Performance Comparison of Optical Character Recognition Approaches for the Hindi Language

Sushmitha Jogula
Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Hyderabad, Telangana, India.

Asfan Sajid
Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Hyderabad, Telangana, India.

Abstract – Optical Character Recognition (OCR) is the digital conversion of images of typed, handwritten, or printed text from a scanned document or a photo into machine-encoded text. OCR engines are typically designed and used to read typed (machine-printed) characters of popular languages such as English, which are generally used as a primary mode for communication. Although Hindi language is a preferred medium of communication in many parts of India, much research has not been carried out in the domain of Hindi character recognition. The objective of this paper is to investigate the principles of OCR and compare the efficiency of character recognition functionality and its subsequent electronic conversion to text for the Hindi language. This work involves using the MSER (Maximally Stable Extremal Regions) algorithm and various additional pre-processing techniques for enhancing the performance of OCR for Hindi in the MATLAB environment. The results of this work are subjected to comparison with an existing open-source OCR engine Tesseract

Index Terms – Optical Character Recognition (OCR), Maximally Stable Extremal Regions (MSER), Character Accuracy (CA), Character Error Rate (CER).

1. INTRODUCTION

1.1. Optical Character Recognition

Optical Character Recognition (OCR) is a comprehensive process of converting scanned or printed images containing textual instances or otherwise handwritten text into a readable and writable format, i.e., editable text to enable its further processing. This technology allows a machine to recognize the text automatically when provided with necessary input [1]. It can be referred to as an offline character recognition process in which the system scans and recognizes static images of the characters. It is a field of research in artificial intelligence, signal processing, pattern recognition, and machine vision [2].

OCR can be subdivided into Handwritten Character Recognition and Printed Character Recognition. Handwritten character recognition is more challenging to be implemented than Printed character recognition due to diverse human handwriting styles and customs. In Printed character recognition, the images to be processed contain standard fonts such as Times New Roman, Arial, Courier [2]. OCR is designed in a way to enable processing of images that contain primarily textual instances, with very little to non-existent non-text clutter. [3]

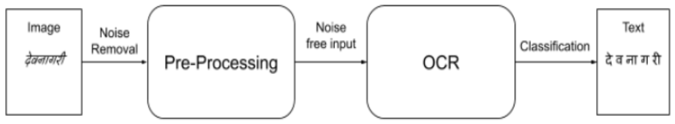


Figure 1 Block Diagram of OCR

1.2. Working of OCR

The three primary phases of the OCR process include:

1.2.1. Pre-processing

OCR software often “pre-processes” images to improve the chances of successful recognition. Few techniques include:

- De-skew: If the input image or document was not correctly aligned during scanning, it might need to be tilted a few degrees counter clockwise or clockwise to make lines of text perfectly horizontal or vertical.
- Binarization: Converts an image from color or grayscale to black-and-white image, also known as a “binary image” because there are two colors. This task is performed for separating the text from the background and is necessary as most recognition algorithms work only on binary images, and the effectiveness of this step impacts the quality of the character recognition stage significantly [4][5].
- Line and word detection: Establishes baseline for word and character shapes and separates words if necessary.

- Character isolation or Segmentation: For per-character OCR, multiple characters that are connected due to image artifacts must be separated, and single characters that are broken into multiple pieces must be connected.

1.2.2. Character Recognition

There are two ways in which character recognition is performed in OCR:

Matrix Matching involves comparing an isolated glyph in the input image to a stored glyph of similar font and the same scale on a pixel-by-pixel basis. This technique works best with printed text and does not work well when new fonts are introduced.

Feature Extraction breaks down glyphs into "features" like lines, closed loops, line direction, and line intersections and makes the recognition process computationally efficient [6]. Nearest neighbor classifiers such as the k-nearest neighbor algorithm are used to compare image features with stored glyph features and select the closest match.

1.2.3. Post-processing

OCR accuracy can be improved with the knowledge of the semantics of a language being scanned, and with the output being constrained by a lexicon – a list of words that are allowed to occur in a document; this can either be all the words in the language, or a more technical lexicon of a particular field. Near-neighbor analysis makes use of words that frequently co-occur to correct errors. The Levenshtein Distance algorithm can also be used to optimize OCR results further [7].

2. RELATED WORK

2.1. Existing System

Today, many types of OCR software are available in the market whose accuracy rate varies from 71% to 98%, but only a few of them are open source and free. Tesseract is one of the open-source and free OCR engines providing a high accuracy rate of up to 95% [8], but in the case of some complex images having multi-layered backgrounds or fancy text, Tesseract offers better accuracy in results if the pictures are in the grayscale mode instead of color. It is written using C and C++, so it is platform-independent. Initially developed for English language recognition, its later versions provide support for more than 100 languages out of the box, where each language comes with a trained language data file which must be kept in the Tesseract home folder. Working of Tesseract is similar to that of a scanner. It has a simple interface to take the input using basic commands. The basic Tesseract command takes two arguments: input image containing text and output text file with default .txt extension, respectively. First, an image with text is given as input to Tesseract, which is then processed by Tesseract, and the output file gets generated [8]. Tesseract OCR 3.01 is capable of detecting the Hindi language, but it still needs some enhancements to improve the performance. Hindi language recognition accuracy is quite low as the combinations of conjunct characters are not easily separable due to partial overlapping [9].

Some limitations of the Tesseract OCR Engine include:

- Tesseract was originally developed for the English language only. Any language that has different punctuation and numbers is going to be misinterpreted to an extent.
- Tesseract can handle only left-to-right languages.
- It is not able to recognize handwriting and is limited to about 64 fonts in total.
- It requires pre-processing to improve the OCR results, and the images need to be scaled appropriately, have as much image contrast as possible, and have text with horizontal alignment.

Recognition of text from an image is still a challenging task due to a wide range of text appearing in images. Text in images can have different font sizes, styles, distortions, and contrasts due to different lighting conditions [10]. As the accuracy of text regions of the image increases with efficient pre-processing methods and algorithms, the accuracy of OCR for that image also increases. This problem is even more severe for images containing non-English text (in this case, Hindi) because the conventional OCR engines are developed for recognizing English characters only and produce highly accurate results for OCR for English language only. For recognizing other language characters, the existing OCR engines must be trained accordingly with the respective language packages and datasets. Even after training the existing OCR engines, they fail to recognize Indic and Devanagari scripts such as Hindi, Sanskrit, Bangla to the utmost accuracy due to the complex nature of the language, lack of large unique word list, linguistic resources and reliable language models [11][12].

3. PROPOSED MODELLING

The proposed work uses different pre-processing techniques like Grayscaling, Thresholding and Binarization and character recognition algorithms like MSER (Maximally Stable Extremal Regions) for achieving the highest possible accuracy of OCR for Hindi language characters and text, when compared to existing OCR engines which do not make use of these improved pre-processing methods, and hence produce less accurate OCR results for Hindi language characters. MATLAB is used for implementation because of the available inbuilt OCR language packages and functions and predefined methods for grayscaling, binarizing and MSER detection. The results of the work are then compared with that of Tesseract. In the first step, an image containing Hindi text is given as input to both Tesseract and the proposed algorithm in MATLAB. In the second step, the image undergoes various enhanced pre-processing stages before it is given as input to OCR. In the third step, the results from both MATLAB and Tesseract are obtained and are compared with each other.

