

Optical Character Recognition System

Prepared by
Dhavalbhai Barevadiya (16IT005)

Under the supervision of
Prof. Pinal Shah

A Report Submitted to
Charotar University of Science and Technology
for Partial Fulfillment of the Requirements for the
Degree of Bachelor of Technology
in Information Technology

IT345 Software Group Project-II (5th sem)

Submitted at



DEPARTMENT OF INFORMATION TECHNOLOGY

Chandubhai S. Patel Institute of Technology

At: Changa, Dist: Anand – 388421

November 2018

CERTIFICATE

This is to certify that the report entitled “**Optical Character Recognition System**” is a bonafied work carried out by **Mr. Dhavalbhai Barevadiya** under the guidance and supervision of **Prof. Pinal Shah** for the subject **Software Group Project-II(IT345)** of 5th Semester of Bachelor of Technology in **Information Technology** at Faculty of Technology & Engineering – CHARUSAT, Gujarat.

To the best of my knowledge and belief, this work embodies the work of candidate **himself**, has duly been completed, and fulfills the requirement of the ordinance relating to the B.Tech. Degree of the University and is up to the standard in respect of content, presentation and language for being referred to the examiner.

Under supervision of,

Prof. Pinal Shah
Assistant Professor
Dept. of Information Technology
CSPIT, Changa, Gujarat.

Prof. Parth Shah
Head & Associate Professor
Department of Information Technology
CSPIT, Changa, Gujarat.

Chandubhai S Patel Institute of Technology

At: Changa, Ta. Petlad, Dist. Anand, PIN: 388 421. Gujarat

ACKNOWLEDGEMENT

I hereby take this opportunity to thank each and every one who has helped me in creating this project. I especially thank **Prof. Sanket Suthar** for guiding me through the whole period of preparation and presentation. I express my gratitude towards my guide **Prof. Pinal Shah and HOD Parth Shah**, for giving me moral and academic support. At last I thank all those who directly or indirectly helped me in preparing the report. I would like to thank all my friends, colleagues and classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. Last but not least I would like to thank my family members who provide us enormous support during this works directly and indirectly.

ABSTRACT

Optical Character Recognition system is used to convert text in an image into text format. This system smartly recognizes text from an image. This is also trained to recognize handwritten text with an accuracy of about 40-50%. It can read almost all types of image formats including JPEG, PNG, BMP. The solution developed for this project consists of three major parts: image pre-processing, character recognition (using deep learning based CNN), character segmentation and result presentation. This System can be used on Toll Booths to read number plates of different cars passing through the booth. This can save manual hard work and time.

TABLE OF CONTENTS

• Abstract.....	1
• Acknowledgement.....	3
• Chapter 1 Introduction.....	5
1.1 Project Overview	5
1.2 Problem Statement and solution.....	5
1.2 Scope	5
1.3 Objective	5
• Chapter 2 System Analysis.....	6
2.1 User Characteristics.....	6
2.2 Tools & Technology used	6
• Chapter 3 System Design.....	7
3.1 Flow of System.....	7
3.1 Major Functionality	8
• Chapter 4 Implementation.....	9
4.1 Implementation Strategy	9
4.2 Deep Convolution Neural Network for character recognition	10
4.3 Coding Standards	13
4.4 Snapshots of project	17
• Chapter 5 Limitations and Future Enhancement.....	20
• Chapter 6 Conclusion	21
• References.....	22

LIST OF FIGURE

• Fig 3.1 Project flow.....	9
• Fig 4.1 CNN architecture.....	13
• Fig 4.2 Loading character dataset	19
• Fig 4.3 Model training	20
• Fig 4.4 Snapshot of output	21

1. INTRODUCTION

- **1.1 Project Overview**

Optical Character recognition is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a scene-photo or from subtitle text superimposed on a image.

- **1.2 Problem Statement and Solution**

Many times we need to write text from some hard paper like magazine, newspaper or some images. They could all involve you spending hours retyping manually and correcting typos. Or you could take a more modern approach and convert any and all of them into a digital format with fully editable text in a matter of minutes. Using OCR we can capture the image of that page and give that to OCR engine, OCR will convert the text from that image.

- **1.3 Scope**

Optical Character Recognition system can read printed and handwritten text from images and can also capture images through webcam. Printed characters are read with an accuracy of about 90% and handwritten characters are read with an accuracy of 40-50%.

- **1.4 Objective**

The goal of Optical Character Recognition (OCR) is to classify optical patterns (often contained in a digital image) corresponding to alphanumeric or other characters.

2. SYSTEM ANALYSIS

- **2.1 User Characteristics**

- Layman with basic computer knowledge can use this software.

- **2.2 Tools and Technologies used**

- Python
- Spyder IDE
- Convolutional Neural Networks
- OpenCV 3.4.2
- Tensorflow (python API for implementing deep learning)
- Keras (high level neural network API in python)

3. SYSTEM DESIGN

- **3.1 Flow of System**

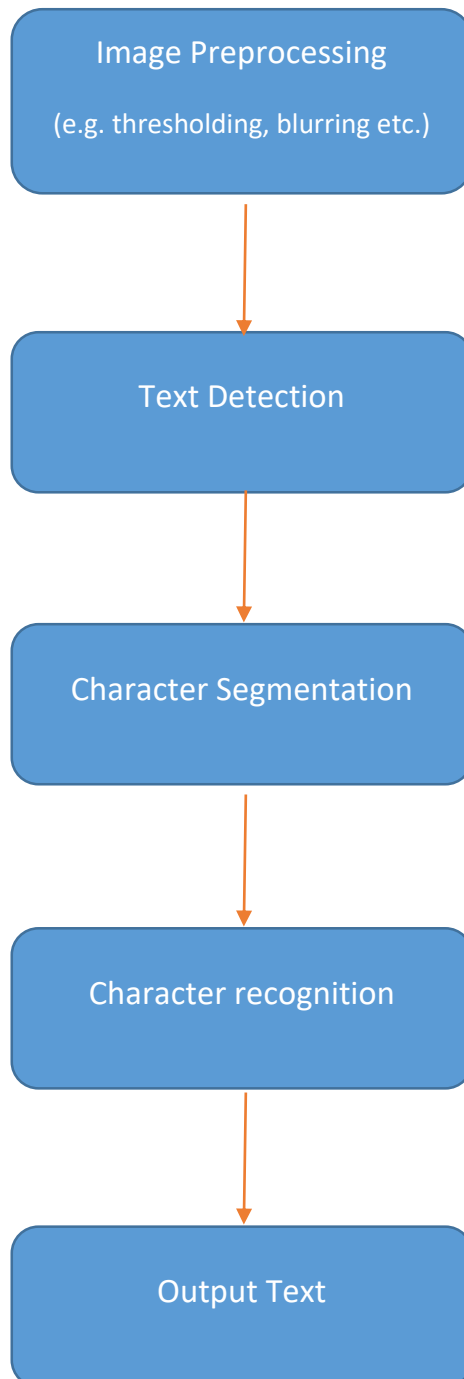


Figure 3.1 / Project Flow

- **3.2 Major Functionality**

Optical Character Recognition (OCR), is very helpful software that converts scanned or photographed documents into text form, which is then readable via computer. It takes scanning to the next level ! It is an excellent option when the original data is on paper. Business cards, passport information, receipts, bank receipts, old paperwork and important documents all benefit from OCR. Great software opens up new possibilities and new options, and with OCR software, you'll discover a new way to approach all of your document based information.

4. IMPLEMENTATION

• 4.1 Implementation Strategy

It will take 4 steps to compute the output from given input image. Which are :

- **Image Preprocessing :** In this stage input image will go through the various image preprocessing techniques like thresholding, blurring, noise removal etc. This stage is important because it removes some extra unnecessary information from input image.
- **Text Detection :** Text detection from image is done using opencv's EAST text detector which give detected text.
- **Character Segmenation :** After detection we have to segment each character from detected text. The segmentation is done with opencv's contour detection.
- **Character Recognition :** The each segmented character will now pass to the our trained model which will predict the character. The model is trained using convolutional neural networks. The model is trained on 4627 images and validated on 1157 images.
- **Output text :** After recognition we have to collect each predicted character and make it in the text form.

