Prüfungsleistung in „Digitale Bildverarbeitung und Mustererkennung“

Dokumentation

Tim Lucas Halt (MatrNr. 6682645)

28. Dezember 2023

1 Methodik

Die Arbeit gliedert sich in zwei Teile. Der Erste enthält Versuche mit dem MNIST-Datensatz. Der Zweite einige Experimente mit dem TinySchiller-Datensatz
Bei MNIST wird primär eine hohe maximale Genauigkeit durch automatisierte Untersuchungen angestrebt. Als Tool wird dafür Weigts&Biases [5] verwendet. Es ermöglicht automatische Sweeps, sowie Werkzeuge zur Analyse und Dokumentation. Alle Untersuchungen gehen von dem Netz in Listing 1 aus, mit dem der erste Accuracy-Benchmark gesetzt wurde. Im ersten Schritt werden die Hyperparameter optimiert. Um möglichst viele automatisiert Kombinationen zu testen, wird GPU-Rechenleistung von Kaggle und Colab verwendet. Als Zweites werden die Callback in das Training integriert, gefolgt von einer Data-Augmentation. Die endgültige Qualität wird anhand einer Versuchsreihe mit XY Trainings ermittelt.
Abschließend werden Versuche gezeigt, zur Reduktion der Label- und Parameter-Anzahl und Ergebnisse mit dem TinySchiller-Datensatz.

2 MNIST

2.1 Grundlage

Das Ausgangsnetz, von dem die Betrachtungen ausgehen ist in Listing 1 dargestellt. Es hat etwa 62-tausend Parameter. Kompiliert mit dem Adam Optimizer, einer Learning-Rate von 0.003 und einem Training mit dem vollständigen Trainingsdatensatz (60000 Label) bei einer Batch-Size von 512 über 100 Epochen erreichte es 99,47% Genauigkeit.

Listing 1: Grundlagen-Netz

```
1  InputLayer(input_shape=(28,28,1)),
2  Conv2D(filter=28, kernel_size=5, padding='same', activation='relu'),
3  MaxPooling2D(2,2),
4  Dropout(0.2),
5  Conv2D(filter=16, kernel_size=5, padding='same', activation='relu'),
6  MaxPooling2D(2,2),
7  Dropout(0.2),
8  Flatten(),
9  Dense(64, kernel_regularizer = tf.keras.regularizers.l2(0.07),
     activation = 'relu'),
10 Dropout(0.2),
11 GaussianNoise(0.1),
12 Dense(10, activation='softmax')
```

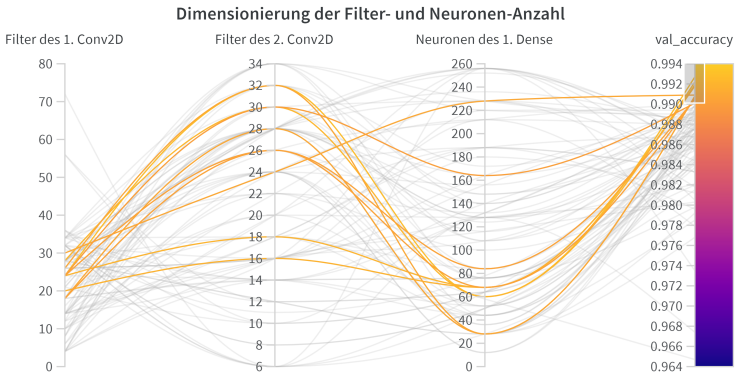

Abbildung 1: Etwa 100 Kombinationen aus Filter- und Neuronen-Anzahlen bewertet nach der val_accuracy. Dokumentiert mit Weight & Biases

