Date of Performance :

Date of Submission :

EXPERIMENT NUMBER: 6

Aim: **Design and implement a CNN model for digit recognition application.**

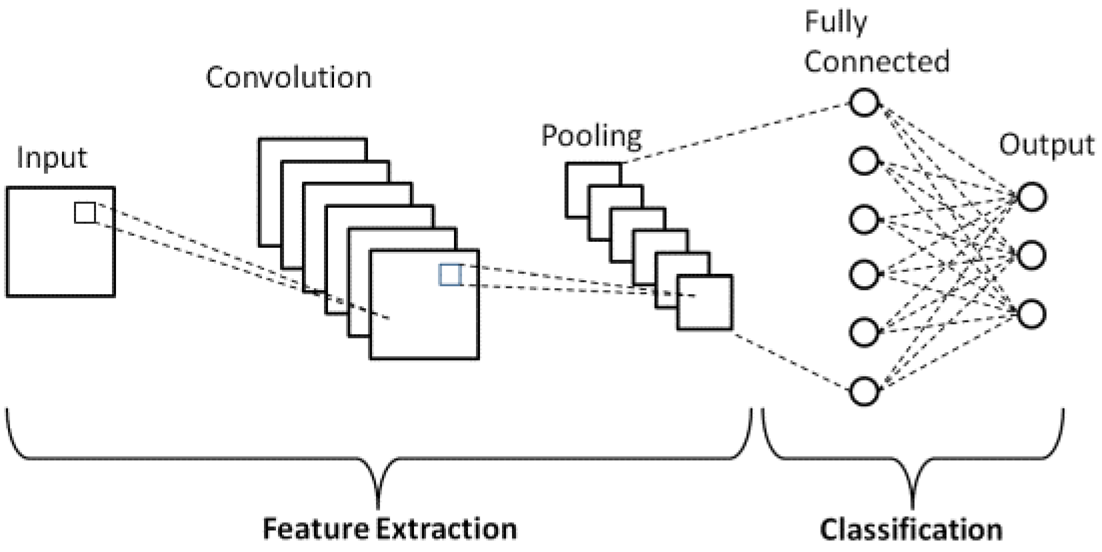
Objective:

- Acquired knowledge of advanced concepts of Convolution Neural Networks

Software Used : Python

Theory:

- Convolution Neural Networks (CNN)



1) Convolutional layer:This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).
2) Pooling Layer:The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations like max pooling, average pooling

3) Fully Connected Layer: The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture. in this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.
4) Dropout: Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data. o overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model.
5) Activation Functions: It decides which information of the model should fire in the forward direction and which ones should not at the end of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class classification, generally softmax us used.

Algorithm:

1) download `mnist_dataset from keras`
2) Split data in training and testing part.
3) Save image parameters to the constants that we will use later for data re-shaping and for model traning.
4) Normalized data
5) Add convolutional layer
6) Add max pooling layer
7) Add convolutional layer
8) Add max pooling layer
9) Add Flatten layer
10) Add dropout layer and fully connected layer
11) Train model and find accuracy from confusion matrix

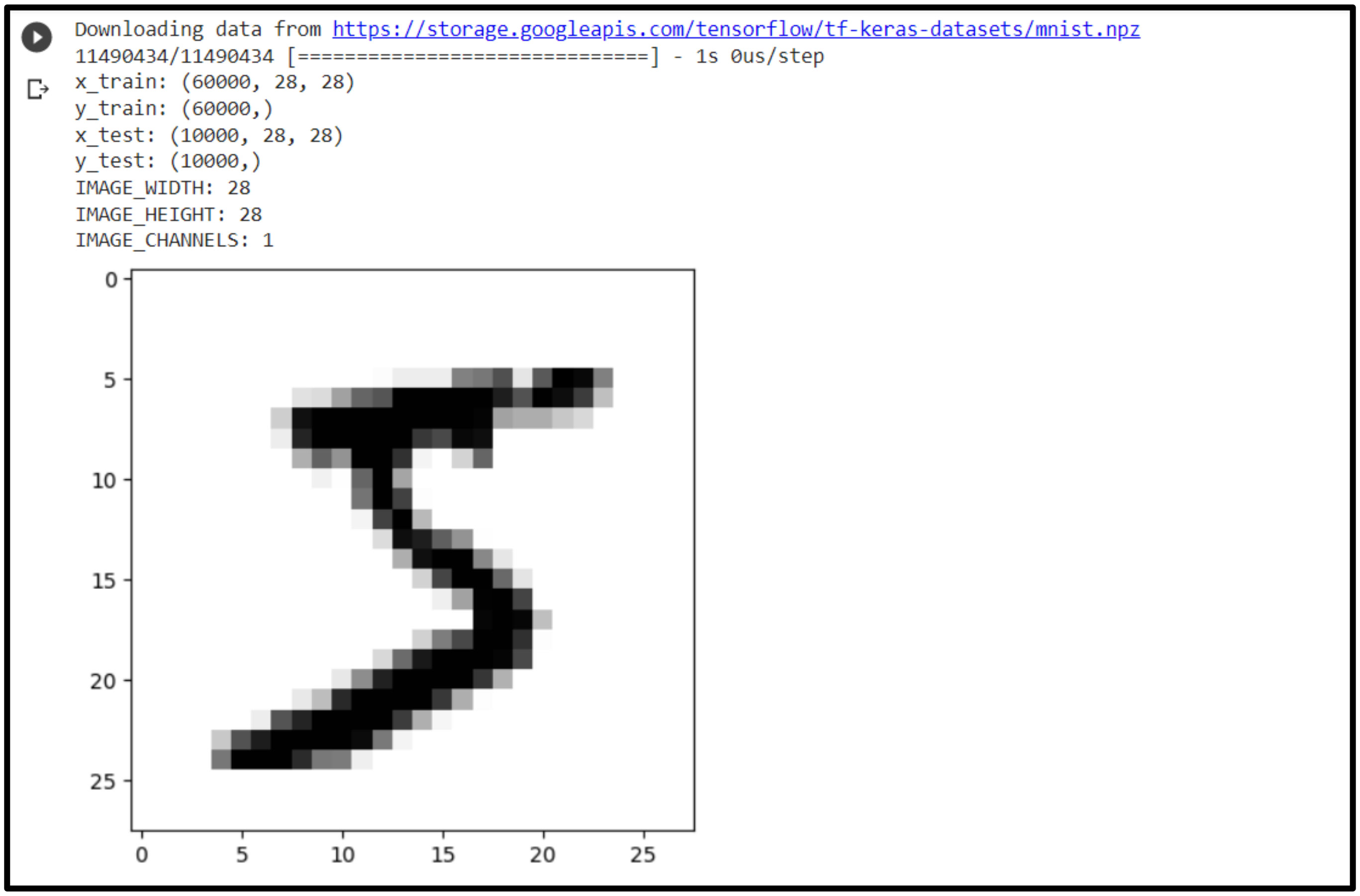
Program:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sn
import numpy as np
import pandas as pd
import math
import datetime
import platform

print('Python version:', platform.python_version())
print('Tensorflow version:', tf.__version__)
print('Keras version:', tf.keras.__version__)
%load_ext tensorboard
# Clear any logs from previous runs.
!rm -rf ./.logs/
mnist_dataset = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist_dataset.load_data()
print('x_train:', x_train.shape)
print('y_train:', y_train.shape)
print('x_test:', x_test.shape)
print('y_test:', y_test.shape)
# Save image parameters to the constants that we will use later for data re-
shaping and for model traning.
(_, IMAGE_WIDTH, IMAGE_HEIGHT) = x_train.shape
IMAGE_CHANNELS = 1

print('IMAGE_WIDTH:', IMAGE_WIDTH);
print('IMAGE_HEIGHT:', IMAGE_HEIGHT);
print('IMAGE_CHANNELS:', IMAGE_CHANNELS);
pd.DataFrame(x_train[0])
plt.imshow(x_train[0], cmap=plt.cm.binary)
plt.show()
numbers_to_display = 25
num_cells = math.ceil(math.sqrt(numbers_to_display))
plt.figure(figsize=(10,10))
for i in range(numbers_to_display):
   plt.subplot(num_cells, num_cells, i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(x_train[i], cmap=plt.cm.binary)
plt.xlabel(y_train[i])
plt.show()
x_train_with_chanels = x_train.reshape(
   x_train.shape[0],
   IMAGE_WIDTH,
   IMAGE_HEIGHT,
   IMAGE_CHANNELS
)

x_test_with_chanels = x_test.reshape(
   x_test.shape[0],
   IMAGE_WIDTH,
   IMAGE_HEIGHT,
   IMAGE_CHANNELS
)
print('x_train_with_chanels:', x_train_with_chanels.shape)
print('x_test_with_chanels:', x_test_with_chanels.shape)
x_train_normalized = x_train_with_chanels / 255
x_test_normalized = x_test_with_chanels / 255
x_train_normalized[0][18]
x_train_normalized[0][18]
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Convolution2D(
   input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS),
   kernel_size=5,
   filters=8,
   strides=1,
   activation=tf.keras.activations.relu,
   kernel_initializer=tf.keras.initializers.VarianceScaling()
))

model.add(tf.keras.layers.MaxPooling2D(
   pool_size=(2, 2),
   strides=(2, 2)
))

model.add(tf.keras.layers.Convolution2D(
   kernel_size=5,
   filters=16,
   strides=1,
   activation=tf.keras.activations.relu,
   kernel_initializer=tf.keras.initializers.VarianceScaling()
))
```

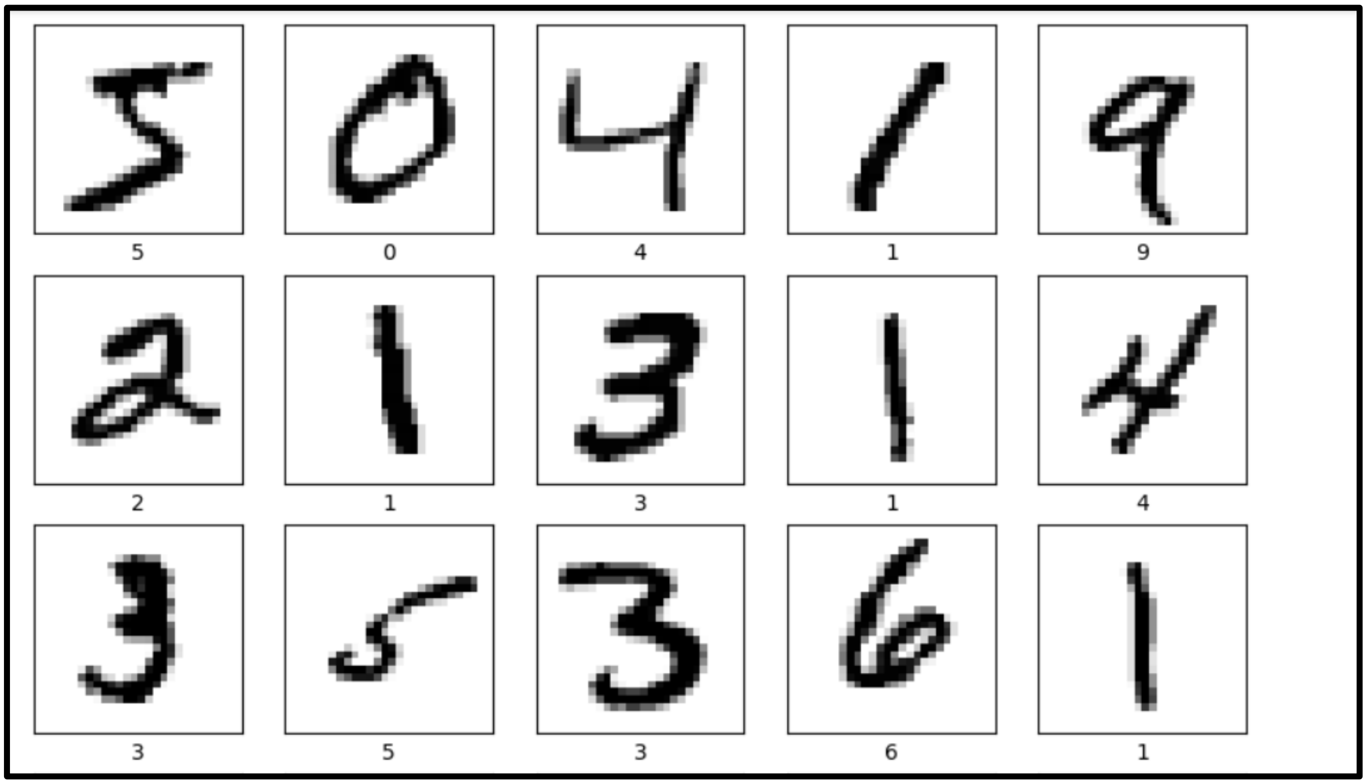
```
model.add(tf.keras.layers.MaxPooling2D(
   pool_size=(2, 2),
   strides=(2, 2)
))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(
   units=128,
   activation=tf.keras.activations.relu
));

model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(
   units=10,
   activation=tf.keras.activations.softmax,
   kernel_initializer=tf.keras.initializers.VarianceScaling()
))
model.summary()
tf.keras.utils.plot_model(
   model,
   show_shapes=True,
   show_layer_names=True,
)
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(
   optimizer=adam_optimizer,
   loss=tf.keras.losses.sparse_categorical_crossentropy,
   metrics=['accuracy']
)
log_dir=".logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

training_history = model.fit(
   x_train_normalized,
   y_train,
   epochs=10,
   validation_data=(x_test_normalized, y_test),
   callbacks=[tensorboard_callback]
)
```

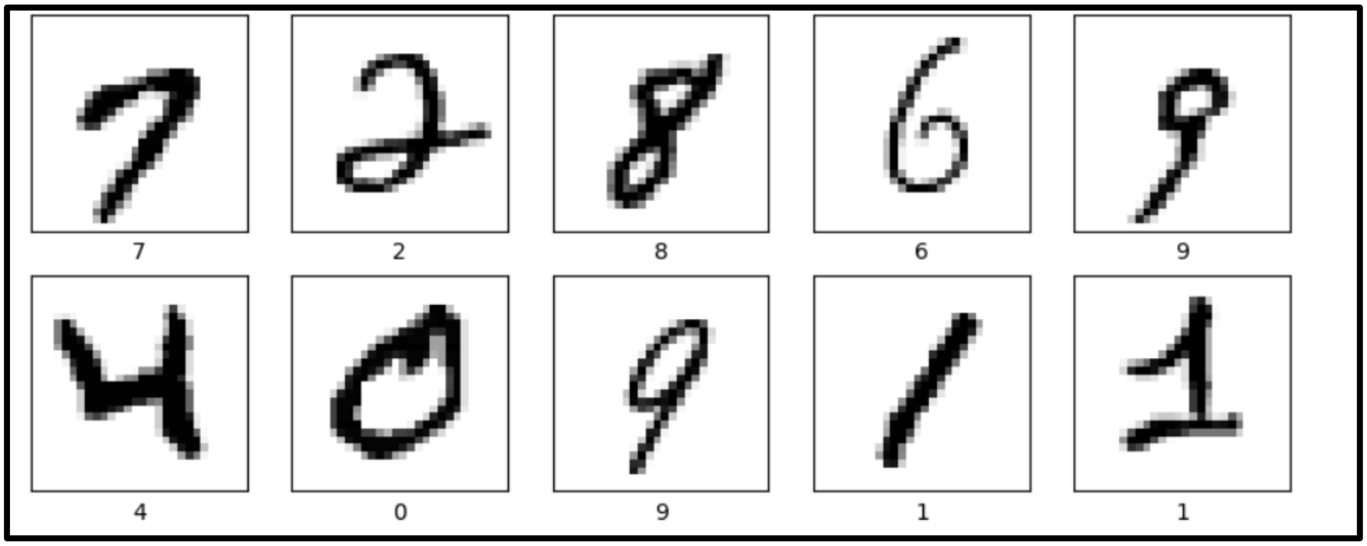
```
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.plot(training_history.history['loss'], label='training set')
plt.plot(training_history.history['val_loss'], label='test set')
plt.legend()
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.plot(training_history.history['accuracy'], label='training set')
plt.plot(training_history.history['val_accuracy'], label='test set')
plt.legend()
%%capture
train_loss, train_accuracy = model.evaluate(x_train_normalized, y_train)
print('Training loss: ', train_loss)
print('Training accuracy: ', train_accuracy)
%%capture
validation_loss, validation_accuracy = model.evaluate(x_test_normalized, y_test)
print('Validation loss: ', validation_loss)
print('Validation accuracy: ', validation_accuracy)
model_name = 'digits_recognition_cnn.h5'
model.save(model_name, save_format='h5')
loaded_model = tf.keras.models.load_model(model_name)
predictions_one_hot = loaded_model.predict([x_test_normalized])
print('predictions_one_hot:', predictions_one_hot.shape)
# Predictions in form of one-hot vectors (arrays of probabilities).
pd.DataFrame(predictions_one_hot)
predictions = np.argmax(predictions_one_hot, axis=1)
pd.DataFrame(predictions)
print(predictions[0])
plt.imshow(x_test_normalized[0].reshape((IMAGE_WIDTH, IMAGE_HEIGHT)),
cmap=plt.cm.binary)
plt.show()
numbers_to_display = 196
num_cells = math.ceil(math.sqrt(numbers_to_display))
plt.figure(figsize=(15, 15))

for plot_index in range(numbers_to_display):
   predicted_label = predictions[plot_index]
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   color_map = 'Greens' if predicted_label == y_test[plot_index] else 'Reds'
   plt.subplot(num_cells, num_cells, plot_index + 1)
   plt.imshow(x_test_normalized[plot_index].reshape((IMAGE_WIDTH,
IMAGE_HEIGHT)), cmap=color_map)

   plt.xlabel(predicted_label)

plt.subplots_adjust(hspace=1, wspace=0.5)
plt.show()
confusion_matrix = tf.math.confusion_matrix(y_test, predictions)
f, ax = plt.subplots(figsize=(9, 7))
sn.heatmap(
   confusion_matrix,
   annot=True,
   linewidths=.5,
   fmt="d",
   square=True,
   ax=ax
)
plt.show()
```

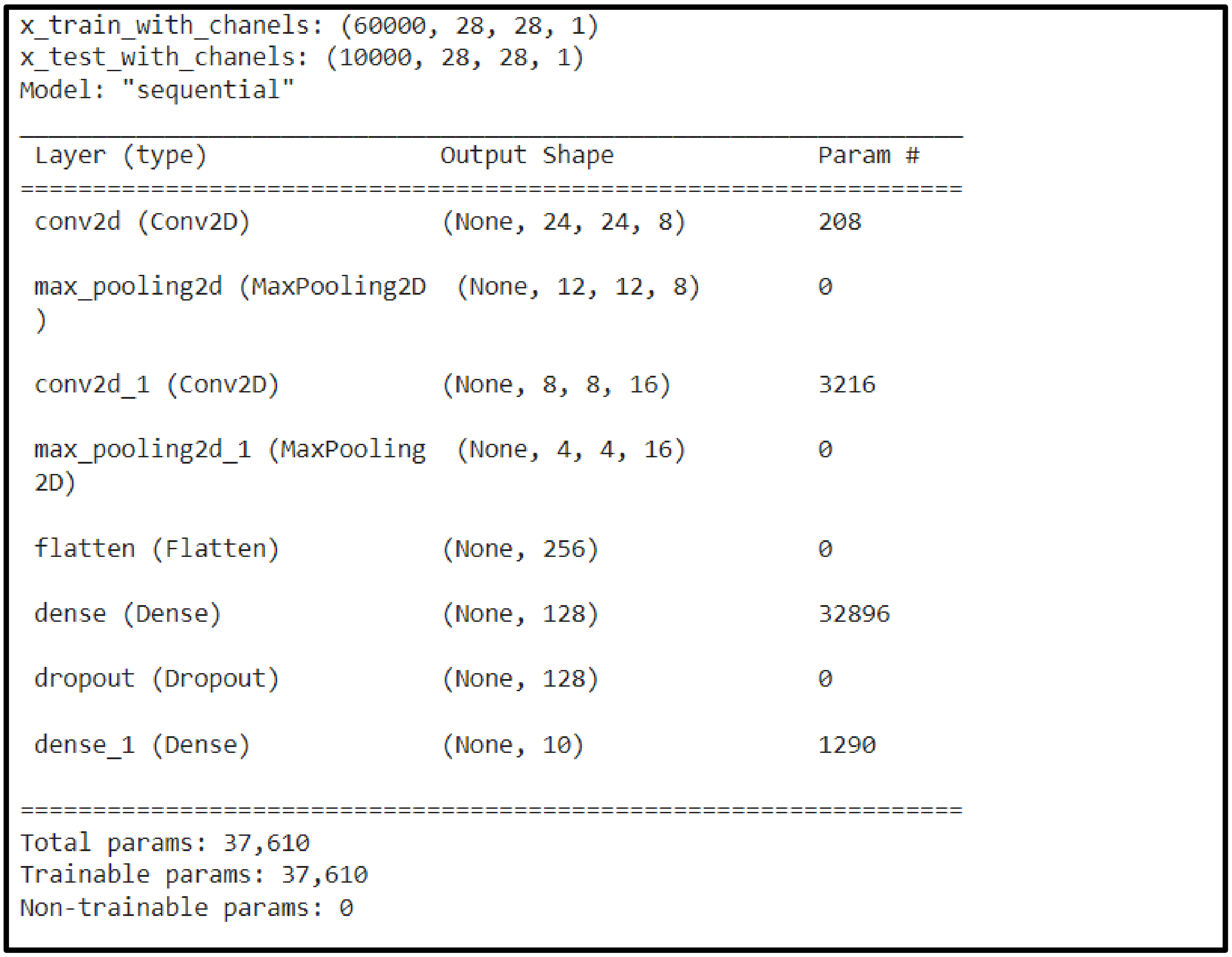
output:



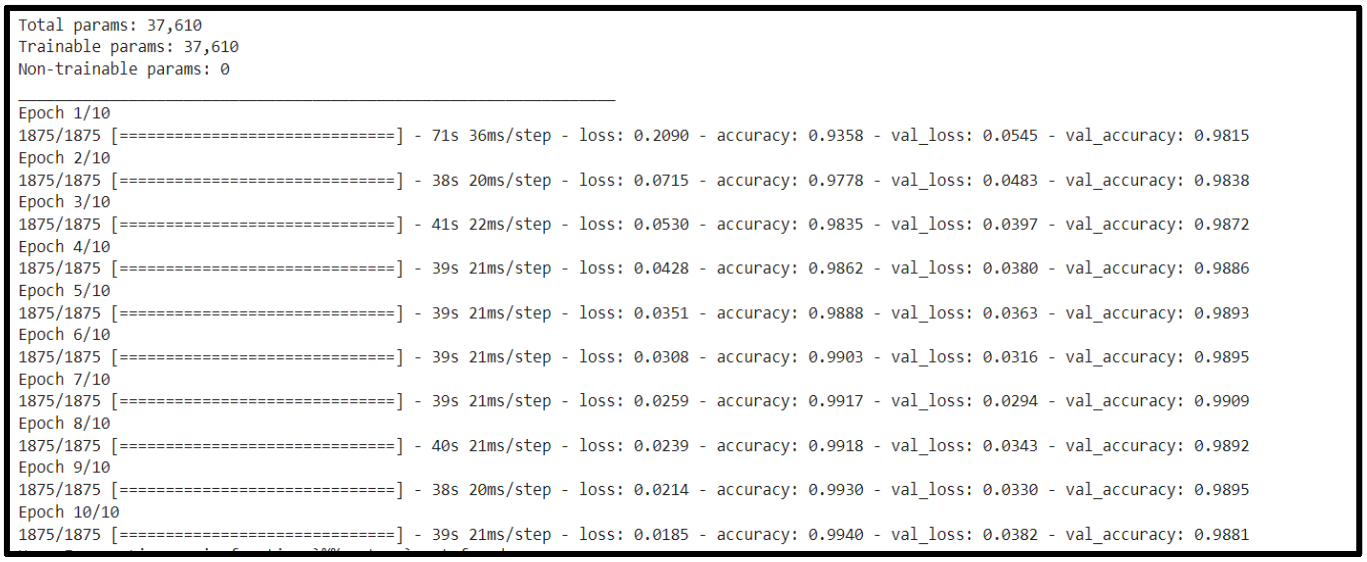
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
x_train: (60000, 28, 28)
y_train: (60000,)
x_test: (10000, 28, 28)
y_test: (10000,)
IMAGE_WIDTH: 28
IMAGE_HEIGHT: 28
IMAGE_CHANNELS: 1
```


5
0
4
1
9
2
1
3
1
4
3
5
3
6
1

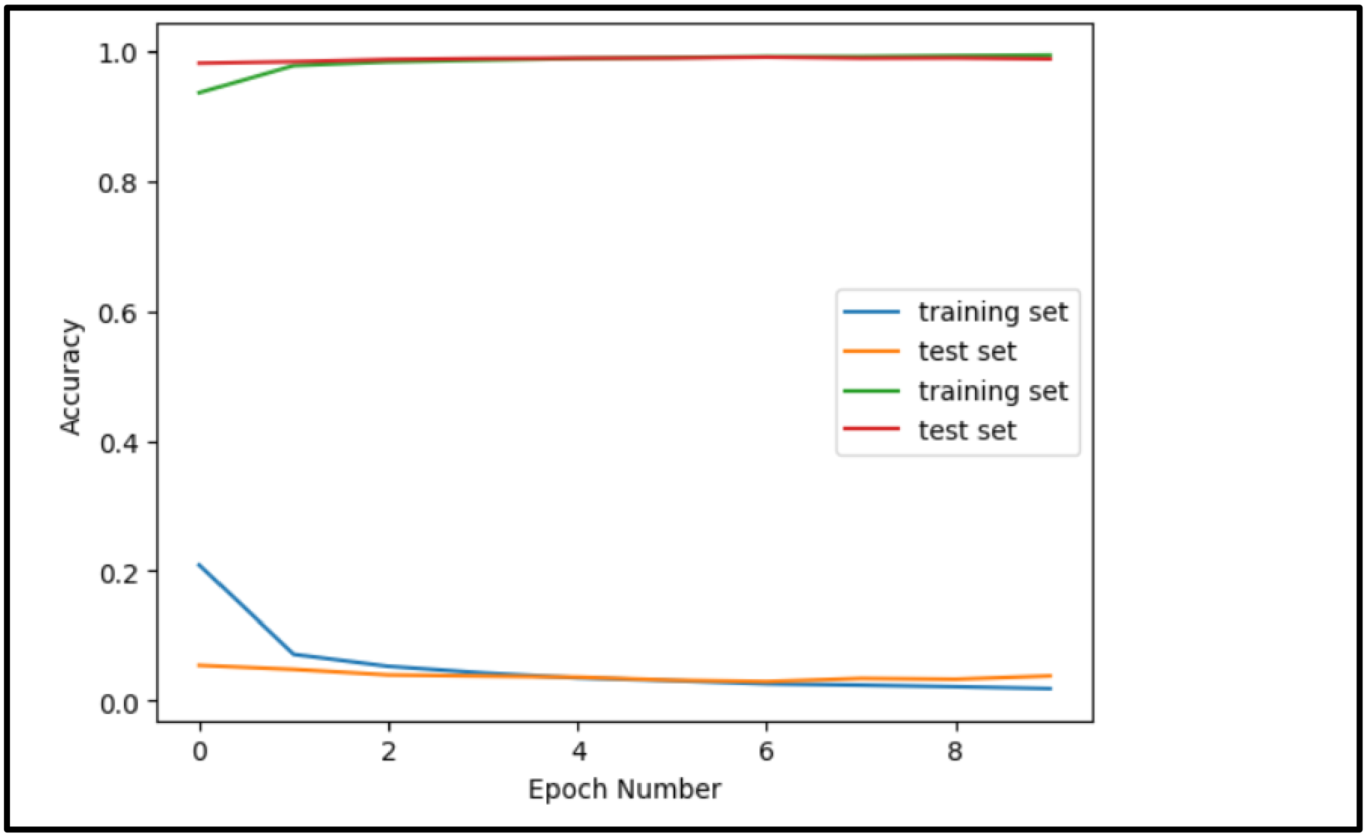

7
2
8
6
9
4
0
9
1
1



```
x_train_with_chanels: (60000, 28, 28, 1)
x_test_with_chanels: (10000, 28, 28, 1)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 8)         208

 max_pooling2d (MaxPooling2D  (None, 12, 12, 8)        0
 )

 conv2d_1 (Conv2D)           (None, 8, 8, 16)          3216

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 16)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 37,610
Trainable params: 37,610
Non-trainable params: 0
```



```
Total params: 37,610
Trainable params: 37,610
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1875/1875 [==============================] - 71s 36ms/step - loss: 0.2090 - accuracy: 0.9358 - val_loss: 0.0545 - val_accuracy: 0.9815
Epoch 2/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0715 - accuracy: 0.9778 - val_loss: 0.0483 - val_accuracy: 0.9838
Epoch 3/10
1875/1875 [==============================] - 41s 22ms/step - loss: 0.0530 - accuracy: 0.9835 - val_loss: 0.0397 - val_accuracy: 0.9872
Epoch 4/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0428 - accuracy: 0.9862 - val_loss: 0.0380 - val_accuracy: 0.9886
Epoch 5/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0351 - accuracy: 0.9888 - val_loss: 0.0363 - val_accuracy: 0.9893
Epoch 6/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0308 - accuracy: 0.9903 - val_loss: 0.0316 - val_accuracy: 0.9895
Epoch 7/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0259 - accuracy: 0.9917 - val_loss: 0.0294 - val_accuracy: 0.9909
Epoch 8/10
1875/1875 [==============================] - 40s 21ms/step - loss: 0.0239 - accuracy: 0.9918 - val_loss: 0.0343 - val_accuracy: 0.9892
Epoch 9/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0214 - accuracy: 0.9930 - val_loss: 0.0330 - val_accuracy: 0.9895
Epoch 10/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0185 - accuracy: 0.9940 - val_loss: 0.0382 - val_accuracy: 0.9881
```



Conclusion/Outcome:

Thus we have implemented CNN model for digit recognition

Marks & Signature:

R1 (4 Marks)	**R2 (4 Marks)**	**R3 (4 Marks)**	**R4 (3 Mark)**	**Total (15 Marks)**	**Signature**