ERKENNUNG VON HANDGESCHRIEBENEN ZIFFERN MNIST DIGITS

Tien Dee Lin
Student Wirtschaftsinformatik - Data Science
Duale Hochschule Baden-Württemberg
Stuttgart
tiendee.lin@gmail.com

ABSTRACT

Diese Arbeit demonstriert den Einsatz eines kompakten Convolutional Neural Networks (CNN) zur Erkennung handgeschriebener Zahlen aus dem MNIST Digits Datensatz. Mithilfe von PyTorch und dem Tool „Weights & Biases“ wird ein Modell mit geringen Parametern und Trainingsepochen entwickelt, das effektive Bildklassifikationen durchführen kann. Die Ergebnisse zeigen die grundsätzliche Einfachheit der Klassifikationsaufgabe und lassen vermuten, dass mit zunehmender Perfektionierung der Klassifikationsgenauigkeit der erforderliche Aufwand exponentiell steigt. Der dokumentierte Quellcode dieses Projektes samt Anleitung zum Start befindet sich im folgenden GitHub-Repository: `https://github.com/TienDeeLInPrivate/mnist_digits_cnn.git`

***K*eywords** MNIST Digits · Convolutional Neural Network · Bildklassifikation

1 Einleitung

In dieser Arbeit wird die Anwendung eines Convolutional Neural Networks zur Erkennung handgeschriebener Zahlen aus dem MNIST-Digits-Datensatz dargelegt. Aufgrund des Projektschwerpunkts auf der Demonstration korrekter Methodik wird ein kompaktes Modell mit einer geringen Anzahl an Trainingsepochen und einer begrenzten Auswahl an Hyperparametern verwendet. Die Methoden zur Datenvorverarbeitung und die Modellarchitektur wurden durch den Beitrag von Gregor Köhler auf www.nextjournal.com inspiriert und teils mit Veränderungen übernommen.[1] Die Implementierung erfolgte in PyTorch, ergänzt durch das ML-Lifecycle-Management-Tool „Weights & Biases“.

2 MNIST Digits Datensatz

Der MNIST Digits-Datensatz, der aus der MNIST-Datenbank stammt, umfasst 70.000 handgeschriebene Ziffern, jeweils versehen mit dem zugehörigen Label, das die intendierte Ziffer angibt. Für diese Arbeit wird der Datensatz von www.kaggle.com verwendet.[2] Die Daten sind in einen Trainingsdatensatz mit 60.000 und einen Testdatensatz mit 10.000 Einträgen unterteilt. Jede Ziffer ist zentral auf einem 28x28 Pixel großen Bild positioniert und ausschließlich in Graustufen dargestellt, wobei jeder Pixel Werte zwischen 0 und 255 annehmen kann. Ein Pixelwert von 0 repräsentiert dabei den Hintergrund, während höhere Werte die handgeschriebene Ziffer selbst darstellen und die Intensität mit steigendem Wert bis maximal 255 zunimmt. In der nachfolgenden Visualisierung (Abbildung 1), die mit matplotlib.pyplot erstellt wurde, repräsentieren die 0-Werte einen schwarzen Hintergrund und die höheren Werte werden als weiße, gezeichnete Pixel dargestellt.

[1] https://nextjournal.com/gkoehler/pytorch-mnist (letzter Zugriff am 27.07.2024)
[2] https://www.kaggle.com/datasets/hojjatk/mnist-dataset (letzter Zugriff am 27.07.2024)

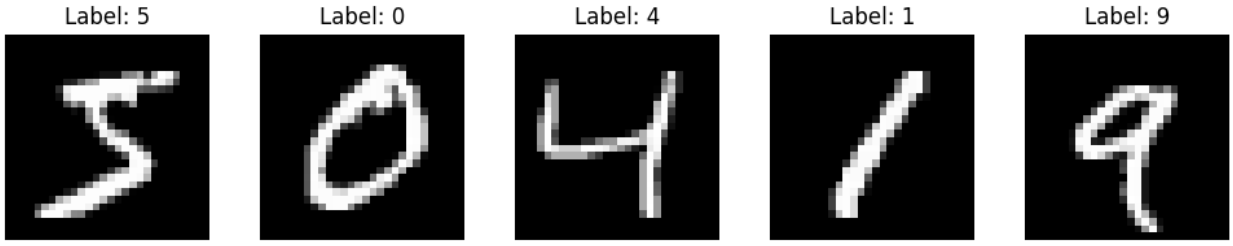

Abbildung 1: Visualisierung von 5 Ziffern aus dem Trainingsdatensatz

2.1 Optimierungsziel

In der vorliegenden Arbeit wird die Erkennung handgeschriebener Ziffern als Bildklassifizierungsaufgabe definiert, wobei die Ziffern 0 bis 9 die möglichen Klassen darstellen. Jedes Bild wird einer dieser Ziffern zugeordnet und dementsprechend klassifiziert. Es wird explizit darauf verzichtet, bestimmte Ziffern als besonders signifikant für die Klassifizierungsleistung zu priorisieren. Stattdessen besteht das primäre Optimierungsziel darin, ein Modell zu entwickeln, das die Klassifizierungsgenauigkeit über das gesamte Spektrum der Ziffern maximiert. Die zur Bewertung der Modell- und Klassifizierungsleistung herangezogene Metrik wird in Kapitel 4 erläutert.

2.2 Datenqualität und Balanciertheit

Der verwendete Datensatz besteht aus Ziffern, die von 250 verschiedenen Personen handgeschrieben wurden, wobei 50 Prozent der Beiträge von Schülern einer Sekundarschule und die anderen 50 Prozent von Mitarbeitern eines Zensusbüros stammen. Die Varianz in der Handschriftqualität, insbesondere die tendenziell unordentlicheren Ziffern der Schüler, trägt dazu bei, den Datensatz realistischer zu gestalten, indem auch weniger geordnete Handschriften zuverlässig erkannt werden müssen. Es lässt sich feststellen, dass der Datensatz Daten von hoher Qualität umfasst, die mit minimaler Vorverarbeitung einsatzbereit sind. Die Trainings- und Testdatensätze sind separiert und enthalten keine überlappenden Datenpunkte. Aufgrund der Beliebtheit des Datensatzes und der bisherigen Nutzung durch verschiedene Autoren wird davon ausgegangen, dass der Datensatz frei von Fehlern und schwerwiegenden Ausreißern ist. Die Verteilung der Bilder auf die entsprechenden Labels wird als ausgewogen betrachtet, da alle Ziffern relativ gleichmäßig repräsentiert sind (siehe Tabelle 1).

Ziffer-Label	0	1	2	3	4	5	6	7	8	9
Anzahl der Bilder	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
Prozentualer Anteil	9.87	11.24	9.93	10.22	9.74	9.04	9.86	10.44	9.75	9.92

Tabelle 1: Ziffernverteilung im Datensatz nach Anzahl und Prozentanteil

3 Datenvorverarbeitung

Wie in Kapitel 2.2 ausgeführt, erfordert die hohe Qualität und Einsatzbereitschaft des Datensatzes keine zusätzliche Datenbereinigung. Auch werden alle Features der Bilder verwendet und keine Features entfernt. Ursprünglich im ubyte-Format vorliegend, werden die Daten nach dem Herunterladen in Numpy-Arrays konvertiert und anschließend in Pytorch-Tensoren umgewandelt. Um die Stabilität des Lernprozesses zu erhöhen und Verzerrungen zu vermeiden, wird eine Normalisierung vorgenommen, indem alle Pixelwerte durch den Maximalwert 255 dividiert werden, was die Werte auf einen Bereich von 0 bis 1 skaliert. Anschließend werden der Mittelwert und die Standardabweichung aller Pixelwerte über den gesamten Datensatz berechnet. Im folgenden Schritt der Standardisierung wird von jedem Pixelwert der Mittelwert subtrahiert und durch die Standardabweichung dividiert. Diese Standardisierung transformiert die Pixelwerte derart, dass die Gesamtverteilung einen Mittelwert von 0 und eine Standardabweichung von 1 aufweist. Der Zweck dieser Maßnahme liegt darin, dass durch die Standardisierung ein Teil der Pixelwerte negativ wird, was eine effektivere Nutzung des nichtlinearen Bereichs der Aktivierungsfunktionen ermöglicht, ohne dass ein Bias zwingend erforderlich ist.

