IDE FINAL DOCUMENATION

NAME: HIMAKAR ANANTHASETTY

ROLL NO: 121810316009

TITLE: HAND WRITTEN AND PRINTED DIGIT RECOGNITION USING CONVOLUTIONAL NEURAL NETWORKS.

Abstract:

Handwriting recognition systems in the past depended on handmade characteristics and a lot of prior information. It's difficult to train an optical character recognition (OCR) system based on these requirements. Deep learning approaches are at the centre of handwriting recognition research, which has yielded breakthrough results in recent years. Nonetheless, the increasing development in the volume of handwritten data combined with the availability of vast computing capacity necessitates an increase in recognition accuracy and warrants additional exploration. Convolutional neural networks (CNNs) are extremely successful in perceiving the structure of handwritten characters/words in ways that allow for the automated extraction of different characteristics, making CNN the best solution for solving handwriting recognition challenges and printed digits recognition challenges.

Introduction:

Image classification is not trivial task which can be achieved using various approaches. However, recently deep learning has been successfully applied to a wide range of machine learning applications. Accordingly, in this work we proposed a subtle Convolutional Neural Networks (CNNs) which is used to train and test our handwritten digits and printed digits. Constructing CNNs plays an essential role in justifying both performance and time consumption. Thus, in our implementation, we designed an elegant CNN after carefully investigating its parameters. In general, using CNNs for handwritten digits recognition consists of a certain number of steps described below:

- Preparing patterns before feeding to the CNN. All images are pre-processed before passing into the network. In our experiments, CNN is designed to receive an image size of 64x64 pixels. Hence, all images have been cropped to the same size to be fed to the model.

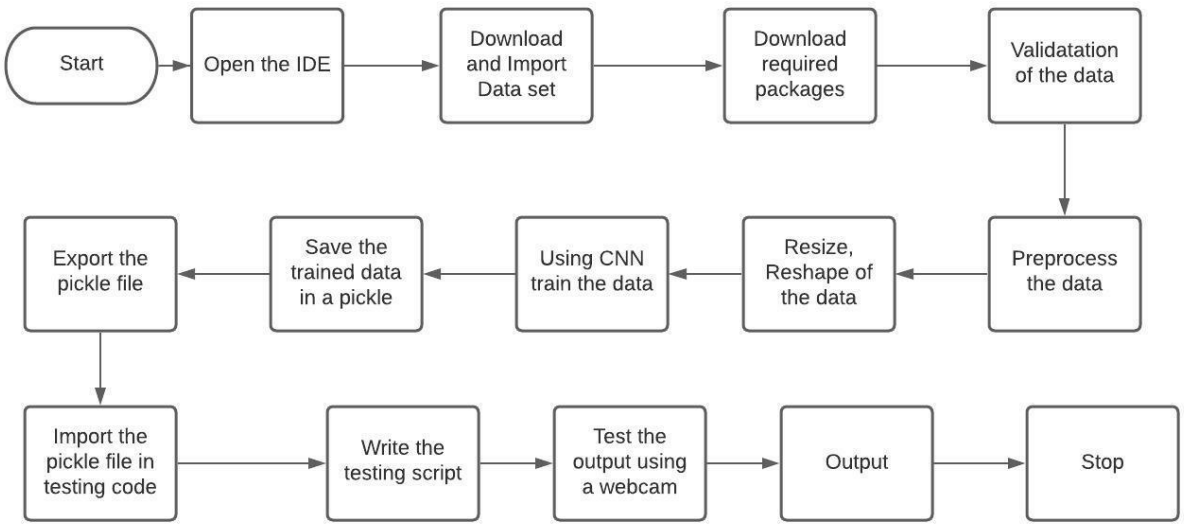
- After preparing images, they are fed to the deep model to extract features. As demonstrated earlier a robust CNN is used in this experiment to extract robust features used in the final decision to justify the class to which they belong to.

• Finally, the last layer named softmax layer is used at the top of CNN to minimize the error. In this work, we carefully explored the architecture of CNN consisting of five layers.

Requirements:

1) Testing and Training Data.
2) Collected nearly 10,000 sample images for each class i.e (0 to 9) , for Testing and Training Data for printed digits.
3) Collected nearly 20,000 sample images for each class i.e (0 to 9) , for Testing and Training Data for hand written digits.
4) Split the Data into Testing , Training and Validation.
5) Pre-process the data.
6) Perform some modifications like fit, zoom, rotate on the data sets.
7) Train the Data
8) Test the model with an example
9) Print the Graphs
10) Results

Flow chart



Packages Used:

1)Keras :

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

and TensorFlow and allows you to define and train neural network models in just a few lines of code.

2)Numpy :

NumPy is a Python library used for working with arrays.It also has functions for working in domain of linear algebra, fourier transform, and matrices.NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.NumPy stands for Numerical Python.

3)Tensor Flow:

TensorFlow is an open source software library for numerical computation using data flow graphs. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture enables you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow provides stable Python and C APIs as well as non-guaranteed backward compatible API's for C++, Go, Java, JavaScript, and Swift.

4)Open CV:
OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.
This OpenCV tutorial will help you learn the Image-processing from Basics to Advance, like operations on Images, Videos using a huge set of Opencv-programs and projects.

