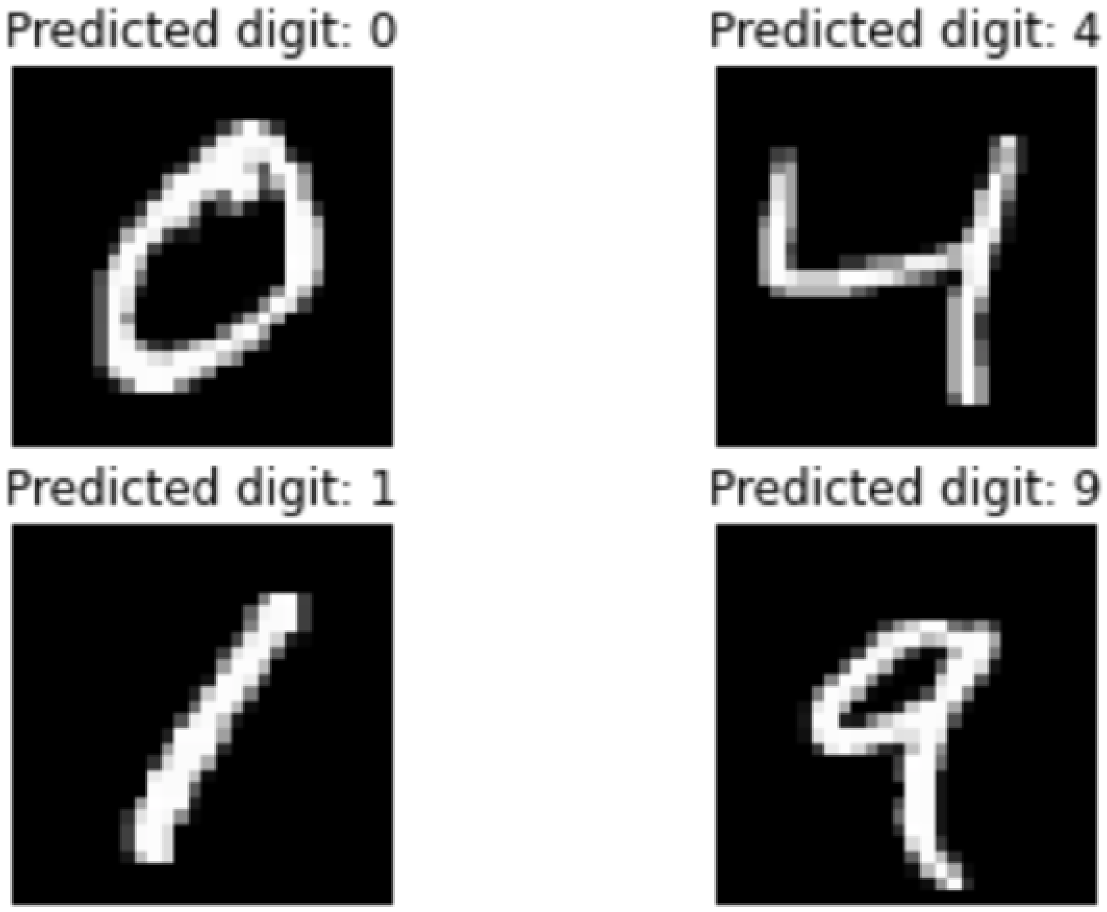
ML PROJECT

Handwritten Digit Recognition using python and MNIST



Submitted To:

Mr. Deepak Yadav

Submitted By:

Ashutosh Pande

Introduction & Objective:

The 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 be made with many different flavors. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

MNIST (Modified National Institute of Standards and Technology database) is probably one of the most popular datasets among machine learning and deep learning enthusiasts. The MNIST dataset contains 60,000 small square 28×28 pixel grayscale training images of handwritten digits from 0 to 9 and 10,000 images for testing. So, the MNIST dataset has 10 different classes.

Technologies:

Deep learning is a machine learning technique that lets computers learn by example. Ever wondered Amazon knows what you would want to buy next and those suggestions are exactly what you need but just never knew it before? You guessed it. It's deep-learning algorithms at work. Whether it's Alexa, Siri or Cortana, deep learning helps them understand speech and the language.

Deep Learning uses different types of neural network architectures like object recognition, image and sound classification, and object detection for different types of problems. The more data a Deep Learning algorithm is trained on, the more accurate it is.

Steps & Methodology

1. **Import the libraries and load the dataset:** Importing the necessary libraries, packages, and MNIST dataset
2. **Pre-process the data**
3. **Create the model**
4. **Train and Evaluate the Model**
5. **Saving the model**
6. **Make Predictions**

1. Import the libraries and load the dataset

Before starting anything, make-sure Tensorflow, Keras, numpy, and pillow are installed on your computer. We need to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset through keras. The mnist.load_data() method returns the training data, its labels along with the testing data and its labels.

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K# the data, split between train and test sets
from keras.utils import np_utils
from sklearn.model_selection import KFold
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.optimizers import SGD# the MNIST data is split between
train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

2. Preprocess the data

The dimension of the training data is (60000, 28, 28). CNN accepts four dimensions. So we need to reshape the images to have dimensions (samples*width*height*pixels)

```
# Reshape to be samples*pixels*width*height
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric. The technique to achieve this is called One-Hot Code.

```
# One hot Code
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

We need to normalize inputs from 0–255 to 0–1 as to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of value. This involves first converting the data type from unsigned integers to floats, then dividing the pixel values by the maximum value.

```
# convert from integers to floats
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')# normalize to range [0,1]
X_train = X_train / 255.0
X_test = X_test / 255.0
```

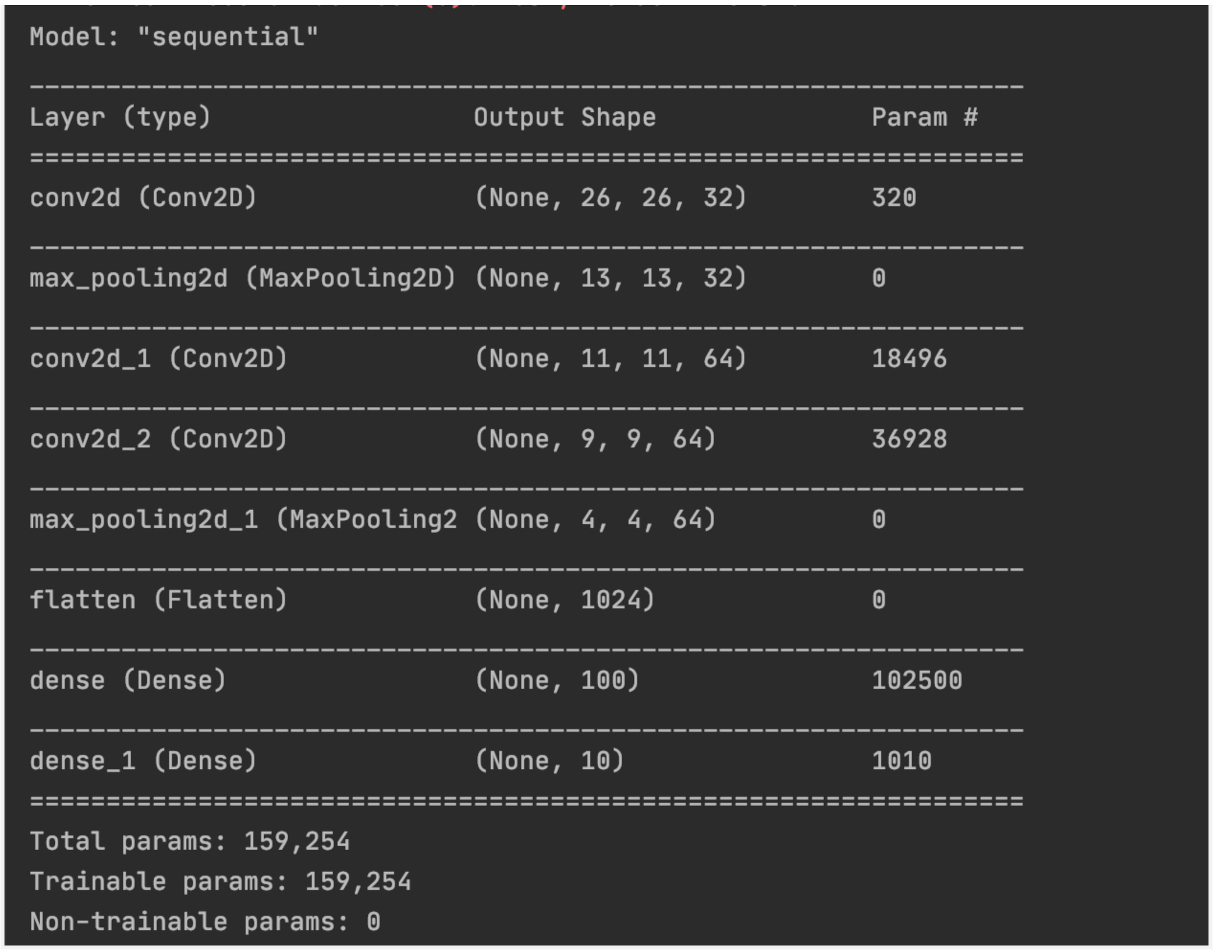
3. Create the model

Next, we need to define a baseline Convolutional Neural Network (CNN) model.

What is CNN? In simpler words, CNN is an artificial neural network that specializes in picking out or detect patterns and make sense of them. Thus, CNN has been most useful for image classification**.** A CNN model has various types of filters of different sizes and numbers. These filters are essentially what helps us in detecting the pattern.

Convolutional layers: There are a number of ways to make covolutional model. I tried and tested a lot of those and found the following work the best.

```
# Create model
# Building CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
```



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 64)          36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 100)               102500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 159,254
Trainable params: 159,254
Non-trainable params: 0
```

Model Summary

4. Train and Evaluate the Model

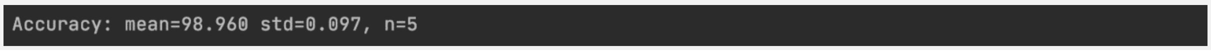
After the model is defined, we need to evaluate it. We will evaluate the model using five-fold cross-validation.

Before we move further, let us understand what cross-validation is. Suppose you have *n* images of pens and pencils. You want to train a deep Learning algorithm so that it can differentiate between the two. The idea behind training and testing any data model is to achieve maximum learning rate and maximum validation. Better Learning rate and better validation can be achieved by increasing the train and test data respectively. Since our data is limited, there is a sweet spot where we can have optimal Learning rate and validation. To find that sweet spot, we use **cross-validation** which divides the data set into *k* subsets and looks for the best test to train data ratio.

In our case, value of *k is 5*. Thus, each test set will be 20% of the training dataset, or about 12,000 examples.

```
def evaluate_model(X_train, y_Train, n_folds=5):    accuracy, data = list(), list()
    # prepare 5-cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)

    for x_train, x_test in kfold.split(X_train):
        # create model
        model = create_model()        # select rows for train and test
        trainX, trainY, testX, testY = X_train[x_train], y_Train[x_train], X_train[x_test],
y_Train[x_test]        # fit model
        data_fit = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=10,
batch_size=32)        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)        # stores Accuracy
        accuracy.append(acc)
        data.append(data_fit)
    return accuracy, data
```

You can see a visual representation of the accuracies achieved during the evaluation with the help of pyplot**.**

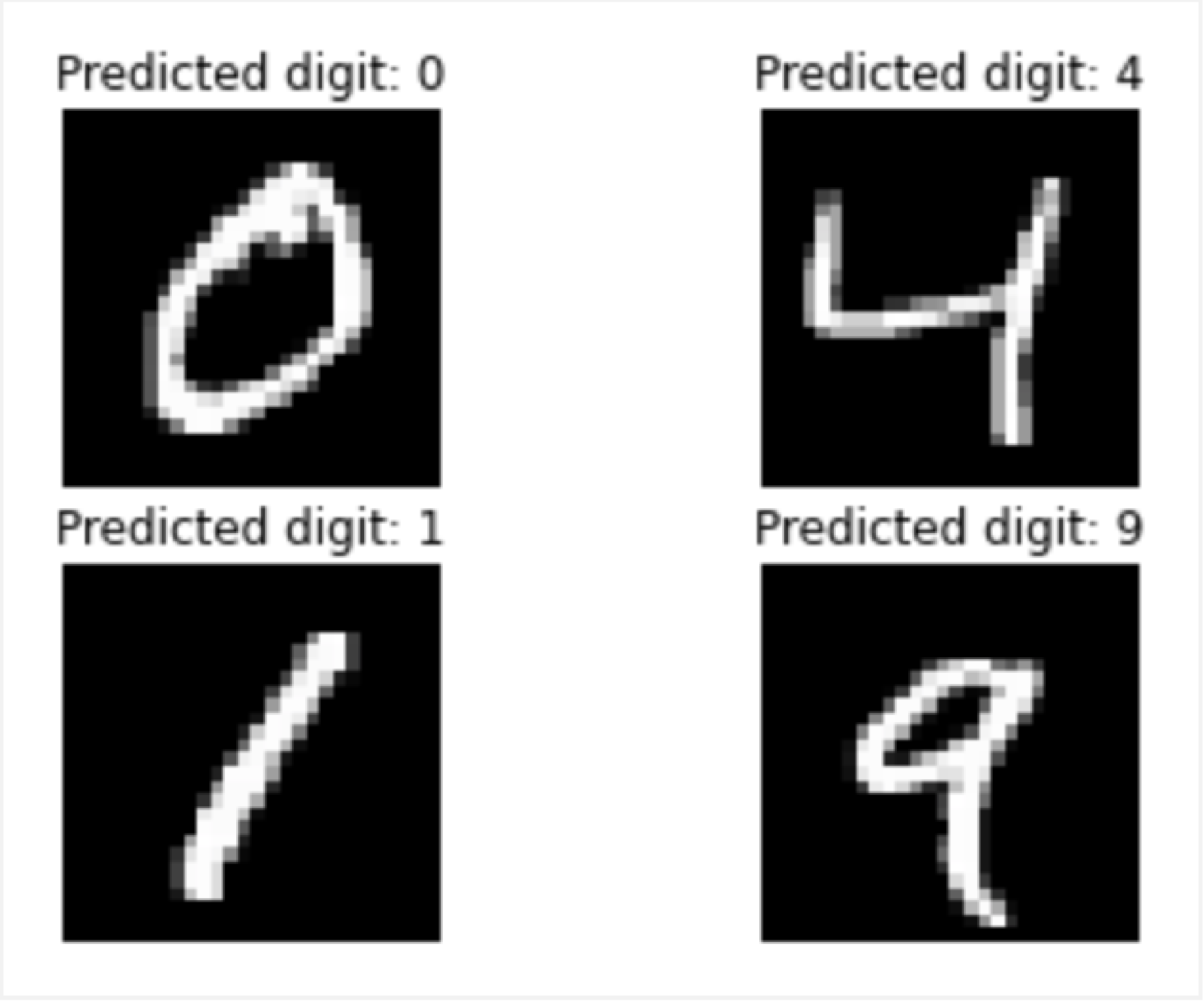


