|  |  |
| --- | --- |
|  | M1 E3A - Electrical and Optical Engineering (EOE)  Signal processing and statistical Data Analysis |

Report

**Lab Work: Convolutional Neural networks.**

Made by: Botirov Khojiakbar

khojiakbar.botirov@telecom-sudparis.eu

**Introduction**

The goal of this laboratory work is to implement Convolutional Neural Networks to classify MNIST handwritten digit images using Keras and Tensorflow frameworks. A dataset is consisted of 70000 images of digits, which were taken from different handwritten documents, each image has a size of 28x28 pixels, the gray level of each being between 0 and 255.

A dataset of handwritten numerals is splitted into two groups, training set and testing set, in following proportions, 60000 and 10000 respectively.

Training set: x\_train = 60000;

Results of training set: y\_train;

Testing set: x\_test = 10000;

Results of testing set: y\_test;

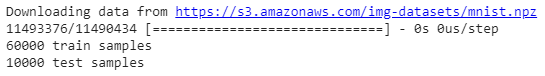


Figure 1. Output of dataset splitting.

In order to visualize loaded dataset, following library is used: **matplotlib.pyplot.**

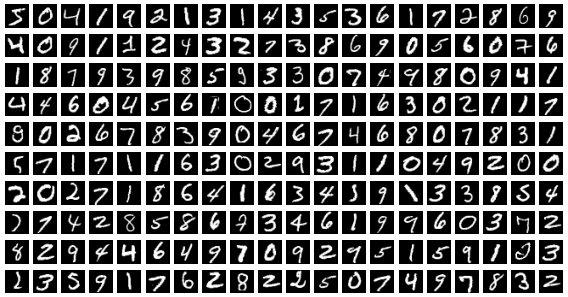


Figure 2. First 200 images of training set.

**Description of the methodology:**

**Model architecture:**

The given model grouped a linear stack of layers into sequential model. CNN architecture is defined as follows:

1. The first hidden layer is a convolutional layer called a Convolution2D. The layer has 32 feature maps or filters, which with the size of 3×3 and a rectifier (“relu”) activation function. This is the input layer, expecting images with the structure outline above [pixels][width][height].

Conv2D:

model.add(Conv2D(32, kernel\_size=(3, 3),

activation=**'relu**',

input\_shape=**(28,28,1)**))

**32** - is a number of filters,

kernel\_size - An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

relu is used as a activation function

input\_shape = (batch\_size, rows, cols, channels) for data\_format = “channels last”, at output\_shape rows and cols have changed due to padding, that is it equals to **(26, 26, 32)**

1. The second layer also convolutional layer with 64 feature maps or filters and it’s input and shape equals as follows: **(26,26,32)** and **(24,24,64): 2Dmatrix**

model.add(Conv2D(64, (3,3), activation = ‘relu’))

1. Next l a pooling layer that takes the max called MaxPooling2D. It is configured with a pool size of 2×2, integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.

Input and output shape equals as follows: **( 24, 24, 64)** and **(12, 12, 64): 2Dmatrix**

model.add(MaxPooling2D(pool\_size = (2,2)))

1. The next layer is a regularization layer using dropout called Dropout. It is configured to randomly exclude 25% of neurons in the layer in order to reduce overfitting. This layer randomly sets 0 to input values with a frequency of rate and inputs not set to 0 are scaled up by 1/(1 – rate), so the sum of all inputs are unchanged.

Input and output shape are equal: **( 12, 12, 64)** and **(12, 12, 64): 2Dmatrix**

Model.add(Dropout(0.25))

1. Next is a layer that converts the 2D matrix data to a vector called Flatten. It allows the output to be processed by standard fully connected layers.

Input and output shape equals as follows: **( 12, 12, 64) 2Dmatrix** and **(9216) vector**

model.add(Flatten())

1. Next a fully connected layer with 128 neurons and rectifier activation function.

Input and output shape equals as follows: **( 9216) vector** and **(128) vector**

model.add(Dense( 128, activation=’relu’)

1. Second Dropout layer again implemented with frequency rate 0.5 to reduce overfitting.

Input and output shape equals as follows: **(128) vector** and **(128) vector**

model.add(Dropout(0.5))

1. Finally, the output layer has 10 neurons for the 10 classes, which represents 10 numbers and a softmax activation function to output probability-like predictions for each class.

