Handschriftliche Ziffernerkennung mithilfe von Deep Learning unter Verwendung des MNIST-Datensatzes und eigens erstellter Daten

Larissa [redacted]
Matrikelnr.: [redacted]
E-Mail: [redacted]

Abstract

Das Ziel der vorliegenden Projektarbeit ist es, ein Modell zur handschriftlichen Ziffernerkennung zu implementieren und dieses mit dem MNIST-Datensatz zu trainieren und zu testen. Anschließend unterlaufen zusätzlich eigens erstellte Daten einem entsprechenden Pre-Processing und werden an das trainierte Modell übergeben. Dabei soll eine ähnliche *Accuracy* wie die der MNIST-Daten erreicht werden. Dazu werden unter anderem die Programmbibliothek PyTorch und ein *Convolutional Neural Network* verwendet. Die Ergebnisse zeigen zum einen mit etwa 99% eine gute Performance des Modells auf den Testdaten von MNIST und zum anderen eine etwas schlechtere *Accuracy* von circa 92 % auf dem eigenen Datensatz. Dieser Unterschied in der *Accuracy* kann nach eingehender Untersuchung dem Pre-Processing zugeschrieben werden. Um weitere Verbesserungen anzustreben, wäre dies folglich ein geeigneter Ansatzpunkt.

1 Einleitung

Die handschriftliche Ziffernerkennung ist die Fähigkeit eines Programms menschliche, handschriftlich verfasste Ziffern einlesen und korrekt erkennen zu können. Dies stellt für Maschinen kein einfaches Problem dar, da von Hand geschriebene Zahlen Charakteristiken des Schreibstiles des Verfassers enthalten. Eine *Neun* wird somit von einigen Menschen mit einem geraden Strich dargestellt, während sie von anderen mit einem kleinen Bogen unten geschrieben wird.
Diese Ausarbeitung befasst sich mit eben diesem Problem. Zunächst folgt ein Überblick über den verwendeten Datensatz. Im Anschluss daran werden die für den Programmcode verwendeten Bibliotheken näher beleuchtet. Überdies wird ein Modell zur handgeschriebenen Ziffernerkennung implementiert und mithilfe des MNIST-Datensatzes trainiert und getestet. Infolgedessen werden zusätzlich eigens erstellte Bilder, entsprechend den Bildern von MNIST, vorbereitet und an das trainierte Modell übergeben. Darauf folgend werden Untersuchungen an den Bildern des MNIST-Datensatzes, sowie der eigens erstellten Daten präsentiert. Abschließend folgen eine Evaluation der erzielten Ergebnisse, sowie ein Fazit.

2 Datensatz

Die MNIST-Datenbank (*Modified National Institute of Standards and Technology database*) stellt eine Vielfalt von 70.000 verschiedenen Bildern von einzelnen Ziffern zur Verfügung, von denen 60.000 Bilder für das Training gedacht sind und die restlichen 10.000 als Testdatensatz verwendet werden. Die MNIST-Datenbank basiert auf der älteren NIST-Datenbank, deren Daten auf Fotos von handschriftlich verfassten Ziffern und Buchstaben von Mitarbeitern der amerikanischen Bundesbehörde *United States Census Bureau*, sowie amerikanischen High School Schülern basieren. Die Trainingsbilder setzen sich ausschließlich aus der Handschrift der Mitarbeiter zusammen. Die NIST-Testbilder entstammen Bildern der High School Schüler. Für die MNIST-Datenbank wurden einzig die Bilder der Ziffern extrahiert und verwendet. Zudem bietet die MNIST-Datenbank den Vorteil, dass die Trainings- und Testbilder neu gemischt wurden, damit die neuen Trainings- und Testdatensätze beide nun sowohl das Schriftbild der Mitarbeiter, als auch das Schriftbild der High School Schüler enthalten. (Wikipedia 2021)
Der eigens erstellte Datensatz umfasst eine Anzahl von 60 Bildern und beinhaltet Handschriften von zwei Personen. Diese Bilder werden ausschließlich zum Testen des bereits trainierten Modells benutzt.