Im letzten Schritt der Datenvorverarbeitung werden die Trainings- und Testdaten zusammengeführt und anschließend neu aufgeteilt. Die neue Verteilung umfasst 60 Prozent Trainingsdaten, 20 Prozent Validationsdaten und 20 Prozent Testdaten. Diese Aufteilung wird gewählt, um die Optimierung der Hyperparameter des Klassifikationsmodells an den Validierungsdaten vorzunehmen. Nach Abschluss dieser Optimierung ermöglicht der Testdatensatz, der ausschließlich aus zuvor ungesehenen Daten besteht, eine objektive Bewertung der Gesamtleistung des Modells mit den optimierten Hyperparametern. Für die neue Aufteilung des Datensatzes wird die „random_split“ Methode des PyTorch-Moduls verwendet, wodurch Trainings-, Validations- und Testdatensätze ebenfalls eine balancierte Verteilung über alle Ziffern hinweg aufweisen.

4 Zielmetrik

In dieser Arbeit wird die Accuracy (Genauigkeit) als Zielmetrik verwendet, die den Anteil der korrekten Vorhersagen im Verhältnis zu allen getätigten Vorhersagen angibt. Speziell für diese Aufgabe der Bildklassifizierung handgeschriebener Zahlen beschreibt die Accuracy-Metrik den Anteil der korrekt erkannten Ziffern im Vergleich zur Gesamtzahl der klassifizierten Testbilder innerhalb eines bestimmten Durchlaufs oder einer Betrachtungsspanne. Die Berechnung der Genauigkeit erfolgt nach folgender Formel:

$$\text{Accuracy} = \frac{\text{Anz. korrekt erkannter Ziffern}}{\text{Gesamtzahl der Testbilder}}$$

Die Entscheidung für diese Metrik stützt sich wesentlich auf drei Gründe: Erstens zeichnet sich der Datensatz durch eine balancierte Verteilung aus, da alle Ziffern annähernd gleich häufig auftreten. Zweitens wird im Rahmen dieser Arbeit keinen spezifischen Fokus auf die differenzierte Erkennung einzelner Ziffern gelegt. Drittens fördert die Verwendung der Accuracy-Metrik die Vergleichbarkeit der Ergebnisse erheblich, da sie auch in einschlägigen wissenschaftlichen Publikationen zu MNIST Digits herangezogen wird, wie beispielsweise in den Arbeiten von An et al. (2020)[3] und Hossain, Ali (2019).[4]

5 Klassifikationsmodell

Für die vorliegende Klassifikationsaufgabe wurde ein Convolutional Neural Network (CNN) ausgewählt, welches die normalisierten und standardisierten Pixelwerte eines Bildes als Eingabe verwendet. Als Ausgabe liefert das Modell eine Wahrscheinlichkeitsverteilung über die möglichen Ziffern, die anzeigt, mit welcher Wahrscheinlichkeit das Bild der handgeschriebenen Zahl jeder spezifischen Ziffer zugeordnet wird. Im Folgenden werden die Architektur des CNN sowie der Trainingsprozess detailliert erläutert.

