Dokumentation zum Programmentwurf

Digitale Bildverarbeitung und Mustererkennung

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Kaan Aydin

Abgabedatum:	05. Januar 2024
Bearbeitungszeitraum:	01.10.2023 - 05.01.2024
Matrikelnummer:	9752327
Kurs:	TFE21-2
Ausbildungsfirma:	ZF Friedrichshafen AG
Dozent des Kurses:	Dr.-Ing. Mark Schutera
Studiengangsleiter der Dualen Hochschule:	Prof. Dr.-Ing. Thomas Kibler

Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 25.07.2018.

Ich versichere hiermit, dass ich meinen Programmentwurf mit dem Thema:

Dokumentation zum Programmentwurf

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Friedrichshafen, den 4. Januar 2024

Kaan Aydin

Um die Genauigkeit eines neuronalen Netzes zu steigern, stehen eine Vielzahl von Methoden zur Verfügung, die mit Hilfe von Experimenten und Überlegungen durchgeführt werden können. Dieses neuronale Netzwerkmodell wird trainiert und für Bildklassifizierungsaufgaben verwendet, um beispielsweise handgeschriebenen Ziffern zu erkennen. Erhöhung der Anzahl an Trainingsdaten, Verbesserung der Funktionsauswahl, Entfernen von fehlerhaften Daten aus der Datenverarbeitung oder einfaches Ausprobieren sollten dazu beitragen können, die Leistung des neuronalen Netzes zu verbessern und eine präzisere Modellierung zu erreichen.

Zu Beginn sollte die Architektur des neuronalen Netzwerkes aufgebaut werden. Der bereitgestellte Code in Abbildung 2 definiert ein Convolutional Neural Network (CNN), das mit der Keras-API von TensorFlow erstellt wurde und für die Bildklassifizierung aus dem MNIST-Datensatz genutzt wird. Es erstellt ein Modell namens 'marvin', das aus mehreren Schichten besteht. Die Funktion 'create_model' definiert die Architektur des neuronalen Netzwerkmodells. Es verwendet 'Sequential', um ein Modell zu definieren. Dieses Modell umfasst Faltungsschichten (engl. convolution layers), vollständig verbundene Schichten (engl. fully connected layers), welche die flatten und dense layers beinhaltet. Die Dropout-Schichten sind zur Regularisierung und die Batch-Normalisierungsschichten (engl. batch normalization layers) für eine bessere Konvergenz. Die Ausgabeschicht des Modells (engl. output layer) verwendet eine 'Softmax'-Aktivierungsfunktion, die für Klassifizierungsprobleme mit mehreren Klassen geeignet ist [**1**; **2**].

