HSB
Hochschule Bremen
City University of Applied Sciences

Assignment IV

Image Processing and Pattern Recognition.
Winter Semester 2021-2022

MSc E.E.

Lecturer: Yannik Steiniger

Date of Submission:

1/27/2022

Leknesh Ravinthiran - 5209457
Jose Luis Quinones Gonzalez - 5172885

Contents

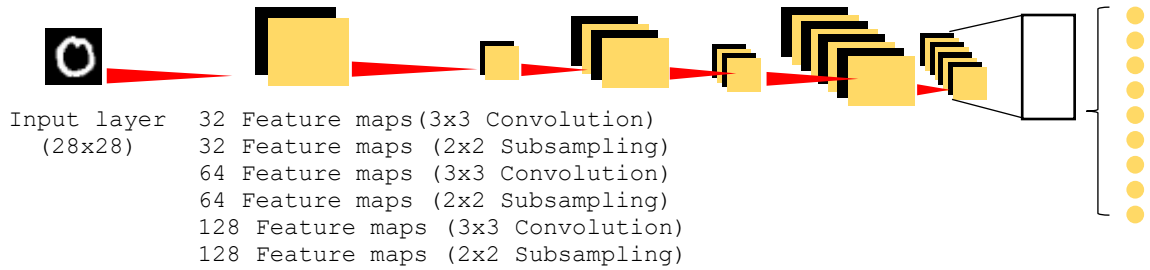
1.Introduction

The following document represents a common and simple use in creating a sequential deep learning architecture. The experiment implements common libriaries, application programming interface (API) as keras, sklearn and tensorflow.

Such API facilities the use and drastically reduces the code when designing/ creating deep learning architectures. Keras provides funtions for deep learning models and uses tensorflow as main backend. [1]

The model in use will be the so called "sequential model" from keras. Its structure is determined by the folling table.

Table 1. Selected sequential model.



Input layer (28x28)	32 Feature maps(3x3 Convolution) 32 Feature maps (2x2 Subsampling) 64 Feature maps (3x3 Convolution) 64 Feature maps (2x2 Subsampling) 128 Feature maps (3x3 Convolution) 128 Feature maps (2x2 Subsampling)	Flatten data: 1 fully connector layer (512 neurons) 1 fully connector layer, output layer (10 neurons)
Input patches	2D Convolution (3,3) Max Pooling (2,2)	

Deep learning architecture

The designed sequential model starts with an input layer, beign the corresponding image in turn. Then it goes through a several feature extraction process. Wherein, first 32 feature maps are extracted, using a 2D convolution (3,3), couple with a max pooling for extracting the max filtered value, as part of a down sampled process, with size (2,2). The process starts with a 32 feature maps, then continues with a 64 and finally with a 128 features maps. At the end of this feature extraction process, the data is flatten into a 512 neurons layer and then this data is connected to a 10 neurons output layer. Each neuron representing a specific category/ class in the experiment (0,1,2,3,4,5,6,7,8,9).

Identifying and as a result, classifying digits is the main task of this experiment. The dataset in use is named, "MNIST", which is a database of handwritten digits. This dataset being a subset of a larger digits' database from the National institute of satandards and technology (NIST). [2]

The activation function which was used in the feauture extraction maps and in the dense layer, was ReLU.

Dense layer is known as the layer which conects the input with the output, for such tranformation to be possible such dense layer is necessary. [3]

The ReLU function is a common choice due to simplicity and good enough performance on a variety of tasks when prediction is the main purpose.

Graphically can be seen as follows:

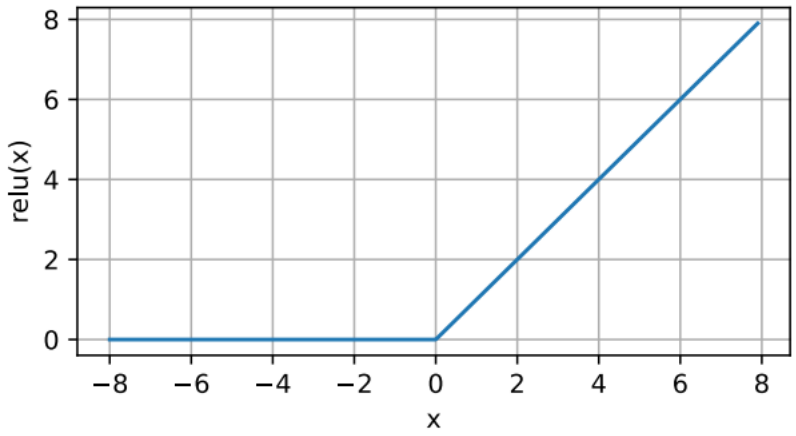


Figure 1. ReLU function. [3]

For an element x, the function is defined by this element's maximum and 0. $ReLU(x) = \max(x, 0)$

This function keeps only positive elements. While negative elements are discarded (setting activations to 0).

When an input is negative, its derivative (applying this function) is 0, as for the positive input, its derivative is 1.

As for optimizing the model, the algorithm "Adam" will be implemented. This optimizer works as to compensate the momentum regarding the current loss, while training. Loss being an essential factor to diminish in any learning model. Accuracy might be in a good value, but if the loss value is notorious then the overall performance of the model may be not that optimal. Adam uses exponential weighted moving averages, as to obtain momentum, couple with the second momentum of gradient. [3]

Adam is expressed by the mathematical function [3]:

Equation 1. Adam expression.

$$v_t \leftarrow \beta_1 v_{t-1} + (1 - \beta_1) g_t,$$

$$s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2) g_t^2,$$

Where v and s represent the first and second momentum, β_1 and β_2 are nonnegative weighting parameters. And g_t, g_t^2 are the corresponding gradients.

Graphically speaking this function can be displayed as:

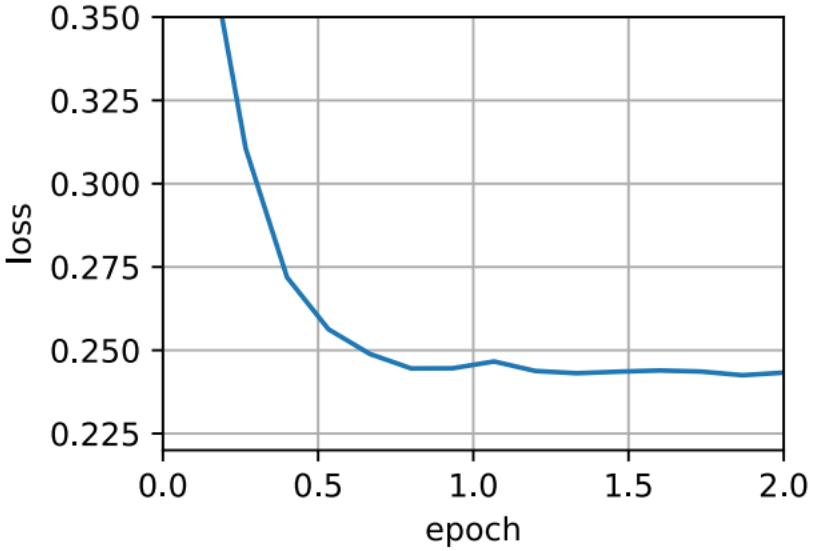
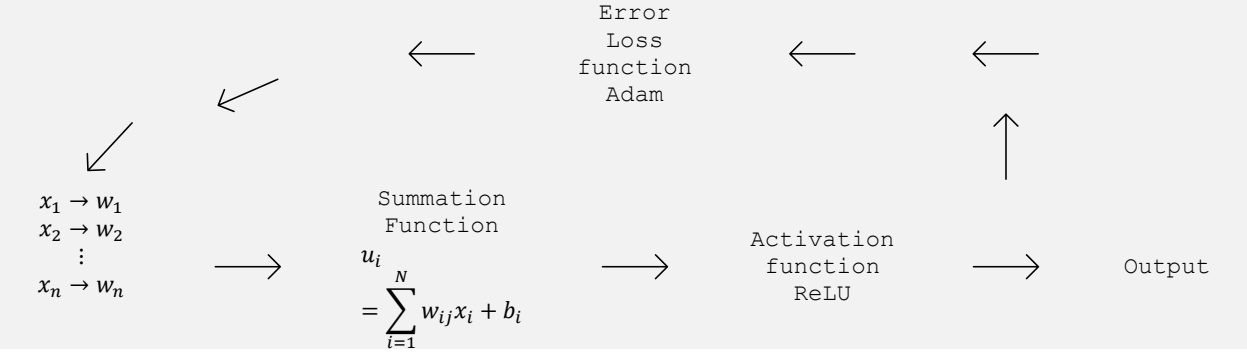


Figure 2. Adam function. [3]

Finally, the model layout can be graphically explained as follows:

Table 2. Implemented learning model.

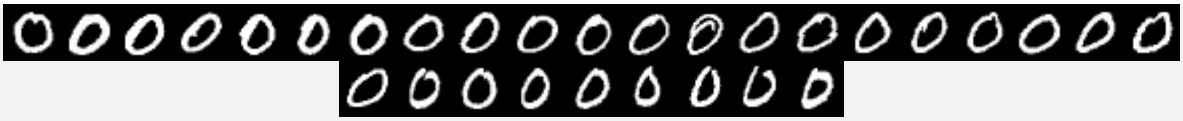


2.Data preparation

For starting the training, the proper "MNIST" data set was downloaded from its original source. "THE MNIST DATABASE of handwritten digits". This dataset is available for implemententing/ putting into practice learning models, or any other type of learning techniques.

It must be clarified that complete use of the enclosed images was not put into practice. For means of analysis regarding the selected designed learning architecture, a total of 1000 digits, 100 per class (0,1,2,3,4,5,6,7,8,9) were selected.

Table 3.Example of Zeros from MNIST dataset



Afterwards, the data was saved within two folders:

Table 4. Preparation of data.

Train	→	0	→	[0]70 images
		1		[1]70 images
		2		[2]70 images
		3		[3]70 images
		4		[4]70 images
		5		[5]70 images
		6		[6]70 images
		7		[7]70 images
		8		[8]70 images
		9		[9]70 images
Test	→	0	→	[0]30 images
		1		[1]30 images
		2		[2]30 images
		3		[3]30 images
		4		[4]30 images
		5		[5]30 images
		6		[6]30 images
		7		[7]30 images
		8		[8]30 images
		9		[9]30 images

The data was splitted 70% in training data, while 30% in testing data.

3.Results

After the data preparation, two programs were developed. One for running and saving the model. And other one for performing the predictions. It is of utmost importance to mention, that along the process for obtaining the best results, best model "configuration", more than one modification was done.

Starting with the "baseline" model without utilizing drop out regularization, data augmentation, padding and filter optimization. The results of the first run are as follows:

Base line
Number of epochs = 8.
Data augmentation = none.
Drop out layer = none.
Filters in layers: 32(layer 1), 64(layer 2), 128 (layer 3)

```
Epoch 8/8
11/11 [==============================] - 1s 126ms/step - loss: 0.0377 - accuracy:
0.9986 - val_loss: 0.4233 - val_accuracy: 0.8833
Confusion Matrix
[[4 4 3 3 4 2 4 0 3 3]
 [6 1 2 1 5 1 5 4 3 2]
 [3 3 3 4 4 4 0 3 2 4]
 [4 0 0 3 3 3 4 6 4 3]
 [1 2 5 1 1 7 1 4 4 4]
 [2 4 3 4 3 1 2 3 3 5]
 [3 4 5 2 4 1 4 0 5 2]
 [3 1 2 5 3 7 3 2 2 2]
 [3 4 2 5 3 2 4 1 3 3]
 [3 6 3 5 3 2 2 2 2 2]]
Classification report
              precision    recall  f1-score   support

           0       0.12      0.13      0.13        30
           1       0.03      0.03      0.03        30
           2       0.11      0.10      0.10        30
           3       0.09      0.10      0.10        30
           4       0.03      0.03      0.03        30
           5       0.03      0.03      0.03        30
           6       0.14      0.13      0.14        30
           7       0.08      0.07      0.07        30
           8       0.10      0.10      0.10        30
           9       0.07      0.07      0.07        30

    accuracy                           0.08       300
   macro avg       0.08      0.08      0.08       300
weighted avg       0.08      0.08      0.08       300
```

As seen in the confusion matrix, most of the values regarding the classes of this experiment were not predicted correctly. A curious fact, the accuracy and loss values are good enough for having a successful prediction. *Hint, the function flow_from_directory had the parameter, shuffle = True.

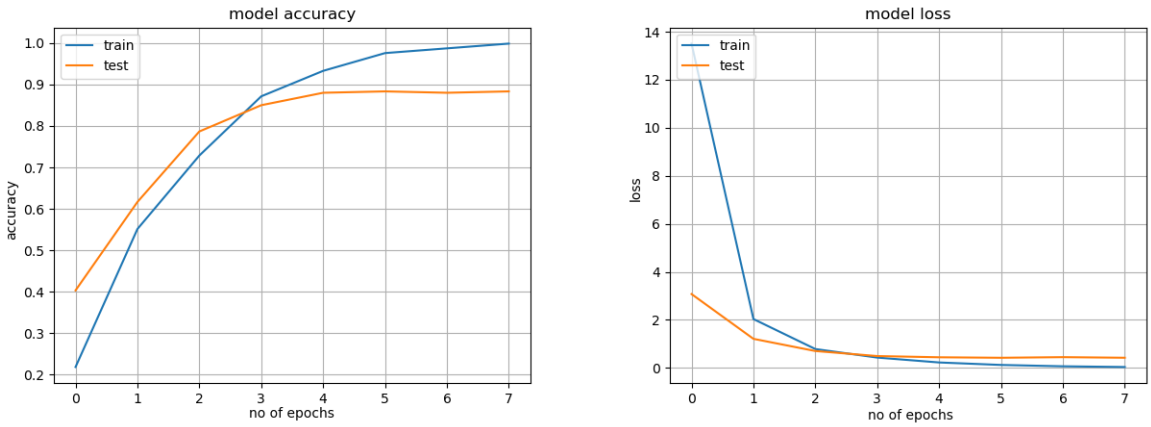


Figure 3.Model baseline, accuracy, and loss.

For the second attempt, a few modifications were done. A drop out layer was implemented at every filter, 32,64,128. The drop out layer is an option which randomly disables some connections from some neurons (in accordance with a certain factor). For this case a factor of 0.2 was used.

Drop out regularization (randomly turns off neural connections)

```
Epoch 8/8
11/11 [==============================] - 1s 48ms/step - loss: 1.1456 -
accuracy: 0.6129 - val_loss: 0.7455 - val_accuracy: 0.7800

Confusion Matrix
[[2 2 3 7 1 2 4 3 2 4]
 [3 3 3 5 4 3 3 1 1 4]
 [5 5 3 3 1 1 1 0 5 6]
 [1 4 2 3 5 2 6 2 1 4]
 [3 2 2 5 4 1 5 1 3 4]
 [6 4 2 2 2 2 1 2 6 3]
 [4 3 3 5 3 0 3 2 2 5]
 [3 0 2 4 3 2 6 5 2 3]
 [5 3 1 3 5 0 3 3 7 0]
 [1 3 4 3 1 5 2 4 3 4]]
Classification report
              precision    recall  f1-score   support

           0       0.06      0.07      0.06        30
           1       0.10      0.10      0.10        30
           2       0.12      0.10      0.11        30
           3       0.07      0.10      0.09        30
           4       0.14      0.13      0.14        30
           5       0.11      0.07      0.08        30
           6       0.09      0.10      0.09        30
           7       0.22      0.17      0.19        30
           8       0.22      0.23      0.23        30
           9       0.11      0.13      0.12        30

    accuracy                           0.12       300
```

```
   macro avg       0.12      0.12      0.12       300
weighted avg       0.12      0.12      0.12       300
```

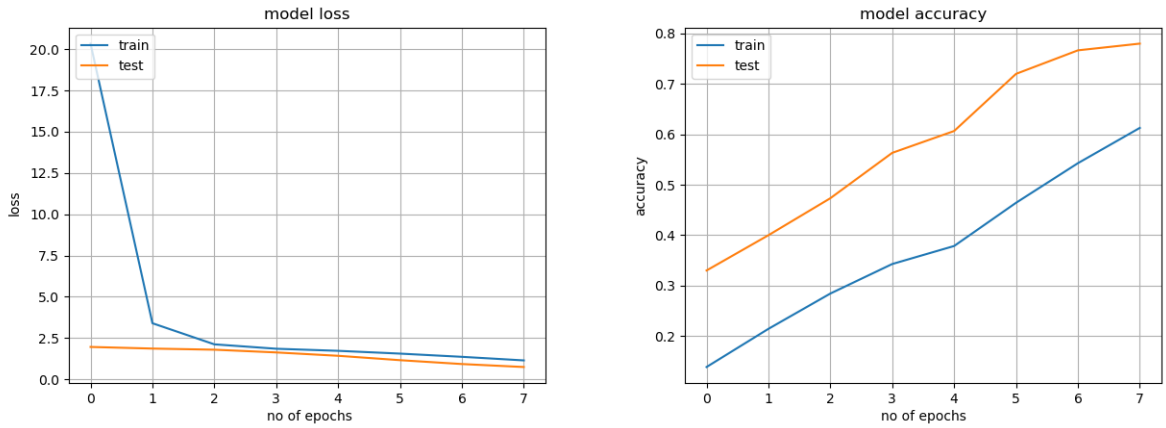


Figure 4. Model with drop out layer (0.2), loss and accuracy.

The last modification within the model before selecting the final model and save it for future predictions, data augmentation couple with padding and increasing filters, (64,128,256) were added.

Data Augmentation (rescale, horizontal_flip, rotation_range, width_shift_range, height_shift_range) + padding + 64,128,256 filters

```
Epoch 8/8
11/11 [==============================] - 2s 163ms/step - loss: 0.8470 - accuracy:
0.7186 - val_loss: 0.6886 - val_accuracy: 0.7700
Confusion Matrix
[[7 5 0 5 1 3 3 2 1 3]
 [3 4 2 8 2 0 3 3 1 4]
 [3 2 3 7 2 1 5 1 2 4]
 [2 4 3 3 2 4 0 1 5 6]
 [1 3 4 6 5 5 4 1 1 0]
 [2 4 3 5 1 1 2 2 3 7]
 [3 5 2 6 4 0 1 4 1 4]
 [5 1 4 4 5 3 1 0 4 3]
 [2 3 2 7 4 2 6 0 1 3]
 [2 5 1 5 3 4 2 3 1 4]]
Classification report
              precision    recall  f1-score   support

           0       0.23      0.23      0.23        30
           1       0.11      0.13      0.12        30
           2       0.12      0.10      0.11        30
           3       0.05      0.10      0.07        30
           4       0.17      0.17      0.17        30
           5       0.04      0.03      0.04        30
           6       0.04      0.03      0.04        30
           7       0.00      0.00      0.00        30
           8       0.05      0.03      0.04        30
           9       0.11      0.13      0.12        30

    accuracy                           0.10       300
   macro avg       0.09      0.10      0.09       300
weighted avg       0.09      0.10      0.09       300
```

For this last attempt in improving the model overall performance, a decrease in loss was seen, nonetheless none of the previous modifications could imitate the same performance from the first, baseline model.

For the final model, a modification in the number of epochs, from 8 to 12 was performed. The padding parameter was also kept, and the major change, perhaps, the shuffle in the flow_from_directory was disabled.

Final model results:

```
Epoch 12/12
11/11 [==============================] - 1s 60ms/step - loss: 0.1140 - accuracy: 0.9743
- val_loss: 0.4001 - val_accuracy: 0.8833
Confusion Matrix
[[29  0  0  0  0  0  1  0  0  0]
 [ 0 29  1  0  0  0  0  0  0  0]
 [ 1  0 27  0  0  0  0  0  2  0]
 [ 0  0  0 20  0  4  1  0  5  0]
 [ 1  0  0  0 28  0  1  0  0  0]
 [ 0  0  0  2  1 26  0  0  0  1]
 [ 0  0  0  0  0  0 30  0  0  0]
 [ 1  0  0  2  1  0  0 24  0  2]
 [ 0  0  1  2  0  0  0  0 27  0]
 [ 0  0  0  4  1  0  0  0  0 25]]
Classification report
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        30
           1       1.00      0.97      0.98        30
           2       0.93      0.90      0.92        30
           3       0.67      0.67      0.67        30
           4       0.90      0.93      0.92        30
           5       0.87      0.87      0.87        30
           6       0.91      1.00      0.95        30
           7       1.00      0.80      0.89        30
           8       0.79      0.90      0.84        30
           9       0.89      0.83      0.86        30

    accuracy                           0.88       300
   macro avg       0.89      0.88      0.88       300
weighted avg       0.89      0.88      0.88       300
```

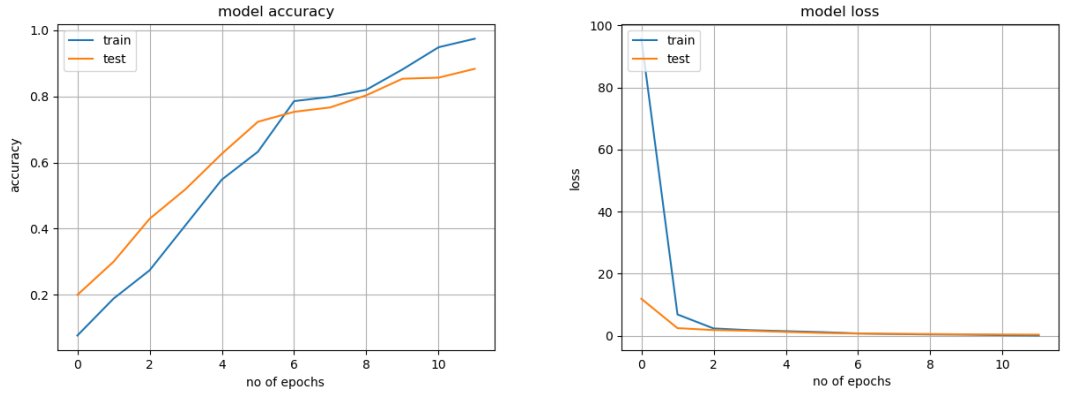


Figure 5. Final model results.

The class **3** was the one which apparently was the most difficult to accurately predict using this model, as one sees the confusion matrix results. After all, the shuffle parameter was the one which prevent the "real" results to be seen and analyzed. During the experimentation of these past parameters, a certain skepticism was presented. At the end it was somehow "eradicated" by the no use of shuffle.
Using the best model to make predictions, these were the results:

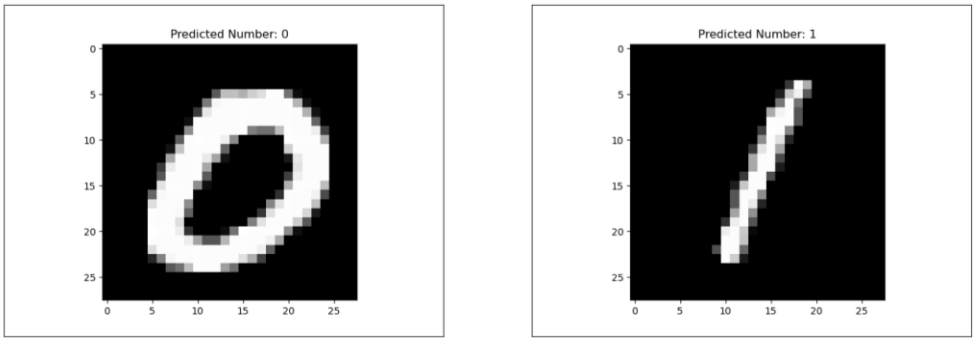


Figure 6. Prediction 1.

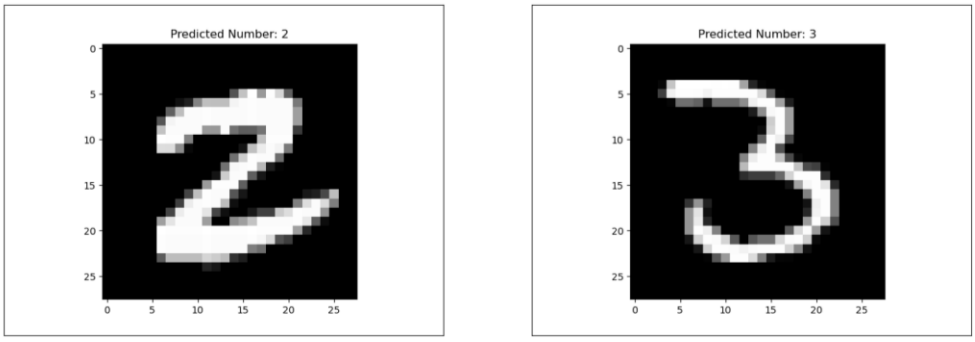


Figure 7. Prediction 2.

