



LEHRGEBIET - TECHNISCHE INFORMATIK

MASTERARBEIT

**Auswirkungen der
Leistungsbegrenzung der CPU auf die
Energieeffizienz und
Leistungsfähigung beim Training
neuronaler Netze**

Betreuerin: Prof. Dr. Lena Oden

Semester: SoSe 2024

Name: Wandke, Keanu

Matr.-Nr.: 4140346

E-Mail: keanu.wandke@gmail.com

Semester: 04

Studiengang: Master Praktische Informatik

Abgabedatum: 16. September 2024

Inhaltsverzeichnis

Selbstständigkeitserklärung	IV
Abkürzungsverzeichnis	V
Abbildungsverzeichnis	VI
Quellcodeverzeichnis	IX
Kurzfassung	X
1 Einleitung	1
2 Grundlagen	3
2.1 Tiefes Lernen	3
2.1.1 Neuron	4
2.1.2 Neuronales Netz	6
2.1.3 Convolutional Neural Networks (CNN)	7
2.2 CNN-Modelle	11
2.2.1 AlexNet	11
2.2.2 VGG16	11
2.2.3 ResNet101	12
2.2.4 DenseNet121	13
2.3 Datensätze	14
2.3.1 CIFAR10	14
2.3.2 Fashion-MNIST	14
2.4 Rolle der CPU und GPU beim tiefen Lernen	15
2.4.1 CPU	15

2.4.2	GPU	17
2.4.3	Zusammenfassung: CPU-GPU	19
2.5	Powermanagement	20
3	Verwandte Arbeiten	22
3.1	GPU DVFS beim tiefen Lernen: Energie und Leistung	22
3.2	Untersuchung der Energie-Leistung-Trade-Offs	22
3.3	Einordnung der verwandten Arbeiten	23
4	Methodik	24
4.1	Experimentelles Setup	24
4.1.1	Eingesetzte Hardware	24
4.1.2	Verwendete Werkzeuge	25
4.1.3	Verwendete Datensätze	28
4.1.4	Verwendete Modelle	28
4.1.5	Aufbau der Modelle/Jupyter-Notebooks	28
4.1.6	Konfiguration der Modelle	37
4.1.7	Anpassungen der Modell-Architekturen	37
4.2	Datenerfassung	40
4.2.1	Verfahren der Messung	40
4.2.2	Konfiguration von HWiNFO	40
4.3	Datenanalyse	43
5	Experimente	45
5.1	Messplan	45
5.2	Versuchsdurchführung	46

6 Ergebnisse	49
6.1 Trainingsdauer	49
6.1.1 Trainingsdauer bei Variation von Frequenz und Spannung . . .	49
6.1.2 Trainingsdauer bei Variation der Spannung	52
6.2 Energieverbrauch	54
6.2.1 Energieverbrauch bei Variation von Frequenz und Spannung . .	55
6.2.2 Energieverbrauch bei Variation der Spannung	57
6.2.3 CPU-Energieverbrauchsanteil	60
6.3 Durchschnittliche Kernfrequenz	61
6.4 Genauigkeit	63
7 Diskussion	64
7.1 Betrachtung der Trainingsdauer	64
7.2 Betrachtung des Energieverbrauchs	66
7.3 Betrachtung der Genauigkeit	70
8 Zusammenfassung und Ausblick	71
Literatur	74
A Anhang	82
A.1 Ergebnisse: Trainingsdauer	82
A.2 Ergebnisse: Energieverbrauch	86
A.3 Ergebnisse: Genauigkeit	90

Selbstständigkeitserklärung

Ich, Keanu Wandke, erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich, inhaltlich oder sinngemäß entnommenen Stellen als solche den wissenschaftlichen Anforderungen entsprechend kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate überprüft und ausschließlich für Prüfungszwecke gespeichert wird.

Hagen im Bremischen, 16.09.2024



Ort, Datum

Unterschrift

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ALU	Arithmetic Logic Unit / Arithmetische Logikeinheit
Cifar	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma-Separated Values
CUDA	Compute Unified Device Architecture
DenseNet	Densely Connected Convolutional Network
DNN	Deep Neural Network / Tiefes Neuronales Netz
DVFS	Dynamische Spannungs- und Frequenzskalierung
FLOPS	Floating Point Operations Per Second
FSB	Front-Side Bus
GPGPU	General-Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
KI	Künstliche Intelligenz
LSVRC-2012	Large Scale Visual Recognition Challenge 2012
MMU	Memory Management Unit / Speicherverwaltungseinheit
RAM	Random Access Memory / Arbeitsspeicher
ReLU	Rectified Linear Unit
ResNet	Residual Network
SSD	Solid State Drive
VGG	Visual Geometry Group
V-RAM	Video Random Access Memory / Videospeicher

Abbildungsverzeichnis

1	Vereinfachte Darstellung eines Neurons in einem neuronalen Netz. [7]	4
2	Detaillierte Darstellung des Neurons. Angelehnt an [11]	5
3	Schematische Darstellung eines neuronalen Netzes. Angelehnt an [11].	6
4	Aufbau/Architektur eines CNN Modells am Beispiel von VGG16 [13].	7
5	Darstellung der Faltung aus den Eingabedaten. Angelehnt an [16]	8
6	Darstellung vom Max- und Average-Pooling. Angelehnt an [18]	9
7	Residual Block. [25]	13
8	Vereinfachter Aufbau eines Dense Convolutional Neural Network. [26].	13
9	Schematische Darstellung des Zusammenhangs zwischen Taktfrequenz, Versorgungsspannung und dem Energieverbrauch. a) Keine Anpassung. b) Verringerung der Taktfrequenz. c) Verringerung der Taktfrequenz und der Versorgungsspannung. [56]	21
10	Allgemeine Sensoreinstellungen von HWiNFO.	41
11	Protokollierungseinstellungen von HWiNFO.	42
12	Auswahl der erweiterten Ansicht in AMD Ryzen Master.	46
13	Auswahl des Tabs "Profile 1" in AMD Ryzen Master.	46
14	Auswahl des Tabs "Profile 1" in AMD Ryzen Master.	47
15	Startmöglichkeiten des Trainings im Jupyter-Notebook	47
16	Trainingsdauer beim AlexNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	49
17	Trainingsdauer beim VGG16 bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	50
18	Laufzeitveränderungen von allen Modellen und Datensätzen bei Variation der Frequenz und Spannung.	51
19	Trainingsdauer beim AlexNet bei Anpassung der CPU-Spannung.	52
20	Trainingsdauer beim VGG16 bei Anpassung der CPU-Spannung.	53

21	Laufzeitveränderungen über alle Modelle und Datensätze bei Variation der Spannung.	54
22	Energieverbrauch beim AlexNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	55
23	Energieverbrauch beim DenseNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	56
24	Energieeinsparung beim Training der verschiedenen Modelle bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	57
25	Energieverbrauch beim AlexNet bei Anpassung der CPU-Spannung. . .	58
26	Energieverbrauch beim DenseNet bei Anpassung der CPU-Spannung. .	58
27	Energieeinsparung beim Training der verschiedenen Modelle bei Anpassung der CPU-Spannung.	59
28	CPU-Energieverbrauchsanteile beim Training der verschiedenen Modelle bei einer Taktfrequenz von 3600 MHz und einer Versorgungsspannung von 1,4 V.	60
29	Durchschnittliche effektive Kerntakt der CPU Kerne bei einer maximalen CPU-Frequenz von 3600 MHz und variabler Spannung.	61
30	Durchschnittliche, maximale und minimale effektive Kernfrequenz der CPU Kerne bei einer maximalen CPU-Frequenz von 3600 MHz und 1,1 V beim ResNet-Modell.	62
31	Genauigkeiten beim ResNet-Modell bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.	63
32	Trainingsdauer beim DenseNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	82
33	Trainingsdauer beim ResNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	83
34	Trainingsdauer beim DenseNet bei ausschließlicher Anpassung der CPU-Spannung.	84
35	Trainingsdauer beim ResNet bei ausschließlicher Anpassung der CPU-Spannung.	85

36	Energieverbrauch beim ResNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	86
37	Energieverbrauch beim VGG16 bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.	87
38	Energieverbrauch beim ResNet-Modell bei ausschließlicher Anpassung der CPU-Spannung.	88
39	Energieverbrauch beim VGG16-Modell bei ausschließlicher Anpassung der CPU-Spannung.	89
40	Genauigkeiten beim AlexNet bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.	90
41	Genauigkeiten beim DenseNet bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.	91
42	Genauigkeiten beim VGG16 bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.	92
43	Genauigkeiten beim AlexNet bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.	93
44	Genauigkeiten beim DenseNet-Modell bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.	94
45	Genauigkeiten beim ResNet-Modell bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.	95
46	Genauigkeiten beim VGG16-Modell bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.	96

Quellcodeverzeichnis

1	Code zum Einstellen des Hardwaregerätes und der deterministischen Ausführung	29
2	Code zum Laden des Datensatzes Cifar10	29
3	Architektur des Modells AlexNet	31
4	Funktion zum automatischen Starten der Messung	32
5	Funktion zum Verbessern der Modelleistung	33
6	Funktion zur Berechnung der Genauigkeit	33
7	Implementierung des Trainings	34
8	Implementierung zum Logging nach einem Training	36

Kurzfassung

Diese Masterarbeit untersucht die Möglichkeit beim Training neuronaler Netze auf der Graphics Processing Unit (GPU), Energie mithilfe von Power-Capping auf der Central Processing Unit (CPU) einzusparen. Dazu werden im Rahmen dieser Masterarbeit die Effekte einer Reduzierung der maximalen CPU-Frequenz und -Spannung auf die Energieeffizienz und Leistung von vier tiefen Lernmodellen – AlexNet, VGG16, ResNet101 und DenseNet121 – untersucht. Diese Modelle werden mit den Datensätzen CIFAR-10 und Fashion-MNIST auf einer GPU trainiert. Es werden dabei verschiedene Messreihen durchgeführt, bei denen nach jedem Trainingsdurchlauf die maximale CPU-Frequenz und die CPU-Spannung reduziert werden. Außerdem werden während des Trainings verschiedene Metriken, wie die CPU-Frequenz, die CPU-Leistungsaufnahme, die Trainingsdauer sowie die Trainingsgenauigkeit mithilfe der Programme PyCharm, HWiINFO und AMD Ryzen Master aufgezeichnet. Die daraus resultierenden Ergebnisse dienen dazu, die Frage zu beantworten, inwiefern das Verringern der maximalen CPU-Frequenz und der CPU-Spannung die Energieeffizienz und Leistungsfähigkeit des Trainingsprozesses beeinflusst.

1 Einleitung

Durch die Anwendung neuronaler Netze ist künstliche Intelligenz (KI) in unserer heutigen Gesellschaft weit verbreitet und in verschiedenen Anwendungen allgegenwärtig und somit kaum noch wegzudenken. So sind zum Beispiel Spamfilter zum Sortieren der E-Mails, automatisch generierte Berichte oder auch virtuelle Assistenten, wie zum Beispiel Alexa, fester Bestandteil des modernen Alltags. Es gibt jedoch einige Nachteile, die durch die Nutzung neuronaler Netze entstehen. Einer dieser Nachteile ist zum Beispiel der Energieverbrauch beim Trainieren dieser Netze. [1] Aber auch die Anwendung, also die Inferenz, von bereits trainierten neuronalen Netzen verbraucht eine Menge Energie. [2]

Das tiefe Lernen hat in den letzten Jahren zwar viele Fortschritte beim maschinellen Lernen ermöglicht und erreicht, jedoch erfordern die Modelle für das tiefe Lernen in der Regel große Datenmengen und eine hohe Rechenleistung, was mit einem hohen Energieverbrauch verbunden ist. Dementsprechend steigen die Bedenken über den entstehenden Energieverbrauch und den damit hervorgerufenen finanziellen und ökologischen Kosten. [3]

Die steigende Bedeutung des tiefen Lernens in verschiedenen Anwendungsgebieten hat zu einer verstärkten Nutzung von neuronalen Netzwerken geführt [4, S. 4]. Der bereits erwähnte Energieverbrauch während des Trainingsprozesses stellt dabei eine Herausforderung dar, die es zu bewältigen gilt. Das Forschungsinteresse liegt nun darin, die Energieeffizienz des Trainings von neuronalen Netzwerken durch gezieltes Power-Capping der CPU zu verbessern. Da hier ein proportionaler Zusammenhang zwischen Taktfrequenz, Spannung und Leistungsaufnahme der CPU besteht [5].

Die konkrete Fragestellung, die sich daraus ergibt, lautet:

„Inwiefern beeinflusst das Power-Capping der CPU, während des Trainings von neuronalen Netzen auf der GPU, die Energieeffizienz und Leistungsfähigkeit des Trainingsprozesses?“

Um diese Frage zu beantworten, wird durch die Profilierung der Leistungsaufnahme während des Trainings geprüft, ob eine gezielte Reduzierung der maximalen CPU-Frequenz und Spannung zu signifikanten Energieeinsparungen führen kann. Dabei soll die Performance des neuronalen Netzwerks möglichst wenig negativ beeinflusst werden. Das Training der neuronalen Netze findet auf der GPU statt. Allerdings werden diverse Einstellungen der CPU untersucht, um diese nicht zu vernachlässigen und eine Forschungslücke zu schließen, die in Kapitel 3 erläutert wird.

Das Ziel, das sich daraus ergibt, ist es, den signifikanten Energieverbrauch beim Training neuronaler Netze durch gezieltes Power-Capping der CPU zu optimieren und dabei potenzielle Kompromisse in der Leistungsfähigkeit des neuronalen Netzes zu identifizieren und hervorzuheben.

Im Fokus dieser Untersuchung stehen die Profilierung der CPU-Frequenz, der Leistungsaufnahme, der Trainingsdauer, der Modellgenauigkeit und des Energieverbrauchs während des Trainingsprozesses. Mithilfe geeigneter Werkzeuge werden die Auswirkungen auf die Leistungsfähigkeit und den Energieverbrauch bei verschiedenen CPU-Taktfrequenzen und Spannungen analysiert. In welchem Umfang eine gezielte Reduzierung der CPU-Frequenz und Spannung während des Trainings zu einer bedeutsamen Einsparung von Energie führen kann, ohne die Leistungsfähigkeit des neuronalen Netzwerkes spürbar zu beeinträchtigen, ist dabei von hohem Interesse. Dabei sollen unterschiedliche Reduktionsstufen der CPU-Taktfrequenz und der Spannung in unterschiedlichen Modellen des tiefen Lernens betrachtet werden. Die Frequenz und die Spannung der CPU wird mit dem Werkzeug AMD Ryzen Master [6] angepasst.

2 Grundlagen

Dieses Kapitel führt wesentliche Konzepte ein, die für das Verständnis der vorliegenden Masterarbeit unverzichtbar sind. Es beginnt mit einer eingehenden Erklärung des tiefen Lernens, wobei dessen verschiedene Komponenten genauer betrachtet werden. Zudem werden die verwendeten Modelle und Datensätze erläutert sowie die jeweiligen Rollen der CPU und GPU genauer beschrieben. Abschließend wird auf das Powermanagement durch dynamische Spannungs- und Frequenzskalierung (DVFS) eingegangen.

2.1 Tiefes Lernen

Das tiefe Lernen ist ein Teilbereich des maschinellen Lernens und ist einer der faszinierendsten Forschungsbereiche aus dieser Disziplin. Die Modelle des tiefen Lernens liefern für sehr viele verschiedene Fragestellungen revolutionäre Ergebnisse und das vor allem im Bereich der Bild- und Spracherkennung. Das tiefe Lernen wird aber nicht nur dafür eingesetzt, sondern es findet auch Anwendung im Bereich der Fahrzeugindustrie. Als Beispiel sind hier selbstfahrende Autos zu nennen. [7] Des Weiteren werden die Modelle des tiefen Lernens auch in der Finanzwelt für Aktenvorhersagen oder Risikoprognosen eingesetzt. Aber auch die Biologie profitiert vom tiefen Lernen, beispielsweise in der Genomik. [7] Genomik ist die Erforschung der genetischen Information von Lebewesen [8]. Es gibt allerdings eine Vielzahl an Aufgaben. Einige dieser Aufgaben wurden bereits im Kapitel 1 benannt.

Das eben erwähnte maschinelle Lernen ist ein Teilbereich der künstlichen Intelligenz. Dieser Bereich beschäftigt sich mit der Entwicklung von Algorithmen und statistischen Modellen, die aus großen Datensätzen lernen und Muster in diesen Daten erkennen, ohne dass bestimmte Regeln explizit programmiert werden müssen. Maschinelle Lernsysteme können eigenständig aus Beispieldaten lernen und ihre Leistung in einer bestimmten Aufgabe, wie Bild- und Spracherkennung, stetig verbessern. Diese Eigenschaft haben klassische Algorithmen nicht, da diese auf festen Regeln und Anweisungen basieren. Aus diesem Grund treffen die maschinellen Lernsysteme oft auch präzisere Vorhersagen als die klassischen Algorithmen. Diese Lernalgorithmen sind flexibel und können sich an neue und unbekannte Daten anpassen, während klassische Algorithmen auf vordefinierten Regeln basieren und nicht so anpassungsfähig sind. [9]

Abschließend sei gesagt, dass beim tiefen Lernen künstliche neuronale Netze zum Einsatz kommen, die sich durch einen komplexen Aufbau aus Neuronen und Schichten auszeichnen. [10] Im Folgenden werden die wichtigsten Bestandteile erklärt, die in den Architekturen des tiefen Lernens enthalten sind.

2.1.1 Neuron

Die künstlichen neuronalen Netze bestehen aus den Neuronen und den Verbindungen zwischen diesen. Diese beiden Bestandteile sind die grundlegenden Elemente eines neuronalen Netzes. Es gibt heutzutage viele verschiedene Variationen von neuronalen Netzen, die allerdings alle auf einem grundlegenden Aufbau basieren. [11] Es lässt sich hier also bereits feststellen, dass das Neuron der Grundbaustein eines neuronalen Netzes ist. Dieses Neuron kann mehrere Eingangssignale (Inputs) empfangen und mittels einer sogenannten Aktivierungsfunktion weiterverarbeiten und als Ausgangsignal (Output) weiterleiten, siehe hierzu auch **Abbildung 1**. [7]

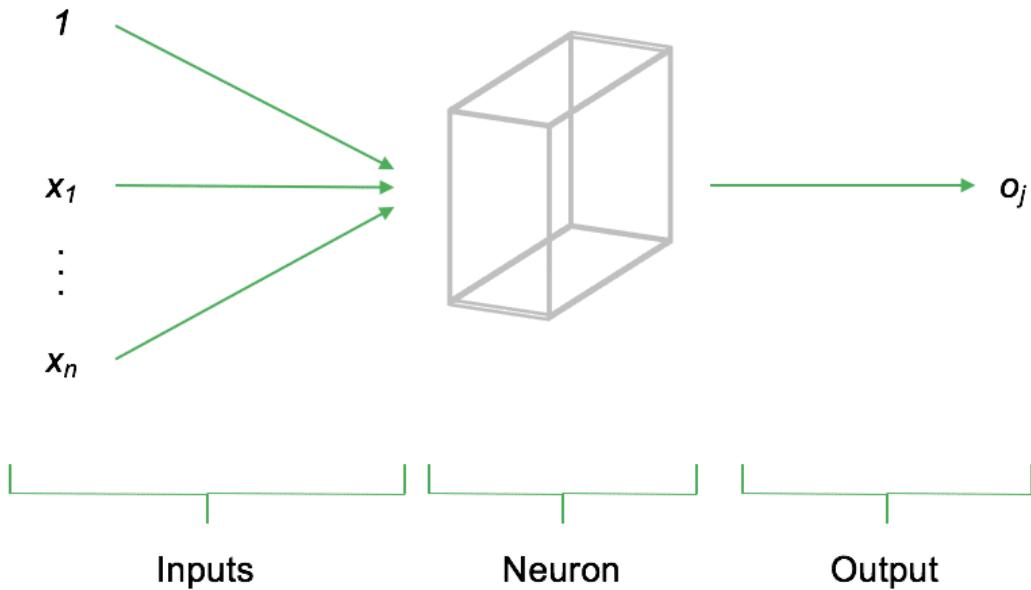


Abbildung 1: Vereinfachte Darstellung eines Neurons in einem neuronalen Netz. [7]

Das Neuron i ist mit mindestens einen oder mehreren Vorgängerneuronen verbunden. Dabei dient die Ausgabe eines Vorgängerneurons als Eingabe für das Neuron i , siehe **Abbildung 2**. Außerdem haben die Verbindungen zwischen den Neuronen Gewichte, die auf den jeweiligen Eingabewert angewendet werden. Zudem kann es einen Bias-Wert geben, der einen konstanten Wert darstellt und zu der Eingabe, für die Aktivierungsfunktion (f_{act}), addiert wird. Die Netzeingabe (net), die aus der Ausgabe des Vorgängerneurons, den Gewichten und beispielsweise dem Bias gebildet wird, wird von der Aktivierungsfunktion des Neurons genutzt, um die sogenannte Aktivierung (act) des Neurons zu berechnen. Wobei hier Schwellwerte eine Rolle spielen können. Beispielsweise kann erst dann die Ausgabe gegeben werden, wenn die Aktivierung einen bestimmten Schwellenwert übertrifft. Ist der Schwellwert nicht erreicht, dann wird die Aktivierungsfunktion immer wieder aufgerufen. [11]

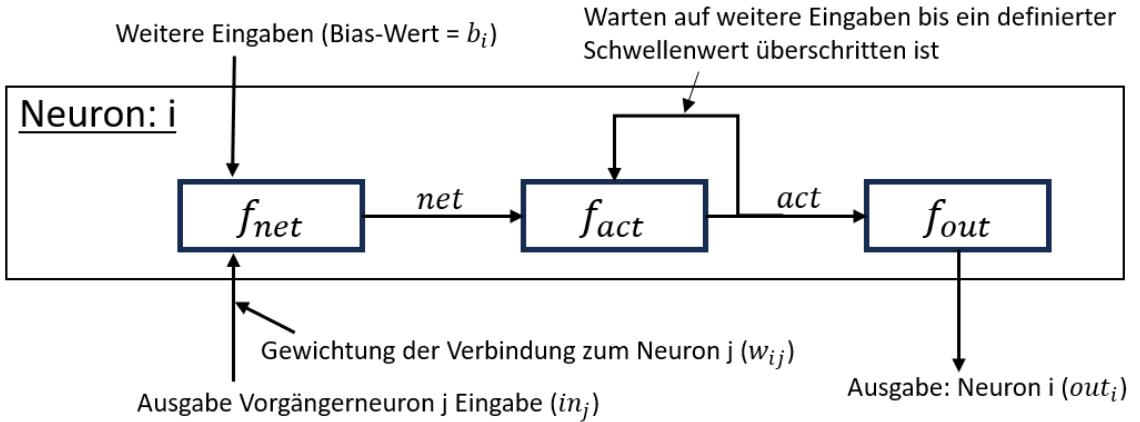


Abbildung 2: Detaillierte Darstellung des Neurons. Angelehnt an [11]

Die Ausgabe eines Neurons errechnet sich also wie folgt, siehe **Gleichung 1**.

$$out_i = f_{act} \left(\sum_j^n w_{ij} * in_j + b_i \right) \quad (1)$$

Hier ist out_i die Ausgabe, f_{act} ist die Aktivierungsfunktion, w_{ij} ist das Gewicht der Eingabe in_j und b_i ist der Bias-Wert.

Es gibt viele verschiedene Aktivierungsfunktionen, die verwendet werden können. Die Aktivierungsfunktion, die im Rahmen dieser Arbeit in den verwendeten Modellen des tiefen Lernens implementiert wurde, ist die sogenannte Rectifier-Funktion oder auch ReLU-Funktion. [11] Diese Funktion gibt für alle Eingaben < 0 den Wert 0 zurück. Für Werte ≥ 0 wird der Ausgabewert dem Eingabewert entsprechen, siehe **Gleichung 2.** [12]

$$R(x) = \begin{cases} x, & \text{wenn } x \geq 0 \\ 0, & \text{wenn } x < 0 \end{cases} \quad (2)$$

Weitere Aktivierungsfunktionen wären beispielsweise die Sigmoidfunktion, die Schwellwertfunktion, die Softmax-Funktion oder auch die Identitätsfunktion. Die vom Neuron errechnete Ausgabe wird dann über die Verbindungen als Eingabe für die folgenden Neuronen genutzt. [11]

2.1.2 Neuronales Netz

Ein künstliches neuronales Netz wird als mathematisches Modell verstanden. Sie bestehen aus einem gerichteten Graphen, der aus Knoten und Kanten besteht. Die Knoten werden von den Neuronen gebildet und die Verbindungen zwischen diesen Neuronen sind die Kanten. Hieraus lässt sich die sogenannte Schichten-Architektur ableiten. Diese Schichten werden von der Eingabeschicht, der versteckten Schicht und der Ausgabeschicht gebildet. Dabei ist es möglich, dass unterschiedliche Anzahlen von versteckten Schichten vorhanden sind, die keine Verbindung zur Umgebung besitzen und damit von außen nicht direkt beeinflussbar sind. [11] Zusätzlich stellen diese versteckten Schichten den zentralen Bestandteil der Architekturen von Modellen des tiefen Lernens dar [7].

Die Eingabeschicht ist dafür da, um das neuronale Netz mit den notwendigen Informationen zu versorgen. Sie nimmt also die Eingabedaten auf. Die in der Eingabeschicht enthaltenen Neuronen verarbeiten die Daten und führen diese gewichtet an die nächste Schicht im Netzwerk weiter. [10] Die nächste Schicht, die versteckte Schicht, gewichtet die Informationen erneut, extrahiert Merkmale und Muster und reicht die Informationen bis zur Ausgabeschicht weiter [11]. In der Ausgabeschicht wird die final resultierende Entscheidung, basierend auf den Eingabedaten, ermittelt und ausgegeben [10].

Eine schematische Darstellung des neuronalen Netzes mit zwei versteckten Schichten ist in **Abbildung 3** zu sehen. Diese Abbildung zeigt eine vergleichsweise einfache Architektur. In solchen einfachen Architekturen sind alle Neuronen der benachbarten Schichten miteinander verbunden [7].

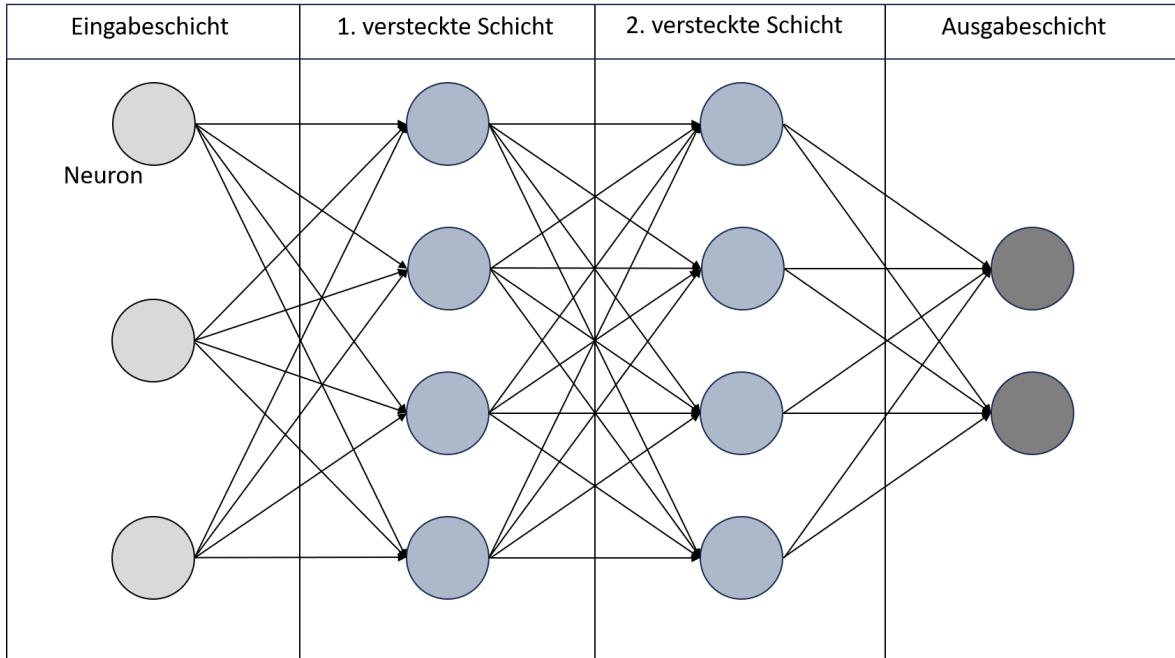


Abbildung 3: Schematische Darstellung eines neuronalen Netzes. Angelehnt an [11].

In einem künstlichen neuronalen Netz können unterschiedliche Verbindungen zwischen den Schichten und Neuronen vorhanden sein. Vorwärtsgerichtete Netze haben nur Verbindungen von der Eingabe- zur Ausgabeschicht, während rückgekoppelte Netze Schleifen und Verbindungen zu vorherigen Schichten haben. Ein Repräsentant eines vorwärtsgerichteten künstlichen neuronalen Netzes sind die Convolutional Neural Networks (CNN), die vor allem in der Bildverarbeitung zum Einsatz kommen. [11]

2.1.3 Convolutional Neural Networks (CNN)

Die Architektur von Convolutional Neural Networks (CNN) besteht zusätzlich aus der Faltungsschicht (Convolutionalschicht) und der Poolingschicht. Die in der Praxis implementierten CNN-Modelle besitzen einige weitere Schichten. [13] Die gesamte Architektur eines CNN-Modells am Beispiel der VGG16 Architektur aus der Praxis ist in **Abbildung 4** dargestellt.

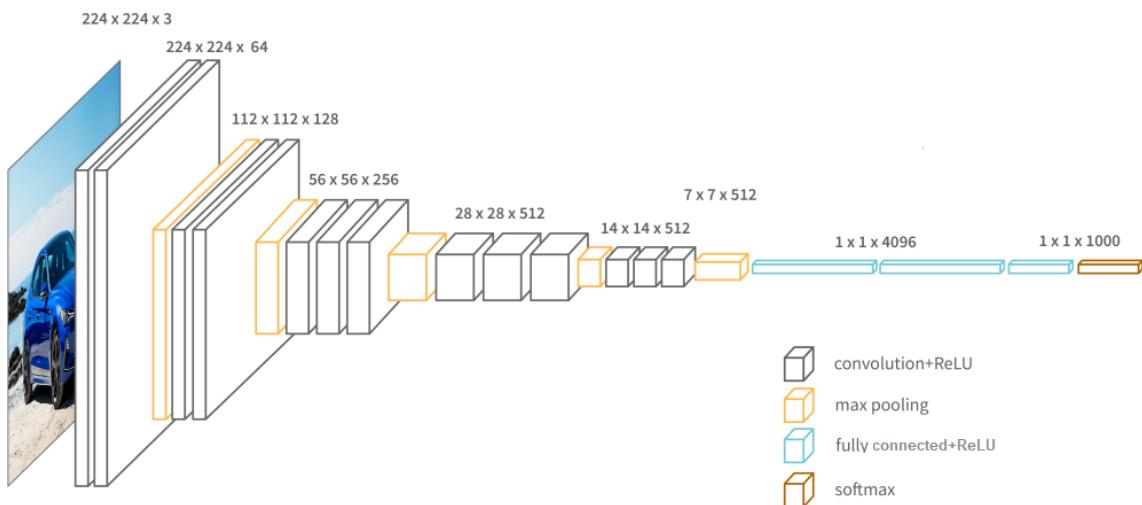


Abbildung 4: Aufbau/Architektur eines CNN Modells am Beispiel von VGG16 [13].

In der Abbildung 4 sind mehrere verschiedene Schichten zu erkennen. Für jeden Schichtblock sind Zahlen bzw. Dimensionen angegeben. Diese Dimensionen, beispielsweise 224x224x3, stehen zum einen für die Größe des Bildes und zum anderen für die Anzahl angewandter Filter bzw. hier für die Anzahl der Farbkanäle des Eingangsbildes [14]. So ist für das genannte Beispiel die Höhe = 224 px, die Breite = 224 px und die Anzahl an Farbkanälen = 3, die für den RGB-Farbraum stehen. Die x128, beispielsweise in 112x112x128, steht für die Anzahl der unterschiedlichen angewendeten Filter, die auf die Eingabedaten angewendet werden.

Jeder dieser 128 Kanäle repräsentiert eine andere Merkmalskarte, die verschiedene Merkmale des Eingangsbildes extrahiert, wie zum Beispiel Kanten, Texturen oder andere Merkmale. [14] In der Abbildung 4 ist für die letzte Fully-Connected-Schicht und die Softmax-Schicht 1x1x1000 angegeben. Dies bedeutet, dass die Bilder bei der Klassifizierung in 1000 Kategorien eingeteilt werden können [14]. Außerdem sind die bereits erwähnten Faltungs- und Poolingschichten zu sehen.

Die Faltungsschicht wendet Filter auf die Eingabedaten an und verarbeitet die darin enthaltenen Informationen, um Merkmale aus den Eingabedaten zu extrahieren. [11] Diese Informationen können bei einem Bild zum Beispiel einzelne Linien oder Formen sein [15]. In der Faltungsschicht findet also die Faltung statt, um bestimmte Merkmale zu extrahieren. Bei der Faltung wird über die Eingabe ein Filter gelegt und mittels Multiplikation der Daten wird der Filterausschnitt ermittelt [16], siehe **Abbildung 5**. Das Ergebnis aus der Abbildung 5 ergibt sich wie folgt. $\text{Filterausschnitt} = \mathbf{1*1+0*1+1*1+0*0+1*1+0*0+1*0+0*0+1*1} = 4$. Hierbei sind die fett geschriebenen Zahlen aus dem Filter und die anderen aus der Eingabe. Es wird also jede Koordinate aus dem Filter mit der dazugehörigen Koordinate aus der Eingabe, wo der Filter aktuell steht, verrechnet. Als Beispiel $\text{Filter}(0, 0) * \text{Eingabe}(0, 0) + \text{Filter}(0, 1) * \text{Eingabe}(0, 1) \dots$

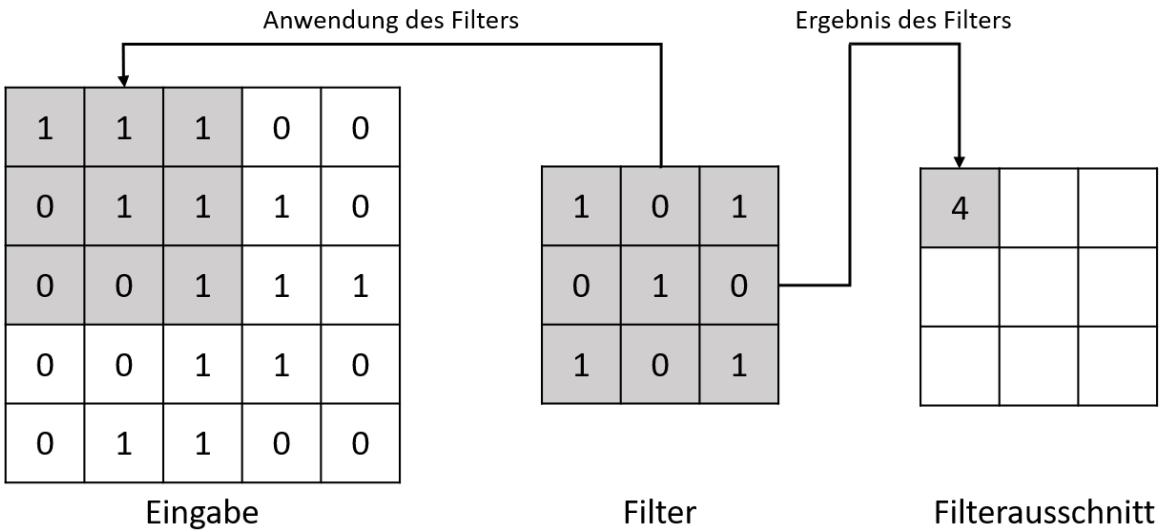


Abbildung 5: Darstellung der Faltung aus den Eingabedaten. Angelehnt an [16]

Das Netz erhält somit wichtige Informationen, die zur Lösung der gegebenen Problemstellung beitragen. Die Faltungsschicht führt eine sehr genaue Analyse der Eingabedaten durch, wodurch große Datenmengen erzeugt werden, die dann in der Poolingschicht verdichtet werden müssen [11]. Die Poolingschicht dient dazu, die räumliche Größe der Filterausschnitte aus der Faltung zu reduzieren, wodurch sich die Anzahl der zu verarbeitenden Parameter verringert. Dabei wird das stärkste Merkmal herausgefiltert und die Schwächeren werden verworfen.

Die Poolingschichten, die am häufigsten verwendet werden, sind Max-Pooling und Average-Pooling. Beim Max-Pooling wird der höchste Wert innerhalb des Filterausschnittes gewählt und beim Average-Pooling wird der Durchschnittswert aus dem Filterausschnitt gewählt und weiterverarbeitet, siehe **Abbildung 6.** [17]

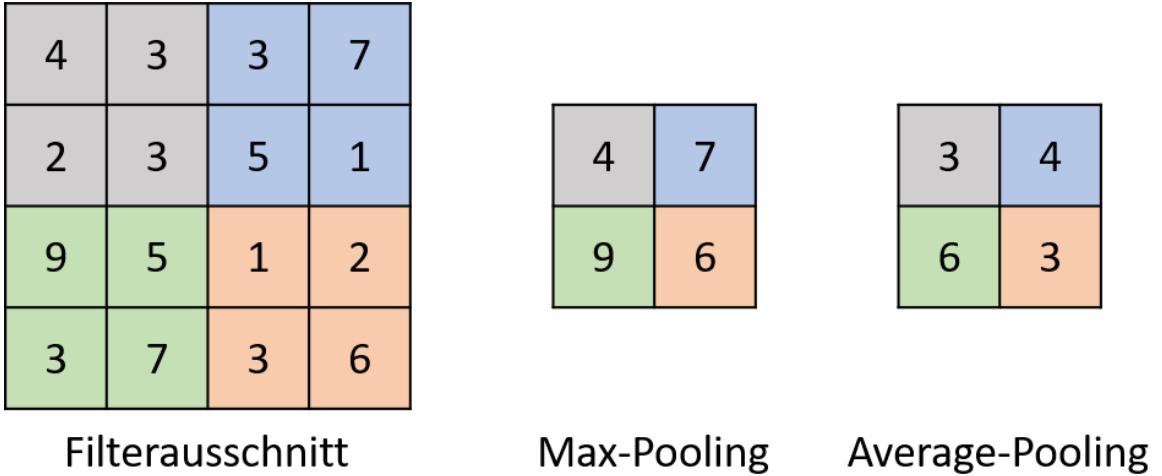


Abbildung 6: Darstellung vom Max- und Average-Pooling. Angelehnt an [18]

Fully-Connected-Schicht:

In der Fully-Connected-Schicht, in manchen Quellen auch als "Dense" bezeichnet, werden die extrahierten Merkmale aus den Faltungsschichten in eine endgültige Ausgabe bzw. Klassifikation umgewandelt. Diese Schicht ist in der Regel mit einer Aktivierungsfunktion, wie der Softmax-Funktion, verknüpft, um die Wahrscheinlichkeiten der verschiedenen Klassen zu berechnen. [17] In dieser Schicht sind alle Neuronen mit jedem Vorgängerneuron verbunden [14].

Softmax-Funktion:

Die Softmax-Funktion ist wie die ReLU-Funktion eine Aktivierungsfunktion im Bereich des maschinellen Lernens. Sie bildet die Eingangswerte auf den Bereich zwischen "0" und "1" ab und ist besonders als letzte Schicht in einem neuronalen Netzwerk geeignet, weil sie die Ausgabewerte des neuronalen Netzes so skaliert, dass sie als Wahrscheinlichkeiten interpretiert werden können, siehe **Gleichung 3.** So kann eine Auswahl der Klasse mit der höchsten Wahrscheinlichkeit als Vorhersage des Modells getroffen werden. Die Softmax-Funktion (Gleichung 3) nimmt einen Vektor mit den Werten x_i als Eingabe und rechnet diese Werte, abhängig von deren Größe, in Wahrscheinlichkeiten um. Ein hoher Wert führt zu einer hohen Wahrscheinlichkeit im Endergebnis. [19]

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (3)$$

Anhand von solchen Architekturvorgaben kann ein CNN-Modell implementiert werden. Wie so ein Modell trainiert werden kann, wird im Folgenden erläutert.

Um das neuronale Netz zu trainieren, gibt es beispielsweise das überwachte Lernen mittels Backpropagation. Das überwachte Lernen wird hauptsächlich bei Klassifikationsaufgaben eingesetzt. Beim überwachten Lernen wird eine Information benötigt, um die Datensätze einordnen zu können. Diese Information kann zum Beispiel ein Label sein, das in kodierter Form die Klassenzugehörigkeit eines Datensatzes angibt. Diese Label dienen dann dazu, die vom neuronalen Netz ermittelte Klassenzugehörigkeit mit der tatsächlichen Zugehörigkeit zu vergleichen. Dabei wird die Abweichung zwischen der ermittelten und der tatsächlichen Zugehörigkeit als Grundlage für das Lernen des Netzes verwendet. Diese Abweichung wird dann durch das Netz zurückgeführt und die Gewichte werden entsprechend angepasst. [11] Das neuronale Netz lernt beim überwachten Lernen also auf Basis von bekannten Daten und Beispielen.

Die unterschiedlichen Lernalgorithmen beim überwachten Lernen beruhen auf dem Backpropagation-Algorithmus. Der Backpropagation-Algorithmus durchläuft drei Schritte. Im ersten Schritt werden die Gewichte mit Zufallswerten initialisiert. Im zweiten Schritt werden die Trainingsdaten eingegeben und die Netzausgabe wird berechnet. Der dritte Schritt führt einen Vergleich der berechneten Ausgabe mit der erwarteten Ausgabe durch, um den Gesamtfehler zu ermitteln. Der Fehler wird durch das Netz zurückgeführt und die Gewichte werden angepasst, um den Fehler zu minimieren. Dies wird wiederholt, bis das Netz einen akzeptablen Gesamtfehler aufweist. [11]

2.2 CNN-Modelle

Es wurde im vorherigen Abschnitt erläutert, wie ein CNN-Modell aufgebaut ist und was die einzelnen Schichten in diesen Modellen für Aufgaben haben. Im folgenden Abschnitt werden die in dieser Arbeit verwendeten CNN-Modelle kurz erläutert und die Besonderheiten geklärt.

2.2.1 AlexNet

Die Architektur AlexNet wurde aus der Notwendigkeit heraus geboren, die Ergebnisse der ImageNet-Challenge zu verbessern. AlexNet war eines der ersten Deep-Convolution-Netze, das bei dem ImageNet-Wettbewerb 2012 (LSVRC-2012) mit einer Genauigkeit von 84,7 % sehr gut abschloss [20]. Dagegen erreichte das zweitplatzierte Netz nur eine Genauigkeit von 73,8 % [20]. AlexNet besitzt fünf Faltungsschichten, mit jeweils einer Max-Poolingschicht. Dazu kommen dann noch drei Fully-Connected-Schichten. Die Aktivierungsfunktion ist hier die Rectified Linear Unit (ReLU) Funktion. Außerdem besitzt dieses Netzwerk etwa 62 Millionen trainierbare Parameter. [20] Die detaillierte Struktur der Architektur des Netzwerkes kann im Tutorial von Anwar [20] nachgeschlagen werden.

2.2.2 VGG16

Das Modell VGG16 (Visual Geometry Group) ist ein CNN mit 16 Schichten und etwa 138 Millionen trainierbaren Parametern. Das VGG-Modell wurde ursprünglich von Simonyan und Zisserman in der Arbeit *"Very Deep Convolutional Networks For Large-Scale Image Recognition"* [21] vorgestellt. VGG16 verbessert AlexNet und ersetzt die großen 11x11- und 5x5-Filter (Filterausschnitte) durch Sequenzen von kleineren 3x3-Filters. Ein 3x3-Filter ist die optimale Größe, da eine kleinere Größe die Informationen von links-rechts und oben-unten nicht erfassen kann. Somit ist das VGG das kleinstmögliche Modell, um die räumlichen Merkmale eines Bildes noch zu erkennen. Außerdem verwendet das VGG-Modell viele kleinere Schichten im Vergleich zum AlexNet, wodurch das Netz schneller konvergiert. Das VGG-Modell nutzt ebenfalls die Aktivierungsfunktion ReLU wie das AlexNet, um die Trainingszeit zu reduzieren. [22] Die VGG16-Architektur kann auf der Internetseite von Interviewbit [13] oder in Abbildung 4 angesehen werden.

2.2.3 ResNet101

Das sogenannte Residual network (ResNet) ist ein Modell des tiefen Lernens und eine CNN-Architektur, die hunderte bis tausende Schichten unterstützen kann. Dieses Modell wird hauptsächlich für Computer-Vision-Anwendungen eingesetzt. Bei den klassischen Architekturen kam es beim Hinzufügen weiterer Schichten zu dem Problem des "verschwindenden Gradienten". [23] Dieses Problem tritt auf, wenn die Gradienten bei zu vielen Schichten durch wiederholte Multiplikation so weit reduziert werden, dass die Gewichtsanpassungen ebenfalls sehr klein werden, was das Training ineffektiv macht oder zum Stillstand bringt [24]. An dieser Stelle kommt das ResNet zum Einsatz, denn es bietet eine innovative Lösung für dieses Problem. Die Lösung nennt sich "Skip Connections". Das ResNet stapelt mehrere Faltungsschichten, überspringt diese Schichten und verwendet die Aktivierungen der vorherigen Schicht wieder. Dieses Überspringen beschleunigt das initiale Training, indem es das Netz in weniger Schichten komprimiert. Wenn das Netz dann neu trainiert wird, werden alle Schichten erweitert und die verbleibenden Teile des Netzes, die sogenannten residual Blöcke, können einen größeren Teil des Feature-Space des Eingangs erkunden. [23]

Ein Residual-Block besteht aus mehreren Faltungsschichten, die durch sogenannte "Shortcut Connections" miteinander verbunden sind. Die Idee dabei ist es, dass jede Schicht nicht nur die Filterausschnitte ihrer Eingaben lernt, sondern auch die Residual-Features zwischen den Eingaben und den gewünschten Ausgaben. Diese Residual-Features werden dann direkt zu den Ausgaben der Schicht hinzugefügt, anstatt sie zu überschreiben.

Durch das Hinzufügen dieser Shortcut-Verbindungen kann das Netzwerk besser lernen. Dies macht das Training tieferer Netze effektiv, da das Durchlaufen der Gradienten durch das Netzwerk verbessert wird. [25] In **Abbildung 7** ist ein Residual-Block zu sehen und dessen Funktionsweise mit den "Skip Connections".

Darüber hinaus wird in Abbildung 7 aufgezeigt, wie die Ausgabe x der vorherigen Schicht als Eingabe für den Block gilt und zu der Ausgabe des Blocks addiert wird. Diese Residual Blöcke glätten den sogenannten Gradientenfluss während der Backpropagation. Somit können Netze auf über 100 Schichten skaliert werden. [25] Die von He et al. [25] vorgestellte ResNet101 Architektur kann in der Arbeit von He et al. [25] angesehen werden.

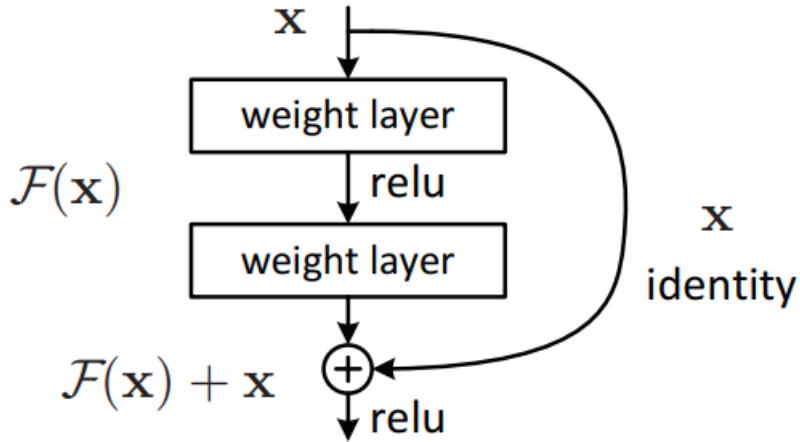


Abbildung 7: Residual Block. [25]

2.2.4 DenseNet121

In verschiedenen Arbeiten wurde bereits gezeigt, dass die convolutional network genauer und effizienter trainiert werden können, wenn sie kürzere Verbindungen zwischen den Schichten aufweisen. An dieser Stelle kommt das Dense Convolutional Network (DenseNet) zum Einsatz. Diese Architektur geht ebenfalls das Problem des "verschwindenden Gradienten" an. In dieser Architektur ist, in einem sogenannten Dense-Block, jede Schicht mit jeder anderen Schicht in der Feed-Forward-Methode direkt verbunden. Daraus ergibt sich, dass ein convolutional network mit L Schichten $\frac{L*(L+1)}{2}$ direkte Verbindungen besitzt, während klassische convolutional networks mit L Schichten auch nur L direkte Verbindungen besitzen. Für das sogenannte DenseNet bedeutet dies, dass die Filterausschnitte der vorherigen Schichten als Eingaben genutzt werden und die Filterausschnitte der eigenen Schicht als Eingabe an alle folgenden Schichten weitergegeben werden. Somit hat beispielsweise die L -te Schicht genau L Eingaben. [26]

In Abbildung 8 ist das DenseNet mit der vollen Vernetzung innerhalb der Dense-Blöcke zu sehen. Dabei bestehen die Dense-Blöcke normalerweise aus mehreren Faltungsschichten, gefolgt von einer Pooling-Schicht. [26]

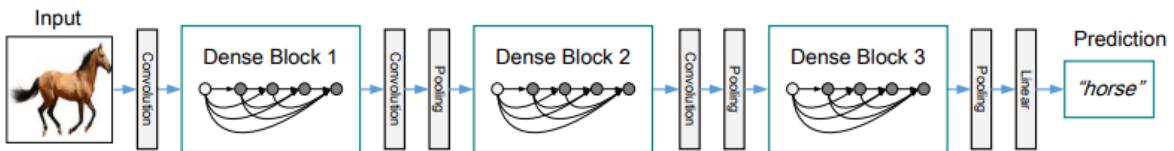


Abbildung 8: Vereinfachter Aufbau eines Dense Convolutional Neural Network. [26].

Beim DenseNet fördert die vollständige Vernetzung in einem Dense-Block den Informationsaustausch und die Wiederverwendung von Merkmalen erheblich, was zu einer verbesserten Leistung und weniger Berechnungen führt. [26] Die DenseNet121-Architektur kann in der Arbeit von Huang et al. [26] angesehen werden.

2.3 Datensätze

Die verschiedenen Modelle des tiefen Lernens werden anhand von verschieden gekennzeichneten Daten aus einem Datensatz trainiert. Diese Modelle können die Merkmale direkt über diese Daten lernen, ohne manuelle Merkmalsextraktion. [27] Im Folgenden werden die zwei Datensätze kurz vorgestellt, die in dieser Masterarbeit verwendet wurden.

2.3.1 CIFAR10

Der CIFAR10 Datensatz wurde in der Arbeit von Krizhevsky et al. [28] vorgestellt. Der Name des Datensatzes steht für "Canadian Institute for Advanced Research", wobei die "10" die Anzahl der Klassen bezeichnet. Dieser Datensatz enthält insgesamt 60.000 beschriftete (labeled) 32x32-Farbbilder in zehn verschiedenen Klassen. Dabei sind 6.000 Bilder pro Klasse vorhanden. Die unterschiedlichen Klassen in diesem Datensatz sind airplane, automobile, bird, cat, deer, dog, frog, horse, ship und truck. Dabei gibt es keinerlei Überlappungen zwischen den einzelnen Klassen. Die Klasse automobile enthält beispielsweise keine trucks. Außerdem sind keinerlei Duplikate in diesem Datensatz vorhanden. Die Zuordnung zu einer Klasse erfolgt anhand der Beantwortung der Frage "Was ist auf dem Bild zu sehen?". Wenn diese Frage gestellt wird, sollte der Klassennname ganz oben auf der Liste der Antworten stehen. Des Weiteren sollte das Bild fotorealistisch sein und hauptsächlich das Objekt enthalten, auf das sich der Klassenname bezieht. Dabei darf das Objekt auf dem Bild beispielsweise verdeckt sein, solange die Identität des Objektes klar ist. [28]

2.3.2 Fashion-MNIST

Der Fashion-MNIST ist ein Datensatz von vielen verschiedenen Zalando-Artikeln. Der Trainingsdatensatz besteht insgesamt aus 60.000 Bildern und der Testdatensatz aus weiteren 10.000 Bildern. Bei diesen Bildern handelt es sich um bereits beschriftete (labeled) 28x28-Graustufenbilder, die in zehn verschiedene Klassen unterteilt sind. [29] Die unterschiedlichen Klassen in diesem Datensatz sind t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag und ankle boot [30]. Zalando-Research beabsichtigt den Fashion-MNIST Datensatz als direkten Ersatz für den ursprünglichen MNIST-Datensatz für das Benchmarking von Algorithmen des maschinellen Lernens zu etablieren [29].

2.4 Rolle der CPU und GPU beim tiefen Lernen

2.4.1 CPU

Die CPU, auch als Central Processing Unit bekannt, ist der Hauptprozessor des Computers. Die CPU verarbeitet verschiedene Befehle, wie zum Beispiel Ein- und Ausgabeoperationen. Die Leistung, die eine CPU leisten kann, wird dabei unter anderem von der Taktrate und dem Cache-Speicher beeinflusst, der wiederum von der eingesetzten technologischen Implementierung abhängig ist. Die Taktrate ist für die CPU-Leistung wichtig, da sie angibt, wie viele Taktzyklen die CPU pro Sekunde ausführen kann. [31] Der Cache-Speicher ist ebenfalls für die CPU-Leistung wichtig, da er als schneller Zwischenspeicher genutzt wird, um häufig benötigte Daten und Befehle zu sichern. Dadurch müssen diese nicht ständig aus dem langsameren Arbeitsspeicher (RAM) geholt werden, wodurch sich die Verarbeitungsgeschwindigkeit erhöht. Dabei gibt es mehrere Ebenen von Cache-Speichern, die sich in ihrer Größe und Geschwindigkeit unterscheiden. Der L1-Cache ist der schnellste und kleinste Speicher. Er speichert die am häufigsten benötigten Befehle und Daten, um Verzögerungen bei der Datenübermittlung zu vermeiden und die CPU optimal auszulasten. Der L2-Cache ist größer und etwas langsamer. Der L3-Cache ist der größte, aber langsamste Cache-Speicher und dient zur Leistungsoptimierung der anderen Caches. [32] Die Leistung der CPU wird in sogenannten Flops (Floating-Point-Operations per Second) gemessen. Dieser Wert gibt an, wie viele Fließkommaberechnungen in einer Sekunde von der CPU durchgeführt werden können. [31]

Um die Gesamtausführungszeit zu verringern und die Gesamtleistung der CPU zu verbessern gibt es noch zwei weitere aufeinander aufbauende Techniken. [33] Moderne CPUs besitzen dafür eine digitale Schaltung, die sogenannte Sprungvorhersage. Diese Technik versucht das Ergebnis einer Operation zu erraten und das wahrscheinlichste Ergebnis vorzubereiten. Wenn nämlich ein Verzweigungsbefehl auftritt, weiß die CPU nicht sofort, welchen Befehl sie als Nächstes ausführen soll, da dies vom Ergebnis des aktuellen Befehls abhängt. Das führt dazu, dass die CPU darauf wartet, dass die nächste Anweisung abgerufen wird, sobald die aktuelle Anweisung ihre Ausführung beendet hat. Um diesem Problem entgegenzuwirken, sagt die Sprungvorhersage die Verzweigung voraus, wodurch die CPU diesen Befehl lädt. Hierbei kann es allerdings passieren, dass die Vorhersage falsch war und die CPU die bereits abgerufene Anweisung wegwirft und somit Zyklen verschwendet. Eine korrekte Vorhersage führt zu einer deutlich schnelleren Ausführung. Man unterscheidet zwischen statischer und dynamischer Sprungvorhersage. Die statische sagt immer dasselbe voraus und die dynamische passt ihre Vorhersage an die vergangenen Ergebnisse an. [34]

Auf der eben erklärten Sprungvorhersage baut die sogenannte spekulative Ausführung auf. Dies ist eine Optimierungstechnik, bei der die CPU Aufgaben ausführt, bevor sie dazu aufgefordert wird. Damit hat die CPU bereits Informationen bereit, die eventuell zu einem bestimmten Zeitpunkt benötigt werden. Dadurch wird die Wartezeit bis zum Eintreffen geeigneter Anweisungen für den nächsten Schritt vermieden. Die spekulative Ausführung nutzt die Sprungvorhersage, um vorherzusagen, welche Anweisungen in naher Zukunft benötigt werden. [33]

Die CPU besteht aus diversen Komponenten. Diese einzelnen Komponenten haben ihre eigenen Aufgabenbereiche und verarbeiten Daten und führen Anweisungen aus. Die Hauptkomponenten einer CPU sind die Steuereinheit, das Register, die ALU (Arithmetische Logikeinheit), die MMU (Speicherverwaltungseinheit), der Cache und der Taktgeber. Zusätzlich kommen dann noch der Adressbus, der Datenbus und der Steuerbus hinzu. [35, S. 356-358]

Der sogenannte Taktgeber erzeugt in regelmäßigen Abständen Impulse, die dafür sorgen, dass sich die Operationen der CPU synchronisieren. Daraus ergibt sich die sogenannte Taktrate, die in Hertz [Hz] angegeben wird. Die Taktrate bestimmt dabei die Anzahl der Befehle, die die CPU pro Sekunde ausführen kann. Heutzutage passen allerdings die CPUs ihre Taktraten an die Auslastung an, um ein Gleichgewicht zwischen der Leistung und dem Stromverbrauch zu erreichen. [36, S. 32] Der Zusammenhang zwischen Taktrate und Stromverbrauch wird im Kapitel 2.5 genauer erklärt.

Die bereits erwähnten Begriffe Arbeitsspeicher, MMU und Cache geben schon einen kleinen Einblick in die Speicheranbindung der CPU. Was der Cache ist und wofür dieser zuständig ist, wurde bereits am Anfang dieses Kapitels geklärt. Die CPU und damit auch der L1-Cache sind über den Front-Side-Bus (FSB) mit dem Memory-Controller-Hub verbunden, der die CPU mit dem Arbeitsspeicher verbindet. Dieser Controller verwaltet die Leitungen, die vom Arbeitsspeicher zur CPU führen. [37] Die L2- und L3-Caches werden vom Backside-Bus mit der CPU verbunden [38].

Die CPUs bestehen meistens nicht nur aus einem einzigen physischen Kern, sondern aus mehreren Kernen, die die Prozesse bzw. Aufgaben nacheinander in wenigen Threads parallel ausführen können [39]. CPUs mit nur einem Kern sind häufig in eingebetteten Systemen zu finden, wobei auch in diesem Bereich der Anteil an CPUs mit mehreren Kernen stetig steigt [40]. CPUs können sequenzielle Aufgaben mit komplexen Berechnungen schnell und effizient ausführen, allerdings sind sie bei der parallelen Verarbeitung von mehreren Aufgaben nicht besonders geeignet [41].

2.4.2 GPU

Die GPU, auch als Graphical processing unit bekannt, ist ein Grafikprozessor, der auf Berechnungen für Grafikdarstellungen optimiert ist [42]. Die GPU kann in der CPU integriert, im Chipsatz verbaut oder als Grafikkarte installiert sein [43]. Die in der CPU oder im Chipsatz integrierte GPU besitzt keinen eigenen Videospeicher (V-RAM), sondern greift auf den Arbeitsspeicher (RAM) des Systems zu. Die als Grafikkarte installierte GPU besitzt einen eigenen Videospeicher, wodurch sie deutlich leistungsfähiger ist als die integrierte GPU. [44] Einer der größten Unterschiede zwischen der CPU (multi-core Prozessor) und der GPU (many-core Prozessor) liegt in der Anzahl an verfügbaren Kernen. Die GPU kann mehrere tausend Kerne besitzen, während die CPU nur sehr wenige Kerne besitzt [42]. Allerdings sind die Kerne der GPU deutlich einfacher aufgebaut und leistungsfähiger als CPU-Kerne. Außerdem ist die GPU auf die Parallelverarbeitung optimiert. Des Weiteren haben GPU-Kerne auch weniger Speicher als CPU-Kerne. [45] Durch diese Eigenschaften ist es der GPU möglich, viele komplexe Berechnungen parallel auszuführen und große Datenmengen schnell zu verarbeiten. Dies führt zu einer merklichen Leistungsverbesserung gegenüber CPUs [46].

Die GPU besitzt, genau wie die CPU, auch Kerne, einen Speicher und eine Steuereinheit [45]. Außerdem besitzt die Grafikkarte auch einen Cache, der eine wichtige Rolle beim Rendering von Grafiken spielt. Dadurch können komplexe Rendering-Vorgänge beschleunigt werden, wobei häufig verwendete Daten in diesen Caches zwischengespeichert werden. [47] Außerdem ist die Grafikkarte mittels der PCIe-Schnittstelle mit dem Memory-Controller-Hub verbunden. Der Memory-Controller-Hub steuert also den Datenfluss zwischen den verschiedenen Komponenten - CPU, GPU und RAM - und setzt dazu Schreib- und Leseanforderungen der CPU und GPU um. [48] Wie bereits erwähnt besitzen die Grafikkarten auch einen Grafikspeicher (V-Ram), der sie leistungsfähiger als die integrierten GPUs macht, die einen Teil des Arbeitsspeichers nutzen [44]. Der V-RAM ist speziell zur Speicherung von Bilddaten geeignet und ist vor allem bei Anwendungen wichtig, die komplexe Bildtexturen oder polygonbasierte 3D-Strukturen berechnen [49]. Die GPU ist also für 3D-Berechnungen ausgelegt. Darunter fallen beispielsweise geometrische Berechnungen wie Transformationen, Rotationen, Shading oder Interpolation. Für diese geometrischen Berechnungen sind Matrix- und Vektorberechnungen notwendig. [43]

Des Weiteren sind GPUs sehr stark parallelisiert, weshalb sie mehrere Datenblöcke gleichzeitig verarbeiten können [43]. Die GPU kann dadurch sich wiederholende Aufgaben sehr schnell durchführen. Hierbei wird eine Aufgabe in mehrere kleine Teilaufgaben unterteilt, die dann parallel abgearbeitet werden können. [45]. Die GPU ist, aufgrund ihres stark parallelisierten Aufbaus, im Gegensatz zur CPU, nicht für sequentielle Aufgaben geeignet. [50]. Durch die Kombination der Parallelisierbarkeit und der angewandten Matrizenberechnung sind GPUs sehr gut für die Berechnung neuronaler Netze einsetzbar. [43].

Die GPU bringt also, im Vergleich zur CPU, für das Training der neuronalen Netze einige Vorteile. Durch eine deutlich erhöhte Anzahl an Tausenden von Kernen bringt die GPU eine sehr gute Parallelisierbarkeit der anfallenden Aufgaben mit. Des Weiteren ist die GPU aufgrund der Optimierung für 3D- bzw. geometrischen Berechnungen und der damit einhergehenden Vektor- und Matrixberechnungen beim Training der neuronalen Netze deutlich effektiver. Damit kann die GPU die anfallenden Kalkulationen beim Training merklich schneller ausführen, als es die CPU könnte. Die Matrixberechnungen sind zum Beispiel bei der Aktivierung eines Neurons oder auch beim Gradient Descent (Verlust einer Funktion minimieren) und bei der Backpropagation (Feinabstimmung der Gewichte) erforderlich. [43] Ein weiterer Vorteil für das Training auf der GPU sind die beliebten Frameworks für das tiefe Lernen, wie TensorFlow und PyTorch, die optimierte Backends für die GPU-Beschleunigung bieten. Durch diese Frameworks ist es den Entwicklern möglich, die Modelle mittels CUDA auf GPUs zu trainieren und sich dabei voll und ganz auf die Aufgaben des maschinellen Lernens zu konzentrieren. [51] Das Ausführen von allgemeinen Berechnungen auf GPUs wird als "General Purpose Computing on GPUs" (GPGPU) bezeichnet. Um dies zu ermöglichen werden GPU-Programmiermodelle eingesetzt. Ein solches Modell ist zum Beispiel das sogenannte CUDA (Compute Unified Device Architecture), das von NVIDIA entwickelt wurde. CUDA ermöglicht Entwicklern und Ingenieuren, die Rechenleistung von NVIDIA-Grafikprozessoren (GPUs) für allgemeine Berechnungen zu nutzen. CUDA bietet die Möglichkeit beispielsweise den rechenintensiven Teil eines Programms auf der GPU zu parallelisieren und damit merklich zu beschleunigen. Das Grundkonzept ist dabei, dass CUDA die Parallelität moderner GPUs nutzt, indem es eine große Anzahl von Threads gleichzeitig ausführt. Dabei führt jeder Thread den selben Code auf verschiedenen Daten aus. Diese Threads sind in einem hierarchischen Modell organisiert. Die Threads bilden sogenannte Thread-Blöcke und diese bilden die Grids. [46] Um CUDA und damit die GPU-Beschleunigung in Python zu nutzen kann beispielsweise das bereits angesprochene Framework PyTorch genutzt werden. [52]

Das Training findet also vorzugsweise auf der GPU statt, während die Anwendung eines bereits trainierten neuronalen Netzes oft auf der CPU stattfindet. Die Gründe, die Anwendung auf der CPU auszuführen, sind zum einen die weniger rechenintensiven Aufgaben und zum anderen sind CPUs kostengünstiger als GPUs und auch die Betriebskosten sind hier geringer. [53]

2.4.3 Zusammenfassung: CPU-GPU

Die CPU hat beim Training sehr vielfältige Aufgaben. Sie übernimmt die Datenverarbeitung und -vorbereitung bevor die Daten beispielsweise zur GPU gesendet werden. Des Weiteren übernimmt die CPU die Steuerung und Koordination, was beispielsweise das Planen und Überwachen für den Fortschritt beim tiefen Lernen umfasst. Die CPU verwaltet Ein- und Ausgabeoperationen, wie das Anzeigen von Trainingsfortschritten und das Protokollieren von Metriken. [54] Es gibt allerdings noch einige weitere Aufgaben auf die hier nicht weiter eingegangen wird. Die genannten Beispiele geben bereits einen guten Überblick.

Abschließend lässt sich durch die im Kapitel 2.4.1 und 2.4.2 angesprochenen Fakten sagen, dass die GPU mit dem Lernprozess und den damit einhergehenden Berechnungen beschäftigt ist, während die CPU Daten entsprechend vorbereitet und austauscht, den Fortschritt protokolliert, weitere Hintergrundprozesse ausführt und mit der Prozessverwaltung beschäftigt ist. Die CPU wird zudem oft bei der Anwendung von neuronalen Netzen genutzt.

2.5 Powermanagement

Der Stromverbrauch von einem Computer hängt von vielen Faktoren ab. Dazu gehören beispielsweise die Hardwarekonfiguration, die Nutzungsdauer und auch die Art der Anwendungen, die auf dem Computer ausgeführt werden. Vor allem verbrauchen Anwendungen viel Strom, wenn es sich um rechenintensive Aufgaben handelt. Der Großteil des Stromverbrauchs entfällt jedoch auf die CPU, die GPU und den Bildschirm. Dabei verbrauchen leistungsstarke CPUs und GPUs auch mehr Strom. [55] Die CPU eines Computers verbraucht dabei den größten Anteil an Strom im gesamten System, dieser Anteil beträgt etwa 50 %. [56] Eine Möglichkeit, den Stromverbrauch bzw. den Energieverbrauch der CPU zu senken, ist die dynamische Spannungs- und Frequenzskalierung (DVFS).

Die dynamische Spannungs- und Frequenzskalierung (DVFS) ist eine Methode, die verwendet wird, um beispielsweise den Energieverbrauch von Prozessoren zu verringern [56]. DVFS ist eine weitverbreitete Technik, die zu einer Energieeinsparung in diversen Systemen führt. Als Beispiele sind hier Computersysteme, eingebettete Systeme und Hochleistungssysteme zu nennen. Um den Energieverbrauch in CMOS-Schaltkreisen zu senken, wird die Taktfrequenz sowie die Versorgungsspannung der CPU verringert. [57] Der proportionale Zusammenhang zwischen Leistungsaufnahme, Taktfrequenz und Versorgungsspannung ist in **Gleichung 4** zu sehen [58].

$$P \sim C * f * V^2 \quad (4)$$

Dabei ist C die Kapazität des Transistors, V die Versorgungsspannung und f die Taktfrequenz. Anhand von Gleichung 4 ist sehr gut zu erkennen, dass sich auch die Leistungsaufnahme P verringert, wenn die Frequenz f verringert wird. Dadurch, dass die Versorgungsspannung V quadratisch in die Gleichung einfließt, bedeutet die Verringerung der Versorgungsspannung eine etwas stärkere Verringerung der Leistungsaufnahme. Durch die Kombination dieser beiden physikalischen Werte ist eine erhebliche Energieeinsparung möglich. [56]

Wenn nur die Taktfrequenz verringert wird, dann wird zwar die Leistungsaufnahme reduziert, jedoch erhöht sich die Durchlaufzeit der Anwendung, was zu einem gleichbleibenden oder sogar erhöhten Energieverbrauch führen kann. Wird allerdings die Taktfrequenz und die Versorgungsspannung reduziert, dann verringert sich letztendlich auch der Energieverbrauch, ohne dass sich die Durchlaufzeit weiter erhöht. [56] Dieser Zusammenhang ist in **Abbildung 9**, mit einer beispielhaften Annahme einer halbierten Taktfrequenz und Versorgungsspannung, anschaulich dargestellt.

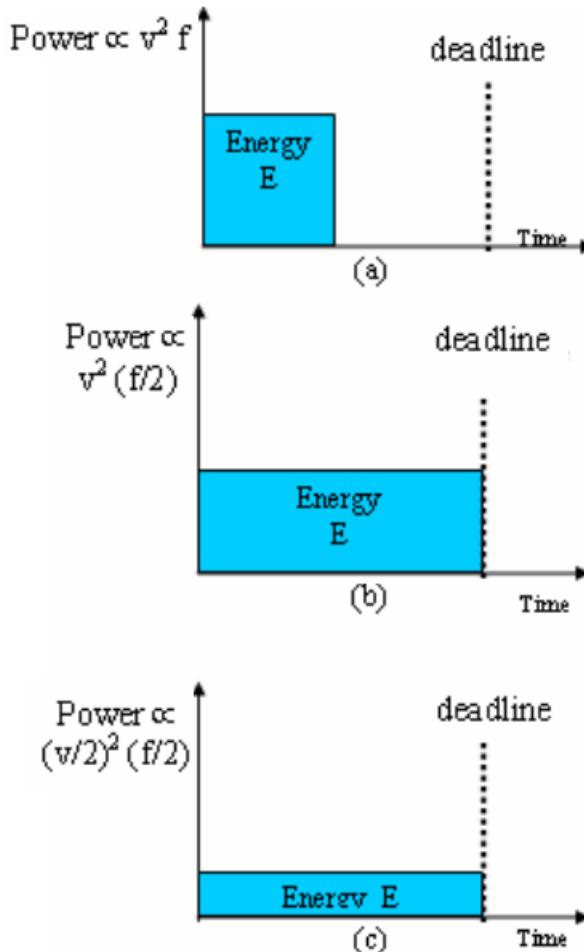


Abbildung 9: Schematische Darstellung des Zusammenhangs zwischen Taktfrequenz, Versorgungsspannung und dem Energieverbrauch. a) Keine Anpassung. b) Verringerung der Taktfrequenz. c) Verringerung der Taktfrequenz und der Versorgungsspannung. [56]

Die elektrische Energie kann durch **Gleichung 5** berechnet werden [59].

$$E = \int_a^b P(t) * dt \quad (5)$$

Diese Gleichung 5 lässt sich beispielsweise bei einer konstanten Leistung vereinfacht darstellen, siehe **Gleichung 6** [60].

$$E = P * t \quad (6)$$

Hier stellt E die elektrische Energie in Joule, P die elektrische Leistung in Watt und t die Zeit in Sekunden dar. Es lässt sich erkennen, dass eine Verringerung der Leistung nicht notwendigerweise zu einer Verringerung der Energie führt, da sich die Energie aus dem Produkt der Leistung und der Zeit ergibt.

3 Verwandte Arbeiten

In diesem Kapitel wird auf verwandte Arbeiten eingegangen, die auf die Techniken Power Capping bzw. DVFS eingehen, um Energie einzusparen.

3.1 GPU DVFS beim tiefen Lernen: Energie und Leistung

Die Studie [61] befasst sich mit dem Thema GPU-DVFS und untersucht dabei den Energieverbrauch und die Modellleistung beim tiefen Lernen. Dabei wird in der Arbeit [61] bereits darauf hingewiesen, dass das Training sowie die Inferenz von tiefen neuronalen Netzen (DNN) sehr viele Rechenressourcen benötigen und hohe Mengen an Energie verbrauchen. In dieser Arbeit wurden Experimente mit unterschiedlichen DVFS Einstellungen, DNN-Konfigurationen und Faltungsalgorithmen durchgeführt. Beispielsweise wurden die Grafikkarten GTX 2080TI und die Tesla P100 verwendet. Als Modelle des tiefen Lernens wurden AlexNet, VGG16, GoogleNet und das ResNet50 untersucht. Die Ergebnisse der Arbeit [61] zeigen, dass die optimalen Frequenzen im Vergleich zu den Standardeinstellungen eine Energieeinsparung von 8,7 % bis 23,1 % beim Training und 19,6 % bis 26,4 % bei der Inferenz ermöglichen. Außerdem konnte hier beobachtet werden, dass eine Erhöhung der Frequenz, die Leistung des Trainings und der Inferenz um bis zu 38,2 % bzw. 33,0 % verbessern kann.

3.2 Untersuchung der Energie-Leistung-Trade-Offs

Die Arbeit [62] befasst sich mit der Untersuchung von Energie-Performance-Trade-Offs beim Training von tiefen convolutional neural networks (CNNs) für die Bilderkennung. Dabei konzentriert sich die Arbeit auf die Anwendung von Power Capping auf der GPU, um Energieeinsparungen bei gleichzeitiger Minimierung der Leistungsreduzierung zu erzielen. Dabei werden die Grafikkarten Nvidia Quadro RTX 6000 und Nvidia V100 verwendet. Zudem werden unterschiedliche Modelle des tiefen Lernens, wie AlexNet, Vgg19, InceptionV3/V4 und ResNet untersucht. Die Ergebnisse der Arbeit [62] zeigen, bei optimierten Energie-Leistungs-Einstellungen, eine Energieeinsparung bis zu 33 % bei nur geringen bis mittleren Leistungsverlusten. Dabei war es das Ziel, Einstellungen zu finden, die größere Energieeinsparungen als Leistungsverluste ermöglichen. Durch diese Arbeit konnte gezeigt werden, dass durch sorgfältige Anpassung der GPU-Leistungsgrenzen erhebliche Energieeinsparungen bei vertretbaren Leistungseinbußen möglich sind.

3.3 Einordnung der verwandten Arbeiten

Die im Abschnitt 3.1 vorgestellte Arbeit markiert einen wichtigen Standpunkt in der Forschung, weil sie die Wirksamkeit von DVFS auf der GPU für die Energieeinsparung im Bereich des tiefen Lernens untersucht hat.

Die zweite Arbeit aus Abschnitt 3.2 ist ebenfalls bedeutsam, da sie sich auch mit der Thematik Energieeinsparungen durch Power Capping auf der GPU befasst. Auch hier zeigen die Ergebnisse, dass durch die GPU-Leistungsgrenzen erhebliche Energieeinsparungen möglich sind, ohne dabei die Leistung zu stark zu beeinträchtigen.

Die beiden vorgestellten Arbeiten in den Kapiteln 3.1 und 3.2 liefern bereits wertvolle Erkenntnisse zur Energieeffizienz beim tiefen Lernen unter Verwendung von Power-Capping bzw. DVFS auf der GPU. Jedoch gibt es hier noch eine Forschungslücke bezüglich der Anwendung von DVFS auf der CPU, die im Rahmen dieser Masterarbeit untersucht wird. Während bereits gute Ergebnisse bezüglich der Energieeinsparungen auf der GPU erzielt wurden, bleibt die Frage offen, ob ähnliche Einsparungen auch auf der CPU möglich sind und wie sich diese auf die Leistung der Modelle auswirken. Des Weiteren wurde bereits in Kapitel 2.5 erläutert, dass die CPU etwa 50 % des gesamten Energieverbrauchs des Computersystems ausmacht. Deshalb sind Untersuchungen zur Energieeinsparung bei der CPU besonders relevant. Diese Masterarbeit zielt darauf ab, diese Lücke zu schließen und herauszufinden, ob und wie DVFS auf der CPU beim Training neuronaler Netze zur Energieeinsparung genutzt werden kann.

4 Methodik

Da es im Rahmen dieser Masterarbeit um die Energieeinsparung beim tiefen Lernen geht, ist der Power Capping Ansatz ein vielversprechender Ansatz zur Energieeinsparung in modernen Computersystemen. Dieser Ansatz, stützt sich auf die dynamische Spannungs- und Frequenzskalierung (DVFS). Es wurde bereits im Kapitel 2.5 der Zusammenhang zwischen dem Energieverbrauch, der Frequenz und der Spannung anhand von Gleichung 4 aufgezeigt. Durch die Anpassung von Frequenz und Spannung können somit signifikante Energieeinsparungen erzielt werden. Diese Methodik wird im Rahmen dieser Arbeit angewandt, um den Einfluss durch Anpassung von Spannung und Frequenz genauer zu untersuchen. Wie diese Untersuchungen strukturiert und aufgebaut sind, wird in den folgenden Kapiteln erklärt.

4.1 Experimentelles Setup

Dieses Kapitel beschreibt die verwendete Hardware, die Werkzeuge, die Datensätze, die Modelle, ihre Struktur sowie die Konfiguration und Anpassungen dieser Modelle.

4.1.1 Eingesetzte Hardware

In diesem Kapitel wird die Hardware vorgestellt, die im Rahmen dieser Masterarbeit genutzt wurde. Die Hardware bildet die Grundlage für die Durchführung der Experimente und Analysen im Zusammenhang mit dem Power Capping Ansatz zur DVFS-Energieeinsparung.

Das verwendete System besteht aus verschiedenen Hardwarekomponenten, die zum Zeitpunkt der Messungen zur Verfügung standen. Die Hauptkomponenten des Systems waren wie folgt:

Mainboard: Verwendet wurde das ASRock B550 Phantom Gaming 4 Mainboard mit der BIOS-Version American Megatrends Inc. P1.10, 09.06.2020. Der Formfaktor ist ATX. Es werden folgende Sockel unterstützt: AMD AM4 Socket Ryzen 3000, 3000 G-Series, 4000 G-Series, 5000 und 5000 G-Series. Alle weiteren technischen Informationen können auf der Herstellerseite [63] nachgelesen werden.

Prozessor: Der verwendete Prozessor war der AMD Ryzen 5 3600 mit sechs CPU-Kernen, einer Basistaktrate von 3.6 GHz und einem AM4 Sockel. Die Produktlinie entspricht AMD Ryzen 5 Desktop Processors. Weitere technische Informationen zum Prozessor können beim Hersteller [64] eingesehen werden.

Grafikkarte: Die eingesetzte Grafikkarte, auf der das tiefe Lernen durchgeführt wurde, ist die NVIDIA GeForce RTX 3060 Ti. Diese Grafikkarte hat 4864 NVIDIA CUDA-Recheneinheiten, einen Boost-Takt von 1.67 GHz und einen Basistakt von 1.41 GHz. Außerdem hat die Grafikkarte eine Speichergröße von 8 GB. Weitere technische Informationen über die Grafikkarte sind der Herstellerseite [65] zu entnehmen.

Arbeitsspeicher: Im Einsatz ist das VENGEANCE LPX Speicherkit mit zwei 8 GB DDR4 DRAM Riegeln mit einer Geschwindigkeit von $2666 \frac{MT}{s}$. Weitere technische Informationen sind der Herstellerseite [66] zu entnehmen.

Speicher: Verwendet wird die M.2 Crucial P1 SSD, die eine Kapazität von 1 TB bereitstellt. Die genaue Bezeichnung ist CT1000P1SSD8. Diese SSD hat eine Leseschwindigkeit bis zu $2000 \frac{MB}{s}$ und eine Schreibgeschwindigkeit bis zu $1700 \frac{MB}{s}$. Weitere technische Informationen sind der Herstellerseite [67] zu entnehmen.

4.1.2 Verwendete Werkzeuge

Die verwendeten Werkzeuge, die in dieser Masterarbeit zum Einsatz kamen, sind HWiNFO [68], AMD Ryzen Master [6], Jupyter [69] und das Framework PyTorch [70]. Diese werden im Folgenden detailliert erläutert.

HWiINFO:

HWiINFO ist ein kostenloses Werkzeug, das für die Systemanalyse eingesetzt wird. Dieses Programm bietet Funktionen, die es einem ermöglichen sehr genaue Einblicke in die Metriken der Hardwarekomponenten, wie beispielsweise Temperaturen, Spannungen, Taktgeschwindigkeiten und Leistungsaufnahmen in Echtzeit zu erhalten. Dazu analysiert HWiINFO das gesamte System und erfasst dabei alle verfügbaren Hardwarekomponenten, wie zum Beispiel Prozessoren, Grafikkarten, Festplatten und Mainboards. Durch dieses Werkzeug können also die Metriken des gesamten Systems permanent überwacht werden. Dadurch können auch potenzielle Probleme frühzeitig erkannt werden. Darüber hinaus ermöglicht HWiINFO die Erstellung von Berichten über die Hardwarekonfiguration des Systems, die bei der Diagnose von Problemen oder der Durchführung von Systemverbesserungen hilfreich sein können. [68]

Eine Funktionsweise von HWiINFO ist auch die Möglichkeit, das Programm zu konfigurieren. Durch die Konfigurationsmöglichkeiten kann HWiINFO genau an die eigenen Bedürfnisse angepasst werden. Dabei sind die Auswahl bestimmter Sensoren, die Einrichtung benutzerdefinierter Alarme und die Konfiguration verschiedener Anzeigeeoptionen oder die Einstellung, welche Informationen protokolliert werden sollen, nur beispielhafte Nennungen. Dies ermöglicht beispielsweise nur die Leistungsaufnahme, die Frequenz und die Spannungen der CPU zu messen und zu protokollieren, wodurch eine übersichtliche Analyse gewährleistet werden kann. [68]

Zudem wird das Werkzeug von renommierten Unternehmen wie der NASA, Intel, AMD, DELL, GIGABYTE und Asus genutzt. Dies spricht dafür, dass HWiINFO ein bewährtes Werkzeug für wissenschaftliche Untersuchungen ist. [68]

Zusammenfassend bietet HWiINFO einen sehr guten Überblick über alle im System installierten Hardwarekomponenten und ermöglicht eine genaue Überwachung ebendieser Komponenten. Durch die anpassbare Benutzeroberfläche und die Protokollierungs- und Exportfunktion ist HWiINFO ein sehr nützliches Werkzeug, um die Leistung seines Systems zu überwachen und zu protokollieren.

Diese genannten Funktionen sprechen für den Einsatz dieses Werkzeugs, um im Rahmen dieser Masterarbeit die Metriken CPU-Taktfrequenz, CPU-Spannung und die CPU-Leistungsaufnahme zu überwachen und zu protokollieren.

AMD Ryzen Master

Das Werkzeug AMD Ryzen Master stellt Funktionen zur Steuerung und Überwachung des Prozessors bereit. Mithilfe dieser Funktionen kann die Prozessorleistung angepasst werden. Zur manuellen Steuerung der CPU können die maximal möglichen Taktfrequenzen und Spannungen der CPU-Kerne direkt in der Benutzeroberfläche angepasst werden. Dabei können die Anpassungen direkt, ohne Systemneustart, erfolgen, wodurch AMD Ryzen Master eine präzise Optimierung der Leistung in Echtzeit ermöglicht. Außerdem bietet dieses Programm die Möglichkeit verschiedene Leistungsprofile zu erstellen und zu speichern. Diese Profile können dann entsprechend der eigenen Anforderungen angepasst werden, die dann je nach Situation und Aufgabe ausgewählt werden können, um die maximale Leistung zu erzielen. Außerdem bietet AMD Ryzen Master Echtzeitinformationen über verschiedene Metriken des Prozessors. Darunter fallen Taktfrequenz, Spannung, Temperatur, CPU-Kernauslastung sowie die Leistungsaufnahme. [6]

AMD Ryzen Master wird aufgrund der direkten manuellen Einstellmöglichkeiten der CPU-Frequenz und der CPU-Versorgungsspannung eingesetzt. Somit ist eine vereinfachte Einstellung der Systemparameter möglich, ohne das System neu starten zu müssen. Zudem wird im Rahmen dieser Masterarbeit ein AMD-Prozessor verwendet, was ebenfalls für dieses Programm spricht. Weitere detaillierte Informationen über dieses Programm können in [71] eingesehen werden.

Jupyter-Notebooks

Die Verwendung des Begriffs Jupyter ist nicht eindeutig. Einerseits bezieht sich Jupyter auf das gesamte Projekt, andererseits wird der Begriff Jupyter häufig für das Jupyter-Projekt Jupyter-Notebooks verwendet. Jupyter Notebooks ist eines der Hauptprodukte von Jupyter. Dieses Werkzeug ermöglicht das Erstellen interaktiver Arbeitsblätter, so genannter Zellen, was es zu einem wertvollen Werkzeug in verschiedenen Anwendungsbereichen macht. Die Aufgabenbereiche von Jupyter Notebooks liegen insbesondere in wissenschaftlichen Bereichen, wie der Datenauswertung, der Statistik, der Visualisierung und dem maschinellen Lernen. Hier hat sich das Jupyter Notebook als Standard etabliert. Die nutzbaren Programmiersprachen sind Julia, Python und R, woraus sich auch der Name Jupyter ergab. Mittlerweile kann durch sogenannte Kernels die Anzahl an nutzbaren Programmiersprachen, wie C++, Ruby, Haskell, PHP und Java, erweitert werden. [72]

Verwendet wird dieses Werkzeug, weil es sich im Bereich der Datenanalyse und dem maschinellen Lernen zum Standard entwickelt hat und auch hier eine ausreichend große Tutorial-Landschaft vorhanden ist.

PyTorch

PyTorch ist ein Open-Source-Framework für das tiefe Lernen. Dieses Framework wird zum Aufbau neuronaler Netze verwendet. Durch die Flexibilität und Benutzerfreundlichkeit hat sich dieses Framework zum führenden Werkzeug im Bereich des maschinellen Lernens in der akademischen Welt entwickelt. Pytorch unterstützt dabei neuronale Netzwerkarchitekturen von einfachen linearen Regressionsalgorithmen bis hin zu komplexen convolutional neuronal networks. Das Framework basiert auf der Programmiersprache Python. Aufgrund der umfangreichen Bibliotheken bietet es auch vorkonfigurierte und vortrainierte Modelle an.[73]

Dieses Framework wird wegen der Flexibilität, der Benutzerfreundlichkeit, der einfachen Implementierung und der großen Gemeinschaft auf GitHub und Tutorial-Seiten genutzt. Zudem ist es ein Python-basiertes Framework, weshalb es auch in den genutzten Jupyter-Notebooks genutzt werden kann.

4.1.3 Verwendete Datensätze

Die Datensätze, die in dieser Masterarbeit zum Trainieren der Modelle verwendet wurden, sind der Cifar10-Datensatz [28] [74] und der Fashion-MNIST-Datensatz [29]. Diese Datensätze werden verwendet, da sie typische Benchmark-Datensätze für Bilderkennung sind. Darüber hinaus sind sie der Öffentlichkeit leicht zugänglich und lassen sich somit hervorragend für wissenschaftliche Zwecke nutzen.

4.1.4 Verwendete Modelle

Die Modelle des tiefen Lernens, die in dieser Masterarbeit untersucht wurden, sind DenseNet121, Resnet101, Alexnet und VGG16. Diese Modelle werden verwendet, da sie sogenannte Convolutional Neural Networks (CNNs) sind und sich besonders gut für 2D-Daten, wie Bilder, eignen [75]. Außerdem wurden Modelle mit unterschiedlicher Komplexität gewählt, um die Ergebnisse möglichst allgemein zu halten. Die ersten Basisimplementierungen und Ideen der Modelle werden aus einer Sammlung verschiedener Architekturen, Modelle und Tipps aus dem GitHub-Repository von Raschka [76] inspiriert. Diese werden angepasst und erweitert, damit zum Beispiel die Trainingszeiten protokolliert werden und die in dieser Arbeit genutzten Datensätze nutzbar werden. Zusätzlich soll beim Start des Trainingsprozesses die Messung mit HWiNFO starten und beim Ende des Trainingsprozesses auch wieder beendet werden. Des Weiteren müssen diese Modelle an die Bildeingangsgröße der verwendeten Datensätze angepasst werden. Bei diesen Anpassungen wird Bezug auf andere Modelle aus dem GitHub-Repository von Raschka [76] genommen. Dabei soll sich die Implementierung möglichst nahe an der originalen Architektur des Modells halten, siehe Kapitel 2.2.

4.1.5 Aufbau der Modelle/Jupyter-Notebooks

Die in dieser Masterarbeit verwendeten Modelle des tiefen Lernens wurden bereits im vorherigen Kapitel 4.1.4 erwähnt und im Kapitel 2.2 erklärt. In diesem Abschnitt geht es um die Implementierung eines solchen Modells im Jupyter-Notebook. Dabei wird der Fokus auf dem AlexNet mit dem CIFAR10 Datensatz liegen, da alle Modelle bzw. Jupyter-Notebooks grundsätzlich denselben Aufbau haben und dieses Modell eine recht simple Architektur besitzt. Die größten Unterschiede zwischen den Implementierungen liegen im Aufbau der Architektur des Modells. Die Anpassungen, die für den Fashion-MNIST Datensatz durchgeführt wurden, werden im Kapitel 4.1.7 angesprochen. Alle erstellten Jupyter-Notebooks sind in [77] hinterlegt.

Die Implementierung beginnt damit, dass eine Variable namens "device" definiert wird, die den Typ der Hardware-Geräte festlegt, auf denen die Tensorberechnungen ausgeführt werden sollen. Wenn eine CUDA-fähige GPU verfügbar ist, wird das Gerät auf "cuda:0" gesetzt. Zudem wird bei Verfügbarkeit der GPU die deterministische Ausführung von CUDA-Kerneln aktiviert. Dies ist notwendig, damit die Berechnungen auf der GPU deterministisch sind, sodass bei gleichen Eingabedaten immer das gleiche Ergebnis berechnet wird. Wenn diese Einstellung nicht vorhanden ist, könnten Berechnungen auf der GPU aufgrund von Faktoren wie Reihenfolge der Operationen oder Rundungsfehlern variieren, was zu unterschiedlichen Ergebnissen führen könnte. Somit wären die Ergebnisse nicht mehr reproduzierbar. Der Quellcode in Python sieht dazu wie folgt aus, siehe **Quellcode 1**.

```
1 if torch.cuda.is_available():
2     # Typ des Hardware-Gerätes festlegen
3     device = torch.device("cuda:0")
4
5     # Deterministische Ausführung von CUDA-kerneln aktivieren
6     torch.backends.cudnn.deterministic = True
```

Quellcode 1: Code zum Einstellen des Hardwaregerätes und der deterministischen Ausführung

Darauf folgen einige Konfigurationsmöglichkeiten, um beispielsweise die Batchgröße, die Lernrate und die Anzahl an Epochen, den Seed oder die Transformation für die Trainingsdaten einzustellen. Die Batch-Größe ist die Anzahl an Trainingsbeispielen, die in einem Durchgang durch das Modell gleichzeitig verarbeitet werden [78]. Die Lernrate beeinflusst, wie groß die Schritte sind, mit denen sich die Modellgewichte ändern [79]. Eine Epoche ist ein vollständiger Trainingsdurchlauf des Trainingsdatensatzes [80]. Anschließend werden die Daten für das Training geladen, siehe **Quellcode 2**.

```
1 # Laden der Daten zum Trainieren
2 trainset = torchvision.datasets.CIFAR10(
3     root='./data', train=True, download=True, transform=transform)
4
5 trainloader = torch.utils.data.DataLoader(
6     trainset, batch_size=batch_size, shuffle=True, num_workers=2)
7
8 # Laden der Daten zum Testen
9 testset = torchvision.datasets.CIFAR10(
10    root='./data', train=False, download=True, transform=transform)
11
12 testloader = torch.utils.data.DataLoader(
13     testset, batch_size=batch_size, shuffle=False, num_workers=2)
```

Quellcode 2: Code zum Laden des Datensatzes Cifar10

Die Daten werden in zwei Pakete aufgeteilt: den "trainset" für das Modelltraining und den "testset" zur Berechnung der Genauigkeit nach dem Training.

Der Hauptteil der Implementierung ist die eigentliche Architektur des Modells, siehe **Quellcode 3**. Hier wird zunächst der Seed gesetzt (Zeile 1), um die Ergebnisse, also die Genauigkeit, eines Durchlaufs möglichst identisch werden zu lassen. Dies erhöht die Reproduzierbarkeit und erleichtert das Debugging und die Forschung. Das Modell ist als Klasse definiert, indem zunächst einmal die verschiedenen Schichten und Aktivierungsfunktionen realisiert sind. Die Faltungsschichten enthalten Parameter wie die in_channels, out_channels, kernel_size, stride und padding. Die in_channels sind die Eingabekanäle, hier beispielsweise drei für den RGB-Farbraum. Die out_channels sind die Ausgabekanäle. Die Kernelgröße ist die Größe des Filters für die Faltung. Dieser wurde bereits im Kapitel 2.1.3 in Abbildung 5 erklärt. Das Padding ist dafür da, um das Eingabebild mit zusätzlichen Pixeln zu füllen, damit die Ausgabe den gewünschten Ausgabeabmessungen entspricht [81]. Bei einem Padding von zwei würde also der Rand des Bildes um zwei Pixel erweitert werden. Der Stride gibt die Schrittweite (Anzahl der Pixel) an, um die der Filter bei der Faltung über das Eingabebild geschoben wird [81]. Anschließend wird eine Adaptive-Average-Pooling-Schicht definiert (Zeile 29). Diese Schicht passt die Eingabedaten an, um eine feste Ausgangsgröße für den nächsten Teil des Netzes zu gewährleisten, unabhängig von der Größe der Eingabe.

Darauf folgt der "classifier" (Zeile 30), in dem eine Sequenz von Schichten definiert wird, die für die Klassifikation verwendet werden. Dieser spezielle Abschnitt besteht aus mehreren Schichten, darunter die Dropout-Schichten, die lineare Schichten (fully connected Schichten) und die ReLU-Aktivierungsfunktionen zwischen den Schichten.

Zum Schluss der AlexNet Klasse (Zeile 39) gibt es noch eine "forward"-Methode. Hier werden die Eingabedaten durch eine Reihe von Feature-Extraktions-Schichten geleitet, die wichtige Merkmale aus den Daten extrahieren. Dann wird die Ausgabe durch eine Adaptive-Average-Pooling-Schicht geleitet, um die Größe anzupassen. Dann wird die Form des Tensors angepasst und anschließend durchläuft die vorbereitete Eingabe die Klassifikationsschichten, um die finale Klassifizierung der Daten durchzuführen. Die Methode gibt anschließend die Ausgabe des Modells zurück, die die Wahrscheinlichkeiten für jede Klasse des Datensatzes angibt. Zum Schluss (Zeile 46) wird das Modell initialisiert und dem bereits definierter Gerät "cuda:0", also der Grafikkarte, übergeben.

```
1 torch.manual_seed(SEED)
2 class AlexNet(nn.Module):
3     def __init__(self, num_classes):
4         super(AlexNet, self).__init__()
5         self.features = nn.Sequential(
6             nn.Conv2d(in_channels=3, out_channels=64,
7                     kernel_size=3, stride=1, padding=1),
8             nn.ReLU(inplace=True),
9             nn.MaxPool2d(kernel_size=2, stride=2),
10
11             nn.Conv2d(in_channels=64, out_channels=192,
12                     kernel_size=3, stride=1, padding=1),
13             nn.ReLU(inplace=True),
14             nn.MaxPool2d(kernel_size=2, stride=2),
15
16             nn.Conv2d(in_channels=192, out_channels=384,
17                     kernel_size=3, stride=1, padding=1),
18             nn.ReLU(inplace=True),
19
20             nn.Conv2d(in_channels=384, out_channels=256,
21                     kernel_size=3, stride=1, padding=1),
22             nn.ReLU(inplace=True),
23
24             nn.Conv2d(in_channels=256, out_channels=256,
25                     kernel_size=3, stride=1, padding=1),
26             nn.ReLU(inplace=True),
27             nn.MaxPool2d(kernel_size=2, stride=2),
28         )
29         self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
30         self.classifier = nn.Sequential(
31             nn.Dropout(0.5),
32             nn.Linear(256 * 6 * 6, 4096),
33             nn.ReLU(inplace=True),
34             nn.Dropout(0.5),
35             nn.Linear(4096, 4096),
36             nn.ReLU(inplace=True),
37             nn.Linear(4096, num_classes),
38         )
39     def forward(self, x):
40         x = self.features(x)
41         x = self.avgpool(x)
42         x = x.view(x.size(0), 256 * 6 * 6)
43         x = self.classifier(x)
44         return x
45
46 net = AlexNet(number_classes).to(device)
```

Quellcode 3: Architektur des Modells AlexNet

Im nächsten Codeblock wird eine Funktion definiert, die den Tastendruck einer angegebenen Taste simuliert. Wenn diese Funktion aufgerufen wird, kann die Protokollierung mittels HWiNFO automatisch gestartet werden, siehe **Quellcode 4**. Diese Taste, hier "key", kann in HWiNFO gewählt werden und dieser Funktion entsprechend übergeben werden.

```
1 def simulate_key_press(key):
2     keyboard.press(key)
3     time.sleep(0.1)
4     keyboard.release(key)
```

Quellcode 4: Funktion zum automatischen Starten der Messung

Die Möglichkeit, die Messung mittels der Funktion in Quellcode 4 beim Beginn des Trainings automatisch starten zu können, ist aus folgenden Gründen notwendig und äußerst hilfreich. Wenn die Messung nach dem Starten des Trainings manuell gestartet werden müsste, dann würden unter Umständen wichtige Messpunkte aus den ersten Momenten des Trainings fehlen. Wenn die Messung vor dem Start des Trainings gestartet wird, dann wären Messpunkte vorhanden, die nicht mit dem Training in Verbindung stehen. Genau das Gleiche gilt auch für das Beenden der Messung.

Wenn man nun einen Schritt weitergeht und die Auswertungen durchführen möchte, kann es dazu kommen, dass die Ergebnisse verfälscht werden. Beispielsweise könnte der Energieverbrauch geringer oder höher sein, als er hätte sein müssen, weil eine unbestimmte Anzahl an falschen Messwerten in den Daten vorhanden ist. Zudem ist es äußerst schwer die Zeit zwischen dem Starten der Messung und dem Starten des Trainings zu protokollieren, da dieser Wert aufgrund menschlicher Variationen beim Starten der Messungen immer ein anderer sein kann. Außerdem wäre es ein erheblicher und vermeidbarer Mehraufwand, die Messwerte zu ermitteln, die nicht zum eigentlichen Training gehören. Der Hauptgrund für die Implementierung des Quellcodes 4 liegt in der geforderten Präzision der Messungen. Die Idee, die Messungen automatisch per Tastendruck zu starten, entstand, weil nur die für das Training relevanten Messwerte protokolliert werden sollen. Außerdem bietet HWiNFO die Möglichkeit, die Messungen per Tastendruck zu starten und zu beenden. Um die Messung per Tastendruck in HWiNFO zu starten, muss lediglich in den Einstellungen die Tastenkombination angegeben werden, siehe Kapitel 4.2.2.

Kurz bevor das eigentliche Training des Modells beginnt, kommt noch ein weiterer kleiner Codeblock, siehe **Quellcode 5**. In diesem Codeblock wird die Verlustfunktion "CrossEntropyLoss()" verwendet, um den Kreuzentropieverlust zwischen den Vorhersagen des Modells und den tatsächlichen Klassenlabels zu berechnen. Je besser die Vorhersage, desto geringer der Kreuzentropieverlust. Des Weiteren wird der Optimierungsalgorithmus "SGD()" verwendet, um die Gewichte des neuronalen Netzwerks schrittweise anzupassen und das Modell zu trainieren. Die Parameter des SGD-Optimierers wurden so ausgewählt, dass eine effiziente Konvergenz beim Training möglich ist und die Modellleistung verbessert wird.

```
1 criterion = nn.CrossEntropyLoss()  
2 optimizer = optim.SGD(net.parameters(), lr=learn_rate, momentum=0.9)
```

Quellcode 5: Funktion zum Verbessern der Modellleistung

Im nächsten Codeabschnitt, siehe **Quellcode 6**, wird zunächst die Funktion "calculate_accuracy" definiert, damit am Ende des Trainings durch Aufruf dieser Funktion das Modell auf den Testdatensatz überprüft werden kann. Hier wird dann die Genauigkeit des Modells nach dem Training berechnet.

```
1 # Funktion zur Berechnung der Genauigkeit auf den Testdaten  
2 def calculate_accuracy(loader):  
3     correct = 0  
4     total = 0  
5     with torch.no_grad():  
6         for data in loader:  
7             inputs, labels = data[0].to(device), data[1].to(device)  
8             outputs = net(inputs)  
9             _, predicted = torch.max(outputs.data, 1)  
10            total += labels.size(0)  
11            correct += (predicted == labels).sum().item()  
12    return correct / total
```

Quellcode 6: Funktion zur Berechnung der Genauigkeit

Darauf folgt dann der Codeabschnitt für das Training des Modells, siehe **Quellcode 7**. In Zeile 2 wird die bereits beschriebene Simulation des Tastendrucks durchgeführt, damit die Messung gestartet wird. Außerdem wird in Zeile 3 die Startzeit des gesamten Trainings in einer Variable gespeichert, um später die Gesamttrainingszeit berechnen zu können.

```

1 time.sleep(1)
2 simulate_key_press('p')
3 start_time = time.time()
4
5 for epoch in range(epochs):
6     epoch_start_time = time.time()
7     for i, data in enumerate(trainloader, 0):
8         # Verschiebe die daten auf die GPU
9         inputs, labels = data[0].to(device), data[1].to(device)
10
11         # Setze parameter gradients auf 0
12         optimizer.zero_grad()
13
14         # forward + backward + optimize
15         outputs = net(inputs)
16         loss = criterion(outputs, labels)
17         loss.backward()
18         optimizer.step()
19
20         # print Statistiken
21         if i % 2000 == 1999:
22             print('[%d, %5d]' % (epoch + 1, i + 1))
23
24         epoch_end_time = time.time()
25         elapsed_time = epoch_end_time - epoch_start_time
26         print(f"Epoch {epoch + 1}/{epochs}, Elapsed Time: {elapsed_time:.2f} seconds")
27
28 total_training_time = time.time() - start_time
29
30 # Berechnung der Genauigkeit auf den Testdaten
31 total_accuracy = calculate_accuracy(testloader) * 100
32 print(f"Accuracy on test data: {total_accuracy:.2f}")
33 print(f"Total Training Time: {total_training_time:.2f} seconds")
34 print('Finished Training')
35
36 time.sleep(1)
37 simulate_key_press('p')
```

Quellcode 7: Implementierung des Trainings

Dann wird im Quellcode 7, ab Zeile 5, das Training in einer Schleife über die Epochen durchlaufen. Hierbei ist jeder Epochendurchlauf ein vollständiger Durchlauf des Trainingsdatensatzes. Dann erfolgt eine Iteration durch die Trainingsdaten (ab Zeile 7). Hierbei wird der Trainloader verwendet. Dabei ist der Trainingsdatensatz in Batches aufgeteilt. Für jeden Batch werden die Eingaben und die zugehörigen Labels abgerufen und auf die Grafikkarte verschoben (Zeile 9). Anschließend werden die Gradienten der Parameter des neuronalen Netzwerks zurückgesetzt, um eine neue Berechnung für den aktuellen Batch zu starten (Zeile 12).

Der Rückwärtsdurchlauf wird durchgeführt (Zeile 17), nachdem die Vorwärtsdurchlaufberechnung (Zeile 15) durchgeführt wurde, die Vorhersagen generiert und der Verlust zwischen den Vorhersagen und den tatsächlichen Labels berechnet wurde. Am Ende eines Batches werden die Statistiken ausgegeben. Außerdem wird am Ende jeder Epoche die vergangene Zeit, seit dem Start der Epoche, berechnet und zusammen mit der aktuellen Epoche ausgegeben. Am Ende des Trainings wird die Genauigkeit sowie die Gesamtdurchlaufzeit des Trainings ausgegeben.

Zum Schluss der Implementierung folgt das Protokollieren der Laufzeit und der Genauigkeit des gesamten Trainingsdurchlaufs, siehe **Quellcode 8**. Hier wird ab Zeile 5 eine Funktion definiert, die zunächst die Daten für die neue Zeile in der CSV-Datei festlegt. Darauf folgt die Überprüfung, ob es bereits eine Datei mit dem angegebenen Dateinamen gibt. Dann wird die Datei geöffnet und die Spaltenüberschriften sowie die Daten selbst werden in die CSV-Datei geschrieben. Die Spaltenüberschriften werden allerdings nur beschrieben, wenn die CSV-Datei zu dem Zeitpunkt noch nicht existiert bzw. erstellt wurde. Ab Zeile 26 folgen dann einige Variablendefinitionen, die beispielsweise den Dateinamen und den Datensatznamen beschreiben.

```
1 #Zum loggen der Trainingslaufzeit und Genauigkeit des Models
2 def add_row_to_csv(filename, datensatz_name, frequency,
3                     voltage, runtime, accuracy):
4     # Daten fuer die neue Zeile
5     new_row = {'Datensatz': datensatz_name, 'Frequenz': frequency,
6                'voltage': voltage, 'Laufzeit': runtime,
7                'Accuracy': accuracy}
8
9     # Ueberpruefe, ob die Datei existiert
10    file_exists = os.path.isfile(filename)
11
12    # Oeffne die CSV-Datei im Modus 'a' (append), um Werte
13    # hinzuzufuegen
14    with (open(filename, 'a', newline='')) as csvfile:
15        # Erstelle einen CSV-Writer, falls die Datei neu erstellt wird
16        writer = csv.DictWriter(
17            csvfile, fieldnames=['Datensatz', 'Frequenz',
18                                  'voltage', 'Laufzeit', 'Accuracy'])
19
20        # Schreibe die Kopfzeile, falls die Datei neu erstellt wurde
21        if not file_exists:
22            writer.writeheader()
23
24        # Schreibe die neue Zeile in die CSV-Datei
25        writer.writerow(new_row)
26
27    # Daten fuer das Logging festlegen
28    filename = 'alexnet_cifar.csv'
29    datensatz_name = 'Cifar10'
30    frequency = '3000'
31    voltage = '1.6'
32    runtime = round(total_training_time, 2)
33    accuracy = total_accuracy
34
35    add_row_to_csv(filename, datensatz_name, frequency,
36                   voltage, runtime, accuracy)
```

Quellcode 8: Implementierung zum Logging nach einem Training

4.1.6 Konfiguration der Modelle

Die Konfiguration der Modelle variiert von Modell zu Modell. Die Lernrate liegt beispielsweise in einem Intervall von 0,0005 bis 0,01. Da die Lernrate die Größe der Schritte beeinflusst, mit denen sich die Modellgewichte ändern, wurden diese so angepasst, dass das verwendete Modell damit arbeiten kann, die Lerngeschwindigkeit angemessen ist und die Verlustfunktion konvergieren kann. Die Anzahl an Epochen sowie die Batchgröße werden bei allen Modellen gleich behandelt. So ist die verwendete Epochenzahl = 30 und die Batchgröße = 128. Ein gesamter Durchlauf des Trainingsdatensatzes ist eine Epoche. Die Batchgröße gibt die Anzahl der Trainingsbeispiele an, die in einem Durchlauf des Trainingsalgorithmus gleichzeitig verarbeitet werden. Die Trainingsausführung findet bei allen Modellen auf dem Gerät "CUDA:0" also der Grafikkarte statt. Einige Modelle besitzen noch die Möglichkeit den "Grayscale" auf True oder False zu setzen. Wenn beispielsweise der Fashion-MNIST verwendet wird, dann wird dieser auf True gesetzt, damit die Graustufenbilder verarbeitet werden können. Des Weiteren wird die Anzahl an Klassen der Datensätze auf "10" gesetzt, da beide Datensätze die Bilder in zehn Klassen bzw. Kategorien einteilen.

4.1.7 Anpassungen der Modell-Architekturen

Die Anpassungen, die durchgeführt wurden, orientieren sich an den Modellen aus dem GitHub-Repository von Raschka [76]. In diesem GitHub-Repository wurden die verwendeten Modelle bereits für den Cifar10 Datensatz angepasst. Dabei ist die Originalarchitektur weitestgehend erhalten geblieben. Außerdem wurde bei den Anpassungen hauptsächlich darauf geachtet, dass die Ausgabegrößen der einzelnen Schichten nicht zu klein werden und entsprechend weiterverarbeitet werden können. Dabei wurde sich an **Gleichung 7** orientiert [82].

$$\text{Ausgabegröße} = \frac{\text{Eingabegröße} - \text{Kernelgröße} + 2 * \text{padding}}{\text{stride} + 1} \quad (7)$$

Die Kernelgröße ist die Größe des Filters für die Faltung. Das Padding gibt an, um wie viele Pixel das Bild erweitert wird und der Stride gibt die Schrittweite (Anzahl der Pixel) des Filters an. Diese Werte sind den Schichten aus der Implementierung der entsprechenden Modelle zu entnehmen.

AlexNet:

Das originale Modell von AlexNet nutzt 224x224 Pixel große Bilder. Die verwendeten Datensätze Cifar10 32x32 Pixel und Fashion-MNIST 28x28 Pixel nutzen eine kleinere Auflösung, weshalb einige Änderungen durchgeführt werden mussten. Die spezifischen Anpassungen, die in der umgebauten Version des AlexNet-Modells für CIFAR-10 vorgenommen wurden, sind:

1. Die Eingabegröße und Eingangskanäle der Faltungsschichten wurden so angepasst, dass sie mit den kleineren Eingangsbildern und Graustufenbildern funktionieren.
2. Die Pooling-Schichten wurden so modifiziert, dass sie die Bildgröße angemessen reduzieren und dennoch die wichtigsten Merkmale beibehalten.
3. Die Fully Connected Schichten am Ende des Modells wurden entsprechend angepasst, um mit den neuen Dimensionen der Features nach den Faltungsschichten zu arbeiten.
4. Eine AdaptiveAvgPool2d-Schicht wurde hinzugefügt, um sicherzustellen, dass die Ausgabe der Faltungsschichten eine feste Größe hat, unabhängig von der Eingabegröße.

Diese Anpassungen ermöglichen es dem AlexNet-Modell, effektiv mit den kleineren Bildern des CIFAR-10- und Fashion-MNIST-Datensatzes zu arbeiten, während gleichzeitig das ursprüngliche Design und die Architektur des AlexNet beibehalten wird. Für den Fashion-MNIST wurden in der ersten Faltungsschicht die Eingangskanäle auf eins gesetzt, damit das Modell mit den Graustufenbildern arbeiten kann.

DenseNet121:

Um das DenseNet121-Modell für Cifar10 und Fashion-MNIST anzupassen, müssen einige Änderungen vorgenommen werden, um sicherzustellen, dass das Modell für die jeweiligen Datensätze geeignet ist. Zunächst wurde die erste Faltungsschicht kernel_size, stride und padding angepasst, damit die output_size die korrekte Größe hat und die kleineren Bilder der Datensätze in dem Modell verarbeitet werden können. Für den Fashion-MNIST Datensatz wurde in der Konfiguration, aufgrund der Graustufenbilder, die Grayscale-Einstellung auf True gesetzt.

VGG16:

Auch in dem Modell VGG16 müssen einige Änderungen vorgenommen werden, um sicherzustellen, dass das Modell für die jeweiligen Datensätze geeignet ist. Der Cifar10-Datensatz besteht aus RGB-Bildern mit einer Größe von 32x32 Pixeln und drei Farbkanälen und der Fashion-MNIST besteht aus 28x28 Pixeln und einem Farbkanal. Im Gegensatz dazu verwendet das originale VGG-16-Modell Eingabebilder mit einer Größe von 224x224 Pixeln. Daher wurde die erste Faltungsschicht in VGG-16 entsprechend angepasst, um mit den Eingabedimensionen vom Cifar10 und dem Fashion-MNIST kompatibel zu sein. Als zusätzliche Anpassung für den Fashion-MNIST wurden die Eingabekanäle, aufgrund der Graustufen, auf eins gesetzt und die Maxpoolingschicht im Block vier angepasst, damit der Ausgabetensor die korrekte Größe hat und die anderen Blöcke durchlaufen kann.

ResNet101:

Auch im ResNet101-Modell wurden Änderungen in der Eingabegröße durchgeführt, damit die kleineren Bilder in diesem Modell verarbeitet werden können. Ebenso wurde die Anpassung der Eingabekanäle für den Fashion-MNIST-Datensatz durchgeführt.

4.2 Datenerfassung

In diesem Kapitel wird darauf eingegangen, wie das Messverfahren abläuft und wie das Werkzeug HWiNFO für die Messungen konfiguriert wurde.

4.2.1 Verfahren der Messung

Beim Starten des Modelltrainings wird automatisch mittels HWiNFO die Messung von Hardwareinformationen, wie Versorgungsspannung, Frequenz und die Leistungsaufnahme der CPU gestartet und in einer CSV-Datei geschrieben, die im Installationsverzeichnis von HWiNFO gespeichert wird. Der gewählte Abfragezeitraum der Hardwaredaten beträgt 20 ms. Um diesen Wert zu ermitteln, wurde bei allen Modellen die Durchlaufzeit der einzelnen Batches gemessen. Dabei kam heraus, dass AlexNet 3-4 ms, DenseNet 70-75 ms, ResNet 50-60 ms und VGG16 6-8 ms für einen Batch benötigt. HWiNFO lässt allerdings keine Abfragezeiträume, die kleiner als 20 ms sind, zu, weshalb der kleinste Wert, also 20 ms, gewählt wurde. Das entspricht 50 Messungen pro Sekunde. Hierbei kommt es dazu, dass die Batches beim AlexNet und VGG16 zu schnell berechnet werden und keine Messpunkte innerhalb der Batchberechnungen garantiert sind. Die schnelle Berechnung der Batches lässt sich durch die äußerst kleinen 32x32 Pixel Bilder erklären. Außerdem werden während bzw. nach dem Training die gesamte Trainingsdurchlaufzeit, die Genauigkeit des Modells sowie die verwendete Frequenz, die genutzte Spannung und der Datensatz in eine CSV-Datei geschrieben. Somit können diese Daten später zur weiteren Auswertung verwendet werden. Um das automatische Starten zu gewährleisten, ist in den Modell-Jupyter-Notebooks kurz vor der Trainingsimplementierung eine Funktion eingebaut, die die Messung in HWiNFO startet, siehe Quellcode 4 und Quellcode 7 in Zeile 2. Um dies zu ermöglichen, muss HWiNFO gestartet und korrekt konfiguriert sein. Die Konfiguration von HWiNFO wird im nachfolgenden Kapitel 4.2.2 erläutert.

4.2.2 Konfiguration von HWiNFO

Beim Starten von HWiNFO muss die Checkbox "Nur Sensoren" bestätigt werden, damit die korrekte Benutzeroberfläche geöffnet wird. In den Sensoreinstellungen von HWiNFO werden im Allgemein-Tab einige Einstellungen vorgenommen. Der Abfragezeitraum wird auf 20 ms gesetzt und die Tastenkombination zum Starten der Sensor-Protokollierung wird für die automatische Messung auf "P" gestellt. Dabei müssen zusätzlich die Kreuze bei "Tastenkombination zum Zurücksetzen von Werten aktivieren" und "Aktivieren der Tastenkombination zum Umschalten" gesetzt sein.

Für die Auswertung ist es erforderlich, dass das Dezimal-Trennzeichen auf Punkt und das Tausender-Trennzeichen auf Komma eingestellt ist. In **Abbildung 10** sind die genauen Einstellungen zu sehen. Darüber hinaus sind keine weiteren Einstellungen notwendig.

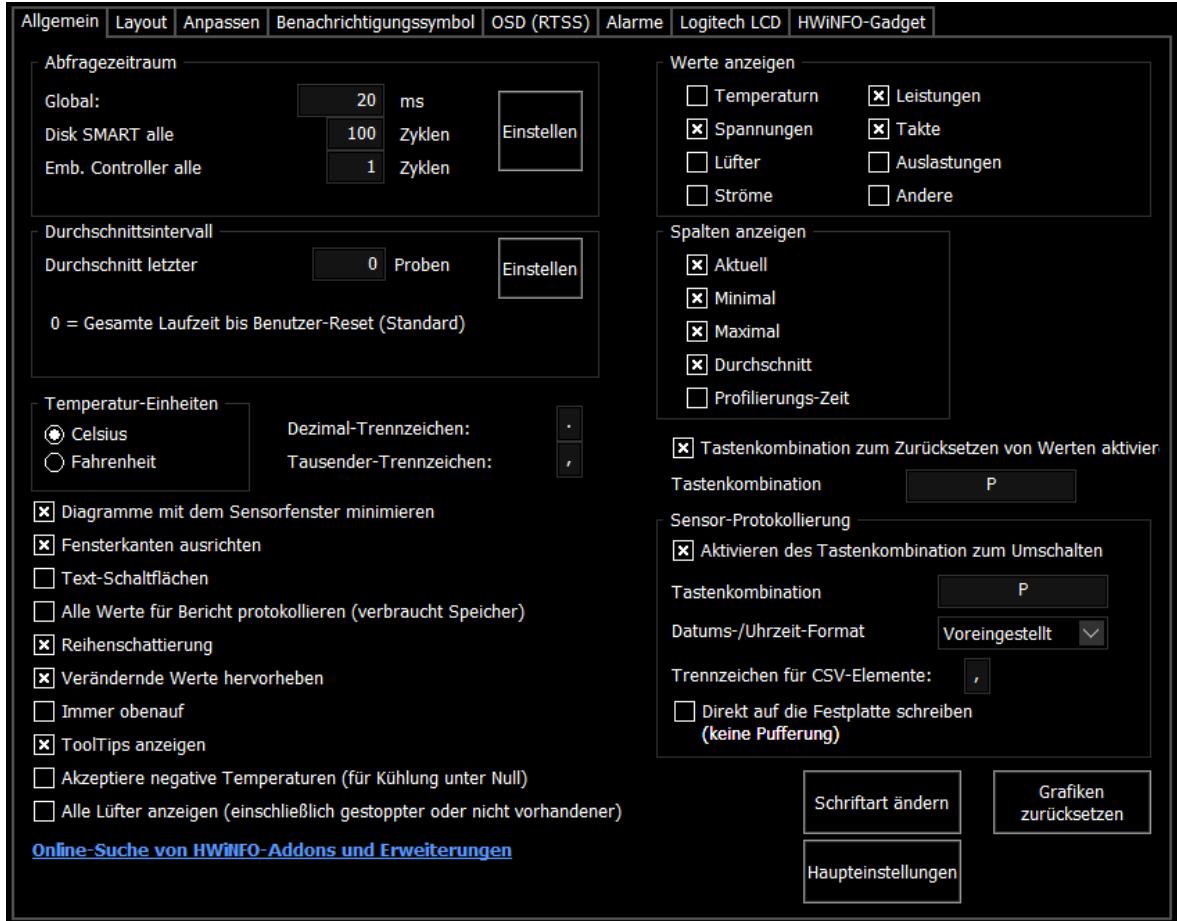


Abbildung 10: Allgemeine Sensoreinstellungen von HWiNFO.

In Bezug darauf gilt es noch zu erwähnen, dass im Layout-Tab angegeben werden kann, ob und welche Messwerte in der Benutzeroberfläche angezeigt und überwacht werden sollen. Hier wurden allerdings keine Änderungen vorgenommen. Des Weiteren kann im Anpassen-Tab festgelegt werden, welche Hardwareinformationen protokolliert werden sollen. Hierbei ist es wichtig, dass die CPU-Gesamt-Leistungsaufnahme, Vcore, und die effektiven Taktraten für die weitere Auswertung protokolliert werden. Diese Informationen sind allerdings standardmäßig aktiviert und erfordern kein eigenständiges Einstellen. In den durchgeföhrten Messungen werden die Cache-Taktraten und einige GPU-Metriken explizit nicht mitprotokolliert. Die Leistungsaufnahme der GPU wird mitprotokolliert. Es können auch noch weitere Einstellungen durchgeföhrt werden, wenn weitere Metriken nicht mitprotokolliert werden sollen. Gründe hierfür könnten mangelnder Speicherplatz, verbesserte Rechnerleistung durch weniger Metriken, bessere Übersichtlichkeit der Daten oder einfach die fehlende Relevanz der Daten sein.

In **Abbildung 11** wird die Einstellungsmöglichkeit für die Protokollierung im Anpassen-Tab aufgezeigt.

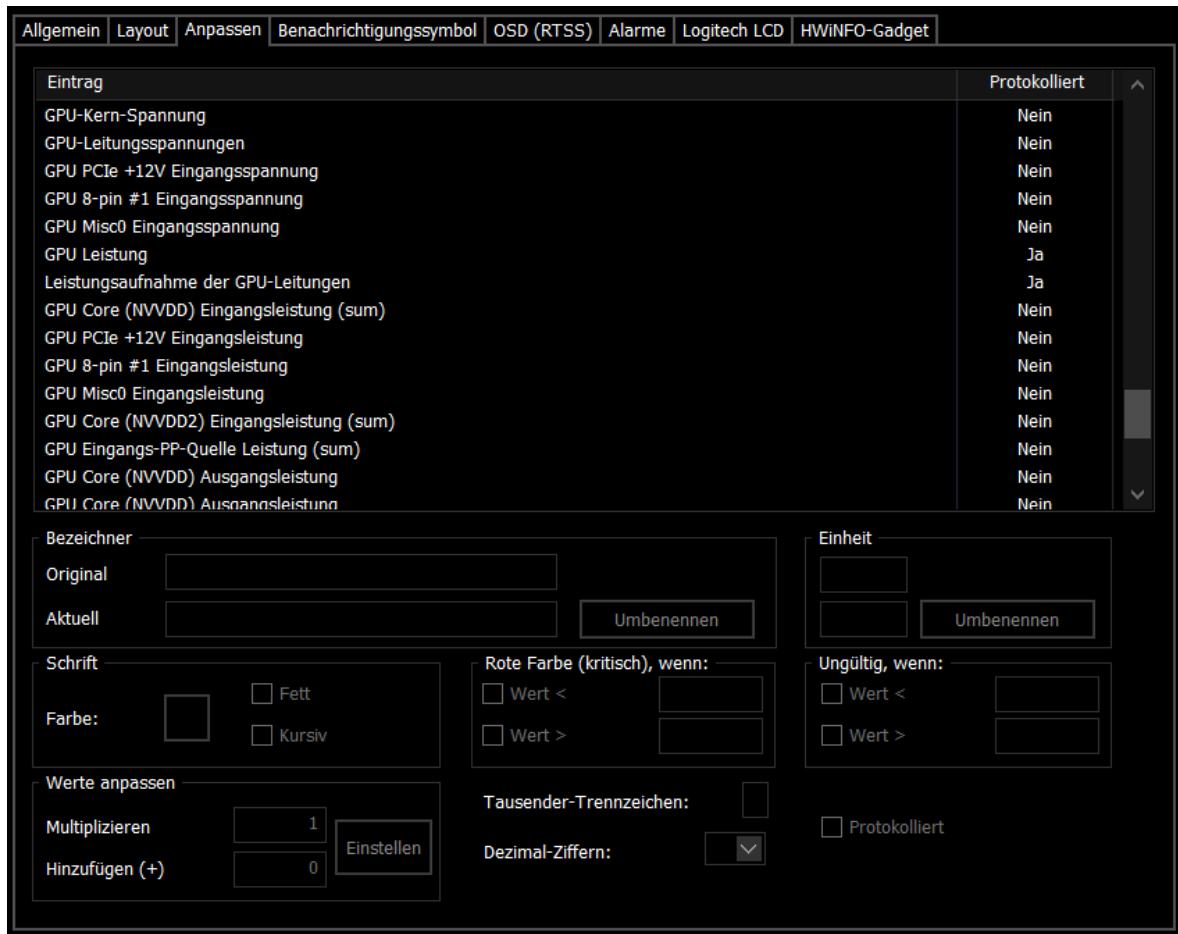


Abbildung 11: Protokollierungseinstellungen von HWiNFO.

Hier sind beispielhaft die GPU-Metriken in einer Liste mit zwei Spalten abgebildet. In der rechten Spalte ist entweder "Ja" oder "Nein" angegeben. Das "Nein" steht dafür, dass diese Metrik nicht mitprotokolliert werden soll. Das "Ja" steht für die protokollierung der Metrik. Diese Einstellung kann man vornehmen, indem man beispielsweise die Metrik "GPU-Kern-Spannung" in der linken Spalte der Liste anklickt und dann unten rechts das Kreuz bei "Protokolliert" setzt.

4.3 Datenanalyse

Nach jedem Training müssen die ersten drei Zellen im Jupyter-Notebook, das für die Auswertung verantwortlich ist, ausgeführt werden. Somit wird für die jeweilige Messung der Energieverbrauch der CPU und der GPU protokolliert. Das Messwerkzeug HWiINFO ermittelt nicht den Energieverbrauch der CPU oder der GPU. HWiINFO misst nur die CPU-Gesamt-Leistungsaufnahme bzw. die GPU-Leistung über die Zeit. Aus diesem Grund muss im Rahmen der Auswertung der Energieverbrauch bei einem Training rechnerisch ermittelt werden. Die Leistungsaufnahme muss über die Zeit integriert werden, da es sich hierbei nicht um eine konstante Leistungsaufnahme handelt, sondern um eine von der Zeit abhängige Leistungsaufnahme, siehe Gleichung 5. Da der Zeitstempel und die CPU-Gesamt-Leistung pro Messpunkt in den CSV-Dateien der HWiINFO-Messung vorhanden sind, kann die Summe der Produkte aus Leistung und Zeit über die Messpunkte gebildet werden, siehe hierfür **Gleichung 8**.

$$E = \sum_{i=0}^n (t_{i+1} - t_i) * \left(\frac{p_i + p_{i+1}}{2} \right) \quad (8)$$

Zwischen den zwei Zeitstempeln t wird die Differenz bzw. das Delta gebildet und mit dem Mittelwert der dazugehörigen Leistungswerte p multipliziert. Diese Werte werden dann über alle Messwerte aufsummiert, woraus sich letztendlich der gesamte Energieverbrauch ergibt.

Zudem werden Darstellungen der Rohdaten für die Messwerte "CPU-Kernfrequenz" und "CPU-Leistungsaufnahme" erzeugt und abgespeichert. Es ist jedoch nicht zwingend notwendig unmittelbar nach einer Messung die eben genannten Auswertungen durchzuführen. Diese können auch erst ausgewertet werden, wenn alle Messungen durchgeführt wurden. Allerdings kann hier nach einer Vielzahl an Messungen schnell die Übersicht verloren gehen und die Auswertung ist anfälliger für menschliche Fehler.

Wurden alle Messungen für einen Datensatz bzw. für beide Datensätze in einem Modell durchgeführt, so können die restlichen Zellen, bis auf die letzten drei, in dem Auswertungs-Jupyter-Notebook ausgeführt werden. Dort werden dann Balkendiagramme erzeugt, die unter anderem beide Datensätze direkt miteinander vergleichen. Ferner wird ein Diagramm für die Genauigkeit, eins für die Trainingsdauer und ein weiteres für den Energieverbrauch erzeugt. Der Energieverbrauch wird dabei nach Gleichung 8 berechnet.

Sobald alle Messungen abgeschlossen und die bisher beschriebenen Auswertungen korrekt durchgeführt wurden, können die letzten drei Auswertungszellen im Jupyter-Notebook ausgeführt werden. Dadurch werden alle Modelle direkt miteinander verglichen. Hierfür werden die Veränderungen in Laufzeit und der Energieverbrauch pro Modell in Balkendiagrammen dargestellt. Zudem wird der prozentuale Anteil des Energieverbrauchs der CPU im Vergleich zur GPU berechnet und dargestellt, um zu ermitteln, wie hoch der Energieverbrauchsanteil der CPU beim Training der neuronalen Netze ist.

Vor jedem Auswertungsschritt sind die Pfade zu den CSV-Dateien und die zusätzlichen Informationen, die in den CSV-Dateien gespeichert werden und die nicht direkt aus der Protokollierung der Messung stammen, an die entsprechende Messung anzupassen. Dazu gehören der Dateiname, die verwendete Frequenz, die Spannung und der Datensatzname. Somit können die Daten korrekt voneinander unterschieden werden. Das Jupyter-Notebook für die Auswertung ist in [77] hinterlegt.

5 Experimente

Im Folgenden wird auf den Messplan eingegangen und die Durchführung der Messungen erklärt.

5.1 Messplan

Die im Rahmen dieser Masterarbeit durchgeführten Versuchsreihen, die den Einfluss auf die Trainingsdauer und den Energieverbrauch untersuchen, werden im Folgenden dargestellt. Jede Versuchsreihe (**Tabelle 1** und **Tabelle 2**) wurde für jeden Datensatz im entsprechenden Modell durchgeführt. Somit wurden pro Modell vier Versuchsreihen ausgeführt, wobei jeweils zwei Versuchsreihen zu Cifar10 und Fashion-MNIST durchgeführt wurden. Um den Einfluss der Verringerung der CPU-Taktfrequenz und der Spannung auf die Trainingsdauer und den Energieverbrauch zu untersuchen, wurden Messungen durchgeführt, bei denen sowohl die Frequenz als auch die Spannung der CPU angepasst wurden, siehe Tabelle 1.

Tabelle 1: Vorgegebene Kombination von Frequenz und Spannung der CPU für die Messungen der ersten Versuchsreihe. Jeweils für Cifar10 und Fashion-MNIST.

Frequenz [MHz]	Spannung [V]
3600	1,4
3000	1,3
2600	1,2
2000	1,1
1600	1,0

Des Weiteren wurden Versuchsreihen durchgeführt, bei denen lediglich die Spannung verringert wurde, während die Frequenz gleich blieb, siehe Tabelle 2. Die Kombination von 3600 MHz und 1,0 V wurde allerdings nicht gemessen, weil sie zu Systemabstürzen führte. Diese Kombination wurde in der Tabelle 2 durchgestrichen.

Tabelle 2: Vorgegebene Kombination von Frequenz und Spannung der CPU für die Messungen der zweiten Versuchsreihe. Jeweils für Cifar10 und Fashion-MNIST.

Frequenz [MHz]	Spannung [V]
3600	1,4
3600	1,3
3600	1,2
3600	1,1
3600	1,0

5.2 Versuchsdurchführung

Zunächst wurden alle notwendigen Programme gestartet. Darunter zählen PyCharm, AMD Ryzen Master und HWiINFO. In der Entwicklungsumgebung PyCharm wurde der Quellcode der Jupyter Notebooks verwaltet und gestartet. AMD Ryzen Master wurde zum Einstellen der CPU genutzt und HWiINFO wurde zum Messen der Metriken beim Trainieren der neuronalen Netze verwendet. Sobald alle notwendigen Programme gestartet wurden, wurde im Programm AMD Ryzen Master die für den aktuellen Versuch korrekte, maximal mögliche CPU-Frequenz pro Kern und Spannung eingestellt. Dazu wurde im Programm AMD Ryzen Master die erweiterte Ansicht ausgewählt, siehe rote Markierung in **Abbildung 12**.

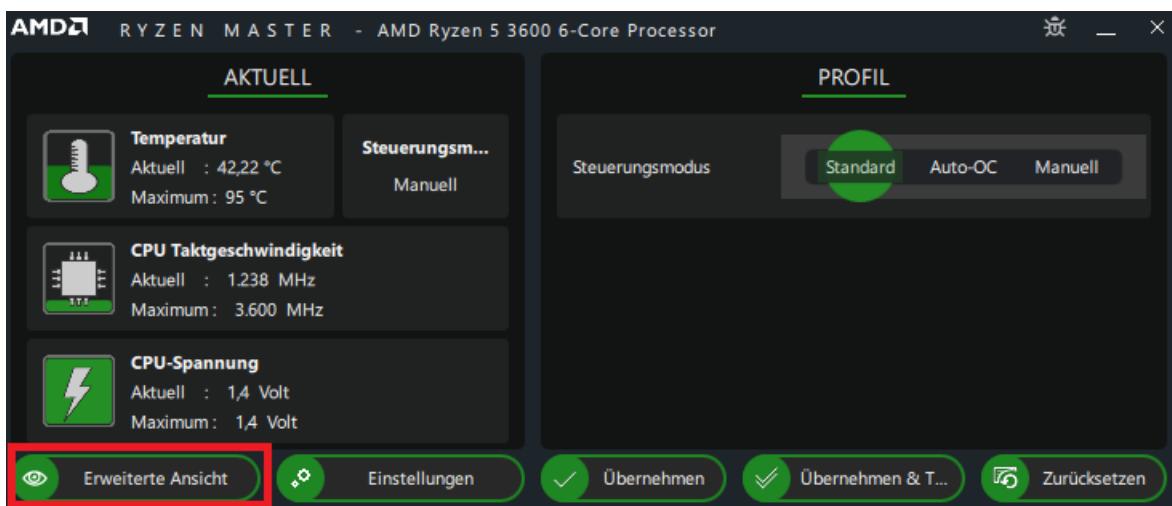


Abbildung 12: Auswahl der erweiterten Ansicht in AMD Ryzen Master.

Anschließend wurde der Tab "Profile 1", der sich oben links in der erweiterten Ansicht befindet, ausgewählt, um die Einstellungen der CPU-Frequenz und CPU-Spannung durchzuführen, siehe rote Markierung in **Abbildung 13**.



Abbildung 13: Auswahl des Tabs "Profile 1" in AMD Ryzen Master.

Im Anschluss daran wurde dafür gesorgt, dass der Steuerungsmodus, der Kern-Abschnitt sowie die Spannungssteuerung enthalten sind. Dafür wurden die Schaltflächen "Enthalten" ausgewählt, siehe rote Markierung **Abbildung 14**. Außerdem wurde der Modus "Manuell" im Steuerungsmodus gewählt, siehe ebenfalls rote Markierung in Abbildung 14. So konnten die maximalen CPU-Frequenzen der einzelnen Kerne und die CPU-Spannung eingestellt werden.

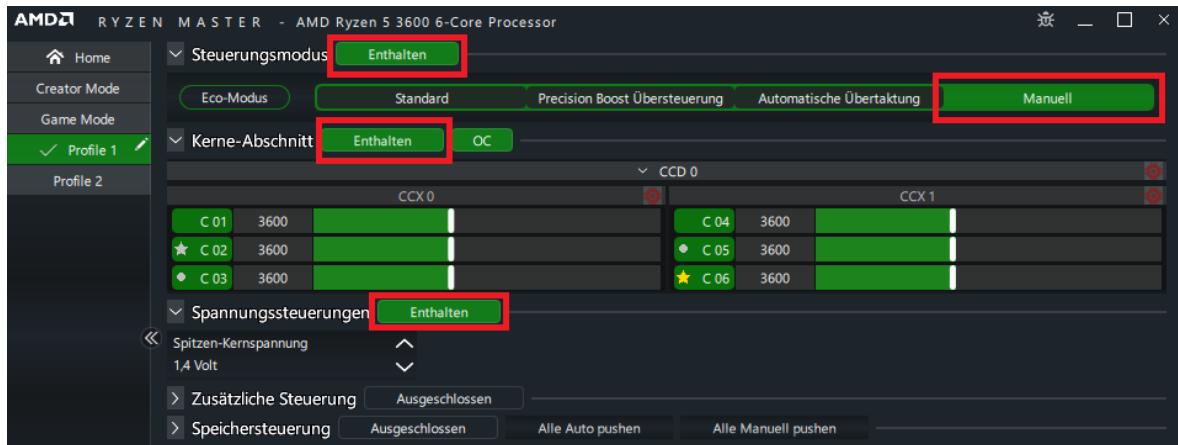


Abbildung 14: Auswahl des Tabs "Profile 1" in AMD Ryzen Master.

Nach jeder Einstellung wurde unten links in der AMD Ryzen Master Anwendung die Schaltfläche "Übernehmen" ausgewählt. Die Einstellungen für die Frequenz und die Spannung wurden für jede Messung durchgeführt.

Sobald die Frequenz und die Spannung der CPU mittels AMD Ryzen Master eingestellt wurden, wurden in PyCharm, im zu messenden Modell, die einzelnen Zellen ausgeführt. Für das Starten des Trainings gibt es zwei Möglichkeiten: Entweder werden alle Zellen automatisch nacheinander mit "Run All" ausgeführt oder jede Zelle wird einzeln mit "Run Cell and Select Below" ausgeführt, siehe **Abbildung 15**. In Abbildung 15 markiert die linke rote Markierung den "Run Cell and Select Below"-Knopf und die rechte rote Markierung den "Run All"-Knopf. Diese Knöpfe sind oben links im Jupyter-Notebook zu finden.

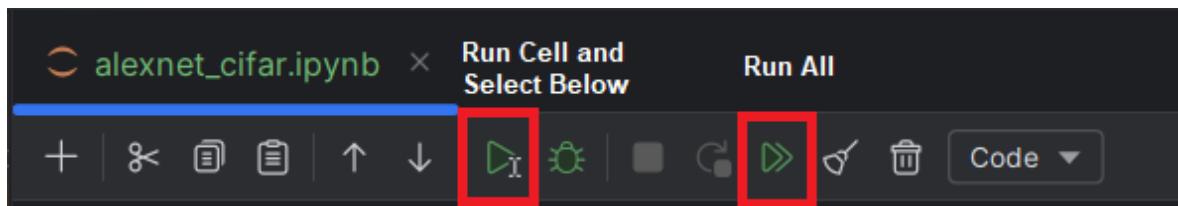


Abbildung 15: Startmöglichkeiten des Trainings im Jupyter-Notebook

Im Rahmen der Versuche dieser Masterarbeit wurden die Zellen mittels des "Run Cell and Select Below"-Knopfs ausgeführt. Dadurch fand eine zusätzliche Kontrolle der Zellen und der darin enthaltenen Konfigurationen statt. Es musste jedoch jede Zelle einzeln gestartet werden. Der Vorteil lag hier einzig und allein in der zusätzlichen Kontrolle vor dem Start der Messung. Sobald die Zelle, die für das Training zuständig ist, ausgeführt wurde, wurde automatisch die Protokollierung mittels HWiNFO gestartet. Das Programm HWiNFO musste dafür lediglich gestartet sein. Die Protokollierung wurde nach dem Training automatisch gestoppt. Wenn das Training mittels "Run All" ausgeführt wird, muss darauf geachtet werden, dass die Pfade und Randbedingungen in der letzten Zelle korrekt angegeben sind, da die Daten sonst falsch abgespeichert werden. Somit empfiehlt sich, diese Daten anzupassen bevor das Training mit "Run All" gestartet wird.

Sobald nun eine Messung eines Modells und dem dazugehörigen Datensatz stattgefunden hat, wurde die Protokollierung entsprechend des Modells, des Datensatzes und der Randbedingungen benannt und für die spätere Auswertung in einem separaten Ordner zwischengespeichert. Anschließend wurde im Programm AMD Ryzen Master wieder die Frequenz und Spannung angepasst und die hier beschriebene Versuchsdurchführung wurde erneut durchlaufen.

6 Ergebnisse

Im Folgenden werden die Ergebnisse aus den Versuchsreihen vorgestellt und entsprechend beschrieben.

6.1 Trainingsdauer

In den **Abbildungen 16 - 21** werden die Ergebnisse zu den verschiedenen Messungen der Trainingszeiten dargestellt. Dabei werden nur ausgewählte Diagramme aufgezeigt, die Besonderheiten aufweisen, da sich sonst gleiche Verhaltensweisen erkennen lassen. Die Vollständigen Ergebnisse zur Trainingsdauer sind im Anhang A.1 zu finden. Die Ergebnisse sind durch gleichzeitige Variation der CPU-Frequenz und die CPU-Spannung sowie durch ausschließliche Variation der CPU-Spannung bei gleichbleibender Frequenz entstanden. Die aufgezeichneten Daten dienen dazu, den Einfluss von CPU-Power-Capping auf die Trainingsdauer und somit auf die Leistungsfähigkeit zu untersuchen.

6.1.1 Trainingsdauer bei Variation von Frequenz und Spannung

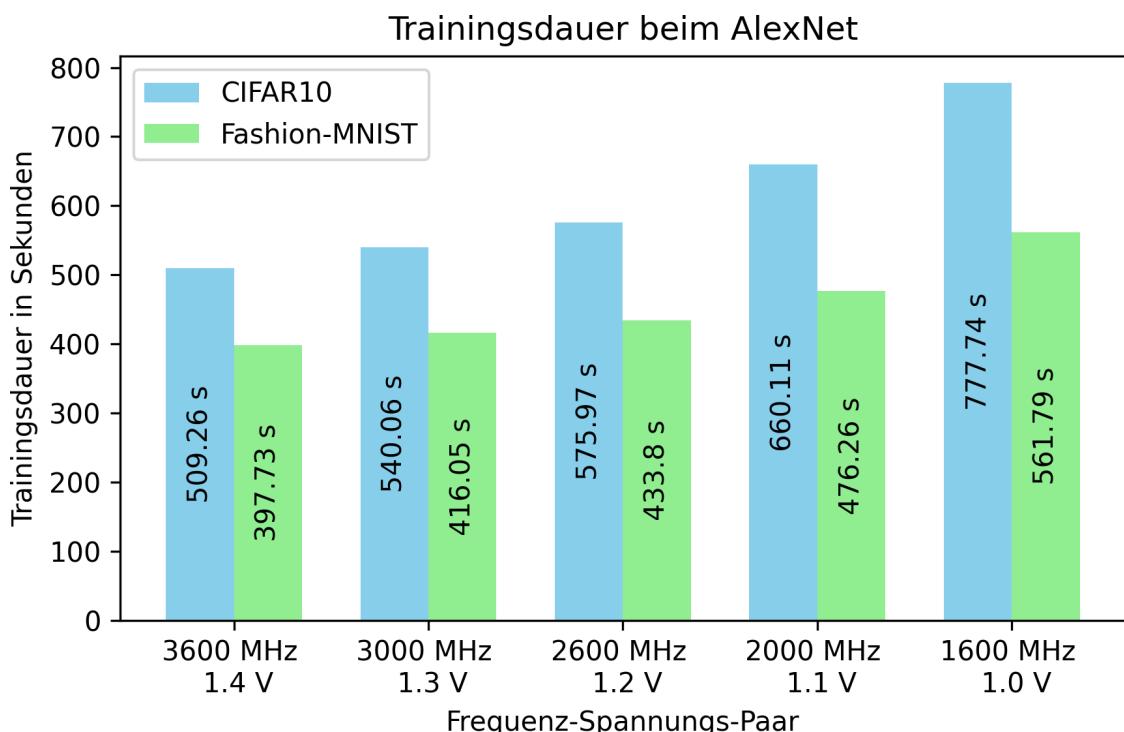


Abbildung 16: Trainingsdauer beim AlexNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In Abbildung 16 sind die Trainingszeiten des Modells AlexNet mit den Datensätzen Cifar10 und Fashion-MNIST dargestellt. Hier ist zu erkennen, dass die Trainingszeit mit dem Verringern der Frequenz und der Spannung sowohl beim Training des Cifar10-Datensatzes als auch beim Fashion-MNIST-Datensatz ansteigt. Dabei ist die Trainingsdauer beim Training des Cifar10-Datensatzes etwas höher als beim Fashion-MNIST-Datensatz. Der Unterschied beträgt hier etwa 112 Sekunden bei 3600 MHz und 1,4 V und etwa 216 Sekunden bei 1600 MHz und 1,0 V. Der relative Unterschied der Datensätze im Bezug auf den Cifar10-Datensatz beträgt bei 3600 MHz etwa 21,90 % und bei 1600 MHz etwa 27,77 %.

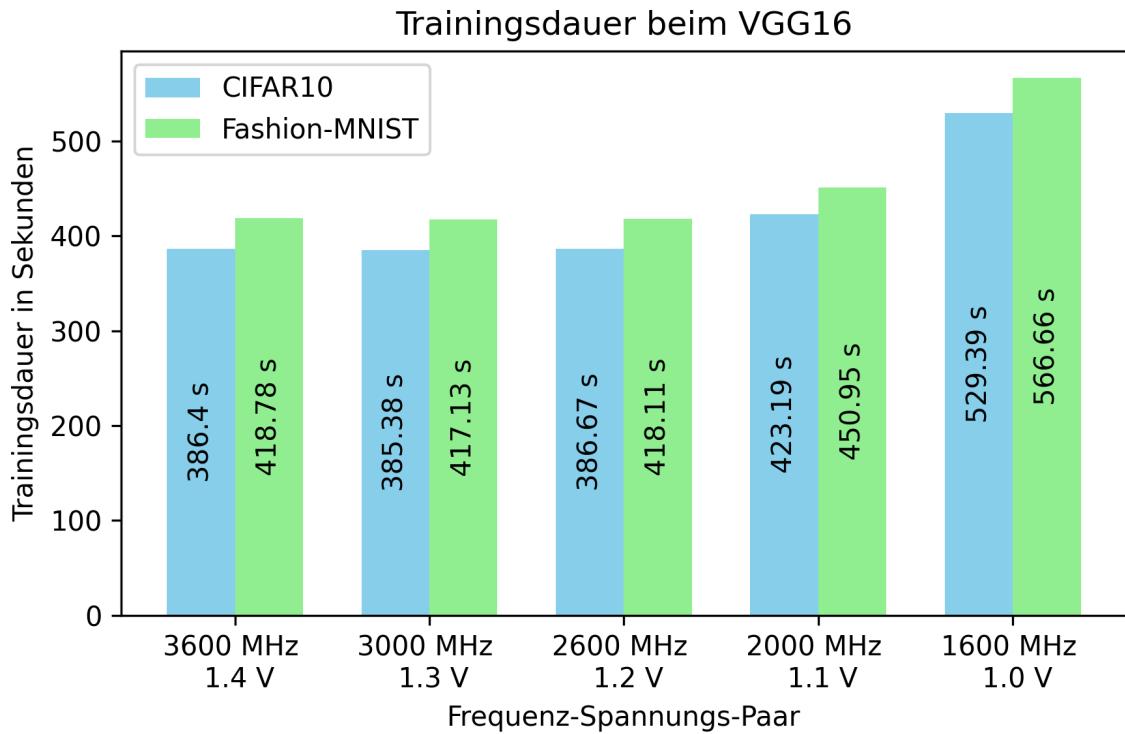


Abbildung 17: Trainingsdauer beim VGG16 bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In Abbildung 17 sind ebenfalls die Trainingszeiten mit den Datensätzen Cifar10 und Fashion-MNIST dargestellt, allerdings hier im VGG16-Modell. Im Vergleich zum AlexNet in Abbildung 16 ist die Trainingsdauer beim Training mit dem Fashion-MNIST minimal höher als beim Training mit dem Cifar10-Datensatz. Der Unterschied beträgt hier etwa 32 Sekunden bei 3600 MHz und 1,4 V und etwa 37 Sekunden bei 1600 MHz und 1,0 V. Der relative Unterschied der Datensätze in Bezug auf den Cifar10-Datensatz beträgt bei 3600 MHz etwa 8,38 % und bei 1600 MHz etwa 7,04 %. Die Trainingszeit steigt hier ebenfalls mit Verringern der CPU-Frequenz und der CPU-Spannung. Im Vergleich mit den Daten aus Abbildung 16 ist zu erkennen, dass die Trainingsdauer bei gleicher Frequenz und Spannung unterschiedlich hoch ist.

Zum Beispiel beträgt die Trainingsdauer des Cifar10-Datensatzes beim AlexNet bei 3600 MHz etwa 509,26 Sekunden, während die Trainingsdauer beim Cifar10-Datensatz im VGG16-Modell nur 386,40 Sekunden beträgt. Abschließend zur Trainingsdauer mit Variation der Frequenz und Spannung ist in Abbildung 18 die Laufzeitveränderung in Prozent von allen Modellen und Datensätzen dargestellt. Hierbei werden die Messwerte der Messungen bei 1,4 V bei 3600 MHz und 1,0 V bei 1600 MHz miteinander verglichen.

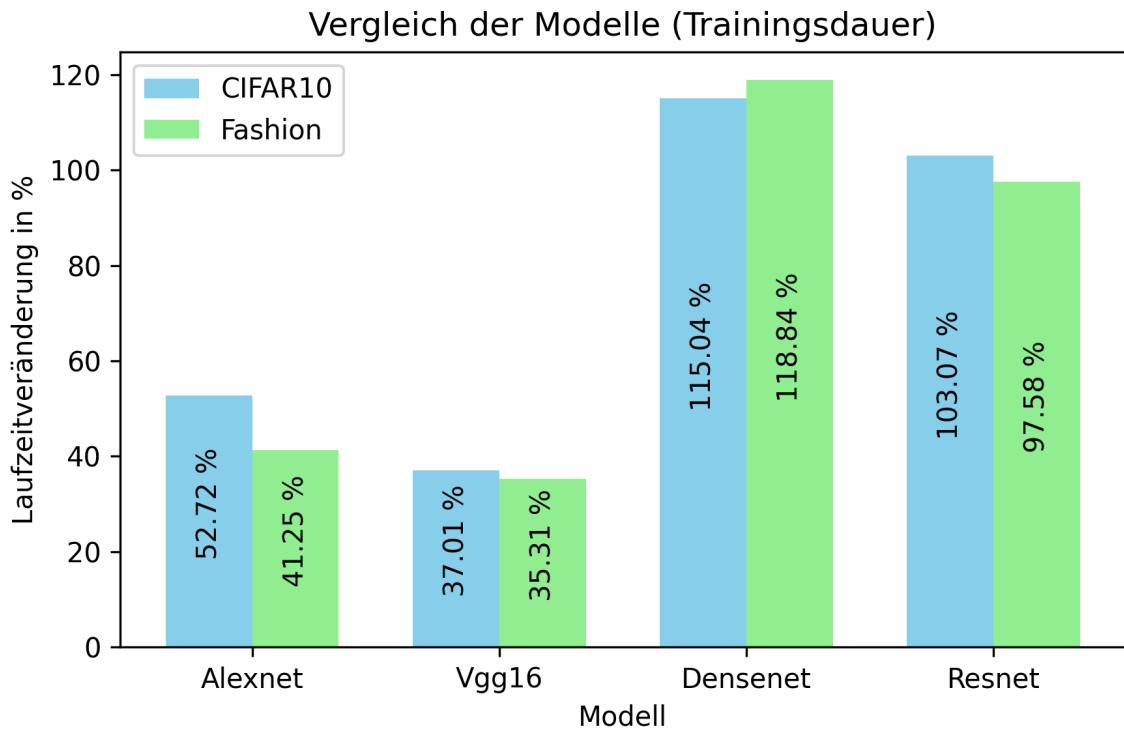


Abbildung 18: Laufzeitveränderungen von allen Modellen und Datensätzen bei Variation der Frequenz und Spannung.

In Abbildung 18 ist erkennbar, dass die Laufzeit des Trainings bei jedem Modell und Datensatz zunimmt. Zudem sind die Laufzeitveränderungen vom AlexNet sowie vom VGG16 im Vergleich zum DenseNet und ResNet relativ gering. AlexNet hat eine Laufzeitzunahme von etwa 52,72 % beim Cifar10-Datensatz und das DenseNet hat eine Laufzeitzunahme beim Cifar10-Datensatz von etwa 115,04 %. Des Weiteren ist die Laufzeitveränderung beim Fashion-MNIST, beim AlexNet, beim VGG16 und beim ResNet im Vergleich zum Cifar10 etwas geringer. Beim DenseNet ist im Gegensatz zu den anderen Modellen die Laufzeitzunahme beim Fashion-MNIST höher als beim Cifar10-Datensatz. Zwischen den beiden Datensätzen liegt ein Unterschied von etwa 3 %. Außerdem ist zu erkennen, dass sich die Laufzeit bei einer Halbierung der CPU-Frequenz nicht verdoppelt. Die CPU-Frequenz wird maximal um den Faktor 2,25 verringert. Der Faktor der maximalen Laufzeitveränderung beträgt aber nur 2,18 beim DenseNet-Modell.

6.1.2 Trainingsdauer bei Variation der Spannung

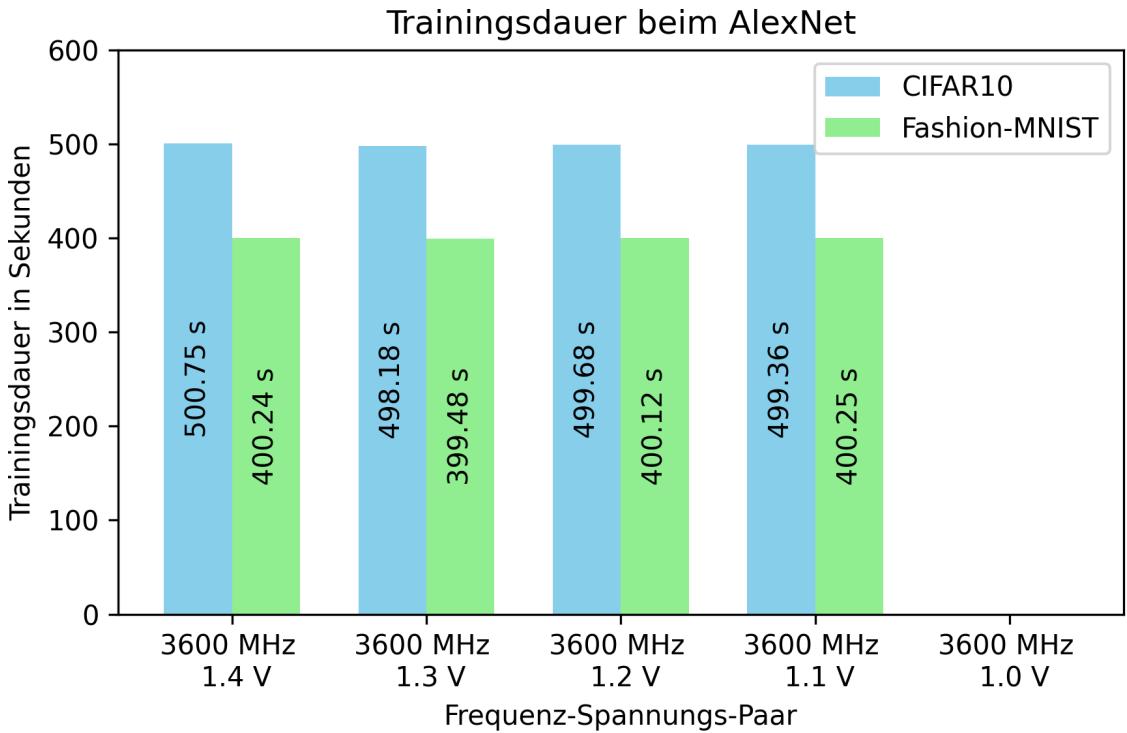


Abbildung 19: Trainingsdauer beim AlexNet bei Anpassung der CPU-Spannung.

In Abbildung 19 sind die Trainingszeiten des AlexNet-Modells mit den Datensätzen Cifar10 und Fashion-MNIST dargestellt. Hier ist zu erkennen, dass die Trainingszeit beim Training des Cifar10-Datensatzes und des Fashion-MNIST-Datensatzes mit Verringern der Spannung nicht essentiell ansteigt. Dabei ist die Trainingsdauer beim Training des Cifar10-Datensatzes um etwa 100 Sekunden höher. Die Trainingszeiten sind nahezu konstant und betragen beim Cifar10-Datensatz um die 500 Sekunden und beim Fashion-MNIST-Datensatz um die 400 Sekunden. Der relative Unterschied der Datensätze im Bezug auf den Cifar10-Datensatz beträgt etwa 20,08 %. Die Messung mit 3600 MHz bei 1,0 V liegt nicht vor, da die Spannungsversorgung für diese Frequenz nicht ausreichend ist.

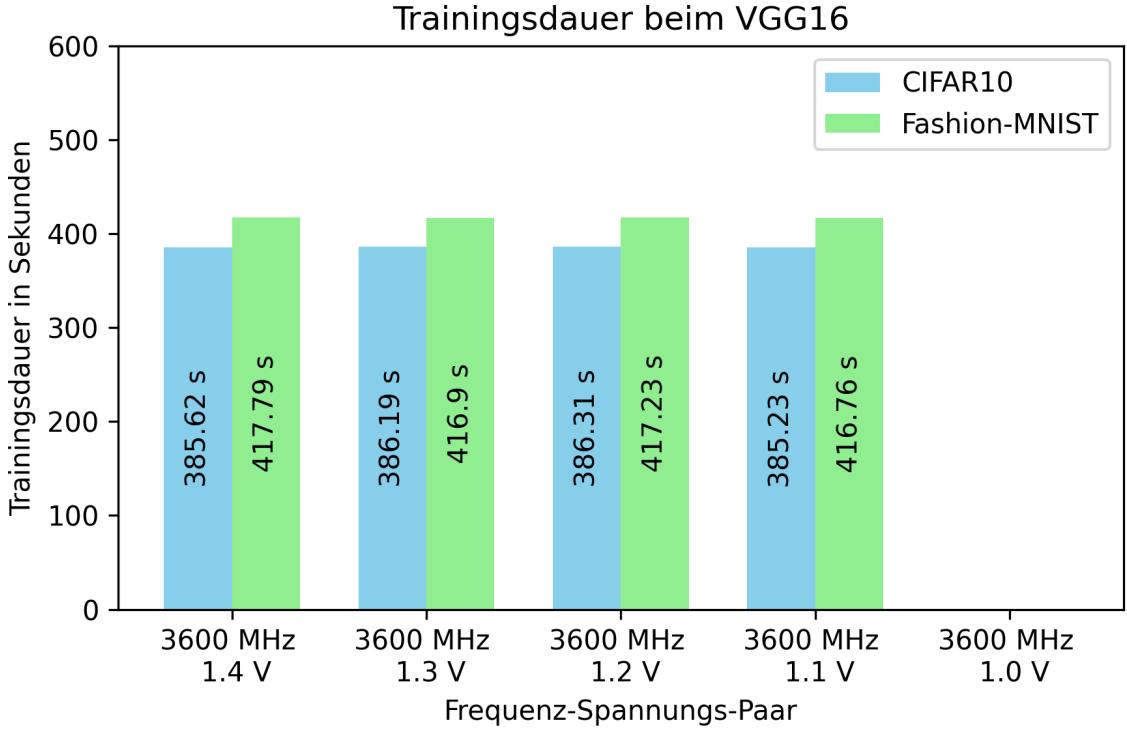


Abbildung 20: Trainingsdauer beim VGG16 bei Anpassung der CPU-Spannung.

In Abbildung 20 sind ebenfalls die Trainingszeiten des Cifar10- und Fashion-MNIST-Datensatzes dargestellt, allerdings hier im VGG16-Modell. Im Vergleich zum Alex-Net aus Abbildung 19 ist die Trainingsdauer beim Training des Fashion-MNIST-Datensatzes etwas höher als beim Training des Cifar10-Datensatzes. Der Unterschied beträgt hier etwa 30-32 Sekunden. Der relative Unterschied der Datensätze in Bezug auf den Cifar10-Datensatz beträgt etwa 5,60 %. Auch hier ist die Trainingszeit nahezu als konstant anzusehen.

Abschließend zur Trainingsdauer, bei ausschließlicher Variation der Spannung, wird in Abbildung 21 die Laufzeitveränderung in Prozent über alle Modelle und Datensätze dargestellt.

In Abbildung 21 ist zu erkennen, dass die Laufzeitveränderung bei jedem Modell und Datensatz zunimmt. Allerdings beträgt diese Laufzeitzunahme deutlich weniger als 1 %. Des Weiteren ist zu erkennen, dass der Cifar10-Datensatz eine höhere Zunahme der Laufzeit aufweist als der Fashion-MNIST-Datensatz. Dabei weist das DenseNet-Modell mit seinen 0,86 % beim Cifar10-Datensatz die größte Zunahme/Schwankung auf.

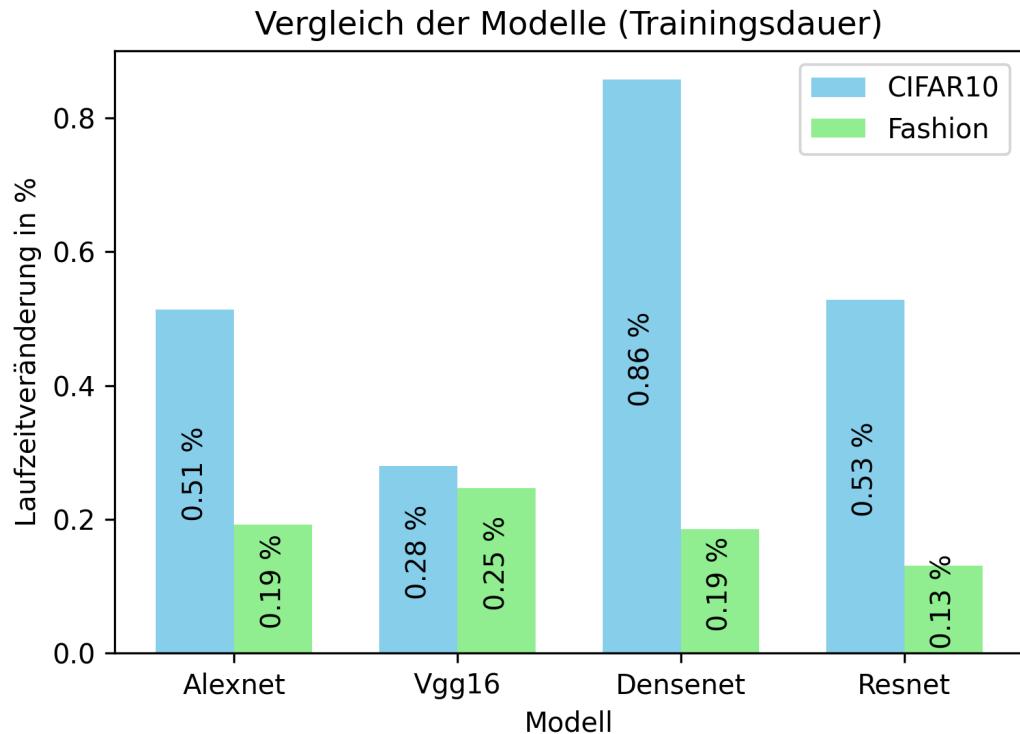


Abbildung 21: Laufzeitveränderungen über alle Modelle und Datensätze bei Variation der Spannung.

6.2 Energieverbrauch

In den **Abbildungen 22 - 28** werden die Ergebnisse zu den verschiedenen Messungen des Energieverbrauchs der CPU dargestellt. Dabei werden nur Diagramme, die Besonderheiten aufweisen, aufgezeigt, da sonst gleiche Verhaltensweisen erkennbar wären. Es handelt sich bei den Ergebnissen um den Energieverbrauch der CPU und nicht um den Energieverbrauch des gesamten Trainings. Vollständige Ergebnisse zum Energieverbrauch und zur Energieeinsparung sind im Anhang A.2 zu finden. Die Diagramme sind durch gleichzeitige Variation der CPU-Frequenz und CPU-Spannung und durch ausschließliche Variation der CPU-Spannung bei gleichbleibender Frequenz entstanden. Die aufgezeichneten Daten dienen dazu, die Auswirkungen von CPU-Power-Capping auf den Energieverbrauch beim Training verschiedener neuronalen Netze zu untersuchen und entsprechend zu bewerten.

6.2.1 Energieverbrauch bei Variation von Frequenz und Spannung

In Abbildung 22 ist der Energieverbrauch der Trainingseinheiten vom AlexNet-Modell mit dem Cifar10-Datensatz und dem Fashion-MNIST-Datensatz bei Variation der CPU-Frequenz und CPU-Spannung dargestellt. Hier ist zu erkennen, dass der Energieverbrauch zunächst, mit Abnahme der maximalen CPU-Frequenz und CPU-Spannung, bis etwa 2600 MHz bei 1,2 V abnimmt und dann bei 2000 MHz bei 1,1 V wieder leicht ansteigt. Des Weiteren ist der Energieverbrauch beim Training mit dem Cifar10-Datensatz höher als beim Training mit dem Fashion-MNIST-Datensatz.

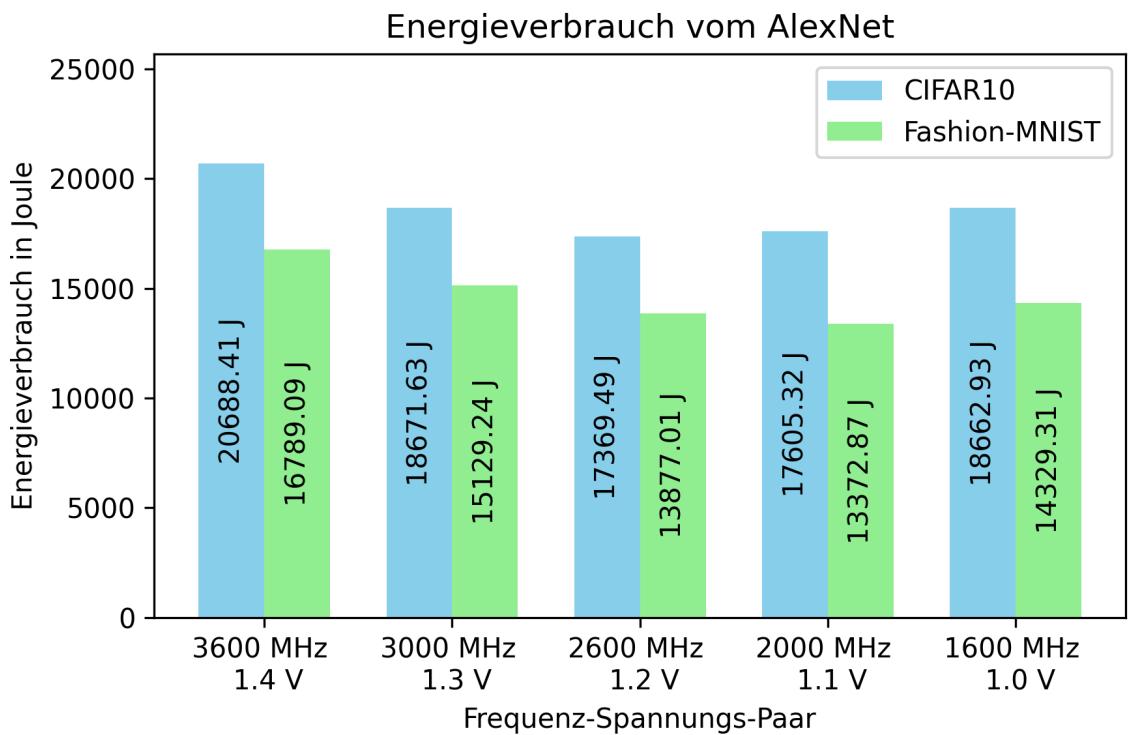


Abbildung 22: Energieverbrauch beim AlexNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In Abbildung 23 ist der Energieverbrauch der Trainingseinheiten vom DenseNet-Modell mit dem Cifar10-Datensatz und dem Fashion-MNIST-Datensatz bei Variation der CPU-Frequenz und CPU-Spannung dargestellt. Im Gegensatz zum AlexNet-Modell, Abbildung 22, ist der Energieverbrauch bei 3600 MHz bei 1,4 V bis 2600 MHz bei 1,2 V nahezu unverändert. Die Werte variieren hier maximal um wenige 100 Joule. Ab 2000 MHz bei 1,1 V ist eine deutliche Steigerung des Energieverbrauchs zu erkennen. Zusätzlich ist zu sehen, dass im Gegensatz zum Training mit dem AlexNet-Modell der Energieverbrauch beim Training des Cifar10-Datensatzes weniger Energie verbraucht als beim Training des Fashion-MNIST-Datensatzes, siehe Abbildungen 22 und 23.

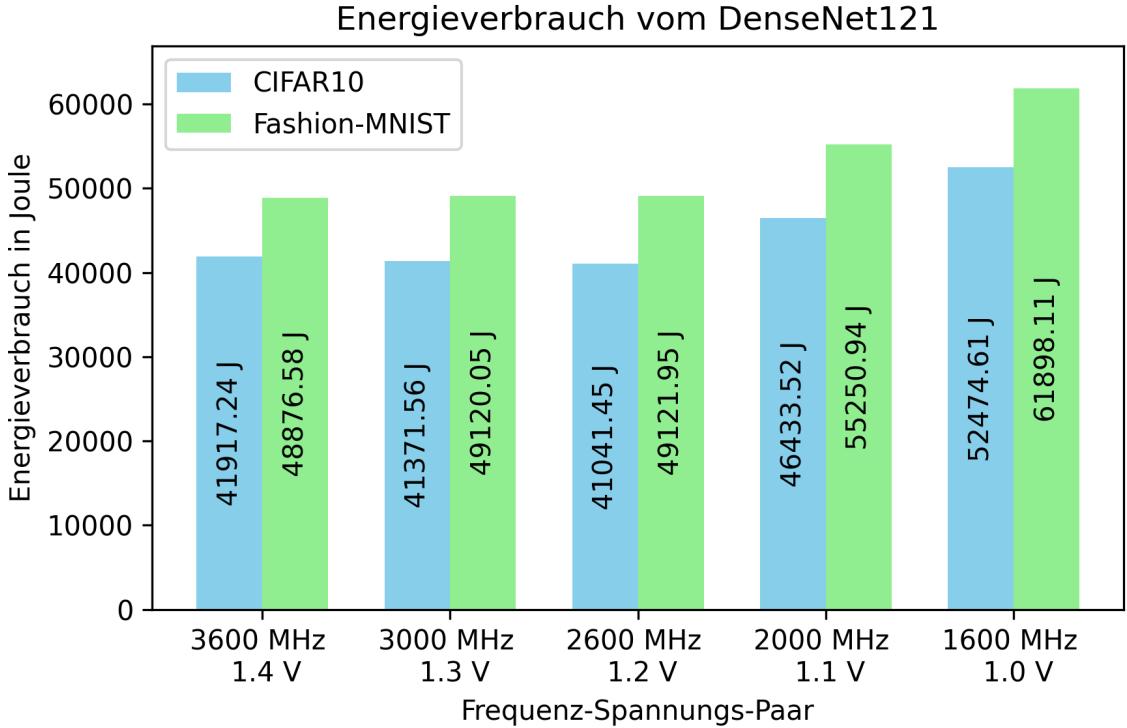


Abbildung 23: Energieverbrauch beim DenseNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In Abbildung 24 ist die maximal mögliche Energieeinsparung in % von jedem Modell bei Variation der CPU-Frequenz und CPU-Spannung dargestellt. Es ist zu erkennen, dass die Energieeinsparungen bei den meisten Modellen zwischen 10 und 26 % liegen. Die Ausnahme bildet hier das DenseNet-Modell, bei dem vergleichsweise nur eine Energieeinsparung von gerade mal 2,09 % erreicht werden konnte. Des Weiteren ist zu erkennen, dass die Energieeinsparung beim Fashion-MNIST-Datensatz etwas höher ist als beim Cifar10-Datensatz. Die Ausnahme bildet hier ebenfalls das DenseNet-Modell, bei dem ausschließlich beim Cifar10-Datensatz eine geringe Energieeinsparung erreicht wurde. Außerdem zeigen die Daten, dass die Energieeinsparung je nach Modell unterschiedlich hoch ist.

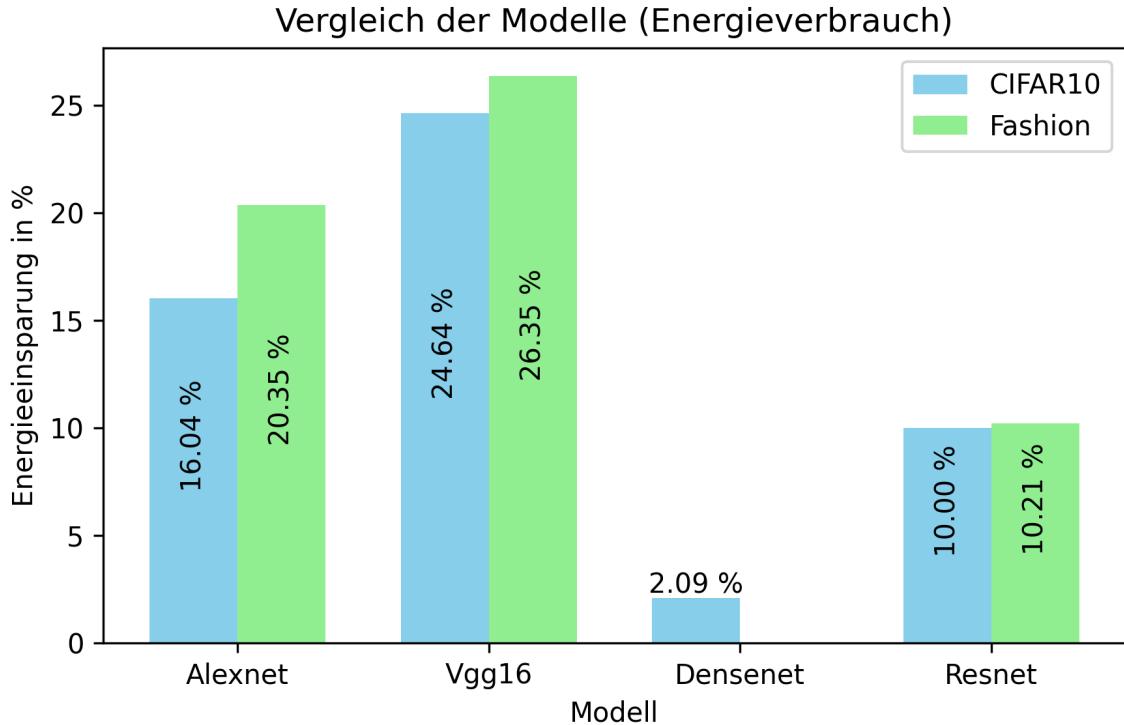


Abbildung 24: Energieeinsparung beim Training der verschiedenen Modelle bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

6.2.2 Energieverbrauch bei Variation der Spannung

In Abbildung 25 ist der Energieverbrauch vom AlexNet-Modell bei ausschließlicher Anpassung der CPU-Spannung dargestellt. Hier ist zu erkennen, dass sich der Energieverbrauch mit Absenken der CPU-Spannung deutlich verringert. Zudem ist der Energieverbrauch beim Training mit dem Cifar10-Datensatz höher als mit dem Fashion-MNIST-Datensatz. Die Messung bei 3600 MHz und 1,0 V war, wie bereits in Kapitel 5.1 erläutert, nicht möglich.

In Abbildung 26 ist der Energieverbrauch vom DenseNet-Modell bei ausschließlicher Anpassung der CPU-Spannung dargestellt. Hier ist ebenfalls zu erkennen, dass sich der Energieverbrauch mit Absenken der CPU-Spannung verringert. Im Gegensatz zum AlexNet-Modell, siehe Abbildung 25, ist hier der errechnete Energieverbrauch beim Fashion-MNIST-Datensatz höher als beim Cifar10-Datensatz. Auch in Abbildung 26 ist die Messung bei 3600 MHz und 1,0 V nicht vorhanden.

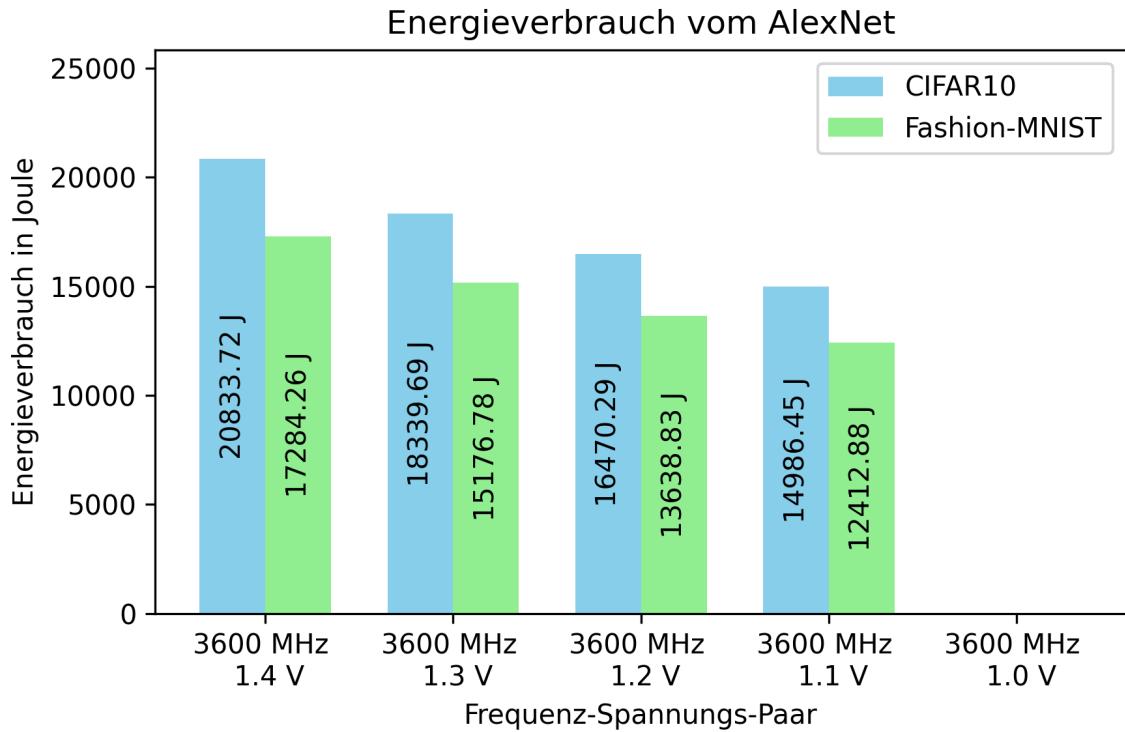


Abbildung 25: Energieverbrauch beim AlexNet bei Anpassung der CPU-Spannung.

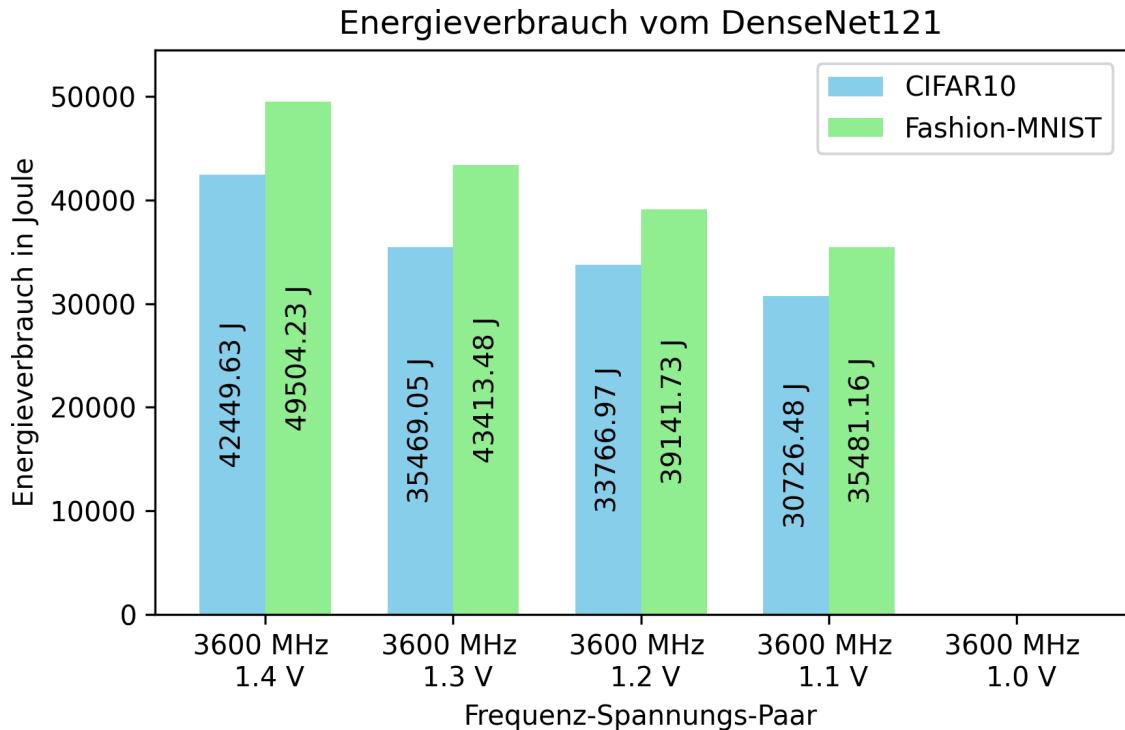


Abbildung 26: Energieverbrauch beim DenseNet bei Anpassung der CPU-Spannung.

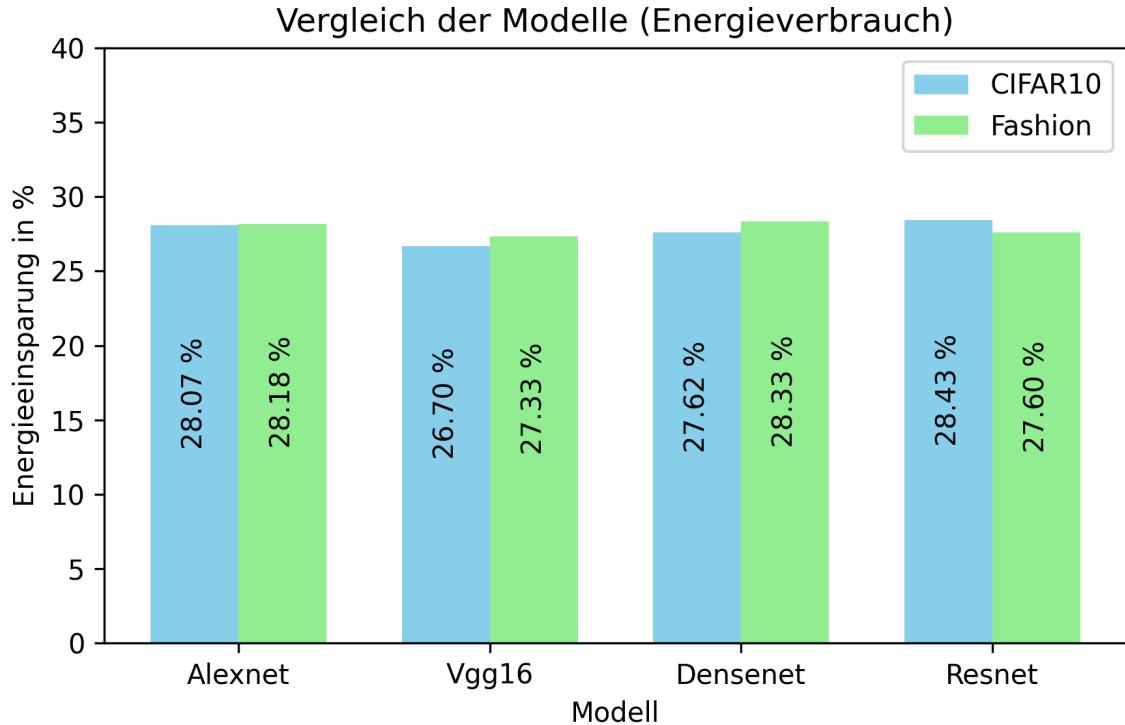


Abbildung 27: Energieeinsparung beim Training der verschiedenen Modelle bei Anpassung der CPU-Spannung.

