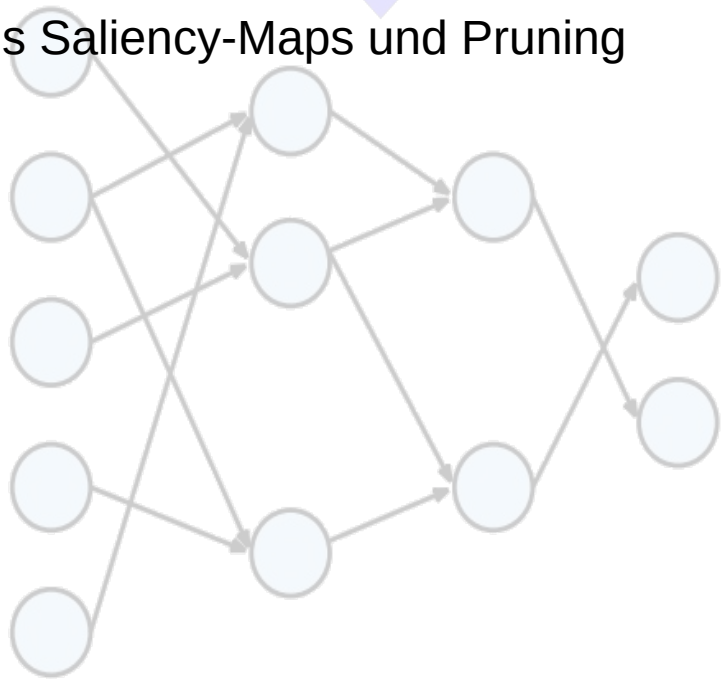
Hardware-Beschleunigung von Edge-AI

Erklärbarkeit und Optimierung von Netzarchitekturen mittels Saliency-Maps und Pruning



Hardware-Beschleunigung von Edge-AI

Erklärbarkeit und Optimierung von Netzarchitekturen mittels Saliency-Maps und Pruning

Verfasser: Béla H. Böhnke

Matrikelnummer: 64048

Betreuer Betrieb: Norbert Link

Betreuer Hochschule: Astrid Laubenheimer

Arbeit: Master Projektarbeit 2

Zusammenfassung

In dieser Arbeit wurden Saliency-Maps, eine Methode aus dem Bereich der Erklärbarkeit neuronaler Netze, genutzt, um eine Optimierung einer vorhandenen Netzarchitektur in Bezug auf Inferenzgeschwindigkeit und Speicherverbrauch durch die Anwendung von Pruning durchzuführen.
Durch das Nutzen von Gradient-Saliency-Maps kann bestimmt werden, wie wichtig einzelne Neuronen und Gewichte für die Netzentscheidung sind. Ohne größere Verluste in der Accuracy kann so die Anzahl an Parametern durch Weglassen unwichtiger Parameter stark reduziert werden.

Abstract

In this work saliency maps, a method from the field of network explainability, were used to optimize a given network architecture in respect of inference speed and memory usage through the usage of pruning.
By using gradient saliency maps, it is possible to determine the importance of single neurons and weights for the network decision. Without major loss in accuracy, the parameter count can be greatly reduced by removing unimportant parameters.

Inhaltsverzeichnis

Abbildungsverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

Abkürzung	Begriff	Erklärung
X		Schicht- oder Netz-Input
Y		Schicht- oder Netz-Output
W	Weight	Gewicht(e)
B	Bias	Schwellwert(e)
SM	Saliency-Maps	
IM	Importance-Maps	
Tanh	Tangens hyperbolicus	
NN	Neuronales Netz	
CNN	Convolutional Neural Network	
CRF	Conditional Random Field	
L2-Loss	Least-Square Error	Loss / Kostenfunktion
SSE	Sum-of-Square Error	Loss / Kostenfunktion
SLIC		Superpixel-Bildung
APoZ	Average Percentage of Zeros	
MI	Mutual Information	
IG	Information Gain	
L1-Loss	Summe der Fehler-Beträge	Loss / Kostenfunktion
L1-Norm	absolute Größe	Regularisierung
Relu	Rectified Linear Unit	Aktivierungsfunktion
MNIST	Modified National Institute of Standards and Technology	Datensatz
MSE	Mean Square Error	Loss / Kostenfunktion
GT	Ground-Truth	Grundwahrheit
FCL	Fully-Connected-Layer	
CI	Cumulated-Importance	
PF	Pruning-Faktor	
PT	Pruning-Schwellwert	
SF	Sparsification-Factor	

Notationen

Notation	Bedeutung
$*$	Skalarprodukt bei Vektoren, ansonsten Multiplikation
$\circ$	elementweise Multiplikation
$\cdot$	Multiplikation mit Skalar
X	Skalar
X	Vektor
$\boldsymbol{X}$	Matrix
X_x	Eintrag x eines Vektors X
$\boldsymbol{X}_x$	Eintrag x einer Matrix $\boldsymbol{X}$

1 Einleitung

Im Teil *2 Grundlagen* werden die Begriffe Pruning und Saliency erläutert sowie Beispiele für mögliche Einsatzgebiete gegeben. Anschließend folgt in Teil *3 Stand der Forschung* eine ausführliche Erläuterung der Literatur zu diesen beiden Gebieten.

In Abschnitt *4 Umsetzung* wird die Umsetzung des praktischen Teils dieser Arbeit besprochen. Dieser Teil hat das Ziel, zu untersuchen, wie geeignet Gradient-Saliency-Maps zum Netzpruning sind, und es werden verschiedene Importance-Maße verglichen.

Dazu gehören Vorüberlegungen die in *4.1 Vorüberlegungen* erläutert sind, in *4.2 Aufbau Testprogramm* ist der Aufbau des Testprogramms geklärt und in *4.3 Implementierungsdetails* werden dann genauere Details des Programms besprochen. Zuletzt ist in *5 Ergebnisse* eine Auswertung der Ergebnisse, die mit dem Testprogramm erzielt wurden, durchgeführt.

2 Grundlagen

2.1 Erklärbarkeit

Ein grundlegendes Problem der neuronalen Netze ist die Erklärbarkeit. Ein neuronales Netz bildet die Eingabedaten (X) auf eine Ausgabe (Y) ab. Das Netz ist hierbei äquivalent zu einer nichtlinearen mathematischen Funktion mit vielen freien Parametern, die während des Trainings des Netzes mit bekannten X und Y im Rahmen eines Optimierungsproblems optimiert werden soll. In Anlehnung an natürliche neuronale Netze werden die freien Parameter hierbei als als Gewichte / Weights (W) und Schwellwerte / Biases (B) bezeichnet. Sind diese gefunden, kann das neuronale Netz im Idealfall für noch unbekannte X die passenden Y berechnen. Durch die hohe Anzahl an Parametern der mathematischen Funktion ist es für einen Menschen allerdings nur schwer nachvollziehbar, welche Muster in den Eingabedaten X zur Entscheidung des Netzes und zur Ausgabe der entsprechenden Y geführt haben.

Dies ist ein großes Problem. Das neuronale Netz hat einen komplexen Sachverhalt erlernt, beinhaltet in seinen W und B also Wissen. Es kann dieses Wissen aber nicht so präsentieren, dass es auch zur Wissenssteigerung des menschlichen Wissens dienen kann oder zumindest zur groben Erklärung und Transparenz der Entscheidungsfindung.

Ohne Transparenz bei der Entscheidungsfindung ist ein neuronales Netz allerdings in vielen Bereichen nicht einsetzbar, insbesondere in Bereichen, in denen das Wohl von Menschen von der Entscheidung abhängt. Zudem können durch die Erklärung von Entscheidungen frühzeitig Fehler in der Entscheidungsfindung festgestellt werden.

Das Themenfeld der Erklärbarkeit von neuronalen Netzen beschäftigt sich daher mit der Entwicklung von Algorithmen und Netzarchitekturen, die für gegebene Ausgabedaten Y beschreiben, welche Daten aus X die Entscheidung herbeigeführt haben und wenn möglich, warum.

In dieser Arbeit geht es insbesondere darum, festzustellen, welche Bereiche im Input jeder Schicht besonders wichtig für die Ausgabe des Netzes sind. Hierfür können Saliency-Maps bzw. Importance-Maps genutzt werden.

2.2 Pruning

Neuronale Netze haben häufig eine hohe Anzahl an Parametern. Die Technik entwickelt sich dahingehend, immer tiefere neuronale Netze zu verwenden, womit die Parameterzahl ständig zunimmt. Mit der Parameterzahl nimmt aber auch der Speicherverbrauch und die Rechenzeit zu. Zudem besteht die Gefahr, zu viele Parameter zu verwenden, wodurch das Netz den Trainingsdatensatz auswendig lernt. In diesem Fall kann das Netz nicht mehr verallgemeinern, das heißt, neue Bilder können vom neuronalen Netz nicht mehr erkannt werden.

Viele der Gewichte, die in einem tiefen neuronalen Netz vorkommen, sind entweder redundant oder zu spezifisch auf eine Teilaufgabe spezialisiert. Sie sind zum Lösen der Aufgabe also unwichtig.

Pruning beschäftigt sich damit, die Rechenzeit und den Speicherverbrauch zu reduzieren, indem solche unwichtigen Gewichte entfernt werden. Als positiver Nebeneffekt ist das Netz mit weniger Parametern nun auch gezwungen zu verallgemeinern.

Hierfür muss allerdings bestimmt werden, welche Gewichte als wichtig und welche als unwichtig für die Entscheidungsfindung gelten. Hierfür können verschiedene Maße zurate gezogen werden, die auch bei der Erklärbarkeit von neuronalen Netzen zum Einsatz kommen.

2.3 Saliency und Importance

Saliency laut [1]:

> das Hervortreten, Herausstechen eines Reizes; Auffälligkeit eines Ereignisses, einer Sache oder Person.

In Bezug auf neuronale Netze sind „Saliency-Maps“ Karten, die anzeigen, welche Bereiche eines Bilds ein neuronales Netz besonders reizen bzw. welche Bereiche

das Netz als herausstechend für die Entscheidungsfindung sieht. An einer Position S_i in einer Saliency-Map ist also die Saliency der Position X_i eingetragen.

Im Idealfall sollten sich diese wichtigen Bereiche im Bild mit den zu erkennenden Objekten decken.

Es gibt eine Vielzahl von Verfahren um Saliency-Maps zu berechnen, die in *3 Stand der Forschung* betrachtet werden.

Im einfachsten Fall kommt die Oracle-Saliency zum Einsatz. Hier wird zum Erstellen einer Saliency-Map nacheinander jeder einzelne Input auf 0 gesetzt (weggelassen) und beobachtet, wie sich der Loss (die Kostenfunktion des Netzes) verändert. Hierbei wird die Annahme getroffen, dass Inputs die beim Entfernen eine große Änderung am Loss bewirken, wichtiger für die Erfüllung der Aufgabe sind. Der Nachteil an dieser Methode ist der hohe Rechenaufwand. Es muss eine Vorwärts-Berechnung des kompletten Netzes für jeden Input durchgeführt werden.

Saliency-Maps (SM) besitzen sowohl positive als auch negative Werte, je nachdem in welche Richtung sie den Loss beeinflussen. Für die Bestimmung, wie wichtig ein einzelner Input ist, ist allerdings die Richtung egal. Nur die Größe der Änderung ist hier relevant. Als Importance-Map (IM) kann also einfach der Absolutwert der Saliency-Map (SM) gebildet werden.

Formel I: Importance-Map aus Saliency-Map

$$\mathrm{IM} = |\mathrm{SM}|$$

2.4 Verwendung von Saliency- und Importance-Maps

Aus der Erkenntnis, welche Bereiche der Eingabedaten für das Netz besonders relevant für die Ausgabedaten sind, können verschiedene Verwendungsszenarien abgeleitet werden.

2.4.1 Prüfung der Entscheidungsfindung

Zum einen kann man sich ein Bild vom korrekten Arbeiten des Netzes machen. Sollte das Netz z.B. besonders viel Wert auf Bereiche in den Eingabedaten legen, die eigentlich keinen Einfluss auf das Ergebnis haben sollten, kann daraus geschlossen

werden, dass der Trainingsdatensatz ungewollte Korrelationen enthält. Ein bekanntes Beispiel hierfür ist ein neuronales Netz, das Hunde und Wölfe auseinanderhalten soll. Stattdessen unterscheidet es Bilder mit und ohne Schnee im Hintergrund, denn der Trainingsdatensatz zeigte vor allem Bilder von Wölfen im Schnee und von Hunden ohne Schnee. Wölfe ohne Schnee sind für das Netz also Hunde.

2.4.2 Stärkere Überwachung des Trainings

Eine weitere Anwendung ist, bereits während der Trainingsphase die Daten zu nutzen, um die Aufmerksamkeit des Netzes direkt auf wichtige Bildbereiche zu fokussieren. Die Trainingsdaten müssen dazu semantisch annotiert sein. Die Bereiche, auf die das Netz Wert legen sollte, gegenüber gestellt den Bereichen, auf die es Wert legt, können nun als Teil der Fehlerfunktion genutzt werden.

2.4.3 Schwach überwachtes Training

Sehr nahe damit verwandt ist der Einsatz für die Segmentierung. So kann ein neuronales Netz zur Klassifizierung dazu genutzt werden, eine Segmentierung durchzuführen, indem die Bereiche hoher Aufmerksamkeit bezogen auf eine Klasse als Segmentierung der entsprechenden Klasse verwendet werden. Es werden hier nur Annotationen der Trainingsdaten für die Klassifizierung benötigt und trotzdem kann damit ein Modell zur Segmentierung trainiert werden.

2.4.4 Netz Surgery, Spezialisierung auf Teilprobleme

Ein weiterer Anwendungsfall ist der Aufbau bzw. die Optimierung der Architektur eines neuronalen Netzes für spezielle Aufgaben. So kann z.B. für ein bereits vorhandenes Netz zur Klassifikation für alle Schichten berechnet werden, welche Neuronen besonders wichtig für die Klassifikation einer bestimmten Klasse sind, und nur diese behalten werden. Dies wird erreicht, indem die Aufmerksamkeit jedes Neurons, gemittelt über alle Daten des Trainingsdatensatzes, für die gewünschte Klasse bestimmt wird. Mit dieser Information (welche Neuronen sind wichtig zum Erkennen der Klasse) kann das Netz nun zerlegt oder verkleinert werden. Ein Netz, das auf viele Klassen trainiert wurde, kann also zu einem kleineren Netz konvertiert werden, das nur einen Teil der Klassen erkennen muss.

2.4.5 Pruning

Ganz analog dazu ist die Problematik des Pruning. Ein großes Netz soll verkleinert werden, um trotz begrenzter Hardware-Ressourcen noch einen angemessenen Durchsatz zu erreichen, siehe *2.2 Pruning*. Hierbei wird ein Verlust in der Genauigkeit hingenommen. Die Fragestellung ist nun allerdings, welche Gewichte und Neuronen vernachlässigt werden können, ohne wichtiges Wissen zu verlieren. Auch hier ist die Auswirkung jeder Schicht auf die Ausgabeschicht eine guter Richtwert dafür, welche Neuronen wichtig sind und was vernachlässigt werden kann. In diesem Fall sollte die Aufmerksamkeit über alle relevanten Daten aus dem Trainingsdatensatz gemittelt werden.

3 Stand der Forschung

Im Rahmen der Arbeit ist eine ausführliche Literatur-Recherche mit den Schwerpunkten der Bestimmung der Saliency bzw. der Importance in Hinsicht auf Erklärbarkeit, die Bestimmung der Importance in Hinsicht auf Pruning sowie Verfahren zum Durchführen von Pruning oder ähnlichen Techniken anhand eines der Importance-Maße durchgeführt worden. Im Folgenden sind jeweils kurze Zusammenfassungen und Erklärungen der einzelnen Quellen gegeben.

3.1 Saliency im Rahmen der Erklärbarkeit

Die Literatur in diesem Bereich beschäftigt sich insbesondere damit, die Saliency des Inputs in Bezug auf die Klassifikationsentscheidung zu bestimmen. Dieselben Methoden können allerdings auch auf andere Bereiche übertragen werden, die Saliency-Information benötigen, was in dieser Arbeit gemacht wird.

3.1.1 Saliency als Anteil an der Neuronen-Aktivierung

In [2] wird zunächst ausführlich das Prinzip eines Multilayer-Perzeptrons beschrieben. Hierbei wird gezeigt, dass bei einer Sigmoid-Aktivierung ein Input mit dem Wert 0 keinen Beitrag zur Klassifizierung leisten kann.

Anschließend wird die Orakle-Saliency eingeführt, die zur Bestimmung der wichtigsten Inputs einzelne Inputs entfernt und beobachtet, wie sich der Output verändert. Hierbei muss beachtet werden, dass theoretisch auch alle Kombinationen

von Inputs getestet werden müssten, da der Output vom Auftreten mehrerer Inputs zusammen abhängen kann. Dies wäre folglich mit sehr viel Rechenaufwand verbunden.

Eine effizientere Möglichkeit, um die Saliency einzelner Inputs für einen Output zu bestimmen, ist, die einzelnen Terme der Summe eines Neurons, also Input*Gewicht, zu betrachten. Die Anteile dieser an der Neuronen-Summe und somit der Aktivierung des Neurons entsprechen dann der Saliency, siehe *Formel II*.

Formel II: Saliency als Anteil an der Neuronen-Aktivierung

$$S = \frac{1}{X * W} \cdot X \circ W$$

Diese Anteile können Schichtweise bis zur Eingabeschicht zurückverfolgt werden, womit dann der Anteil am Output bekannt ist.

Die Orakle-Saliency wird in [2] als Referenzmaß genutzt, um die Vorhersage zu prüfen.

3.1.2 Saliency als Änderung des Outputs

Die Saliency als Änderung des Outputs beim Weglassen eines bestimmten Inputs wurde bereits in [2] eingeführt, in [3] wird diese Idee erweitert, indem die Änderung des Outputs bei einer Änderung des Inputs betrachtet wird. Dies geschieht, indem die Gradienten in Bezug auf den Input gebildet werden. Dieses Vorgehen ist sehr effizient, da die Gradientenberechnung für die Backpropagation beim Training eines Netzes bereits benötigt werden.

Formel III: Saliency als Änderung des Outputs bei Änderung des Inputs

$$S = \frac{\partial J}{\partial X}$$

Als Output wird hier nicht der normale Trainings-Loss genutzt, sondern ein spezieller „Class Score“, der maximiert werden soll. Dieser „Class Score“ entspricht dem Output des Netzes der zu betrachtenden Klasse vor einer Softmax-Layer (da bei Softmax der Score auch maximiert werden könnte, indem die anderen minimiert werden).

Der Absolutwert des Gradienten des „Class Score“ in Bezug auf den Input wird dann direkt als Saliency verwendet. Der „Class Score“ wird im Folgenden in dieser Arbeit als Pruning-Loss bezeichnet, die Absolutwerte der Gradienten als Importance, wohingegen die Saliency direkt durch die Gradienten gegeben ist.

Die Saliency-Map wird dann in Kombination mit Graph-Cut dazu verwendet, eine Objekt-Segmentierung durchzuführen.

In [4] und [5] wird eine Anpassung des Verfahrens aus [3] vorgenommen. Anstatt die Gradient-Saliency-Map nur für den Input zu berechnen, wird sie in einem CNN für die ersten N Layer-Inputs also die ersten N Feature-Maps des CNN berechnet. Die so entstandenen Maps werden dann auf dieselbe Größe gebracht (Bilineares Upsampling). Als Saliency-Map jedes Layers wird wiederum das Maximum des Absolutwerts aus den Kanälen des Layers gewählt, um eine einkanalige Map zu erhalten. Diese einkanaligen Maps jedes Layers werden zu einer Saliency-Map kombiniert, indem sie über den Tangens hyperbolicus (Tanh) der einzelnen Maps gemittelt werden, siehe *Formel IV*.

Formel IV: Kombination von Saliency-Maps

$$S = \frac{1}{|L|} \sum_{l \in L} \tanh(\alpha \cdot S_l)$$

Es wurde hier außerdem festgestellt, dass eine klassenspezifische Map oft auch noch Bereiche anderer Klassen abdeckt. Aus diesem Grund werden die Maps aller Klassen berechnet und von der relevanten Klasse c die Maps der anderen Klassen x abgezogen, wie in *Formel V*.

Formel V: Saliency-Klassen-Differenz

$$S = \sum_{x \in C \setminus c} max(S_c - S_x, 0)$$

Für Relu-Aktivierungen wird im Paper die Guided-Backpropagation eingesetzt, bei der nur positive Gradienten zurückpropagiert werden, dies scheint bessere Maps zu ergeben.

Die Maps können dann unter der Verwendung von Fully Connected CRF (Conditional Random Field) nachbearbeitet werden, um die Objektausdehnung besser zu bestimmen.

In [6] werden die Saliency-Maps in Bezug auf Objektausdehnung weiter verbessert, indem zusätzlich Bounding-Box-Annotierungen verwendet werden. Diese Veröffentlichung ist für diese Arbeit nicht weiter relevant, da in dieser Arbeit keine zusätzlichen Annotationen verwendet werden sollen.

Auch [7] baut auf Gradient-Saliency auf. Allerdings wird hier nicht direkt der Gradient verwendet, sondern der Input des Netzes in mehreren Schritten mittels Backpropagation so optimiert, dass die Klasse anschließend nicht mehr erkannt wird. Hierzu wird ein spezieller Loss verwendet, auf den Objektpixel einen stärkeren Einfluss haben als Hintergrundpixel. Die Differenz zwischen Originalbild und verändertem Bild entspricht dann der Saliency. Durch die höhere Anzahl an Iterationen wird die Map hier exakter als bei nur einer einfachen Gradientenberechnung wie in [3].

Die Loss-Funktion wird so gewählt, dass Outputs die nicht zur erkannten/untersuchten Klasse gehören konstant gehalten werden. Es wird also als Zielwert der Loss-Funktion der originale Output bei unverändertem Bild gewählt. Der Output, der zur zu untersuchenden Klasse gehört, soll möglichst gering werden (Zielwert = 0) . Die Kosten werden dann einfach als L2-Loss (Least-Square Error) bzw. SSE (Sum-of-Square Error) berechnet.

Es wird dabei angenommen, dass alle Outputs, die nicht zur Klasse gehören, auf den Hintergrund zurückzuführen sind und nur der Output der entsprechenden Klasse auf das Objekt im Bild. Das Bild wird dann so verändert, dass die für die Klasse relevanten Pixel entfernt werden. Von den Bildwerten werden dabei, wie beim Gradientenabstieg üblich, die Gradienten abgezogen. Negative Gradienten werden dabei auf 0 geclippt, damit werden die Werte wichtiger Pixel immer nur kleiner und nicht größer. Das Differenzbild hat damit nur positive Werte.

Zuletzt wird eine Objektsegmentierung durchgeführt, bei der die Saliency-Maps mittels SLIC (Superpixel Bildung) und Low-Level-Features (Farbe, Kontrast, etc.) verbessert werden.

3.2 Importance im Rahmen von Pruning

Als Maß, wie wichtig ein Neuron innerhalb eines Netzes ist, kann betrachtet werden, wie häufig es durch Eingabedaten aktiv wird bzw. wie häufig es inaktiv bleibt. In [8] wird nach diesem Prinzip die Average Percentage of Zeros (APoZ) der Aktivierungen jeder Schicht mithilfe eines Training Samples bestimmt. Dies wird erreicht, indem berechnet wird, wie oft die Aktivierung eines Neurons im Durchschnitt 0 ist. Neuronen mit hoher APoZ werden als unwichtig gesehen und können entfernt werden.

Oft wird beim Training von Netzen ein Regularisierungsterm verwendet, der dafür sorgt, dass die Gewichte klein bleiben. Es wird in [9] davon ausgegangen, dass sehr kleine Gewichtswerte unwichtiger sind und entfernt werden können.

In [10] wird eine Übersicht verschiedener Maße zur Bestimmung der Wichtigkeit einzelner Neuronen gegeben.

Zunächst wird als Referenzmaß die auch in [2] vorgestellte Orakle-Saliency erklärt. Hier wird jedes einzelne Neuron nacheinander entfernt und die Kostendifferenz bestimmt. Neuronen, die beim Entfernen eine geringe Kostendifferenz hervorrufen, können entfernt werden.

Dieses Verfahren hat einen sehr hohen Rechenaufwand und kann in der Praxis deshalb nicht eingesetzt werden. Es kann aber als ideales Referenzmaß verwendet werden.

Als weiteres Maß wird „Minimum Weight“ vorgestellt. Dieses ist angelehnt an [9], aber auf Neuronen-Ebene anstelle von einzelnen Gewichten. Es wird hier die Höhe des durchschnittlichen Gewichts für jedes Neuron bestimmt und die Neuronen mit dem geringsten durchschnittlichem Gewicht entfernt.

Diese Methode ist sehr einfach und hat einen sehr geringer Rechenaufwand. Sie kann außerdem mit der L1- oder L2-Regularisierung unterstützt werden, liefert aber auch weniger gute Ergebnisse.

Eine weitere Methode basiert auf der Verwendung der Aktivierung der Neuronen. Neuronen, die selten aktiviert werden, sind nicht so wichtig und können entfernt werden. Als Maß kann die Mittlere Aktivierung oder die Standardabweichung der

Aktivierung jedes Neurons bestimmt werden. Hier kann man auch APoZ aus [8] einordnen.

Ein Problem hierbei ist, dass in frühen Schichten sehr ähnliche Werte für alle Neuronen geliefert werden.

Mutual Information (MI) oder für vereinfachte Berechnung Information Gain (IG) beschreibt die Abhängigkeit einer Variablen von einer anderen. Es wird die Abhängigkeit des Outputs des Netzes von der Aktivierung jedes Neurons für ein Trainings Sample bestimmt, also wie oft eine hohe Aktivierung zusammen mit einem hohen Output auftritt. Neuronen mit geringer Abhängigkeit werden entfernt.

Beim in [10] neu eingeführten Maß Taylor Expansion / Impact wird Pruning als Optimierungsproblem betrachtet. Ziel ist eine bestimmte (und geringere) Anzahl an Parametern zu finden, die die absolute Differenz des Loss möglichst gering halten. Wobei der neue Loss nach Entfernen eines Parameters (eines Neurons) durch ein Taylor Polynom ersten Grads geschätzt wird. Es werden also Parameter bzw. Neuronen entfernt, die einen geringen Gradient in Bezug auf den Loss in Kombination mit einer geringe Aktivierung haben. Für ein Trainings-Sample wird dieser Wert für jedes Neuron gemittelt. In [11] wurde parallel zu [10] dieselbe Technik verwendet und als Impact bezeichnet, um in CNNs die Convolutons nur an bestimmten Stellen der Feature Maps zu berechnen.

Formel VI: Saliency mittels Impact des Entfernen eines Inputs

$$S = \frac{\partial J}{\partial X} \circ X$$

3.3 Pruning-Verfahren

Dropout [12] und Dropconnect [13] zählen zwar nicht zum Pruning, doch auch hier werden Neuronen oder einzelne Gewichte entfernt. Dies geschieht allerdings nur mit einer gewissen Wahrscheinlichkeit während des Trainings, um das Netz robuster zu machen und Overfitting zu vermeiden. Während der Inferenz werden dagegen wieder alle Gewichte und Neuronen genutzt, weshalb es dort keinen Geschwindigkeitsvorteil bringt.

In [9] wird ein einfaches iteratives Vorgehen verwendet, bei dem das Netz nach Entfernen einzelner Gewichte neu trainiert wird (Fine-Tuning). Dieser Ablauf wird wiederholt, bis die Anzahl der noch verbleibenden Gewichte einen bestimmten Grenzwert unterschritten hat oder die Accuracy unter eine Grenze gefallen ist.

Pruning kann auch ganz ohne Importance-Maß durchgeführt werden, wie in [14]. Hier wird eine Optimierung direkt auf der Adjazenzmatrix der Gewichte durchgeführt. Jedes Layer wird hierbei getrennt optimiert. Ziel dabei ist es, den Output des Layers für die Trainingsdaten gleich zu lassen, aber mit weniger Gewichten zu erreichen. Als Loss pro Layer wird hier der L1-Loss (Summe der Fehler-Beträge) verwendet. Der Vorteil an dieser Lösung ist, dass das Problem, das gesamte Netz zu optimieren, in kleine einfache Teilprobleme zerlegt wird.

Auch in [15] wird ähnlich vorgegangen. Allerdings werden hier nicht einzelne Gewichte, sondern ganze Filter (Neuronen) und somit ganze Kanäle in einer Feature-Map entfernt. Es wird versucht, mit den verbleibenden Kanälen in der Folgeschicht einen ähnlichen Output zu erhalten, wobei der L2-Loss eingesetzt wird.

In [8] wird wiederum ein iteratives Vorgehen mit Fine-Tuning eingesetzt, bei dem APoZ als Maß der Importance einzelner Neuronen eingesetzt wird.

In [16] wird Mutual Information als Maß eingesetzt. Die Korrelation zwischen Neuronen wird hier allerdings nicht mit Testdaten bestimmt, sondern mit zufälligem Rauschen. Neuronen, die stark korreliert sind, werden durch ein einzelnes Neuron ersetzt. Das Netz wird iterativ weiter trainiert, bis die Accuracy unter einen festgelegten Grenzwert fällt. Zusätzlich ist es möglich, die Loss-Funktion anzupassen um Korrelation zwischen Neuronen zu fördern.

Das Entfernen von Neuronen und damit Feature-Maps wird in [10] durch eine Maske gelöst, die mit dem Output eines Neurons multipliziert wird. Die Maske wird als „Pruning Gate“ bezeichnet und kann die Werte 0 (Neuron entfernt) oder 1 (Neuron vorhanden) annehmen.