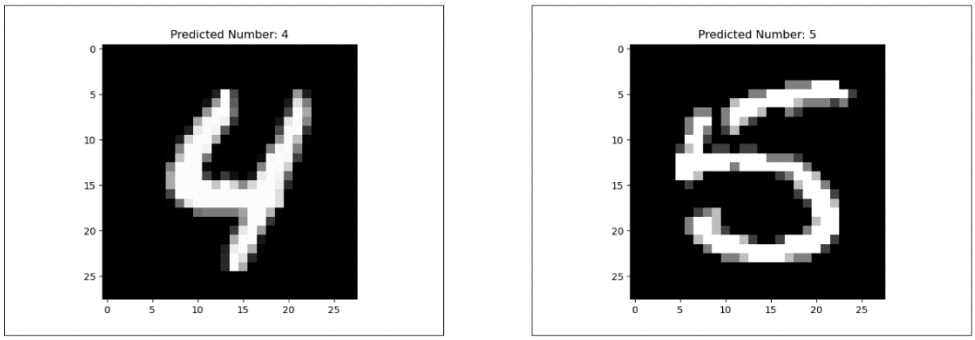


Figure 8. Prediction 3.

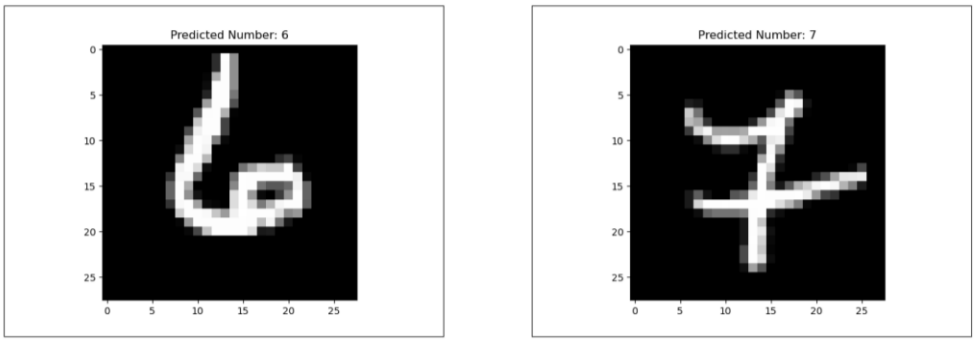


Figure 9. Prediction 4.

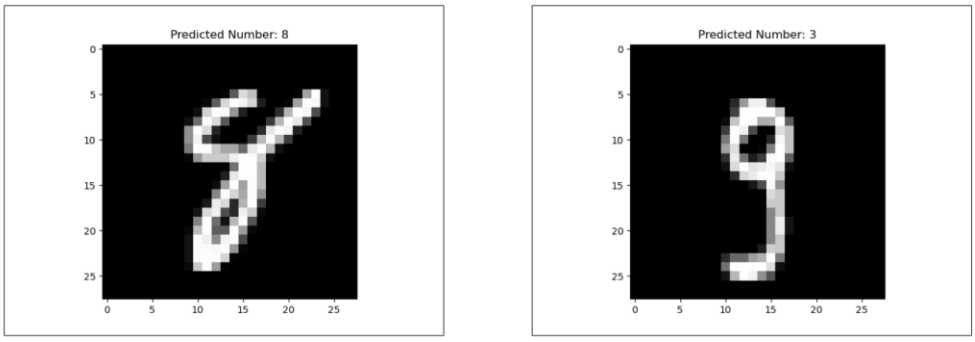


Figure 10. Prediction 5.

4. Conclusion

Throughout the undertaking of this experiment, the main challenge was regarding the number and proper configurations of such learning architecture. Since some literature suggested some methods and examples, at the end, it is still difficult to state that such architecture and configuration may yield the expected "good enough" results. Sometimes even the type of data and application may determine and demand for the experimenter to simply "play around" with the values, until desired results are obtained. Indeed, having a starting point is always advisable.

In brief, this model yielded some interesting results. Predictions could be made, though one of the numbers was not predicted correctly. Considering the loss and accuracy value, is understandable that such behavior may be not perfect. The model could err, but it is not prone to have such poor predictions regularly. In this case was one out of ten values which were mistaken. Yet, overall performance was decent, and one can assume that such model was designed properly for the application of hand-writing digits classification.