- **4.2 Deep Convolutional Neural Networks for character recognition**

In a regular Neural Network there are three types of layers

- **Input Layers:** It's the layer in which we give input to our model. The number of neurons in this layer is equal to total number of features in our data (number of pixels incase of an image).
- **Hidden Layer:** The output from Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layers can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by addition of learnable biases followed by activation function which makes the network nonlinear.
- **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into probability score of each class.
- The data is then fed into the model and output from each layer is obtained, this step is called feedforward, we then calculate the error using an error function, some common error functions are cross entropy, square loss error etc. After that, we backpropagate into the model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.
- **Convolutional Neural Network**
Convolution Neural Networks are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image) and height (as image generally have red, green, and blue channels).

Now imagine taking a small patch of this image and running a small neural network on it, with say, k outputs and represent them vertically. Now slide that neural

network across the whole image, as a result, we will get another image with different width, height, and depth. Instead of just R, G and B channels now we have more channels but lesser width and height. this operation is called Convolution. If patch size is same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

Now let's talk about a bit of mathematics which is involved in the whole convolution process.

Convolution layers consist of a set of learnable filters (patch in the above image). Every filter has small width and height and the same depth as that of input volume (3 if the input layer is image input). For example, if we have to run convolution on an image with dimension $34 \times 34 \times 3$. Possible size of filters can be $a * a * 3$, where 'a' can be 3, 5, 7, etc but small as compared to image dimension.

During forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have value 2 or 3 or even 4 for high dimensional images) and compute the dot product between the weights of filters and patch from input volume.

As we slide our filters we'll get a 2-D output for each filter and we'll stack them together and as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

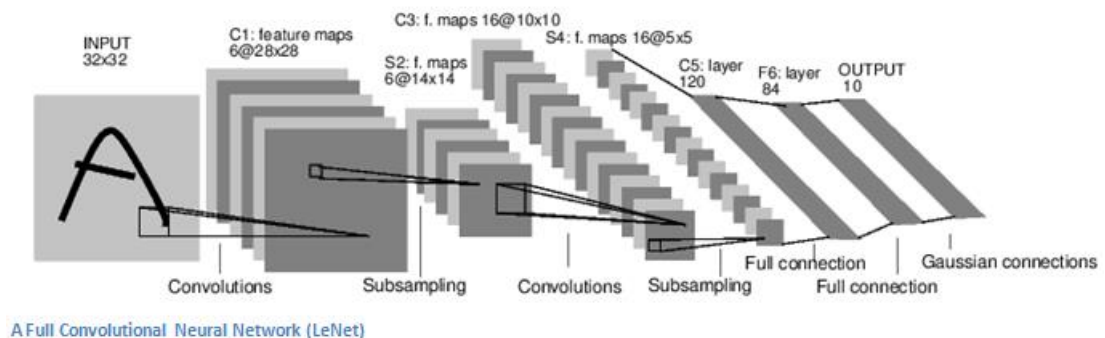


Figure 4.1 | CNN Structure

- **Types Of Layeres used in our training Model**

- **Input Layer:** This layer holds the raw input of image with width 28, height 28 and depth 3.
- **Convolution Layer:** This layer computes the output volume by computing dot product between all filters and image patch. Suppose, we have $32 \times 32 \times 3$ size image and we apply 6 filters then we'll get output of 6 feature maps of $28 \times 28 \times 1$ size after applying a convolution layer.
- **Activation Function Layer:** This layer will apply element wise activation function to the output of convolution layer. We have used ReLU in our trained model. The purpose of ReLU layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers. In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better than others.
- **Pool Layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents from overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2×2 filters and stride 2 on $32 \times 32 \times 12$ size filters, the resultant volume will be of dimension $16 \times 16 \times 12$. We have used maxpooling in our model.
- **Dropout Layer :** Now, dropout layers have a very specific function in neural networks. In the last section, we discussed the problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. The idea of dropout is simplistic in nature. This layer "drops out" a random set of activations in that layer by setting them to zero. It forces the network to be redundant. The network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out.

- **4.3 Coding Standards**

- **Training.py:**

```
import numpy as np
import pickle
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
import random
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

from sklearn.model_selection import train_test_split
from keras.utils import np_utils

DATADIR = "E:\\Datasets\\MyDataset"

CATEGORIES =
["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

IMG_SIZE = 28

#create training data
training_data = []

def create_training_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category) # create path
        to dataset
        class_num = CATEGORIES.index(category)

        for img in tqdm(os.listdir(path)): # iterate over
            each image per alphabets
            try:
                img_array =
cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                img_array = cv2.GaussianBlur(img_array,
(3,3), 0)

                img_array = cv2.adaptiveThreshold(img_array,
255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV, 7, 10)
```

```
        img = cv2.resize(img_array, (IMG_SIZE,
IMG_SIZE))
        training_data.append([img, class_num])
    except Exception as e:
        pass

#loading data from dataset folders
create_training_data()

#print(len(training_data))

pickle_out = open("Thresholded_Blurred_data.pickle","wb")
pickle.dump(training_data, pickle_out)
pickle_out.close()

#randomize data
random.shuffle(training_data)
for sample in training_data[:10]:
    print(sample[1])

X = []
Y = []

for features,label in training_data:
    X.append(features)
    Y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)

seed = 785

# split the data into training and testing
(X_train, X_test, Y_train, Y_test) = train_test_split(X, Y,
test_size=0.20, random_state=seed)

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')

X_train = X_train / 255
X_test = X_test / 255

Y_train = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)

num_classes = Y_test.shape[1]
```



```

#model training
model = Sequential()
model.add(Conv2D(64, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
model.add(Conv2D(64, (3, 3), input_shape=(26, 26, 1),
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.05))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
model.fit(X_train, Y_train, validation_data=(X_test,
Y_test), epochs=15, batch_size=100, verbose=2)

#model evaluation
val_loss, val_acc = model.evaluate(X_test, Y_test)
print(val_loss)
print(val_acc)

#save the model
model.save('Thresholded_Blurred_Trained')

#load saved model
new_model =
tf.keras.models.load_model('Thresholded_Blurred_Trained')

#predict image on test data
predictions = new_model.predict(X_test)

print(CATEGORIES[np.argmax(predictions[505])])
img = cv2.resize(X_test[505], (28,28))
plt.imshow(img)

```

- **Main.py**

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

CATEGORIES =
["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O"
, "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

```

```
new_model =
tf.keras.models.load_model('Thresholded_Blurred_Trained')
img = cv2.imread('myTestImages/string/shridhar.jpg',
cv2.IMREAD_GRAYSCALE)
img = cv2.GaussianBlur(img, (5, 5), 0)
img = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_MEAN_C,
                        cv2.THRESH_BINARY_INV, 13, 10)
plt.imshow(img)
image, contours, hier = cv2.findContours(img,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
contours = sorted(contours, key=lambda ctr:
cv2.boundingRect(ctr)[0])
d=0
string = ""
for ctr in contours:
    # Get bounding box
    x, y, w, h = cv2.boundingRect(ctr)
    #print(x, y, w, h)
    if h > 4 and w > 2:
        # Getting ROI
        roi = image[y-1:y+h+2, x-1:x+w+2]
        roi = cv2.resize(roi, (28,28))

        #pass the data into model
        roi = np.expand_dims(roi, axis = 0)
        roi = np.expand_dims(roi, axis = 3)
        roi =
roi.reshape(roi.shape[0],28,28,1).astype('float32')
        roi = roi / 255