2.2 Dimensionierung der Filter- und Neuronen-Anzahl

Zunächst erfolgt die Untersuchung der Filter-Anzahlen in den Convolution-Layern und der Neuronen-Anzahl des ersten Dense-Layer. Das Output-Dense-Layer wird bei 10 Neuronen und der *softmax*-Aktivierungsfunktion belassen, damit jeder Klasse eine Wahrscheinlichkeit zugeordnet wird.
Als Methode ist in Weights&Biases *bayes*, die Bayes'sche Optimierung, zur Maximierung der val_accuracy gewählt. Dabei werden die Hyperparameter aus einem vorgegebenen Raster gewählt, ein Training durchgeführt und die Ergebnisse dokumentiert. Das Raster für die Filter und Neuronen wurde so gewählt, dass (1) eine Abstufung im Convolution-Teil erfolgt, (2) auch Zahlen nicht zur Basis zwei getestet werden und (3) die Parameteranzahl nicht exorbitant groß wird. Das erste Kriterium erwies sich als kontraproduktiv und wird später verworfen.
Die Ergebnisse von circa 100 Kombinationen sind in Abbildung 1 dargestellt. Hervorgehoben sind die besten Zehn. Nach weiter eingegrenztem Sweeps sind in beiden Convolution-Layern 28 Filter gewählt und 54 Neuronen für das Dense-Layer. Den Quellcode des Testes enthält die Datei *pepsi.wandb*.

2.3 Wahl der Aktivierungsfunktionen

Für die Wahl der Aktivierungsfunktionen wurden ebenfalls Sweeps durchgeführt. Die Ergebnisse sind in Abbildung 2 abgebildet und die besten fünf hervorgehoben. Sie haben die Sigmoid-Funktion im Dense-Layer gemeinsam. Bei den Convolution-Layern

zeigte über alle Kombinationen hinweg die Relu-Funktion die stärkste Tendenz zu hoher Genauigkeit. Entsprechend wurde gewählt: Relu, Relu, Sigmoid.

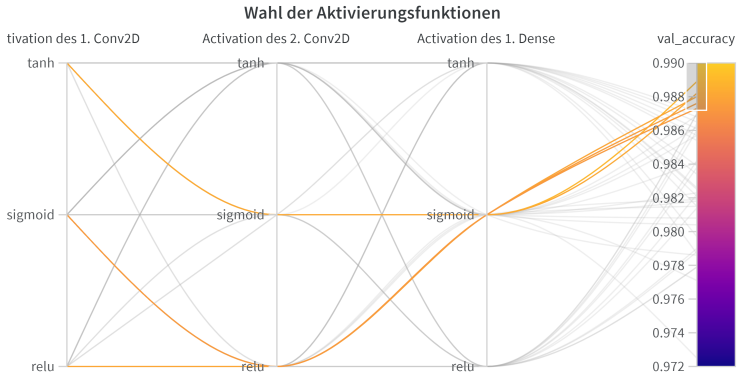

Abbildung 2: Kombinationen der Aktivierungsfunktionen bewertet nach der val_accuracy. Dokumentiert mit Weight & Biases

2.4 BatchNormalization, Dropout und GaussianNoise

Als drittes wird das Netz um BatchNormalization vor den MaxPool und Dropout erweitert, um das Model zu stabilisieren. Bei anderen Reihenfolgen kommt es zu Widersprüchen [3] Anschließend wurden die Dropout- und GaussianNoise-Werte getestet, ebenfalls mit Weigts & Biasses. Als erfolgreich erwiesen sich Dropouts von 0,4 und GaussianNoise von 0,75. [2].

2.5 Callbacks - Early Stopping und Variable Learning Rate

Nach der Optimierung der Hyperparameter des Netzes, wird der Trainingsprozess verbessert, indem er mit Callbacks angereichert wird. Zum einen wird EarlyStopping eingesetzt, um wenig erfolgreiche Trainings abzubrechen. Um darüber hinaus schneller den Grenzwert zu erreichen wird eine Variable Learning Rate verwendet. In Kombination mit dem EarlyStopping zeigte eine gleichmäßige Reduktion mit jeder Epoche bessere Ergebnisse, als die Methode ReduceLROnPlateau. In Kombination mit einer Batch Size von 32 ergaben sich die besten Ergebnisse

Listing 2: Callbacks

```
1  er = EarlyStopping(
2    monitor="val_accuracy",
```

```
    patience=10,
      restore_best_weights=True
  )
  vlr = LearningRateScheduler(lambda x: 1e-3 * 0.995 ** x)
```

2.6 Data Augmentation

Die Anzahl an Trainingsdaten wird mit einer Augmentation erweitert. Dazu kommt der ImageDataGenerator von Keras zum Einsatz. Wie er Listing 3 zu entnehmen ist. Die Bilder werden nicht gespiegelt, weil die Zahlen nicht entsprechend symmetrisch sind. Die gewählten Werte wurden ebenfalls mit Weight&Biases Sweeps getestet.