5)**Sklearn** :

This library implements multi-layer perceptrons, auto-encoders and recurrent neural networks with a stable future proof interface as a wrapper for the powerful existing libraries such as lasagne currently, with plans for blocks which is compatible with Scikit-learn for a more user-friendly and Pythonic interface. By importing the sknn package provided by this library, you can easily train deep neural networks as regressors (to estimate continuous outputs from inputs) and classifiers (to predict discrete labels from features).

Due to the underlying Lasagne implementation, the code supports the following neural network features

- Activation Functions: Sigmoid, Tanh, Rectifier, Softmax, Linear.
- Layer Types: Convolution (greyscale and color, 2D), Dense (standard, 1D).
- Learning Rules: sgd, momentum, nesterov, adadelta, adagrad, rmsprop, adam.

- Regularization: L1, L2, dropout, and batch normalization.
- Dataset Formats: Numpy.ndarray, scipy.sparse, pandas.DataFrame and iterators (via callback).

Matplotlib and Seaborn:

Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPythonotTkinter. Both are the data visualization library used for plotting graph which will help us in understanding the data.

Dataset:

The MNIST dataset is a handwritten digital picture collection centred in 28*28 fields with 20 20pixel pictures. Figure 6 shows some examples from the MNIT dataset. The data is split into four sections. The first part contains 60000 photographs for training; the second part has 10000 pictures for testing; and the third and fourth parts contain the associated labels, or the numbers that correspond to the handwritten pictures. About 250 different writers contributed digital photos to the training programme. About half of them are regular high school pupils, and the other half are American census employees. The number of authors in the test set is unknown, however there is no overlap with the training set writer.

Methodology:

Convolutional Neural Network Architecture:

A basic convolutional neural network comprises three components, namely, the convolutional layer, the pooling layer and the output layer. The pooling layer is optional sometimes. The typical convolutional neural network architecture with three convolutional layers is well adapted for the classification of handwritten images as shown in Figure 1. It consists of the input layer, multiple hidden layers (repetitions of convolutional, normalization, pooling) and a fully connected and an output layer.

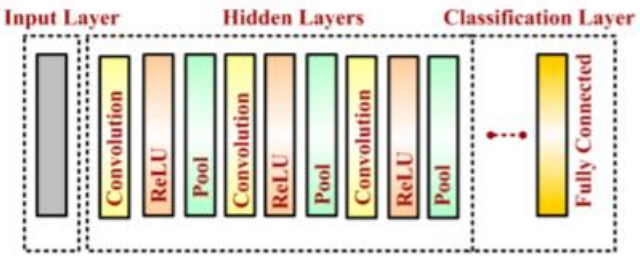


Fig 1: Convolutional Neural Network Layer

Input Layer:

The input data is loaded and stored in the input layer. This layer describes the height, width and number of channels (RGB information) of the input image.

Hidden Layer :

The hidden layers are the backbone of CNN architecture. They perform a feature extraction process where a series of convolution, pooling and activation functions are used. The distinguishable features of handwritten digits are detected at this stage.

Convolutional Layer :

The convolutional layer is the first layer placed above the input image. It is used for extracting the features of an image. The n × n input neurons of the input layer are convoluted with an m × m filter and in return deliver (n − m + 1) × (n − m + 1) as output. It introduces non-linearity through a neural activation function. The main contributors of the convolutional layer are receptive field, stride, dilation and padding, as described in the following paragraph. CNN computation is inspired by the visual cortex in animals . The visual cortex is a part of the brain that processes the information forwarded from the retina. It processes visual information and is subtle to small sub-regions of the input. Similarly, a receptive field is calculated in a CNN, which is a small region of an input image that can affect a specific region of the network. It is also one of the important design parameters of the CNN architecture and helps in setting other CNN parameters . It has the same size as the kernel and works in a similar fashion as the foveal vision of the human eye works for producing sharp central vision.

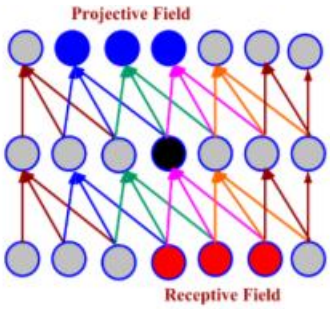


Fig 2: Receptive field and projective field

Pooling Layer

A pooling layer is added between two convolutional layers to reduce the input dimensionality and hence to reduce the computational complexity. Pooling allows the selected values to be passed to the next layer while leaving the unnecessary values behind. The pooling layer also helps in feature selection and in controlling overfitting. The pooling operation is done independently. It works by extracting only one output value from the tiled non-overlapping sub-regions of the input images. The common types of pooling operations are max-pooling and avg-pooling (where max and avg represent maxima and average, respectively). The max-pooling operation is generally favorable in modern applications, because it takes the maximum values from each sub-region, keeping maximum information. This leads to faster convergence and better generalization . The max-pooling operation for converting a 4 × 4 convolved output into a 2 × 2 output with stride size 2 .. The maximum number is taken from each convolved output (of size 2 × 2) resulting in reducing the overall size to 2 ×2.

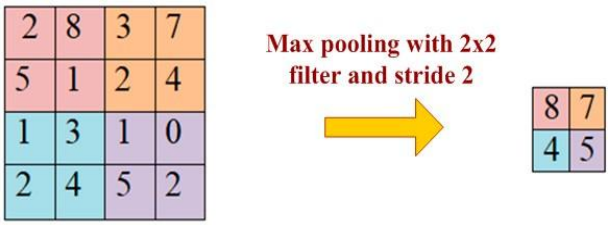


Fig 3. Max pooling with 2*2 filter

Classification Layer: The classification layer is the last layer in CNN architecture. It is a fully connected feed forward network, mainly adopted as a classifier. The neurons in the fully connected layers are connected to all the neurons of the previous layer. This layer calculates predicted classes by identifying the input image, which is done by combining all the features learned by previous layers. The number of output classes depends on the number of classes present in the target dataset. In the present work, the classification layer uses the 'softmax' activation function for classifying the generated features of the input image received from the previous layer into various classes based on the training data.

Code:

Train.py:

```
import numpy as np
import cv2
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from keras.layers import Dropout,Flatten
from keras.layers.convolutional import Conv2D,MaxPooling2D
import pickle

#path = 'MNIST Dataset/MNIST Dataset/MNIST - JPG - training'
path = 'myData'
testRatio = 0.2
valRatio = 0.2
imageDimensions= (32,32,3)
batchSizeVal= 100
epochsVal = 3
stepsPerEpochVal = 10

count = 0
images = []     # LIST CONTAINING ALL THE IMAGES
classNo = []    # LIST CONTAINING ALL THE CORRESPONDING CLASS ID OF IMAGES
myList = os.listdir(path)
print("Total Classes Detected:",len(myList))
noOfClasses = len(myList)
print("Importing Classes!!!!!!!")
for x in range (0,noOfClasses):
    myPicList = os.listdir(path+"/"+str(x))
    for y in myPicList:
        curImg = cv2.imread(path+"/"+str(x)+"/"+y)
        curImg = cv2.resize(curImg,(32,32))
        images.append(curImg)
        classNo.append(x)
    print(x,end= " ")
print(" ")
print("Total Images in Images List = ",len(images))
print("Total IDS in classNo List= ",len(classNo))

images = np.array(images)
classNo = np.array(classNo)
print(images.shape)

#### SPLITTING THE DATA
X_train,X_test,y_train,y_test = train_test_split(images,classNo,test_size=testRatio)
X_train,X_validation,y_train,y_validation = train_test_split(X_train,y_train,test_size=valRatio)
print(X_train.shape)
print(X_test.shape)
print(X_validation.shape)

#### PLOT BAR CHART FOR DISTRIBUTION OF IMAGES
numOfSamples= []
for x in range(0,noOfClasses):
    #print(len(np.where(y_train==x)[0]))
    numOfSamples.append(len(np.where(y_train==x)[0]))
print(numOfSamples)
```

```
plt.figure(figsize=(10,5))
plt.bar(range(0,noOfClasses),numOfSamples)
plt.title("No of Images for each Class")
plt.xlabel("Class ID")
plt.ylabel("Number of Images")
plt.show()

def preProcessing(img):
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img = cv2.equalizeHist(img)
    img = img/255
    return img
# img = preProcessing(X_train[30])
# img = cv2.resize(img,(300,300))
# cv2.imshow("PreProcesssed",img)
# cv2.waitKey(0)

X_train= np.array(list(map(preProcessing,X_train)))
X_test= np.array(list(map(preProcessing,X_test)))
X_validation= np.array(list(map(preProcessing,X_validation)))


X_train = X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)
X_validation = X_validation.reshape(X_validation.shape[0],X_validation.shape[1],X_validation.shape[2],1)

dataGen = ImageDataGenerator(width_shift_range=0.1,
                             height_shift_range=0.1,
                             zoom_range=0.2,
                             shear_range=0.1,
                             rotation_range=10)
dataGen.fit(X_train)

y_train = to_categorical(y_train,noOfClasses)
y_test = to_categorical(y_test,noOfClasses)
y_validation = to_categorical(y_validation,noOfClasses)

def myModel():
    noOfFilters = 60
    sizeOfFilter1 = (5,5)
    sizeOfFilter2 = (3, 3)
    sizeOfPool = (2,2)
    noOfNodes= 500

    model = Sequential()
    model.add((Conv2D(noOfFilters,sizeOfFilter1,input_shape=(imageDimensions[0],
                      imageDimensions[1],1),activation='relu')))
    model.add((Conv2D(noOfFilters, sizeOfFilter1, activation='relu')))
    model.add(MaxPooling2D(pool_size=sizeOfPool))
    model.add((Conv2D(noOfFilters//2, sizeOfFilter2, activation='relu')))
    model.add((Conv2D(noOfFilters//2, sizeOfFilter2, activation='relu')))
    model.add(MaxPooling2D(pool_size=sizeOfPool))
    model.add(Dropout(0.5))

    model.add(Flatten())
```

```
    model.add(Dense(noOfNodes,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(noOfClasses, activation='softmax'))

    model.compile(Adam(learning_rate=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
    return model

model = myModel()
print(model.summary())

history = model.fit(dataGen.flow(X_train,y_train,
                                 batch_size=batchSizeVal),
                                 steps_per_epoch=stepsPerEpochVal,
                                 epochs=epochsVal,
                                 validation_data=(X_validation,y_validation),
                                 shuffle=1)

plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('Loss')
plt.xlabel('epoch')
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Accuracy')
plt.xlabel('epoch')
plt.show()

score = model.evaluate(X_test,y_test,verbose=0)
print('Test Score = ',score[0])
print('Test Accuracy =', score[1])

pickle_out= open("reviewtest.p", "wb")
pickle.dump(model,pickle_out)
pickle_out.close()
```

Test.py:

