NISHANT PAWAR IS 3

The MNIST database contains 60,000 training images and 10,000 testing images.Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support-vector machine to get an error rate of 0.8%.

In [3]:
```
import tensorflow
import keras
from tensorflow.keras import Sequential
from keras.layers import Flatten, Dropout,Dense,Activation
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
```

In [4]:
```
(X_train,y_train),(X_test,y_test) = mnist.load_data()
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
ist.npz
11493376/11490434 [==============================] - 5s 0us/step
11501568/11490434 [==============================] - 5s 0us/step
```

Out[4]:
```
((60000, 28, 28), (60000,), (10000, 28, 28), (10000,))
```

In [5]:
```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

In [6]:
```
X_train /= 255
X_test /= 255
```

In [7]:
```
X_train.shape , X_test.shape
```

Out[7]:
```
((60000, 784), (10000, 784))
```

In [8]:
```
from keras.utils import np_utils
```

In [9]:
```
n_classes = 10
Y_train = np_utils.to_categorical(y_train, n_classes)
Y_test = np_utils.to_categorical(y_test, n_classes)
Y_train.shape
```

Out[9]:
```
(60000, 10)
```

In [10]:
```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[10]:
```
((60000, 784), (60000,), (10000, 784), (10000,))
```

In [11]:
```
model= Sequential()
```

In [12]:
```
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))
```

In [13]:
```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam
```

In [14]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               401920

 activation (Activation)     (None, 512)               0

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 activation_1 (Activation)   (None, 512)               0

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 10)                5130

 activation_2 (Activation)   (None, 10)                0

=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
```

In [15]:
```
model.fit(X_train, Y_train,batch_size=128, epochs=20,verbose=2,validation_data=(X_te
```

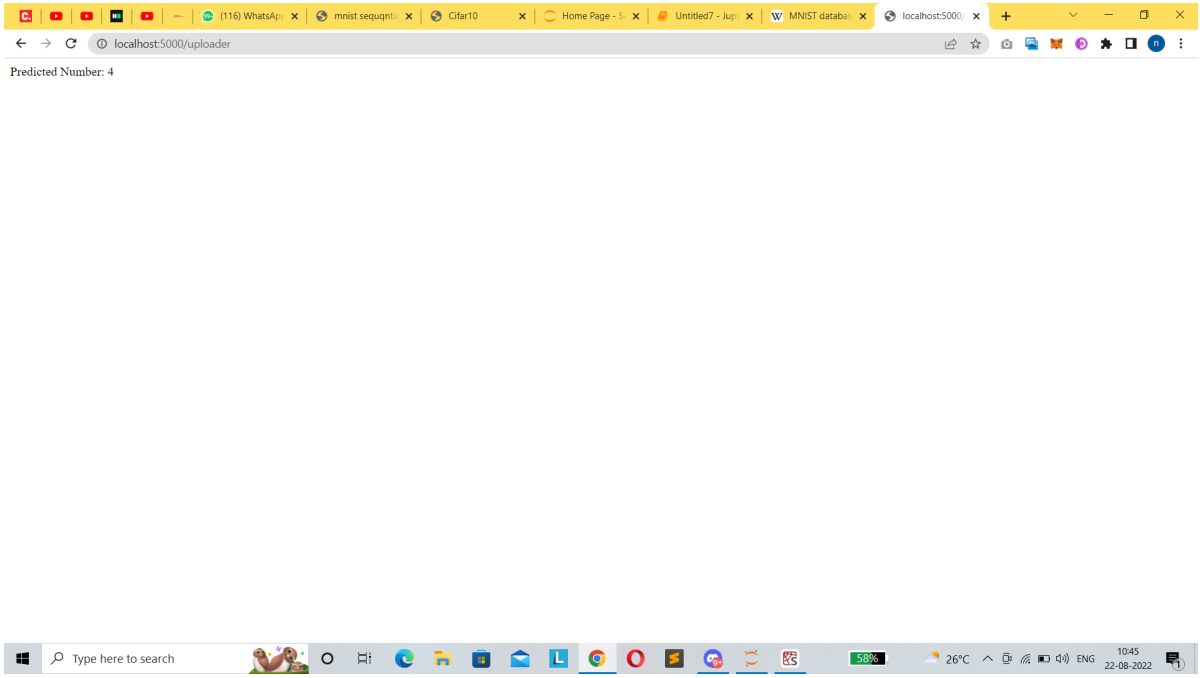
```
Epoch 1/20
469/469 - 5s - loss: 0.2512 - accuracy: 0.9249 - val_loss: 0.0969 - val_accuracy: 0.
9691 - 5s/epoch - 12ms/step
Epoch 2/20
469/469 - 4s - loss: 0.1000 - accuracy: 0.9689 - val_loss: 0.0924 - val_accuracy: 0.
9707 - 4s/epoch - 9ms/step
Epoch 3/20
469/469 - 4s - loss: 0.0714 - accuracy: 0.9773 - val_loss: 0.0673 - val_accuracy: 0.
9793 - 4s/epoch - 9ms/step
Epoch 4/20
469/469 - 4s - loss: 0.0554 - accuracy: 0.9823 - val_loss: 0.0656 - val_accuracy: 0.
9791 - 4s/epoch - 8ms/step
Epoch 5/20
469/469 - 3s - loss: 0.0457 - accuracy: 0.9857 - val_loss: 0.0686 - val_accuracy: 0.
```

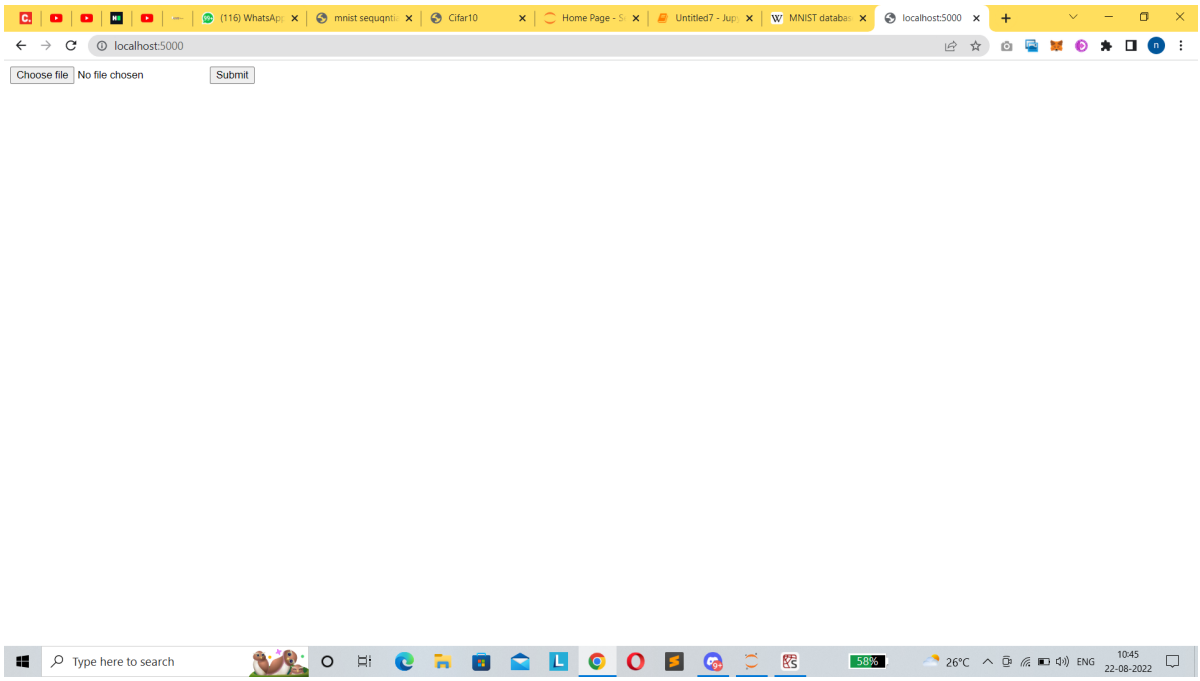
```
9788 - 3s/epoch - 7ms/step
Epoch 6/20
469/469 - 4s - loss: 0.0399 - accuracy: 0.9868 - val_loss: 0.0628 - val_accuracy: 0.
9814 - 4s/epoch - 8ms/step
Epoch 7/20
469/469 - 3s - loss: 0.0365 - accuracy: 0.9877 - val_loss: 0.0705 - val_accuracy: 0.
9803 - 3s/epoch - 7ms/step
Epoch 8/20
469/469 - 4s - loss: 0.0319 - accuracy: 0.9890 - val_loss: 0.0726 - val_accuracy: 0.
9802 - 4s/epoch - 8ms/step
Epoch 9/20
469/469 - 3s - loss: 0.0270 - accuracy: 0.9909 - val_loss: 0.0623 - val_accuracy: 0.
9839 - 3s/epoch - 7ms/step
Epoch 10/20
469/469 - 4s - loss: 0.0261 - accuracy: 0.9911 - val_loss: 0.0649 - val_accuracy: 0.
