Technische Universität Dortmund
Fakultät Informatik

Design Automation for Embedded Systems Group

Entwurf und Training eines Convolutional Neuronal Networks auf dem CIFAR 10-Datensatz

Fachprojekt

Jens Müller, Nour Ajlan

Betreuung: M.Sc. Mikail Yayla

23. März 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Eine der größten Forschungsgebiete und Herausforderungen der aktuellen Informatik ist es, eine Energie-, Ressourcen- und Zeiteffiziente und gleichzeitig sehr genaue Bilderkennung zu erreichen. Um diese Ziele zu erreichen werden zur Zeit vermehrt Neuronale Netzwerke benutzt.

1. 1. Motivation

Das Ziel unserer Gruppe war es, die Bilderkennung des CIFAR 10-Datensatzes mit möglichst guten Erkennungsergebnissen bei einer geringen Laufzeit zu erreichen. Dafür wurden Convolutional Neuronal Networks benutzt, welche auch auf 8-Bit-Werte quantisiert wurden.

Der CIFAR 10-Datensatz wurde ausgewählt, da er mit insgesamt 60.000 Bildern sehr umfangreich ist und auch schon die Klassifizierung der Bilder erhält, außerdam ist dieser oft in der Wissenschaft zum Training benutzt und so sehr erprobt.

CNNs, also Convolutional Neuronal Networks wurden verwendet, da diese als eine besonders effektive Art von Netzwerken für Machine-Learning gelten und auch dieses auch mehrfach belegt wurde.

2. CIFAR 10

CIFAR 10 ist ein Datensatz, welches vom CIFAR, dem Canadian Institute For Advanced Research, zusammengestellt wurde. Dieser wird aufgrund seiner mit 60.000 Bildern sehr großen Datenmenge sehr häufig zur Forschung und Training von CNNs benutzt.

Die Bilder haben ein Format von 32x32 Pixeln und werden in zehn Klassen aufgeteilt: Flugzeuge (airplane), Frösche (frog), Hirsche (deer), Hunde (dog), Katzen (cat), LKWs (truck), PKWs (automobile), Pferde (horse), Schiffe (ship) und Vögel (bird).

Die Bilder werden außerdem in zwei Teilmengen technisch unterteilt, in die Trainingsmenge mit 50.000 Bildern und einer Verifikationsmenge von 10.000 Bildern. Bei jedem Bild wird auch die Klasse, in welche dieses gehört in den Daten hinterlegt.

Beispiele aus dem Datensatz:

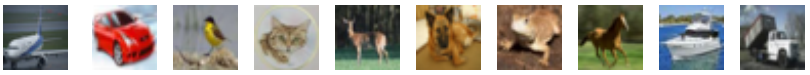

Abb. 1 Abb. 2 Abb. 3 Abb. 4 Abb. 5 Abb. 6 Abb. 7 Abb. 8 Abb. 9 Abb. 10

airplane car bird cat deer dog frog horse ship truck [CIFAR]

3. Neuronale Netzwerke

Ein Neuronales Netz ist eine häufig genutzte Methode im Machine Learning. Neuronale Netzwerke haben sich bei der Entwicklung künstlicher Intelligenzen als ein effektives Werkzeug herausgestellt.

Der Begriff Neuronales Netzwerk bezieht sich darauf, dass diese ähnlich menschlicher Gehirne aufgebaut sind mit Neuronen und Verbindungen zwischen diesen. Im Netzwerk werden die Knoten auch Neuronen genannt und diese sind mit gewichteten Kanten in Schichten miteinander verbunden. Die Werte er Neuronen beziehen sich immer im Zusammenhang von vorhergehenden Neuronen mit den Gewichtswerten der Kanten.

Das System lernt, indem es versucht den berechneten Output-Wert an den vorbestimmten Sollwert des Outputs anzupassen. Dafür werden die vorgegebenen Schichten vom Input aus der Reihenfolge nach wiederholt (Forward Propagation) und mit dem Sollwert mit dem aus einen Rückwärtslauf (Backward Propagation) ermittelten Istwert verglichen wird. Diese Verbesserungen werden durch automatische Veränderung der Gewichte versucht zu erreichen.

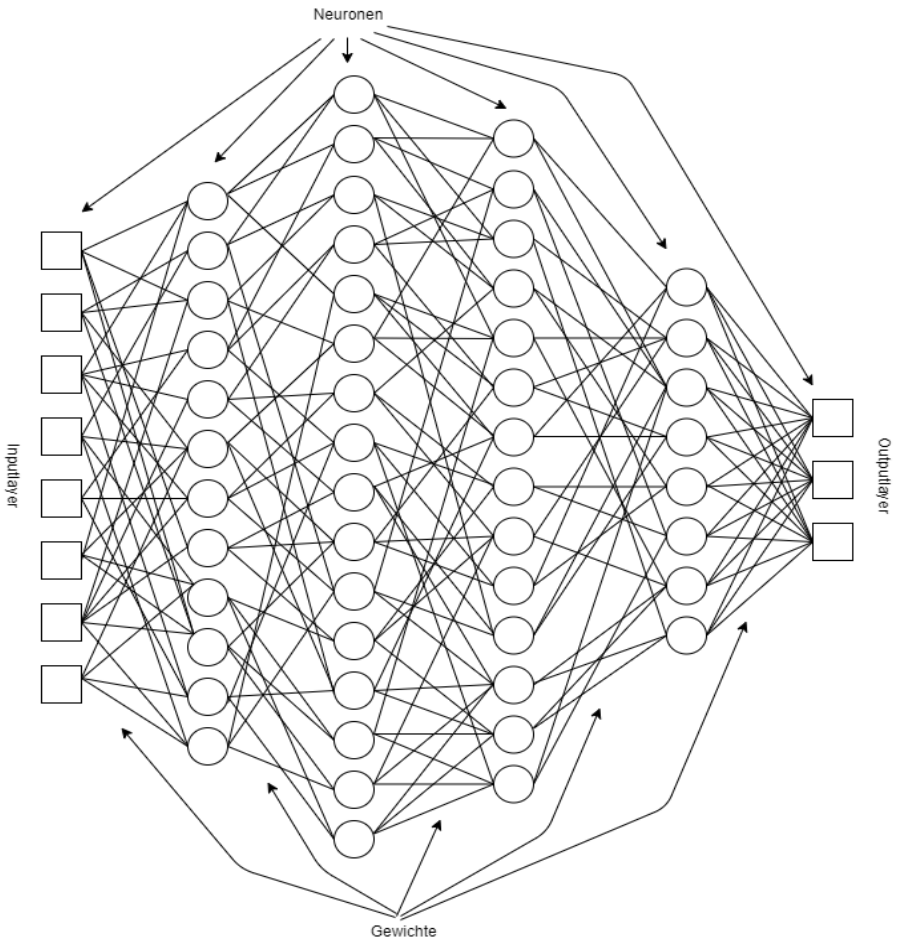

Abb. 11 Neuronales Netzwerk

3. 1. Convolutional Neuronal Network (CNN)