Input and output shape equals as follows: **(128) vector** and **(10) vector**

model.add(Dense(10, activization = ‘softmax’))

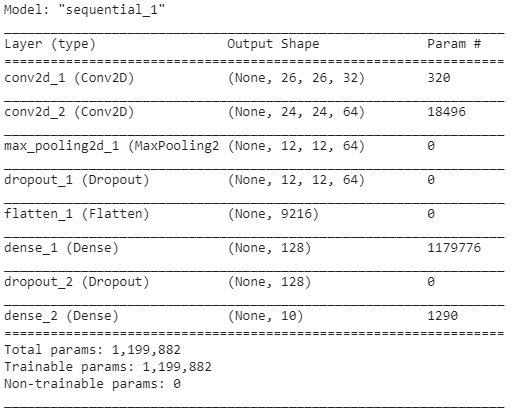


Figure 3. Summary of the given model.

**Fitting**

There are 3 important hyperparameters are presented: batch\_size, epochs, validation\_split

Batch\_size is a number of samples represented as training set, full size of the dataset is too high to neural networks , that is why it should be divided into smaller pieces called batches.

Epochs – is a reference that dataset went thought CNN and optimum number of epochs is vital in fitting of NN, small number can lead to have underfitting and high number will be result of overfitting, in both cases the accuracy of NN will be low.

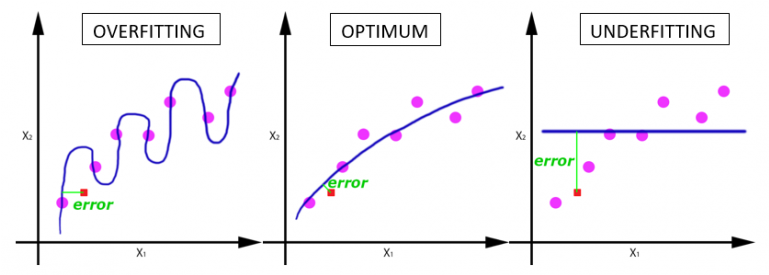


Figure 4.

With the increase in the number of epochs, the weights of the neural network change more and more times. Each time, the curve adjusts better to the data, moving sequentially from a poorly trained state (last graph) to optimal (central graph). If we don’t stop in time, then retraining may occur (the first graph) - when the curve very precisely adjusted to the points, and the generalizing ability disappeared.

Validation\_split – is ratio between training and testing sets, in order to score the accuracy of model it is necessary to have set of data, which will be new for model. Using the same training examples for testing is unlikely to give an accurate representation of the predictive accuracy of the model as the model is likely to be biased towards the training set.

Figure 5 shows the result of the following configuration:

**Batch\_size = 200**

**Epochs = 4**

**Validation\_split = 0.3**

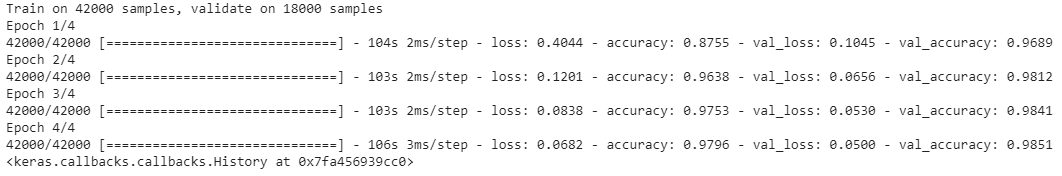


Figure 5. Output of the model fitting

The graph below shows evaluation on the given model in different values of Batch\_size, epochs = 4,

validation\_split = 30%. With the increase of batch\_sizeto 800 the accuracy is slightly dopped at 97.7%, best accuracy 98.6% is obtained at batch\_size = 200.

