POLITECHNIKA GDAŃSKA

Imię i nazwisko autora rozprawy: Mariusz Pietrołaj
Dyscyplina naukowa: Informatyka techniczna i telekomunikacja

ROZPRAWA DOKTORSKA

Tytuł rozprawy w języku polskim: Ograniczanie precyzji arytmetyki zmiennoprzecinkowej w celu treningu sieci neuronowych przy ograniczonych zasobach

Tytuł rozprawy w języku angielskim: Limitation of Floating-Point Precision for Resource Constrained Neural Network Training

Promotor *podpis*
dr hab. inż. Marek Blok

Gdańsk, rok 2024

POLITECHNIKA GDAŃSKA

The author of the doctoral dissertation: Mariusz Pietrołaj

Scientific discipline: Technical Informatics and Telecommunications

DOCTORAL DISSERTATION

Title of doctoral dissertation: Limitation of Floating-Point Precision for Resource Constrained Neural Network Training

Title of doctoral dissertation (in Polish): Ograniczanie precyzji arytmetyki zmiennoprzecinkowej w celu treningu sieci neuronowych przy ograniczonych zasobach

Supervisor *signature*
dr hab. inż. Marek Blok

Gdańsk, year 2024

POLITECHNIKA GDAŃSKA

OŚWIADCZENIE

Autor rozprawy doktorskiej: Mariusz Pietrołaj

Ja, niżej podpisany(a), oświadczam, iż jestem świadomy(a), że zgodnie z przepisem art. 27 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 poz. 1062), uczelnia może korzystać z mojej rozprawy doktorskiej zatytułowanej:

Ograniczanie precyzji arytmetyki zmiennoprzecinkowej w celu treningu sieci neuronowych przy ograniczonych zasobach

do prowadzenia badań naukowych lub w celach dydaktycznych.[1]

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym i nauce (Dz.U.2021.478 t.j.), a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładana rozprawa doktorska została napisana przeze mnie samodzielnie.

Oświadczam, że treść rozprawy opracowana została na podstawie wyników badań prowadzonych pod kierunkiem i w ścisłej współpracy z promotorem dr hab. inż. Marek Blokiem.

Niniejsza rozprawa doktorska nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem stopnia doktora.

Wszystkie informacje umieszczone w ww. rozprawie uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami, zgodnie z przepisem art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji pracy doktorskiej z załączoną wersją elektroniczną.

Gdańsk, dnia ....

podpis doktoranta

Ja, niżej podpisany(a), wyrażam zgodę/~~nie wyrażam zgody~~* na umieszczenie ww. rozprawy doktorskiej w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej.

Gdańsk, dnia ....

podpis doktoranta

**niepotrzebne usunąć*

[1] Art. 27. 1. Instytucje oświatowe oraz podmioty, o których mowa w art. 7 ust. 1 pkt 1, 2 i 4–8 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce, mogą na potrzeby zilustrowania treści przekazywanych w celach dydaktycznych lub w celu prowadzenia działalności naukowej korzystać z rozpowszechnionych utworów w oryginale i w tłumaczeniu oraz zwielokrotniać w tym celu rozpowszechnione drobne utwory lub fragmenty większych utworów.
2. W przypadku publicznego udostępniania utworów w taki sposób, aby każdy mógł mieć do nich dostęp w miejscu i czasie przez siebie wybranym korzystanie, o którym mowa w ust. 1, jest dozwolone wyłącznie dla ograniczonego kręgu osób uczących się, nauczających lub prowadzących badania naukowe, zidentyfikowanych przez podmioty wymienione w ust. 1.

POLITECHNIKA GDAŃSKA

STATEMENT

The author of the doctoral dissertation: Mariusz Pietrołaj

I, the undersigned, declare that I am aware that in accordance with the provisions of Art. 27 (1) and (2) of the Act of 4th February 1994 on Copyright and Related Rights (Journal of Laws of 2021, item 1062), the university may use my doctoral dissertation entitled:

Limitation of Floating-Point Precision for Resource Constrained Neural Network Training

for scientific or didactic purposes.[1]

Gdańsk,..... .....

signature of the PhD student

Aware of criminal liability for violations of the Act of 4th February 1994 on Copyright and Related Rights and disciplinary actions set out in the Law on Higher Education and Science(Journal of Laws 2021, item 478), as well as civil liability, I declare, that the submitted doctoral dissertation is my own work.

I declare, that the submitted doctoral dissertation is my own work performed under and in cooperation with the supervision of dr hab. inż. Marek Blok.

This submitted doctoral dissertation has never before been the basis of an official procedure associated with the awarding of a PhD degree.

All the information contained in the above thesis which is derived from written and electronic sources is documented in a list of relevant literature in accordance with Art. 34 of the Copyright and Related Rights Act.

I confirm that this doctoral dissertation is identical to the attached electronic version.

Gdańsk,..... .....

signature of the PhD student

I, the undersigned, agree/~~do not agree~~* to include an electronic version of the above doctoral dissertation in the open, institutional, digital repository of Gdańsk University of Technology.

Gdańsk,..... .....

signature of the PhD student

**delete where appropriate*

[1] Art 27. 1. Educational institutions and entities referred to in art. 7 sec. 1 points 1, 2 and 4–8 of the Act of 20 July 2018 – Law on Higher Education and Science, may use the disseminated works in the original and in translation for the purposes of illustrating the content provided for didactic purposes or in order to conduct research activities, and to reproduce for this purpose disseminated minor works or fragments of larger works.

2. If the works are made available to the public in such a way that everyone can have access to them at the place and time selected by them, as referred to in para. 1, is allowed only for a limited group of people learning, teaching or conducting research, identified by the entities listed in paragraph 1.

POLITECHNIKA GDAŃSKA

OPIS ROZPRAWY DOKTORSKIEJ

Autor rozprawy doktorskiej: Mariusz Pietrołaj

Tytuł rozprawy doktorskiej w języku polskim: Ograniczanie precyzji arytmetyki zmiennoprzecinkowej w celu treningu sieci neuronowych przy ograniczonych zasobach

Tytuł rozprawy w języku angielskim: Limitation of Floating-Point Precision for Resource Constrained Neural Network Training

Język rozprawy doktorskiej: angielski

Promotor rozprawy doktorskiej: dr hab. inż. Marek Blok

~~**Drugi promotor rozprawy doktorskiej***: <imię, nazwisko>~~

~~**Promotor pomocniczy rozprawy doktorskiej***: <imię, nazwisko>~~

~~**Kopromotor rozprawy doktorskiej***: <imię, nazwisko>~~

Data obrony:

Słowa kluczowe rozprawy doktorskiej w języku polskim: sieci neuronowe, uczenie głębokie, uczenie maszynowe, ograniczenie precyzji, asymetryczny wykładnik, liczby zmiennoprzecinkowe

Słowa kluczowe rozprawy doktorskiej w języku angielskim: neural networks, deep learning, machine learning, precision limitation, asymmetric exponent, floating point

Streszczenie rozprawy w języku polskim:

Niewystarczająca dostępność wymaganej mocy obliczeniowej i pamięci jest istotnym ograniczeniem w przypadku eksperymentów obejmujących domenę sztucznej inteligencji. Niniejsza praca skupia się na rozwiązaniu zarysowanego problemu poprzez trening sieci neuronowych z użyciem arytmetyki o ograniczonej precyzji. Na podstawie przeprowadzonych badań autor proponuje nową metodę ograniczenia arytmetyki na potrzeby treningu sieci neuronowych przy wykorzystaniu niestandardowej reprezentacji formatu zmiennoprzecinkowego o ograniczonej precyzji z zaokrąglaniem wartości współczynników. Rozprawa prezentuje wyniki uzyskane dla proponowanej metody na przykładzie wybranych referencyjnych konwolucyjnych sieci neuronowych, takich jak LeNet, AlexNet i ResNet. Zaproponowane rozwiązanie pozwala na ograniczenie zasobów wymaganych podczas treningu wybranych sieci neuronowych poprzez zmniejszenie złożoności obliczeniowej wykonywanych operacji oraz zapotrzebowania na pamięć. W ramach pracy wykazano również, że zaproponowana procedura umożliwia wytrenowanie badanych sieci neuronowych bez znaczącej utraty dokładności przy użyciu niestandardowych 8-bitowych liczb zmiennoprzecinkowych. Udowodniono również, że zaproponowany sposób

ograniczenia precyzji treningu nie wpływa negatywnie na jego zbieżność i nie jest wymagane kosztowne wydłużanie treningu poprzez zwiększoną liczbę epok.

Streszczenie rozprawy w języku angielskim:

Insufficient availability of computational power and runtime memory is a major concern when it comes to experiments in the field of artificial intelligence. One of the promising solutions for this problem is an optimization of internal neural network's calculations and its parameters' representation. This work focuses on the mentioned issue by the application of neural network training with limited precision. Based on this research, the author proposes a new method of precision limitation for neural network training leveraging a custom, constrained floating-point representation with additional rounding mechanism. Its application allows to limit the resources required during neural network training thanks to the reduction of computational complexity and memory usage. The work shows that the proposed procedure allows to train commonly used benchmark networks such as LeNet, AlexNet and ResNet without significant accuracy degradation while using only 8-bit custom floating-point variables. It has also been proven that the proposed method of precision limitation does not negatively affect the network's convergence, therefore, it is not required to extend the training by increasing the number of costly training epochs.

** niepotrzebne skreślić*

POLITECHNIKA GDAŃSKA

DESCRIPTION OF DOCTORAL DISSERTATION

The Author of the doctoral dissertation: Mariusz Pietrołaj

Title of doctoral dissertation: Limitation of Floating-Point Precision for Resource Constrained Neural Network Training

Title of doctoral dissertation in Polish: Ograniczanie precyzji arytmetyki zmiennoprzecinkowej w celu treningu sieci neuronowych przy ograniczonych zasobach

Language of doctoral dissertation: English

Supervisor: dr hab. inż. Marek Blok

~~**Second supervisor***: <first name, surname >~~

~~**Auxiliary supervisior***: <first name, surname >~~

~~**Cosupervisior***: <first name, surname >~~

Date of doctoral defense:

Keywords of doctoral dissertation in Polish: sieci neuronowe, uczenie głębokie, uczenie maszynowe, ograniczenie precyzji, asymetryczny wykładnik, liczby zmiennoprzecinkowe

Keywords of doctoral dissertation in English: neural networks, deep learning, machine learning, precision limitation, asymmetric exponent, floating point

Summary of doctoral dissertation in Polish:

Niewystarczająca dostępność wymaganej mocy obliczeniowej i pamięci jest istotnym ograniczeniem w przypadku eksperymentów obejmujących domenę sztucznej inteligencji. Niniejsza praca skupia się na rozwiązaniu zarysowanego problemu poprzez trening sieci neuronowych z użyciem arytmetyki o ograniczonej precyzji. Na podstawie przeprowadzonych badań autor proponuje nową metodę ograniczenia arytmetyki na potrzeby treningu sieci neuronowych przy wykorzystaniu niestandardowej reprezentacji formatu zmiennoprzecinkowego o ograniczonej precyzji z zaokrąglaniem wartości współczynników. Rozprawa prezentuje wyniki uzyskane dla proponowanej metody na przykładzie wybranych referencyjnych konwolucyjnych sieci neuronowych, takich jak LeNet, AlexNet i ResNet. Zaproponowane rozwiązanie pozwala na ograniczenie zasobów wymaganych podczas treningu wybranych sieci neuronowych poprzez zmniejszenie złożoności obliczeniowej wykonywanych operacji oraz zapotrzebowania na pamięć. W ramach pracy wykazano również, że zaproponowana procedura umożliwia wytrenowanie badanych sieci neuronowych bez znaczącej utraty dokładności przy użyciu niestandardowych 8-bitowych liczb zmiennoprzecinkowych. Udowodniono również, że zaproponowany sposób ograniczenia precyzji treningu nie wpływa negatywnie na jego zbieżność i nie jest wymagane kosztowne wydłużanie treningu poprzez zwiększoną liczbę epok.

Summary of doctoral dissertation in English:

Insufficient availability of computational power and runtime memory is a major concern when it comes to experiments in the field of artificial intelligence. One of the promising solutions for this problem is an optimization of internal neural network's calculations and its parameters' representation. This work focuses on the mentioned issue by the application of neural network training with limited precision. Based on this research, the author proposes a new method of precision limitation for neural network training leveraging a custom, constrained floating-point representation with additional rounding mechanism. Its application allows to limit the resources required during neural network training thanks to the reduction of computational complexity and memory usage. The work shows that the proposed procedure allows to train commonly used benchmark networks such as LeNet, AlexNet and ResNet without significant accuracy degradation while using only 8-bit custom floating-point variables. It has also been proven that the proposed method of precision limitation does not negatively affect the network's convergence, therefore, it is not required to extend the training by increasing the number of costly training epochs.

**delete where appropriate*

STRESZCZENIE ROZSZERZONE W J. POLSKIM

W dzisiejszych czasach informacja stała się jednym z najcenniejszych zasobów. Wzrost liczby urządzeń komunikujących się ze sobą oraz zbierających informacje na temat swoich użytkowników sprawił, że dostęp do obszernych źródeł danych z wielu dziedzin jest większy niż kiedykolwiek wcześniej. W wielu przypadkach analiza lub określanie zależności pomiędzy dużymi zbiorami danych stanowią realną przeszkodę w ich efektywnym wykorzystaniu [1]. Odpowiedzią na ten problem mają być algorytmy sztucznej inteligencji, w tym szeroko rozumiane sieci neuronowe [2].

Rosnąca złożoność problemów i wymagania jakościowe stawiane przed współczesnymi architekturami sieci neuronowych zapoczątkowały gwałtowny wzrost zapotrzebowania na moc obliczeniową i zasoby sprzętowe wymagane do ich inferencji oraz znacznie bardziej złożonego treningu [3]. Wraz ze wzrostem wymagań, wykorzystanie ogólnodostępnych procesorów okazało się niewystarczające, co skłoniło badaczy do poszukiwania alternatywnych rozwiązań sprzętowych takich jak karty graficzne lub niestandardowe architektury oparte na bezpośrednio programowalnej macierzy bramek [4]. Rosnący koszt energii i sprzętu niezbędnego do wytrenowania współczesnych sieci neuronowych był naturalnym następstwem wykorzystywania coraz większej mocy obliczeniowej i złożonych architektur docelowych modeli. Czynniki te skłoniły badaczy do zgłębienia tematu możliwego ograniczenia wymagań zasobów stawianych przez standardowe procedury wykorzystania sieci neuronowych [5] [6] [7] [8].

Początkowe badania dotyczące optymalizacji sieci neuronowych skupiały się głównie na mniej skomplikowanej, lecz znacznie częściej wykonywanej fazie inferencji. Techniki takie jak kwantyzacja, pruning czy uwspólnianie wag pozwalają na znaczne zmniejszenie rozmiaru jak i złożoności obliczeniowej modelu [9]. W ślad za tym pojawiły się rozwiązania sprzętowe, w tym akceleratory, pozwalające na efektywne wykorzystanie zoptymalizowanych topologii modeli [10] [11] [12]. Pomimo iż wymienione metody zostały na stałe ugruntowane w rozwiązaniach wdrażanych produkcyjnie oraz bibliotekach programistycznych, temat optymalizacji treningu jest nadal otwartym zagadnieniem [13].

W rozprawie przedstawiony jest przegląd propozycji i rozwiązań opracowany na bazie dostępnej literatury. Techniki obejmują zarówno rozwiązania programistyczne [5] [6] [14] jak i czysto sprzętowe [15] [16] [17]. Ze względu na kierunek przedstawionej pracy i skupieniu się na zagadnieniu treningu sieci neuronowych z wykorzystaniem liczb o ograniczonej precyzji, szczególnej uwadze poddano rozwiązania dotyczące kwantyzacji, ograniczonej i mieszanej precyzji liczb zmiennoprzecinkowych oraz wspierające je rozwiązania sprzętowe.

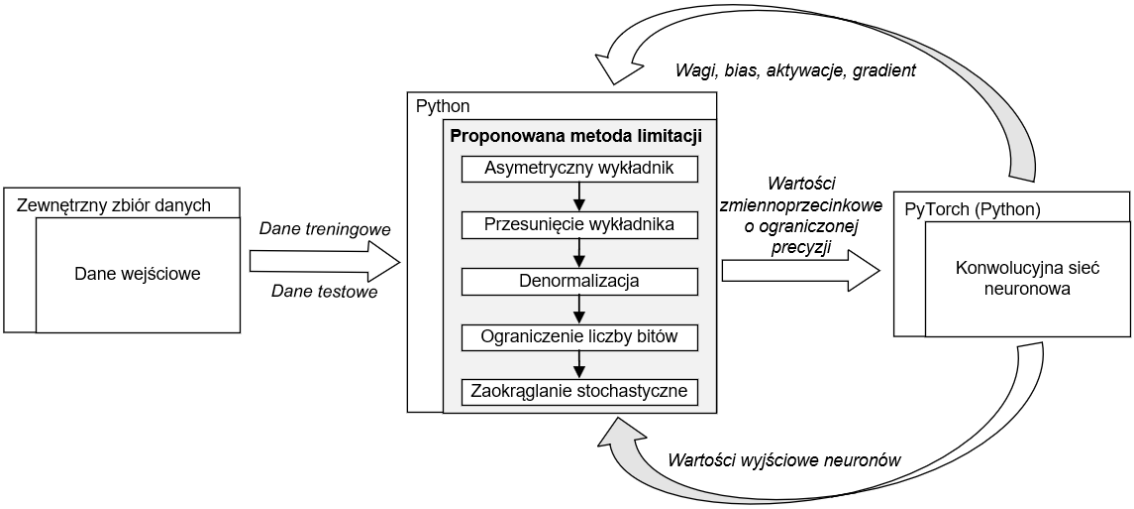
Po zarysowaniu teorii dotyczącej wybranych architektur sieci neuronowych oraz ich optymalizacji, przedstawiony został cel pracy, którym było opracowanie nowej metody ograniczenia arytmetyki na potrzeby treningu konwolucyjnych sieci neuronowych przy wykorzystaniu zmniejszonej precyzji liczb zmiennoprzecinkowych. Dodatkowo przyjęto założenie

braku znaczącej utraty jakości wytrenowanych w ten sposób modeli. W ramach tego zagadnienia postawione i udowodnione zostały dwie tezy:

1. **Możliwe jest wytrenowanie referencyjnych sieci neuronowych takich jak LeNet, AlexNet i ResNet-18 przy użyciu niestandardowych 8-bitowych liczb zmiennoprzecinkowych bez znaczącej utraty dokładności w porównaniu do treningu z użyciem 32-bitowych liczb zmiennoprzecinkowych opisanych w standardzie IEEE-754.**
2. **Zastosowanie zaproponowanej metody ograniczenia precyzji arytmetyki do treningu konwolucyjnych sieci neuronowych przy zmniejszonej precyzji liczb zmiennoprzecinkowych pozwala na ograniczenie wymaganej do realizacji treningu mocy obliczeniowej oraz pamięci.**

Autor rozprawy przeprowadza analizę wykorzystania wykładnika 32-bitowych liczb zmiennoprzecinkowych podczas treningu wybranych sieci neuronowych na publicznie dostępnych zbiorach treningowych. W ramach tej analizy zaobserwowana została nieefektywność wykorzystania bitów wykładnika dostępnych w 32-bitowym typie IEEE-754. Przedstawione zjawisko wskazuje, że z dużym prawdopodobieństwem użycie liczb zmiennoprzecinkowych o mniejszym zakresie bitowym do treningu wskazanych architektur sieci neuronowych nie spowoduje pogorszenia skuteczności tego treningu.

Następnie zaprezentowana została autorska metoda ograniczonej arytmetyki na potrzeby treningu sieci neuronowych opierająca się na wykorzystaniu niestandardowego, ograniczonego bitowo typu danych zmiennoprzecinkowych. Wskazana technika, oprócz ograniczenia pamięci wymaganej przez poszczególne parametry sieci neuronowej, wprowadza również modyfikację interpretacji wykładnika poprzez zastosowanie jej asymetrycznej reprezentacji z przesunięciem. Oznacza to, że wszystkie bity przeznaczone na wykładnik reprezentują tylko wybrany zakres jej ujemnych wartości. Dodatkowe przesunięcie pozwala na odpowiednie dopasowanie zakresu wartości reprezentowanych przez wykładnik do wybranej architektury sieci. Oprócz wymienionych elementów, metoda uwzględnia również zdenormalizowaną reprezentację wartości oraz zaokrąglanie stochastyczne niwelujące ograniczenia wynikające z dostępnego podczas treningu bardzo zawężonego zbioru wartości zmiennoprzecinkowych. W celu zastosowania przedstawionej metody opracowano środowisko programistyczne wspierające wymienione techniki, co zostało przedstawiona na Rys. 1.



Rys. 1. Środowisko programistyczne proponowanej metody ograniczenia arytmetyki treningu sieci neuronowych z użyciem liczb zmiennoprzecinkowych o zmniejszonej precyzji

W celu sprawdzenia efektywności zaprezentowanej metody przeprowadzono liczne treningi referencyjnych sieci neuronowych. Jako punkt odniesienia prezentowanych wyników przyjęto rezultaty referencyjnych sieci neuronowych trenowanych z użyciem wielu wariantów długości bitowej liczb zmiennoprzecinkowych. Następnie przeprowadzono analogiczne treningi wraz z zastosowaniem przedstawionej metody w pełnym zakresie dostępnych kombinacji liczby bitów wykładnika i mantysy. We wszystkich przypadkach ograniczenie precyzji było realizowane programowo i wykonywane na ogólnodostępnych procesorach z wykorzystanie tymczasowej translacji limitowanych zmiennych do typu 32-bitowego. Tabela 1 przedstawia rezultaty porównania wyników treningu sieci neuronowych przy użyciu proponowanej metody z uwzględnieniem różnych wariantów reprezentacji 8-bitowego typu zmiennoprzecinkowego.

Dodatkowo Tabela 1 przedstawia porównanie dokładności trenowanych sieci neuronowych z ograniczoną precyzją ze standardowym treningiem 32-bitowym. Podkreślone wartości wskazują konfiguracje w których proponowana metoda pozwoliła na uzyskanie dokładności powyżej standardowego podejścia 32-bitowego. Wyniki te wskazują również, że użycie proponowanej metody nie wymaga zwiększenia liczby epok treningowych, co mogłoby negatywnie wpłynąć na zakres zużytej energii i czasu wykorzystania zasobów obliczeniowych.

Uzyskane rezultaty pozwoliły na potwierdzenie obu postawionych tez. Pierwsza teza: **„Możliwe jest wytrenowanie referencyjnych sieci neuronowych takich jak LeNet, AlexNet i ResNet-18 przy użyciu niestandardowych 8-bitowych liczb zmiennoprzecinkowych bez znaczącej utraty dokładności w porównaniu do treningu z użyciem 32-bitowych liczb zmiennoprzecinkowych opisanych w standardzie IEEE-754”** została bezpośrednio potwierdzona licznymi treningami sieci neuronowych nie tylko w zakresie liczb 8-bitowych, ale również typów danych o mniejszej liczbie bitów. W przypadku wszystkich weryfikowanych sieci udało się uzyskać wyniki na poziome standardowych treningów z użyciem 32-bitowych liczb zmiennoprzecinkowych.

Tabela 1. Porównanie wyników 10-tej epoki treningów z użyciem zaprezentowanej metody na różnych 8-bitowych wariantach danych zmiennoprzecinkowych, ostatni wiersz przedstawia wyniki dla bazowego 32-bitowego formatu IEEE-754

Liczba zmiennoprzecinkowa			**LeNet**	**AlexNet**		**ResNet-18**	
Liczba bitów znaku	**Liczba bitów wykładnika**	**Liczba bitów mantysy**	**MNIST**	**CIFAR10**	**CIFAR100**	**CIFAR10**	**CIFAR100**
1	**1**	**6**	54.84	22.23%	1.98%	8.02%	0.91%
1	**2**	**5**	77.81	62.93%	1.46%	9.97%	1.02%
1	**3**	**4**	**96.15%**	72.94%	38.59%	7.34%	1.17%
1	**4**	**3**	95.98%	**74.50%**	**38.69%**	**76.01%**	40.21%
1	**5**	**2**	95.78%	71.10%	36.02%	62.85%	**42.62%**
1	**6**	**1**	94.66%	66.11%	30.00%	63.39%	39.68%
Bazowy format zmiennoprzecinkowy IEEE-754 32-bit							
1	**8**	**23**	96.18%	74.39%	38.93%	77.08%	39.54%

Druga teza stawiana w rozprawie: **„Zastosowanie zaproponowanej metody ograniczenia arytmetyki do treningu konwolucyjnych sieci neuronowych przy zmniejszonej precyzji liczb zmiennoprzecinkowych pozwala na ograniczenie wymaganej do realizacji treningu mocy obliczeniowej oraz pamięci”** została udowodniona pośrednio. Wykorzystanie typów danych o 75% mniejszej liczbie bitów pozwala na jednoznaczne zaoszczędzenie zarówno cykli procesora niezbędnych na przeprowadzenia operacji mnożenia i dodawania liczb zmiennoprzecinkowych oraz składowanie ich wyników w pamięci operacyjnej [18]. Dodatkowo zmniejszone typy danych jednoznacznie przyczyniają się do ograniczenia pamięci wymaganej do zapisywania i przechowywania współczynników wytrenowanego modelu.

Osiągnięcie zakładanych rezultatów i udowodnienie tez pracy nie oznacza jednak, że analizowany problem został ostatecznie rozwiązany. Zaprezentowana metoda otwiera wiele kierunków dalszych badań i optymalizacji. Oprócz weryfikacji przedstawionych technik na większej grupie architektur sieci neuronowych, kluczowa jest dalsza optymalizacja samej metody i jej elementów do poszczególnych parametrów sieci. Dzięki zastosowanej technice przesunięcia asymetrycznego wykładnika, przedstawiona metoda może zostać użyta w realizacjach o mieszanej precyzji, wraz ze zmianą przesunięcia istnieje możliwość określenia różnych zakresów precyzji dla:

- Kolejnych epok treningowych, wprowadzając możliwość regularyzacji uczenia sieci wraz z postępem procesu treningu,
- poszczególnych warstw danej architektury sieci neuronowej,

- parametrów sieci neuronowych z rozróżnieniem na wagi, bias [19], gradient i aktywacje.

Dodatkowym krokiem mającym na celu dokładniejsze określenie oszczędności zasobów stawianych przez zaprezentowaną metodę jest jej implementacja przy pomocy bezpośrednio programowalnej macierzy bramek. Prace w tym kierunku zostały już zapoczątkowane przez Aleksiuk et al. (2023) [20] w ramach implementacji 8-bitowego mnożnika wspierającego założenia prezentowanej metody.

ACKNOWLEDGMENTS

*I would like to thank my supervisor, **dr hab. inż. Marek Blok**, for his invaluable help and constant support during the preparation of this dissertation. Without his extraordinary commitment and immense amount of time spent on steering me in the right direction regarding the conducted research, it wouldn't be possible. His insights were instrumental in shaping this dissertation into its current form, and I appreciate them greatly.*

*Many thanks to **dr inż. Bartosz Czaplewski** and **mgr Wojciech Borkowski** for giving me a constant access to their computational infrastructure which allowed me to continue my research.*

*I am extremely grateful to **my mother and sister** for encouraging me throughout the whole time and giving me huge motivation to continue this work.*

*Finally, I wouldn't have done it without the remarkable patience and understanding of my beloved **fiancée Asia**. Despite the circumstances, she continues to support me, even though I should spend more time with her rather than working.*

LIST OF THE MOST IMPORTANT SYMBOLS AND ABBREVIATIONS

AI	-	Artificial Intelligence
ASIC	-	Application Specific Integration Circuits
BPTT	-	Backpropagation Through Time
CIFAR	-	Canadian Institute for Advanced Research
CNN	-	Convolutional Neural Network
CPT	-	Cyclic Precision Training
CPU	-	Common Processing Unit
DLA	-	Deep Learning Accelerator
DNN	-	Deep Neural Network
DSP	-	Dynamic Precision Scaling
FGMP	-	Fined-grained Mixed Precision
FLOP	-	Floating-point Operation
FP	-	Floating-point
FP32	-	32-bit Floating-point
FP8	-	8-bit Floating-point
FP8-SEB	-	8-bit Floating-point Type with a Shared Exponent Bias
FPGA	-	Field Programmable Gate Array
FPU	-	Floating Point Unit
GD	-	Gradient Descent
GPU	-	Graphical Processing Units
GRU	-	Gated Recurrent Units
IEEE	-	Institute of Electrical and Electronics Engineers
ILSVRC	-	International Large Scale Visual Recognition Challenge
INT	-	Integer
INT32	-	32-bit Integer
INT8	-	8-bit Integer
IoT	-	Internet of Things
LDP	-	Learnable Dynamic Precision
LNPU	-	Learning Processing Unit
LSTM	-	Long-Short Term Memory
LUT	-	Lookup Table
MAC	-	Multiply-accumulate
MB	-	Megabyte
ML	-	Machine Learning
MNIST	-	Modified National Institute of Standards and Technology
MoFQ	-	Mixture-of-Formats Quantization
NaN	-	Not a Number
NLP	-	Natural Language Processing

NN	-	Neural Network
NPU	-	Neural Processing Unit
PCM	-	Precision-Controlled Memory
PE	-	Processing Engine
PTQ	-	Post Training Quantization
QAT	-	Quantization Aware Training
R&D	-	Research and Development
ResNet	-	Residual Neural Network
RNN	-	Recurrent Neural Network
TPU	-	Tensorflow Processing Unit
UNPU	-	Unified Neural Processing Unit
VGG	-	Visual Geometry Group
XLA	-	Tensorflow Accelerated Linear Algebra

TABLE OF CONTENTS

1 INTRODUCTION

Nowadays, information can be treated as one of the most precious resources. The continuing expansion of Internet of Things (IoT) along with an increasing number of various mobile devices, sensors and smart utilities produces an unprecedented amount of data every day. The ability to efficiently process, analyze, and filter data is one of the main challenges facing the big data domain [1]. Application of Artificial Intelligence (AI) is often presented as a viable solution for this problem, especially in case of extremely large, differentiated datasets [2]. Unfortunately, increasing complexity and abundance of incoming data requires more resources for effective processing. This issue is especially vivid in the case of neural networks (NN) evolution and its adaptation by the industry [21]. Along with the difficulty of the problems that need to be solved, there can be observed a growth of NN architectures size and complexity [3]. Many recent improvements have been implemented at the cost of additional computational power, runtime memory and storage required by NN designs which poses a question of both financial and environmental profitability in terms of NN applications if such a trend remains unchanged [22].

Although big data is much more accessible today for the industry and researchers, its applicability to machine learning is not always straightforward. Lack of accurately labeled data is still a relevant concern for many classification tasks, especially in case of supervised learning [23]. Continuous improvement of NN models, depending on training data accessibility, was followed by increasing storage and computational requirements [24]. Leveraging Graphical Processing Units (GPU), Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuits (ASIC) was a common response from the research community to overcome high computational demands [4]. Further development started a rapid increase of neurons in broadly used NN architectures and shifted researchers focus to deep neural network (DNN) topologies. Fig. 1.1 gives a good perspective on a growing computational and memory complexity as an aim to achieve better classification accuracy.

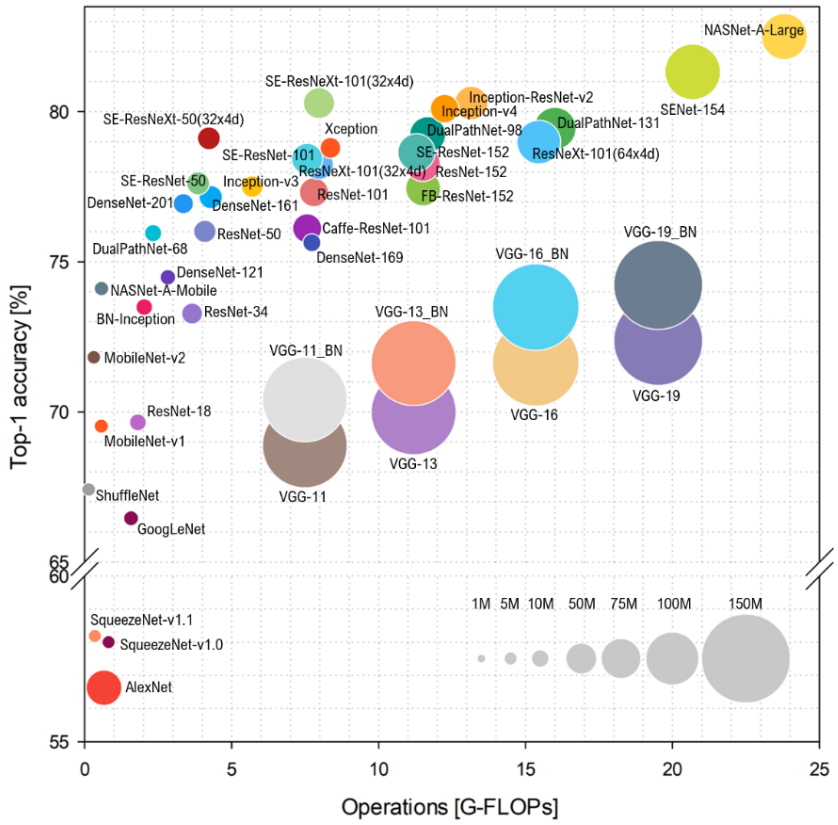


Fig. 1.1. Top-1 accuracy compared to the computational complexity of NN models in floating-point operations (FLOPs) required for a single forward pass. The size of each ball corresponds to the model's complexity (required memory in MB) [3]

Utilization of deeper NN topologies brought a disadvantage of a significant number of power-consuming floating-point operations. Additionally, the growing size of neural network architectures translated to bigger memory footprint [3]. These problems were apparent in the case of both training and inference. Over recent years, many researchers pursued the subject of resource efficiency of neural networks [5] [6] [7] [8]. The utilization of less resource demanding operations was one of the paths which showed satisfactory results. Focusing on restraining power-hungry floating-point operations proved to be an effective way for limiting both computational and memory requirements. Multiple techniques such as pruning or quantization of regular IEEE-754 32-bit floating-point parameters have been successfully adapted by the industry to improve resource efficiency of the inference process [9]. Nevertheless, this technique is not always easily applicable to the training phase of neural models which requires much more time and resources.

Modern machine learning domain relies on high performance GPUs and cloud computing with robust datacenter backends [25]. Nevertheless, plenty of neural network-based solutions are developed and deployed on mobile and low-power devices [26]. Software optimization is not always sufficient for running AI applications in such a constrained environment. Hence, innovation on the hardware side of the machine learning (ML) applicability is crucial for broader productization of modern AI [15] [16] [17]. The most vivid response from the market in the hardware field are dedicated chips and accelerators, often integrated into common processing

units (CPU). Market leaders such as Nvidia, Google, Intel and Qualcomm continue development on this path as general purpose hardware is not always efficiently utilized in the case of NN processing [27]. The influence of AI specific hardware is especially visible in the case of functionalities that need to be continuously enabled on the device such as phrase recognition, voice translation or image augmentation [28]. Offloading such computation to low-power neural accelerators significantly increases energy efficiency of the device and related user experience.

However multiple effective methods of neural network optimization have been proposed, the aspect of low-resource training process is still an open issue [13]. The initial focus on the inference is understandable as the network, once it is trained, can be used and deployed on multiple devices. Additionally, experiments showed that inference is much more resilient to parameters precision limitation [29]. The training phase, required for NN model preparation, requires much more resources and a significant amount of input data. Although in theory this process can be done only once, in practice creating a good quality model requires extensive experimentation and hyper-parameterization [30]. Limiting the time and resources required for such a model preparation would allow for a broader experimentation phase within the research community and faster productization of less resource demanding products [22]. Fig. 1.2 presents an overview of the inference and training comparison.

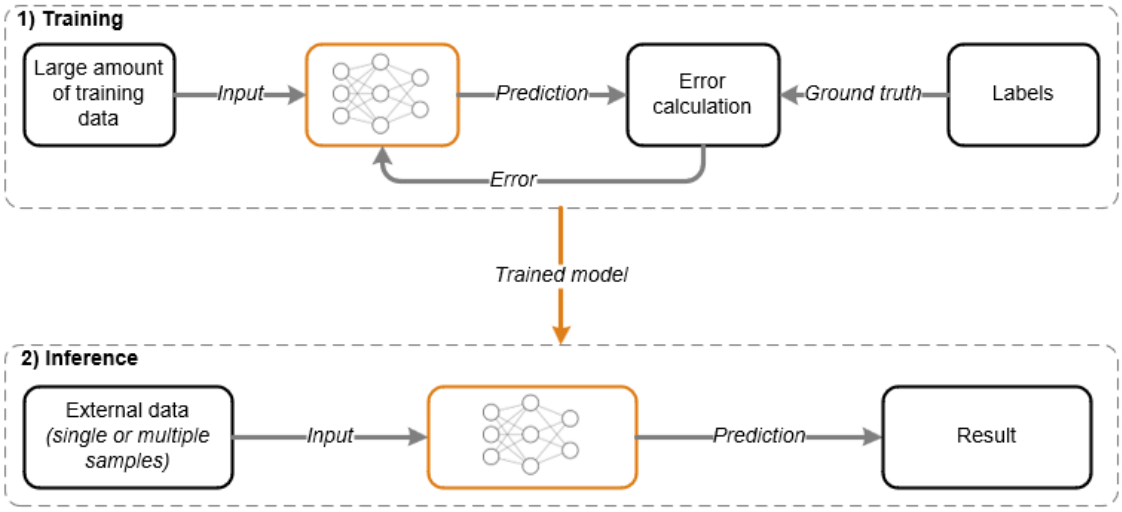


Fig. 1.2. Simplified overview of NN training and inference processes

Although resource utilization and efficiency of neural network-based solutions is an extremely important aspect, factors of privacy and security cannot be overlooked, especially in terms of low-power devices. A growing number of mobile applications base their functionalities on various forms of NN architectures. Such a model can be inferred on the device itself or offloaded to a cloud backend [31]. Inference on a user's device requires usage of a memory constrained neural network model which is usually trained before the deployment into the device takes place [32]. Such an approach often limits the performance of the model and requires re-deployment of a new instance in case there is an update of the functionality. Usage of a cloud infrastructure for NN related computations resolves issues related to the device's hardware limitations as long as there is a stable network connection available. However, transferring user

data to an external server can be treated as a potential privacy and security risk [33]. Ideally, the model should be able to continuously adapt to users' private data without transferring it outside the device. Optimization of the training process may be a promising solution for these problems.

In the face of the contradictory relation between AI solutions and scarcity of available resources, the aim of the study considers the following subjects:

1. Verification of influence of limited precision variables to the training quality of popular benchmark convolutional neural networks.
2. Consideration of limited precision influence on resource consumption of neural network training process.
3. Proposition of neural network training process with limited precision floating-point variables, including new data type parameters format.

Based on the aims set for this dissertation, the following theses are to be proven by the conducted studies:

1. **It is possible to train popular convolutional neural networks as LeNet, AlexNet and ResNet-18 with custom 8-bit floating-point variable's type without significant classification accuracy degradation in comparison to regular IEEE-754 32-bit floating-point.**
2. **Application of the proposed arithmetic precision limitation method for convolutional neural networks training with low level bit count floating-point variables allows to decrease computational power and memory requirements.**

This dissertation is organized in a form of 5 separate chapters. The following, chapter 2, starts with a concise presentation of the theory behind NN training with a major focus on several common neural topologies used in this area. Afterwards, it outlines a background behind floating-point representation with a perspective on its limitations and popular rounding techniques that are also applicable to the ML domain. The wide area of NN optimization and acceleration is covered at the end of chapter 2 including software and hardware-based solutions. The next, 3rd chapter, is solely focused on neural network precision limitation. It opens with a related study section containing a wide presentation of the dissertation's results in comparison to other researchers' outcomes. Subsequently starts the description of experiments on NN limitations conducted by the author, giving an insight into the influence of limited precision parameters to the network's classification accuracy. Additionally, the exponent utilization results are presented and commented as an introduction to the proposed limitation method. Chapter 4 depicts the proposed method along with an explanation of the techniques incorporated into the training process. Moreover, details of the conducted experiments are presented for several neural network architectures. The impact of incorporated method's features is also discussed with a strong focus on neural network training convergence. The last, 5th chapter, closes the dissertation with a summary and directions for the future work on the presented method of NN training with limited precision. Fig. 1.3 gives a detailed overview of the dissertation's structure.