Eine Untergruppe der Neuronalen Netzwerke ist das Convolutional Neuronal Network, CNN genannt. Diese Klasse benutzt besondere Schichten, sogenannte Convolutional-Layer. Auf deutsch heisst Convolution Diskrete Faltung, welche in diesen Schichten stattfindet.

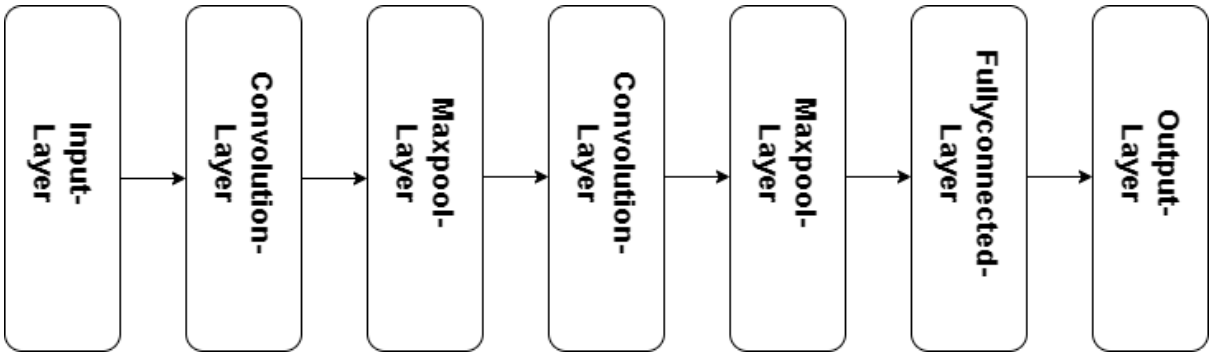

Abb. 12 Beispiel eines CNN in der Schichtweise

3. 2. CNN Layer und Aktivierungsfunktion ReLU:

Ein CNN besteht gewöhnlich aus folgenden Schichten, auch Layer genannt:

3. 2. 1. Convolutionlayer

Im Convolutionlayer, wird mittels eines Faltungskern eine diskrete Faltung durchgeführt auf die Eingangsdaten. Im 2D wird das durch eine Faltungsmatrix die über die über die Eingangsmatrix sich bewegt und in der Überlappung eine Matrizenmultiplikation durchführt. In Pytorch wird aus technischen Gründen die der Faltung ähnliche sogenannte Kreuzkorrelation genutzt.