Statt ganze Feature-Maps zu entfernen, werden in [11] nur einzelne Positionen in diesen nicht berechnet („Sparse Feature-Map“). Die fehlenden Werte der unberechneten Positionen werden durch nächste Nachbarn aus der Menge der berechneten Positionen ersetzt. Dazu werden die unberechneten Positionen für ein Test-Sample evaluiert und mit zufälligen bleibenden berechneten Positionen mittels Euklidischer Distanz verglichen. Die unberechnete Position wird dann durch den Wert der berechneten Position mit der geringsten Distanz ersetzt. Die zu berechnenden Positionen werden mittels einer „Perforation Mask“ bestimmt. Diese Maske kann auf verschiedene Weisen bestimmt werden, auf die nun eingegangen wird.

a) Uniform

Zufällig normalverteilt ausgewählte Positionen werden als Maske verwendet.

b) Grid

Ein zufälliges, aber gleichmäßiges Gitter, das mittels einer Pseudozufallszahlenfolge erstellt wird.

c) Pooling Structure

Beim Pooling überlappen einzelne Pooling-Filter. Es werden die Positionen gewählt, die in möglichst vielen Filtern vorkommen, also an denen sich viele Filter überlappen.

d) Impact

Siehe *3.2 Importance im Rahmen von Pruning*. Schätzen der Auswirkung des Entfernens einer Position auf den Loss und Entfernen der Positionen mit geringster Auswirkung. Die Schätzung des Impacts wird mittels Taylor-Polynom ersten Grads durchgeführt, also der Gradient für diese Position multipliziert mit dem Wert der Position. Der Gradient wird dabei für die „Sparse Feature-Map“ ohne Interpolation berechnet. Für die Maske wird der Mittelwert des Impacts über ein Trainings Sample

und die Summe über alle Kanäle an einer Position verwendet. Behalten werden dann nur die N Positionen mit dem höchsten Impact.

Nach dem das Netz perforiert wurde, wird Fine-Tuning durchgeführt.

Eines der neusten Pruningverfahren [17] adaptiert das Vorgehen aus Dropout [12] und Dropconnect [13] auf Pruning. Das Pruning wird hier bereits während des Trainings durchgeführt, die Gewichte werden nicht dauerhaft, sondern mit einer Pruning-Wahrscheinlichkeit entfernt. Diese wird anhand einer Rangfolge der Gewichte eines Neurons bestimmt, die wiederum die L1-Norm (absolute Größe) der Gewichte als Importance verwendet. Hier könnten aber auch andere Maße zum Erstellen der Rangfolge Anwendung finden.

Für alle Gewichte, die in der Rangfolge unterhalb der Position liegen, die durch ein Pruning-Verhältnis vorgegeben ist, wird die Wahrscheinlichkeit erhöht.

Pruning wird dann zufällig anhand der Wahrscheinlichkeiten durchgeführt, und das Netz wird weiter trainiert. In jeder Iteration wird die Pruning-Maske neu anhand der Wahrscheinlichkeiten festgelegt.

4 Umsetzung

Ziel des praktischen Teils dieser Arbeit ist es, zu untersuchen, wie geeignet Gradient-Saliency-Maps zum Netzpruning sind.

Hierzu werden im Folgenden Vorüberlegungen zum Aufbau eines Testprogramms, das zur Validierung verschiedener Importance-Maße und zur Durchführung von Pruning dient, ausgeführt. Anschließend wird der Aufbau des Testprogramms detailliert erläutert, und zuletzt werden Implementierungsdetails besprochen.

4.1 Vorüberlegungen

Zum Beginn der Arbeit galt es festzulegen, welche Netzarchitektur mit welchen Aktivierungsfunktionen eingesetzt werden und mit welchem Datensatz gearbeitet werden sollte.

4.1.1 Aktivierungsfunktion

Als Aktivierungsfunktionen wurden Sigmoid, Tanh und Relu getestet, es wurde schließlich Tanh gewählt. Beim Testen der Aktivierungsfunktionen wurden Testbilder verarbeitet und die Accuracy sowie die Importance-Maps bewertet. Trotz des verbreiteten Einsatzes von Relu, wies Tanh die höchste Accuracy auf, Relu die niedrigste.

Bei der Bewertung der Importance-Maps wurden die zuvor festgelegten Kriterien, siehe Abschnitt *4.1.3 Importance-Map,* zurate gezogen. Bei der Relu-Aktivierung wurden hier die schlechtesten Maps erzeugt, was vermutlich auf die Nicht-Differenzierbarkeit im negativen Bereich zurückzuführen ist. Leaky Relu könnte dieses Problem lösen.

Es wurden auch Kombinationen aus verschiedenen Aktivierungen getestet, mit diesen wurden allerdings auch keine nennenswerten Verbesserungen in Bezug auf Accuracy und Importance-Maps erzielt.

4.1.2 Datensatz

Der verwendete MNIST-Datensatz liefert Bilder mit den Werten zwischen 0 und 255, die eine handgeschriebene Ziffer zeigen. Diese ist um ihren Schwerpunkt zentriert

und auf einheitliche Zifferngröße normiert. Zur besseren Verarbeitung werden diese Werte in der Regel auf einen Wertebereich zwischen 0 und 1 transformiert und ohne weitere Veränderung zum Training verwendet. Von diesem Vorgehen wurde in dieser Arbeit allerdings Abstand genommen und stattdessen ein mit Rauschen augmentierter und auf den Wertebereich von -1 bis 1 transformierter Datensatz verwendet. Im Folgenden sind die Gründe dazu aufgeführt.

Im nicht transformierten Bild sind nur die Ziffern selbst mit Werten von größer 0 dargestellt, der gesamte Hintergrund ist gleich 0. Es wurde als Aktivierungsfunktion Tanh gewählt, die Werte zwischen -1 und 1 liefert, wie in *4.1.1 Aktivierungsfunktion* dargestellt.

Zunächst liegt der Wertebereich des Input der ersten Schicht durch die Transformation im selben Wertebereich wie der Input aller Folgeschichten, was eine einheitliche Visualisierung ermöglicht.

Ein Problem, das durch nicht-transformierte Bilder entsteht, ist die Berechnung der Gradienten $\frac{\partial J}{\partial \boldsymbol{W}}$ an Positionen des Inputs mit dem Wert 0. Denn unabhängig von der Änderung eines Gewichts wird sich nach der Multiplikation mit dem Input immer 0 ergeben und sich so auch der Output nicht ändern [2]. Der Gradient wird für diese Stellen 0 sein. Die Gewichte könnten nur an den Stellen, die sich mit einer Ziffer decken, angepasst werden. Durch die Transformation können Gewichte an jeder Stelle des Bilds angepasst werden.

Dieses Problem ist besonders relevant, da über die Gradienten auch die Saliency und so die Importance der Gewichte bestimmt werden kann. Es können folglich im Fall der nicht-transformierten Daten nur hohe (positive und negative) Gradienten an den Stellen auftreten, an denen im Input die Ziffern stehen. Da das Ziel dieser Arbeit auch ist, zu überprüfen, ob sich die Gradienten als Vorhersage für wichtige Inputs und Gewichte eignen, ist es ungünstig, wenn bereits der Input dafür sorgt, dass große Gradienten nur an den Stellen der Ziffern und somit immer nur an vermeintlich wichtigen Stellen berechnet werden können. Zur Überprüfung sollte die Berechnung der Gradienten an möglichst allen Stellen möglich sein und dennoch nur an wichtigen Stellen hohe Gradienten auftreten.

Aus diesem Grund wurde als weitere Maßnahme und um die Klassifikation zu erschweren eine Augmentierung des Datensatzes mit Gaus- und „Salz und Pfeffer“-Rauschen durchgeführt. Das Netz musste so auch erlernen, zwischen Ziffern und zufälligen Aktivierungen zu unterscheiden. Die Gradienten an Stellen mit Rauschen sollten bei einem gut trainierten Netz voraussichtlich trotz hoher Inputs gering bleiben.

Geht man von der (vereinfachten) Annahme aus, dass ein spezialisiertes Neuron der ersten Schicht jeweils Teile einer bestimmten Ziffer erlernt, indem es an den entsprechenden Stellen hohe (positive) Gewichte ausbildet und an allen anderen Stellen geringe (positive und negative) Gewichte, hat das folgende Konsequenzen für nicht-transformierte Daten.

Fall 1: Ein Neuron, das ein Segment einer Ziffer gelernt hat, die gerade auch im Input anliegt, hat an allen hohen Gewichten auch hohe Inputs anliegen, am Rest der Gewichte 0. Die Summe seiner Inputs ist damit positiv. Es feuert (gibt +1 aus), einen entsprechenden Bias größer oder ungefähr gleich 0 vorausgesetzt.

Fall 2: Ein Neuron, das ein Segment gelernt hat, das nicht in der gerade gezeigten Ziffer vorkommt, hat an all seinen Gewichten 0 anliegen und so ist auch die Summe der Inputs 0. Die Gewichte an den Stellen mit hohem Input haben durch ihre geringen Werte einen geringen positiven oder negativen Einfluss. Es würde bei einer Tanh-Aktivierungsfunktion also einen (hohen) negativen Bias benötigen, um eine eindeutige -1 auszugeben.

Fall 3: Ein Neuron, das ein Segment erlernt hat, das nur teilweise durch die Ziffer abgedeckt ist, erhält nur für den abgedeckten Teil ein Signal, die restlichen Inputs liefern 0. Es hat in Summe also ein positives Signal, bei einem Bias größer gleich 0 gibt es eine eindeutige 1 aus, bei einem negativen Bias eine -1 oder etwas Uneindeutiges. Wünschenswert ist hier etwas Uneindeutiges um die 0, also ein negativer Bias.

Die Summe aller Inputs tendiert hier immer zu einem positiven Wert, entsprechend wird in Fall 2 und 3 ein negativer Bias benötigt, um diese Tendenz auszugleichen. Dies steht allerdings im Widerspruch zu Fall 1, in dem ein positiver Bias benötigt wird.

Diese Auswirkung kann durch die Transformation einfach gelindert werden.

Fall 1: Ein Neuron, das ein Segment einer Ziffer gelernt hat, die gerade auch im Input anliegt, hat an allen hohen Gewichten auch hohe Inputs anliegen, am Rest der (geringen) Gewichte -1. In Summe erhält es einen positiven Input.

Fall 2: Ein Neuron, das ein Segment gelernt hat, das nicht in der gerade gezeigten Ziffer vorkommt, hat an all seinen hohen Gewichten -1 anliegen, und so ist auch die Summe der Inputs negativ.

Fall 3: Ein Neuron, das ein Segment erlernt hat, das nur teilweise durch das Bild abgedeckt ist, erhält nur für den abgedeckten Teil ein positives Signal, für den anderen Teil ein negatives. In Summe also etwas Uneindeutiges nahe 0, wie gewünscht.

Der Bias kann hier in allen drei Fällen nahe 0 liegen, es besteht also kein Widerspruch mehr.

4.1.3 Importance-Map

Ziel ist es, die Importance von Inputs und Gewichten des Netzes zu bestimmen. Also den Einfluss auf die Netzentscheidung. Inputs, die die Netzentscheidung stark beeinflussen, gelten als wichtiger als Inputs, die kaum eine Auswirkung auf die Entscheidung haben.

Es wurden hier Recherchen angestellt und verschiedene mögliche Maße zusammen getragen, siehe *3.1 Saliency im Rahmen der Erklärbarkeit* und *3.2 Importance im Rahmen von Pruning*. Diese Maße mussten dann verglichen und eines ausgewählt werden. Der Vergleich wurde auf Basis der Veröffentlichungen, der Intuition hinter den Maßen sowie dem Rechenaufwand und Implementierungsaufwand getroffen. Zuletzt blieben zwei sehr ähnliche Maße, die Gradient-Saliency-Map und der Impact. Um diese zu vergleichen, wurden beide implementiert und die Importance-Maps verglichen, siehe *5.1 Vergleich von Importance-Maßen*. Dazu mussten allerdings zunächst Thesen aufgestellt werden, welche Bereiche in einem Bild überhaupt wichtig sein könnten und diese Bereiche dann mit den durch das Maß bestimmten Bereichen verglichen werden.

Folgende Bereiche wurden im Voraus als vermutlich wichtig identifiziert.

Bereich 1: Die Positionen, an denen die im Input gezeigte Ziffer selbst liegt. Mit einer positiven Auswirkung auf die Klassenentscheidung.

Bereich 2: Die Positionen, an denen Ziffern des Datensatzes liegen, die nicht mit der Ziffer aus dem momentanen Input übereinstimmen. Mit einem negativen Einfluss auf die Klassenentscheidung.

Bereich 3: Positionen an den Kanten der Ziffern, da Kanten häufig zur Erkennung von Objekten verwendet werden. Mit einer positiven Auswirkung auf die Klassenentscheidung.

Zum Pruning werden Importance-Maps für ein Test-Sample zusammengefasst, siehe hierzu *4.3.4 Pruning*. Der MNIST-Datensatz besteht, wie in *4.1.2 Datensatz* beschrieben, aus zentrierten Ziffern. Es ist also anzunehmen, dass die wichtigen Bereiche in der Mitte des Bilds liegen und etwa die Größe der Ziffern haben sollten. Ein weiteres Kriterium zum Vergleich stellt also die Zusammenfassung der Importance-Maps dar.

Nach den Tests wurde entschieden, die Gradient-Saliency-Maps zu verwenden, da diese ähnliche Regionen als wichtig erkennen aber deutlich weniger Rechenzeit benötigen.

4.1.4 Loss-Funktion

a) Trainings-Loss

Als Loss für das Training wurde einfachheitshalber der MSE (Mean Square Error) verwendet. Dieser konnte direkt auf den Netzoutput angewendet werden, unabhängig von der Aktivierungsfunktion der letzten Schicht. Die GT (Ground-Truth) musste dafür nur minimal angepasst werden. Die Eigenschaft, dass mit beliebigen Aktivierungsfunktionen gearbeitet werden kann, war wichtig, da verschiedene Funktionen getestet werden sollten. Aus diesem Grund wurde auch auf die gängige Cross-Entropie oder Softmax verzichtet, die eine Summe aller Outputs von 1 voraussetzt.

Für die Berechnung der Importance wurden zwei verschiedene Loss-Funktionen getestet. Einerseits der Trainings-Loss MSE, andererseits ein spezieller Pruning-Loss der in [3] beschrieben ist.

b) Pruning-Loss

Um zu entscheiden, welche Bereiche wichtig sind, ist die Auswirkung dieser Bereiche auf den Netzoutput von Bedeutung. Das Verhalten des direkten Netzoutputs kann sich vom Verhalten des Loss stark unterscheiden. So kann, z.B. nach einem Softmax-Layer, ein Output auch maximiert werden, indem alle anderen Outputs minimiert werden [3]. Der Pruning-Loss sollte also direkt am rohen Netzoutput berechnet werden.

Es können auch nur die Auswirkungen auf spezielle Teile des Outputs betrachtet werden. Dies ermöglicht es, z.B. Netze, die für viele Klassen trainiert wurden, so zu prunen, dass sie nur noch eine Untermenge der Klassen erkennen, dafür aber auch deutlich kleiner sind.

Hierzu wird der Netzoutput maskiert. So haben nur noch die zu den gewünschten Klassen gehörenden Positionen im Output eine Auswirkung. Es werden dann direkt die vom Input abhängige Änderungen an den entsprechenden Positionen betrachtet. Zum Fine-Tuning nach dem Pruning muss hier eine entsprechend angepasste Loss-Funktion angesetzt werden, die nur in den relevanten Bereichen der GT den Loss berechnet. Dies wurde in dieser Arbeit allerdings umgangen, indem die Anzahl der Klassen nicht reduziert und trotzdem der Pruning-Loss verwendet wurde. So konnte der unangepasste Trainings-Loss weiter verwendet werden und dennoch die Funktion des Pruning-Loss überprüft werden.

4.2 Aufbau Testprogramm

Im Verlauf der Arbeit wurde ein Testprogramm entwickelt, mit dem die Thesen überprüft werden konnten. Der interne Ablauf des Programms ist in *Abbildung 1: Ablauf des Testprogramms* dargestellt.

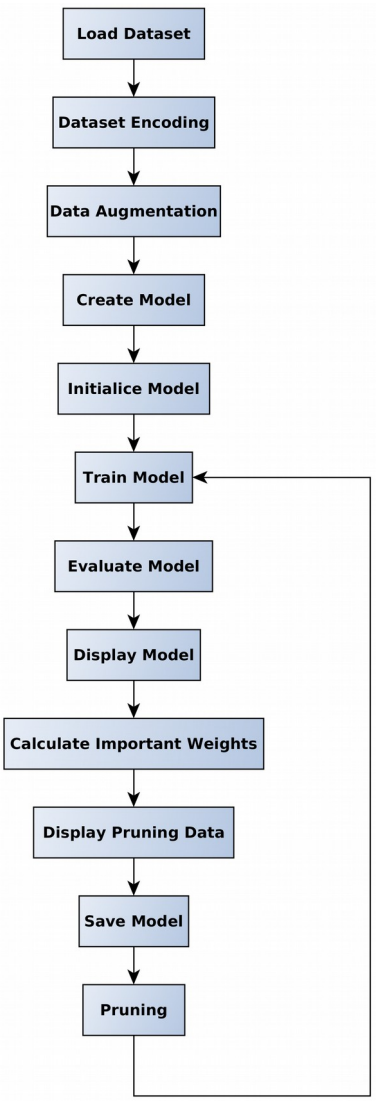


Abbildung 1: Ablauf des Testprogramms

4.2.1 Datensatz

Im ersten Schritt wird der Datensatz geladen. Hier kommt als Datensatz der klassische MNIST-Datensatz zum Einsatz. Er beinhaltet die Ziffern von 0 bis 9 mit den entsprechenden Integer-Werten als Label. Es ist ein sehr einfach zu klassifizierender Datensatz, bei dem alle Ziffern als Graustufenbild der selben Größe, um ihren Schwerpunkt zentriert und auf einheitliche Zifferngröße normiert, gegeben sind.

4.2.2 Encoding

Die Label des MNIST-Datensatz müssen für das Netz in eine verarbeitbare Form gebracht werden. Hierzu werden die Integer-Label in ein One-Hot-Encoding umgewandelt. Es wird ein Tensor der Länge 10 erstellt, bei dem alle Werte auf 0 gesetzt sind, bis auf den, dessen Index mit dem Label übereinstimmt. Zudem wird ein One-Hot-Others-Cold-Encoding erstellt. Hier handelt es sich um eine Variante des One-Hot-Encoding, bei dem anstelle von 0 alle anderen Werte auf -1 gesetzt werden. Dieses Encoding kann mit einer Tanh anstelle einer Sigmoid-Schicht verwendet werden.

4.2.3 Datenaugmentierung

Eine detaillierte Erklärung zu diesem Punkt ist in *4.1.2 Datensatz* zu finden. Hier sollen noch einmal die wichtigsten Punkte aufgezeigt werden.

Der MNIST-Datensatz hat im Graustufenbild nur an den Stellen hohe Werte, an denen die Ziffern stehen, an allen anderen Stellen ist der Wert 0. Durch die Multiplikation mit 0 sind die Aktivierungen und so auch die Gradienten an diesen Stellen immer 0, egal was für Gewichtswerte hier gelernt wurden. Bei der Berechnung der Saliency-Map kann die Saliency also zwangsweise nur in den Ziffernregionen ungleich 0 sein. Tests auf diesen Daten würden also immer zeigen, dass das Netz auf die richtigen Regionen achtet.

Um die Aufgabe nicht so trivial zu machen, wurden die Bilder daher mit einem Gaus- und einem „Salz und Pfeffer“-Rauschen überlagert. Die Bilder wurden außerdem von einem Wertebereich von 0 bis 255 auf einen Bereich von -1 bis +1 skaliert, was den verwendeten Tanh-Aktivierungen Rechnung trägt.

4.2.4 Modell

Ausführlicher ist der Netzaufbau in *4.3.1 Netzaufbau* beschrieben. Einige wichtige Punkte sollen hier zum Grundverständnis dennoch genannt werden. Als Basis-Modell wurden drei Fully-Connected-Layer (FCL) verwendet. Die FCL wurden selbst implementiert, um das Pruning zu ermöglichen. Hierfür wurde der Gewichtstensor mit einem nicht-trainierbaren variablen Tensor als Maske elementweise multipliziert. Der Output jeder Schicht wurde ebenfalls mit einem Masken-Tensor verrechnet. Zu Beginn werden diese Tensoren mit 1 initialisiert. Auf diese Weise ist die Möglichkeit gegeben, ganze Neuronen oder einzelne Gewichte abzuschalten, indem die Maske an der entsprechenden Position auf 0 gesetzt wird.

4.2.5 Modell-Initialisierung

Das Modell wird entweder mit einer vorhandenen Gewichtsdatei initialisiert, oder wenn keine vorhanden ist, zufällig.

4.2.6 Training

Das Modell wird trainiert, bis es nicht mehr konvergiert. Alle 100 Schritte wird eine Evaluierung durchgeführt, bei der die Accuracy für das aktuelle Batch berechnet wird.

4.2.7 Evaluierung

Nach dem Training wird das Modell mit den Validations-Daten evaluiert, hierzu wird die Accuracy berechnet.

4.2.8 Display Modell und Pruning

Die Inputs jeder Schicht, die Gradienten für die Gewichtswerte und Inputs und die Input-Importance werden geplottet, wenn das Flag „display_model“ auf True gesetzt ist. Das Flag „display_pruning“ sorgt dafür, dass die Cumulated-Importance und die Pruning-Masken angezeigt werden.

4.2.9 Berechnung wichtiger Gewichte und Pruning

Diese Berechnungen sind so durchgeführt, wie in *4.3.4 Pruning* und *4.3.3 Weight-Importance* ausgeführt.

4.3 Implementierungsdetails

Im Folgenden wird auf Details in der Implementierung des Testprogramms eingegangen.

4.3.1 Netzaufbau

Zum Testen wurde ein einfaches, dreischichtiges, voll verbundenes Netz verwendet, wie in *Abbildung 2: Aufbau des Test-Netzes* zu sehen. Jedes Gewicht und jeder Output eines Neurons wurde dabei elementweise mit einer Maske multipliziert (im Bild als gestrichelte Linie dargestellt). Mit Hilfe dieser Maske kann Pruning durchgeführt werden.

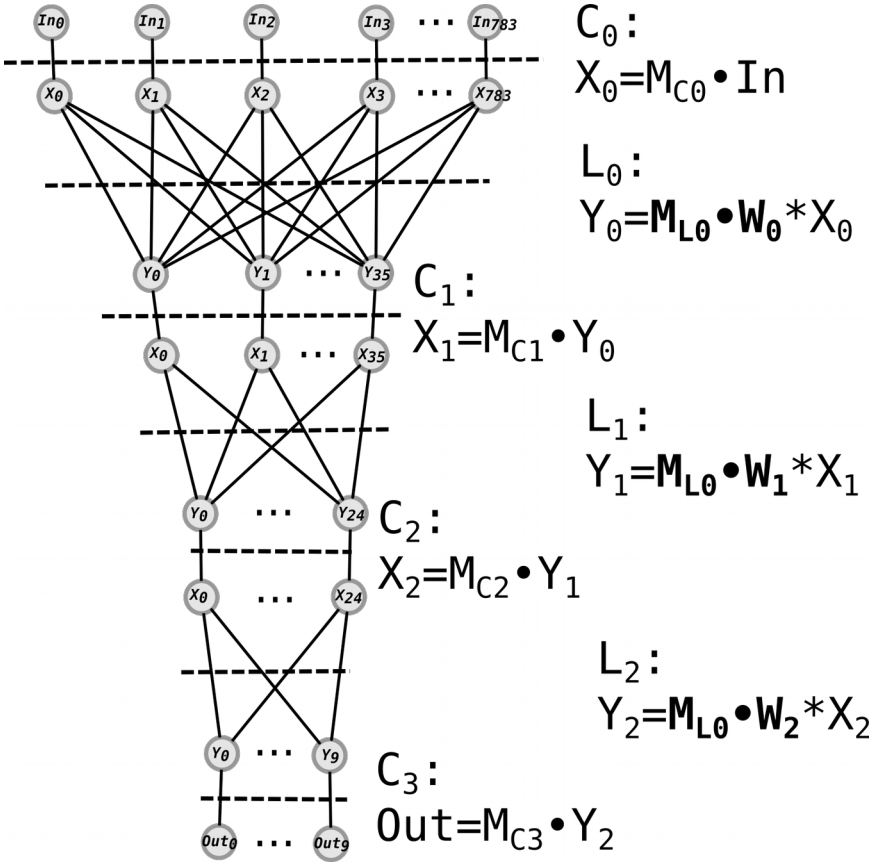


Abbildung 2: Aufbau des Test-Netzes

a) Trainings-Modell

An das Basis-Modell wurde für das Training eine Loss-Funktion angehängt. Es wurde hier der Mean Square Error (MSE) und ein Gradientenabstiegs-Optimierer verwendet.

b) Evaluations-Modell

Zur Evaluierung wurde an das Basis-Modell die Berechnung für die Accuracy angehängt.

c) Pruning-Modell

Das Basis-Modell kann einen vom Trainings-Loss unterschiedlichen Pruning-Loss verwenden. Welcher Loss hier Verwendung findet, kann umgestellt werden. Der Pruning-Loss entspricht hierbei dem Netzoutput an der Stelle, die für die im Input gegebene Ziffer steht. Er wurde aus [3] übernommen, wie auch die Berechnung der Saliency.

Es werden hier außerdem zur jeder Schicht noch die nötigen Operationen hinzugefügt, um die Gradienten der Schicht-Inputs und die Gewichte der Schichten zu berechnen.

4.3.2 Input-Importance

Für jede Schicht wird der Gradient des Loss J in Bezug auf den Input der Schicht X bestimmt, dies ergibt die Saliency-Map SM .

Formel VII: Gradient-Saliency zur Bestimmung der Input-Importance

$$\mathrm{SM}=\frac{\partial J}{\partial \mathrm{X}}$$

Große positive und negative Gradienten deuten beide auf große Auswirkungen des Inputs auf den Loss hin. Es wird daher der Absolutwert gebildet, um die Importance des Inputs I_X zu bestimmen.

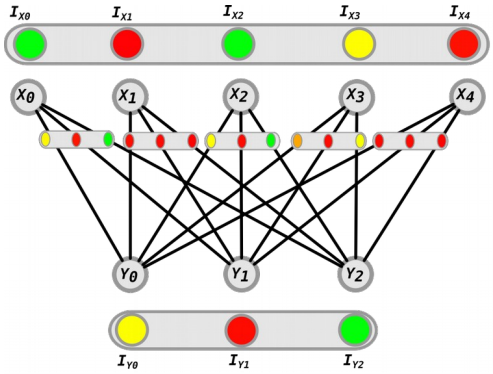
Formel VIII: Input-Importance

$$\mathrm{I}_X=|\mathrm{SM}|$$

Die Importance wird pro Neuron noch auf den Wertebereich zwischen 0 und 1 skaliert (Min-Max-Skalierung), um sie besser vergleichbar zu machen.

4.3.3 Weight-Importance

Aus der Importance des Inputs einer Schicht I_{X_l} und der Importance des Inputs der Folgeschicht I_{Y_l} kann nun die Importance I_{W_l} der einzelnen Gewichte $\boldsymbol{W}$ berechnet werden. Hierbei ist zu beachten, dass der Input der Folgeschicht X_{l+1} immer dem Output Y_l der vorhergehenden Schicht entspricht.



$$W_{xy}: \quad I_{Wxy} = I_{Xx} * I_{Yy}$$

Abbildung 3: Weight-Importance-Berechnung

Ist der Input der Folgeschicht Y_y unwichtig ($I_{Y_y}=0$) sind folglich alle Gewichte W_{xy} die zu diesem Input führen, ebenfalls unwichtig.

Ähnlich ist ein Gewicht W_{xy} das einen unwichtigen Input X_x mit einem beliebigen Input der Folgeschicht Y_y verbindet, ebenfalls unwichtig.

Ist der Input Y_y wichtig ($I_{Y_y}=1$) und der Input der Vorgängerschicht X_x ebenfalls wichtig, muss auch das Gewicht W_{xy} das die beiden verbindet, wichtig sein.

Es muss also jeder Eintrag der Importance der beiden Schichten mit jeweils jedem anderen multipliziert werden, um die Importance der Gewichte zu ermitteln, wie in *Abbildung 3: Weight-Importance-Berechnung* zu sehen.

4.3.4 Pruning

Mit Hilfe der Weight-Importance kann nun die Cumulated-Importance (CI) der Gewichte über verschiedene Inputs des Netzes hinweg bestimmt werden. Diese wird gebildet, indem für ein Trainings-Sample die Importance für jedes Neuron getrennt aufaddiert wird.

Gewichte, die für den größten Teil der Inputs unwichtig waren, werden hier eine geringe CI erhalten.

Auch Gewichte, die nur bei sehr wenigen Inputs wichtig waren und sonst immer unwichtig, erhalten eine geringe CI. Solche Gewichte entstehen durch Overfitting, da sie nur auf einen ganz bestimmten Input reagieren. Eine geringe CI ist also auch angebracht.

Anhand der CI kann nun ein Pruning-Durchlauf durchgeführt werden.

Hierzu wurde zunächst ein Pruning-Faktor (PF) verwendet, der in jedem Zeitschritt PF % der verbleibenden Gewichte entfernt. Dieser exponentielle Decay (Zerfall) würde allerdings dafür sorgen, dass nach unendlich vielen Schritten alle Gewichte entfernt werden.

Deshalb wurde auf einen Pruning-Schwellwert (PT) gewechselt. Alle Gewichte die unterhalb dieses liegen, werden entfernt. Der Schwellwert bildet sich dabei aus dem Median der (nicht entfernten) Gewichte eines einzelnen Neurons, der Sparsity (S) des Neurons und dem Sparsification-Factor (SF), einem Hyperparameter.

Die Sparsity berechnet sich dabei aus der Anzahl der ursprünglichen Gewichte C_0 zum Zeitpunkt 0 und der Anzahl der noch verbleibenden Gewichte C_t zum Zeitpunkt t.

Formel IX: Sparsity

$$S = \frac{C_0}{C_t}$$

Umso mehr Gewichte entfernt werden, umso größer wird also die Sparsity. Beginnend bei eins läuft sie gegen unendlich, wenn alle Gewichte entfernt würden.

Der Sparsification-Factor (SF) gibt an, bei wie viel Prozent des Medians der Pruning-Schwellwert liegen soll. Er wird allerdings mit der Sparsity potenziert, was zu einem Decay führt, sodass nie alle Gewichte entfernt werden können.

Formel X: Pruning-Schwellwert

$$PT = Median * SF^{S}$$

Umso weniger Gewichte noch vorhanden sind, umso weniger Gewichte werden im nächsten Schritt entfernt.

Beträgt der SF genau 1, werden alle Gewichte, die kleiner sind als der Median, entfernt. Da der Median in der Mitte aller Werte liegt, wird also in jedem Pruning-Schritt die Hälfte aller Gewichte entfernt. Das entspricht einem exponentiellen Zerfall, der für unendlich viele Schritte gegen 0 läuft.

Für SF < 1 findet ebenfalls ein exponentieller Zerfall statt, der den SF gegen 0 laufen lässt. Dies bedeutet, dass pro Zeitschritt immer weniger Gewichte entfernt werden. Also zwei exponentielle Funktionen, die gegeneinander arbeiten.

Der Grenzwert, gegen den sie laufen, kann allerdings nicht exakt bestimmt werden, da die Verteilung der Gewichte im Voraus nicht bekannt ist. SF = 50% des Medians bedeutet folglich nicht 50 % * 50 % = 25 % der Gewichte werden entfernt. Wenn man dies aber als Approximation annimmt, können folgende Gleichungen aufgestellt werden, mit $m = 0.5$, da der Median immer bei 50 % aller Gewichte liegt:

$$C_t = C_{t-1} - \lfloor C_{t-1} * PT_{t-1} \rfloor$$

und eingesetzt:

Formel XI: Iterative Vorschrift zum Schätzen der verbleibenden Gewichte

$$C_t = C_{t-1} - \lfloor C_{t-1} * m * SF^{\frac{C_0}{C_{t-1}}} \rfloor$$

Numerisch kann hier ein Grenzwert bestimmt werden, da die Anzahl an Gewichten C hier nur ganzzahlige Werte annehmen kann, wie in *Abbildung 4: Numerische berechnete Konvergenz der Anzahl der Gewichte* für SF von 0 bis 0.9 gezeigt.

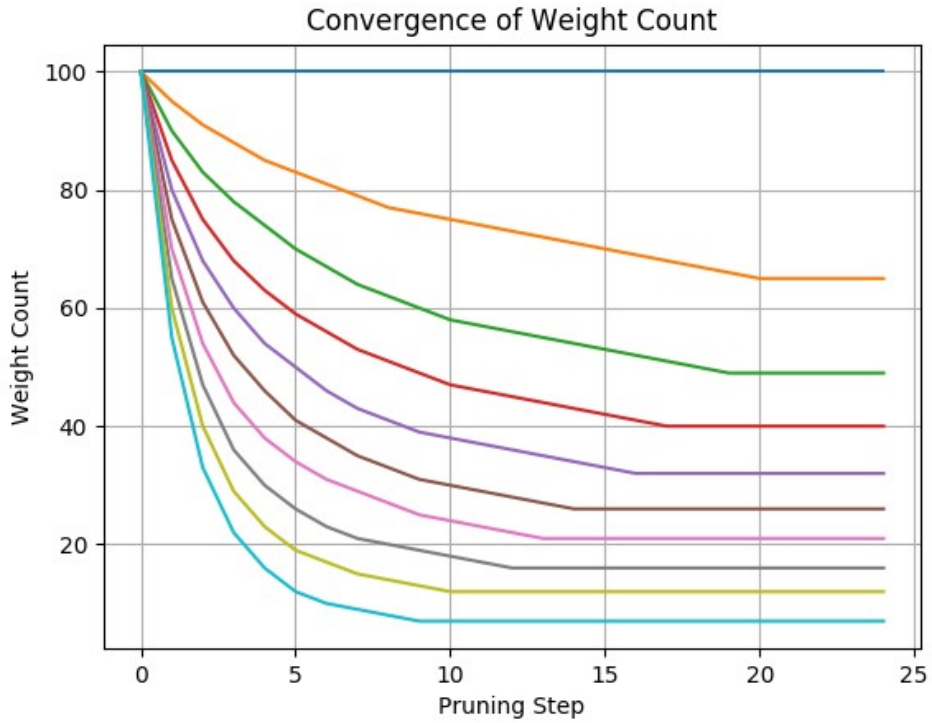


Abbildung 4: Numerische berechnete Konvergenz der Anzahl der Gewichte

Mathematisch hat es zunächst den Anschein, als würde diese Folge nicht konvergieren. Dadurch, dass es aber immer nur eine ganzzahlige Anzahl an Gewichten geben kann, durch die Gausklammern dargestellt, kann allerdings dennoch eine Umformung vorgenommen werden und so der Grenzwert in geschlossener Form bestimmt werden. Die Herleitung ist im Folgenden zu sehen:

Für Konvergenz gilt:

$$C_t = C_{t-1}$$

Damit ergibt sich:

$$C_t = C_t - \lfloor C_t * m * SF^{\frac{C_0}{C_t}} \rfloor$$

und vereinfacht:

$$0 = \lfloor C_t * m * SF^{\frac{C_0}{C_t}} \rfloor$$

Durch Weglassen der Gausklammern unter Beachtung der Rundung erhält man eine Ungleichung:

$$1 > C_t * m * SF^{\frac{C_0}{C_t}}$$

Diese kann umgeformt werden zu:

$$1 > C_t * m * e^{\frac{C_0}{C_t} * ln(SF)}$$

und mit der Umkehrfunktion von $x * \mathrm{e}^x$ (der LambertW-Funktion W) ergibt sich:

Formel XII: Analytische Vorschrift zum Schätzen der verbleibenden Gewichte

$$C_t = -\frac{C_0 * log(SF)}{W(-C_0 * m * log(SF))}$$

Durch Einsetzen von Beispielwerten erhält man ähnliche Grenzwerte wie bei der numerischen Bestimmung, siehe rote Linie in *Abbildung 4: Numerische berechnete Konvergenz der Anzahl der Gewichte*.

$$\left\{-\frac{c \log(s)}{W(-c\, m \log(s))}, c = 100, s = 0.3, m = 0.5\right\} \qquad -\frac{c \log(s)}{W(-c\, m \log(s))} \approx 40.1421$$

An dieser Stelle könnten auch andere Vorschriften zur Bestimmung der Anzahl an zu entfernenden Gewichte verwendet werden. So wird in [17] anstelle SF % des Median eine Rangfolge gebildet und alle Gewichte unterhalb eines Rangs entfernt. Die statistische Verteilung der Gewichte hat dadurch keinen Einfluss auf die Anzahl der zu entfernenden Gewichte. Es war nicht Ziel der Arbeit, hier verschiedene Vorgehen zu vergleichen, weshalb Alternativen nicht getestet wurden.

5 Ergebnisse

Im Folgenden wird auf die Ergebnisse der durchgeführten Tests eingegangen. Es wurden Importance-Maße verglichen, eine Gegenüberstellung der direkt berechneten Weigh-Importance zur Berechnung aus der Input-Importance durchgeführt und das Pruning getestet.

5.1 Vergleich von Importance-Maßen

Aus den bei Recherchen gefundenen Maßen für die Importance wurden zwei sehr ähnliche Maße, der Impact mittels Taylor-Polynom und die Gradient-Saliency, ausgewählt und verglichen. Beide Maße sind in *3.1.2 Saliency als Änderung des Outputs* und *3.2 Importance im Rahmen von Pruning* im Detail beschrieben. Die Ergebnisse des Vergleiches sind im Folgenden dargestellt.

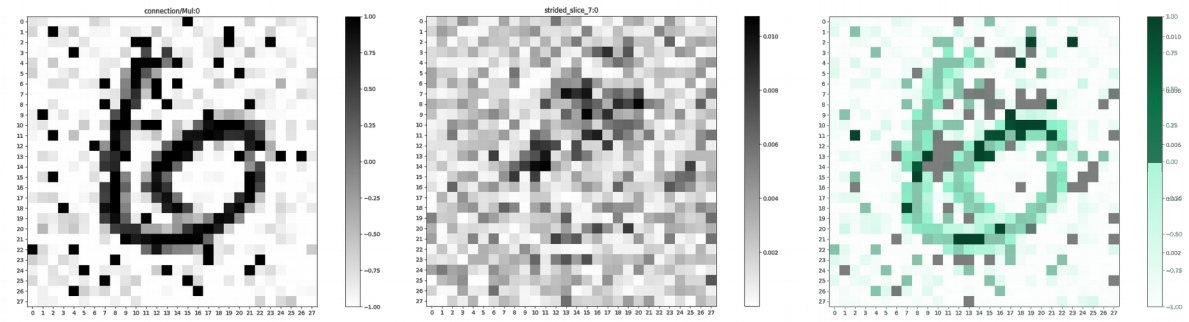


Abbildung 5: Impact der Eingabe auf den Trainings-Loss

In *Abbildung 5* ist links der Netzinput, in der Mitte der Impact des Inputs auf den Trainings-Loss und rechts ein überlagertes Bild beider dargestellt. In der Überlagerung sind die dunkelgrauen Pixel jeweils hohe Impact Werte, die sich nicht mit der Ziffer decken. Die dunkelgrünen Pixel entsprechen hohen Werten, die sich mit der Ziffer überlagern. Wie hier gut zu sehen ist, tritt der höchste Impact an den Rändern der Ziffer sowie auf der Ziffer selbst auf. Er scheint nach den in *4.1.3 Importance-Map* bestimmten Kriterien also an plausiblen Stellen hoch zu sein.

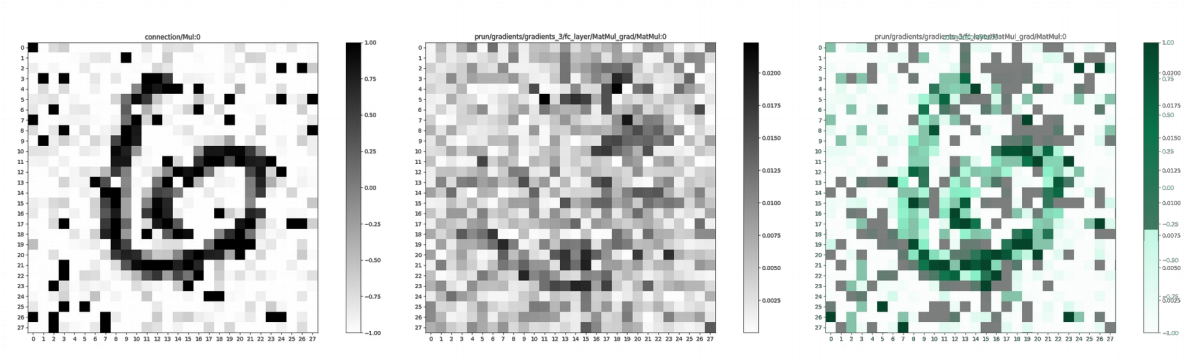


Abbildung 6: Gradient-Saliency der Eingabe in Bezug auf den Trainings-Loss

In *Abbildung 6* ist links der Netzinput, in der Mitte die Saliency des Inputs auf den Trainings-Loss und rechts ein überlagertes Bild beider dargestellt. In der Überlagerung sind die dunkelgrauen Pixel jeweils hohe Saliency Werte, die sich nicht mit der Ziffer decken. Die dunkelgrünen Pixel entsprechen hohen Werten die sich mit der Ziffer überlagern. Wie hier gut zu sehen ist, tritt die höchste Saliency an den Rändern der Ziffer sowie auf der Ziffer selbst auf. Die Saliency scheint also ebenfalls an plausiblen Stellen hoch zu sein.

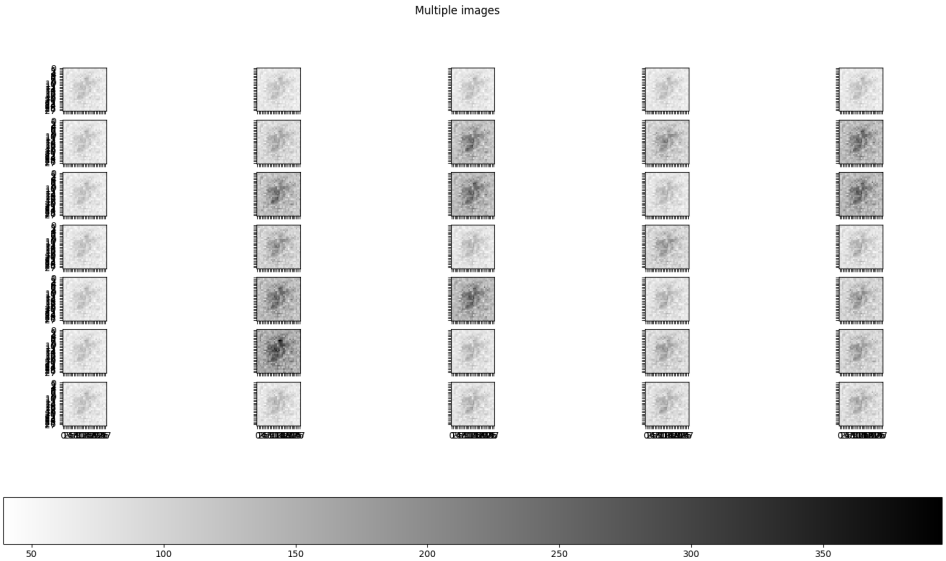


Abbildung 7: Weight-Importance anhand Impact

In *Abbildung 7* ist die Weight-Importance anhand des Impacts dargestellt. Jedes einzelne Bild ist hierbei die Importance aller Gewichte eines Neurons. Wie zu sehen ist, gibt es hier Neuronen, die generell wichtiger sind (durch die dunklere Färbung zu erkennen). Die einzelnen Gewichte sind dabei alle in der Mitte des Inputs (wo sich auch alle Ziffern des MNIST-Datensatzes befinden) dunkler. Der Impact scheint sich also zu eignen, um die wichtigen Gewichte zu finden.

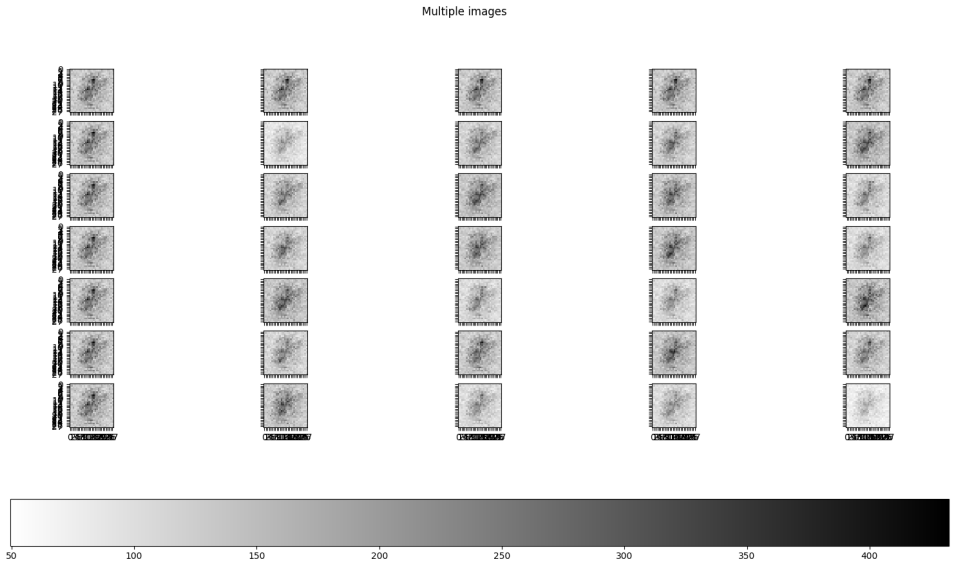


Abbildung 8: Weight-Importance anhand Saliency

In *Abbildung 8* ist die Weight-Importance anhand der Saliency dargetellt. Jedes einzelne Bild zeigt die Importance der Gewichte eines Neurons. Die einzelnen Gewichte sind auch hier in der Mitte des Inputs dunkler. Die Saliency scheint sich also gleichermaßen zu eignen, um die wichtigen Gewichte zu finden.

Die Maße scheinen also beide ähnlich gute Werte zu liefern, sie unterscheiden sich allerdings in einem wichtigen Punkt, der Rechenzeit. Die Saliency benötigt pro Bild nur den Gradienten des Inputs in Bezug auf den Output, während der Impact zusätzlich noch für jeden Input eine Multiplikation mit dem Gradienten benötigt. Also eine zusätzliche Multiplikation pro Input und pro Neuron des Netzes. Bereits bei dem kleinen Testnetz führte dies zu hohem Zeit- und Performance-Verlust, weshalb im späteren Verlauf der Arbeit bevorzugt die Importance anhand der Saliency verwendet wurde.

5.2 Weight-Importance

Die Importance der Gewichte kann entweder direkt mit einem Importance-Maß bestimmt werden oder indirekt aus der Input-Importance zweier aufeinanderfolgenden Schichten berechnet werden. Die Arbeit wurde mit dem Ziel der Berechnung aus der Input-Importance begonnen. Die Möglichkeit der direkten Berechnung kam dann während der Recherche auf, weshalb ein Vergleich der beiden Verfahren durchgeführt wurde. Hierzu wurden wiederum die entstehenden Importance-Maps nach den Kriterien in *4.1.3 Importance-Map* verglichen. Im Folgenden sind die Ergebnisse für die Importance anhand der Gradient-Saliency mit Trainings-Loss aufgeführt, in *8 Anhänge* sind weitere Ergebnisse für den Pruning-Loss, den Impact und den Fashion-MNIST-Datensatz zu finden. Das eigentliche Pruning wurde nur mit der Bestimmung der Weight-Importance aus der Input-Importance validiert und ist in *5.3 Pruning* beschrieben.

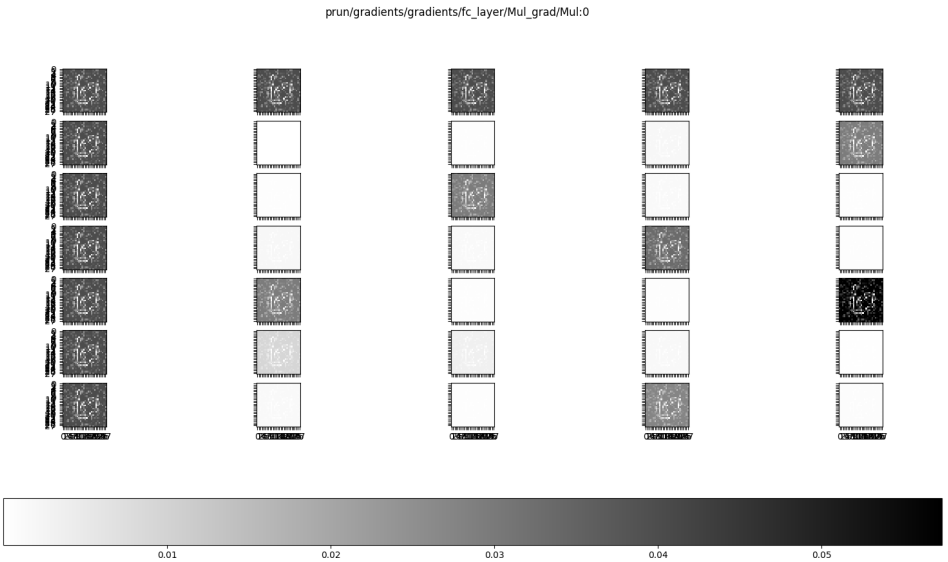


Abbildung 9: Importance direkt von Gewichten

Bei der Importance der Gewichte, direkt bestimmt durch die Gradienten des Loss in Bezug auf die einzelnen Gewichte, siehe *Abbildung 9,* erhält man an den Stellen an denen sich die Ziffer befindet, niedrige Gradienten, an den anderen Stellen hohe.

Eine Änderung an wichtigen Gewichten hat damit kaum eine Auswirkung auf die Entscheidungsfindung und steht damit im Widerspruch zum gewünschten Verhalten.

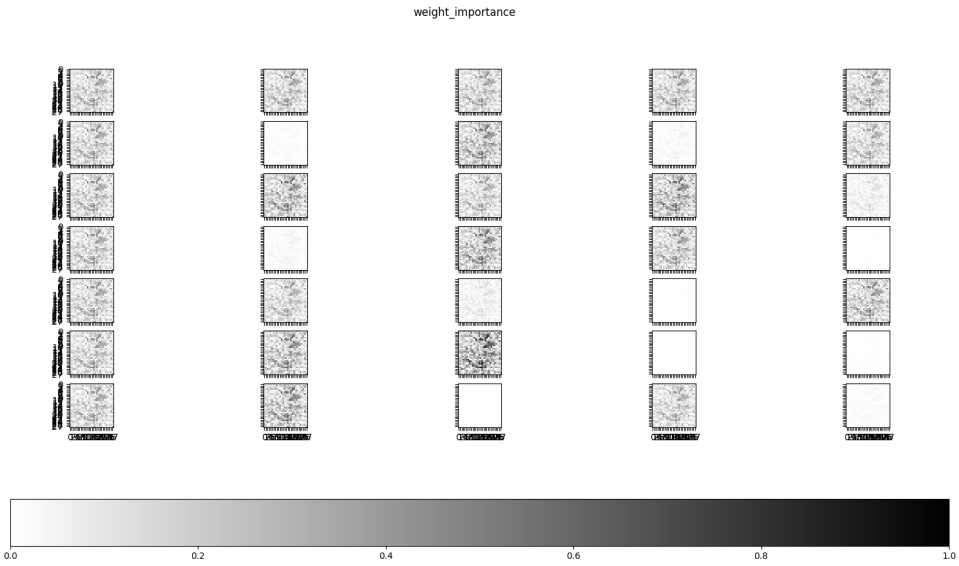


Abbildung 10: Importance der Gewichte anhand des Inputs

Die Importance der Gewichte, bestimmt durch zwei aufeinanderfolgende Input-Importance-Maps, siehe *Abbildung 10*, zeigt hingegen deutlich, dass die wichtigen Bereiche mit Bereichen hoher Importance übereinstimmen. Sie eignet sich folglich besser für die Bestimmung wichtiger Gewichte und wird im Rest der Arbeit verwendet.

5.3 Pruning

Pruning wurde auf einem initialen Netz für verschiedenen Sparsity-Faktoren durchgeführt. Das Netz wurde dabei 150 Pruning-Iterationen unterzogen und der Verlauf der Anzahl der Gewichte und Sparsity jeder Schicht sowie die Accuracy des Netzes gespeichert. Im Folgenden ist immer nur der Gewichtsverlauf der ersten Schicht dargestellt. Die anderen Schichten verhalten sich ähnlich und sind in *8 Anhänge* zu finden. Die Sparsity ist für jede Schicht dargestellt, allerdings ohne zeitlichen Verlauf. Dieser ist an der Dichte der Linien zu erkennen. Hier werden beispielhaft drei Sparsity-Faktoren verglichen, in *8 Anhänge* sind die Diagramme weiterer Faktoren zu finden. Bei allen Diagrammen ist zur besseren Darstellung die Inverse der Sparsity aufgetragen, da diese immer im Bereich zwischen 0 und 1 liegt.

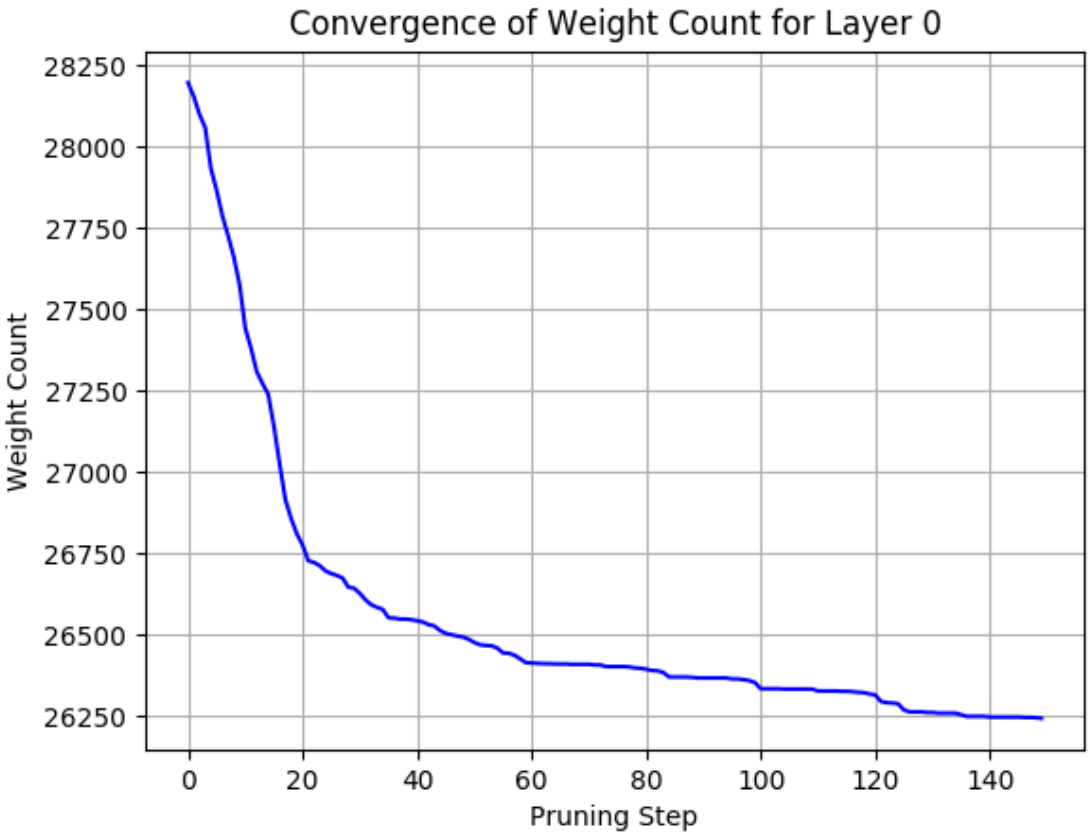


Abbildung 11: Verlauf der Anzahl der Gewichte bei SF 0.6

Gut zu erkennen ist in *Abbildung 11: Verlauf der Anzahl der Gewichte bei SF 0.6,* dass sich die Anzahl der Gewichte einem Grenzwert nähern. In *Abbildung 12* hingegen ist keine stetige Annäherung an einen Grenzwert für Layer 2 erkennbar. Dies sollte eigentlich, wie bei Layer 0, durch eine zunehmende Dichte der Linien erkennbar sein. Stattdessen wirkt es eher wie ein linearer Verlauf. Dies ist auf die Importance-Verteilung der Gewichte zurückzuführen. Hier wird deutlich, dass SF=60 % des Medians nicht automatisch heißt, dass immer 50 % * 60 % = 30 % der Gewichte entfernt werden.

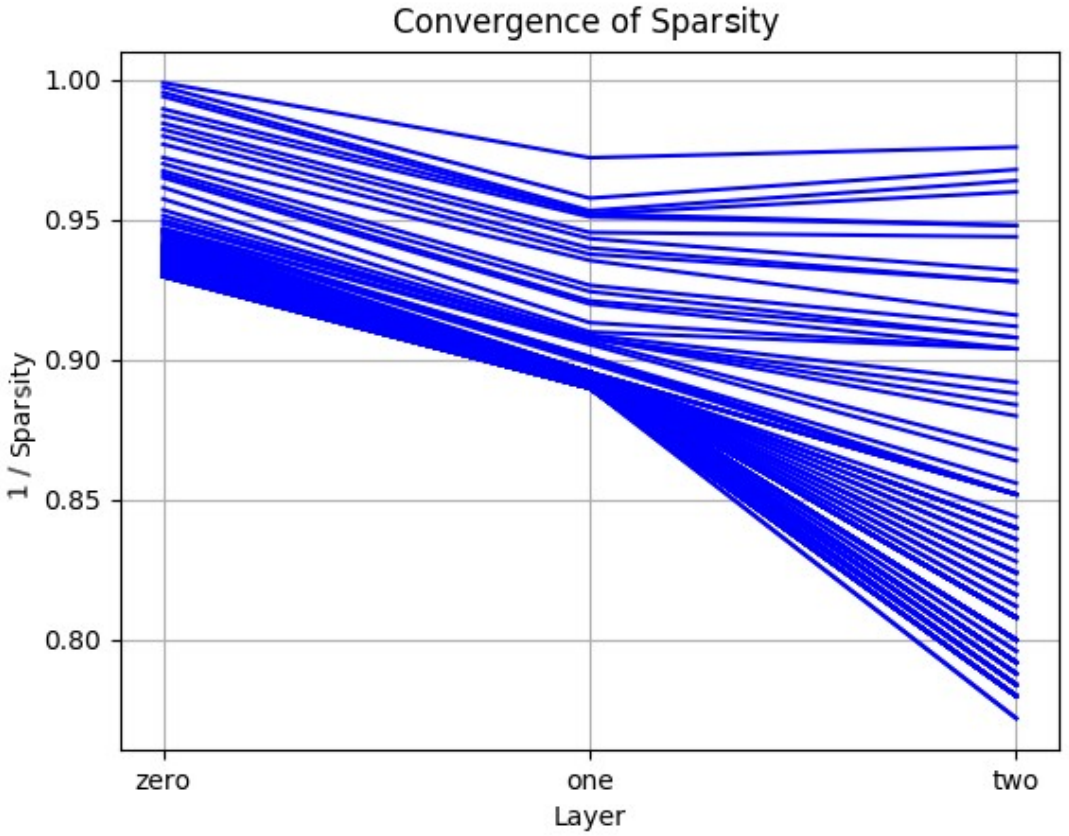


Abbildung 12: Verlauf der Sparsity bei SF 0.6

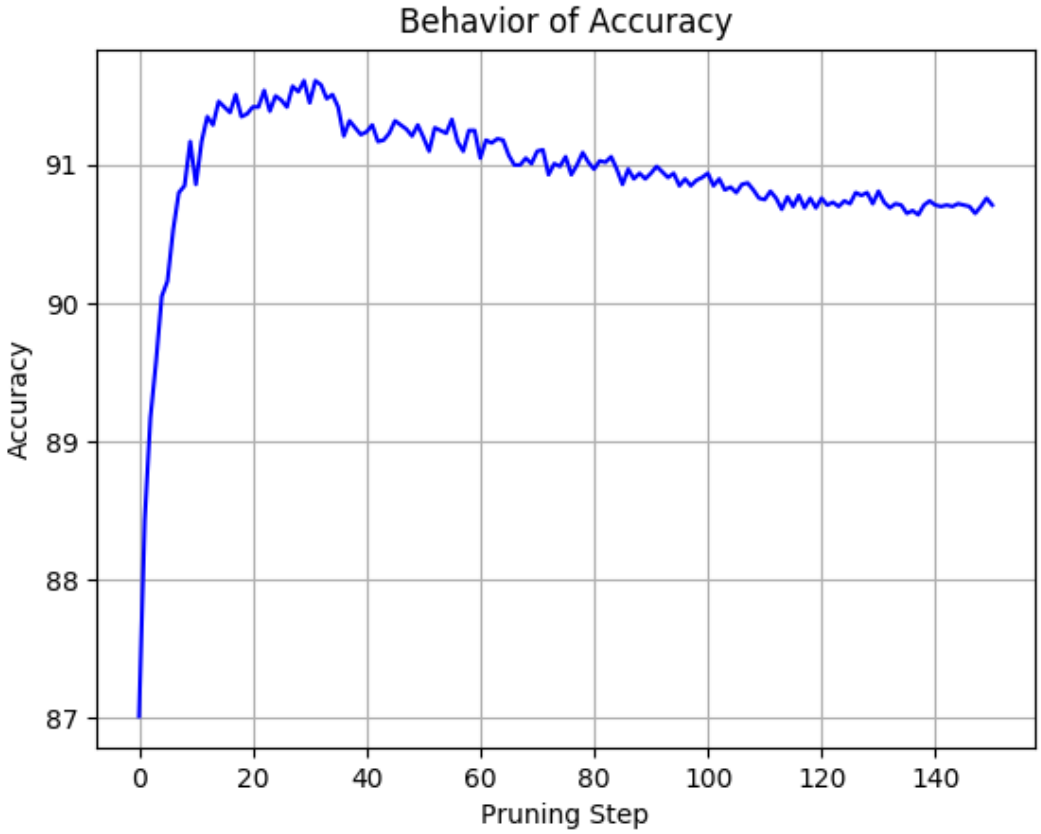


Abbildung 13: Verlauf der Accuracy bei SF 0.6

Überraschend ist zu beobachten, dass die Accuracy in *Abbildung 13* in den ersten 30 Pruning-Durchläufen nicht fällt, sondern steigt. Dies hat nichts mit der zusätzlichen Trainingszeit beim Fine-Tuning zu tun, sondern ist auf das Vermeiden von Overfitting zurückzuführen. Das Netz wird in jedem Schritt trainiert, bis sich die Accuracy nicht mehr verändert. Durch das Entfernen von Gewichten, die nur für eine sehr geringe Anzahl an Trainingsdaten relevant sind, wird das Netz allerdings gezwungen, mit den verbleibenden Gewichten besser zu verallgemeinern.

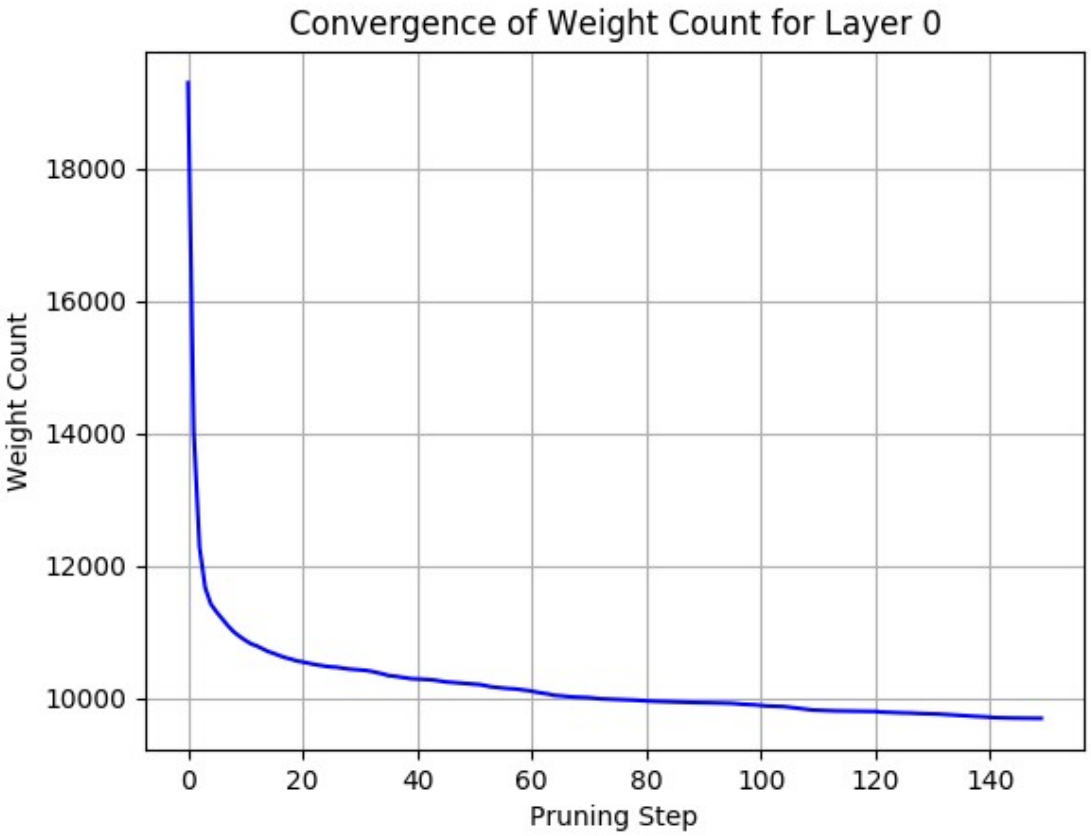


Abbildung 14: Verlauf der Anzahl der Gewichte bei SF 0.9

Auch bei einem SF von 0.9 ist ein exponentieller Verlauf gegen einen Grenzwert zu beobachten, siehe *Abbildung 14.* Dieser fällt gerade im vorderen Bereich deutlich schneller, entfernt also bereits zu Beginn des Pruning deutlich mehr Gewichte.

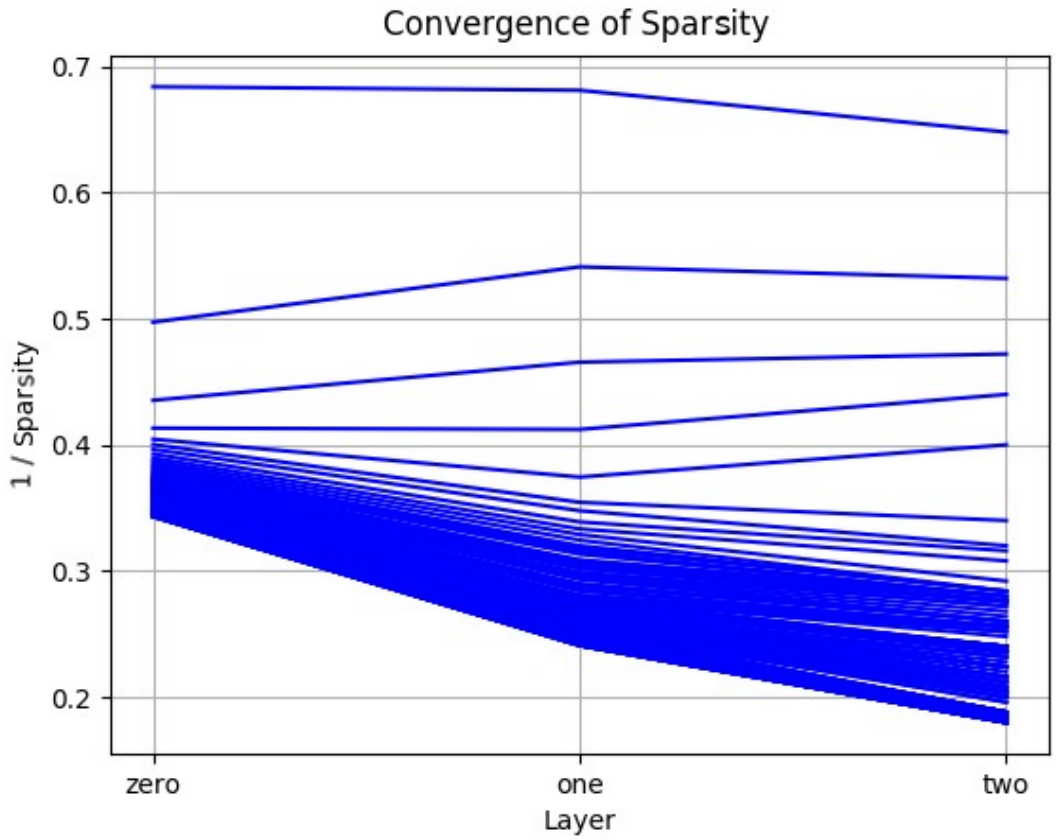


Abbildung 15: Verlauf der Sparsity bei SF 0.9

Bei einem SF, der zu einem PT so nahe am Median führt, spielt auch die Verteilung der Gewichte eine deutlich geringere Rolle. So kann man in *Abbildung 15* nun für alle Schichten eine Annäherung der Sparsity an einen Grenzwert beobachten. Hier treten nur noch kleine Artefakte in der Dichte der Linien auf.

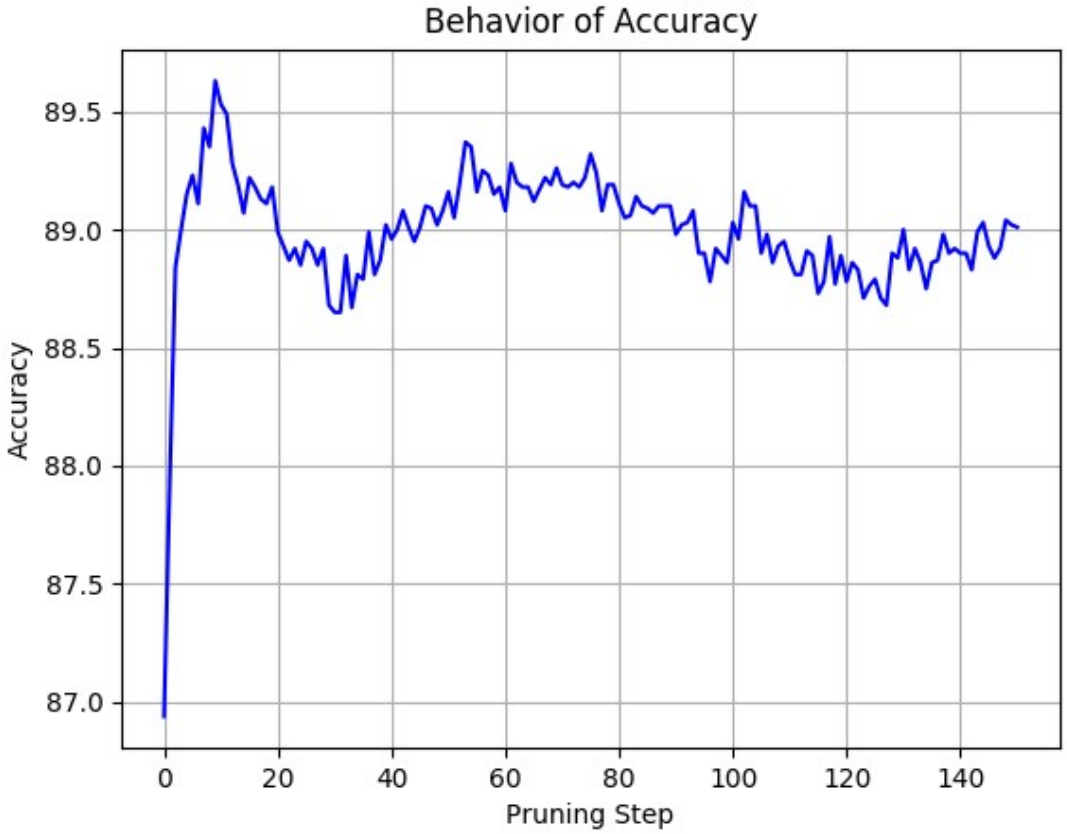


Abbildung 16: Verlauf der Accuracy bei SF 0.9

Durch das schnelle Entfernen so vieler Gewichte kann das Netz nicht mehr die Accuracy erreichen, die es beim langsamen Entfernen der Gewichte erreicht hat, siehe *Abbildung 16*. Dennoch ist zu beobachten, dass die Accuracy in den ersten Pruning-Durchläufen zunächst zunimmt, bevor sie sich dann bei einem Grenzwert einpendelt. Das Netz erreicht hier immer noch eine höhere Accuracy als das Original, bei nur noch ca. 30 % der Gewichte.

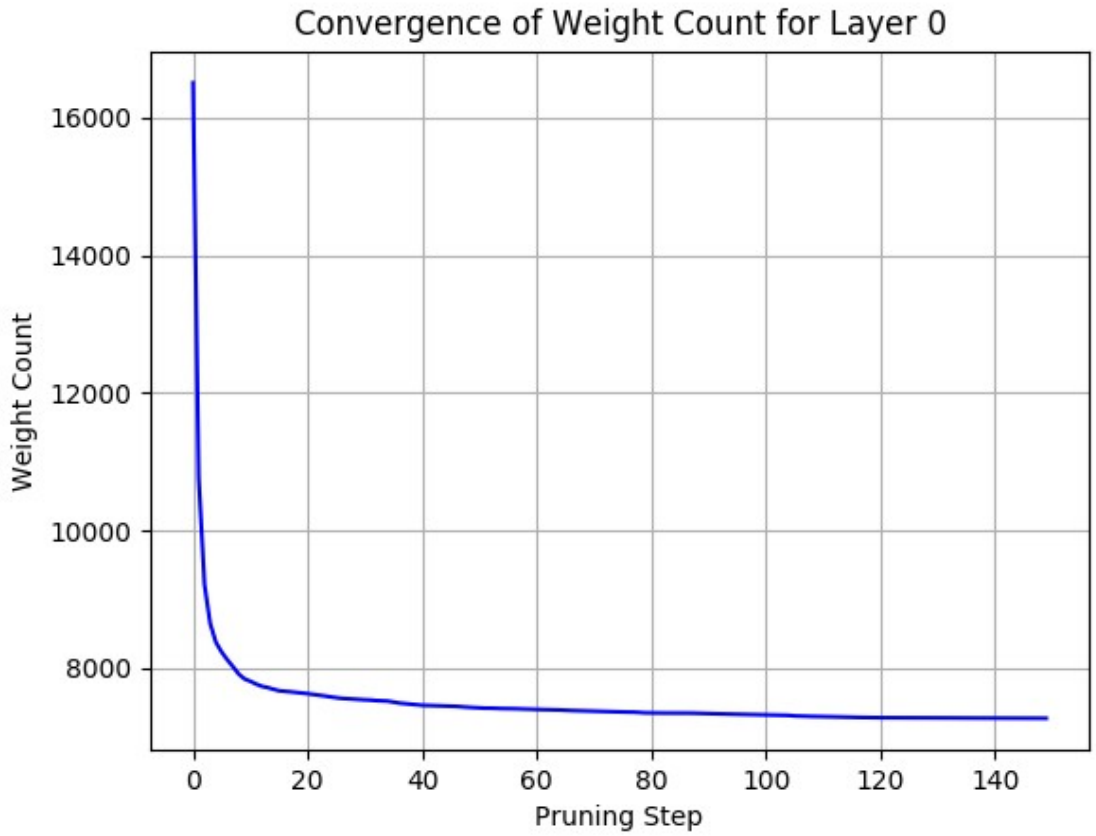


Abbildung 17: Verlauf der Anzahl der Gewichte bei SF 0.95

Bei einem SF von 0.95 werden sehr viele Gewichte entfernt. Es werden jedoch immer noch nicht 100 % der Gewichte entfernt, wie der Verlauf der Gewichte von 28224 (nicht mehr in der Grafik) gegen den Grenzwert von 7500 zeigt, siehe *Abbildung 17*. Tatsächlich fällt die Anzahl der Gewichte hier noch langsam, eine numerische sowie eine mathematische Schätzung liefert einen Grenzwert von ca. 300 verbleibenden Gewichten.

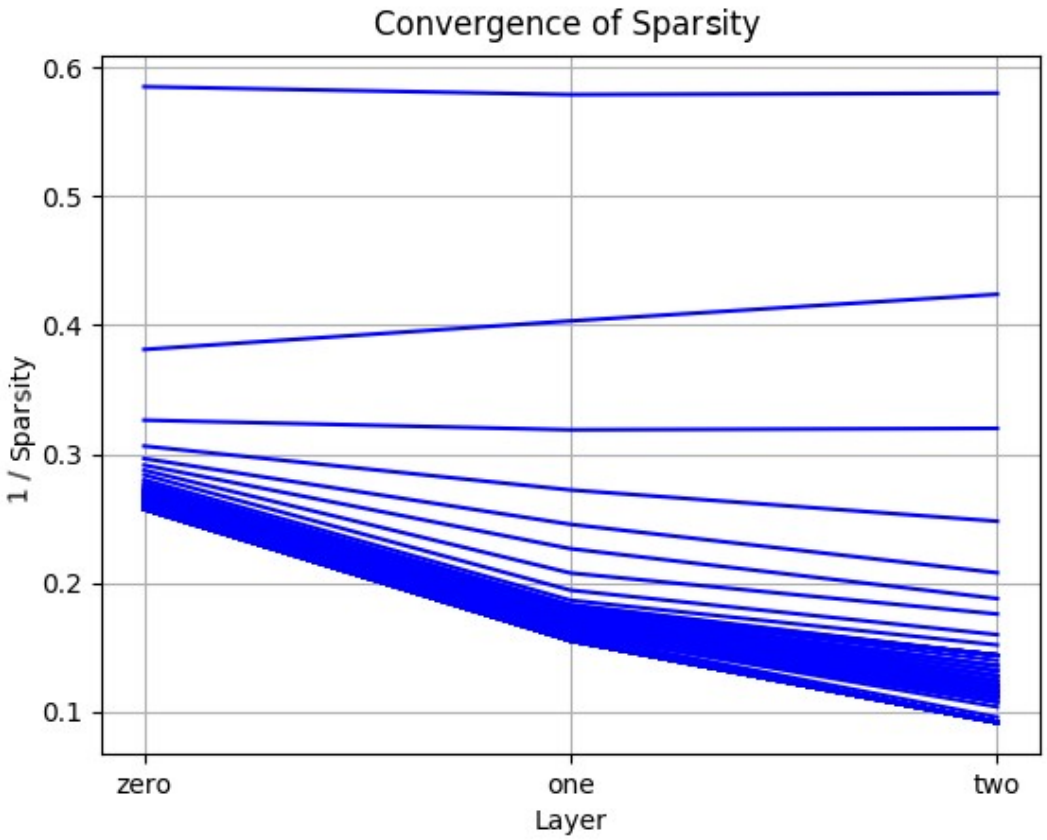


Abbildung 18: Verlauf der Sparsity bei SF 0.95

Der Verlauf der Sparsity in *Abbildung 18* ist nun noch weniger von der Importance-Verteilung der Gewichte abhängig. Es bleiben hier für die verschiedenen Schichten nur noch 10 % bis 25 % aller Gewichte übrig.

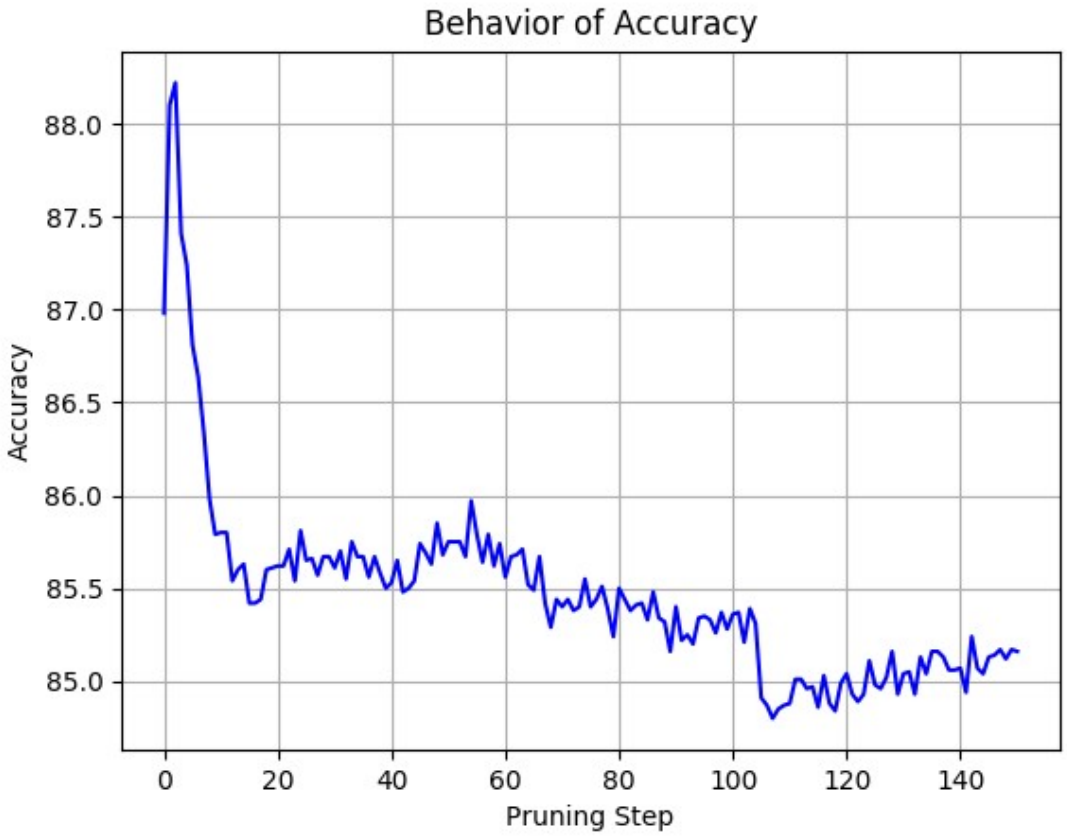


Abbildung 19: Verlauf der Accuracy bei SF 0.95

Dies ist nun auch das erste Mal, dass die Accuracy unter die des Original-Netzes fällt. Allerdings auch nur 2 % darunter, siehe *Abbildung 19*. Das ist für eine Reduzierung der Gewichte auf ca. 10 % bis 25 % ein beachtlich geringer Verlust.

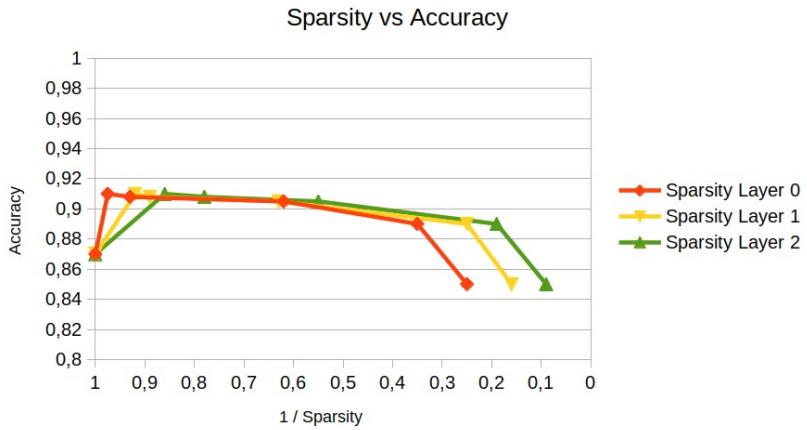


Abbildung 20: Sparsity vs Accuracy

Ein Vergleich der Sparsity mit der Accuracy für alle Messwerte ist in *Abbildung 20* zu sehen. Offensichtlich kann die Anzahl der Parameter im getesteten Netz auf unter 30 % reduziert werden, ohne dabei einen Verlust in der Accuracy verbuchen zu müssen.

Der Sparsity-Faktor scheint dabei im Bereich oberhalb von 0.5 einen nahezu linearen Zusammenhang mit der Sparsity zu haben, siehe *Abbildung 21*.

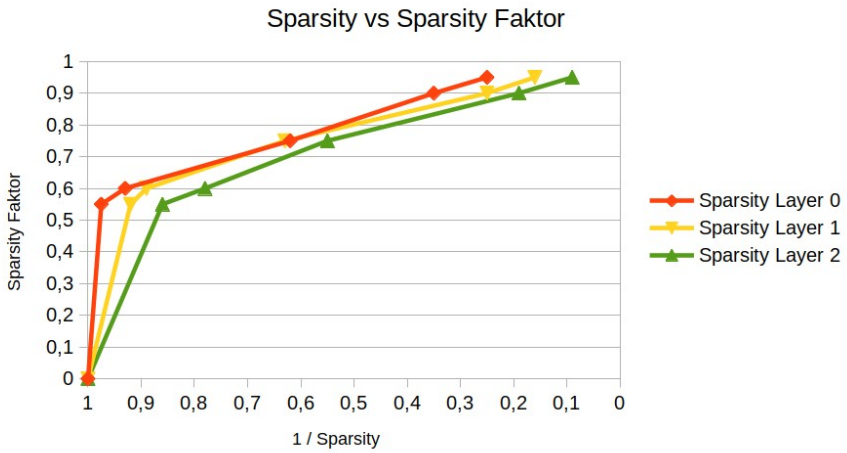


Abbildung 21: Sparsity vs Sparsity Faktor (SF)

6 Fazit

In dieser Arbeit wurden verschiedene Maße für die Wichtigkeit einzelner Gewichte, basierend auf Saliency, verglichen. Als vielversprechendstes Maß wurde die Weight-Importance, mittels Gradient-Saliency der Schicht-Inputs berechnet, genutzt, um Netz-Pruning durchzuführen. Pruning mit diesem Maß lieferte für ein einfaches Testnetz und den MNIST-Datensatz Ergebnisse, die die Erwartungen übertrafen. So wurde die Accuracy beim Pruning zunächst nicht schlechter, sondern besser, was auf die Vermeidung von Overfitting durch Pruning zurückzuführen ist. Auch bei einer sehr hohen Anzahl entfernter Parameter des Netzes musste nur ein geringer Verlust in der Accuracy hingenommen werden.

Eine ausführliche Recherche und eine Darstellung der wichtigsten Veröffentlichungen in diesem Bereich ergänzt diese Arbeit.

Als Erweiterungen könnte man das in [17] beschriebene Structured-Probabilistic-Pruning mit dem der in dieser Arbeit verwendeten Saliency-Information kombinieren, um Pruning bereits während des regulären Trainings durchzuführen und Overfitting zu vermeiden.

7 Quellen

Literaturverzeichnis

[1]: o.J., Duden Online, 2019, https://www.duden.de/suchen/dudenonline/saliency
[2]: Marilyn Lougher Vaughn, Interpretation and Knowledge Discovery from the Multilayer Perceptron Network: Opening the Black Box, 1996, Neural Comput & Applie (1996) 4
[3]: Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, 2014, arXiv:1312.6034
[4]: Wataru Shimoda, Keiji Yanai, Distinct Class Saliency Maps For Multiple Object Images, 2016, International Conference on Learning Representations 2016 (ICLR 2016)
[5]: Wataru Shimoda, Keiji Yanai, Distinct Class-specific Saliency Maps for Weakly Supervised Semantic Segmentation, 2016, The 14th European Conference on Computer Vision 2016 (ECCV 16)
[6]: Seong Joon Oh, Rodrigo Benenson, Anna Khoreva, Zeynep Akata, Mario Fritz, Bernt Schiele, Exploiting saliency for object segmentation from image level labels, 2017, The IEEE Conference on Computer Vision and Pattern Recognition 2017 (CVPR 2017)
[7]: Hengyue Pan, Hui Jiang, A Fast Method for Saliency Detection by Back-Propagating A Convolutional Neural Network and Clamping Its Partial Outputs, 2017, International Joint Conference on Neural Networks 2017 (IJCNN 2017)
[8]: Hengyuan Hu, Rui Peng, Yu-Wing Tai, Chi-Keung Tang, Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures, 2016, arXiv:1607.03250
[9]: S. Han, J. Pool, J. Tran, and W. Dally, Learning both weights and connections for efficient neural network, 2015, In Advances in Neural Information Processing Systems
[10]: Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz, Pruning Convolutional Neural Networks for Resource Efficient Inference, 2017, 5th International Conference on Learning Representations (ICLR 2017)
[11]: Michael Figurnov, Aijan Ibraimova, Dmitry Vetrov, Pushmeet Kohli, PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions, 2016, 30th Conference on Neural Information Processing Systems (NIPS 2016)
[12]: N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, 2014, The Journal of Machine Learning Research, 15(1)
[13]: L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, Regularization of neural networks using dropconnect, 2016, In Proceedings of the 33rd International Conference on Machine Learning 2016 (ICML 2016)
[14]: Alireza Aghasi, Afshin Abdi, Nam Nguyen, Justin Romberg, Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee, 2017, 31st Conference on Neural Information Processing Systems (NIPS 2017)

[15]: Yihui He, Xiangyu Zhang, Jian Sun, Channel Pruning for Accelerating Very Deep Neural Networks, 2017, International Conference on Computer Vision (ICCV 2017)
[16]: Mohammad Babaeizadeh, Paris Smaragdis, Roy H. Campbell, NoiseOut: A Simple Way to Prune Neural Networks, 2016, 29th Conference on Neural Information Processing Systems (NIPS 2016)
[17]: Huan Wang, Qiming Zhang, Yuehai Wang, Haoji Hu, Structured Probabilistic Pruning for Convolutional Neural Network Acceleration, 2018, Proceedings of the British Machine Vision Conference (BMVC 2018)

8 Anhänge

In Digitaler Form unter:

https://github.com/bela127/Pruning_with_Saliency_Information.git