Group assignment 1: Image recognition using deep networks

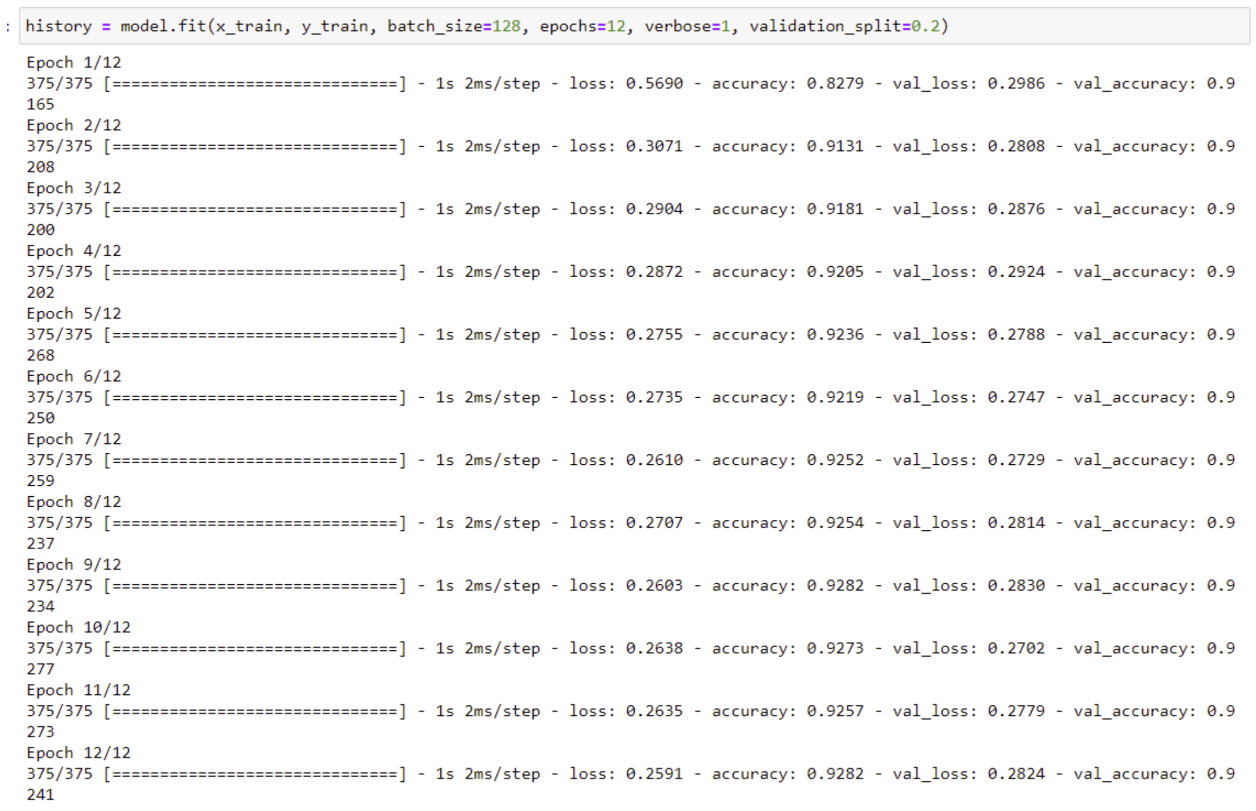
Date: 05-03-2021
Authors: Eva Steenhoven (9969632), Rosanne Vreugdenhil (4163869), Robin Reijers (5069769), and Mirthe Hendriks (6866999)

Exercise 1: identifying handwritten numbers

Discuss with your group, then describe to your teacher, a list of applications where automatic recognition of hand-written numbers would be useful. (Question 1, 3 points)

- Grading numerical tests, for example highschool math tests
- Sorting post by automatic postal code recognition
- Digitalizing primary sources for economical history like censuses or other organizational records
- Automatic reading of bank records
- Reading a doctor's note with a handwritten number for medication

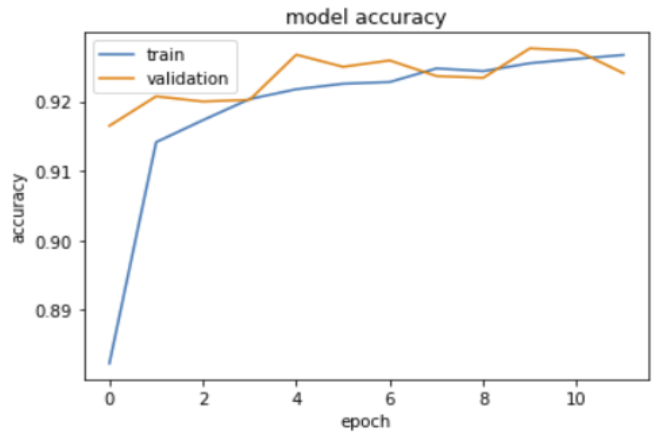
Show your teacher the text from your console, with how long it took for each epoch to run and the training performance history. (Question 2, 5 points)

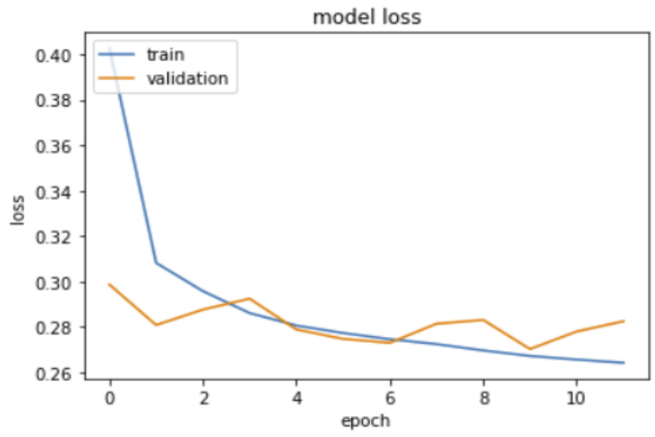


```
history = model.fit(x_train, y_train, batch_size=128, epochs=12, verbose=1, validation_split=0.2)
```

```
Epoch 1/12
375/375 [==============================] - 1s 2ms/step - loss: 0.5690 - accuracy: 0.8279 - val_loss: 0.2986 - val_accuracy: 0.9
165
Epoch 2/12
375/375 [==============================] - 1s 2ms/step - loss: 0.3071 - accuracy: 0.9131 - val_loss: 0.2808 - val_accuracy: 0.9
208
Epoch 3/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2904 - accuracy: 0.9181 - val_loss: 0.2876 - val_accuracy: 0.9
200
Epoch 4/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2872 - accuracy: 0.9205 - val_loss: 0.2924 - val_accuracy: 0.9
202
Epoch 5/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2755 - accuracy: 0.9236 - val_loss: 0.2788 - val_accuracy: 0.9
268
Epoch 6/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2735 - accuracy: 0.9219 - val_loss: 0.2747 - val_accuracy: 0.9
250
Epoch 7/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2610 - accuracy: 0.9252 - val_loss: 0.2729 - val_accuracy: 0.9
259
Epoch 8/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2707 - accuracy: 0.9254 - val_loss: 0.2814 - val_accuracy: 0.9
237
Epoch 9/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2603 - accuracy: 0.9282 - val_loss: 0.2830 - val_accuracy: 0.9
234
Epoch 10/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2638 - accuracy: 0.9273 - val_loss: 0.2702 - val_accuracy: 0.9
277
Epoch 11/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2635 - accuracy: 0.9257 - val_loss: 0.2779 - val_accuracy: 0.9
273
Epoch 12/12
375/375 [==============================] - 1s 2ms/step - loss: 0.2591 - accuracy: 0.9282 - val_loss: 0.2824 - val_accuracy: 0.9
241
```

Plot the training history and show this to your teacher (Question 3, 3 points)





Discuss with your group, then describe to your teacher, how the accuracy on the training and validation sets progress differently across epochs, and what this tells us about the generalisation of the model. (Question 4, 5 points).

The more the model trains - with increasing epochs - the higher the accuracy score. This is what we see happen for the train dataset; the accuracy score increases from 0.83 to 0.93. The accuracy of the validation dataset is relatively stable, it fluctuates around the 0.92. An accuracy score of 1 indicates perfect prediction. We can conclude that the accuracy score is good. The accuracy score for the validation dataset shows us that the model can be generalised to other datasets (other than the training dataset) and still predict the handwritten numbers accurately.
Robin: Also the training time reduces with each epoch, starting at 3ms for the first epoch and ending at 1ms for the latest epoch (while still retaining the same amount of images in each epoch)

Evaluate the model performance on the test set using the following command: loss, accuracy = model.evaluate(x_test, y_test, verbose=0) Show your teacher what values you get for the model's accuracy and loss. (Question 5, 2 points)

```
loss  0.2943028211593628
accuracy  0.9200000166893005
```

Discuss with your group, then describe to your teacher, whether this accuracy is sufficient for some uses of automatic hand-written digit classification. (Question 6, 5 points)

Whether the accuracy is sufficient for uses of automatic hand-written digit classification is dependent on the classification task in question. An accuracy of 0.92 would be sufficient for sorting post by automatic postal code recognition, with the amount of post that needs the be sorted automated classification would be beneficial. If number classification is inaccurate a letter might not be sorted correctly, but there is still a human check in the delivery of the post.

For the digitalisation of historical sources this accuracy would also be sufficient as the source would still be critically analyzed and mistakes could be recognized. However, this accuracy score would be a problem for the grading of numerical (math) tests. The consequence would be that some written numbers are not classified correctly and students will receive a wrong grade; as such a small inaccuracy score would still be problematic and it might be easier for teachers to grade tests themselves.
Other cases where an accuracy of 0.92 would not be adequate is for medication and bank records. The consequences of a faulty prediction would be too severe. If a doctors' note containing a handwritten number for a certain type of medication is misclassified by the algorithm, and as a result a patient receives too much or too little medication this would be severely problematic. Similarly, the consequences of mistakes in classification of numbers in bank records would be severely problematic too. For these tasks, the accuracy is too low.

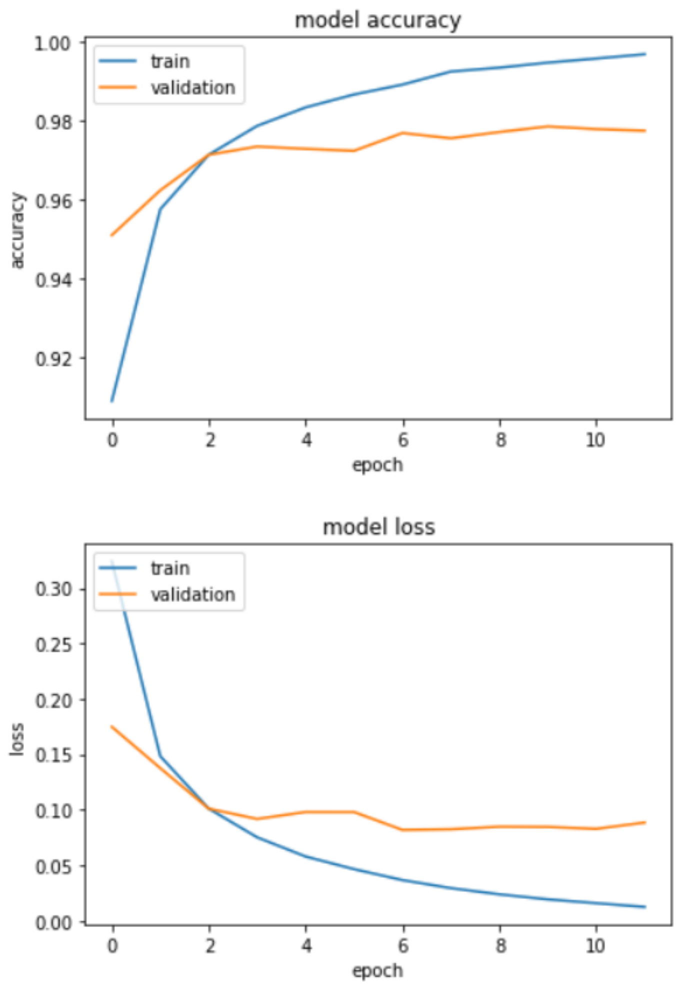
Discuss with your group, then describe to your teacher, how linear activation of units limits the possible computations this model can perform. (Question 7, 5 points)

Since the model uses linear activation of units, it is not possible to use backpropagation (gradient descent, see below) to train the model. The derivative of the function is a constant, and has no relation to the input, X. So it's not possible to go back and understand which weights in the input neurons can provide a better prediction.

All layers of the neural network collapse into one—with linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function). So a linear activation function turns the neural network into just one layer. Meaning that as soon as the data is nonlinear you will have massive performance loss because non-linearity is not captured in these functions.

Backpropagation is an algorithm commonly used to train neural networks. When the neural network is initialized, weights are set for its individual elements, called neurons. Inputs are loaded, they are passed through the network of neurons, and the network provides an output for each one, given the initial weights. Backpropagation helps to adjust the weights of the neurons so that the result comes closer and closer to the known true result.

Plot the training history and show it to your teacher (Question 8, 2 points)



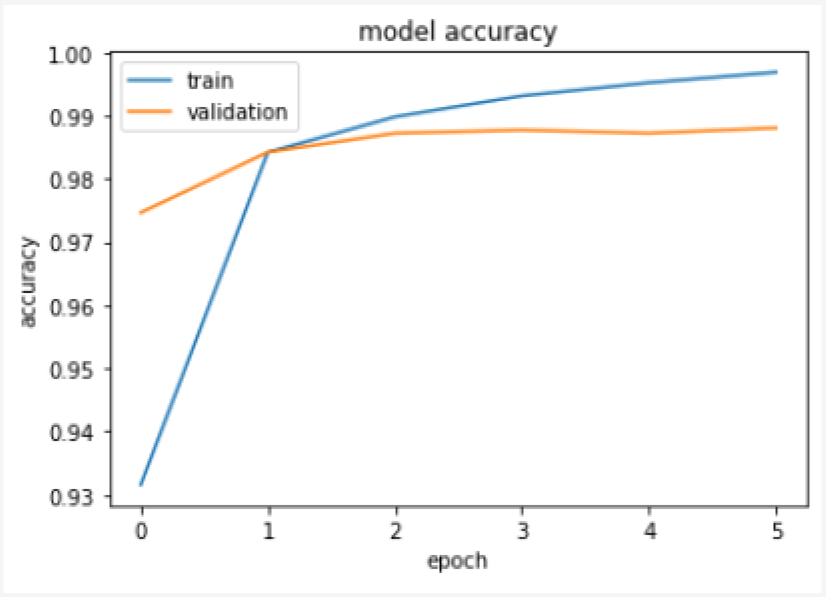
Discuss with your group, then describe to your teacher, how this training history differs from the previous model, for the training and validation sets. Describe what this tells us about the generalisation of the model. (Question 9, 5 points)

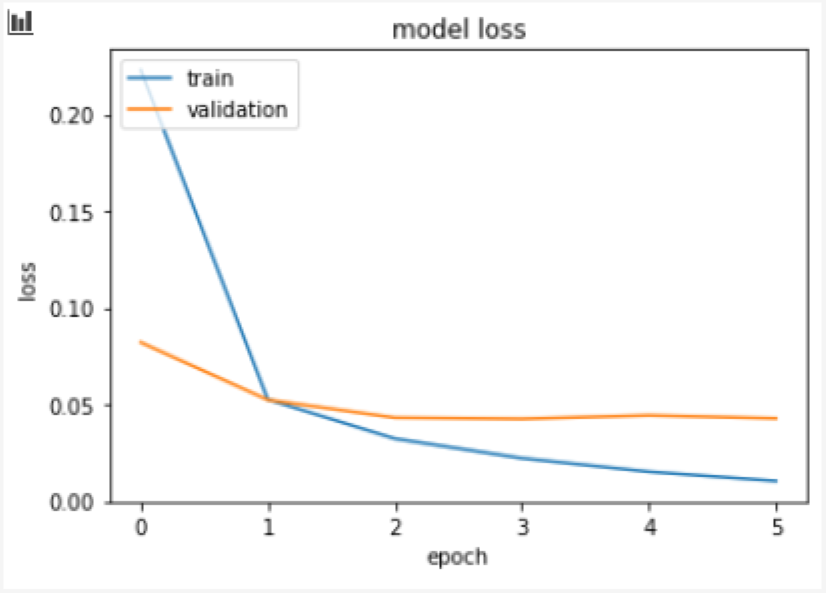
The model accuracy is higher and the model loss is lower for both the train and the validation sets. The model accuracy for the train set in the first epoch is a little higher, but the main difference between the models is that for the model with rectified activation the accuracy score increases significantly (and even reaches the 1.00). Besides, for the model with rectified activation, the accuracy for the validation training set is already higher for the first epoch (0.95

instead of 0.92) and it continues to increase; the model continues to learn in the validation set and reaches an accuracy of 0.97. For the model with rectified activation in the first hidden layer, the accuracy score for the validation shows a slight and gradual increase, while in the initial model the accuracy score fluctuated over the epochs. As such, the second model seems to show a more stable performance.

However, the model with rectified activation shows worse performance in the generalization from the training set to the validation set. The gap between the accuracy for the train set and the validatiation is notably large after epoch 2; while the accuracy of the train set continues to increase (almost to 1.00) the accuracy score for the validation set does not. The gap in accuracy between the train and validation dataset grows larger. It seems that the prediction is overfitted on the training data and therefore the model does not generalise well to new/validation data.

DEEP CONVOLUTIONAL NETWORKS
Plot the training history and show this to your teacher. (Question 10, 2 points)





Discuss with your group, then describe to your teacher, how the training history differs from the previous model, for the training and validation sets. What does this tell us about the generalisation of the model? (Question 11, 5 points)
Compared to the previous model, the training history is more consistent. This model, as a convolutional network, overall obtains a higher accuracy, especially the accuracy score for the validation dataset is higher in this model compared to the previous model, a linear/non-linear model. There is still a gap in the accuracy between the training data and the validation data. The

distance between the train and validation accuracy increases from epoch 1 to epoch 6. However this gap is smaller than the gap in accuracy of the model before, this indicates that the generalisation of this model is better than the one before.

Show your teacher what values you get for the model's accuracy and loss. (Question 12, 2 points)
Loss 0.03142546862363815
Accuracy 0.9894999861717224

Discuss with your group, then describe to your teacher, whether this accuracy is sufficient for some uses of automatic hand-written digit classification. (Question 13, 5 points)
This new model has high accuracy, almost 0.99. We think it is high enough for tasks like grading numerical tests, sorting the post and digitizing primary sources. Yet it is not enough for prescription medication or working with bank records. It is a high accuracy, but considering the high amount of numbers that will need to be processed and the consequences the mistakes made will still be too big.

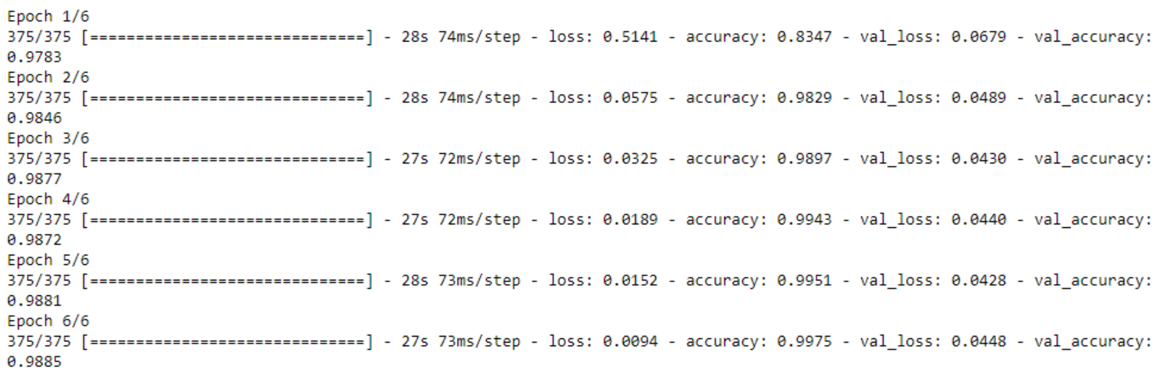
DROPOUT

Discuss with your group, then describe to your teacher, how the training history differs from the previous (convolutional) model, for both the training and validation sets, and for the time taken to run each model epoch (Question 14, 3 points)

For the previous model there was a higher accuracy for the train set than for the validation set. In this model, with the dropout, the accuracy of the validation data is better than the accuracy of the train data as can be seen in the graphs below. The dropout model results in better accuracy and model loss values of the train and validation data than the previous model.

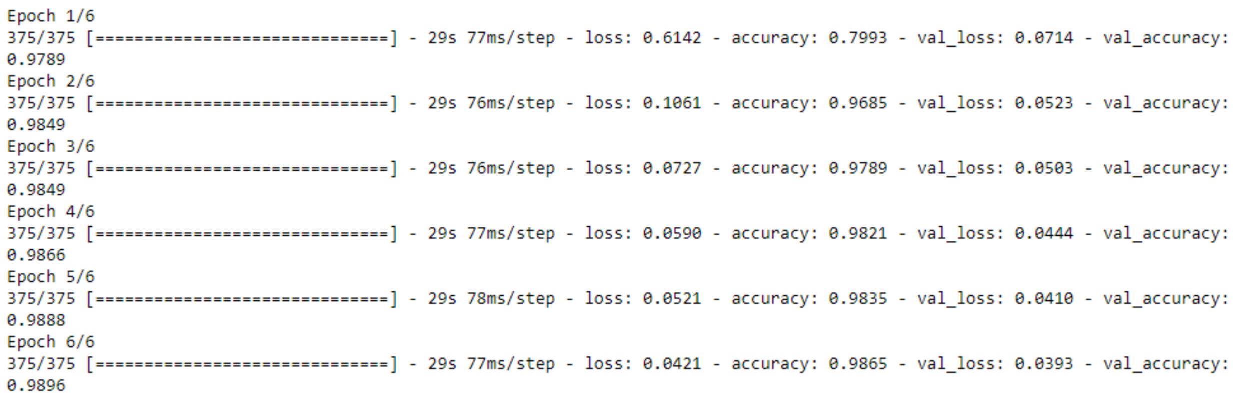
The difference in time between the two models in epoch duration is 165 seconds total duration without dropout and 174 seconds with dropout included (5,45% increase in total time). Interestingly, the amount of time for each epoch is fluctuating when there is no drop-out implemented (average fluctuation 0,5 s per epoch) but when there is one implemented you have no fluctuation at all in epoch duration.

Model of question 10 epoch stats (with the Adadeta optimizer but without dropout)

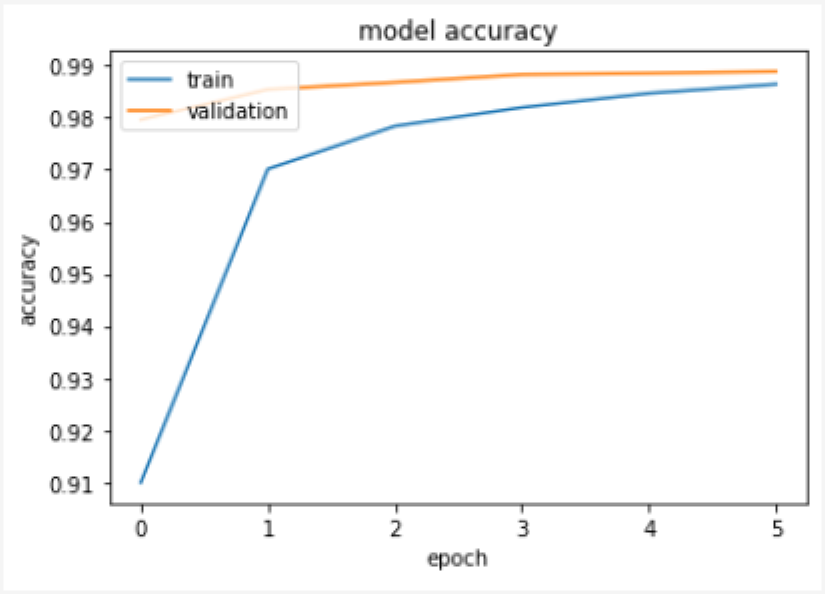


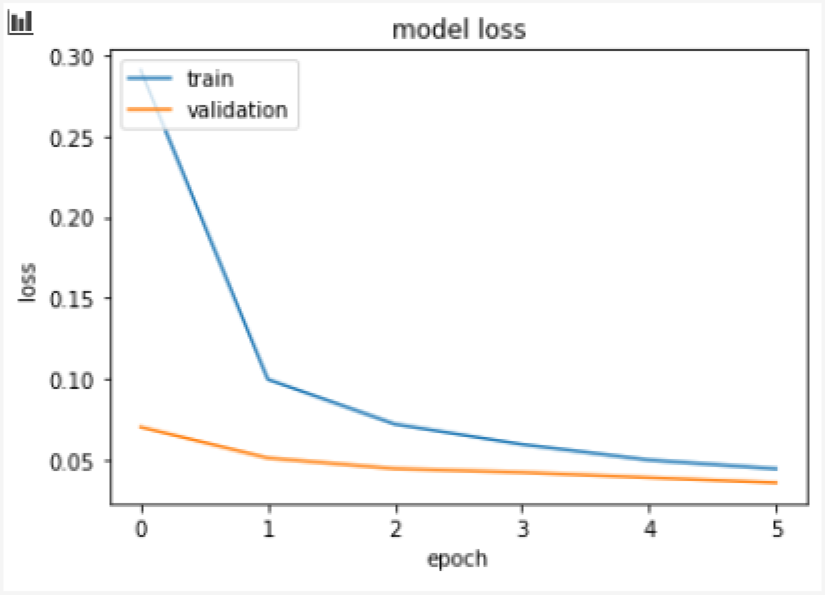
```
Epoch 1/6
375/375 [==============================] - 28s 74ms/step - loss: 0.5141 - accuracy: 0.8347 - val_loss: 0.0679 - val_accuracy:
0.9783
Epoch 2/6
375/375 [==============================] - 28s 74ms/step - loss: 0.0575 - accuracy: 0.9829 - val_loss: 0.0489 - val_accuracy:
0.9846
Epoch 3/6
375/375 [==============================] - 27s 72ms/step - loss: 0.0325 - accuracy: 0.9897 - val_loss: 0.0430 - val_accuracy:
0.9877
Epoch 4/6
375/375 [==============================] - 27s 72ms/step - loss: 0.0189 - accuracy: 0.9943 - val_loss: 0.0440 - val_accuracy:
0.9872
Epoch 5/6
375/375 [==============================] - 28s 73ms/step - loss: 0.0152 - accuracy: 0.9951 - val_loss: 0.0428 - val_accuracy:
0.9881
Epoch 6/6
375/375 [==============================] - 27s 73ms/step - loss: 0.0094 - accuracy: 0.9975 - val_loss: 0.0448 - val_accuracy:
0.9885
```

Drop-out model epoch stats (with Adadeta optimizer and dropout)



```
Epoch 1/6
375/375 [==============================] - 29s 77ms/step - loss: 0.6142 - accuracy: 0.7993 - val_loss: 0.0714 - val_accuracy:
0.9789
Epoch 2/6
375/375 [==============================] - 29s 76ms/step - loss: 0.1061 - accuracy: 0.9685 - val_loss: 0.0523 - val_accuracy:
0.9849
Epoch 3/6
375/375 [==============================] - 29s 76ms/step - loss: 0.0727 - accuracy: 0.9789 - val_loss: 0.0503 - val_accuracy:
0.9849
Epoch 4/6
375/375 [==============================] - 29s 77ms/step - loss: 0.0590 - accuracy: 0.9821 - val_loss: 0.0444 - val_accuracy:
0.9866
Epoch 5/6
375/375 [==============================] - 29s 78ms/step - loss: 0.0521 - accuracy: 0.9835 - val_loss: 0.0410 - val_accuracy:
0.9888
Epoch 6/6
375/375 [==============================] - 29s 77ms/step - loss: 0.0421 - accuracy: 0.9865 - val_loss: 0.0393 - val_accuracy:
0.9896
```





Discuss with your group, then describe to your teacher, what this tells us about the generalisation of the two models. (Question 15, 3 points)
Adding dropout after the max pooling stage and after the fully-connected (dense) layer prevents the model from overfitting on the training data. This improves the generalizability of the learning model for new data. When comparing the plots from the previous model (without dropout) and this model (with dropout) we see that while in the previous model there was a large gap between the accuracy of the training data and the validation data (the model was overfitted on the training data), the gap is significantly smaller in this model. This model with dropout has a better generalizability; the accuracy scores for the validation dataset are high, even higher than

for the training data set. The accuracy scores tell us that the model is not overfitted on the training data and it will most likely perform well in new data (with an accuracy of approximately 0.985). Hence, adding dropout in deep network training learns the model to generalise better to new data.

Code:

```
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
#Data preparation
(x_train, y_train),(x_test, y_test)=keras.datasets.mnist.load_data()
# reshape
x_train = np.reshape(x_train, (60000, 784))
x_test = np.reshape(x_test, (10000, 784))
# rescale to values between 0 and 1
x_train = x_train/255
x_test = x_test/255
y_train=keras.utils.to_categorical(y_train, 10)
y_test=keras.utils.to_categorical(y_test, 10)
#Model definition
model = keras.Sequential()
model.add(keras.layers.Dense(256, input_shape=(784,)))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()
#compile the model
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.RMSprop(), metrics='accuracy')
#model training and evaluation
history = model.fit(x_train, y_train, batch_size=128, epochs=12, verbose=1, validation_split=0.2)

#visualize in plot
   #accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
   # "Loss"
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

loss,accuracy = model.evaluate(x_test, y_test, verbose=0)


# Model rectified activation
model = keras.Sequential()
model.add(keras.layers.Dense(256, input_shape=(784,), activation='relu'))  #the first hidden layer
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()
#model compile
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.RMSprop(), metrics='accuracy')
#model fit
```

```
history = model.fit(x_train, y_train, batch_size=128, epochs=12, verbose=1, validation_split=0.2)
#visualize plot (same code as before)
loss,accuracy = model1.evaluate(x_test, y_test, verbose=0)


#Convolution Neural Network
(x_train1, y_train1),(x_test1, y_test1)=keras.datasets.mnist.load_data()
x_train1 = np.reshape(x_train1, (60000, 28, 28, 1))
x_test1 = np.reshape(x_test1, (10000, 28, 28, 1))
print(x_train1.shape)
print(x_test1.shape)
# rescale to values between 0 and 1
x_train1 = x_train1/255
x_test1 = x_test1/255
y_train1=keras.utils.to_categorical(y_train1, 10)
y_test1=keras.utils.to_categorical(y_test1, 10)

#model define
model1 = keras.Sequential()
model1.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu", input_shape=(28, 28, 1)))
model1.add(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation="relu"))
model1.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model1.add(keras.layers.Flatten())
model1.add(keras.layers.Dense(128, activation="relu"))
model1.add(keras.layers.Dense(10, activation="softmax"))
model1.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adadelta(learning_rate=1), metrics='accuracy')
model1.summary()
#model fit
history1 = model1.fit(x_train1, y_train1, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
#vizualize in plot
loss,accuracy = model1.evaluate(x_test1, y_test1, verbose=0)

model2 = keras.Sequential()
model2.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu", input_shape=(28, 28, 1)))
model2.add(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation="relu"))
model2.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model2.add(keras.layers.Dropout(rate=0.25))
model2.add(keras.layers.Flatten())
model2.add(keras.layers.Dense(128, activation="relu"))
model2.add(keras.layers.Dropout(rate=0.5))
model2.add(keras.layers.Dense(10, activation="softmax"))
model2.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adadelta(learning_rate=1), metrics='accuracy')
history2 = model2.fit(x_train1, y_train1, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
#vizualize in plot
loss,accuracy = model1.evaluate(x_test1, y_test1, verbose=0)
```

