Document Scanner

CS 6643 - Computer Vision

Shuhan Zhang
sz2898@nyu.edu

Yinjia Huang
yh3257@nyu.edu

Xuanbin Luo
xl2806@nyu.edu

1 Introduction

We always meet situations that a electronic version of a document is need for communication or archiving, while a scanner is not available. People should not be bound by lacking of equipments. In this project, we use computer vision methods to develop a system that transform pictures of document into a more formal and readable version and also extract the text from the document for more purposes. In this way, we can replace and even go beyond a scanner with a camera and a computer which are more common and convenient. To achieve this goal, we use traditional methods of the computer vision the features of image processing and a deep learning method to achieve optical character recognition. More specifically, we take in the image, find the edge of the document to cut it out, apply geometric transformation to it to get a standard top-down view, enhance it to get a readable binary document, segment all the words out and apply optical character recognition to extract the text.

In the report, we give the approaches to achieve all the functions in details in Section 2. We give the data we used and show the results of our system in Section 3. We discuss the strength and weakness in Section 4. We also propose some future works we might do in Section 5.

2 Approach

2.1 Edge Detection and Contour Find

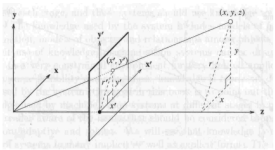
To detect the edge of the document in the picture, we firstly convert the image to grayscale, blur it, and find edges in the image. Here, we use Gaussian Sobel to blur the image use Canny to detect the edge and get its binary image. To make the edges more distinguished, we also use the dilation and erosion. Then, perform topology analysis to image, we get the contours. Figure 1 gives result of different steps.

(a) Original image (b) Gaussian Blur (c) Edge Detection (d) Dilation (e) Erosion

Figure 1: Steps of Edge Detection and Contour Find

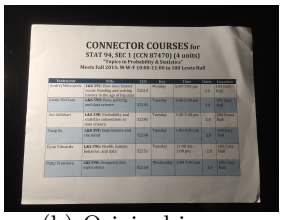
2.2 Image Transformation

Perspective Transformation is the projection of an image onto a new Viewing Plane, also known as Projective Mapping. Figure 2 gives the matrix notation and the perspective transformation result of the picture.

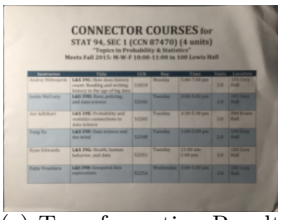


$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(a) Perspective Transformation Matrix



(b) Original image



(c) Transformation Result

Figure 2: Demonstration and Result of Transformation

2.3 Image Enhancement

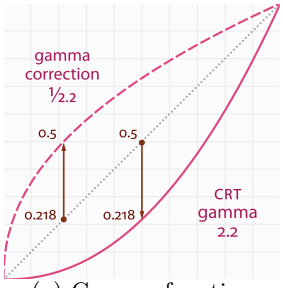
The goal of image enhancement is to produce a binary (black-white) image of the document with clear words. In other words, we want the background to be white but not gray and the words to be black. Also, the words should be neither too thin nor to bold so that both human and our OCR network can distinguish the words. We use three algorithms to enhance the image.

2.3.1 Gamma Correction

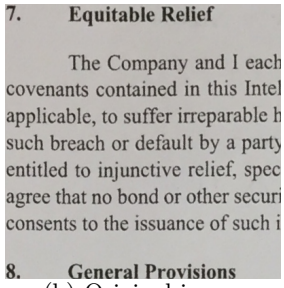
The purpose of gamma correction is to change the luminance of the image. We use the following function to achieve gamma correction:

$$f(I) = I^{\gamma} \tag{1}$$

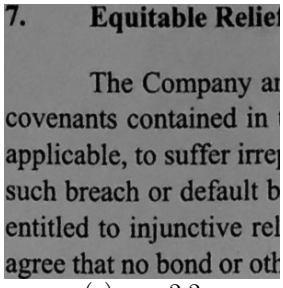
As shown in Figure 3, when $\gamma < 1$, the contrast of the image is enhanced in the area with low gray value and the whole image turns to be lighter. It will also make the words likes thiner. When $\gamma > 1$, it is the opposite. Contrast of the image is enhanced in the area with high gray value, the image turns to be darker and the words likes bolder. In this project, we are dealing with documents and we always want the words to be bold thus they will be clearer for OCR, so we use $\gamma = 2.2 > 1$ to enhance the contrast of words.



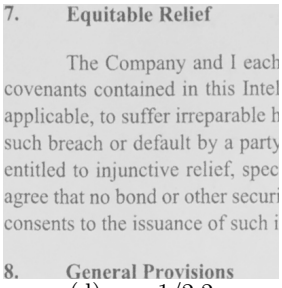
(a) Gamma function



(b) Original image



(c) $\gamma = 2.2$



(d) $\gamma = 1/2.2$

Figure 3: Effect of Gamma Correction

The process of gamma correction is simple. For each pixel in the image, we normalize it to keep it gray value in $[0, 1]$. Then we use equation 1 to calculate the new value of the pixel. Finally, we do inverse normalization to set its gray value to $[0, 255]$. Consider that we are processing large images that always have millions of pixels, I optimize the algorithm by first calculating gamma correction for every gray scale, and store the results in a table. Then for every pixel, we look into the table and find its corresponding value. In this way, we significantly reduces the number of exponential operations and saves a lot of time.

2.3.2 Image Sharpening

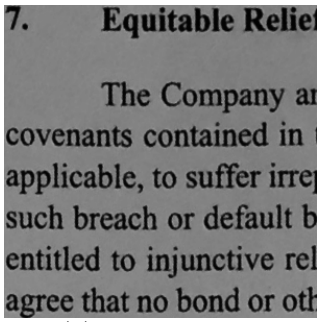
Image sharpening compensates the outlines in the image. It enhance the edges and the area where gray scales change rapidly to make the image clearer. The purpose of image sharpening is to give words sharp and clear edges. It makes easier for image binarization. Clear words also benefit both human reading and OCR.

To achieve the effect of image sharpening, we perform convolution operation on the image with the Laplacian operator in Figure 4(a). It makes dark pixels at the edge area darker, light pixels at the edge area lighter, and

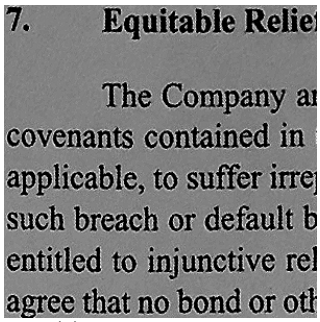
does almost no change to pixels whose neighborhood have same or similar value. We successfully sharpen the words by increasing contrast at edge area. Figure 4 compares the original and processed image and prove our algorithm to be useful.

$$G = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(a) Laplacian Operator



(b) Original image



(c) Processed image

Figure 4: Image Sharpening Using Laplacian Operator

2.3.3 Image Binarization

Image binarization turns a gray-scale image into a binary image, where there are only black (with value 0) and white (with value 255) pixels.

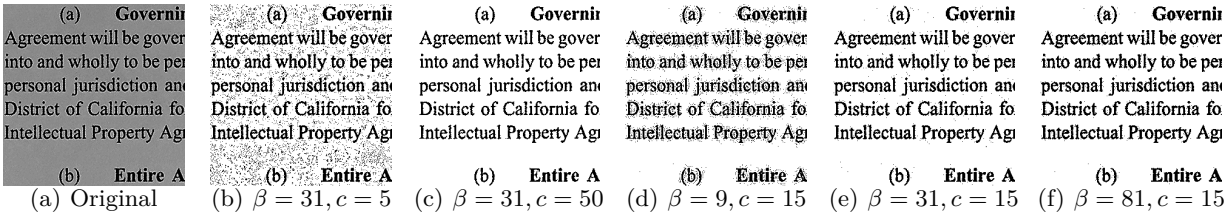
I use adaptive threshold for image binarization. For a given pixel p, it first choose a neighborhood area of size β, and use a Gaussian filter of size size to calculate the weighted average w. Using Gaussian filter as weights can help us better focus on the center area of the neighborhood. Considering that we have an image with sharp edges, this is better than just calculate the mean of whole area. Then, by subtracting a constant c from w, we get the threshold t of p. If $p > t$, it will be a white pixel; Otherwise, it will be a black pixel.

Choosing proper value for factors β and c can help us get a balanced result that fits the following procedures the most.

- For β: A small block size help us focus only on the edge of a word. It produces words that are sharp and thin, surrounded by noises. A large block produce words that are bold,and with less noise. However, the result will be affected if the brightness of the image is uneven in the large block we choose. Salt&pepper noise we produced is not primary issue since they will be eliminated in the next step. In conclusion, a block that is large enough to get bold words with less noise is good enough.

- For c: This factor has more effect on the final result, for it has direct influence on the threshold. A large c will lower the threshold, both decreasing noise and making words thinner. A small c will increase the threshold, thus increase noises and make bolder words. We choose the smallest possible value for c as long as the noises are not overwhelming so we get bolder words for reading and OCR.

Figure 5 compares results with different β and c. We choose $c = 15$ for a smaller c does not improve word quality and produces noises we cannot handle well. We choose $\beta = 31$ for a larger size does not improve word quality while smaller size makes words thinner and even hollow characters.



(a) Original (b) $\beta = 31, c = 5$ (c) $\beta = 31, c = 50$ (d) $\beta = 9, c = 15$ (e) $\beta = 31, c = 15$ (f) $\beta = 81, c = 15$

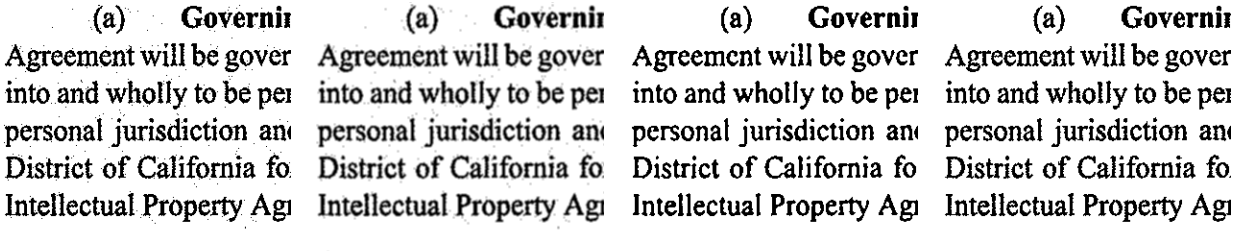
Figure 5: Effect of Adaptive Threshold

2.3.4 Lossless Noise Reduction

Image enhancement will produce images with salt&pepper noise, in other words, black dots. We are looking for a way to eliminate the noise while not damage the sharpness of our image. Given our noises are dots or small group of dots, I think we could use connected components algorithm to finish the task.

For a image, we find all its 4-connected components, and also the size of each component. Generally, characters, punctuations and even the dot on letter 'i' and 'j' have more pixels in its component than a noise. If the size of the component is smaller than a certain number, we think it is a noise and we discard it from the image. In this way, we can eliminate noises without affecting the characters. That is why I call it lossless. However, this method is not universal and only works on binary images.

Figure 6 compares the results using different noise reduce methods. As we can see, Gaussian filter does no help to reduce salt&pepper noise. On contrary, it blurs the image a lot and this is not acceptable. Median filter works really well. It successfully gets rid of almost all noise points. And so does our method. The differences are median filter produce characters with smooth edges but the characters seem uneven in size and are a little blurry, while our method produce clear characters but with jagged edges. It is really hard to say that one method is significantly better than the other. However, considering that median filter actually may change pixels on characters, while the connected components method only eliminates the noise and keeps the pixels on characters unchanged, we choose to use the connected components algorithm to denoise.



(a) Original image (b) Gaussian Blur, $size = 3, \sigma = 1$ (c) Median Blur, $size = 3$ (d) Ours, discard components ≤ 5

Figure 6: Effect of Image Denoise

2.4 Image Segmentation

The goal of image segmentation is to extract words or characters from the document for OCR. Normally, optical character recognition system takes picture of words or characters as input and output the predicted words or characters. There might be a system that can take the whole document as input to do recognition, but it would be too hard for us to implement. Also, in this section, we want to use traditional method to do segmentation instead of a deep learning method. Deep learning method requires a lot of data for training, and we cannot find or manually create the amount of data for training.

The first thing to decide is whether we want to segment the document into words or characters. Characters are much easier for OCR, and that is what we try first. We have tried several methods to segment individual characters, but none of them produce satisfying results. We have to give up and turn to segment document into words. With a proper trained network, it is possible to recognize pictures of words. We will discuss the segmentation of characters and words next.

Segmentation of characters: As we have said, we decide to use traditional methods to do this job, which means we are dealing with groups of black pixels. Three different methods are considered:

- Use connected components to extract characters. After denoising, most black pixels in the image belong to characters, and others belong to the edges of the document. Using the connected components algorithm, we can get almost all characters separately. However, letter 'i' and 'j' will be split into two parts. This is not a good result for OCR because 'i' without the dot can be easily recognize as other characters, '1' or 'l' for example. This method has very obvious shortcomings, so it is not even tested.

- Use Maximally Stable Extremal Regions algorithm. MSER is a region detect algorithm usually performed on gray-scale image. It binarize the image using threshold from 0 to 255. If a region is not covered until the threshold reaches the gray value of the characters, then this region is a maximally stable extremal regions. However, applying MSER on a binary image works very similar to connected components algorithm. Figure 8(a) shows the result. Though it separate most characters, it fails when the character is already connected together. Also, it split the dot from 'i' and 'j'.
- Use image projection and threshold to segment characters. For a given image, we first project it horizontally and get the number of black pixels for every line. Using this projection and a threshold, we can locate the range of every line of text. Then, for every line, we project it vertically, which will help us locate the position of every character. Figure 7 shows the horizontal and vertical projection of a line and also the segment result of it. As show in Figure 8(b), this method will not leave the dot out, but still, it might separate several characters together. It will split 'm', 'n' or 'h' into two parts sometimes. The gap between characters are so small that small threshold may cause groups of characters together while large threshold may cause a single character to split into two.

From the analysis above, we can see that traditional methods we use have great drawbacks on character segmentation. Every fail in segmentation will cause a fail in character recognition. With so many characters in a sentence, we may get multiple wrong recognitions in one sentence. That will make the sentence unreadable. We decided to turn to word segmentation. Segment a word is much easer, which leads to higher accuracy in segmentation and recognition. And under most circumstances, people can still guess and recover a sentence even with a misplaced word. That makes our result more readable.

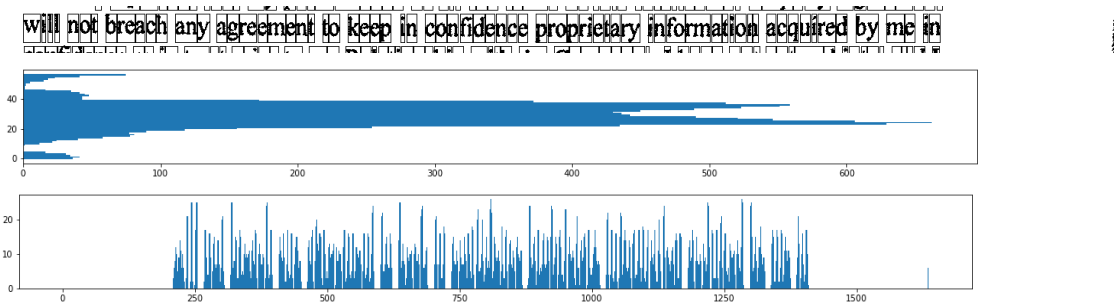
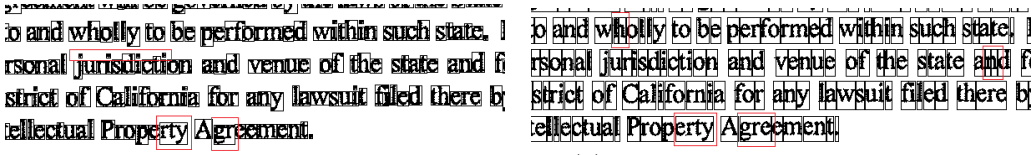


Figure 7: Segmentation and Projection

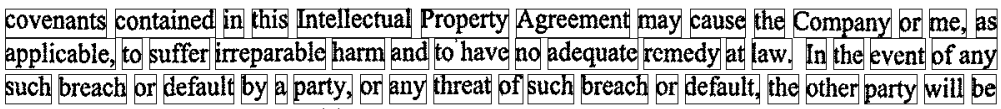
Segmentation of words: Word segmentation use the same projection method as character segmentation. There are large gap between words, making it a lot easier for us to locate every word. Figure 8(c) shows the result of word segmentation.

The main problem is that we always segment punctuations into a word, causing failures in recognition. By comparing the height of last character in a word, we can locate commas and periods, but there is still no good way to tell other punctuations from English characters. We will eliminate commas, periods and other large noises when we actually cut the image into small image for OCR.



(a) Characters Using MSER

(b) Characters Using Image Projection



(c) Words Using Image Projection

Figure 8: Result of Segmentation

2.5 Optical Character Recognition

For OCR part, our goal is to recognize the words. Since word is a sequence, not a single character, we have to consider about relations between characters in a single word. Thus, we use a **CRNN** network, which is a combination of **CNN** and **RNN**: **CNN** will extract the features of word image, then **RNN** will learn about the word sequence.

After we train our **CRNN** model, we find that the model can't recognize single character word very well, for example, **I** and **a**. We deduce the reason is the word dataset we use to train has no single character word. So in addition, we train another single **CNN** to deal with this situation.

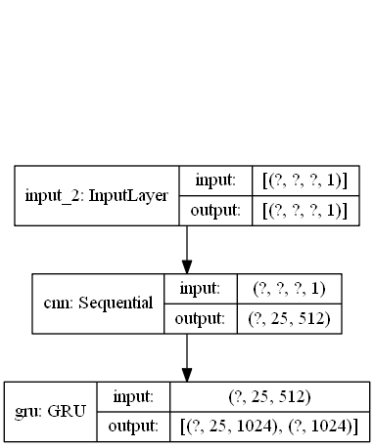
2.5.1 CRNN for word recognition

The **CRNN** model will have 2 parts: **Encoder** and **Decoder**.

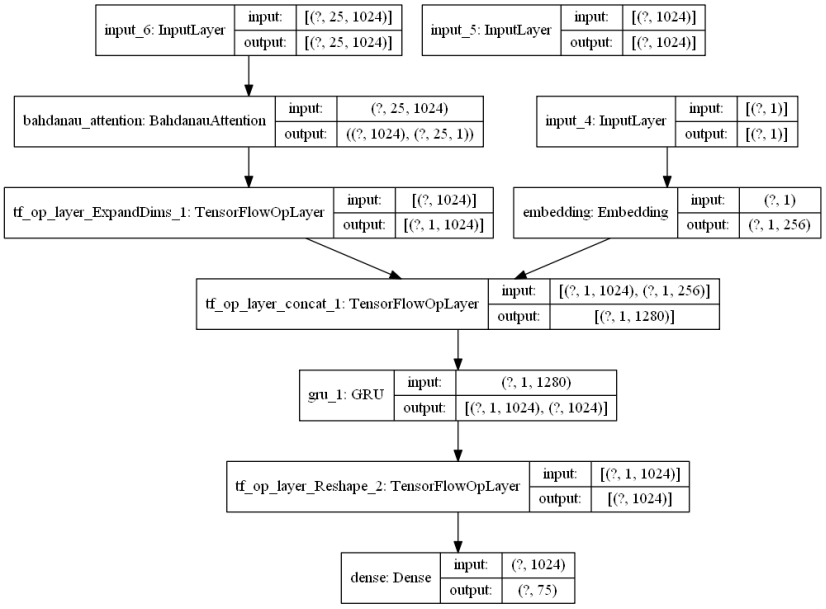
Encoder is a pure **CRNN** structure. The input image will first go through **CNN**, then **RNN**. The **CNN** is a 7-layer convolution network, and **RNN** we use the **GRU** model implement in keras.

Decoder will use **Attention** module to process the output of **Encoder**, then combine with the label of last time, and put into a **GRU** again. this **GRU** will output the predicted label of current time.

Here's detail structure of **Encoder** and **Decoder** (plot by keras tools):



(a) Encoder



(b) Decoder

Figure 9: CRNN structure

The number of parameters for this **CRNN** is about 2 million.

For **CRNN** code, we mainly refer to this github repository: `https://github.com/koibiki/CRNN-ATTENTION`.

2.5.2 CNN for single character word recognition

Single character word is just one character so we can just train a simple **CNN** to recognize it. We use 6-layer convolution network and it's enough to achieve a satisfying accuracy for single character recognition.

The number of parameters for this **CNN** is about 1 million. Large part of parameters are from FC-layers and convolution layers have only about 50 thousand parameters, So it's a much simpler model than the **CRNN** we use.

For simple **CNN** code, we mainly refer to this github repository: `https://github.com/WasifArmanHaque/english-character-recognizer`.

3 Experiments

3.1 Data

For image processing:

We use pictures taken by cameras of documents as input. The document in the picture should have clear border. We can also take special types of documents such as receipts, dollar bills, handwritten notes and others as input and produce a highly readable image, but it would be very hard to do character recolonization on these types of inputs.

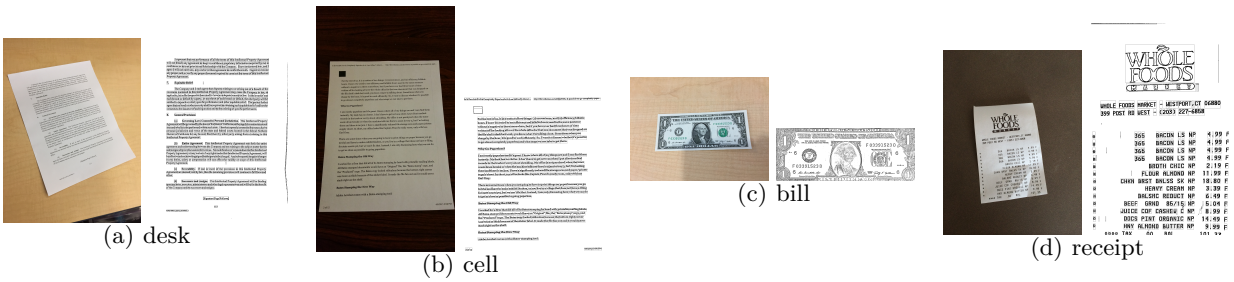
For OCR:

We use MJSynth dataset to train our **CRNN**. This dataset has 9 million images, covering 90 thousand English words. The dataset size is about 10G. The dataset link: `http://www.robots.ox.ac.uk/~vgg/data/text/`.

We use Chars74k dataset to train our simple **CNN**. We use the computer fonts dataset in it. This dataset has 60 thousand images, covering all 62 characters and digits. The dataset size is about 50M. The dataset link: `http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/#download`.

3.2 Results

Image processing: Figure 10 shows the results of image processing and segmentation. A full view of results can be found in folder `data&result/result/image_process_results`. In regular text-based document, we can do a decent job on both enhancement and segmentation. However, on non-regular types of document such like dollar bill or receipts, we can only get the scanned results without a satisfying segmentation. It is because the limitation of the traditional methods we choose.



(a) desk (b) cell (c) bill (d) receipt

Figure 10: Results of Image Processing

OCR training:

- For **CRNN**, we train about 5 days on a google platform compute engine machine, with a Nvidia Tesla T4 equipped on it. We train 12 epochs, which means train the whole dataset for 12 times. The model approximately converges on 10th epoch, and the final accuracy is about 92%.
- For simple **CNN**, we train about 1 hour on a google colab notebook, also with a Nvidia Tesla T4 equipped on it. We train 50 epochs, and the final accuracy is about 91%.

OCR testing: We test OCR on the segmentation results of picture 'desk' and 'cell' in Figure 10. The segmented words and predictions can be found in folder `data&result/result`. For picture 'desk', our prediction reaches 83.8% of accuracy. For picture 'cell', our prediction reaches 87.8% of accuracy. In the text result, we can see that we made most mistakes on mis-predict letter 'e' to 'c', 'f' to 't' or mis-predict words with punctuations. Based on the quality of the image and similarity between those characters, these mistakes are explainable and do not affect human reading severely in most situations. In a word, we found

our results acceptable based on the methods we use, but there are still a lot of improvements that can be done.

4 Conclusion

We have discussed the why we choose such approaches to the project and how they works in details. We also showed and analyzed the results we got with our approaches.In this section, we will discuss the strength and weakness of our system.

Strength:

- Our system goes beyond a traditional scanner with our OCR function. Being able to get selectable text from an image is useful for various work scenes.
- Dispite the training progress of OCR, our system is quick enough to get the result. With a trained network, it only takes seconds to get the processed image in a very high resolution and about 2 minutes to get a prediction of hundreds of words in the document.
- Our system is of high scalability. The different functions of the system is modular and can be replaced or improved individually with little effect on other parts.
- Our system is portable to different platform with some changes. System can be run in different computer systems, and with a trained network it can be transplant to a mobile device.

Weakness: :

- Our system has some strict demands on input data. The document must have clear borders to be successfully processed.
- The segmentation function still needs improvements. Current method can only take documents of specific patterns to get a good segmentation, and still make mistakes under many special circumstances.
- Words segmentation results are not as good as characters for OCR. Accuracy of word recognition is lower than character recognition.
- OCR accuracy can be improved by training with data that fits our results. Current training data is not perfect for our system.

5 Future Work

Even though our system is currently functional, it is not perfect and a lot of work can be done to improve it. We have some proposals of future work:

- Improve contour algorithm so we can take more kinds of pictures as inputs.
- Use deep learning method for segmentation. In this way we can segment characters with higher accuracy.
- Use word dataset contains both single characters and words so we do not have to use a second model for OCR.
- Use some more complex **CNN** to be part of our **RCNN** model, like **VGG19** or **ResNet34**. Maybe we can have a better result but there's also a risk of overfitting since the dataset images are small.
- Transplant the system to mobile device so we can do scan and OCR right after we take a picture with our smart-phone.