5.1 CNN-Architektur

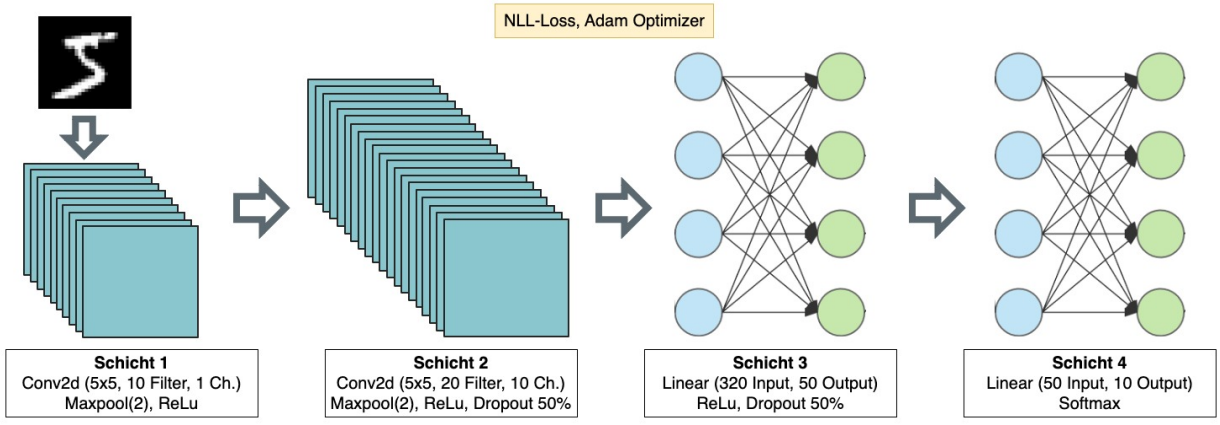

Abbildung 2: CNN-Architektur mit 4 Schichten

[3] An et al. 2020: „An ensemble of simple convolutional neural network models for MNIST digit recognition“
[4] Hossain, Ali 2019: „Recognition of Handwritten Digit using Convolutional Neural Network (CNN)“

Das CNN besteht aus vier Schichten: zwei Convolution-Schichten und zwei Fully-Connected-Schichten. Die erste Convolution-Schicht verarbeitet 28x28 Pixel große Bilder mit normalisierten und standardisierten Graustufenwerten, wobei 10 Filter mit 5x5 Kernels (ohne Padding, Stride von 1) eingesetzt werden, um 10 24x24 Feature Maps zu erzeugen. Nach einem 2x2 Max-Pooling werden diese zu 10 12x12 Feature Maps reduziert, auf die anschließend die ReLu-Aktivierungsfunktion angewendet wird und die als Input für die zweite Schicht dienen.

In der zweiten Schicht, die ebenfalls 5x5 Kernels ohne Padding und mit Stride 1 verwendet, wird der Output der ersten Schicht als ein Input mit 10 Kanälen betrachtet, wobei jeder Kanal einer Feature Map der ersten Schicht entspricht. Jeder der 20 Filter in dieser Schicht besitzt 10 solcher Kernels, um jeden Kanal individuell zu verarbeiten. Die Ergebnisse der 10 Kernels jedes Filters werden zu einem einzelnen 8x8 Feature Map zusammengefügt, indem die vollständigen Ergebnisse nach Anwendung der zugehörigen Kernels (also Feature-Maps) addiert werden. Nach der Convolution wird ein Dropout von 50% auf die Feature Maps angewendet, um Overfitting entgegenzuwirken. Dies führt zu 20 aggregierten 8x8 Feature Maps, die nach einem weiteren 2x2 Max-Pooling und der ReLu-Aktivierung zu 20 verschiedenen 4x4 Feature Maps reduziert werden.

Die dritte Schicht nimmt die 320 Werte aus den 20 4x4 Feature Maps der zweiten Schicht direkt als Eingabefeatures auf. Diese Fully-Connected-Schicht verfügt über 320 Eingabefeatures und 50 Ausgangsneuronen, welche die ReLu-Aktivierungsfunktion nutzen. Der Output dieser Schicht sind 50 Werte aus den Ausgangsneuronen, auf die ebenfalls ein Dropout von 50% angewendet wird.

Die letzte Schicht, ebenfalls eine Fully-Connected-Schicht, verarbeitet 50 Ausgabewerte aus der dritten Schicht als Eingabe. Sie besteht aus 10 Neuronen, auf welche die Softmax-Aktivierungsfunktion angewendet wird. Diese Funktion transformiert Neuronenausgaben in eine Wahrscheinlichkeitsverteilung, die für jede Ziffer die Wahrscheinlichkeit ihrer Zugehörigkeit anzeigt. Das endgültige Bildlabel wird anhand der Ziffer mit der höchsten Wahrscheinlichkeit bestimmt.

