Michelle Tan, Maitri Mangal, Song Chen

Abstract

Text recognition has emerged as a very important problem when parsing through large volumes of image and text data. The United States postal service has to process handwritten and non-handwritten text on parcels and mail. Attorneys, accountants, and various government agencies need to quickly and accurately process and store legal and financial documents, containing both hand and non-handwritten text. Text Recognition can also be important for capturing highway and street roadway signs to make sure they have adequate signs in place. Thus, we seek to write an algorithm for processing images containing text into plain text. First, an image needs to be properly segmented into textual and non-textual regions. Next the textual region will need to extracted and sentences or words will need to be further segmented to extract characters. Finally, those characters will need to be compared to each alphanumeric (and grammar characters) using an image correlation technique. We compare our methods against existing Optical Character Recognition algorithms using varying images containing and not containing text. We use accuracy and speed as measures of a successful algorithm.

Introduction

Optical Character Recognition techniques can generally be broken down into preprocessing, feature extraction, and then classification. Most raw images of text usually contain noise and other non-text objects. The text may also be skewed (not vertically aligned). Preprocessing is usually done to remove this extra unnecessary information to retrieve only the text regions and make it easier to extract features from the image such as smoothing, zoning, and

binarization. Newer techniques for Character Recognition involve machine Learning Techniques such as Nearest Neighbor and Neural Network based algorithms. First we give a general overview of the techniques used for Character and Textual Recognition, broken down into 3 categories: preprocessing, extraction, and classification. We then give a brief summary of a completely different technique, crowdsourcing humans to do simple tasks such as text classification.

There are multiple preprocessing techniques for OCR. One technique that is important and common in preprocessing is thresholding. There are multiple different ways to threshold. Thresholding is a technique where an image gets segmented based on specific pixel values in a neighborhood. There are different ways to calculate the threshold for different neighborhoods. F or example a simple method for thresholding is to take the average value of all the pixels in the image and threshold the whole image based off of that value. Another preprocessing technique is deskewing. Deskewing involves taking a tilted or skewed image and rotate it back so that all the text is vertically aligned. One method for deskewing involves calculating the rotation matrix of the image. One final preprocessing technique used is erosion and dilation. These techniques can be used to clarify letters or close small imperfection in the original image.

One technique for extracting features is through contour profiles. By tracing along the curve of a single given character/ glyph, a parametric curve can be approximated. Next the coefficients of the curve can be used as a set of features for a classifier, such as K nearest

neighbors. A drastically different technique for extracting features is to focus on retrieving the number of intersections, loops, and junctions within the glyph. Given an gray-level image input character, the image is treated as a topographic map, with increasing elevation as gray level is increased. Ridge lines and saddle points can be retrieved to create a graph consisting of arcs, loops, and line segments. Similarly, if the binarized character is first thinned, discrete features such as the # of T-joints, isolated dots, total # of endpoints, # of crossings, etc. can be extracted. All of these techniques can be used to retrieve features of a given input but these features still need to used in some way to classify the given character as one of a pre-set database of glyphs.

The most simplest of classification techniques is through pattern matching or template matching (which is the method we use). Essentially, for each preset glyph, some type of pixel by pixel comparison is done between the glyph and the given input to calculate a score. The glyph with the highest score is the best match. Some more advanced techniques involve supervised machine learning, such as K- Nearest Neighbors and Neural Network based algorithms. Using one of the feature extraction methods mentioned earlier, a given inputs features can be compared against the feature vectors of a set of labelled training points. The classification matches the training point with the closest distance (usually Euclidean Distance metric). To remove the effect of noise or unbalanced training sets, the classification can be done against the k closest points (either weighted or unweighted combination). Feed forward Neural Networks is another supervised learning algorithm that classifies texts and characters well. For example, a given input character can be scaled down to a 16*16 image and input as a 256 data points vector. By feeding this vector forward through hidden layers, the Neural Network is able to extract features (hidden from us) and then at the last layer/node perform classification.

A completely different approach to text recognition is through massive online human computation. For example, Amazon uses “mechanical turks”, paying a small amount to people who solve simple tasks that people post online. By applying mechanical turks to the problem of text recognition, the problem completely changes from a feature extraction/classification problem to a distribution networks problem.

Method

For preprocessing we use multiple techniques in order to prepare the text for segmentation. To start the image gets thresholded with the opencv adaptive threshold method. In the adaptive threshold method the algorithm passes a gaussian window through the image and this method calculates the threshold based on the weighted sum of the pixels in the window. This results in a threshold is that it considers the local dark and light values or a pixel neighborhood. For example global thresholding an image of a crumpled piece of paper text in a shadowed area would be blacked out along with the shadow, while with adaptive thresholding only the text gets thresholded.

Another technique used in preprocessing is finding the rotational matrix of the image. In linear algebra a rotation matrix may be applied to a matrix to rotate it. Because an image is a fancy matrix a rotation matrix can help. OpenCv has a function that calculates a rotational matrix using the image and angle of rotation. After calculating the rotation matrix we apply it using the warpaffine function from opencv. Warpaffine applies an affine transformation to the image. An affine transformation combines linear transformation with translations.

Once the preprocessing of the image is done, it is important to start segmenting the image. Segmentation is essentially the process that allows for the extraction of the text from the

image background. Once the characters have been extracted from the image, they will be passed onto the next step which is actually recognizing these characters. The first step of text segmentation is to downsample the image itself. First, the image is blurred down by convolving the source image with the following Gaussian kernel.

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Following this, the even rows and columns are rejected to downsample the image. Downsampling of the image is important because it allows the process to not pick up as many noises that may be present in the image, and may thus occlude the image for text segmentation. It also allows the processor to not pick up as many "not so strong" edges. These edges may be present as part of the background of an image, and not the text itself. Thus, it is important to eliminate these edges from the image before actually connecting components for text segmentation.

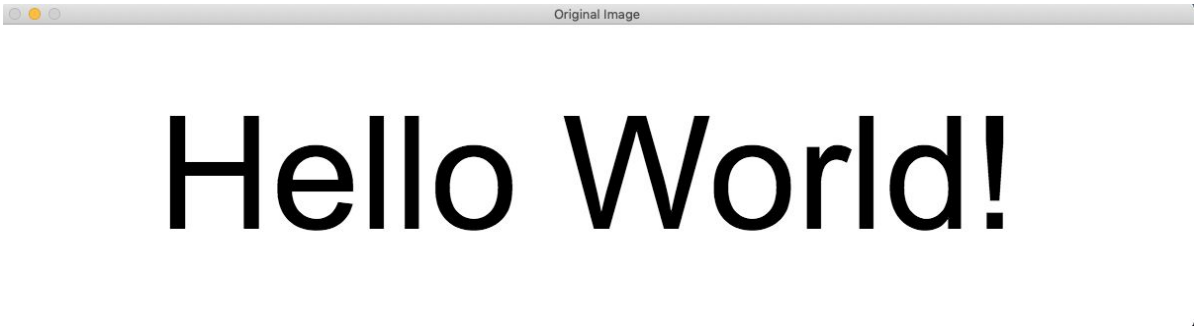

Image A: Original Image

Image B: Downsample Image

The next step is to create a structuring element in an ellipse shape. This shape will be of size(3, 3). Once this kernel has been created, the morphological extraction is carried out to perform a morphological transformation on the image. For this transformation the same elliptical kernel will be used.

Image C: Morphological Ellipse Image

Once this transformation is finished, the image is then transformed into a binary image using OpenCV's Binary threshold value and the OTSU threshold value. Next another morphological transformation is performed on this binary image. But this time, the morphological kernel used is of a rectangle with the size(9, 1). This allows for all the horizontally oriented regions to come together. It is essentially getting the horizontal oriented regions connected to one another.

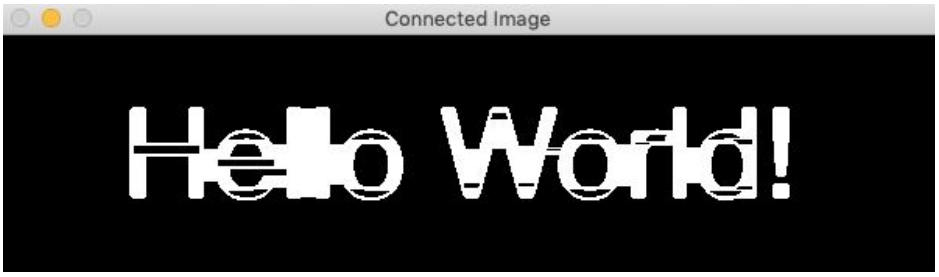

Image D: Morphological Rectangular Connected Components Image

Once the image has been identified for horizontal connected components, the next step is to essentially find all the connected contours. Contours are the curve that joins all the continuous points with the same color or intensity. Thus, it was important to convert the image into a binary image, and then find the connected components of the image. In this way, all of the individual words were already connected together, and were in the same color value. Thus, to find the contour of this image, it was essentially just detecting which were connected, and with 255 pixel value as seen in the following Image.

Once all of the contours were detected, it was necessary to filter out these contours. That is because the connected components of the image may not have necessarily only gotten the components that are supposed to be detected as words or characters. Thus, in order to filter out the contours, a ratio is selected to determine if there are texts in that region. This ratio is represented the nonzero pixel values in the region to the image size itself. If the ratio was greater than .5, then a rectangular box was drawn around this region of interest. This cropped section was also put into a vector of matrices for later use to detect the characters themselves. In this part of the project only individual words were detected.

Image E: Final Image after Extraction

Once each individual words have been extracted, it is important to extract each individual characters. This will help in the next few steps where the images are put into the character recognition methods. For the extraction of each individual characters, we follow the 4 connected components algorithm. For this algorithm, we use a queue to do a breadth first search on all the pixels to see which ones have been visited, and which ones have not. We keep track of whichever ones have been visited, by coloring the pixel a certain given color. Here we also keep track of the min and max x and y coordinates of each connected component. This will help in the extraction of the individual characters. Once each character has been identified by the connected components method, it is cropped out of the image and stored.

Image F: Character Segmentation

Each character image then reduce it to its morphological skeleton. To reduce the image the algorithm uses morphological operations such as erosion, dilation, opening and closing. For this creating the morphological skeleton the algorithm performs a thinning operation. The thinning operation uses the erosion operator with a structuring element. The operation also uses an erosion kernel which weights specific values in the structuring element. The reason thinning gets used because it preserves the topology of the image it also produces a skeleton of one pixel, which is good for comparing the image to a glyph. The algorithm then takes the skeleton and performs a pixel by pixel comparison. The comparison returns a percent of pixels inside the glyph and then guesses the character whose glyph has the highest percent. Image G demonstrates the skeletonization process.

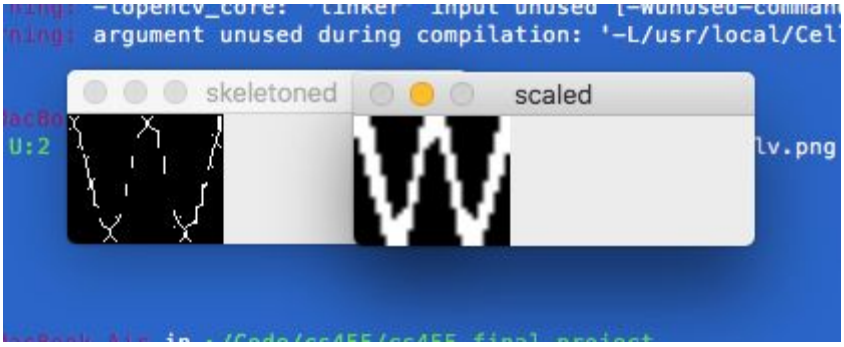

Image G: comparison between the skeletonization of the character W and the glyph for W

Results

For preprocessing the results look very good when the text is type on a flat surface. For example the following image shows the before and after of the text preprocessing for an image created on a computer.

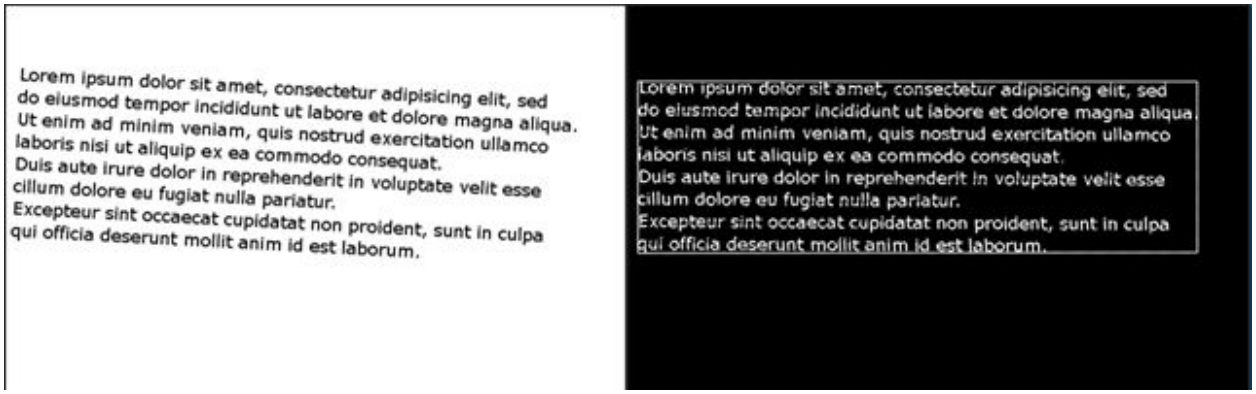

Image H

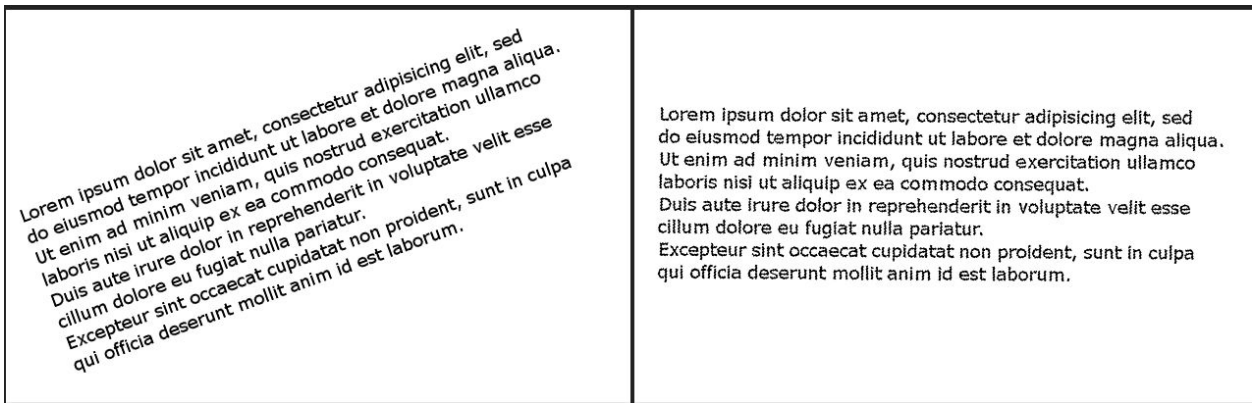

Image I

These images are vertically aligned. The quality of the text gets a little degraded but it can be fixed with a dilation. However, for handwritten text the deskewing does not work as well. The following is an example of the algorithm applied to handwritten characters.

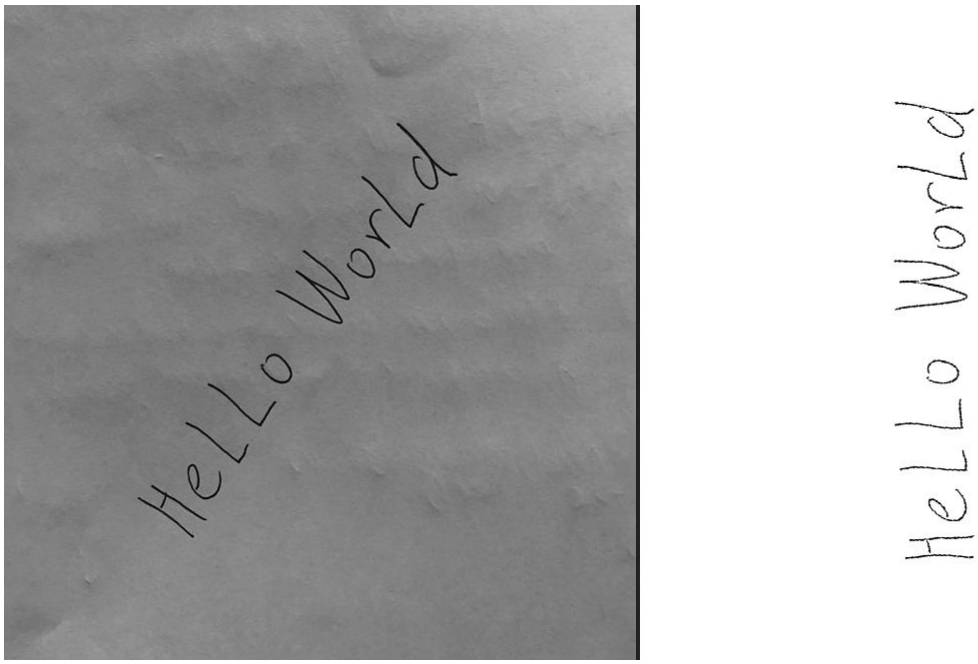

Image J

The text gets flipped to be horizontal. Other tests show that the text gets adjusted but does not become fully vertical. For example in the following photo the text gets adjusted but still has a slight tilt.

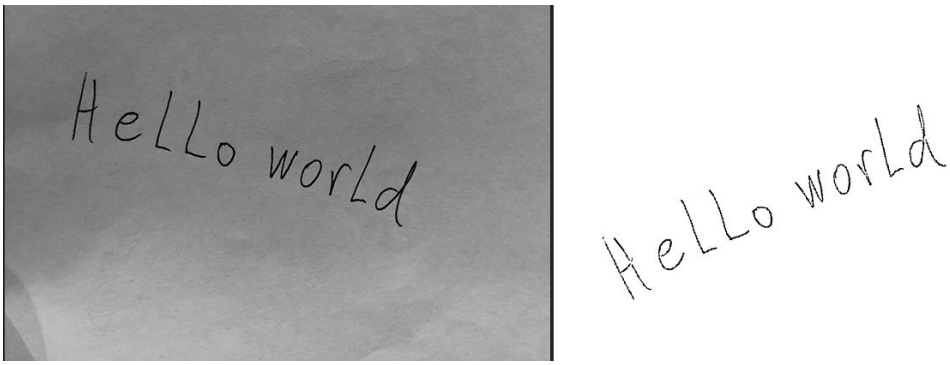

Image K

Tesseract OCR vs. Our Proposed Algorithm

	Correct Out	Tesseract Out	Tesseract Speed	Proposed Out	Proposed Speed
Basic Image	“HELLO WORLD”	“HELLO WORLD”	Not noticeable	“HELLo WoRLD”	315.745 seconds
Skewed Image	“Hello World”	“Mello Worlg”	Not noticeable	“Hello wOrld”	331.44 seconds
Different Font (Times)	“Hello World”	“Hello World”	Not noticeable	“.ElIo world”	355.247 seconds
Text Parag.	“Lorem …”	“Lorem ...” (100% correct)	Not noticeable	“”	NA
Slightly Skewed Book Title	“HARRY POTTER and the Goblet of Fire”	“”	Not noticeable	“”	NA

Table 1

We ran an experiment to compare our proposed algorithm to the popular Deep Learning based Tesseract OCR open source program developed by GOOGLE (Table 1). We compared the two based on speed and accuracy against a variety of images that included different fonts, skew, and text paragraphs. We can see from the results that the main difference between the two was the speed. Because our proposed algorithm did a pixel by pixel comparison to every glyph in our database (~60 alphanumeric) our algorithm was much slower than the Tesseract OCR algorithm. Also, our algorithm could not correctly segment long paragraphs into individual characters, only into separate paragraphs. Many of our outputs had somewhat harmless inaccuracies. Characters where the lowercase and uppercase looked very similar were easily confused for the other. Also,

the proposed algorithm only works for one type of font: Helvetica, because these are the only types of glyphs we have saved into our database. Therefore, using other types of fonts produced much lower accuracy. Finally, our skewness detection algorithm did not work properly for the real example book title; it produced a blank image and therefore text regions and characters could not be recognized.