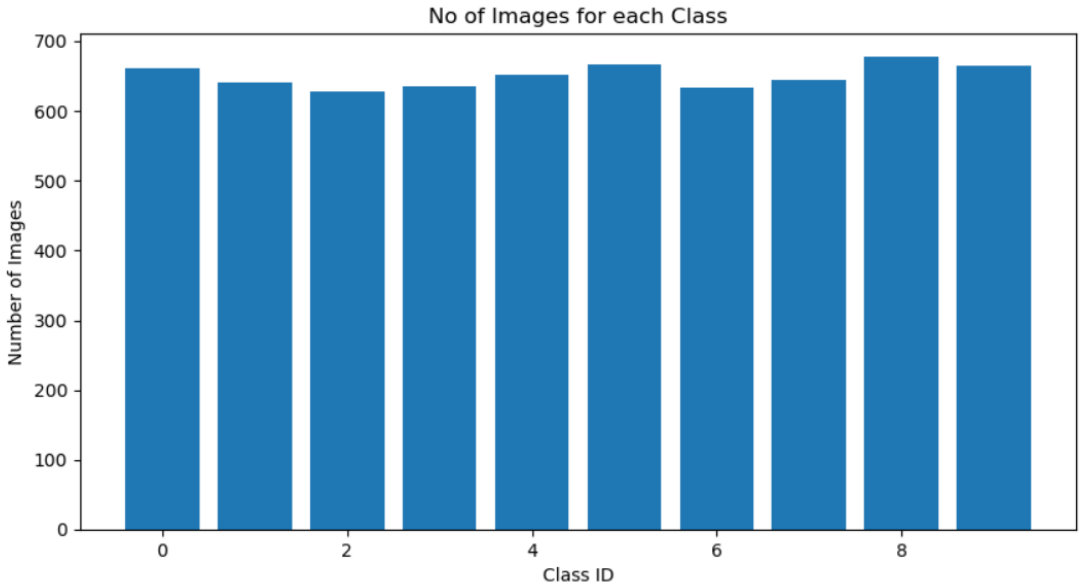
```
import numpy as np
import cv2
import pickle


width = 640
height = 480
threshold = 0.65 # MINIMUM PROBABILITY TO CLASSIFY
cameraNo =1
cap = cv2.VideoCapture(cameraNo)
cap.set(3,width)
```

```
cap.set(4,height)

#pickle_in = open("model_trained_20_2000.p","rb")
pickle_in = open("model_trained_minst2.p","rb")
model = pickle.load(pickle_in)

def preProcessing(img):
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img = cv2.equalizeHist(img)
    img = img/255
    return img

while True:
