Prüfungsleistung im „Digitale Bildverarbeitung und Mustererkennung“

Dokumentation

Tim Lucas Halt

15. Dezember 2023

1 Methodik

Die Arbeit gliedert sich in zwei Teile. Der Erste enthält Versuche mit dem MNIST-Datensatz vor allem hinsichtlich der maximalen Genauigkeit. Der Zweite einige Experimente mit dem TinySchiller-Datensatz
Angestrebtes Ziel war es die Untersuchungen an MNIST teilautomatisiert ablaufen zu lassen. Als Tool wird dafür Weigts&Biases verwendet. Es ermöglicht automatische Sweeps (https://docs.wandb.ai/guides/sweeps .und Werkzeuge zur Analyse und Dokumentation. Als Grundlage der Untersuchungen dient das Netz in Listing 1, 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. Die Ergebnisse sind in **??** erläutert. Im zweiten Schritt werden die Callback in das Training integriert, gefolgt von einer Data-Augmentation. Danach wird die Layer-Struktur nochmal verbessert.
Anschließend werden Versuche gezeigt, die Anzahl an Labeln und Parametern zu reduzieren und die Dokumentation mit Einblicken in die Experimente TinySchiller-Datensatz abgeschlossen.

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'),
```

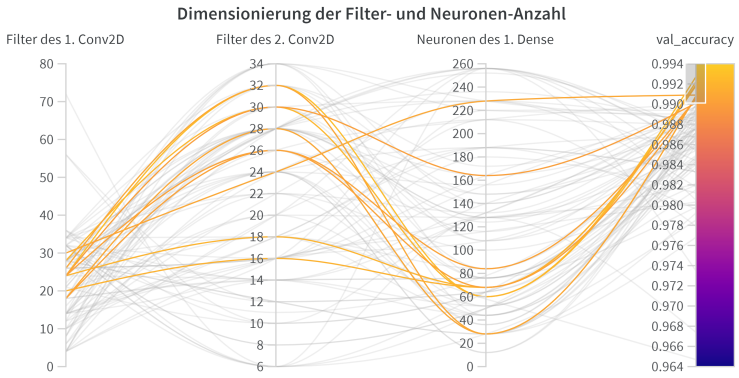

Abbildung 1: Etwa 100 Kombinationen aus Filter- und Neuronen-Anzahlen bewertet nach der val_accuracy. Dokumentiert mit Weight & Biases

```
    Dropout(0.2),
    GaussianNoise(0.1),
    Dense(10, activation='softmax')
```

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. (https://docs.wandb.ai/guides/sweeps. 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 noch verworfen. Die Ergebnisse von circa 100 Kombinationen sind in Abbildung 1 dargestellt. Hervorgehoben sind die besten Zehn. Die val_acc ist hoch, für eine Filteranzahl der Convolution-Layers um den Wert 28. Für die Neuronen des ersten Dense-Layers konzentrieren sich der erfolgreichen Kombinationen bei den kleineren Werten. Nach einem weiter eingegrenztem Sweep sind 28 Filter in beiden Convolution-Layern als gewählt und 54 Neuronen für das Dense-Layer. Das Symmetrie-Bedürfnis erkennt Ähnlichkeiten zur Bildgröße von 28x28 Pixeln und wird gleichzeitig enttäuscht, weil 54 weder eine 2er-Potenz noch ein Vielfaches von 28 ist.

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 höher Genauigkeit. Entsprechend wurde gewählt: Relu, Relu, Sigmoid.

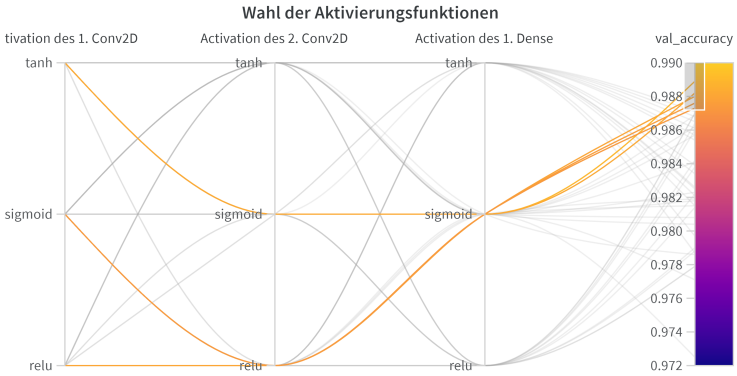

Abbildung 2: Kombinationen der Aktivierungsfunktionen bewertet nach der val_accuracy. Dokumentiert mit Weight & Biases

2.4 BatchNormalization, Dropout und GaussianNoise

Als drittes wird das Netz um BatchNormalization vor den MaxPool und Dropout erweitert, um das Model zu stabilisieren. 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. papers (dropout gege noverfitting).

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 den automatisierten Trainingsprozess zu verkürzen und wenig erfolgreiche Druchläufe hinsichtlich der val_accuracy abzubrechen. Gleichzeitig werden die besten Ergebnisse gespeichert. 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

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

2.6 Data Augmentation

Des Weiteren wird die Anzahl an Trainingsdaten mit einer Augmentation erweitert. Dazu kommt der ImageDataGenerator von Keras zum Einsatz. Für Rotation und Zoom werden 0,15 verwendet. Für Shear und Shift 0,1. Die Bilder werden nicht gespiegelt, weil die Zahlen nicht entsprechend symmetrisch sind. Die gewählten Werte haben sich bei Weight&Biases Sweeps als geeignet herausgestellt.

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

Nach den vorausgegangenen Methoden die Genauigkeit zu optimieren, folgt nun abschließend noch eine Anpassung des Netzes. Zum einen werden anstelle eines Convolutional-Layers mit einer Filtergröße von 3 nun zwei aufeinanderfolgende mit Größe 3 verwendest. Dadurch ist das Netzwerk in der Lage sein, komplexere und nichtlineare Muster

zu erlernen. https://doi.org/10.48550/arXiv.2202.01653 Des Weiteren wird anstatt MaxPooling Strided Convolution für das Downsampling verwendet. Dadurch ergibt sich die gleiche Reduktion der Dimensionen, aber weitere trainierbare Parameter. Wie zuvor bereits angedeutet, wurde zudem die Filtergröße des 2. Convolution-Blocks 56 erhöht. Bei mehreren Versuchen wurden hiermit bessere Ergebnisse als mit 28 erreicht. Zusätzlich wurde außerdem noch ein Dense-Layer mit 82 Neuronen ergänzt(sprich 28 mehr als das folgende Layer). Für einen besseren Schutz gegen Overfitting wurde außerdem nach jedem Dropout ein GaussianNoise eingefügt.
Das dadurch entstandene Netz ist dem Jupyterdateiname zu entnehmen

Mit diesen Änderungen wurde bei 50 Netzen... maximal 99.77 und damit der benchmark

2.8 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 cosinussquere.jupyterdatei.

3 Tiny Schiller

tiny schiller referenz Gandalf in Moria: „Im Zweifelsfall sollte man immer seiner Nase folgen . . .“

Motiviert durch die Vorlesung zu den Large Language Model wurde neben den Versuchen an MNIST mit dem TinySchiller-Datensatz „gespielt“. Ausgehend von dem Code https://www.askpython.com/python/examples/predict-shakespearean-text. Wurde mit gru und lstm andere Layertypenversucht. Liebevoll trägt das Modell den Namen *drunkenSchiller* und ihm steht die Ehre zu einen abschließenden Satz über DeepLearning zu teilen: