Final Report / Task 2

by Benjamin Seeger

Die Software Contribution ist über folgenden Link zu finden und kann mittels Binder erkundet werden: https://github.com/BenjaminSeeger/my-first-binder

In dem Binder ist der Code jeweils mit Kommentaren versehen, welcher näher die Gedankengänge zu den einzelnen Zeilen erklärt. Daher versuche ich hier im Final Report zusätzliche Informationen und Ideen zu klären, welche im Binder noch nicht stehen.

Einleitung

Meine Schwerpunkte für die Aufgabe sind: Performance – Explainability – Data Analysis. Dementsprechend habe ich mich mir sehr viele Mühe bei der Performancesteigerung und der Entwicklung meines Netzes gegeben. Im Binder ist ganz unten ein Archiv eingefügt, welches die Entwicklungsstufen des Modells enthält. Dem Punkt Explainability wollte ich gerecht werden, durch begleitende Hinweise im Code, die näher die Entwicklung und den Code generell erklären. Speziell wird sich folgender Abschnitt weiter mit der Erklärung meines Netzes beschäftigen. Auf Data Analysis werde ich vor allem beim Evaluieren des Netzes eingehen.

Mein Konzept

Im Folgenden erkläre ich Stück für Stück mein entwickeltes Modell und die Ideen dahinter. Dazu habe ich Abschnitte eingefügt, ähnlich meinem Binder, für die bessere Übersichtlichkeit. Der Fokus meiner Modell-Entwicklung und auch die hauptsächlichen Verbesserungen konnten im Aufbau des Netzes selbst erzielt werden, jedoch sind meine Gedanken und Ideen zum Import und Pre-Processing auch aufgeführt.

Import

Beim Importieren des MNIST-Datasets gehe ich wie üblich vor und lade die Daten in ein Trainings- und Validierungsset. An dieser Stelle könnte man durch Augmentation die Datenmenge weiter erhöhen. Bevor ich das in meiner Software Contribution umgesetzt habe, wollte ich jedoch die Notwendigkeit beurteilen. Dazu habe ich mir ein paar Hundert Elemente der Datensets ausgeben lassen und die Bilder miteinander vergleichen. Dabei waren alle möglichen Formen und Verdrehungen der Zahlen bereits gegeben. Zum Teil waren sogar Zahlen dabei, die ich selbst nicht identifizieren konnte, weshalb eine 100% accuracy des MNIST-Sets ausgeschlossen ist. Auf Grund der hohen Vielfalt der Daten hatte ich mich dann gegen die Daten Augmentation entschieden. Das macht wohl erst bei geringer Datenmenge und wenig Varianz der Daten wirklich Sinn. Sprich das standardmäßige Laden der Daten bleibt.

Pre-Processing

Beim Pre-Processing des MNIST-Datasets sind in meiner Software Contribution nur noch die üblichen Vorgänge enthalten. Zum einen das Skalieren der Graustufenwerte von dem Bereich [0; 225] auf den Bereich [0; 1]. Zum anderen das Umformen des Datasets für den ersten Layer des Modells. An dieser Stelle hatte ich jedoch noch zusätzlich eine weitere Idee. Beim Analysieren der Daten zuvor ist mir nämlich aufgefallen, dass die Zahlen recht unscharf sind und weiche Kanten haben. Meine Idee war es nun die Kanten der Zahlen zu schärfen, um dem Netz die Erkennung zu vereinfachen. Umgesetzt hatte ich das in dem ich den zuvor skalierten Graustufenwerte-Bereich [0; 1] in der Mitte getrennt habe und alle Werte kleiner 0,5 auf 0 und alle Werte größer oder gleich 0,5 auf 1 gesetzt habe. Dadurch erhielten die Zahlen scharfe Kanten und waren mit dem Auge sehr gut erkennbar. Leider

hatte sich mein Netz nicht wie gewünscht über die Anpassung gefreut. Ich konnte keinen merklichen Mehrwert durch Analyse von loss und accuracy feststellen, der diesen Aufwand gerechtfertigt hätte. Daher habe ich das wieder gelöscht und das Pre-Processing beim Skalieren der Graustufenwerte und Reshapen belassen.

Modell

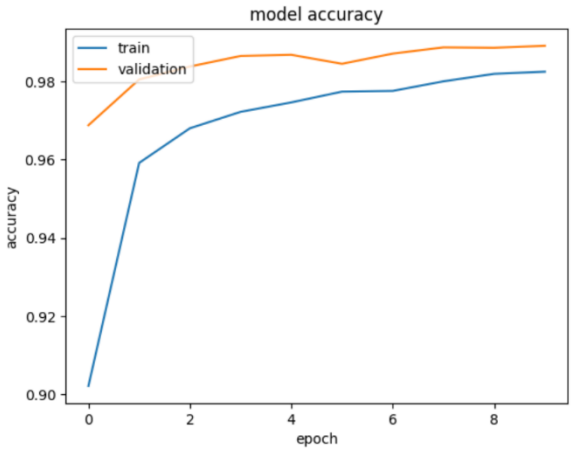
In diesem Abschnitt erkläre ich mein entwickeltes Modell und vor allem auch meine Ideen dahinter. In dem Binder ist ganz unten ein Archiv eingefügt, welches einige der Entwicklungsstufen und Versuche bis zum aktuellen Modell beinhaltet. Da dort bereits eine Entwicklungshistorie aufgeführt ist, gehe ich hier auf mein aktuelles und performantestes Modell ein. Hier rechts ist die Zusammenfassung des Modells zu erkennen. Die Grundidee hinter dem Modell ist das Komprimieren der Daten und das Verdichten der Daten über das Modell hinweg. Da die Zahlen in einem Quadratischen Bild gespeichert sind, ist die Idee, Anfangs auch weiter die zweidimensionale Form beizubehalten. Um dennoch die Idee des Verdichtens umsetzen zu können folgender Aufbau. Der erste Layer ist ein 2D convolutional Layer mit Parameter 64. Daraufhin wird die Abtastung der Eingabe entlang der Höhe und Breite mit einem MaxPooling2D Layer verringert. Um das overfitten des Netzes zu verhindern, folgt ein Dropout Layer. Damit der die Verdichtungsidee fortgesetzt wird, wiederholen sich diese drei Layer erneut, mit dem Unterschied, dass nun der 2D convolutional Layer den Paramater 32 hat. Damit sind die ersten sechs Layer noch bei der zweidimensionalen Form geblieben. Um weiter zu verdichten, wird an dieser Stelle ein flatten Layer eingesetzt, welcher die vorigen zwei Dimensionen in eine Dimension überführt. Nun wird der Verdichtungsgedanke mit einem Dense Layer fortgeführt, welcher auf 1x128 runterbricht. Da die Zahlen von 0-9 detektiert werden sollen folgt zuletzt ein Dense Layer mit der Größe 1x10, um die zehn Zahlen zuordnen zu können. Mit diesem Modell Aufbau habe ich nach zehn Epochen eine Validation Accuracy von 99% Und einen Loss von 0,03 erzielt. Zu der Auswertung und einer spannenden Eigenschaft des Netzes im nächsten Abschnitt mehr.

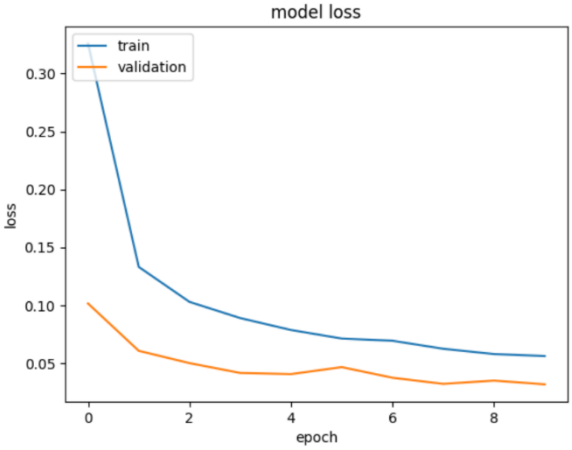
Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d_4 (MaxPooling 2D)	(None, 13, 13, 64)	0
dropout_4 (Dropout)	(None, 13, 13, 64)	0
conv2d_5 (Conv2D)	(None, 11, 11, 32)	18464
max_pooling2d_5 (MaxPooling 2D)	(None, 5, 5, 32)	0
dropout_5 (Dropout)	(None, 5, 5, 32)	0
flatten_2 (Flatten)	(None, 800)	0
dense_4 (Dense)	(None, 128)	102528
dense_5 (Dense)	(None, 10)	1290