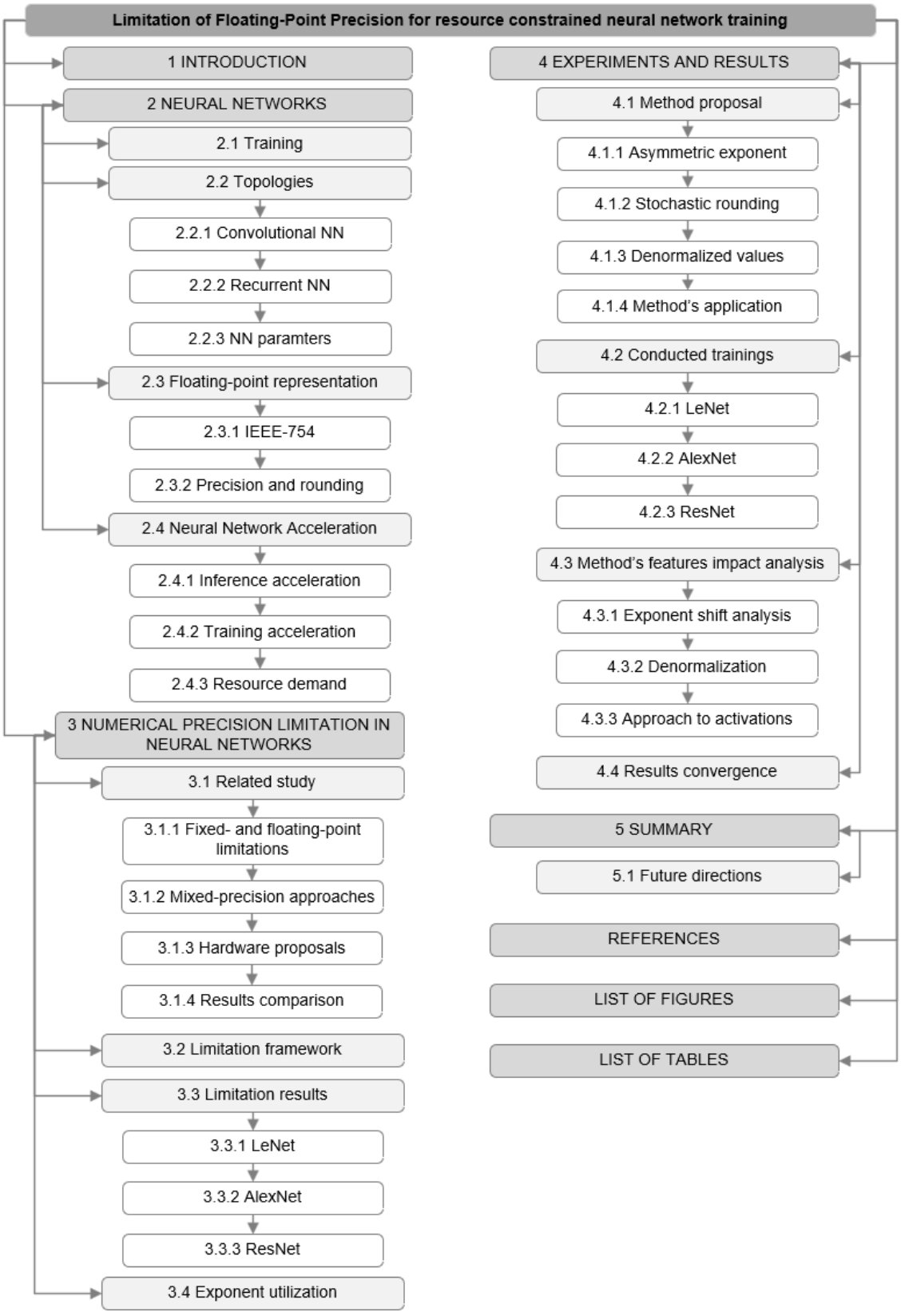


Fig. 1.3. An overview of the dissertation's structure

2 NEURAL NETWORKS

The foundation of NN creation lies in the attempt to model a human brain behavior with a mathematical algorithm [34]. In general, this structure aims to approximate the relation between input and output data during the process of training. As in the case of a human brain, neurons are core building elements of the whole network entity. In the initial proposal in 1958, a single artificial neuron, also known as perceptron, includes three basic features as weights, bias, and activation [35]. Based on its state, a neuron can react accordingly to the input data and fire with a response providing output for other elements of the network. The mechanics behind this functionality can be described by the following equation.

$$y = f_a\left(\left(\sum_{i=1}^{n} x_i \cdot w_i\right) + b\right) \tag{2.1}$$

where:

y – output of the neuron,

$f_a(\cdot)$ – activation function,

x_i – i-th input of the neuron,

w_i – weight assigned to the i-th input of the neuron,

b – bias.

Although a single neuron cannot be used for any complicated task, it proves to be extremely useful when scaled into a form of a larger structure. Modern NN architectures consist of millions of artificial neurons connected with each other. In most cases those neurons are grouped in the form of layers which output is then passed as an input to other deeper parts of the structure forming a deep neural network [3]. Fig. 2.1 depicts a single artificial neuron.

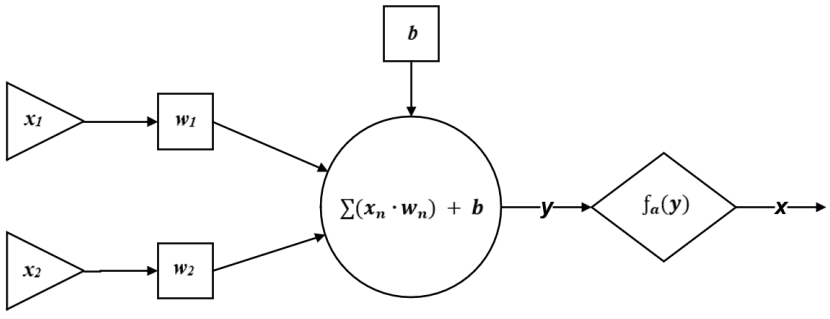


Fig. 2.1. Single artificial neuron with two inputs [36]

2.1 Training

The procedure of neural network training is a heavily time and resource consuming process. In order to adapt the network to a particular problem, weights and biases of each neuron have to be tediously adjusted based on the training input and the current network's predictions. This issue is especially vivid in the case of deep neural networks where mentioned adjustments

need to be continuously applied to millions of neurons grouped in hundreds of layers. In fact, this problem was the reason behind the longtime lack of interest for NN in the field of AI [37].

The mechanism of backpropagation proved to be a solution for deep neural network training. It leverages chain rule for derivatives calculation in order to establish a gradient of a loss function with respect to the parameters of the models [38]. Thanks to this mechanism it is possible to establish the influence of a single parameter on a final error of the network. Along with the application of optimizers such as Gradient Descent (GD), and its variations like Stochastic GD, Batch GD, Mini-Batch GD [39], it enables deep neural network training by adjusting trainable parameters. Additional hyper-parameters such as learning rate and batch size enable the algorithm to define the size of the update steps or frequency of gradient calculation in relation to input data samples. Fig. 2.2 presents a simple one-layer neural network structure with a single backpropagation example for one of the paths in a network.

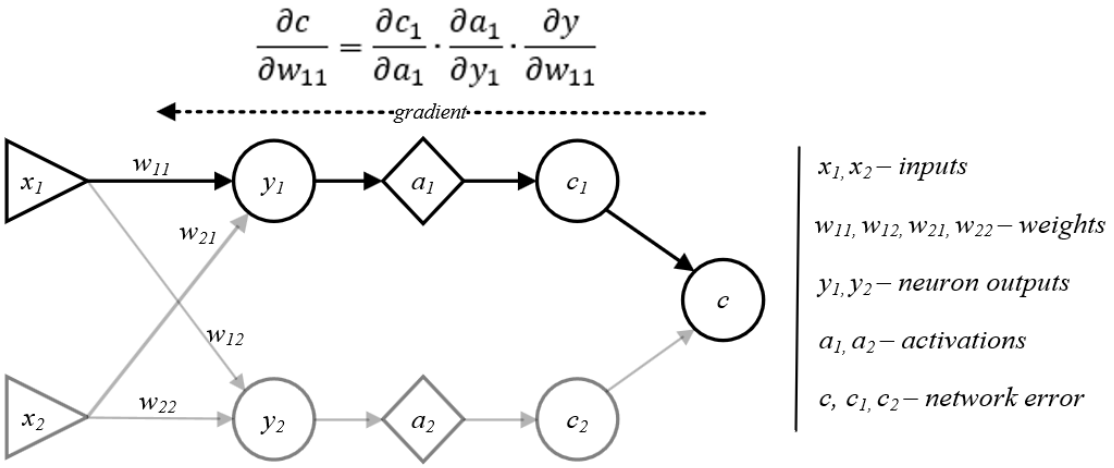


Fig. 2.2. Backpropagation chain rule example - gradient calculation with respect to a single weight parameter (biases excluded)

Frequent gradient calculation and parameters adjustment requires many floating-point based multiplications which are one of the most power-hungry operations. Hence, gradient calculation and update of the network's parameters are the most computationally expensive parts of the neural network training [40]. It needs to be mentioned that there are several issues connected with gradient calculation such as gradient explosion or gradient vanishing [41]. The latter one is especially important in case of using limited precision parameters for NN training. Additional reduction of precision increases the vanishing effect of small floating-point numbers that require high negative exponent values [42]. Those are only a few reasons why researchers still pursue a more efficient way for DNN training.

2.2 Topologies

The transition from a perceptron to a multilayer perceptron can be treated as the beginning of deeper NN architectures [43]. The ability to stack neurons in layers enabled researchers to touch much more complex problems that led to the growing size of proposed architectures. It is generally accepted that a structure with more than two layers can be called a

DNN, however, modern architectures often significantly exceed this minimum [44] [45]. Fig. 2.3 gives an example of a simple multilayer perceptron with input layer, output layer and two hidden layers.

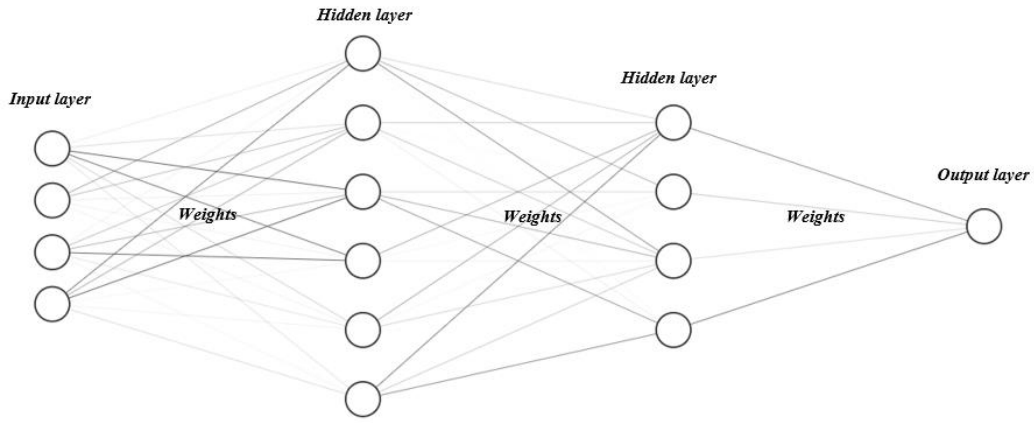


Fig. 2.3. Deep Neural Network with two hidden layers [46]

The evolution of robust deep learning architectures enabled neural networks to represent multiple levels of abstraction in the scope of available training data [47]. Easier access to large amounts of data and high computational power led to multiple breakthroughs in image, video [48] [49] and natural language processing [50], making this domain a main ML solution when it comes to solving complex problems. The input data format was an additional factor that influenced neural topology designs. In general, initial artificial NNs treated input as a vector of correlated numbers. However, such an approach proved to be insufficient for multi-dimensional data such as images [51]. Additionally, in many cases the time relation between input data is crucial for its understanding and proper classification or new data generation. Those, among many, factors highly contributed to the evolution of convolutional [52] and recurrent [53] neural networks that are known today and were used as a starting point for further architectural improvements in the field.

2.2.1 Convolutional NN

Convolutional Neural Networks (CNN), which are a focus of this dissertation, are one of the architectures which have been highly adopted by the industry. They proved to be especially useful in case of image and audio pattern processing [52]. The key element differentiating CNNs from regular NN is the ability to efficiently handle spatial dimensionality of the input. A batch of images is a good example of such data [51]. We can distinguish two basic dimensions as height and width, the third dimension based on the number of samples can be treated as the depth of the input. It needs to be mentioned that we could still use a multilayer perceptron for processing such data, but it would be much more complex and require additional data preparations such as flattening. Although regular fully connected layers are still used by CNN architectures, there are two main additional mechanisms that differ from this architecture: convolution and pooling. The combination of multiple instances of these layer types is the reason behind such appreciated effectiveness of CNNs.

Convolution layer treats its 1D or 2D input as a map of features. This map is scanned by a filter which is applied to the whole input including its depth which is perceived as input channels. The result of the filter operation creates an output features map. After applying activation, it can be used as an input for deeper layers of the network, commonly this structure is called an activation map. The number of filters used for the convolution layer defines output channels which state for the depth of the output. Fig. 2.4 depicts an example of a convolution with one channel 4x4 input, single 2x2 filter, stride equal to 1 and no padding.

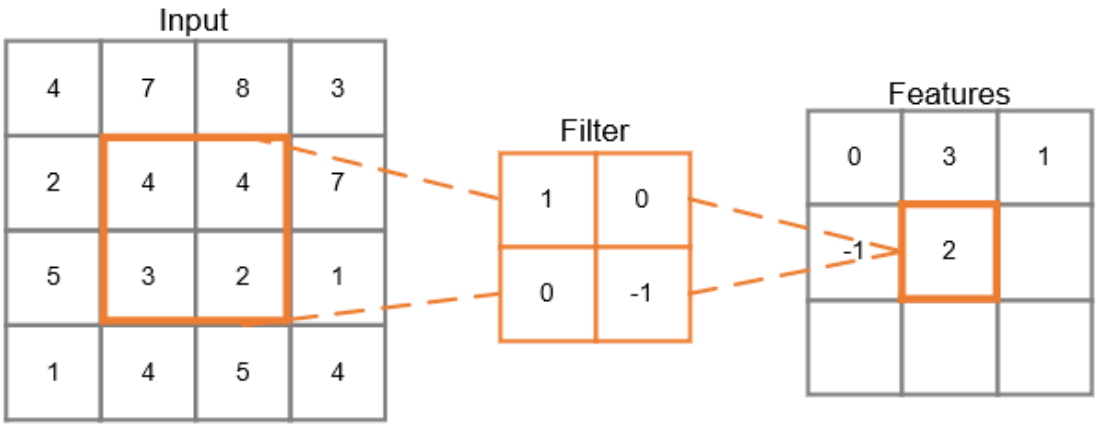


Fig. 2.4. Example of a simple 1 channel 2d convolution

In case of high-resolution input, applying multiple convolutions might be a computationally heavy procedure. In order to reduce dimensionality of feature maps, neural network models leverage pooling layers which allow for limiting the number of operations in a model. Max and average pooling are the two most commonly used layers for this purpose. In a similar fashion to convolutions, there are multiple parameters that can be adjusted in case of pooling such as kernel size or overlapping. Fig. 2.5 gives an example of 2x2 max pooling with stride 2, applied to a one channel 4x4 input.

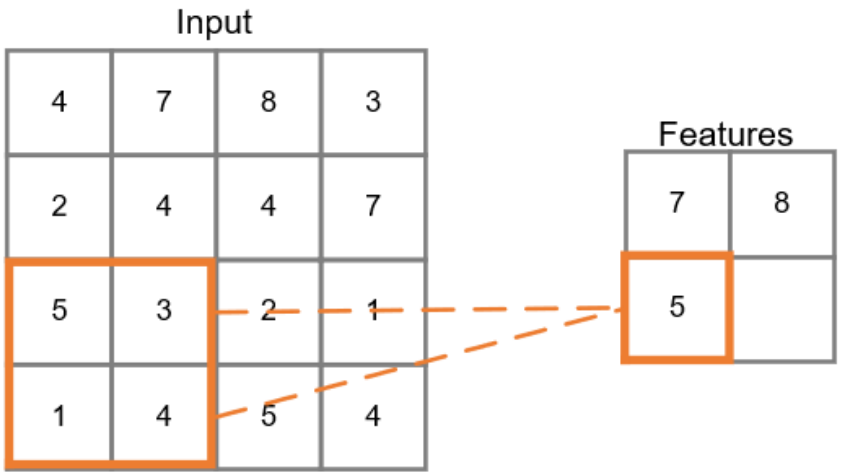


Fig. 2.5. Example of a simple 1 channel max pooling

Fully connected layers are usually applied at the end of convolutional network topology. A meaningful showcase of abilities of such CNN architecture was LeNet-5 designed for zip codes

reading in the US postal offices [54]. The originally presented neural network consisted of three convolutional layers, two pooling layers followed by two fully connected layers and a softmax classifier. The overall number of trainable parameters of this network totals to 60000. Fig. 2.6 presents the original LeNet-5 network as proposed by (LeCun et al 1998) [55].

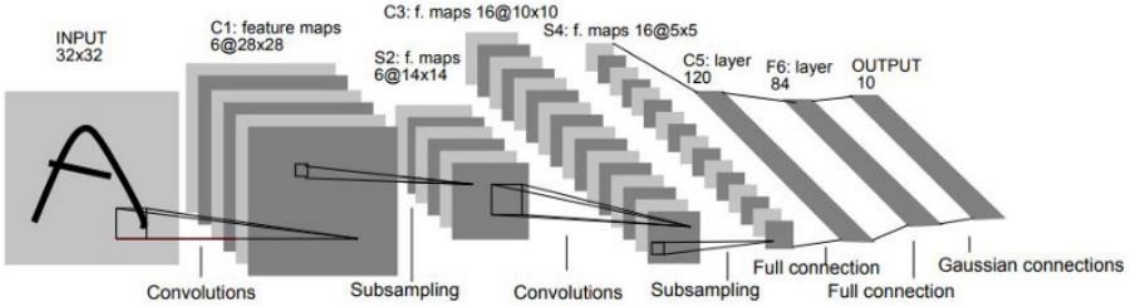


Fig. 2.6. Original LeNet-5 architecture [55]

The modern interest in deep CNN topologies has been restored by the winner of the 2012 ImageNet competition AlexNet. The challenge involved a trainset of 1.2 million images split into 1000 categories. Evaluation of models was done on a separate classification test data that has not been previously seen by the competing models [56]. The AlexNet architecture outperformed other competitors by almost over 10 percentage points of accuracy [48]. The results have been achieved thanks to a large CNN topology which was executed on GPU instead of, popular at this time, CPU. This event initiated a rapid shift to GPU NN training, which is confirmed by the change in ImageNet challenge submissions. In 2012 four entrants used GPUs, when in 2014 almost all 110 were using such devices [40]. Fig. 2.7 depicts the original topology of AlexNet architecture. The size of the network was so big for current standards that it had to be trained on two GPU cards due to hardware memory constraints [48].

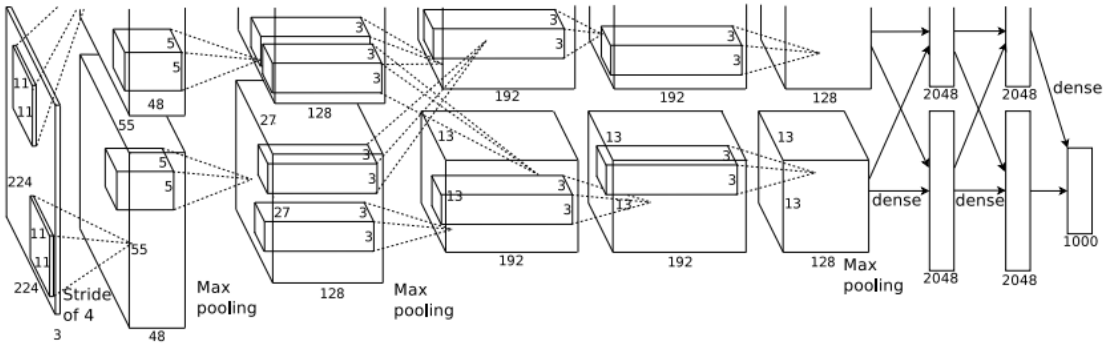


Fig. 2.7. Original AlexNet architecture [48]

The success of AlexNet started the pursuit of continuously growing network architectures as VGG [57] and GoogLeNet [58], where improved accuracy of the model has been achieved at the cost of increasing the number of layers and computational complexity. Although, in theory, more robust neural networks should provide better prediction accuracy, it became apparent that increasing the number of layers enhances the problem of exploding and vanishing gradient [59] [60]. The novel architecture change proposed by Zhang et al. (2016) [61] addressed this problem by introducing ResNet (Residual Network) which won the 2015 ImageNet competition. The key

idea behind ResNet revolves around shortcut connections which skip one or more layers in the network's topology providing an identity mapping by adding the output of such a layer to the output of skipped layers. An example building block of a residual connection is presented in Fig. 2.8.

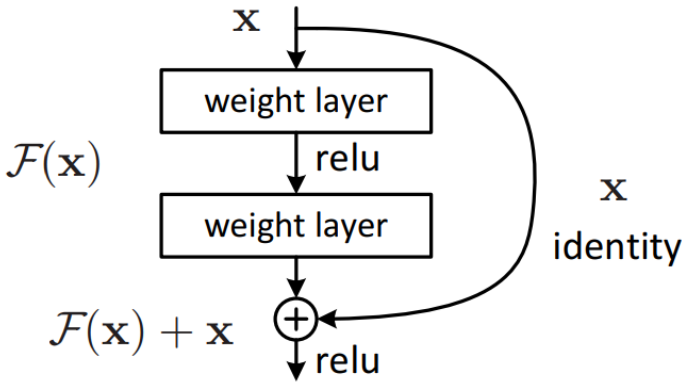


Fig. 2.8. An example of a residual building block with a shortcut connection [61]

ResNet provides a scalable architecture for building a deep neural network topology. Although in order to lower training time and the complexity of the model bottleneck blocks with 1x1 convolutions are introduced as a replacement for standard building blocks. Fig. 2.9 gives an example of a building block used for ResNet with 18 and 34 layers, and a bottleneck block introduced for 40, 101 and 152 layers ResNet versions.

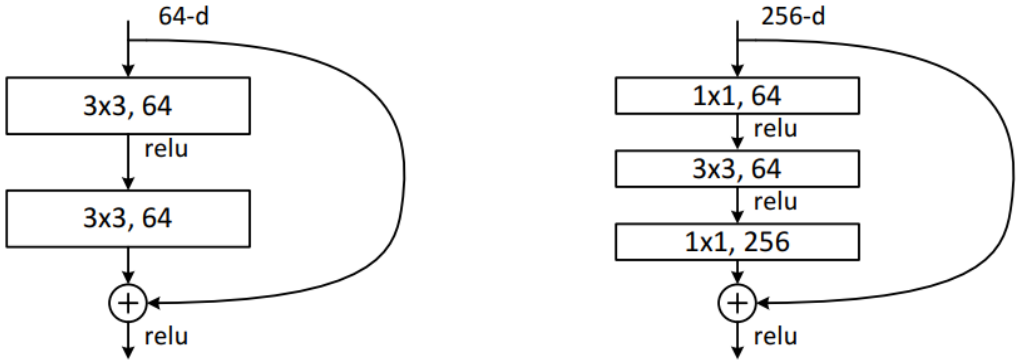


Fig. 2.9. Comparison of a building block (left) and a bottleneck block (right) used in ResNet topology [61]

The proposed approach of building blocks enabled further increases of neural network depth without degradation of the prediction quality. According to the authors, the presented architecture allowed to train a ResNet architecture with over 1000 layers with no optimization difficulty on the method's side.

2.2.2 Recurrent NN

Another path in the neural network development domain has been directed by the data which contains a strict time relation between the following input samples. In many input formats such as audio, video or time labeled statistical data, the sequential characteristic is crucial for its understanding. Recurrent neural networks (RNN) were developed to address this issue [53]. The architecture of RNN strongly depends on maintaining a hidden state of a neuron and is based on the previous output. Such functionality is possible due to recurrent connections in the network,

allowing for applying information from the previously seen information to a present input. Fig. 2.10 presents an example of simple RNN architecture with one hidden layer.

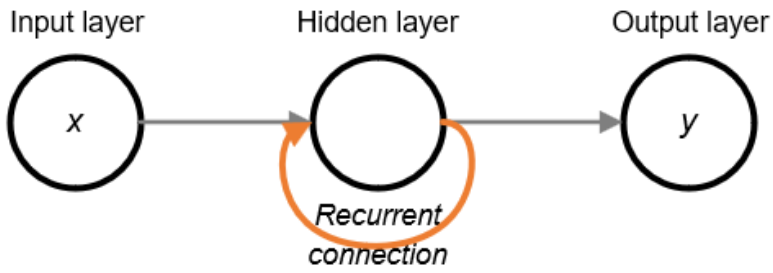


Fig. 2.10. Example of a simple Recurrent Neural Network

Although powerful, RNN often struggles with learning long-term connections in the provided data. Moreover, due to sequence dependent characteristics, backpropagation for RNN networks needs to be extended, this mechanism is known as backpropagation through time (BPTT) [62]. The RNN design also struggles with problems of vanishing and exploding gradient due to a fact that a back-propagated error either grows or shrinks in every calculated time step [63].

The mentioned RNN related issues and the need for more flexible adaptation to sequential data lies at the architecture of Long-Short Term Memory (LSTM) network. Although this topology provides additional complexity on the design itself, it proved to be extremely efficient for sequence data where crucial information is widely spread through time as in natural language processing [63]. The basic building block of an LSTM layer is a memory block which stores two states of the unit. The first one called cell states plays a role of long-term memory, the second one called hidden state is treated as short-term memory. Such a mechanism is achieved with the application of three gates responsible for forgetting data, storing information in memory, and adapting the output based on the cell's memory [64]. The implementation of gates is commonly accomplished by using a sigmoid function. Fig. 2.11 gives an example of a single LSTM memory cell.

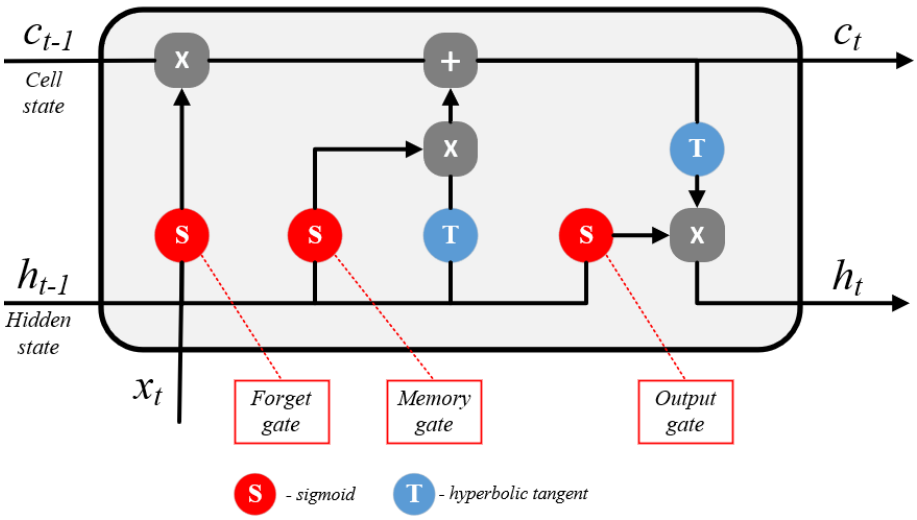


Fig. 2.11. An example of a single LSTM memory cell

There are multiple implementation variants when it comes to LSTM architecture [65]. Moreover, a simplified version of the gating mechanism has been proposed in the Gated Recurrent Unit (GRU), which gained popularity in the research community [66]. The domain of NN architecture is still evolving, often new designs are proposed and experimented on. The one that gathers much of the attention in the context of processing sequential data is transformer [67].

2.2.3 NN parameters

Regardless of the size or topology leveraged by a particular NN architecture, the core implementation depends on simple mathematical operations. Matrix addition and multiplications stand for the majority of calculations required for both forward and backward passes through the network [68]. To achieve a sufficient dynamic range of variables for weights, biases and activations most current implementations depend on IEEE-754 32-bit floating-point representation available in general purpose hardware [69]. This case is especially important during NN training where a vanishing gradient problem might be intensified with lower precision variables.

The need for a high number of floating-point multiplications has a direct impact on computational requirements during neural network training. Floating-point operations are one of the most power demanding hardware operations. This not only translates to high computational demands while training large architectures but also the time required for finishing such a process. These two factors create difficulties not only related to financial effectiveness but also impose a long time of experimentation and tuning for research and development tasks [70].

Computational requirements are not the only issue related to the usage of 32-bit variables for NN. In the scale of million parameters, reserving 4 bytes for each parameter can create problems with storing the network itself, especially in case of low-power devices or chipset's internal memory. Even if the instance of the network is stored in the backend it usually takes more than hundreds of megabytes. VGG-19 with a size of 550MB is a good example of such a case [57]. Memory issue is much more crucial in terms of runtime memory requirements, in the majority of cases inferring the network requires loading the trained topology to a runtime memory which is much scarcer than the regular storage [71]. This is often a blocking constraint for deploying larger topologies on edge and mobile devices.

2.3 Floating-point representation

Digital systems enforce a binary format for representation of numeric values. Storing them in a form of ones and zeroes hardly ever easily translates into a commonly used decimal system. Moreover, due to limited variable's bit count there are strict constraints when it comes to storing numerical data in digital memory. Over the years, engineers came up with multiple formats for addressing this issue [72] [73]. Although, translation of integers to binary system is straightforward, complications appear in the case of real numbers.

There are two common ways for storing real numbers in digital variables, floating-point and fixed-point [74]. The first one dedicates a specific number of bits for integers and factorial

parts of the number, which makes the implementation less complicated. This translates to economical savings due to lower hardware cost, power and time of computation [75]. Such an approach, however, not computationally complex, limits the dynamic range of numeric values that could be stored on a particular number of bits. The answer to this issue, coming at a cost of additional complexity, was a floating-point representation which stores a number in a form of the exponent and mantissa. This format can be depicted in the form of the following equation.

$$x = S \cdot M \cdot B^{E} \tag{2.2}$$

where:

x – floating-point numeric value,

S – sign of the value,

M – mantissa,

B – base of the number system, two for binary,

E – exponent.

Fig. 2.12 shows a bit level comparison between an 8-bit fixed-point and 32-bit floating-point formats representations for numeric value 6.75.

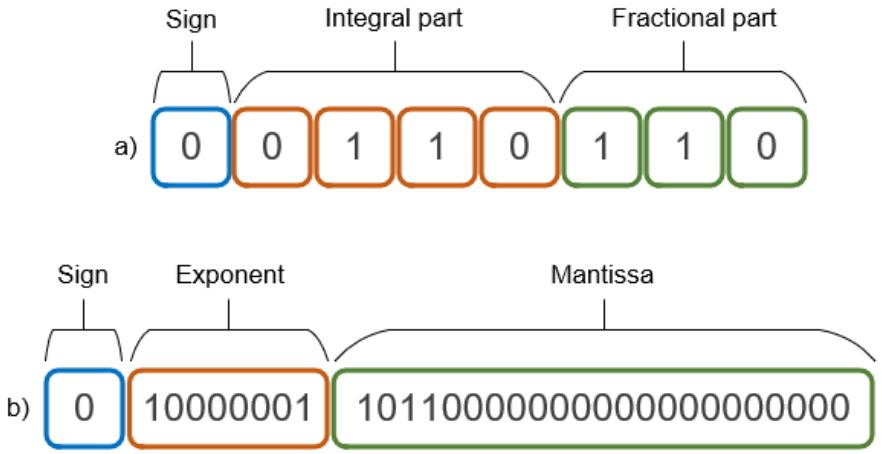


Fig. 2.12. The number 6.75 represented in a) 8-bit fixed-point with 4-bits integral and 3 bits fractional part b) IEEE-754 32-bit floating-point

Values distribution is another important factor when it comes to a number format type applicability to an optimized NN training algorithm. As presented in Fig. 2.13 fixed-point type uniformly distributes its values, the difference between adjacent numbers is always equal to one. In contrast, floating-point values are distributed non-uniformly providing more representations of values closer to zero. Moreover, a denormalized range of values specific for floating-point implementations provides an additional numbers' representation that otherwise would be rounded to zero [76]. Such characteristic may be especially important in case where NN training requires multiple low-value gradient updates.

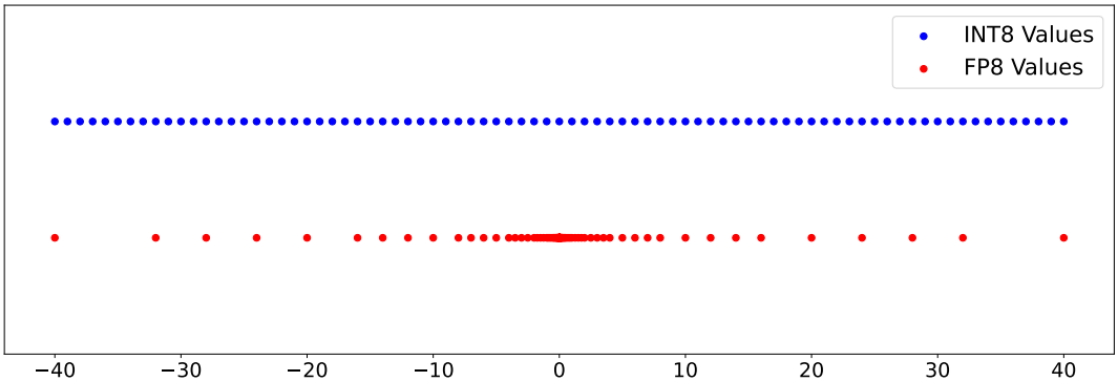


Fig. 2.13. Values distribution in 8-bit floating-point (FP8) and 8-bit fixed-point (INT8) variable [77]

2.3.1 IEEE-754

The common standard for general purpose computing is IEEE-754 floating-point providing definitions for 16-, 32-, 64- and 128-bit formats. Higher bit counts which are a multiplication of 32-bits are also included in the standard [78]. It is mostly appreciated for its high dynamic range and ease of use when it comes to software implementation. Due to wide global adaptation, most modern processors contain floating-point processing units (FPU).

The format of a single-precision 32-bit floating-point variable contains:

- 1-bit sign (set to 1 if the number is negative),
- 8-bit exponent with a base of 2,
- 23-bit mantissa.

The 8-bit exponent is split into a range of $< -126, 127 >$ with a bias equal to 127. The format includes special representations for zero, infinity and “not a number” (NaN) values. In order to support a wider range of close to zero numbers it introduces denormalized values, also known as subnormal values, which interpret the leading hidden bit of mantissa as 0. Thanks to this feature, which is achieved at the expense of significant mantissa’s bits, it is possible to limit underflow cases as limited exponent range could be easily exceeded during floating-point arithmetic.

The standard had its beginning in 1985. It specified formats, rounding, exceptions and operations for floating-point arithmetic. Before then, multiple available hardware architectures defined their own arithmetic, forcing engineers to support and maintain cumbersome conversion mechanisms. The IEEE-1987 revision introduced radix-independent floating-point arithmetic. Other important updates were submitted in 2008 covering binary and floating-point arithmetic, extensions of types, supplementary functions and attributes. The latest changes were included in the IEEE-754 2019 providing among others optional augmented arithmetic calculations. It is argued that future revisions may provide more machine learning focused updates allowing for sacrificing precision over pure accuracy [79]. Table 2.1 gives an example of several bit variants of floating-point types.

Table 2.1. Comparison of several floating-point types [80]

Type	Bit count details	Min (normalized)	Max	Unit roundoff
16-bit brain float (bfloat)	8-bit exponent 8-bit mantissa	1.18×10^{-38}	3.39×10^{38}	3.91×10^{-3}
16-bit floating-point	5-bit exponent 11-bit mantissa	6.10×10^{-5}	6.55×10^{4}	4.88×10^{-4}
32-bit floating-point	8-bit exponent 24-bit mantissa	1.18×10^{-38}	3.40×10^{38}	5.96×10^{-8}
64-bit floating-point	11-bit exponent 53-bit mantissa	2.22×10^{-308}	1.80×10^{308}	1.11×10^{-16}

Selecting a specific variable type for neural network representation has a direct impact on its resource requirements. This trend has been especially important in case of inference optimization when quantization to smaller, often fixed-point, formats enabled minimization of latency or output size of the model [29]. Although FPUs are commonly available in modern processors for both GPUs and CPUs, there are still devices that can benefit from using fixed-point arithmetic, digital signal processing (DSP) units are a good example of that [81]. In case of small low-power devices, various factors such as speed, power consumption or chip's area are crucial requirements for the final productization.

Consideration of pros and cons regarding usage of floating- and fixed-point representation is not a new problem. Over 25 years ago (Inacio & Ombres 1996) [82] described their point of view for selecting one of these numeric types to DSP implementations. The domains that were considered are not so different from those investigated today. The major factors included cost of the mathematical unit, number of cycles required for computation, ease of use and software support. Today we consider the same aspects in order to efficiently execute ML specific computations [77].

Binary representation of a real number format is not the only factor that impacts calculations' precision and performance. Besides accessibility to hardware computation units that can be optimized to support chosen formats of numeric operations, a crucial role is played by the size of variables that are used. Despite the precision-wise disadvantages of this solution, limiting bit count of variables is a straightforward method to limit both computational complexity and memory consumption when it comes to neural network training and inference [40]. Even modern neural network frameworks such as Tensorflow [83] or Pytorch [84] introduced similar mechanisms to enable neural network training on 16-bit half-precision or lower floating-point variables [26] [85].

Considering the bit-width of used variables, interesting research on custom floating- and regular fixed-point usage has been conducted by (Barrois & Sentieys 2017) [75] in a relation to k-

means clustering. The proposed custom floating-point format loosens some IEEE-754 restrictions on normalization or special values. Their comparison showed a significant overhead in case of floating-point energy consumption which needs 5-12x more energy for adders and 2-10x in case of multipliers. However, when both numeric types were applied to k-mean clustering task, the 8-bit floating-point algorithm was 80% more accurate than the 8-bit fixed-point with only a 1.6 energy increase. As the increased accuracy enabled the algorithm to converge faster, the overall energy cost of using fixed-point was higher. It is important to highlight that the advantage of floating-point variables was not observed for higher bit counts than 16-bit. In such scenarios 16-bit fixed point was a more efficient choice. The authors see such a scenario as an opportunity for development of energy-efficient microcontrollers with small bit-width floating-point variables as a compromise between accuracy and energy consumption.

A few years later Zhang et al. (2023) presented research in the same domain as Barrois & Sentieys (2017) [75] with focus on low-bit fixed-point and floating-point comparison in relation to large language models quantization. In their proposal of Mixture-of-Formats Quantization (MoFQ), the authors proved that although floating-point support translates to higher hardware cost due to required area size, the difference between fixed-point and floating-point decreases along with the limited bit count of the supported variables (Fig. 2.14). In terms of 8-bit variables, the overall required multiply–accumulate (MAC) area is comparable for both types.

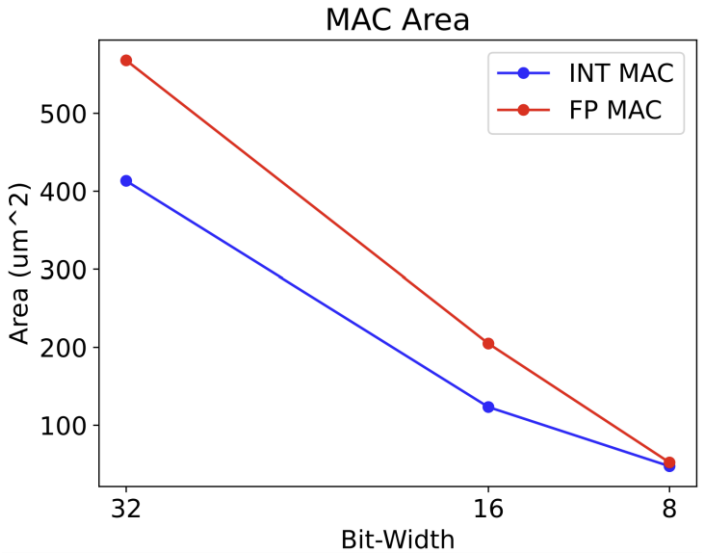


Fig. 2.14. Required area size of fixed-point integer and floating-point operators across various bit widths [77]

When it comes to power efficiency, an interesting study conducted by (Tong et al 2000) [86] verified the idea of floating-point variables limitation with an aim of energy savings. Their implementation of optimized floating-point representation included the change of the implied radix, simplification of rounding modes and most importantly a reduction in mantissa and exponent bit-width. The presented work confirmed that energy per operation increases linearly with growing bit count of operands. As presented in Fig. 2.15, an 8-bit multiplication consumes 78% less energy in comparison to 24 x 24 Wallace tree multiplier [18] used as a baseline. Even in the case of limiting mantissa to 16-bit, the energy consumption was 32% lower.

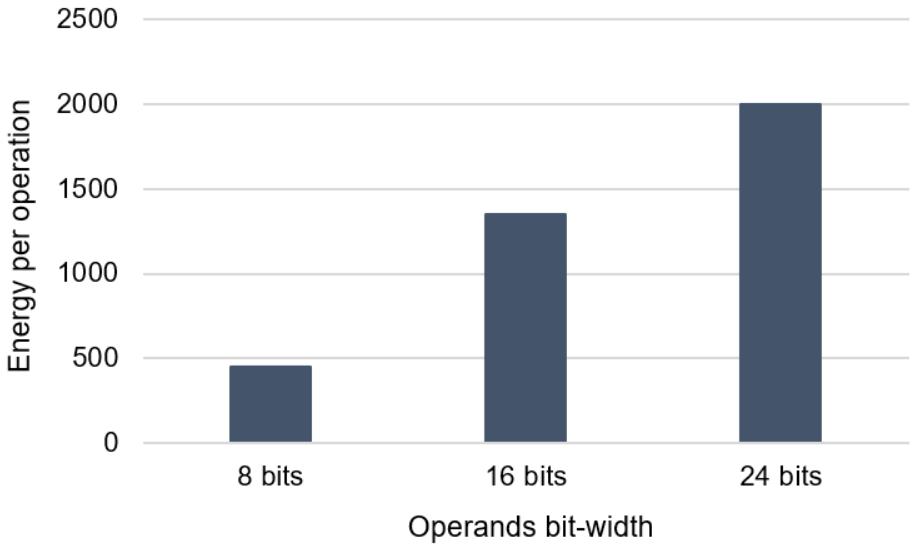


Fig. 2.15. Performance of the digital multiplier across selected floating-point variables bit-widths [86]

The above results should be interpreted along with power consumption per functional block in a single precision multiplier. As presented in Table 2.2, the authors measured that over 80% of power is used by the unit responsible for mantissa multiplication. These results show that limiting mantissa bit count may be especially important in terms of power efficiency for precision limited NN training. Even earlier studies of (Meier et al. 1996) [87] and (Callaway et al. 1997) [88] confirm that limitation of variables' bit count translates to a significant reduction of the multiplier's power consumption.

Table 2.2. Power consumption of a single precision floating-point multiplier [86]

Functional Block	**Power consumption (% of total)**
Mantissa Multiplier	81.2
Rounding Unit	17.9
Exponent Unit	0.833
Others (exception handling etc.)	0.066

2.3.2 Precision and rounding

The uniform placement of a decimal point in a fixed-point format allows for more natural translation of real numeric values to binary representation. Although such simplified representation may require less complicated hardware arithmetic, it significantly reduces the dynamic range of a numeric variable [69] . Leveraging fixed-point representation may also create more complication on the software development side when it comes to handling overflow and underflow scenarios. Additionally, fitting real values into fixed-point arithmetic may especially suffer due to quantization noise caused by enforced rounding. Such a problem has been already investigated as early as in 1993 by (Choi et al 1993) [89] who signaled the issue of accumulating arithmetic rounding and quantization errors with large feedforward neural networks.

It is worth mentioning that rounding related issues may be a vital problem in case of several aspects of neural network applications, especially in domains where correctness of inference results is an aspect of security. The work presented by (Jia & Rinard 2021) [90] states that floating-point error might be used to exploit real-valued neural network verifiers. Such cases are especially important where space of the input of the model is not constrained or limited.

The necessity of allocating analog numerical values into a limited number of bits, forced engineers to apply various rounding techniques on digital numeric representations [5]. Although, rounding to nearest is sufficient for most common use cases, it may pose a significant problem in the case of neural network training where rounding errors tend to accumulate over time. Hence, stochastic rounding rose in popularity in recent years around deep learning researchers.

Wide application of quantization and precision limitation in the ML field pushed researchers in the direction of better ways of handling rounding and truncation errors. The standard to-nearest method, which is a default for IEEE-754 floating-point arithmetic, introduced many issues due to accumulating errors over the time of training or inferencing the network, especially in case of low bit-width variables. This reinstated interest in the stochastic rounding method that was initially proposed in the 1950s [91].

Stochastic rounding, in contrast to the to-nearest technique, proposes a non-deterministic approach to rounding numbers. In general, two flavors of the method can be distinguished. The first one randomly rounds the number up or down with 50% probability. The other one, more commonly used in low-precision machine learning computations, determines the direction of rounding based on the number's relative distance to the nearest upper or lower boundary [80].

The following equation sums up the second version of the stochastic rounding algorithm, which is also presented in Fig. 2.16.

$$r(x) = \lfloor x \rfloor + p, where\ p = \begin{cases} 0 & with\ probability\colon\ \ 1 - (x - \lfloor x \rfloor) \\ 1 & with\ probability\colon\ \ x - \lfloor x \rfloor \end{cases} \tag{2.3}$$

The following alternative of this equation can be considered:

$$r(x) = \lfloor x + u \rfloor, where\ u \in [0,1)\ with\ \ uniform\ distribution \tag{2.4}$$

which emphasize hardware design efficiency improvement relying on the possibility of using a random bit stream generator for the generation of binary representation of the stochastic parameter u.

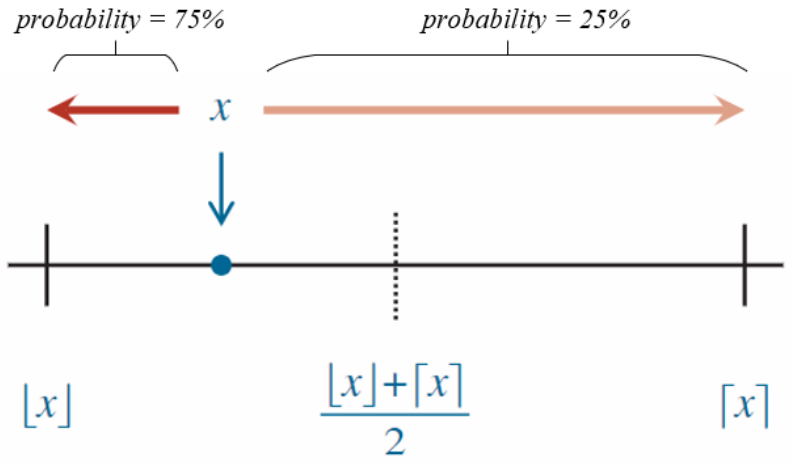


Fig. 2.16. Stochastic rounding with distance-based probability [92]

Although the error with this approach can be larger in a singular case than with to-nearest rounding, the statistical characteristic enables the reduction of accumulated error on a larger scale. Fig. 2.17 provides a comparison of accumulated error between 16- and 32-bit IEEE-754 floating-point, and 8-bit floating-point with stochastic and to-nearest rounding. The experiment involved multiplication of two randomly generated vectors containing one hundred 64-bit floating-point elements each. The average limitation and rounding error have been calculated for each iteration.

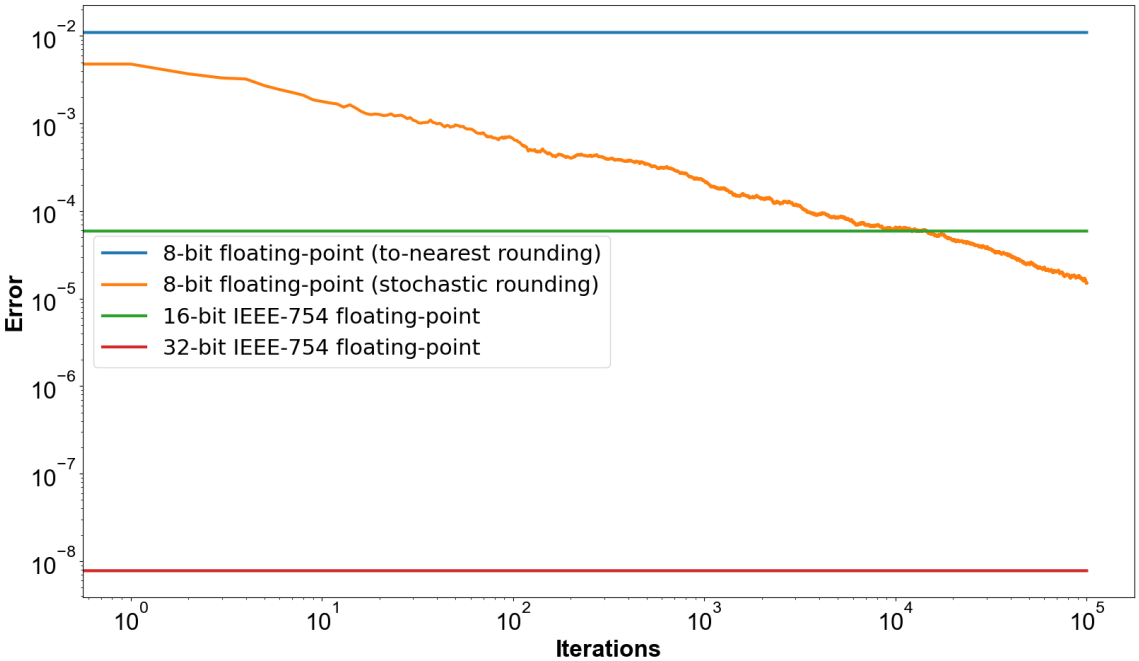


Fig. 2.17. Comparison of average error after multiplication of two vectors with 64-bit floating-point variables for various variable types and rounding techniques

Based on Fig. 2.17 it can be observed that along with increasing test iterations the average error of 8-bit floating-point with stochastic rounding is decreasing and after 100000 iterations gets lower than the one for 16-bit IEEE-754 floating-point. On the other hand, types with to-nearest rounding maintain a constant level of error. As explained by (Connolly et al 2021) [80] rounding errors produced by stochastic rounding are mean independent. Additionally, it allows to

avoid stagnation which is especially important for tiny parameters updated in NN. Random based rounding increases a chance for error cancellation, especially in case of low precision numbers where mantissa bit count is highly limited.

An important aspect that needs to be considered when it comes to the application of additional rounding techniques is their overhead on the overall algorithm. Although accuracy results may improve due to the rounding, it is important to not sacrifice possible efficiency gains. This opens a subject of hardware support for efficient stochastic rounding. First hardware implementation of stochastic rounding was presented in the 1950s [93]. Researchers continue the work on this subject with dedicated stochastic rounding accelerators which may make software implementations redundant in the highly efficient implementations. For example, the proposal presented by (Mikaitis 2021) [94] focuses on algorithms and hardware-based acceleration for various fixed-point types commonly used in ML. The solution includes mixing of the used formats and, as the authors suggest, it should be applicable to floating-point arithmetic adders and multipliers.

The review of patents and devices done by (Croci et al 2022) [92] shows that multiple major hardware manufacturers own patents or products supporting stochastic rounding. Graphcore IPU parallel machine learning accelerators include stochastic rounding of 32-bit values to 16 bits [95]. IBM patents include using stochastic rounding for floating-point adders and multipliers [96]. AMD depicts methods for using stochastic rounding for integer adders and accumulators for 32- and 16- bit mixed precision [97]. Similar methods are presented by NVIDIA for 64-, 32- and 16- binary and floating-point types [98].

The number of published papers using stochastic rounding with limited precision confirms its advantages for neural network training. The method includes experiments on 12- and 14-bit fixed point variables [5], 12-bit floating- and fixed-point with additional context representation [6] or dynamic precision scaling for 14- and 16-bit fixed-point [7].

2.4 Neural Network Acceleration

Regular purpose hardware rarely provides optimal performance when it comes to specific calculation tasks. Modern NN architectures give good examples of structures that require significant computational power and large amounts of high-speed memory. These constraints were the major reason behind the rapid shift of researchers and industry from standard CPUs to more efficient GPUs [68]. The parallel computation provided by GPUs proved to be a perfect fit for huge amounts of multi-add floating-point operations required for neural networks training. Although such a change usually results in an order of magnitude improvement, the necessity of having a powerful GPU on the device limits the possible productization use cases especially when AI models have doubled the usage of computational power every 3.4 months since 2012 [99].

Recent years have shown increasing interest in neural network-based solutions for low-power devices. The growing domain of mobile and edge devices enforced researchers to avoid pursuing the best possible classification accuracy at any cost. From now on, computational and

memory requirements became an important factor in terms of NN architecture's productization potential. As presented by Dhar et al. (2020) [100] it applied not only to the hardware design but also algorithms, software ML libraries and general NN learning theory. New factors, closely related to mobile devices, as battery life and cost of additional hardware components had to be considered in the perspective of AI-based features introduction.

In the face of new requirements, both researchers and hardware designers responded with new ideas. Multiple software techniques focusing on trimming or compressing existing neural network architectures have been proposed and rapidly adapted by the industry. The same is true for the hardware side, where the era of AI accelerators has already begun [40].

2.4.1 Inference acceleration

The inference is the most common, user faced functionality provided by a neural network. Once the network is trained, it can be deployed on a variety of devices as a part of a larger software application or cloud-based solution. Then the results, for a specific input data, can be generated with a feedforward mechanism. Such scale of adoption was followed by multiple software and hardware acceleration techniques for the inference, this section presents a few of the most popular methods.

Most software techniques for neural network acceleration depend on modifications of an already trained full-precision model. The aim is to generalize or remove non indispensable information from the network without heavy impact on the final classification accuracy. The most common approaches include:

Pruning is aimed to reduce the size of the network by its parameters removal. There are various ways on how it can be applied to an already trained neural network. Two most common techniques focus on weights and neurons [101]. The idea is that connections with weights below a particular threshold have a smaller influence on the final network's prediction, hence they can be removed. Similar approach applies to nodes, if the resulting activations are low or close to zero then there is a big chance that a particular neuron has a minor role in the inference outcome. It is worth highlighting that the fine-tuning approach can be also applied to pruning. The pruned network can be retrained to recover some of the lost accuracy [102]. In many cases, removal of weights or nodes produce a sparse network (Fig. 2.18) which may result in computational inefficiencies with the usage of general-purpose hardware or modern machine learning libraries.

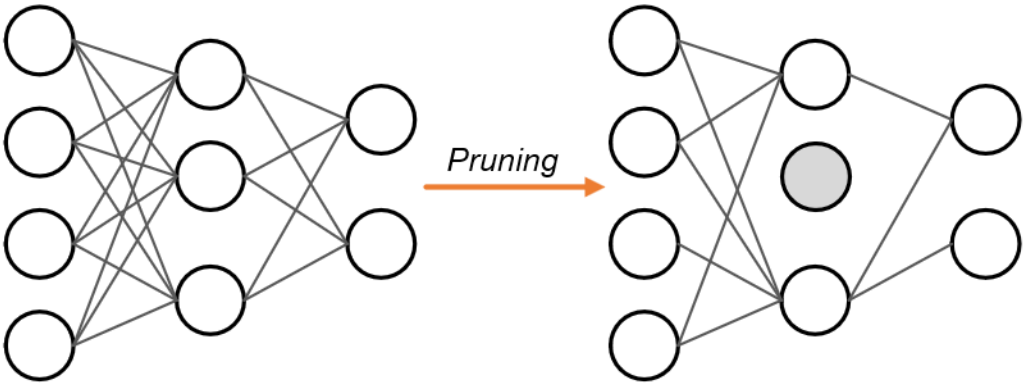


Fig. 2.18. Example of neural network pruning

Quantization is one of the most common inference acceleration techniques (Fig. 2.19). The method relies on replacing regular 32-bit floating-point parameters with low-bit integer variables, however floating-point variants are also considered. It leverages the fact that, in general, fixed-point operations are much more efficient on general purpose hardware. Post Training Quantization (PTQ) is a common technique that modifies an already trained neural network, which no longer requires a high dynamic range for backpropagation steps [103]. In order to limit quantization error, many frameworks already provide an option of Quantization Aware Training (QAT) that emulates quantized inference during training time in order to prepare the model for the quantization step [104]. The usual target for quantization is 8-bit integer but there are several studies showing ways to compress a network to 4- or lower-bit fixed-point variables [103]. The extreme case in terms of quantization is represented by binary neural networks which store parameters values on a single bit [105]. Such architectures enforce changing the regular neural network mechanisms to bit level operations.

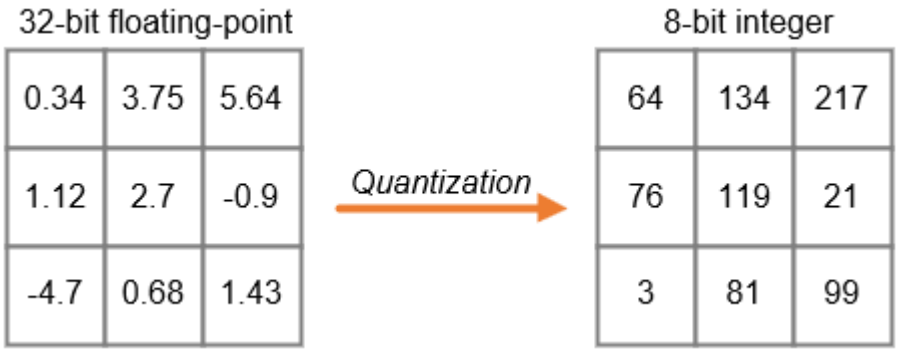


Fig. 2.19. Example of neural network parameters quantization [104]

Weight sharing is another technique focusing on limiting the number of parameters in the neural network and reducing redundancy. As the name suggests, it relies on reducing trainable parameters in the network by sharing them between multiple nodes [106]. The most common scenario for weight sharing applications is using the same weights across convolutional filters. The other case may include sharing weights between initial layers of single or multiple neural networks.

Knowledge distillation aims to improve resource efficiency of inference at the cost of more complicated training steps [107]. It uses a large neural network to supervise and train smaller topology with similar accuracy results. This technique is commonly referred to as the teacher-learner approach [108]. It provides a utility to train a smaller neural topology that could not be trained from scratch on the same dataset as the teacher model. The main advantage of this method is minimizing entropy and the distance between probabilistic estimates of the network and in result compressing the final NN model. Fig. 2.20 gives an example of this technique.

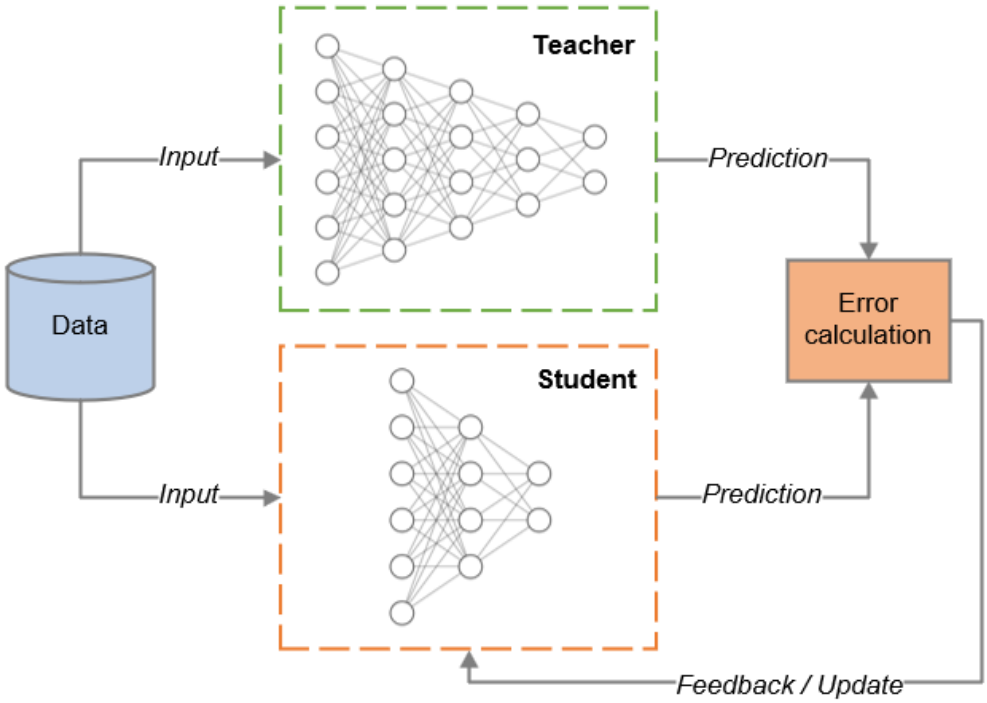


Fig. 2.20. Example of a teacher-learner training technique

In the pursuit for the best accuracy to efficiency ratio for neural network inference, a lot of the above methods are used jointly. There are various examples of such approaches as Han et al. (2015) [29] where pruning, quantization and Huffman coding are combined to compress CNNs. In another example Tung et al. (2018) [109] presents CLIP-Q method that leverages in-parallel pruning and quantization for networks such as AlexNet, VGGNet, GoogleNet and ResNet.

Along with the software proposals, there has been continuous development on the hardware side of NN acceleration. Using more efficient general-purpose devices was not always an option to limit the latency or energy consumption of a particular model. This opened an opportunity for custom neural accelerators and hardware optimizations aiming for better computational parallelism and memory access reduction.

Such a trend is already visible on the market and supported by major corporations dedicated to AI development. A few years ago, Google introduced its TensorFlow Processing Unit (TPU) [10] and Nvidia followed with Deep Learning Accelerator (DLA) [11]. Other vendors as Intel, Samsung or Qualcomm provide specialized Neural Processing Units (NPU) for AI related workloads [27]. The key idea is to provide specialized, highly efficient hardware for common ML

tasks as convolutions, matrix multiplications or activations. Depending on the hardware placement, ML accelerators can be briefly divided into the following groups.

On-chip accelerators which aim to offload ML related computation from the main CPU. With the use of multiple Processing Engines (PE), such devices can speed up repetitive matrix-based computations [12]. This is especially important in case of edge and mobile devices where inference needs to be done on the device itself. Often the network is stored on the chip memory, hence there might be a strong limitation when it comes to supported network topologies [100]. Another constraint can be posed by operations supported by a given accelerator which may limit the use of newer topologies or network layers. Various NPUs [110] or a NN specific accelerator RENO [111] are good examples of on-chip accelerators.

Standalone accelerators present a domain of often highly specific powerful devices for ML tasks. Such architecture is not limited by the constraints related to CPU or GPU chip designs and can be used in separation to general purpose hardware architecture. Additionally, standalone accelerators are often designed to support both training and inference use cases. The family of DianNao devices [112] or TPUs [113] shows benefits of such architectures in terms of deep learning acceleration. Many standalone accelerators leverage field programmable array gates (FPGA) or application specific integrated circuits (ASIC) for its designs, which manifest a better performance density than GPUs despite lower throughput [114]. Zhang et al (2015) [115], Guo et al (2017) [116], Nguyen et al (2019) [117] provide examples of such an approach.

Hardware acceleration can have a significant impact on efficient execution of ML algorithms. This is especially important in case of on-device inference or algorithms that had to be close to the data source or sensors such as smart cameras or smartphones [71]. In many situations, the network throughput is limited, and algorithms response time is crucial for the application use cases. Nevertheless, in general the common way to provide AI based functionality is a cloud base backend leveraging powerful datacenters [33] [100]. This scenario combines a wide range of CPUs, GPUs and ML accelerators that handle ML tasks on a large scale. Many cloud market leaders such as Amazon, Google, Microsoft support AI cloud acceleration [118]. The infrastructure is used for providing computational power behind common AI applications leveraging natural language processing, image recognition or risk identification.

2.4.2 Training acceleration

In contrast to the inference, training acceleration is a much less examined subject. Up to recently, spending days or even weeks on training a particular model was acceptable as long as proper computational power was available [119]. The continuously growing domain of IoT and environment related consequences of energy consumption, put into question the rapidly growing neural network resource requirements. Additionally, usability and privacy aspects force the industry to focus on on-device NN training [33].

Reducing computational complexity of an algorithm has been often a much cheaper way to improve overall efficiency. Upgrading the hardware in order to get additional computational

power can be costly and does not always scale appropriately along with growing input data or topology sizes [100]. The same goes for custom hardware designs which are usually much more expensive and require a long research and development (R&D) phase before entering the market [70]. Due to these reasons, researchers pursue multiple ways of NN training acceleration on existing general-purpose hardware including the following:

Topology and hyper-parameters tuning are common approaches when looking for reduction in NN resource consumption. Selecting a smaller topology is an obvious choice to limit the number of parameters that need to be trained, however, it usually comes at the cost of decreased performance of the final network. Applying additional normalization and regularization techniques may also impact the time required for the network to efficiently converge for a given problem. Tweaking with batch-size or input data size may be an additional factor in speeding up the training process [120].

Transfer learning is an interesting technique which decreases the time and resources required for training the neural network. The key element of this method revolves around using already trained weights of an existing network to solve a different problem, as presented in Fig. 2.21. It has been proved that even if the network has been trained for a different task, the training time required for adjusting the network to other problems is much shorter than starting the process from random weights [121]. Unfortunately, this method has its limitations when it comes to training completely new architectures.

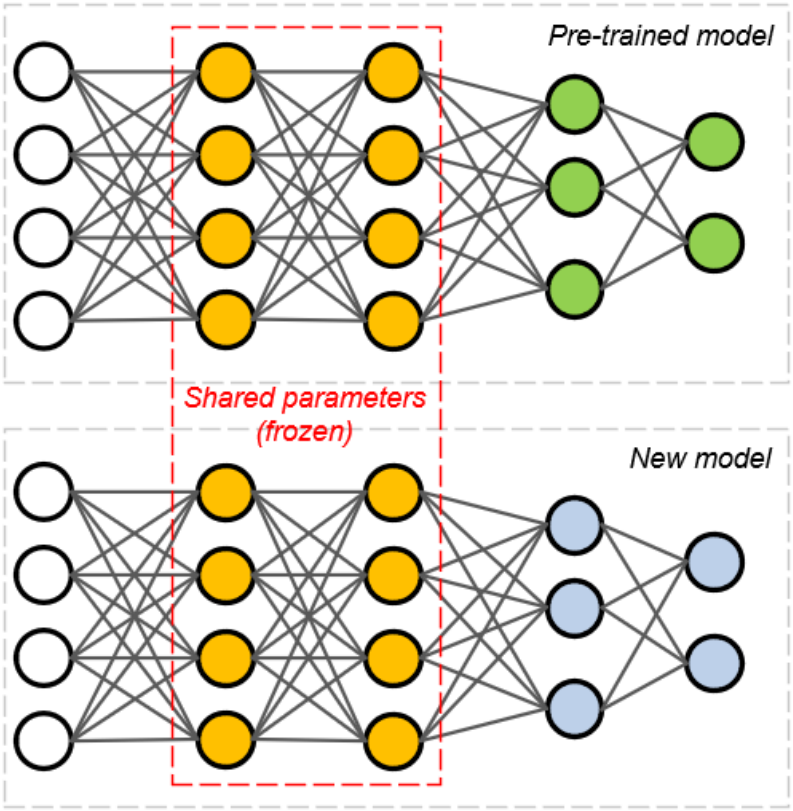


Fig. 2.21. Example of transfer learning

Usage of **half precision variables** is a common optimization technique that focuses on the format of network parameters. IEEE-754 32-bit floating-point is a standard type used for neural network training. The bit count of parameters used in this process has a direct impact on energy consumption and memory required for loading and storing the model [86]. Leveraging half precision 16-bit floating point format for all or part of model parameters is a simple way for resource savings. A vivid disadvantage of this solution is the limitation of available precision and dynamic range of the variable. In case of 16-bit floating-point overflow and underflow scenarios may be much more common, especially during gradient calculations where values can be extremely low. The response to this problem has been proposed by NVIDIA in a form of mixed precision training [122]. The method chooses half precision types for parameters where it does not impact the final accuracy of the network. Additionally, a tool for mixed precision training called "A Pytorch Extension" (Apex) has been developed in order to support this technique in modern ML training frameworks [123].

In a similar way to inference, using specific hardware accelerators can have an enormous impact on NN training optimization. Progressive parallelization and increasing computational power of available GPUs allowed for training big modern architectures in a reasonable time. Nevertheless, resource constraints and long training time limit the ability to experimentation, debugging or touching complex problems in an efficient manner. Increasing the gap between the continuous growth of modern deep neural models and general-purpose hardware brings concerns in terms of future scalability of edge-based AI solutions [124]. There are two main paths when it comes to hardware-based advancements for NN training. The first one revolves around optimization of currently executed ML operations in relation to memory access, caching and data throughput. The other one focuses on allowing network designers for more flexibility in terms of parameters bit count and its precision which often requires specialized software support.

NN accelerators are much less common to support the training stage of the network as it is a much more complex task. However, there are examples of architectures which support this use case [27]. Google TPU [10] is one of them, providing its functionality via cloud infrastructure. Another one is presented by Tensorflow Accelerated Linear Algebra (XLA) which optimizes GPU operation for specific NVIDIA hardware [125].

Low precision operations support is a crucial advancement when it comes to supporting a power efficient NN architecture. Fixed point integer values are often not sufficient for inference of large topologies, the more for their training. Moreover, the general-purpose hardware usually supports 16-bit variables as the smallest floating-point format. IEEE-754 is not an ideal type for neural network training, hence enabling experiments on smaller and custom floating-point representation is an important step to low-power network architectures [122]. Several design proposals of custom precision accelerators have been presented in the literature, along with custom precision support as in (Lee et al 2018) [15]. Additionally, such research may enhance development on efficient binary neural networks.

In general AI hardware supports high-bit floating-point operations and low-bit fixed-point operations [77]. Nevertheless, the shift in this approach is noticeable as the global GPU manufacturers as NVIDIA does not stay behind custom NN accelerators. The recently introduced NVIDIA H100 Tensor Core GPU provides support for efficient 8-bit floating-point operations, considering the types with 4-bit and 5-bit exponents with comparable performance to the 8-bit fixed point [126]. Such products show that focusing on training algorithms for NN with low-bit floating-point variables may be especially fruitful when it comes to the future energy efficient ML development.

2.4.3 Resource demand

Despite hardware and software advancements in the field of NN acceleration, the rapid growth of future state-of-the-art models' performance may be no longer sustainable due to the continuously growing computational demand [119]. Many novel breakthroughs in the field of image classification, voice recognition or text generation came from incremental growth of resources used for the model's preparation. It is especially visible in the case of hardware applied to NN training. The advancements over the years were often related to faster CPUs availability and then overall switch to GPUs. Once using GPU was not enough, then the era of ML accelerators began with the additional increase of devices used in the process of multi-GPU training [68]. Finally, the largest, most complicated topologies were pushed out to the cloud due to their extreme resource requirements [127].

According to (Thompson et al. 2020) [119] over-parameterization of deep learning models is a key factor contributing to AI sustainability issues as it strictly depends on the growing number of network parameters and input data points. The cost of model training scales with the product of its parameters and data points in at least quadratic scaling, highly limiting performance improvement of existing deep learning architectures. ImageNet competition focused on image classification task can be a good example of this phenomenon [56]. Fig. 2.22 presents ImageNet state-of-the-art models along with their number of operations required.

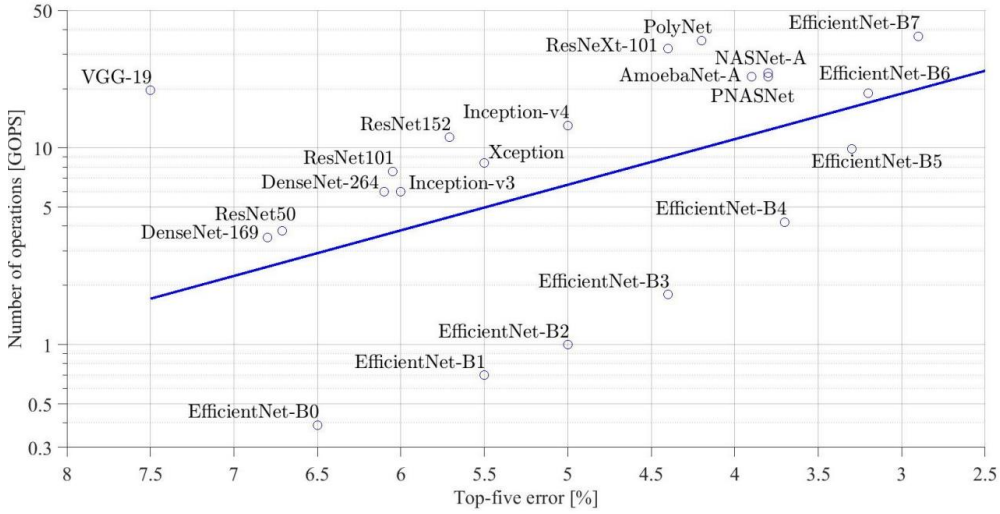


Fig. 2.22. ImageNet competition top-5 error in comparison to the number of operations required by the NN [114]

The continuous increase in the image classification accuracy is followed by the growth of the model itself. This is especially vivid in the case of the newest models as CoCa where the number of parameters almost doubles for a slight accuracy improvement [127]. This subject has been deeply investigated by Canziani et al. (2016) [45] in the analysis of practicality of deep neural network models based on the ImagNet competition submissions. The authors argue that key factors such as upper computational boundaries or inference time should be a part of benchmarking for real-life use case models.

Optimization and proper acceleration are potential ways to increase profitability and market adaptation of cutting-edge AI solutions. Nevertheless, growing cost of developing and productization of modern AI solutions may become a possible issue for its adaptation by a broader market. According to recent online publications, keeping a novel ChatGPT [128] chatbot running costs around $100000 a day which may hinder the profitability of such applications in the future [129]. Moreover, the estimations suggested by Thompson et al. (2020) [119] based on their models say that achieving smaller error rates on ImageNet and other benchmark datasets may be financially and environmentally unprofitable. Reducing 4% percentage points of error on ImageNet dataset provides, at least, a major polynomial increase in required computation, CO_2 production and overall economic cost. Table 2.3 presents their summary for the ImageNet benchmark.

Table 2.3. Implication of achieving performance benchmarks on the computational requirements from polynomial and exponential models' projections [119]

Benchmark	Error Rate	Polynomial			Exponential		
		Computation Required (flops)	Environmental Cost (CO_2)	Economic Cost ($)	Computation Required (flops)	Environmental Cost (CO_2)	Economic Cost ($)
ImageNet	Today: 9.00%	10^{23}	10^{5}	10^{6}	10^{24}	10^{6}	10^{7}
	Target 1: 5%	10^{26}	10^{8}	10^{9}	10^{30}	10^{13}	10^{14}
	Target 2: 1%	10^{33}	10^{16}	10^{16}	10^{92}	10^{74}	10^{75}

In the past decades, we have observed an enormous growth in various processor improvements [130]. Deep learning is not the only field where computational power drives the increase in performance. In his work (Thompson 2017) [131] shows that modern computer chips and Moore's Law had a direct impact on productivity growth in the mid-2000s. Similar observation has been made by Thompson et al (2020) [132] in terms of computational power influence on progress in the areas of weather prediction, oil exploration and protein folding. In order to maintain the continuous growth in deep learning and related areas, the computational requirements for new architectures need to be met by the hardware. One of the promising domains for providing such computational capacity is quantum computing, however, it is still an open domain for extensive research [133]. The other one is a possible breakthrough in terms of NN resource consumption, hence experiments in this area are especially important.

The issue of a narrow approach to deep learning development is also investigated by Martínez-Plumed et al. (2018) [134] who states that many advancements are not coming from the architecture or software improvements but are a side effect of computational hardware advancements. The proposal is to look at the AI advancements in a multi-dimensional way instead of following a specific task performance. These would include such aspects as economic value, social value, scientific progress, computer efficiency, data efficiency, automation, reproducibility and generality. In essence, the idea revolves around comparing the resources spent on preparing a particular solution as engineering effort, data preparation, implementation and deployment to the profitability of the outcome.

Although the focus is often put on computational power, the memory limitations are also an important subject. As stated by Gholami et al. (2021) [135] NLP models have been increasing in size by 240x every 2 years, however, DRAM memory growth is only 2x over the same period. Such a situation creates a bottleneck for rapidly growing NN architectures. Similarly, other researchers state that exponential growth of resource consumption cannot be maintained and needs to be substituted with architectural, hardware and methodical advancements. The solution for this problem, depicted as a “memory wall”, was also investigated by Jain et al. (2020) [136] with a novel approach to tensor re-materialization.

Fortunately, examples of more resource focused NN architectures are also present in the deep learning domain. The proposals of MobileNet topologies for vision use cases proves that resource efficiency improvements can be executed also at the architectural level of the model [137]. The family of EfficientNet designs, the successor of MobileNet, is another example of such resource constrained approach to the NN design [138]. Both mentioned architectures proved their strength by winning ImageNet competition.

The global discussion about computational demand raised questions regarding environmental and social impacts of power-hungry deep learning developments. The carbon emission of the largest models seems to be noticed by researchers and loudly stated as a possible issue [107]. Energy consumed by the public cloud providers, and computational requirements of recent NLP models tend to raise questions about the overall profitability of such designs in terms of required resources [139]. According to Strubell et al. (2019) [70] the NLP BERT model training on a GPU is comparable to a trans-America flight in the matter of carbon emission. Patterson et al. (2021) [140] calculated energy use and carbon footprint for several modern deep neural models. The authors admit that not only the training step is a problematic factor. In case of leading AI companies as NVIDIA, Amazon or Google the overall inference cost states for approximately 90% of energy consumed.

Another important aspect of deep learning growth focuses on the financial requirements for developing new models. Along with larger, more computationally complex topologies comes an increased cost of energy and hardware utilization required for model preparation. Table 2.4 provides a few examples of training resource required per model based on estimations of Schwartz et al. (2019) [107]:

Table 2.4. Selected neural network models comparison in terms of training hardware, time and application type [107]

Model	Device	Time	Type
BERT-large	64 TPUs	4 days	NLP
Grover	256 TPUs	14 days	Fake news detection
XLNet	512 TPUs	2.5 days	NLP
AlphaGo	1920 CPUs and 280 GPUs	N/A	Playing GO

However, the advancements in the field are important, the question arises if results provided by some of the new, extremely large neural network models are justifiable. Often a small benchmark improvement in specific classification task is achieved only with disproportional scaling of the network's architecture [86]. Huge investments required for creation of the state-of-the-art models might have a negative impact on the deep learning field in general. The ability to work on such topologies is currently limited by access to resources and can be continued only by the largest companies with their own cloud infrastructure. Such a situation can limit the possibility of scientific discoveries by smaller, independent companies or the academic community [22]. Future development directions of ML should take into consideration both environmental and financial overhead. As discussed by Patterson et al. (2021) [140], there are several aspects such as increased deep neural network model sparsity, geographic location of ML workloads based on an available energy mix or even improved datacenters infrastructure that can benefit environmental footprint and reduce the overall cost of AI development.

3 NUMERICAL PRECISION LIMITATION IN NEURAL NETWORKS

As already discussed, floating-point multiplications, dominant in the process of NN training, are one of the most power-expensive low-level, digital operations [70]. Limiting the length of variables by reducing their bit count is one of the most effective methods to reduce computational overhead and hardware area required for such calculations [114]. Although plenty of architectures already leverage this approach for inference step [9] [29] [109], NN training with limited precision is still an open subject for the research community [8] [141] [142].

Once the literature review for this specific domain is introduced, then the presented chapter focuses on the impact of floating-point precision limitation on a selected, common convolutional neural networks' prediction quality. The experiments consisted of numerous NN trainings, where parameters such as weights, gradients, biases and activations were constrained in order to fit into low-bit representations. The limitation operation was based on 32-bit IEEE-754 floating-point format and included all possible bit count variants, starting with as low as 1-bit exponent and 1-bit mantissa, up to the total 32-bit limit of the baseline variable.

3.1 Related study

Based on high adaptation of precision limitation for NN inference, various researchers pursued a similar path to optimize the computationally demanding training procedure. The scope of work conducted in this field can be grouped into two general categories of software and hardware designs. According to available literature, training optimization experiments are concentrated on leveraging low bit count for fixed- and floating-point variables. Along with further advancements, the focus has been shifted to mixed-precision NN implementations for both inference and training. Hardware-based inventions and architectural proposals aim to efficiently support the mentioned arithmetic allowing engineers to overcome limitations of general-purpose processors. The inventions in these fields create a promising view for further advancements and overcoming resource related constraints associated with NN training.

3.1.1 Fixed- and floating-point limitations

Software approach to NN training optimization focuses on modified representation or bit count changes of the network's parameters. The approach adopted by Gupta et al. (2015) [5] examined training NN using fixed-point variables with the limited bit count. Their experiments show that 12- and 14-bit fixed-point variables are sufficient for NN training as long as stochastic rounding is applied. The results have been verified on MNIST [143] and CIFAR [144] datasets achieving almost no accuracy degradation in comparison to the 32-bit floating-point baseline. Additionally, their work introduced a proposal of a low-precision fixed-point arithmetic hardware accelerator with support of stochastic rounding.

In a similar way to Gupta et al. (2015), Ortiz et al. (2018) [6] followed experimentation on 12-bit fixed-point parameters. Their work showed that such a limited CNN cannot be trained without accuracy degradation on the CIFAR10 dataset. The results have been improved with

application of stochastic rounding during the training process. This operation allowed to train the mentioned CNN with approximately two percentage points of accuracy degradation. Finally, the authors proposed a context-based float variable format which allowed to improve the 32-bit floating-point baseline results by two percentage points. The limitation has been emulated in the software layer while using 32-bit floating-point variables. It is worth mentioning that in addition to precision limitation experiments, Ortiz et al. (2018) proposed a power of two network which uses only bit level operations for limiting memory usage and computational requirements.

Park et al. (2018) [14] proposed a variation of stochastic gradient descent leveraging Kahan summation to overcome issues with updating low precision parameters. The lazy update method used in this solution allowed the authors to achieve 32-bit baseline accuracy with 8-bit signed integer variables. The results have been validated with multiple datasets as MNIST, CIFAR and SVHN [145].

The work of Fuketa et al. (2018) [146] focused on floating-point variables limitation. The proposed 9-bit floating-point type included a 5-bits exponent, 3-bits mantissa with the hidden most significant bit. The authors were able to achieve accuracy on par with 16-bit floating-point variables. The solution has been verified on two network topologies AlexNet and ResNet-50. The ImageNet ILSVRC2012 dataset has been used as input training data. Along with the results, the authors proposed a hardware design required for supporting the method and its size estimations.

Park et al. (2021) [8] approach to limited precision training is based on a custom 8-bit floating-point type with a shared exponent bias (FP8-SEB). The underlying hardware proposal for this method introduces multiple-way fuse multiply-add (FMA) trees. The FP8-SEB leverages tensor with variables consisting of 1-bit sign, 4-bit exponent and 3-bit mantissa. Each tensor can use a different bias depending on required dynamic range. According to the authors, their hardware proposal requires 78.1 times less power than standard GPU and overhead related to additional biasing is negligible. The provided data shows that results for ResNet-18 on ImageNet achieve 69% accuracy.

3.1.2 Mixed-precision approaches

Various experiments showed that searching for one fit all approach is not always the best path for finding an optimal solution. The precision required from NN parameters is often dependable on their role in the network's topology or phase of the training process. In the spirit of this principle, the technique proposed by Na and Mukhopadhyay (2016) [147] touches both the software and hardware side of the optimization problem by introducing Dynamic Precision Scaling (DPS). The proposed mechanism allows for dynamic adjustment of parameter precision based on its value. In order to address the need for multiplication of variables with flexible sizes, the authors introduced multiplayer-accumulator (MAC) design. According to the experiments' results, this solution allowed to shorten the training time of LeNet and AlexNet networks by a few times.

Another approach on mix-precision floating-point utilization for NN training has been presented by Taras and Stuart (2018) [7]. In a similar fashion to Na and Mukhopadhyay (2016),

their work focused on leveraging DPS technique. The NN has been trained on the MNIST dataset with parameters limited to 14-bits for weights and 16-bits for activations. The achieved accuracy was at the level of 98.8%.

Lee (2020) [148] in his research towards energy-efficient neural network training proposed a fine-grained mixed precision (FGMP) method. In contrast to applying one type of variable to a specific parameter, the author dynamically adjusts the ratio of 8- and 16-bit floating-point types during the training. The aim is to achieve the lowest possible power and memory requirements without decreasing the final accuracy of the NN. Along with the limitation method, a deep learning neural processing unit (LNPU) is proposed, which aims to double the energy efficiency of the training process. The results provided by the author stated that this method allowed to reduce external memory accesses during ResNet-18 training by 38.9%. While tested on CIFAR10 and ImageNet datasets, the accuracy of the ResNet-18 network was on par with 16-bit floating-point baseline.

The path of dynamic adjustment of floating-point variable bit count has been also followed by Rios et al. (2021) [149]. In this case, the technique combines regular 32-bit floating-point type with brain floating-point half-precision type. The author claims that 16-bit type stands for up to 96.4% of all computations required during the training. The method achieved results close to 32-bit floating-point baseline on AlexNet, Inception and ResNet-50 architectures.

The Cycling Precision Training (CPT) developed by Fu et al. (2021) [141] relies on initializing the training process with low precision variables and incrementation of their bit count along further iterations. The main idea behind this method states that a parameter's precision can be treated as a hyper-parameter in a similar way to the learning rate. Low bit count of initial training epochs aims to improve generalization abilities of the trained NN. The results have been verified on multiple topologies as Transformer, LSTM, ResNet and MobilNet. According to the authors, the achieved accuracies were on par with 32-bit floating-point baseline.

Another idea of dynamic precision adjustments for internal NN parameters has been established by Yu et al. (2022) [142]. The proposed Learnable Dynamic Precision (LDP) framework uses additional parameters for selecting optimal precision for each network layer. According to conducted evaluation on multiple ResNet models, their results surpass both SBM [150] and CPT [142] methods.

Junaid et al. (2022) [151] proposes a combination of 32-, 24- and 16-bits floating-point parameters for mixed precision neural network training. The research includes an additional hardware accelerator engine, which allows for reduction of energy consumption by 3.91 in comparison to regular 32-bit floating-point architecture. The results have been verified on a CNN with MNIST dataset providing 93.32% accuracy in comparison to 96% 32-bit baseline.

In their work (Micikevicius et al. 2022) [152] investigate two 8-bit floating-point variants with 4-bit exponent and 3-bit mantissa, and 5-bit exponent and 2-bit mantissa. Although the 5-bit exponent type follows IEEE-754 convention, the 4-bit exponent type is modified by removal of

infinite representations and one of the mantissa patterns for NaN. The authors provide the results on a wide scope of NN topologies where their method's accuracy is on par with the 16-bit training baseline. Both presented 8-bit floating-point types are used depending on the chosen network topology, but the proposed direction is to use 5-bit exponent for gradients and 4-bit exponent for weights and activations. The 8-bit tensors were simulated by clipping the original 16-bit values to a target type, including an additional scaling factor and saturation, and then once again converted to a 16-bit floating-point type. According to the authors, the output tensors were represented in higher precision.

Another research on 8-bit floating-point utilization for NN training has been presented by (Noune et al. 2022) [153]. They consider multiple alternative formats including those with 3-, 4- and 5-bit exponents. In addition to precision limitation, the bias offset is considered as a replacement for a fixed scaling factor. Moreover, only one representation is used for Inf and Nan special values. According to the results presented by the authors, their method achieves the level of 32-bit float-point baseline with a mix of 8-bit floating-point types with 4- and 5-bit exponents. It is important to remark that the input to the first layer of the network must remain unquantized in order to avoid the network's accuracy decrease.

3.1.3 Hardware proposals

However, the software level definition of new variable formats and training procedures is crucial for NN training optimization, there is a necessity of an efficient hardware that supports the mentioned advancements. A good example of such a step is the Unified Neural Processing Unit (UNPU) designed by Lee et al. (2018) [15]. The accelerator enables flexible precision variable definition in the range from 1 to 16 bits. The support includes convolutional, fully connected, and recurrent layers covering a wide spectrum of modern NN architectures. The key features include an additional speed up due to reduction of off-chip memory accesses. According to the authors, this architecture allows for a 50% reduction of energy consumption and external memory accesses for specific NN definitions.

Another proposal that can be placed on the edge of hardware improvements is a quantization-based method introduced by Onishi et al. (2020) [16]. The proposal assumes the utilization of lookup tables (LUT) for optimization of memory and power usage. According to the authors, LUT allows to limit memory usage by up to 22% for a forward pass and 60% for a backward pass while training LeNet-5. The validation has been conducted with MNIST dataset and achieved accuracy with degradation of 1.41 percentage points in comparison to the baseline. It is worth mentioning that the overall number of multiplications has been reduced by 11.7%.

Kim et al. (2020) [17] introduced a precision-controlled memory system (PCM) which aims to reduce power requirements for NN trained with limited precision parameters. The authors state that, in comparison to regular GPU architectures, their method provides 34% lower energy consumption and 20% speedup. The solution has been evaluated on ResNet-20 with CIFAR100 dataset.

3.1.4 Results comparison

In order to present a comparison of methods reviewed in this literature study, an overview of key aspects of each proposal has been prepared in a form of Table 3.1. The summary includes training optimization details including variable types and applied techniques. Due to the variety of evaluation methods, implementation details and topologies used during investigated research, it is extremely difficult to fully compare performance of each method. Hence, the provided details include information about the baseline and post limitation accuracy to give the reader a better view on improvements reported by each author. The results are provided along with NN architectures and datasets used for accuracy validation.

Table 3.1. Detailed summary of the related study with comparison to the proposed precision limitation method for neural network training [36] [154]

Paper	Variable type	Technique	Dataset	Topology	Baseline accuracy	Accuracy after limitation
Gupta et al. (2015) [5]	12-bit fixed-point 14-bit fixed-point	Stochastic rounding	MNIST	Custom LeNet	99.23%	99.17% (14-bit fixed-point) 99.11% (12-bit fixed-point)
			CIFAR10	3-layer CNN	75.4%	74.6% (14-bit fixed-point) 71.2% (12-bit fixed-point)
Na and Mukhopadhyay (2016) [147]	16-bit fixed-point 32-bit fixed-point	Dynamic Precision Scaling (DPS) Flexible multiplier-accumulator (MAC)	MNIST	LeNet	Not given (only loss charts presented)	32-bit fixed-point accuracy achieved on 16-bit fixed point with DPS
			Flickr images	AlexNet (pre-trained)		64-bit fixed-point accuracy achieved on 32-bit fixed point with DPS
Ortiz et al. (2018) [6]	12-bit floating-point 12-bit fixed-point	Stochastic rounding Context representation	CIFAR10	3-layer CNN	75,6%	63.03% (12-bit fixed-point) 74.20% (12-bit floating-point) 78.02% (12-bit context-float) 76.32% (12-bit context-fixed)
Taras and Stuart (2018) [7]	14-bit fixed-point (weights) 16-bit fixed-point (activations)	Stochastic rounding Dynamic Precision Scaling (DPS)	MNIST	LeNet	98.80%	98.80%

Paper	Variable type	Technique	Dataset	Topology	Baseline accuracy	Accuracy after limitation
Park et al. (2018) [14]	Combination of 8-bit and 16-bit integers	Stochastic gradient descent with Kahan summation Lazy update	MNIST	LeNet-like CNN	99.10%	99.24%
			SVHN	4-layer CNN	97.06%	96.99%
			CIFAR10	3-layer CNN	81.56%	81.17%
				ResNet-20	90.16%	90.23%
			ImageNet	AlexNet	80.81%	80.62%
Fuketa et al. (2018) [146]	9-bit floating point format with hidden most significant bit and sign bit	Custom float representation Custom MAC unit	ILSVRC	AlexNet	48.27%	46.18%
				ResNet-50	68.84%	67.55%
Lee et al. (2018) [15]	Fully variable weight bit-precision from 1b to 16b	Original hardware accelerator for CNN-RNN networks	Not applicable	AlexNet VGG-16	Not applicable	Operation based power savings presented
Onishi et al. (2020) [16]	No strict parameters limitation, factorization based on LUT is used for limiting memory consumption and multi-adds operations.	Lookup-Table (LUT) based quantization Cluster swap	MNIST	LeNet	99.28%	97.87% Memory consumption reduced: -22.2% (forward pass) -60% (backward pass)
Lee (2020) [148]	Mix of: 16-bit floating-point 8-bit floating-point	Fine-Grained Mixed Precision	CIFAR10	ResNet-18	72.48% (16-bit floating-point)	72.45% (up to 94% of 8-bit floating-point)
			ImageNet		68.25% (16-bit floating-point)	99.11% (up to 90% of 8-bit floating-point)
Kim et al. (2020) [17] *Subset of results presented*	Mix of: 7-bit floating-point 9-bit floating-point	Precision-controlled memory system (PCM)	CIFAR10	ResNet-200	69% (16-bit floating-point)	~69% (9-bit floating-point)
Rios et al. (2021) [149]	Mix of: 32-bit floating-point 16-bit Brain floating-point	Mixed precision training	ImageNet	AlexNet	60.79% (32-bit floating-point)	60.32% (BF16FMA 94.60%)
				Inception	74.01% (32-bit floating-point)	72.80% (BF16FMA 95.55%)
				ResNet-50	75.69% (32-bit floating-point)	92.70% (BF16FMA 96.40%)

Paper	Variable type	Technique	Dataset	Topology	Baseline accuracy	Accuracy after limitation
Fu et al. (2021) [141] **Subset of results presented**	Dynamic range: From 2-bit floating-point to 32-bit floating-point	Cycling Precision Training (CPT) Last two stages trained with full precision	CIFAR10	ResNet-74	91.15% (SBM 6 bit)	92.4% (CPT 3-t o 6-bit, grad 6 bit)
				MobileNetV2	91.56% (SBM 6 bit)	91.81% (CPT 4- to 6-bit, grad 6 bit)
			CIFAR100	ResNet-74	70.31% (SBM 6 bit)	70.83% (CPT 3- to 6-bit, grad 6 bit)
				MobileNetV2	72.31% (SBM 6 bit)	73.18% (CPT 4– to 6-bit, grad 6 bit)
			ImageNet	ResNet-18	69.76% (32-bit floating-point)	70.67% (CPT: 8- to 32- bit)
Park et al. (2021) [8]	8-bit floating-point	Floating point with shared exponent bias multiple-way fuse multiply-add trees	ImageNet	ResNet-18	Not defined	69% (8-bit floating-point + SEB)
Junaid et al. (2022) [151]	Mix of: 32-bit floating-point 24-bit floating-point 16-bit floating-point	Mixed precision training	MNIST	Custom CNN	96% (32-bit floating-point)	93.32%
Yu et al. (2022) [142] *Subset of results presented*	Dynamic range: From 3-bit floating-point to 16-bit floating-point	Learnable Dynamic Precision (LDP)	CIFAR10	ResNet-18	91.86% (SBM 8 bit)	92.08% (LDP 3- to 8- bit, grad 8 bit)
			CIFAR100		67.24% (SBM 8 bit)	67.88% (LDP 3- to 8- bit, grad 8 bit)
			ImageNet		69.60% (SBM 8 bit)	69.62% (LDP 4- to 8- bit, grad 8 bit)

Paper	Variable type	Technique	Dataset	Topology	Baseline accuracy	Accuracy after limitation
Micikevicius et al. (2022) [152] *Subset of image classification results presented*	8-bit FP (4-bit exponent and 3-bit mantissa for weights and activations 5-bit exponent and 2-bit mantissa for gradients)	Scaling factor	ImageNet	VGG-16	71.27% (16-bit floating-point)	71.11%
				Inception v3	77.23% (16-bit floating-point)	77.06%
				ResNet-18	70.58% (16-bit floating-point)	70.12%
				ResNeXt50	77.68% (16-bit floating-point)	77.62%
				MobileNet v2	71.65% (16-bit floating-point)	71.04%
Noune et al. (2022) [153] *Subset of image classification results presented*	8-bit FP (4-bit exponent and 3-bit mantissa for weights and activations 5-bit exponent and 2-bit mantissa for gradients) 32-bit FP input to the first layer of the network	Bias offset (per parameter type)	CIFAR100	ResNet-32	70.26%	70.42% (32-bit floating-point used for first layer activations and activations gradients)
			ImageNet	ResNet-18	70.35%	70.29% (32-bit floating-point used for first layer activations and activations gradients)
				ResNet-50	76.61%	76.57% (32-bit floating-point used for first layer activations and activations gradients)
				EfficientNet-B4	82.42%	82.34% (16-bit floating-point used for first layer activations and activations gradients)
Pietrołaj and Blok (2022) [36]	8-bit floating-point 12-bit floating-point 14-bit floating-point	Asymmetric exponent No additional rounding	MNIST	LeNet	96.04%	75.89% (8-bit floating-point) 95.01% (12-bit floating-point) 97.13% (14-bit floating-point)

Paper	Variable type	Technique	Dataset	Topology	Baseline accuracy	Accuracy after limitation
Pietrołaj and Blok (2024) [154]	8-bit floating-point (4-bit exponent and 3-bit mantissa)	Asymmetric exponent Exponent offset Stochastic rounding	MNIST	LeNet	96.18% (10 epochs) 98.35% (30 epochs)	95.98% (10 epochs) 98.38% (30 epochs)
			CIFAR10	AlexNet	74.39% (10 epochs) 79.53% (30 epochs)	74.5% (10 epochs) 80.06% (30 epochs)
				ResNet-18	77.08% (10 epochs) 83.41% (30 epochs) 94.99% (200 epochs)	76.01% (10 epochs) 82.22% (30 epochs) 94.58% (200 epochs)
			CIFAR100	AlexNet	38.93% (10 epochs) 51.82% (30 epochs)	38.69% (10 epochs) 51.91% (30 epochs)
				ResNet-18	39.54% (10 epochs) 51.69% (10 epochs) 75.08% (200 epochs)	40.21% (10 epochs) 55.16% (30 epochs) 74.25% (200 epochs)

An additional conclusion that can be drawn from the above summary is that mixed-precision proposals combining multiple bit count varying variables are especially popular in recent years [148] [149] [141] [142] [151] [17]. The software advancements are also backed up by innovation from the hardware side. Although such solutions provide satisfactory results, the possible disadvantages may be caused by the overhead required for mixing multiple variable types [15] [155]. The proposal presented in this dissertation, although the bit count of NN parameters is constant, allows for mixed cross tensor, layer or epoch precision approach. The mechanism used here is in the common domain with the FP8-SEB technique [8] or offset bias [153] and leverages floating-point format manipulation, especially offset of exponent values.

3.2 Limitation framework

In order to conduct limitation experiments, a custom software framework had to be created to support all required use cases. The implementation leveraged Python [156] programming language and PyTorch [84] machine learning framework as a foundation for the development of further features. The method used generally available CPU and GPU hardware with 32-bit floating-point parameters as base variables. The limitation mechanism has been implemented in a software layer in a similar manner to Ortiz et al. (2018) [6] and Micikevicius et al. (2022) [152]. At each step of NN execution, layer by layer, all parameters such as weights, biases, activations, and gradients were limited to the target bit count. Then, such a limited numerical value has been temporarily stored in a regular 32-bit floating-point variable supported

by the hardware. Fig. 3.1 depicts an overview of the training environment provided by the designed framework.

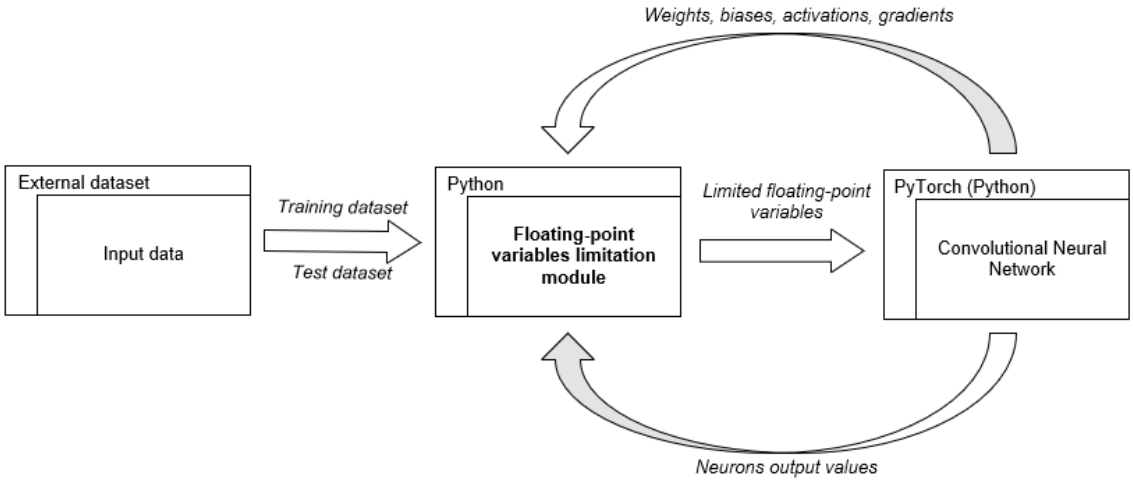


Fig. 3.1. An overview of the neural network training environment for floating-point limitation

The floating-point limitation method used in the following experiments assumed a custom technique of fitting a variable into targeted, constrained bit count. During each limitation step, the input 32-bit variable is split into sign, exponent and mantissa parts. Then the algorithm shortens the mantissa based on the selected bit-width target. This is done by the selection of the most significant mantissa's bits. Due to the specification of the IEEE-754 floating-point format, the exponent must first undergo a procedure of bias removal. Then based on the bit count target, maximum and minimum boundaries for exponent values are calculated. If an exponent cannot be contained in the selected range, its value is assigned respectively to the nearest maximum or minimum representation. It is important to note that once the exponent's limitation is completed, the new bias value, adjusted to targeted bit count, is applied to the final exponent. Then, if required, the mantissa's value is adjusted to reduce the potential rounding error resulting from operations conducted on the exponent. Once the described steps are finalized, the result of combined exponent and mantissa is translated to a 32-bit floating-point format. Fig. 3.2 presents a simplified pseudo code of the outlined limitation algorithm. To maintain clarity, the code snippet omits some of the details used in the original algorithm's implementation such as input parameters, definition of constant and temporary variables, bit shift operations, tensor operations and exponent bias handling.

```
procedure limit_variable(x)
      sign = extract_sign(x)
      mantissa = extract_mantissa(x)
      exponent = extract_exponent(x)

      limited_mantissa =
            cut_least_significant_bits(target_mantissa_bitcount)
      limited_exponent_max =
            get_max_limited_exponent_value(target_exponent_bitcount)
      limited_exponent_min =
            get_min_limited_exponent_value(target_exponent_bitcount)

      if exponent > limited_exponent_max:
            exponent = limited_exponent_max
      else if exponent < limited_exponent_min:
            exponent = limited_exponent_min

      limited_mantissa =
                  adjust_mantissa_value(exponent, limited_mantissa)

      limited_variable = sign | exponent | limited_mantissa

      return limited_variable
```

Fig. 3.2. A simplified pseudocode depicting the algorithm used for the limitation of parameters used in the neural network training

Fig. 3.3 gives an example of applying the algorithm to a 32-bit floating-point variable. The targeted format consists of 4-bit exponent and 3-bit mantissa. Please note that in case of the framework implementation, all limitation operations are done on the tensor level instead of a single variable to improve operations efficiency.

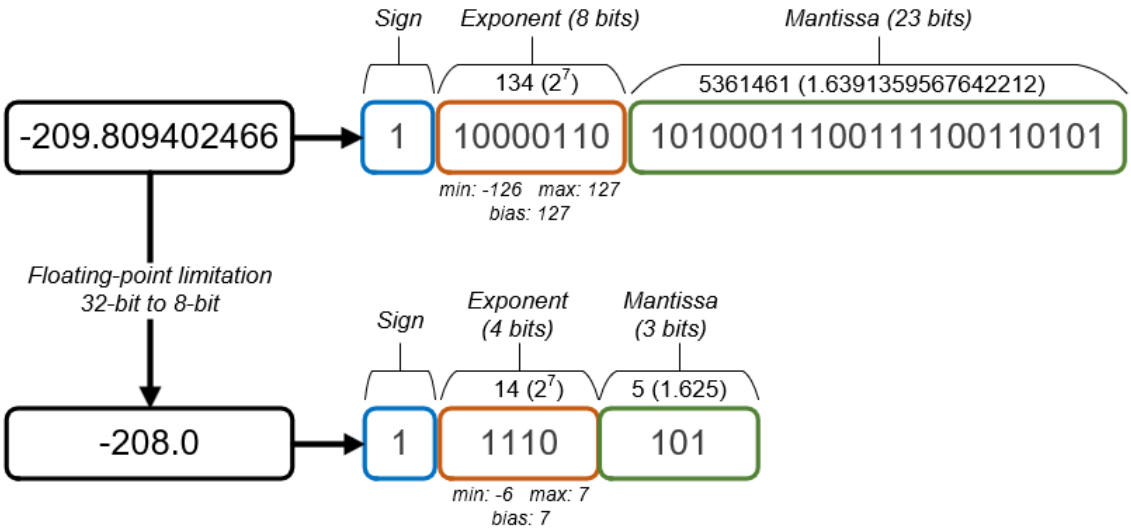


Fig. 3.3. Representation of a 32-bit floating-point value with an 8-bit floating-point format

Due to the much larger range of the original exponent representation, it is often required to truncate its value due to targeted bit count limitation. Fig. 3.4 gives an example of such a case. It is important to remark that the sign of a limited floating-point number does not impact the method in any form.

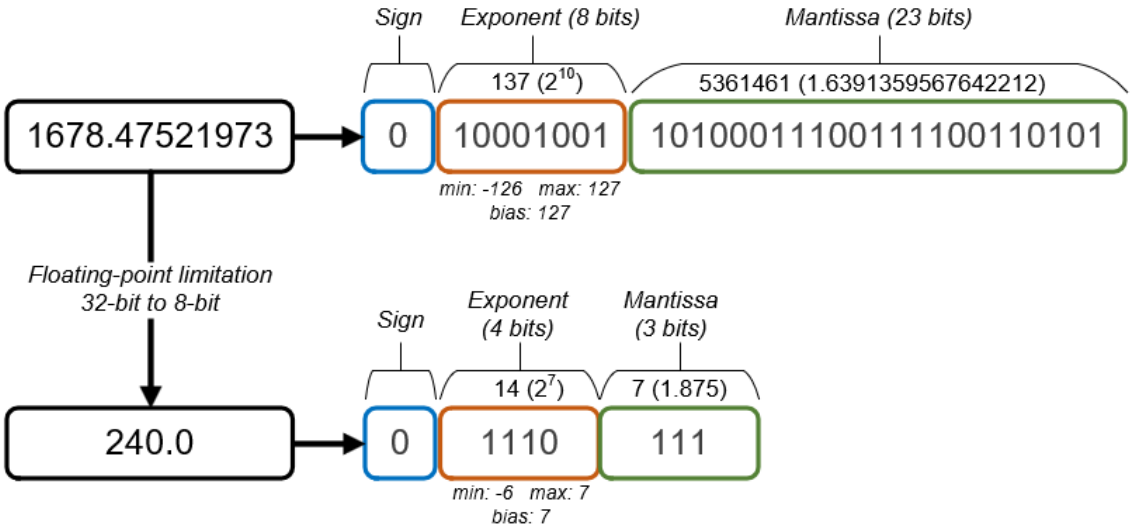


Fig. 3.4. Representation of a 32-bit floating-point value with an 8-bit floating-point format with exponent truncation.

3.3 Limitation results

Limitation experiments have been conducted on three CNN architectures LeNet, AlexNet and ResNet-18. The selection has been dictated by the popularity of these network designs in the research community and relative easiness of experiments reproducibility by other peers. In addition, the previously presented related study (Table 3.1) shows that similar NNs have been used in several papers related to training with precision limitation. Hence, there is more data available for results comparison.

The limitation conditions were applied in the same form to all tested NN topologies. Each model has been trained multiple times in the range from 3 to 32 bits over 10 epochs. It includes all possible exponent and mantissa configurations available in IEEE-754 32-bit floating-point representation boundaries. Training environment's configuration remained unchanged for each of the topologies across varying bit count trainings. Comparison of results was done based on the test set part of each of the used datasets. Table 3.2 presents targeted baseline IEEE-754 32-bit accuracies achieved for the selected neural network topologies in the developed experimentation environment after 10 training epochs.

Table 3.2 Baseline IEEE-754 32-bit accuracies per neural network after 10 training epochs

Neural network topology	Dataset	Test accuracy
LeNet	MNIST	96.18%
AlexNet	CIFAR10	74.39%
	CIFAR100	38.93%
ResNet-18	CIFAR10	77.08%
	CIFAR100	39.54%

3.3.1 LeNet

Although the current trend in the domain of NN architecture focuses on growing topologies with millions of parameters, there are multiple low-level hardware use cases where smaller topologies are still required, especially in case of embedded or edge devices [157]. LeNet-5 is commonly treated as an introductory CNN used as a benchmark for various optimization experiments [5] [147] [7] [14] [16]. The conducted training with limited precision combined LeNet with MNIST dataset [143] as an input. MNIST is a dataset of hand-written digits often used for CNN verification tasks. It consists of 60000 training and 10000 test examples. Implementation details of LeNet architecture utilized in the experiment are summed up in Table 3.3. Additionally, Table 3.4 presents hyper-parameters set for the training phase.

Table 3.3. Summary of the LeNet-5 architecture used in the experiment [55] [158]

Layer		Feature Map	Size	Kernel size	Stride	Padding	Activation
Input	**Image**	1	28x28	-	-	-	-
1	**1st Convolution**	6	28x28	5x5	1	2	tanh
2	**Average Pooling**	-	14x14	2x2	-	-	-
3	**2nd Convolution**	16	10x10	5x5	1	-	tanh
4	**Average Pooling**	-	5x5	2x2	-	-	-
5	**3rd Convolution**	120	1x1	5x5	1	-	tanh
6	**1st Fully Connected**	-	84	-	-	-	tanh
Output	**2nd Fully Connected**	-	10	-	-	-	softmax

Table 3.4. Hyperparameters used during LeNet-5 training

Hyperparameter	Value
Optimizer	Stochastic Gradient Descent
Learning rate	0.01
Batch size	64
Loss function	Cross entropy

Fig. 3.5 presents LeNet-5 training results with the use of the described limitation algorithm. The evaluation has been conducted across exponent bit counts ranging from 1 to 8 and mantissa bit count from 1 to 23. It is worth highlighting that with 32-bit floating-point parameters this network achieves accuracy of 96.18% over 10 epochs in the presented training environment.

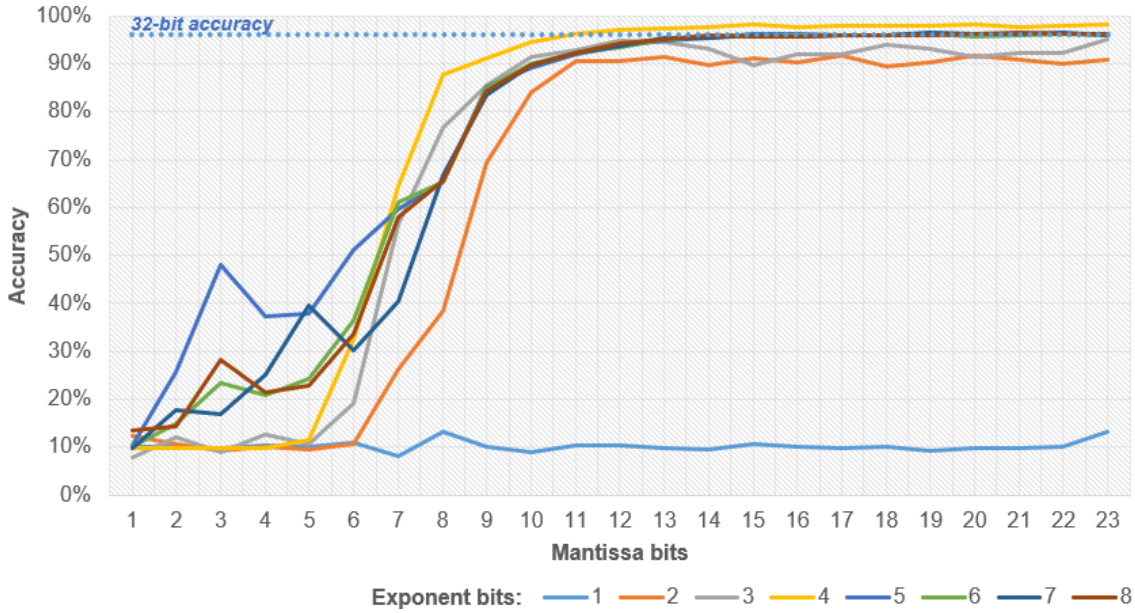


Fig. 3.5. LeNet-5 accuracy across various limited bit count configurations

The expected effect can be observed, the reduced number of bits dedicated to exponent and mantissa has a negative impact on the network training and decreases the overall classification accuracy. As already discussed, a lower number of bits narrows the dynamic range that can be represented by a particular variable format [77] [95]. It is also vivid that 1-bit exponent is insufficient for training this particular NN, giving around 10% accuracy over all possible mantissa sizes. The results on 2-bit exponent are much better, however, as presented in previous research [154], the accuracy achieved with such low exponent bit counts is often unstable and can vary between following training executions. Exponents with 3 or higher bit counts provide much better output along with the increasing mantissa bit width. The observed results vary in the range from 90% to 98%. Surprisingly, the best result of 98.11% has been achieved for 4-bit exponent, exceeding 32-bit floating-point baseline (Table 3.2). Such observations for low bit count

variables have been also confirmed by other researchers [141]. The common explanation of this phenomenon is that limiting the number of available bits in a variable, same as learning rate manipulation, can be treated as a form of an additional NN regularization which helps with knowledge generalization based on previously learned data samples [142].

The important conclusion that can be drawn from this experiment is inability to train LeNet-5 based on 8-bit floating-point parameters to match 32-bit accuracy. The lowest combined bit count of sign, exponent and mantissa which allowed for achieving matching accuracy of 96.32% was 16 bits. Such a format combines 1 bit to sign, 4 bits to exponent and 11 bits to mantissa. Training neural networks on 16-bits variables is a common technique for reducing complexity of a NN topology both in computation and memory domains. Such an option is available in most of the popular machine learning frameworks as Pytorch or Tensorflow [26]. Although in this case the author showed that 4-bit exponent with 11-bit mantissa gives the best results, the more common, generally available type used by other frameworks leverages the IEEE-754 16-bit floating-point with 5-bit exponent and 10-bit mantissa. In case of this experiment, such a type achieved significantly lower accuracy of 89.09% over 10 epochs.

3.3.2 AlexNet

The analogous experiment has been conducted on another popular CNN topology, AlexNet. It is commonly treated as a pivot point in terms of development of deep learning architectures, especially in terms of image recognition. The training leveraged two benchmark image datasets focusing on small image classification, CIFAR10 and CIFAR100 [144]. Both datasets contain 50000 training and 10000 test 32x32 images divided into 10 and 100 classes accordingly. In modern machine learning tasks AlexNet is often treated as a comparison point or benchmark for various researchers [14] [146] [15] [149] [147]. Hence, it was selected for investigation in this dissertation. Table 3.5 gives an overview of convolutional network architecture used for the training. Table 3.6 summarizes its main hyper-parameters.

.

Table 3.5. Summary of the AlexNet architecture used in the experiment [48]

Layer		Feature map	Size	Kernel size	Stride	Padding	Activation
Input	**Image**	3	32x32	-	-	-	-
1	**1st Convolution**	64	34x34	3x3	1	2	ReLU
2	**1st Max Pool**	64	17x17	2x2	-	-	-
3	**2nd Convolution**	192	19x19	3x3	-	2	ReLU
4	**2nd Max Pool**	192	9x9	2x2	-	-	-
5	**3rd Convolution**	384	9x9	3x3	-	1	ReLU
6	**4th Convolution**	256	9x9	3x3	-	1	ReLU
7	**5th Convolution**	256	9x9	3x3	-	1	ReLU
8	**3rd Max Pool**	256	4x4	3x3	2	-	-
9	**1st Dropout (*rate = 0.6*)**	-	4096	-	-	-	-
10	**1st Fully Connected**	-	2048	-	-	-	ReLU
11	**2nd Dropout (*rate = 0.6*)**	-	2048	-	-	-	-
12	**2nd Fully Connected**	-	2046	-	-	-	ReLU
13	**3rd Fully Connected**	-	10/100	-	-	-	softmax

Table 3.6. Hyperparameters used during AlexNet training

Hyperparameter	Value
Optimizer	Adam
Learning rate	0.0001
Batch size	128
Loss function	Cross entropy

Similarly, to LeNet-5, AlexNet has been trained on various bit count configurations over 10 epochs. Baseline 32-bit floating-point accuracy of this network is equal to 74.39% on CIFAR10 and 38.39% on CIFAR100 (Table 3.2). Fig. 3.6 and Fig. 3.7 depict the results of the experiments for both datasets.

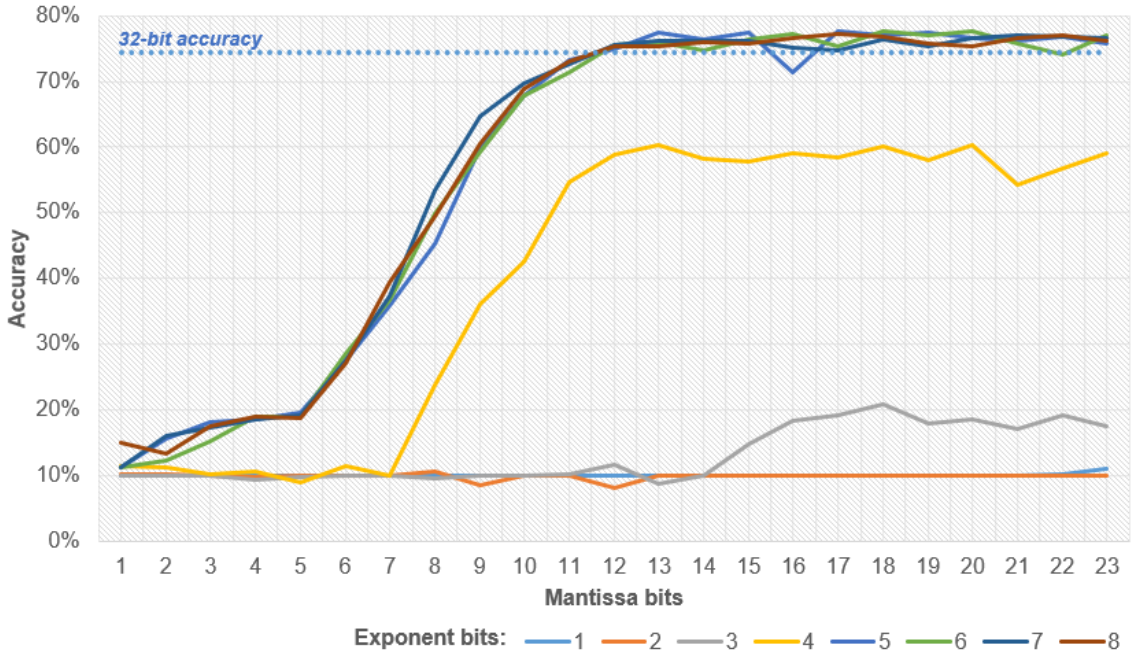


Fig. 3.6. AlexNet accuracy across various limited bit count configurations on CIFAR10 dataset

As depicted by Fig. 3.6, 4- and lower-bit exponents are insufficient for AlexNet training on CIFAR10 dataset without significant decrease in the model's accuracy. Even 16-bit floating-point composed of 5-bit exponent and 10-bit mantissa achieves only 67.83% of accuracy which is over 6 percentage points worse than the baseline (Table 3.2). First comparable results can be observed for combined bit count of 18 bits with 5-bit exponent and 12-bit mantissa giving 75.02% which slightly surpasses the previously calculated 32-bit baseline result (Table 3.2).

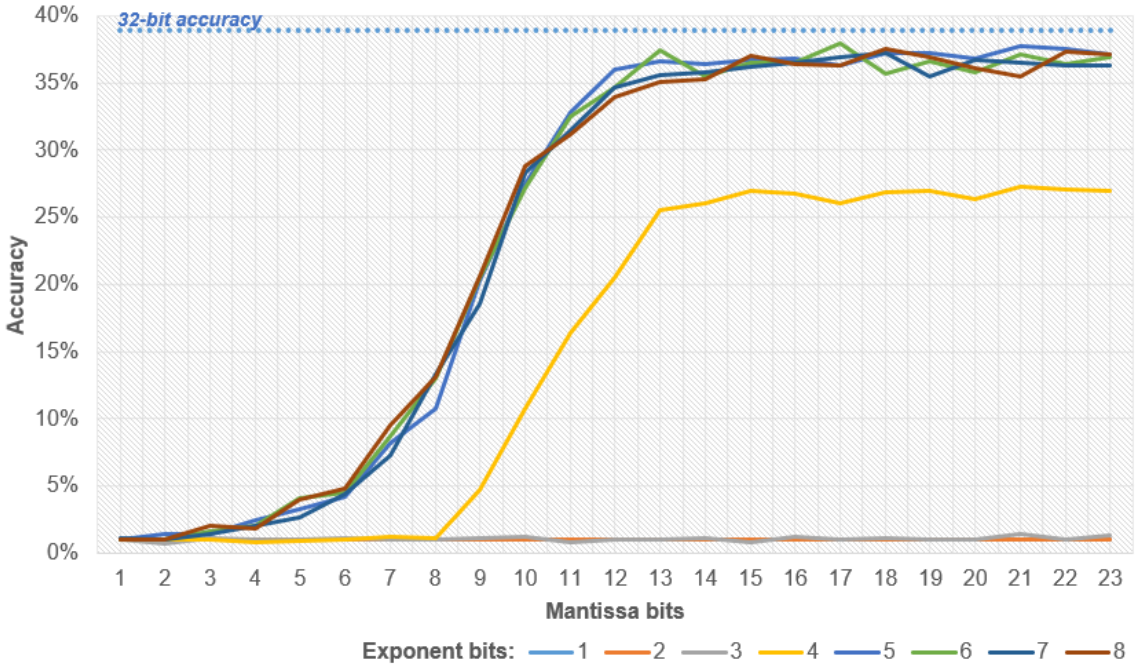


Fig. 3.7. AlexNet accuracy across various limited bit count configurations on CIFAR100 dataset

A similar case can be observed with AlexNet results on CIFAR100 dataset (Fig. 3.7). Again, 4- and lower-bit exponents do not provide sufficient range for the training process of this topology. The minimum bit width of the required exponent is 5 bits which gives a comparable accuracy of 37.23% for 18 bits of mantissa.

3.3.3 ResNet

In order to check limitation impact on more modern NN architecture, the last topology selected for the experiment was ResNet. It was a winner of ImageNet Competition in 2015 and has been established as the most cited neural network in the 21st century [159]. There are multiple popular versions of ResNet implementation available based on the number of the network's layers, regularly used by researchers as a benchmark architecture [14] [146] [148] [17] [149] [141] [8] [127]. In the case of this dissertation, the version with 18 layers has been used, also known as ResNet-18. Table 3.7 and Table 3.8 summarize the architecture of ResNet-18 used in this experiment along with ResNet Building Block details. Table 3.9 provides main hyper-parameters used during the training process.

Table 3.7. Summary of the ResNet-18 architecture used in the experiment

Layer		Feature map	Size	Kernel size	Stride	Padding	Activation
Input	**Image**	3	32x32	-	-	-	-
1	**1st Convolutional Layer**	64	32x32	3x3	-	1	-
2	**Bach Normalization**	64	32x32	-	-	-	ReLU
3	**[2nd Convolution *(Block)*] x 2**	64	32x32	3x3	1	1	ReLU
4	**[3rd Convolution *(Block)*] x 2**	128	16x16	3x3	2	1	ReLU
5	**[4th Convolution *(Block)*] x 2**	256	8x8	3x3	2	1	ReLU
6	**[5th Convolution *(Block)*] x 2**	512	4x4	3x3	2	1	ReLU
7	**Average Pooling**	512	1x1	-	-	-	-
8	**1st Fully Connected**	-	10/100	-	-	-	softmax

Table 3.8. Summary of the ResNet-18 Convolutional Basic Block architecture used in the experiment. The values examples based on the first block of the network

Convolution Block Layer		**Feature map**	**Size**	**Kernel size**	**Stride**	**Padding**	**Activation**
Input	**1st Convolution output**	64	32x32	-	-	-	-
1	**1st Convolution layer**	64	32x32	3x3	1	1	-
2	**1st Batch Normalization**	64	32x32	-	-	-	ReLU
3	**2nd Convolution layer**	64	32x32	3x3	1	1	-
4	**2nd Batch Normalization**	64	32x32	-	-	-	ReLU
5 ***(Optional)***	**3rd Convolutional layer** ***(shortcut)***	64	32x32	1x1	1	1	-
6 ***(Optional)***	**3rd Batch Normalization** ***(shortcut)***	64	32x32	1x1	1	1	
Output	**4th layer output + 6th Layer output**	64	32x32	-	-	-	ReLU

Table 3.9. Hyperparameters used during ResNet-18 training

Hyperparameter	Value
Optimizer	Stochastic Gradient Descent
Learning rate	0.1
Momentum	0.9
Weight decay	5e-4
Learning rate scheduler milestones	60, 120, 160
Learning rate scheduler gamma	0.2
Batch size	128
Loss function	Cross entropy

As in the previous trainings, ResNet-18 has been iterated over 10 epochs. The datasets are the same as in the case of AlexNet and include CIFAR10 and CIFAR100. The baseline 10 epoch, 32-bit floating-point accuracies achieved for this network are equal to 77.08% for CIFAR10 and 39.54% for CIFAR100 (Table 3.2). Fig. 3.8 and Fig. 3.10 present the results of the experiments.

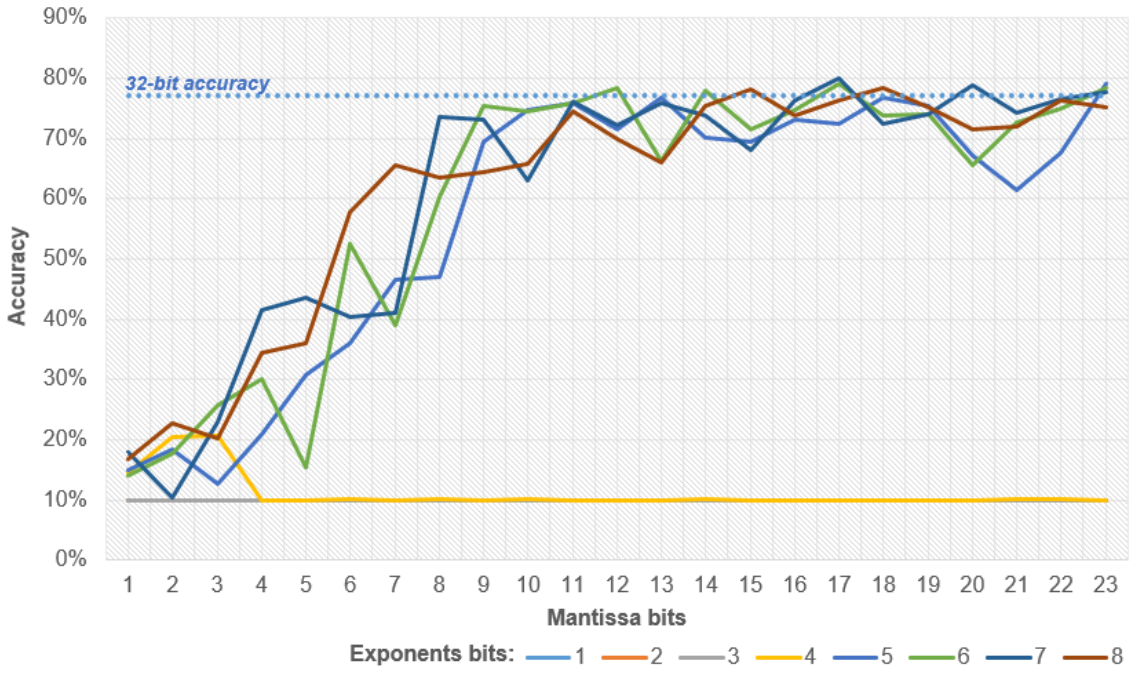


Fig. 3.8. ResNet-18 accuracy across various limited bit count configurations on CIFAR10 dataset

Based on the results presented in Fig. 3.8, it can be noticed that 4-bit or lower exponent bit counts are insufficient for training ResNet-18 on CIFAR10 dataset. For most of the mantissa bit count values verified in this scenario, the accuracy line remains almost flat around 10%. The

improvement can be observed for higher exponent bit counts, however, the results appear to be highly volatile even with increasing mantissa. This behavior may be caused by the fact that only the tenth epoch's result is presented on the chart in each case. In contrast to LeNet and AlexNet, ResNet-18 may usually require more epochs to achieve its results convergence [160]. In order to limit this side effect for presentation purposes, Fig. 3.9 shows results for the best of 10 epochs in the exact same training process for each bit count variant.

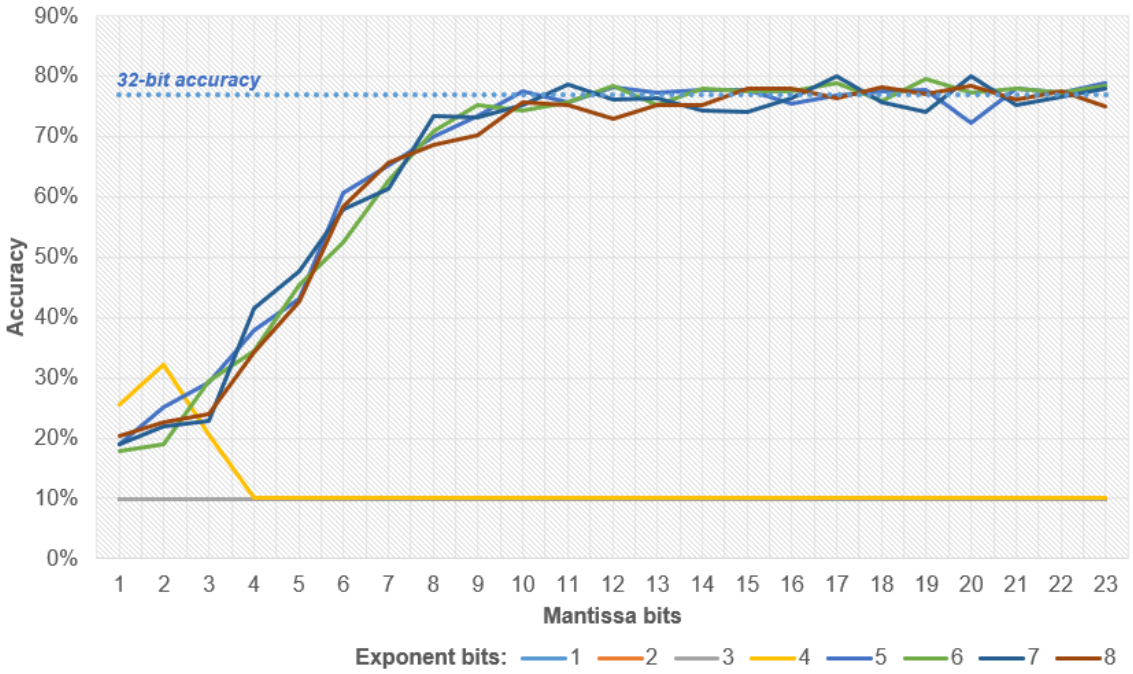


Fig. 3.9. ResNet-18 accuracy across various limited bit count configurations on CIFAR10 dataset – the best validation epoch results selected

It can be observed that accuracy achieved by the tested network grows with exponent and mantissa bit counts. First results that can be compared to the 32-bit baseline are achieved for 16-bit floating point consisting of 5-bit exponent and 10-bit mantissa giving 77.45% accuracy. This result is consistent with observation of other researchers and ML framework developers which frequently use this data type for more efficient NN training [161]. Results with higher bit counts tend to maintain similar network's accuracy with the deviation of a few percentage points. These differences can be further reduced with a higher number of training epochs.

The analogous case appears in the ResNet-18 training on CIFAR100 dataset which is presented in Fig. 3.10. Once again exponent bit counts below 5-bits do not provide enough range for achieving accuracy close to the 32-bit baseline (Table 3.2). The volatility of the last epochs results makes it difficult to observe the accuracy growth. Fig. 3.11 corrects this inconvenience and focuses on best epochs results only.

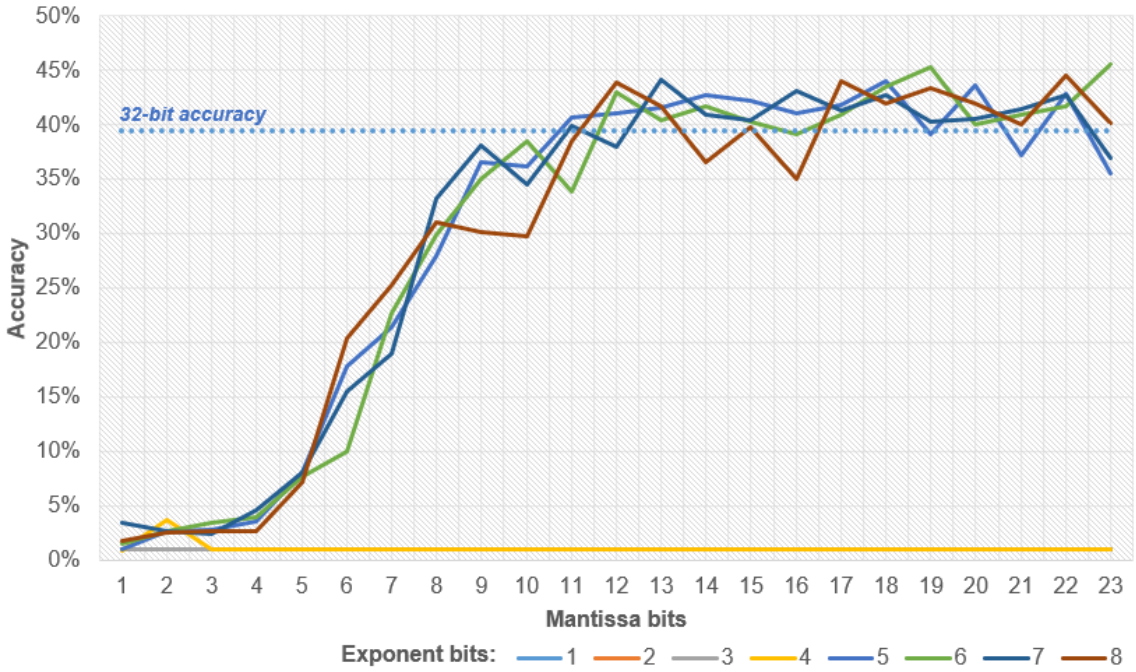


Fig. 3.10. ResNet-18 accuracy across various limited bit count configurations on CIFAR100 dataset

CIFAR100 states a more difficult training scenario for ResNet-18. The previously sufficient 16-bit floating-point does not maintain the baseline accuracy (Table 3.2) for this dataset achieving 36.15% of accuracy. Interestingly, higher bit counts tend to exceed the 32-bit baseline with over 5 percentage points. An example of such a scenario is a 30-bit floating point with 6-bit exponent and 23-bit mantissa. This behavior can be explained by a form of regularization that can be introduced due to limited precision variables [142].

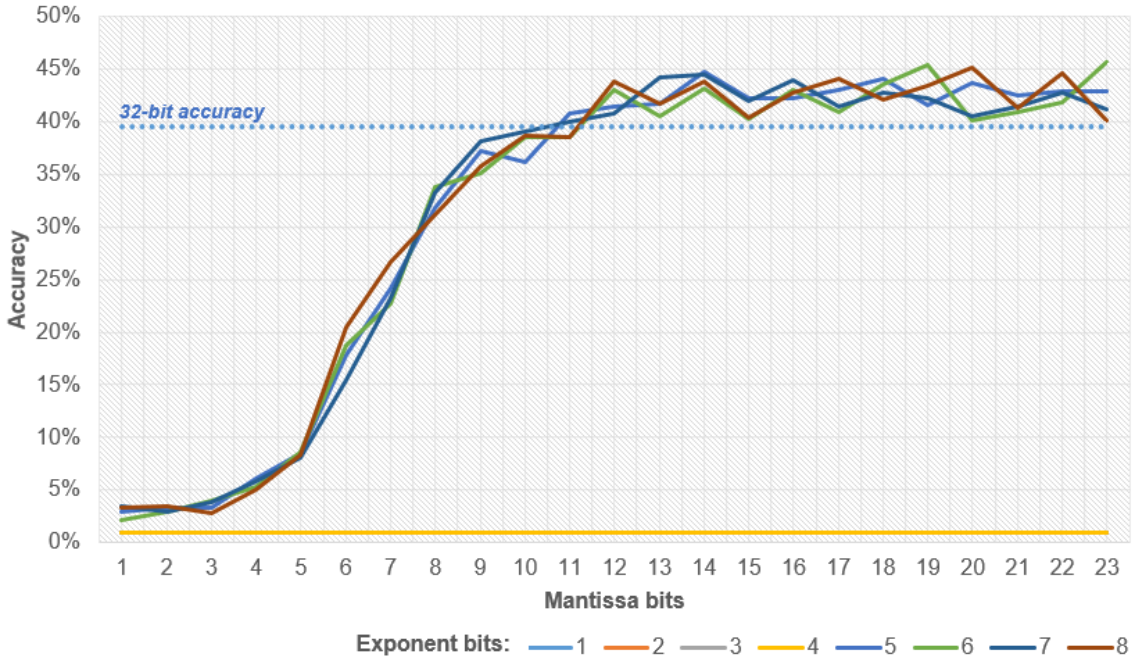


Fig. 3.11. ResNet-18 accuracy across various limited bit count configurations on CIFAR100 dataset – the best validation epoch results selected

It is important to remark that all tested scenarios showed that 8-bit floating point variables are not sufficient for training the presented NNs. Hence, a simple floating-point limitation cannot be used for such a purpose. Accurate and efficient NN training with low precision variables requires a more sophisticated approach presented in the later chapters of this dissertation.

3.4 Exponent utilization

Neural network's ability to conduct a proper prediction is a major factor that dictates its usability. However, besides focusing solely on accuracy, an interesting observation can be made regarding bit count utilization. The described experiment, and various previous research [141] [142] [8], showed that the full variable precision is not mandatory for NN convergence during the training process. Hence, in a similar fashion to quantized inference [103] [104], it is important to examine possible format and bit width optimizations when it comes to the NN learning process.

An important factor of this research was the investigation of a bit count usage of 32-bit floating-point variables during training of the selected NN architectures. The conducted analysis showed that the majority of tested IEEE-754-based network parameters use negative exponent values to store their numeric representation. This observation poses a question if exponent bits in a regular 32-bit floating-point representation are optimally utilized during trainings of the investigated NN topologies. As mentioned previously, the IEEE-754 exponent bias, with a value of 127, splits its values range into negative and positive halves. Considering that most of the positive portion is not utilized during a specific neural network architecture training, translates to omitting a subset of the bits assigned to a 32-bit floating-point representation and therefore partially wasting resources.

The analysis focused on the IEEE-745 32-bit floating-point based trainings of previously presented networks, LeNet-5, AlexNet and ResNet-18. Each architecture has been trained over 10 epochs with continuous logging of their most important parameters. The scope included a focus on the exponent values utilization for weights, gradients, biases and activations of each network layer.

Fig. 3.12 gives a detailed summary of exponent values utilization for the main groups of parameters used during neural network training including weights, gradients, activations and biases. It can be observed that only a portion of the 8-bits exponent is utilized during the training. Including special values, regular exponent provides 255 numeric representations in the range from -126 to 127. The combined exponents of weights, gradients and activations in the presented training can be situated in the narrow range from -17 to 1. This means that only around 8% of all possible exponent representations have been used during the LeNet-5 training phase over 10 epochs with regular IEEE-754 32-bit floating point variables.

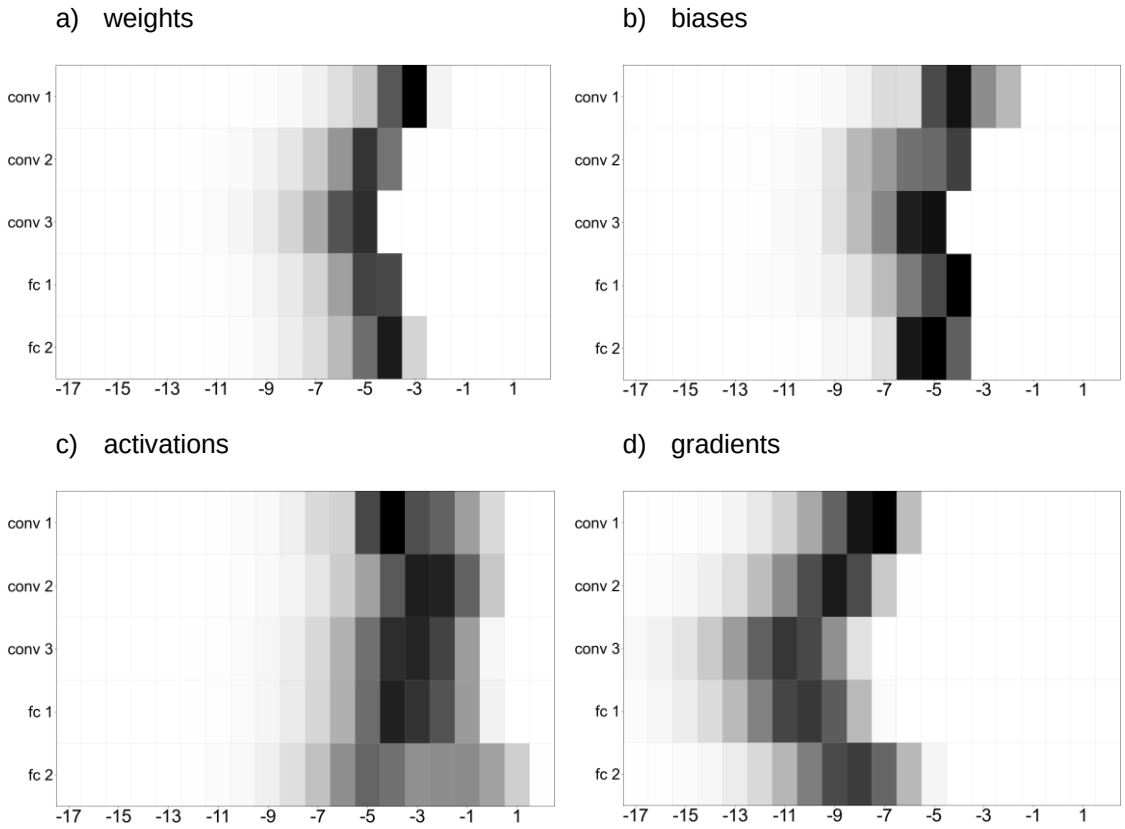


Fig. 3.12. LeNet-5 exponent utilization (normalized over layer) during training on MNIST dataset. The darker the color the higher the utilization. Layers: conv – convolution, fc – fully connected

It should be noticed that the utilized range of exponent values varies by a parameter type. The smallest one is represented by weights and biases which can be fitted between -9 and -2. The situation is different for activations, which with the range of -9 to 1 tend to also use positive values of the exponent. In the case of gradients, another behavior can be remarked in the form of utilizing much lower values of the exponent ranging from -17 to -5. This observation is expected as researchers have already stated that both gradients and activations may be problematic when it comes to NN training with limited precision, hence, various methods tend to leverage a higher bit count variables or their mix especially for those two types of parameters [7] [117] in comparison to weights and biases where much lower bit count is often sufficient [157].

Similar analysis has been carried out for AlexNet. Fig. 3.13 presents results of exponent values utilization for this network architecture trained with full precision variables on CIFAR10 dataset over 10 epochs.

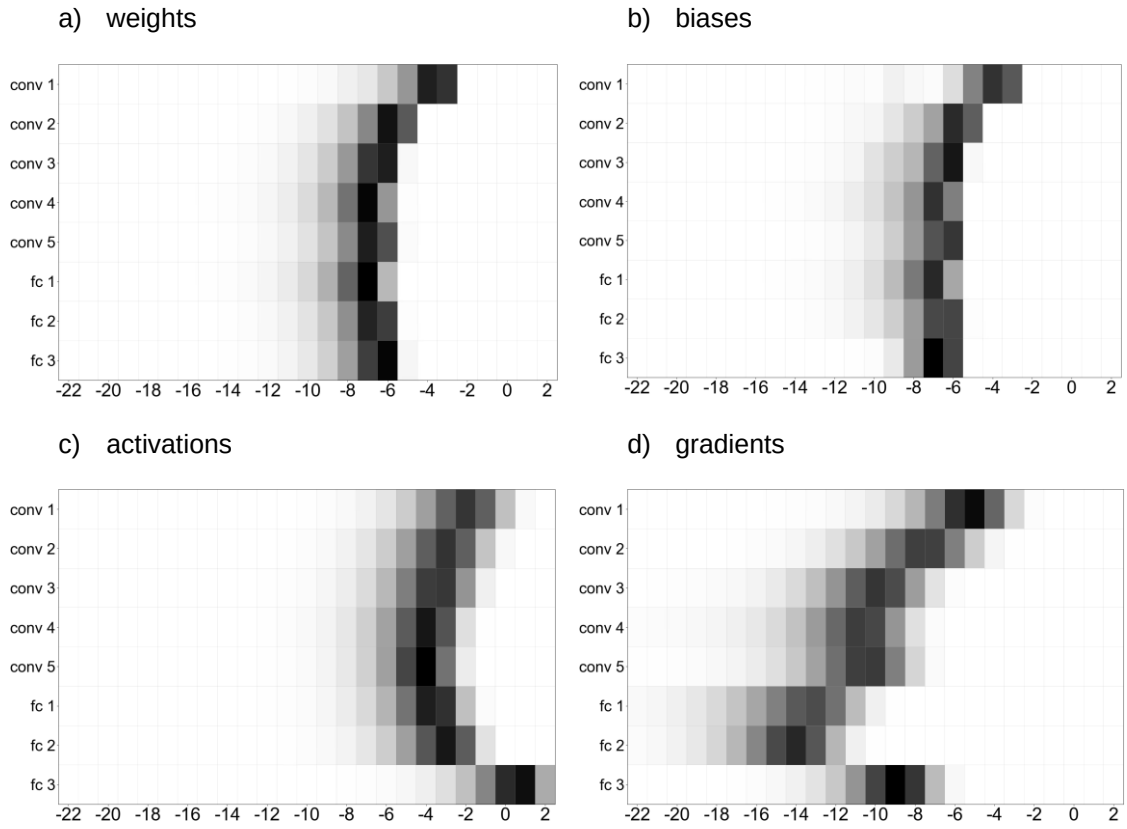


Fig. 3.13. AlexNet exponent utilization (normalized over layer) during training on CIFAR10 dataset. The darker the color the higher the utilization. Layers: conv – convolution, fc – fully connected

Although, in the case of AlexNet the utilization of exponent values is bigger than with LeNet-5, it still uses around 10% of the available 8-bit range. The numeric values used during the training can be contained between -22 and 3. The most common exponent values in this range could be narrowed to a scope between -17 and 2 which makes it close to the observations of LeNet-5. It is important to remark that a similar characteristic can be observed in the case of AlexNet regarding weights and biases. The effective exponent range needed to represent these parameters can be limited to values between -10 and -3. The move in the direction of positive exponent values can be observed for activations in the range from -8 to 3. The wide area is utilized by gradients which require exponent values from -22 to -4. Same as with LeNet both activations and gradients tend to require higher precision for a NN training step.

Another test related to AlexNet has been conducted with the CIFAR100 dataset. The gathered results are analogous to those achieved with CFIAR10. Fig. 3.14 presents exponent utilization observed during this training.

The same procedure has been repeated for the ResNet-18 NN. Fig. 3.15 presents results for a full precision training on CIFAR10 over 10 epochs. In order to improve readability of the ResNet-18 related diagrams some of the layers and basic blocks details have been incorporated into groups combining one or multiple basic blocks of the architecture. This is especially visible in case of charts displaying the network's activations.

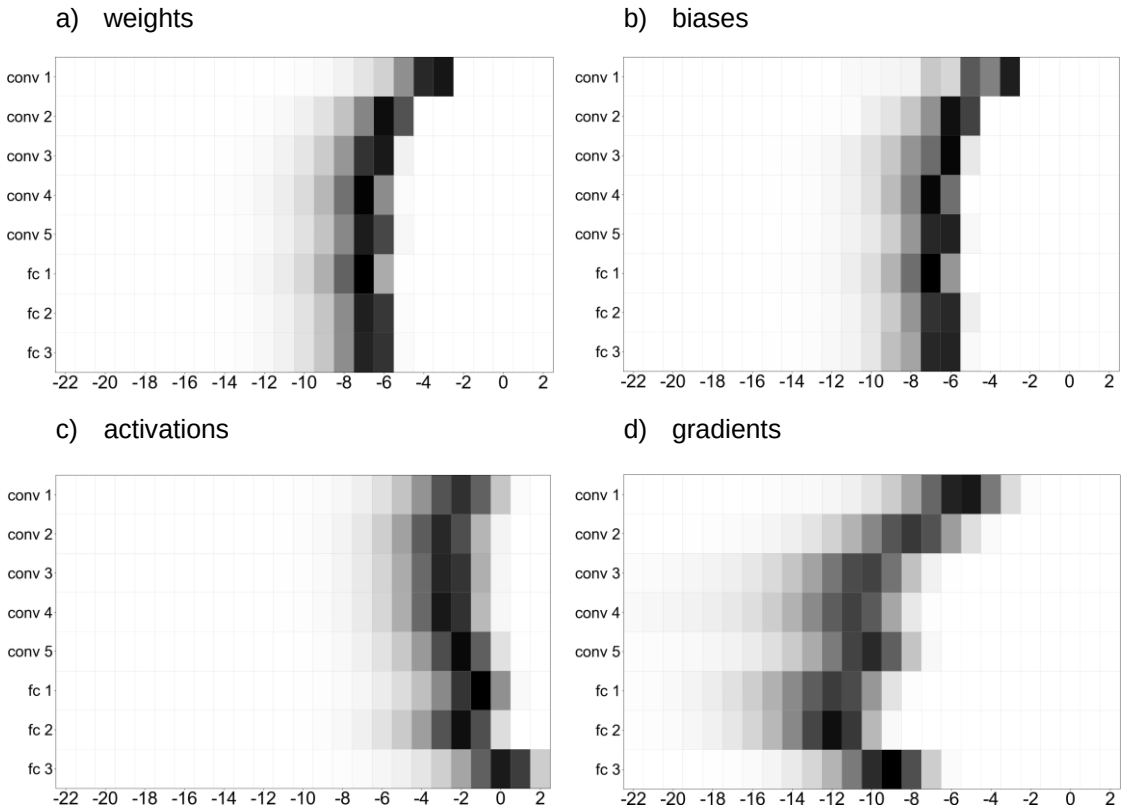


Fig. 3.14. AlexNet exponent utilization (normalized over layer) during training on CIFAR100 dataset. The darker the color the higher the utilization. Layers: conv – convolution, fc – fully connected

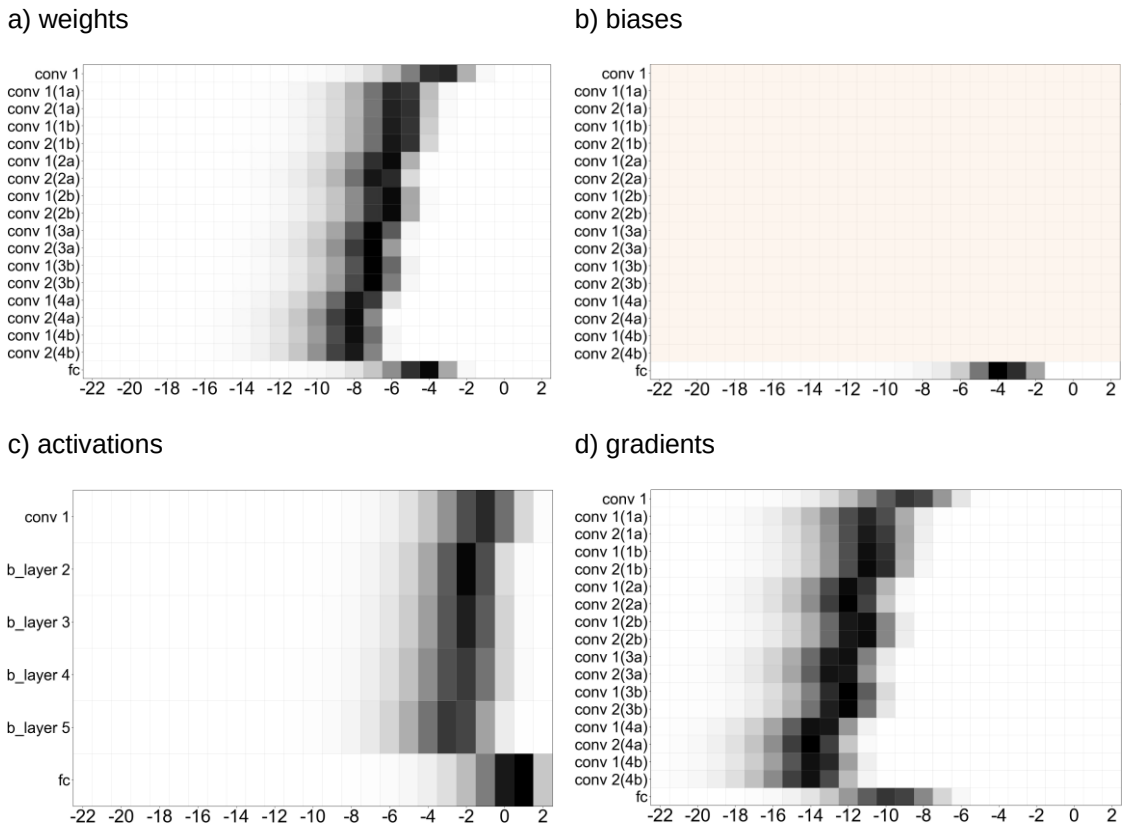


Fig. 3.15. ResNet-18 exponent utilization (normalized over layer) during training on CIFAR10 dataset. The darker the color the higher the utilization. Activations were grouped for clarity of the diagram. Layers: conv (x) – convolution (basic block), fc – fully connected, b_layer – combined layers into basic blocks

Despite differences in NN architecture, it can be noticed that the utilization of exponent values in ResNet-18 is similar to that in AlexNet and is equal to around 10%. The required range can be estimated to values between -20 and 3. As expected, gradients and activations seem to be the most precision-wise demanding parameters. In contrast to the previously tested networks, this architecture implementation does not include biases for convolutional layers which can be observed in Fig. 3.15 b) and Fig. 3.16 b).

The application of CIFAR100 does not impact ResNet-18 exponent utilization in a major way. Nevertheless, the results of such analysis are presented in Fig. 3.16.

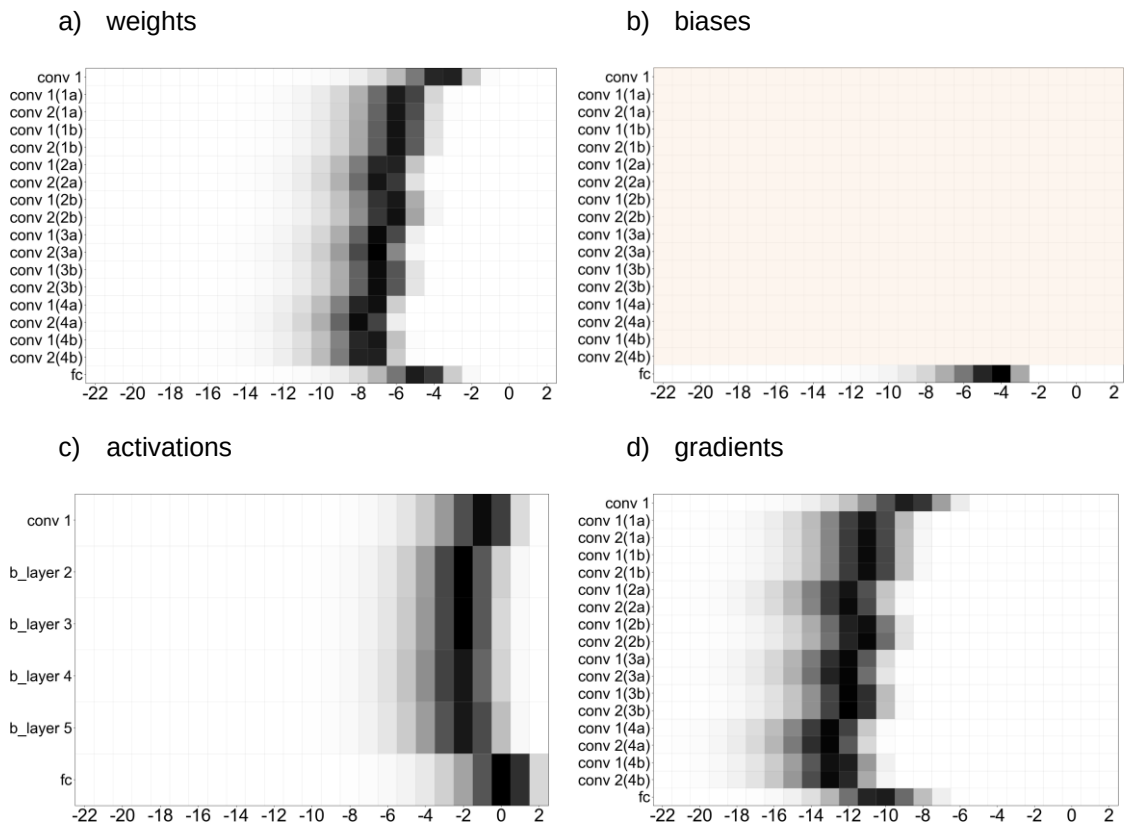


Fig. 3.16. ResNet-18 exponent utilization (normalized over layer) during training on CIFAR100 dataset. The darker the color the higher the utilization. Activations were grouped for clarity of the diagram. Layers: conv (x) – convolution (basic block), fc – fully connected, b_layer – combined layers into basic blocks

The presented results clearly show that in case of the tested networks, the bit count reserved for a regular IEEE-754 32-bit floating-point type is underutilized and causing some of the exponent range to be wasted. According to Tong et al. (2000) [86] usage of 8-bit floating-point representation requires up to 4 times less storage capacity and runtime memory in comparison to 32-bit data type. Moreover, switching to 8-bit floating-point multiplications allows to reduce power consumption to less than a third of a regular 32-bit based unit. Such observations give an interesting ground for further experiments on floating-point precision limitation for NN training. Besides possible mantissa bit count limitation, focusing on the efficient exponent representation gives an additional room for computational and memory resource savings. As presented, in many cases numeric values used during the training could be effectively stored in 4 or 5 bits. This poses a question regarding new resource efficient floating-point formats for NN parameters. Modification

of parameters' type and optimization of their bit count utilization would help to limit the cost of the NN training process and maintain its regular 32-bit floating-point accuracy. The next chapter presents the method's proposal in this area.

4 EXPERIMENTS AND RESULTS

The presented findings (section 3.3) clearly show that the straightforward limitation of parameters' bit count does not provide a solution for more efficient NN training. Although the operations are less computationally complex, the decline of the model's accuracy undermines legitimacy of resource savings. Various research confirmed that more sophisticated methods are necessary for NN training with limited precision [8] [141] [142]. However, the optimal method has not been established yet, the areas with promising outcome include parameters' format changes [75], rounding techniques [5] [6] and quantization [7] [16].

This dissertation describes a new method of precision limitation for convolutional neural network training with low bit count variables. It combines recent knowledge and techniques verified by the research community as application of rounding with new proposals of floating-point representation and exponent utilization. Moreover, the results of the extensive method's verification are presented based on commonly used, benchmark CNN architectures.

This chapter starts with a detailed explanation of the proposed method including key elements in terms of incorporated techniques and implementation details. Then the conducted experiments are discussed in a step-by-step manner. The last part presents a summary of the results achieved for NN training with limited precision on the key target parameter representation formats.

4.1 Method proposal

The proposed method provides a mechanism for training NN with limited precision, floating-point variables. The regular neural topologies are based on the standard 32-bit IEEE-754 format. The technique developed in this thesis enables weights, biases, activations and gradient limitation to custom floating-point representation in the range of an initial 32-bit variable. It includes the following mechanisms:

- Introduction of asymmetric exponent representation.
- Support for exponent offset mechanism allowing for values range adjustment.
- Application of stochastic rounding during the process of variable limitation.
- Denormalized values utilization for a limited precision floating-point type.

Analogous to the previously presented floating-point limitation flow, the proposed framework is executed solely in the software layer. This allows for direct comparison of the previously used method of raw floating-point limitation presented in section 3.2 of this dissertation and follows similar techniques for experimentation provided by utilities as QPytorch [162] or CPFloat [163]. The proprietary, hardware independent implementation has been selected for this research in order to avoid the limitations introduced by existing devices or software libraries when it comes to specific bit ranges, custom data type formats or supported topologies. Fig. 4.1 presents an overview of the framework used for the experiments described in this chapter.

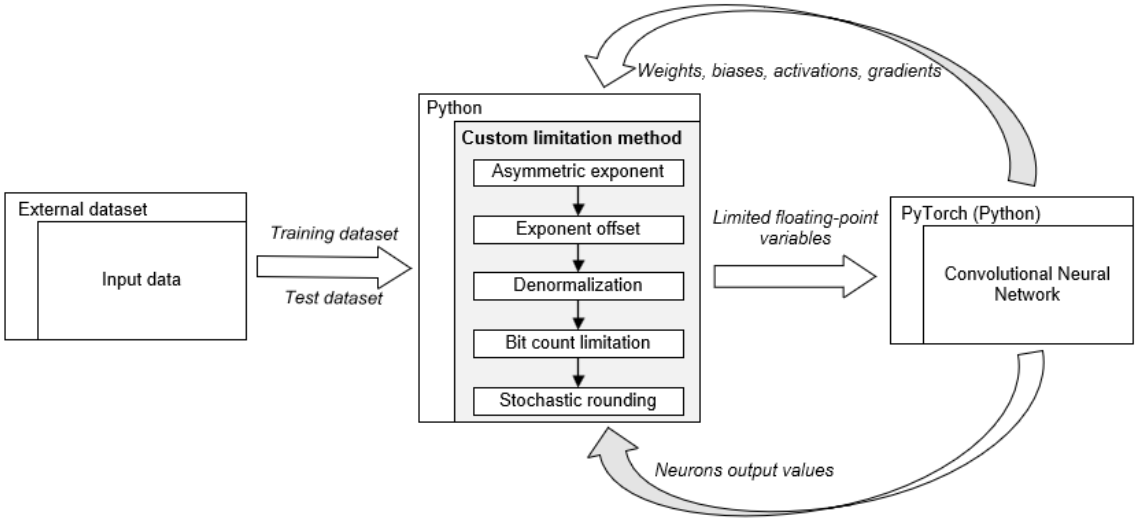


Fig. 4.1. Overview of the proposed custom floating-point limitation method [154]

4.1.1 Asymmetric exponent

The exponent utilization experiments conducted in section 3.4 of this dissertation showed that a significant portion of bit usage focuses on a limited range of its negative values. Hence, a part of the available exponent's bits is not utilized. Such a characteristic is especially inefficient in the case of floating-point variables with limited precision. The proposed method addresses this issue with the introduction of the asymmetric exponent representation which assigns all exponent bits to its negative values only. This way the variable can represent a wider range of small values required during the training and maintain an unchanged bit count. Although such an approach significantly improves the availability of efficiently utilized exponent values, 8- or lower-bit count variables may still not provide a sufficient range for NN training. In the case of 3- or 4-bit exponents, it is crucial to support representation for most often occurring values in the network. Hence, the proposed asymmetric exponent includes an additional offset which can shift available exponent values by a scalar in order to adjust the parameters' dynamic range to a particular topology and dataset. This functionality can be treated as an additional hyper-parameter during the training and then be flexibly adjusted across the following epochs or specific layers of the network. Table 4.1 presents a comparison between different representations of 8-bit floating point variables. It enlists details of IEEE-754-like data types with different exponent's representation, including variants with an asymmetric and asymmetric exponent with an offset.

Table 4.1. Comparison of the proposed exponent representations with IEEE-754 types

	Total bits	Exponent bits	Exponent bias	Exponent min	Exponent max
32-bit IEEE floating-point	32	8	127	-126	127
16-bit IEEE floating-point	16	5	15	-14	15
8-bit floating-point	8	4	7	-6	7
8-bit floating-point with asymmetric exponent	8	4	14	-13	0
8-bit floating-point with asymmetric exponent and offset set to -2	8	4	16	-15	-2

An important limitation when it comes to implementation of the asymmetric exponent format over a regular 32-bit floating-point type is the maximum exponent range available for 8 bits. In order to map the asymmetric exponent into a full precision variable, it is important to not exceed the range between -126 and 127 defined by the IEEE-754 standard. Although this might be a blocking issue for larger variables, it is negligible for the main focus of this dissertation which revolves around low bit variables as 8-bit floating-point. In the case of higher exponent bit counts investigated in this work, the exponent range was strictly limited within 32-bit IEEE-745 range boundaries. Fig. 4.2 shows the difference between precision limitation with and without application of the asymmetric exponent method. Although the final result is always mapped to a generally available IEEE-754 32-bit floating-point variable, the range of the exponent changes. In case of 4 bits, it can contain values between -6 and 7, however, the value shift provided by both asymmetric exponent and exponent offset techniques significantly modifies this range without any changes to the constrained bit count. This way the final variable's representation range from -15 to -2 is much more aligned with the numeric range required by the chosen NN training.

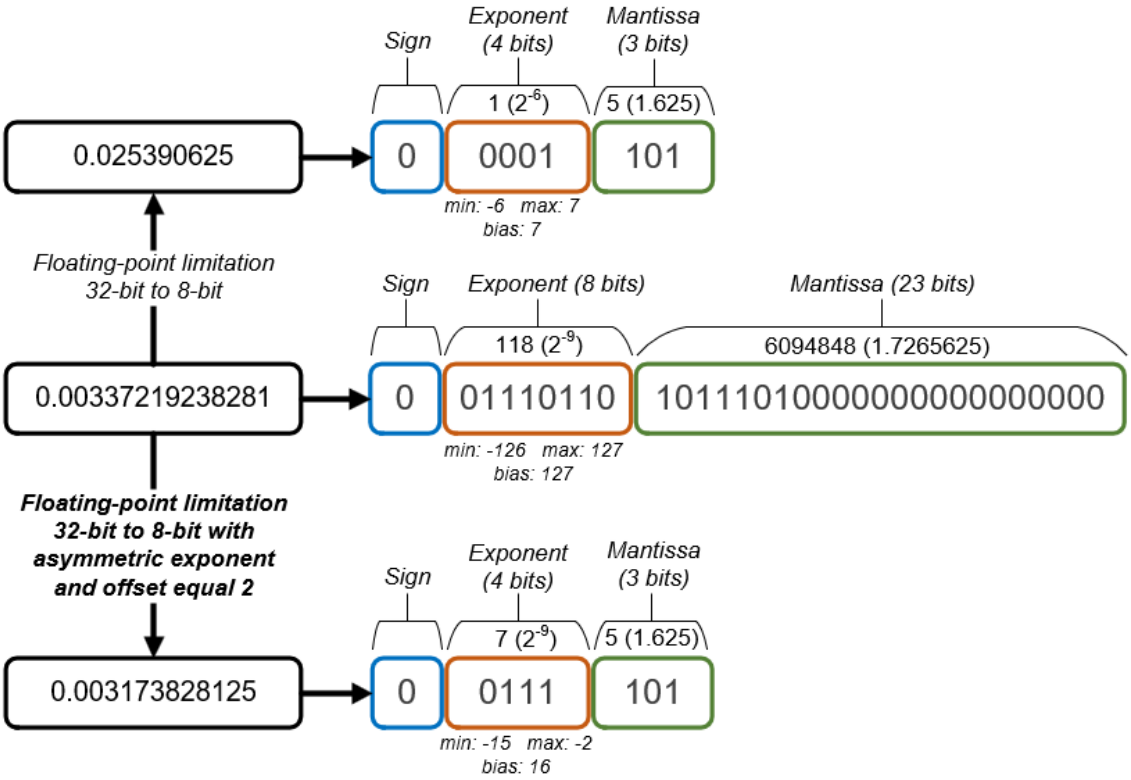


Fig. 4.2. IEEE-754 32-bit floating-point conversion to 8-bit format with and without an asymmetric exponent

As a part of the proposed limitation method, the asymmetric exponent functionality is implemented in the software layer only. Thus, the final values of the limited variables are stored in 32-bit floating-point format. The translation required for the 8- to 32-bit mapping is shown in Fig. 4.3. Additionally, Fig. 4.4 presents a simplified pseudo code showing implementation details related to this feature. The operations related to the mantissa, presented in Fig. 3.2, have been omitted to improve readability of the code snippet.

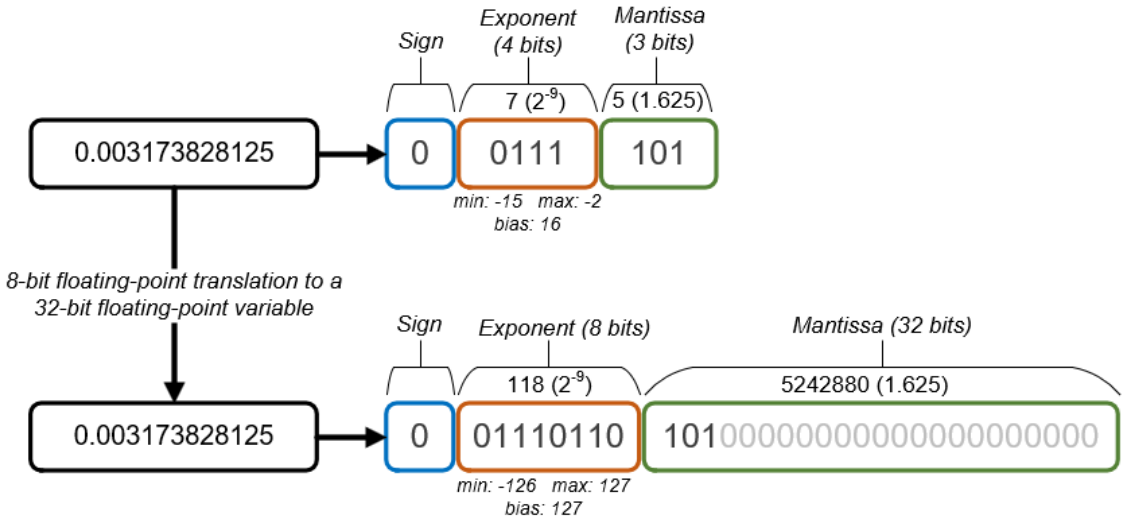


Fig. 4.3. An example of 8-bit floating-point translation to a 32-bit floating-point variable

```
procedure apply_asymmetric_exponent(x, new_exponent_bit_count,
                                    exponent_offset)
      sign = extract_sign(x)
      mantissa = extract_mantissa(x)
      exponent = extract_exponent(x)

      asymmetric_exponent_min =
                  get_assymetric_exponent_min(new_exponent_bit_count)
      asymmetric_exponent_max =
                  get_assymetric_exponent_max(new_exponent_bit_count)

      asymmetric_exponent_min_shifted =
                  asymmetric_exponent_min + exponent_offset
      asymmetric_exponent_max_shifted =
                  asymmetric_exponent_max + exponent_offset

      asymmetric_exponent_min_shifted_trimmed =
      trim_to_ieee754_exponent_range(asymmetric_exponent_min_shifted)
      asymmetric_exponent_max_shifted_trimmed =
      trim_to_ieee754_exponent_range(asymmetric_exponent_max_shifted)

      new_exponent_value =
                     fit_into_assymetric_exponent_boundries(exponent,
                     asymmetric_exponent_min_shifted_trimmed,
                     asymmetric_exponent_max_shifted_trimmed)

      return asymmetric_variable = sign | new_exponent_value |
             mantissa
```

Fig. 4.4. Simplified pseudocode of asymmetric exponent transformation implementation

4.1.2 Stochastic rounding

Rounding or trimming of a numerical value is an inevitable part of its precision limitation. Stochastic rounding is one of the techniques successfully adapted for NN training by multiple researchers [6] [7]. According to recent studies, this method proved to be helpful for maintaining the NN accuracy in case of both floating-point and fixed-point parameters limitation as it aims to statistically preserve information about limited values which is a key factor while using it for NN training [5] [92]. Thanks to this property, the expected error of rounding is zero [164]. Fig. 4.5 shows an example of the floating-point limitation with stochastic rounding.

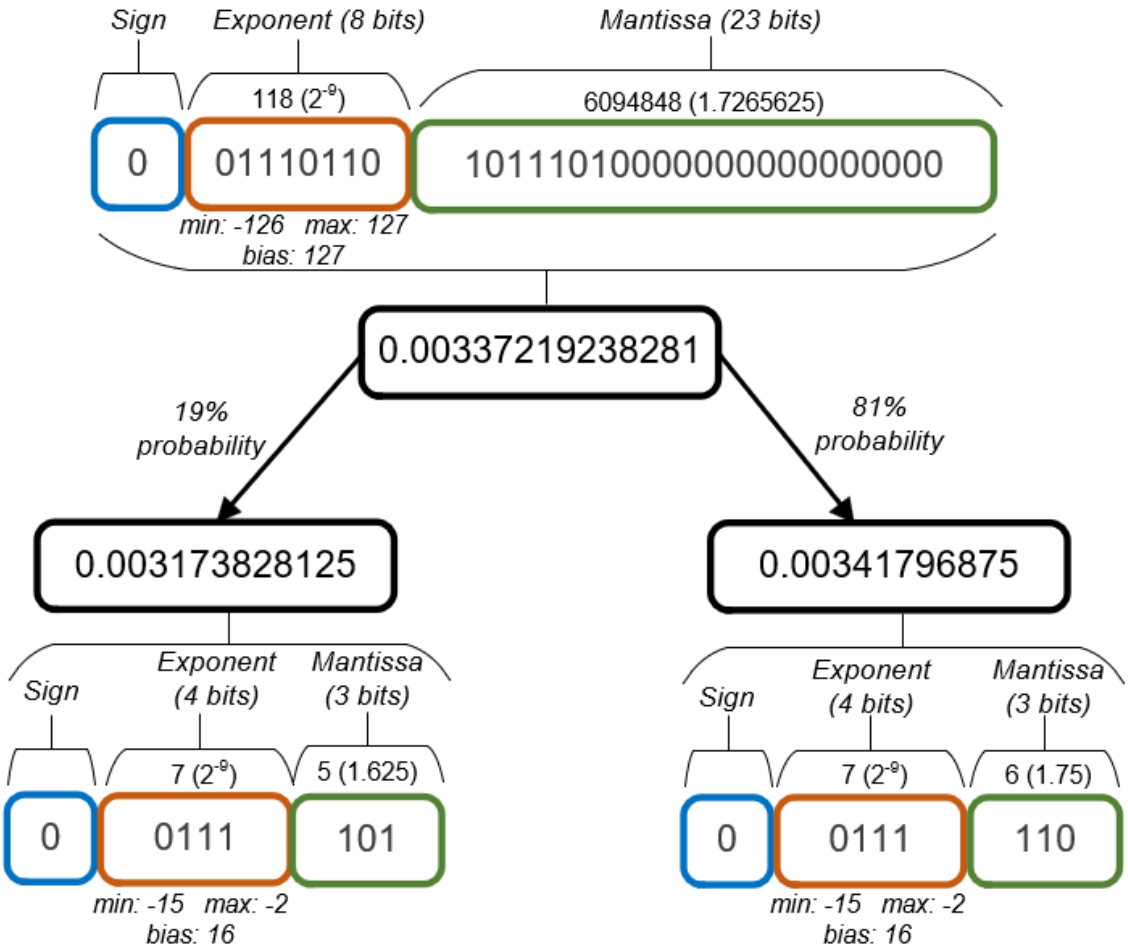


Fig. 4.5. Floating-point limitation results variants and their probabilities with stochastic rounding enabled

As already presented in section 2.3.2 with equations (2.3) and (2.4), the implementation of stochastic rounding techniques may vary depending on how the probability of the rounding is determined. In the case of the presented framework, it is based on the distance of the variable to the limited counterparts. First, the upper and lower boundary of the limitation algorithm is calculated for a 32-bit floating point input. Then the relative distances between the limited input and the boundaries are measured. Based on that, the framework establishes the probability of rounding direction which depends on the output of the software random number generator. Fig. 4.6 gives an overview of such an implementation in a form of pseudo code.

```
procedure apply_stochastic_rounding(x, exponent_bitcount,
                                    mantissa_bitcount)

      upper_boundry, lower_boundry = get_limitation_boundries(x,
                                exponent_bitcount, mantissa_bitcount)

      upper_distance = abs(abs(x) - upper_boundry)
      lower_distance = abs(abs(x) - lower_boundry)

      round_up_probability = upper_distance / (upper_distance +
                             lower_distance)
      random_float_number = get_random()

      if random_float_number <= round_up_probability:
            rounded_x = upper_boundry
      else:
            rounded_x = lower_boundry

      return rounded_x
```

Fig. 4.6. Simplified pseudocode of stochastic rounding implementation in the precision limitation framework

In order to increase readability of Fig. 4.6 the details of establishing limitation boundaries have been hidden under a *get_limitation_boundries* method as it heavily depends on details already presented in Fig. 3.2 and Fig. 4.4 which cover limitation internals. It is important to mention that the limitation process must be executed only once for one of the boundaries as establishing the other one is a simple operation of mantissa decrease or increase and exponent adjustment in case of its under or overflow.

4.1.3 Denormalized values

An additional feature that can be enabled in the proposed method is a denormalization of the limited values. As with the IEEE-754 approach presented in section 2.3.1, the variable format proposed in the training optimization method introduces denormalized range that covers a wider range of values close to zero. It has been introduced in order to reduce the rounding error between original and limited value. Analogously to IEEE-754, the leading bit of mantissa is interpreted as zero for denormalized values which results in a wider exponent range at the expense of mantissa bits. Fig. 4.7 shows the results of this mechanism based on the proposed framework calculations.

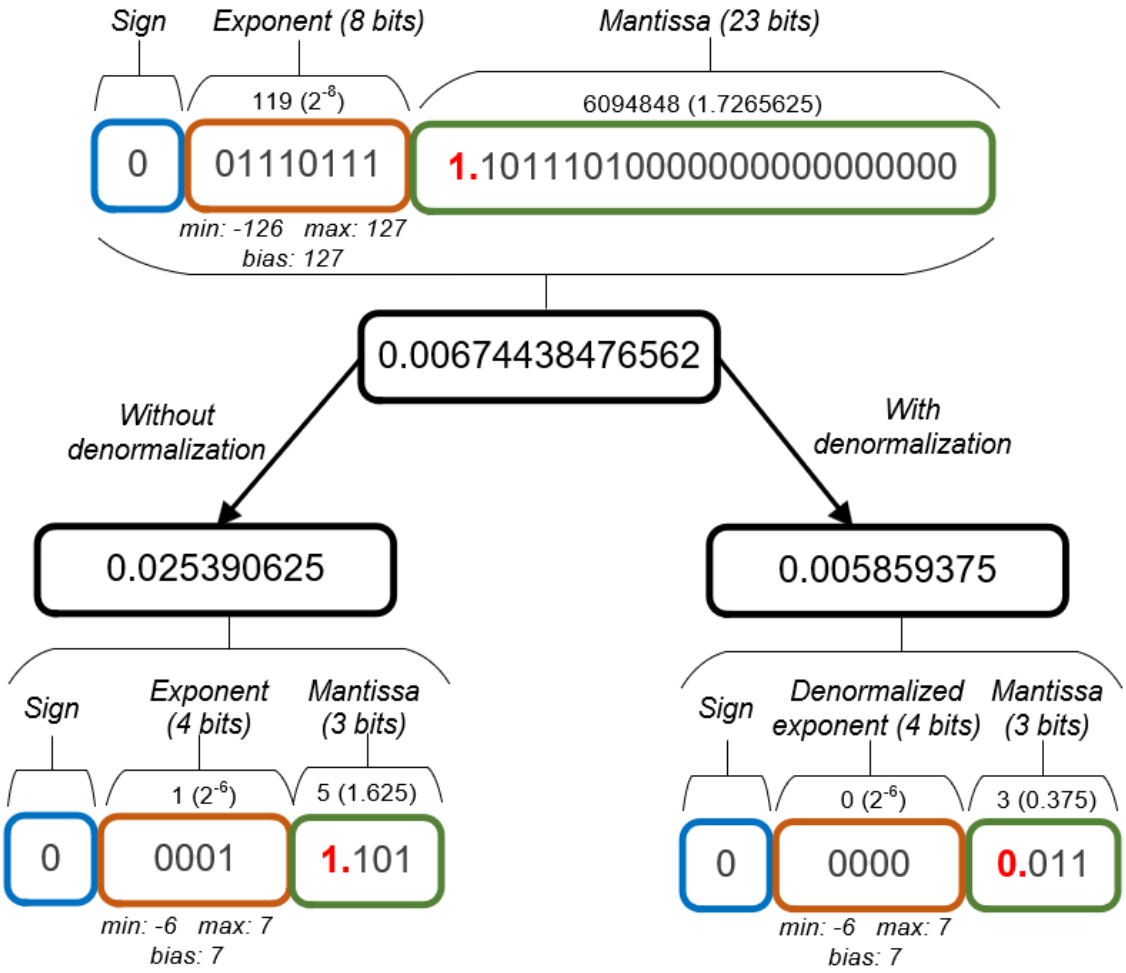


Fig. 4.7. Limited 8-bit floating-point denormalization (without mantissa rounding). The hidden leading mantissa bit is marked in red.

The software framework implementation of a denormalization feature is achieved by extending limited exponent range based on available mantissa bits. With the usage of right bit shift on mantissa's value, it is possible to gradually divide it by 2 as long as there is at least one significant bit left in the mantissa's representation. Such a limited mantissa value can be then once again translated to normalized floating-point representation with analogous left bit shift operations and adjustment of exponent's value in order to match the same number in a standard IEEE-754 format. Fig. 4.8 considers an example of simple software layer implementation of denormalization.

```
procedure apply_denormalization(x, exponent_bitcount,
                                mantissa_bitcount)
      sign = extract_sign(x)
      mantissa = extract_mantissa(x)
      exponent = extract_exponent(x)

      lowest_exponent_value =
                        get_lowest_exponent_value(exponent_bitcount)

      exponents_difference = exponent – lowest_exponent_value

      if exponents_difference <= mantissa_bitcount
            limited_mantissa = limited_mantissa >>
                               exponents_difference
            limited_mantissa = limited_mantissa <<
                               exponent_difference

            denormalized_x = sign | lowest_exponent_value |
                                    limited_mantissa
            return denormalized_x
      else
            return 0
```

Fig. 4.8. Simplified pseudocode of a custom denormalization implementation in the precision limitation framework

4.1.4 Method's application

Although all code representations in this chapter had been focused on a single value method application to avoid presentation complexity, the real-life implementation focuses solely on tensor-based operations for efficiency purposes. The productization of the proposed technique may strictly depend on the training hardware capabilities and should be focused on its optimal utilization. It is especially important in case of stochastic rounding and denormalization where simplified implementation may highly limit the complexity and computational power required for additional steps such as random number generation or denormalization enablement.

The presented technique aims to provide a consistent precision limitation method for training NN with low bit count variables. It is important to highlight that applicability to various NN architectures may vary, thus establishing an optimal bit count of limited floating-point variables can differ per chosen architecture. Selection of proper bit count should be perceived as a hyper-parameterization in a training stage. Although the initial experiments used constant values of these hyper-parameters during the whole training process, they can be dynamically modified for specific epochs, layers or parameter types. This is often the case with recent NN limitation studies [146] [142]. A similar approach can be applied to the asymmetric exponent with offset or denormalization. Dynamic modification of these parameters can be used to achieve a mixed-precision approach without variable's bit count modification, which creates an advantage when simplification of hardware design requirements is needed.

4.2 Conducted trainings

The proposed method has been validated on the same set of NN architectures and datasets as presented in section 3.2. This includes LeNet with MNIST, AlexNet and ResNet-18 with CIFAR10 and CIFAR100. Such an approach allows a straightforward, one to one comparison and measurements of accuracy gains achieved with the use of the proposed precision limitation method for low bit count variables. In order to ensure that the training environment is the same, all hyper-parameters remain unchanged, and no stop-loss mechanism has been enabled. The only difference introduced during the process was the enablement of the features vital to the proposed method as asymmetric exponent with offset, stochastic rounding and denormalization mechanism as presented in Table 4.2. It is important to highlight that activations are the only parameters that do not leverage asymmetric exponent and exponent offset features. The reason behind it is a different range of exponent values required by these parameters' values, as explained in section 3.4.

Table 4.2. Features of the proposed method per neural network's parameter type

Feature / Parameter	**Weights**	**Biases**	**Activations**	**Gradients**
Bit count limitation	✓	✓	✓	✓
Asymmetric exponent	✓	✓	✗	✓
Exponent offset	✓	✓	✗	✓
Denormalization	✓	✓	✓	✓
Stochastic rounding	✓	✓	✓	✓

4.2.1 LeNet

LeNet architecture provides the simplest test case scenario for the proposed precision limitation method in terms of both parameters' count and classification task complexity. Hence, a considerably lower number of bits is required in order to achieve the 32-bit baseline set to 96.18% of the network's accuracy. Fig. 4.9 provides classification results for LeNet architecture trained across a wide range of bit count variants with asymmetric exponent offset set to -2.

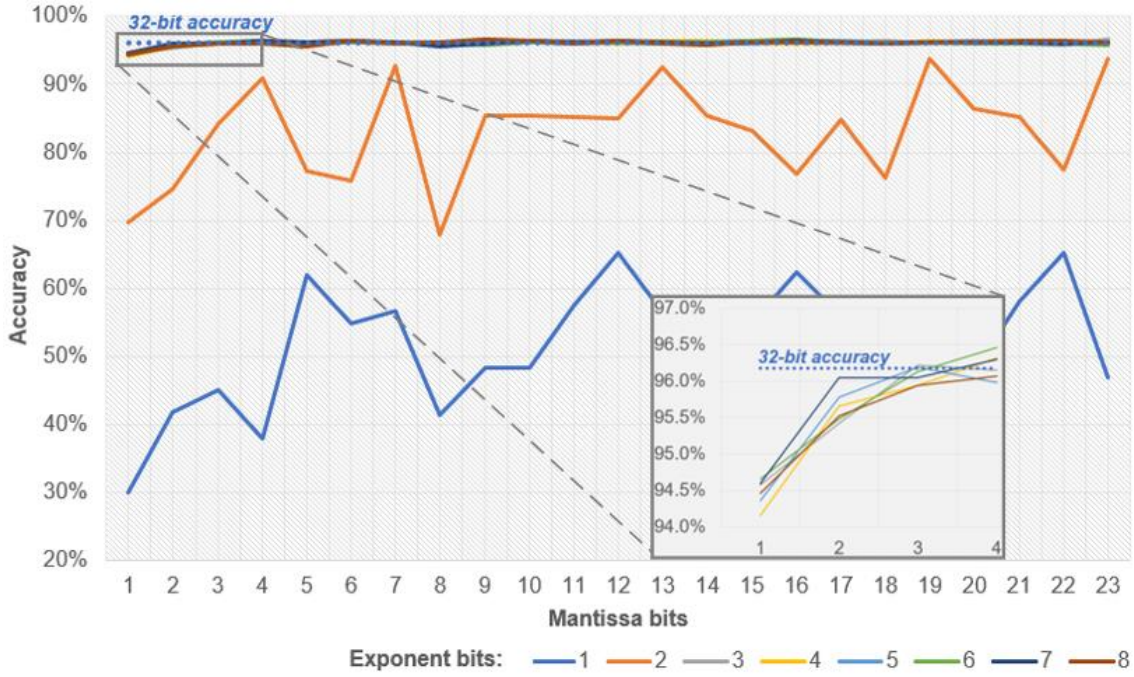


Fig. 4.9. LeNet (MNIST) training results with the proposed limitation method

Both 1- and 2-bit exponents do not provide a stable base for LeNet training with MNIST dataset. The network cannot achieve the baseline accuracy with such exponents no matter how many mantissa bits are available during the training. The only exception, close to the baseline result, can be observed with 2-bit exponent and 23-bit mantissa. It is important to remark that in case of such low exponent bit counts the training appears to be unstable and introduces a lot of accuracy variations between repetitive iterations which may be also reinforced by randomization of initial model's weights for each training scenario. Although 2-bit exponent proved to be insufficient for this training example, there is a clear improvement for variables with 3- and larger bit exponents. In case of this network topology, even 3-bit exponent and 1-bit mantissa floating point provides close to baseline accuracy of 94.58%. Even better results can be observed for the targeted 8-bit floating-point variable with 4-bit exponent. This type provides the ability to train the network to the accuracy of 95.98%. The best accuracy of 96.15% is achieved for 8-bit floating-point type with 3-bit exponent and 4-bit mantissa. It is important to remark that it provides above 20 percentage points improvement in comparison to the previously proposed method, utilizing a solely asymmetric exponent feature with a result of 75.89% [36]. As presented in Fig. 4.9, the further increase of both exponent and mantissa bit counts does not provide any significant improvement in comparison to the proposed 8-bit floating point parameters which is represented by flattened accuracy lines for the trainings with exponents above 2-bits.

4.2.2 AlexNet

The next experiment involved AlexNet with CIFAR10 dataset. The targeted 32-bit baseline amounted to 74.39% of accuracy. As previously mentioned, in comparison to LeNet training an additional parameterization change of asymmetric exponent offset has been introduced for AlexNet and ResNet networks by decreasing this parameter value from -2 to -3.

The change was dictated by different exponent utilization observed during FP32 trainings (section 3.4). Fig. 4.10 presents training results achieved with the use of the proposed limitation method across multiple bit counts variants over 10 epochs.

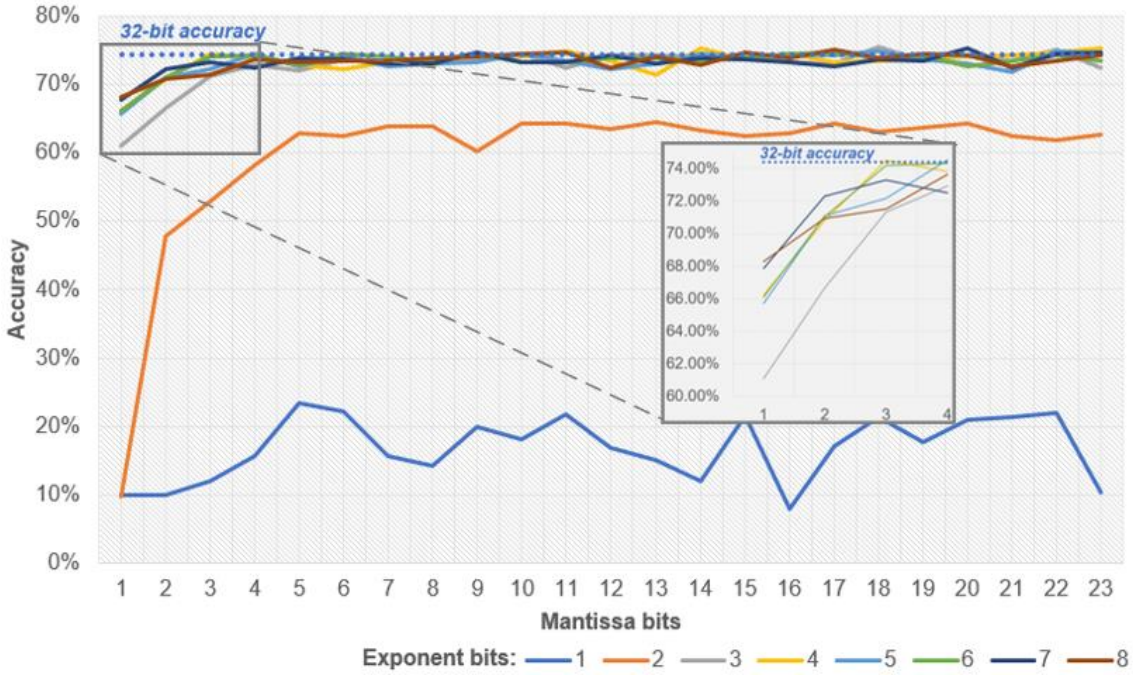


Fig. 4.10. AlexNet (CIFAR10) training results with the proposed limitation method

Based on the cross-validation results conducted on AlexNet with CIFAR10 it can be observed that the proposed technique allows to train this network without accuracy degradation on 8-bit floating-point type. As expected, 1-bit and 2-bit exponents could not provide a sufficient dynamic range for AlexNet training, which is understandable based on the poor results of LeNet trainings with such low bit count types. The convergence of training results can be observed for 3-bit and higher exponent bit counts. Although 5-bit floating point does not provide satisfactory accuracy with the result of 61.09%, the 3-bit exponent and 4-bit mantissa type achieves much better accuracy of 72.94%. It should be noticed that along the increasing bit count the results fluctuation is higher than in case of the flattened LeNet chart, which can be explained by increased size of the topology and complexity of the classification problem. Nevertheless, 4-bit exponent and 3-bit mantissa type provide a satisfactory result of 74.5% which surpasses the 10 epochs baseline result of 74.39%.

The same network has been also validated on CIFAR100 dataset providing a bit more complex classification scenario. Both parameterization and experiment settings remained unchanged. The results of such cross validation are presented in Fig. 4.11.

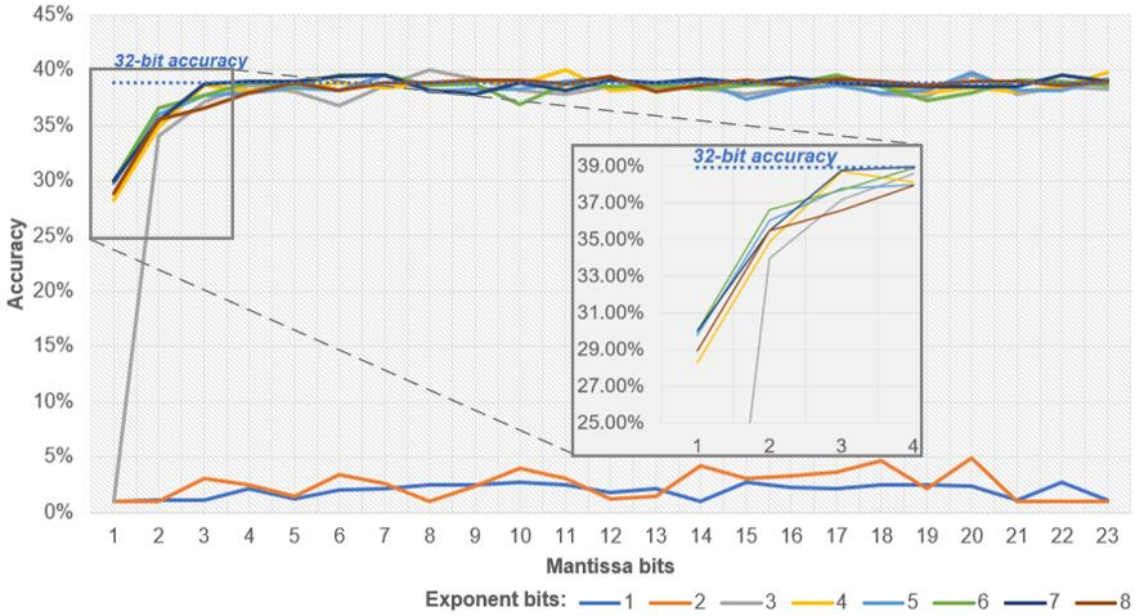


Fig. 4.11. AlexNet (CIFAR100) training results with the proposed limitation method

Application of a more demanding dataset for the AlexNet based classification had a clear influence on its accuracy across lower bit counts. In comparison to CIFAR10 classification, the degradation can be observed for 2-bit exponent, which provides much worse results across all mantissa bit counts. Similar cases can be observed for 5-bit type with 3-bit exponent. It is vivid that such parameters are unable to train the network. The first satisfactory results can be observed for 6-bit floating-point type with accuracy of 33.98%, still showing a significant degradation from the baseline of 38.93%. Unsurprisingly, the 8-bit floating-point type with 4-bit exponent surpassed the 3-bit one with an accuracy of 38.69% versus 38.59%, both slightly underperformed in comparison to the 32-bit baseline of 38.93%.

4.2.3 ResNet

ResNet-18 topology with CIFAR dataset was the last neural architecture used for the validation of the proposed method. The CIFAR10 32-bit baseline for the presented environment for this model has been established to 77.08%. Fig. 4.12 shows training results for the ResNet-18 network with the use of the proposed limitation method on multiple bit count variants across 10 training epochs. It is important to remark that both the environment and parameterization of the training scenarios remained unchanged in comparison to previously presented AlexNet trainings.

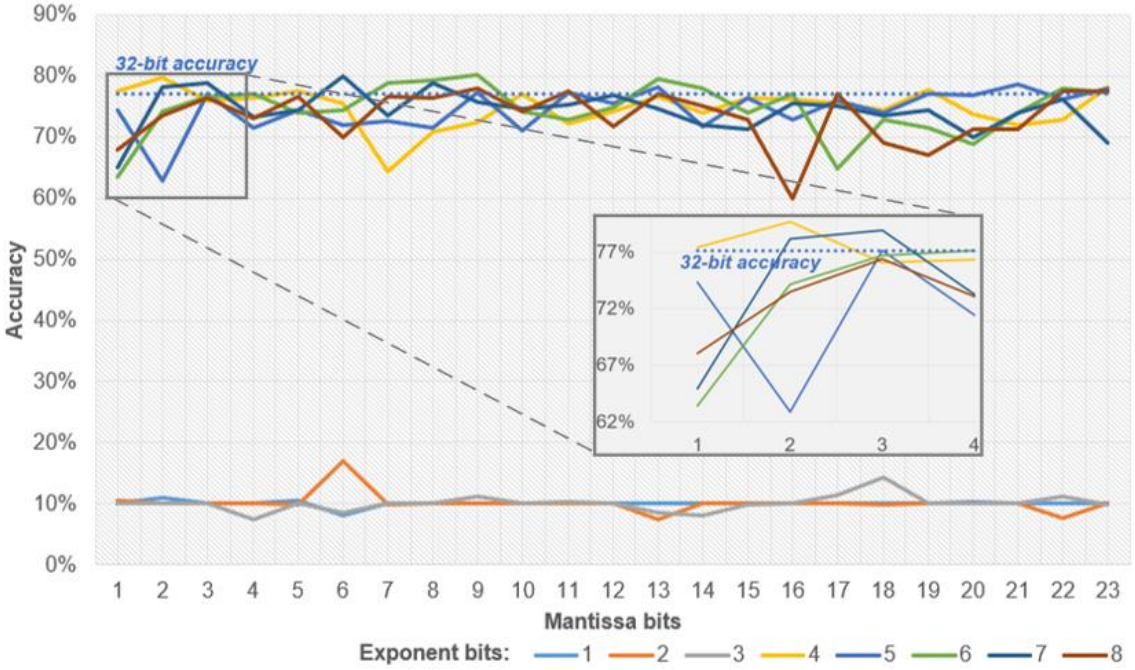


Fig. 4.12. ResNet-18 (CIFAR10) training results with the proposed limitation method

Based on the cross-validation results, it can be observed that limitation of the network's parameters bit count has a much larger influence on the ResNet-18 classification in comparison to AlexNet, even though the training dataset remains unchanged. In this case, not only 1- and 2-bit exponents are unable to provide sufficient range for training the network, similar cases can be observed for 3-bit exponent limitation across the full range of mantissa bit counts. The network is able to converge classification results for 4-bit and higher exponents. The best 76.01% accuracy comparable to the baseline is achieved for 4-bit exponent and 3-bit mantissa type with degradation of around 1 percentage point. Interestingly, the best low bit count accuracy for ResNet-18 on CIFAR10 has been achieved for 7-bit type with 4-bit exponent and 2-bit mantissa surpassing the baseline with 79.65% of accuracy.

It had to be mentioned that much higher accuracy fluctuation can be observed for all tested scenarios. This behavior may be the result of a too short training period for a ResNet-18 network resulting with volatile 10 epoch results. Such a negative effect is greatly reduced when we consider best epochs for ResNet-18 trainings presented in Fig. 4.13. In this case the fluctuations are much lower and the overall shape of the chart lines for higher bit counts correlates with those seen for LeNet and Alexnet.

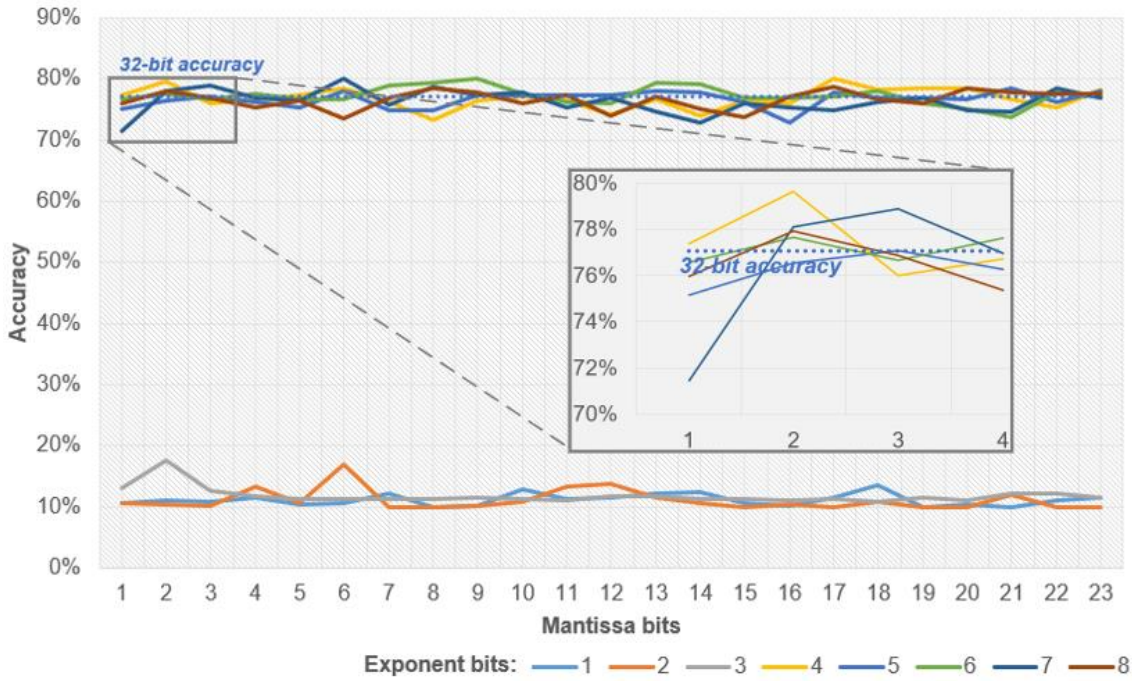


Fig. 4.13. ResNet-18 (CIFAR10) training results with the proposed limitation method – best validation epoch results selected

Even higher fluctuation of results can be observed for ResNet-18 training with the CIFAR100 dataset presented in Fig. 4.14. The baseline 32-bit accuracy for ResNet-18 on CIFAR100 in the present experimental environment has been established at 39.54% of accuracy.

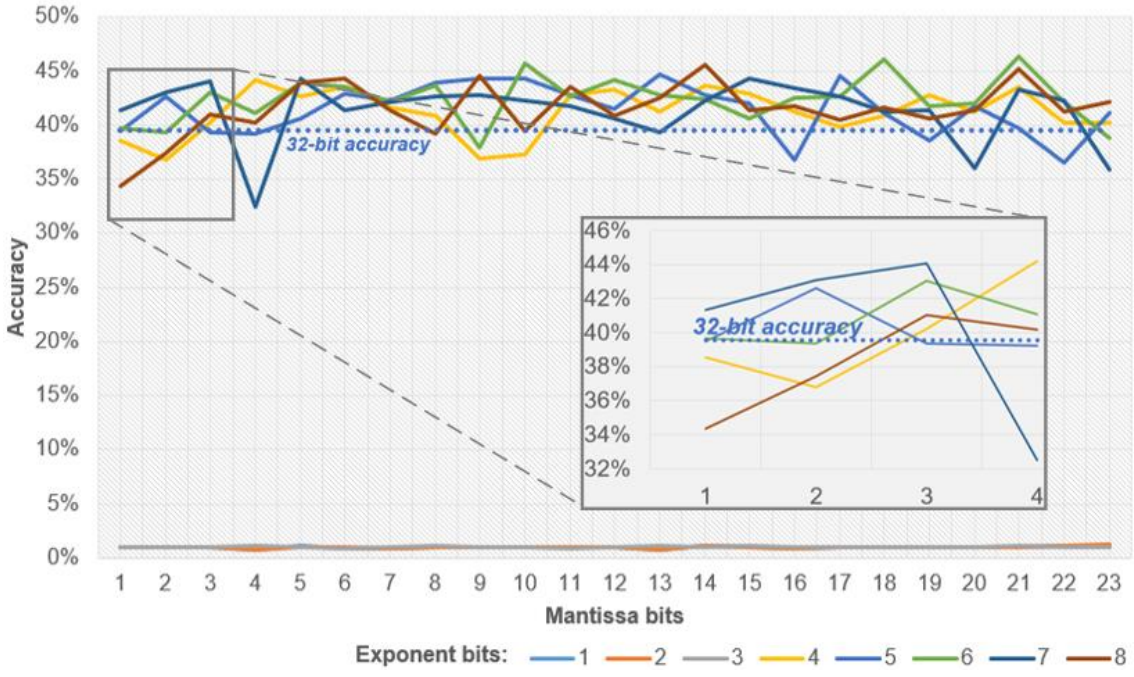


Fig. 4.14. ResNet-18 (CIFAR100) training results with the proposed limitation method

Similarly to CIFAR10, 3-bit and lower exponents do not provide enough range for neural network results convergence. Fig. 4.15 shows the best accuracy across 10 epochs in order to present a better picture of the training results. Although training on 8-bit floating-point types

provides range accuracy of 40.21%, much better results are observed in the case of 5-bit exponent. The 5-bit exponent and 2-bit mantissa type achieves a surpassing accuracy of 42.62%.

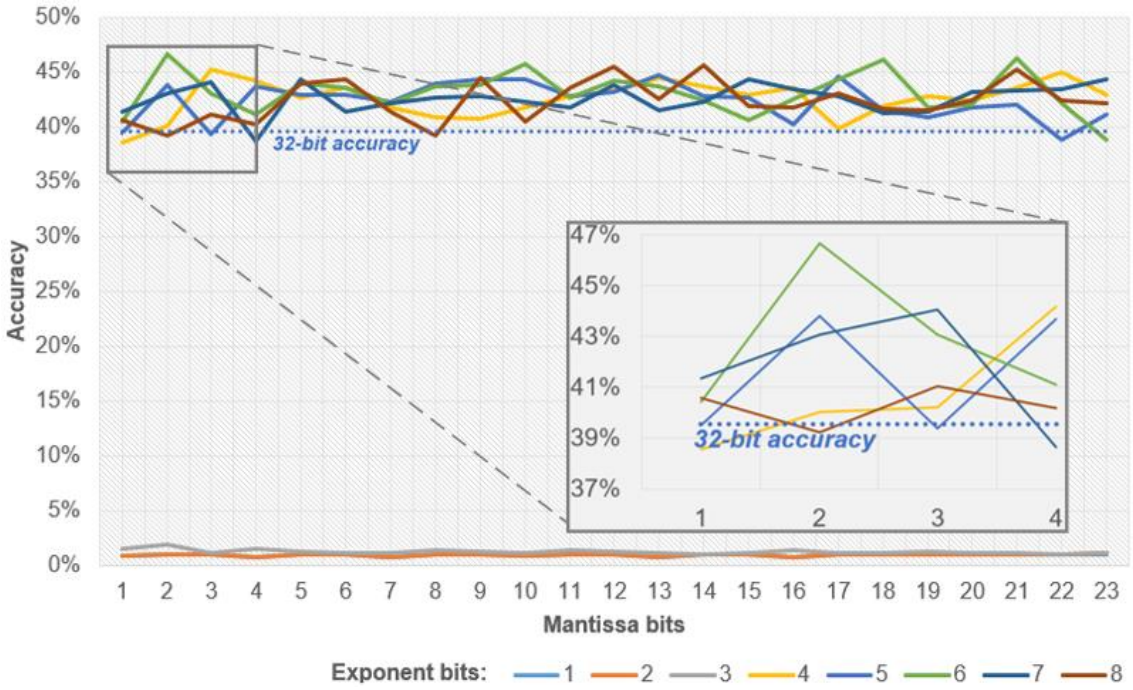


Fig. 4.15. ResNet-18 (CIFAR100) training results with the proposed limitation method – best validation epoch results selected

Although the main focus of the conducted experiments was 8-bit floating-point accuracy, it is clear that, similarly to training hyper-parameterization and the proposed method configuration, the selection of the optimal bit count and parameters type has a key role in efficient utilization of NN training with limited precision. Multiple factors such as NN parameters count, classification task complexity or the epochs number may influence the variables range required by the training procedure. Hence, the proposed method should be considered as an optimization technique rather than the one-fits-all solution.

The experiments outlined in the previous section proved that it is possible to train tested NN architectures with custom 8-bit floating-point parameters without significant accuracy degradation. In less complex cases such as LeNet with MNIST or AlexNet with CIFAR10 it was possible to achieve satisfactory results with only a 5-bit range for utilized variables. Moreover, the experiment's outcome showed that a proper assignment of bits to exponent and mantissa parts may noticeably influence the final accuracy of the tested NN. As confirmed by multiple research, the popularly used 4-bit exponent and 3-bit mantissa 8-bit floating-point type is not always the optimal solution while training the neural topology with limited precision [8] [152] [153]. Hence, additional factors such as topology size, complexity or dataset must be considered. Table 4.3 summarizes the accuracy of the tested networks across all investigated custom 8-bit floating-point variants during 10 epochs. Additionally, results exceeding regular 32-bit training procedures have been underlined in the table.

Table 4.3. Accuracy results of the proposed limitation method across several 8-bit floating-point formats during 10 epochs, the last row presents the IEE-754 32-bit baseline results.

Floating-point variant			**LeNet**	**AlexNet**		**ResNet-18**	
Sign bit count	**Exponent bit count**	**Mantissa bit count**	**MNIST**	**CIFAR10**	**CIFAR100**	**CIFAR10**	**CIFAR100**
1	**1**	**6**	54.84%	22.23%	1.98%	8.02%	0.91%
1	**2**	**5**	77.81%	62.93%	1.46%	9.97%	1.02%
1	**3**	**4**	**96.15%**	72.94%	38.59%	7.34%	1.17%
1	**4**	**3**	95.98%	**74.50%**	**38.69%**	**76.01%**	40.21%
1	**5**	**2**	95.78%	71.10%	36.02%	62.85%	**42.62%**
1	**6**	**1**	94.66%	66.11%	30.00%	63.39%	39.68%
IEEE-754 32-bit baseline							
1	**8**	**23**	96.18%	74.39%	38.93%	77.08%	39.54%

The presented dissertation focuses largely on 8-bit floating-point representation of limited NN parameters because handling of such types is much more efficient from the hardware perspective due to the binary representation. Nevertheless, further limitation of variables used for NN training is possible. The conducted experiments have shown that 5-bit parameters are sufficient for LeNet training with MNIST dataset. Both AlexNet and ResNet required at least 6-bit parameters to converge without significant degradation of classification accuracy for CIFAR10 and CIFAR100 datasets. It is important to remark that in all cases a decrease of the network's accuracy is still noticeable and varies depending on the topology and the dataset used. Table 4.4. presents the bit count variants for each of the tested scenarios that were selected by the author as minimal parameter size for training a particular network topology in the experimentation environment. Although the majority of the results are not at the level of the 32-bit floating-point baseline, there should be a possibility of improving below 8-bit floating-point accuracy for selected networks by further training parameterization or modification of the proposed limitation method. An interesting case is presented by ResNet-18 with CIFAR10 dataset in case of which the achieved result is better than the targeted 8-bit type with 4-bit exponent.

Table 4.4. Accuracy results of the proposed limitation method with minimal usable floating-point bit counts (compared to the 32-bit baseline)

Neural network	Dataset	Total bit count	Exponent bit count	Mantissa bit count	Accuracy	Accuracy change (in percentage points)
LeNet	MNIST	5	3	1	94.58%	-1.68
AlexNet	CIFAR10	6	3	2	73.42%	-0.97
	CIFAR100	6	3	2	33.98%	-4.95
ResNet	CIFAR10	6	4	1	77.37%	0.29
	CIFAR100	6	4	1	38.54%	-1.0

4.3 Method's features impact analysis

The proposed method incorporates multiple techniques as denormalization, stochastic rounding and modification of exponent representation in order to train selected NN architectures with 8-bit floating-point variables at the accuracy level represented by common 32-bit trainings. It is important to analyse how the proposed method's elements impact the network's classification quality. All comparisons presented in this chapter were conducted based on 8-bit floating point-type with 4-bit exponent and 3-bit mantissa.

4.3.1 Exponent shift analysis

Stochastic rounding proved to be a vital solution in terms of NN training with limited precision as it allows to minimize rounding errors impact on the network's quality [5] [6] [7] [92]. The conducted experiment aims to compare the influence of exponent representation change on trainings with 8-bit floating-point and stochastic rounding. The analysis involved trainings on various representations of shifted exponent, including asymmetric exponent representation, regular symmetric exponent and intermediate shifts that can be achieved by exponent's bias modification. In the same manner as with other experiments presented in the dissertation, the analysis has been conducted on LeNet, AlexNet and ResNet models. Fig. 4.16 presents results of such an experiment on the LeNet network with the MNIST dataset.

As presented in Fig. 4.16, a representation of the exponent has a marginal impact when it comes to LeNet accuracy on MNIST dataset. Both regular exponent with a range from -6 to 7 and asymmetric exponent with a range from -13 to 0, achieve high results with a slight advantage for the regular exponent representation. Additional shifts do not significantly modify the accuracy until as far as an asymmetric exponent with offset -6 which represents values range from -6 to -19 and notably degrades accuracy of the network. In order to enable an easier comparison of the proposed method with its previous version proposed by Pietrołaj and Blok (2022) [36], the red line has been marked in Fig. 4.16 to show the 8-bit floating-point LeNet accuracy. The mentioned method did not include denormalization or stochastic rounding implementation, hence accuracy

difference can be seen as a scale of improvement achieved thanks to newly introduced method's techniques. Such a comparison shows that stochastic rounding is a vital element of the method and without its application the network would not be able to achieve 32-bit baseline results with an 8-bit floating-point precision limitation.

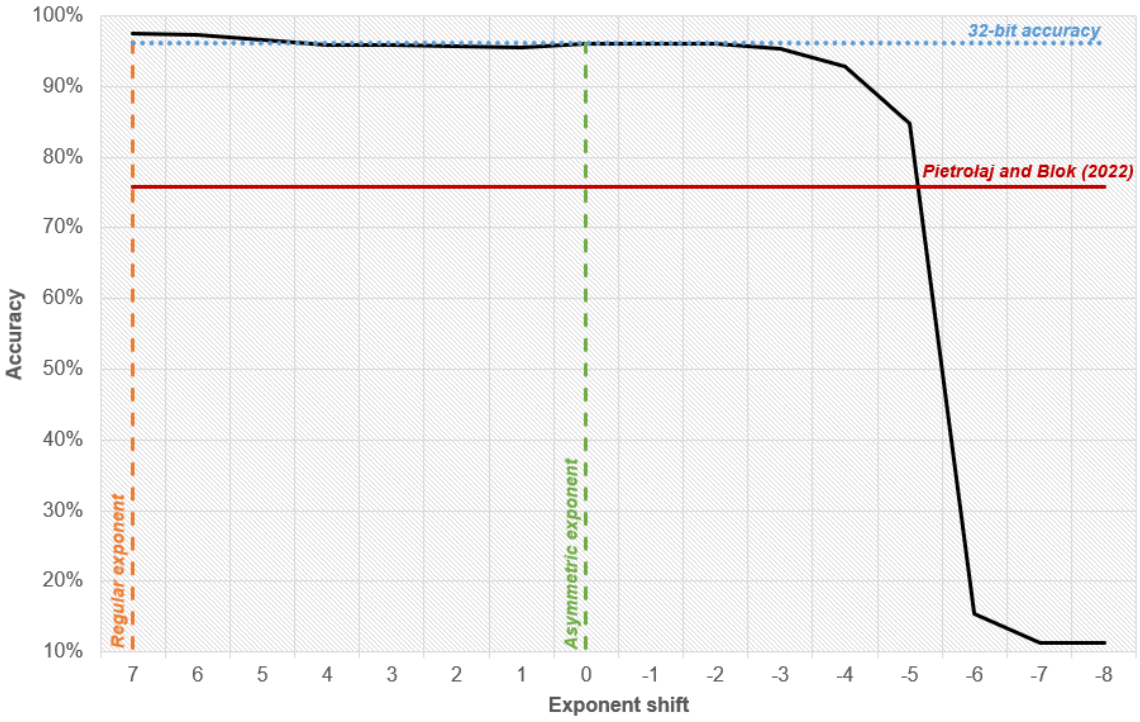


Fig. 4.16. Accuracy of 8-bit LeNet (MNIST) across various exponent shift scenarios

The following analysis involved a more complex convolutional neural network AlexNet. Although with a small model as LeNet the influence of an asymmetric exponent with offset is negligible, it is much more beneficial in this case. The accuracy of AlexNet on CIFAR10 dataset in various exponent representation scenarios is presented in Fig. 4.17.

The AlexNet case shows that the asymmetric exponent has a significant advantage of above 15 percentage points over a regular exponent. Additional shift of -2 of asymmetric exponent allows to tune the training and gain a few additional percentage points of accuracy. Such case noticeably exceeds the previously selected result in section 4.2 for offset of -3 with over 2 percentage points. As with LeNet there is a visible degradation of results once the offset is set below -5 which interferes with the network's ability to converge.

The same analysis of AlexNet results has been conducted on CIFAR100 to verify the impact of a slightly more complex dataset. Fig. 4.18 sums up the results achieved during this experiment.

Both with CIFAR10 and CIFAR100 training variants, the positive influence of asymmetric exponent on AlexNet training accuracy can be noticed. The offset of -3 provides almost 6 percentage points of accuracy improvement in comparison to asymmetric exponent. Similarly to CIFAR10, the regular exponent does not provide enough range to train the network on par with the 32-bit baseline.

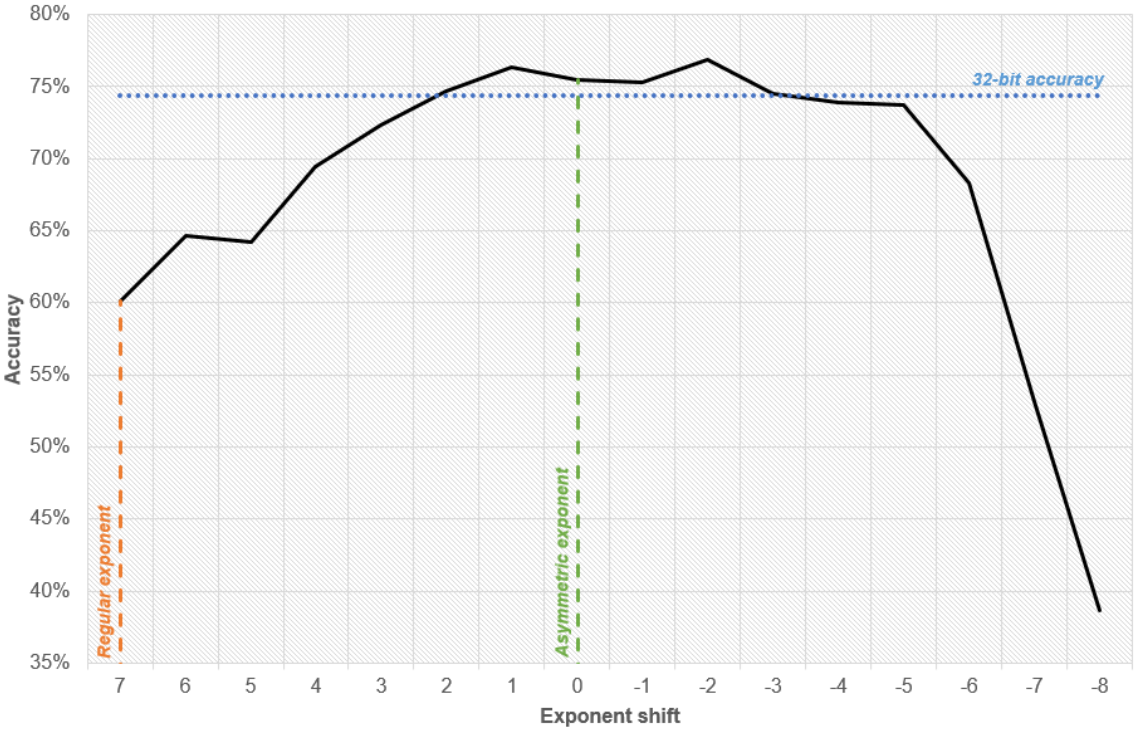


Fig. 4.17. Accuracy of 8-bit AlexNet (CIFAR10) across various exponent shift scenarios

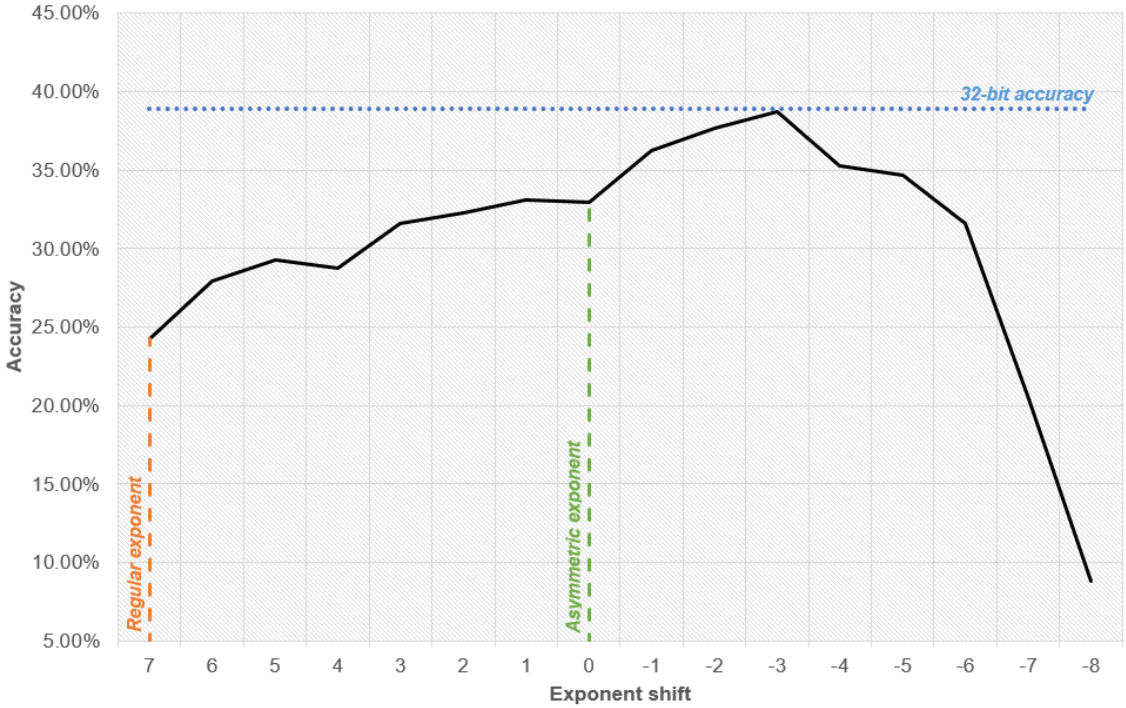


Fig. 4.18. Accuracy of 8-bit AlexNet (CIFAR100) across various exponent shift scenarios

ResNet was the last network tested in terms of various exponent representation variants. The verification involved both CIFAR10 and CIFAR100 datasets. Fig. 4.19 presents ResNet results on the CIFAR10 dataset.

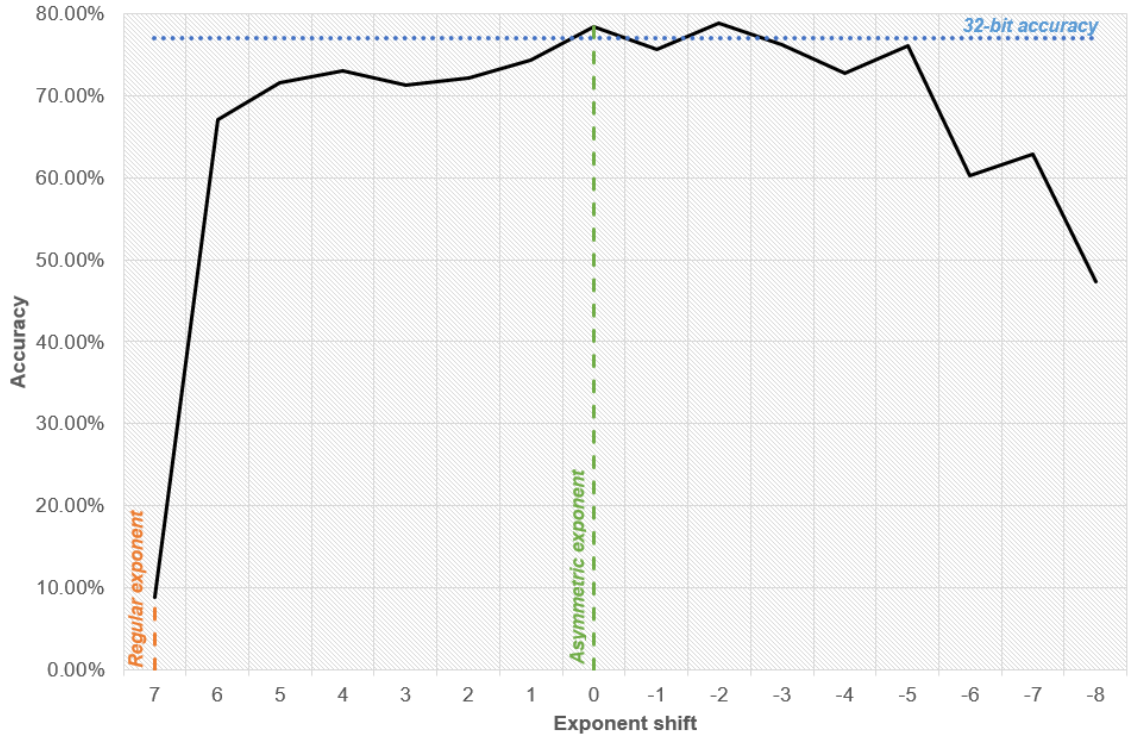


Fig. 4.19. Accuracy of 8-bit ResNet-18 (CIFAR10) across various exponent shift scenarios

As presented in Fig. 4.19, ResNet-18, in contrast to LeNet and AlexNet, could not be trained on CIFAR10 dataset with a regular exponent representation. The best result can be observed for an asymmetric exponent with an offset equal to -2. A similar case can be observed in the case of ResNet and CIFAR100 dataset (Fig. 4.20), although in this case the exponent offsets of 1 and -1 show a clear advantage with the accuracy above 43%.

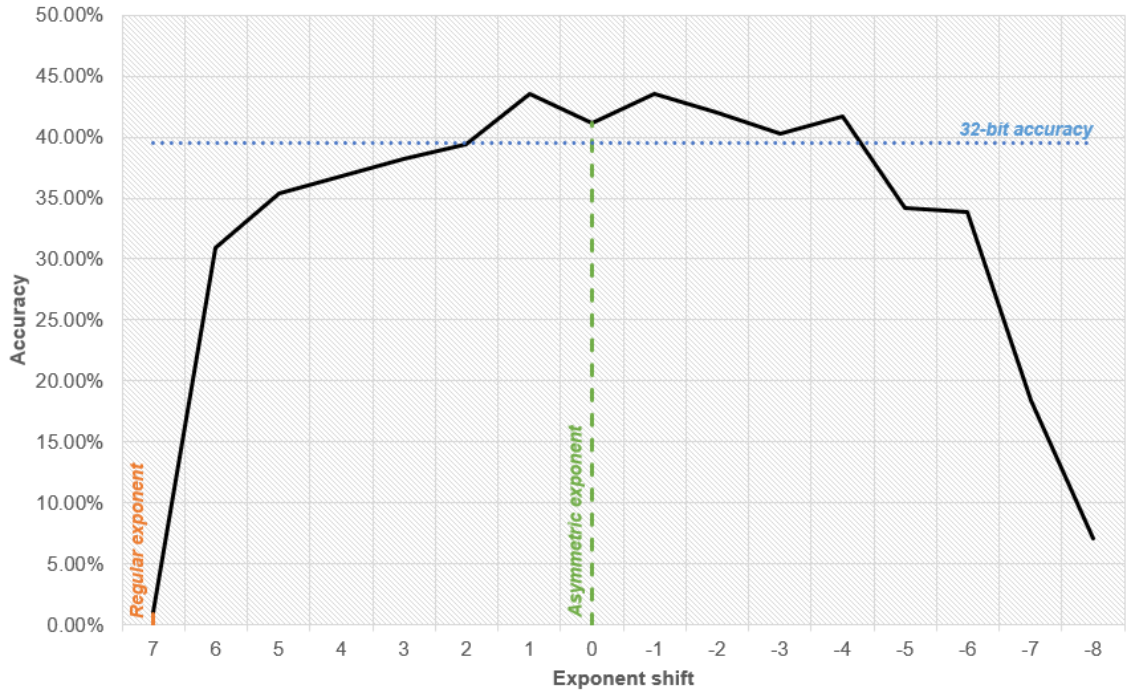


Fig. 4.20. Accuracy of 8-bit ResNet-18 (CIFAR100) across various exponent shift scenarios

The presented results show that an asymmetric exponent with offset is a vital technique for improving accuracy of CNN training with limited precision. Its influence is especially noticeable in the case of more complex architecture and datasets where requirements regarding exponent range are more demanding.

4.3.2 Denormalization

Although stochastic rounding and asymmetric exponent representation can be stated as the most impactful features of the presented method, the influence of IEEE-754-like denormalization mechanism cannot be overlooked. In order to provide 32-bit baseline accuracy on 8-bit floating-point, an additional values range is crucial, especially for more complex NN architectures. The extended margin of supported values provided by the denormalization feature enables the proposed method to limit possible decrease of accuracy on a low-precision floating-point type. The influence of denormalization feature on the presented experiments can be assessed based on Table 4.5.

Table 4.5. Comparison of experiment results with and without denormalization feature

Neural network	**Dataset**	**Accuracy without denormalization**	**Accuracy with denormalization**	**Improvement (percentage points)**
LeNet	MNIST	95.59%	95.98%	0.39
AlexNet	CIFAR10	73.79%	74.50%	0.71
	CIFAR100	29.28%	38.69%	9.41
ResNet	CIFAR10	67.07%	76.01%	8.94
	CIFAR100	40.41%	40.21%	-0.2

It can be observed that in almost all training scenarios the denormalization feature provides improvement when it comes to the final accuracy of the network. The difference is especially vivid in the case of more complex networks. The highest improvement is visible for AlexNet on CIFAR10 which achieves over 9 percentage point better accuracy. Similar scenarios can be observed for the ResNet network on CIFAR10 and with almost 9 percentage points of improvement. The smallest difference is reported for the less complex training scenarios as LeNet and AlexNet on CIFAR10. Interestingly, in the case of ResNet-18 and CIFAR100 the denormalization does not provide any improvement to the accuracy result.

4.3.3 Approach to activations

As described in section 4.2 the approach applied to activations limitation was different from the one used for weights, biases and gradients. In case of activations both asymmetric exponent and asymmetric offset features were disabled. This decision was dictated by a different

range of exponent values utilized specifically by activations based on previously conducted 32-bit trainings for all tested NN architectures. In order to confirm this decision, the author conducted trainings with and without exponent-related features as presented in Table 4.6.

Table 4.6. Asymmetric exponent influence on activations

Neural network	Dataset	Activations with asymmetric exponent	Activation without asymmetric exponent
LeNet	MNIST	19.50%	95.98%
AlexNet	CIFAR10	15.42%	74.50%
	CIFAR100	1.75%	38.69%
ResNet	CIFAR10	10.02%	76.01%
	CIFAR100	1.00%	40.21%

The presented results show that using the asymmetric exponent feature for activations significantly degrades training accuracy of each of the presented architectures. The range of values used by activation parameters includes numbers with positive values of exponent which are not represented by variables with asymmetric exponent. In order to apply the asymmetric exponent in this scenario, additional experiments would be required with exponent offset that includes most commonly used positive exponent values for a particular neural network topology and dataset.

The analysis presented in this chapter shows that none of the method's features from the proposed method can be used as a standalone technique when it comes to training NN with limited precision. Only a precise combination of these features with appropriate tuning allows to achieve the 32-bit baseline results on a limited number of bits.

4.4 Results convergence

The number of epochs required to train a specific NN architecture is crucial when it comes to resource consumption. The more time is required, the longer is the utilization of a given computational unit. Moreover, additional memory may be needed to store logs or temporary epoch results for the final accuracy selection. Due to these concerns, it is crucial that the proposed limitation method does not negatively affect the time of the training convergence. In order to verify this scenario and confirm that there is no such negative impact on a presented method, a comparison of IEEE-754 32-bit floating-point and the proposed 8-bit floating-point training results has been conducted for each NN architecture investigated in this dissertation. The verification of the results' convergence of networks such as LeNet and AlexNet has been checked over 30 epochs. In case of ResNet-18, to ensure that there is enough time for the network to converge, the training length has been extended to 200 epochs.

Fig. 4.21 shows a comparison of LeNet training accuracy on 32-bit IEEE-754 floating-point and the proposed 8-bit floating-point over 30 epochs. It can be observed that starting from

seventh epoch both accuracy lines tend to follow the same path with minor deviations of no more than 0.2 of a percentage point, giving consistent results above 98% for both trainings. An interesting behavior can be noticed during initial epochs of the training where the proposed 8-bit floating-point type provides slightly better results over 6 epochs. This observation is consistent with the assumption of Yu et al. (2022) [142] where limited precision tends to provide a regularization mechanism and faster convergence similarly to manipulation of learning rate hyper-parameter.

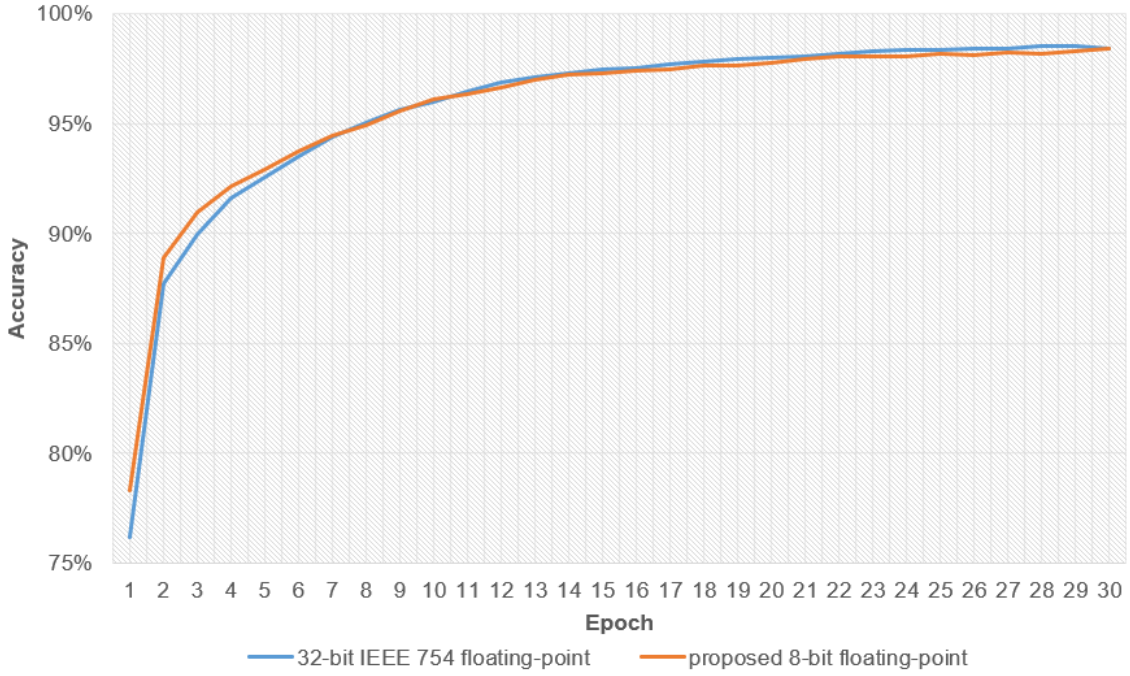


Fig. 4.21. Comparison of LeNet (MNIST) 32-bit IEEE-754 and proposed 8-bit floating point trainings convergence

The comparison of training results between 8-bit and 32-bit floating-point AlexNet architectures on CIFAR10 is presented in Fig. 4.22. Although the 8-bit validation accuracy of the training closely follows the baseline across all 30 epochs, there is a noticeable degradation of results visible across the whole chart, giving 1 percentage point of difference in the results for the 30th epoch. An additional tuning or more training epochs may be required in order to minimize this phenomenon. Nevertheless, it is visible that both convergence trends follow the same path without significant irregularities. It is important to remark that the achieved results for both 32-bit and 8-bit scenarios exceed the previously presented results [154], most likely due to favorable random weights initialization.

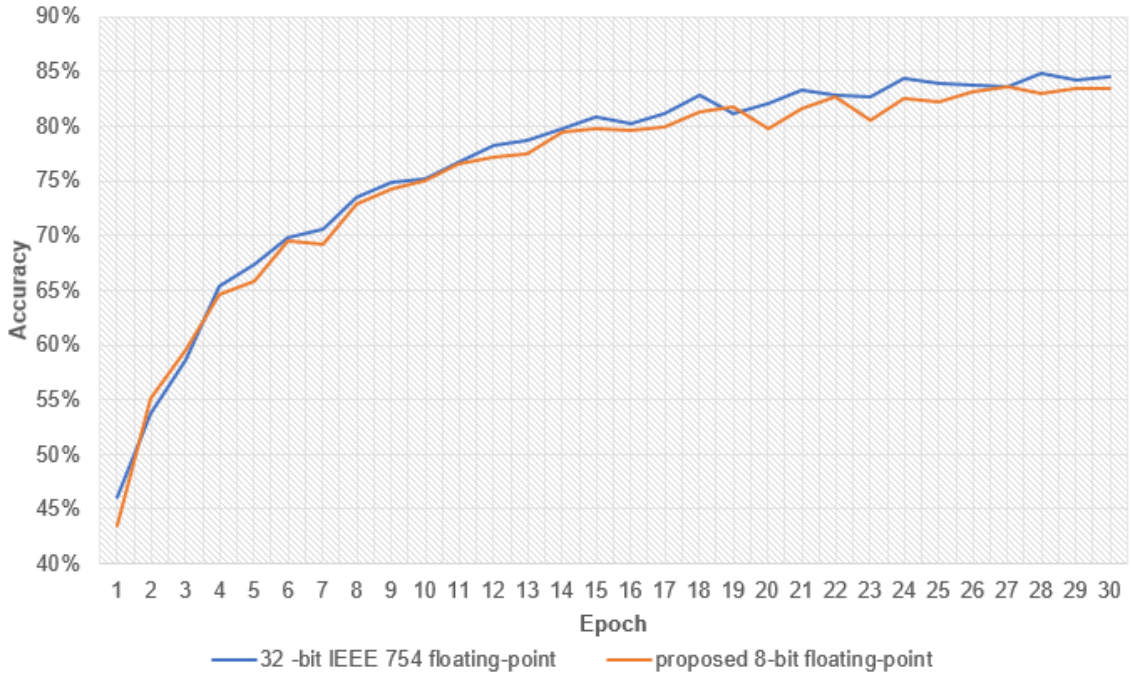


Fig. 4.22. Comparison of AlexNet (CIFAR10) 32-bit IEEE-754 and proposed 8-bit floating point trainings convergence

Much better results can be observed for the same topology trained with the CIFAR100 database (Fig. 4.23). From the initial epochs, both 8-bit and 32-bit accuracies follow almost identical convergence paths. The small accuracy differences can be spotted for epochs between 11 and 22. Although final epochs of the training show a more significant difference between the baseline, the end result during the 30th epoch gives a better accuracy in the case of 8-bit training with around 0.5 percentage point of advantage.

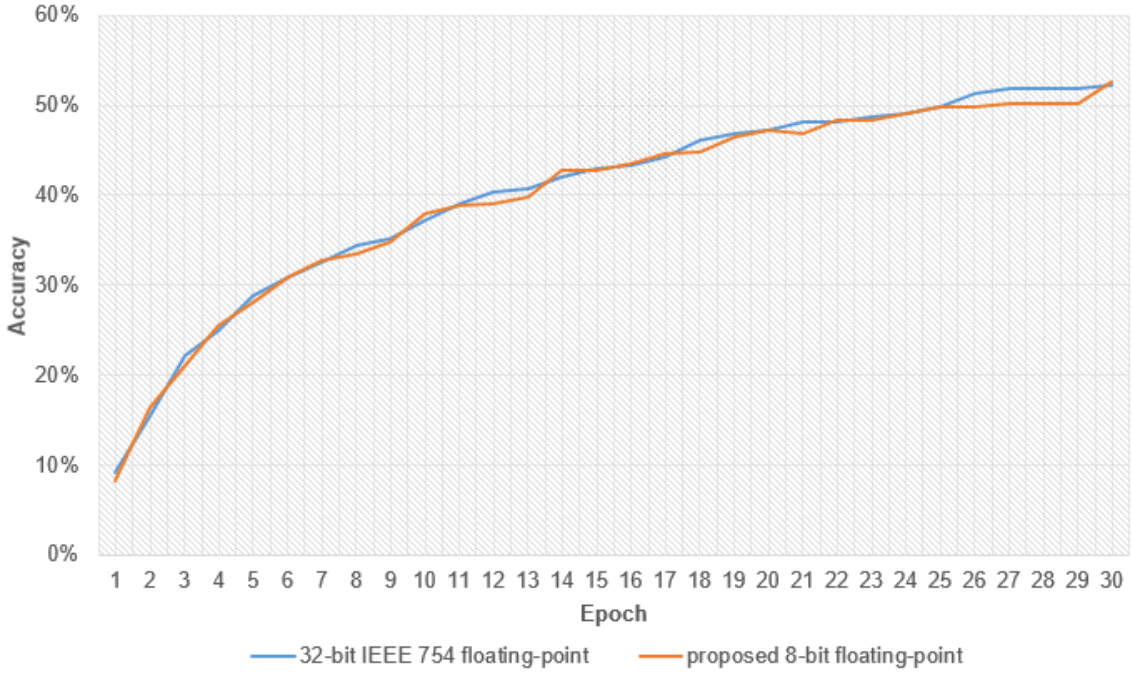


Fig. 4.23. Comparison of AlexNet (CIFAR100) 32-bit IEEE-754 and proposed 8-bit floating point trainings convergence

Fig. 4.24 presents validation accuracy for each of 200 training epochs with ResNet-18 on CIFAR10 dataset. In case of this topology, the number of epochs has been increased for better presentation of the convergence process. It can be observed that accuracy achieved on the proposed 8-bit floating-point follows the 32-bit IEEE-754 baseline, however, the same as with previous charts related to ResNet-18, the accuracy lines behave irregularly during initial epochs. The steep changes in accuracy can be attributed to learning rate scheduler's milestones which were set to 60, 120 and 160 epochs. The chart shows smooth results starting from the 120th epoch once the second scheduler milestone is hit. From this point the baseline slightly exceeds the proposed limitation method result till the end of the training with a difference below 0.5 of percentage point.

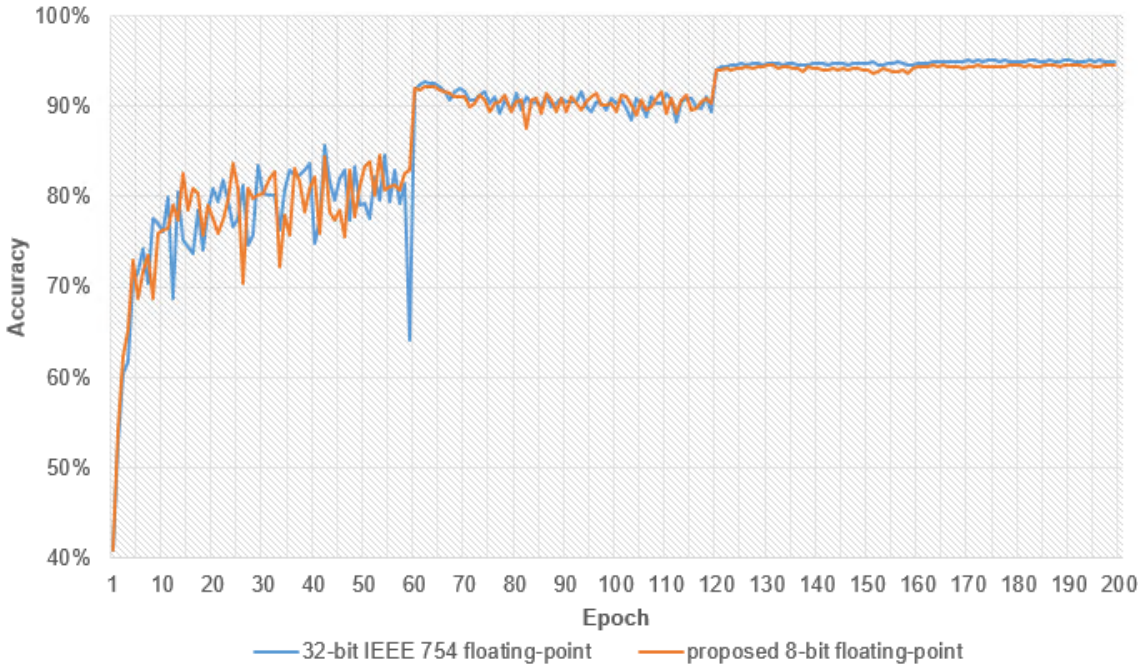


Fig. 4.24. Comparison of ResNet-18 (CIFAR10) 32-bit IEEE-754 and proposed 8-bit floating point trainings convergence

The same situation can be observed while feeding the ResNet-18 network with a bit more complex CIFAR100 dataset (Fig. 4.25). Similar irregularity for initial epochs can be observed along with steep changes due to learning rate scheduler milestones. Although accuracy achieved with limited precision closely follows the baseline, it can be noticed that a small degradation of results is visible during the last 40 epochs. In order to resolve this issue, an additional training tuning may be required.

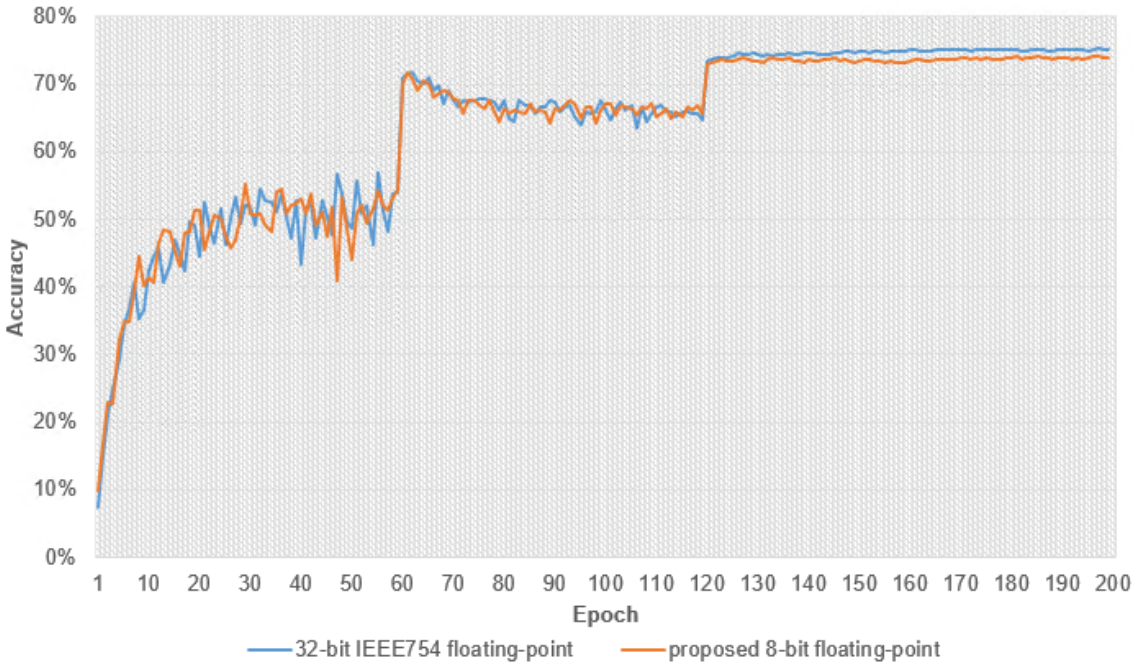


Fig. 4.25. Comparison of ResNet-18 (CIFAR100) 32-bit IEEE-754 and proposed 8-bit floating point trainings convergence

As already discussed in this dissertation, preparation of an optimal machine learning solution to a particular problem is a repetitive task which often includes multiple iterations. Moreover, NN solutions highly depend on the provided training and test datasets. Extensive hyper-parameterization stands for another factor which is often investigated with multi-training procedures as grid search or custom tuning [30]. The provided training convergence examples for LeNet, AlexNet and ResNet-18 show that there is no negative impact of the presented method across multiple neural topologies and datasets. Achieving comparable results for a similar number of epochs confirms that bit count limitation of network architecture's parameters can be done without prolongation of the training time which would hinder the possible power and memory savings.

5 SUMMARY

The dissertation broadly presents results of the performed investigation of NN training with limited precision variables. The author proposed a new precision limitation method for neural network training with an extensive verification on commonly used benchmark convolutional models. Based on a broad literature review, the presented method has been compared to existing techniques of limitation, mixed-precision and hardware approaches.

The presented research can be divided into two general sections. The first one covers experiments on commonly used convolutional benchmark NNs in order to investigate the impact of the proposed limitation of the IEEE-754 32-bit floating-point bit count on their classification accuracy. A wide comparison of bit count variants is validated with detailed description of the gathered results for each of the topologies and the datasets used. Moreover, a deep dive insight on exponent utilization during NN training is featured, showing a suboptimal exponent's bit count usage during training of the selected neural models. Such an observation not only showed a room for data type related improvements when it comes to NN training but also provided a brief introduction to the efficiency related features established by a presented, proprietary training method. The second section of the dissertation was focused on explanation of the proposed NN training method with limited precision. Besides the method itself, its components are described with a relation to its applicability to ML experiments. Analogously to the previous research section, a wide range of bit count variants has been tested in order to validate the method's influence on selected CNNs. The gathered results have been presented to the reader to allow an easy comparison of enhancements provided by the proposed technique.

The conducted experiments allowed the author to validate the theses introduced at the beginning of this dissertation. Firstly, multiple trainings with the use of the proposed method confirmed that **it is possible to train popular convolutional neural networks as LeNet, AlexNet, and ResNet-18 with a custom 8-bit floating-point variable's type without significant classification accuracy degradation in comparison to regular IEEE-754 32-bit floating-point**. This statement has been directly proven by NN training results, especially ones with variables containing 1-bit sign, 4-bit exponent and 3-bit mantissa. In addition to data type changes, a stochastic rounding technique and denormalization have been introduced for achieving the presented results. Secondly, the author was able to indirectly prove the thesis that **application of the proposed arithmetic precision limitation method for convolutional neural networks training with low level bit count floating-point variables allows to decrease computational power and memory requirements.** The relation between lowering bit-count and resource requirements has been showcased based on multiple domain related research and hardware designs. Hence, limiting data type bit count required for NN training would greatly lower resource consumption during that process. Additionally, the method has not increased the number of epochs required for convergence to the results similar to full precision training which could spoil the benefits of the proposed optimization. Finally, the stochastic rounding technique leveraged for the training limitation does not enforce a significant computational overhead to the presented

method. This rounding technique has been frequently used by other researchers and industry leaders providing specific AI hardware accelerators adjusted for stochastic rounding in order to efficiently manage limited resources.

The dissertation presents original achievements of the author in terms of presented method and conducted experiments:

- In-depth analysis of exponent values utilization during convolutional neural network training.
- Investigation of bit count limitation influence to convolutional neural network accuracy.
- Proposition of a new low-precision floating-point format with multiple variants of exponent and mantissa configurations, including a custom approach to exponent range representation.
- Proposition of an original method focusing on low-precision floating-point arithmetic for neural network training combining techniques such as asymmetric exponent, stochastic rounding and denormalization of low-precision variables.
- Extensive experiments on the proposed method impact on selected neural network architectures training accuracy, proving the method's achievements.
- Experiments on asymmetric exponent and its possible offset variants.
- Verification of the convergence between regular 32-bit floating-point training and the proposed 8-bit floating-point technique.

Although the presented research was highly focused on 8-bit floating-point utilization for neural network training, it is important to remark that the method introduces much wider possibilities for future optimizations. As stated in the experiments' overview, it is important to treat the provided mechanism as a form of training hyper-parameterization including the selected bit count and format of the exponent characteristic chosen in this process. Focusing on more directly shaped appliance of the proposed method to a specific topology or training data may greatly influence further outcomes of the proposed method.

5.1 Future directions

The mentioned flexibility of the proposed method introduces a variety of directions for its further development. One of the factors that should be investigated is a much broader application of the mixed precision approach to NN training. At this point, only a single data type has been used for the whole training process of a selected NN training with an exception of activations. Based on the insights of other researchers publishing their results, it may be beneficial to modify the precision along the training process. Three aspects are considered by the author of the dissertation:

- Increasing the bit count and exponent's capacity along with following epochs of the training process. This should allow for NN regularization during the initial epochs. Finding a closest training loss minimum at the early training stage may accelerate the search for the optimal solution.

- Further analysis of neural network training requirements for specific data type formats across single layers of the topology. This way a specific per layer approach can be applied to a given NN model by selecting bit count or floating-point format.
- Applying varying bit counts to specific topology parameters as weights, biases or activations based on their utilization in a specific NN topology. Such an approach might be especially beneficial for activations and biases which, as shown in the dissertation, tend to use a bit different range of exponent values in a floating-point type.

Naturally, along with experimentation progress the proposed variants could be mixed and matched in a form of grid-search-like approach to achieve the best method's variant. Nevertheless, the author does not state that those are the only possible customization that could additionally enhance the proposed training's limitation technique.

Another crucial direction of the presented research is the preparation of a custom hardware design with full support of the presented method's features. In contrast to software simulation techniques, a hardware acceleration would allow for much better power and memory savings measurements during NN trainings. Based on that, it would be also possible to approximate costs and requirements for future productization of such NN training accelerators. Moreover, efficient hardware design opens new ways for power effective implementation of stochastic rounding or the application of mixed-precision. The initial step in this direction has been already made by research done by Aleksiuk et al. (2023) [20]. The work involved the design and implementation of the FPGA based 8-bit floating-point multiplier proposed in this dissertation, required for the support of the presented method.

Usage of full software simulation for utilization and verification of the presented method significantly limits the validation scope that can be executed on broadly available common use hardware. The emulation of limited bit count and additional rounding introduce additional operations during the training which is both time and resource consuming. Once custom hardware is available, validation of the proposed method on much larger NN topologies and datasets would be more approachable. This direction should also include training and verification of other network topologies such as recurrent networks or transformers.

6 REFERENCES

[1] Y. Duan, J. S. Edwards and Y. K. Dwivedi, "Artificial intelligence for decision making in the era of Big Data–evolution, challenges and research agenda," *International journal of information management,* vol. 48, pp. 63-71, 2019.

[2] R. M. Amir, A. Elham, A. Saqib, M. Mokhtar, A. H. Omed, G. Y. Marwan, A. H. Sarkar and H. Mehdi, "Artificial intelligence approaches and mechanisms for big data analytics: a systematic study.," *PeerJ Computer Science,* no. 7, p. e488, 2021.

[3] S. Bianco, R. Cadene, L. Celona and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE access,* vol. 6, pp. 64270-64277, 2018.

[4] Y. Hu, Y. Liu and Z. Liu, "A survey on convolutional neural network accelerators: GPU, FPGA and ASIC.," in *2022 14th International Conference on Computer Research and Development (ICCRD)*, 2022.

[5] S. Gupta, A. Agrawal, K. Gopalakrishnan and P. Narayanan, "Deep Learning with Limited Numerical Precision," in *International conference on machine learning*, 2015.

[6] M. Ortiz, A. Cristal, E. Ayguadé and M. Casas, "Low-Precision Floating-Point Schemes for Neural Network Training," *arXiv:180405267,* 2018.

[7] I. Taras and D. M. Stuart, "Quantization Error as a Metric for Dynamic Precision Scaling in Neural Net Training," *arXiv:180108621,* 2018.

[8] J. Park, S. Lee and D. Jeon, "A neural network training processor with 8-bit shared exponent bias floating point and multiple-way fused multiply-add trees," *IEEE Journal of Solid-State Circuits,* vol. 57, no. 3, pp. 965-977, 2021.

[9] R. Mishra, H. P. Gupta and T. Dutta, "A survey on deep neural network compression: Challenges, overview, and solutions," *arXiv:2010.03954,* 2020.

[10] "Cloud TPU," Google, 2023. [Online]. Available: https://cloud.google.com/tpu. [Accessed 12 March 2023].

[11] "Deep Learning Accelerator," Nvidia, 2023. [Online]. Available: https://developer.nvidia.com/deep-learning-accelerator. [Accessed 12 March 2023].

[12] Y. Chen, Y. Xie, L. Song, F. Chen and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering,* vol. 6, no. 3, pp. 264-274, 2020.

[13] X. Zhang, S. Liu, R. Zhang, C. Liu, D. Huang, S. Zhou, J. Guo, Q. Guo, Z. Du and T. Zhi, "Fixed-point back-propagation training," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020.

[14] H. Park, J. H. Lee, Y. Oh, S. Ha and S. Lee, "Training Deep Neural Network in Limited Precision," *arXiv:1810.05486,* 2018.

[15] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim and H.-J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," 2018.

[16] K. Onishi and M. Hashimoto, "Memory Efficient Training using Lookup-Table-based Quantization for Neural Network," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020.

[17] B. Kim, S. H. Lee, H. Kim, D.-T. Nguyen, M.-S. Le, I. J. Chang, D. Kwon, J. H. Yoo, J. W. Choi and H.-J. Lee, "PCM: precision-controlled memory system for energy efficient deep neural network training," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[18] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers,* no. 1, pp. 14-17, 1964.

[19] T. Ryszard and M. Szaleniec, Leksykon sieci neuronowych, Projekt Nauka. Fundacja na rzecz promocji nauki polskiej, 2015.

[20] H. Aleksiuk, A. Bogucki and K. Repiński, "8-bitowy mnożnik i sumator zmiennoprzecinkowy, Projekt grupowy pod opieką prof. Marka Bloka," Politechnika Gdańska, Gdańsk, 2023. Available at: https://git.pg.edu.pl/5-ksti-2023

[21] L. Yu, S. Wang and K. K. Lai, "Data preparation in neural network data analysis," *Foreign-Exchange-Rate Forecasting With Artificial Neural Networks,* pp. 36-62, 2007.

[22] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *The Journal of Machine Learning Research,* vol. 21, no. 1, pp. 10039-10081, 2020.

[23] T. Fredriksson, D. I. Mattos, J. Bosch and H. H. Olsson, "Data Labeling: An Empirical Investigation into Industrial Challenges and Mitigation Strategies," in *International Conference on Product-Focused Software Process Improvement*, 2020.

[24] Y. Roh, G. Heo and S. E. Whang, "A survey on data collection for machine learning: a big data-ai integration perspective," *IEEE Transactions on Knowledge and Data Engineering,* vol. 33, no. 4, pp. 1328-1347, 2019.

[25] K. Y. Chan, B. Abu-Salih, R. Qaddoura, A. M. Al-Zoubi, V. Palade, D.-S. Pham, J. Del Ser and K. Muhammad, "Deep neural networks in the cloud: Review, applications, challenges and research directions," *Neurocomputing,* p. 126327, 2023.

[26] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj and T. Wang, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems,* vol. 3, pp. 800-811, 2021.

[27] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi and J. Kepner, "AI and ML Accelerator Survey and Trends," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, 2022.

[28] G. Alexandru-Lucian, P. Alessandro, C. Horia and B. Michaela, "Performance vs. hardware requirements in state-of-the-art automatic speech recognition," *EURASIP Journal on Audio, Speech, and Music Processing,* vol. 2021, no. 1, 2021.

[29] S. Han, H. Mao and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149,* 2015.

[30] X. Zhang, X. Chen, L. Yao, C. Ge and M. Dong, "Deep Neural Network Hyperparameter Optimization with Orthogonal Array Tuning," in *Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part IV 26*, 2019.

[31] J. Lee, N. C. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik and M. Grundmann, "On-Device Neural Net Inference with Mobile GPUs," *arXiv:1907.01989,* 2019.

[32] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv:1610.05492,* 2016.

[33] S. A. Osia, A. S. Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H. R. Rabiee, N. D. Lane and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet of Things Journal,* vol. 7, no. 5, pp. 4505-4518, 2020.

[34] M. C. Nwadiugwu, "Neural networks, artificial intelligence and the computational brain," *arXiv:2101.08635,* 2020.

[35] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review,* vol. 65, no. 6, p. 386, 1958.

[36] M. Pietrołaj and M. Blok, "Neural network training with limited precision and asymmetric exponent," *Journal of Big Data,* vol. 9, no. 1, p. 63, 2022.

[37] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *nature,* vol. 323, no. 6088, pp. 533-536, 1986.

[38] R. Hecht-Nielsen, "Theory of the backpropagation neural network," *Neural networks for perception,* pp. 65-93, 1992.

[39] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison and G. E. Dahl, "On Empirical Comparisons of Optimizers for Deep Learning," *arXiv:1910.05446,* 2020.

[40] V. Sze, Y.-H. Chen, T.-J. Yang and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE,* vol. 105, no. 12, pp. 2295-2329, 2017.

[41] B. Hanin, "Which neural net architectures give rise to exploding and vanishing gradients?," *Advances in neural information processing systems,* vol. 31, 2018.

[42] L. Xia, M. E. Hochstenbach and S. Massei, "On the Convergence of the Gradient Descent Method with Stochastic Fixed-point Rounding Errors under the Polyak-Lojasiewicz Inequality," *arXiv:2301.09511,* 2023.

[43] F. Rosenblatt, Principles of neurodynamics. perceptrons and the theory of brain mechanisms, vol. 55, Spartan books Washington, DC, 1962.

[44] H. Schulz and S. Behnke, "Deep learning: Layer-wise learning of feature hierarchies," *KI-Künstliche Intelligenz,* vol. 26, pp. 357-363, 2012.

[45] A. Canziani, A. Paszke and E. Culurciello, "An analysis of deep neural network models for practical applications," *arXiv:1605.07678,* 2016.

[46] A. LeNail, "NN-SVG: Publication-Ready Neural Network Architecture Schematics," *J. Open Source Softw.,* vol. 4, no. 33, p. 747, 2019.

[47] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *nature,* vol. 521, no. 7553, pp. 436-444, 2015.

[48] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural,* vol. 25, 2012.

[49] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang and S. Wang, "Image and Video Compression With Neural Networks: A Review," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 30, no. 6, pp. 1683-1698, 2019.

[50] B. Alshemali and K. Jugal, "Improving the Reliability of Deep Neural Networks in NLP: A Review," *Knowledge-Based Systems,* no. 191, p. 105210, 2020.

[51] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv:1511.08458,* 2015.

[52] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE transactions on neural networks and learning systems,* 2021.

[53] H. Salehinejad, S. Sankar, J. Barfett, E. Colak and S. Valaee, "Recent advances in recurrent neural networks," *arXiv:1801.01078,* 2017.

[54] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems,* vol. 2, 1989.

[55] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[56] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, Karpathy, rej, A. Khosla and M. Bernstein, "Imagenet large scale visual recognition challenge," *International journal of computer vision,* vol. 115, pp. 211-252, 2015.

[57] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556,* 2014.

[58] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper With Convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[59] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010.

[60] H. H. Tan and K. H. Lim, "Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization," in *2019 7th international conference on smart computing \& communications (ICSCC)*, 2019.

[61] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[62] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1550-1560, 1990.

[63] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM - a tutorial into long short-term memory recurrent neural networks," *arXiv:1909.09586,* 2019.

[64] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[65] Y. Yu, X. Si, C. Hu and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural computation,* vol. 31, no. 7, pp. 1235-1270, 2019.

[66] K. Cho, B. Van Merriënboer, D. Bahdanau and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259,* 2014.

[67] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.

[68] Y. LeCun, "1.1 deep learning hardware: past, present, and future," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2019.

[69] W. Kahan, "IEEE standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE,* vol. 754, no. 94720-1776, p. 11, 1996.

[70] E. Strubell, A. Ganesh and A. McCallum, "Energy and policy considerations for deep learning in NLP," *arXiv:1906.02243,* 2019.

[71] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing,* vol. 16, no. 3, pp. 82-88, 2017.

[72] M. J. Flynn and S. F. Oberman, "Advanced Computer Arithmetic Design," 2001.

[73] E. E. Swartzlander and C. E. Lemonds, Computer Arithmetic: Volume III, World Scientific, 2015.

[74] E. L. Oberstar, "Fixed-point representation & fractional math," *Oberstar Consulting,* vol. 9, no. 19, 2007.

[75] B. Barrois and O. Sentieys, "Customizing fixed-point and floating-point arithmetic—a case study in k-means clustering," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2017.

[76] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM computing surveys,* vol. 23, no. 1, pp. 5-48, 1991.

[77] Y. Zhang, L. Zhao, S. Cao, W. Wang, T. Cao, F. Yang, M. Yang, S. Zhang and N. Xu, "Integer or Floating Point? New Outlooks for Low-Bit Quantization on Large Language Models," *arXiv:2305.12356,* 2023.

[78] "IEEE Standard for Floating-Point Arithmetic," *In IEEE Std 754-2019 (Revision of IEEE 754-2008),* pp. 1-84, 2019.

[79] D. G. Hough, "The IEEE standard 754: One for the history books," *Computer,* vol. 52, no. 12, pp. 109-112, 2019.

[80] M. P. Connolly, N. J. Higham and T. Mary, "Stochastic rounding and its probabilistic backward error analysis," *SIAM Journal on Scientific Computing,* vol. 43, no. 1, pp. A566-A585, 2021.

[81] G. Frantz and R. Simar, "Comparing fixed-and floating-point DSPs," *Texas Instruments, Dallas, TX, USA,* 2004.

[82] C. Inacio and D. Ombres, "The DSP decision: Fixed point or floating?," *IEEE Spectrum,* vol. 33, no. 9, pp. 72-74, 1996.

[83] "TensorFlow," 2023. [Online]. Available: https://www.tensorflow.org/. [Accessed 21 January 2023].

[84] "PyTorch," 2023. [Online]. Available: https://pytorch.org/. [Accessed 21 January 2023].

[85] C. Luo, X. He, J. Zhan, L. Wang, W. Gao and J. Dai, "Comparison and benchmarking of ai models and frameworks on mobile devices," *arXiv:2005.05085,* 2020.

[86] J. Y. F. Tong, D. Nagle and R. A. Rutenbar, "Reducing power by optimizing the necessary precision/range of floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 8, no. 3, pp. 273-286, 2000.

[87] M. C. Pascal, R. A. Rutenbar and R. L. Carley, "Exploring multiplier architecture and layout for low power," in *Proceedings of Custom Integrated Circuits Conference*, 1996.

[88] T. K. Callaway and E. E. Swartzlander, "Power-delay characteristics of CMOS multipliers," in *Proceedings 13th IEEE Sympsoium on Computer Arithmetic*, 1997.

[89] H. Choi, W. Burleson and D. Phatak, "Fixed-point roundoff error analysis of large feedforward neural networks," 1993.

[90] K. Jia and M. Rinard, "Exploiting verified neural networks via floating point numerical error," in *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings 28*, 2021.

[91] G. E. Forsythe, "Round-off errors in numerical integration on automatic machinery-preliminary report," *In Bulletin of the American Mathematical Society,* vol. 56, pp. 61-62, 1950.

[92] M. Croci, M. Fasi, N. J. Higham, T. Mary and M. Mikaitis, "Stochastic rounding: implementation, error analysis and applications," *Royal Society Open Science,* vol. 9, no. 3, p. 211631, 2022.

[93] C. M. Barnes R., E. H. Cooke-Yarborough and G. A. Thomas D, "An electronic digital computor using cold cathode counting tubes for storage," *Electron,* vol. 23, pp. 286-291, 1951.

[94] M. Mikaitis, "Stochastic rounding: algorithms and hardware accelerator," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021.

[95] S. Felix, M. Gore and A. G. Alexander, "Converting floating point numbers to reduce the precision". Washington, DC: U.S. Patent 11,169,778, 9 November 2021.

[96] J. D. Bradbury, S. R. Carlough, B. R. Prasky and E. M. Schwarz, “Stochastic rounding floating-point multiply instruction using entropy from a register”. Washington, DC: U.S. Patent 10,445,066, 15 October 2019.

[97] G. H. Loh, "Stochastic rounding logic". Washington, DC: U.S. Patent 10,628,124, 18 March 2019.

[98] J. M. Alben, P. Micikevicius, H. Wu and M. Y. Siu, "Stochastic Rounding of Numerical Values". Washington, DC: U.S. Patent 10,684,824, 12 12 2019.

[99] D. Amodei and D. Hernandez, "AI and compute," OpenAI, 16 May 2018. [Online]. Available: https://openai.com/blog/ai-and-compute/. [Accessed 8 April 2023].

[100] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup and M. Shah, "A survey of on-device machine learning An algorithms and learning theory perspective," *ACM Transactions on Internet of Things,* vol. 2, no. 3, pp. 1-49, 2021.

[101] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle and J. Guttag, "What is the state of neural network pruning?," in *Proceedings of machine learning and systems*, 2020.

[102] A. Renda, J. Frankle and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," *arXiv:2003.02389,* 2020.

[103] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv:1808.04752,* 2018.

[104] N. Zmora, W. Hao and J. Rodge, "Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT," Nvidia, 20 July 2021. [Online]. Available: https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/. [Accessed 15 June 2023].

[105] H. Qin, R. Gong, X. Liu, X. Bai, J. Song and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition,* vol. 105, p. 107281, 2020.

[106] K. Ullrich, E. Meeds and M. Welling, "Soft weight-sharing for neural network compression," *arXiv:1702.04008,* 2017.

[107] R. Schwartz, J. Dodge, N. A. Smith and O. Etzioni, "Green AI," *Communications of the ACM,* vol. 63, no. 12, pp. 54-63, 2020.

[108] T. Furlanello, Z. Lipton, M. Tschannen, L. Itti and A. Anandkumar, "Born again neural networks," in *International Conference on Machine Learning*, 2018.

[109] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE transactions on pattern analysis and machine intelligence,* vol. 42, no. 3, pp. 568-579, 2018.

[110] H. Esmaeilzadeh, A. Sampson, L. Ceze and D. Burger, "Neural acceleration for general-purpose approximate programs," in *2012 45th annual IEEE/ACM international symposium on microarchitecture*, 2012.

[111] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell and Q. Wu, "RENO: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015.

[112] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News,* vol. 42, no. 1, pp. 269-284, 2014.

[113] Y. E. Wang, G.-Y. Wei and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for deep learning," *arXiv:1907.10701,* 2019.

[114] L. Baischer, M. Wess and N. TaheriNejad, "Learning on hardware: A tutorial on neural network accelerators and co-processors," *arXiv:2104.09252,* 2021.

[115] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015.

[116] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE transactions on computer-aided design of integrated circuits and systems,* vol. 37, no. 1, pp. 35-47, 2017.

[117] D. T. Nguyen, T. N. Nguyen, H. Kim and H.-J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 27, no. 8, pp. 1861-1873, 2019.

[118] J. Yao, S. Zhang, Y. Yao, F. Wang, J. Ma, J. Zhang, Y. Chu, L. Ji, K. Jia and T. Shen, "Edge-cloud polarization and collaboration: A comprehensive survey for ai," *IEEE Transactions on Knowledge and Data Engineering,* vol. 35, no. 7, pp. 6866-6886, 2022.

[119] N. C. Thompson, K. Greenewald, K. Lee and G. F. Manso, "The computational limits of deep learning," *arXiv:2007.05558,* 2020.

[120] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv:2003.05689,* 2020.

[121] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE,* vol. 109, no. 1, pp. 43-76, 2020.

[122] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev and G. Venkatesh, "Mixed precision training," *arXiv:1710.03740,* 2017.

[123] "Apex (A PyTorch Extension)," Nvidia, 2018. [Online]. Available: https://nvidia.github.io/apex/. [Accessed 17 July 2023].

[124] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics,* vol. 1, no. 4, pp. 216-222, 2018.

[125] "XLA: Optimizing Compiler for Machine Learning," Google, 30 March 2023. [Online]. Available: https://www.tensorflow.org/xla. [Accessed 19 August 2023].

[126] "NVIDIA H100 Tensor Core GPU Architecture," Nvidia, 2023. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core. [Accessed 19 August 2023].

[127] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini and Y. Wu, "Coca: Contrastive captioners are image-text foundation models," *arXiv:2205.01917,* 2022.

[128] "ChatGPT: Optimizing Language Models for Dialogue," OpenAI, 30 11 2022. [Online]. Available: https://openai.com/blog/chatgpt/. [Accessed 10 February 2023].

[129] "OpenAI's ChatGPT Reportedly Costs $100,000 a Day to Run," CIOCoverage, 2023. [Online]. Available: https://www.ciocoverage.com/openais-chatgpt-reportedly-costs-100000-a-day-to-run/. [Accessed 10 February 2023].

[130] J. L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach, Elsevier, 2011.

[131] N. Thompson, "The economic impact of moore's law: Evidence from when it faltered," *SSRN Electronic Journal,* 2017.

[132] N. C. Thompson, S. Ge and G. F. Manso, "The importance of (exponentially more) computing power," *arXiv:2206.14007,* 2020.

[133] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli and S. Woerner, "The power of quantum neural networks," *Nature Computational Science,* vol. 1, no. 6, pp. 403-409, 2021.

[134] F. Martínez-Plumed, S. Avin, M. Brundage, A. Dafoe, S. Ó. hÉigeartaigh and J. Hernández-Orallo, "Between Progress and Potential Impact of AI: the Neglected Dimensions," *arxXiv preprint,* 2018.

[135] A. Gholami, Z. Yao, S. Kim and M. W. Mahoney, "AI and Memory Wall," Medium, 29 March 2021. [Online]. Available: https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8. [Accessed 16 09 2023].

[136] P. Jain, A. Jain, A. Nrusimha, A. Gholami, P. Abbeel, J. Gonzalez, K. Keutzer and I. Stoica, "Checkmate: Breaking the memory wall with optimal tensor rematerialization," *Proceedings of Machine Learning and Systems,* vol. 2, pp. 497-511, 2020.

[137] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861,* 2017.

[138] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019.

[139] S. A. Budennyy, V. D. Lazarev, N. N. Zakharenko, A. N. Korovin, O. Plosskaya, D. V. Dimitrov, V. Akhripkin, I. Pavlov, I. V. Oseledets and I. S. Barsola, "Eco2AI: carbon emissions tracking of machine learning models as the first step towards sustainable AI," in *Doklady Mathematics*, 2022.

[140] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier and J. Dean, "Carbon emissions and large neural network training," *arXiv:2104.10350,* 2021.

[141] Y. Fu, H. Guo, M. Li, X. Yang, Y. Ding, V. Chandra and Y. Lin, "CPT: Efficient deep neural network training via cyclic precision," *arXiv:2101.09868,* 2021.

[142] Z. Yu, Y. Fu, S. Wu, M. Li, H. You and Y. Lin, "LDP: Learnable Dynamic Precision for Efficient Deep Neural Network Training and Inference," *arXiv:2203.07713,* 2022.

[143] Y. LeCun, C. Cortes and C. J. C. Burges, "The MNIST database of handwritten digits," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed 14 October 2022].

[144] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[145] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[146] H. Fuketa, T. Ikegami, W. Nogami, T. Matsukawa, T. Kudoh and R. Takano, "Image-Classifier Deep Convolutional Neural Network Training by 9-bit Dedicated Hardware to Realize Validation Accuracy and Energy Efficiency Superior to the Half Precision Floating Point Format," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.

[147] T. Na and S. Mukhopadhyay, "Speeding up Convolutional Neural Network Training with Dynamic Precision Scaling and Flexible Multiplier-Accumulator," 2016.

[148] J. Lee, "Energy-efficient deep-neural-network training processor with fine-grained mixed precision," 2020.

[149] J. O. Ríos, A. Armejach, E. Petit, G. Henry and M. Casas, "Dynamically Adapting Floating-Point Precision to Accelerate Deep Neural Network Training," in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2021.

[150] R. Banner, I. Hubara, E. Hoffer and D. Soudry, "Scalable methods for 8-bit training of neural networks," *Advances in neural information processing systems,* vol. 31, 2018.

[151] M. Junaid, S. Arslan, T. Lee and H. Kim, "Optimal Architecture of Floating-Point Arithmetic for Neural Network Training Processors," *Sensors,* vol. 22, no. 3, p. 1230, 2022.

[152] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd and J. Kamalu, "FP8 formats for deep learning," *arXiv:2209.05433,* 2022.

[153] B. Noune, P. Jones, D. Justus, D. Masters and C. Luschi, "8-bit numerical formats for deep neural networks," *arXiv:2206.02915,* 2022.

[154] M. Pietrołaj and M. Blok, "Resource constrained neural network training," *Scientific Reports,* vol. 14, no. 1, pp. 1-13, 2024.
Available at: https://github.com/MariuszPPP/resource_constrained_nn_training

[155] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[156] "Python," Python Software Foundation, 2023. [Online]. Available: https://www.python.org/. [Accessed 12 August 2023].

[157] M. Rüb and A. Sikora, "A Practical View on Training Neural Networks in the Edge," *IFAC-PapersOnLine,* vol. 55, no. 4, pp. 272-279, 2022.

[158] M. Rizwan, "LeNet-5-A Classic CNN Architecture," Data Science Central, 16 October 2018. [Online]. Available: https://www.datasciencecentral.com/lenet-5-a-classic-cnn-architecture/. [Accessed 20 January 2022].

[159] J. Schmidhuber, "The most cited neural networks all build on work done in my labs," *AI Blog,* pp. 4-30, 2022.

[160] L. N. Smith and N. Topin, "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates," in *Artificial intelligence and machine learning for multi-domain operations applications*, 2019.

[161] J. Yun, B. Kang, F. Rameau and Z. Fu, "In Defense of Pure 16-bit Floating-Point Neural Networks," *arXiv:2305.10947,* 2023.

[162] "QPyTorch," 15 March 2023. [Online]. Available: https://github.com/Tiiiger/QPyTorch. [Accessed 24 August 2023].

[163] "CPFloat: Custom-Precision Floating-point numbers," 13 April 2023. [Online]. Available: https://github.com/north-numerical-computing/cpfloat. [Accessed 20 August 2023].

[164] L. Xia, M. Anthonissen, M. Hochstenbach and B. Koren, "A Simple and Efficient Stochastic Rounding Method for Training Neural Networks in Low Precision," *arXiv:2103.13445,* 2021.

7 LIST OF FIGURES

8 LIST OF TABLES