Tutorial on OCR using pytesseract library with an example application

Trusha Talati
Department of Computer Engineering
Sardar Patel Institute of Technology
Mumbai, India
trusha.talati@spit.ac.in

Akshat Bhat
Department of Computer Engineering
Sardar Patel Institute of Technology
Mumbai, India
akshat.bhat@spit.ac.in

***Abstract*—The recognition of text from images is still an important aspect of computer vision applications such as image segmentation and object detection. Optical Character Recognition (OCR) has become a popular application of Computer Vision in recent years. OCR recognises text in a written or printed document and converts it to machine-readable code. Text from a scanned document, a photo of a document, a scene photo, or subtitle text placed on an image can all be recognised using OCR. In this paper, we aim to provide a step-by-step guide on how to apply OCR to images using pytesseract and OpenCV, along with an example application to further illustrate the application of OCR.**

***Index Terms*—Optical Character Recognition, Tesseract OCR, Image Processing, Text Extraction, Text Recognition, Computer Vision, Jaro-Winkler Similarity**

I. Introduction

The electronic conversion of typed, handwritten, or printed text images into machine-encoded text is known as optical character recognition (OCR). OCR allows a large number of paper-based documents in a variety of languages and formats to be digitised into machine-readable text, which not only simplifies storage but also makes previously inaccessible data available to anyone with a single click.

Various techniques which are used to carry out Optical character recognition (OCR):

1) **Preprocessing:** For successful recognition, preprocessing is performed on the given images to improve the chances. Various techniques include - De-skew involves aligning documents to make text either horizontal or vertical. Despeckling is removal of spots and smoothing edges. Images are also converted to black and white to make distinction between text and background. This technique is called binarization. Techniques like line removal, layout analysis, segmentation and normalization are also done as part of preprocessing.
2) **Text recognition:** This involves two important OCR algorithms - Matrix matching and Feature Extraction both of which rank candidate characters. Matrix matching involves pattern recognition by comparing image to stored glyph. This method is usually employed on typewritten texts with known fonts. Feature Extraction as name suggests extracts various features from glyphs such as lines, closed loops,line intersections, etc. This works for handwritten text recognition and hence modern OCR method. It uses KNN to choose nearest match when comparing image features to glyph features.

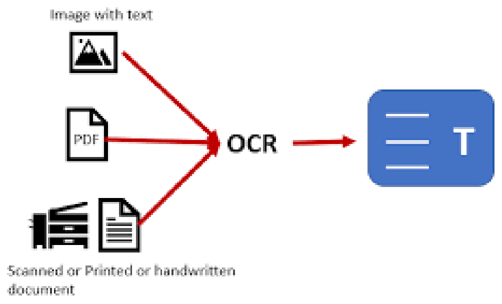


Fig. 1. Optical Character Recognition

3) **Post-processing:** OCR accuracy can be improved by constraining the output with a lexicon – a list of words that are permitted to appear in a document. This could be everything in the English language, or a more technical lexicon for a specific field. This method can be problematic if the document contains words that are not in the lexicon, such as proper nouns. Tesseract influences the character segmentation step with its dictionary to improve accuracy.
4) **Application-specific optimizations:** Application specific OCR is implemented for better performance.This includes taking into account the business rules, standard expressions or rich information contained in color images. This is applied to variety of applications like vehicle's license plates, bill invoices, identity cards, screenshots of handwritten works etc.

The most famous libraries for OCR are - Open Source Computer Vision which is an open source library for computer vision and image processing Tesseract which is an open-source, free software available under the Apache License. It is currently sponsored by Google and is compatible with several programming languages with the help of wrappers. Pytesseract is one such wrapper that is used for the Python programming language.