In Abbildung 27 ist die maximal mögliche Energieeinsparung der verschiedenen Modelle bei ausschließlicher Anpassung der CPU-Spannung in % dargestellt. Es ist zu erkennen, dass die Energieeinsparung bei allen Modellen zwischen 26 und 29 % liegt. Dabei sind die Energieeinsparungen pro Datensatz nahezu identisch, wobei nur sehr geringe Unterschiede von weniger als 1 % zu sehen sind. Die durchschnittliche relative Anpassung der Energieeinsparung über die verschiedenen Modelle beträgt etwa 2,06 %. Trotz der sehr geringen Variation des Energieverbrauchs zwischen den Datensätzen hat der Fashion-MNIST-Datensatz, bis auf das ResNet-Modell, minimal mehr Energie eingespart.

6.2.3 CPU-Energieverbrauchsanteil

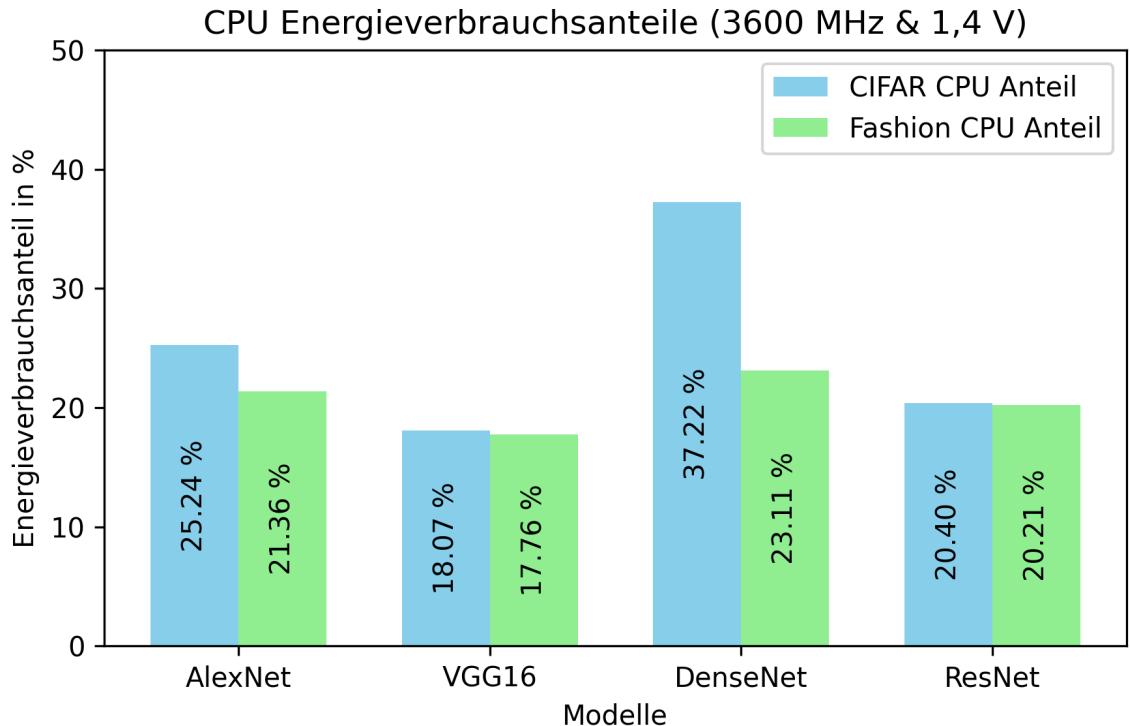


Abbildung 28: CPU-Energieverbrauchsanteile beim Training der verschiedenen Modelle bei einer Taktfrequenz von 3600 MHz und einer Versorgungsspannung von 1,4 V.

In Abbildung 28 ist der Energieverbrauchsanteil der CPU in % bei einer maximalen CPU-Taktfrequenz von 3600 MHz und einer CPU-Versorgungsspannung von 1,4 V dargestellt. Ein weiteres Bauteil, das beim Training einen großen Teil des Energieverbrauchs ausmacht, ist die GPU. Erkennbar ist, dass die CPU-Anteile je nach Modell variieren und dass, je nachdem welcher Datensatz trainiert wurde, die Anteile innerhalb des Modells eine leichte Varianz aufweisen. So wird deutlich, dass der Fashion-MNIST-Datensatz weniger CPU-Energieverbrauchsanteile hat als der Cifar10-Datensatz, wobei beim VGG16-Modell und dem ResNet-Modell ein Unterschied von weniger als 1 % zwischen dem Cifar10-Datensatz und dem Fashion-MNIST-Datensatz vorliegt. Das DenseNet-Modell hat den höchsten CPU-Energieverbrauchsanteil.

6.3 Durchschnittliche Kernfrequenz

In Abbildung 29 werden die durchschnittlichen effektiven Kernfrequenzen der sechs genutzten CPU-Kerne bei gleichbleibender maximaler CPU-Frequenz und variabler CPU-Spannung dargestellt. Diese Auswertung wurde beim ResNet-Modell mit dem Cifar10-Datensatz durchgeführt, da in diesem Fall am meisten Energie eingespart wurde, siehe Abbildung 27. Diese Daten dienen zur Erklärung der Energieeinsparungen bei den Versuchen, bei denen ausschließlich die CPU-Spannung angepasst wurde.

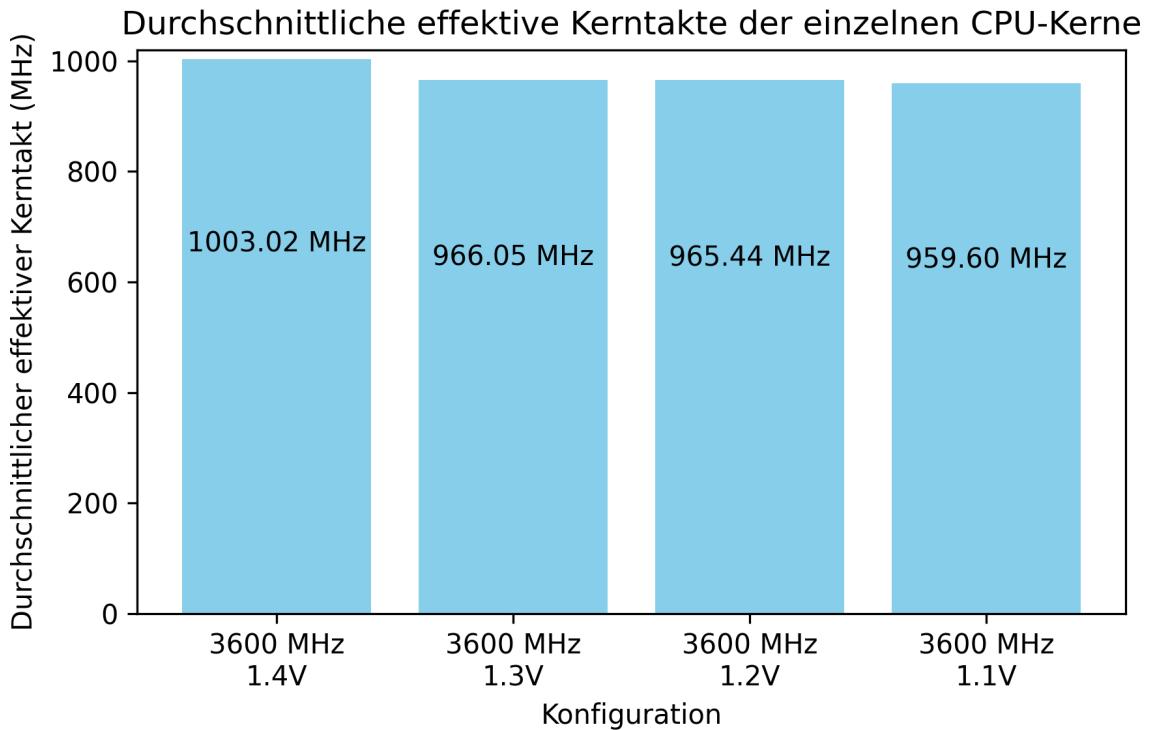


Abbildung 29: Durchschnittliche effektive Kerntakt der CPU Kerne bei einer maximalen CPU-Frequenz von 3600 MHz und variabler Spannung.

In Abbildung 29 ist eine nahezu gleichbleibende durchschnittliche effektive Kernfrequenz zu erkennen. Die Abweichungen zwischen den Messungen mit unterschiedlicher Konfiguration sind von 1,4 V auf 1,3 V vergleichsweise hoch (3,69 %), allerdings sind die weiteren Unterschiede zwischen den Messungen kleiner als 1 %. Die Verringerung der effektiven Kernfrequenz beträgt nach der Messung mit 1,3 V nur wenige MHz. Hier wird auch deutlich, dass die durchschnittliche, effektive Kernfrequenz deutlich unterhalb der maximal eingestellten CPU-Frequenz liegt. Es wird zwar für alle sechs verwendeten Kerne eine maximale Kernfrequenz von beispielsweise 3600 MHz eingestellt, aber die tatsächliche Kernfrequenz in den einzelnen CPU-Kernen kann geringer sein, siehe auch Abbildung 30. Hier ist zu erkennen, dass die Kernfrequenzen unterschiedlich sein können, da die CPU ihre Kerne einzeln regeln kann.

So ist zum Beispiel eine minimale Frequenz, wie in Abbildung 30 möglich, wodurch sich die durchschnittlichen, effektiven Kernfrequenzen aus Abbildung 29 ergeben.

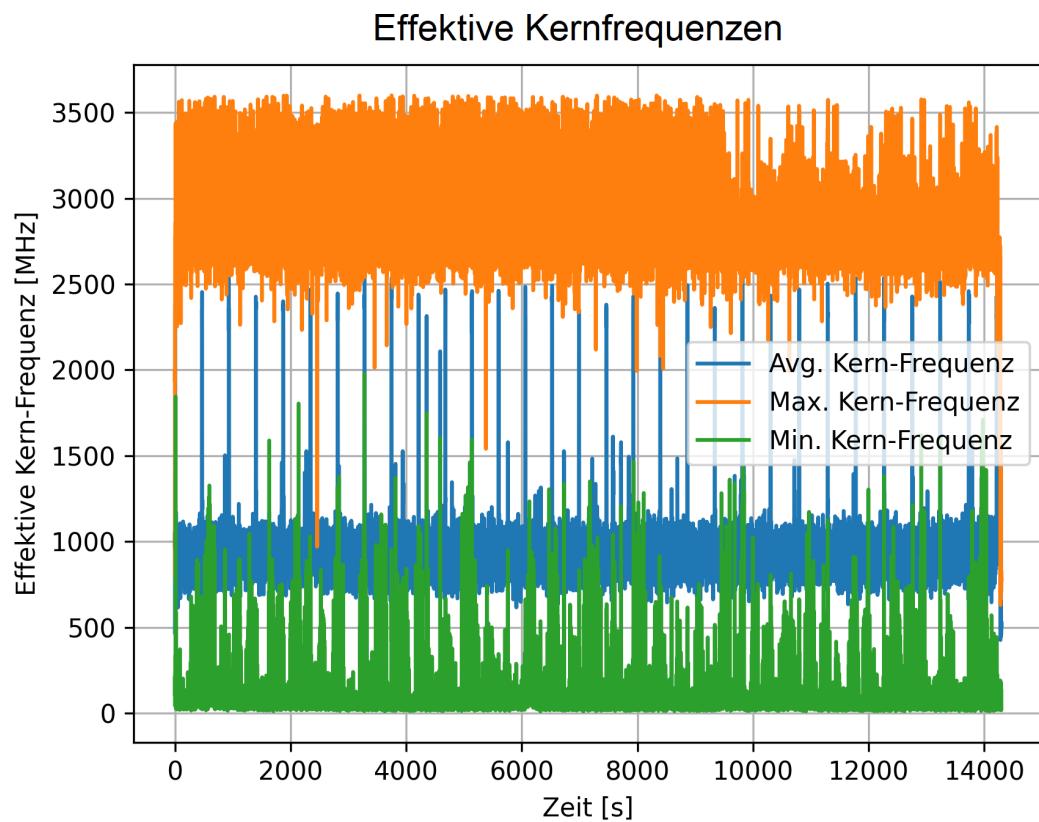


Abbildung 30: Durchschnittliche, maximale und minimale effektive Kernfrequenz der CPU Kerne bei einer maximalen CPU-Frequenz von 3600 MHz und 1,1 V beim ResNet-Modell.

6.4 Genauigkeit

In der Abbildung 31 werden die Genauigkeiten des ResNet-Modells bei unterschiedlichen maximalen CPU-Frequenzen und variablen CPU-Spannungen dargestellt. Die Genauigkeiten der anderen Modelle können dem Anhang A.3 entnommen werden, wobei bei allen Modellen dasselbe Verhalten zu erkennen ist. Die aufgezeichneten Daten dienen dazu, die Auswirkungen von CPU-Power-Capping auf die Genauigkeit der Modelle und somit auf die Leistungsfähigkeit der Modelle zu untersuchen und entsprechend zu bewerten.

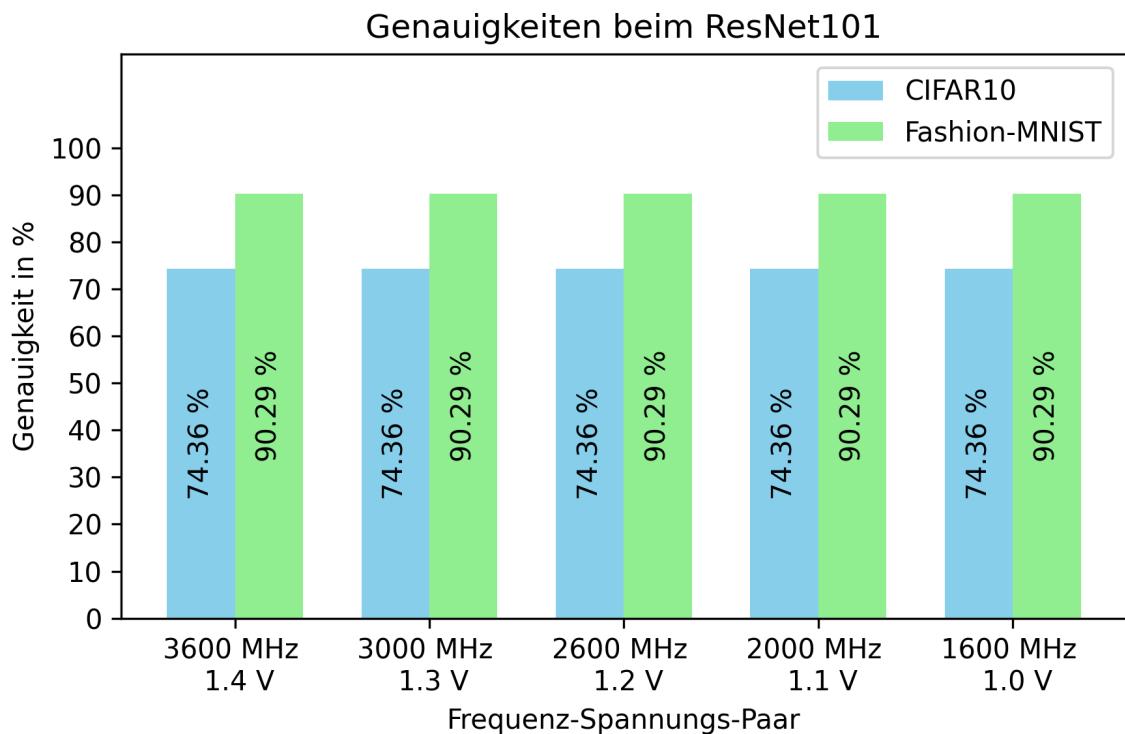


Abbildung 31: Genauigkeiten beim ResNet-Modell bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.

In Abbildung 31 ist zu erkennen, dass die Genauigkeiten beim Verringern der maximalen CPU-Frequenz und CPU-Spannung unverändert bleiben und damit konstant sind. Des Weiteren ist zu sehen, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz 74,36 % und beim Training mit dem Fashion-MNIST-Datensatz 90,29 %.

7 Diskussion

Im Folgenden werden die bereits vorgestellten Ergebnisse aus den Versuchsreihen interpretiert, diskutiert, miteinander verglichen und geschlussfolgert.

7.1 Betrachtung der Trainingsdauer

Die Datenlage zeigt, dass die Trainingsdauer beim Verringern der CPU-Frequenz stetig steigt. Der Grund dafür ist, dass die CPU-Frequenz bestimmt, wie viele Anweisungen die CPU pro Sekunde ausführen kann. Eine niedrigere Frequenz bedeutet also, dass die CPU weniger Anweisungen pro Sekunde verarbeiten kann, was direkt eine längere Verarbeitungszeit zur Folge hat. Dieses Verhalten ist auch in Abbildung 9 dargestellt. Eine Ausnahme in diesem Verhalten bildet die Datenlage des VGG16-Modells, siehe Abbildung 17. Hier bleiben die Trainingszeiten bis zu einer Frequenz von 2600 MHz nahezu unverändert, obwohl die CPU-Frequenz verringert wird. Aus dieser Abbildung lässt sich erkennen, dass die CPU-Frequenz zunächst nicht der limitierende Faktor ist und die CPU die anfallenden Aufgaben im Bereich von 3600 MHz und 2600 MHz ausreichend schnell, ohne Engpässe, ausführen kann. Da dieses Verhalten bei den anderen Modellen nicht beobachtet werden konnte, liegt die Vermutung nahe, dass dieses Verhalten auf die Architektur des VGG16-Modells zurückzuführen ist, sodass die Trainingszeiten bei einer Verringerung der CPU-Frequenz zunächst nahezu konstant bleiben.

Die GPU übernimmt die meisten rechenintensiven Aufgaben und in der Zeit in der die GPU das Training ausführt, ist die CPU nicht mit dem Training beschäftigt und wartet darauf, die nächsten Daten eines Batches zur GPU zu senden. Wenn nun die Batchzeit sehr hoch ist, kann dies zu einer langen Wartezeit der CPU führen und erst bei einer deutlichen Reduktion der CPU-Frequenz werden die unterstützenden Aufgaben der CPU zu einem Flaschenhals und die Trainingszeiten verlängern sich entsprechend. Es kann an dieser Stelle gesagt werden, dass eine niedrigere CPU-Frequenz zu einer längeren Verarbeitungszeit führt.

Weiterhin bildet das AlexNet-Modell eine Ausnahme in den Daten. Denn dieses Modell ist das einzige gemessene Modell, das eine höhere Trainingsdauer beim Training des Cifar10-Datensatzes vorweist als beim Training des Fashion-MNIST-Datensatzes, siehe Abbildung 16. Der Grund liegt hier im Aufbau des AlexNet-Modells. Obwohl das AlexNet-Modell tiefer ist als andere Netzwerke, ist es im Vergleich zu modernen Architekturen, wie dem VGG16-Modell oder auch dem ResNet-Modell, relativ flach und weniger komplex. Dies kann dazu führen, dass es bei vergleichsweise komplexeren Datensätzen wie Cifar10 weniger effizient ist.

Andere Modelle wie VGG16 sind besser auf die spezifischen Eigenschaften vom Cifar10 abgestimmt und können daher effizienter trainieren.

Die signifikanten Laufzeitverlängerungen beim DenseNet-Modell und beim ResNet-Modell im Vergleich zum AlexNet-Modell und dem VGG16-Modell bei einer Reduktion der CPU-Frequenz können auf die unterschiedliche Komplexität der Modellarchitekturen zurückgeführt werden. Jedes Modell benötigt unterschiedlich viel Zeit, um die zur GPU gesendeten Daten zu verarbeiten, was anhand der unterschiedlichen Batchzeiten erklärt werden kann, siehe Kapitel 4.2.1. Demnach können komplexe Modelle wie das DenseNet-Modell und das ResNet-Modell, die eine vergleichsweise hohe Batchzeit (50-75 ms) aufweisen, die GPU effizienter auslasten, da die GPU länger gebraucht wird. Dadurch sind diese Modelle anfälliger für Verzögerungen durch CPU-Engpässe [83], was zu einer stärkeren Zunahme der Trainingsdauer führt. Im Gegensatz dazu sind die einfacheren Modelle, wie AlexNet und VGG16, weniger effizient in der GPU-Auslastung, da diese eine deutlich geringere Batchzeit (3-8 ms) und somit auch eine geringere Laufzeitzunahme haben. Dieses Verhalten wird in Abbildung 18 dargestellt. An dieser Stelle wären Untersuchungen der CPU- und GPU-Auslastung bei unterschiedlichen Modellen bei gleicher Batchgröße oder sogar variabler Batchgröße durchaus interessant, da sie im Rahmen dieser Masterarbeit den Rahmen gesprengt hätten. Diese Untersuchungen könnten interessante Ergebnisse liefern und diese Vermutung bestätigen.

Bei den Versuchen, bei denen ausschließlich die Spannung verringert wurde, zeigt die Datenlage eine nahezu konstante Trainingsdauer, siehe Abbildung 19 und 20. Weil die CPU-Frequenz nicht angepasst wurde, hat die CPU trotz der Verringerung der Spannung weiterhin ausreichend Versorgungsspannung, um alle sechs CPU-Kerne versorgen zu können. Wodurch die Leistung vorhanden ist, um die Trainingseinheiten nahezu immer zur selben Zeit auszuführen, siehe hierzu Abbildung 29. Auch hier weist das AlexNet-Modell eine höhere Trainingsdauer beim Cifar10-Datensatz auf. An dieser Stelle gilt die gleiche Erklärung wie bei den Versuchen mit der angepassten CPU-Frequenz. In Abbildung 21 ist ebenfalls zu erkennen, dass die maximale Trainingszeitzunahme gerade mal 0,86 % beträgt und damit als nahezu konstant angesehen werden kann. Abschließend lässt sich hieraus erschließen, dass eine Reduktion der Spannung, ohne Anpassung der Frequenz, keinen signifikanten Einfluss auf die Trainingsdauer hat.

7.2 Betrachtung des Energieverbrauchs

Die Datenbasis zum Energieverbrauch zeigt, dass der Energieverbrauch mit sinkender Frequenz und Spannung zunächst abfällt und dann wieder ansteigt, siehe Abbildung 22. Dieses Verhalten zeigt sich beim AlexNet-Modell, beim ResNet-Modell und beim VGG16-Modell. Der Grund für den zunächst geringeren Energieverbrauch ist auf den Zusammenhang zwischen der Leistung, der Frequenz und der Spannung aus Gleichung 4 zurückzuführen.

Im Kapitel 6.1.2 wurde bereits erwähnt, dass sich die Ausführungszeit beim Verringern der Frequenz erhöht. Wenn dabei die Frequenz der CPU zu stark reduziert wird, verlängert sich die Zeit, die die CPU braucht, um die anfallenden Aufgaben abzuarbeiten. Außerdem sind die CPUs für einen Betrieb in einem bestimmten Frequenz- und Spannungsbereich optimiert. Wenn nun die Kombination der maximalen Frequenz und Spannung nicht für einen optimalen Betrieb der CPU geeignet ist, kann die Effizienz der CPU sinken. Dies führt dazu, dass die CPU für die gleiche Aufgabe mehr Energie verbraucht. So ist zum Beispiel der Messwert bei 3600 MHz bei 1,0 V nicht vorhanden, da dort die CPU unter Last nicht ausreichend mit Spannung versorgt werden kann. Daraus lässt sich schließen, dass Energie eingespart werden kann, wenn die Frequenz und Spannung moderat verringert wird, wobei auch hier das Optimum durch verschiedene Experimente gefunden werden muss. Eine zu starke Reduktion der CPU-Frequenz und -Spannung verschlechtert die Energieeffizienz der CPU und führt zur Steigerung des Energieverbrauchs.

Das DenseNet-Modell zeigt im Vergleich zu den anderen Modellen ein anderes Verhalten beim Energieverbrauch bei sinkender Frequenz und Spannung, siehe Abbildung 23. Hier bleibt der Energieverbrauch zunächst nahezu konstant und steigt dann im Verlauf immer weiter an. Wobei beim Cifar10-Datensatz zunächst etwa 2,06 % eingespart werden können. Diese Einparungen machen bei einem CPU-Energieverbrauchsanteil von 37,22 % etwa 5,53 % des ursprünglichen CPU-Energieverbrauchs aus. In den vorherigen Kapiteln wurde bereits der Grund des starken Anstiegs der Laufzeit beim DenseNet-Modell erklärt. Durch diesen Anstieg wird die verringerte Leistungsaufnahme wieder ausgeglichen. Wodurch letztendlich das beobachtete Verhalten beim DenseNet-Modell zustande kommt. Hieraus lässt sich ableiten, dass die Energieverbrauchsverläufe eines Modells und damit das Optimum der CPU-Frequenz und -Spannung vom verwendeten neuronalen Netzwerk abhängig sind.

Das Training mit dem Cifar10-Datensatz beim AlexNet-Modell verbraucht im Vergleich zum Fashion-MNIST-Datensatz mehr Energie. Bei den anderen Modellen verbraucht hingegen der Fashion-MNIST-Datensatz mehr Energie, da diese besser für komplexe-re Datensätze optimiert sind. Das AlexNet-Modell ist bei vergleichsweise komplexen Datensätzen wie Cifar10 weniger effizient. Dadurch erhöht sich die Laufzeit und der erhöhte Energieverbrauch kommt zustande. Die Modellarchitektur spielt also eine wichtige Rolle beim Energieverbrauch und ist somit ausschlaggebend für die Dauer des Trainings und für dessen Energieverbrauch.

Die unterschiedlichen prozentualen Energieeinsparungen, die beim Training der ver-schiedenen neuronalen Netze durch das Absenken der maximalen CPU-Frequenz und -Spannung beobachtet wurden (Abbildung 24), lassen sich durch die Architektur der Modelle und der daraus resultierenden Berechnungsdauer der einzelnen Batches erklä-ren.

Das AlexNet-Modell und das VGG16-Modell haben beispielsweise eine vergleichsweise einfache Architektur mit vielen aufeinanderfolgenden Faltungsschichten sowie großen und vollständig miteinander verbundenen Schichten. Die Berechnungsdauer eines Bat-ches beträgt beim AlexNet etwa 3-4 ms und beim VGG16 etwa 6-8 ms. Durch die häufige Auslastung der CPU bei diesen Modellen und die daraus folgende Reduktion der CPU-Wartezeiten, über die gesamte Trainingsdauer hinweg, hat die Verringerung der maximalen Frequenz und Spannung einen relativ großen Einfluss auf die Energie-einsparung der CPU. Dies ist anhand der Abbildung 24 zu erkennen. Hier weisen die Modelle mit einer deutlich höheren Batchzeit eine erheblich geringere Energieeinspa-rung auf als die Modelle mit einer sehr geringen Batchzeit.

Das DenseNet-Modell und das ResNet-Modell sind komplexer. Sie sind durch tiefe Architekturen und Mechanismen, wie Densely Connected Layers beim DenseNet und Residual Verbindungen beim ResNet, gekennzeichnet. Die Berechnungsdauer eines Bat-ches beträgt beim DenseNet etwa 70-75 ms und beim ResNet etwa 50-60 ms, je nach Datensatz. Beispielsweise weist das DenseNet-Modell mit einer Batchzeit von 70-75 ms die geringste CPU-Energieeinsparung auf. Die geringeren Energieeinsparungen im Vergleich zu den anderen Modellen (AlexNet, VGG16) lassen sich anhand der Laufzeit-veränderung aus Abbildung 18 erklären. Durch die vergleichsweise sehr starke Laufzeit-zunahme beim DenseNet und beim ResNet sind die Energieeinsparungen sehr gering, da sich die Energie aus der Leistungsaufnahme und der vergangenen Zeit ergibt. Die Leistungsaufnahme verringert sich zwar, jedoch steigt die vergangene Zeit an, wodurch die Energieeinsparungen vergleichsweise sehr gering oder gar nicht vorhanden sind.

Die unterschiedlichen prozentualen Energieeinsparungen bei der Verwendung verschiedener Datensätze in den Modellen könnten durch mehrere unterschiedliche Faktoren beeinflusst werden. Zu diesen Faktoren gehören die Komplexität der Daten und daraus resultierend die Berechnungsdauer der einzelnen Batches. Der Fashion-MNIST-Datensatz ist ein einfacher Datensatz bestehend aus Graustufenbildern mit einer Größe von 28x28 Pixeln. Die Einfachheit und geringe Größe der Bilder bedeuten, dass die Berechnungsdauer der Batches geringer ist als beispielsweise beim Cifar10-Datensatz. Dies hat zur Folge, dass die CPU weniger warten muss und mehr von der Frequenz- und Spannungsänderung der CPU betroffen ist. Dadurch ist die eingesparte Energie beim Fashion-MNIST-Datensatz auch höher als beim Cifar10-Datensatz, der mit den farbigen Bildern mit einer Größe von 32x32 Pixeln und drei Farbkanälen eine höhere Komplexität und damit auch eine leicht erhöhte Berechnungsdauer der Batches aufweist.

Der Grund für den stetigen Abfall der Energie, bei denen ausschließlich die CPU-Spannung angepasst wurde, liegt darin, dass sich die Spannung quadratisch auf den Energieverbrauch auswirkt, siehe Gleichung 4. Dadurch kann eine hohe Energieeinsparung erzielt werden, ohne dass der Verbrauch nach einiger Zeit wieder ansteigt. Zusätzlich gibt es hier keinen nennenswerten Laufzeitzuwachs (Abbildung 21), der dafür sorgen könnte, dass sich die Abnahme des Energieverbrauchs verlangsamt und wieder ansteigt. Der Nachteil ist hierbei, dass ab einer zu geringen Spannung die CPU nicht mehr korrekt arbeitet und dazu neigt abzustürzen. Dies ist auch der Grund weshalb die Messung bei 3600 MHz bei 1,0 V nicht durchgeführt werden konnte. Auch hier zeigt sich das Verhalten, dass der Cifar10-Datensatz beim AlexNet-Modell mehr Energie verbraucht als der Fashion-MNIST-Datensatz.

In Abbildung 27 wurde die gesamte prozentuale Energieeinsparung bei den Modellen durch ausschließliche Anpassung der CPU-Spannung dargestellt. Im Vergleich zu den prozentualen Energieeinsparungen bei der Anpassung der CPU-Frequenz und Spannung (siehe Abbildung 24) ist hier eine nahezu konstante Energieeinsparung von etwa 27 % über alle Modelle hinweg möglich. Solange die CPU mit der jeweiligen Frequenz-Spannungs-Kombination arbeiten kann, lässt sich stetig Energie einsparen. Hierbei lassen sich die Energieeinsparungen vermutlich auf die Hardwareumsetzungen und -mechanismen von modernen CPUs zurückführen. An dieser Stelle wäre es hilfreich, weitere Untersuchungen durchzuführen, um zu ermitteln, ab wann eine zusätzliche Frequenzverringerung mehr Energie einspart als eine reine Spannungsreduktion. Darüber hinaus sind Untersuchungen nötig, die die mögliche Hardware-Ursache der beobachteten Energieeinsparung genauer erforschen.

Eine weitere Erklärung ist mit Blick auf die Daten aus Abbildung 29, dass trotz der verringerten Spannung, die CPU-Kerne ihre durchschnittliche effektive Taktfrequenz nicht großartig verändern (maximal 3,68 %) und trotzdem einiges an Energie eingespart wird. Der Grund dafür liegt möglicherweise darin, dass nicht alle sechs verwendeten CPU-Kerne auf der maximalen Frequenz, den 3600 MHz, arbeiten. Einige CPU-Kerne werden runter reguliert und andere Kerne wiederum gleichen dies aus, wodurch sich der Durchschnitt nicht großartig verändert. Die Energieeinsparung nimmt aber mit abnehmender Spannung, durch das "Abschalten" einiger Kerne, weiterhin zu.

Ein weiterer Grund dafür könnte die quadratische Abhängigkeit der Leistung von der Spannung darstellen, wodurch eine Reduktion der Spannung zu einer merklichen Reduktion der Leistungsaufnahme führt, selbst wenn die Frequenz konstant bleibt. Außerdem könnte an dieser Stelle auch der optimale Betriebspunkt der CPU eine Rolle spielen, bei dem die CPU energiesparender arbeitet [84].

Abschließend ist anhand der Daten aus Abbildung 28 zu sagen, dass die Energiesparmaßnahmen der CPU, beim Training neuronaler Netze auf der GPU, einen hohen Einfluss auf die verbrauchte Energie haben. Die CPU-Energieverbrauchsanteile liegen nämlich je nach Modell zwischen 17 % und 37 %. Aus diesem Grund sollten die Energiesparmaßnahmen der CPU nicht vernachlässigt werden.

7.3 Betrachtung der Genauigkeit

Die Datenlage zur Modell-Genauigkeit zeigt, wie erwartet, einen konstanten Verlauf der unterschiedlichen CPU-Frequenzen und -Spannungen, siehe Abbildung 31. Hierbei ist die Genauigkeit vom Fashion-MNIST-Datensatz höher als vom Cifar10-Datensatz, weil es sich bei dem Fashion-MNIST-Datensatz um einen weniger komplexen Datensatz mit 28x28 Pixel großen Graustufenbildern handelt und die Bilder weniger komplex aufgebaut sind. So sind zum Beispiel klare und einfache Hintergründe beim Fashion-MNIST-Datensatz vorhanden, wohingegen beim Cifar10-Datensatz weitaus komplexere Hintergründe zur Verfügung stehen. Außerdem handelt es sich im Cifar10-Datensatz um 32x32 Pixel große Farbbilder. Hiermit lässt sich bestätigen, dass die Verringerung der CPU-Frequenz und -Spannung keinen Einfluss auf die Modellgenauigkeit hat, was darauf hindeutet, dass die Leistungsfähigkeit des Modells nicht durch die Hardwaredingungen beeinträchtigt wird. Darüber hinaus bestehen aber Zusammenhänge zwischen der Komplexität der Daten und der Modellgenauigkeit, wodurch weniger komplexe Daten auch eine höhere Genauigkeit erreichen können. Abschließend lässt sich sagen, dass die Modellgenauigkeit nicht von den getesteten Hardwarevariationen beeinflusst wird, wodurch Energiesparmaßnahmen durch Reduktion der CPU-Frequenz und -Spannung umgesetzt werden können, ohne dabei die Modellgenauigkeit zu beeinträchtigen.

8 Zusammenfassung und Ausblick

Im Rahmen dieser Masterarbeit wurde erfolgreich untersucht und aufgezeigt, wie sich das Power-Capping der CPU während des Trainings neuronaler Netze auf der GPU auf die Energieeffizienz und Leistungsfähigkeit (Trainingsdauer und Genauigkeit) des Trainingsprozesses auswirkt. Die Untersuchungen wurden zunächst an den Modellen bzw. an den neuronalen Netzen AlexNet, VGG16, ResNet101 und DenseNet121 jeweils mit dem Cifar10-Datensatz und dem Fashion-MNIST-Datensatz durchgeführt. Die hierbei protokollierten Metriken waren die Frequenz, die Trainingsdauer, die Modellgenauigkeit und der Energieverbrauch. Außerdem wurde für die Aussagekraft der Energieeinsparungen, die durch das Power-Capping erreicht wurden, der CPU-Energieverbrauchsanteil zwischen der GPU und der CPU ermittelt.

Die Untersuchung der Trainingsdauer zeigt, dass die Trainingsdauer bei reduzierter maximaler CPU-Frequenz steigt, da die CPU weniger Anweisungen pro Sekunde verarbeiten kann. Das VGG16-Modell bildet eine Ausnahme. Hier bleiben die Trainingszeiten bis 2600 MHz nahezu konstant und steigen erst nach 2600 MHz merklich an. Das AlexNet-Modell weist, aufgrund seiner vergleichsweise weniger komplexen Architektur, beim Training mit dem CIFAR-10-Datensatz eine längere Trainingsdauer auf als mit dem Fashion-MNIST-Datensatz. Die Abbildung 18 zeigt, dass die Trainingszeiten je nach Modell zwischen 35 % und 118 % zunehmen. Besonders bei dem DenseNet-Modell und dem ResNet-Modell sind die Laufzeitenveränderungen sehr stark ausgeprägt. Im Gegensatz dazu sind das AlexNet-Modell und das VGG16-Modell weniger stark von der Laufzeitveränderung betroffen. Nichtsdestotrotz kann an dieser Stelle bestätigt werden, dass eine niedrigere CPU-Frequenz zu einer längeren Verarbeitungszeit des Trainings führt. Allerdings bleibt bei einer ausschließlichen Reduktion der CPU-Spannung ohne Anpassung der CPU-Frequenz die Trainingsdauer nahezu konstant.

Die Untersuchung des Energieverbrauchs zeigt, dass der Energieverbrauch zunächst durch die Verringerung der CPU-Frequenz und -Spannung sinkt und dann, bedingt durch die optimalen Betriebsbedingungen der CPU, wieder ansteigt. Dieses beobachtete Verhalten tritt bei den Modellen AlexNet, ResNet und VGG16 auf. Eine moderate Reduktion der CPU-Frequenz und -Spannung führt jedoch zu merklichen Energieeinsparungen zwischen 2,09 % und 26,35 %. Dies ist jedoch vom verwendeten Modell und dem Datensatz abhängig. Das DenseNet-Modell zeigt ein anderes Verhalten mit zunächst konstantem und dann steigendem Energieverbrauch. Beim AlexNet-Modell verbraucht das Training mit dem Cifar10-Datensatz mehr Energie als mit dem Fashion-MNIST-Datensatz, während andere Modelle beim Fashion-MNIST-Datensatz mehr Energie für das Training benötigen.

