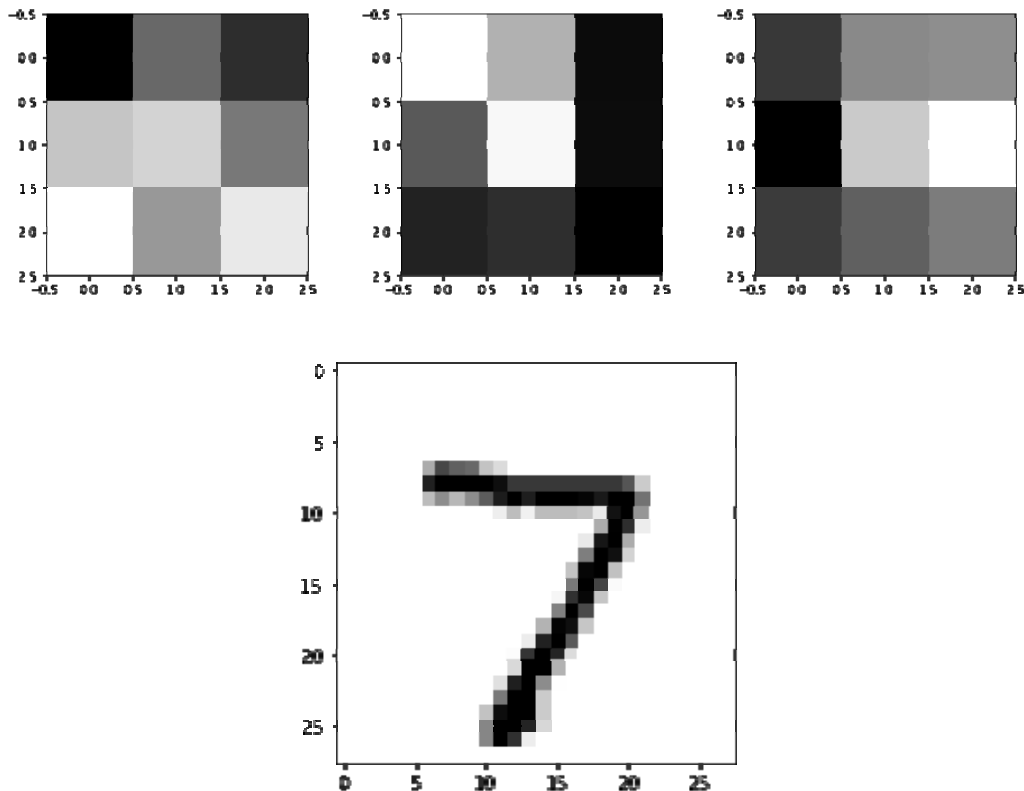
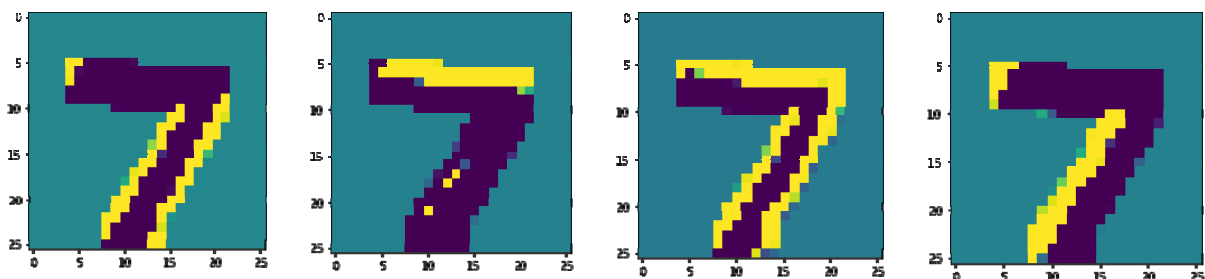


Convolutional Neuronal Networks 1

Grundsätzlich ist es möglich, mit einem Neuronalen Netz (NN), wie wir es in den letzten Übungen kennen gelernt habe, vorherzusagen, was auf einem Bild zu sehen ist. Der vorgestellte Ansatz hat aber (auch praktische) Grenzen, die im Wesentlichen darauf zurückzuführen sind, dass das NN kein allgemein gültiges „Verständnis“, wie ein Bild aufgebaut ist, hat. Genau das ist die Idee bei sog. **Convolutional Neural Networks** (CNN). Dieses entwickelt ein Verständnis über den Aufbau von Bildern. Genau genommen werden hierbei lokale Muster gelernt. Das sind kleine zweidimensionale Fenster (z.B. 3x3 Pixel).



Nachfolgende Abbildungen illustrieren 4 Beispiele, anhand welcher Kanten das CNN erkennt, dass es sich hierbei um die Ziffer ‚7‘ handelt.



CNNs sind das Maß der Dinge, wenn es um die Bildverarbeitung/-klassifizierung (= maschinelles Sehen) geht.

Grundlage für nachfolgende Übung bildet das Keras-MNIST-Dataset. Wie die Daten aufzubereiten sind, sollte soweit einmal klar sein; wobei es bei der Shape einen kleinen aber wichtige Erweiterung gibt. Aber der Reihe nach!