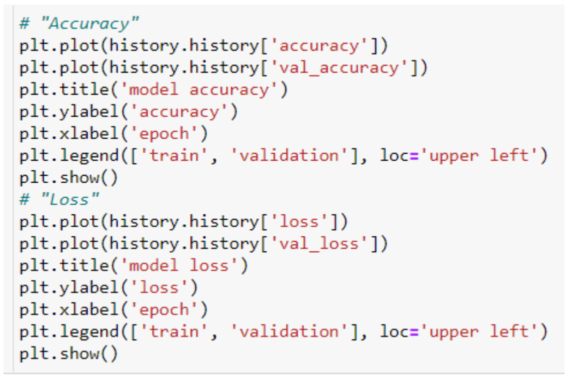
Exercise two: Identifying objects from images

Before fitting the model, show your teacher the code you used to define the model described here. (Question 16, 6 points)

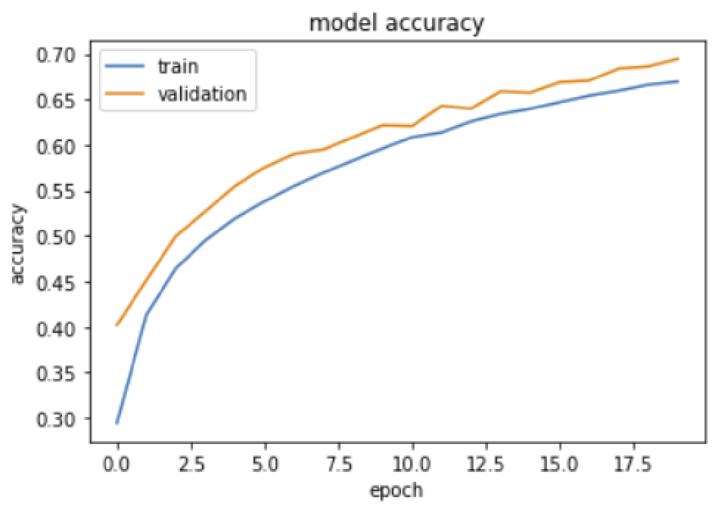
```
model = keras.Sequential()
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
activation="relu", input_shape=(32, 32, 3), padding = "same"))
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(rate=0.25))

model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
activation="relu"))
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(rate=0.25))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation="relu"))
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(10, activation="softmax"))
model.compile(loss='categorical_crossentropy',
optimizer=keras.optimizers.RMSprop(lr=0.0001, decay=1e-6),
metrics='accuracy')

history = model.fit(x_train, y_train, batch_size=32, epochs=20, verbose=1,
validation_data=(x_test, y_test), shuffle = True)
```

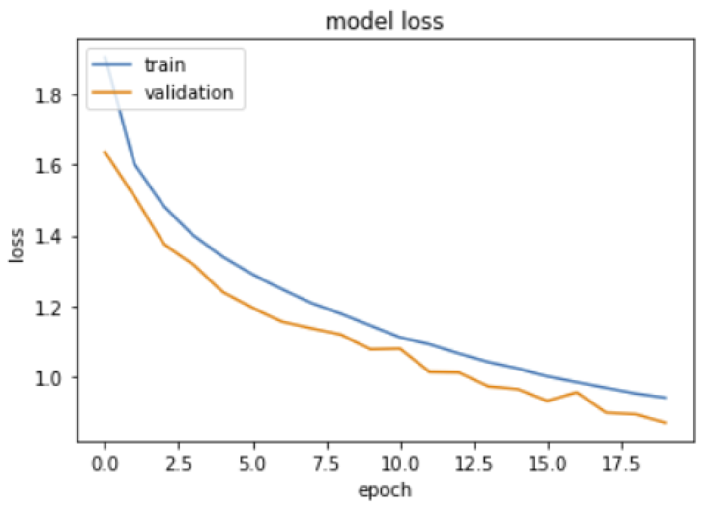
Plot the training history and show it to your teacher (Question 17, 2 points)

Code



```python
# "Accuracy"
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# "Loss"
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





```
(0.8721948266029358, 0.6945000290870667)
```

Discuss with your group, then describe to your teacher, how the training history differs from the convolutional model for digit recognition and why. (Question 18, 5 points)

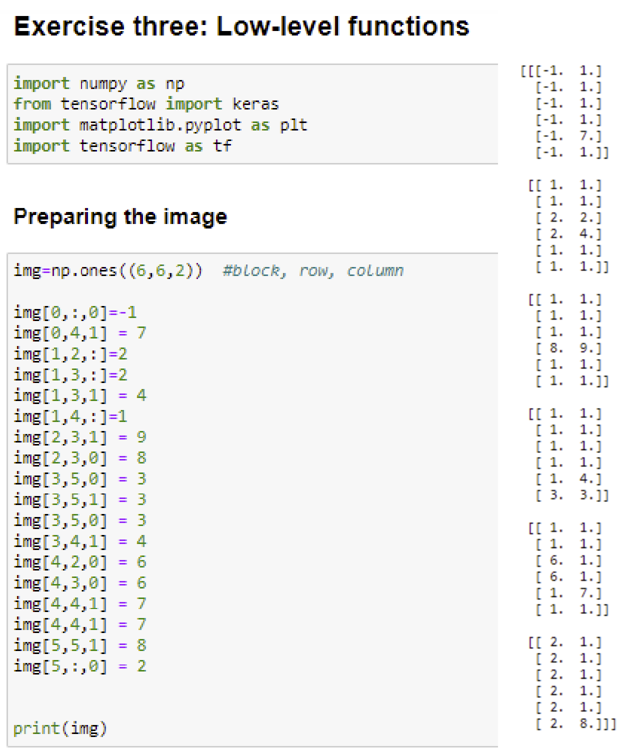
In comparison to the convolutional model for digit recognition the overall accuracy score is significantly lower; in this new model we reach an accuracy score of only 0.69. This is most likely because the task of identifying objects from images is a more complex one. For this reason we built a network with more layers that is more computationally-intensive. The accuracy scores tell us that the model is not overfitted on the training data and even performs better on new data (with an accuracy of approximately 0.7).

Discuss with your group, then describe to your teacher, how the time taken for each training epoch differs from the convolutional model for digit recognition. Give several factors that may contribute to this difference (Question 19, 4 points)

The time taken for each training epoch was noticeably lower for the convolutional model for digit recognition, where each epoch took about 41 seconds. In this new model with object recognition in images an epoch lasted approximately 67 seconds. This network for object recognition in images has more layers and is more computationally-intensive than the convolutional model for digit recognition.

This difference is mainly caused by the amount of layers in the models. In the convolutional model there were only 2 convolutional layers & 2 dense layers, while in the new model 4 convolutional layers & 2 dense layers are used in the model. There are more object classes, different types of images, higher resolution, different viewing angles and the images are in colour.

Exercise Three: Low-level functions



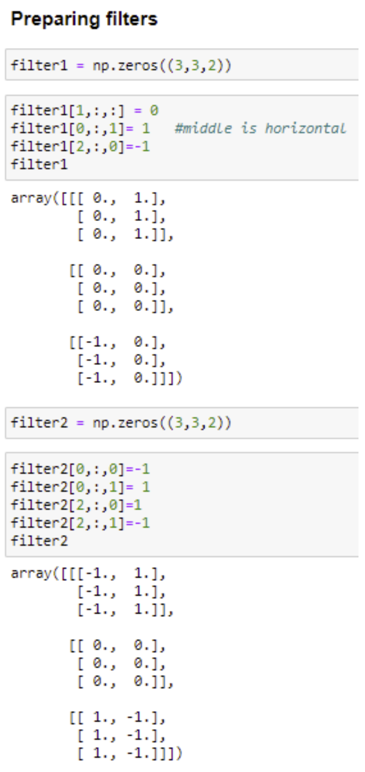
Exercise three: Low-level functions

```
import numpy as np
from tensorflow import keras
import matplotlib.pyplot as plt
import tensorflow as tf
```

Preparing the image

```
img=np.ones((6,6,2))  #block, row, column

img[0,:,0]=-1
img[0,4,1] = 7
img[1,2,:]=2
img[1,3,:]=2
img[1,3,1] = 4
img[1,4,:]=1
img[2,3,1] = 9
img[2,3,0] = 8
img[3,5,0] = 3
img[3,5,1] = 3
img[3,5,0] = 3
img[3,4,1] = 4
img[4,2,0] = 6
img[4,3,0] = 6
img[4,4,1] = 7
img[4,4,1] = 7
img[5,5,1] = 8
img[5,:,0] = 2


print(img)
```

```
[[[-1.  1.]
  [-1.  1.]
  [-1.  1.]
  [-1.  1.]
  [-1.  7.]
  [-1.  1.]]

 [[ 1.  1.]
  [ 1.  1.]
  [ 2.  2.]
  [ 2.  4.]
  [ 1.  1.]
  [ 1.  1.]]

 [[ 1.  1.]
  [ 1.  1.]
  [ 1.  1.]
  [ 8.  9.]
  [ 1.  1.]
  [ 1.  1.]]

 [[ 1.  1.]
  [ 1.  1.]
  [ 1.  1.]
  [ 1.  1.]
  [ 1.  4.]
  [ 3.  3.]]

 [[ 1.  1.]
  [ 1.  1.]
  [ 6.  1.]
  [ 6.  1.]
  [ 1.  7.]
  [ 1.  1.]]

 [[ 2.  1.]
  [ 2.  1.]
  [ 2.  1.]
  [ 2.  1.]
  [ 2.  1.]
  [ 2.  8.]]]
```



Preparing filters

```
filter1 = np.zeros((3,3,2))
```

```
filter1[1,:,:] = 0
filter1[0,:,1]= 1   #middle is horizontal
filter1[2,:,0]=-1
filter1
```

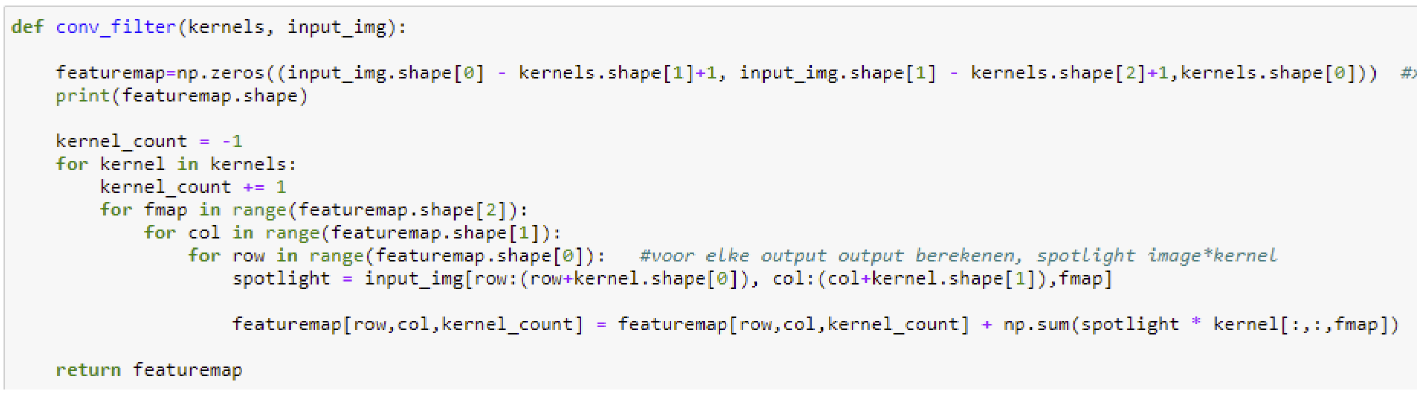
```
array([[[ 0.,  1.],
        [ 0.,  1.],
        [ 0.,  1.]],

