Language Identification through Optical Character Recognition (OCR)

Brenda Tam
Toronto Metropolitan University
Toronto, Ontario, Canada
b1tam@torontomu.ca

Gurveen Kaur
Toronto Metropolitan University
Toronto, Ontario, Canada
gurveen.kaur@torontomu.ca

Manav Patel
Toronto Metropolitan University
Toronto, Ontario, Canada
manav1.patel@torontomu.ca

***Abstract*—This Optical Character Recognition (OCR) is the process of converting images to their text counterparts. It is a mature technology, but still faces some challenges. This paper will outline the basics of OCR, covering a naive implementation, presenting its limits, and exploring more advanced algorithms. Different applications of this include document processing, autonomous vehicles, augmented reality, etc. The MLT2019 [1] dataset is being used for this project, which consists of 10,000 images. The task of extracting text from an image is completed using OCR. This paper presents the task at hand, the dataset used, and the overall results and conclusions made from this project.**

Keywords—OCR; Tesseract; EAST; CRAFT;

Source Code:

https://drive.google.com/file/d/15SOO6bT3N815yivHb7Ipmqkngtto_yOa/view?usp=sharing

I. Introduction

Being able to read text from an image is important in various applications such as for an autonomous vehicle to be able to read road signs, or helping visually impaired individuals. In this project, the challenge of identifying and detecting text from an image is faced. The focus was not only on using Optical Character Recognition (OCR) [3] for text recognition but also on detecting the Latin script. Throughout this project, the historical evolution of OCR was further explored in-depth. We then looked into a few advanced text detection algorithms such as CRAFT and EAST. CRAFT stands for Character Region Awareness for Text Detection, and is an algorithm designed to recognize text regions within images, and focuses on multi-oriented and curved text. EAST stands for Efficient and Accurate Scene Text Detector. This is used for scene text detection and focuses on identifying text regions in natural scenes. Here, our main objective was to examine and analyze these algorithms and see how they can be used to solve the problem at hand. We initially started by using Tesseract, surrounded by the PyTesseract library, which provides a reliable and accurate platform for text recognition over the dataset. The low-level idea of image processing in this situation is to implement noise reduction techniques, contrast, sharpening and overall enhancement of the quality of the image so that it is more clear. The main parts of this OCR system are preprocessing, feature extraction, text recognition and post-processing [3]. Preprocessing is where the potential outcomes are regulated. Images are processed using processing techniques to make them clearer for feature extraction. In feature extraction, key patterns are identified in images from original features, which allows the image data to be displayed. Techniques for doing this include Edge Detection, Corner Detection, Texture Analysis, and capturing of histograms. In text recognition, we compare the extracted features to the original features and different patterns are identified for different texts. We use techniques such as Template Matching, Character Segmentation and Bounding Box Detection for recognizing the text present in the image. In post-processing, the text is finally extracted from the image and saved to a text file for accuracy evaluation. The final dataset we used was the MLT2019 dataset, which consisted of 10,000 images of one-word images.. In short, using Optical Character recognition, we were able to recognize Latin text present in images.

The first OCR machine dates back to the 1950s, made by David Shepard and Harvey Cook Jr. called GISMO [8]. Throughout the 1950s, these machines continued to be updated. In the 1960s, a group of MIT students attempted to improve the capabilities of these machines but were not able to due to the workstation's computational power [8]. At the same time, another technique was developed by another group of researchers called Hough's Transform and this technique allowed these machines to capture shapes along with the letters of Latin text [8]. Two new technologies were also introduced around this time: ICR AND MICR, which

eventually played an important role in transforming the OCR technology [8]. Throughout the 1980s and the 1990s, OCR technology started to advance even more as more improved algorithms were appearing with improved capabilities and results. Commercial OCR software also started to be introduced around this time, as these software and their results had become more reliable over the years [8]. In 2005, the Tesseract OCR technology was resurrected under Google, which eventually incorporated many advancements in this field [8]. In the present day, deep learning topics and ideas such as Convolutional Neural Networks (CNNs) have been introduced and incorporated into the OCR technology [8]. This combination introduced several significant changes to OCR including the ability to interpret handwritten text.

II. Method, Algorithms, and Models

The code implemented is a very basic approach to OCR using pyTesseract, a Python wrapper for the Tesseract OCR engine developed by Google, and openCV for computer vision functions. Our implementation does not use the EAST or CRAFT, though these algorithms will be briefly discussed later.