9827 - 4s/epoch - 7ms/step
Epoch 11/20
469/469 - 3s - loss: 0.0234 - accuracy: 0.9921 - val_loss: 0.0696 - val_accuracy: 0.
9821 - 3s/epoch - 7ms/step
Epoch 12/20
469/469 - 3s - loss: 0.0200 - accuracy: 0.9933 - val_loss: 0.0694 - val_accuracy: 0.
9811 - 3s/epoch - 7ms/step
Epoch 13/20
469/469 - 3s - loss: 0.0213 - accuracy: 0.9928 - val_loss: 0.0760 - val_accuracy: 0.
9813 - 3s/epoch - 7ms/step
Epoch 14/20
469/469 - 3s - loss: 0.0209 - accuracy: 0.9933 - val_loss: 0.0699 - val_accuracy: 0.
9818 - 3s/epoch - 7ms/step
Epoch 15/20
469/469 - 3s - loss: 0.0200 - accuracy: 0.9934 - val_loss: 0.0736 - val_accuracy: 0.
9827 - 3s/epoch - 7ms/step
Epoch 16/20
469/469 - 3s - loss: 0.0172 - accuracy: 0.9945 - val_loss: 0.0687 - val_accuracy: 0.
9844 - 3s/epoch - 7ms/step
Epoch 17/20
469/469 - 3s - loss: 0.0143 - accuracy: 0.9953 - val_loss: 0.0772 - val_accuracy: 0.
9825 - 3s/epoch - 7ms/step
Epoch 18/20
469/469 - 3s - loss: 0.0180 - accuracy: 0.9938 - val_loss: 0.0740 - val_accuracy: 0.
9845 - 3s/epoch - 7ms/step
Epoch 19/20
469/469 - 3s - loss: 0.0187 - accuracy: 0.9938 - val_loss: 0.0637 - val_accuracy: 0.
9855 - 3s/epoch - 7ms/step
Epoch 20/20
469/469 - 4s - loss: 0.0141 - accuracy: 0.9953 - val_loss: 0.0708 - val_accuracy: 0.
9836 - 4s/epoch - 8ms/step
```

Out[15]: <keras.callbacks.History at 0x12090456c10>

In [16]:
```
model.save('model.h5')
```

FINAL OUTPUT


localhost:5000/uploader
Predicted Number: 4
Type here to search
58%
26°C
ENG
10:45
22-08-2022


localhost:5000
Choose file
No file chosen
Submit
Type here to search
58%
26°C
ENG
10:45
22-08-2022

In [ ]: