International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 05 | May 2022 www.irjet.net p-ISSN: 2395-0072

Optical Recognition of Handwritten Text

Soham Bhagwat[1], Pratik Dharu[2], Abhishek Dixit[3], Bhadrayu Godbole[4]

[1,2,3,4]*Dept. of Computer Engineering, P.E.S. Modern College of Engineering, Maharashtra, India*

Abstract - *The project focus on the creation of OCR software for the off-line recognition of handwriting. OCR programs can recognize printed text with nearly perfect accuracy. The recognition of handwriting is harder due to the many different styles and inconsistent nature of handwriting. Handwritten text recognition (HTR) is an open field of research and a relevant problem that helps automatically process historical documents.*

In recent years great advances in deep learning and computer vision have allowed improvements on document and image processing including HTR. Handwritten text recognition plays an important role in the processing of vital information. Processing of digital files is cheaper than processing traditional paper files even though a lot of information is available on paper.

The aim of an OCR software is to convert handwritten text into machine readable formats. Despite such advances in this field, little has been done to produce open-source projects that address this problem as well as methods that utilize graphical process units (GPUs) to speed up the training phase.

Key Words: OCR, HTR, GPU, CNN, BRNN, CTC, DIA, MRZ

1. INTRODUCTION

Optical Character Recognition (OCR) deals with recognition of different characters from a given input that might include an image, a real time video, or a manuscript / document. By using OCR, one can transform the text into a digital format, thus allowing rapid scanning and digitization of documents in physical format as well as real time text-recognition (in case of videos). Similarly, this interpretation of OCR methodology involves pre-processing of input, text area detection, application of the best pre-trained models and finally, detection of text as an output. All of this is made possible with an offline software UI compatible with a Windows operating system.

1.1 Problem Definition and Objectives

To develop Optical Character Recognition (OCR) software for the recognition of handwriting using following algorithms:

- CNN
- BRNN
- CTC

1.2 Project Scope

Presently, OCR is capable in reading screenshots which has facilitated the transferring of information between incompatible technologies. By using OCR for handwritten text manual entries on paper will also be legible to computer systems. Additionally, OCR can be used to perform Document Image Analysis (DIA) by reading and recognizing text in research, governmental, academic, and business organizations that are having a large pool of documented, scanned images. Thirdly, OCR can be used to automate documentation and security processes at airports by automatically reading the Machine-Readable Zone (MRZ) and other relevant parts of a passport. In this way, OCR has a scope in a wide range of applications.

1.3 Limitations

Having considered some of the benefits of using OCR software, it also comes along with its own shortcomings. To begin with, straight OCR without additional AI or technology specifically trained to recognize ID types will lack the requisite accuracy one needs to deliver a good user experience. Thus, structuring the extracted/detected data involves more than just OCR. Secondly, considering pictures of ID documents - these images usually need to be de-skewed if the image was not aligned properly and reoriented so that the OCR technology can properly extract the data. Thus, OCR must combine with image rectification. Lastly, when there is glare or blurriness in the ID image, the probability of data extraction mistakes is significantly higher. Thus, glare and blur can cause mistakes.

2. SOFTWARE REQUIREMENTS SPECIFICATION

Mentioned below are some requirement specifications for the efficient working of the OCR software.

2.1 Assumptions and Dependencies

Before the commencement of the project, there are some assumptions that the project works with:

- The input image selected by the user is in a jpg/png format.
- The text to be detected and recognized is in English.
- The input image provided by the user is upright.

2.2 System Features

- The user should be able to choose and upload an image of his choice through the file browser window upon clicking of the upload image button on the software UI.

- The user should be able to get instant output of the chosen image in the output window, and user should also be shown intermediate stages in output detection for better understanding upon clicking of the button successively on the software UI.

2.3 External Interface Requirements

- The software is a single screen display where the user uploads an image as per interest.

- The image is cropped, resized and detected text output is displayed after the user presses indicated buttons.

- The software is designed to run on all PCs having at least a Windows 8 OS along with Python 3.9 installed.

- The frontend is managed using Tkinter library of Python while the backend is handled by os library of Python.

- The software runs on the saved model files that have been trained on a cloud infrastructure named Google Colaboratory.

2.3 Non-functional Requirements

- The models have already been trained and optimized on Google Colaboratory GPUs beforehand, so the performance requirements of users have been reduced to a minimum 8 GB RAM along with a suitable quad core processor and minimum 5 GB free HDD to make space for the whole software suite.

- With the above hardware specifications, a user takes around 30 to 50 seconds to get an output in the provided output window in the UI.