Schlussfolgernd beeinflusst die Architektur der Modelle die Berechnungsdauer der Batches und dadurch die Effizienz des Trainings bei verschiedenen Datensätzen und damit wird letztendlich der Energieverbrauch beeinflusst. Der Grund dafür ist, dass die CPU bei einer geringen Batchzeit häufiger in einer bestimmten Zeit aktiv ist und weniger warten muss, wodurch sie anfälliger für Frequenzänderungen ist. Dies ist dann abhängig vom Modell und Datensatz.

Die Reduktion der CPU-Spannung allein führt zu deutlichen Energieeinsparungen, ohne dabei die Trainingszeiten zu verlängern. Diese Variante des Power-Cappings ist jedoch begrenzt durch die Stabilität der CPU. Insgesamt zeigen die Ergebnisse, dass durch die Anpassung der CPU-Frequenz und -Spannung merkliche Energieeinsparungen beim Training neuronaler Netze auf der GPU möglich sind.

Die Datenlage zeigt, dass die Modellgenauigkeit bei unterschiedlichen CPU-Frequenzen und -Spannungen konstant bleibt. Die Genauigkeit des Fashion-MNIST-Datensatzes ist höher als die des CIFAR-10-Datensatzes, da der Fashion-MNIST-Datensatz weniger komplex ist. Dies bestätigt, dass die Reduktion der CPU-Frequenz und -Spannung keinen Einfluss auf die Modellgenauigkeit hat. Energiesparmaßnahmen durch die Verringerung der CPU-Frequenz und -Spannung können daher ohne Beeinträchtigung der Modellgenauigkeit umgesetzt werden.

Abschließend lässt sich an dieser Stelle sagen, dass durch eine moderate Verringerung der CPU-Frequenz und -Spannung erhebliche Energieeinsparungen erreicht werden können, ohne dabei die Modellgenauigkeit zu beeinträchtigen. Allerdings wird durch die Verringerung der maximalen CPU-Frequenz die Trainingsdauer erhöht, wodurch wiederum der gesamte Energieverbrauch ansteigen kann. Außerdem kann eine erhebliche Menge an Energie eingespart werden, wenn ausschließlich die CPU-Spannung angepasst wird. Dies ist jedoch nur möglich, wenn die CPU nicht mehr mit ausreichend Spannung versorgt werden kann. Die Energieeinsparungen variieren je nach Modellarchitektur und Art des Datensatzes. Insbesondere zeigte sich, dass komplexere Modelle wie das DenseNet-Modell und das ResNet-Modell weniger Potenzial für Energieeinsparungen durch CPU-Power-Capping haben. Dies lässt sich mit der relativ hohen Batchzeit sowie der Neutralisierung der Einsparungen bei der Leistungsaufnahme durch den Laufzeitzuwachs, erklären. Das AlexNet-Modell und das VGG16-Modell haben dafür allerdings ein höheres Energieeinsparungspotenzial durch die CPU, weil dort die Batchzeit und der Laufzeitzuwachs vergleichsweise gering sind.

Die Energieeinsparung durch das Power-Capping der CPU sind eine sehr vielversprechende Möglichkeit, den Energieverbrauch beim Training neuronaler Netze zu reduzieren. Außerdem werden dadurch neue Möglichkeiten für energieeffiziente Trainingsprozesse eröffnet. Vor allem bei Anwendungen, bei denen die Einsparung von Ressourcen von wichtiger Bedeutung ist.

Ein interessanter Ansatz für weitere zukünftige Untersuchungen wäre es, eine adaptive Steuerung der CPU-Frequenz und -Spannung zu entwickeln, die sich dynamisch an den Anforderungen des jeweiligen Modells und Datensatzes anpasst. Hierfür sind auch Analysen notwendig, um herauszufinden, ob es möglich ist, das Power-Capping nur in ausgewählten Phasen des Trainings zu reduzieren, damit die Energieeffizienz gezielt optimiert werden kann, ohne die Trainingsdauer unnötig zu verlängern. Außerdem wäre es vorteilhaft, diese Untersuchungen mit anderen Architekturen und komplexeren Datensätzen durchzuführen, um die Generalisierbarkeit der Ergebnisse zu überprüfen.

Literatur

- [1] Adrian Lobe. *Künstliche Intelligenz verbraucht für den Lernprozess unvorstellbar viel Energie*. Zugriffsdatum: 04.02.2024. 2019. URL: <https://www.spektrum.de/news/kuenstliche-intelligenz-verbraucht-fuer-den-lernprozess-unvorstellbar-viel-energie/1660246>.
- [2] N3XTCODER. *Der große Energieverbrauch Künstlicher Intelligenz*. Zugriffsdatum: 08.06.2024. 2023. URL: <https://n3xtcoder.org/de/energy-impact-of-ai>.
- [3] Vanessa Mehlin, Sigurd Schacht und Carsten Lanquillon. *Towards energy-efficient Deep Learning: An overview of energy-efficient approaches along the Deep Learning Lifecycle*. 2023. arXiv: 2303.01980 [cs.LG].
- [4] Fraunhofer Institute. *Maschinelles Lernen - Kompetenzen, Anwendungen und Forschungsbedarf*. Zugriffsdatum: 04.02.2024. 2018. URL: https://www.bigdata-ai.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/BMBF_Fraunhofer_ML-Ergebnisbericht_Gesamt.pdf.
- [5] *Dynamic Voltage and Frequency Scaling in Power Management*. Zugriffsdatum: 04.02.2024. 2011. URL: <https://www.itwissen.info/dynamic-voltage-and-frequency-scaling-power-management-DVFS.html>.
- [6] Advanced Micro Devices, Inc. *Ryzen Master*. Zugriffsdatum: 17.02.2024. Advanced Micro Devices, Inc. 2024. URL: <https://www.amd.com/de/technologies/ryzen-master>.
- [7] Sebastian Heinz. *Deep Learning Teil 1 – Einführung*. Zugriffsdatum: 29.02.2024. Statworx. 2017. URL: <https://www.statworx.com/content-hub/blog/deep-learning-teil-1-einfuehrung/>.
- [8] Studysmarter. *Genomik*. Zugriffsdatum: 11.09.2024. 2024. URL: <https://www.studysmarter.de/studium/biologie-studium/genetik-studium/genomik/#:~:text=innovative%20Durchbr%C3%BCche%20erm%C3%A4glichen.%20Wichtigste,Genen%20und%20nicht%20kodierende%20Abschnitte..>
- [9] Prof. Dr. Ute Schmid. *Maschinelles Lernen*. Zugriffsdatum: 25.04.2024. 2024. URL: <https://www.bidt.digital/glossar/maschinelles-lernen/>.
- [10] Laurenz Wuttke. *Neuronale Netzwerke: Einführung*. Zugriffsdatum: 29.02.2024. 2023. URL: <https://datasolut.com/neuronale-netzwerke-einfuehrung/>.

- [11] Detlev Frick u. a. *Data Science Konzepte, Erfahrungen, Fallstudien und Praxis: Konzepte, Erfahrungen, Fallstudien und Praxis*. Jan. 2021, S. 225–238. ISBN: 978-3-658-33402-4. DOI: 10.1007/978-3-658-33403-1.
- [12] Artem Oppermann. *Aktivierungsfunktionen in neuronalen Netzen: Sigmoid, tanh, ReLU*. Zugriffsdatum: 27.04.2024. 2021. URL: <https://artemoppermann.com/de/aktivierungsfunktionen/>.
- [13] Interview Bit. *CNN Architecture – Detailed Explanation*. Zugriffsdatum: 26.04.2024. 2024. URL: <https://www.interviewbit.com/blog/cnn-architecture/>.
- [14] github.io. *Convolutional Neural Networks (CNNs / ConvNets)*. Zugriffsdatum: 26.04.2024. 2024. URL: <https://cs231n.github.io/convolutional-networks/>.
- [15] Max-Ludwig Stadler. *Convolutional Neural Network (CNN)*. Zugriffsdatum: 01.03.2024. 2021. URL: <https://mindsquare.de/knowhow/convolutional-neural-network/>.
- [16] Daniel Johnson. *CNN-Bildklassifizierung in TensorFlow mit Schritten und Beispielen*. Zugriffsdatum: 26.04.2024. 2023. URL: <https://www.guru99.com/de/convnet-tensorflow-image-classification.html>.
- [17] J.O. Schneppat. *Convolutional Neural Networks (CNNs / ConvNets)*. Zugriffsdatum: 26.04.2024. 2023. URL: <https://gpt5.blog/convolutional-neural-networks-cnns/>.
- [18] EpyNN. *Pooling (CNN)*. Zugriffsdatum: 27.04.2024. 2021. URL: <https://epynn.net/Pooling.html>.
- [19] databsecamp. *Was ist die Softmax-Funktion?* Zugriffsdatum: 26.04.2024. 2023. URL: <https://databsecamp.de/ki/softmax>.
- [20] Aqeel Anwar. *Difference between AlexNet, VGGNet, ResNet, and Inception*. Zugriffsdatum: 09.03.2024. 2019. URL: <https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>.
- [21] Karen Simonyan und Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].
- [22] datagen.tech. *Understanding VGG16: Concepts, Architecture, and Performance*. Zugriffsdatum: 09.03.2024. URL: <https://datagen.tech/guides/computer-vision/vgg16/#>.

- [23] DataGen.techn. *ResNet: The Basics and 3 ResNet Extensions*. Zugriffsdatum: 08.03.2024. URL: <https://datagen.techn/guides/computer-vision/resnet/>.
- [24] Jörg-Owe Schneppat. *Verschwindende Gradienten (Vanishing Gradient)*. Zugriffsdatum: 08.03.2024. 2022. URL: <https://gpt5.blog/verschwindende-gradienten-vanishing-gradient/#:~:text=Das%20Problem%20der%20verschwindenden%20Gradienten&text=Das%20Ph%C3%A4nomen%20der%20verschwindenden%20Gradienten,berechnet%20werden%2C%20sukzessive%20kleiner%20werden..>
- [25] Kaiming He u. a. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].
- [26] Gao Huang u. a. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].
- [27] MathWorks. *Was ist Deep Learning?* Zugriffsdatum: 15.03.2024. 2024. URL: <https://de.mathworks.com/discovery/deep-learning.html#:~:text=Deep%2DLearning%2DModelle%20werden%20mit,eine%20manuelle%20Merkmalsextraktion%20erforderlich%20ist..>
- [28] Alex Krizhevsky. „Learning Multiple Layers of Features from Tiny Images“. In: *University of Toronto* (Mai 2012).
- [29] Zalando Research. *Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms*. <https://github.com/zalandoresearch/fashion-mnist>. Zugriffsdatum: 10.02.2024. 2017.
- [30] Zalando Research. *FashionMNIST*. <https://www.kaggle.com/datasets/zalando-research/fashionmnist>. Zugriffsdatum: 15.03.2024. 2018.
- [31] Futura. *Was ist CPU? Eine Definition*. Zugriffsdatum: 17.03.2024. 2022. URL: https://www.futura-sciences.com/de/was-ist-cpu-definition_11602/.
- [32] Ben Lutkevich. *cache memory*. Zugriffsdatum: 27.04.2024. 2020. URL: <https://www.techtarget.com/searchstorage/definition/cache-memory>.
- [33] Tech Target Contributor. *speculative execution*. Zugriffsdatum: 28.04.2024. 2024. URL: <https://www.techtarget.com/whatis/definition/speculative-execution>.
- [34] Muhammad Zubai. *What is branch prediction?* Zugriffsdatum: 28.04.2024. 2024. URL: <https://www.educative.io/answers/what-is-branch-prediction>.
- [35] Dirk W. Hoffmann. *Grundlagen der Technischen Informatik*. 3. Auflage. HANSER, 2013. ISBN: 978-3446477797.

- [36] Jürgen Ortmann. *Einführung in die PC-Grundlagen*. 9. Auflage. ADDISON-WESLEY, 2006. ISBN: 978-3827323392.
- [37] Speichermarkt Team. *Zusammenarbeit von CPU und Arbeitsspeicher*. Zugriffsdatum: 29.04.2024. 2016. URL: <https://www.speichermarkt.de/blog/wissen/zusammenarbeit-von-cpu-und-speicher/>.
- [38] Margaret Rouse. *Backside Bus*. Zugriffsdatum: 29.04.2024. 2011. URL: <https://www.techopedia.com/definition/296/backside-bus-bsb>.
- [39] Philipp Goeller. *GPU vs CPU - Performance von maschine learning mordellen um 500 Prozent steigern?* Zugriffsdatum: 21.03.2024. 2020. URL: <https://paraboost.de/ki-computer/gpu-vs-cpu-wie-kann-ich-die-performance-von-machine-learning-modellen-um-ueber-500-steigern/>.
- [40] André Schmitz. *Software-Entwicklung für Multicore-Systeme*. Zugriffsdatum: 28.04.2024. 2019. URL: <https://www.embedded-software-engineering.de/software-entwicklung-fuer-multicore-systeme-a-811008/>.
- [41] Pure Storage. *CPU versus GPU für maschinelles Lernen*. Zugriffsdatum: 21.03.2024. 2024. URL: <https://blog.purestorage.com/de/purely-bildung/cpu-versus-gpu-fuer-maschinelles-lernen/#:~:text=Modelle%20lernen%20schneller%20wenn%20alle,schneller%20verarbeiten%20als%20eine%20CPU>.
- [42] Andrea Manconi u. a. „G-CNV: A GPU-based tool for preparing data to detect CNVs with read-depth methods“. In: *Frontiers in Bioengineering and Biotechnology* 3 (März 2015). DOI: 10.3389/fbioe.2015.00028.
- [43] Databraino. *GPU-PROZESSOREN FÜR KÜNSTLICHE INTELLIGENZ*. Zugriffsdatum: 21.03.2024. 2019. URL: <https://databraineo.com/ki-training-resources/gpu-prozessoren-fuer-kuenstliche-intelligenz/>.
- [44] Johanna leierseder. *Tech-Facts: dedizierte vs. integrierte Grafikkarte – das ist der Unterschied*. Zugriffsdatum: 28.04.2024. 2023. URL: <https://www.cyberport.de/blog/erstmalverstehen/erstmalverstehen-dedizierte-vs-integrierte-grafikkarte-das-ist-der-unterschied/>.
- [45] Amazon Web Services. *Was ist der Unterschied zwischen GPUs und CPUs?* Zugriffsdatum: 21.03.2024. 2023. URL: <https://aws.amazon.com/de/compare/the-difference-between-gpus-cpus/>.
- [46] José Miguel Mantas, Marc De la Asunción und Manuel J. Castro. „An Introduction to GPU Computing for Numerical Simulation“. In: *Numerical Simulation in Physics and Engineering: Lecture Notes of the XVI 'Jacques-Louis Li-*

- ons' Spanish-French School.* Hrsg. von Inmaculada Higueras, Teo Roldán und Juan José Torrens. Cham: Springer International Publishing, 2016, S. 219–251. ISBN: 978-3-319-32146-2. DOI: 10.1007/978-3-319-32146-2_5. URL: https://doi.org/10.1007/978-3-319-32146-2_5.
- [47] Margaret Rouse. *Cache-Speicher*. Zugriffsdatum: 29.04.2024. 2024. URL: <https://www.techopedia.com/de/definition/cache-speicher#text=Auch%20moderne%20Grafikkarten%20haben%20einen,des%20Systems%20angewiesen%20zu%20sein..>
- [48] Wolfram Schiffmann, Helmut Bähring und Udo Höning. „Aufbau und Funktion eines Personal Computers“. In: *Technische Informatik 3: Grundlagen der PC-Technologie*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 1–58. ISBN: 978-3-642-16812-3. DOI: 10.1007/978-3-642-16812-3_1. URL: https://doi.org/10.1007/978-3-642-16812-3_1.
- [49] Wesley Chai. *VRAM (Video RAM)*. Zugriffsdatum: 29.04.2024. 2020. URL: <https://www.computerweekly.com/de/definition/VRAM-Video-RAM>.
- [50] Andreas Knoll. *Die GPU als Co-Prozessor der CPU*. Zugriffsdatum: 28.04.2024. 2011. URL: <https://www.elektroniknet.de/automation/die-gpu-als-co-prozessor-der-cpu.75167.html>.
- [51] Geekforgeeks (yuvraj10). *7 Best Deep Learning Frameworks You Should Know in 2024*. Zugriffsdatum: 28.04.2024. 2024. URL: <https://www.geeksforgeeks.org/deep-learning-frameworks/>.
- [52] run.a: *PyTorch GPU*. Zugriffsdatum: 08.06.2024. 2024. URL: <https://www.run.ai/guides/gpu-deep-learning/pytorch-gpu>.
- [53] Jason Bell. *Using CPU for AI Inference: A Cost-Effective Alternative to GPU-Based Cloud Solutions*. Zugriffsdatum: 28.04.2024. 2024. URL: <https://jasebell.medium.com/using-cpu-for-ai-inference-a-cost-effective-alternative-to-gpu-based-cloud-solutions-a46a6bdf8913>.
- [54] William Stallings. *Betriebssysteme - BAFÖG-Ausgabe: Prinzipien und Umsetzung*. 4. Auflage. Pearson Studium, 2005. ISBN: 978-3827371850.
- [55] Boris Stippe. *Wie viel Strom verbraucht ein PC?* Zugriffsdatum: 26.03.2024. 2023. URL: [https://solarwissen.selfmade-energy.com/wie-viel-strom-verbraucht-ein-pc/#text=Im%20Allgemeinen%20liegt%20der%20Stromverbrauch,\(GPU\)%20und%20den%20Bildschirm..](https://solarwissen.selfmade-energy.com/wie-viel-strom-verbraucht-ein-pc/#text=Im%20Allgemeinen%20liegt%20der%20Stromverbrauch,(GPU)%20und%20den%20Bildschirm..)
- [56] Muhammed Ibrahim und Ibrahim Hamarash. „Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction“. In: (Dez. 2005).

- [57] Etienne Sueur und Gernot Heiser. „Dynamic voltage and frequency scaling: The laws of diminishing returns“. In: *2010 HotPower* (Okt. 2010).
- [58] Paul J. Kuehn und Maggie Mashaly. „DVFS-Power Management and Performance Engineering of Data Center Server Clusters“. In: *2019 15th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. 2019, S. 91–98. DOI: [10.23919/WONS.2019.8795470](https://doi.org/10.23919/WONS.2019.8795470).
- [59] Landesbildungsserver. *Die Energie einer stromdurchflossenen Spule*. Zugriffsdatum: 30.04.2024. 2016. URL: https://www.schule-bw.de/faecher-und-schularten/mathematisch-naturwissenschaftliche-faecher/physik/unterrichtsmaterialien/e_lehre_2/selbstinduktion/energie_spule_exp.htm.
- [60] SFC Energy AG. *Elektrische Energie*. Zugriffsdatum: 27.03.2024. 2024. URL: <https://www.sfc.com/glossar/elektrische-energie/>.
- [61] Zhenheng Tang u. a. *The Impact of GPU DVFS on the Energy and Performance of Deep Learning: an Empirical Study*. 2019. arXiv: 1905.11012 [cs.PF].
- [62] Adam Krzywaniak, Paweł Czarnul und Jerzy Proficz. „GPU Power Capping for Energy-Performance Trade-Offs in Training of Deep Convolutional Neural Networks for Image Recognition“. In: *Computational Science – ICCS 2022*. Hrsg. von Derek Groen u. a. Cham: Springer International Publishing, 2022, S. 667–681. ISBN: 978-3-031-08751-6.
- [63] ASRock. *B550 Phantom Gaming 4*. Zugriffsdatum: 09.04.2024. 2024. URL: <https://pg.asrock.com/mb/AMD/B550%20Phantom%20Gaming%204/index.de.asp#Overview>.
- [64] AMD. *AMD Ryzen™ 5 3600 Drivers und Support*. Zugriffsdatum: 09.04.2024. 2024. URL: <https://www.amd.com/de/support/cpu/amd-ryzen-processors/amd-ryzen-5-desktop-processors/amd-ryzen-5-3600>.
- [65] NVIDIA. *GEFORCE RTX 3060-Familie*. Zugriffsdatum: 09.04.2024. 2024. URL: <https://www.nvidia.com/de-de/geforce/graphics-cards/30-series/rtx-3060-3060ti/>.
- [66] corsair. *VENGEANCE LPX Speicherkit 16 GB (2 x 8 GB) DDR4 DRAM 2666 MHz C16 – Schwarz*. Zugriffsdatum: 09.04.2024. 2024. URL: <https://www.corsair.com/de/de/p/memory/cmk16gx4m2a2666c16/vengeance-lpx-16gb-2-x-8gb-ddr4-dram-2666mhz-c16-memory-kit-black-cmk16gx4m2a2666c16#tab-techspecs>.

- [67] crucial. *Crucial P1 SSD*. Zugriffsdatum: 09.04.2024. 2024. URL: <https://www.crucial.de/products/ssd/p1-ssd>.
- [68] REALiX. *HWiNFO*. Zugriffsdatum: 09.02.2024. 2024. URL: <https://www.hwinfo.com/>.
- [69] Project Jupyter. *Jupyter*. Zugriffsdatum: 09.02.2024. 2001. URL: <https://jupyter.org/>.
- [70] PyTorch. *Die Energie einer stromdurchflossenen Spule*. Zugriffsdatum: 30.04.2024. 2024. URL: <https://pytorch.org/>.
- [71] AMD. *Ryzen*. Zugriffsdatum: 01.05.2024. 2022. URL: <https://www.amd.com/content/dam/amd/en/documents/products/software-tools/ryzen-master-quick-reference-guide.pdf>.
- [72] Dipl.-IngStefan Luber, Nico Litzel. *Was ist Jupyter?* Zugriffsdatum: 01.05.2024. 2021. URL: <https://www.bigdata-insider.de/was-ist-jupyter-a-1004504/>.
- [73] IBMi. *Was ist PyTorch?* Zugriffsdatum: 01.05.2024. 2024. URL: <https://www.ibm.com/de-de/topics/pytorch>.
- [74] Canadian Institute for Advanced Research (CIFAR). *CIFAR-10 and CIFAR-100 datasets*. Zugriffsdatum: 10.02.2024. URL: <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [75] Technik und Wissen. *Wie erstelle ich ein Deep Learning Modell?* Zugriffsdatum: 10.02.2024. 2019. URL: <https://www.technik-und-wissen.ch/wie-erstelle-ich-ein-deep-learning-modell.html>.
- [76] Sebastian Raschka. *Deep Learning Models Repository*. Zugriffsdatum: 09.02.2024. 2017. URL: <https://github.com/rasbt/deeplearning-models>.
- [77] Keanu Wandke. *Jupyter-Notebooks Masterarbeit*. Zugriffsdatum: 16.09.2024. 2024. URL: https://github.com/Keanu99/Masterarbeit_Wandke.
- [78] Max. *Was ist die Batch-Größe in Deep Learning?* Zugriffsdatum: 02.05.2024. 2023. URL: <https://metaprompting.de/ai/what-should-be-the-batch-size-in-deep-learning/>.
- [79] Ed Burns. *Deep Learning*. Zugriffsdatum: 02.05.2024. 2021. URL: <https://www.computerweekly.com/de/definition/Deep-Learning>.

- [80] Datascientest. *Epoch: Definition, Funktionsweise und Verwendung*. Zugriffsdatum: 02.05.2024. 2023. URL: <https://datascientest.com/de/epoch-definition-funktionsweise-und-verwendung>.
- [81] d2l.ai. *Padding (Machine Learning)*. Zugriffsdatum: 05.05.2024. 2024. URL: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html.
- [82] KNOK. *Handcrafted CNN from scratch - convolutional layer*. Zugriffsdatum: 05.05.2024. 2023. URL: <https://knownoknow.net/2023/07/24/20.html>.
- [83] Lenovo. *Was bedeutet CPU Bottlenecking? - Lenovo*. Zugriffsdatum: 23.08.2024. 2024. URL: <https://www.lenovo.com/de/de/glossary/cpu-bottlenecking/>.
- [84] Stefan. *Sweet-Spot einer CPU über das Prozessor Power Limit finden und einstellen*. Zugriffsdatum: 30.07.2024. 2023. URL: https://www.elefacts.de/test-sweet_spot_einer_cpu_ueber_das_prozessor_power_limit_finden_und_einstellen.

A Anhang

A.1 Ergebnisse: Trainingsdauer

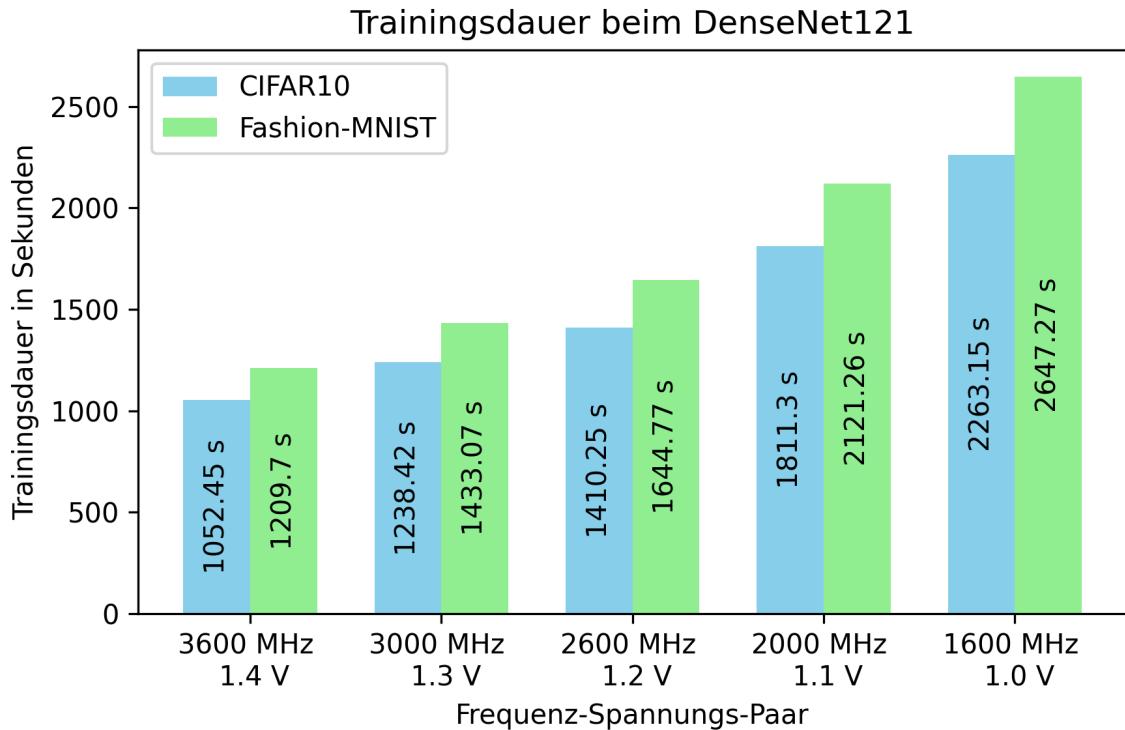


Abbildung 32: Trainingsdauer beim DenseNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In **Abbildung 32** sind die Trainingszeiten des Modells DenseNet mit den Datensätzen Cifar10 und Fashion-MNIST dargestellt. Hier ist zu erkennen, dass die Trainingszeit mit Verringern der Frequenz und Spannung, sowohl beim Training des Cifar10-Datensatzes als auch beim Fashion-MNIST-Datensatz, ansteigt. Dabei ist die Trainingsdauer beim Training des Fashion-MNIST-Datensatzes etwas höher als beim Cifar10-Datensatz. Der Unterschied beträgt hier etwa 157 Sekunden bei 3600 MHz und 1,4 V und etwa 284 Sekunden bei 1600 MHz und 1,0 V.

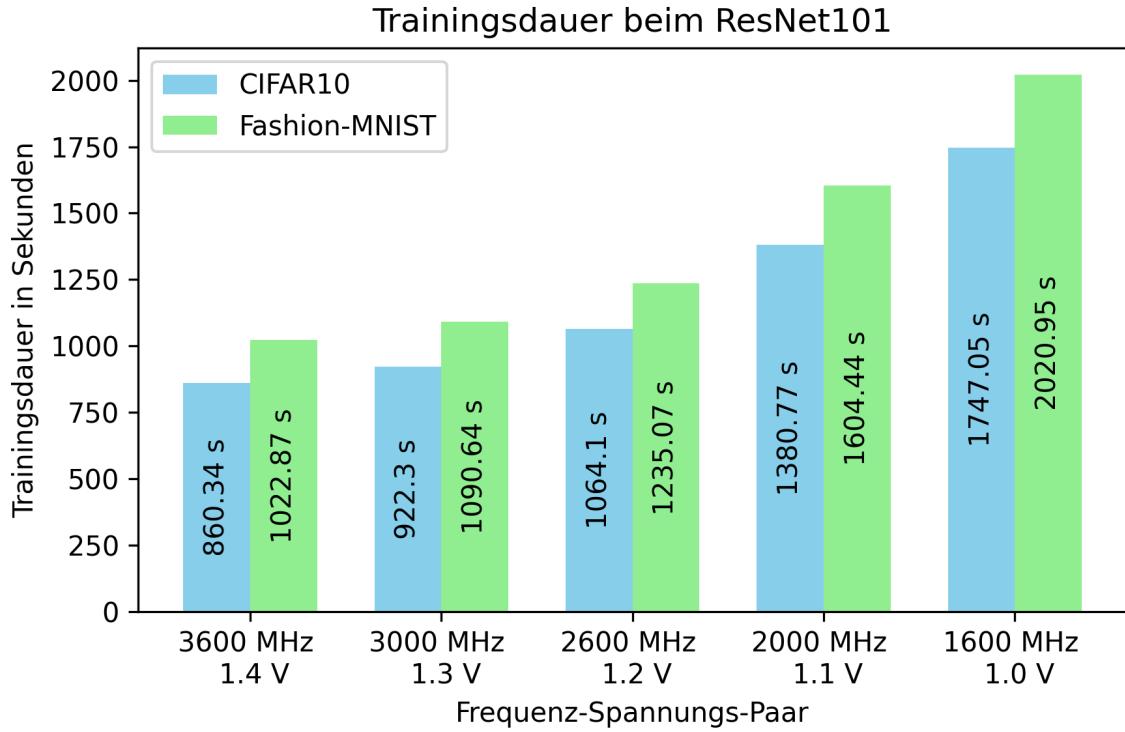


Abbildung 33: Trainingsdauer beim ResNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In **Abbildung 33** sind die Trainingszeiten des ResNet-Modells mit den Datensätzen Cifar10 und Fashion-MNIST dargestellt. Hier ist zu erkennen, dass die Trainingszeit mit Verringern der Frequenz und Spannung, sowohl beim Training des Cifar10-Datensatzes als auch beim Fashion-MNIST-Datensatz, ansteigt. Dabei ist die Trainingsdauer beim Training des Fashion-MNIST-Datensatzes etwas höher als beim Cifar10-Datensatz. Der Unterschied beträgt hier etwa 162 Sekunden bei 3600 MHz und 1,4 V und etwa 273 Sekunden bei 1600 MHz und 1,0 V.

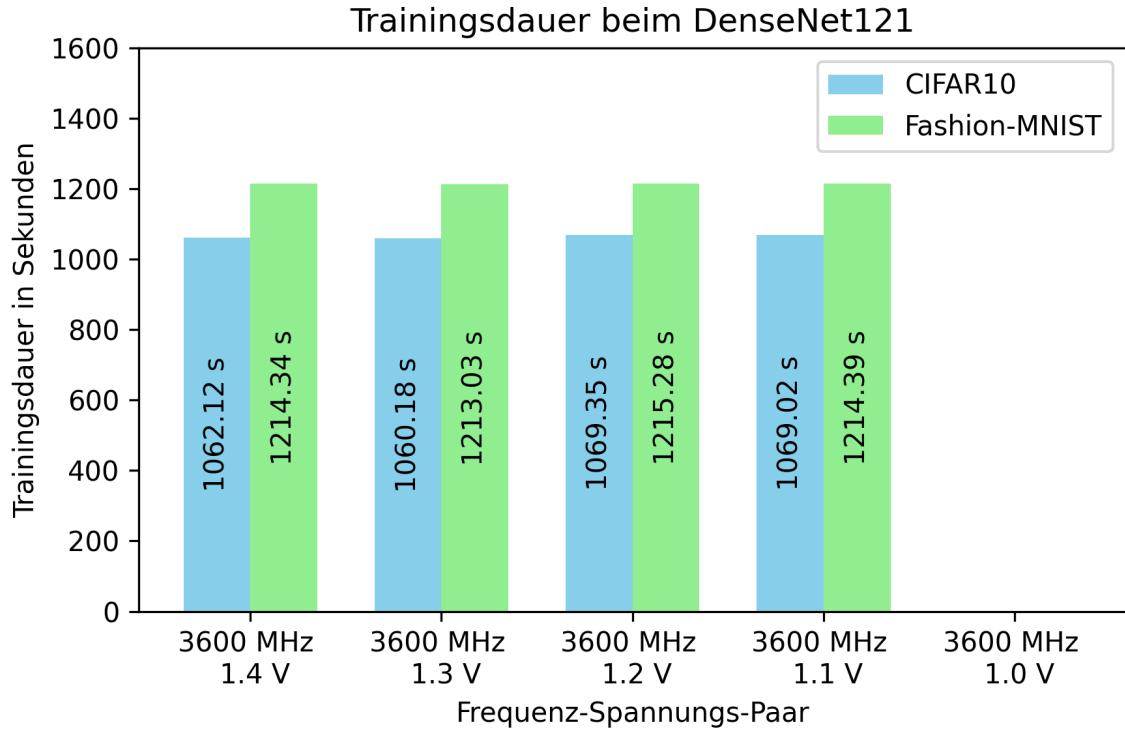


Abbildung 34: Trainingsdauer beim DenseNet bei ausschließlicher Anpassung der CPU-Spannung.

In **Abbildung 34** sind die Trainingszeiten des DenseNet-Modells mit den Datensätzen Cifar10 und Fashion-MNIST dargestellt. Hier ist zu erkennen, dass die Trainingszeit beim Training des Cifar10-Datensatzes und des Fashion-MNIST-Datensatzes mit dem Verringern der Spannung nicht ansteigt. Dabei ist die Trainingsdauer beim Training des Fashion-MNIST-Datensatzes um etwa 150 Sekunden höher. Die Trainingszeiten sind nahezu konstant und betragen beim Cifar10-Datensatz um die 1060 Sekunden und beim Fashion-MNIST-Datensatz um die 1214 Sekunden. Die Messung bei 3600 MHz bei 1,0 V liegt nicht vor, da die Spannungsversorgung für diese Frequenz nicht ausreichend ist.

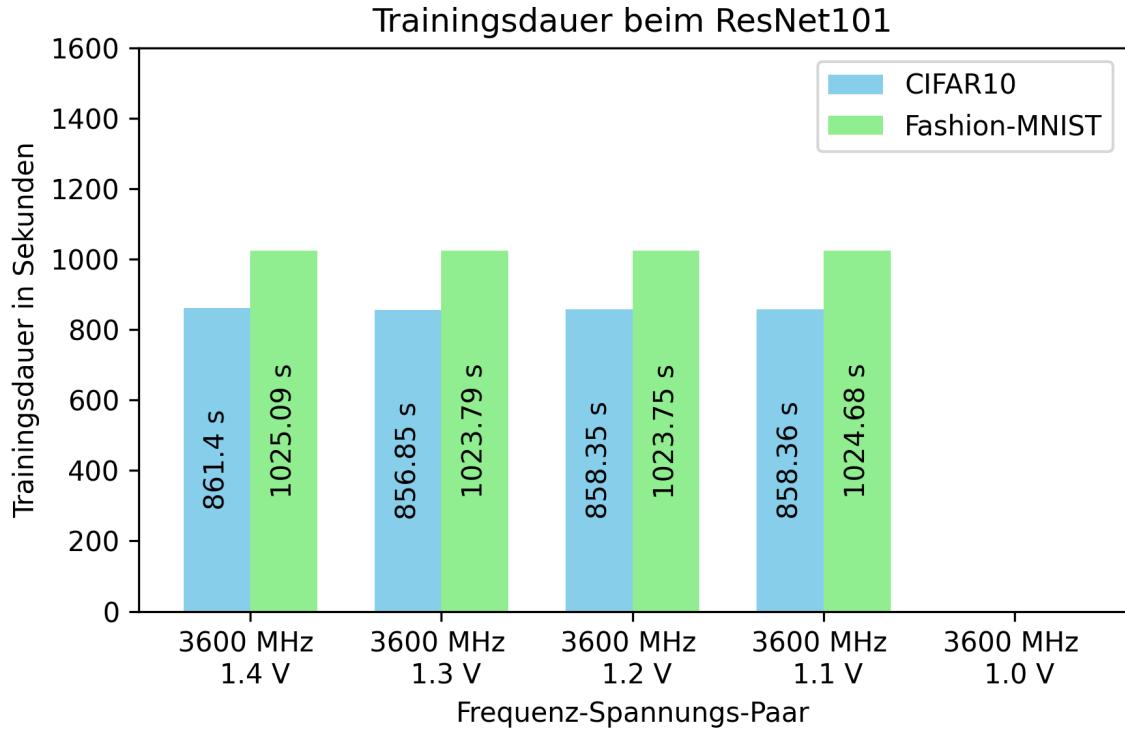


Abbildung 35: Trainingsdauer beim ResNet bei ausschließlicher Anpassung der CPU-Spannung.