Listing 3: Data Augmentation

```

  datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range = 15,
    zoom_range = 0.15,
    shear_range = 0.1,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    rescale = 0,
    fill_mode = 'nearest',
    horizontal_flip=False,
    vertical_flip=False)
  datagen.fit(x_train)
```

2.7 Netzanpassung

Um die Genauigkeit weiter zu optimieren, wurde die Layer-Struktur nochmals angepasst. Zum einen werden anstelle eines Convolutional-Layers mit einer Filtergröße von 5 nun zwei aufeinanderfolgende mit Größe 3 verwendest. Dadurch ist das Netzwerk in der Lage sein, komplexere und nichtlineare Muster zu erlernen [4].
Des Weiteren wird anstatt MaxPooling ein Strided Convolution für das Downsampling verwendet. Dadurch ergibt sich die gleiche Reduktion der Dimensionen, aber weitere trainierbare Parameter. Außerdem wird die Filtergröße des 2. Convolution-Blocks auf 56 erhöht. Bei Versuchen wurden hiermit bessere Ergebnisse als mit den zuvor gewählten 28 erreicht.
Zusätzlich wird auch noch ein Dense-Layer mit 82 Neuronen ergänzt hinzugefügt. Für einen besseren Schutz gegen Overfitting wurde zudem nach jedem Dropout ein GaussianNoise eingefügt.

2.8 Auswertung

Die Genauigkeit erhöht sich damit zwar, aber auf Kosten von Rechenzeit. Mit diesen Änderungen wurde bei Zahl Trainingsdurchläufen durchschnittlich mit standartabweichung erzielt, Histogramm. Bei Trainings ohne EarlyStopping wurden maximal 99,77 Prozent Genauigkeit am Testdatensatz erreicht. Des finalen Netzes und der Evaluation mit Histogramm enthält das Notebook *pepsi.evaluation*.

2.9 Sonstiges: Reduktion der Label

Um die Label zu Reduzieren wurde für jede Zahl das Label im Trainingsdatensatz ausgewählt, dass die höchste Kosinus-Ähnlichkeit zu den anderen besitzt. Der Code für diese Auswahl ist in *minLabel*. Die Ergebnisse wurden allerdings nicht erfolgreich in ein Training integriert.

3 Tiny Schiller

Motiviert durch die Vorlesung zu den Large Language Model wurde neben den Versuchen an MNIST mit dem TinySchiller-Datensatz **[Schutera.2023]** „gespielt“. Ausgehend von dem Code in [1] wurde mit GRU und LSTM-Layern experimentiert. Liebevoll trägt das Modell den Namen *drunkenSchiller*. Es hat den folgenden Satz zu DeepLearning formuliert: „Deep Learningen und der alte schwert in dieser stadt geschlagen wird, der der geschichte...“Allerdings hat es eine schwäche für Geschichte und wiederholt danach in Dauerschleife die Worte „der“und „Geschichte“. Hier besteht noch Optimierungsbedarf. Der Quellcode und das Modell sind unter dem Namen *drunkenSchiller* gespeichert.

Literatur

[1] Isha Bansal. *Predict Shakespearean Text Using Keras TensorFlow - AskPython*. en-US. Nov. 2021. URL: https://www.askpython.com/python/examples/predict-shakespearean-text (besucht am 26. 12. 2023).

[2] Shaofeng Cai u. a. *Effective and Efficient Dropout for Deep Convolutional Neural Networks*. Techn. Ber. arXiv:1904.03392 [cs] type: article. arXiv, Juli 2020. DOI: 10.48550/arXiv.1904.03392. URL: http://arxiv.org/abs/1904.03392 (besucht am 26. 12. 2023).

[3] Xiang Li u. a. *Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift*. Techn. Ber. arXiv:1801.05134 [cs, stat] type: article. arXiv, Jan. 2018. DOI: 10.48550/arXiv.1801.05134. URL: http://arxiv.org/abs/1801.05134 (besucht am 26. 12. 2023).

[4] Rachid Riad u. a. *Learning strides in convolutional neural networks*. Techn. Ber. arXiv:2202.01653 [cs] type: article. arXiv, Feb. 2022. DOI: 10.48550/arXiv.2202.01653. URL: http://arxiv.org/abs/2202.01653 (besucht am 26. 12. 2023).

[5] *Tune Hyperparameters | Weights & Biases Documentation*. en. URL: https://docs.wandb.ai/guides/sweeps (besucht am 26. 12. 2023).