6 Training

Das Modell wurde in einem Train-Validation-Test Zyklus entwickelt. Während der Trainingsphase wurden die trainierbaren Parameter des Modells auf die Minimierung der Verlustfunktion hin angepasst, in der Validierungsphase wurden die Hyperparameter bezüglich der Zielmetrik optimiert, und die Testphase wurde einmalig am Ende des Prozesses durchgeführt, um die Leistung des finalen Modells mit optimierten Hyperparametern auf einem ungesehenen Datensatz zu evaluieren.

Das Training des neuronalen Netzes erfolgte auf einem Trainingsdatensatz mit 42.000 annotierten Bildern, welcher über 5 Epochen trainiert wurde. Die Daten wurden mittels eines Dataloaders in Mini-Batches in den Trainingsprozess eingeführt. Die Gewichtsanpassungen wurden mithilfe des Adam-Optimizers vorgenommen, der wegen seiner hohen Effizienz und adaptiven Lernrate ausgewählt wurde. Zur Berechnung des Verlustes wurde der Negative Log-Likelihood-Loss verwendet, welcher die Unsicherheit bei der Vorhersage der korrekten Klasse bestraft. Um den Entwicklungsprozess zu unterstützen, wurde das ML-Lifecycle-Management-Tool „Weights & Biases“ verwendet, mit dem der Trainingsverlust über die 5 Epochen dokumentiert wurde.

Abbildung 3 illustriert den Trainingsverlauf, aufgezeichnet pro 100 Batches (X-Achse). Aus der Visualisierung ist ersichtlich, dass der Trainingsverlust (Y-Achse) zunächst stark abnimmt und in späteren Epochen leicht oszilliert, jedoch ein insgesamt stabiles Verhalten mit nur geringfügigen Verlustabnahmen zeigt. Dies legt nahe, dass die Wahl von 5 Trainingsepochen angemessen war, da zusätzliche Epochen das Modell nicht signifikant verbessern würden.

7 Optimierung der Hyperparameter

Die Optimierung der Hyperparameter erfolgte auf einem Validierungsdatensatz, der 14.000 annotierte Bilder umfasst, basierend auf der in Kapitel 4 beschriebenen Accuracy-Metrik. Im Rahmen der Optimierung wurden die Batch-Größe und die Lernrate angepasst. Für die optimale Parameterkombination wurden Batch-Größen von 16, 32, 64 und 128 sowie Lernraten von 0.0001, 0.001 und 0.01 evaluiert. Die Optimierung wurde mithilfe von Weights & Biases Sweeps durchgeführt und alle Parameterkombinationen mit dem Gridsearch-Algorithmus druchsucht. Dabei erwies sich eine Batchgröße von 16 und eine Lernrate von 0,001 als optimale Parameterkombination, die auf dem Validierungsdatensatz nach 5 Epochen eine Accuracy von etwa 98,336% erreichte.

Abbildung 4 zeigt drei der zwölf Optimierungsdurchläufe und illustriert die Validierungs-Accuracy (Y-Achse) über die 5 Epochen (X-Achse). Jeder Graph verdeutlicht die Unterschiede in der Accuracy, abhängig von der Parameterkombination und Epoche. Hier zeigt sich ebenfalls, dass die Modellleistung in der ersten Epoche signifikant zunimmt und sich in späteren Epochen stabilisiert. Insgesamt trugen die Hyperparameter innerhalb der getesteten Bandbreite zu einer Schwankung der Klassifikations-Accuracy von etwa 5% bei.