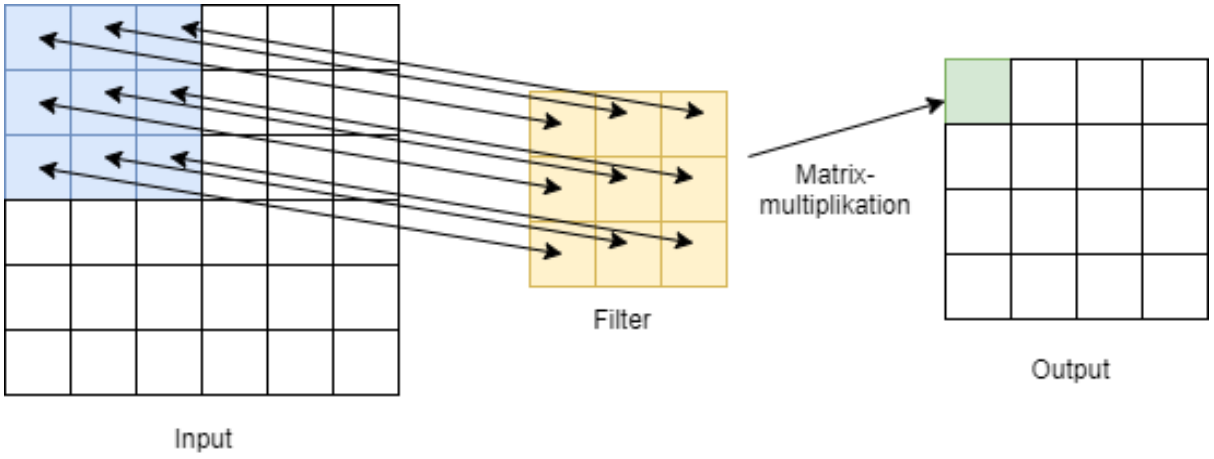

Abb. 13 Allgemeines Beispiel für ein Convolutionlayer

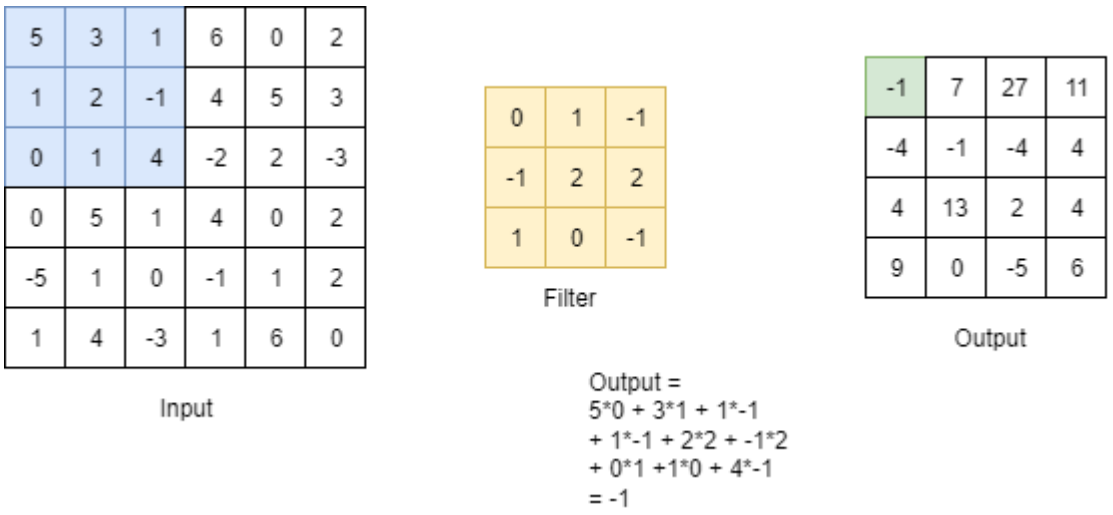

Abb. 14: Beispiel eines Convolutionlayer mit Zahlen

3. 2. 2. Pooling bzw. Maxpooling

Im Poolinglayer werden die Daten reduziert und so verfeinert und ihre Aussagekraft erhöht. Meistens wird ein Maxpooling benutzt, in dem im Kern der Poolingfunktion nur das Maximum der überdeckten Eingangsdaten genommen wird.

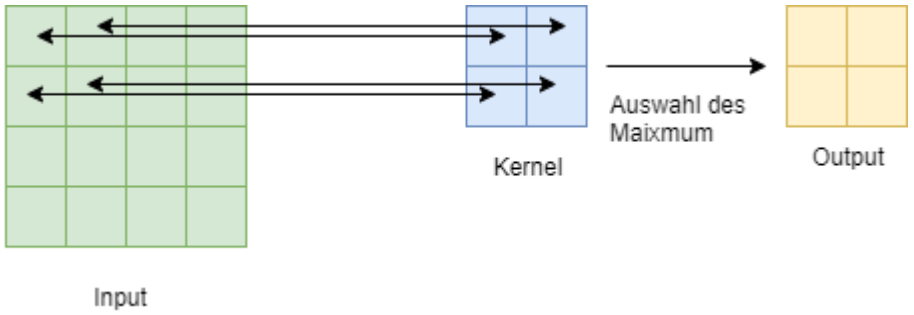

Abb. 15 Allgemeines Beispiel eines Maxpool2D-Layers

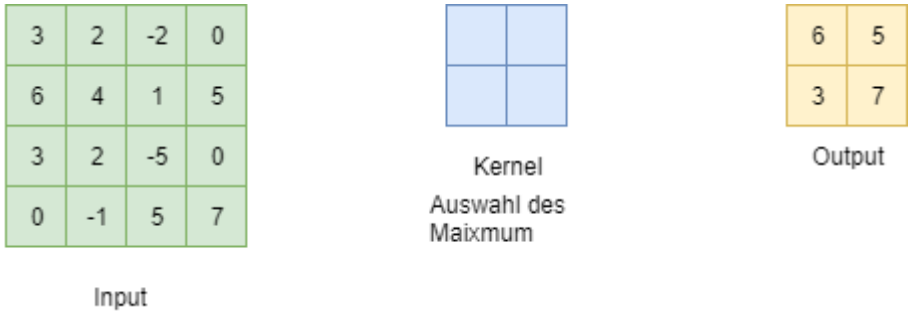

Abb. 16 Beispiel eines Maxpool2D-Layers mit Zahlen

3. 2. 3. Flattening

Im Flatteninglayer werden die Eingabedaten, welche mehrere Dimensionen haben können, zum Beispiel bei der Bilderkennung aus zwei Dimensionen, zu einen eindimensionalen Tensor umgebildet.

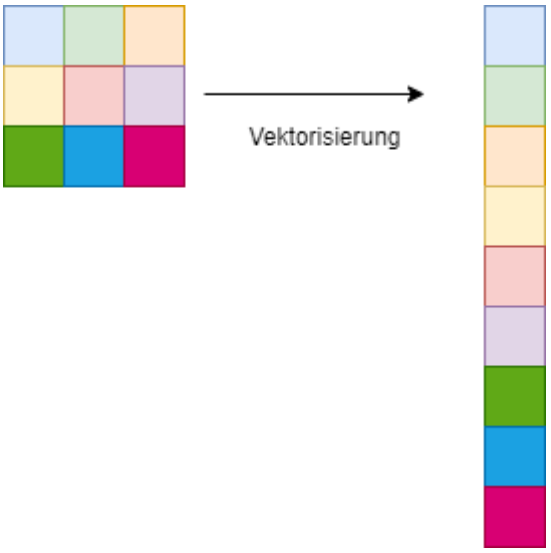

Abb. 17 Beispiel eines Flatteninglayer

3. 2. 4. Fullyconnected Layer

Im Allgemeinen werden im Fullyconnected Layer alle Inputdaten mit den aktivierten Neuronen der nächsten Schicht verbunden. In der Regel sind in CNNs die letzten Layer Fullyconnected Layer bis zum erreichen der endgültigen Ausgabeschicht.

In Pytorch wird in den Fullyconnected Layern die nn.Linear-Funktion benutzt, welche auf einer Matrizenmultiplikation der form o = W * i +b basiert, wobei o die Outputdaten, i die Inputdaten, und b die Biase als Vektoren gegeben sind und W als Gewichtsmatrix.

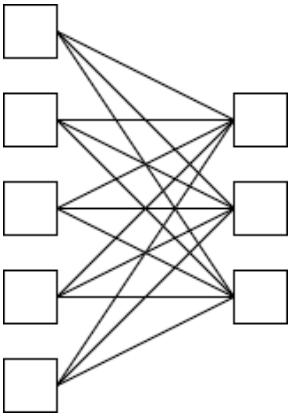

Abb. 18 Schema eines Fullyconnected Layer

3. 2. 5. Aktivierugnsfunkton ReLU

ReLU ist die abkürzung für die Einheit in der die Funktion ausgeführt, der rectified linear unit. Die Funktion an sich heißt Gleichrichter, oder auch Rectifier. Hier werden alle Argumente mit f(x) = max(0,x) mit x als Ausgabe berechnet, also werden negative Werte auf 0 gesetzt.