Conclusion

Our approach to this problem resulted in the development of a pixel by pixel comparison algorithm. We take the skeleton of the character and compare it to all the glyphs. The algorithm then calculates the percent of overlap of the skeleton with the glyph. It then guesses which letter the skeleton came from. The approach has certain limitations, for example our algorithm is limited to fonts similar to helvetica. For example consider the following image M, image M has typography that has a unique font with wide letters. The width of the letters would change the skeleton of each character. This makes it harder to match up character skeletons to glyphs.

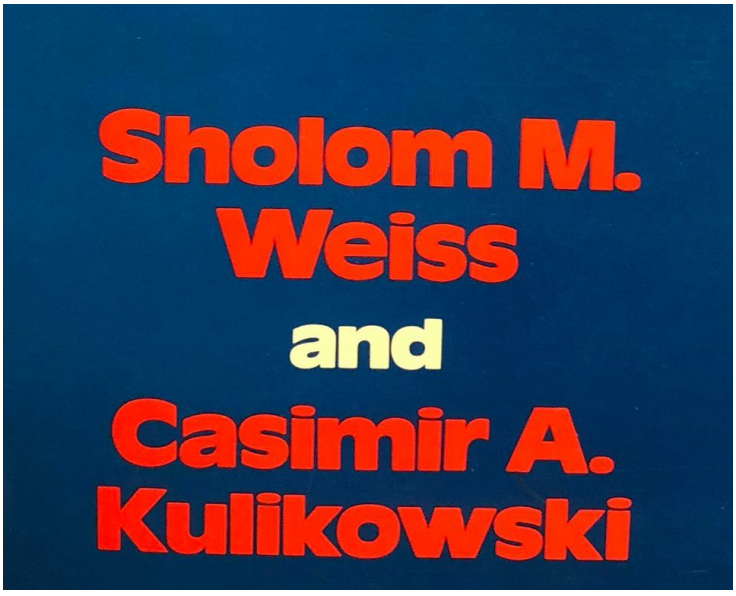

Image M: Cover with short bold text

We get a similar issue with handwritten characters when comparing the skeleton of the characters to glyphs. Handwritten characters have too much variance to be able to rely on the glyph comparisons.

Another issue we have with our approach involves preprocessing warped text. While current preprocessing deals with rotated text boxes, it does not deal with other types of warped text. For example in image N has text on an item.

Image N: example of warped text on a mug

The text on the mug is different from our test images in two ways. The text is facing away from the camera and as a result it is warped in a way that a simple rotation will not fix. Another thing is that the background of the image is not a consistent like in image A, where the background is all blue. Our algorithm may mistake the mug for text or not even notice the text at all. One way to fix the previous two problems would be to use machine learning and train the machine on a dataset of images with objects that have text on them. The problem with this approach is that it would require a large amount of labelled data.

One thing our group can implement in the future is identifying text and performing OCR on a image of a street with multiple signs. One example of this type of image would be image O.

Image O: example of a photo of a neighborhood with text

In image O there is a street with multiple signs in multiple fonts with different colors. The right most sign even has strangely warped text. In order for our algorithm to be able to read the text from this image, we first must work on a way to locate multiple texts in a large image. Because this image contains more than textboxes (buildings, trees, etc) we would need to use pattern matching to separate possible text boxes from other elements in the image.

It is important to note that our algorithm contains many inefficiencies that would not be practical for real applications. Pixel by pixel comparison for classification against a database of glyphs is extremely slow. It is also very inaccurate and is extremely limited. Even in perfect conditions matching against the correct font, minor errors and confusion against lowercase and uppercase still occurs. Therefore, it is recommended to those building an OCR system NOT use any type of pixel by pixel comparison for their classification portion.

References

[1] https://www.irjet.net/archives/V4/i6/IRJET-V4I629.pdf

[2] https://docs.opencv.org/2.4/

[3] https://ieeexplore.ieee.org/abstract/document/5169511 Real world uses: Licence plate recognition

[4]https://www.semanticscholar.org/paper/Feature-extraction-methods-for-character-survey-Trier-Jain/1215253e17ea39b1b31ae6777721880af08e5fbf?navId=paper-header Feature Extraction Methods for Character Recognition - A survey

[5] https://dl.acm.org/citation.cfm?id=1866040 Using Mechanical Turks (crowdsourcing)