        predictions = new_model.predict(np.array(roi))
        string += CATEGORIES[np.argmax(predictions[0])]
print("Predicted string is : " ,string)
```

• 4.4 Snapshots of project

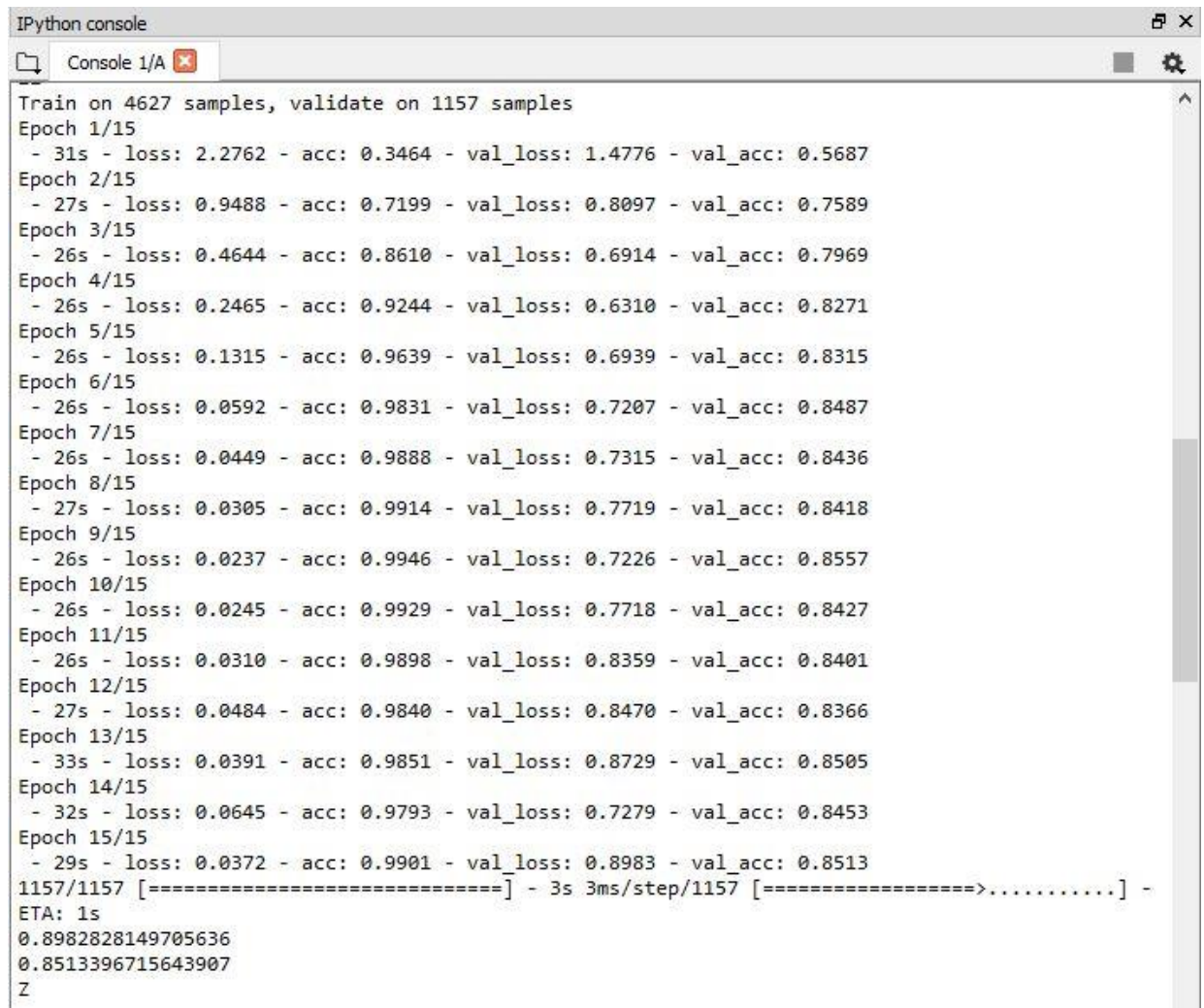
```

IPython console
Console 1/A

In [1]: runfile('E:/Dhaval/study/IT sem 5/IT 345 SGP-II/opencv practice/Project_work/
myDatasetTraining.py', wdir='E:/Dhaval/study/IT sem 5/IT 345 SGP-II/opencv practice/
Project_work')
C:\Users\Dhaval\Anaconda3\lib\site-packages\h5py\_init_.py:34: FutureWarning: Conversion of the
second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be
treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
100%|██████████| 227/227 [00:03<00:00, 74.52it/s]
100%|██████████| 240/240 [00:01<00:00, 140.84it/s]
100%|██████████| 209/209 [00:02<00:00, 69.96it/s]
100%|██████████| 236/236 [00:01<00:00, 181.49it/s]
100%|██████████| 222/222 [00:01<00:00, 146.06it/s]
100%|██████████| 214/214 [00:01<00:00, 153.95it/s]
100%|██████████| 234/234 [00:01<00:00, 158.87it/s]
100%|██████████| 206/206 [00:01<00:00, 175.80it/s]
100%|██████████| 229/229 [00:02<00:00, 98.88it/s]
100%|██████████| 230/230 [00:02<00:00, 103.59it/s]
100%|██████████| 232/232 [00:01<00:00, 182.74it/s]
100%|██████████| 230/230 [00:01<00:00, 184.10it/s]
100%|██████████| 233/233 [00:01<00:00, 188.80it/s]
100%|██████████| 224/224 [00:01<00:00, 178.44it/s]
100%|██████████| 226/226 [00:01<00:00, 157.48it/s]
100%|██████████| 224/224 [00:01<00:00, 183.84it/s]
100%|██████████| 223/223 [00:01<00:00, 174.88it/s]
100%|██████████| 226/226 [00:01<00:00, 200.92it/s]
100%|██████████| 240/240 [00:01<00:00, 194.05it/s]
100%|██████████| 210/210 [00:01<00:00, 161.95it/s]
100%|██████████| 203/203 [00:01<00:00, 187.63it/s]
100%|██████████| 188/188 [00:00<00:00, 200.22it/s]
100%|██████████| 217/217 [00:01<00:00, 177.87it/s]
100%|██████████| 214/214 [00:01<00:00, 201.46it/s]
100%|██████████| 223/223 [00:01<00:00, 175.34it/s]
100%|██████████| 224/224 [00:01<00:00, 214.31it/s]
15
1
12
5
10

```

Figure 4.2 | Loading character dataset



```
IPython console
Console 1/A
Train on 4627 samples, validate on 1157 samples
Epoch 1/15
- 31s - loss: 2.2762 - acc: 0.3464 - val_loss: 1.4776 - val_acc: 0.5687
Epoch 2/15
- 27s - loss: 0.9488 - acc: 0.7199 - val_loss: 0.8097 - val_acc: 0.7589
Epoch 3/15
- 26s - loss: 0.4644 - acc: 0.8610 - val_loss: 0.6914 - val_acc: 0.7969
Epoch 4/15
- 26s - loss: 0.2465 - acc: 0.9244 - val_loss: 0.6310 - val_acc: 0.8271
Epoch 5/15
- 26s - loss: 0.1315 - acc: 0.9639 - val_loss: 0.6939 - val_acc: 0.8315
Epoch 6/15
- 26s - loss: 0.0592 - acc: 0.9831 - val_loss: 0.7207 - val_acc: 0.8487
Epoch 7/15
- 26s - loss: 0.0449 - acc: 0.9888 - val_loss: 0.7315 - val_acc: 0.8436
Epoch 8/15
- 27s - loss: 0.0305 - acc: 0.9914 - val_loss: 0.7719 - val_acc: 0.8418
Epoch 9/15
- 26s - loss: 0.0237 - acc: 0.9946 - val_loss: 0.7226 - val_acc: 0.8557
Epoch 10/15
- 26s - loss: 0.0245 - acc: 0.9929 - val_loss: 0.7718 - val_acc: 0.8427
Epoch 11/15
- 26s - loss: 0.0310 - acc: 0.9898 - val_loss: 0.8359 - val_acc: 0.8401
Epoch 12/15
- 27s - loss: 0.0484 - acc: 0.9840 - val_loss: 0.8470 - val_acc: 0.8366
Epoch 13/15
- 33s - loss: 0.0391 - acc: 0.9851 - val_loss: 0.8729 - val_acc: 0.8505
Epoch 14/15
- 32s - loss: 0.0645 - acc: 0.9793 - val_loss: 0.7279 - val_acc: 0.8453
Epoch 15/15
- 29s - loss: 0.0372 - acc: 0.9901 - val_loss: 0.8983 - val_acc: 0.8513
1157/1157 [=====] - 3s 3ms/step/1157 [=====>.....] -
ETA: 1s
0.8982828149705636
0.8513396715643907
Z
```

Figure 4.3 | Model training

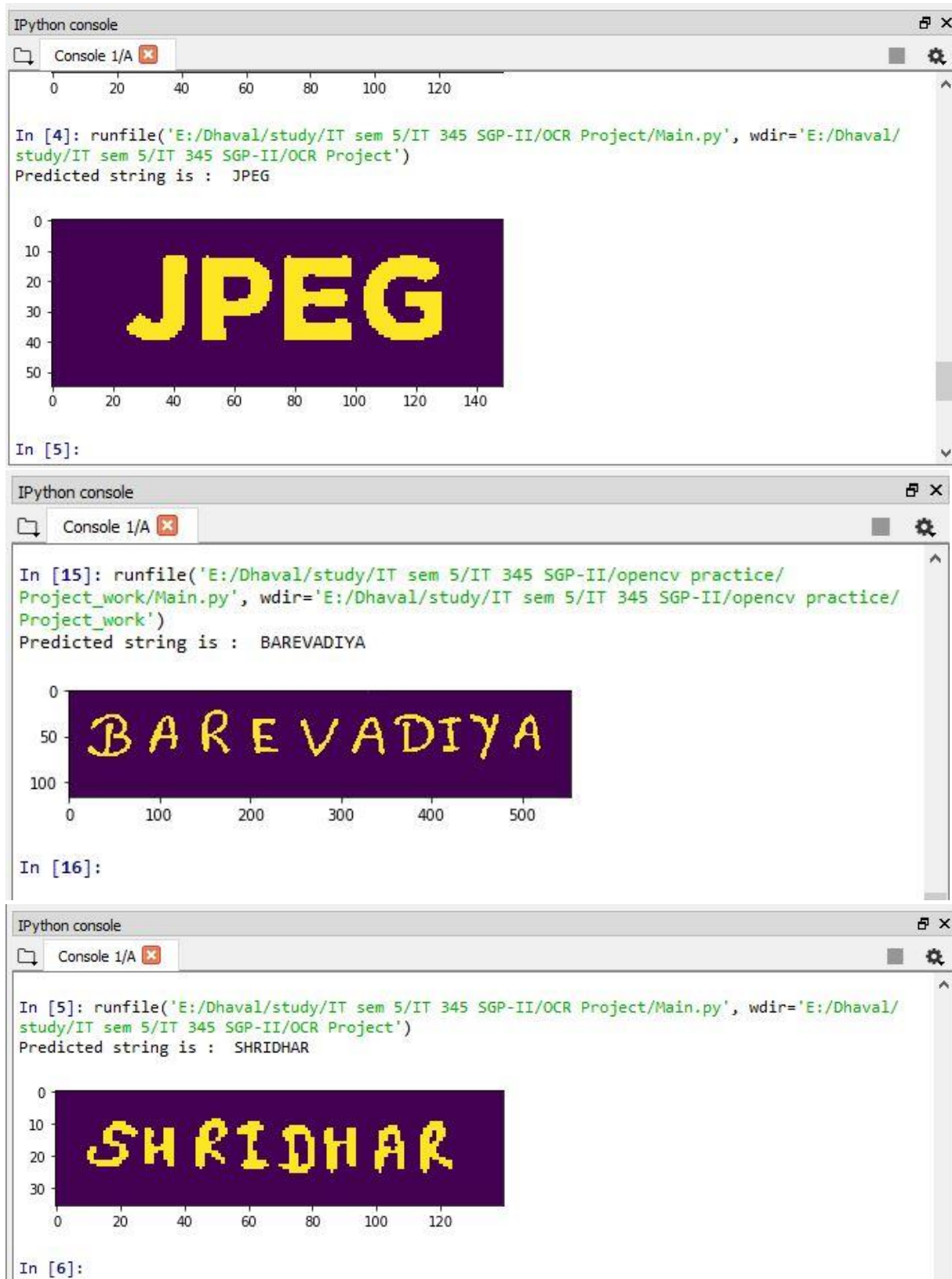


Figure 4.4 | Snapshots of output

5. Limitations and Future Enhancement

- **5.1 Limitations**

