
---

**Project Name:-** Characters Recognition(Digits)

---

**Name:-** PANKAJ KUMAR

---

**Section:-** KM046 **Roll Number:-** 57

---

**Registration Number:- 11910254**

---

**Course Name:-** INT(246)

---

**Class Teacher Name:-** ISHAN KUMAR

---

**University Name:-** LOVELY PROFESSIONAL UNIVERSITY

---

# INTRODUCTION

Handwriting recognition is the ability of a machine to receive and interpret handwritten input from multiple sources like paper documents, photographs, touch screen devices etc. The main aim of this project is to design expert system for, “Handwritten Character (Digit) Recognition using Neural Network” that can effectively recognize a particular character of type format using the Convolutional Neural Network (CNN) approach. Handwriting recognition has been one of the active and challenging research areas in the field of image processing and pattern recognition. It has numerous applications which include, reading aid for blind, bank cheques and conversion of any handwritten document into structural digit form.

# TECHNOLOGY USED

This project is based on neural networks and for this purpose Python is used. The libraries used in this project are:

(i) Tensor flow

(ii) Keras

(iii) Pandas

(iv) Numpy

(v) Seaborn

(vi) Matplotlib

# TENSORFLOW

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow is mainly used for: Classification, Perception, Understanding, Discovering, Prediction and Creation.

# KERAS

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML. Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano. Keras allows users to productize deep models. Keras give some convenient API to use.

# PANDAS

Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, and Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

# NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

# SEABORN

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

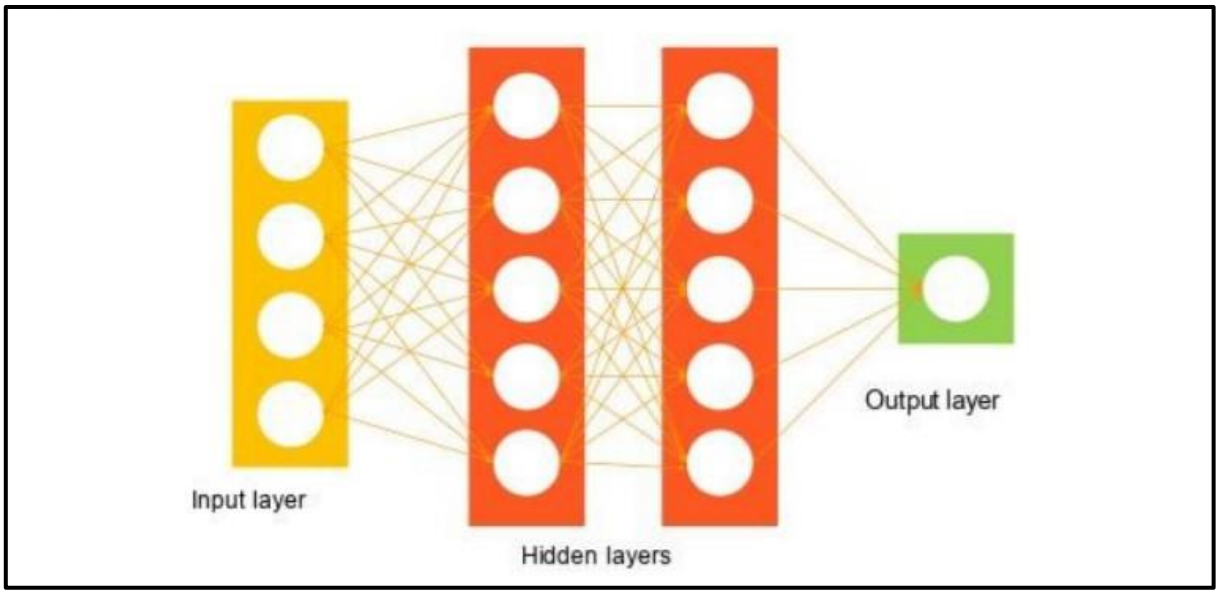
# MATPLOTLIB

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

# NEURAL NETWORK

Neural networks are computing systems with interconnected nodes that work much like neurons in the human brain. Using algorithms, they can recognize hidden patterns and correlations in raw data, cluster and classify it, and – over time – continuously learn and improve. A neural network has many layers. Each layer performs a specific function, and the complex the network is, the more the layers are. That's why a neural network is also called a multi-layer perceptron.

The purest form of a neural network has three layers:-



**(1) The input layer: -**

The input layer picks up the input signals and transfers them to the next layer. It gathers the data from the outside world.

**(2) The hidden layer:-**

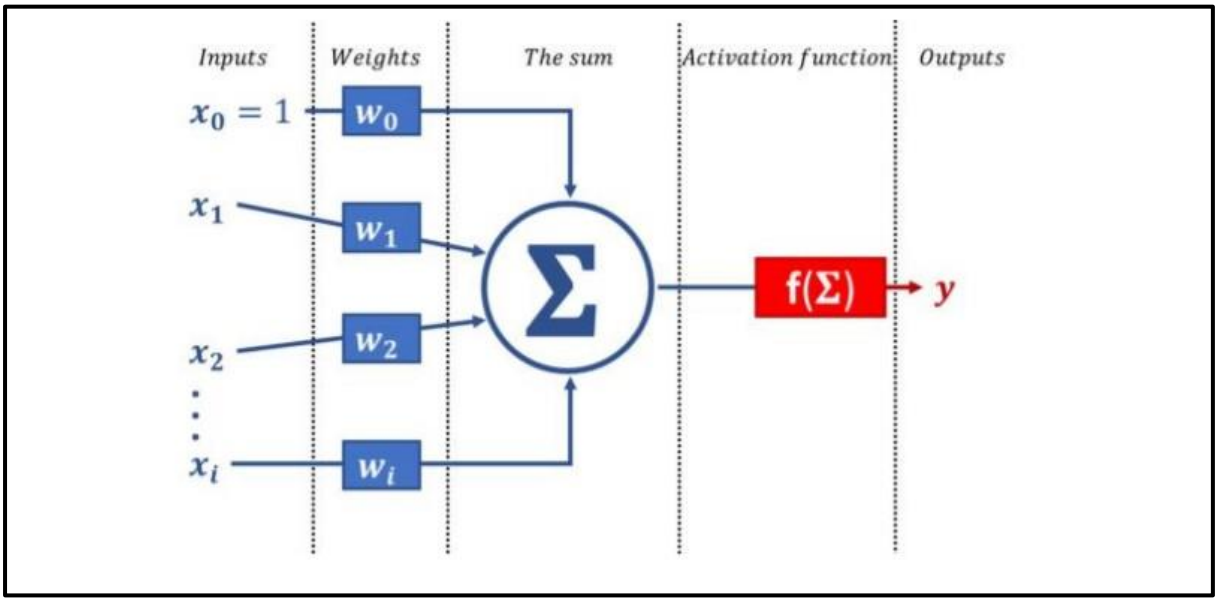
There can be multiple hidden layers in a neural network according to the requirements. The hidden layer performs all the back-end tasks of calculation. A network can even have zero hidden layers. However, a neural network has at least one hidden layer.

**(3) The output layer:-**

The output layer transmits the final result of the hidden layer's calculation. There can be single or multiple nodes in the output layer. If we have a binary classification problem the output node is 1 but in the case of multi-class classification, the output nodes can be more than 1.

# NEURAL NETWORK WORKING

A neuron takes input, performs weighted sum of all inputs, applies activation function to it and gives the output. As shown in following figure.

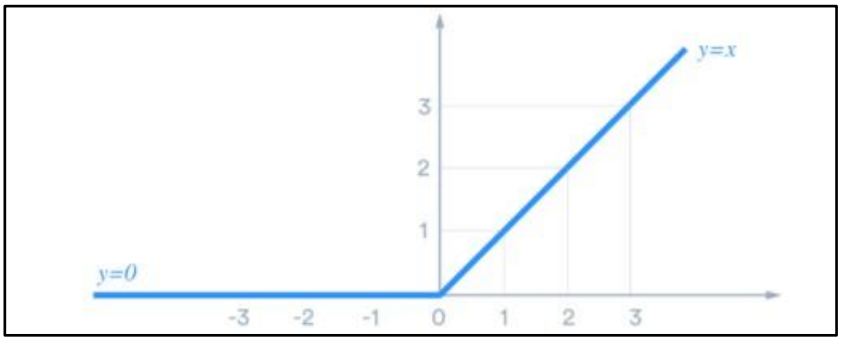



# ACTIVATION FUNCTIONS

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model. In this project work, two activation functions namely, ReLU and Sigmoid are used which can be described as:

# RELU

ReLU stands for rectified linear activation unit and is considered one of the few milestones in the deep learning revolution. ReLU is a non-linear activation function that produces identity output for positive sums. For negative values, its output is zero. If x is weighted sum of neuron then ReLU activation function can be described as – f(x) = max (0, x) Thus it gives an output that has a range from 0 to infinity. Graphical representation of this activation function looks like following diagram.




# SIGMOID

This function takes any real value as input and outputs values in the range of 0 to 1.

The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below.

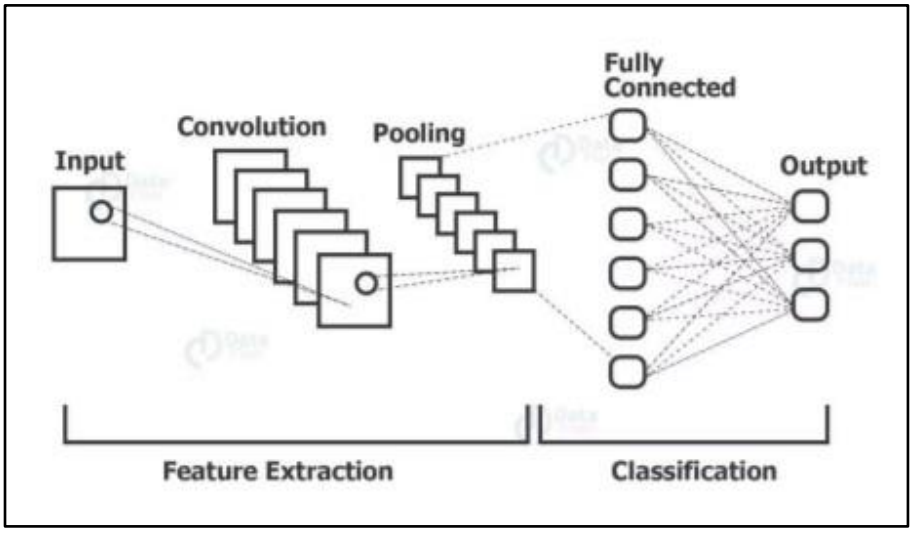
Here's why sigmoid/logistic activation function is one of the most widely used functions:

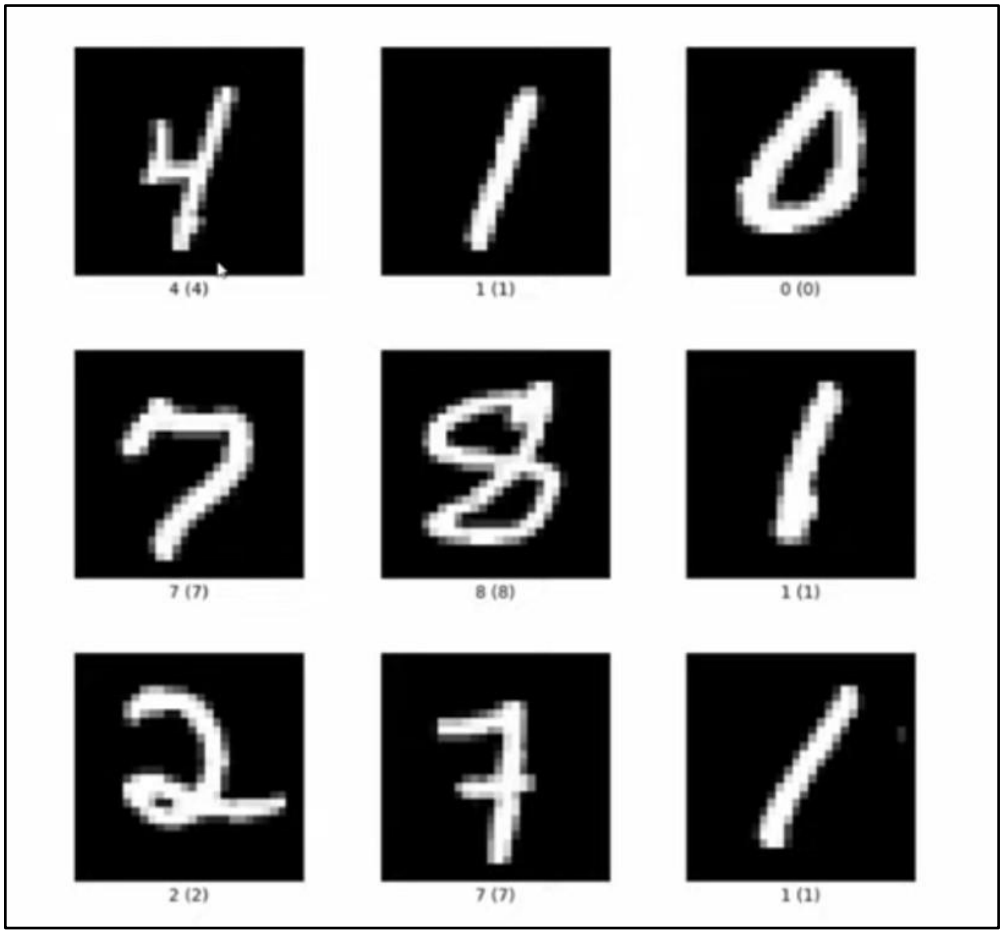
- It is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.
- The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

$$f(x) = \frac{1}{1 + e^{-x}}$$

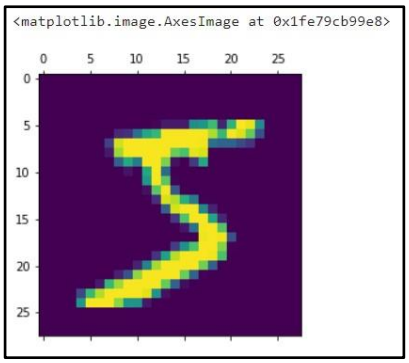
# CNN

CNN stands for Convolutional Neural Networks that are used to extract the features of the images using several layers of filters. The convolution layers are generally followed by maxpool layers that are used to reduce the number of features extracted and ultimately the output of the maxpool and layers and convolution layers are flattened into a vector of single dimension and are given as an input to the Dense layer (The fully connected network).

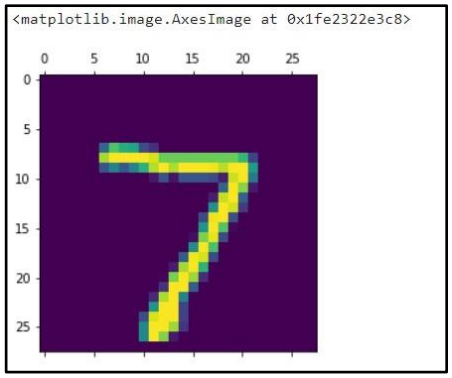


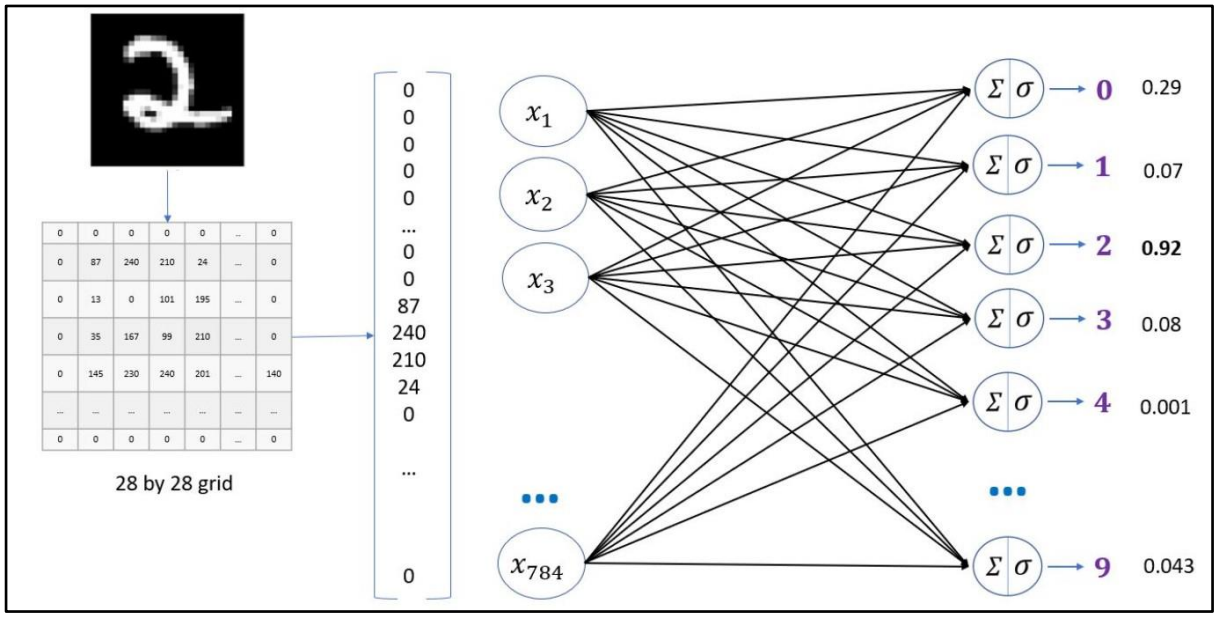
# CODE




```
import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

matplotlib inline

import numpy as np

(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()

len(X_train)

len(X_test)

X_train[0].shape

X_train[0]

plt.matshow(X_train[0])

y_train[0]

X_train = X_train / 255

X_test = X_test / 255

X_train[0]

plt.matshow(X_train[0])

y_train[0]

X_train = X_train / 255

X_test = X_test / 255

X_train[0]

X_train_flattened = X_train.reshape(len(X_train), 28*28)

X_test_flattened = X_test.reshape(len(X_test), 28*28)

X_train_flattened.shape

X_train_flattened[0]

model = keras.Sequential([keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')])

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```



model.fit(X_train_flattened, y_train, epochs=5)




model.evaluate(X_test_flattened, y_test)

y_predicted = model.predict(X_test_flattened)

y_predicted[0]

plt.matshow(X_test[0])

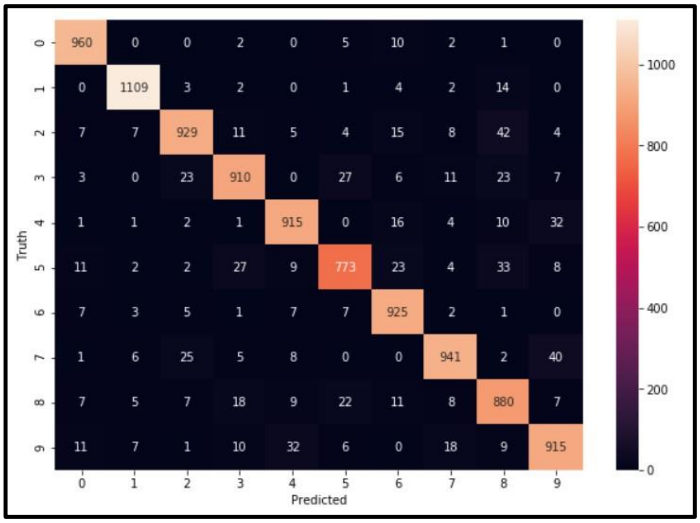
np.argmax(y_predicted[0])

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

y_predicted_labels[:5]

cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)

cm




import seaborn as sn

plt.figure(figsize = (10,7))

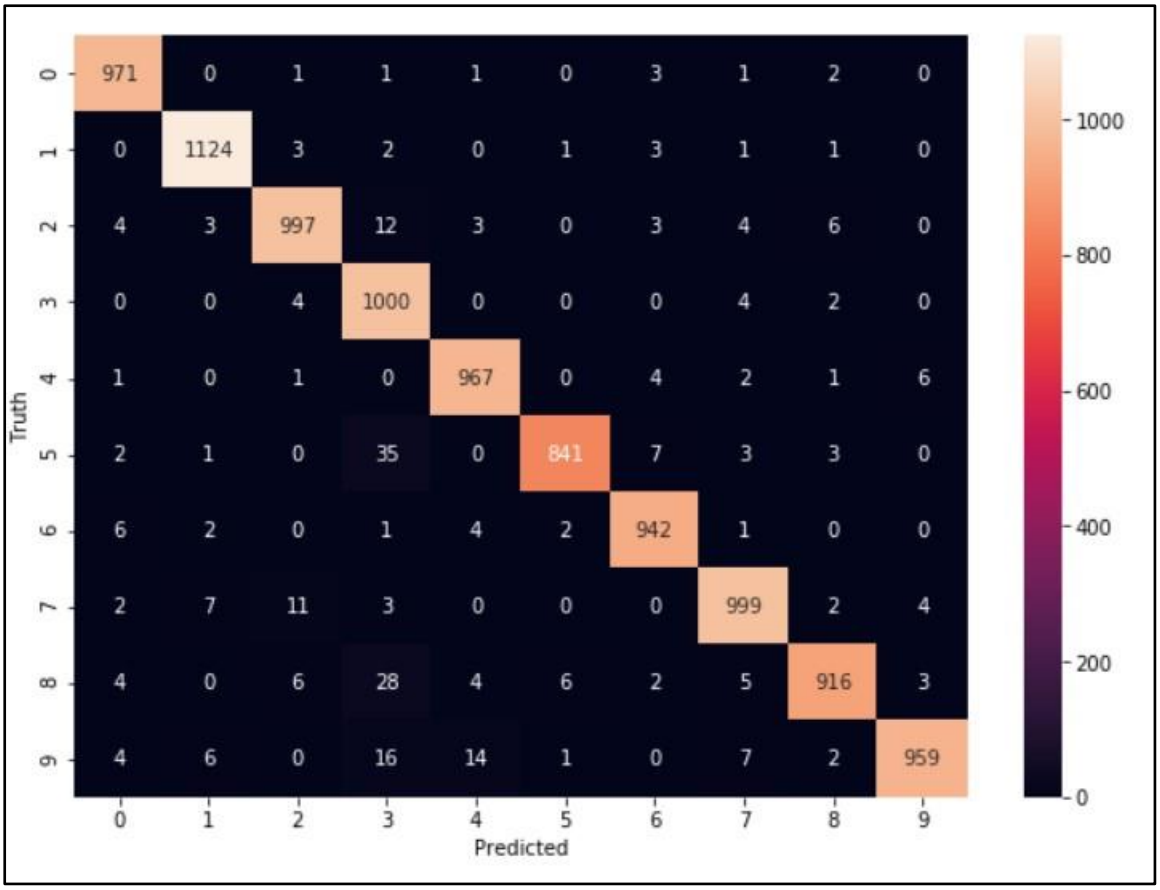
sn.heatmap(cm, annot=True, fmt='d')

plt.xlabel('Predicted')

plt.ylabel('Truth')

model = keras.Sequential([