Damit aus einem NN ein CNN wird, braucht es eigentlich nur einen einzigen neuen Layer, den Conv2D-Layer¹.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten
from tensorflow.keras.utils import to_categorical ←


model = Sequential()
model.add(Conv2D(10, kernel_size=(3,3), activation="relu",
input_shape=(28,28,1)))
model.add(Flatten())
model.add(Dense(10, activation="softmax"))


model.compile(optimizer="rmsprop", loss="categorical_crossentropy",
metrics=["accuracy"])

model.fit(train_images, train_labels, epochs=10, batch_size=1000)
```


Datenaufbereitung

Wie der Definition der Input-Shape zu entnehmen ist, braucht es die Shape (28, 28, **1**). Zurückzuführen ist das auf die Anforderungen, die der Conv2D-Layer mit sich bringt. Um die Bilddaten als solches verarbeiten zu können, braucht es die entsprechende Form einer Abbildung (x-Achse bzw. y-Achse sowie Information über den Farbkanal). Nachdem im Beispielfall Graustufenbilder gegeben sind, reicht **1** Kanal.

 20.1.1 Welche Shape wäre bei RGB-Bildern vonnöten?

 20.1.2 Implementieren und trainieren Sie das CNN.

Flatten, wozu?

 20.1.3 Warum braucht es den Flatten-Layer? Transparenz liefert das Attribut `output_shape` der Klasse `Sequential`. Dessen Anwendung siehe Klasse `Flatten`.

Filter visualisieren

¹ https://keras.io/api/layers/convolution_layers/convolution2d/

Einleitend (siehe oben) sind drei beispielhafte Filter, die der `Conv2D`-Layer gelernt hat, dargestellt. Um diese zu visualisieren, muss man wissen, dass die `model`-Variable (vom Typ `Sequential`) die entsprechenden Informationen bereitstellt.

```
model.layers
```

Output:

```
[<tensorflow.python.keras.layers.convolutional.Conv2D at 0x1a1db155978>,  
<tensorflow.python.keras.layers.core.Flatten at 0x1a1db1c9828>,  
<tensorflow.python.keras.layers.core.Dense at 0x1a1db1c96d8>]
```

Über `layers` kann man auf sämtliche Layers des Modells zugreifen. Uns interessiert der erste, der `Conv2D`-Layer. Nachdem es sich um eine Liste handelt, sollte die Art und Weise der Selektion klar sein:

```
model.layers[0].weights
```

Output:

```
[<tf.Variable 'conv2d_8/kernel:0' shape=(3, 3, 1, 10) dtype=float32, numpy=  
array([[[[-0.09856407,  0.1903866 , -0.05910416, -0.10853323,  
          -0.20164576, -0.08738692, -0.15775552,  0.0388436 ,  
           0.10379335,  0.15335637]],  
        [[ 0.20154625,  0.08066699, -0.02593814, -0.19852525,  
          0.13874826, -0.13848028,  0.10793245,  0.05125847,  
          -0.04102194,  0.1893729 ]],  
        [[ 0.04165038,  0.1503039 , -0.2032386 , -0.03458954,  
          0.01996356, -0.13969208, -0.13842005,  0.18356541,  
          -0.0280593 , -0.1511026 ]]],  
...  
0 (das je-  
weils 0.  
Pixel)  
1 (das  
jeweils 1.  
Pixel)  
2
```

Die Filter, die Layer gelernt hat, sind über `weights` einsehbar. Hierbei handelt es sich um `3x3` Numpy-Arrays mit jeweils 10 Einträgen. Zurückzuführen ist das einerseits auf die Anzahl der definierten Filter (vgl. Argument `filters`) sowie die Kernel-Size (vgl. Argument `kernel_size`) beim Instanzieren des der `Conv2D`-Layers. Daraus ergeben sich 9 Pixel umfassende Filter. Diese jeweils 9 Werte sind wie folgt enthalten: Das Pixel 0 vom Filter 0 ist an Position 0 (siehe gelbe Markierung) der 3. Dimension (siehe ID „0“) enthalten. Das Pixel 0 vom Filter 1 ist an Position 1 (siehe grüne Markierung) der 3. Dimension (siehe ID „0“) enthalten.

Gespeichert sind die Werte in einer Variabel des Typs `tf.Variable`. Der Zugriff ist zugegebenermaßen etwas umständlich:

```
import tensorflow.keras.backend as K  
data = K.eval(model.layers[0].weights[0])  
data
```

Output:

```
array([[[[-0.09856407,  0.1903866 , -0.05910416, -0.10853323,  
          -0.20164576, -0.08738692, -0.15775552,  0.0388436 ,  
           0.10379335,  0.15335637]]],
```

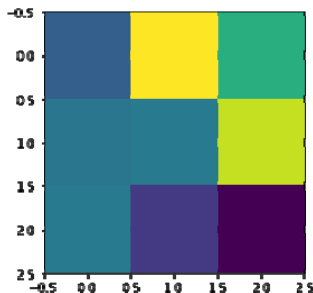
```
[[ 0.20154625,  0.08066699, -0.02593814, -0.19852525,
   0.13874826, -0.13848028,  0.10793245,  0.05125847,
  -0.04102194,  0.1893729  ]],

[[ 0.04165038,  0.1503039 , -0.2032386 , -0.03458954,
   0.01996356, -0.13969208, -0.13842005,  0.18356541,
  -0.0280593 , -0.1511026  ]]],
...
```

Nachdem der Aufbau der Datenstruktur mit den gelernten Filtern klar ist, kann man diese ratzfatz visualisieren:

```
plt.imshow(data[?].reshape(3,3))
plt.show()
```

Output:



Na ja, so einfach ist das dann doch wieder nicht! Denn wenn man die Shape von `data` ausgibt, wird einem leicht schwindelig:


```
data.shape
```

Output:

```
(3, 3, 1, 10)
```



Wir wissen, dass alle an der Stelle 0, der Stelle 1 usw. zusammengehören und somit einen Filter bilden. Es ist somit ausschließlich die letzte Achse, die uns interessiert. Und damit man zu dieser gelangt, muss man alle der 1., dann alle der 2. und alle der 3. auswählen. Dann kann man mit den IDs von 0 bis 9 das gewünschte Filter selektieren.

 20.1.4 Visualisieren Sie die 10 gelernten Filter.