At the preprocessing step, basic computer vision techniques are applied to increase the accuracy of the output. Conversion to grayscale is a preliminary step that is required to perform many of the other image enhancement procedures that openCV offers. Various denoising methods such as one of OpenCV's built-in denoisers as well as the addition of a Gaussian blur to clean up the background. Thresholding is applied to binarize the image for a stark contrast between backgrounds and text. The thresholding used is openCV's adaptive or local thresholding, this was chosen to mitigate unwanted binarization effects that can arise when globally thresholding an unevenly lit image for example. Deskewing is an important step which was attempted, however it produced varying results and was thus left out. For similar reasons, morphological processes, such as erosion and dilation, are not included.

The final order of our preprocessing step is OpenCV's built-in denoising function, conversion to grayscale, Gaussian blur, and then local thresholding. This process was developed through trial and error. This processed image is passed onto the Tesseract OCR engine.

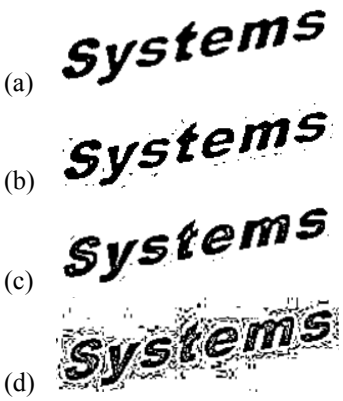

Fig. 1. (a) the output image after all preprocessing steps are performed. (b) preprocessing excluding OpenCV's built-in denoiser, fastNlMeansDenoisingColored(). (c) preprocessing excluding Gaussian blur. (d) preprocessing excluding both (b) and (c).

The Tesseract engine uses a line-finding and word-finding algorithm. First finding the text regions within an image and then identifying the text lines as seen in [2, Fig. 1]. It then determines whether a word is present by analyzing these text lines for fixed spacing before moving on to word recognition. This involves chopping low-confidence polygons in an attempt to create recognizable characters. If this process fails to yield results, the polygons are passed to an *associator* which will run through possible segmentation combinations to maximize the chances of correct character classification [2].

In post-processing, the extracted text is written to a separate text file so that its accuracy can be evaluated via the Levenshtein distance on a character level. We followed the same equation as given in [4, eq. (1) and eq. (2)]:

$$C_{error} = C_{ins} + C_{del} + C_{sub} \quad (1)$$

$$C_{correct} = C_{aln} - C_{error} \quad (2)$$

(1) is the basic concept of the Levenshtein distance, which is the number of edits needed to alter an input word into the target word [4]. Edits are categorized as insertions, deletions, and substitutions; C_{ins}, C_{del}, C_{sub} respectively. (2) gives the number of correct characters when compared to the target string.

Taking (1) and (2), we used them to determine the accuracy of our OCR implementation. Essentially, given the Levenshtein distance between two strings, the accuracy is determined by dividing the number of correct characters by max(length of target string, length of detected string):

$$Accuracy\% = C_{correct} / max(len(target), len(detected))$$

This is to take into account that the OCR engine may produce false positives when recognizing characters.

Efficient Accurate Scene Text Detector (EAST) is another text detection technique. We will talk about how this algorithm is implemented even though it isn't used in our experiments. The algorithm, as its name implies, is highly reliable, operates in real-time, and is both accurate and efficient.

Since the architecture of EAST is a Fully Convolutional Network(FCN), image pixels are converted to pixel classes using a convolutional neural network. The height and width of intermediate feature maps are transformed back to those of the input image using a fully convolutional network following the pixel classification process. Hence, the classification results for the input pixel at the same spatial position are stored in the channel dimension at any output pixel [7]. The network captures and processes information at multiple scales which helps it to handle text of different sizes.

The text recognition method can generate word or text-line predictions directly by eliminating the slow and unnecessary intermediate processes by using the FCN model. The eliminated intermediate steps include candidate proposal, text-region formation and word partition. The image is sent into the FCN during the preprocessing step, where labels are created for the pictures. Many channels of pixel-level text, score maps, and geometry make up the labels that are produced[5].

One of the anticipated channels is the score map, which is used to show the degree of confidence in the expected geometric shape at a given position by representing the likelihood of text existence in a specific area of the input image. From each pixel's point of view, the rest of the channels depict shapes that surround the word. A quadrangle (QUAD) or a rotated box (RBOX) can be used for this geometry map. In the study paper, the RBOX generation method is demonstrated. For each pixel with a positive score, the distances to the text box's four boundaries are computed and entered into the four channels of the RBOX ground truth. The distances between each pixel and the four text box boundaries are computed and recorded in the QUAD geometry map for datasets containing text regions marked in the QUAD style. A rotated rectangle covering the region with the least area is created similarly[5].

For each geometry, different loss functions are designed using the formula given in [5, eq 4]

$$L = L_s + \lambda_g L_g \quad (3)$$

L_s , L_s and λ_g indicate the losses for the score map, the geometry and the relative importance of the two losses respectively.

Ultimately, the geometries undergo post-processing processes that solely involve Thresholding and Non-Maximum Suppression (NMS). Because the algorithm's reliability depends on these two steps, they are extremely important to the algorithm. Only score maps that are above a predetermined score map are regarded as legitimate text regions when thresholding is performed to the score map channel. This is done to exclude the algorithm's less reliable predictions and concentrate on more accurate text region predictions. Following that filtering, NMS is used to exclude data that is duplicated or overlaps with other text detections.

Experiments on conventional benchmarks show that the suggested approach delivers greatly improved performance while running much faster than previous methods, both qualitatively and quantitatively[5].

Another text detection technique is CRAFT. CRAFT stands for Character Region Awareness for Text Detection. This is designed with a convolutional neural network that produces a character region score and an affinity score [6]. *"The region score is used to localize individual characters in the image, and the affinity score is used to group each character into a single instance"* [6].

For each training image, a ground truth label is generated for the affinity score and the region score with the characters bounded by a box [6]. The region box is the probability that the pixel given is in the middle of that character and the affinity score is the probability of there being space in the middle of the characters [6]. We use a Gaussian heatmap to find the probability of the character center [6]. To generate this ground truth, we use the following three steps: [6] *"1) prepare a 2-dimensional isotropic Gaussian map; 2) compute perspective transform between the Gaussian map region and each character box; 3) warp Gaussian map to the box area"* [6].

This method combines using real and synthetic images. It involves generating a character box for each word. For images where it's not clear enough to generate a character box, a model is used to predict the character region using the affinity and region scores. Then a confidence map is created over each character box to see if the characters and words have been accurately identified [6].

The following equation is used to calculate the confidence value [6, eq 1]

$$s_{conf}(w) = l(w) - min(l(w), |l(w) - lc(w)|)/l(w) \quad (3)$$

$l(w)$ the word length of the sample *w*, and lc(w) is the corresponding length of characters lc(w).

Thresholding, linkage refinement and bounding box refinement are all part of the CRAFT post-processing step. Like the EAST algorithm, CRAFT preserves the text region with a confidence score above the threshold while eliminating poor datasets. The affinity score is used to capture the association between adjacent characters. Subsequently, that score is employed to connect discrete character regions with entire text sections. This process of connecting the character regions is called linkage refinement. It also improves the overall quality of the discovered text regions. The process of creating bounding boxes is completed in three steps, which is summed up as follows. *"First, the binary map M covering the image is initialized with 0. M(p) is set to 1 if Sr(p) > τr or Sa(p) > τa, where τr is the region threshold and τa is the affinity threshold. Second, Connected Component Labeling (CCL) on M is performed. Lastly, QuadBox is obtained by finding a rotated rectangle with the minimum area enclosing the connected components corresponding to each of the labels."*[6]. This summarizes the post-processing step of the CRAFT algorithm. CRAFT has the advantage of not requiring any additional post-processing processes, like non-maximum suppression.

III. Experiments

As was previously outlined, text recognition has been accomplished using the Tesseract OCR engine. The MLT2019 dataset [1] used contains 23000 images of single words or characters. The first 10087 images from the dataset were processed. Of these, 5154 are images where the text is in Latin script, we will only be considering these images. However, only 1502 images were able to be identified as containing a word, ~29% of the Latin script set. The number of case-sensitive and insensitive characters in the images is listed

in Table 1, along with the corresponding number of characters that the engine was able to identify.

The overall average accuracy increases a small percentage when the case of the text is ignored, 0.009854575 for total images processed and 0.0338152328 for total images where words were detected. Given this small increase, it means that our implementation does not have a large issue with determining the letter case and that this does not impact the accuracy greatly.

Table 1. Overall Average Accuracy

	Total Processed	Total Detected
Case Sensitive	0.1674331012	0.5745340902
Ignore Case	0.1772876762	0.6083493230

As Fig. 2 displays, the difference between the case sensitivities is not noticeable, similar to the overall average accuracy. While (a), (c) appear to plateau at their average, it seems that (b), (d) are steadily increasing. This is due to the number of total detected images being much lower than the total processed images. It is likely that if (a), (c) were to be cut off at the 1500 mark, they would display the graph shape as that is before the plateau begins. Therefore, the averages calculated using the number of total images where a word was detected may increase slightly before plateauing if given a larger sample size.

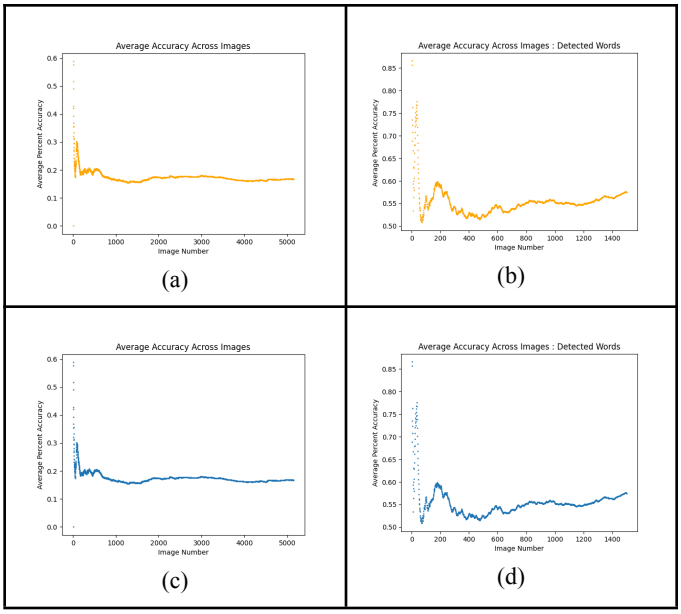

Fig. 2. (a), (b) case sensitive, the average accuracy each time a new image was processed, for total processed and total detected respectively. (c), (d) same as (a), (b) but for results where the letter case is ignored.

While the performance accuracy is not extremely high, it seems that the main issue with our implementation is that it has trouble with text detection, as over half of the possible Latin script images were passed over. Given our simple dataset of one-word images, it is likely that the performance will drop significantly given a natural scene image. The previously discussed EAST and CRAFT algorithms were aimed at solving this problem. The results displayed in [5, Fig. (6)] and [6, Fig. (8)] when compared to the results of our solely pyTesseract solution show that our implementation has rigid limitations, especially when dealing with any large degree of skew or 3D distortion.

IV. Conclusion

The historical evolution of OCR was first discussed. We then went more in-depth into two advanced algorithms for text recognition: CRAFT and EAST. CRAFT (Character Region Awareness for Text Detection) is an algorithm designed for multi-oriented text in images. It produces character and affinity scores that are used with Gaussian heatmaps to present more accurate results. It also can identify various text scripts. EAST (Efficient and Accurate Text Detector) is an algorithm that is similar and also addresses issues with text orientations and sizes. Tesseract, via the PyTesseract library, was used for text recognition in our project. The main parts of the OCR system were outlined, including pre-processing, feature extraction, text recognition, and post-processing. Pre-processing enhances the quality of an image which makes the feature extraction a more effective process. In feature extraction, key patterns found in the image are extracted. These key patterns are then compared to original features and patterns for detecting and identifying text are then found. Finally, in post-processing, the text received from the model is then saved in a text file and further evaluated. In short, in this project, we were able to use OCR and Tesseract Engine to successfully detect Latin text from images.

This project explored the concept of text detection and recognition in images. The overall challenge being addressed was the identification and detection of text in images using Optical Character Recognition for text recognition. The combination of openCV and Tesseract is limited by requiring optimal images to locate text. The burgeoning algorithms using deep learning such as EAST and CRAFT continue to tackle these problems.

Table 2. Team Members and Their Contributions

Member Name	**Contribution**
Manav Patel	Abstract, Research, Technical Implementation, Experiments
Gurveen Kaur	Introduction, Research, Technical Implementation, Experiments
Brenda Tam	Analysis, Research, Technical Implementation, Experiments

V. References

[1]“MLT2019 Dataset - Downloads - ICDAR 2019 Robust Reading Challenge on Multi-lingual scene text detection and recognition - Robust Reading Competition,” Cvc.uab.es, 2019. https://rrc.cvc.uab.es/?ch=15&com=downloads

[2] R. Smith, “An overview of the tesseract OCR engine,” *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*, 2007. doi:10.1109/icdar.2007.4376991

[3] K. Karthick, K. B. Ravindrakumar, R. Francis, and S. Ilankannan, “Steps Involved in Text Recognition and Recent Research in OCR; A Study,” *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, no. 1, May 2019.

[4] R. Karpinski, D. Lohani, and A. Belaid, “Metrics for Complete Evaluation of OCR Performance,” *IPCV’18 - The 22nd Int’l Conf on Image Processing, Computer Vision, & Pattern Recognition*, Jul. 2018.

[5] X. Zhou *et al.*, “East: An efficient and accurate scene text detector,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi:10.1109/cvpr.2017.283

[6] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee, “Character region awareness for text detection,” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. doi:10.1109/cvpr.2019.0095

[7] A. Zhang, M. Li, Z. Lipton, and A. J. Smola, *Dive into Deep Learning*. Cambridge, UK: Cambridge University Press, 2023.

[8] P. Tripathi, “A journey through history: The evolution of OCR technology,” A Journey Through History: The Evolution of OCR Technology, https://www.docsumo.com/blog/optical-character-recognition-history (accessed Dec. 15, 2023).