [13] handelt es sich um eine invertierbare, parametrische und nichtlineare Transformation, die für die lokale Normalisierung und Gaussianisierung von Daten natürlicher Bilder konzipiert wurde. Sie ist eine Weiterentwicklung der Divisive Normalization [14] und wird in neuronalen Netzen typischerweise auf die Ergebnisse von linearen Filteroperationen $z = \boldsymbol{H}\boldsymbol{x}$ (z.B. convolutional Operationen) angewendet. Dabei ist $\boldsymbol{x}$ eine Eingabematrix und $\boldsymbol{H}$ die Transformationsmatrix. Die Autoren aus [13] definieren die nichtlineare und parametrische GDN-Transformation, wie in Gleichung (2.1) beschrieben. (vgl. [13], [14], [15])

$$\boldsymbol{y} = g(\boldsymbol{x}; \boldsymbol{\theta}) \qquad \text{s.t.} \qquad y_i = \frac{z_i}{\left(\beta_i + \sum_j \gamma_{ij} |z_j|^{\alpha_{ij}}\right)^{\varepsilon_i}} \tag{2.1}$$

Dabei ist $\boldsymbol{y}$ die normalisierte und gaussianisierte Ergebnismatrix der GDN-Transformation. Die Vektoren β und ε, sowie die Matrizen α und γ sind die Parameter der Transformation, die während des Trainings optimiert werden. Darüber hinaus spezifizieren die Indizes i und j die lokalen Patches in der Eingangsmatrix, die während der GDN-Transformation einzeln normalisiert werden. Dieses Prinzip der lokalen Normalisierung entstammt der Divisive Normalization. (vgl. [13], [14], [15])

In Softwarebibliotheken, wie CompressAI, wird bei der Implementierung der GDN-Transformation für neuronale Bildkompressionsmodelle eine gewichtete L2-Norm verwendet. Dafür werden die Parameter α und ε auf die Werte 2 und $\frac{1}{2}$ gesetzt. Der Grund dafür ist, dass die Divisive Normalization und damit auch die GDN-Transformation mit einer gewichteten L2-Norm die beste Gaussianisierung erreicht, was einen positiven Einfluss auf die Kompression des Modells hat (vgl. [16]). Die angepasste Version der Gleichung (2.1) ist in Gleichung (2.2) dargestellt. (vgl. [13], [15], [17]).

$$\boldsymbol{y}_n = g(\boldsymbol{x}_n; \boldsymbol{\theta}) \qquad \text{s.t.} \qquad y_n = \frac{x_n}{\sqrt{\beta_n + \gamma_n (x_n \cdot)^2}} \tag{2.2}$$

Dabei ist x_n die n-te Eingangsmatrix in die nichtlineare GDN-Transformation und y_n die n-te normalisierte und gaussianisierte Ergebnismatrix. Zusätzlich ist die Spezifikation der Parameter i und j nicht mehr notwendig, da für den Term unter der Wurzel eine convolutional Operation mit den Gewichten γ und den Biases β durchgeführt wird. Bei dieser convolutional Operation wird die Aufteilung in Patches und die lokale Normalisierung innerhalb dieser Patches implizit durchgeführt (vgl. [13], [15], [17]).

Nutzen von GDN in Kompressionsmodellen

In Kompressionsmodellen erfolgt die Implementierung der GDN-Transformationen in Form von GDN-Schichten. Diese GDN-Schichten kommen bei der Kompression von Daten zum Einsatz. Die Parameter für die Transformationen aller GDN-Schichten werden während des Trainings des Kompressionsmodells gelernt. Im Modell fungieren die GDN-Schichten als nichtlineare Aktivierungsfunktionen für die Feature-Maps der convolutional Schichten. In Forschungsarbeiten wie [15] konnte gezeigt werden, dass die Verwendung von GDN-Schichten im Vergleich zu häufig verwendeten Aktivierungsfunktionen wie ReLU, leaky ReLU oder Tanh zu deutlich besseren Kompressionsraten und Rekonstruktionsqualitäten führt.
Dies lässt sich im Wesentlichen auf zwei zentrale Aspekte zurückführen. Zum einen reduzieren die GDNs die Korrelationen bzw. Interpixel-Redundanzen zwischen den einzelnen Werten der Feature-Maps. Durch diese Dekorrelation der Daten wird eine effizientere Kompression ermöglicht, da redundante Informationen entfernt werden. Zum anderen führen

die GDNs dazu, dass die Verteilung der Daten näher an eine Gaußverteilung herangeführt wird. Dieser Prozess wird als Gaussianisierung bezeichnet. Gaußverteilte Daten besitzen Eigenschaften, die eine einfachere statistische Modellierung ermöglichen, was zu effizienteren Kompressionen führen kann (vgl. [13], [15], [17]).

2.1.4 Metriken der verlustbehafteten Bildkompression

Da bei der verlustbehafteten Bildkompression neben den redundanten Daten auch weniger essenzielle Informationen entfernt werden, ist keine fehlerfreie Rekonstruktion des Originalbildes möglich. Daher muss bei der Bewertung von verlustbehafteten Kompressionsverfahren, wie dem verwendeten Autoencoder, neben der Kompressionsrate auch die Rekonstruktionsqualität berücksichtigt werden. In diesem Abschnitt werden die drei Metriken vorgestellt, die zur Bewertung des Kompressionsmodells in dieser Arbeit verwendet werden.

PSNR

Die PSNR ist eine leicht berechenbare Metrik, die in der verlustbehafteten Bildkompression zur Messung der Qualität von rekonstruierten Bildern eingesetzt wird. Wie bereits durch den Namen angedeutet, berechnet diese Metrik das Verhältnis zwischen dem höchsten Pixelwert L (Peak-Signal) und der Differenz zum Originalbild (Noise), das in Form des Mean Square Error (MSE) zwischen den Pixeln des Originalbildes und des rekonstruierten Bildes ermittelt wird. Die Gleichung (2.3) stellt die allgemeine Berechnung der PSNR-Metrik dar. (vgl. [18, S.270-273])

$$PSNR = 10 \cdot log_{10} \left(\frac{L}{MSE} \right) \tag{2.3}$$

Bei der PSNR-Metrik handelt es sich folglich um ein dimensionsloses Verhältnismaß, bei dem größere Werte auf kleinere Rekonstruktionsfehler hinweisen. Ein typischer Wertebereich liegt zwischen 20 dB und 40 dB. Es ist allerdings zu beachten, dass die PSNR-Bewertungen keine ausreichende Aussagekraft für die subjektive Qualität der rekonstruierten Bilder besitzen. Aus diesem Grund ist diese Metrik besser geeignet, um die Qualität der Rekonstruktionen zwischen verschiedenen Verfahren zu vergleichen. (vgl. [18, S.270-273])

Im Rahmen dieser Arbeit werden die Pixelwerte der Originalbilder und der rekonstruierten Bilder als Fließkommazahlen im normalisierten Wertebereich zwischen 0 und 1 betrachtet. Demzufolge gilt für den höchsten Pixelwert $L = 1$. Die Gleichung (2.3) kann hierzu wie folgt umgestellt werden:

$$PSNR = -10 \cdot log_{10}(MSE) \tag{2.4}$$

MS-SSIM

Im Rahmen der Rekonstruktion von Bildern können höhere Pixelabweichungen auftreten, die zwar zu einer schlechteren PSNR-Bewertung führen, aber vom menschlichen Auge nicht als störend wahrgenommen werden. Aus diesem Grund ist die reine Verwendung der PSNR-Metrik unzureichend, um die Rekonstruktionsqualität eines Kompressionsverfahrens zu bewerten. Infolgedessen wurden Metriken basierend auf der Structural Similarity (SSIM) [19] und später der Multiscale Structural Similarity (MS-SSIM) [20] entwickelt. Beide dienen der Ähnlichkeitsbewertung zwischen zwei Bildern und werden häufig für die Bewertung der Rekonstruktionsqualität zwischen einem Originalbild X und dem rekonstruierten Bild Y verwendet. (vgl. [19], [20])

Da die SSIM-Metrik die Grundlage der in dieser Arbeit verwendeten MS-SSIM-Metrik ist, wird zunächst deren Berechnung erläutert. Dazu werden die beiden Bilder X und Y in N Bild-Patches: $\boldsymbol{x} = \{x_i | i = 1, 2, ..., N\}$ und $\boldsymbol{y} = \{y_i | i = 1, 2, ..., N\}$ zerlegt. Diese Patches liegen dabei an den exakt gleichen räumlichen Positionen im Bild. Für jedes Patch wird ein Vergleich der Helligkeit $l(\boldsymbol{x}, \boldsymbol{y})$, des Kontrasts $c(\boldsymbol{x}, \boldsymbol{y})$ und der Struktur $s(\boldsymbol{x}, \boldsymbol{y})$ durchgeführt. Die Berechnung des SSIM-Werts erfolgt auf der Grundlage der drei Komponenten, wie in Gleichung (2.5) dargestellt. (vgl. [19], [20])

$$SSIM(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha} \cdot [c(\mathbf{x}, \mathbf{y})]^{\beta} \cdot [s(\mathbf{x}, \mathbf{y})]^{\gamma} \tag{2.5}$$

Die Parameter α, β und γ bestimmen die relative Wichtigkeit der drei Komponenten. Gängigerweise werden alle drei Komponenten gleich gewichtet, d.h. $\alpha = \beta = \gamma = 1$. Um den SSIM-Wert des gesamten Bildes zu berechnen, wird der Durchschnitt der SSIM-Werte der einzelnen Patches gebildet. (vgl. [19], [20])

Die subjektive Wahrnehmung der Ähnlichkeit zwischen zwei Bildern wird von verschiedenen Faktoren beeinflusst, wie z.B. dem Abstand des Betrachters zur Bildebene oder seiner individuellen Wahrnehmungsfähigkeit. Je nach Ausprägung dieser Faktoren variiert die subjektive Ähnlichkeitsbewertung, da manche Details wichtiger oder unwichtiger werden. Eine Metrik wie SSIM nutzt nur eine feste Skalierung bzw. Auflösung und kann daher nur in bestimmten Fällen eine Bewertung ermöglichen, die mit der menschlichen Wahrnehmung übereinstimmt. Aus diesem Grund erweitert die MS-SSIM-Metrik die SSIM-Metrik um mehrere Skalierungen. (vgl. [19], [20])
Zur Berechnung werden auf die extrahierten Bild-Patches $\boldsymbol{x}$ und $\boldsymbol{y}$ Tiefpassfilter und Downsampling-Operationen mit dem Faktor zwei angewendet. Dadurch entstehen $j = 1, 2, ..., M$ Skalierungen bzw. Auflösungen für jeden Patch. Anschließend wird ein Vergleich des Kontrasts $c_j(\boldsymbol{x}, \boldsymbol{y})$ und der Struktur $s_j(\boldsymbol{x}, \boldsymbol{y})$ für jede Skalierung j durchgeführt. Der Helligkeitsvergleich erfolgt ausschließlich auf der M-ten Skalierung mit $l_M(\boldsymbol{x}, \boldsymbol{y})$. Der MS-SSIM-Wert berechnet sich über alle Skalierungen mit der Gleichung (2.6). (vgl. [19], [20])

$$MS - SSIM(\mathbf{x}, \mathbf{y}) = [l_M(\mathbf{x}, \mathbf{y})]^{\alpha M} \cdot \prod_{j=1}^{M} [c_j(\mathbf{x}, \mathbf{y})]^{\beta_j} [s_j(\mathbf{x}, \mathbf{y})]^{\gamma_j} \tag{2.6}$$

Die Exponenten α_M, β_j und γ_j geben die relative Bedeutung der Komponenten an. Die MS-SSIM-Gesamtbewertung wird aus dem Durchschnitt der MS-SSIM-Bewertungen der einzelnen Patches berechnet. (vgl. [19], [20])

BPP

Die Metrik „Bits per Pixel (BPP)“gibt die durchschnittliche Anzahl der Bits an, die zur Kodierung eines Pixels in einem Bild benötigt werden. In einem unkomprimierten RGB-Bild werden für die Kodierung eines Pixels 24 Bits verwendet. Für ein unkomprimiertes RGB-Bild ergibt sich eine BPP-Bewertung von 24 BPP. Die durchschnittliche Anzahl der Bits, die zur Kodierung eines Pixels in einem komprimierten Bild benötigt werden, ist in der Regel deutlich kleiner. Je kleiner der BPP-Wert ist, desto stärker ist die Kompression. (vgl. [18, S. 11])

2.2 Stückweise lineare Approximation

Das Ziel der PLA besteht darin, komplizierte und meist rechenintensive Funktionen $f(x)$ in einem Intervall $[a, b]$ durch einfachere, stückweise linear zusammengesetzte Funktionen $L(x)$ anzunähern. Bei der Approximation von Funktionen mit nur einer Variablen wird dazu zunächst eine Punktfolge von n Stützstellen $s_1, s_2..., s_n$ bestimmt, wobei $s_1 = a$ und $s_n = b$ und $s_i < s_{i+1}$ ist. Anschließend wird in jedem Teilintervall $[s_1, s_2], ..., [s_{n-1}, s_n]$ eine lineare Funktion $L_i(x)$ interpoliert. In dieser Arbeit wird nicht die lineare Interpolationsdarstellung genutzt, die verwendet wird, um die lineare Funktion mit Bezug auf die Stützstelle s_i darzustellen. Stattdessen wird jedes lineare Segment durch eine allgemeine Geradengleichung (2.7) ausgedrückt. (vgl. [21], [22])

$$L_i(x) = m_i \cdot x + c_i \tag{2.7}$$

Dabei ist m_i die Steigung des linearen Segments und c_i der y-Achsenabschnitt. Für die Berechnung der Steigung und des y-Achsenabschnitts in allen Teilintervallen gilt:

$$m_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{2.8}$$

$$c_i = f(x_i) - m_i \cdot x_i \tag{2.9}$$

Die Funktion $L(x)$ stellt eine Vereinigung der linearen Verbindungsstrecken (Geraden) zwischen den Punkten der n-elementigen Punktfolge dar und wird daher auch als Streckenzug, Polygonzug oder linearer Spline bezeichnet (vgl. [23]). Mathematisch definiert sich $L(x)$ als:

$$L(x) = \begin{cases} L_1 & \text{für } x \in [s_1, s_2] \\ L_2 & \text{für } x \in [s_2, s_3] \\ ... & \\ L_n & \text{für } x \in [s_{n-1}, s_n] \end{cases} \tag{2.10}$$

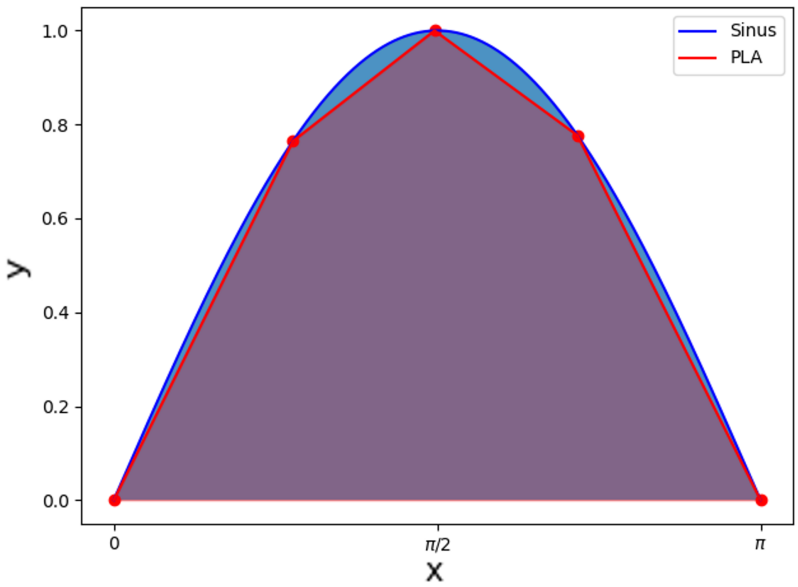


Abbildung 2.3: PLA an einer Sinusschwingung im Intervall [0, π] mit fünf Stützstellen

Bei der PLA handelt es sich zusammenfassend um eine lineare Spline-Interpolation, mit dem Unterschied, dass keine n-elementige Punktfolge gegeben ist, sondern die Punktfolge direkt für die definierte Funktion bestimmt wird. Mit diesem Vorgehen ist es möglich, eine komplexe Funktion, wie z.B. eine trigonometrische, exponentielle oder logarithmische Funktion durch einen einfachen linearen Spline $L(x)$ zu approximieren (vgl. [21], [23], [22]). Die Abb. 2.3 dient der Veranschaulichung der stückweise linearen Approximation am Beispiel einer halben Sinusschwingung im Intervall von $[0, \pi]$. Es ist zu erkennen, dass fünf Stützstellen von der Funktion abgetastet wurden und diese mithilfe einer linearen Interpolation zu einem linearen Spline verbunden wurden.

PLA wird vorwiegend für die Linearisierung von nichtlinearen Funktionen verwendet, um eine effizientere Berechenbarkeit der nichtlinearen Funktionen zu ermöglichen. So verwenden die Autoren in [24] die PLA zur Approximation der nichtlinearen Sigmoidfunktion in einem neuronalen Netz, um eine effiziente Implementierung zu ermöglichen. Mit einem ähnlichen Problem wird sich in [17] beschäftigt. Hierbei werden die nichtlinearen Potenzfunktionen in den Aktivierungsfunktionen durch PLAs approximiert, um die Inferenzgeschwindigkeit und damit den Datendurchsatz des neuronalen Modells auf einer eingebetteten Hardware zu erhöhen.

2.3 Festkommazahlen

Grundsätzlich bestehen alle N-Bit Binärzahlen aus einer Abfolge von Nullen und Einsen. Der tatsächliche Zahlenwert sowie die Behandlung einer Binärzahl hängen vollständig von der Art und Weise ab, wie sie interpretiert wird. In der Programmierung existieren verschiedene Interpretationsmethoden, darunter die Ganzzahlen, die Festkommazahlen oder die Fließkommazahlen. Im Gegensatz zu Fließkommazahlen (IEEE-745 Norm [25]) besitzen Festkommazahlen keinen festgelegten Standard. Bei der Beschreibung von Festkommazahlen existieren in der Literatur unterschiedliche Herangehensweisen. Im Rahmen dieser Arbeit wird sich auf die Definition von Festkommazahlen durch Skalierungsfaktoren bezogen, da diese im Vergleich zum Q-Format [26] oder der vorzeichenlosen U(a,b)-Skalierung [27] eindeutiger und leichter verständlich ist. Darüber hinaus beziehen sich sämtliche Ausführungen auf die vorzeichenlosen Binärzahlen, da diese für die Arbeit relevant sind.
Im Folgenden erfolgt zunächst eine Definition der Festkommazahlen mithilfe von Skalierungsfaktoren. Anschließend wird beschrieben, wie die Skalierungsfaktoren zur Festkommaquantisierung einer Menge von Fließkommazahlen bestimmt werden. Abschließend wird auf die für diese Arbeit relevantesten arithmetischen Operationen mit Festkommazahlen eingegangen.

2.3.1 Definition von Festkommazahlen

In der Festkomma-Interpretation werden Binärzahlen implizit in einen ganzzahligen und einen gebrochenen Anteil unterteilt, die durch a Ganzzahlbits und b Fractionbits dargestellt werden. Diese beiden Anteile werden durch ein festes Komma bzw. einen festen impliziten Binärpunkt voneinander getrennt. Im Gegensatz zu Fließkommazahlen, bei denen die Größe der beiden Anteile und die Stelle des Kommas von Zahl zu Zahl variieren kann, sind die Anteile und somit auch die Position des Kommas bzw. des Binärpunktes für eine Festkommazahl fest. Als Beispiel hat eine vorzeichenlose 8 Bit Binärzahl $x_7x_6x_5x_4bx_3x_2x_1x_0$ mit sechs Ganzzahl- und zwei Fractionbits die folgende allgemeine Form:

$$x_5x_4x_3x_2x_1x_0.x_{-1}x_{-2} \tag{2.11}$$

In diesem Beispiel befindet sich das Komma zwischen dem zweiten und dritten Bit von rechts. In dieser impliziten Festkommadarstellung erhält jedes Bit x_k abhängig von seiner

relativen Position zum Binärpunkt eine neue Bedeutung bzw. einen neuen Wert 2^k. Der dezimale Wert einer solchen vorzeichenlosen N-Bit-Binärzahl X in der impliziten Festkomma-Interpretation ergibt sich anhand der Gleichung (2.12). Der Wertebereich einer solchen Festkommazahl liegt zwischen 0 und $2^a - 2^{-b}$. (vgl. [27], [28])

$$y = (1/2^b) \sum_{0}^{N-1} 2^n x_n \tag{2.12}$$

Die implizite Festkomma-Interpretation verändert somit die Bedeutung der Bits, nicht aber deren Anordnung im Speicher, anders als bei Fließkommazahlen, wo die Binärzahl in Exponent, Mantisse und Vorzeichen aufgeteilt wird. Das most significant bit (MSB) steht weiterhin an der vordersten und das least significant bit (LSB) an der hintersten Stelle. Aus diesem Grund ist es möglich, die Binärzahl sowohl als Ganzzahl als auch als Festkommazahl zu interpretieren. Die Ganzzahl- und die Festkomma-Interpretation hängen dabei über einen festen Skalierungsfaktor 2^b miteinander zusammen. Daher können Festkommazahlen als festkommaskalierte Ganzzahlen angesehen werden. Durch das folgende Beispiel kann dies veranschaulicht werden. Die Binärzahl 1010_2 kann in einer Festkommadarstellung mit einem 2-Bit-Ganzzahl- und einem 2-Bit-Bruchanteil interpretiert werden. In dieser Darstellung ergibt sich der Wert der Binärzahl wie folgt:

Binärzahl: 10.10

Ganzzahlanteil: $10_2 = 2_{10}$

Bruchanteil: $0.10_2 = 0.5_{10}$

Gesamtwert: $2_{10} + 0.5_{10} = 2.5_{10}$

Wird die Binärzahl allerdings als reine Ganzzahl interpretiert, dann hat sie einen Wert von 10_{10}. Um den impliziten Wert der Festkommadarstellung explizit darzustellen, kann der Wert der Ganzzahl durch den zugrunde liegenden Skalierungsfaktor 2^2 geteilt werden:

Gesamtwert: $\frac{10_{10}}{2^2_{10}} = 2.5_{10}$

Zusammenfassend haben Festkommazahlen den Vorteil, dass sie eine begrenzte, feste Anzahl an Nachkommabits abbilden und ermöglichen so eine höhere Genauigkeit als Ganzzahlen. Gleichzeitig können sie deutlich effizienter durch die ganzzahlbasierte Rechenlogik der Hardware berechnet werden als Fließkommazahlen. Sie stellen somit einen Mittelweg dar, da sie zwar ungenauer als Fließkommazahlen sind, aber effizienter berechnet werden können und gleichzeitig genauer als Ganzzahlen sind, aber die Anwendung einer Festkomma-Skalierung erfordern.

2.3.2 Festkommaquantisierung

Die Substitution von Fließkomma- durch Ganzzahloperationen ist entscheidend, um die effiziente Berechnung rechenintensiver Verfahren und Funktionen auf eingebetteten Systemen wie FPGAs zu ermöglichen. Der Prozess der Substitution bzw. Umwandlung wird als Quantisierung bezeichnet. Im Folgenden wird theoretisch dargelegt, wie eine Festkomma-Quantisierung von vorzeichenlosen Fließkommaoperationen durchgeführt wird und wie die Festkommazahlen wieder in Fließkommazahlen umgewandelt werden. (vgl. [27], [28])

Eine mathematische Operation besteht in der Regel aus mehreren Variablen. Jede dieser vorzeichenlosen Fließkomma-Variablen hat einen Wertebereich $[min_{value}, max_{value}]$, der entweder empirisch oder analytisch bestimmt werden kann. Um eine Festkomma-Quantisierung der gesamten Operation zu erreichen, müssen alle Fließkomma-Variablen in Festkomma-Variablen umgewandelt werden. Dazu wird im ersten Schritt der Quantisierung ein Skalierungsfaktor 2^b für jede der Variablen bestimmt. Dabei steht b für die Anzahl an Nachkommabits, die von den festkommaskalierten Ganzzahlen abgebildet werden. Um b zu berechnen, wird zum einen der Maximalwert max_{value} des vorzeichenlosen Wertebereichs jeder Variable ermittelt und zum anderen die Bitanzahl n festgelegt, die für die Kodierung jeder Variable verwendet werden soll. Anschließend kann b mithilfe der Gleichung (2.13) bestimmt und der Skalierungsfaktor abgeleitet werden. (vgl. [27], [28])

$$b = \left\lfloor \log_2 \left(\frac{2^n}{max_{value}} \right) \right\rfloor \tag{2.13}$$

Sobald die Skalierungsfaktoren bestimmt sind, können die Fließkommazahlen x_{float} jeder Fließkomma-Variable in Festkommazahlen x_{fest} umgewandelt werden. Dazu werden

die Fließkommazahlen mit den entsprechenden Skalierungsfaktoren multipliziert und anschließend alle verbleibenden Nachkommastellen abgeschnitten, wie in Gleichung (2.14) dargestellt. (vgl. [27], [28])

$$x_{fest} = \text{int}\left(x_{float} \cdot 2^{b}\right) \tag{2.14}$$

Die Multiplikation mit dem Skalierungsfaktor entspricht auf der binären Ebene einer Schiebeoperation um b-Bit. Wenn b positiv ist, wird die Fließkommazahl vor der Rundung um b Bit nach links verschoben. Dadurch übernimmt die festkommaskalierte Ganzzahl b Bits des Nachkommateils der Fließkommazahl und stellt diesen in der Ganzzahlform dar. Dies lässt sich an einem Beispiel einfacher veranschaulichen. Die Fließkommazahl $101.101_2 = 5.625_{10}$ soll in eine vorzeichenlose 5 Bit Festkommazahl umgewandelt werden. Mit $max_{value} = 5.625$ ergibt sich der Skalierungsfaktor als 2^2. Die Multiplikation mit dem berechneten Skalierungsfaktor und die anschließende Entfernung der verbleibenden Nachkommastellen sind im Folgenden vereinfacht dargestellt:

$$\text{int}\left(5.625_{10} \cdot 2^2_{10}\right) = 22_{10}$$
$$\text{int}\left(101.101_2 \ll 2\right) = 10110_2$$

In der vorliegenden Beispielrechnung weist die Festkommazahl drei ganzzahlige Bits und zwei Bruchbits auf. Der implizite Dezimalwert dieser Festkommazahl ist $101.10_2 = 5.5$. Aufgrund der Tatsache, dass lediglich zwei von den drei Bits des Bruchanteils der Fließkommazahl durch die festkommaskalierte Ganzzahl abgebildet werden, kommt es zu einer Beeinträchtigung der Nachkommagenauigkeit. Die Vorkommastellen hingegen können verlustfrei abgebildet werden. (vgl. [27], [28])
Wenn b negativ ist, können nicht alle Bits des ganzzahligen Anteils der Fließkommazahl durch die Festkommazahl bzw. festkommaskalierte Ganzzahl abgebildet werden. Aus diesem Grund erfolgt eine Schiebeoperation um $|b|$ Bits nach rechts. Dabei gehen Informationen in der Vorkommastelle der Fließkommazahl verloren, und es können zusätzlich keine Informationen der Nachkommastellen abgebildet werden. Dieser Sachverhalt wird erneut mit dem vorherigen Beispiel der Fließkommazahl $101.101_2 = 5.625_{10}$ betrachtet. In diesem Fall erfolgt die Umwandlung in eine vorzeichenlose 2 Bit Festkommazahl. Die Multiplikation mit dem berechneten Skalierungsfaktor 2^{-1} und die anschließende Entfernung der

verbleibenden Nachkommastellen sind im Folgenden vereinfacht dargestellt:

$$\text{int}\left(5.625_{10} \cdot 2_{10}^{-1}\right) = 2_{10}$$
$$\text{int}\left(101.101_2 \gg 1\right) = 10_2$$

Die resultierende Festkommazahl weist zwei ganzzahlige Bits und keine Bruchbits auf. Um die Größe des Quantisierungsfehlers zu bestimmen, ist eine Rückumwandlung in eine Fließkommazahl erforderlich. Dazu muss eine Division der Festkommazahl durch den Skalierungsfaktor gemäß der Gleichung (2.15) erfolgen. In diesem Beispiel beträgt der Quantisierungsfehler $5.625 - 4 = 1.625$. Daran lässt sich erkennen, dass bei einem negativen b der Quantisierungsfehler der Festkommaquantisierung die gesamte Nachkommagenauigkeit und einen Teil der Vorkommagenauigkeit betrifft. (vgl. [27], [28])

$$\hat{x}_{float} = \frac{x_{fest}}{2^b} \tag{2.15}$$

2.3.3 Arithmetische Operationen

Die beiden für diese Arbeit besonders relevanten arithmetischen Operationen sind die vorzeichenlose Addition und Multiplikation von Festkommazahlen. Daher werden die beiden im Folgenden detailliert betrachtet.

Addition

Für eine korrekte Addition zweier Festkommazahlen bzw. festkommaskalierter Ganzzahlen x_1 und x_2 müssen beide vorzeichenbehaftet oder vorzeichenlos sein und über den gleichen Skalierungsfaktor 2^b verfügen. Wenn diese Voraussetzungen erfüllt sind, ergibt die Summe zweier Festkommazahlen bzw. festkommaskalierter Ganzzahlen eine neue Festkommazahl x_s. Diese benötigt ein zusätzliches, ganzzahliges Bit, da bei der Addition ein Überlauf auftreten kann. Im folgenden Beispiel wird zur Verdeutlichung eine Addition zwischen zwei vorzeichenlosen 4 Bit festkommaskalierten Ganzzahlen vorgenommen, die beide mit dem Faktor 2^2 skaliert wurden. (vgl. [27], [28])

$$\begin{aligned}\text{sum} &= 11_{10} + 9_{10} \\ \text{sum} &= 1011_2 + 1001_2 = 10100_2 \\ \text{sum} &= 10100_2 = 20_{10}\end{aligned}$$

Der implizite Wert der beiden Summanden kann mithilfe der Gleichung (2.12) berechnet werden und lautet $10.11_2 = 2.75_{10}$ und $10.01_2 = 2.25_{10}$. Das erwartete dezimale Ergebnis der Addition ist demnach 5. Die Rückumwandlung des festkommaskalierten Ergebnisses 20_{10} durch die Division mit dem Skalierungsfaktor 2^2 ergibt das gleiche Ergebnis.

Multiplikation

Bei der Multiplikation zweier Festkommazahlen müssen beide vorzeichenbehaftet oder vorzeichenlos sein. Die Festkommaskalierungen können voneinander abweichen. Das Produkt zweier Festkommazahlen mit N und M Bits und Skalierungsfaktoren 2^{b_1} und 2^{b_2} ergibt eine neue $N+M$-Bit Festkommazahl mit einem Skalierungsfaktor von $2^{b_1+b_2}$. Im folgenden Beispiel wird dies anhand der Multiplikation von zwei vorzeichenlosen 4 Bit festkommaskalierten Ganzzahlen mit den Skalierungsfaktoren 2^1 und 2^3 dargestellt. (vgl. [27], [28])

$$\begin{aligned}\text{prod} &= 6_{10} \cdot 5_{10} \\ \text{prod} &= 0110_2 \cdot 0101_2 \\ \text{prod} &= 0011110_2 = 30_{10}\end{aligned}$$

Der implizite Wert der beiden Faktoren ergibt sich anhand der Gleichung (2.12) als $011.0_2 = 3_{10}$ und $0.101_2 = 0.625_{10}$. Das Produkt der beiden sollte demnach einen Wert von 1.875_{10} annehmen. Die Rückumwandlung des Ergebnisses der festkommaskalierten Multiplikation 30_{10} durch die Division mit dem Skalierungsfaktor 2^{1+3} ergibt das gleiche Ergebnis.

2.4 Grundlagen Quantisierung

In diesem Abschnitt werden die theoretischen Grundlagen für die Quantisierung der convolutional und deconvolutional Operationen des verwendeten Modells geschaffen. Dazu erfolgt zunächst eine Definition der Quantisierung im Kontext von neuronalen Netzwerken. Anschließend wird auf Quantisierungskonfigurationen (QConfigs) und deren Parameter eingegangen, die die Durchführung der Quantisierung eines Modells bestimmen. Zuletzt wird der allgemeine Ablauf einer PTQ in PyTorch beschrieben. Diese Beschreibung dient als Grundlage für die PTQ, die in dieser Arbeit durchgeführt wird.

2.4.1 Quantisierung von neuronalen Netzen

Um bestmögliche Trainingsergebnisse zu erzielen, werden die Gewichte vieler neuronaler Netzwerke in FP32 trainiert. Die Netze mit den trainierten FP32-Gewichtstensoren erreichen hohe Genauigkeiten, sind jedoch rechenintensiv und haben einen hohen Speicherbedarf. Beim Prozess der Quantisierung werden diese Tensoren in einen ganzzahligen Wertebereich mit einer niedrigeren Bitauflösung (z.B. 8 Bit) konvertiert, um eine effiziente Modellkomprimierung und eine höhere Recheneffizienz zu ermöglichen. Betrachtet man z.B. die Quantisierung von FP32 in Integer 8 Bit (INT8), so lässt sich das Modell um den Faktor 4 komprimieren und der Rechenaufwand typischerweise um einen linearen bis quadratischen Faktor reduzieren. Allerdings führt eine Quantisierung zwangsläufig zu einem Genauigkeitsverlust, der je nach Modell und angewandter Quantisierungsmethode unterschiedlich stark ausfällt (vgl. [29]).

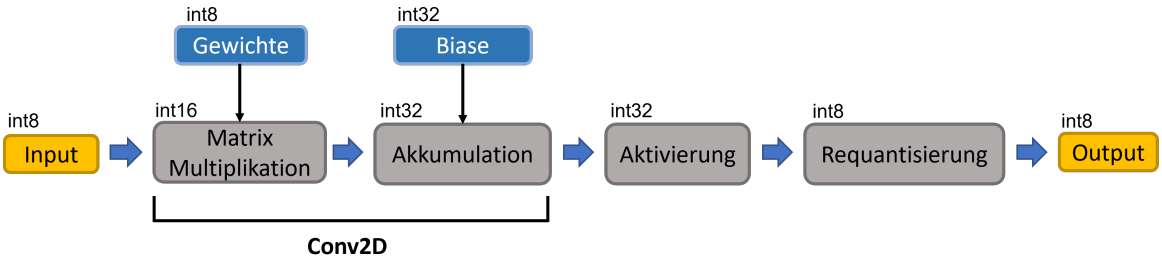


Abbildung 2.4: Schematischer Überblick der Bitbreiten einer quantisierten Convolution

Um eine vollständige Quantisierung eines neuronalen Netzes zu erreichen, müssen seine Eingabedaten und convolutional Operationen quantisiert werden. Besonders die Quantisierung der convolutional Operationen stellt dabei eine Herausforderung dar, da nicht nur

die Gewichtstensoren, sondern auch die Aktivierungstensoren dieser Operationen quantisiert werden müssen. Bei den Aktivierungstensoren handelt es sich um die Ausgabetensoren einer linearen oder convolutional Operation. Der Grund, aus dem eine Quantisierung beider Tensoren erfolgen muss, um eine vollständige INT8-Quantisierung zu ermöglichen, wird durch die Betrachtung der Convolution in Abb. 2.4 verdeutlicht. Zunächst wird eine Matrix-Multiplikation zwischen den INT8-Gewichtswerten und den INT8-Eingabewerten durchgeführt, wobei Integer 16 Bit (INT16)-Ergebnisse entstehen. Diese Werte werden während der Convolution akkumuliert, weshalb zur Vermeidung von Überläufen Integer 32 Bit (INT32)-Akkumulatoren verwendet werden. Die Akkumulatoren werden anschließend mit einem INT32-Bias aufsummiert. Um die nachfolgenden Berechnungen mit INT8 zu ermöglichen, muss eine Quantisierung des INT32-Aktivierungstensors in INT8 erfolgen. Dies wird oft als Requantisierung bezeichnet. (vgl. [29])

2.4.2 Konfigurationen der Quantisierung

Die Quantisierung der Gewichts- und Aktivierungstensoren von convolutional und deconvolutional Operationen wird mittels einer Quantisierungskonfiguration (QConfig) definiert. Für jede zu quantisierende Operation muss eine QConfig bestimmt und zugewiesen werden. Da die Anforderungen an die Quantisierung von Operation zu Operation variieren können, müssen die Parameter der QConfigs so gewählt werden, dass der Quantisierungsfehler minimiert wird. Besonders wichtige Parameter, die einen starken Einfluss auf das Ergebnis der Quantisierung haben, sind das Quantisierungsschema, die Granularität sowie die verwendeten Observer. (vgl. [29]–[32])

Quantisierungsschema

Das Quantisierungsschema bestimmt, wie die Abbildung eines Fließkommatensors x_f auf einen ganzzahligen Tensor x_{int} definiert ist. Dabei werden grundsätzlich zwei Formen unterschieden: die uniforme, asymmetrische und die uniforme, symmetrische Quantisierung. In Abb. 2.5 ist eine **uniforme, asymmetrische Quantisierung** dargestellt. Bei diesem Quantisierungsschema werden die minimalen und maximalen Fließkommawerte von x_f auf die minimalen und maximalen Werte des ganzzahligen Wertebereichs abgebildet. Diese Abbildung bzw. Quantisierung erfolgt mithilfe eines Skalierungsfaktors s, einem Nullpunkt z und einer Bitbreite b. Eine häufig verwendete Bitbreite für die Quantisierung ist 8 Bit.

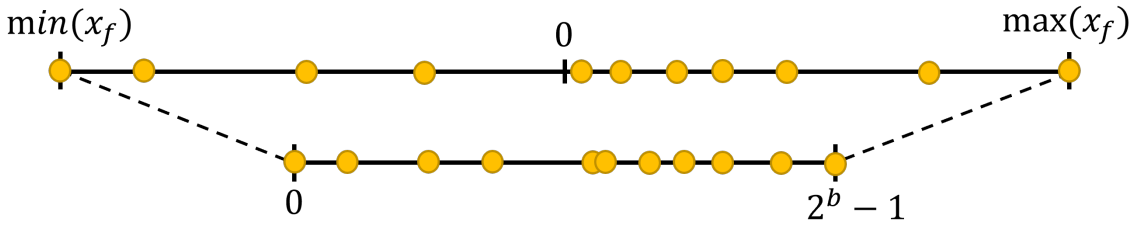


Abbildung 2.5: Schematische Darstellung eines asymmetrischen Quantisierungsschemas (vgl. [30])

Der Skalierungsfaktor und der Nullpunkt werden aus den minimalen und maximalen Werten des Fließkomma- und des ganzzahligen Wertebereichs abgeleitet. Der Skalierungsfaktor definiert die Schrittweite der Quantisierung. Das bedeutet, dass er den Wertebereich des Fließkommatensors in 2^b gleich große Intervalle unterteilt. Alle Fließkommazahlen eines Intervalls werden bei der Quantisierung auf eine der 2^b Ganzzahlen des ganzzahligen Wertebereichs abgebildet. Der Nullpunkt sorgt dafür, dass die Null ohne Quantisierungsfehler im quantisierten, ganzzahligen Bereich abgebildet wird. Im Allgemeinen findet die asymmetrische Quantisierung Anwendung bei vorzeichenlosen Quantisierungen. Es besteht zudem die Möglichkeit, vorzeichenbehaftete Quantisierungen durchzuführen. In diesem Fall ist es erforderlich, dass von jeder quantisierten Zahl x, 2^{b-1} subtrahiert wird. Die Quantisierung eines Fließkommatensors x_f in einen Ganzzahltensor x_{int} erfolgt mithilfe der drei Quantisierungsparameter (QParameter): s, z und b, wie in Gleichung (2.16) und Gleichung (2.17) dargestellt. (vgl. [30])

$$x_{int} = clamp\left(round\left(\frac{x_{float}}{s}\right) + z, 0, 2^b - 1\right) \tag{2.16}$$

$$x_{int} = clamp\left(round\left(\frac{x_{float}}{s}\right) + z - 2^{b-1}, 2^{b-1}, 2^{b-1} - 1\right) \tag{2.17}$$

Bei der **uniformen, symmetrischen Quantisierung** werden für die Abbildung nicht die minimalen und maximalen Werte des Fließkommatensors verwendet. Wie in Abb. 2.6 dargestellt, wird stattdessen der höchste absolute Wert $max\left(|x_f|\right)$ bestimmt und von $-max\left(|x_f|\right)$ bis $max\left(|x_f|\right)$ auf die minimalen und maximalen Werte des ganzzahligen Wertebereichs abgebildet. Da sich der quantisierte Wertebereich somit symmetrisch um die Null verteilt, wird die symmetrische Quantisierung meist für die vorzeichenbehaftete Quantisierung eingesetzt. Der Vorteil dieses Quantisierungsschemas gegenüber der asymmetrischen Quantisierung besteht darin, dass kein Nullpunkt für die Berechnung der Quan-

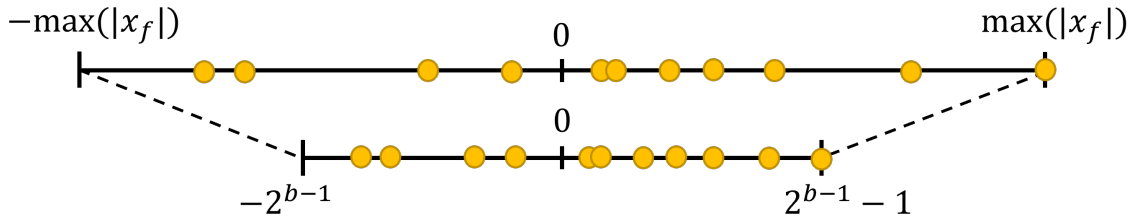


Abbildung 2.6: Schematische Darstellung eines symmetrischen Quantisierungsschemas (vgl. [30])

tisierung notwendig ist. Dies führt zu erheblich einfacheren und effizienteren Berechnungen. Der Nachteil ist, dass die Daten des Fließkommatensors bereits symmetrisch um die Null verteilt sein sollten, da ansonsten nicht der gesamte ganzzahlige Wertebereich bei der Quantisierung ausgenutzt werden kann und dies zu schlechteren Quantisierungsergebnissen führt. Die symmetrische Quantisierung eines Fließkommatensors x_f erfolgt unter der Berücksichtigung der zwei QParameter: s und b, gemäß Gleichung (2.18). (vgl. [30])

$$x_{int} = clamp\left(round\left(\frac{x_{float}}{s}\right), 2^{b-1}, 2^{b-1}-1\right) \tag{2.18}$$

Beide Quantisierungsschemata haben Vor- und Nachteile. In der Fachliteratur wird vielfach darauf hingewiesen, dass für die Quantisierung der Gewichtstensoren eine symmetrische Quantisierung verwendet werden sollte. Der Hauptgrund dafür ist, dass der Nullpunkt der asymmetrischen Quantisierung zu einem erheblichen Mehraufwand in der Berechnung führt, was die Inferenzgeschwindigkeit stark negativ beeinflusst. Darüber hinaus wird für die Aktivierungstensoren eine asymmetrische Quantisierung empfohlen, da diese eine vorzeichenbehaftete und eine vorzeichenlose Abbildung ermöglicht. (vgl. [29], [31])

Granularität

Ein weiterer Parameter für die QConfigs der Gewichts- und Aktivierungstensoren ist die Granularität der Quantisierung. Dieser Parameter lässt sich am einfachsten anhand der Gewichtstensoren veranschaulichen, gilt aber gleichermaßen für jeden beliebigen Fließkommatensor x_f. Ein Gewichtstensor in einer Convolution hat eine Anzahl von c_{out} Filtern. Jeder dieser Filter hat die Höhe h, Breite w und Anzahl an Eingabekanälen c_{in}. Der gesamte Gewichtstensor hat folglich eine Dimension von $w \times h \times c_{in} \times c_{out}$.

