Recognize and extract font letters from an image

Ran Liu
Yuchen Wang
Nolan Chen

Abstract

We aimed to develop a program that crops letters from the image regardless of the perspective of the letters. When users upload an image with their desired font, the project would first detect corners of the target paper in this image by using Morphological operations. Then, the paper would be re-projected where its corners wwould be matched to the image's corners. Upper and lower cases of 26 letters would be recognized by functions in the pytesseract module. The final step is to crop these letters out from the original image, resize them into the same size and convert the image to binary image by thresholding method. The final results are a set of pngs for every single letter.

1. Introduction

1.1. Motivation

Fonts have been an important medium for communication, and after entering the digital era, its property of being reproducible enables fancy fonts on every digital documents. Imagine when people hang their posters on the poster wall, they find that other posters have fancier fonts than theirs, which they might want to use for their next posters. Also, sometimes people want to use their own handwritten fonts that can be typed in computers and don't know how to transform their handwritten font from physical to digital. Thus, people need to create the fonts they like. The existing methods are limited in flexibility and time consuming. As far as we know, people need to download a template, write on the template and upload it to a website to generate fonts, or they need to use the complicated font creating software, which requires time to learn. This motivates us to develop this program, which we wish to be flexible and simple in the process and generates the result conveniently and efficiently by reducing human's work.

1.2. Aims

When we started to design the algorithm, we first worked on letters detection on simple inputs that do not require any perspective transformations. Then, we gradually increased the complexity and variety of the input and added different methods and assumptions to handle different inputs. Eventually, we came up with four different aims to achieve our goal of generating pngs of letters.

Our first aim was to develop an algorithm to detect the corners of the target paper for later perspective transformation. Initially, we planned to ask users to input the coordinates of corners rather than using an algorithm, but we thought it would contradict our motivation of reducing human's work. Our second aim was to reproject paper to the top view for letter recognition and cropping. Since the image may not present a perfect top view of the paper, transforming its perspective would increase the accuracy of letter recognition and cropping. It also helped to delete the noise around the paper. Our third aim was to recognize upper and lower cases of 26 letters, and the fourth aim was to crop the letters from the original image, resize them to the same size, and output them as pngs. For the fourth aim, we initially planned to generate an OpenType font file (otf) or a TrueType font file (ttf), but our algorithm cannot guarantee to recognize all of the letters, or the image might not include all the letters. Thus, we finally decided to generate pngs rather than font files.

2. BACKGROUND INFORMATION

2.1. Tesseract

Tesseract is an open source optical character recognition platform, which extracts text from images and documents without a text layer and outputs the document into a new searchable text file, PDF, or most other popular formats. [4] It mainly includes three emphasis areas, which are line finding, features/classification methods, and the adaptive classifier, where it assumes that its input is a binary image with optional polygonal text regions defined. The recognition process of Tesseract includes two passes. The first pass attempts to recognize each word in turn, where the satisfied ones will be passed to an adaptive classifier as training data. The second pass will run over the page again and recognize the words that are not recognized well.

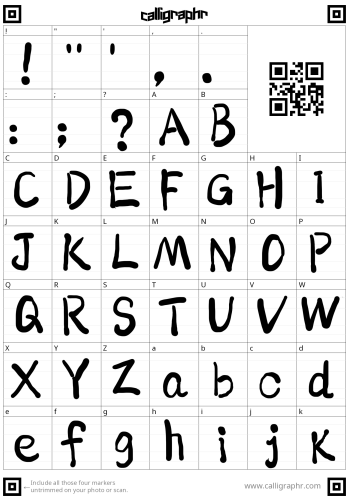


Figure 1. Calligraphr

2.2. Projective Transformation

Projective Transformation is a combination of affine transformations and projective wraps. It is used to describe what happens to the perceived positions of observed objects when the point of view of the observer changes. This transformation does not preserve sizes and angles of the object, but it does preserve incidence and cross-ratio, which are important properties in projective geometry. [2]

3. RELATED WORK

3.1. Calligraphr

Calligraphr is s one of the most popular font creating platforms. It's a website where people can download the font template, write on it and upload it to get a ttf and an otf file. However, its limitation is that it only takes fonts on the template but not any paper with text. If people have created their fonts somewhere else, they must write again on the template, which we tried to overcome in our project. There are four rectangles at the four corners of the template used for perspective transformation, which is also an important inspiration for us [1]. The Figure 1 shows an example input for the website.

4. METHODS STRATEGIES

4.1. Corner Detection

Our first aim was to detect corners of the paper in the image. At first, we used the Harris corner function in OpenCV to detect corners of paper, but it couldn't differentiate the corners of paper and the corners in the letters. Thus, we realized that we must recognize the paper first, so we decided to apply a contour to the paper.

Figure 2. Morphological Operation

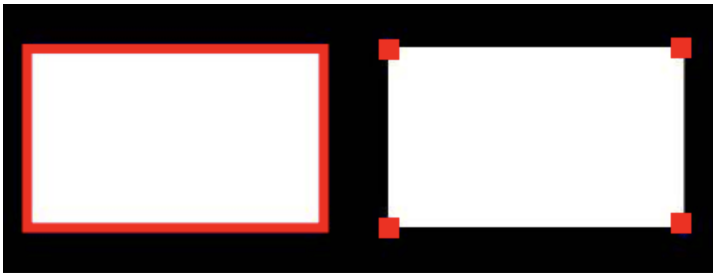


Figure 3. Detect corners using the shape of the contour

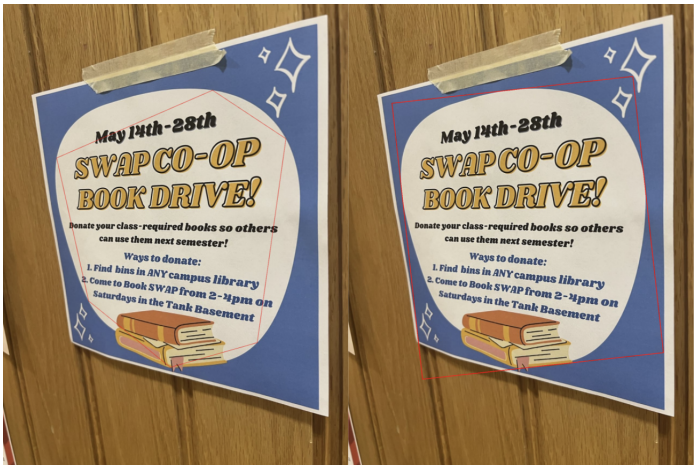


Figure 4. When the contour has more than four edges, we apply the smallest rectangle bounding to the contour.

However, directly applying the contour to the image would cause inaccuracy in edge recognition because of the noise and the vague edges, so we tried various methods to make the shapes clear, including transforming the image to grayscale, blurring the image using a gaussian filter, and thresholding the image using a mean threshold. None of the worked well, but eventually, we found a useful method called morphological operation in the OpenCV library [3].

It processes images based on shapes. It has seven tools, and we were using the Opening and Closing modes of this function. Both of them were generally used to remove the noise in the image, where the Opening involves erosion followed by dilation in the outer surface, while Closing involves dilation followed by erosion. Erosion primarily involves eroding the foreground(which is the outer surface) of the image, while Dilation primarily involves expanding that. Both of them would suggest the foreground as white. Since morphological operation requires a binary image, and blurring and thresholding the image would also increase the quality of morphological transformation by reducing the noise and enhancing the variation of pixels, we decided to keep all the previous pre-processing methods we applied. The result of pre-processing is shown in figure2.

The next step was to find the contour of the paper. The findContours function in OpenCV finded every possible contour based on the change in color through the image, and then we used the contourArea function to find the largest contour in the image, which we assumed was the contour of the paper. It means that the paper with text must be the largest shape in the image, or the algorithm cannot recognize the paper correctly.

Then, we applied approxPolyDP function to the largest contour, which returned a set of corners representing the shape of the contour. However, when we tried to use the corners of contour to transform the image, we found that in some cases the contour has more than 4 corners because the contour is confused by the real edges of paper and the shapes in the paper. Since we couldn't transform the image properly by using more than 4 corners, and we struggled to remove shapes in the paper, we decided to handle this case by applying the meanAreaRect function in OpenCv, (https://theailearner.com/tag/cv2-minarearect/) which draws the smallest rectangle bounding to the contour. The perspective transformation would be limited to rotation, but this is the only solution we have so far.

4.2. Perspective transformation

As we had the corners of paper, we applied perspective transformation for a better quality of letters recognition and cropping. We spent a long time figuring out where to reproject the image to keep the aspect ratio of the paper. We thought about asking for the ratio or the type of paper, but the aspect ratio is not easy to estimate by looking at the image. In the end, we decided to project the corners of paper to the corners of the image, making an assumption that the paper in the image has the same aspect ratio as the image. This method worked well for most cases, such as using smartphones to take pictures of American letter paper or A4 paper.

After the projection, the resulting image might be rotated due to the order of the projected corners. We tried to adjust

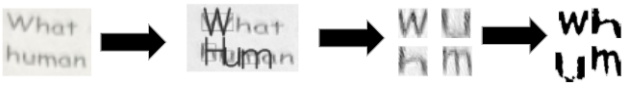


Figure 5. Process

the corners by comparing their x and y coordinates, but then we realized that the relation between the coordinates of the four corners is not constant. For example, sometimes the upper-left corner had a larger x then the lower-right corner. What's more, the paper may not be in the right perspective from the beginning. Then, we tried to use the image_to_osd function in Pytesseract to rotate the image. The function returned the orientation of the text, but after a few tests, we found out that the rotated text would cause significant low accuracy in letters detection. Thus, We finally decided to ask users to rotate the image to the right angle.

4.3. Letter Recognition

Now we had the image with text in the correct direction. Then to get a better recognition result, we used a filter to deblur the image and transferred the image into grayscale. This step used pytesseract(tesseract in python library). As introduced earlier, it is a wrapper for Tesseract-OCR Engine, which is used to recognize and "read" the text embedded in images. In this project, we aimed to recognize the text in the image and also get the locations of those text. So we used two functions from the module: image_to_string and image_to_boxes. We used the image_to_boxes function first, it allowed the program to print out the exact position of text. It createed the imaginary boxes around each text and returned the coordinates of the upper left corner and lower right corner of this imaginary box. Then we got the location coordinates of all text in the image. Then we used the image_to_string function to recognize the letters in this image and convert them to a string.

Then we matched the recognized letters with the positions returned by the imagetoboxes function. In order to avoid confusion, we only kept the image of letter when the letter is detected for the first time by saving them into an array. However, we may get some bad results in this way since we cannot make sure the first time detection is the best detection for the letter. We tried the function image_to_data which can return the confidence of detection. But the function only returned the detection of the words but not a single letter. We found it was quite difficult to separate letters from the result of this function and turned back to the image_to_string function. Then after this step, we would get 52 upper and lower case English letters in the best situation.

4.4. Cropping and resizing

Then to make the letter image easier for people to use, we cropped, resized and turned the image to binary image at

last. The figure2 shows the whole process. We used the coordinates of the letter positions to calculate the range of the letter array within the original image array. Then we could take the letter images out of the original image. For the resize part, we initially defined a standard size for letters, but we realized that the sizes of letters might be vary depending on the input. Thus, we decided to calculate the mean height of each box of letters, resize the height of boxes to the mean height and set the width of boxes based on the ratio of original width and height. The program successfully cropped the letters and resized them in most of the case, but the resulting images were blur and had various background color. To solve this problem, we transfered each resized letter image to the binary image by convolving and thresholding method.

5. RESULTS

In the case shown in figure 6, the original image came from a screenshot of a handwriting on an iPad. The program successfully recognized and cropped 47 of 52 letters, and there is no obvious noise in the binary images, which is the best results we got so far. We notice that besides the failure to detect upper A, lower f, lower l, and lower g, it recognized i and j together as lower i, which could be solved by leaving more space between the letters. Another problem is that our program resized all the letters to the same size, while sometimes the lower cases should be smaller than the upper cases. For "c", "k", "m" and other letters of which the lower and upper cases are the same, it is hard to differentiate between the lower and the upper letters.

Figure 7 shows a general result from our program, which indicates our motivation of improving the flexibility and convenience of font creating. All the pre-processing methods before letters recognition were used in this case. The results are not in good quality, as only 41 of 52 letters were detected, and 26 of 52 letters were cropped accurately. We think the reason is that after pre-processing the image, the noise cannot be fully removed from the image, which confuses our letter detecting engine (tesseract). Also, the quality of the image also affects the final result, since the letters are not clear after pespective transformation.

Figure 8 shows a bad result. According to our analysis, we think there are two reasons why the result is not ideal. First, since there are a circular shape inside the paper, which cannot be removed during the pre-processing phase, the program incorrectly recognizes it as the paper and handles it by drawing a rectangle bounding to the contour as the paper. Thus, the only available transformation is rotating the image and the letters are difficult to detect. As we investigated further, we discovered that in the pre-processing phase, some parts of the edges of paper were removed, so there won't be a contour on paper, and this causes the shape inside the paper has the largest contour As we mentioned before, the program assumes the largest contour as the contour on the paper. The second reason is that the fonts in the paper might be too fancy to be recognized by tesseract.

6. CONCLUSION

As shown in the result part, we can successfully recognize letters from paper with a clean background which basically achieve our original aim of this project. And the project helps us understand more about the computer vision techniques we learned from the class. However, there are also some limitations. We mentioned in the Aims part that the approach did not perform well in a messy background and also it cannot choose the detected version based on confidence.

7. FUTURE WORK

Since our original aim of the project was finding a way for people to use the font in the image directly, the next step of the project should be to find a way to turn those single letter images into a OTF/TTF file which people can use directly.

Then, it is reasonable to deal with the limitations of the projects, such as detecting in messy environments and choosing detected letters according to the confidence level(find a good way to use image_to_data function). It is also valuable to train our own model of font detector, in which we could get the easy access to the confidence. At last, we think it would be nice to check if there is any existing font is similar to the detected image and people can easily use the font by knowing the name of it. When we did the research, we found the all existing approach to detect fonts utilized CNN which we have no time to train. But it would be a good direction to work on in the future.

Besides letter detection, another potential future development is to normalize the letters in a more flexible way depending on the standard aspect ratios of letters. For now, the upper and lower cases have the same size.

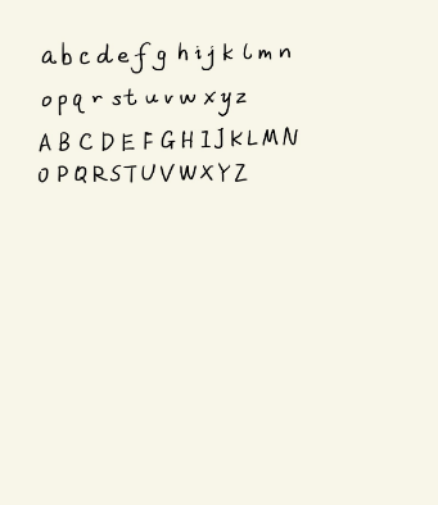
References

[1] Calligraphr. https://www.calligraphr.com/en/. 2

[2] Aurigma. Affine and Projective Transformations. *Aurigma Graphics Mill 5.5 for .NET*. 2

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 2

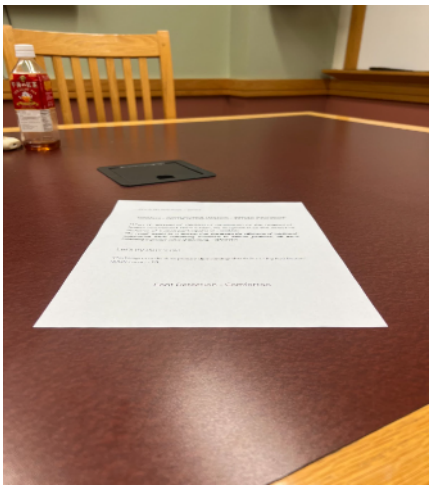
[4] R. Smith. An overview of the tesseract ocr engine. *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 2:629–633, 2007. 1



(a) original image

(b) binary result

Figure 6. Best Result



(a) original image

(b) binary result

Figure 7. General Result

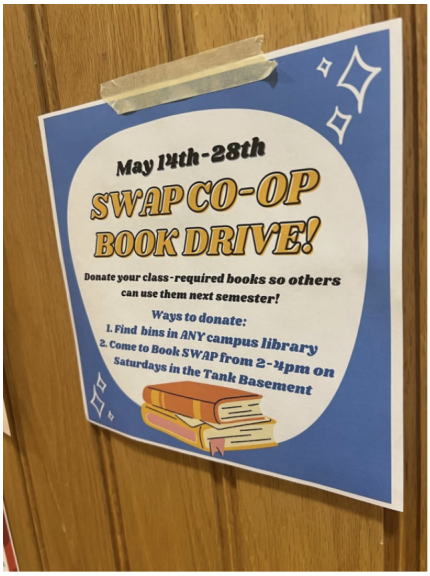


Figure 8. bad result