CAS MAIN, deep learning

## Finger Exercises day 5

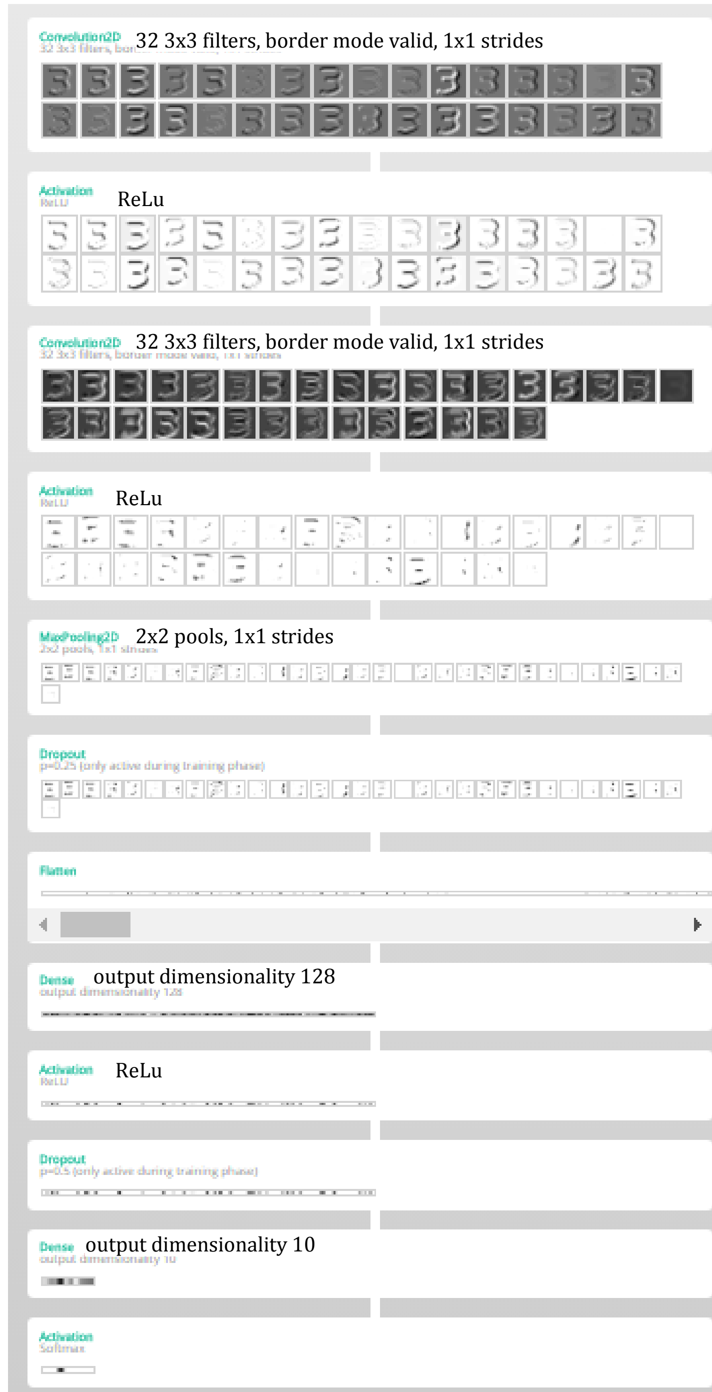
Topic: live CNN, Keras Syntax and CNN feature layer

Follow the first link of day 5 on our course-webpage “live cnn in browser”
https://transcranial.github.io/keras-js/#/mnist-cnn

a) Draw some digits and check out what is happening in the CNN running live in the browser.

b) Fill the gaps in the following Keras Code:

```
model = Sequential()
model.add(Convolution2D(… , …, …,
            border_mode='valid',
            input_shape=(… ,… ),
            dim_ordering='tf'))
model.add(Activation('…………….'))
model.add(Convolution2D(… , …, …,
            border_mode='valid',
            dim_ordering='tf'))
model.add(Activation('…………….'))
model.add(MaxPooling2D(pool_size=(… , …),
            border_mode='valid',
            dim_ordering='tf'))
model.add(Dropout(…………….))
model.add(Flatten())
model.add(Dense(…………….))
model.add(Activation('relu'))
model.add(Dropout(…………….))
model.add(Dense(…………….))
model.add(Activation('softmax'))
```




c) Mark the layer in which the “code” can be found, that is used to decide on the digit class.

d) How long is in the genertated CNN “code” - often called CNN feature vector representation?

## Convolutional Neural Networks for image data

Exercise 1:
We want to investigate, if a CNN outperforms a fc NN on image data.

First we recall the design of the fc NN which performed so far best on MNIST when only keeping 4000 examples in the training data set (see below). With this NN we have reached ~91% accuracy on the validation data set. 07_fcn_MNIST_keras_solution.ipynb

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 500)               392500    
_________________________________________________________________
batch_normalization_1 (Batch (None, 500)               2000      
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0         
_________________________________________________________________
activation_1 (Activation)    (None, 500)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 50)                25050     
_________________________________________________________________
batch_normalization_2 (Batch (None, 50)                200       
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0         
_________________________________________________________________
activation_2 (Activation)    (None, 50)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510       
=================================================================
Total params: 420,260.0
Trainable params: 419,160.0
Non-trainable params: 1,100.0
_________________________________________________________________
```

a) Where do we spend most learnable parameter? Can you explain the “Param #” of the dense_1 layer?
Remark: dense layer is the same as fully connected layer.

b) Now we want to use our first CNN with only 1 convolutional and 1 dense layer:

```
________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
================================================================================================
convolution2d_1 (Convolution2D)  (None, 28, 28, 32)    320         convolution2d_input_1[
________________________________________________________________________________________________
activation_1 (Activation)        (None, 28, 28, 32)    0           convolution2d_1[0][0]
________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 25088)         0           activation_1[0][0]
________________________________________________________________________________________________
dense_1 (Dense)                  (None, 10)            250890      flatten_1[0][0]
________________________________________________________________________________________________
activation_2 (Activation)        (None, 10)            0           dense_1[0][0]
================================================================================================
Total params: 251,210
Trainable params: 251,210
Non-trainable params: 0
________________________________________________________________________________________________
```

In which layer do we need to learn most parameter/weights?

Do you expect with this cnn1 more or less overfitting then in the fc NN above? Why?

Please open the ipython-Notebook 08_cnn1_mnist.ipynb and try to understand the code and run the code.

How large is the accuracy on the validation set? Do you observe overfitting? Describe how you check for overfitting and/or sketch the corresponding graph.

Restart the kernel and run the model without first standardizing the data which is done in code cell 7 after # here we center and standardize the data. Instead of commenting out each command, you can turn the cell in Cell Type "markdown" and then run the code.

How large is now the accuracy on the validation set? Can you explain, what happened?

c) Please open the ipython-Notebook 08_cnn2_mnist_No_solution.ipynb. Search for the position in the code which is marked by "# here is your code coming:" and add the missing layers - the missing layers are marked below. How is the performance of cnn2?

```
________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
================================================================================================
convolution2d_1 (Convolution2D)  (None, 28, 28, 32)    320         convolution2d_input_1[
________________________________________________________________________________________________
batchnormalization_1 (BatchNorma (None, 28, 28, 32)    128         convolution2d_1[0][0]
________________________________________________________________________________________________
activation_1 (Activation)        (None, 28, 28, 32)    0           batchnormalization_1[0
________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 28, 28, 32)    9248        activation_1[0][0]
```

```
batchnormalization_2 (BatchNorma (None, 28, 28, 32)    128          convolution2d_2[0][0]

activation_2 (Activation)        (None, 28, 28, 32)    0            batchnormalization_2[0

maxpooling2d_1 (MaxPooling2D)    (None, 14, 14, 32)    0            activation_2[0][0]

convolution2d_3 (Convolution2D)  (None, 14, 14, 64)    18496        maxpooling2d_1[0][0]

batchnormalization_3 (BatchNorma (None, 14, 14, 64)    256          convolution2d_3[0][0]

activation_3 (Activation)        (None, 14, 14, 64)    0            batchnormalization_3[0


--------------------START OF THE MISSING LAYERS------------------------------------------

convolution2d_4 (Convolution2D)  (None, 14, 14, 64)    36928        activation_3[0][0]

batchnormalization_4 (BatchNorma (None, 14, 14, 64)    256          convolution2d_4[0][0]

activation_4 (Activation)        (None, 14, 14, 64)    0            batchnormalization_4[0

maxpooling2d_2 (MaxPooling2D)    (None, 7, 7, 64)      0            activation_4[0][0]

--------------------END OF THE MISSING LAYERS--------------------------------------------


flatten_1 (Flatten)              (None, 3136)          0            maxpooling2d_2[0][0]

dense_1 (Dense)                  (None, 200)           627400       flatten_1[0][0]

batchnormalization_5 (BatchNorma (None, 200)           800          dense_1[0][0]

dropout_1 (Dropout)              (None, 200)           0            batchnormalization_5[0

activation_5 (Activation)        (None, 200)           0            dropout_1[0][0]

dense_2 (Dense)                  (None, 10)            2010         activation_5[0][0]

activation_6 (Activation)        (None, 10)            0            dense_2[0][0]
==========================================================================================
Total params: 695,970
Trainable params: 695,186
Non-trainable params: 784
```

# Deep Learning Course

## 8 faces fine tuning

In this excercise we work with the 8 faces dataset. We want to improve the performance by using a
pretrained vgg16 network. We predict the features on the fc1 layer with the already learned weights on imagenet and then train a small fully connected network for our own labels. The feature extraction was done in this notebook vgg16_feature_extraction_8_faces

a) What do you expect, will it increase our performance? Why? What's the idea behind this so called fine tuning?

b) Open the notebook 8 faces fine tuning and bulit this network and then train it.

```
____________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================
dense_1 (Dense)                  (None, 400)           1638800     dense_input_1[0][0]
____________________________________________________________________________________
batchnormalization_1 (BatchNorma (None, 400)           1600        dense_1[0][0]
____________________________________________________________________________________
activation_1 (Activation)        (None, 400)           0           batchnormalization_1
____________________________________________________________________________________
dropout_1 (Dropout)              (None, 400)           0           activation_1[0][0]
____________________________________________________________________________________
dense_2 (Dense)                  (None, 400)           160400      dropout_1[0][0]
____________________________________________________________________________________
batchnormalization_2 (BatchNorma (None, 400)           1600        dense_2[0][0]
____________________________________________________________________________________
activation_2 (Activation)        (None, 400)           0           batchnormalization_2
____________________________________________________________________________________
dropout_2 (Dropout)              (None, 400)           0           activation_2[0][0]
____________________________________________________________________________________
dense_3 (Dense)                  (None, 8)             3208        dropout_2[0][0]
____________________________________________________________________________________
```

```
activation_3 (Activation)        (None, 8)             0           dense_3[0][0]
====================================================================================
Total params: 1,805,608
Trainable params: 1,804,008
Non-trainable params: 1,600
------------------------------------------------------------------------------------
```

c) Complete the code to get the predicted labels out of the probability vector and look at the accuracy on the test data.