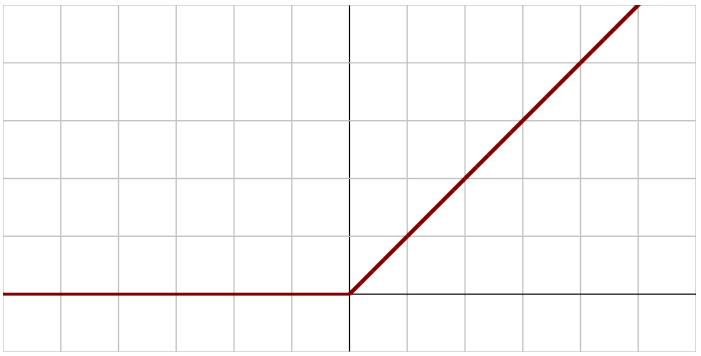

Abb. 19 Aktivierungsfunktion Rectifier [WikiReLU]

4. Die Ausgearbeiten Programme mit CNNs zur Bilderkennung

4. 1. Vorbemerkungen

Aus dem Grund, dass wir sehr beschränkte Kenntnisse in der Programmiersprache Python haben, entschlossen wir uns dazu, dass wir den Großteil des Codes von unserem Programm aus einem Tutorial [TOTU] übernehmen und dieses ein wenig mit unseren eigenen Kenntnissen modifizieren. Die Definition der Layer, Forward Propagation und des Quantisierungsabschnittes wurden von uns ausgearbeitet und ergänzt.

Auch wären Verbesserungen die helfen würden noch eine Eingabe des Benutzers, wie die Lernrate sein sollte und wie viele Epochen berechnet werden sollte. Auch wurde um immer frische Daten haben zu können auf eine Dateladefunktion verzichtet worden, obwohl es eine Speicherfunktion existiert. Außerdem fehlt auch eine Quantisierungsfunktion “nach Außen” und dessen Ausgabe.

Damit Effekte bezüglich der Genauigkeit der Berechnung und Laufzeit wurden zwei CNNs und eine leichte Modifikation von einem CNN von uns selbst definiert und auch das im Tutorial definierte CNN aus Statistikzwecken benutzt.

Die Programme mit je einem CNN sind unter
https://github.com/NussStulle/FachprojektCIFAR10CNN/
abzurufen, wobei CNNToturial.py das CNN aus den Toturial ist und CNNComplex.py, dessen Abwandlung CNNQuadComplex.py und CNNFaster.py die selbstdefinierten CNNs sind.

4. 2. Das Programm kurz erklärt

Zunächst werden alle Bibliotheken importiert und die CIFAR 10 Daten gedownloadet, soweit dies noch nicht geschehen ist.

Anschließend wird eine Bildvorschau erstellt, soweit dieses mit dem Gerät möglich ist, anderenfalls werden nur die Label der ausgewählten Bilder angezeigt.

Darauf folgend werden die CNNs definiert und die Forward Propagation. Diese werden später noch vorgestellt.

Der nächste Abschnitt definiert die benutzte Quantisierung.Und daran schließt sich die Optimierung an, in welcher auch die Lernrate festgelegt wird.

Nachdem zunächst ermittelt wird, ob über die CPU oder GPU gearbeitet wird beginnt das Training des CNN mit einer Schleife über die festgelegte Anzahl der Epochen. Auch wird in der Schleife für jeden Durchgang der Loss berechnet, also die Abweichung vom Sollwert. Als Ausgabe nach der Schleife wird der Loss im Durchschnitt für die 2000 benutzten Minibatches und die Laufzeit der Gesamten Epoche ausgegeben.

Ist das Training abgeschlossen wird dies mit einem Text bestätigt und schließlich die Daten gespeichert.

Darauf Folgt die Evaluation der Daten, dafür wieder Bilder ausgegeben mit deren Label und wofür das CNN diese hält. Anschließend werden alle 10.000 Testbilder überprüft und die prozentuale Korrektheit der Klassenvorhersage zur korrekten Klasse angegeben und ausgegeben. Als letztes wird schließlich dann für Jede Klasse eine separate Güteanalyze der Vorhersage gemacht und die Korrektheit in Prozente ausgegeben.