3.1. Methodology

3.1.1. System Design

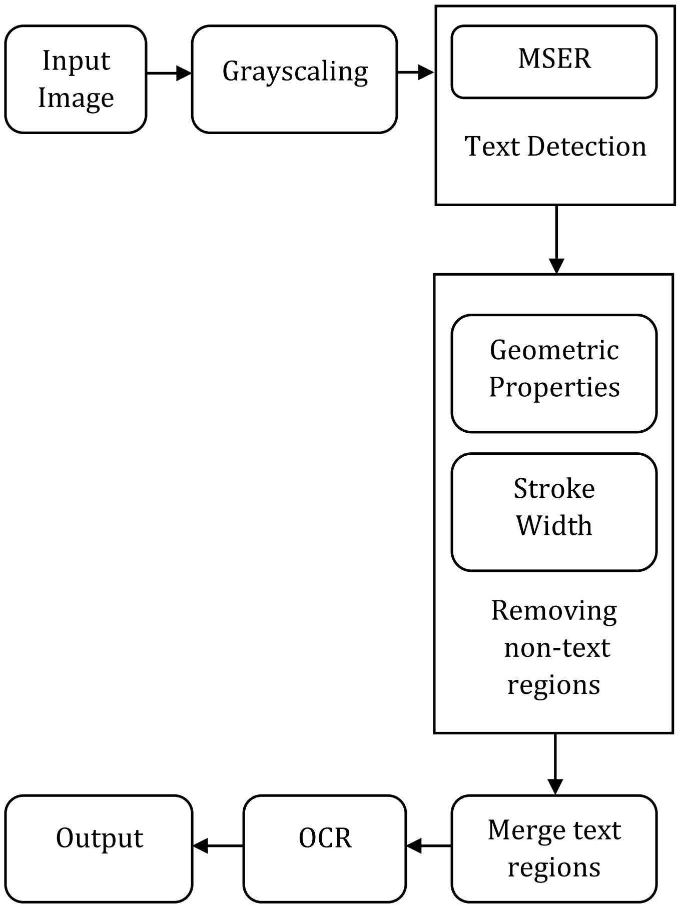


Figure 2 System Design for Proposed System

3.1.2 Algorithm Used: MSER

Maximally Stable Extremal Regions (MSER) is a feature detection algorithm. It extracts several co-variant regions, called MSERs, and enables identification of blobs from an image. An MSER is a stable connected component with uniform intensity levels under varying levels of threshold. ‘Extremal’ refers to the property that all pixels inside the MSER have either higher (bright extremal regions) or lower (dark extremal regions) intensity than all the pixels on its outer boundary [13][14]. This method is efficient for finding text regions because the consistent color and high contrast of text leads to stable intensity profiles [15].

In this operation, firstly, all pixels by sorted by gray value. Then, pixels are incrementally added to each connected component as the threshold is changed, and the area is monitored. Regions whose variation with respect to the threshold is minimal are defined as maximally stable [13].

- A simple luminance thresholding of the image is performed, and all the pixels below a threshold are made white, and the others black.
- Connected components (“Extremal Regions”) are extracted.
- A threshold for which an extremal region is “Maximally Stable,” i.e., the local minimum of the relative growth of its square is determined [14].
- Approximate a region with an ellipse and keep those regions descriptors as features.

3.2 Implementation

This work comprises of many phases [10] such as:

1. Input image selection

In the first step, the user is prompted to select an input image.

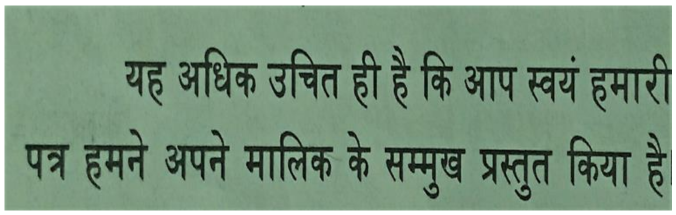

यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है।

Figure 3 Selected input image

2. Conversion of input RGB image to Grayscale image

The input image is then converted into a grayscale format and displayed to the user. All further operations are performed on this grayscale image.

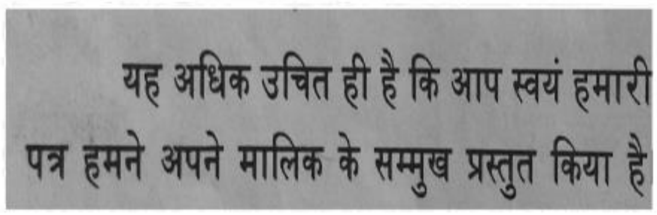

यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है

Figure 4 Grayscale format of the selected input image

3. Detecting text regions using MSER

MSER algorithm is used for detecting and segregating text regions from non-text regions of the image, which helps in identifying text in the image correctly, thereby increasing the accuracy of OCR [16].

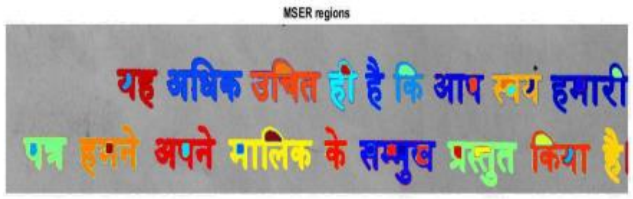

MSER regions
यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है।

Figure 5 Detected MSER Regions

4. Removing non-text regions from the image based on:

(i) Basic geometric properties

In the next step, the non-text regions are removed from the input image based on basic geometric properties. In addition to text regions, the MSER algorithm also detects many other stable regions in the image that are non-text. Several geometric properties can be used for discriminating between text and non-text regions such as Aspect Ratio, Eccentricity, Euler Number, Extent, Solidity [15].

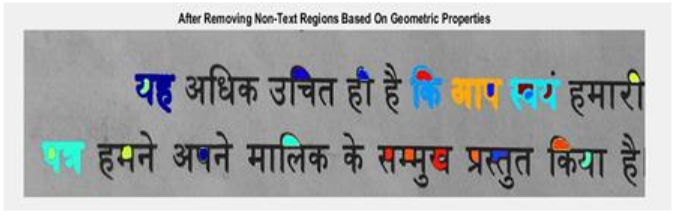
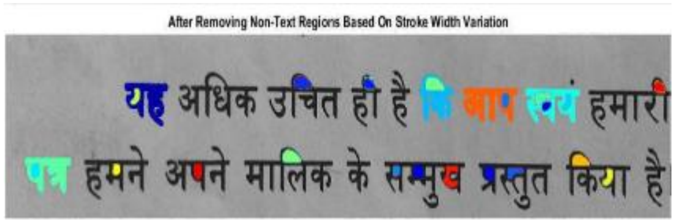

After Removing Non-Text Regions Based On Geometric Properties
यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है

Figure 6 Removing non-text regions based on Geometric Properties

(ii) Stroke Width variation

Stroke width, a common metric used to discriminate between text and non-text regions, can be defined as a measure of the width of the curves and lines that make up a character. Text regions tend to have less variation of stroke width than non-text regions [15].


After Removing Non-Text Regions Based On Stroke Width Variation
यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है

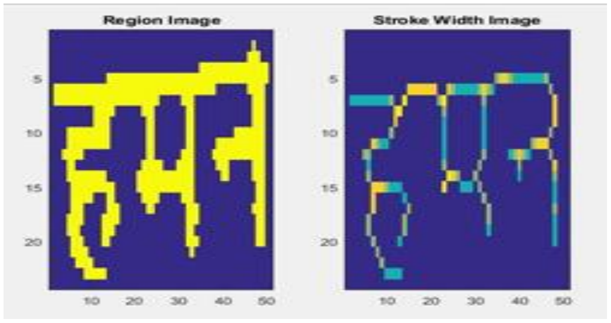


Figure 7 Removing non-text regions using Stroke width variation

5. Merge text regions for final detection region

At this stage, all the detection results are composed of individual text characters. Then, the individual text characters must be merged into words or text lines for recognizing actual words in the image, which has more meaningful information than unique characters. The approach for merging individual text regions into words or text lines is to determine the neighboring text regions and form a bounding box around them. Due to this, the bounding boxes of neighboring text regions overlap in such a way that the text regions that belong to the same word or text line form a chain of overlapping bounding boxes. Now, the overlapping bounding boxes are merged to form a single bounding box around individual words or text lines [15].

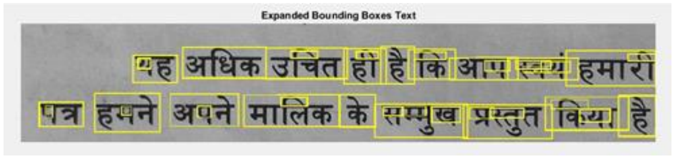

Expanded Bounding Boxes Text
यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है

Figure 8 Expanded Bounding Boxes text

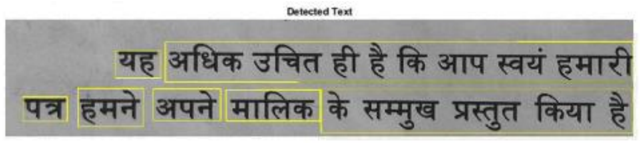
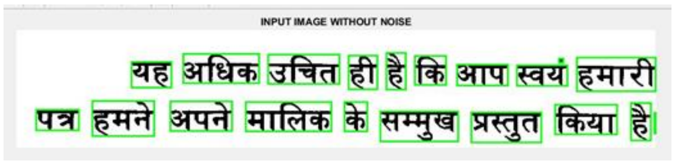

