# Milestone 1: Report

## The Dataset

The dataset assigned to our group (accessible here) is the MNIST database, which is a subset of a larger database NIST that consists of Special Database 3 (SD-3) and Special Database 1 (SD-1). In the NIST, the SD-3 was used as the training set and the SD-1 database was used as the test set. But SD-3 is much cleaner and easier to recognize than SD-1, therefore the databases are mixed in the MNIST.

The original bilevel images from NIST were size normalized to fit a 20x20 pixel box and centered in a 28x28 image. The MNIST training set consists of 30 000 patterns from SD-3 and 30 000 patterns from SD-1 and it contains examples from approximately 250 writers. The test set consists of 60 000 patterns as well, however, only a subset of this test set is available on the site. This subset is of 10 000 test images, 5 000 from each database.

There are 4 files available on the website:

- training set images (60 000 patterns)
- test set images (10 000 patterns)
- training set labels
- test set labels

The data is stored in the IDX file format for vectors and multidimensional matrices of various numerical types.

The machine learning algorithm used on this dataset solves a classification problem, because our goal is to predict a categorical output/value. We classify each image into one of ten possible categories depending on which number is written in the image (0-9).

## The Code

The code base that was assigned to us (accessible here) is from the Keras Team and is stored in their examples folder. The code uses the Keras package which requires installation - will be described below.
The code base uses convolutional neural network to train the algorithm, which is commonly applied to analyse visual imaginery.

## The .py File

To get the code in a .py format, we click on Raw, copy the URL of the page and use the command:

```
$ wget https://raw.githubusercontent.com/keras-team/keras/master/examples/mnist_cnn.py
```

This code saves the code as a .py file called mnist_cnn.py to the current working directory. In our case, it was saved to the milestone_1 branch of our git repository.
After that, we need to take the following steps to commit the file to the repository.

```
$ git add mnist_cnn.py

$ git commit

$ git push
```

After this set of commands, the file can be found on the github page of our repository (on the milestone_1 branch).

## Running the Code

As already mentioned above, installation of the Keras package was needed to run the code. We start the installation by installing pip, if it is not yet installed, with the command:

```
$ sudo apt-get install -y python3-pip
```

In the next step, we upgrade the setuptools.

```
$ pip3 install ——upgrade setuptools
```

After that, we move on to the installation of tensorflow.

```
$ pip3 install tensorflow
```

Now we can finally turn to the installation of the keras package.

```
$ pip3 install keras
```

To make sure the installationo was successful, we can check with the following commands:

```
$ pip3 show tensorflow

# after the installation of tensorflow
# since everything was okay, we proceeded to the installationo of keras

$ pip3 show keras
```

To run the .py file and run the code, we use the command:

```
$ python3 mnist_cnn.py
```

To find out which version of Python is being used to run the code, we can use the following command:

```
$ python3 --version
```

The output is: Python 3.8.5

The dependencies required to run the code with keras package can be seen after using the pip show keras command which shows us the needed packages under required.

```
$ pip show keras
```

In the output, we can see that the Keras version is 2.4.3 and the required packages are scipy, pyyaml, numpy and h5py.

To check which versions of the required packages are used, we use the command pip list and check the versions in the list.

```
$ pip list
```

The following versions of the packages were used:

| Package | Version |
|---|---|
| scipy | 1.5.2 |
| pyyaml | 5.3.1 |
| numpy | 1.18.5 |
| h5py | 2.10.0 |

The versions could be dependent on the system the code is being run on. We checked the versions on both of our machines and all versions are the same except for scipy, where on my machine the version is 1.5.2 and on Vitor's machine is version 1.5.0.

## The Code - What Does It Do?

To understand the code, we will divide into several parts and explain them. First the used packages are loaded.

```
from __future__ import print_function
```

Thereby we can ignore the first loaded library because we are working with Python 3 and not with 2. (Used so that Python 2 can also print so cool).

The most important library is of course the Keras, which is used here for simple Convolutional Neural Network (CNN). Then the second most important library is loaded, Sequential, so that we can actually make the CNN. ( We loaded the Data by importing the mnist Data).

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
```

Then dense, dropout and flatten are loaded.

```
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

Dense is needed to implement the operation.output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument. Dropout is a simple but powerful regularization technique used in CNN.dropout causes neurons to fall out of the network randomly during training! So… The other neurons have to step in and handle the representation needed to make predictions for the missing neurons. Result: Several independent internal representations are learned by the network! Flattening is required to unroll values starting at the last dimension! This will allow our output to be processed by the fully connected layers.

Let us now jump to the code. What is the main idea? With this code we try to recognize handwritten numbers from the MNIST database. Each image has 28 * 28 pixels, 784 pixels in total.

The second part is the processing of our data for CNN. We defined this:

```
batch_size = 128
num_classes = 10
epochs = 12
```

The algorithms train the first 128 samples. Then they train the next 128 samples. The total number of classes is needed to derive the largest number in y + 1.

If the epoch is defined, so that the whole data set is passed forward and backward through the nn! (Defines how many times).

Like now, we have to flatten our image data sets, which are structured as 28 by 28 pixels!

```
img_rows, img_cols = 28, 28
```

Now it is time to normalize! To do this, we use the if else command, so we try to normalize the pixel values from the gray scale so that they are between 0 and 255 up to a range between 0 and 1. How do we do that? By dividing each by 255!

```
if K.image_data_format() == 'channels_first':
  x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
  x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
  input_shape = (1, img_rows, img_cols)
else:
  x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
  x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
  input_shape = (img_rows, img_cols, 1)
```

Then we use hot-coding (to create more columns so we have better prediction). Then we want to convert the numbers into vectors from 0 to 9. For example, the number would be 4 (0 0 0 0 4 0 0 0 0 0). This regularization is done by the function utils.to_categorical() provided by Keras.

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

The next part is the definition of the model. Let's get to the interesting part! First we have defined the first hidden layer Conv2D is a convolution layer that has 32 feature maps, each with a size of 3x3.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                activation='relu',
                input_shape=input_shape))
```

Then we use a rectified linear activation function - relu. (In short, this is a piecewise linear function that outputs the input directly if it is positive, otherwise it will output zero!) We also add convolution layer with 64 feature maps.

```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

We also add a pooling layer MaxPooling2D.

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

Then we apply a regularization layer with dropout, which is set to randomly exclude 25% of the neurons in the layer.

```
model.add(Dropout(0.25))
```

Then we convert the 2-dimensional matrix into a vector with flattening.

```
model.add(Flatten())
```

Next, we add a fully connected layer with 128 neurons and a ReLU activation function.

```
model.add(Dense(128, activation='relu'))
```

Then we add another regularization layer to reduce overmatching. This time we randomly exclude 50% of the neurons. This will output a prediction of the probability that a digit belongs to each class.

```
model.add(Dropout(0.5))
```

We end the neural network with an output layer that has 10 neurons.

```
odel.add(Dense(num_classes, activation='softmax'))
```

Then we had to compile our model. We do this by working with a logarithmic loss and the Adeadelta optimizer.

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

Then we train our model.

```
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

At the same time we print the results so that we can see how well our model is doing!

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```