References

[1] keras, "Keras," [Online]. Available: https://keras.io/about/.

[2] C. C. C. J. B. Yann LeCun, "The MNIST database".

[3] Z. C. L. M. L. a. A. J. S. Aston Zhang, Dive into Deep Learning.

Index

Python's Code

```
"""
Created on Sat Jan 22 13:19:00 2022

@author: jlqgj
"""


from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt


#construct an instance of ImageDataGenerator class
train_datagen = ImageDataGenerator(rescale=1)
test_datagen = ImageDataGenerator(rescale=1)

#defining the sequential model for multiclass classification
model = keras.Sequential()

##########################Feature extraction
#32 features map, kernel size 3,3 (convolution), kerne size 2,2 (pooling),
model.add(layers.Conv2D(32, (3,3),
activation='relu',input_shape=(28,28,3),padding='same'))
model.add(layers.MaxPool2D((2,2)))


#64 features map, kernel size 3,3 (convolution), kerne size 2,2 (pooling),
model.add(layers.Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(layers.MaxPool2D((2,2)))


#128 features map, kernel size 3,3 (convolution), kerne size 2,2 (pooling),
model.add(layers.Conv2D(128,(3,3),activation='relu',padding='same'))
```

```
model.add(layers.MaxPool2D((2,2)))



#########################End of feature extraction


model.add(layers.Flatten())
#512 neurons
model.add(layers.Dense(512,activation='relu'))


#Final layer, Number of neurons, one per class (multiclass classification)
model.add(layers.Dense(10,activation='softmax'))


#prepare iterators
training_iterator =
train_datagen.flow_from_directory('C:/Users/jlqgj/programs/CNN_digits_model/digits/tra
in',

batch_size=64,target_size=(28,28),shuffle=False)
testing_iterator =
test_datagen.flow_from_directory('C:/Users/jlqgj/programs/CNN_digits_model/digits/test
',

batch_size=64,target_size=(28,28),shuffle=False)


batch_size = 64
num_of_train_samples = 70
num_of_test_samples = 30

#compile model
model.compile(loss = 'categorical_crossentropy', metrics=['accuracy'],
optimizer='adam')

#fit model
history = model.fit(training_iterator, validation_data=testing_iterator, epochs=12)

# confusion matrix
Y_pred = model.predict(testing_iterator,num_of_test_samples//batch_size)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(testing_iterator.classes, y_pred))
print('Classification report')
target_names = ['0','1','2','3','4','5','6','7','8','9']
print(classification_report(testing_iterator.classes, y_pred,
target_names=target_names))

#Plotting of results
import matplotlib.pyplot as plt

#loss vs epochs
fig, ax = plt.subplots(1)
ax.plot(history.history['loss'])
ax.plot(history.history['val_loss'])
plt.title('model loss')
```

```
plt.xlabel('loss')
plt.ylabel('no of epochs')
plt.legend(['train','test'],loc='upper left')
plt.grid(True)
fig.show()

#accuracy vs epochs
fig2, ax2 = plt.subplots(1)
ax2.plot(history.history['accuracy'])
ax2.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.xlabel('accuracy')
plt.ylabel('no of epochs')
plt.legend(['train','test'],loc='upper left')
plt.grid(True)
fig.show()


#save the model
model.save('models/digits_with_shuffle.h5')

#get class names
class_labels = testing_iterator.class_indices
print(class_labels)
train_labels = class_labels


#dictionary
# {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9}
```

Program for Predictions

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
import glob

#load saved model
model = load_model('models/digits_with_shuffle.h5')

#load sample image for prediction

img = load_img('samples/57477.png', target_size=(28,28))


#image to array and add dimesion
samples = img_to_array(img)
samples = samples.reshape(1,28,28,3)


#prediction
results = model.predict(samples)
```

```
#class labels
class_labels = {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8,
'9': 9}

#prediction results
print(results)

#print max probability
results = np.argmax(results)
print('Predicted number: ', results)

#Indexing class to predicted result
prediction = [key for key in class_labels][results]
print(prediction)
print(type(prediction))

#plotting results
plt.imshow(img)
plt.title('Predicted Number: '+ prediction)
plt.show()
```