```
Accuracy: mean=98.960 std=0.097, n=5
```

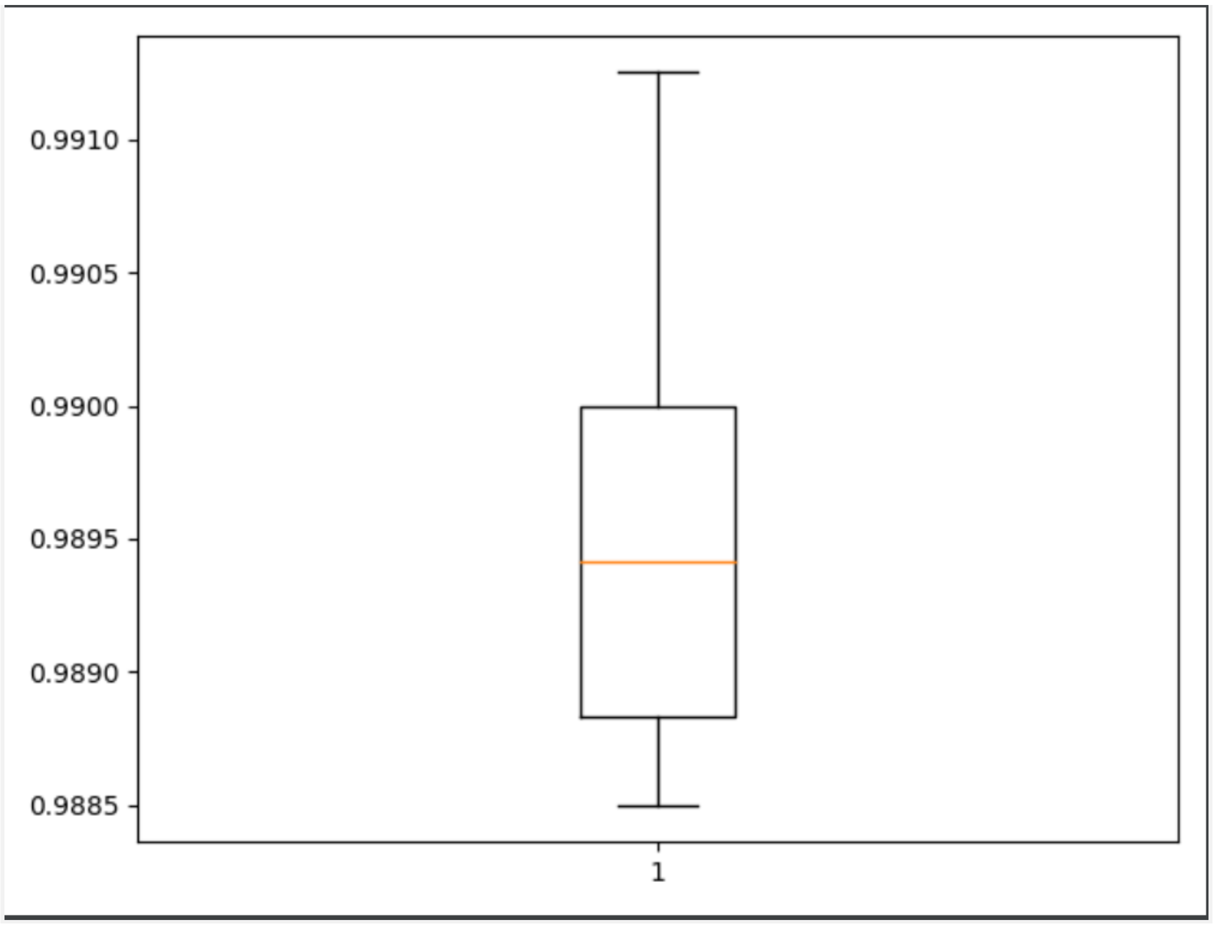
Accuracy

```
# summarize model performance
def summarize_performance(acc):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (numpy.mean(acc) * 100,
numpy.std(acc) * 100, len(acc)))
```

```
# box and whisker plots of results
pyplot.boxplot(acc)
pyplot.show()
```

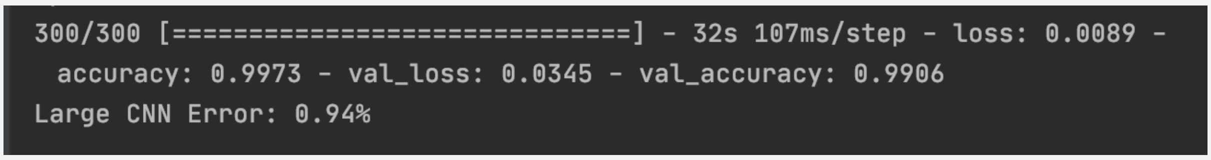


Predicting using the dataset images



Accuracy Plot

```
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200)
```



```
300/300 [==============================] - 32s 107ms/step - loss: 0.0089 -
  accuracy: 0.9973 - val_loss: 0.0345 - val_accuracy: 0.9906
Large CNN Error: 0.94%
```

5. Saving the Model

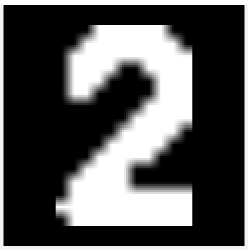
If you are satisfied with your model, you can save it using *model.save(“model_name.h5”).* The accuracy of the neural network was **99.73%**.

```
# serialize model to JSON and save the model
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights("final_model.h5")
```

6. Make Predictions

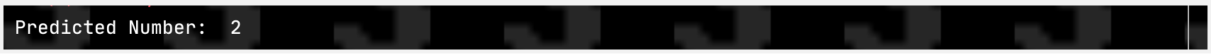
We need to resize and reshape the image to **(1, 28, 28, 1)**. (**Note:** The image must be in grayscale.) We need to load the saved model by using load_model.

```
def predict(img):
    image = img.copy()
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # image = cv2.threshold(image, 140, 255, cv2.THRESH_BINARY)[1]
    image = cv2.resize(image, (28, 28))
    # display_image(image)
    image = image.astype('float32')
    image = image.reshape(1, 28, 28, 1)
    image /= 255

    # plt.imshow(image.reshape(28, 28), cmap='Greys')
    # plt.show()
    model = load_model('cnn.hdf5')
    pred = model.predict(image.reshape(1, 28, 28, 1), batch_size=1)

    print("Predicted Number: ", pred.argmax())

    # return pred.argmax()
```



Test Image

Using the test image, we will predict the number.



```
Predicted Number:  2
```

Result of the Prediction.

Conclusion:

In this article, we have successfully built a Python deep learning project on handwritten digit recognition. We have built and trained the Convolutional neural network which is very effective for image classification purposes.

__________________________END_______________________________