Bei der **per-Tensor** Quantisierung werden die Wertebereiche aller c_{out} Filter des Gewichtstensors zu einem gemeinsamen Wertebereich kombiniert. Für die Quantisierung wird demnach nur ein Satz QParameter benötigt, der auf den gesamten Tensor angewendet wird. Dadurch wird die Implementierung auf der Hardware vereinfacht. Aus diesem Grund wird diese Form der Quantisierungsgranularität häufig verwendet. (vgl. [29])
Bei der **per-Channel** Quantisierung werden alle c_{out} Filter des Gewichtstensors mit einem eigenen Satz QParameter quantisiert. Das führt insbesondere dann zu besseren Quantisierungsergebnissen, wenn sich die Wertebereiche der einzelnen Filter signifikant voneinander unterscheiden. Allerdings führt die per-Channel Quantisierung zu einem starken Mehraufwand während der Quantisierung und Requantisierung und wird nicht von allen Hardwareplattformen unterstützt. (vgl. [29])
In der Literatur wird für die Quantisierung der Gewichte eine per-Channel Quantisierung empfohlen, da diese in der Regel zu deutlich besseren Quantisierungsergebnissen führt. Für die Aktivierungstensoren sollte hingegen eine per-Tensor Quantisierung verwendet werden, da die Requantisierungen ansonsten sehr rechenaufwendig werden. Des Weiteren ist zu berücksichtigen, dass per-Channel Quantisierungen für Aktivierungstensoren von den meisten Hardwareplattformen nicht unterstützt werden. (vgl. [29], [31])

Observer

Für die Berechnung der QParameter wird ein minimaler und ein maximaler Wert aus dem Wertebereich des zu quantisierenden Fließkommatensors x_f benötigt. In PyTorch existieren dafür sogenannte Observer. Diese werden während des Quantisierungsprozesses für die Ermittlung der Datenverteilung des jeweiligen Tensors verwendet. Abhängig vom verwendeten Observer werden unterschiedliche Strategien, wie z.B. die Min-Max-Strategie verwendet, um aus den ermittelten Datenverteilungen den minimalen und maximalen Wert zu bestimmen. Aus diesen können die QParameter für die Quantisierung des entsprechenden Tensors bestimmt werden. In PyTorch gibt es verschiedene Observer, die für unterschiedliche Anforderungen und Datenverteilungen entwickelt wurden. Zu den gängigen Observern gehören:

- MinMaxObserver
- HistogramObserver
- MovingAverageMinMaxObserver

Üblicherweise wird ein MinMaxObserver für die Überwachung von Gewichtstensoren und ein MovingAverageMinMaxObserver für die Überwachung von Aktivierungstensoren verwendet. (vgl. [31], [32])

2.4.3 Ablauf einer PTQ in PyTorch

Der Ablauf einer PTQ im PyTorch Framework kann in folgende fünf Schritte unterteilt werden:

1. Festlegung der zu quantisierenden Bereiche und Schichten des Modells
2. Fusion von Schichten und Aktivierungsfunktionen (Optional)
3. Festlegung der QConfigs
4. Berechnung der QParameter
5. Konvertierung des Modells

Die ersten drei Schritte dienen der Vorbereitung und Definition der durchzuführenden PTQ für das verwendete Modell. Die beiden letzten Schritte beschreiben den eigentlichen Prozess der Quantisierung. Die folgenden Beschreibungen sollen ein tieferes Verständnis des Quantisierungsprozesses innerhalb des verwendeten Frameworks ermöglichen und gleichzeitig die praktische Vorgehensweise in dieser Arbeit verdeutlichen. (vgl. [31])

Festlegung der zu quantisierenden Bereiche und Schichten

Im ersten Schritt des Quantisierungsprozesses müssen die Bereiche und Schichten des Modells gekennzeichnet werden, die während der PTQ quantisiert werden sollen. Ein Bereich bezieht sich in diesem Kontext auf einen definierten Abschnitt des Modells. Dies kann eine einzelne Schicht, mehrere Schichten, nichtlineare Funktionen, arithmetische Operationen oder das gesamte Modell umfassen. Für die Kennzeichnung stellt PyTorch sogenannte QuantStubs und DeQuantStubs zur Verfügung. Ein QuantStub besitzt eigene QParameter, mit denen ein FP32-Tensor in einen INT8-Tensor quantisiert werden kann. Ein DeQuantStub kann eine Dequantisierung eines INT8-Tensors in einen FP32-Tensor vornehmen. Ein Bereich im Netzwerk, der im Rahmen der PTQ quantisiert werden soll, wird demnach mit einem vorangestellten QuantStub und einem nachfolgenden DeQuantStub gekennzeichnet. Da eine beliebige Anzahl von Stubs an frei wählbaren Positionen im Modell platziert werden

kann, können die zu quantisierenden Bereiche leicht festgelegt werden. (vgl. [31], [32]). Darüber hinaus müssen arithmetische Operationen, die mit den quantisierten Werten durchgeführt werden, durch sogenannte FloatFunctionals (FFs) ersetzt werden. Diese PyTorch-Funktionalitäten besitzen eigene QParameter, die während der Quantisierung bestimmt werden. Mit diesen können sie eine Requantisierung der Ergebnisse der arithmetischen Operationen durchführen, um sicherzustellen, dass die Werte weiterhin im INT8-Wertebereich liegen. (vgl. [31], [32])

Fusion der Schichten und Aktivierungsfunktionen

In diesem zweiten, optionalen Schritt werden Fusionen zwischen den convolutional Operationen und den Aktivierungsfunktionen des Netzwerks definiert. Eine Fusion ermöglicht es, dass die Aktivierungsfunktion auf den Aktivierungstensor angewendet wird, bevor eine Requantisierung in INT8 erfolgt. Dies hat den Vorteil, dass die convolutional Operationen und die Aktivierungsfunktionen in einem Schritt quantisiert werden können, was eine effizientere Berechnung und einen geringeren Quantisierungsfehler ermöglicht. Da PyTorch zum Zeitpunkt der Erstellung dieser Arbeit nur die Fusion zwischen convolutional Operationen, ReLU-Funktionen und Batchnormalisierungen unterstützt, kann keine Fusion der Convolutions bzw. Deconvolutions und der Generalized Divisive Normalizations (GDNs) und Inverse Generalized Divisive Normalizations (IGDNs) des betrachteten Autoencoders erfolgen. Aus diesem Grund wird dieser optionale, aber empfohlenen Schritt in dieser Arbeit übersprungen. (vgl. [31], [32])

Festlegung der QConfigs

Der dritte Schritt besteht in der Bestimmung und Zuweisung von QConfigs. Die Zuweisung kann individuell für jede zu quantisierende Operation bzw. Schicht erfolgen oder es wird dem Modell global eine QConfig zugewiesen, die auf alle Operationen bzw. Schichten in den Quantisierungsbereichen angewendet wird. Da in diesem Schritt festgelegt wird, wie die Gewichts- und Aktivierungstensoren jeder zu quantisierenden Schicht quantisiert werden, hat er den größten Einfluss auf die Qualität und Effizienz der Quantisierung (vgl. [31], [32]).

Berechnung der QParameter

In diesem Schritt werden auf Grundlage der zuvor festgelegten QConfigs die QParameter für die Gewichts- und Aktivierungstensoren für jede zu quantisierende Operation bzw. Schicht bestimmt. Darüber hinaus werden die QParameter der Quant- und DeQuantStubs und der FFs ermittelt. Die Bestimmung der QParameter gestaltet sich bei den Gewichtstensoren relativ unkompliziert, da die Gewichte nach dem Trainingsprozess fixiert und ihre Verteilungen bekannt und konstant sind. Zur Bestimmung der QParameter müssen lediglich mithilfe der ausgewählten Observer die minimalen und maximalen Fließkommawerte ermittelt werden. Diese dienen als Grundlage für die Berechnung der Skalierungsfaktoren sowie gegebenenfalls der Nullpunkte (vgl. [31], [32]).
Komplexer gestaltet sich die Berechnung der QParameter der Aktivierungstensoren, der Eingabetensoren der Stubs und der Ergebnistensoren der FFs. Die Statistiken dieser Tensoren ergeben sich aus den Eingabedaten, die an der entsprechenden Position des Modells verarbeitet werden. Diese Datenabhängigkeit erschwert es, die QParameter bereits vor der Inferenz zu bestimmen. Im Rahmen der PTQ wird diese Berechnung durch die Verwendung eines Kalibrierungsdatensatzes ermöglicht. Dabei wird der Kalibrierungsdatensatz als Eingabe für das Netzwerk verwendet. Während die Datenpunkte des Datensatzes durchlaufen werden, überwachen die Observer die Aktivierungstensoren, die Eingabetensoren der Stubs und die Ergebnistensoren der FFs. Nachdem alle Daten des Kalibrierungsdatensatzes durch das Netzwerk verarbeitet wurden, bestimmen die Observer anhand der gesammelten Statistiken die minimalen und maximalen Werte. Auf Basis dieser Werte werden dann die QParameter berechnet. Je besser der Kalibrierungsdatensatz die tatsächlichen Datenverteilungen, die später während der Inferenz auftreten, widerspiegelt, desto präziser und effektiver ist das Quantisierungsergebnis des Modells (vgl. [31], [32]).

Konvertierung des Modells

Im letzten Schritt werden die bestimmten QParameter verwendet, um eine Quantisierung der Gewichtstensoren zu realisieren. Diese Quantisierung kann bereits vor der Inferenz vorgenommen und die quantisierten Gewichte gespeichert werden. Die QParameter für die Requantisierung der Aktivierungstensoren, die Quantisierung und Dequantisierung durch die Stubs und die Requantisierung nach den FFs werden hingegen nur gespeichert, um während der Inferenz verwendet zu werden. (vgl. [31], [32])

3 Technische Grundlagen

In diesem Kapitel werden technische Komponenten erläutert, die als Grundlage für diese Arbeit dienen. Eine dieser Komponenten ist der Autoencoder, der für diese Arbeit vorgegeben ist. Konkret handelt es sich dabei um das vortrainierte PyTorch Modell *„bmshj2018_factorized“* [33] aus der CompressAI Bibliothek, das auf dem Modell von Ballé et al. [9] basiert. Im ersten Teil dieses Kapitels wird die Architektur des Modells detailliert beschrieben. Anschließend werden die beiden für die Untersuchungen in dieser Arbeit verwendeten Datensätze vorgestellt.

3.1 Architektur des Autoencoders

Wie in Abb. 3.1 dargestellt, besteht der Autoencoder aus einem Encoder-Netzwerk, einem latenten Raum (auch Bottleneck genannt) und einem Decoder-Netzwerk. Der Autoencoder wurde darauf trainiert, eine Kompression und Rekonstruktion von RGB-Bildern mit den Dimensionen $256 \times 256 \times 3$ Pixel durchzuführen.

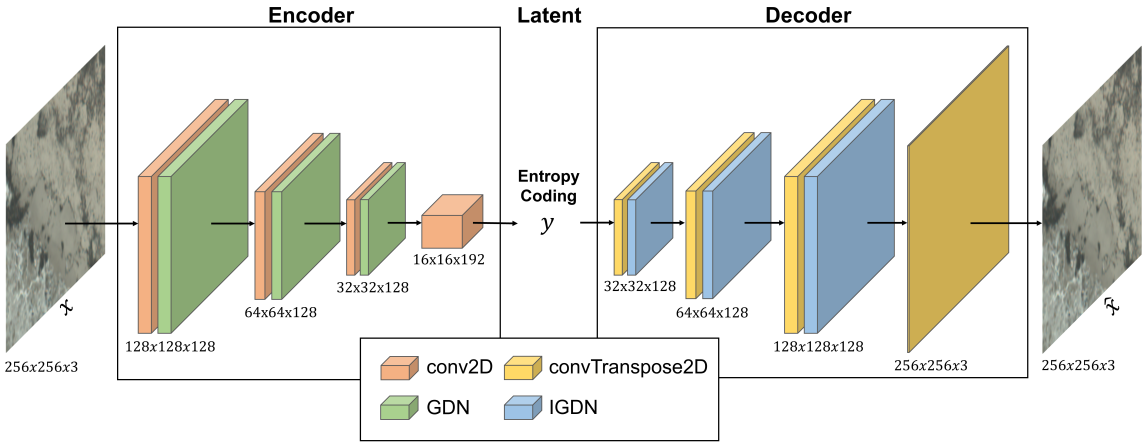


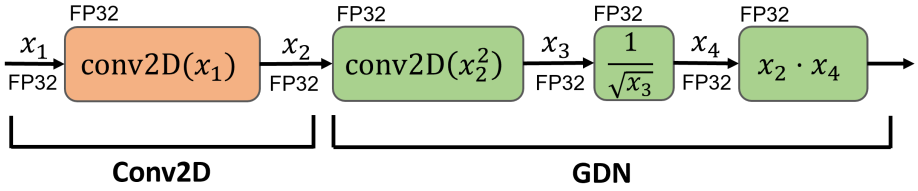
Abbildung 3.1: Vereinfachte Darstellung des verwendeten Autoencoders

Die Funktion des Encoder-Netzes besteht in der Transformation eines Eingabebildes x in eine latente Darstellung y. Dazu werden insgesamt sieben Schichten verwendet. In den ersten sechs Schichten erfolgt ein alternierender Wechsel aus conv2D- und GDN-Schichten.

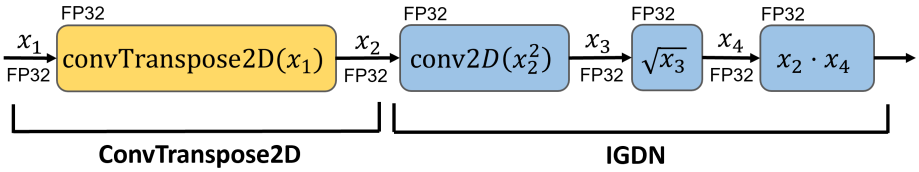
Die conv2D-Schichten extrahieren die wichtigsten Merkmale aus den Eingabedaten und reduzieren gleichzeitig die Bildinformationen, indem sie die Dimensionen der Feature-Maps vierteln. Die anliegenden GDN-Schichten dienen als nichtlineare Aktivierungsfunktionen, die es den conv2D-Schichten ermöglichen, komplexe und nichtlineare Merkmale zu extrahieren. Zudem führen sie eine Gaussianisierung der Datenverteilung der Feature-Maps durch und reduzieren die Interpixel-Redundanzen. Die GDN-Schichten verändern jedoch nicht die Dimensionen der Feature-Maps. Die letzte conv2D-Schicht unterscheidet sich von den vorherigen, da sie statt 128 Filtern über 192 Filter verfügt und nicht durch eine GDN-Schicht aktiviert wird. Nach Anwendung der letzten conv2D-Schicht wird die Transformation des Eingabebildes in die latente Darstellung abgeschlossen.
Anschließend erfolgt im Bottleneck eine Entropie-Kodierung der latenten Darstellung. Diese bewirkt eine noch stärkere Kompression. Das Ergebnis der Entropie-Kodierung ist eine stark komprimierte Version des Eingabebildes. Diese kann effizient gespeichert oder übertragen werden. Um das Bild zu rekonstruieren, wird eine Entropie-Decodierung angewendet, um die latente Darstellung y wiederherzustellen.
Die latente Darstellung wird dem Decoder-Netz übergeben, das eine Transformation in das rekonstruierte Bild $\hat{x}$ durchführt. Das Decoder-Netz hat den gleichen strukturellen Aufbau wie der Encoder. Es besteht aus sieben aufeinander folgenden Schichten, von denen die ersten sechs alternierend zwischen einer parametrischen Operation und einer nichtlinearen Aktivierung wechseln. Konkret werden convTranspose2D-Schichten verwendet, die ein Upsampling des Bildes durchführen, was zu einer Erhöhung der Dimensionen der Feature-Maps führt. Als nichtlineare Aktivierungen der convTranspose2D-Schichten werden die IGDN-Schichten verwendet. Diese führen eine inverse Operation zu den Operationen der GDN-Schichten des Encoders aus. Durch die Kombination aus dem Upsampling der convTranpose2D-Schichten und den IGDN-Schichten kann das Decoder-Netz eine inverse Transformation zu der des Encoders durchführen. Auf diese Weise ermöglicht es die Rekonstruktion des Bildes.

Detaillierte Darstellung der Schichten

Im Folgenden werden detaillierte Darstellungen der Netzschichten abgebildet, die einen besseren Überblick über die Operationen in den Schichten ermöglichen. In Abb. 3.2a sind die Operationen einer conv2D- und einer GDN-Schicht detailliert dargestellt. Hierbei ist zu erkennen, dass jede der GDN-Schichten aus drei Operationen besteht. Erstens eine Convolution der quadrierten Feature-Map der vorhergehenden conv2D-Schicht, zweitens die Be-



(a) conv2D- und GDN-Schicht des Encoders



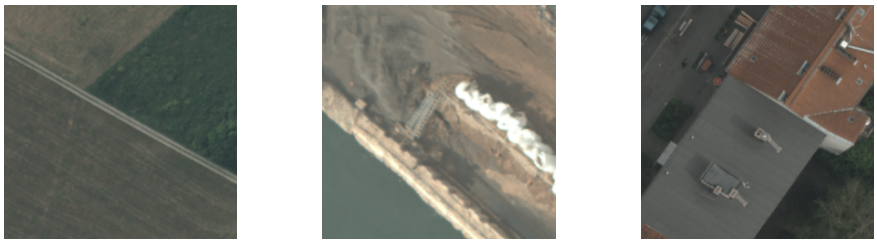
(b) convTranspose2D- und IGDN-Schicht des Decoders

Abbildung 3.2: Detaillierte Darstellung der Schichten des Autoencoders

rechnung der nichtlinearen, inversen Wurzel (RSQRT) für das Ergebnis dieser Convolution, und drittens die Multiplikation des Ergebnisses mit der originalen Feature-Map. Abb. 3.2b zeigt die detaillierte Darstellung einer conv2DTranspose- und einer IGDN-Schicht. Es ist zu erkennen, dass sich die GDN- und die IGDN-Schicht kaum voneinander unterscheiden. Der größte Unterschied ist die verwendete Potenzfunktion. In den IGDN-Schichten wird anstelle einer RSQRT eine konventionelle Wurzel (SQRT) verwendet.

3.2 Datensätze

Der erste der beiden Datensätze wird im Folgenden als Kalibrierungsdatensatz bezeichnet. Dieser Datensatz wird sowohl für die Quantisierung als auch für die Approximation verwendet. Er besteht aus 2280 RGB-Bildern. Dabei handelt es sich um 256×256 Pixel große Bildausschnitte, die aus hochauflösenden Luftbilddaten des Modular Aerial Camera System (MACS) extrahiert wurden. Der Datensatz deckt viele unterschiedliche Motive ab, wie z.B. Waldlandschaften, Felder, Küsten, Meere sowie urbane Fragmente wie Häuser, Straßen, Autos, Industrieanlagen etc. Die Vielfalt der Motive wurde bewusst gewählt, um eine große Variabilität der Bildinhalte in der Kalibrierung zu berücksichtigen. Auf diese Weise wird sichergestellt, dass das approximierte und quantisierte Modell robust gegenüber verschiedenen Eingaben ist und nicht auf bestimmte Szenarien optimiert wird. Das minimiert das Risiko schlechter Kompressionsergebnisse, falls das Modell auf bisher unbekannte Daten trifft. Abb. 3.3 zeigt beispielhaft drei Bildausschnitte.



(a) Feld-Motiv (b) Küsten-Motiv (c) Stadt-Motiv

Abbildung 3.3: Beispielbilder aus dem Kalibrierungsdatensatz

Zusätzlich zum Kalibrierungsdatensatz wird ein Datensatz zur Evaluierung des verwendeten Kompressionsmodells definiert. Der Evaluierungsdatensatz besteht aus sechs RGB-Bildern mit einer Größe von 4864×3072 Pixel. Die Bilder zeigen ähnliche Motive wie die des Kalibrierungsdatensatzes und wurden ebenfalls mit dem MACS aufgenommen. Bei der Evaluierung der vorgestellten Ansätze mit diesem Datensatz wird wie folgt vorgegangen: Zunächst werden die großen Bilder des Datensatzes in 256×256 Pixel große Kacheln unterteilt, da der Autoencoder auf diese Kachelgröße trainiert wurde. Anschließend wird jede Kachel einzeln vom Autoencoder komprimiert und rekonstruiert, wobei die Kompressions- und Rekonstruktionsleistung anhand entsprechender Metriken gemessen wird. Nachdem alle Kacheln eines großen Bildes berechnet wurden, werden die Ergebnisse gemittelt und das große Bild wieder zusammengesetzt. Dieser Vorgang wird für alle großen Bilder des Datensatzes wiederholt, und die Ergebnisse werden erneut gemittelt.

4 Methodik der Quantisierung

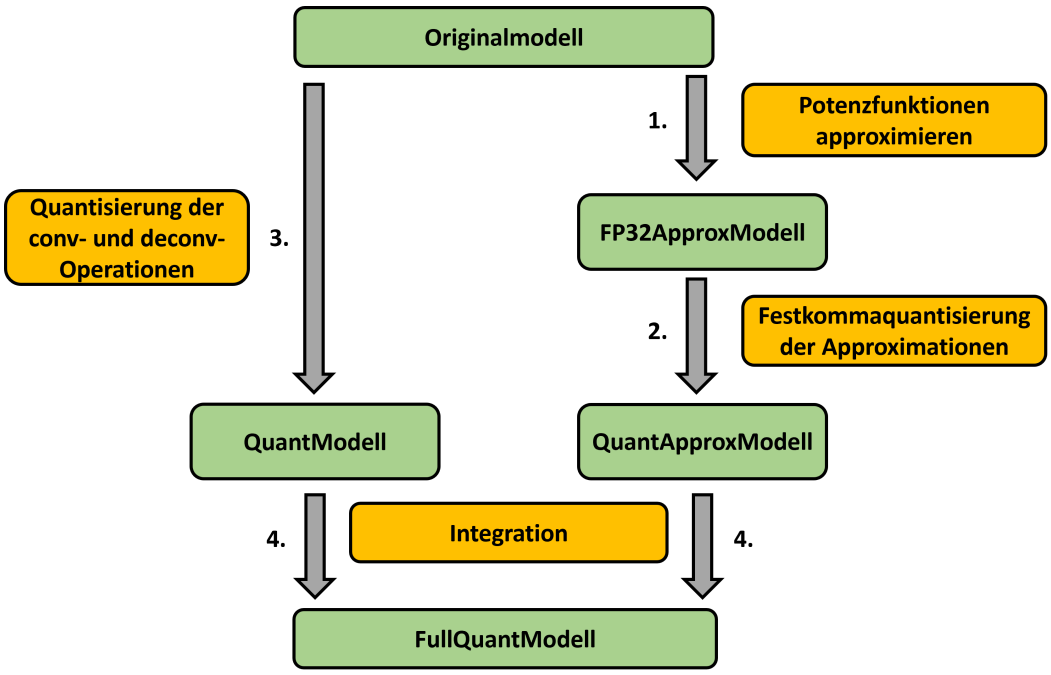


Abbildung 4.1: Flussdiagramm zur Darstellung der Reihenfolge der Methodikbeschreibung

Dieses Kapitel befasst sich mit der Beschreibung der Methodiken zur vollständigen Quantisierung des Encoder- und des Decoder-Netzes des verwendeten Autoencoders. Diese Quantisierung beinhaltet die lineare Approximation und Festkommaquantisierung der nichtlinearen Potenzfunktionen sowie die Quantisierung der convolutional und deconvolutional Operationen in den Netzwerkschichten. Wie in Abb. 4.1 zu erkennen, wird im ersten Schritt die Methodik zur linearen Approximation der sechs Potenzfunktionen (RSQRT, SQRT) in den GDN- und IGDN-Schichten des originalen Autoencoders aufgezeigt. Daraufhin wird die Methodik der Festkommaquantisierung der sechs ermittelten linearen Approximationen dargelegt. Im dritten Schritt wird die Methodik der Quantisierung der convolutional und deconvolutional Operationen beschrieben. Abschließend wird aufgezeigt, wie durch die Integration der festkommaquantisierten linearen Approximationen und der quantisierten Operationen eine vollständige Quantisierung des Encoder- und Decoder-Netzes erreicht werden kann.

4.1 Approximation Potenzfunktionen

Für die Berechnung des Autoencoders auf der inferenzbeschleunigten Hardware stellen die RSQRT und die SQRT der GDN- und IGDN-Schichten eine besondere Herausforderung dar. Der Grund dafür ist, dass für die Berechnung nichtlinearer Funktionen iterative Verfahren, wie beispielsweise das Newton-Raphson-Verfahren [34] oder das CORDIC-Verfahren [35] verwendet werden müssen. Diese benötigen für die Berechnung solcher Funktionen mehrere Iterationsschritte, um ein konvergentes Ergebnis zu erhalten. Aufgrund ihres iterativen Charakters führen diese Verfahren bei der Berechnung auf der FPGA-Hardware zu ungewissen Verzögerungen bei den parallelen Berechnungen und ermöglichen im Allgemeinen nur einen geringeren Datendurchsatz als lineare Funktionsberechnungen, die keine Iterationsschritte erfordern. (vgl. [17], [36])
Da eine hohe Inferenzgeschwindigkeit und ein hoher Datendurchsatz bei der Inferenz des Autoencoders auf der FPGA-Hardware erreicht werden sollen, müssen iterative Verfahren und somit die Verwendung von Potenzfunktionen vermieden werden. Eine passende Lösung ist die Verwendung einer linearen Approximation der Potenzfunktionen. Ein geeignetes Verfahren hierfür ist die PLA, die bereits in mehreren Forschungsarbeiten erfolgreich zur linearen Approximation nichtlinearer Funktionen eingesetzt wurde (vgl. [17], [24]). Ein Vorteil dieses Verfahrens ist, dass es vor der Inferenz berechnet werden kann, sodass die linearen Segmente während der Inferenz geladen und verwendet werden können, ohne dass sie auf der FPGA-Hardware neu berechnet werden müssen.
Im ersten Teil dieses Abschnitts wird die zweischrittige Methodik zur Berechnung der PLAs für die RSQRT- und die SQRT-Funktionen im Autoencoder beschrieben. Wie in Abb. 4.2 dargestellt, muss zunächst eine individuelle Schätzung des Approximationsintervalls jeder Potenzfunktion erfolgen. Anschließend wird für jede Potenzfunktion in ihrem Approximationsintervall eine n-elementige Punktfolge bzw. PLA ermittelt. Im Anschluss an die Beschreibung der Methodik zur PLA-Erstellung wird eine Untersuchung durchgeführt. Bei dieser wird ermittelt, wie viele Stützstellen für die PLAs der Potenzfunktionen erforderlich sind, damit der Autoencoder mit approximierten Potenzfunktionen im Vergleich zum originalen Autoencoder einen maximalen Verlust der Rekonstruktionsqualität von 2 dB PSNR und $1\,\%$ MS-SSIM aufweist.

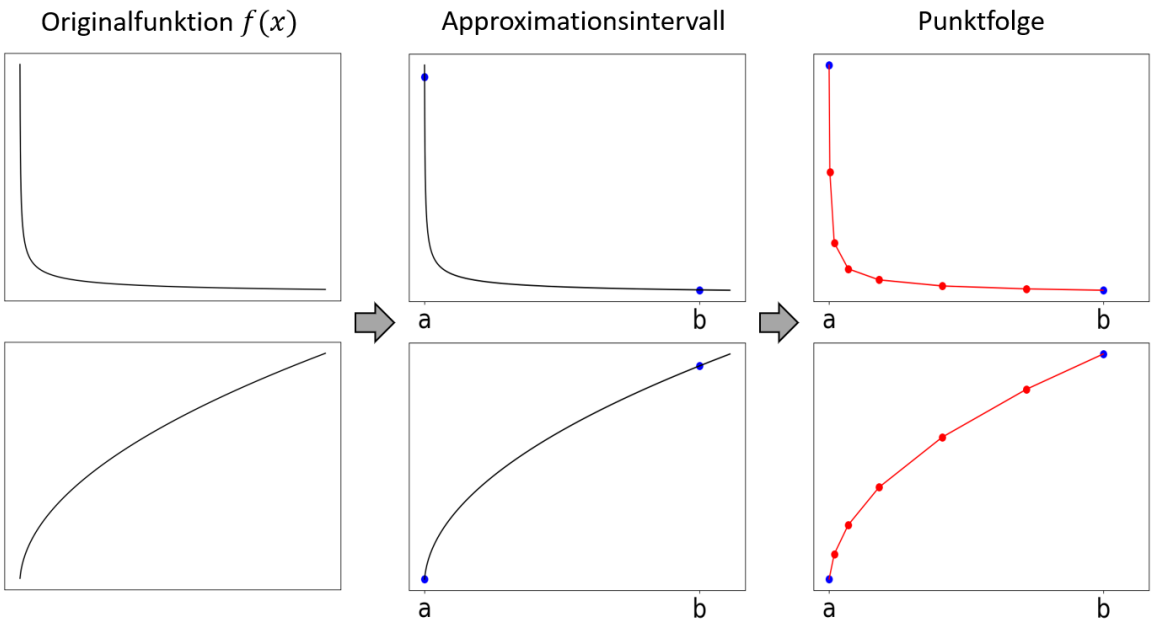


Abbildung 4.2: Hauptschritte zur Erstellung von PLAs

4.1.1 Schätzung der Approximationsintervalle

Bevor eine PLA erzeugt werden kann, muss zunächst das Intervall $[a, b]$ für die Approximation einer RSQRT- bzw. einer SQRT-Funktion des Autoencoders bestimmt werden. Dieses Intervall ist eine Schätzung des Wertebereichs der Eingangsdaten der jeweiligen RSQRT- bzw. SQRT-Funktion. Um eine möglichst realistische Schätzung zu ermöglichen, wird der Kalibrierungsdatensatz verwendet, da dieser im Idealfall die zu erwartenden Eingangsdaten statistisch weitestgehend abdeckt. Zur Ermittlung des Intervalls werden alle 2280 Bilder des Kalibrierungsdatensatzes durch den Autoencoder geleitet. Dabei wird für jedes durchlaufene Bild der Minimal- und Maximalwert der Eingangsdaten für die sechs Potenzfunktionen ermittelt und gespeichert. Nachdem alle 2280 Bilder durchlaufen wurden, liegen für jede Potenzfunktion zwei seperate Listen vor. Die eine beinhaltet die 2280 Minimalwerte und die andere enthält die 2280 Maximalwerte. Aus den beiden zugehörigen Listen jeder Potenzfunktion werden die kleinsten Minimalwerte und die größten Maximalwerte bestimmt und als Intervallgrenzen a und b verwendet. Am Ende dieses Prozesses haben alle sechs Potenzfunktionen ein eigenes Approximationsintervall, für das im nächsten Schritt eine PLA berechnet werden kann.

Mit dieser Intervallbestimmungsmethode konnten die in Tabelle 4.1 dargestellten Intervalle für die Potenzfunktionen bestimmt werden. Es ist zu erkennen, dass es zwischen den Schichten große Unterschiede in den Wertebereichen der Eingabedaten gibt. Dies verdeut-

licht, dass eine genaue, schichtweise Bestimmung der Approximationsintervalle erforderlich ist und eine einheitliche Schätzung der Intervalle zu einer ineffizienten und ungenauen Approximation führen würde.

	GDN 0	GDN 1	GDN 2	IGDN 0	IGDN 1	IGDN 2
Min	0.1135	0.5121	1.5140	2.2865e-06	1.0567e-06	2.0583e-05
Max	304.3966	997.2007	10797.9893	0.7377	4.2467	5.6274

Tabelle 4.1: Eingabewertebereiche der Potenzfunktionen der n-ten GDN- bzw. IGDN-Schicht des Autoencoders

Es sei erwähnt, dass Änderungen des Kalibrierungsdatensatzes zu Änderungen der Wertebereiche der Eingangsdaten der Potenzfunktionen führen. Daher muss in diesem Fall sowohl die Berechnung der Intervalle als auch die Berechnung der PLAs neu durchgeführt werden.

4.1.2 Bestimmung der Punktfolgen

Für die Bestimmung n-elementiger Punktfolgen zur Approximation der Potenzfunktionen in den berechneten Approximationsintervallen können unterschiedliche Verfahren eingesetzt werden. Im Rahmen dieser Arbeit wird ein genetischer Algorithmus verwendet, da er im Vergleich zu anderen getesteten Methoden, wie der Zufallssuche und dem Ansatz von Hamann et al. [37], die besseren Ergebnisse liefert. Um den genetischen Algorithmus für die Bestimmung einer n-elementigen Punktfolge mit kleinstmöglichem Approximationsfehler anwenden zu können, wird diese Aufgabe als Optimierungsproblem definiert. In diesem Optimierungsproblem werden sämtliche Punktfolgen mit n Stützstellen als potenzielle Lösung betrachtet, sofern alle Stützstellen innerhalb des vorgegebenen Approximationsintervalls liegen. Da jede Stützstelle somit einen Freiheitsgrad darstellt, hat das Optimierungsproblem einen n-dimensionalen Suchraum, der nach einer bestmöglichen Lösung durchsucht werden kann.
Im ersten Teil des folgenden Abschnitts wird eine Metrik definiert, mit der der genetische Algorithmus den Approximationsfehler der Punktfolgen bewerten kann. Im zweiten Teil wird der Ablauf des genetischen Algorithmus beschrieben und die gewählten Parameter erläutert, die einen Kompromiss aus Exploration und Exploitation im Suchraum ermöglichen. Zuletzt erfolgt die Bestimmung einer 10-elementigen Punktfolge, um die Ergebnisse des Algorithmus darzustellen. (vgl. [38])

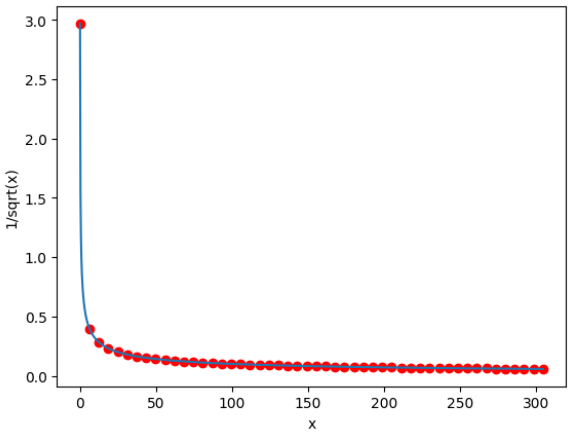


Abbildung 4.3: Gleichmäßige Abtastung der RSQRT

Approximationsfehler-Metrik

Der einfachste Ansatz zur Quantifizierung des Approximationsfehlers besteht darin, die zu approximierende Funktion $f(x)$ an möglichst vielen gleichverteilten Stellen abzutasten, die Funktionswerte von $f(x)$ und die der stückweise linearen Funktion $L(x)$ zu bestimmen und den Root Mean Square Error (RMSE) zu berechnen:

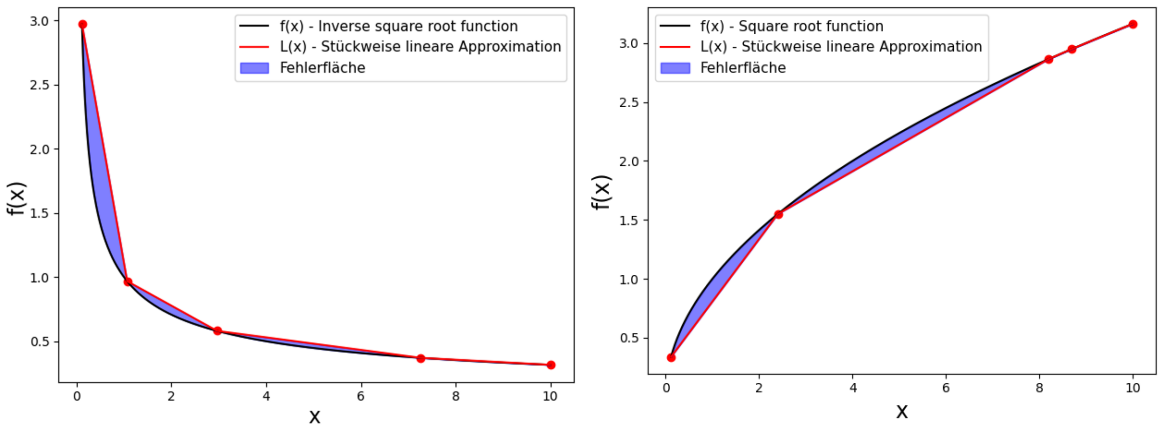
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - L(x_i))^2} \tag{4.1}$$

Die Problematik dieses Ansatzes besteht darin, dass der Verlauf der RSQRT durch die gleichverteilte Abtastung nicht korrekt erfasst wird. Das wird anhand der Abb. 4.3 verdeutlicht. Dies führt dazu, dass der genetische Algorithmus deutlich mehr Stützstellen hinter dem Knick der Funktion platziert, da eine präzise Approximation dieses Bereichs aufgrund der hohen Anzahl an Punkten zu einem geringeren RMSE führt. Eine solche Approximation führt jedoch zu großen Approximationsfehlern, da der Bereich vor und um den Knick nicht genug berücksichtigt wird. Daher ist die gleichmäßige Abtastung und Verwendung des RMSE zur Bestimmung der Stützstellen keine geeignete Methode.

Aus diesem Grund wird eine Metrik verwendet, die auf der prozentualen Flächendifferenz Δ zwischen den Flächeninhalten von $f(x)$ und $L(x)$ basiert. Für die Berechnung von Δ gilt:

$$\Delta = \left| \frac{A_{approx} \cdot 100}{A_{original}} - 100 \right| \tag{4.2}$$

In dieser Gleichung (4.2) steht $A_{original}$ für den Flächeninhalt unter der tatsächlichen Potenzfunktion im Approximationsintervall $[a, b]$. A_{approx} ist die kumulierte Fläche unter den linearen Abschnitten der Approximation. Für die Flächenberechnungen müssen keine negativen Flächen berücksichtigt werden, da gilt: $\forall x : f(x) > 0$. Um die prozentuale Abweichung zwischen der approximierten und der tatsächlichen Fläche zu berechnen, wird $A_{original}$ als $100\,\%$ angenommen. Die Subtraktion mit 100 stellt sicher, dass die direkte Abweichung vom Idealwert ($100\,\%$) bestimmt wird. Der Betrag sorgt schließlich dafür, dass die Flächendifferenz immer als positiver Wert betrachtet wird, unabhängig davon, ob die approximierte Fläche größer oder kleiner als die tatsächliche Fläche ist. Auf diese Weise kann die Metrik zur Bestimmung des Approximationsfehlers der linearen Approximation der RSQRT- und der SQRT-Funktionen eingesetzt werden.



(a) RSQRT (b) SQRT

Abbildung 4.4: Approximationsfehler-Metrik an den beiden Potenzfunktionen im Intervall $[0.1, 100]$ mit einer fünf elementigen Punktfolge

In Abb. 4.4 ist die Metrik anhand der beiden Potenzfunktionen dargestellt. Hierbei sind die beiden Potenzfunktionen schwarz, die Approximationen rot und die Fehlerflächen blau. In dieser exemplarischen Darstellung kann man erkennen, dass der Bereich vor dem Knick, sowie der Bereich nach dem Klick gleichwertig bewertet wird und somit das Problem der vorherigen Metrik gelöst wurde. Für den genetischen Algorithmus wird diese Metrik als Fitnessfunktion verwendet, mit der die Qualität der Punktfolgen bestimmt wird.

Ablauf des genetischen Algorithmus

Der in dieser Arbeit verwendete genetische Algorithmus wird für die Bestimmung n-elementiger Punktfolgen mit kleinstmöglichem Approximationsfehler eingesetzt. Dabei folgt der Algorithmus dem iterativen Ablauf aus Fitnessbewertung, Selektion, Rekombination und Mutation. Im Folgenden werden die einzelnen Schritte sowie die verwendeten Strategien und Parameter dieses Prozesses näher erläutert. (vgl. [38])

1. **Initialisierung:** Die Initialisierung ist ein einmaliger Schritt, bei dem eine Population von m Individuen erzeugt wird. Jedes Individuum besteht aus einem Chromosom mit n zufällig ausgewählten Genen. Im Rahmen dieses Optimierungsproblems repräsentiert jedes dieser n Gene eine Stützstelle s_i im vorgegebenen Approximationsintervall. Folglich enthält jedes Individuum eine Punktfolge von n Stützstellen und stellt eine potenzielle Lösung des Optimierungsproblems dar. Bei der Initialisierung des verwendeten genetischen Algorithmus werden 100 Individuen mit zufälligen Chromosomen aus dem Suchraum erzeugt. Vereinfacht gesagt, werden für jedes Individuum $n-2$ zufällige Werte im Approximationsintervall ermittelt, die zusammen mit den Intervallgrenzen eine n-elementige Punktfolge bzw. ein Chromosom bilden. (vgl. [38])

2. **Fitnessbewertung:** In diesem Schritt wird mithilfe der vorgestellten Metrik (siehe Gleichung (4.2)) der Approximationsfehler und damit die Fitness jedes Individuums bestimmt. Aufgrund der Definition der Metrik haben Individuen mit einem kleineren Approximationsfehler eine höhere Fitness. Das vorliegende Optimierungsproblem lässt sich folglich als Minimierungsproblem charakterisieren.

3. **Selektion:** Im Rahmen der Selektion werden m Individuen aus der Population ausgewählt, die ihre Gene in die nächste Generation weitergeben dürfen. Hierfür existieren verschiedene Selektionsstrategien, wie z.B. die Roulette-Selektion, die rangbasierte Selektion, die Turnierselektion und so weiter. Für den in dieser Arbeit verwendeten genetischen Algorithmus wird die Turnierselektion genutzt. Bei dieser werden $n_{turnier}$ zufällige Individuen aus der Population ausgewählt, von denen das mit der besten Fitness seine Gene in die nächste Generation weitergeben darf. Da die Auswahl mit Zurücklegen erfolgt, können manche Individuen mehrfach ausgewählt werden. Durch die Anzahl der Turnierteilnehmer kann die Exploration und Exploitation präzise gesteuert werden. Bei einer hohen Anzahl werden Individuen mit einer schlechteren Fitnessbewertung seltener ausgewählt, während sich die besten Individuen schnell

durchsetzen. Dies fördert die Exploitation der bereits bekannten Lösungen und reduziert die Exploration, da es schwieriger wird, sich von diesen Optima zu entfernen und neue Bereiche des Suchraums zu erkunden. Die Gefahr einer zu hohen Exploitation besteht darin, dass nur lokale Optima gefunden werden. Eine Anzahl von fünf Turnierteilnehmern schafft für das vorliegende Optimierungsproblem allerdings eine gute Balance zwischen Exploration und Exploitation. (vgl. [38])

4. **Rekombination:** Bei der Rekombination werden aus je zwei Individuen (Eltern) der selektierten Population zwei neue Individuen erzeugt. Die neuen Individuen bilden die neue Generation. Die Chromosomen dieser neuen Individuen setzen sich aus einer Kombination von Teilen der Chromosomen beider Eltern-Individuen zusammen. Die Bestimmung dieser Anteile hängt von der verwendeten Rekombinationsstrategie ab. Für den in dieser Arbeit vorgestellten genetischen Algorithmus wird das parametrische, uniforme Crossover verwendet, da es eine gute Balance zwischen Exploration und Exploitation ermöglicht. Bei dieser Strategie werden zwei Eltern-Individuen zufällig ohne Zurücklegen aus der selektierten Population ausgewählt, die je zwei neue Individuen erzeugen. Abb. 4.5 zeigt, dass für jedes Gen individuell entschieden wird, ob ein Genaustausch stattfindet. In dieser Arbeit wird eine Austauschwahrscheinlichkeit von $p_a = 0.5$ verwendet. Das bedeutet, dass beide neuen Individuen im Durchschnitt die gleiche Anzahl zufällig ausgewählter Gene von beiden Eltern erhalten. (vgl. [38])

5. **Mutation**: Im letzten Schritt werden die Gene jedes neuen Individuums mit einer bestimmten Wahrscheinlichkeit p_m mutiert. Mutation meint hierbei, dass der ursprüngliche Wert des Gens bzw. einer Stützstelle durch einen neuen zufälligen Wert im Approximationsintervall ausgetauscht wird. Die beiden Gene, die die Intervallgrenzen repräsentieren, sind von dem Prozess der Mutation ausgeschlossen. Die verwendete Mutationsstrategie ist die uniforme Mutation. Bei dieser Strategie ist die Mutationswahrscheinlichkeit für jedes Gen bei jedem Individuum und über alle Generationen hinweg gleich. Für den hier vorgestellten genetischen Algorithmus wird eine Mutationswahrscheinlichkeit von $\frac{1}{n}$ verwendet. Dies ist eine übliche Wahl bei der uniformen Mutation, da die durchschnittliche Mutation eines Gens in jedem Individuum für viele Optimierungsprobleme einen guten Kompromiss zwischen Exploration und Exploitation darstellt. (vgl. [38])

6. **Wiederholung:** Die Schritte 2 bis 4 werden über mehrere Generationen wiederholt, bis eine Abbruchbedingung erreicht ist. Für diesen genetischen Algorithmus werden

zwei Abbruchbedingungen definiert. Zum einen bricht der Algorithmus ab, wenn sich der beste Fitnesswert über 100 Generationen um weniger als 1×10^{-5} geändert hat. Diese Konvergenzschwelle stellt sicher, dass eine vollständige Konvergenz des Algorithmus erreicht wird. Zum anderen begrenzt eine maximale Anzahl von 2000 Generationen die Laufzeit des Algorithmus, falls keine Konvergenz entsprechend der gewählten Konvergenzschwelle eintritt.

Parameter wie die Populationsgröße m, die Austauschwahrscheinlichkeit p_a, die Mutationswahrscheinlichkeit p_m, die Anzahl der Turnierteilnehmer $n_{turnier}$, die Konvergenzschwelle usw. wurden durch Trial-and-Error angepasst. Eine ausführliche Analyse war aus zeitlichen Gründen nicht möglich. Die implementierten Strategien und Parameter führen in der Regel zu einer Konvergenz des genetischen Algorithmus und liefern Punktfolgen mit sehr guten Approximationsergebnissen. Darauf wird im Folgenden genauer eingegangen. (vgl. [38])

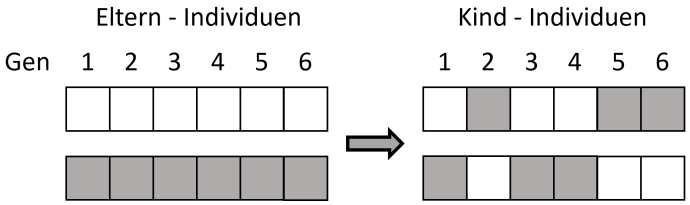


Abbildung 4.5: Schematische Darstellung einer parametrischen, uniformen Rekombination mit einer Austauschwahrscheinlichkeit von $p_a = 0.5$

Anwendung des genetischen Algorithmus

Im Folgenden wird veranschaulicht, dass durch den genetischen Algorithmus genaue und konvergente PLAs für die Potenzfunktionen des Autoencoders erzeugt werden können. Diese Veranschaulichung erfolgt am Beispiel von 10-elementigen PLAs der Potenzfunktion der ersten GDN- und der ersten IGDN-Schicht in deren Approximationsintervall (siehe Tabelle 4.1). Für diese wird zum einen der Konvergenzgraph des genetischen Algorithmus und zum anderen die erzeugte PLA dargestellt.

Die Abb. 4.6 zeigt die Konvergenzgraphen des genetischen Algorithmus für die beiden Potenzfunktionen. Auf der x-Achse sind die Generationen dargestellt, die bis zum Erreichen einer Abbruchbedingung durchlaufen wurden. Auf der y-Achse wird hingegen der Approximationsfehler bzw. prozentuale Flächenfehler in der vorgestellten Approximationsfehler-Metrik aufgetragen. Für jede Generation wird der Approximationsfehler des besten Individuums dargestellt. Um den Verlauf besser erkennen zu können, wurde zwischen den Generationen

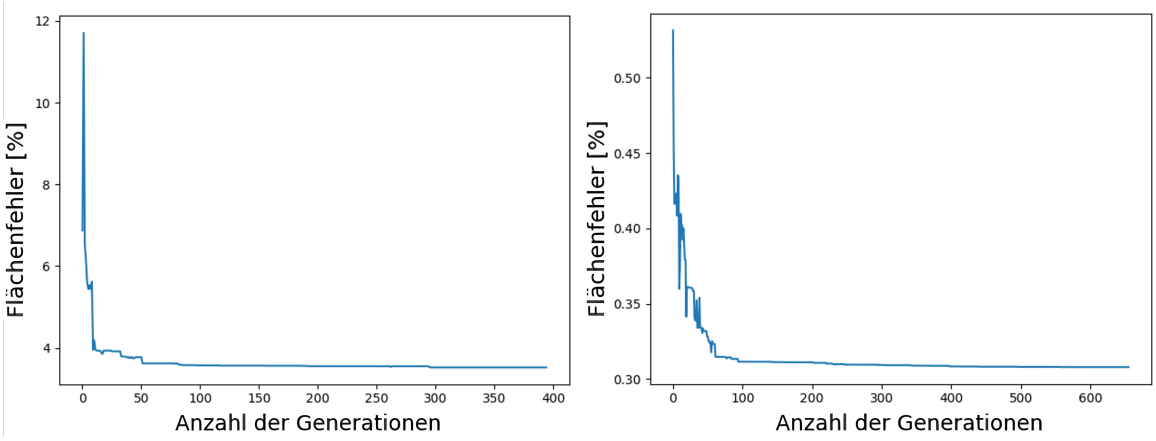
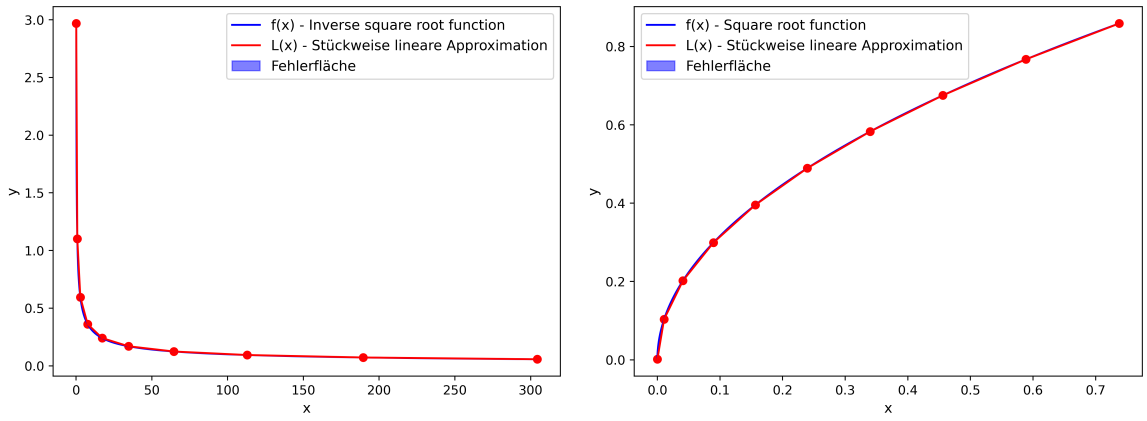


Abbildung 4.6: Konvergenzgraphen des genetischen Algorithmus für die (a) RSQRT der ersten GDN-Schicht und die (b) SQRT der ersten IGDN-Schicht bei der Erzeugung von 10-elementigen Punktfolgen

linear interpoliert. Beide Grafiken zeigen, dass eine Konvergenz vor den maximalen 2000 Generationen erfolgt. Dies weist auf eine gute Exploitation im Suchraum hin. Darüber hinaus ist festzustellen, dass es während des Optimierungsprozesses zu Verschlechterungen im Approximationsfehler des besten Individuums kam. Das ist daran zu erkennen, dass die Graphen über die Generationen nicht dauerhaft sinken, sondern mitunter wieder ansteigen. Das deutet darauf hin, dass der genetische Algorithmus nicht zum ersten Optimum konvergiert, sondern in der Lage ist, lokale Optima zu überspringen. Daraus wird ersichtlich, dass der Algorithmus effektiv im Suchraum explorieren kann. Es sei jedoch darauf hingewiesen, dass nicht bewiesen wird, dass es sich bei den konvergenten PLAs um globale Optima handelt. Allerdings zeigt eine qualitative Begutachtung der Abb. 4.7, dass die Platzierung der Stützstellen der erzeugten PLAs in der Nähe eines globalen Optimums liegen muss. Das kann daran erkannt werden, dass der 'Knick' der RSQRT-Funktion und der niedrige Wertebereich der SQRT-Funktion durch deutlich mehr lineare Segmente abgebildet werden als die asymptotischen Ausläufe der Funktionen. Eine solche Verteilung der Stützstellen ist optimal, um einen geringstmöglichen Approximationsfehler zu erreichen.

Anhand des Beispiels kann gezeigt werden, dass der vorgestellte genetische Algorithmus eine Exploration und Exploitation im 10-dimensionalen Suchraum ermöglicht, wodurch er sehr gute PLAs für die Potenzfunktionen in den gegebenen Approximationsintervallen findet. Des Weiteren wurde erfolgreich geprüft, dass ähnlich gute, konvergente Ergebnisse bei der Erstellung von PLAs mit bis zu 50 Stützstellen für alle Potenzfunktionen des

Autoencoders in deren Approximationsintervallen erzielt werden können.



(a) (b)

Abbildung 4.7: Darstellung von 10-elementigen PLAs der (a) RSQRT der ersten GDN-Schicht und der (b) SQRT der ersten IGDN-Schicht

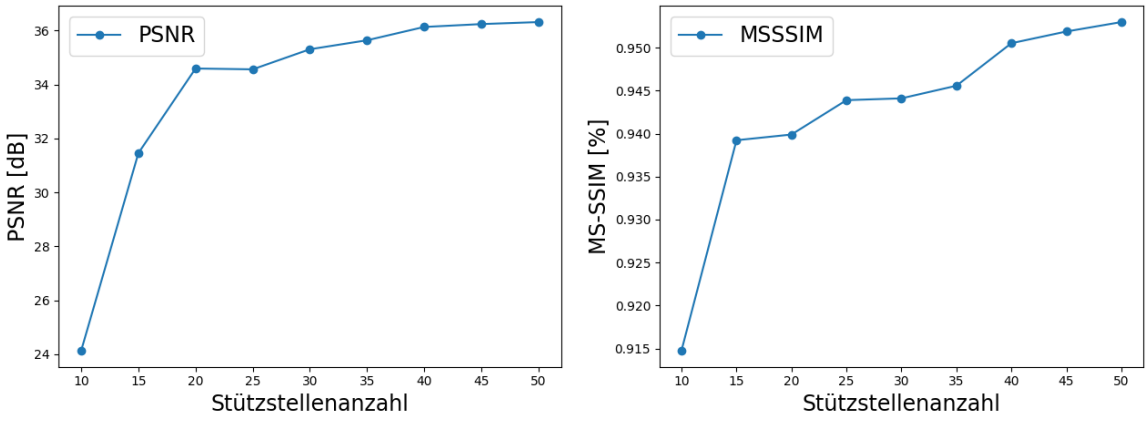
4.1.3 Ermittlung der Punktfolgenlänge

Die Missionsanforderungen setzen einen maximalen Rekonstruktionsfehler von 8 dB PSNR und 6 % MS-SSIM für die gesamten Approximationen und Quantisierungen im Encoder- und im Decoder-Netz fest. Da sich der Approximationsfehler der sechs PLAs bzw. der Kompressions- und Rekonstruktionsfehler des Autoencoders mit approximierten Potenzfunktionen direkt über die Anzahl der verwendeten n Stützstellen steuern lässt, wird in diesem Abschnitt eine geeignete Anzahl von Stützstellen gesucht. Das Ziel ist es, dass der genetische Algorithmus n-elementige Punktfolgen bestimmen kann, die einen durchschnittlichen Rekonstruktionsfehler von 2 dB PSNR und 1 % MS-SSIM erreichen. Dies ermöglicht einen größeren Spielraum, für die Fehler der noch folgenden Quantisierungen. Da der nicht approximierte Autoencoder eine Rekonstruktionsqualität von 36.67 dB PSNR und 95.7 % MS-SSIM auf dem Evaluierungsdatensatz erreicht, sollte der Autoencoder mit approximierten Potenzfunktionen mindestens 34.67 dB PSNR und 94.7 % MS-SSIM erzielen.
Um eine Reproduzierbarkeit der folgenden Untersuchung zu ermöglichen, erfolgt zunächst eine prägnante Darstellung der verwendeten Vorgehensweise. Anschließend wird die Untersuchung durchgeführt, um die kleinste Anzahl von Stützstellen zu finden, mit denen die genannten Anforderungen erfüllt werden können.

Vorgehen der Untersuchung

In der folgenden Untersuchung werden n $= 10, 15, 20, ..., 50$ Stützstellen betrachtet. Frühere Tests mit dem genetischen Algorithmus lassen erwarten, dass die gesuchte Anzahl an Stützstellen in diesem Wertebereich liegt. Für jede dieser Stützstellenzahlen wird ein Untersuchungsdurchlauf durchgeführt, der wie folgt gegliedert ist:

1. Es wird für jede Potenzfunktion im Autoencoder eine PLA mit n Stützstellen erstellt. Hierfür wird der vorgestellte genetische Algorithmus verwendet. Die generierten PLAs ersetzen die ursprünglichen Potenzfunktionen.
2. Der Autoencoder mit den approximierten Potenzfunktionen wird auf den Evaluierungsdatensatz angewendet. Dabei werden die durchschnittlichen PSNR-, MS-SSIM- und BPP-Bewertungen über sämtliche Bilder des Evaluierungsdatensatzes bestimmt.
3. Die Schritte eins und zwei werden fünfmal wiederholt und die Ergebnisse gemittelt. Diese Wiederholung ist erforderlich, da genetische Algorithmen aufgrund der zufälligen Initialpopulation nicht immer zum gleichen Optimum konvergieren. Durch die mehrfache Durchführung und die anschließende Mittelung wird somit eine verlässliche Bewertung der Approximation mit n Stützstellen ermöglicht.



(a) (b)

Abbildung 4.8: Bildrekonstruktion des Autoencoders bei sich ändernder Anzahl von Stützstellen in den verwendeten PLAs, gemessen in (a) PSNR, (b) MS-SSIM

Auswertung

In Abb. 4.8 sind die Ergebnisse der Untersuchung dargestellt. Die x-Achsen der zwei Graphen stellen die untersuchten Stützstellenanzahlen dar, während auf den y-Achsen die entsprechenden Metriken PSNR (a) und MS-SSIM (b) aufgetragen sind. Für jede betrachtete Stützstellenanzahl wurden die gemittelten Metrik-Bewertungen in den Diagrammen dargestellt. Zur besseren Veranschaulichung wurde zwischen den Werten der einzelnen Stützstellenanzahlen eine lineare Interpolation durchgeführt. Es ist jedoch zu betonen, dass diese Interpolation rein illustrativer Natur ist und nicht zwangsläufig ein linearer Zusammenhang zwischen den Werten besteht.
Die Abb. 4.8a zeigt, dass bereits mit 30 Stützstellen eine PSNR-Bewertung von 35.31 dB erreicht werden kann. Dies ist ausreichend, um die Anforderung von 34.67 dB zu erfüllen. Durch die Betrachtung von Abb. 4.8b wird jedoch deutlich, dass mit 30 Stützstellen nur 94.44 % MS-SSIM erreicht werden können. Um auch diese Missionsanforderungen zu erfüllen, müssen 40 Stützstellen verwendet werden, da dann eine MS-SSIM-Bewertung von 95.05 % erreicht wird.
Zusammenfassend zeigt die Untersuchung, dass zur Einhaltung der gestellten Anforderungen durchschnittlich 40-elementige PLAs erforderlich sind. Daher werden diese im weiteren Verlauf der Arbeit für die Approximation der Potenzfunktionen verwendet. Konkret werden die sechs PLAs mit 40 Stützstellen genutzt, die in den fünf Wiederholungen des genetischen Algorithmus die besten Ergebnisse erzielt haben. Mit diesen PLAs erreicht der Autoencoder eine Rekonstruktionsleistung von 36.34 dB PSNR und 95.3 % MS-SSIM, sowie eine Kompressionleistung von 0.147 BPP.

4.2 Festkommaquantisierung der Approximation

Viele FPGAs, wie auch die Zielhardware, besitzen keine Module, die Fließkommaoperationen effizient ausführen können. Mit speziellen FPGA-Designs wäre es zwar möglich, Fließkomma-Operationen zu realisieren, allerdings bräuchte dies eine beträchtliche Menge an logischen Elementen und Speicher, was zu einer größeren genutzten Chipfläche führt. Eine größere Chipfläche begrenzt die maximal erreichbare Taktfrequenz, was wiederum den Datendurchsatz verringert. Zusätzlich benötigen die meisten Fließkomma-Designs mehr Energie. Im Gegensatz dazu sind Festkommaoperationen weniger komplex und lassen sich auf FPGA-Hardware effizienter ausführen. Der Grund dafür ist, dass Festkommaoperatio-

nen effizient auf den integrierten DSP-Blöcken der FPGAs ausgeführt werden können, die auf eine effiziente Durchführung von Additionen, Multiplikationen und Schiebungen mit Festkommazahlen optimiert sind. Abseits davon werden deutlich weniger logische Elemente benötigt. Dadurch können die Festkommaoperationen auf der FPGA-Hardware mit einer deutlich höheren Taktfrequenz, einem höheren Datendurchsatz und einem geringeren Leistungsverbrauch verwendet werden. Um eine effiziente Implementierung des Autoencoders auf der FPGA-Hardware zu realisieren, ist daher eine Festkommaquantisierung der Fließkommaoperationen der linearen Approximationen notwendig. (vgl. [39])
Für eine einfache Substitution der Fließkommaoperationen durch ganzzahlige Berechnungen wäre es allerdings ausreichend, eine INT8-Quantisierung der convolutional und deconvolutional Operationen durchzuführen. Dadurch könnten sämtliche Berechnungen im Encoder- und Decoder-Netzwerk im INT8-Format durchgeführt werden. Die Berechnung der linearen Approximationen der nichtlinearen Potenzfunktionen des betrachteten Modells erfordert jedoch eine höhere Genauigkeit, als durch eine einfache INT8-Quantisierung gewährleistet werden kann. Die eingeschränkte Präzision und der begrenzte Wertebereich führen bei einer direkten Berechnung der quantisierten Werte zu ungenauen Ergebnissen, was die Modellgenauigkeit stark beeinträchtigen kann. Im Gegensatz zu einer reinen INT8-Quantisierung kann die Festkommaquantisierung eine feste Nachkommagenauigkeit in den festkommaskalierten Ganzzahlen abbilden und ermöglicht damit eine wesentlich höhere Genauigkeit und Stabilität bei der Berechnung der komplexen Operationen. Demzufolge kann durch eine Festkommaquantisierung nicht nur eine Substitution der Fließkommaoperationen durch Ganzzahlarithmetik erfolgen, sondern es wird eine deutlich höhere Genauigkeit bei der Berechnung der linearen Approximationen der nichtlinearen Potenzfunktionen ermöglicht, was zu geringeren Fehlern führt.
Aus diesem Grund wird bei der vollständigen Quantisierung des Encoders und Decoders in dieser Arbeit eine separate Quantisierung der linearen Approximationen durch Festkommazahlen und der convolutional und deconvolutional Operationen durch PTQ vorgenommen. Dieser Abschnitt fokussiert sich auf die Beschreibung der Methodik zur Festkommaquantisierung der 40-elementigen PLAs der sechs Potenzfunktionen, die in Abschnitt 4.1.3 bestimmt wurden. Dazu werden zunächst die Bitbreitenbeschränkungen der Rechenoperationen in den DSP-Blöcken der Zielhardware und im festgelegten FPGA-Design erläutert. Anschließend wird begründet dargelegt, inwieweit Vorzeichen für die Berechnung der PLAs beachtet werden müssen. Daraufhin wird die eigentliche Methodik zur Festkommaquantisierung beschrieben.

4.2.1 Bitbegrenzungen im FPGA

Die wichtigste Begrenzung für die Festkommaquantisierung der linearen Approximationen ist die maximale Bitbreite der verwendeten Rechenoperationen, die durch die Hardware oder das gewählte FPGA-Design vorgegeben werden. Da die PLAs aus Multiplikations- und Additionsoperationen bestehen, ist es essenziell, dass die Bitbreiten der beiden Operationen bekannt sind.
Die Berechnung der beiden Operationen soll auf den DSP48-Blöcken [40] erfolgen. Dabei handelt es sich um die DSP-Blöcke, die auf der Zielhardware für die effiziente Berechnung mit Ganzzahlen und Festkommazahlen verwendet werden. Die Multiplikationen können in diesen Hardware-Elementen mit einer maximalen Bitbreite von 25 Bit für den ersten Faktor und 18 Bit für den zweiten Faktor durchgeführt werden, sodass 43 Bit Ergebnisse entstehen. Durch Kaskadieren mehrerer DSP48-Blöcke können die Bitbreiten der Multiplikation noch weiter erhöht werden. Durch das FPGA-Design, das die Architektur und Funktionsweise der Recheneinheiten festlegt, wurde die Bitbreite der Multiplikationsergebnisse auf 32 Bit beschränkt. Die Bitbreiten der beiden Faktoren der Multiplikation können für die Festkommaquantisierung beliebig gewählt werden, solange sie 25 Bit nicht übersteigen. Die Additionsoperationen können in den DSP48-Blöcken mit einer maximalen Bitbreite von 48 Bit durchgeführt werden. Auch hierbei wird eine Begrenzung auf 32 Bit durch das implementierte FPGA-Design vorgenommen (vgl. [40]).

4.2.2 Wahl der Vorzeichenrepräsentation

Ein weiterer wichtiger Aspekt, mit dem sich im Rahmen der Festkommaquantisierung beschäftigt werden muss, ist die Wahl der Vorzeichenrepräsentation für die PLA-Berechnungen. Das heißt es muss entschieden werden, ob eine vorzeichenlose Kodierung der Eingabedaten, Stützstellen, Steigungen und y-Achsenabschnitte möglich ist oder ob eine vorzeichenbehaftete Kodierung erforderlich ist. Dies hat einen direkten Einfluss auf die Festkommaquantisierung, da bei vorzeichenloser Kodierung kein Zweierkomplement verwendet werden muss und das MSB für die Erhöhung der Genauigkeit der festkommaskalierten Ganzzahlen verwendet werden kann.
Um feststellen zu können, ob eine vorzeichenlose Kodierung möglich ist, müssen die Wertebereiche der vier PLA-Variablen und die Funktionswerte der Potenzfunktionen untersucht werden. Da die beiden Potenzfunktionen stets positiv sind, haben die meisten PLA-Variablen ebenfalls einen positiven Wertebereich. Die einzige Ausnahme bilden die Stei-

gungen der RSQRT-Funktion, deren Wertebereich stets negativ ist. Durch eine mathematische Umformung der allgemeinen PLA-Gleichung 2.7 lässt sich zeigen, dass die Berechnung dennoch ohne Vorzeichen durchgeführt werden kann. Zur Veranschaulichung wird die Gleichung (2.7) so angepasst, dass die Steigungen m mit einem negativen Vorzeichen dargestellt werden. Die Umformung kann dann wie folgt durchgeführt werden:

$$
\begin{aligned}
L_i(x) &= -m_i \cdot x + c_i \\
L_i(x) &= -(m_i \cdot x) + c_i \\
L_i(x) &= c_i - (m_i \cdot x)
\end{aligned}
$$

Die Umformung zeigt, dass statt der Multiplikation mit den vorzeichenbehafteten Steigungen eine vorzeichenlose Multiplikation mit den absoluten Steigungen möglich ist. Da der Funktionswert der Potenzfunktionen stets positiv ist, bleibt das Ergebnis der Subtraktion ebenfalls positiv. Folglich konnte gezeigt werden, dass für die Kodierung der PLA-Variablen und somit auch für die PLA-Berechnungen keine Vorzeichen beachtet werden muss. Das MSB kann daher zur Erhöhung der Genauigkeit der Festkommaquantisierung verwendet werden.

4.2.3 Methodik der Festkommaquantisierung

Das Ziel der Festkommaquantisierung besteht darin, die Fließkommaberechnungen der PLAs in INT32-Operationen umzuwandeln, um effizientere Ausführungen auf der FPGA-Hardware zu ermöglichen. In Abb. 4.9 ist das Vorgehen der Festkommaquantisierung anhand der PLA in einer GDN-Schicht dargestellt. Hierbei werden die Eingabedaten der PLA x_3 mit einem Skalierungsfaktor skaliert und in INT32-Werte umgewandelt. Dieser Prozess wird in der Abbildung als FestQuant bezeichnet. Die Festkommaquantisierung der PLA-Variablen wurde bereits vor der Inferenz auf die gleiche Weise durchgeführt. Durch die Quantisierung aller Bestandteile der PLAs kann die gesamte Berechnung mit festkommaskalierten Ganzzahlen durchgeführt werden. Nach der Berechnung werden die Ergebnisse mithilfe eines Skalierungsfaktors dequantisiert, damit die folgenden Berechnungen in FP32 durchgeführt werden können. Dieser Prozess wird in der Abbildung als FestDeQuant bezeichnet. Auf diese Weise kann die Auswirkung der Festkommaquantisierung auf die Kompressions- und Rekonstruktionsergebnisse isoliert betrachtet werden. Die gleiche Methodik gilt analog auch für die PLAs in den IGDN-Schichten. Im Folgenden wird

die Erstellung der Skalierungsfaktoren für Eingabedaten und PLA-Variablen der sechs 40-elementigen PLAs erläutert.

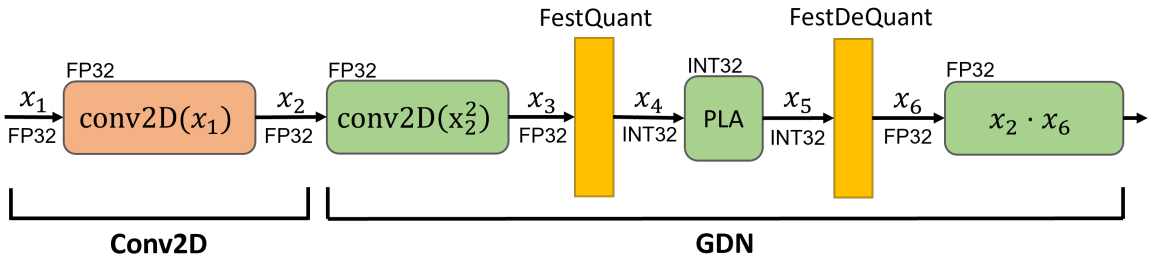


Abbildung 4.9: Schematische Darstellung der Methodik zur Festkommaquantisierung der PLAs anhand einer GDN-Schicht

Erstellung der Skalierungsfaktoren

Um eine Festkommaquantisierung dieser gesamten Berechnung zu realisieren, müssen die Eingaben, Steigungen, Stützstellen und y-Achsenabschnitte in Festkomma-Variablen umgewandelt werden. Dafür ist die Bestimmung geeigneter Skalierungsfaktoren 2^b erforderlich, mit deren Hilfe alle Zahlen im Wertebereich dieser Variablen in vorzeichenlose n-Bit-Festkommazahlen umgewandelt werden können. b repräsentiert in diesem Kontext die Anzahl der Fractionbits, die die Festkommazahlen der entsprechenden Variablen implizit abbilden können und ist somit ein Indikator für die Nachkommagenauigkeit. Für die Quantisierung jeder PLA werden die Skalierungsfaktoren: 2^{b_m} für die Steigungen, 2^{b_x} für die Stützstellen und die Eingaben sowie 2^{b_c} für die y-Achsenabschnitte benötigt. Da die Stützstellen und die Eingabedaten den gleichen Wertebereich haben, besitzen sie den gleichen Skalierungsfaktor.
Um die drei Skalierungsfaktoren jeder PLA zu berechnen, müssen die absoluten Maximalwerte ihrer Wertebereiche bestimmt und die Anzahl an Bits für die Festkommadarstellung der entsprechenden PLA-Variable gewählt werden. Anschließend kann mit der Gleichung (2.13) die Anzahl der Fractionbits b jeder Variablen berechnet und der Skalierungsfaktor abgeleitet werden. Da es nach der Erstellung der PLAs zu keinen Veränderungen in den Wertebereichen kommt, ist die Anzahl der Bits die einzige zu wählende Variable. Die Hardware gibt hierbei eine Begrenzung von 32 Bit für das Ergebnis der PLA-Berechnung vor. Unter der Berücksichtigung des einen Bits, das einen möglichen Überlauf bei der Additionsoperation abfängt, verbleiben 31 Bit, die auf die beiden Faktoren der Multiplikation

aufgeteilt werden können. Um eine möglichst gleichverteilte Aufteilung zu erreichen, werden 15 Bit für die Quantisierung der Steigungen und 16 Bit für die der Eingabedaten und Stützstellen verwendet. Da die y-Achsenabschnitte für eine fehlerfreie Addition die gleiche Skalierung wie das Produkt beider Faktoren benötigen, ergibt sich der Skalierungsfaktor aus $2^{b_m+b_x}$. Nach der Festlegung der zur Verfügung gestellten Bitanzahlen können die Fractionbits und somit die Skalierungsfaktoren mithilfe der Gleichung (2.13) berechnet werden. In Tabelle 4.2 sind die berechneten Skalierungsfaktoren für die sechs 40-elementigen PLAs dargestellt. Zusätzlich wurden die Wertebereiche abgebildet, damit die Berechnung der Skalierungsfaktoren transparenter ist. Für die Steigungen der RSQRT-Funktionen wurden die negativen Wertebereiche dargestellt, aber mit den absoluten Werten gerechnet.

	Steigungen		**Eingabedaten/ Stützstellen**		**y-Achsenabschnitte**	
	Wertebereich	2^{b_m}	Wertebereich	2^{b_x}	Wertebereich	2^{b_y}
RSQRT1	-9.02 - -1.00e-5	2^{12}	0.11 - 304.4	2^{7}	0.0879 - 3.99	2^{19}
RSQRT2	-1.13 - -1.71e-5	2^{16}	0.51 - 997.2	2^{6}	0.0487 - 1.98	2^{22}
RSQRT3	-0.19 - -4.90e-7	2^{19}	1.51 - 10798	2^{2}	0.0149 - 1.10	2^{21}
SQRT1	0.589 - 34.457	2^{11}	2.28e-6 - 0.74	2^{16}	0.0014 - 0.42	2^{27}
SQRT2	0.246 - 19.508	2^{12}	1.05e-6 - 4.25	2^{13}	0.0010 - 1.02	2^{25}
SQRT3	0.214 - 11.924	2^{13}	2.05e-5 - 5.63	2^{13}	0.0043 - 1.17	2^{26}

Tabelle 4.2: Wertebereiche und Skalierungsfaktoren der Steigungen, Eingabedaten und Stützstellen sowie der y-Achsenabschnitte für die drei RSQRT- und die drei SQRT-Approximationen

Die Skalierungsfaktoren der Steigungen liegen je nach PLA zwischen 2^{11} und 2^{19}. Das bedeutet, dass zwischen 11 und 19 Fractionbits in der festkommaskalierten Ganzzahldarstellung abgebildet werden können. Dies ermöglicht eine hohe Nachkommagenauigkeit und einen geringen Quantisierungsfehler. Im Gegensatz dazu haben die Eingaben, insbesondere die der RSQRT-Approximation, einen höheren ganzzahligen Anteil. Da bereits für die Darstellung des ganzzahligen Anteils mehr Bits benötigt werden, bleiben nur noch wenige Bits für die Darstellung der Nachkommastellen übrig. Infolgedessen haben sie kleinere Skalierungsfaktoren. Wie bereits erwähnt, ergeben sich die Skalierungsfaktoren der y-Achsenabschnitte aus dem Produkt der beiden anderen. Bei dieser Berechnung muss ein Sonderfall beachtet werden. Genau wie die anderen drei Variablen dürfen die festkommaskalierten Ganzzahlen der y-Achsenabschnitte maximal 31 Bit besitzen. Falls die Anzahl der Bits des ganzzahligen Anteils und die Anzahl der zu schiebenden Fractionbits $b_y = b_m + b_x$ zusammen die maximalen 31 Bit übersteigen, kann dies zu Überläufen in der Hardware führen. Um dies zu vermeiden, kann das Ergebnis der Multiplikation um einen geeigneten

Skalierungsfaktor zurückskaliert werden. Dadurch lässt sich ein kleinerer Skalierungsfaktor für die y-Achsenabschnitte der jeweiligen PLA verwenden, der zu keinen Überläufen führen kann. Dieser Sonderfall wurde bei der Festkommaquantisierung der sechs 40-elementigen PLAs überprüft, trat jedoch nicht auf. Der Grund dafür ist, dass die y-Achsenabschnitte nur über einen sehr geringen ganzzahligen Anteil verfügen, sodass je nach PLA Skalierungsfaktoren von 2^{29} bis 2^{31} genutzt werden können. Diese sind somit größer als die tatsächlich notwendigen Skalierungsfaktoren, die in Tabelle 4.2 dargestellt sind. Infolgedessen treten keine Überläufe bei der Festkommaquantisierung auf.

4.3 PTQ der Conv- und Deconv-Operationen

Das Encoder- und Decoder-Netz bestehen zusammen aus 14 Schichten, die jeweils eine convolutional oder deconvolutional Operation besitzen. Bei diesen Operationen handelt es sich um die rechenintensivsten Komponenten des Autoencoders. Um eine effiziente Ausführbarkeit des Autoencoders auf der FPGA-Hardware zu ermöglichen, ist es daher erforderlich, eine INT8-Quantisierung der Gewichts- und Aktivierungstensoren dieser Operationen durchzuführen. Die Quantisierung verbessert nicht nur die Recheneffizienz, sondern führt darüber hinaus zu einer Kompression des Modells, wodurch dessen Speicherbedarf reduziert und die Speicherressourcen der Hardware effizienter genutzt werden.
Da es sich beim verwendeten Autoencoder der CompressAI-Bibliothek um ein PyTorch-Modell handelt, erfolgt die Quantisierung der convolutional und deconvolutional Operationen in PyTorch. Als Quantisierungsmethode wird die PTQ verwendet. Mit der Begründung für die Auswahl dieser Methode wird sich im ersten Teil dieses Abschnitts beschäftigt. Im zweiten Teil wird die Methodik der INT8-Quantisierung bzw. der PTQ beschrieben.

4.3.1 Begründung der Quantisierungsmethode

In PyTorch existieren drei Methoden zur Quantisierung von neuronalen Netzwerken. Dabei handelt es sich um die dynamische Quantisierung, die Post Training Quantization (PTQ) und die Quantization Aware Training (QAT). Im Folgenden wird die Eignung dieser drei Methoden für die Quantisierung in dieser Arbeit beschrieben. Aus diesen Beschreibungen wird hervorgehen, dass die PTQ unter den gegebenen Umständen die geeignetste Quantisierungsmethode für die INT8-Quantisierung ist.

Ein wichtiger Umstand ist die Notwendigkeit, die verfügbare Rechenleistung der Zielhardware so effizient wie möglich zu nutzen. Daher sollte die Quantisierung vor der Inferenz des Autoencoders erfolgen, um die Anzahl der erforderlichen Quantisierungsberechnungen während der Inferenz möglichst gering zu halten. Das kann durch die dynamische Quantisierung allerdings nicht gewährleistet werden. Zwar werden hierbei die Gewichte vor der Inferenz quantisiert, allerdings erfolgt die Berechnung der QParameter und die Durchführung der Quantisierung der Eingabe- und Aktivierungstensoren während der Inferenz. Aus diesem Grund ist diese Quantisierungsmethode für diese Arbeit weniger gut geeignet.
Ein weiterer Umstand ist das Fehlen eines großen Datensatzes, der für das Training des Autoencoders verwendet werden kann. Für die Realisierung der QAT müsste jedoch ein Training des Autoencoders erfolgen, bei dem die Quantisierung der Gewichts-, Eingabe- und Aktivierungstensoren gelernt wird. Auch wenn einige Forschungsarbeiten gezeigt haben, dass mit dieser Quantisierungsmethode sehr gute Ergebnisse erzielt werden können, kann sie unter den gegebenen Umständen dieser Arbeit nicht verwendet werden.
Die Methode, die trotz der genannten Umstände verwendet werden kann, ist die PTQ. Der Grund dafür ist, dass bei dieser Methode zum einen die Gewichtstensoren vor der Inferenz quantisiert werden und zum anderen eine Berechnung der QParameter für die Eingabe- und Aktivierungstensoren vor der Inferenz durch einen Kalibrierungsdatensatz erfolgt. Hierfür kann der in Abschnitt 3.2 vorgestellte Kalibrierungsdatensatz verwendet werden. Da die Quantisierung der PTQ vorrangig vor der Inferenz ausgeführt werden kann, ohne dass ein Training des Autoencoders erforderlich ist, ist sie für die Quantisierung in dieser Arbeit besonders geeignet und wird im Folgenden verwendet.

4.3.2 Methodik der PTQ

In diesem Abschnitt wird die Methodik der PTQ detailliert beschrieben. Dabei orientiert sich die Beschreibung an den Schritten des allgemeinen Ablaufs einer PTQ, der in Abschnitt 2.4.3 dargestellt wurde. Begonnen wird mit der Festlegung der zu quantisierenden Bereiche und Schichten. Da PyTorch keine Fusion zwischen den convolutional Operationen und den Aktivierungsfunktionen des verwendeten Modells unterstützt, wird dieser Schritt übersprungen. Stattdessen wird direkt auf die verwendeten QConfigs eingegangen. Die kombinierte Beschreibung dieser beiden Schritte definiert die Kernaspekte der PTQ, die in dieser Arbeit angewendet wird. Das Vorgehen zur Bestimmung der QParameter aus dieser Definition und der Konvertierung des Modells kann Abschnitt 2.4.3 entnommen werden.

Festlegung der zu quantisierenden Bereiche und Schichten

In Abb. 4.10 ist eine schematische Darstellung des Autoencoders und der groben Quantisierungsbereiche der PTQ dargestellt. Die vorangestellten QuantStubs und nachgestellten DeQuantStubs kennzeichnen sowohl das gesamte Encoder-Netz als auch das gesamte Decoder-Netz als zu quantisierenden Bereich. Die De- und Requantisierung um die Entropie-Codierung (EC) im Bottleneck des Autoencoders wurde hierbei bewusst herbeigeführt, da sich im Rahmen dieser Arbeit nicht mit dessen Quantisierung beschäftigt wird.

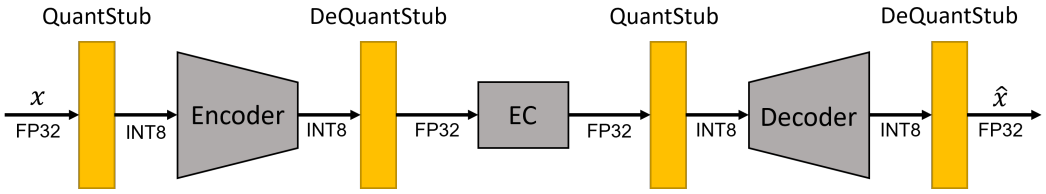


Abbildung 4.10: Grobe Quantisierungsbereiche der PTQ des Autoencoder

Durch diese Anordnung der Quant- und DeQuantStubs befinden sich sämtliche convolutional und deconvolutional Operationen beider Netze in Quantisierungsbereichen, sodass sie während der PTQ quantisiert werden. Zusätzlich sind auch die arithmetischen Operationen und die nichtlinearen Potenzfunktionen der GDN- und IGDN-Schichten in diesen Bereichen enthalten. Um die Multiplikationen und Quadrierungen in GDN- und IGDN-Schichten korrekt auf die INT8-Werte anwenden zu können, werden sie durch entsprechende Float Functionals (FFs) ersetzt. Diese führen Requantisierungen durch, um die Ergebnisse der beiden arithmetischen Berechnungen wieder in INT8-Werte zu requantisieren.
Anders verhält es sich mit den nichtlinearen Potenzfunktionen. Diese könnten zwar auch direkt zur Berechnung der INT8-Werte verwendet werden, allerdings führen die geringe Präzision und der eingeschränkte Wertebereich zu erheblichen Berechnungsfehlern. Um den Einfluss der INT8-Quantisierung der convolutionalen und deconvolutionalen Operationen auf die Rekonstruktionsqualität des Modells isoliert betrachten zu können, wird vor den Potenzfunktionen eine Dequantisierung in FP32 durchgeführt. Dadurch können die Berechnungen in den Potenzfunktionen mit einer erheblich höheren Präzision realisiert werden, sodass keine zusätzlichen Fehler zu denen der quantisierten Operationen entstehen. Die Ergebnisse werden nach der Berechnung in INT8 requantisiert. In Abb. 4.11 sind obige Ausführungen anhand einer conv2D- und einer GDN-Schicht dargestellt. Die gleiche Vorgehensweise gilt auch für die convTranpose2D- und IGDN-Schichten.

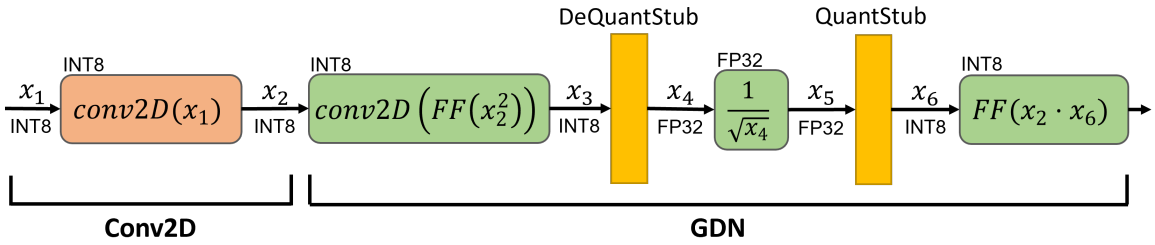


Abbildung 4.11: Schematische Darstellung der feineren Quantisierungsbereiche der PTQ an einer conv2D- und einer GDN-Schicht

Ein Problem bei der Dequantisierung vor den Potenzfunktionen ist das Auftreten von Nullwerten in den dequantisierten FP32-Tensoren x_4. Diese können durch Quantisierungsfehler entstehen und führen zu mathematischen Fehlern bei der Berechnung der RSQRT-Funktionen. Um dieses Problem zu lösen, können die Nullwerte durch den kleinsten Wert des Tensors ersetzt werden, der ungleich null ist. Dieser Ansatz ist dynamisch, da der kleinste Wert vom jeweiligen Tensor abhängt. Ein alternativer Ansatz wäre die Addition des FP32-Tensors mit einem kleinen konstanten Faktor ϵ. Testversuche haben jedoch gezeigt, dass durch den dynamischen Ansatz bessere Ergebnisse erzielt werden können. Zudem zeigte sich, dass bessere Rekonstruktionsqualitäten erreicht werden können, wenn derselbe Ansatz zur Behandlung von Nullwerten in den dequantisierten Eingabetensoren der SQRT-Funktionen in den IGDN-Schichten angewendet wird.

Festlegung der QConfigs

Nachdem die zu quantisierenden Bereiche des Autoencoders gewählt wurden, muss anschließend entschieden werden, welche QConfigs für die Gewichts- und Aktivierungstensoren der convolutional und deconvolutional Operationen verwendet werden. In PyTorch stehen hierbei zwei Standard-QConfigs zur Verfügung. Dabei handelt es sich um die QConfigs: FBGEMM und QNNPACK. Diese sind in Tabelle 4.3 vergleichend dargestellt.

Festlegung der QConfigs

Nachdem die zu quantisierenden Bereiche des Autoencoders gewählt wurden, muss anschließend entschieden werden, welche QConfigs für die Gewichts- und Aktivierungstensoren der convolutional und deconvolutional Operationen verwendet werden. In PyTorch ste-

hen hierbei zwei Standard-QConfigs zur Verfügung. Dabei handelt es sich um die QConfigs: FBGEMM und QNNPACK. Diese sind in Tabelle 4.3 vergleichend dargestellt.

Kriterium	**QNNPACK**		**FBGEMM**	
	Gewichte	**Aktivierungen**	**Gewichte**	**Aktivierungen**
Schema	Symmetrisch	Asymmetrisch	Symmetrisch	Asymmetrisch
Granularität	per-Tensor	per-Tensor	per-Channel	per-Tensor
Observer	MinMax	MovingAvgMinMax	PerChannelMinMax	MovingAvgMinMax

Tabelle 4.3: Vergleich der Standard-QConfigs in PyTorch

Die in der FBGEMM-Konfiguration enthaltenen QConfigs für Gewichts- und Aktivierungstensoren beinhalten exakt die Parameter, die in der Literatur für die Quantisierung empfohlen werden. Daher stellt diese QConfig aus theoretischer Sicht eine sehr vielversprechende Option dar, die wahrscheinlich zu guten Quantisierungsergebnissen führen könnte. Allerdings zeigte sich in ersten Quantisierungsversuchen, dass PyTorch derzeit keine per-Channel-Quantisierung der `convTranspose2D`-Operationen unterstützt. Aus diesem Grund kann die FBGEMM-Konfiguration nicht für die Quantisierung des Autoencoders in dieser Arbeit verwendet werden. Stattdessen wird die Quantisierung der convolutional und deconvolutional Operationen mithilfe der QNNPACK-Konfiguration definiert. Dazu wird in PyTorch jeder Operation die QNNPACK-Konfiguration zugewiesen.

4.4 Integration zur vollständigen Quantisierung

In diesem Abschnitt werden die festkommaquantisierten, approximierten Potenzfunktionen der GDN- und IGDN-Schichten in den Autoencoder mit den quantisierten convolutional und deconvolutional Operationen integriert. Auf diese Weise wird eine vollständige Quantisierung des Encoder- und Decoder-Netzes ermöglicht.

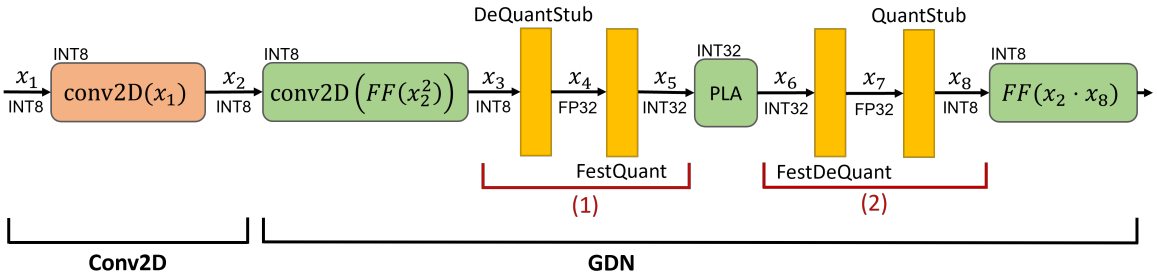


Abbildung 4.12: Schematische Darstellung der Integration der festkommaquantisierten PLAs in den Autoencoder mit quantisierten convolutional und deconvolutional Operationen, am Beispiel einer conv2D- und einer GDN-Schicht

In Abb. 4.12 wird die Methodik zum Erreichen einer vollständigen Quantisierung anhand einer GDN-Schicht im Encoder dargestellt. Es wird deutlich, dass die vollständige Quantisierung durch die Integration der festkommaquantisierten PLAs in den dequantisierten Bereich der PTQ erreicht wird. Im Schritt (1) des Integrationsprozesses, wird der INT8-Aktivierungstensor x_3 durch einen DeQuantStub der PTQ in einen FP32-Tensor x_4 dequantisiert. Daraufhin erfolgt die Festkommaquantisierung von x_4 durch die Anwendung eines Skalierungsfaktors (FestQuant). Anschließend kann die festkommaquantisierte PLA-Berechnung durchgeführt werden. Nach der Berechnung erfolgt der Schritt (2) der Integration. Hierbei wird der festkommaquantisierte Tensor x_6 in einen FP32-Tensor dequantisiert. Zuletzt wird der dequantisierte Tensor x_7 mithilfe der QParameter des QuantStubs der PTQ in einen INT8-Tensor transformiert, sodass die nachfolgenden Berechnungen in INT8 durchgeführt werden können. Das gleiche Vorgehen gilt analog für die vollständige Quantisierung der convTranspose2D- und IGDN-Schichten im Decoder.

Die Anwendung der PLAs und der Skalierungsfaktoren der Festkommaquantisierung auf den dequantisierten Aktivierungstensor x_4 ist aufgrund der Robustheit von convolutional und deconvolutional Operationen gegenüber der INT8-Quantisierung problemlos möglich. Diese Operationen können in den meisten Fällen trotz der Quantisierung stabil arbeiten, wodurch der dequantisierte FP32-Tensor und der ursprüngliche FP32-Aktivierungstensor im unquantisierten Modell nahezu den gleichen Wertebereich besitzen. Die einzigen kleinen Abweichungen entstehen durch den auftretenden Quantisierungsfehler. Da die Approximationen und Festkommaquantisierungen auf der Grundlage der Wertebereiche der ursprünglichen FP32-Aktivierungstensoren erstellt wurden, können sie demnach auch auf die dequantisierten FP32-Tensoren angewendet werden.
Die Requantisierung des dequantisierten Ergebnistensors x_7 ist aus einem ähnlichen Grund

möglich. Die Ergebnisse der reinen Potenzfunktionsberechnung und die der festkommaquantisierten PLA-Berechnungen weichen im Optimalfall nur geringfügig voneinander ab. Folglich haben sie einen ähnlichen Wertebereich, der sich nur minimal durch den auftretenden Approximations- und Quantisierungsfehler unterscheidet. Da die QParameter des QuantStubs für die Wertebereiche der Ergebnistensoren der reinen Potenzfunktionsberechnungen erstellt wurden, können sie auch für die dequantisierten Tensoren x_7 der festkommaquantisierten PLA-Berechnung verwendet werden.

Bei der Implementierung auf der Zielhardware wird es möglich sein, die De- und Requantisierung direkt zwischen INT8 und INT32 durchzuführen, ohne dass der FP32-Zwischenschritt erforderlich ist. Dazu müssen die QParameter der QuantStubs und DeQuantStubs mithilfe der Skalierungsfaktoren der entsprechenden Festkommaquantisierung quantisiert werden. Da die Festkommaquantisierung der QParameter in PyTorch nicht unterstützt wird, werden die Ergebnisse in dieser Arbeit allerdings mit den FP32-Zwischenschritten erzeugt. Da jedoch angenommen werden kann, dass der Quantisierungsfehler durch die Festkommaquantisierung der QParameter gering ist, sind die Resultate dennoch repräsentativ für die Ergebnisse, die durch die spätere Hardware-Implementierung erreicht werden können.

5 Auswertung

In diesem Abschnitt werden die in Kapitel 4 beschriebenen Methodiken auf das Encoder- und Decoder-Netz des Autoencoders angewendet. Auf diese Weise werden zusammen mit dem originalen Autoencoder fünf Modelle mit unterschiedlichem Approximations- und Quantisierungsgrad erstellt, die in Tabelle 5.1 dargestellt sind. In der Tabelle ist ein Haken gesetzt, wenn die jeweilige Methodik angewendet wurde, und ein Kreuz, wenn sie nicht angewendet wurde. Die Vielzahl der getesteten Modelle dient dazu, die Auswirkung der einzelnen Methodiken auf die Rekonstruktionsqualität des Autoencoders systematisch untersuchen zu können. Dadurch kann ermittelt werden, bei welcher der Methodiken das größte Optimierungspotenzial besteht.

Modell	**Approximation**	**Festkommaquantisierung**	**PTQ**
OriginalModell	✗	✗	✗
FP32ApproxModell	✓	✗	✗
QuantApproxModell	✓	✓	✗
QuantModell	✗	✗	✓
FullQuantModell	✓	✓	✓

Tabelle 5.1: Überblick über die zu vergleichenden Modelle und ihre Methodiken

Das OriginalModell ist der FP32-Autoencoder, an dem keine Veränderungen vorgenommen wurden. Beim FP32ApproxModell handelt es sich um das Modell, in dem gemäß Abschnitt 4.1 eine lineare FP32-Approximation der Potenzfunktionen durchgeführt wurde. Im QuantApproxModell erfolgte die Festkommaquantisierung der 40-elementigen PLAs. Das QuantModell repräsentiert die PTQ der convolutional und deconvolutional Operationen in den Netzschichten, ohne dass eine Quantisierung der Potenzfunktionen vorgenommen wurde. Schließlich kombiniert das FullQuantModell die quantisierten Operationen und festkommaquantisierten linearen Approximationen, um eine vollständige Quantisierung des Encoder- und Decoder-Netzes zu erreichen.
Im Folgenden werden diese Modelle auf dem Evaluierungsdatensatz getestet. Dabei werden die Kompressionsraten in der BPP-Metrik und die Rekonstruktionsqualitäten in den PSNR- und MS-SSIM-Metriken für alle Modelle ermittelt und dargestellt. Anschließend werden die Ergebnisse des FullQuantModells mit den Missionsanforderungen verglichen, um zu prüfen, ob die Anforderungen erfüllt werden oder bei welchen Methodiken eine Optimierung notwendig ist.

5.1 Darstellung der Ergebnisse

Wie in Tabelle 5.2 zu erkennen, gibt es einen signifikanten Unterschied zwischen den Kompressionsraten und Rekonstruktionsqualitäten der Modelle. Die Ergebnisse zeigen, dass es zwischen dem OriginalModell und dem FullQuantModell zu einem signifikanten Abfall der PSNR-Bewertung kam, während die MS-SSIM-Bewertungen deutlich stabiler geblieben sind. In Bezug auf die BPP-Metrik zeigen sich keine großen Veränderungen, was zeigt, dass die Quantisierung die Kompressionseffizienz nicht stark beeinflusst hat. Im Folgenden erfolgt eine prozessuale Darstellung der Ergebnisse der einzelnen Modelle.

Modell	PSNR [dB]	MS-SSIM [%]	BPP
OriginalModell	36.67	95.7	0.146
FP32ApproxModell	36.34	95.3	0.147
QuantApproxModell	36.14	95.2	0.147
QuantModell	20.89	91.3	0.148
FullQuantModell	21.40	91.7	0.149

Tabelle 5.2: Vergleich der Modelle hinsichtlich PSNR, MS-SSIM und BPP

Die Ergebnisse des FP32ApproxModells mit den sechs 40-elementigen PLAs wurden bereits in Abschnitt 4.1.3 präsentiert. Die Approximationen führen zu einem Verlust in der Rekonstruktionsgenauigkeit von 0.33 dB PSNR und 0.4 % MS-SSIM und zu einer schlechteren Kompression von 0.001 BPP, was einem Unterschied von lediglich 0.68 % entspricht. Mit dem FP32ApproxModell können demnach ähnlich gute Ergebnisse erzielt werden wie mit dem OriginalModell. Durch den Quantisierungsfehler der Festkommaquantisierung der linearen Approximationen kam es im QuantApproxModell zu einer zusätzlichen Abnahme der Rekonstruktionsqualität von 0.2 dB PSNR und 0.1 % MS-SSIM. Die Kompressionseffizienz wurde allerdings nicht beeinflusst. Insgesamt führte die Approximation und Quantisierung der Potenzfunktionen zu einem Verlust von 0.53 dB PSNR und 0.5 % MS-SSIM, sowie einer schlechteren Kompressionseffizienz von 0.001 BPP gegenüber dem OriginalModell. In Abb. 5.1 sind zur visuellen Veranschaulichung dieser Werte die Rekonstruktionen eines Bildausschnitts aus einem Bild des Evaluierungsdatensatzes vom OriginalModell und vom QuantApproxModell dargestellt. Durch die Abbildung wird deutlich, dass beide Modelle eine nahezu identische Rekonstruktion ermöglichen. Bei sehr genauer Betrachtung von Abb. 5.1b sind leichte Verdunkelungen an den Rändern der Kacheln zu erkennen, in die das große Bild des Datensatzes während der Inferenz zerlegt wird. Darüber hinaus treten leichte Pixelunterschiede auf, die mit dem bloßen Auge kaum wahrnehmbar sind. Diese

(a) OriginalModell (b) QuantApproxModell

Abbildung 5.1: Rekonstruierte Bildausschnitte des OriginalModell und QuantApproxModell

beiden Abweichungen sind allerdings nicht signifikant und führen daher nur zu einem sehr geringen Rekonstruktionsfehler.

Die PTQ der convolutional und deconvolutional Operationen des Autoencoders wirkt sich dagegen deutlich stärker auf die Rekonstruktionsqualität des Autoencoders aus. Hierbei sinkt der PSNR-Wert auf 20.89 dB ab und der MS-SSIM-Wert fällt auf 91.3 %. Die Kompressionseffizienz wird allerdings kaum beeinflusst, da im Vergleich zum OriginalModell lediglich 0.002 BPP mehr für die Kodierung der komprimierten Bilder benötigt wird. In Abb. 5.2 sind die Auswirkungen der PTQ auf die Rekonstruktionen an einem Bildausschnitt eines Bildes des Evaluierungsdatensatzes dargestellt. Das rekonstruierte Bild des QuantModells (Abb. 5.2b) weist im Gegensatz zum rekonstruierten Bild des OriginalModell (Abb. 5.2a) deutliche Farbverschiebungen auf. Außerdem sind in Abb. 5.2b die Ränder an den Übergängen zwischen den kleinen Kacheln sehr markant. Das führt zu einem Kachelmuster im gesamten Bild des QuantModells. Sowohl die Farbverschiebung als auch die Randartefakte sind die Ursache für den Abfall in der PSNR-Bewertung. Andererseits sind die niedrig- und hochfrequenten Details in Abb. 5.2b weitgehend erhalten geblieben. Das heißt kleinere Bilddetails, wie z.B. Kanten im Bild oder der Kies oben rechts in der Ecke, sind vom Betrachter gut zu erkennen. Aus diesem Grund ist die MS-SSIM-Bewertung weniger stark abgefallen.

Das FullQuantModell zeigt eine leichte Verbesserung der Kompressions- und Rekonstruktionswerte gegenüber dem QuantModell. Eine Erklärung dafür könnte sein, dass die Quantisierungsfehler des QuantModells und die Quantisierungs- und Approximationsfehler des

(a) OriginalModell (b) QuantModell

Abbildung 5.2: Rekonstruierte Bildausschnitte des OriginalModell und QuantModell

QuantApproxModells gegeneinander wirken und sich so gegenseitig aufheben. Dies muss in einer späteren Arbeit genauer untersucht werden. Dennoch ist zu erkennen, dass es im Vergleich zum OriginalModell zu einem deutlichen Qualitätsverlust der Rekonstruktion von 15.27 dB PSNR und 4 % MS-SSIM gekommen ist. Dennoch kommt es hier nur zu einer vernachlässigbar schlechteren Kompressionseffizienz von 0.003 BPP, was einem Unterschied von lediglich 2 % entspricht. Die Rekonstruktionsqualität des FullQuantModells ist gegenüber der des QuantModells leicht verbessert, jedoch ist diese Verbesserung nicht signifikant genug, um einen visuellen Unterschied wahrzunehmen. Daher ist eine visuelle Darstellung der Unterschiede nicht zielführend.

5.2 Interpretation der Ergebnisse

Die Missionsanforderungen legen einen maximalen Rekonstruktionsfehler des FullQuantModells gegenüber des OriginalModells fest. Dieser liegt bei 8 dB PSNR und 6 % MS-SSIM. Durch das erstellte FullQuantModell konnte eine MS-SSIM-Bewertung von 91.7 % erreicht werden, was einer Abnahme von 4 % gegenüber dem OriginalModell entspricht und somit die Missionsanforderungen hinsichtlich der Metrik vollständig erfüllt. Bezüglich der PSNR-Bewertung wurden 21.4 dB erreicht, was einen stärkeren Verlust von 15.27 dB bedeutet. Durch die Analyse der einzelnen Modelle bzw. Methodiken ist es möglich, die Hauptursache dieses Qualitätsverlustes auf die PTQ einzugrenzen. Eine Optimierung der PTQ hat daher das größte Potenzial, die Rekonstruktionsqualität zu steigern und damit die PSNR-

Vorgaben vollständig zu erfüllen. Im folgenden Kapitel wird hierzu ein vielversprechender Optimierungsansatz vorgestellt.
Insgesamt konnten durch die dargelegten Methodiken eine vollständige Quantisierung des Encoder- und Decoder-Netzes realisiert werden, die bereits wesentliche Missionsanforderungen erfüllt. Das FullQuantModell bietet eine gute Grundlage, um eine effiziente Ausführbarkeit auf der FPGA-Zielhardware zu ermöglichen und stellt gleichzeitig eine solide Ausgangsbasis für zukünftige Optimierungen dar.

6 Optimierungsansatz

Ein vielversprechender Ansatz zur Optimierung der durchgeführten PTQ besteht darin, dass die sensiblen convolutional und deconvolutional Operationen im Autoencoder mit einer höheren Bitbreite quantisiert werden. Statt einer INT8-Quantisierung könnte eine INT16-Quantisierung der sensiblen Operationen durchgeführt werden. Der Begriff Sensibilität beschreibt in diesem Kontext, wie stark der Informationsverlust in den Gewichten der Operationen durch die Reduzierung der Genauigkeit von 32 Bit auf 8 Bit ausfällt. Operationen, die besonders sensibel auf die Quantisierung reagieren, können ihre 32 Bit-Gewichte nicht adäquat in 8 Bit abbilden. Dies führt zu größeren Quantisierungsfehlern, die sich negativ auf die Rekonstruktionsqualität auswirken. Da PyTorch zum Zeitpunkt dieser Arbeit noch keine höheren Bitbreiten bei der Quantisierung unterstützt und eine manuelle Implementierung aus Zeitgründen nicht möglich ist, werden in diesem Kapitel zwei Voruntersuchungen durchgeführt, in denen die Sensibilität bestimmter Komponenten des Autoencoders auf die INT8-Quantisierung bzw. PTQ analysiert wird.
In der ersten Untersuchung wird die individuelle Sensibilität des Encoder- und des Decoder-Netzes gegenüber der durchgeführten INT8-Quantisierung bewertet. Durch diese erste Untersuchung kann gezeigt werden, wie sich die individuelle PTQ der beiden Netze auf die Rekonstruktionsqualität auswirkt. Die zweite Untersuchung analysiert die Auswirkung der INT8-Quantisierung auf die einzelnen convolutional und deconvolutional Operationen in den Netzschichten des Encoder- und Decoder-Netzes. Diese Untersuchung identifiziert die Operationen, deren Quantisierung zu einer besonders starken Verschlechterung der Rekonstruktionsqualität führt.
Die gewonnenen Erkenntnisse beider Untersuchungen, insbesondere die Identifikation der sensiblen Komponenten des Autoencoders, stellen eine wichtige Grundlage für die Umsetzung des vorgeschlagenen Optimierungsansatzes dar. Künftige Forschungsarbeiten können sich somit gezielt auf die herausgestellten Komponenten fokussieren und die Veränderung der Rekonstruktionsqualität untersuchen.

6.1 Methodisches Vorgehen

Die Untersuchung der separaten PTQ des Encoder- und Decoder-Netzes ist wie folgt strukturiert:

1. Es werden zwei Modelle erstellt: eins mit dem quantisierten Encoder-Netz und eins mit dem quantisierten Decoder-Netz. Beim Modell mit dem quantisierten Encoder werden die convolutional Operationen in den sieben Encoderschichten quantisiert, während der Decoder unquantisiert bleibt. Beim zweiten Modell erfolgt die Quantisierung der sieben convolutional und deconvolutional Operationen im Decoder, während der Encoder unquantisiert bleibt. Die Quantisierungen der convolutional und deconvolutional Operationen erfolgen mittels der PTQ, die in Abschnitt 4.3 beschrieben wurde.

2. Die Kompressionseffizienz und Rekonstruktionsgenauigkeit beider Modelle wird auf den sechs Bildern des Evaluierungsdatensatzes gemessen. Die genaue Durchführung dieser Messung ist in Abschnitt 3.2 beschrieben. Die Bewertung der Rekonstruktionen erfolgt anhand der PSNR- und MS-SSIM-Metrik, während die Kompressionen in der BPP-Metrik gemessen werden.

3. Als Referenzmodell werden das OriginalModell und das QuantModell verwendet. Diese Abkürzungen stehen für den unquantisierten Autoencoder und den Autoencoder, bei dem alle convolutional und deconvolutional Operationen mithilfe der vorgestellten PTQ quantisiert wurden. Die Ergebnisse der beiden erstellten Modelle werden mit denen der Referenzmodelle verglichen. Daraus kann ermittelt werden, ob der Encoder oder der Decoder sensibler auf die PTQ reagiert. Darüber hinaus sollte dadurch erkennbar werden, welches der beiden Netze des Autoencoders einen größeren Anteil am Gesamtverlust der Rekonstruktionsqualität durch die PTQ des gesamten Modells hat.

Bei der zweiten Untersuchung wird die Sensibilität sämtlicher convolutional und deconvolutional Operationen individuell untersucht. Hierbei wird wie folgt vorgegangen:

1. Jede der 14 Schichten des Autoencoders enthält eine einzige convolutional oder deconvolutional Operation. Zur besseren Übersicht werden diese Schichten von 1 bis 14 nummeriert, wobei Index 1 der convolutional Operation der ersten conv2D-Schicht entspricht, Index 2 der convolutional Operation in der ersten GDN-Schicht, usw. Die vollständige Zuweisung der Indizes zu den Schichten bzw. Operationen des

Autoencoders ist in Tabelle 6.1 dargestellt und lässt sich durch die Abb. 3.1 visuell nachvollziehen. Insgesamt werden 14 teilweise quantisierte Modelle erstellt, in denen jeweils nur eine convolutional oder deconvolutional Operation quantisiert wird. Die restlichen Operationen bleiben unquantisiert. Für die Quantisierung wird auch hier das Vorgehen der PTQ verwendet, das in Abschnitt 4.3 beschrieben wurde.

2. Für jedes teilweise quantisierte Modell wird die Kompressionseffizienz und die Rekonstruktionsqualität mithilfe des Evaluierungsdatensatzes bestimmt. Das genaue Vorgehen bei der Messung ist in Abschnitt 3.2 beschrieben. Auch hierbei werden die PSNR-, MS-SSIM- und die BPP-Metrik verwendet.

3. Der Vergleich zwischen den Ergebnissen des nicht quantisierten Autoencoders und denen der teilweise quantisierten Modelle wird aufzeigen, welche convolutional und deconvolutional Operationen besonders sensibel gegenüber der PTQ sind.

Index	**Schicht**	**Operation**
1	Erste conv2D	Convolution
2	Erste GDN	Convolution
3	Zweite conv2D	Convolution
4	Zweite GDN	Convolution
5	Dritte conv2D	Convolution
6	Dritte GDN	Convolution
7	Vierte conv2D	Convolution
8	Erste convTranspose2D	Deconvolution
9	Erste IGDN	Convolution
10	Zweite convTranspose2D	Deconvolution
11	Zweite IGDN	Convolution
12	Dritte convTranspose2D	Deconvolution
13	Dritte IGDN	Convolution
14	Vierte convTranspose2D	Deconvolution

Tabelle 6.1: Zuweisung der Indizes zu den conv- und deconv-Operationen im Autoencoder

6.2 Auswertung

In Tabelle 6.2 sind die Ergebnisse der ersten Untersuchung dargestellt. Der Autoencoder mit dem quantisierten Encoder-Netz erreichte auf dem Evaluationsdatensatz eine Rekonstruktionsgenauigkeit von 21.25 dB PSNR und 92.5 % MS-SSIM sowie eine Kompressionseffizienz von 0.148 BPP. Unter den gleichen Testbedingungen erreichte das Modell mit dem quantisierten Decoder 31.08 dB PSNR, 93.1 % MS-SSIM und 0.146 BPP. Der Vergleich mit den Ergebnissen des OriginalModells zeigt, dass das Encoder-Netzwerk einen erheblichen Rekonstruktionsverlust von 15.42 dB PSNR und 3.2 % MS-SSIM aufweist. Die Abnahme in der PSNR- und MS-SSIM-Bewertung ist somit 2.75-mal und 1.25-mal stärker als bei der Quantisierung des Decoders. Dies verdeutlicht, dass es im Encoder bestimmte convolutional Operationen gibt, die sehr sensibel auf die INT8-Quantisierung reagieren. Die Verschlechterung von 0.002 BPP ist hingegen weniger signifikant. Dennoch zeigte die erste Untersuchung, dass der größte Teil des Rekonstruktionsfehlers des QuantModells auf die Quantisierung des Encoder-Netzwerks zurückzuführen ist.

Modell	PSNR [dB]	MS-SSIM [%]	BPP
OriginalModell	36.67	95.7	0.146
QuantModell	20.89	91.3	0.148
PTQ Encoder	21.25	92.5	0.148
PTQ Decoder	31.08	93.1	0.146

Tabelle 6.2: Vergleich des Autoencoders mit quantisiertem Encoder und mit quantisiertem Decoder zum OriginalModell und QuantModell

Durch die anschließende Untersuchung werden im Folgenden die genauen Schichten identifiziert, bei denen die PTQ zu größeren Verlusten in der Rekonstruktionsqualität des Autoencoders führt. Die Ergebnisse der zweiten Untersuchung sind in Abb. 6.1 dargestellt. Auf der x-Achse des Diagramms sind die Indizes der Schichten nummeriert von 1 bis 14 dargestellt. Jeder Index entspricht somit einem der 14 teilweise quantisierten Modelle, in dem jeweils nur die convolutional oder deconvolutional Operation der Schicht mit dem entsprechenden Index quantisiert wurde. Der Index bei $x = 1$ entspricht beispielsweise dem teilweise quantisierten Modell, in dem nur die convolutional Operation der ersten conv2D-Schicht quantisiert wurde.

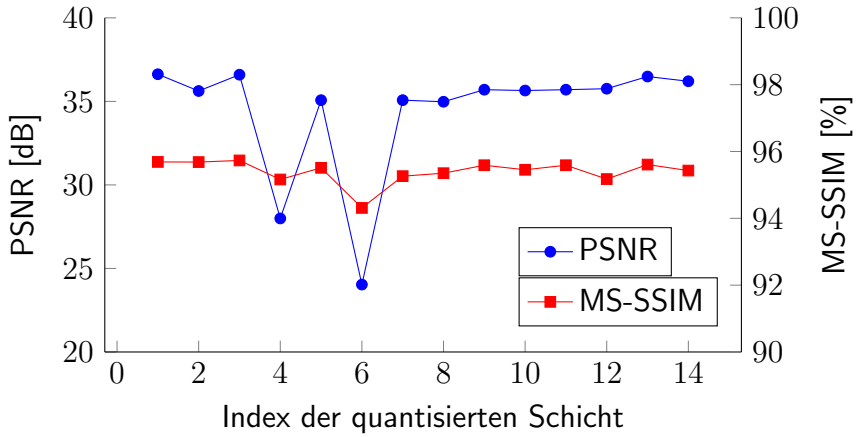


Abbildung 6.1: Rekonstruktionsqualität bei INT8-Quantisierung einzelner conv- u. deconv-Operationen

Auf den y-Achsen sind die Rekonstruktionsmetriken PSNR und MS-SSIM aufgetragen. Für jedes der 14 Modelle ist die durchschnittliche PSNR- und MS-SSIM-Bewertung auf dem Evaluierungsdatensatz dargestellt. Die PTQ der convolutional oder deconvolutional Operationen führt in den meisten Schichten zu einer geringen Reduktion der Rekonstruktionsqualität von weniger als 2 dB PSNR und 0.5 % MS-SSIM. Im Gegensatz dazu führen die PTQ der convolutional Operationen in der zweiten und dritten GDN-Schicht (Index 4 und 6, siehe Tabelle 6.1) des Encoders zu einem erheblichen Verlust der Rekonstruktionsgenauigkeit des Autoencoders. Die Reduktion der Genauigkeit in der zweiten GDN-Schicht führt zu einer Abnahme von 8.68 dB PSNR und 0.54 % MS-SSIM. Noch schwerwiegender ist der Verlust in der dritten GDN-Schicht mit 12.64 dB PSNR und 1.39 % MS-SSIM. Die convolutional Operationen dieser beiden Schichten sind besonders sensibel gegenüber der INT8-Quantisierung. Eine mögliche Erklärung könnte in der Verteilung der Gewichts- oder Eingabetensoren liegen. Falls diese große Ausreißer aufweisen, kann dies die Qualität der Quantisierung negativ beeinflussen. Da die Quantisierung den gesamten Wertebereich abdecken muss, werden die Ausreißer mit einbezogen, was zu einer gröberen Abstufung bei den häufigeren, niedrigeren Werten führt. Dies könnte den hohen Rekonstruktionsfehler erklären. Um die eindeutige Ursache zu bestimmen, sind detaillierte Tests erforderlich. Bei der Messung der Kompressionseffizienz unterschieden sich die Modelle um weniger als 0.002 BPP, daher wird keine visuelle Darstellung vorgenommen.

Zusammenfassend zeigte die erste Untersuchung, dass der Encoder sehr sensibel auf die PTQ reagiert, während das Decoder-Netz robuster gegenüber der Quantisierung ist. Durch die zweite Untersuchung konnten zwei Schichten identifiziert werden, deren convolutional

Operationen durch die PTQ zu erheblichen Verlusten der Rekonstruktionsqualität geführt haben. Dieses gewonnene Wissen der Voruntersuchungen kann für die Umsetzung des vorgestellten Optimierungsansatzes in zukünftigen Forschungsarbeiten verwendet werden. Die praktische Umsetzung stellt eine vielversprechende Möglichkeit dar, um die Missionsanforderungen vollständig erfüllen zu können.

7 Zusammenfassung

Das Ziel dieser Arbeit war die vollständige Quantisierung des Encoder- und Decoder-Netzes des vorgegebenen Autoencoders unter Einhaltung der festgelegten Missionsanforderungen von 8 dB PSNR und 6 % MS-SSIM. Dadurch soll eine effiziente Ausführbarkeit des Autoencoders auf der Xilinx Zynq 7020 FPGA-Hardware ermöglicht werden.

Hierzu wurde sich mit der Erstellung von Methodiken beschäftigt, die die Umsetzung der einzelnen Teilaspekte der vollständigen Quantisierung beschreiben. Zunächst erfolgte dazu die Beschreibung der Erstellung von PLAs zur linearen Approximation der nichtlinearen Potenzfunktionen im Autoencoder. Daraufhin wurde die Methodik zur Festkommaquantisierung der linearen Approximationen aufgezeigt. Anschließend wurde das Vorgehen der Quantisierung der convolutional und deconvolutional Operationen des Encoder- und Decoder-Netzes mittels PTQ beschrieben. Zuletzt wurde dargelegt, wie eine Kombination der festkommaquantisierten, linearen Approximationen und der PTQ erfolgen kann, um eine vollständige Quantisierung des Encoders und Decoders zu ermöglichen. Um den Einfluss jedes Teilaspekts auf die Kompressions- und Rekonstruktionsleistung des Modells zu bewerten, wurden diese einzeln angewendet und auf einem Evaluierungsdatensatz getestet. Auf diese Weise konnte ermittelt werden, bei welchen Aspekten Optimierungen notwendig sind. Darüber hinaus wurde eine Evaluierung des Autoencoders mit dem vollständig quantisierten Encoder und Decoder durchgeführt, um zu überprüfen, ob die Missionsanforderungen erfüllt werden konnten.

Bei den Tests stellte sich heraus, dass sowohl die vorgeschlagenen Approximationen als auch die durchgeführten 32 Bit Festkommaquantisierungen der linearen Approximation nur geringe Verluste in der Rekonstruktionsgenauigkeit verursachen. Die PTQ der convolutional und deconvolutional Operationen hingegen führte zu einer erheblich höheren Abnahme der Rekonstruktionsgenauigkeit des Modells. Trotzdem konnte der Autoencoder mit dem vollständig quantisierten Encoder und Decoder die Missionsanforderungen hinsichtlich der MS-SSIM-Metrik mit einem Verlust von 4 % einhalten. Aufgrund des hohen Verlustes durch die PTQ war eine Einhaltung der Missionsanforderungen hinsichtlich der PSNR-Metrik mit 15.27 dB Verlust nicht möglich. Es konnte jedoch ein vielversprechender Optimierungsansatz aufgezeigt werden, der darin besteht, höhere Bitbreiten für die Quantisierung bestimmter convolutional Operationen zu verwenden, die sehr sensibel gegenüber der durch-

geführten INT8-Quantisierung sind. Durch die Umsetzung dieses Optimierungsansatzes im Rahmen zukünftiger Forschungsarbeiten im ScOSA-Projekt wird es voraussichtlich möglich sein, die Rekonstruktionsqualität zu optimieren und damit auch die PSNR-Anforderungen zu erfüllen.

Insgesamt konnte das Ziel der vollständigen Quantisierung des Encoders und Decoders durch die vorgestellten Methodiken erreicht werden, die bereits ohne weitere Optimierungen einen Teil der Missionsanforderungen erfüllt. Damit stellt die in dieser Arbeit durchgeführte Quantisierung des Autoencoders eine gute Grundlage dar, die im Rahmen von weiteren Forschungsarbeiten im Projekt ScOSA optimiert werden kann.

8 Ausblick

Die in dieser Arbeit vorgestellten Methodiken zur Approximation und Quantisierung bieten eine solide Grundlage, um eine effiziente Ausführbarkeit des Autoencoders auf der Zielhardware zu ermöglichen. Dennoch können aus den bisherigen Erkenntnissen verschiedene Ansätze für zukünftige Forschungsarbeiten und Optimierungen abgeleitet werden.

Die Approximation der nichtlinearen Potenzfunktionen und die Quantisierungen der Fließkommaoperationen ermöglichen in der Theorie eine effiziente Ausführbarkeit des Autoencoders auf der FPGA-Hardware. Die praktische Implementierung und anschließende Tests zur Überprüfung der realen Effizienz des quantisierten Modells stehen allerdings noch aus. Mit dieser Thematik können sich zukünftige Arbeiten des ScOSA-Projekts beschäftigen.

Die vorgestellten Methodiken zur linearen Approximation und Festkommaquantisierung führten zu kleinen Verlusten in der Rekonstruktionsqualität, während dies nicht auf die Methodik der PTQ zutrifft. Allerdings konnte im Rahmen dieser Arbeit ein Optimierungsansatz vorgestellt werden, dessen Umsetzung in weiterführenden Forschungen die Ergebnisse der PTQ erheblich verbessern könnte. Dadurch wäre es unter Umständen möglich, mit der vorgestellten PTQ alle Missionsanforderungen einzuhalten.

Ein weiterer Ansatzpunkt ist die Untersuchung von alternativen Quantisierungsverfahren und -konfigurationen. Die Konfiguration, deren Untersuchung sich in diesem Kontext anbietet, ist die Per-Channel-Quantisierung. Diese wird in der Literatur häufig empfohlen, um bessere Qualitäten der PTQ zu ermöglichen. Allerdings führt sie auch zu einer höheren Rechenkomplexität, weshalb eine Abwägung zwischen der erhöhten Genauigkeit und der niedrigeren Inferenzzeit durchgeführt werden muss. Da die Per-Channel-Quantisierung in PyTorch nicht alle Komponenten des Autoencoders unterstützt, müsste eine manuelle Implementierung vorgenommen werden.
Neben den Konfigurationen der PTQ könnten auch gänzlich andere Methoden der Quantisierung angewendet werden, um deren Auswirkung auf die Rekonstruktionsqualität des Modells zu analysieren. Eine Methode, die sich für den Anwendungsfall dieses Projektes besonders eignen würde, ist die QAT. In zukünftigen Forschungsarbeiten könnte ein umfassender Datensatz für das Training des Modells ermittelt werden, der die Verwendung dieser Methode ermöglicht, mit der die Quantisierung der convolutional und deconvolutional Operationen weiter optimiert werden könnte.

Literaturverzeichnis

[1] V. Alves de Oliveira et al., „Reduced-Complexity End-to-End Variational Autoencoder for on Board Satellite Image Compression“, *Remote Sens.*, Jg. 13, S. 447, 2021. DOI: 10.3390/rs13030447. Adresse: https://doi.org/10.3390/rs13030447.

[2] G. Wallace, „The JPEG still picture compression standard“, *IEEE Transactions on Consumer Electronics*, Jg. 38, Nr. 1, S. xviii–xxxiv, 1992. DOI: 10.1109/30.125072.

[3] P. Schelkens, A. Skodras und T. Ebrahimi, *The JPEG 2000 Suite*. John Wiley & Sons, 2009.

[4] J. Goodwill, D. Wilson, S. Sabogal, A. George und C. Wilson, „Adaptively Lossy Image Compression for Onboard Processing“, in *Proceedings of the 2020 IEEE Aerospace Conference*, 2020, S. 1–15. DOI: 10.1109/AERO47225.2020.9172536.

[5] R. L. Grassa, C. Re, G. Cremonese und I. Gallo, „Hyperspectral Data Compression Using Fully Convolutional Autoencoder“, *Remote Sens.*, Jg. 14, S. 2472, 2022, Zugriff: 13.05.2024. Adresse: https://doi.org/10.3390/rs14102472.

[6] X. Ma, „High-resolution image compression algorithms in remote sensing imaging“, *Displays*, Jg. 79, S. 102 462, 2023, ISSN: 0141-9382. DOI: 10.1016/j.displa.2023.102462. Adresse: https://doi.org/10.1016/j.displa.2023.102462.

[7] o.V., *SCOSA Flugexperiment*, Zugriff: 23.05.2024, o.J. Adresse: https://www.dlr.de/de/sc/forschung-transfer/projekte/scosa-flugexperiment.

[8] T. Freitag, „Acceleration of an Autoencoder using a FPGA-SoC in a High-Performance Node of a Distributed Onboard Computer“, 2022. Adresse: https://elib.dlr.de/192913/.

[9] J. Ballé, D. Minnen, S. Singh, S. J. Hwang und N. Johnston, „Variational image compression with a scale hyperprior“, in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[10] R. C. Gonzalez und R. E. Woods, *Digital Image Processing*. Addison-Wesley Publishing Company Inc., 1992, ISBN: 0-201-50803-6.

[11] V. Dumoulin und F. Visin, *A guide to convolution arithmetic for deep learning*, 2018. arXiv: 1603.07285 [stat.ML]. Adresse: https://arxiv.org/abs/1603.07285.

[12] M. D. Zeiler, D. Krishnan, G. W. Taylor und R. Fergus, „Deconvolutional networks“, in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, S. 2528–2535. DOI: 10.1109/CVPR.2010.5539957.

[13] J. Ballé, V. Laparra und E. P. Simoncelli, *Density Modeling of Images using a Generalized Normalization Transformation*, 2016. arXiv: 1511.06281 [cs.LG]. Adresse: https://arxiv.org/abs/1511.06281.

[14] M. Carandini und D. J. Heeger, „Normalization as a canonical neural computation“, *Nature Reviews Neuroscience*, Jg. 13, Jan. 2012. DOI: 10.1038/nrn3136.

[15] J. Ballé, *Efficient Nonlinear Transforms for Lossy Image Compression*, 2018. arXiv: 1802.00847 [eess.IV]. Adresse: https://arxiv.org/abs/1802.00847.

[16] S. Lyu und E. P. Simoncelli, „Nonlinear image representation using divisive normalization“, in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, S. 1–8. DOI: 10.1109/CVPR.2008.4587821.

[17] H. Shao, B. Liu, Z. Li, C. Yan, Y. Sun und T. Wang, „A High-Throughput Processor for GDN-Based Deep Learning Image Compression“, *Electronics*, Jg. 12, Nr. 10, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12102289. Adresse: https://www.mdpi.com/2079-9292/12/10/2289.

[18] D. Salomon, *Data Compression: The Complete Reference*, 3rd. New York: Springer, 2004, ISBN: 978-0-387-40697-8. Adresse: https://doi.org/10.1007/b97635.

[19] Z. Wang, A. Bovik, H. Sheikh und E. Simoncelli, „Image quality assessment: from error visibility to structural similarity“, *IEEE Transactions on Image Processing*, Jg. 13, Nr. 4, S. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[20] Z. Wang, E. Simoncelli und A. Bovik, „Multiscale structural similarity for image quality assessment“, in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Bd. 2, 2003, 1398–1402 Vol.2. DOI: 10.1109/ACSSC.2003.1292216.

[21] C. D'Ambrosio, A. Lodi und S. Martello, „Piecewise Linear Approximation of Functions of Two Variables in MILP Models“, *Operations Research Letters*, Jg. 38, Nr. 1, S. 39–46, Jan. 2010. DOI: 10.1016/j.orl.2009.09.005.

[22] O. Ernst, I. Busch und T. Etling, *Numerische Mathematik (4V, 2Ü)*, Vorlesungsfolien, Technische Universität Chemnitz, Sommersemester 2013. Adresse: https://www.tu-chemnitz.de/mathematik/numa/lehre/numerik-2013/numerik2013.php.

[23] o.V, *Spline-Interpolation*, Zugriff: 26.06.2024, o.J. Adresse: https://de.wikipedia.org/wiki/Spline-Interpolation.

[24] B. H.-G. H. Amin K.M. Curtis, „Piecewise linear approximation applied to nonlinear function of a neural network“, *IEE Proceedings - Circuits, Devices and Systems*, Jg. 144, 313–317(4), 6 Dez. 1997. Adresse: https://digital-library.theiet.org/content/journals/10.1049/ip-cds_19971587.

[25] „IEEE Standard for Floating-Point Arithmetic“, *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, S. 1–84, 2019. DOI: 10.1109/IEEESTD.2019.8766229.

[26] o.V., *TMS320C64x DSP Library Programmer's Reference*, Zugriff: 28.06.2024, Okt. 2003, Kap. Appendix A.2, S. 150. Adresse: https://web.archive.org/web/20221222210046/https://www.ti.com/lit/ug/spru565b/spru565b.pdf.

[27] R. Yates, „Fixed-Point Arithmetic: An Introduction“, Digital Signal Labs, Technical Reference, Jan. 2013. Adresse: http://www.digitalsignallabs.com.

[28] o.V., *Understanding and implementing fixed point numbers*, Juni 2022. Adresse: http://www.sunshine2k.de/articles/coding/fp/sunfp.html.

[29] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen und T. Blankevoort, „A White Paper on Neural Network Quantization“, *CoRR*, Jg. abs/2106.08295, 2021. arXiv: 2106.08295. Adresse: https://arxiv.org/abs/2106.08295.

[30] o.V., *Quantization Algorithms*, Zugriff: 20.08.2024, o.J. Adresse: https://intellabs.github.io/distiller/algo_quantization.html.

[31] O. W. Savolainen, „How to Quantize a Neural Network Model in PyTorch: An In-Depth Explanation“, *Medium*, 2024, Zugriff: 20.08.2024. Adresse: https://oscar-savolainen.medium.com/how-to-quantize-a-neural-network-model-in-pytorch-an-in-depth-explanation-d4a2cdf632a4.

[32] o.V., *Quantization API Reference*, Zugriff: 20.08.2024, 2023. Adresse: https://pytorch.org/docs/stable/quantization-support.html.

[33] I. InterDigital, *CompressAI Model Zoo*, Accessed: 2024-07-13. Adresse: https://interdigitalinc.github.io/CompressAI/zoo.html.

[34] W. Vogt, „3.2 Das Newton-Verfahren“, in *Zur Numerik nichtlinearer Gleichungssysteme (Teil 1)*, Zugriff: 26.08.2024, 2001, S. 30–38. Adresse: https://www.db-thueringen.de/servlets/MCRFileNodeServlet/dbt_derivate_00008992/IfM_Preprint_M_01_12.pdf#page=30&zoom=auto,-13,488.

[35] J. E. Volder, „The CORDIC Trigonometric Computing Technique“, *IRE Transactions on Electronic Computers*, Jg. EC-8, Nr. 3, S. 330–334, 1959. DOI: 10.1109/TEC.1959.5222693.

[36] o.V., *CORDIC*, Zugriff: 26.08.2024, o.J. Adresse: https://de.wikipedia.org/wiki/CORDIC.

[37] B. Hamann und J.-L. Chen, „Data point selection for piecewise linear curve approximation“, *Computer Aided Geometric Design*, Jg. 11, Nr. 3, S. 289–301, 1994, ISSN: 0167-8396. DOI: https://doi.org/10.1016/0167-8396(94)90004-3. Adresse: https://www.sciencedirect.com/science/article/pii/0167839694900043.

[38] K. Weicker, *Evolutionäre Algorithmen*, 3. Aufl. Springer Vieweg Wiesbaden, 2015, ISBN: 978-3-658-09957-2. Adresse: https://doi.org/10.1007/978-3-658-09958-9.

[39] D. L. N. Hettiarachchi, V. S. P. Davuluru und E. J. Balster, „Integer vs. Floating-Point Processing on Modern FPGA Technology“, in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020, S. 0606–0612. DOI: 10.1109/CCWC47524.2020.9031118.

[40] Xilinx, *7 Series DSP48E1 Slice User Guide*, Version 1.10, Accessed: 2024-07-31, AMD, März 2018. Adresse: https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1.