- Right now the segmentation accuracy is very low and because of this the model predicts some characters wrong.
- After doing thresholding image becomes too smooth to recognize by model correctly. Especially for handwritten characters.
- For noisy image the model will fail to predict right output.

- **5.2 Future Enhancement**

- Training accuracy is around 75% which can be improved using larger dataset.
- The segmentation algorithm needs many improvements to segment the character correctly.
- For noisy images we can use some noise removal techniques.
- A proper GUI Application can be made for better user experience.

6. Conclusion

- This project enabled me to learn more about python programming concepts and Image processing. I also learnt about some **machine learning** and **deep learning** concepts and steps for retrieving text from image.
- This project has given me the opportunity to learn high level APIs like **Tensorflow**, **Keras** and **Opencv**.
- I learnt how to be patient throughout the development phase and troubleshoot problems.

References

1. <https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/>
2. <https://pythonprogramming.net/>
3. <https://www.coursera.org/learn/convolutional-neural-networks/>
4. <https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/>
5. <https://neuralnetworksanddeeplearning.com/chap1.html>
6. <https://datascience.stackexchange.com/>
7. <https://stackoverflow.com/>
8. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
9. <https://www.kaggle.com/about/inclass/overview>
10. <https://www.geeksforgeeks.org/introduction-convolution-neural-network/>