       [[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]],

       [[-1.,  0.],
        [-1.,  0.],
        [-1.,  0.]]])
```

```
filter2 = np.zeros((3,3,2))
```

```
filter2[0,:,0]=-1
filter2[0,:,1]= 1
filter2[2,:,0]=1
filter2[2,:,1]=-1
filter2
```

```
array([[[-1.,  1.],
        [-1.,  1.],
        [-1.,  1.]],

       [[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]],

       [[ 1., -1.],
        [ 1., -1.],
        [ 1., -1.]]])
```

Question 20: Convolutional filter



```
def conv_filter(kernels, input_img):

    featuremap=np.zeros((input_img.shape[0] - kernels.shape[1]+1, input_img.shape[1] - kernels.shape[2]+1,kernels.shape[0]))  #
    print(featuremap.shape)

    kernel_count = -1
    for kernel in kernels:
        kernel_count += 1
        for fmap in range(featuremap.shape[2]):
            for col in range(featuremap.shape[1]):
                for row in range(featuremap.shape[0]):    #voor elke output output berekenen, spotlight image*kernel
                    spotlight = input_img[row:(row+kernel.shape[0]), col:(col+kernel.shape[1]),fmap]

                    featuremap[row,col,kernel_count] = featuremap[row,col,kernel_count] + np.sum(spotlight * kernel[:,:,fmap])

    return featuremap
```

```
filters = np.array([filter1, filter2])
#print(filters)
featuremaps=conv_filter(filters, img)
print(featuremaps)
```

```
(4, 4, 2)
[[[ 0.  6.]
  [-7.  5.]
  [-1. 11.]
  [-1. 11.]]

 [[ 1.  0.]
  [ 4.  2.]
  [ 4. -1.]
  [ 1. -1.]]

 [[-5.  5.]
  [-2. 11.]
  [-2.  5.]
  [ 3.  0.]]

 [[-3.  3.]
  [-3.  3.]
  [ 0.  6.]
  [ 2. -1.]]]
```

Question 21: Rectified linear activation

```
# rectified linear function
def rectified(x):
    return np.maximum(0, x)

def relu(input_img):
    input_img = rectified(input_img)
    return input_img
```

```
relu(featuremaps)
```

```
array([[[ 0.,  6.],
        [ 0.,  5.],
        [ 0., 11.],
        [ 0., 11.]],

       [[ 1.,  0.],
        [ 4.,  2.],
        [ 4.,  0.],
        [ 1.,  0.]],

       [[ 0.,  5.],
        [ 0., 11.],
        [ 0.,  5.],
        [ 3.,  0.]],

       [[ 0.,  3.],
        [ 0.,  3.],
        [ 0.,  6.],
        [ 2.,  0.]]])
```

```
relu_fmap=relu(featuremaps)
relu_fmap.shape
```

```
(4, 4, 2)
```

Question 22: Max pooling ¶

```python
def max_pooling(input_fmap, size=2, stride=2):
    #Preparing the output of the pooling operation.
    pool_out = np.zeros((np.uint16((input_fmap.shape[0]/size)),
                        np.uint16((input_fmap.shape[1]/size)),
                        input_fmap.shape[2]))
    print(pool_out.shape)
    for fmap in range(input_fmap.shape[2]):
        for col in range(0,input_fmap.shape[1], size):
            for row in range(0, input_fmap.shape[0], size):
                max_spotlight = input_fmap[row:(row+size), col:(col+size), fmap]
                pool_out[row//size,col//size,fmap] = np.max(input_fmap[row:row+size, col:col+size, fmap])
    return pool_out
```

```python
pooled_fmap=max_pooling(relu_fmap,2,2)
pooled_fmap
```

```
(2, 2, 2)
```

```
array([[[ 4.,  6.],
        [ 4., 11.]],

       [[ 0., 11.],
        [ 3.,  6.]]])
```

Question 23: Normalisation

```python
def normalize(x, mean, sd):    #mean 0, standard deviation 1
    value = (x - mean) / sd
    return value

def normalization(input_map):
    for fmap in range(input_map.shape[2]):
        mean = np.mean(input_map[:,:,fmap])
        standdev = np.std(input_map[:,:,fmap])
        input_map[:,:,fmap] = normalize(input_map[:,:,fmap], mean, standdev)

    return input_map
```

```python
norm_fmap=normalization(pooled_fmap)
norm_fmap
```

```
array([[[ 0.76249285, -1.        ],
        [ 0.76249285,  1.        ]],

       [[-1.67748427,  1.        ],
        [ 0.15249857, -1.        ]]])
```

Question 24: Fully connected layer

```python
#24: fully connected layer
def connectedlayer(inputmap, noutput, bias):
    flatmap = inputmap.flatten()
    weights = np.random.rand(len(flatmap), noutput) #geeft een numpy array met random waardes. Dimensie van de matrix geef je doc
    output = np.dot(flatmap, weights) + bias #vermenigvuldigd de matrices met elkaar om een output te krijgen
    return output
```

```python
connected_fmap = connectedlayer(inputmap =norm_fmap, noutput=3, bias=1)
connected_fmap
```

```
array([2.48737338, 0.62093306, 2.10904643])
```

Question 25: Classification probabilities softmax

```
[ ]: def softmax(inputmap):
         e_x = np.exp(inputmap)
         return e_x / e_x.sum()
```

```
[ ]: softmax(connected_fmap)
```

```
[ ]: array([0.54357286, 0.08407619, 0.37235094])
```