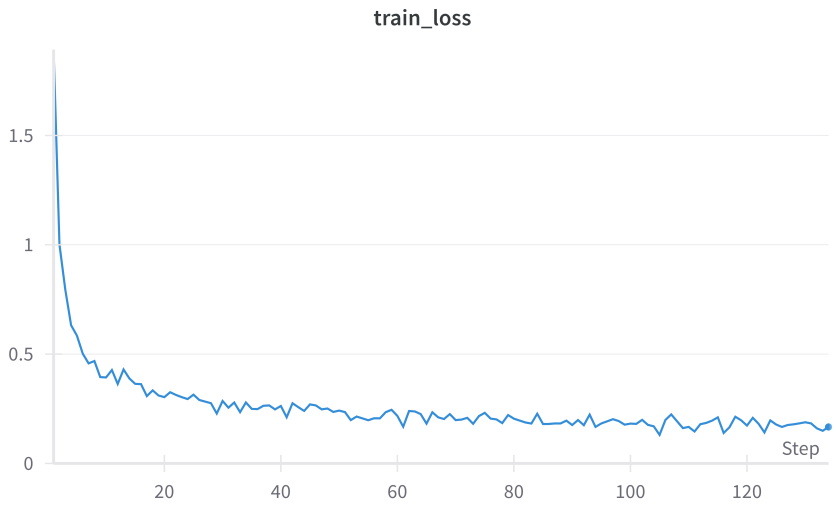

Abbildung 3: Trainingsverlust aufgezeichnet pro 100 Batches

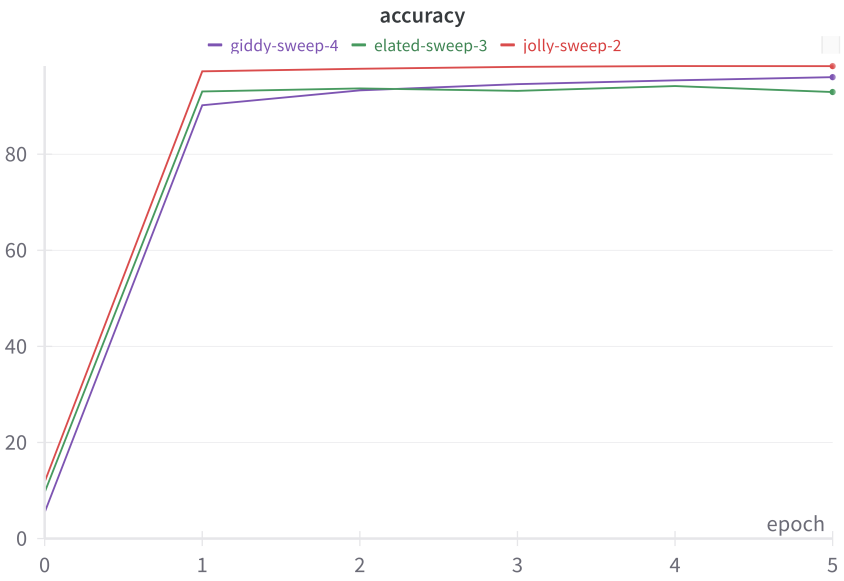

Abbildung 4: Validation-Accuracy für 3 verschiedene Hyperparameter-Kombinationen

8 Ergebnis und Fazit

Das finale Modell erreicht mit optimierten Hyperparametern eine Accuracy von etwa 98,2% auf dem Testdatensatz, wobei 13.748 der 14.000 Testbilder korrekt klassifiziert wurden. Die Confusionsmatrix in Abbildung 5 visualisiert die Klassifikationsergebnisse der Testdaten. Sie zeigt nicht nur eine relativ ausgewogene Verteilung der Labels im Datensatz, sondern auch die hohe Klassifikationsgenauigkeit. Zudem lassen sich einige intuitive Fehlklassifikationen beobachten, wie beispielsweise die häufige Verwechslung der Ziffer „2“ mit der „7“ oder der „8“ mit der „6“.