Detected Text
यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है

Figure 9 Final Detected text

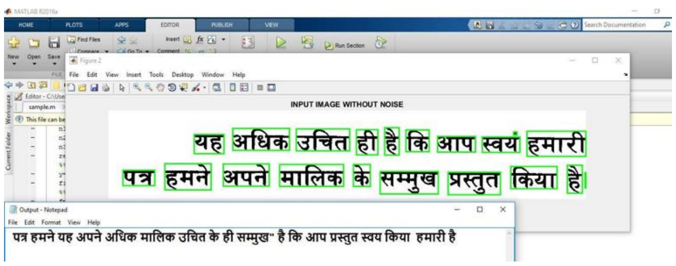
After all the steps have been executed, the resulting image is given as input to the OCR tool to obtain the output in the form of an editable text.



INPUT IMAGE WITHOUT NOISE

यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है।

Figure 10 Input image without noise



INPUT IMAGE WITHOUT NOISE

यह अधिक उचित ही है कि आप स्वयं हमारी
पत्र हमने अपने मालिक के सम्मुख प्रस्तुत किया है।

Output - Notepad

```
पत्र हमने यह अपने अधिक मालिक उचित के ही सम्मुख" है कि आप प्रस्तुत स्वय किया  हमारी है
```

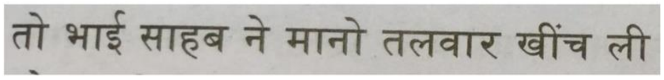
Figure 11 Final output in text file

4. RESULTS AND DISCUSSIONS

4.1. Comparative Analysis

To perform the Comparative Analysis of both the OCR tools, two performance measures are considered: Character Accuracy (CA) and Character Error Rate (CER) [17]. CA is used to evaluate whether all the characters are identified accurately or not. CER helps in determining the percentage of characters that are not appropriately recognized. Here, two types of input images are considered: 1-line text image and handwritten text image.

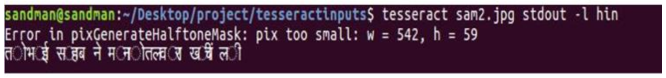
Input image-I:



तो भाई साहब ने मानो तलवार खींच ली

Figure 12 Input Image- I

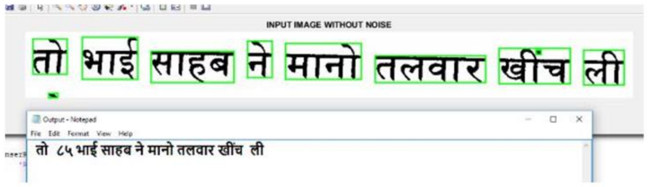
OCR Results in Tesseract:



```
sandman@sandman:~/Desktop/project/tesseractinputs$ tesseract sam2.jpg stdout -l hin
Error in pixGenerateHalftoneMask: pix too small: w = 542, h = 59
त◌ोभ◌ई स◌हब ने म◌न◌ोतल्व◌र ख◌चीं ल◌ी
```

Figure 13 OCR result in Tesseract for Input Image- I

OCR Results in MATLAB:



INPUT IMAGE WITHOUT NOISE

तो भाई साहब ने मानो तलवार खींच ली

Output - Notepad

```
तो  ८५ भाई साहब ने मानो तलवार खींच  ली
```

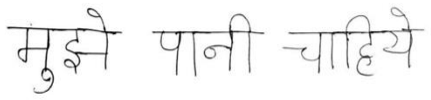
Figure 14 OCR result in MATLAB for Input Image- I

5.1.1. Comparative Analysis for Input Image-I:

S.No.	Original Text	MATLAB Output	Tesseract Output
1.	तो	तो	त◌ो
2.	भाई	भाई	भ◌ई
3.	साहब	साहब	स◌हब
4.	ने	ने	ने
5.	मानो	मानो	म◌न◌
6.	तलवार	तलवार	◌ितल्व◌र
7.	खींच	खींच	ख◌चीं
8.	ली	ली	ल◌ी

Table 1 Comparative Analysis-I

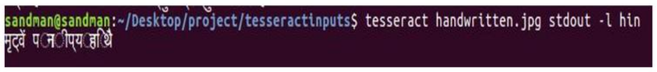
Input image-2:



मुझे पानी चाहिये

Figure 15 Input Image- II

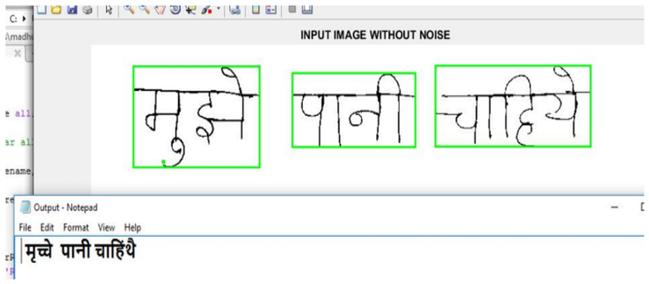
OCR Results in Tesseract:



```
sandman@sandman:~/Desktop/project/tesseractinputs$ tesseract handwritten.jpg stdout -l hin
मृर्वें प◌न◌ीपय◌ह◌थै
```

Fig 16: OCR result in Tesseract for Input Image- II

OCR Results in MATLAB:



INPUT IMAGE WITHOUT NOISE

Output - Notepad

```
मृच्चे  पानी चाहिंथै
```

Figure 17 OCR result in MATLAB for Input Image- II

5.1.2. Comparative Analysis for Input Image-II:

S.No.	Original Text	MATLAB Output	Tesseract Output
1.	मुझे	मृच्चे	मृट्वें
2.	पानी	पानी	प◌म◌
3.	चाहिये	चाहिंथै	िप्य◌ह◌थै

Table 2 Comparative Analysis-II

5.2. Performance Measurement

To find the Character Accuracy (CA) and Character Error Rate (CER) from the resultant text documents, i.e., to verify whether an OCR tool has converted all the characters available in the input image correctly or not, the following formula is used [17]:

Character Accuracy (CA) = (a/n) *100

Character Error Rate (CER) = 100-CA

Where a=Total number of characters identified correctly in the resultant text document

n=Total number of characters in the input image

S.No.	Input Image	CA% for Tesseract	CA% for MATLAB
1	I	56.25	100
2	II	0	57.14

Table 3 Comparison between OCR Results of Tesseract and MATLAB

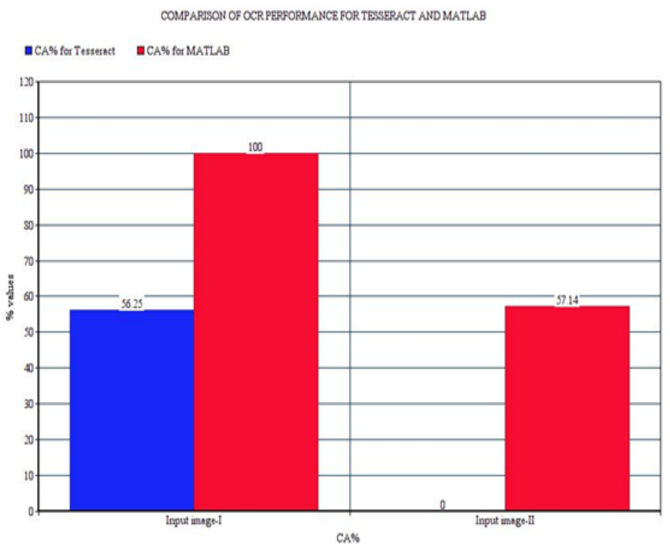


Figure 18 Graph showing comparison of CA% results

5. CONCLUSION

This work focuses on evaluating the efficiency and juxtaposing the accuracy of the two described approaches of OCR for the Hindi language. It analyzes the fundamentals of OCR and the techniques applied to enhance its performance for the Hindi language, which facilitates the users to have a precise and readily accessible tool for recognizing and understanding the language quickly and effectively. In this work, two different methodologies, an open-source tool Tesseract and the proposed system using the MSER algorithm executed in MATLAB, have been implemented, and the outcomes have been assessed using performance measures. In the future, the implemented MSER algorithm can be extended further for recognition of other Indic scripts.

REFERENCES

[1] Chirag Patel, Atul Patel, Dharmendra Patel. (2012). Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study. International Journal of Computer Applications (0975 – 8887) Volume 55– No.10.
[2] Mahesh Jangid. (2011). Devanagari Isolated Character Recognition by using Statistical features. International Journal on Computer Science and Engineering (IJCSE), Vol. 3 No. 6.
[3] Ravina Mithe, Supriya Indalkar, Nilam Divekar. (2013). Optical Character Recognition. International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-2, Issue-1.
[4] Mehmet Sezgin, Bülent Sankur. (2004). Survey over image thresholding techniques and quantitative performance evaluation. Journal of Electronic Imaging 13(1), 146–165.
[5] Oivind Due Trier, Anil K. Jain. (1995). Goal-Directed Evaluation of Binarization Methods. IEEE Transactions on Pattern Analysis and Machine Intelligence. 17 (12): 1191–1201.
[6] What's OCR? (2019, September 22). Retrieved from http://www.dataid.com/aboutocr.htm
[7] Chris Woodford. (2018, December 11). Optical Character Recognition. Retrieved from http://www.explainthatstuff.com/how-ocr-works.html
[8] Sahil Badla. (May 2014). Improving the Efficiency of Tesseract OCR Engine**.** SJSU ScholarWorks.
[9] Nitin Mishra, C. Patvardhan, C. Vasantha Lakshmi, Sarika Singh. (February 2012). Shirorekha Chopping Integrated Tesseract OCR Engine for Enhanced Hindi Language Recognition. International Journal of Computer Applications (0975 – 8887) Volume 39– No.6.
[10] Geethanjali Adlinge, Shashikala Kashid, Tejasvini Shinde, Virendrakumar Dhotre**.** (May 2016). Text Recognition in Images using MSER Approach**.** International Research Journal of Engineering and Technology (IRJET) Volume: 03 Issue: 05.
[11] B.Indira, Muhammad Shuaib Qureshi, Mahaboob Sharief Shaik, Rashad Mahmood Saqib, MV Ramana Murthy. (December 2012). Devanagari Character Recognition: A Short Review. International Journal of Computer Applications (0975–8887) Volume 59– No.6.
[12] Naveen T. S. (December 2014). Word Recognition in Hindi Scripts**.** Center for Visual Information Technology, International Institute of Information Technology Hyderabad, India.
[13] T. Kranthi, Jagadish Gurrala, G. Santhoshi. (2017). An Improved scheme of Optical Character Recognition Algorithm, International Journal of Innovations in Engineering and Technology (IJIET), Volume 9 Issue 1.
[14] Maximally Stable Extremal Region (MSER) Detectors. (2019, September 25). Retrieved from http://www.micc.unifi.it/delbimbo/wp-content/uploads/2010/05/A23_image_features_v_mser.pdf

[15] Automatically Detect and Recognize Text in Natural Images. (2019, September 25). Retrieved from https://au.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html
[16] Chaitanya R. Kulkarni, Ashwini B. Barbadekar. (2017). A Real Time Text Detection & Recognition System to Assist Visually Impaired, International Journal of Science and Research (IJSR), Volume 6, Issue 7.
[17] Dr. S.Vijayarani and Ms. A.Sakila. (July 2015). Performance Comparison of OCR Tools. International Journal of UbiComp (IJU), Vol.6, No.3.
[18] Archana A. Shinde, D.G.Chougule, (2012). Text Pre-processing and Text Segmentation for OCR. International Journal of Computer Science Engineering and Technology, Issue 1, pp 810-812.

Authors

Sushmitha Jogula
Department of Computer Science and Engineering,
Chaitanya Bharathi Institute of Technology,
Hyderabad,
Telangana,
India.

Asfan Sajid
Department of Computer Science and Engineering,
Chaitanya Bharathi Institute of Technology,
Hyderabad,
Telangana,
India.