In **Abbildung 35** sind die Trainingszeiten des ResNet-Modells mit den Datensätzen Cifar10 und Fashion-MNIST dargestellt. Hier ist zu erkennen, dass die Trainingszeit beim Training des Cifar10-Datensatzes und des Fashion-MNIST-Datensatzes mit Ver- ringern der Spannung nicht ansteigt. Dabei ist die Trainingsdauer beim Training des Fashion-MNIST-Datensatzes um etwa 160 Sekunden höher. Die Trainingszeiten sind nahezu konstant und betragen beim Cifar10-Datensatz um die 860 Sekunden und beim Fashion-MNIST-Datensatz um die 1023 Sekunden. Die Messung bei 3600 MHz bei 1,0 V liegt nicht vor, da die Spannungsversorgung für diese Frequenz nicht ausreichend ist.

A.2 Ergebnisse: Energieverbrauch

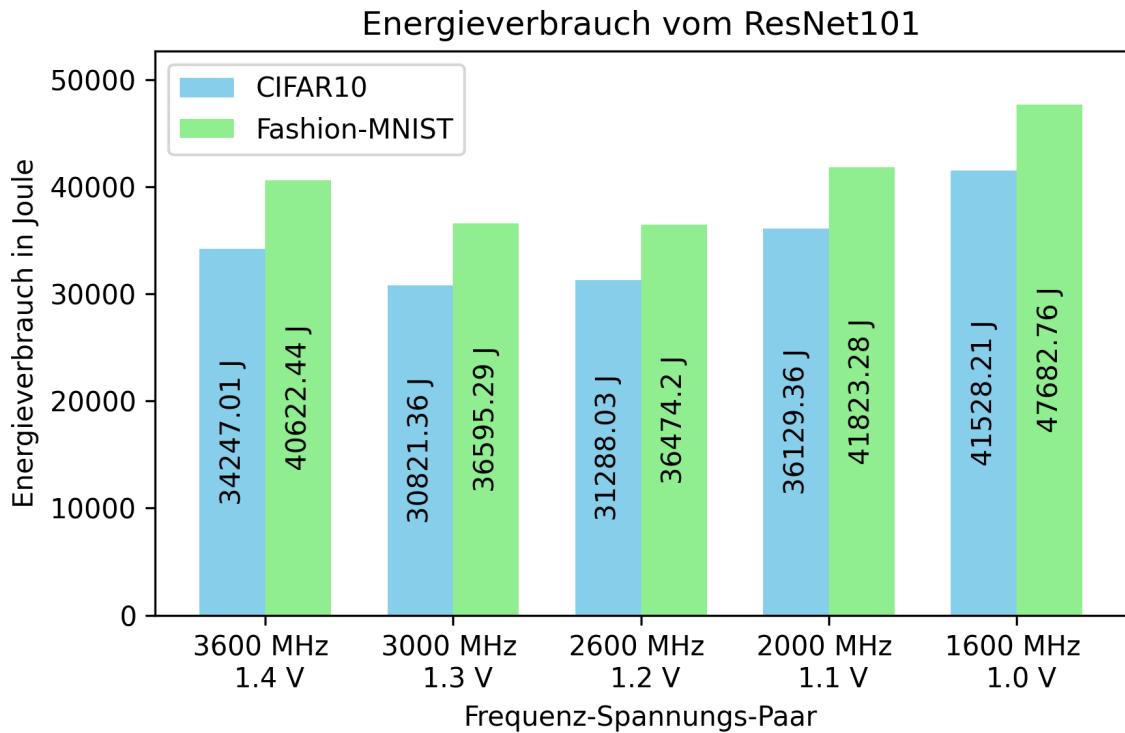


Abbildung 36: Energieverbrauch beim ResNet bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In **Abbildung 36** ist der Energieverbrauch der Trainingseinheiten vom ResNet-Modell mit dem Cifar10-Datensatz und dem Fashion-MNIST-Datensatz bei Variation der CPU-Frequenz und CPU-Spannung dargestellt. Hier ist zu erkennen, dass der Energieverbrauch zunächst, mit Abnahme der maximalen CPU-Frequenz und CPU-Spannung, bis etwa 2600 MHz bei 1,2 V abnimmt und dann bei 2000 MHz bei 1,1 V wieder ansteigt. Des Weiteren ist der Energieverbrauch beim Training mit dem Fashion-MNIST-Datensatz höher als beim Training mit dem Cifar10-Datensatz.

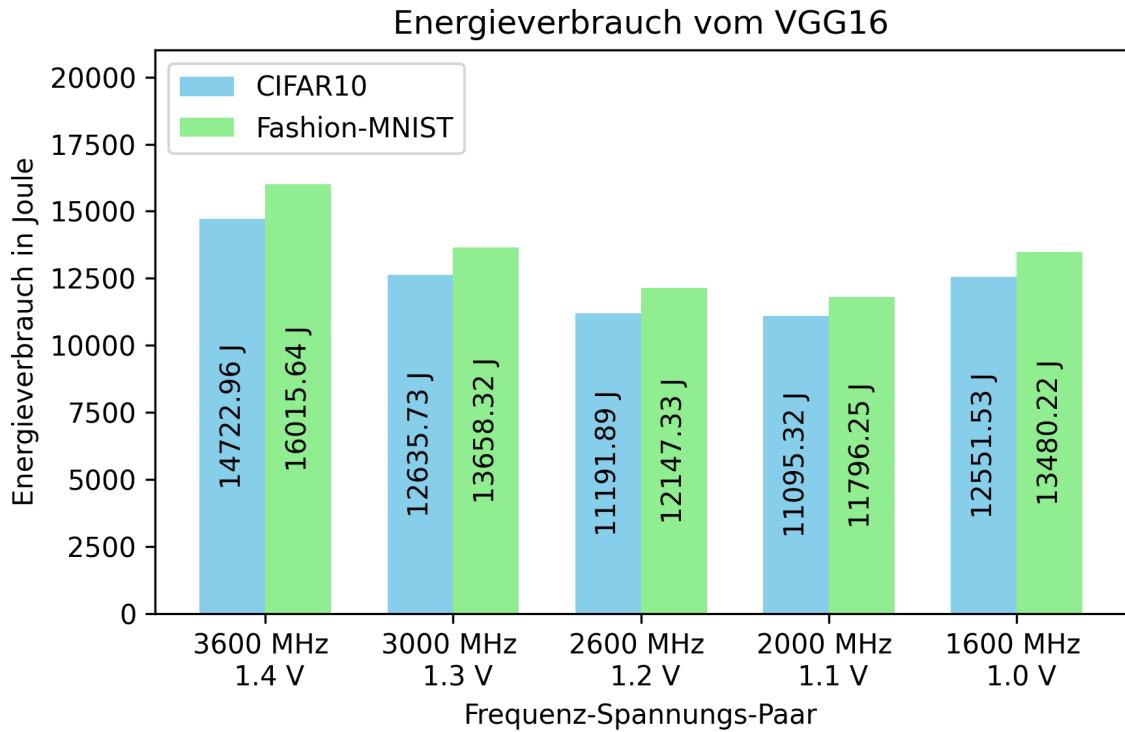


Abbildung 37: Energieverbrauch beim VGG16 bei gleichzeitiger Anpassung der CPU-Frequenz und CPU-Spannung.

In **Abbildung 37** ist der Energieverbrauch der Trainingseinheiten vom VGG16-Modell mit dem Cifar10-Datensatz und dem Fashion-MNIST-Datensatz bei Variation der CPU-Frequenz und CPU-Spannung dargestellt. Hier ist zu erkennen, dass der Energieverbrauch zunächst, mit Abnahme der maximalen CPU-Frequenz und CPU-Spannung, bis etwa 2000 MHz bei 1,1 V abnimmt und dann bei 1600 MHz bei 1,0 V wieder ansteigt. Des Weiteren ist der Energieverbrauch beim Training mit dem Fashion-MNIST-Datensatz höher als beim Training mit dem Cifar10-Datensatz.

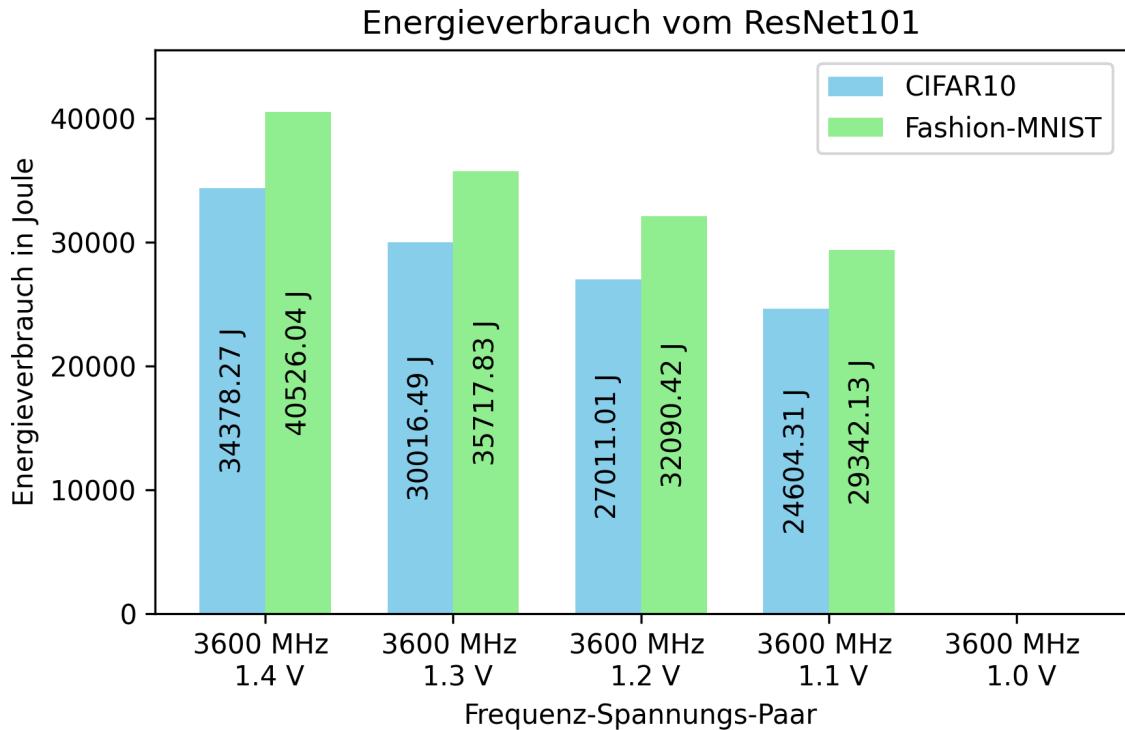


Abbildung 38: Energieverbrauch beim ResNet-Modell bei ausschließlicher Anpassung der CPU-Spannung.

In **Abbildung 38** ist der Energieverbrauch vom ResNet-Modell bei ausschließlicher Anpassung der CPU-Spannung dargestellt. Hier ist ebenfalls zu erkennen, dass sich der Energieverbrauch mit Absenken der CPU-Spannung verringert. Des Weiteren ist der Energieverbrauch beim Training mit dem Fashion-MNIST-Datensatz höher als beim Training mit dem Cifar10-Datensatz. Auch hier ist die Messung bei 3600 MHz und 1,0 V nicht vorhanden.

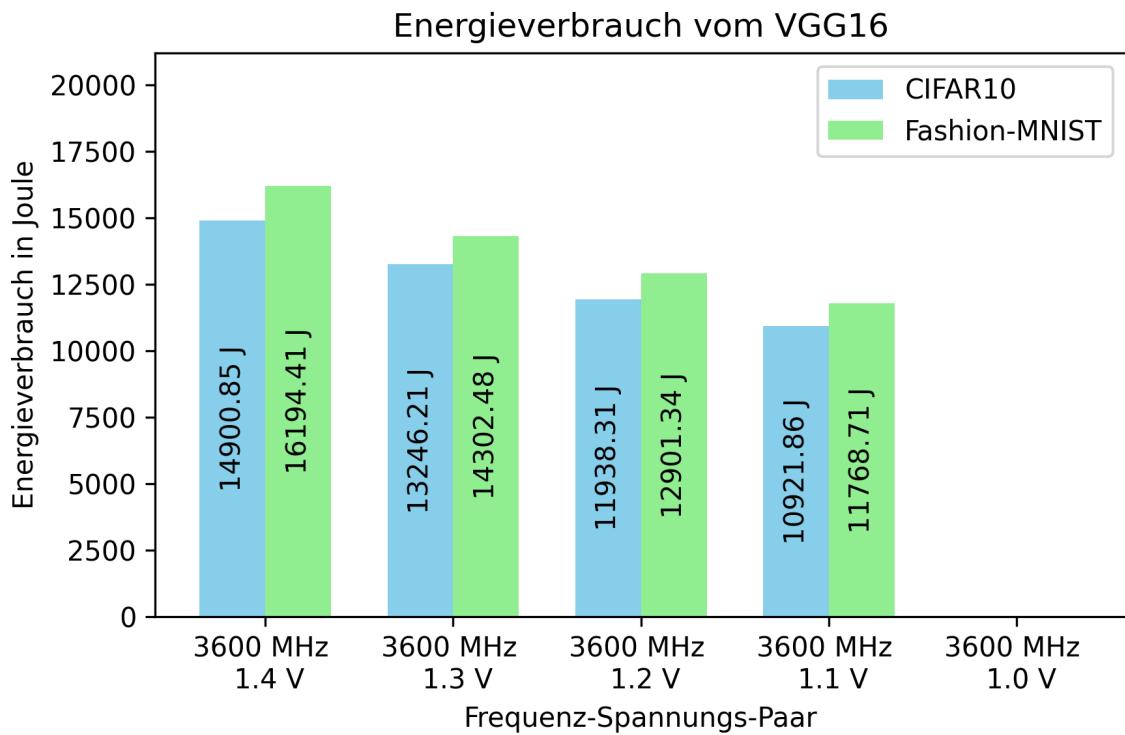


Abbildung 39: Energieverbrauch beim VGG16-Modell bei ausschließlicher Anpassung der CPU-Spannung.

In **Abbildung 39** ist der Energieverbrauch vom VGG16-Modell bei ausschließlicher Anpassung der CPU-Spannung dargestellt. Hier ist ebenfalls zu erkennen, dass sich der Energieverbrauch mit Absenken der CPU-Spannung verringert. Des Weiteren ist der Energieverbrauch beim Training mit dem Fashion-MNIST-Datensatz höher als beim Training mit dem Cifar10-Datensatz. Auch hier ist die Messung bei 3600 MHz und 1,0 V nicht vorhanden.

A.3 Ergebnisse: Genauigkeit

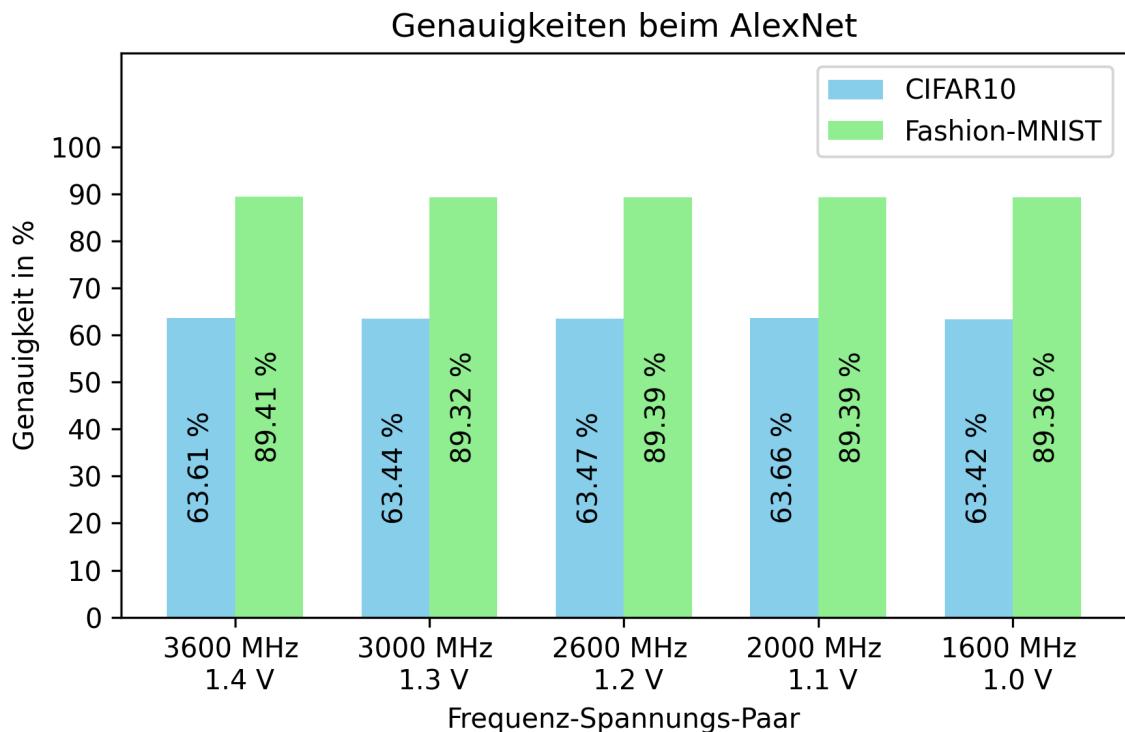


Abbildung 40: Genauigkeiten beim AlexNet bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.

In **Abbildung 40** ist zu erkennen, dass die Genauigkeiten beim Verringern der maximalen CPU-Frequenz und -Spannung nahezu unverändert bleiben und nur eine Abweichung von weniger als 1 % aufweisen. Des Weiteren ist festzustellen, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz etwa 63 % und beim Training mit dem Fashion-MNIST-Datensatz 89 %.

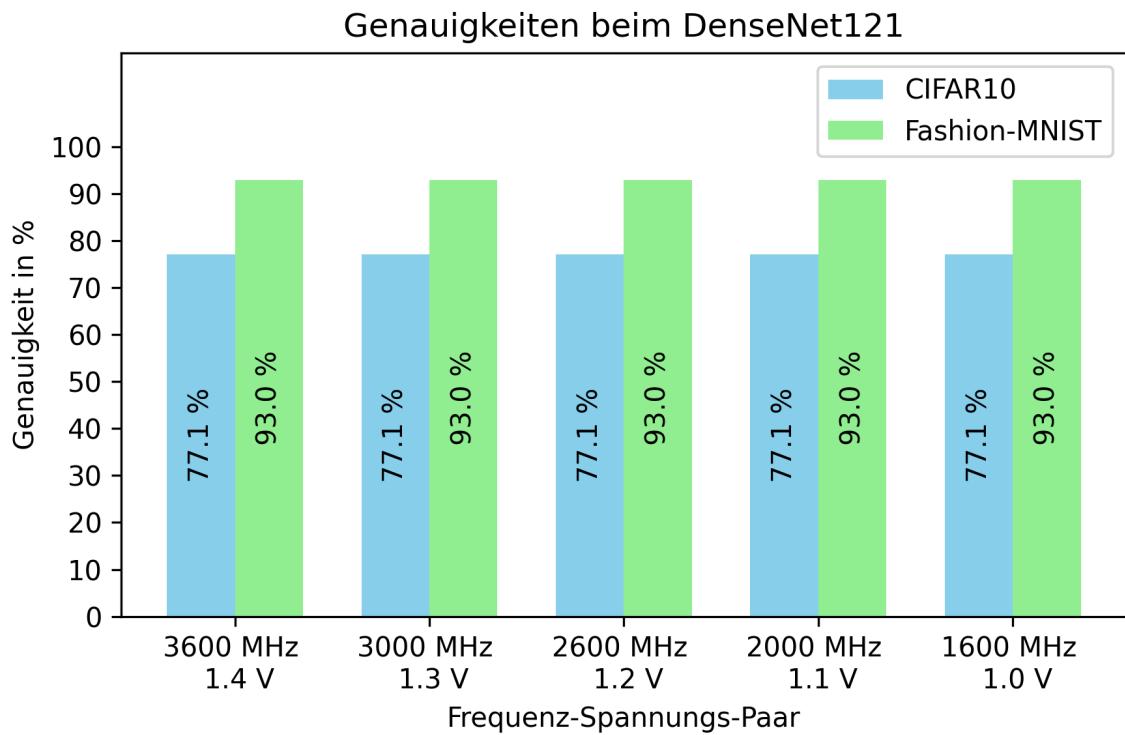


Abbildung 41: Genauigkeiten beim DenseNet bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.

In **Abbildung 41** ist zu erkennen, dass die Genauigkeiten beim Verringern der maximalen CPU-Frequenz und -Spannung unverändert bleiben. Des Weiteren wird deutlich, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz 77,1 % und beim Training mit dem Fashion-MNIST-Datensatz 93 %.

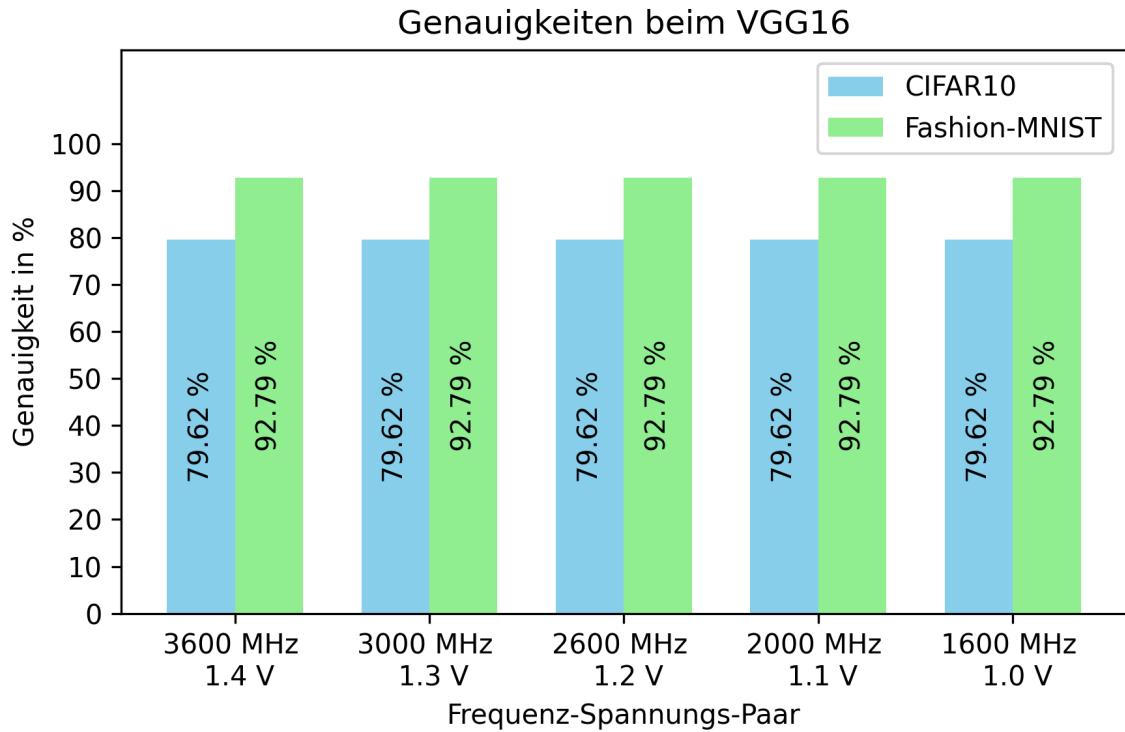


Abbildung 42: Genauigkeiten beim VGG16 bei unterschiedlichen CPU-Frequenzen und CPU-Spannungen.

In **Abbildung 42** ist zu erkennen, dass die Genauigkeiten beim Verringern der maximalen CPU-Frequenz und -Spannung unverändert bleiben. Des Weiteren ist festzustellen, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz 79,62 % und beim Training mit dem Fashion-MNIST-Datensatz 92,79 %.

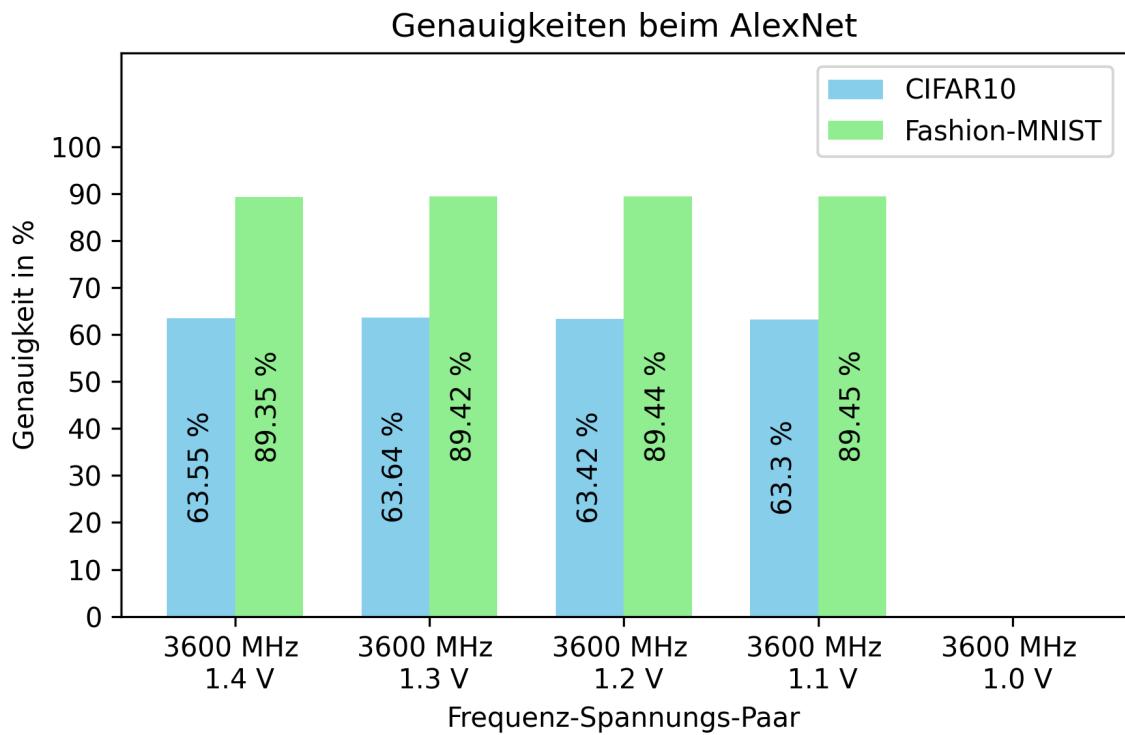


Abbildung 43: Genaugkeiten beim AlexNet bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.

In **Abbildung 43** ist zu erkennen, dass die Genaugkeiten beim ausschließlichen Verrin- gern der maximalen CPU-Spannung nahezu unverändert bleiben und nur eine Abwei- chung von weniger als 1 % aufweisen. Des Weiteren wird deutlich, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz etwa 63 % und beim Training mit dem Fashion-MNIST-Datensatz 89 %.

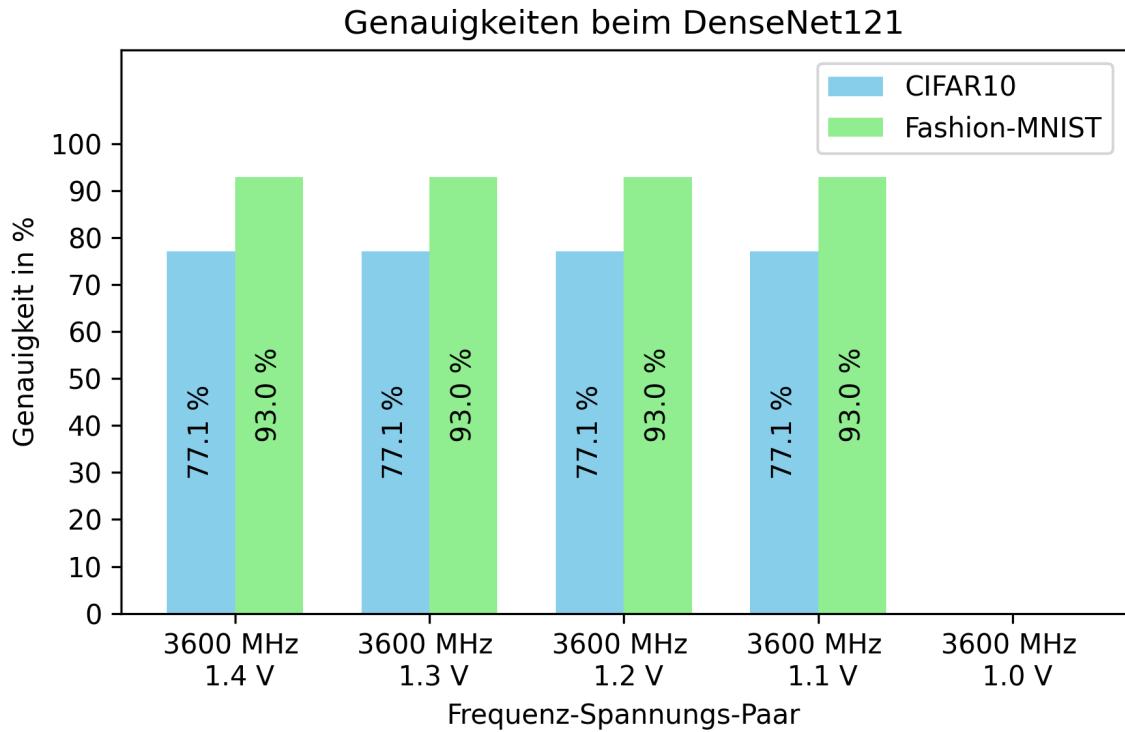


Abbildung 44: Genauigkeiten beim DenseNet-Modell bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.

In **Abbildung 44** ist zu erkennen, dass die Genauigkeiten beim ausschließlichen Verringern der maximalen CPU-Spannung unverändert bleiben. Des Weiteren ist festzustellen, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz 77,1 % und beim Training mit dem Fashion-MNIST-Datensatz 93 %.

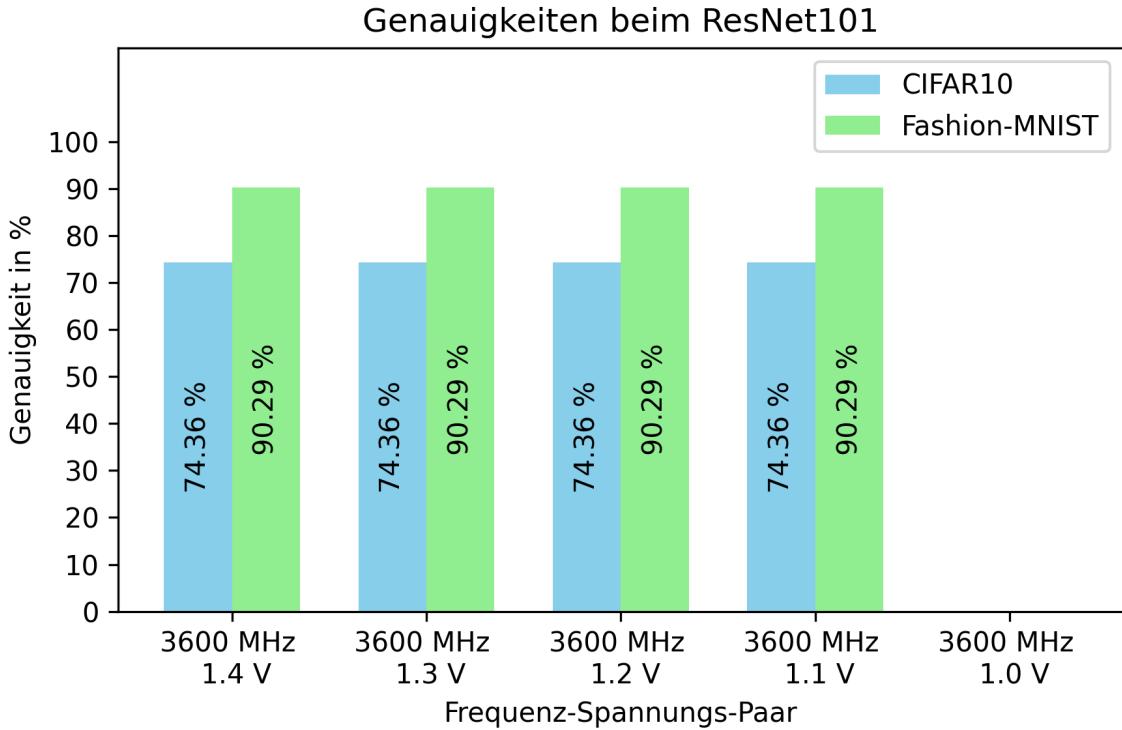


Abbildung 45: Genauigkeiten beim ResNet-Modell bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.

In **Abbildung 45** ist zu erkennen, dass die Genauigkeiten beim ausschließlichen Verringern der maximalen CPU-Spannung unverändert bleiben. Des Weiteren wird deutlich, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz 74,36 % und beim Training mit dem Fashion-MNIST-Datensatz 90,29 %.

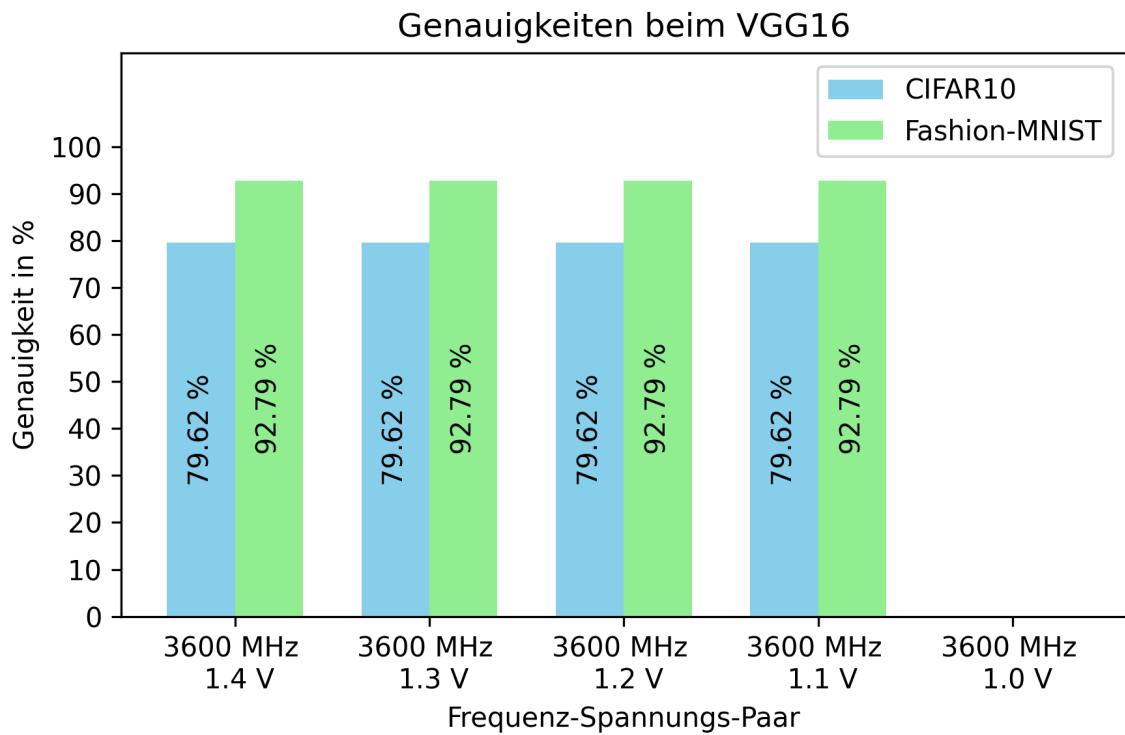


Abbildung 46: Genauigkeiten beim VGG16-Modell bei unterschiedlichen CPU-Spannungen und gleicher CPU-Frequenz.

In **Abbildung 46** ist zu erkennen, dass die Genauigkeiten beim ausschließlichen Verringern der maximalen CPU-Spannung unverändert bleiben. Des Weiteren ist festzustellen, dass die Genauigkeit nach dem Training mit dem Cifar10-Datensatz niedriger ist als nach dem Training mit dem Fashion-MNIST-Datensatz. So beträgt die Genauigkeit beim Training mit dem Cifar10-Datensatz 79,62 % und beim Training mit dem Fashion-MNIST-Datensatz 92,79 %.