Dieses Modell übertrifft deutlich den Dummy Classifier von sklearn.dummy, welcher stets die häufigste Klasse des Testdatensatzes vorhersagt und eine Accuracy von 10,97% erreicht. Zum weiteren Vergleich wird das „M7“-Modell aus der Arxiv-Publikation von An et al. (2020) herangezogen, dessen Architektur in Abbildung 6 dargestellt ist und auf

dem originalen Testdatensatz mit 10.000 annotierten Bildern eine Accuracy von 99,79% erzielt.[5] Trotz seiner 2.336.972 Parameter, im Gegensatz zu den 21.840 Parametern des in dieser Arbeit trainierten Modells, bietet das „M7“-Modell nur eine marginale Leistungssteigerung. Diese Beobachtung könnte darauf hindeuten, dass die Herausforderung des MNIST-Datensatzes weniger in der grundlegenden Klassifikation liegt als vielmehr in der Feinabstimmung um die letzten Prozentpunkte der Genauigkeit. Der hochqualitative MNIST-Datensatz, mit einfachen 28x28 Pixel großen Bildern in einem Farbkanal und Tausenden von Trainingsbeispielen pro Ziffer, könnte vielen Modellen unterschiedlicher Konfigurationen ermöglichen, eine relativ hohe Genauigkeit zu erreichen. Die größte Herausforderung bestünde demnach darin, die Klassifikation nahezu perfekt auszuführen, wobei dafür die erforderliche Modellkomplexität und der Trainingsaufwand exponentiell ansteigen.

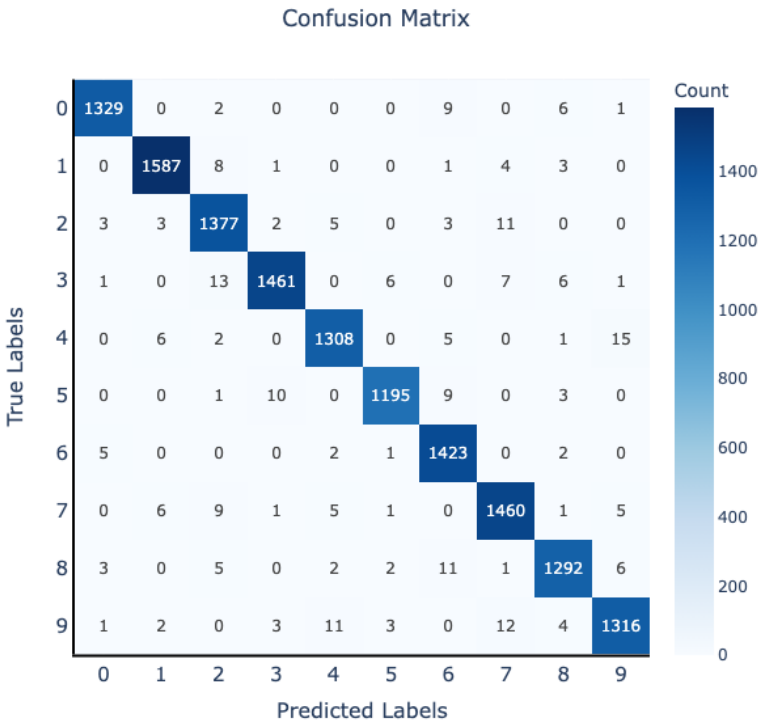

Abbildung 5: Klassifikationsergebnisse der Testdaten

[5] Vgl. An et al. 2020 S. 3

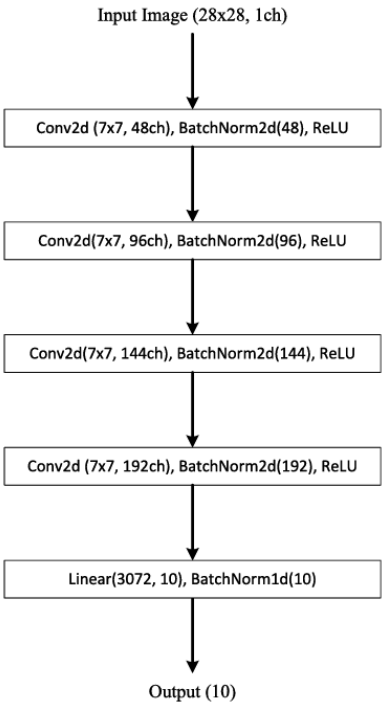

Abbildung 6: Modellarchitektur „M7“ (An et al. 2020 S. 2)