3 Verwendete Bibliotheken

Zur Entwicklung des Modells wird die Programmbibliothek PyTorch gewählt, die auf maschinelles Lernen ausgerichtet ist. Grund für die Auswahl von PyTorch ist die Vertiefung von, im Rahmen des Kurses 'Deep Learning in NLP', erlangtem Wissen. Des Weiteren werden Bibliotheken wie Matplotlib, NumPy, OpenCV, SciPy und Pillow unter anderem für das Pre-Processing und zum Sichten der MNIST-Bilder verwendet.

4 Ablauf

Zunächst wurden die MNIST-Bilder mithilfe der *DataLoader*-Klasse von PyTorchs Package *torch.utils.data* geladen und abgespeichert. Dazu wurde eine Instanz *train_loader* für die Trainingsdaten erstellt und eine weitere *test_loader* für die Testdaten.
Infolgedessen wurde unter Verwendung einer *Feedforward*-Struktur ein Modell implementiert. Dieses Modell beinhaltete drei Linear-Layers und zwei *rectified linear unit*-Funktionen (ReLU). Mithilfe dieser Struktur, einem *CrossEntropyLoss*, einer *batch size* von 100 und einer Lernrate von 0.001 wurde eine *Accuracy* von 97% auf den Testdaten erzielt.
Da die *Accuracy* bereits sehr hoch war, wurden als nächstes eigene Bilder erstellt und anschließend ebenfalls, wie die MNIST-Daten, mithilfe der *DataLoader*-Klasse geladen.
Zum Erstellen der Bilder wurde die Bildbearbeitungsfunktion eines Smartphones genutzt. In einem ersten Schritt wurde der Hintergrund eines Bildes mit schwarzer Farbe gefärbt. Darauf wurde mit einem Touch-Stift eine Ziffer in weißer Schrift mittig auf das Foto geschrieben und anschließend auf den PC geladen. Dort wurden die Bilder entsprechend in ein quadratisches Format geschnitten, sodass die Ziffer nahezu zentriert war und überschüssiges schwarz weggeschnitten wurde.
Das weitere Pre-Processing der Bilder gestaltete sich schwierig, da dieses dem Pre-Processing der MNIST-Bilder gleichen sollte.
Es wurden jedoch zwei Quellen gefunden, in denen jeweils unterschiedliche Verfahren beschrieben worden sind, die im Pre-Processing für die MNIST-Bilder Anwendung fanden. In beiden Fällen wurde nicht präzise genug beschrieben, wie und welche Methoden genau genutzt wurden. Das Pre-Processing konnte aus diesen Gründen nur näherungsweise immitiert werden.

Zunächst wurde das Verfahren, das von Michael Garris, einem der leitenden Wissenschaftler der Interessensgemeinschaft für künstliche Intelligenz am *National Institute of Standards and Technology* (NIST), in einem Vortrag präsentiert worden ist, angewendet. Er war maßgeblich an der Bildverarbeitung und Mustererkennung am Institut beteiligt (NIST o.D.). In seiner Präsentation darüber, wie der MNIST-Datensatz erstellt worden ist, beschreibt er, auf das orignale 128x128 Bild würde zunächst ein *Gaussian Blur* angewendet. Demnach wird das Bild durch eine Gaußsche Funktion weich gezeichnet. Auf das Resultat wird eine ROI Extraktion (*region of interest extraction*) angewendet, die eine Box um die einzelne Ziffer bildet und die entstandene Box aus dem Bild herausschneidet. Anschließend wird das extrahierte Bild zentriert in ein 28x28 Format gebettet (Analytics 2019).
Dieses Pre-Processing brachte im Zusammenhang mit dem Feedforward Netzwerk allerdings nur 3-4 von 10 korrekt erkannten Bildern hervor.

Die zweite Quelle ist die Webseite der MNIST-Datenbank. Auf dieser Seite wird beschrieben, dass die einzelnen originalen, schwarz-weiß Bilder in ein 20x20 Format gebettet wurden. Dazu wurde eine Anti-Aliasing Technik darauf angewendet, um mit der Kantenglättung einen etwas sanfteren Übergang von der weißen Ziffer in den schwarzen Hintergrund zu erlangen und somit weitere Grautöne in die Bilder einzufügen. Mithilfe der Berechnung des Massenmittelpunktes wurden die Ziffern zentriert und in ein 28x28 Format formatiert (LeCun o.D.).
Dieses Verfahren verbesserte die Bilanz nur leicht und gab Resultate aus, in denen 5-6 von 10 Bildern richtig erkannt wurden.