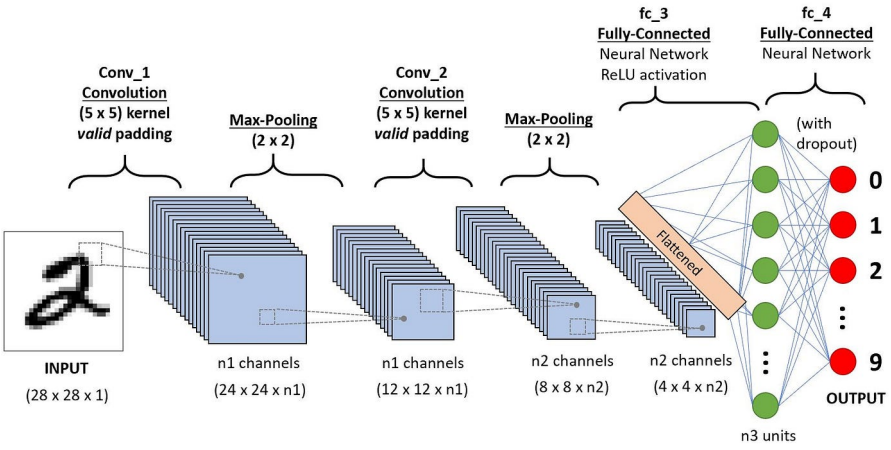

Abbildung 1: Darstellung der verschiedenen Ebenen [**3**].

```python
def create_model(input_shape=(28,28,1)):
    model = Sequential([
        # Convolution layers
        Conv2D(filters=64, kernel_size=5, padding='same', activation='relu', input_shape=input_shape),
        MaxPooling2D(2,2),
        Dropout(0.3),
        BatchNormalization(),
        Conv2D(filters=64, kernel_size=5, padding='same', activation='relu'),
        MaxPooling2D(2,2),
        Dropout(0.3),
        BatchNormalization(),
        # Fully connected layers
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.3),
        BatchNormalization(),
        Dense(32, kernel_regularizer = tf.keras.regularizers.l2(0.07), activation='relu'),
        Dropout(0.3),
        BatchNormalization(),
        GaussianNoise(0.1),
        # Output Layer
        Dense(10, activation='softmax')
    ])

    return model

marvin = create_model()
```

Abbildung 2: Darstellung der Architektur des Netzwerksmodells.

Das TensorFlow Keras 'models.Sequential' wird zum Erstellen eines Modells mit einer Abfolge von Ebenen verwendet. 'layers.InputLayer' definiert die Eingabeform für das Modell, in diesem Fall (28,28,1). Der MNIST-Datensatz enthält Bilder mit jeweils 28 x 28 Pixeln. Die 'Conv2D'-Ebene ist eine Faltungsebene, die 64 Faltungsfilter der Größe 5x5 auf die Eingabebilder anwendet. Das Argument 'padding='same" stellt sicher, dass die Ausgabebilder dieselbe Größe wie die Eingabebilder haben, was dabei hilft, räumliche Informationen während der Faltung beizubehalten. Das Argument activate='relu' wendet die Aktivierungsfunktion Rectified Linear Unit (ReLU) auf die Ausgabe der Faltung an. MaxPooling2D wendet maximales Pooling auf die Ausgabe der vorherigen Ebene an, wodurch die räumlichen Abmessungen der Ausgabe reduziert werden, während nur die Maximalwerte beibehalten werden. Die Dropout-Schicht wird verwendet, um zufällig ausgewählte Neuronen während des Trainings auszuschließen, bzw. zu ignorieren, was dazu beiträgt, eine Überanpassung zu verhindern. In diesem Fall fallen bei jeder Trainingsiteration 30% der Neuronen aus. Die Ebene 'BatchNormalization' normalisiert die Eingabe bei jedem Batch, indem sie die Aktivierungen anpasst und skaliert. Dies kann dazu beitragen, den Trainingsprozess zu beschleunigen und die Leistung des Modells zu verbessern. Anhand dieser ganzen Schichten kann eine bessere Leistung des Netzmodells erreicht werden [**4**].

Der Code ist so strukturiert, dass zunächst eine Vorhersage über die Daten-Durchläufe getroffen wird, dann die Ausführung erfolgt und schließlich eine Auswertung zurückgeben wird.
Um den Ausgang eines Ereignisses oder einer Situation vorherzusagen, wird hierbei eine Vorhersage durchgeführt. Diese sind wichtige Bestandteile des maschinellen Lernens, da sie es dem Modell ermöglichen, das Gelernte zu verallgemeinern und auf neue Daten anzuwenden. Eine Vorhersage sagt also nicht über die absolute Sicherheit eines Ereignisses aus, sondern liefert vielmehr eine Wahrscheinlichkeitsverteilung oder Schätzung möglicher Ereignisse. In dieser Anwendung wird eine binäre Vorhersage, mit Hilfe der Funktion 'prediction.round()', verwendet. Dies geschieht, um die Klasse mit der höchsten Wahrscheinlichkeit zu erhalten **[5]**.
Bei Betrachtung des Modells lässt sich die Genauigkeit durch eine Erhöhung der Anzahl der Epochen und eine Verringerung des sogenannten 'batch size' steigern. Dies führt zu einer erhöhten Anzahl von Daten-Durchläufen, was wiederum zu einer verbesserten Anpassung der Daten führt. In dem Codeblock der Trainingsparameter sind folgende Funktionen beschrieben:

- logdir: Dies ist das Verzeichnis, in das die TensorBoard-Dateien geschrieben werden.

- Epochen: Die Häufigkeit, mit der der gesamte Datensatz vorwärts und rückwärts durch den Trainingsalgorithmus geleitet wird.

- batch_size: Die Anzahl der ausgewerteten Trainingsbeispiele.

- shuffle: Ein boolescher Wert, der angibt, ob die Trainingsdaten nach jeder Epoche gemischt werden sollen.

- validation_data: Dieser Parameter wird verwendet, um den Verlust und alle Modellmetriken am Ende jeder Epoche auszuwerten.

- Rückrufe: Dieser Parameter wird hierbei nicht verwendet, ist aber um Rückrufe für den Modelltrainingsprozess anzugeben.

Als Auswertung ergeben sich die Abbildungen 3 und 4.
Abbildung 3 zeigt die Trainings- und Validierungsgenauigkeit. Hierbei stellt die x-Achse die Epochen dar und die y-Achse die Genauigkeit. Die Genauigkeit des Modells wird sowohl für den Trainingssatz als auch für den Validierungssatz aufgetragen. Wie der Grafik zu entnehmen ist, nimmt die Trainingsgenauigkeit allmählich zu, da das Modell mehr aus den Trainingsdaten lernt. Das bedeutet, dass sich das Modell verbessert und bessere Vorhersagen auf der Grundlage bisher unbekannter Daten treffen kann. Es ist jedoch wichtig zu beachten, dass die Validierungsgenauigkeit nicht im gleichen Maße zunimmt wie die Trainingsgenauigkeit. Abbildung 4 zeigt die Verlustkurven sowohl für die Trainings- als auch für die Validierungssätze über die Trainingsepochen. Im Trainingssatz lernt das Modell aus den gegebenen Daten, einschließlich der Beschriftungen. Wenn das Modell seinen Lernprozess verbessert, verringert sich daher der Trainingsverlust [**6**].

Abbildung 3: Darstellung der Trainings- und Validierungsgenauigkeit.

Abbildung 4: Darstellung des Trainings- und Validierungsverlustes.

Literatur

(1) PySource. “Flatten and Dense layers“. Abgerufen am 30.12.2023 von https://pysource.com/2022/10/07/flatten-and-dense-layers-computer-vision-with-keras-p-6/.

(2) EMaster Class Academy. “Deep Learning in TesnorFlow 4 L5 - CNN“. Abgerufen am 01.01.2024 von https://www.youtube.com/watch?v=UPljwpNKMOM.

(3) Towardsdatascience. “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way“. Abgerufen am 01.01.2024 von https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

(4) Keras.io. “Keras layers API“. Abgerufen am 02.01.2024 von https://keras.io/api/layers/.

(5) RealPython. “How to build a neural network and make predictions“. Abgerufen am 03.01.2024 von https://realpython.com/python-ai-neural-network/.

(6) Atmosera. “Binary Classification with Neural Networks“. Abgerufen am 03.01.2024 von https://www.atmosera.com/blog/binary-classification-with-neural-networks/.

Anhang

```
1875/1875 [==============================] - 50s 26ms/step - loss: 0.6002 - accuracy: 0.9014 - val_loss: 0.1047 - val_accuracy: 0.9797
Epoch 2/25
1875/1875 [==============================] - 50s 27ms/step - loss: 0.1891 - accuracy: 0.9605 - val_loss: 0.0828 - val_accuracy: 0.9867
Epoch 3/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.1642 - accuracy: 0.9657 - val_loss: 0.0672 - val_accuracy: 0.9915
Epoch 4/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.1499 - accuracy: 0.9711 - val_loss: 0.0696 - val_accuracy: 0.9905
Epoch 5/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.1463 - accuracy: 0.9716 - val_loss: 0.0613 - val_accuracy: 0.9937
Epoch 6/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.1336 - accuracy: 0.9741 - val_loss: 0.0646 - val_accuracy: 0.9924
Epoch 7/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.1251 - accuracy: 0.9762 - val_loss: 0.0593 - val_accuracy: 0.9939
Epoch 8/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.1240 - accuracy: 0.9772 - val_loss: 0.0655 - val_accuracy: 0.9919
Epoch 9/25
1875/1875 [==============================] - 52s 28ms/step - loss: 0.1216 - accuracy: 0.9782 - val_loss: 0.0605 - val_accuracy: 0.9933
Epoch 10/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.1164 - accuracy: 0.9787 - val_loss: 0.0628 - val_accuracy: 0.9928
Epoch 11/25
1875/1875 [==============================] - 50s 27ms/step - loss: 0.1203 - accuracy: 0.9783 - val_loss: 0.0572 - val_accuracy: 0.9941
Epoch 12/25
1875/1875 [==============================] - 50s 27ms/step - loss: 0.1142 - accuracy: 0.9801 - val_loss: 0.0525 - val_accuracy: 0.9946
Epoch 13/25
1875/1875 [==============================] - 52s 28ms/step - loss: 0.1083 - accuracy: 0.9808 - val_loss: 0.0570 - val_accuracy: 0.9942
Epoch 14/25
1875/1875 [==============================] - 52s 28ms/step - loss: 0.1067 - accuracy: 0.9811 - val_loss: 0.0591 - val_accuracy: 0.9934
Epoch 15/25
1875/1875 [==============================] - 54s 29ms/step - loss: 0.1036 - accuracy: 0.9819 - val_loss: 0.0584 - val_accuracy: 0.9944
Epoch 16/25
1875/1875 [==============================] - 53s 28ms/step - loss: 0.1028 - accuracy: 0.9822 - val_loss: 0.0657 - val_accuracy: 0.9917
Epoch 17/25
1875/1875 [==============================] - 52s 28ms/step - loss: 0.1043 - accuracy: 0.9823 - val_loss: 0.0585 - val_accuracy: 0.9941
Epoch 18/25
1875/1875 [==============================] - 52s 28ms/step - loss: 0.1067 - accuracy: 0.9812 - val_loss: 0.0581 - val_accuracy: 0.9942
Epoch 19/25
1875/1875 [==============================] - 53s 28ms/step - loss: 0.0958 - accuracy: 0.9831 - val_loss: 0.0560 - val_accuracy: 0.9946
Epoch 20/25
1875/1875 [==============================] - 55s 29ms/step - loss: 0.0971 - accuracy: 0.9840 - val_loss: 0.0540 - val_accuracy: 0.9938
Epoch 21/25
1875/1875 [==============================] - 53s 28ms/step - loss: 0.0976 - accuracy: 0.9839 - val_loss: 0.0577 - val_accuracy: 0.9941
Epoch 22/25
1875/1875 [==============================] - 55s 29ms/step - loss: 0.0987 - accuracy: 0.9836 - val_loss: 0.0560 - val_accuracy: 0.9948
Epoch 23/25
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0942 - accuracy: 0.9846 - val_loss: 0.0570 - val_accuracy: 0.9938
Epoch 24/25
1875/1875 [==============================] - 54s 29ms/step - loss: 0.0980 - accuracy: 0.9838 - val_loss: 0.0537 - val_accuracy: 0.9940
Epoch 25/25
1875/1875 [==============================] - 52s 28ms/step - loss: 0.0907 - accuracy: 0.9848 - val_loss: 0.0520 - val_accuracy: 0.9953
```

Abbildung 5: Darstellung der Genauigkeit und des Verlustes des Netzes.

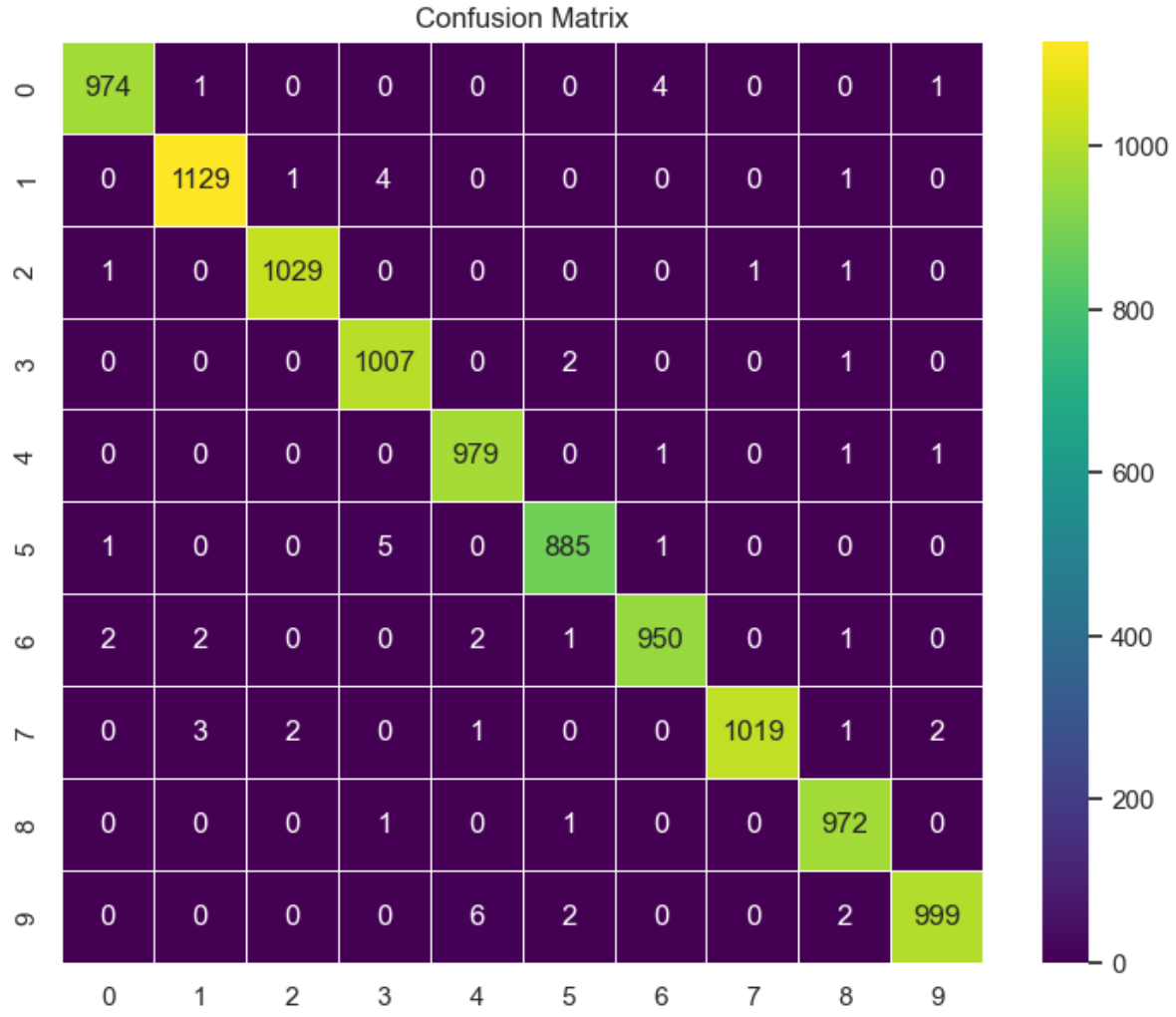

Abbildung 6: Darstellung der 'Heatmap'.