    success, imgOriginal = cap.read()
    img = np.asarray(imgOriginal)
    img = cv2.resize(img,(32,32))
    img = preProcessing(img)
    cv2.imshow("Processsed Image",img)
    img = img.reshape(1,32,32,1)

    classIndex = int(model.predict_classes(img))
    predictions = model.predict(img)
    probVal= np.amax(predictions)
    print(classIndex,probVal)

    if probVal> threshold:
        cv2.putText(imgOriginal,str(classIndex) + "   "+str(probVal),
                    (50,50),cv2.FONT_HERSHEY_COMPLEX,
                    1,(0,0,255),1)

    cv2.imshow("Original Image",imgOriginal)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

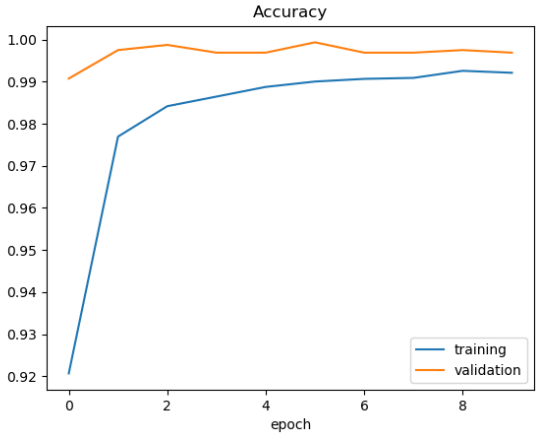
Outputs:

Printed digits:

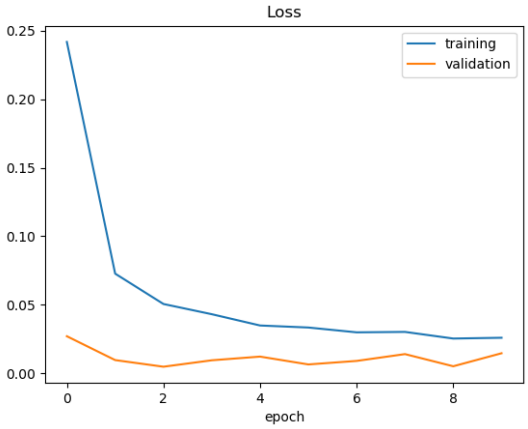
Distribution of the Inputs:



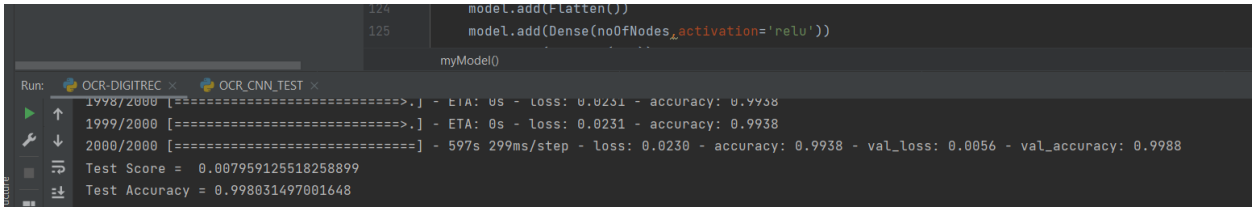
Training accuracy:



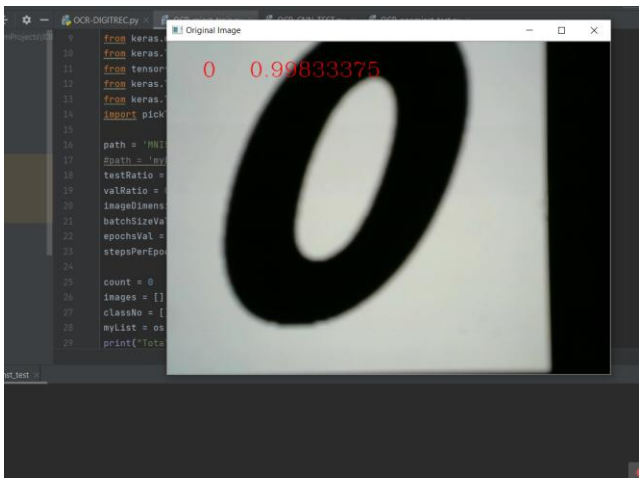
Training Loss:



Final Accuracy:



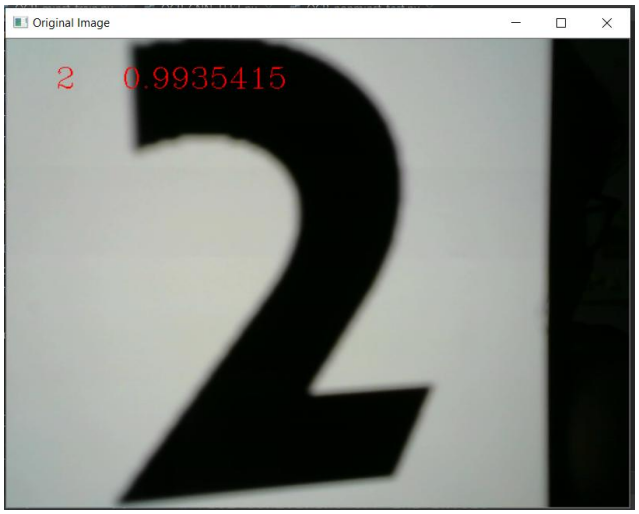
0:



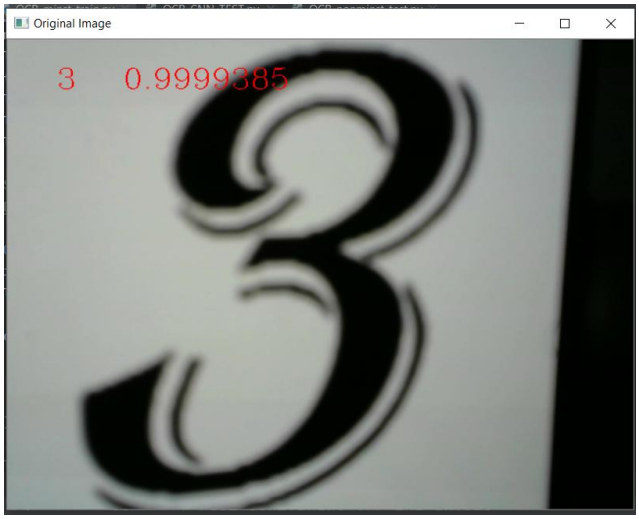
1:


Original Image
1 0.9864031

2:

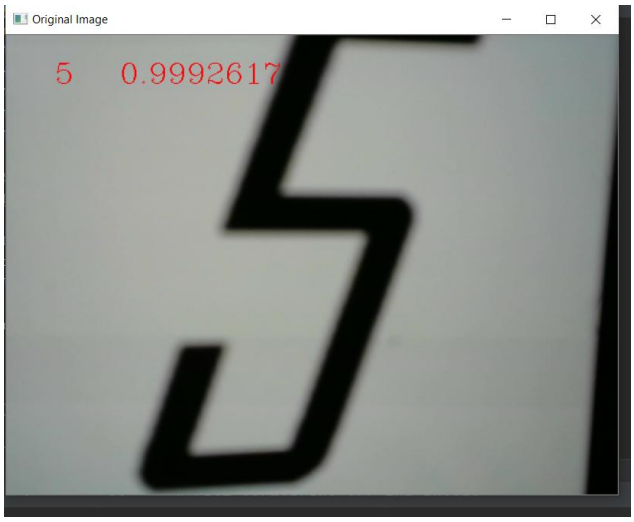

Original Image
2 0.9935415

3:

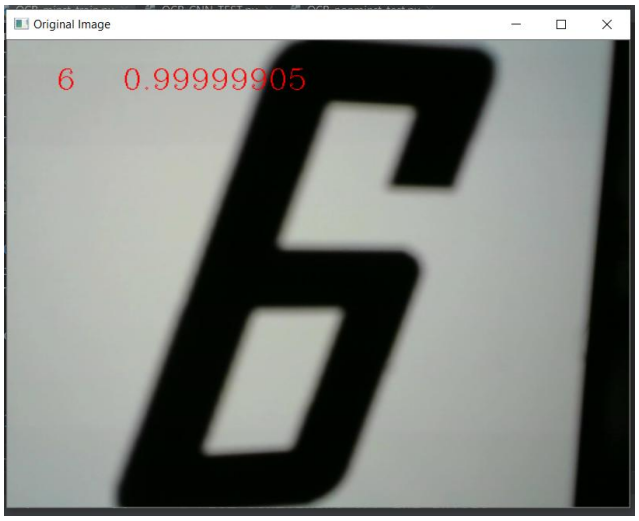

Original Image
3 0.9999385

4:

5:



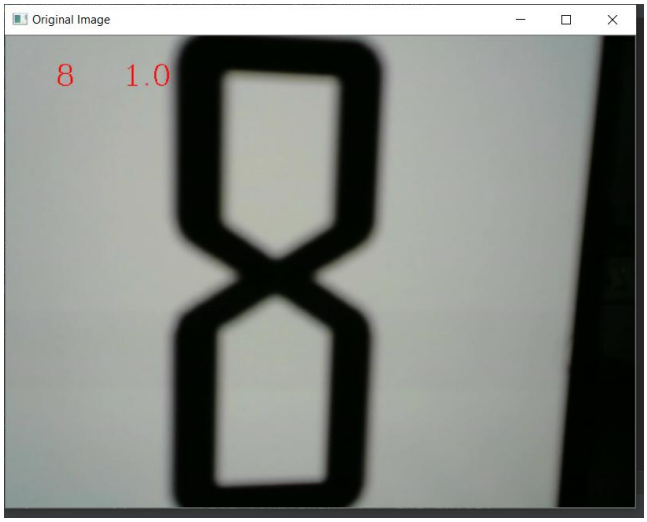
6:


Original Image
6 0.99999905

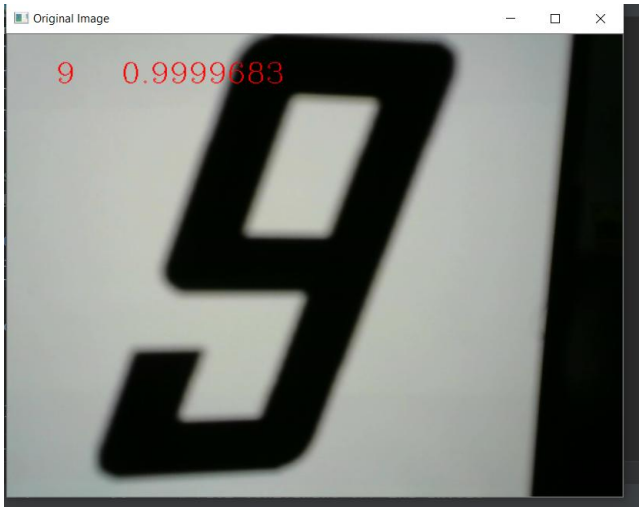
7:


Original Image
7 0.9999982

8:

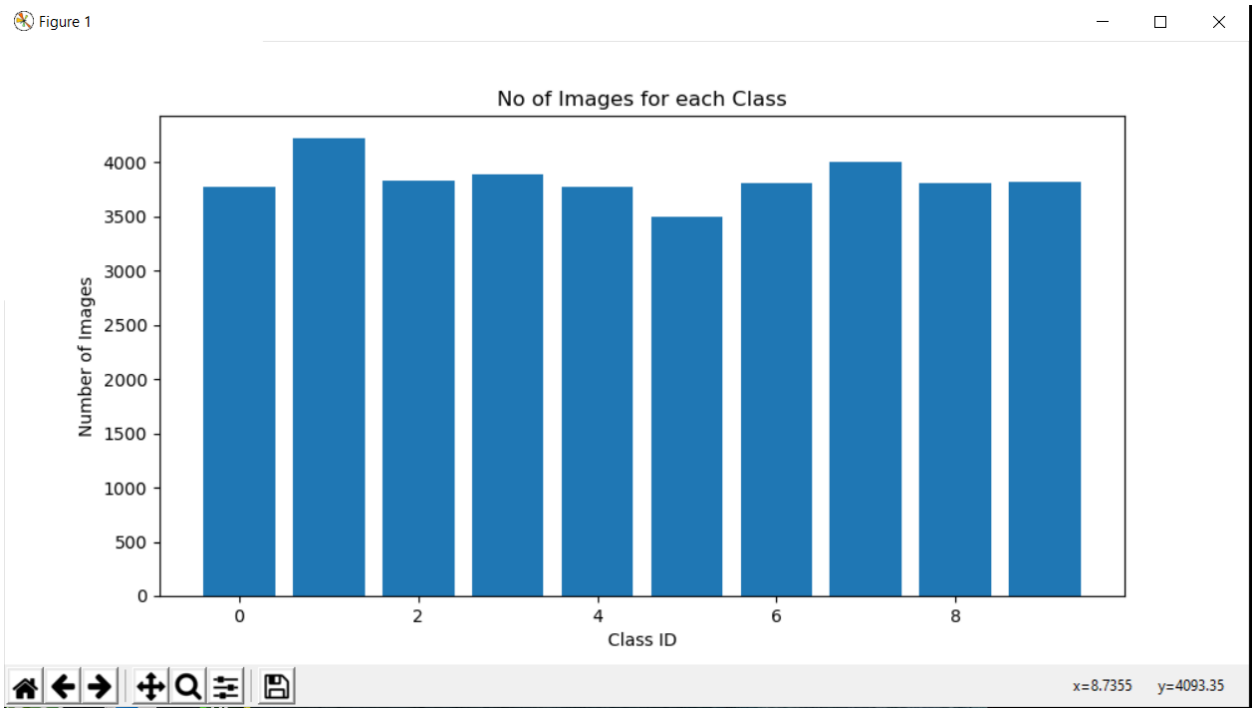

Original Image
8 1.0

9:

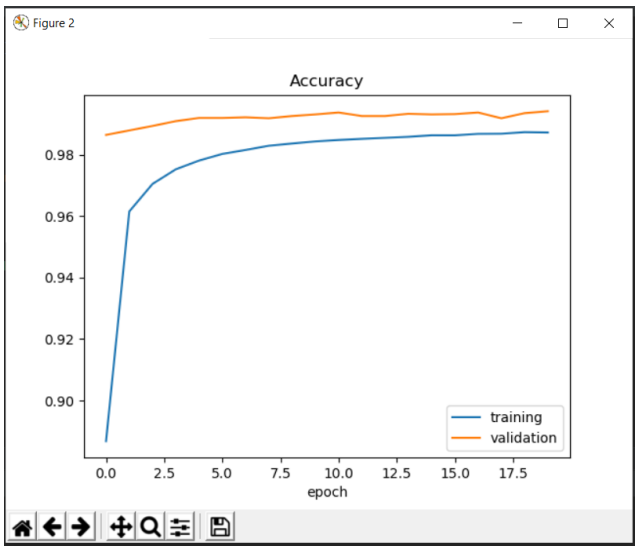


Hand written digits:

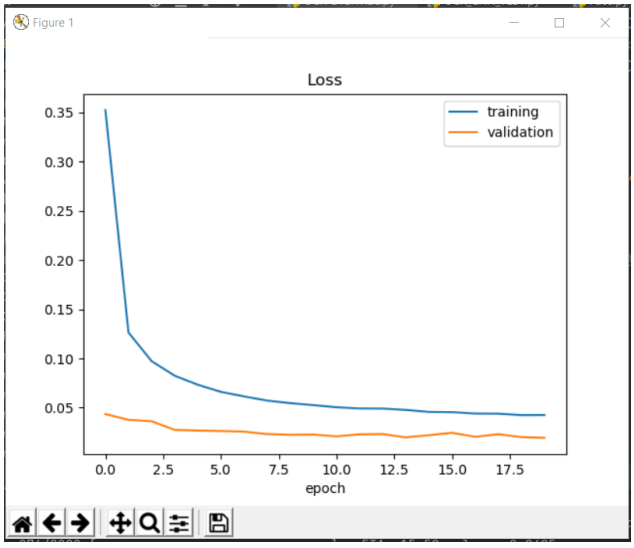
Distribution of the Inputs:



Training Accuracy:



Training Loss:

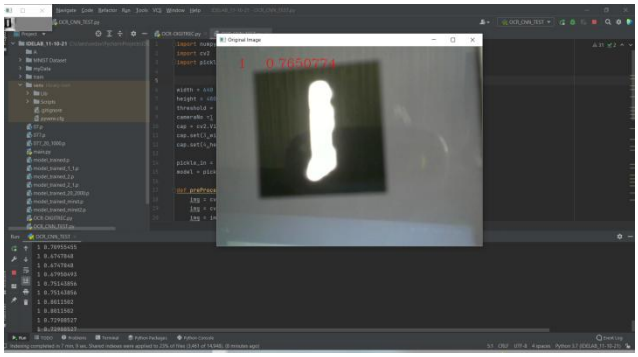


Final Accuracy:

```
 999/1000 [============================>.] - ETA: 0s - loss: 0.0480 - accuracy: 0.9854
1000/1000 [==============================] - 746s 746ms/step - loss: 0.0479 - accuracy: 0.9855 - val_loss: 0.0265 - val_accuracy: 0.9920
Test Score =  0.026177879404819 88
Test Accuracy = 0.9920833110809326
```

0:

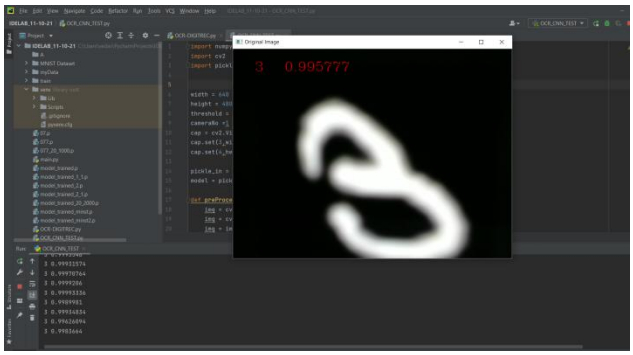
1:



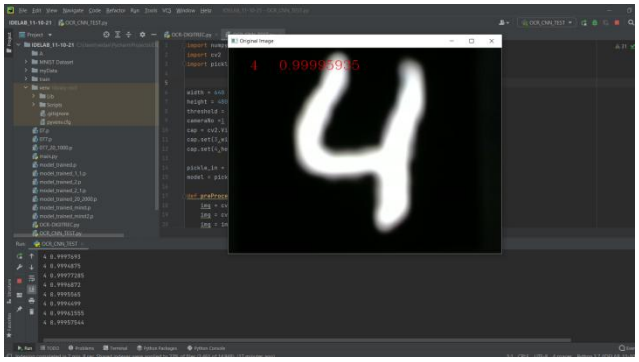
2:


Original Image
2 0.9990007

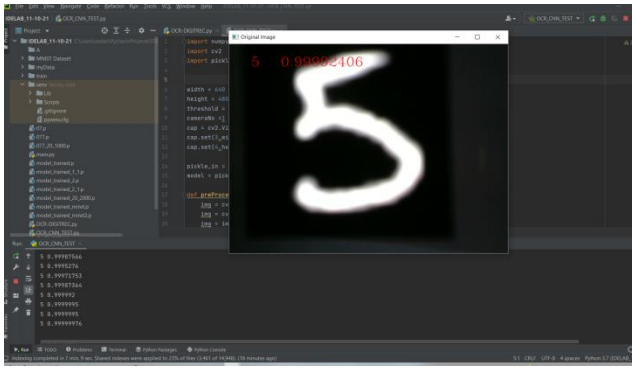
3:

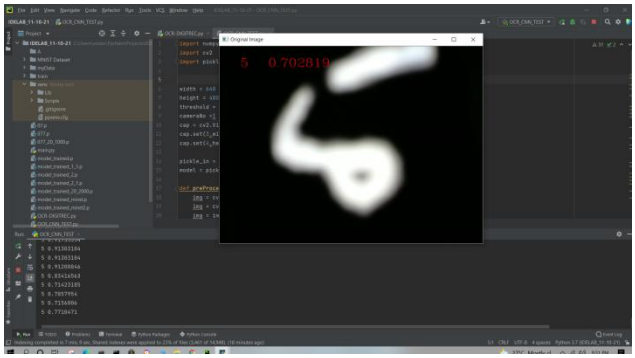

Original Image
3 0.995777

4:

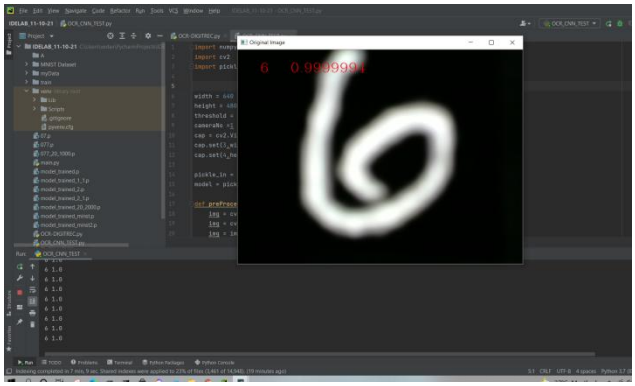

Original Image
4 0.9999385

5:

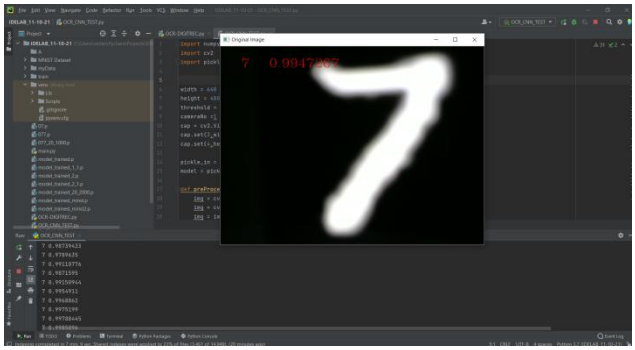

Original Image
5 0.99992406


Original Image
5 0.702819

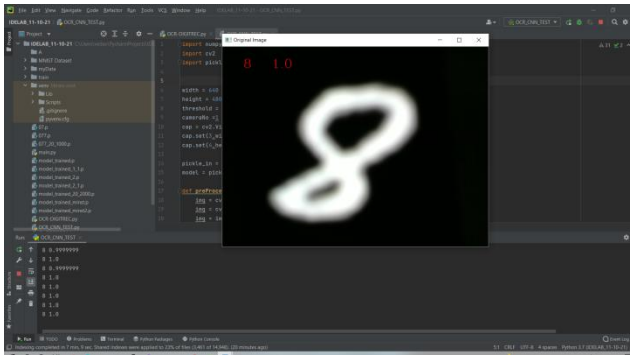
6:


Original Image
6 0.9999994

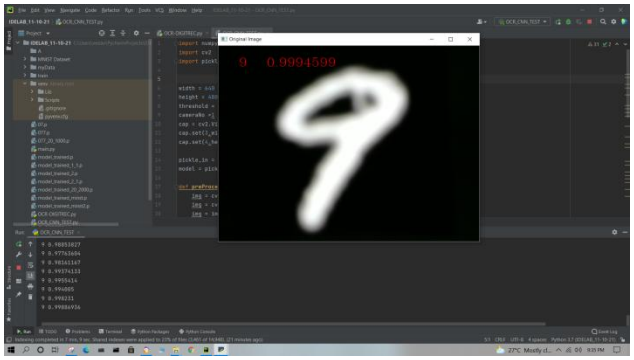
7:



8:



9:



Conclusion and Future Scope:

After putting forth a lot of work to distribute and collect digit forms from hundreds of elementary, high school, and college students, a significant dataset was gathered. After discovering that there are few and not particularly difficult MNIST digit datasets, I expended considerable effort in locating the proper data set.

In addition, the obtained data is trained using a CNN model that represents the current state-of-the-art for a number of applications. As a result, I thoroughly investigated the model by carefully picking its parameters and demonstrating its robustness in handling our dataset.

In future I would like to extend the model from 0 -9 to 0 to 100 and train the data with **Recurrent neural network** (**RNN**)