Durch ein *Debugging* des Programmcodes und ein Vergleichen der Pixel der MNIST-Bilder stellte sich heraus, dass der schwarze Hintergrund der eigens erstellten Bilder nicht die gleiche Farbnuance wie der Hintergrund der MNIST-Bilder hatte. Daher wurde nun anstelle eines schwarzen Hintergrunds ein weißer gewählt und die Farbe der Ziffern änderte sich von weiß zu schwarz. Dem weiteren Pre-Processing wurde ein Invertierungsschritt hinzugefügt, der die weiß-schwarzen Bilder

in schwarz-weiße Bilder invertieren sollte. Die *Accuracy* der eigens erstellten Bilder verbesserte sich mit dieser Methode auf circa 70 %.
Da diese *Accuracy* noch nicht annähernd der der MNIST-Testdaten entsprach, wurde eine andere Modell-Struktur gewählt. Da in der Bildverarbeitung größtenteils mit *Convolutional Neural Networks* (CNN) gearbeitet wird, wurde dieses Netzwerk nun auch für dieses Projekt gewählt.

Die *Accuracy* der Testbilder von MNIST ist mit dieser Art von Netzwerk und einer *batch size* von 64 auf $\approx$ 98,5% und die der eigens erstellten Bilder mithilfe des von Michael Garris beschriebenen Pre-Processings auf $\approx$ 85% gestiegen.
Da die *Accuracies* immer noch eine Differenz von circa 13% mit sich trugen, wurde nun zum einen versucht die beiden verschiedenen Pre-Processing Verfahren zu verfeinern und zu kombinieren und zum anderen ein paar Hyperparameter etwas abzuändern.
Schlussendlich wurden in der Funktion *firstpreprocessing* die Ausgangsbildgröße auf ein 860x860 Format verändert, der *Gaussian Blurr* abgeschwächt und die zuvor verwendete ROI Extraktion-Methode durch eine andere *Bounding Box*-Methode ausgetauscht.
In der zweiten Pre-Processing Funktion wurde die Ausgangsgröße der Bilder ebenfalls angepasst und die Massenmittelpunktberechnung entfernt.
Die Hyperparameter wurden so verändert, dass die *batch size* nun wieder 100 beträgt und die Lernrate von 0.1 auf 0.2 erhöht wurde.
Damit wurden die besten Ergebnisse erreicht. Die *Accuracy* der Testdaten stieg von 98,5% auf circa 98,8 – 99% und die Accuracy der eigens erstellten Bilder stieg mit der *firstpreprocessing*-Funktion auf durchschnittlich 92%.

5 Statistiken und Untersuchungen

Die *Accuracy* der eigens erstellten Bilder ist mit der *firstpreprocessing*-Funktion nun bereits relativ hoch, allerdings reicht sie noch nicht an die 99% der Testdaten heran. Um den bereits geäußerten Verdacht, dass es sich bei der Ursache für die Diskrepanz in den Ergebnissen um Unterschiede im Pre-Processing handelt, zu bestätigen, wurden hierzu weitere Untersuchungen durchgeführt. Dazu wurden ausschließlich Bilder gewählt, die mit der *firstpreprocessing*-Funktion vorbereitet worden sind, da die *Accuracy* damit bereits höher war als mit der zweiten Variante.
Im Folgenden werden nun die Untersuchungen und ihre Ergebnisse präsentiert.

Zunächst wurde das *Blurring* der Bilder überprüft und verglichen. Dazu wurde der Laplace-Operator verwendet, der in der *Laplacian*–Funktion von OpenCV genutzt wird. Diese Funktion kann dazu eingesetzt werden, das Bild auf Kanten zu überprüfen. Wenn ein Bild stark weich gezeichnet wurde, sind weniger Kanten in dem Bild vorhanden. Wenn das Bild allerdings kaum weich gezeichnet wurde, ist das Gegenteil der Fall und das Bild enthält viele Kanten.
Der Laplace-Operator hebt Bereiche eines Bildes hervor, die schnelle Intensitätsänderungen enthalten und tut dies mithilfe des Gradienten.
Die Funktion nimmt einen einzigen Kanal eines Bildes und faltet ihn mit der folgenden 3x3 Matrix:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

(pyimagesearch 2015).
Von der Ausgabe dieser Funktion wurde die Varianz berechnet. Je höher der daraus resultierende Wert ist, desto mehr Kanten bzw. schnelle Intensitätsänderungen existieren in dem Bild.

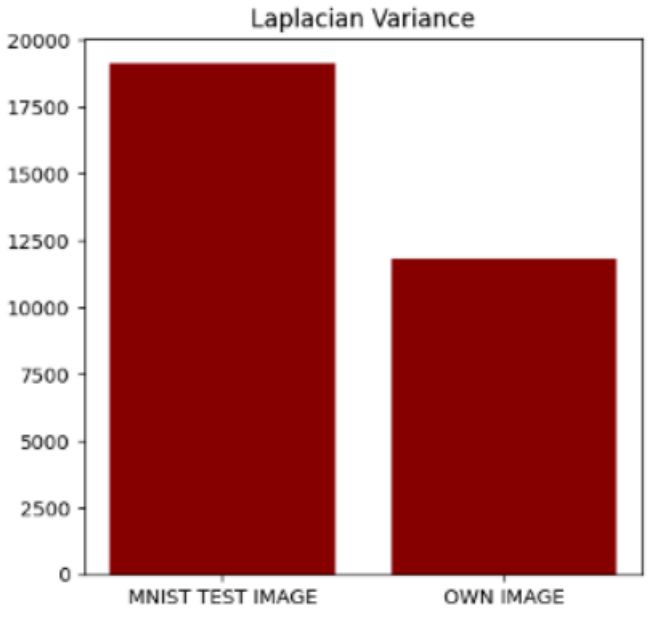

Figure 1: Die Abbildung zeigt die Varianz eines MNIST-Testbildes und eines eigens erstellten Bildes, nachdem die *Laplacian*-Funktion auf die Bilder angewandt worden ist.

Figure 1 stellt den Unterschied dieser Varianz, die

auf der Grundlage des mit dem Laplace-Operator berechneten Wert ermittelt wurde, dar. Mit einem Wert von circa 19.000 ist deutlich zu erkennen, dass das MNIST-Bild mehr Kanten enthält als das eigens erstellte Bild, welches einen Wert von circa 12.000 hat.
Auch nachdem der *Gaussian Blurr* etwas abgeschwächt worden ist und versucht wurde eine dickere Schrift zu verwenden, blieb das Ergebnis das gleiche. In Figure 2 ist oben links

Figure 2: *links oben und unten*: MNIST-Testbild
rechts oben: eigens erstelltes Bild
rechts unten: eigens erstelltes Bild mit weniger *Blurring* und dickerer Schrift

das MNIST-Bild zu sehen, das auch für die Varianz in Figure 1 als Beispiel diente. Rechts daneben ist das Bild aus dem eigens erstellten Datensatz dargestellt, welches ebenfalls in Figure 1 verwendet worden ist. Aus Figure 2 wird deutlich, dass nach dem abgeschwächteren *Blurring* und unter der Anpassung der Schriftdicke (siehe Bild unten rechts) zwar ein intensiveres Weiß entstanden ist, dies jedoch nicht dem Weiß des MNIST-Bildes gleicht.
Die Varianz des eigens erstellten Bildes bewegt sich trotz dessen ebenfalls noch im gleichen Spektrum wie die Varianz des Bildes in Figure 1.

Um zu überprüfen, wie sich die Bilder auf einer kleineren Ebene genau unterscheiden, wurden in einem nächsten Schritt die Pixel der Bilder betrachtet. Auch dort ist ein starker Unterschied in der Intensitätsänderung zu beobachten.
Während in den MNIST-Bildern relativ starke Kontraste vom Weißen ins Schwarze zu beobachten sind, sind bei den eigens erstellten Bildern viele Zwischenstufen vorhanden.

Im folgenden Code-Output ist anhand der Pixel festzustellen, dass das MNIST-Bild deutlich weniger Graustufen beinhaltet als das eigens erstellte Bild. So haben 658 Pixel die Nuance Schwarz (= 0) und 97 Pixel die Nuance Weiß (= 251).
Das MNIST-Bild weist zwei Maxima auf. Das eine Maximum liegt bei Schwarz und das andere bei Weiß. In dem anderen Bild gibt es jedoch nur ein Maximum, welches bei Schwarz liegt. Die restlichen Pixel verteilen sich nicht größtenteils auf ein Weiß, sondern auf mehrere Grautöne und Nuancen von Weiß.

MNIST-Bild:

```
mnist.getcolors()

Output: [(658, 0), (1, 86), (1, 94),
(2, 105), (1, 108), (1, 109), (1, 113),
(2, 114), (1, 115), (1, 118), (4, 124),
(1, 131), (1, 134), (1, 157), (2, 167),
(1, 170), (2, 172), (1, 177), (1, 185),
(1, 205), (1, 210), (1, 218), (1, 229),
(97, 251)]
```

Eigens erstelltes Bild:

```
own_img.getcolors()

Output: [(567, 0), (8, 1), (4, 2),
(1, 3), (4, 4), (3, 5), (2, 6),
(1, 7), (2, 9), (1, 10), (1, 12),
(2, 15), (3, 19), (1, 20), (3, 22),
(2, 24), (1, 26), (2, 27), (2, 28),
(1, 31), (2, 32), (1, 33), (1, 34),
(1, 37), (1, 39), (1, 40), (2, 41),
(1, 43), (1, 48), (1, 50), (1, 51),
(2, 52), (1, 53), (1, 55), (2, 56),
(1, 58), (1, 59), (1, 61), (1, 66),
(1, 67), (1, 68), (1, 69), (1, 72),
(1, 73), (2, 78), (2, 85), (1, 86),
(1, 89), (1, 90), (1, 91), (1, 95),
(1, 97), (1, 99), (2, 101), (1, 103),
(1, 105), (1, 110), (2, 113), (2, 114),
(2, 121), (1, 125), (1, 128), (1, 129),
(1, 131), (1, 132), (2, 134), (1, 135),
(1, 136), (1, 137), (1, 138), (1, 141),
(1, 145), (1, 146), (1, 150), (1, 153),
(2, 162), (1, 168), (1, 169), (1, 170),
(2, 173), (1, 174), (1, 178), (1, 179),
(1, 180), (2, 181), (1, 183), (1, 187),
(1, 188), (1, 189), (1, 190), (3, 191),
(1, 193), (2, 195), (1, 196), (1, 199),
(1, 205), (1, 206), (2, 207), (1, 210),
(1, 212), (1, 213), (2, 214), (3, 215),
(1, 216), (2, 217), (1, 218), (1, 220),
(1, 221), (5, 222), (1, 224), (3, 225),
(2, 226), (2, 227), (2, 228), (1, 229),
(2, 230), (1, 232), (1, 234), (8, 236),
(3, 237), (2, 238), (4, 239), (2, 240),
(2, 241), (4, 242), (4, 243), (3, 244),
(3, 245), (1, 246), (1, 247), (3, 248),
(3, 249)]
```

6 Evaluation und Fazit

Im Zuge dieser Projektarbeit sollte ein Modell zur handschriftlichen Ziffernerkennung mithilfe der Programmbibliothek PyTorch implementiert, trainiert und getestet werden. Des Weiteren sollten eigens erstellte Daten an das trainierte Modell übergeben und eine möglichst gute *Accuracy* erzielt werden, die der *Accuracy* der MNIST-Testdaten ähnelt.
Um dieses Ergebnis zu erreichen, wurde ein *Convolutional Neural Network* verwendet und ein Pre-Processing zusammengestellt, das die eigenen Daten so vorbereitet, dass sie auf das Modell appliziert werden können.
Obwohl die Ergebnisse gut waren und eine *Accuracy* von circa 92% auf dem eigenen Datensatz erreicht worden ist, reichten die Ergebnisse noch nicht an die Performance des MNIST-Datensatzes heran.
Nachdem daher einige Untersuchungen an den Bildern durchgeführt worden sind, hat sich herausgestellt, dass es einen Unterschied in der Intensität der Farbe und der Kantenmenge gibt.

Um das gesetzte Ziel zu erreichen, eine nahezu perfekte *Accuracy* wie die der MNIST-Bilder zu erlangen, gibt es nun verschiedene Möglichkeiten, die noch getestet werden könnten.
Zum einen könnten eigens erstellte Bilder unter die Trainingsdaten von MNIST gemischt werden, um eine bessere Performance auf den eigenen Bildern beim Testen zu erzielen. Im Rahmen dieser Projektarbeit war dies aufgrund der zeitlichen Begrenzung jedoch nicht mehr möglich.
Eine weitere Möglichkeit bestünde darin, das *Blurring* weiter zu untersuchen und zu verbessern. Es könnte nach Methoden gesucht werden, die im Bezug auf die Farben, ebenso wie bei den MNIST-Daten, zwei Maxima hervorbringen und die abundanten, graustufigen Pixel der eigens erstellten Bilder in ein und dasselbe Weiß umwandeln.
Eine mögliche Methode wäre, über die Pixel zu iterieren und jeden Bildpunkt, der einen kleineren Wert als einen vordefinierten Grenzwert hat, gleich null zu setzen. Jedes Pixel, welches über diesen vordefinierten Wert fällt, wird auf einen benutzerdefinierten Wert gesetzt. Somit würde ein binäres Bild entstehen, welches zwei Maxima aufweist. Wird der Grenzwert entsprechend definiert, könnte möglicherweise ein Bild entstehen, das dem MNIST-Bild ähnelt.

Literatur

Analytics, Lander (März 26, 2019). *The Story of the MNIST Dataset[Video].* YouTube. Abgerufen am 13. Februar 2021. URL: https://www.youtube.com/watch?v=oKzNUGz21JM.

blog, Jonathan Hui (Feb. 2018). *PyTorch - Data loading, preprocess, display and torchvision.* Abgerufen am 10. März 2021. URL: https://jhui.github.io/2018/02/09/PyTorch-Data-loading-preprocess_torchvision/.

datahacker (Dezember 11, 2019). *014 PyTorch – Convolutional Neural Network on MNIST in PyTorch.* Abgerufen am 10. März 2021. URL: http://datahacker.rs/005-pytorch-convolutional-neural-network-on-mnist-in-pytorch/.

Electroica, Blog (Aug. 2020). *Select ROI or Multiple ROIs [Bounding box] in OPENCV python.* Abgerufen am 13. Februar 2021. URL: https://blog.electroica.com/select-roi-or-multiple-rois-bounding-box-in-opencv-python/#select-roi-in-opencv-python.

LeCun, Yann (o.D.). *THE MNIST DATABASE of handwritten digits.* Abgerufen am 10. März 2021. URL: http://yann.lecun.com/exdb/mnist/.

NIST (o.D.). *Michael Garris.* Abgerufen am 13. Februar 2021. URL: https://www.nist.gov/blogs/taking-measure/authors/michael-garris.

pyimagesearch (Sep. 2015). *Blur detection with OpenCV.* Abgerufen am 10. März 2021. URL: https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/.

PyTorch (2017). *Training a classifier.* Abgerufen am 10. März 2021. URL: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.

science, towards data (Feb. 2019). *Handwritten Digit Recognition Using PyTorch - Intro to Neural Networks.* Abgerufen am 18. Februar 2021. URL: https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627.

AI-SPECIALS (Juni 2020). *Image Classification using CNN from Scratch in Pytorch- Part1 Training.* YouTube. Abgerufen am 16. Februar 2021. URL: https://www.youtube.com/watch?v=9OHlgDjaE2I.

stackoverflow (Dezember 07, 2018). *Match center of two images (OpenCV, Python)*. Abgerufen am 23. Februar 2021. URL: https://stackoverflow.com/questions/48280828/match-center-of-two-images-opencv-python.

Wikipedia (März 10, 2021). *MNIST database*. Abgerufen am 15. März 2021. URL: https://en.wikipedia.org/wiki/MNIST_database.

wikipedia (Feb. 2020). *Laplace-Operator*. Abgerufen am 07. März 2021. URL: https://de.wikipedia.org/wiki/Laplace-Operator.

A Anhang

Programmcode:https://github.com/larissa0898/deep_learning_project