Graph 1

Number of epochs is vital in model fitting and to obtain best value following evaluations was done with different values of epochs in Graph 2. Best accuracy were obtained at batch\_size = 200, validation\_split = 30% and number of epochs equal to 15.

Graph 2

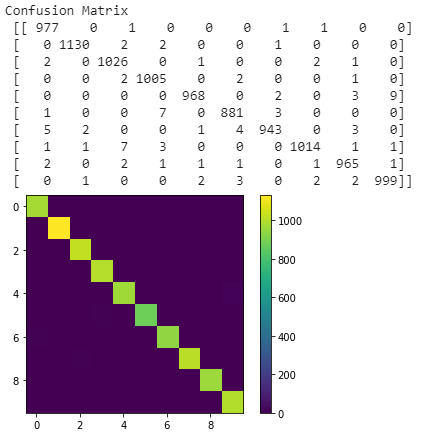


Figure 6.

Confusion matrix – also called error matrix, is a specific table layout that allows us visualize the performance of the model. Figure 6. shows confusion matrix for the given matrix at batch\_size = 200, epochs = 10, validation\_split = 30%.

The architecture of the new model is given by following layers:

model = Sequential()

    model.add(Conv2D(30, (5,5), activation = 'relu', input\_shape = (28,28,1)))

    model.add(MaxPooling2D(pool\_size = (2,2)))

    model.add(Conv2D(15, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size = (2,2)))

    model.add(Dropout(0.2))

    model.add(Flatten())

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

    model.add(Dense(50, activation = 'relu'))

    model.add(Dense(10, activation = 'softmax'))

New model differs from the given model by having extra pooling layer right after first convolution layer with kernel size equal to 5,5 and also one extra fully connected Dense layer was added.

The evaluation of the new model is made at following values of hyperparameters.

Batch\_size = 200

Epochs = 10

Validation\_split = 0.3

Test loss: 0.02782436383238528

Test accuracy: 0.991100013256073

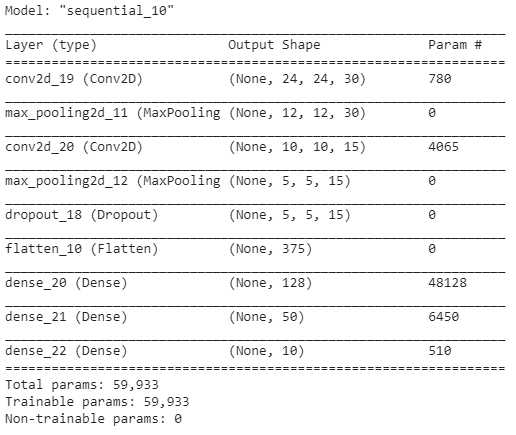


Figure 7. Summary of the new model

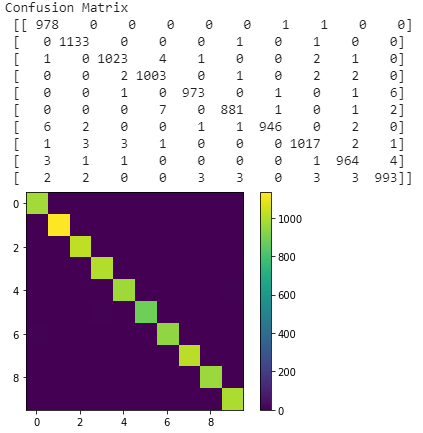


Figure 8. Confusion matrix of the new model

**Conclusion**

For CNN vital role plays hyperparameters as well as the model architecture neural network. In order to calculate the accuracy of model, dataset should be split into training and testing sets, for both models validation split was held at 30% as optimum value, the optimum value of batch\_size and number of epochs were obtained thought several manipulations with these hyperparameters. The performance of the new model is higher than the given model, for same configuration of hyperparameters the accuracy was better 99.1% against 98.6%, moreover average time spent on the fitting of each epoch was also shorter, 26 sec against 105sec for the given model.