SRH University Heidelberg

Professional Technical Report

TASK – 1

Classification of Handwritten Digits based on the Convolutional Neural Network

Author:	Abhijith Pavithran
Matriculation Number:	11018118
Email Address:	11018118@stud.hochschule-heidelberg.de
Supervisor:	Prof. Dr. Milan Gnjatovic
Project Submission Date:	07-12-2022

Table of Contents

Table of Figures

ABSTRACT

In this project, a program for the classification of handwritten digits based on a convolutional neural network approach is explained. In order to implement the program jupyter notebook is used. For training and testing the classifier the program used MNIST dataset. It implemented the training procedure by importing convolutional neural network model and Keras, a Python library for intensive computation of neural nodes that is supported by the Tensor Flow, with all pixels (attributes) of all classes (digits) from the MNIST training dataset. The testing procedure used MNIST test dataset and could correctly predict the digits.

INTRODUCTION

Handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can vary from person to person. Handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

The MNIST dataset contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains grayscale pixel value. In this project uses the MNIST dataset and create a simple neural network using TensorFlow and Keras

If we want to apply a neural network to image classification, it would be convenient to represent a digital image as a vector that can be presented to the input layer. A digital image represented by a two-dimensional matrix a $[M\times N]$ of integers can be alternatively represented by a vector v of dimension $M \cdot N$. However, applying this vector directly to a fully connected feedforward neural network would not be an appropriate solution. The main problems can be summarized as follows:

- The number of neurons in the input layer of a given neural network is constant. On the other hand, input image may vary in size, which implies that vector representing a given input will also vary in size.
- Even if all input images were of constant size, the generated neural network would not be necessarily robust to even small translations of the input image (e.g., shifting pixels of a training set image by a small amount in the same direction).
- The number of parameters in such a neural network would be significant. E.g., each neuron in the first hidden layer is assigned $M \cdot N + 1$ parameters (i.e., $M \cdot N$ weights for each pixel of the input image and a bias)

To overcome these problems, classification of images is performed by convolutional neural networks, a special type of neural network for processing data with grid-data topology that has three components.

CONVOLUTIONAL NEURAL NETWORK

A feedforward neural network is multilayer network of neurons in which the outputs from neurons in each layer are fed to neurons in the next layer, i.e., information is fed forward, there are no cycles. In general, we differentiate between three kinds of neuron layers: a neural network contains one input layer, one or more hidden layers, and one output layer.

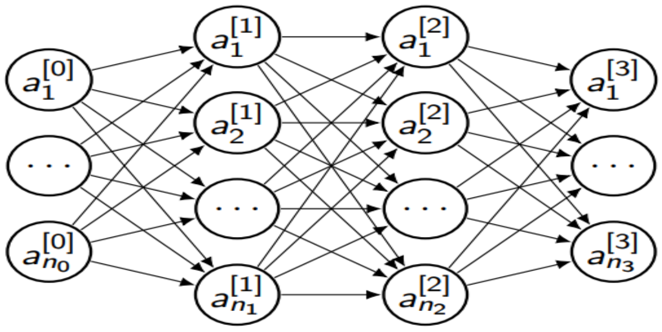


Figure 1 : Illustration of feedforward neural network.

A neuron is the building block of a neural network:

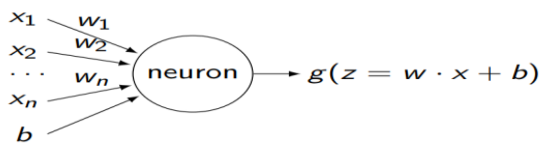


Figure 2 : Illustration of a neuron.

n inputs x ≡ x1, x2, . . . , xn , (1)

n weights: w ≡ w1,w2, . . . ,wn , (2)

Assigned respectively to inputs (1), i.e., each input xi is assigned a weight wi , a bias b.

Its output is defined as:

$$g(z = w \cdot x + b) , \quad (3)$$

where: g is an activation function.