4. 3. 1. CNN-Layer von CNNComplex.py

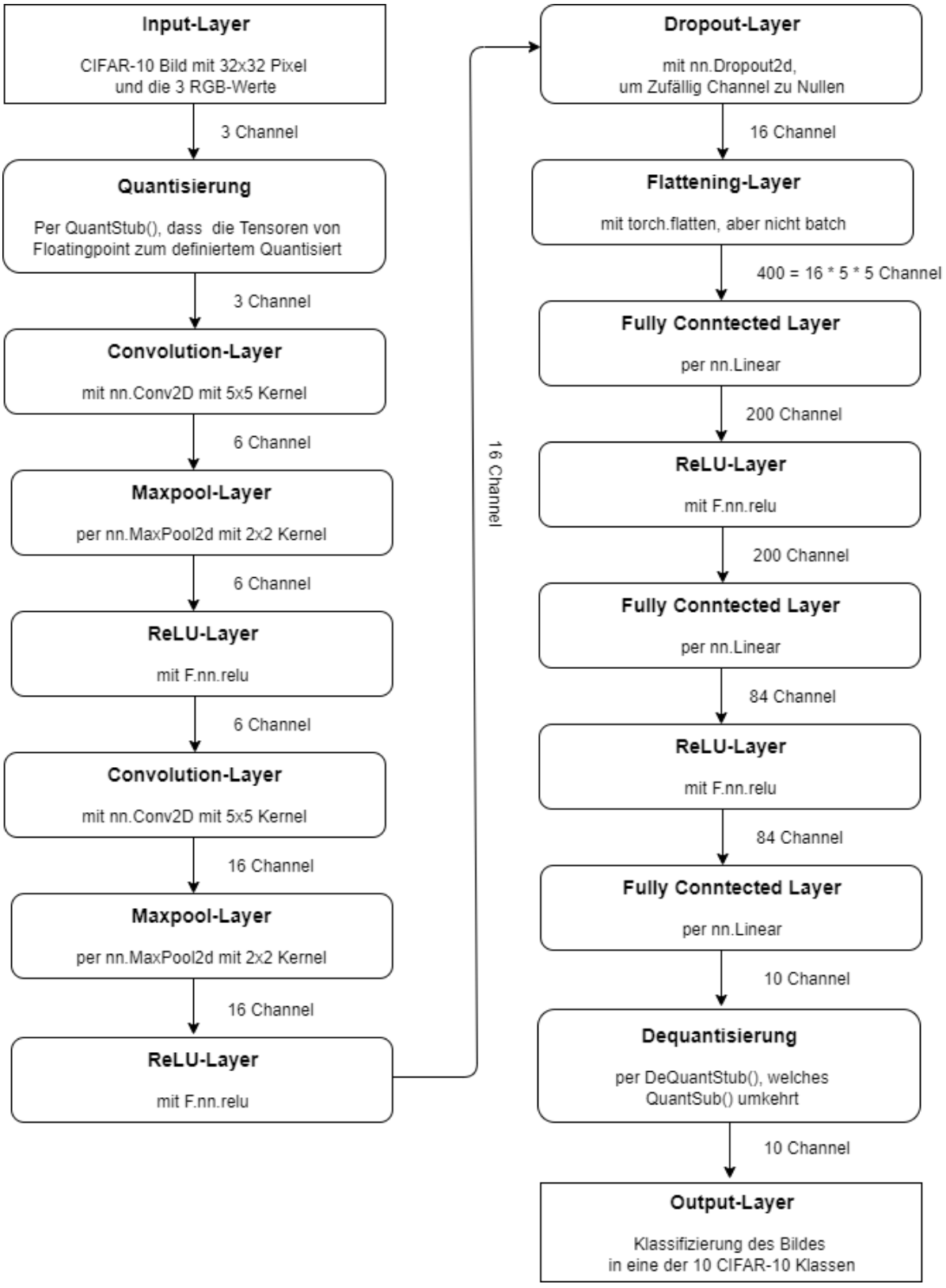

Abb. 20: CNNComplex

4. 3. 2. CNN-Layer von CNNQuadComplex.py

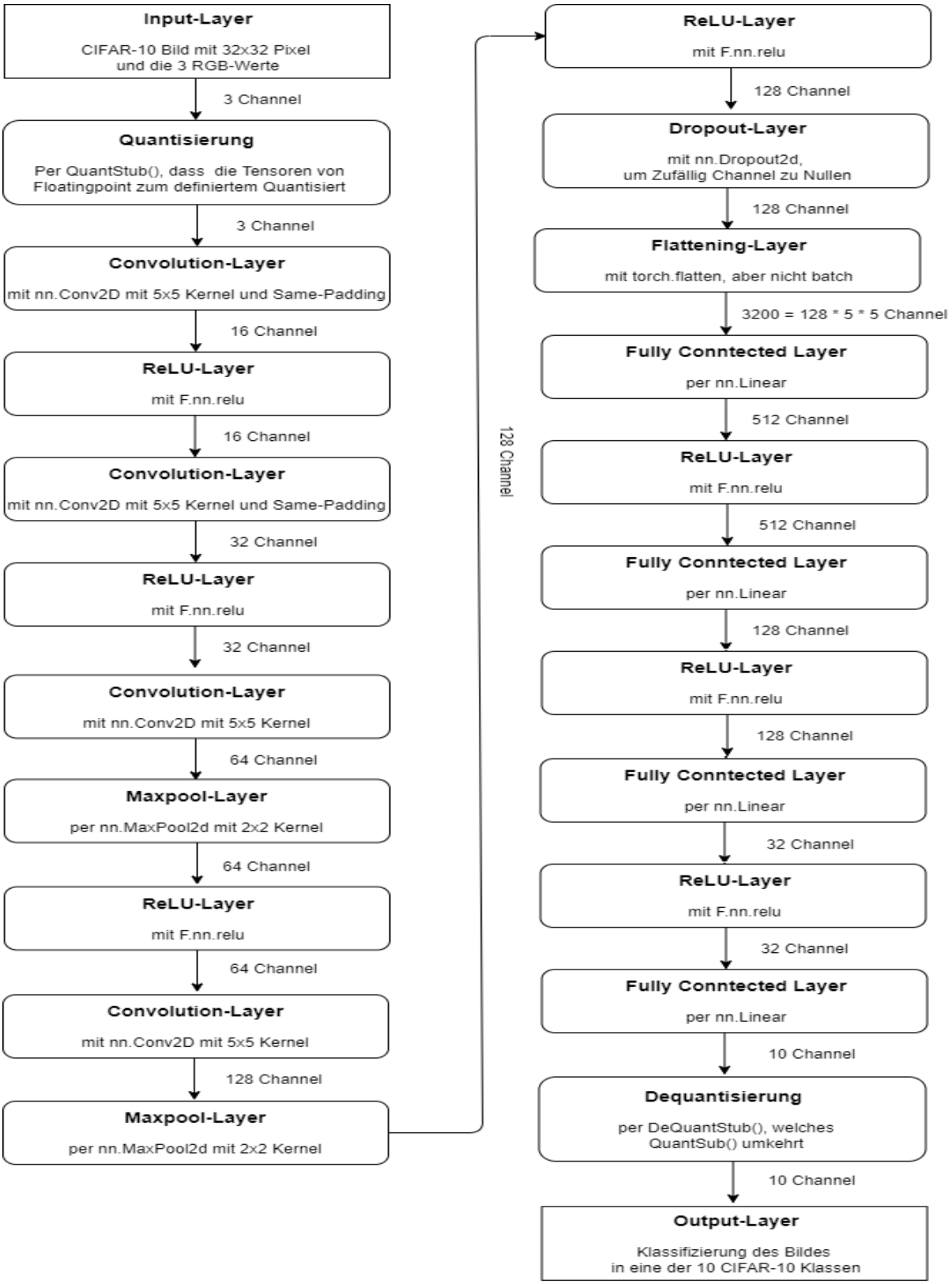

Abb. 21 CNNQuadComplex

4. 3. 3. CNN-Layer von CNNFaster.py

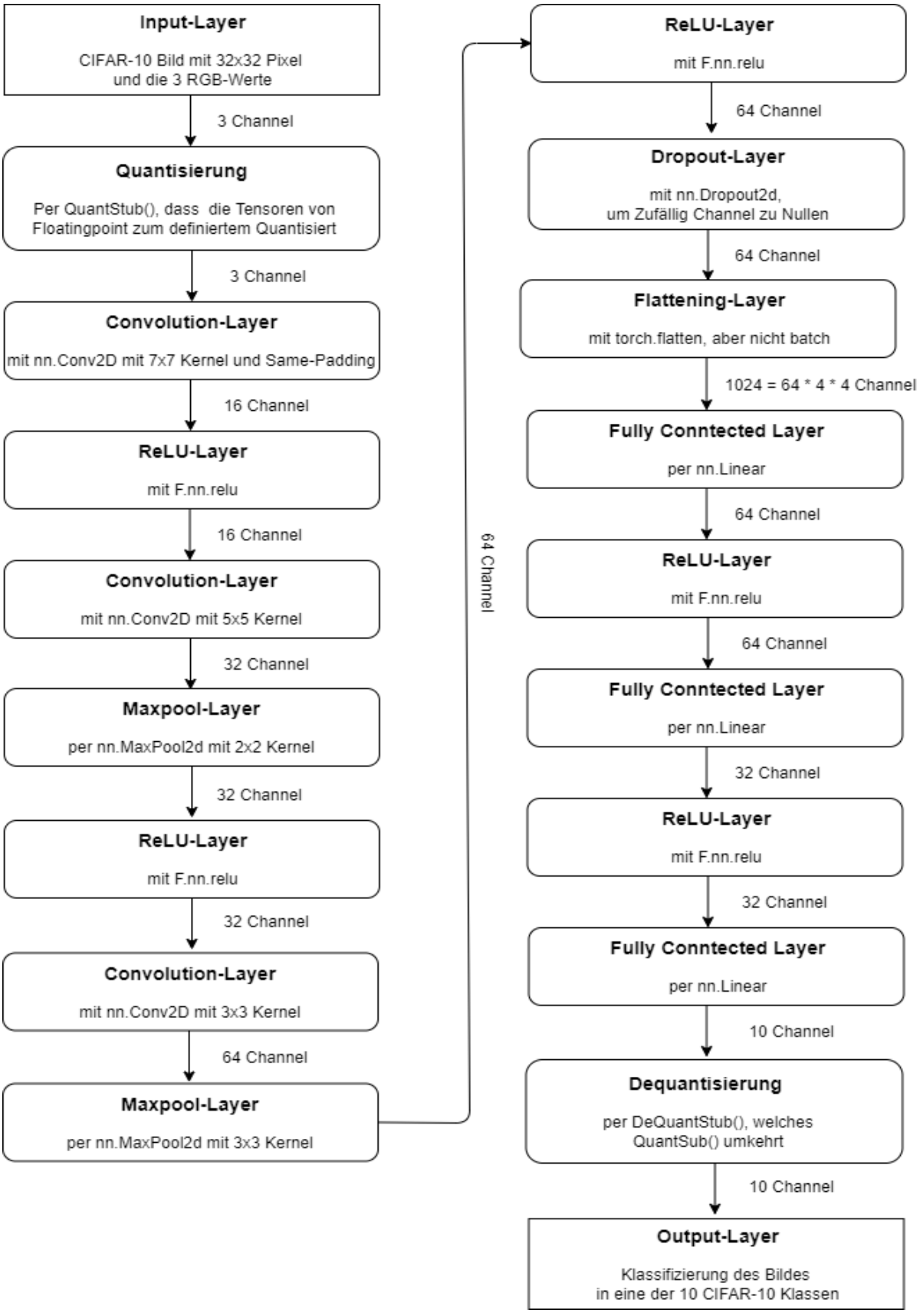

Abb. 22 CNNFaster

4. 3. 4. CNN-Layer von CNNToturial.py

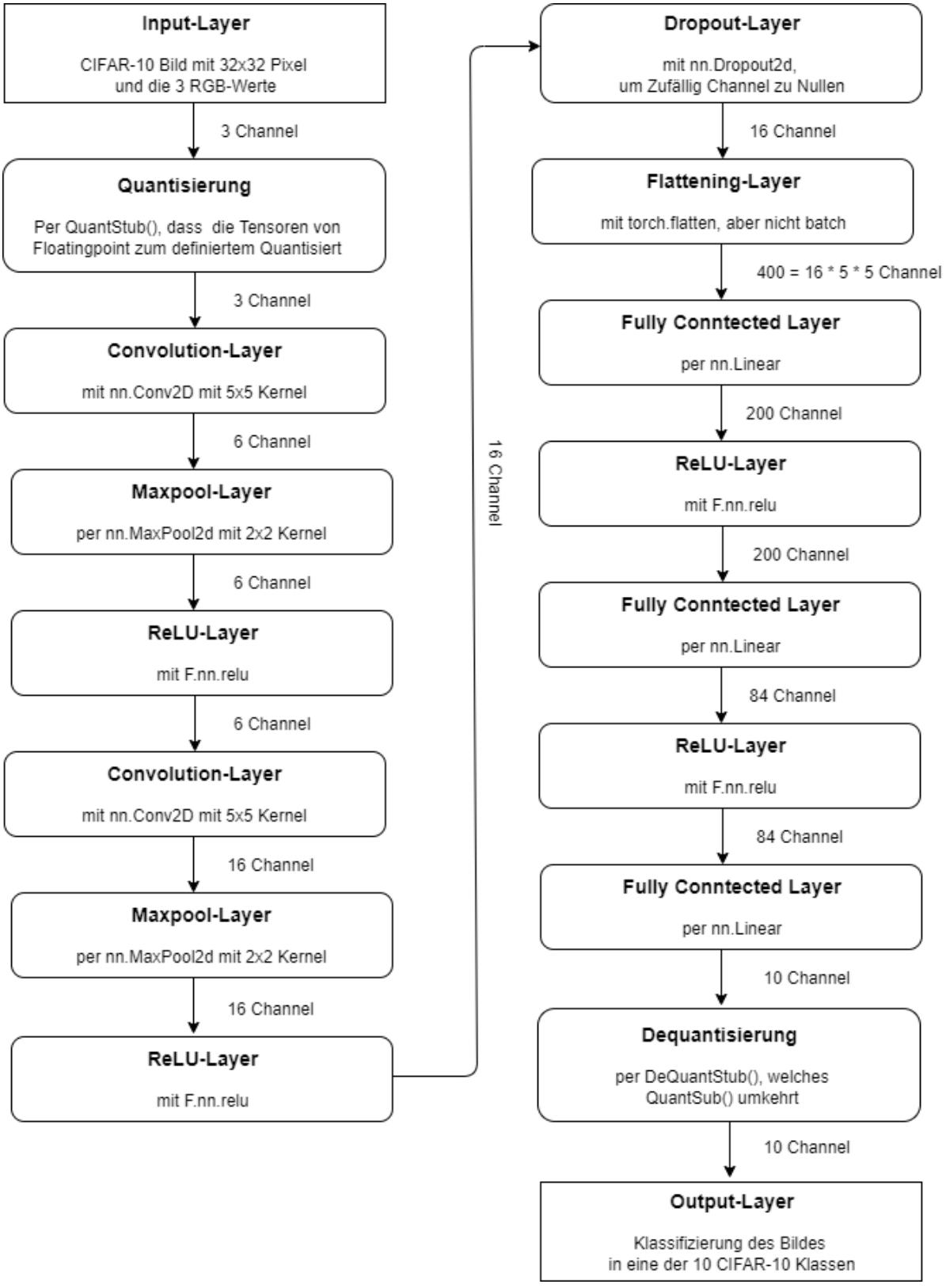

Abb. 23 CNNToturial

4. 4. Statistiken

Zunächst einmal die Durchschnittliche Laufzeit der verschiedenen CNNs:

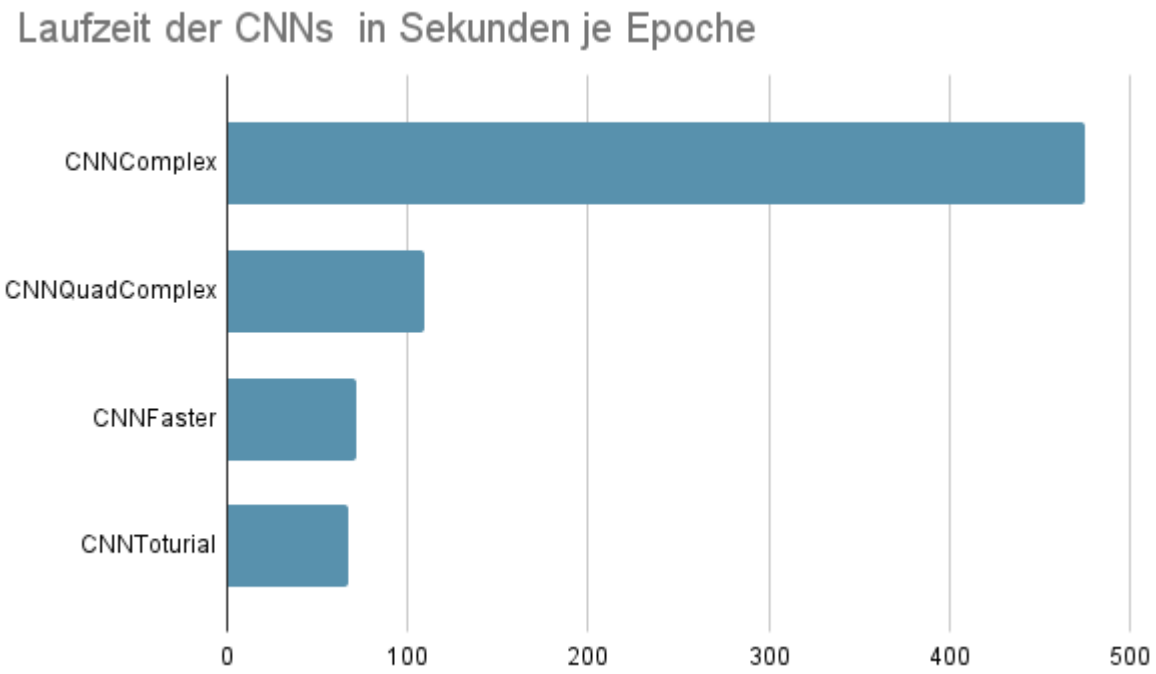

Abb. 24 Laufzeitvergleich

Was interessant zu Beobachten ist, in CNNQuadComplex, welches nur ein Viertel der Ausgangschannel benutzt je Layer ist die Berechnungszeit auch um auf weniger als ein Viertel gesunken.

Als nächstes Vergleiche ich die Korrektheit der jeweiligen CNNs nach 1 Epoche, 10 bis 50 Epochen und noch einmal nach 100 Epochen. Die Werte sind zum einen die Gesamtkorrektheit, die höchste und die niedrigste Korrektheit bei der Klassenweise Evolution. Wobei zu beachten ist, dass CNNComplex eine Lernrate von 0,0001 statt 0,001 hat.

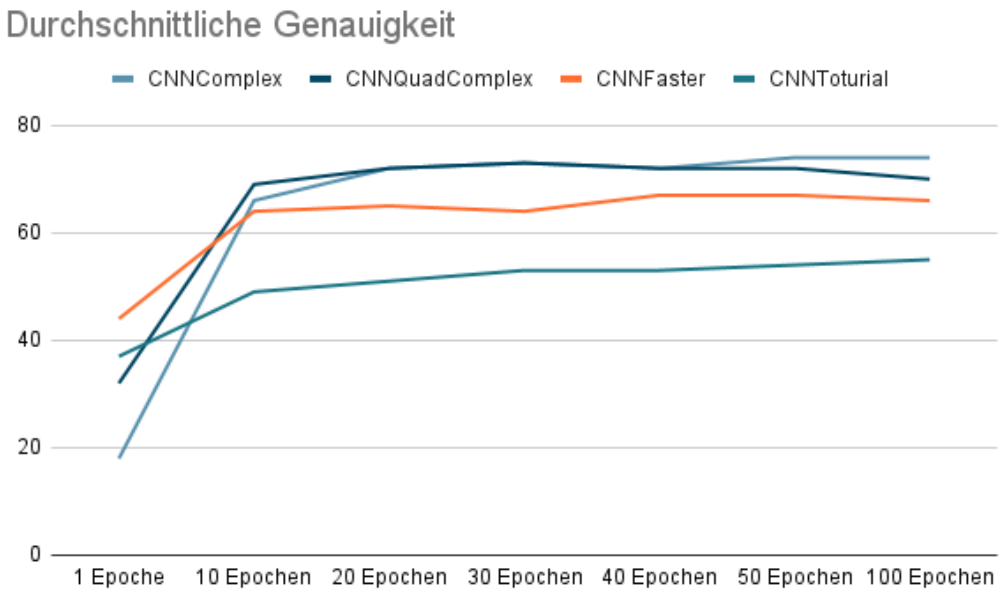

Abb. 25 Durchschnittliche Genauigkeit

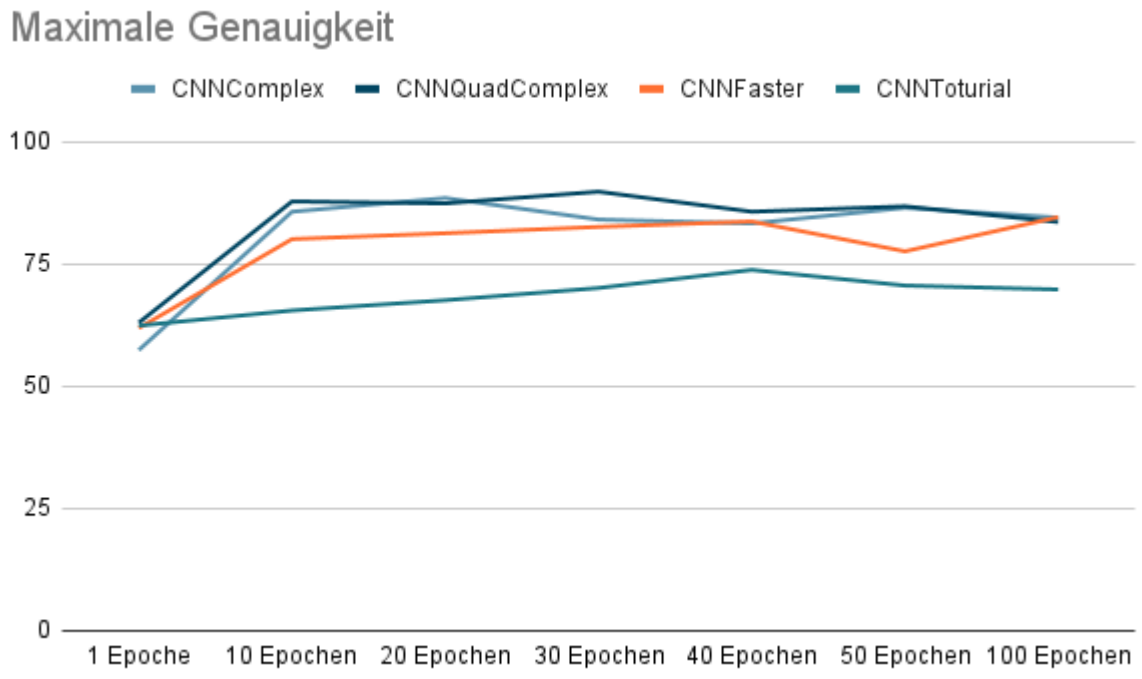

Abb. 26 Maximale Genauigkeit

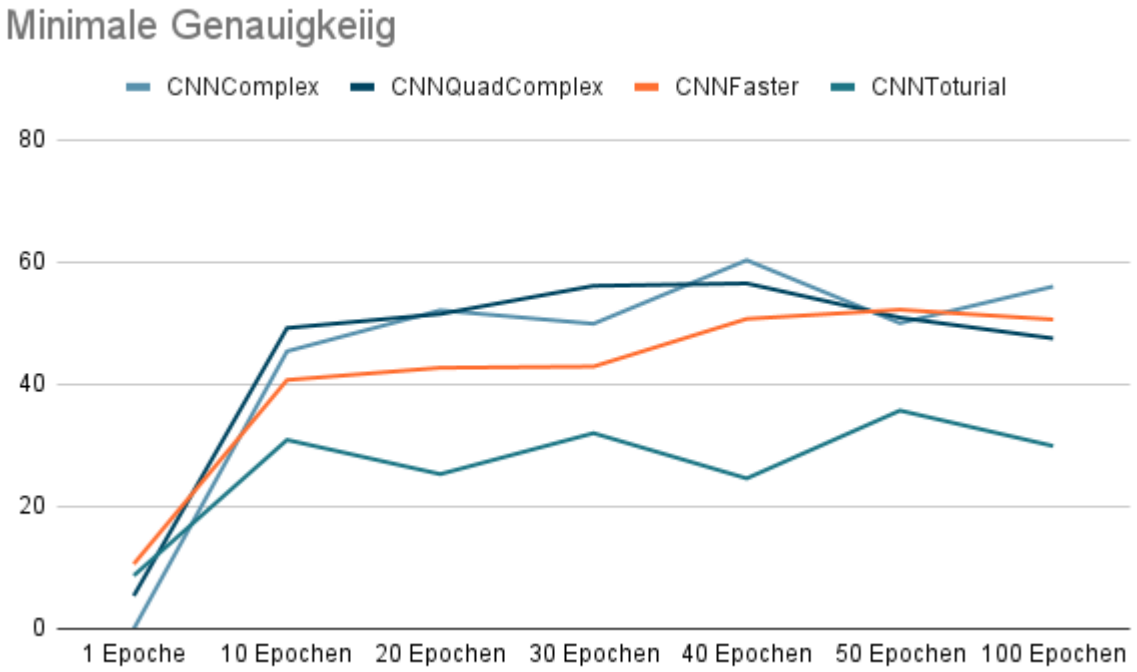

Abb. 26 Minimale Genauigkeit

Hier ist zu Beobachten, dass die Gesamtkorrektheit auch sinken kann, was aber Aufgrund von neuen Gewichten in den Testdurchläufe geschehen kann oder aufgrund von Rundungen. Außerdem ist bemerkenswert, dass nach einer gewissen Anzahl von Epochen die Lernerfolge weniger stark steigen.

4. 5. Fazit

Was beim Testen zum finden eines guten CNN zu sehen war, dass sehr komplexe Netzwerke mit einer zu hohen Lernrate zu einer Überinterpretation zu einer Klasse führen kann und so die Genauigkeit nicht mehr zu erfassen ist, da das CNN dann jedes Bild einer Klasse zuordnet. auch kann eine zu geringe Lernrate bei zu wenigen Epochen zu dem gleichen Effekt führen.

Allgemein kann man sehen, dass auch recht einfache CNNs eine relativ gute Genauigkeit der Erkennung haben können, bei einer schnellen Berechnung, aber will man diese immer genauer haben muss man sie mit mehr Epochen trainieren lassen. Andererseits brauchen komplexe CNNs eine lange Rechenzeit, sind dafür in sehr wenigen Epochen schon sehr genau.

Quellenverzeichnis

[CIFAR] https://www.cs.toronto.edu/~kriz/cifar.html

[WikiReLU] https://de.wikipedia.org/wiki/Rectifier_(neuronale_Netzwerke)

[TOTU] https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

https://de.wikipedia.org/wiki/Convolutional_Neural_Network

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://pytorch.org/docs/stable/quantization.html