- As no user data is collected, there aren't any security concerns.

- As this software is offline, there isn't any vulnerability posed from the network side.

- The traces of data are deleted once the user closes the UI.

3. SYSTEM DESIGN

3.1 System Architecture

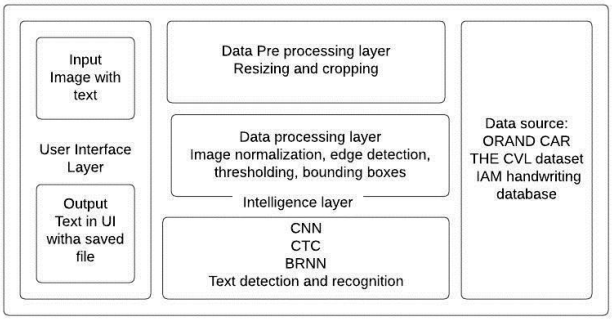


Fig -1: System architecture diagram

3.2 Use Case Diagram

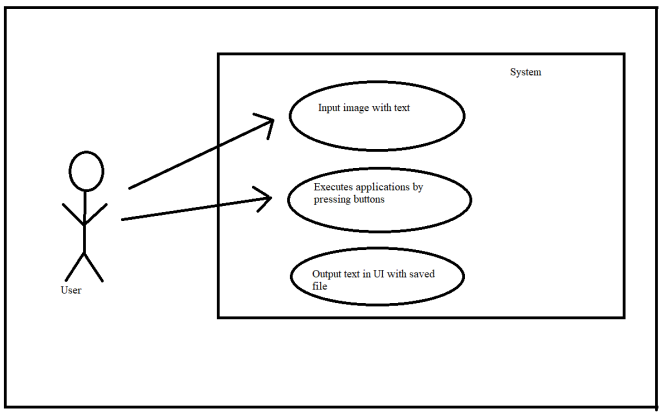


Fig -2: Use-case diagram

3.3 Sequence Diagram

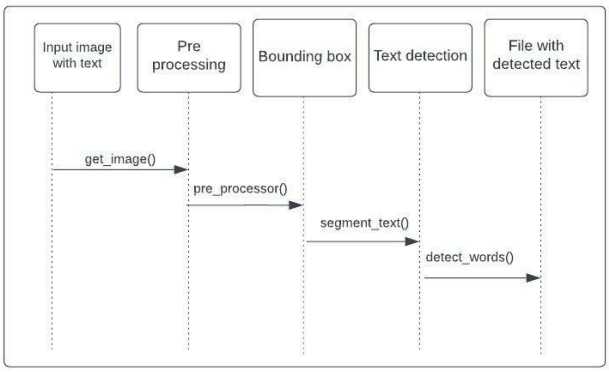


Fig -3: Sequence diagram

3.4 Component Diagram

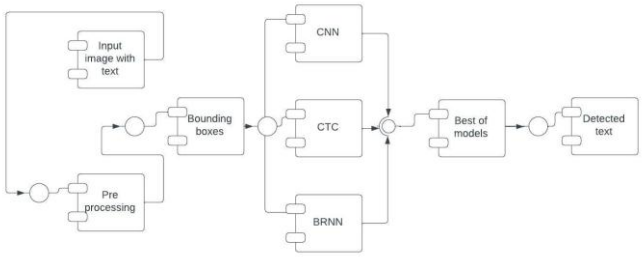


Fig -4: Component diagram

4. PROJECT IMPLEMENTATION

4.1 Overview of Project Modules

- Document Detection: This module helps to detect document present in the image. It further helps in cropping the image by removing the background so that the only the document is visible and further resizes the image so as to make it usable by the other modules.

- Text-area Detection: This module scans over the areas in the image and makes a rough estimate of text areas that might be present in the image. It further draws bounding boxes over the detected text areas in the image.

- Text Recognition: This module scans over the bounding boxes in the image and gives a rough estimate of recognized text that might be present in the bounding boxes. This is done with the help of different pre-trained models.

- Output: This module stores the text that has been recognized from the image. It further generates an output in the output window and makes a text file.

4.2 Algorithm for Hough Line Detection

The algorithm for detecting straight lines can be divided into the following steps:

- Edge detection, e.g. using the Canny edge detector.
- Mapping of edge points to the Hough space and storage in an accumulator.

- Interpretation of the accumulator to yield lines of infinite length. The interpretation is done by thresholding and possibly other constraints.

- Conversion of infinite lines to finite lines. The finite lines can then be superimposed back on the original image.

This in turn will help us in detecting a document present in the image.

This process is used to further crop and resize the image from removing the background in a way that just the text remains.

4.3 Algorithm for Text Recognition

The algorithm for text recognition through different models can be divided into the following common steps:

- Perform pre-processing on the image by removing noise and document background using the Hough line detector mentioned above.

- Detect text areas in the image and draw bounding boxes over the detected text areas by making use of packages in Python.

- Use feature extraction/labelling on existing datasets for un-supervised learning.

- Build different neural networks and train with the pre-processed data viz. CNN, BRNN, CTC for outputs with varied accuracies on test data.

- Record observations after using different models and images and choose the one with the best accuracy for text recognition.

This will help us in detecting and recognizing text from the given image document present in the image.

This output will then be shown on the UI for the user along with a text-file that will be generated.

5. IMAGES OF MODELS

5.1 CTC Model

Model: "handwriting_recognizer"

Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	[(None, 128, 32, 1)]	0	[]
Conv1 (Conv2D)	(None, 128, 32, 32)	320	['image[0][0]']
pool1 (MaxPooling2D)	(None, 64, 16, 32)	0	['Conv1[0][0]']
Conv2 (Conv2D)	(None, 64, 16, 64)	18496	['pool1[0][0]']
pool2 (MaxPooling2D)	(None, 32, 8, 64)	0	['Conv2[0][0]']
reshape (Reshape)	(None, 32, 512)	0	['pool2[0][0]']
dense1 (Dense)	(None, 32, 64)	32832	['reshape[0][0]']
dropout_1 (Dropout)	(None, 32, 64)	0	['dense1[0][0]']
bidirectional_2 (Bidirectional)	(None, 32, 256)	197632	['dropout_1[0][0]']
bidirectional_3 (Bidirectional)	(None, 32, 128)	164352	['bidirectional_2[0][0]']
label (InputLayer)	[(None, None)]	0	[]
dense2 (Dense)	(None, 32, 81)	10449	['bidirectional_3[0][0]']
ctc_loss (CTCLayer)	(None, 32, 81)	0	['label[0][0]', 'dense2[0][0]']

Total params: 424,081
Trainable params: 424,081
Non-trainable params: 0

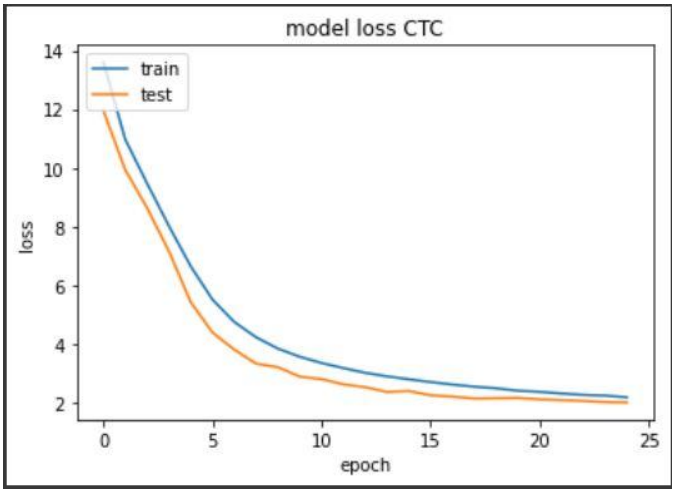
Fig -5: CTC model summary

5.2 BRNN Model

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 128, 1)]      0

 conv2d (Conv2D)             (None, 32, 128, 16)       160

 max_pooling2d (MaxPooling2D  (None, 16, 64, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 64, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 8, 32, 32)        0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 32, 64)         18496

 conv2d_3 (Conv2D)           (None, 8, 32, 64)         36928

 max_pooling2d_2 (MaxPooling  (None, 4, 32, 64)        0
 2D)

 conv2d_4 (Conv2D)           (None, 4, 32, 64)         36928

 batch_normalization (BatchN  (None, 4, 32, 64)        256
 ormalization)

 conv2d_5 (Conv2D)           (None, 4, 32, 64)         36928

 batch_normalization_1 (Batc  (None, 4, 32, 64)        256
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 2, 32, 64)        0
 2D)

 conv2d_6 (Conv2D)           (None, 1, 31, 64)         16448

 lambda (Lambda)             (None, 31, 64)            0

 bidirectional (Bidirectiona  (None, 31, 256)          198656
 l)

 bidirectional_1 (Bidirectio  (None, 31, 256)          395264
 nal)

 dense (Dense)               (None, 31, 63)            16191

=================================================================
Total params: 761,151
Trainable params: 760,895
Non-trainable params: 256
_________________________________________________________________
```

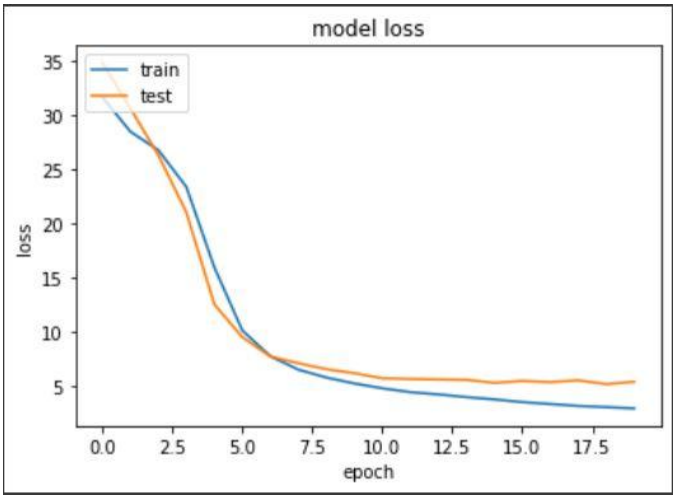
Fig -6: BRNN model summary

5.3 CTC Model Graph (Loss)



Graph -1: CTC model loss

5.3 BRNN Model Graph (Loss)



Graph -2: BRNN model loss

5.4 Sample word detection

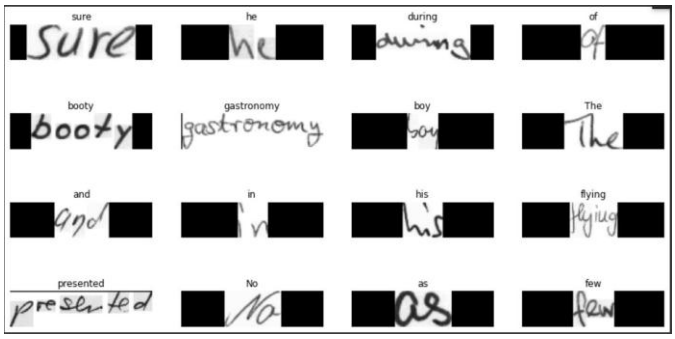


Fig -7: Sample word detection using pre-processed data and saved model files

6. IMAGES OF SOFTWARE

6.1 UI screen

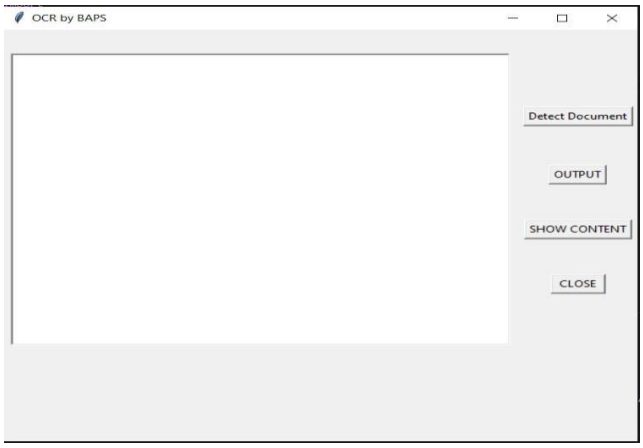


Fig -8: Main UI window

6.2 Input Image

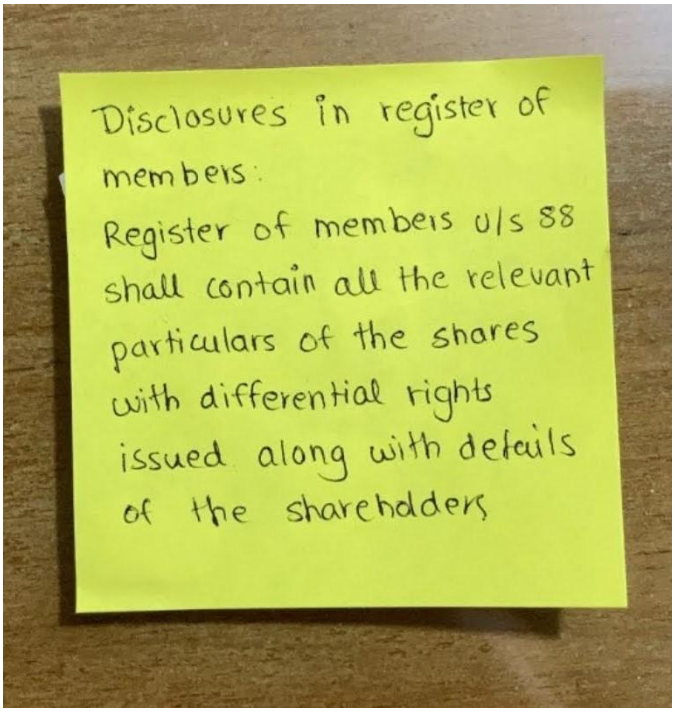


Fig -9: Input image (raw)

6.3 Input Image after Pre-processing

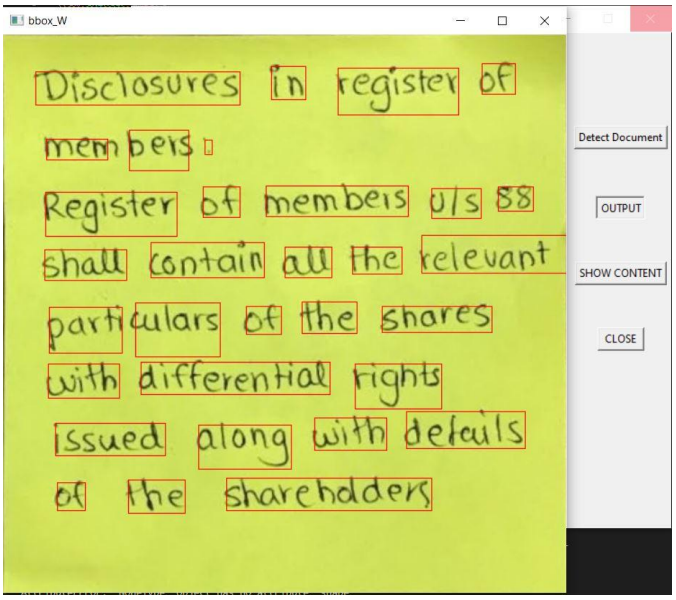


Fig -10: Input image after pre-processing

6.4 Final Output

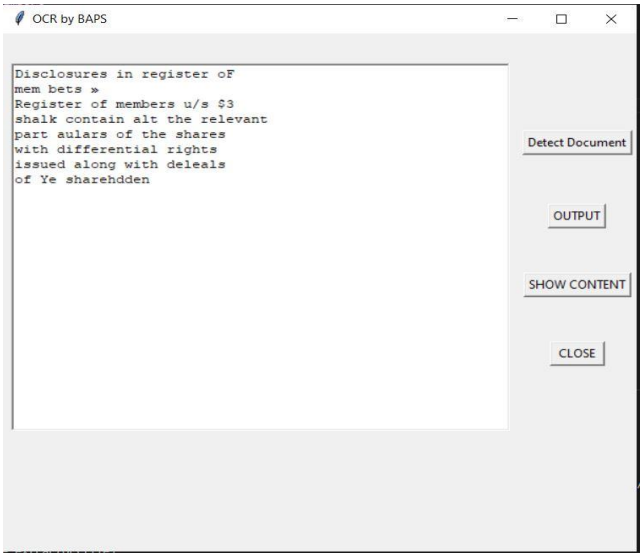


Fig -11: Final output image in UI

7. CONCLUSIONS

In recent years, great advances in deep learning and computer vision have allowed improvements on document and image processing and HTR. Processing of digital files is cheaper than processing traditional paper files. The aim of an OCR software is to convert handwritten text into machine readable formats and by making use of data pre-processing that involved cropping, resizing, normalization, thresholding of an image and then further splitting of datasets and applications of models viz CNN, CTC and BRNN, we're able to perform Optical Character Recognition of Text through the provided UI. The aforesaid models can be further optimized to improve the accuracy of the detected text.

REFERENCES

[1] Rosebrock, A. "Automatically OCR'ing Receipts and Scans," PyImageSearch, 2021, https://pyimagesearch.com/2021/10/27/automatically-ocring-receipts-and-scans/.

[2] H. Li, R. Yang and X. Chen, "License plate detection using convolutional neural network," 2017 3rd IEEE International Conference on Computer and Communications (ICCC), 2017, pp. 1736-1740, doi: 10.1109/CompComm.2017.8322837.

[3] Schuster, Mike & Paliwal, Kuldip. (1997). Bidirectional recurrent neural networks. Signal Processing, IEEE Transactions on. 45. 2673 - 2681. 10.1109/78.650093.

[4] O. Nina, Connectionist Temporal Classification for Offline Handwritten Text Recognition, BYU Conference Center, 2016.