Total params: 122,922
Trainable params: 122,922
Non-trainable params: 0

Evaluation / unerwartetes Verhalten

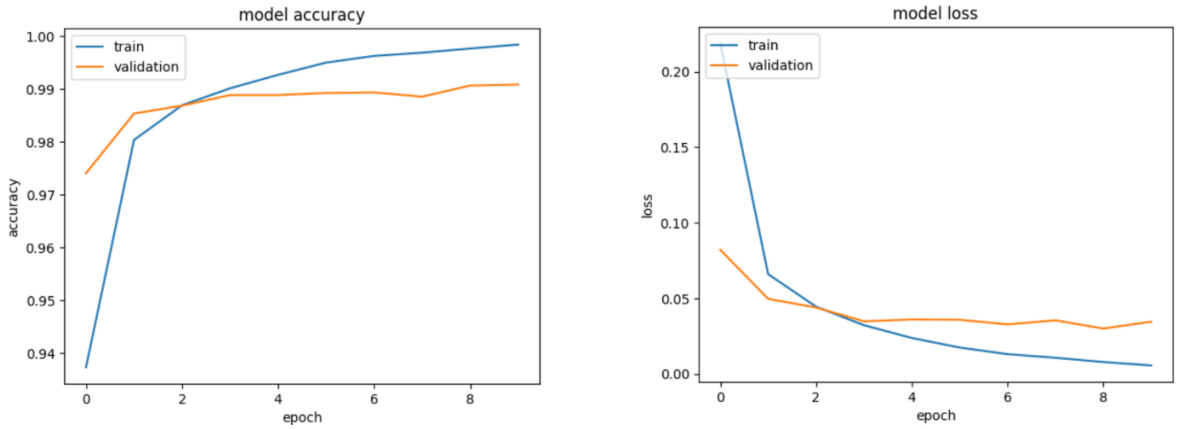
An dieser Stelle möchte ich nochmal zurückblicken und die Leistung des Netzes betrachten. Dazu sind hier zwei Grafiken eingefügt, welche die Entwicklung der Accuracy (links) und des Losses (rechts) über zehn Epochen darstellen. Beide Werte konvergieren recht zügig gegen die Endwerte 99,00%





Accuracy und 0,0301 Loss. Bei diesem Datenset sind jedoch 100% nicht zu erreichen. Wie ich bereits im Binder beschrieben habe, gibt es Zahlen, welche selbst als Mensch nicht zu identifizieren sind und somit von dem Netz auch nicht erkannt werden konnten. Bei meiner Modellentwicklung habe ich für die Vergleichbarkeit immer 10 Epochen verwendet, um die Entwicklungsstufen miteinander vergleichen zu können. An dieser Stelle noch der Versuch dem Modell 30 Epochen zu geben. Dabei hat das Netz eine validation Accuracy von 99,18% und einen loss von 0,0244 erzielt. Also über die Epochen kann mein Modell auch noch etwas besser werden. Dafür benötigt das Netz dann schon ein bisschen Zeit :).

Viel spannender ist jedoch die genauere Betrachtung der beiden Diagramme. Widererwarten ist die Accuracy höher und der Loss niedriger bei dem Validation-Set im Vergleich zum Train-Set. Nach doppelter Prüfung, ob die Achsenbeschriftung der Diagramme richtig gesetzt ist, folgte eine Recherche des Verhaltens im Internet. Laut Internetquellen[1] kann es tatsächlich bei der Verwendung von dropout Layern zu solch einem Verhalten kommen. Demnach habe ich die beiden dropout Layer auskommentiert und das Netz erneut trainiert/validiert mit folgendem Ergebnis:



In den Diagrammen ist eine deutliche Veränderung zu erkennen. Anfangs tritt das Phänomen immer noch auf, doch bereits nach der vierten Epoche stellt sich das gewohnte Verhalten ein. Dem entsprechend waren die dropout Layer sicherlich hauptsächlich die Ursache dafür, aber da das Verhalten anfangs noch nicht komplett weg ist muss es noch andere Gründe geben. Eine weitere mögliche recherchierte Ursache könnte eine ungleiche Verteilung der Daten sein. Gemeint ist damit, dass das Trainings-Set viel schwieriger als das Validation-Set ist. Allerdings ist das bei MNIST eher unwahrscheinlich, da ein stichprobenartiger Vergleich der Datensets dem widerspricht und da das Verhalten bei den vorigen Entwicklungsstufen meines Modells nicht aufgetreten ist. Allerdings könnte das übrig gebliebene „Einschwingen“ noch durch eine Unterscheidung der Berechnung von Training- und Validation-Accuracy zurückzuführen sein. Da aber hauptsächlich das Auskommentieren der dropout Layer die Unterscheidung gebracht hat, wird das als Hauptursache definiert.

An dieser Stelle ist tatsächlich aber der Sinn und Zweck der dropout Layer gezeigt und sogar bewiesen. Ohne die Beiden läuft mein Modell bereits nach der zehnten Epoche stark in Richtung overfitten. Hierbei wird die Funktion, eben das overfitten zu verhindern, der dropout Layer sehr schön dargestellt. Aus diesem Grund bleiben die dropout Layer in meinem Modell und beschriebenes Phänomen ist dementsprechend in meiner Software Contribution zu betrachten. Gerade für längere Trainings mit über zehn Epochen sind die dropout Layer bei meinem Aufbau sogar unerlässlich.

[1] Quelle: https://towardsdatascience.com/what-your-validation-loss-is-lower-than-your-training-loss-this-is-why-5e92e0b1747e

Zusammenfassung

Alles in Allem hat mein Netz eine gute Leistung erzielt und die Hauptidee mit der Verdichtung funktioniert auch. Das spannende Phänomen meines Modells konnte untersucht und überwiegend den dropout Layern zugeschrieben werden. Dabei wurde gleichzeitig die Notwendigkeit dieser nochmal dargestellt und mit Tests bewiesen. Allgemein kann das Netz noch weiter verfeinert werden, doch dazu ist mehr Zeit notwendig. Das Ergebnis war das Beste, was ich in dem gegebenen Zeitraum erzielen konnte und ich hoffe, dass es auch zu eurer Zufriedenheit ist. 😊

Ausblick

Als kleinen Ausblick an welchen Punkten ich gerne noch weitergearbeitet hätte: Bereits während meiner Modellentwicklung hatte ich noch weitere Layer verwendet, welche auch recht gute Ergebnisse erzielt haben. Diese in mein aktuelles Modell einzubinden könnte weiter die Leistung steigern. Meine Idee mit dem Daten Pre-Processing konnte ich nicht mehr in Verbindung mit dem aktuellen Netz testen. Das könnte ebenfalls noch Auswirkungen auf die Leistung haben, wobei das „Einschränken“ des Inputs durchaus auch negativ sein kann. Schließlich nutze ich auch die Aktivierungsfunktion relu, wobei das Setzen auf entweder null oder eins keine kontinuierliche Form mehr benötigt. Im Allgemeinen hätte ich gerne noch komplett andere Architekturen getestet, doch wie bereits erwähnt ging mir dann die Zeit aus. Sehr spannend war das zu untersuchende Phänomen der dropout-Layer, wobei an dieser Stelle auch noch nach weiteren Ursachen geforscht werden könnte. Vielleicht finde ich im Anschluss an dieses Projekt im Weihnachtsurlaub noch etwas Zeit dazu.

Ich wünsche euch viel Spaß beim Erkunden meines Binders und vielen Dank für eure Vorlesung. Sorry das mein Report etwas länger ist, ich hoffe ihr nehmt es mir nicht übel. Sind auch ein paar Abbildungen zwischen dem Text. 😊

Frohe Weihnachten wünscht Benjamin Seeger