The choice of activation function is not arbitrary, since it affects the performance of a whole neural network. The main idea underlying the activation function is that small changes in the weights and the bias produce a small change in a neuron's output. In this project, ReLU (Rectified Linear Unit) are often used as activation functions for hidden layers in the detector stage, and softmax probability functions for the output layer. The number of neurons in the output layer may be equal to the number of classes.

A typical convolutional neural network layer consists of three components:

Convolutional stage: A filter is convolved with the input image and the bias term is added. The filter is usually of dimension 3×3 and its elements are estimated by means of the backpropagation algorithm. The resulting matrix is referred to as the feature map.

Detector stage: The feature map is passed through a nonlinear activation function ReLU (Rectified Linear Unit)

Poling stage: An additional filter is convolved with the feature map obtained in the previous stage. It should be noted that this filter is slid over the feature map without overlapping with itself. In this stage Max pooling function is applied, which returns a maximum value in a rectangular neighborhood covered by the filter. It extract sharp and smooth features. It also reduces variance and computations. Max-pooling helps in extracting low-level features like edges, points, etc.

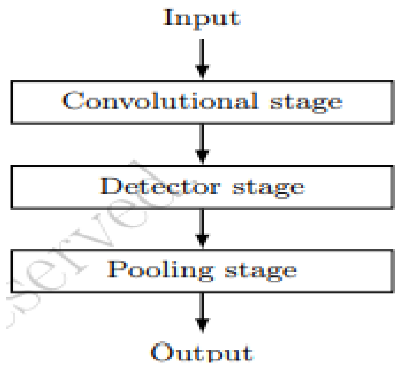


Figure 3 : Formation convolutional neural layer.

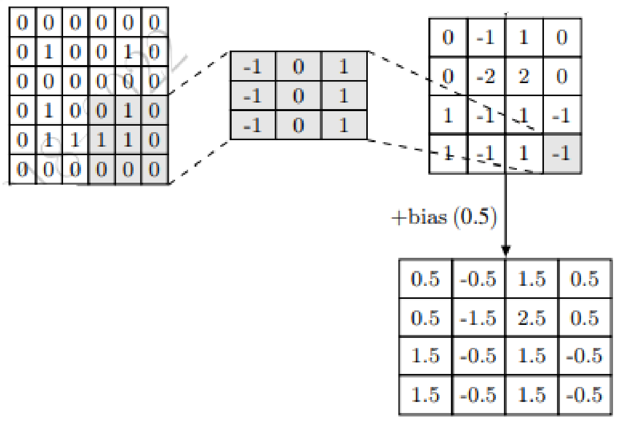


Figure 4 : Formation of convolutional stage.

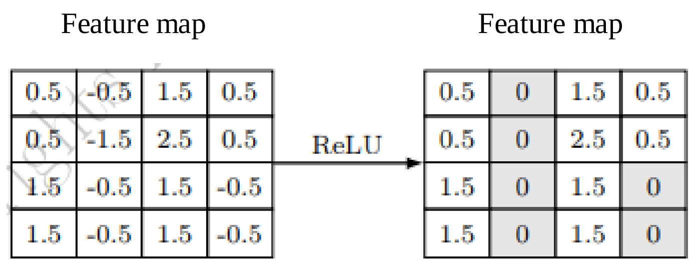


Figure 6 : Formation of detector stage.

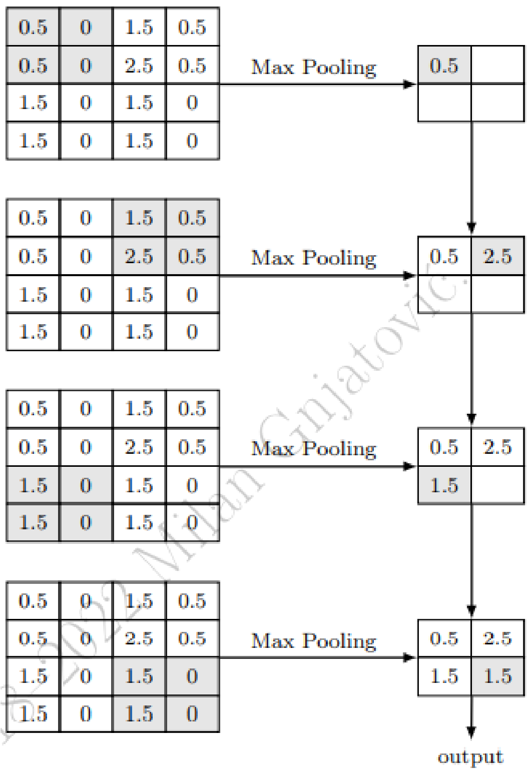


Figure 7 : Formation of pooling stage.

Cost Function

To train and evaluate a neural network, we should select a cost function that quantifies the error between the expected and actual outputs. In this project "sparse categorical cross entropy" used as a cost function for multi-class classification model where the output label is assigned integer value (0, 1, 2, 3…9). It classifies the data by predicting the probability of whether the data belongs to one class or the other class.

To minimize the cost function, we can apply the gradient descent algorithm.

Dropout

In each iteration, a different incomplete neural network is considered. Each of these incomplete neural networks may overfit the training data. The the dropout procedure averages the effects of these networks and thus reduce the overfitting.

FLOW CHART

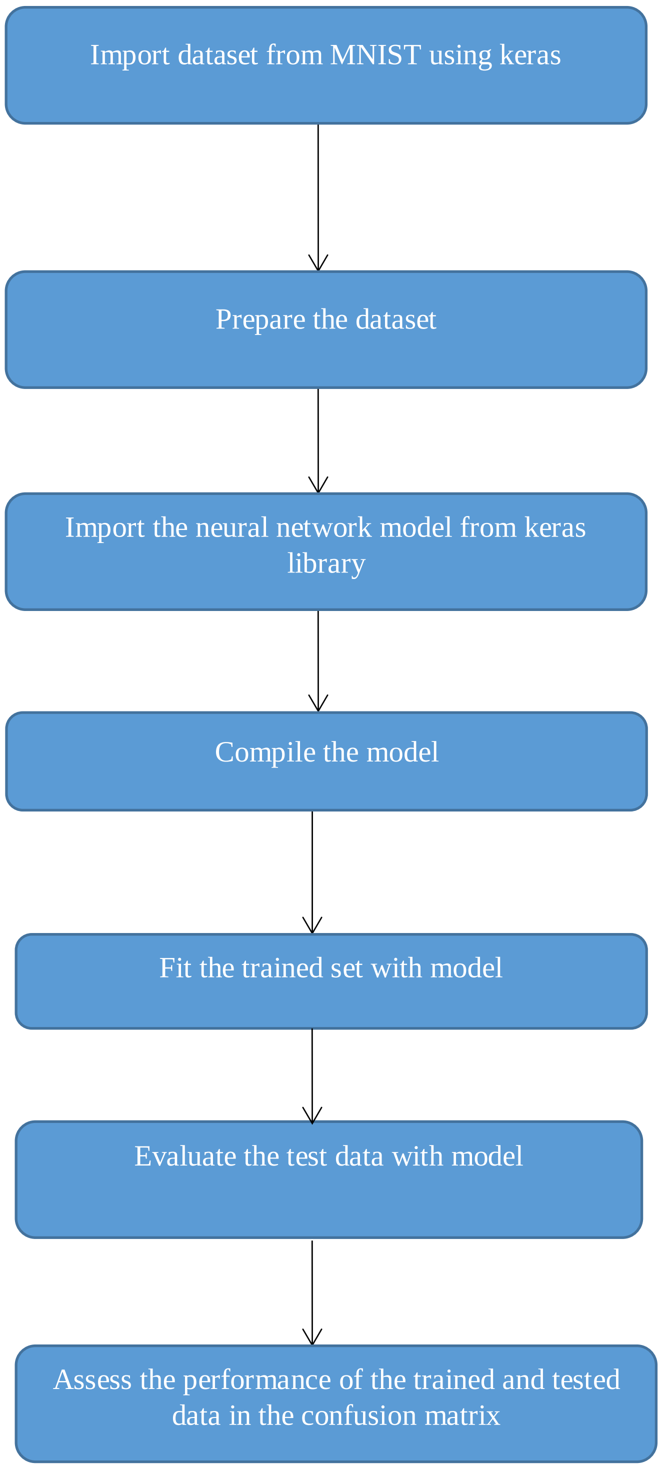
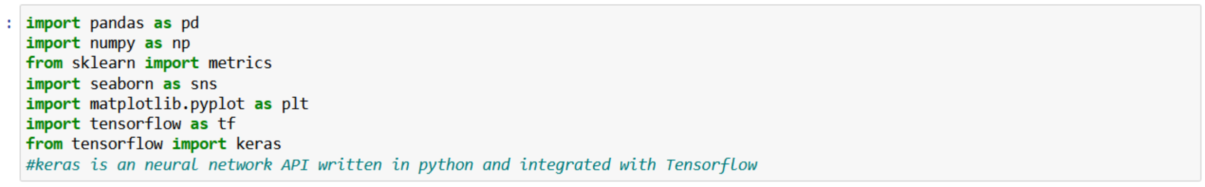

Import dataset from MNIST using keras
Prepare the dataset
Import the neural network model from keras library
Compile the model
Fit the trained set with model
Evaluate the test data with model
Assess the performance of the trained and tested data in the confusion matrix

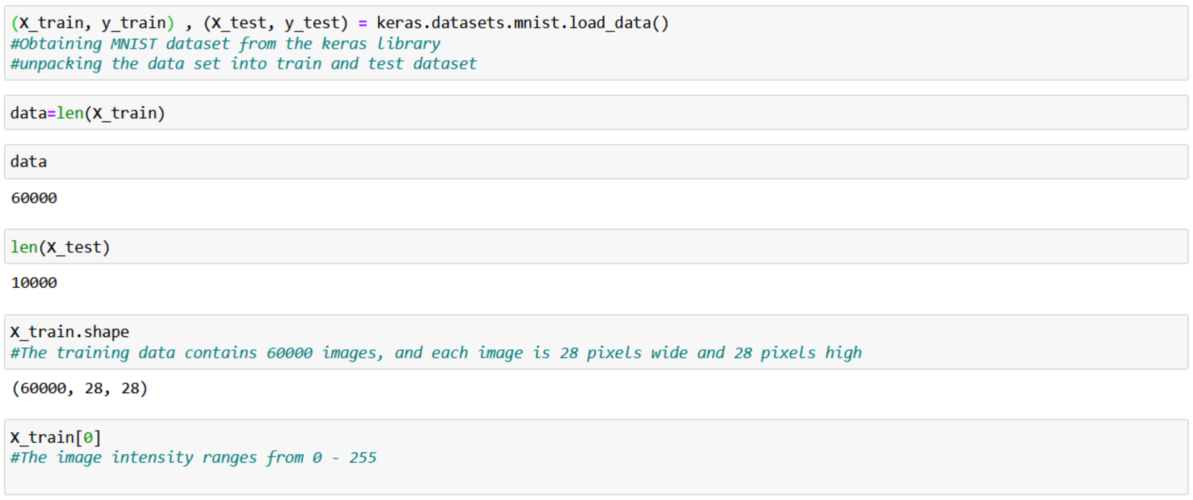
Figure 8: Flow chart

RESULT



```
: import pandas as pd
import numpy as np
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
#keras is an neural network API written in python and integrated with Tensorflow
```

Figure 9 : Importing python libraries



```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
#Obtaining MNIST dataset from the keras library
#unpacking the data set into train and test dataset
```

```
data=len(X_train)
```

```
data
```

```
60000
```

```
len(X_test)
```

```
10000
```

```
X_train.shape
#The training data contains 60000 images, and each image is 28 pixels wide and 28 pixels high
```

```
(60000, 28, 28)
```

```
X_train[0]
#The image intensity ranges from 0 - 255
```

Figure 10 : Importing MNIST dataset using keras library .

```
plt.matshow(X_train[22])
# The image in the training dataset is colour image
```

```
<matplotlib.image.AxesImage at 0x1ace40ddf10>
```

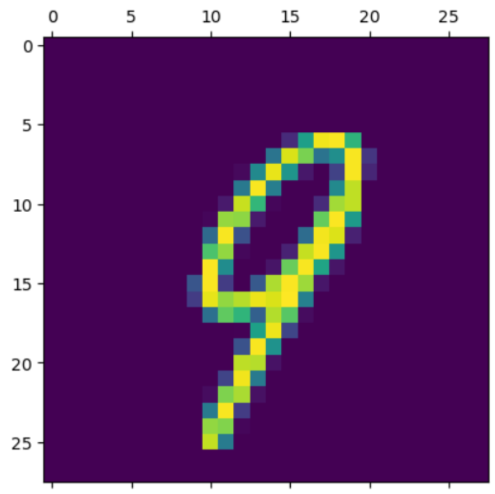


Figure 11 : Illustration of colour image in the training dataset

```
plt.matshow(X_train[22], cmap = plt.cm.binary)
#Convert the dataset colour image into greyscale using matplotlibs build in colourmap
```

```
<matplotlib.image.AxesImage at 0x1acb2623a30>
```

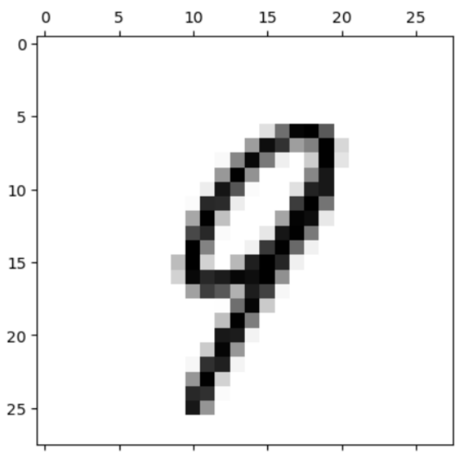


Figure 12 : converting the colour image into inverted greyscale image

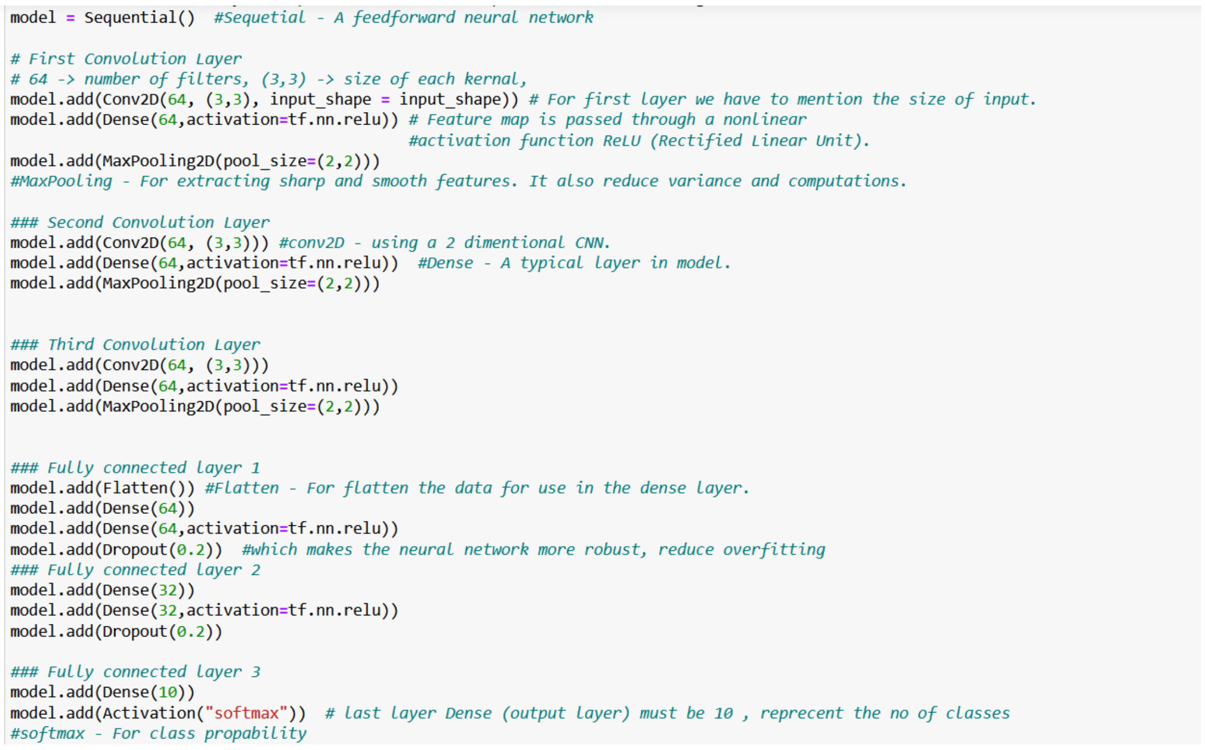
```
X_train = X_train / 255
X_test = X_test / 255
#Normalisation, then the image intensity ranges from 0 - 1
```

```
X_train=X_train.reshape(X_train.shape[0],28,28,1)
X_test=X_test.reshape(X_test.shape[0],28,28,1)
input_shape=(28,28,1)
#Resizing image for convolution , provide 4D array as input to the CNN, 1 additional dimention for kernel operation
```

```
X_train[0]
```

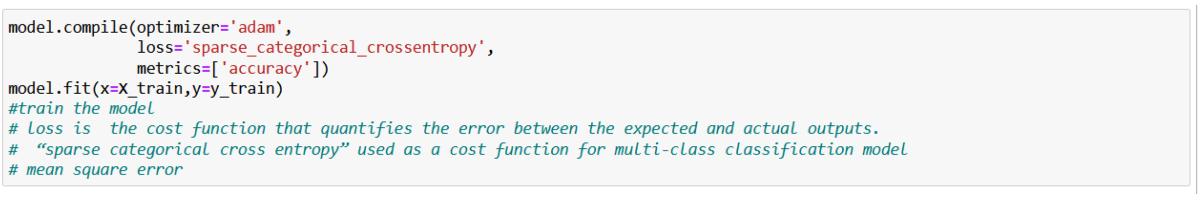
```
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.31372549],
        [0.61176471],
        [0.41960784],
        [0.99215686],
        [0.99215686],
        [0.80392157],
        [0.04313725],
        [0.        ],
        [0.16862745],
        [0.60392157],
        [0.        ],
```

Figure 13 : Normalization and reshaping of datasets



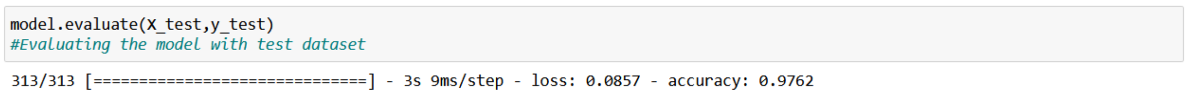
```
model = Sequential()  #Sequetial - A feedforward neural network

# First Convolution Layer
# 64 -> number of filters, (3,3) -> size of each kernal,
model.add(Conv2D(64, (3,3), input_shape = input_shape)) # For first layer we have to mention the size of input.
model.add(Dense(64,activation=tf.nn.relu)) # Feature map is passed through a nonlinear
                                            #activation function ReLU (Rectified Linear Unit).
model.add(MaxPooling2D(pool_size=(2,2)))
#MaxPooling - For extracting sharp and smooth features. It also reduce variance and computations.

### Second Convolution Layer
model.add(Conv2D(64, (3,3))) #conv2D - using a 2 dimentional CNN.
model.add(Dense(64,activation=tf.nn.relu))  #Dense - A typical layer in model.
model.add(MaxPooling2D(pool_size=(2,2)))


### Third Convolution Layer
model.add(Conv2D(64, (3,3)))
model.add(Dense(64,activation=tf.nn.relu))
model.add(MaxPooling2D(pool_size=(2,2)))


### Fully connected layer 1
model.add(Flatten()) #Flatten - For flatten the data for use in the dense layer.
model.add(Dense(64))
model.add(Dense(64,activation=tf.nn.relu))
model.add(Dropout(0.2))  #which makes the neural network more robust, reduce overfitting
### Fully connected layer 2
model.add(Dense(32))
model.add(Dense(32,activation=tf.nn.relu))
model.add(Dropout(0.2))

### Fully connected layer 3
model.add(Dense(10))
model.add(Activation("softmax"))  # last layer Dense (output layer) must be 10 , reprecent the no of classes
#softmax - For class propability
```

Figure 14 : Create the convolutional neural network model and add required layers using tensor flow.



```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=X_train,y=y_train)
#train the model
# loss is  the cost function that quantifies the error between the expected and actual outputs.
#  “sparse categorical cross entropy” used as a cost function for multi-class classification model
# mean square error
```

Figure 15: Compile and fit the model with the training dataset.



```
model.evaluate(X_test,y_test)
#Evaluating the model with test dataset
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0857 - accuracy: 0.9762
```

Figure 16: Evaluate the model with the test dataset.

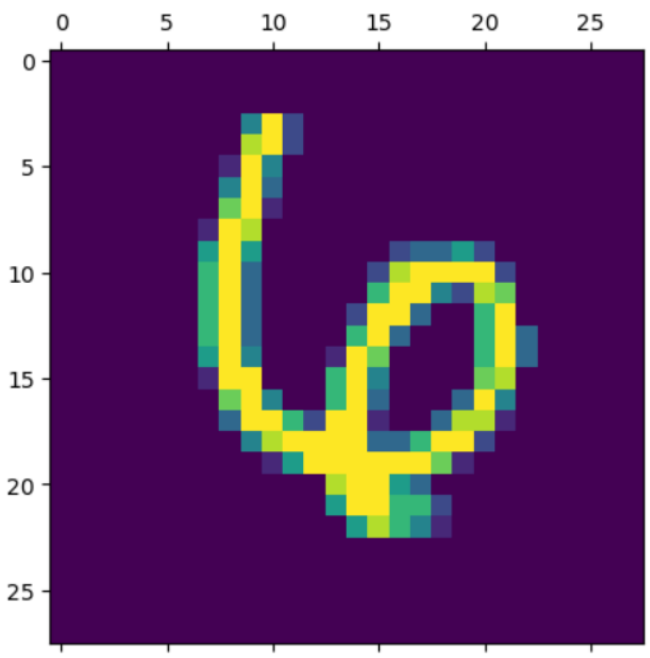
```
y_predicted = model.predict(X_test)  # prediction based on softmax propability
y_predicted[22]
```

```
313/313 [==============================] - 3s 9ms/step

array([1.20806884e-08, 1.93217486e-09, 4.90795912e-07, 3.08749613e-08,
       2.27810392e-06, 3.44731848e-06, 9.99984145e-01, 1.25200515e-07,
       9.53676044e-06, 5.58781217e-08], dtype=float32)
```

Figure 17 : Prediction of Handwritten digits in test dataset based on softmax probability.

```
plt.matshow(X_test[22]) # check data
```

```
<matplotlib.image.AxesImage at 0x21dd35b8340>
```



```
np.argmax(y_predicted[22]) #prediction is same as actual value
```

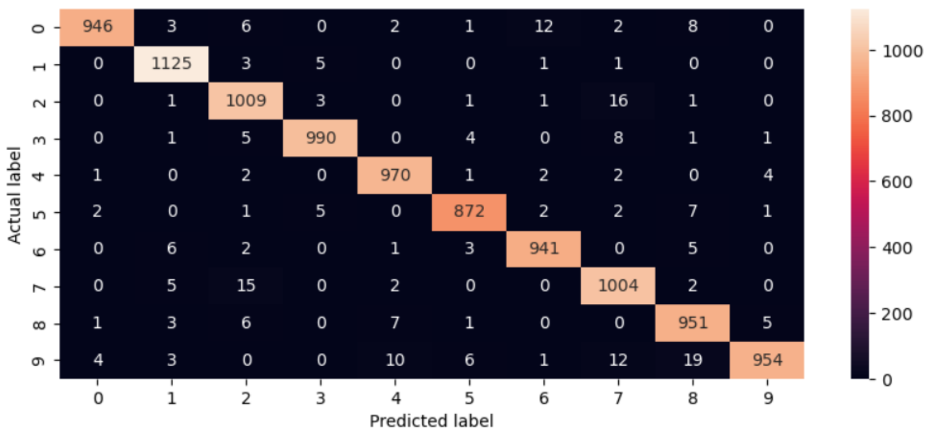
```
6
```

Figure 18 : Illustration of the predicted value is same as the actual value.

```
y_predicted_labels = [np.argmax(i) for i in y_predicted]
```

```
plt.figure(figsize=(10,4))
y_act = pd.Series(y_test, name = 'Actual')
y_pred = pd.Series(y_predicted_labels, name = 'Predicted')
Confusion_matrix = pd.crosstab(y_act, y_pred)
ax = sns.heatmap(Confusion_matrix, annot=True, fmt = "d")
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 14.722222222222216, 'Predicted label')
```



Actual label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	946	3	6	0	2	1	12	2	8	0
1	0	1125	3	5	0	0	1	1	0	0
2	0	1	1009	3	0	1	1	16	1	0
3	0	1	5	990	0	4	0	8	1	1
4	1	0	2	0	970	1	2	2	0	4
5	2	0	1	5	0	872	2	2	7	1
6	0	6	2	0	1	3	941	0	5	0
7	0	5	15	0	2	0	0	1004	2	0
8	1	3	6	0	7	1	0	0	951	5
9	4	3	0	0	10	6	1	12	19	954

Figure 19 : Formation of confusion matrix

```
np.diag(Confusion_matrix) # diagnol element for the confusion matrix
```

```
array([ 959, 1131, 1007,  974,  960,  847,  918,  975,  956,  986],
      dtype=int64)
```

```
Confusion_matrix.sum()  #  calculating the predictions correct predictions
```

```
Predicted
0     976
1    1154
2    1069
3     982
4     978
5     857
6     926
7    1021
8    1015
9    1022
dtype: int64
```

```
Confusion_matrix.sum().sum() # total sum of testing images
```

```
10000
```

```
accuracy = np.diag(Confusion_matrix).sum()/Confusion_matrix.sum().sum()
accuracy
# overoll accuracy from confusion matrix
```

```
0.9713
```

Figure 20 : Accuracy obtained from confusion matrix.

CONCLUSION

In this project, a program for classification of handwritten digits based on the Convolutional neural network approach is successfully realized. The program is written using jupyter notebook. The classifier is able to classify handwritten digits, the accuracy seems high (97.13%).

REFERENCE

- http://gnjatovic.info/misc/feedforward.neural.networks.pdf
- http://gnjatovic.info/machinelearning/
- https://vitalflux.com/keras-categorical-cross-entropy-loss-function/#:~:text=sparse_categorical_crossentropy%3A%20Used%20as%20a%20loss,just%20has%20a%20different%20interface.
- https://github.com/AnikeshChowdhury/Hand-Written-Digit-Classification/blob/main/Hand_Written_Digit_Classification_with_CNN.ipynb