```
    keras.layers.Dense(100, input_shape=(784,), activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(X_train_flattened, y_train, epochs=5)
```




```
model.evaluate(X_test_flattened,y_test)
y_predicted = model.predict(X_test_flattened)
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
```

```
plt.xlabel('Predicted')

plt.ylabel('Truth')

model = keras.Sequential([

   keras.layers.Flatten(input_shape=(28, 28)),

   keras.layers.Dense(100, activation='relu'),

   keras.layers.Dense(10, activation='sigmoid')

])

model.compile(optimizer='adam',

          loss='sparse_categorical_crossentropy',

          metrics=['accuracy'])

model.fit(X_train, y_train, epochs=10)

model.evaluate(X_test,y_test)
```

# RESULTS & DISCUSSION

**Step1:-** In firs step calculating the accuracy without using converting flattened(Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer) and scaling (Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. So, we use Feature Scaling to bring all values to the same magnitudes and thus, tackle this issue) the matrix and hidden layer.

Our Accuracy :-

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 4.0014 - accuracy: 0.3509
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.6839 - accuracy: 0.4795
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.5789 - accuracy: 0.4861
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.5325 - accuracy: 0.4835
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.4938 - accuracy: 0.4763
```

**Step2:-** In second step we flattened and scaling the data.

Our Accuracy:-

```
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4886 - accuracy: 0.8775
Epoch 2/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3060 - accuracy: 0.9156
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2848 - accuracy: 0.9214
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2747 - accuracy: 0.9243
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2677 - accuracy: 0.9262
```

**Step3:-** In third step we add hidden layer “ReLU”.

Our Accuracy:-

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2925 - accuracy: 0.9191
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1366 - accuracy: 0.9602
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0981 - accuracy: 0.9703
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0764 - accuracy: 0.9768
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0618 - accuracy: 0.9812
```

**Step4:-** In fourth step we flattened the matrix through keras pre-built flattened function and again adding some(2) hidden layer.

Our Accuracy:-

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2959 - accuracy: 0.9185
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1368 - accuracy: 0.9603
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0995 - accuracy: 0.9703
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0771 - accuracy: 0.9772
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0628 - accuracy: 0.9806
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0519 - accuracy: 0.9841
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0442 - accuracy: 0.9865
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0369 - accuracy: 0.9886
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0300 - accuracy: 0.9910
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0264 - accuracy: 0.9917
```

# CONCLUSION

In this project I learn how actually neural network work. In Starting we gain very low accuracy rate but using different-different hidden layer, i learn how can train our machine.  In this project I learn about the TensorFlow and keras, pandas, seaborn, numpy, matplotlib libraries. How can implement these libraries in real world project. In this project I train to machine for the digit recognition. In finally I achieved the accuracy of 0.9917 (high accuracy).

----------:-:-:-:----------

----------:-:-:-:----------

----------:-:-:-:----------