MNIST Classification Network Development

Basierend auf Keras und TensorFlow

Ziel & Grundlagen

Das Ziel dieser Arbeit war das Erstellen und Verbessern eines Neuronalen Netzes zur Klassifizierung der MNIST-Datenbank. Dabei handelt es sich um einen Datensatz von 60.000 handgeschriebenen Zahlen, welche das Netz zuverlässig erkennen soll.

Daten

Die Trainingsdaten des MNIST-Datensatzes bestehen aus 60.000 handgeschriebenen Zahlen. Um aus diesen Daten das maximale Trainingspotenzial zu schöpfen, müssen diese vor dem Training noch vorbereitet werden.

Zunächst werden die Eingangsdaten durch eine NumPy-Methode (*np.stack*) in TensorFlow-Tensoren umgewandelt. Dies ermöglicht eine schnellere und effizientere Verarbeitung, da TensorFlow diese auf die GPU verlagern kann und dies den Rechenprozess beschleunigt.

Dann werden die Daten normalisiert und umgeformt, um dem Eingangsformat für das Keras-Modell zu entsprechen.

Das Netz soll nun nicht auf allen 60.000 Bildern trainiert werden, um die Datenmenge kleinzuhalten. Dafür werden zufällig 10.000 der Label für das Training ausgewählt

Diese werden nun für das Training augmentiert. Das bedeutet, dass die Bilder teilweise zufällig gedreht, gezoomt oder zugeschnitten werden, um dem Netz noch mehr Trainingsmaterial zur Verfügung zu stellen und es vor Overfitting zu schützen (A Complete Guide to Data Augmentation | DataCamp).

Dies geschieht in einer integrierten TensorFlow-Methode (*datagen*), in welcher die Parameter zur Änderung der Bilder festgelegt werden können.

Netzwerk

Als Basis wurde zu Beginn das im Notebook vorhandene Netzwerk trainiert.

Dies wurde nach Inspiration durch mehrere Fach-Artikel zur MNIST-Thematik (z.B. How to reduce training parameters in CNNs while keeping accuracy >99% | by Sabrina Göllner | Towards Data Science) mehrfach überarbeitet und getestet, bis folgende Architekur entstand.

Die Anzahl der Parameter ist im Vergleich zum ursprünglichen Netzwerk deutlich geringer, was an den weggelassenen Dense-Layers liegt. Um diese zu kompensieren, ohne das Hauptnetzwerk unnötig zu vergrößern, wurde ein Netzwerk nach LeNet-5 Architektur als Backbone integriert

(*Benali Amjoud, A., & Amrouch, M. (2020). Convolutional Neural Networks Backbones for Object Detection. Image and Signal Processing: 9th International Conference*)

Hauptmodell:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (1, 26, 26, 16)           160

 batch_normalization (Batch  (1, 26, 26, 16)           64
 Normalization)

 max_pooling2d (MaxPooling2  (1, 13, 13, 16)           0
 D)

 conv2d_1 (Conv2D)           (1, 11, 11, 4)            580

 batch_normalization_1 (Bat  (1, 11, 11, 4)            16
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (1, 5, 5, 4)              0
 g2D)

 flatten (Flatten)           (1, 100)                  0

 dropout (Dropout)           (1, 100)                  0

 dense (Dense)               (1, 10)                   1010

=================================================================
Total params: 1830 (7.15 KB)
Trainable params: 1790 (6.99 KB)
Non-trainable params: 40 (160.00 Byte)
_________________________________________________________________
```

Training

Für das Training wurden verschiedene Callbacks implementiert. Der Callback „save_best_only“ speichert immer die Gewichte mit der höchsten Accuracy während des Trainings. Der Callback „early_stopping“ beendet das Training, wenn das Netzwerk seine Genauigkeit nicht mehr verbessert.

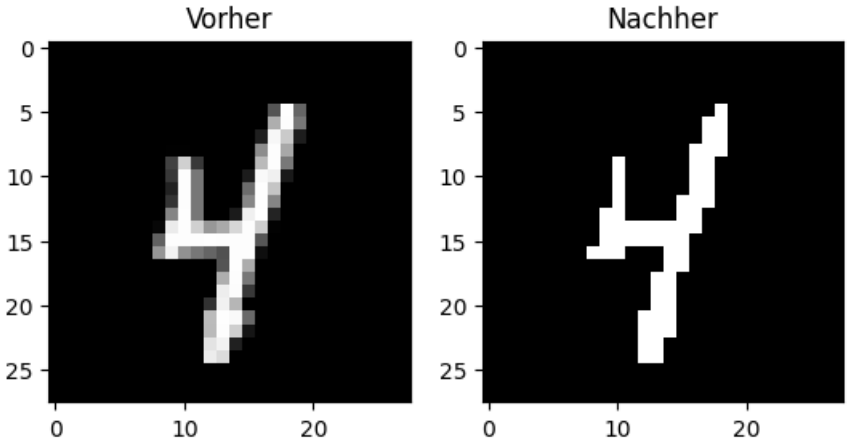
Das Training wird in 50 Epochen mit einer Batch-Size von 64 durchgeführt. Diese Batch-Size hatte sich nach verschiedenen Versuchen als die geeignetste erwiesen. Die Parameter wurden dauerhaft variiert, um die bestmögliche Kombination zu ermitteln.

Im Training wurden eine Validation Accuracy von 0.9752 erreicht. Dies ist in Anbetracht der geringen Größe und verringerten Trainingsdaten ein sehr akzeptabler Wert.

Experimente

Um die Parameter des Netzes und somit seine Größe zu reduzieren, wurde mit verschiedenen Filtermengen in den Convolutional-Layer experimentiert. Dabei wurde mit einer Kombination aus 16 Filtern in der ersten Schicht und 4 Filtern in der zweiten Schicht die beste Kombination aus Parameter-Menge und erreichbarer Accuracy erzielt.

Ein weiterer Versuch war die Verdeutlichung der Bilder durch die Skalierung der Pixel auf eindeutige Schwarz/Weiß Werte:



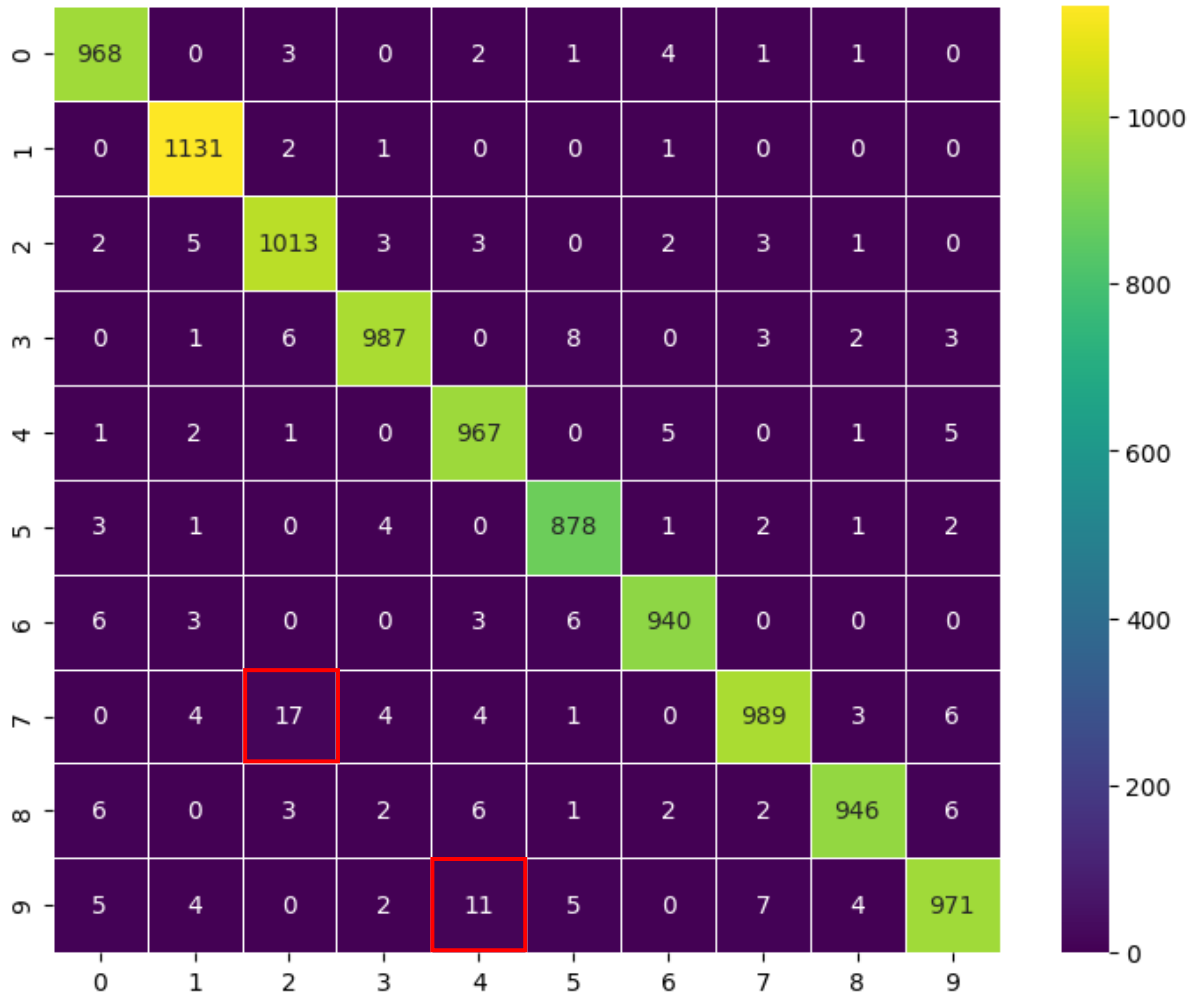
Dies erfolgte durch eine selbstdefinierte Schwellwert-Methode. Das Netzwerk wurde auf beiden Datensätzen trainiert, es konnte durch diesen Ansatz jedoch keine Verbesserung der Genauigkeit erreicht werden:

- Mit B/W: Val_Acc nach 16 Epochen/BS: 64 -> 0.9866
- Ohne B/W: Val_Acc nach 17 Epochen/BS: 64 -> 0.9907

Diskussion & Ausblick

Auf Basis der erzielten Ergebnisse können nun weitere Methoden zur Erweiterung angewendet können, das könnte zum Beispiel das sogenannte „Pruning“ sein, bei welchem zufällig Stränge des Netzwerks gekappt werden, um seine Widerstandsfähigkeit zu testen und zu verbessern. Zudem kann versucht werden, durch direkte Auswahl der Trainingsdaten eine höhere Accuracy zu erzielen.

In folgender Confusion Matrix ist zu erkennen, dass sich das Netzwerk vor allem mit der Unterscheidung der Ziffern 2 und 7 sowie 4 und 9 schwertut. Auch dieses Problem könnte durch eine Umverteilung der Trainingsdaten verbessert werden.



Weitere Quellen:

- Going beyond 99% — MNIST Handwritten Digits Recognition | by Jay Gupta | Towards Data Science
- Pruning in Deep Learning: The efficacy of pruning for model compression | by Mohan Dogra | Medium
- numpy.stack — NumPy v1.26 Manual
- Lecun Y, Bottou L, Bengio Y, et al. "Gradient-based learning applied to document recognition,". Proceedings of the IEEE, 1998, 86(11): 2278--2324.