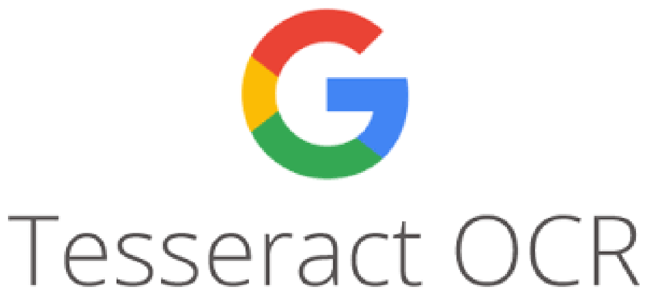


Fig. 2. Tesseract

Fig. 3. OpenCV

The following section is a description of all the different papers referred for the purpose of this research. The sections after that elaborate the proposed methodology along with experiments and results.

II. Literature Survey

The overview of the Tessaract OCR Engine was given in the paper [1] from Google Inc. explaining the engine's novelity in line finding, features/classification methods, and adaptive classifier.

Many research papers have tried to explain the various techniques for Optical Character Recognition using Tesseract or OpenCV. In their paper, "Optical Charater Recognition using Tesseract and Classification" [2] S. Dome and A. P. Sathe review papers whose main objective is to summarize the research that has been conducted on character recognition of handwritten documents and to provide research directions.The authors have analysed 176 handwritten OCR research articles which were published between the years 2000 to 2019. This review article serves the purpose of presenting the state of the art results and techniques on OCR and also provide research directions by highlighting research gaps.

In the paper "Handwritten And Printed Text Recognition Using Tesseract-OCR" [3] the author has created a simple text recognizing model using Pytesseract and OpenCV that can perform several functions such as detecting characters, detecting words,detecting just digits, converting handwritten text to computer-readable text, and detecting multiple language text. The features of this model are also described in this research paper.

In the paper "Optical Character Recognition using Tesseract Engine" [4] the authors discuss the Optical Character Recognition (OCR) technology, its design, and the experimental results of OCR conducted by Tesseract on medical data images.They also provide a comparison of this tool with other detection methods in order to improve detection accuracy. The developed models were able to detect extract text from images.However, the accuracy was found to be maximum for the tesseract engine.

The authors of the paper "Text Recognition And Detection From Images Using Pytesseract" [5] have reviewed and analysed different methods for text recognition from images. They have reviewed and analysed different methods to find text characters from scene images. They have reviewed the basic architecture of text recognition from images in which they discussed different techniques of image processing in a particular sequence for text recognition from scan images. Some applications of the text recognition system have also been discussed.

The paper "A Business Card Reader Application for iOS devices based on Tesseract" [6] aimed to design a business card reader application for iOS devices using Tesseract. A dataset of 55 digital business cards obtained from an online repository was used to test and evaluate the system's accuracy. In terms of text recognition and data detection, the system achieved an accuracy of up to 74%. A comparison test was performed against a commercial business card reader application, and our application performed admirably.

III. Methodology

Our methodology consists of three parts. In the first part, we perform basic operations using OpenCV for preprocessing a handwritten text image. In the second part, the tesseract system architecture is studied and pytesseract is used for text localization, detection and recognition from a hadnwritten text image. Lastly, we use pytesseract to recognize texts from randomly generated textual images containing a single word each.

A. Performing basic operations using OpenCV

OpenCV is a programming library geared mostly at real-time computer vision. It was created by Intel and then sponsored by Willow Garage and Itseez. OpenCV is an useful tool for image processing and computer vision. It is an open-source library which can be used for face detection, objection tracking, landmark detection etc. The following are the operations that we perform:

1) Reading and Displaying an image: To read an image cv2.imread() method is used. This method loads an image from the specified file. And to display an image plt.imshow() method is used. This method displays an image as an output. The image to be read is taken from the Handwritten Text Recognition Dataset. After displaying the image, basic operations are performed on it.

2) Grayscaling of Image: Grayscaling is the process of converting an image from other color spaces e.g. RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white. cv2.cvtColor(image,

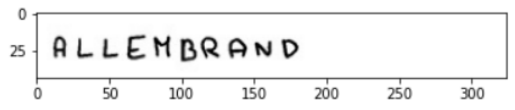


Fig. 4. Displaying the handwritten text image

cv2.COLOR_BGR2GRAY) method is used which takes the image file as the first argument and the second argument signals the conversion from BGR colour scale to gray scale.

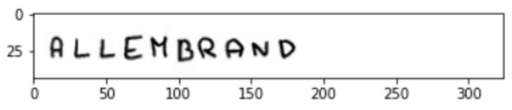


Fig. 5. Displaying the image in grayscale

3) Performing Median Blur: cv2.medianBlur() computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise. One interesting thing to note is that, in the Gaussian and box filters, the filtered value for the central element can be a value which may not exist in the original image. However this is not the case in median filtering, since the central element is always replaced by some pixel value in the image. This reduces the noise effectively. The kernel size must be a positive odd integer. We add a 50% noise to our original image and use a median filter.

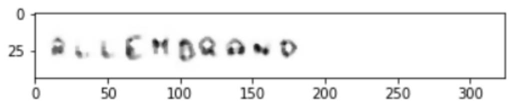


Fig. 6. Image after applying Median Blur

4) Performing Adaptive Thresholding: At a regional level, adaptive thresholding, also known as dynamic or local thresholding, defines the threshold level for deciding whether to convert an image to white or black. The area sampled and the technique of evaluation differ depending on the application. The method cv.adaptiveThreshold() takes three input parameters:

The adaptiveMethod decides how the threshold value is calculated.

cv.ADAPTIVE_THRESH_MEAN_C: The threshold value is the mean of the neighbourhood area minus the constant C.

cv.ADAPTIVE_THRESH_GAUSSIAN_C: The threshold value is a gaussian-weighted sum of the neighbourhood values minus the constant C. The blockSize determines the size of the neighbourhood area and C is a constant that is subtracted from the mean or weighted sum of the the neighbourhood pixels.

B. Basic usage of pytesseract for Text Localization, Detection and Recognition

Tesseract is an open-source, free software available under the Apache License. It is currently sponsored by Google and is compatible with several programming languages with the help of wrappers. Pytesseract is one such wrapper that is used for the Python programming language.

The method pytesseract.image_to_string is used to extract text from an image.

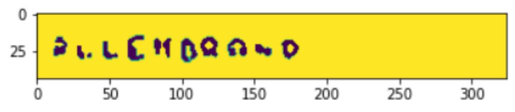
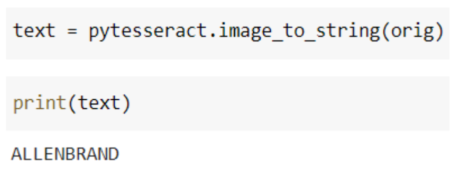


Fig. 7. Image after applying Adaptive Thresholding



```
text = pytesseract.image_to_string(orig)
```

```
print(text)
```

```
ALLENBRAND
```

Fig. 8. Output text after performing pytesseract OCR on image

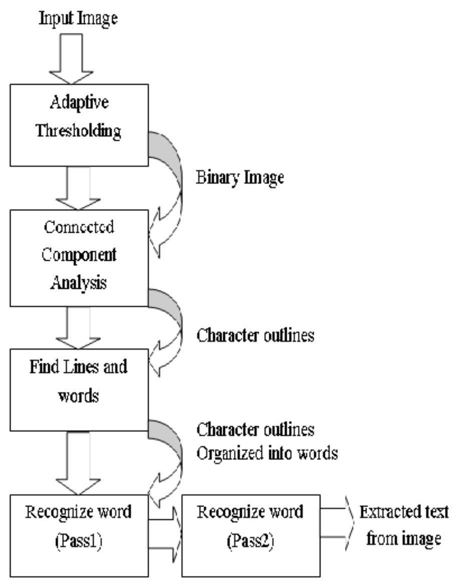


Fig. 9. Architecture of Tesseract OCR

As shown in the block diagram, Tesseract OCR works in a step-by-step manner [7]. Adaptive Thresholding is the first stage, which turns the image into binary images. Connected component analysis is the next stage, which is utilised to extract character outlines. Because it performs OCR on images with white text on a black background, this approach is quite handy. Tesseract was most likely the first to offer this type of processing. The outlines are then turned into Blobs. Text lines are created from blobs, and the lines and regions are examined for a fixed area or equivalent text size. Using definite and fuzzy spaces, the text is separated into words. The text recognition process is then launched as a two-pass procedure. During the first pass, each word in the text is attempted to be recognised.

As training data, each satisfactory word is delivered to an adaptive classifier. The adaptive classifier makes an attempt to detect text more accurately. Because the adaptive classifier learned something new after receiving some training data, the final phase is used to overcome numerous challenges and extract text from photos.

C. Recognising texts from Randomly Generated Textual Images using pytesseract

In order to demonstrate the working of pytesseract and how accurate are its OCR predictions, we create a dataset containing 1000 randomly generated images that have texts in different fonts and styles. The pytesseract package is used to interface with the Tesseract OCR engine. Firstly, we install two python packages - trdg and RandomWords. The trdg package stands for TextRecognitionDataGenerator which is a synthetic data generator for text recognition. The RandomWords package is a helpful package for generating random words in English. We create a RandomWords class object and generate 1000 random words by calling the random_words() method and providing a count argument with value 1000. GeneratorFromStrings() will generate 1000 images containing text in any random font and style (Fig. 10.). The size argument specifies the size in pixels for the images generated. A dataframe is created which will store the filenames of the generated images and their corresponding text labels. We then loop through the images and labels generated and save the images in a Google Drive folder, and also save the details in the dataframe. After

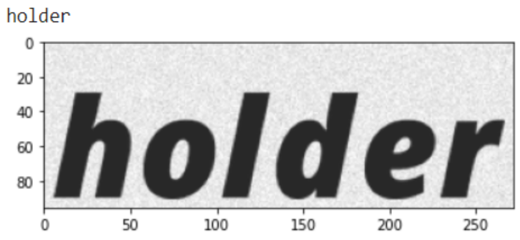
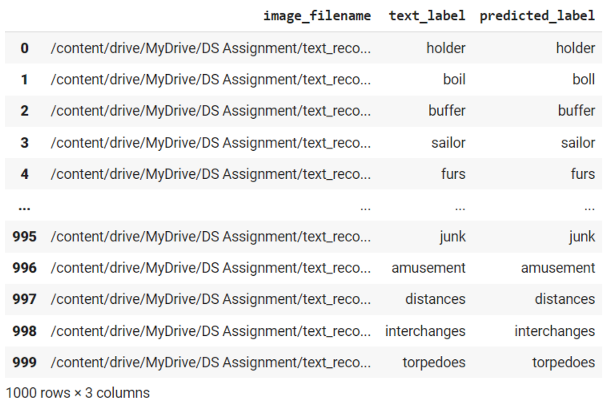


Fig. 10. Example image generated along with its label

successfully storing the image and corresponding text label details in the dataframe, we loop through these images and use the pytesseract.image_to_string() function to extract text from an image. The predicted labels will be stored in a list. The tqdm package acts as a progress indicator for the loop as there are many images to go through. We store the predicted labels as a different column in the dataframe. After that we evaluated the performance using different metrics which will be discussed in the following section.

IV. Results

If we were to simply check with maximum strictness whether the predicted label was the same as the actual label, we would be marking a near-miss (maybe because there's a mismatch in 1 character) as a failure. Upon evaluation, we found that only 87.5% of 1000 images were exactly captured. Since we do not want to mark a near-miss as a failure we

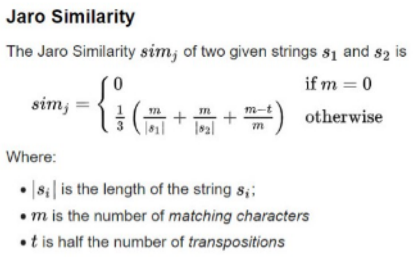


	image_filename	text_label	predicted_label
0	/content/drive/MyDrive/DS Assignment/text_reco...	holder	holder
1	/content/drive/MyDrive/DS Assignment/text_reco...	boil	boll
2	/content/drive/MyDrive/DS Assignment/text_reco...	buffer	buffer
3	/content/drive/MyDrive/DS Assignment/text_reco...	sailor	sailor
4	/content/drive/MyDrive/DS Assignment/text_reco...	furs	furs
...	...	...	...
995	/content/drive/MyDrive/DS Assignment/text_reco...	junk	junk
996	/content/drive/MyDrive/DS Assignment/text_reco...	amusement	amusement
997	/content/drive/MyDrive/DS Assignment/text_reco...	distances	distances
998	/content/drive/MyDrive/DS Assignment/text_reco...	interchanges	interchanges
999	/content/drive/MyDrive/DS Assignment/text_reco...	torpedoes	torpedoes

1000 rows × 3 columns

Fig. 11. Dataframe with actual and predicted labels

will be using the Jaro-Winkler algorithm to detect similarity between the captured text and the actual text. Using this we will get a quantitative understanding of how accurate and practical the tessearct OCR system is. Further information



Jaro Similarity

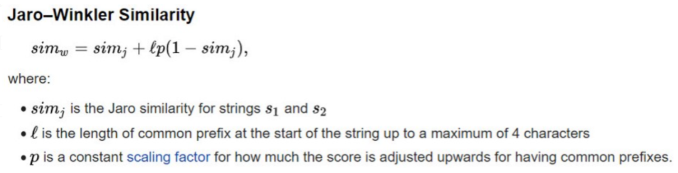
The Jaro Similarity sim_j of two given strings s_1 and s_2 is

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Where:

- $|s_i|$ is the length of the string s_i;
- m is the number of *matching characters*
- t is half the number of *transpositions*

Fig. 12. Jaro Similarity



Jaro–Winkler Similarity

$$sim_w = sim_j + \ell p(1 - sim_j),$$

where:

- sim_j is the Jaro similarity for strings s_1 and s_2
- ℓ is the length of common prefix at the start of the string up to a maximum of 4 characters
- p is a constant scaling factor for how much the score is adjusted upwards for having common prefixes.

Fig. 13. Jaro-Winkler Similarity

[8] on the Jaro Similarity algorithm and the Jaro-Winkler Similarity algorithm is given in Fig. 12. and Fig. 13. Once the similarity score is obtained, another column will be added to the dataframe (Fig. 15.) so that we can analyze the differences between the predicted labels and the actual labels via the similarity score. In order to get an overall picture of the similarity score, we calculate the mean similarity score which in our case comes to around 95.56%. The similarity scores are plotted as an histogram (Fig. 16.) to see the distribution of scores for different images and their corresponding labels. We can see that most scores are 100%. However, 25 scores are 0% (Fig. 17.). We analyzed some of the images for which 0% similarity scores were obtained. We can see that no labels were predicted for the last 3 images (Fig. 18.) whereas an incorrect label was predicted for the 1st image. We can infer

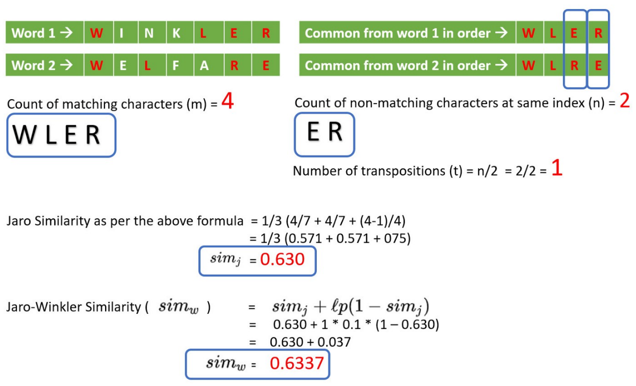
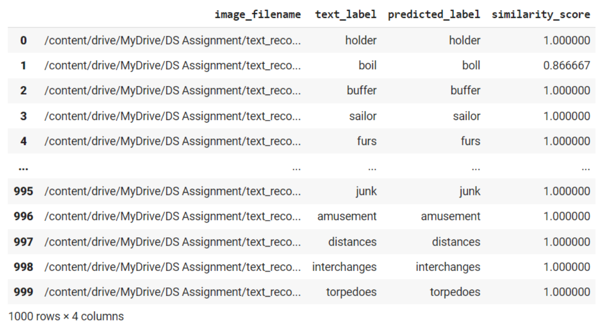

Word 1 → W I N K L E R
Word 2 → W E L F A R E
Common from word 1 in order → W L E R
Common from word 2 in order → W L R E
Count of matching characters (m) = 4
W L E R
Count of non-matching characters at same index (n) = 2
E R
Number of transpositions (t) = n/2 = 2/2 = 1
Jaro Similarity as per the above formula = 1/3 (4/7 + 4/7 + (4-1)/4)
= 1/3 (0.571 + 0.571 + 075)
sim_j = 0.630
Jaro-Winkler Similarity (sim_w) = $sim_j + \ell p(1 - sim_j)$
= 0.630 + 1 * 0.1 * (1 – 0.630)
= 0.630 + 0.037
sim_w = 0.6337

Fig. 14. Example on Jaro-Winkler Algorithm



	image_filename	text_label	predicted_label	similarity_score
0	/content/drive/MyDrive/DS Assignment/text_reco...	holder	holder	1.000000
1	/content/drive/MyDrive/DS Assignment/text_reco...	boil	boll	0.866667
2	/content/drive/MyDrive/DS Assignment/text_reco...	buffer	buffer	1.000000
3	/content/drive/MyDrive/DS Assignment/text_reco...	sailor	sailor	1.000000
4	/content/drive/MyDrive/DS Assignment/text_reco...	furs	furs	1.000000
...	...	...	...	...
995	/content/drive/MyDrive/DS Assignment/text_reco...	junk	junk	1.000000
996	/content/drive/MyDrive/DS Assignment/text_reco...	amusement	amusement	1.000000
997	/content/drive/MyDrive/DS Assignment/text_reco...	distances	distances	1.000000
998	/content/drive/MyDrive/DS Assignment/text_reco...	interchanges	interchanges	1.000000
999	/content/drive/MyDrive/DS Assignment/text_reco...	torpedoes	torpedoes	1.000000

1000 rows × 4 columns

Fig. 15. Dataframe with similarity scores

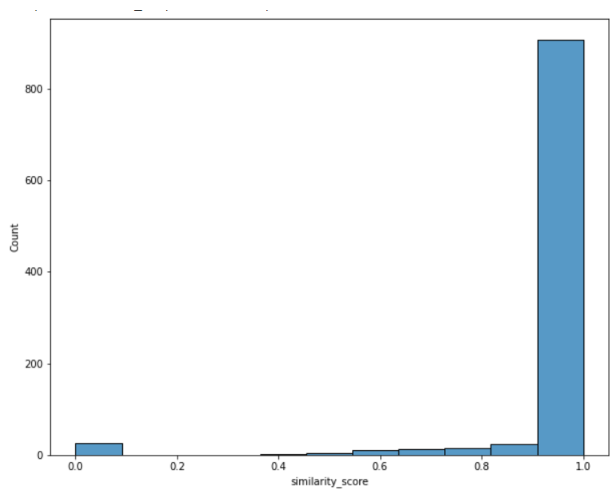
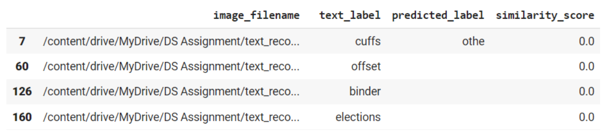
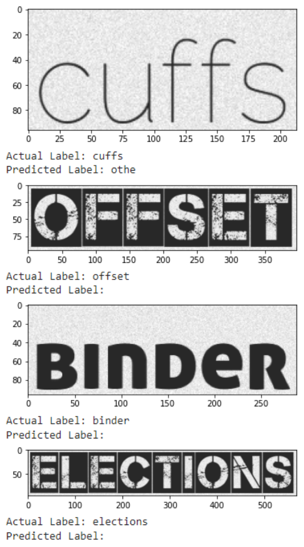


Fig. 16. Histogram of similarity scores



	image_filename	text_label	predicted_label	similarity_score
7	/content/drive/MyDrive/DS Assignment/text_reco...	cuffs	othe	0.0
60	/content/drive/MyDrive/DS Assignment/text_reco...	offset		0.0
126	/content/drive/MyDrive/DS Assignment/text_reco...	binder		0.0
160	/content/drive/MyDrive/DS Assignment/text_reco...	elections		0.0

Fig. 17. Dataframe of those images whose similarity scores were 0%



```
Actual Label: cuffs
Predicted Label: othe
Actual Label: offset
Predicted Label:
Actual Label: binder
Predicted Label:
Actual Label: elections
Predicted Label:
```

Fig. 18. Images with zero percent similarity score

that pytesseract OCR was not able detect text with a bold font style (as seen in the word 'binder') and when bold fonts are used with a darker background (as seen in the words 'elections' and 'offset) for our dataset.

V. Example application: Applying OCR to Business Card Images

To demonstrate real-life usage, we implemented our methodology on a dataset of Business cards. The dataset used is the Stanford Mobile Visual Search Data Set. This dataset includes query and reference images for: (1) CDs, (2) DVDs, (3) books, (4) video clips, (5) landmarks, (6) business cards, (7) text documents, (8) paintings. The format is exactly as described in 2011 ACM Multimedia Systems paper. We use the 6th category of business cards for our implementation in this case. The steps explained in methodology are applied to the dataset to retrive business information. Below is the result of each step performed on a sample business card -

1) **Step 1:** After loading the image it was cloned to extract high-resolution version. It is then resized to a width of 600px.
2) **Step 2:** Image is converted to grayscale.

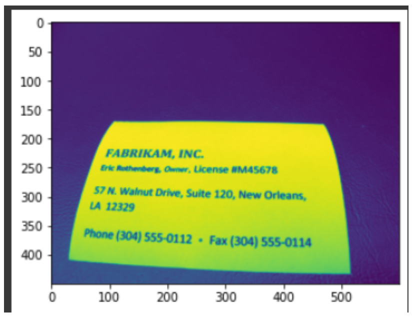

FABRIKAM, INC.
Eric Rothenberg, Owner, License #M45678
57 N. Walnut Drive, Suite 120, New Orleans, LA 12329
Phone (304) 555-0112 • Fax (304) 555-0114

Fig. 19. Step 2

3) **Step 3:** Median Blurring is applied to the image.

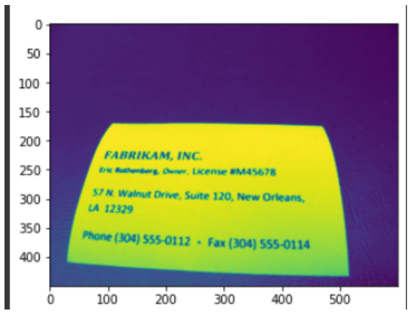

FABRIKAM, INC.
Eric Rothenberg, Owner, License #M45678
57 N. Walnut Drive, Suite 120, New Orleans, LA 12329
Phone (304) 555-0112 • Fax (304) 555-0114

Fig. 20. Step 3

4) **Step 4:** Adaptive Thresholding is applied on the image for edge detection in greyscaled images.

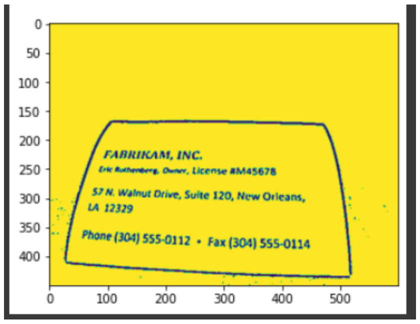
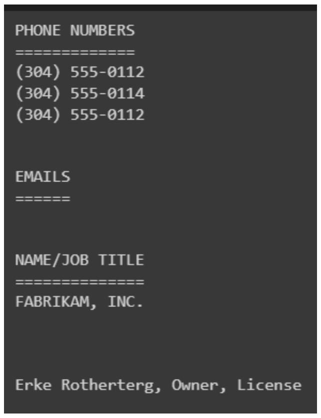

FABRIKAM, INC.
Eric Rothenberg, Owner, License #M45678
57 N. Walnut Drive, Suite 120, New Orleans, LA 12329
Phone (304) 555-0112 • Fax (304) 555-0114

Fig. 21. Step 4

5) **Step 5:** The image is processed using an alternative method which uses binary thresholding and inverse binary thresholding is applied and then pytesseract OCR is performed on the both types of images and a combined result is obtained to retain as much as information from the card as possible.
6) **Step 6:** Regex expressions are applied on the text obtained to find identities like phone numbers, emails and name/job title.



```
PHONE NUMBERS
=============
(304) 555-0112
(304) 555-0114
(304) 555-0112


EMAILS
======


NAME/JOB TITLE
==============
FABRIKAM, INC.



Erke Rotherterg, Owner, License
```

Fig. 22. Result

VI. Conclusion

In this tutorial paper, we went through various steps involoved in the process of extracting texts from images using pytesseract OCR. We learnt how to use OpenCV for image processing and how to apply different techniques such as grayscaling, median blurring, adaptive thresholding, binary thresholding etc. so that the process of using pytesseract to capture text becomes easier. Further, we saw how to measure the performance of an OCR system, by using the Jaro-Winkler Similarity Score method. A set of randomly generated images containing texts was obtained and after performing OCR on them using pytessearct, a mean similarity score of 95.56% was obtained. We also analyzed the cases for which OCR did not work efficiently. Finally, we saw an example application of OCR in which we applied OCR to business card images to extract valuable information such as names, job titles, emails and phone numbers.

References

[1] R. Smith, "An Overview of the Tesseract OCR Engine," Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), 2007, pp. 629-633, doi: 10.1109/ICDAR.2007.4376991.
[2] Jamshed Memon et al., Optical Character Recognition using Tesseract and Classification, IEEE Access, July 28, 2020.
[3] Patel, J. A. (n.d.). Handwritten And Printed Text Recognition Using Tesseract-OCR, IJCRT, September 9, 2021.
[4] Nikita Kotwal, Gauri Unnithan, Ashlesh Sheth, Nehal Kadaganchi, Optical Character Recognition using Tesseract Engine, IJCRT, September 9, 2021.
[5] Saurabh Uday Saoji et al., Text Recognition And Detection From Images Using Pytesseract, Journal of Interdisciplinary Cycle Research, August 2021.
[6] B. A. Dangiwa and S. S. Kumar, "A Business Card Reader Application for iOS devices based on Tesseract," 2018 International Conference on Signal Processing and Information Security (ICSPIS), 2018, pp. 1-4, doi: 10.1109/CSPIS.2018.8642727.
[7] Chirag Indravadanbhai Patel, Atul Patel, Dharmendra Patel, "Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study", International Journal of Computer Applications, DOI: 10.5120/8794-2784
[8] https://srinivas-kulkarni.medium.com/jaro-winkler-vs-levenshtein-distance-2eab21832fd6