EE837 Assignment

Jonas Nikula
20176392

1 Implementation

I did the assignment using Python 3 and the Keras deep learning library. Training was done on the FloydHub cloud computing platform.

1.1 Code details

The code itself is very simple. It's divided into 4 files.

`create_model.py`
`train_and_eval.py`
`load_and_eval.py`
`read_data.py`

The file `create_model.py` contains the actual model, and a function that returns the model. `train_and_eval.py` instantiates and trains the model, printing out the model accuracy on the test data at the end. `load_and_eval` only prints out the accuracy, using a trained model. `read_data.py` contains a helper function to read in the MNIST data set. The bash script `run_floydhub.sh` is just to make model training easier.

When the code is run on FloydHub, it looks for the dataset in the directory `/data`, which is the data directory on the cloud platform. If it can't find it, it assumes that it's being run locally, and looks for the data in `./data`. The same principle holds for the output folder.

1.2 Model architecture and training

The model consists of 4 convolutional "blocks" and one dense block at the end. Each convolutional block consists of three conv-layers, with decreasing filter sizes from 3 to 1. They all use ReLU as the activation function. At the end of a convolutional block there's a max pooling layer with a pool size of 2, and a dropout layer with a 25% chance of dropout. All the conv-layers use zero-padding, so that the output dimensions are the same. The number of filters increases from 64 to 512, doubling at each block. The goal of this architecture was to get a lot of non-linearity, with a manageable parameter count.

The dense layer consists of 128 unit layer, with ReLU activation and a 50% chance of dropout. The final layer has 10 units with softmax activation. The Appendix A contains just the code to define the appendix, for an easy overview. The code should be clear enough to understand without knowledge of Python or Keras.

Training was done with **categorical crossentropy** as the loss function and **stochastic gradient descent** as the optimizer, with a **learning rate** of 0.01, a **momentum** of 0.9 and a **weight decay** of 1×10^{-6}. Training ran for 20 epochs with a batch size of 128. Training took about 16 minutes for the final model.

2 Performance

During training the model reported — at best — an accuracy of 0.9969. With the test set the model evaluated to an accuracy of 0.9946.

A Model code

```
####### CONV ###########################################################################
        filter_count = 64

        model.add( Conv2D(filters=filter_count, kernel_size=(3,3), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(2,2), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(1,1), padding='same'))
        model.add( Activation('relu') )

        model.add( MaxPooling2D(pool_size=(2,2)) )
        model.add( Dropout(0.25) )

####### CONV ###########################################################################
        filter_count = 128

        model.add( Conv2D(filters=filter_count, kernel_size=(3,3), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(2,2), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(1,1), padding='same'))
        model.add( Activation('relu') )

        model.add( MaxPooling2D(pool_size=(2,2)) )
        model.add( Dropout(0.25) )

####### CONV ###########################################################################
        filter_count = 256

        model.add( Conv2D(filters=filter_count, kernel_size=(3,3), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(2,2), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(1,1), padding='same'))
        model.add( Activation('relu') )

        model.add( MaxPooling2D(pool_size=(2,2)) )
        model.add( Dropout(0.25) )
```

```
####### CONV ###########################################################################
        filter_count = 512

        model.add( Conv2D(filters=filter_count, kernel_size=(3,3), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(2,2), padding='same'))
        model.add( Activation('relu') )

        model.add( Conv2D(filters=filter_count, kernel_size=(1,1), padding='same'))
        model.add( Activation('relu') )

        model.add( MaxPooling2D(pool_size=(2,2)) )
        model.add( Dropout(0.25) )

####### DENSE ###########################################################################
        model.add( Flatten() )

        model.add( Dense(128) )
        model.add( Activation('relu') )
        model.add( Dropout(0.5) )

        model.add( Dense(10) )
        model.add( Activation('softmax') )
```