20220614_Wirtschaftsinformatik_FAT2

June 14, 2022

[1]:
```
!pip install keras
```

```
Requirement already satisfied: keras in
/Users/h4/opt/anaconda3/lib/python3.8/site-packages (2.6.0)
```

[3]:
```
import keras
from keras.datasets import mnist
mnist.load_data()
```

[3]:
```
((array([[[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         [[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         [[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         …,

         [[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
```

```
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         [[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         [[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]]], dtype=uint8),
 array([5, 0, 4, …, 5, 6, 8], dtype=uint8)),
(array([[[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         [[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         [[0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          …,
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0],
          [0, 0, 0, …, 0, 0, 0]],

         …,
```

```
          [[0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           …,
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0]],

          [[0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           …,
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0]],

          [[0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           …,
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0]]], dtype=uint8),
  array([7, 2, 1, …, 4, 5, 6], dtype=uint8)))
```

[4]:
```
# imports von packages

import numpy as np
import matplotlib
import matplotlib.pyplot as plt

from keras.models import Sequential, load_model
from keras.layers.core import Dense, Activation
from keras.utils import np_utils
```

[5]:
```
# SPLIT Funktion

(X_train, y_train), (X_test, y_test)=mnist.load_data()
```

[8]:
```
# Umwandlung der Daten

X_train =X_train.reshape(60000, 784) # die 60000 Bilder 28x28 vom Train werden␣
 ↪umgewandelt in 60000 Bilder 1x784
X_test=X_test.reshape(10000, 784) # bei Test gibt es 10000 28x28 Bilder

X_train=X_train.astype('float32')
X_test=X_test.astype('float32')
```

```
# Normalisierung (Graustufen sind zw 0 und 255). Also wir wollen die Daten zw 0␣
↪und 1 haben. Deshalb teilen wir durch 255.

X_train /=255
X_test /=255
```

[9]:
```
# nicht Prüfungsrelevant

# Die zehn Kategorien werden umgewandelt in Zahlenkategorien.
# Dies macht man mit "One-hot Encoding"

n_classes = 10

Y_train = np_utils.to_categorical(y_train, n_classes)
Y_test = np_utils.to_categorical(y_test, n_classes)
```

[10]:
```
# Deep Learning

model = Sequential()

model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

[11]:
```
# nicht prüfungsrelevant
# Compilation vom Model

model.compile(loss='categorical_crossentropy', metrics=['accuracy'],␣
↪optimizer='adam')
```

[12]:
```
# training vom model

history = model.fit(X_train, Y_train, batch_size=128, epochs=20,␣
↪validation_data=(X_test, Y_test))
```

```
Epoch 1/20
469/469 [==============================] - 3s 5ms/step - loss: 0.8066 -
accuracy: 0.7606 - val_loss: 0.3843 - val_accuracy: 0.8913
Epoch 2/20
469/469 [==============================] - 2s 5ms/step - loss: 0.3525 -
accuracy: 0.8969 - val_loss: 0.3100 - val_accuracy: 0.9109
Epoch 3/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2893 -
```

```
accuracy: 0.9165 - val_loss: 0.2645 - val_accuracy: 0.9215
Epoch 4/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2491 -
accuracy: 0.9262 - val_loss: 0.2371 - val_accuracy: 0.9286
Epoch 5/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2128 -
accuracy: 0.9375 - val_loss: 0.1931 - val_accuracy: 0.9424
Epoch 6/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1818 -
accuracy: 0.9466 - val_loss: 0.1776 - val_accuracy: 0.9454
Epoch 7/20
469/469 [==============================] - 3s 5ms/step - loss: 0.1581 -
accuracy: 0.9530 - val_loss: 0.1536 - val_accuracy: 0.9529
Epoch 8/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1371 -
accuracy: 0.9594 - val_loss: 0.1355 - val_accuracy: 0.9584
Epoch 9/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1201 -
accuracy: 0.9639 - val_loss: 0.1249 - val_accuracy: 0.9602
Epoch 10/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1062 -
accuracy: 0.9685 - val_loss: 0.1167 - val_accuracy: 0.9648
Epoch 11/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0943 -
accuracy: 0.9718 - val_loss: 0.1064 - val_accuracy: 0.9664
Epoch 12/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0845 -
accuracy: 0.9744 - val_loss: 0.0978 - val_accuracy: 0.9684
Epoch 13/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0768 -
accuracy: 0.9772 - val_loss: 0.0928 - val_accuracy: 0.9707
Epoch 14/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0683 -
accuracy: 0.9794 - val_loss: 0.0908 - val_accuracy: 0.9722
Epoch 15/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0624 -
accuracy: 0.9810 - val_loss: 0.0857 - val_accuracy: 0.9738
Epoch 16/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0558 -
accuracy: 0.9835 - val_loss: 0.0847 - val_accuracy: 0.9743
Epoch 17/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0512 -
accuracy: 0.9847 - val_loss: 0.0769 - val_accuracy: 0.9760
Epoch 18/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0453 -
accuracy: 0.9865 - val_loss: 0.0814 - val_accuracy: 0.9742
Epoch 19/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0426 -
```

```
accuracy: 0.9871 - val_loss: 0.0812 - val_accuracy: 0.9745
Epoch 20/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0373 -
accuracy: 0.9888 - val_loss: 0.0716 - val_accuracy: 0.9780
```

[ ]: