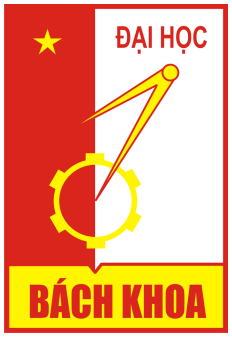

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

IT4343E - 131597 - Computer Vision

OCR for Document Processing

Group 2:

Hoang Nguyen Minh Nhat - 20194445

Pham Thanh Truong - 20194460

Pham Thanh Hung - 20194437

Tran Quoc Lap - 20194443

HANOI 2022

Contents

Chapter 1

Introduction

For many years, document processing has been a concerned topic of many areas such as communication, banking, administration and retail. Among researched methodologies, OCR is one of the most effective approach to solve the problem of document processing. OCR is the conversion of images of handwritten or printed text into electronic text data. OCR makes it more easy and more efficient for important document to be stored, searched, modified, and extracted.

However, large-scale document processing using ready-made OCR solutions is neither cheap nor productive. This is due to the diversity of organizations' specific demand and various forms of document to be processed. OCR itself is a complicated domain because it consists of many sub-domains and neither of them can be seen as a completely solved problem. In order to build a complete and efficient OCR solution, engineers are required to have a broad overview in the field and an accomplished programming skills in image processing and computer vision.

With all those in mind, we decided to develop a simple OCR solution for document processing. We hope that, by working on various techniques required by this project, we will improve our programming skills. We also hope that our final product might come in handy for people in time of need.

For the sake of convenient collaboration and development, we organize our OCR solution in a pipeline. The whole document process is broken into 3 stages: Pre-processing, Text Detection, and Text Recognition. Each of these stages consist of many other steps or options. Particularly, with Pre-processing stage, a corner detector is initialized to extract the paper area from the background, then the cropped image is deblur, cleaned, and deskewed. The processed image is then passed to the Text Detection stage where the text-line areas is determined. Finally, each text-line area is passed to the Text Recognition stage to build a JSON file.

It is noticed that deep learning has been rapidly developed, and indeed it has been beating many traditional image processing techniques in term of accuracy in solving OCR problems. Therefore, it's tempting for lively students like us to follow a models-training-and-comparison procedure in this course project. However, we prefer a different direction. We believe that traditional image processing techniques,

which is often under-estimated due to the emerge of deep learning, is important in the learning path. That means, we will have to compromise between the OCR accuracy and our learning goal, and sometimes we will use a pre-trained model for completeness.

The structure of the report is organized as follow. Chapter 1 introduces an overview of OCR for document processing, our expectation in this course project, and the report structure. Chapter 2 describes the details of the pipeline and techniques used our final software application. Chapter 3 presents theoretical foundations of the techniques introduced in Chapter 2. Chapter 4 summarize the experiment result and our findings.

The repository of our project can be found here, including installation and usage guidance.

Chapter 2

Program pipeline

Our program is organized into stages as described in Figure 2.1.

The system takes an image captured by the camera as input and returns a JSON file containing the location and content of the text information in the image. The input image will be processed through 3 main steps: Pre-processing, Text Detection, and Text Recognition. Details of each step are as follows.

2.1 Pre-processing

In most problems involving image processing and computer vision, pre-processing is an important step. It will help improve the image quality, so that the subsequent steps are more effective.

For our problem, the input is a document image taken from a camera, which is affected by factors such as environmental conditions, photography skills, and paper quality. Bad conditions such as low-light, noise, blur, rotation, patterned paper, or paper placed on an inappropriate background,... often negatively affect the subsequent processing steps, which may lead to incorrect or unrecognizable text.

After careful consideration, we decided to perform the following 4 pre-processing steps with the input image:

* Identify the 4 corners of the paper, then crop and straighten the page area. * Estimate the level of blur. * Clean the noisy components. * Rotate the text to natural view.

The important thing here is that, in each of the above 4 steps, we will automatically calculate the parameters for the image pre-processing functions and present them to the user. If they are satisfied with the proposal, that's a good sign. But if they are not satisfied, they can completely adjust it manually.

2.1.1 Corners detection and image alignment

Detection corners is the first step in our image pre-processing. The purpose of this step is to remove the background behind the paper, which can inadvertently be

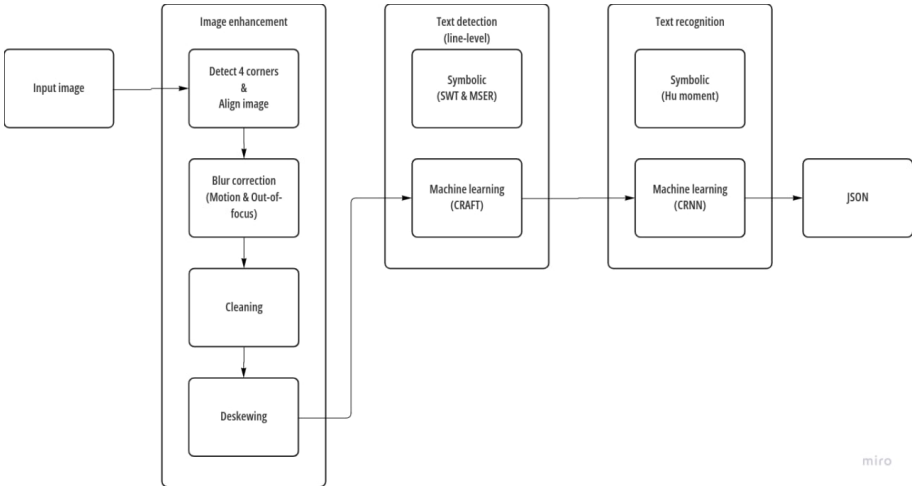

Figure 2.1: Our designed system pipeline.

identified as containing text in many cases.

Our method is as follows. First use the Median blur filter with a large size in order to spread the main colors in the image. Then, the K-means algorithm with K=2 is applied to segment the image into two different categories: the paper and the background. The K-means algorithm with K=2 will act as a binarizer regardless of the color of the background and the paper, as long as the background is relatively the same color and has a different color from the paper. After obtaining the binary image, we use the connected component finding algorithm to determine the pixel region corresponding to the paper and approximate it by a convex quadrilateral. Thanks to this, the four corners of the paper are determined. An example of this process can be found in Figure 2.2.

We consider the case where the paper is distorted because the image can be taken at different angles. With the 4 corners of the paper determined in the previous step, we use the Perspective transform algorithm to align it to a straight line.

2.1.2 Blur correction

Dealing with blurred photos is an interesting job. Because during the shooting process, the user can shake his hand or press the shutter while the camera has not finished focusing. Here, the Fourier transform can be used to recover the blurred image. However, this is difficult because it requires knowing the filter in advance - which can almost only be done by the sensors when the image is captured by the camera. A less expensive solution is to ask the user to take a sharper image. But in case the user can't, we still provide a simple interface where the user can try to recover the motion blur image by manually adjusting the parameters.

To determine if an image is blurred, we use the Fourier transform. And to sharpen the image, we use the Wiener filter (see Figure 2.3).

Figure 2.2: Result of corner detection

Figure 2.3: Image before and after motion deblur.

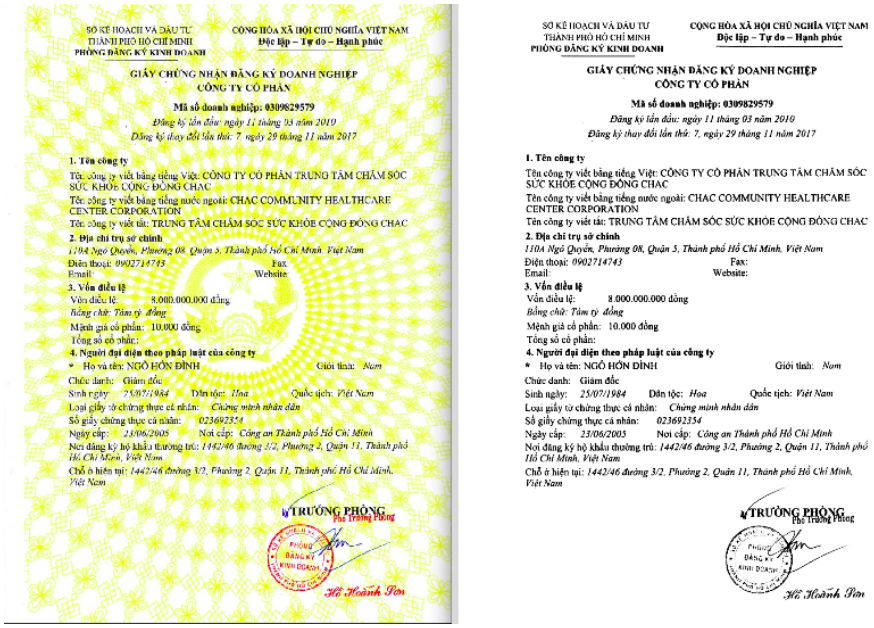

Figure 2.4: Image before and after cleaning.

2.1.3 Cleaning

For many types of documents such as identity cards, vehicle registration, passports, etc., usually the letters are printed on the background with patterns. Some patterns can confuse the Text Detection and Text Recognition steps, for example circles can be thought of as an o, or transparent watermarks in the form of text that are not our concern might also be false positive. Therefore, image cleaning is a necessary process.

The method of image cleaning comes from our observation: normally, the Value channel in the HSV color space of letters are smaller than the Value channel value of the background. That means, the text pixels are likely to sit in the left half of the histogram, and the non-text pixels are likely to sit in the right half of the histogram. The problem now lies in how to find a mechanism to push all non-text pixels to white, and all text pixels to black. To do that, we analyze the histogram of the image and then automatically estimate the appropriate datum. And the results are relatively good (See Figure 2.4).

2.1.4 Deskewing

In some cases, the text in the paper may be not written or printed horizontally, and the user may want to rotate the text for the purpose of document reconstruction.

There are several methods that can be used to determine the rotation of the text even if the text-area is not specified. Hough Transform combined with K-means algorithm can be used in this case. Another method is to use the Fourier Transform, and that is the method we use in our system (Figure 2.5).

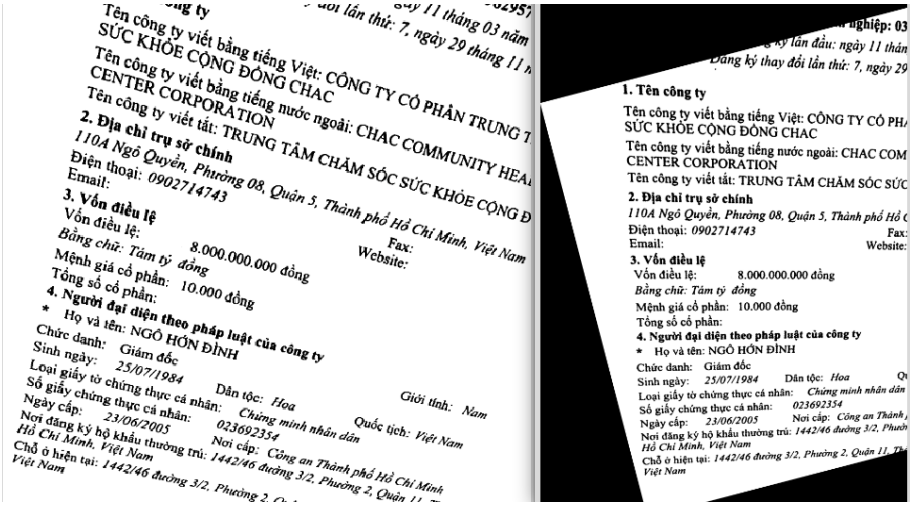

Figure 2.5: Image before and after deskewing.

2.2 Text Detection

The next step after image preprocessing is to identify the areas that contain text in the image. The output of this stage is to return the coordinates of the areas containing the text, which will then be fetched into the Recognition stage. These areas can be character-level, word-level, or line-level. In different aspect, it can be in the form of a bounding box or in the form of pixel-level segmentation. Which coordinate configuration to choose highly depends on the input requirements of the Text Recognition stage. The Text Recognition stage in the system we designed requires a line-level text area, so the output of the Detection stage will follow this configuration.

2.2.1 Using machine learning method

Recently, neural nets have been increasingly used to detect words in image. Many architectures has been proposed such as CRAFT, CTPN, EAST, FOTS, PixelLink, PSENet. After a thorough searching, we notice that CRAFT yields the best performance among all. It detects correctly both horizontal and multi-oriented text areas. CTPN, EAST, FOTS, PixelLink can have detected text areas that overlap each other. Some non-text areas such as a portion of logos or barcodes is detected as a text area by CTPN, EAST, FOTS, PixelLink and PSENet.

2.2.2 Using symbolic method

However, if the documents contain many words and lines, then the network data can be quite slow if it is run on a CPU. That impression, in addition to our desire to learn new image processing techniques, leads us into search of a text detector before deep learning era. And that leads us to Stroke Width Transform proposed in 2010.

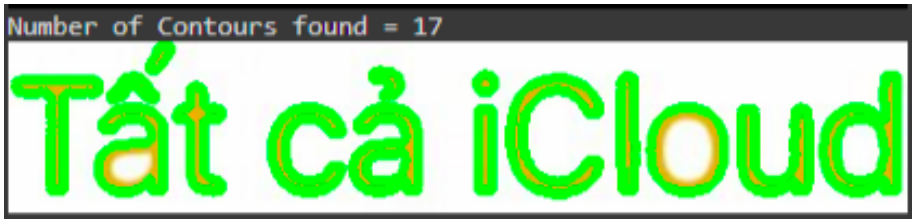

Figure 2.6: Text recognition result using shape matching. The predicted text is *TdA8lOd2TIOlOUd*

In general, the results show that CRAFT is the most competent to most of the cases than other methods in both speed and quality. Therefore, in our system, we fix with CRAFT as the final text detector. But notice that the original CRAFT has a post-processing step to convert from density map into bounding boxes or polygon. We modify this post-processing step a little bit, so that the output is the text area contours. It is because, with document containing handwritten text like birth certification, the previous post-processing produce bounding boxes that merges 2 lines very often. Handwritten character is often much bigger than printed text and consecutive lines might be stick together. That leads to unexpected behaviour of Recognition stage.

2.3 Text recognition

The last step in our system is Text Recognition. It take the text area of the previous stage as input and outputs a JSON file containing the coordinates and contents of the corresponding text.

2.3.1 Using symbolic method

Just like the previous two stages, our first attempt is to use traditional image processing techniques. However, with this approach, we cannot operate on the line-level, but can perform only on character-level.

In general, the result we achieve is not good (Figure 2.6).

After converting the image to gray Image, we find contours for all the appeared characters, and contemporarily leverage those contours to calculate the corresponding inner area. The result is stored in a dictionary whose key is String of character and value is np.array value of that character.

We create a directory which can be matched with contour data of the image of each character. For example, the image of the character “A” would be stored in the letter “A” in our directory.

Compare the Predicted result to True result, we can see the accuracy is about 20%-30%. It can recognize some of the popular characters like in English, for instance: "T", "A", "I”, "O", "U", "D". However, it is challenging to distinguish

between Uppercase and Lower Cases of some letters such as “o”, ”c”, “u”, “l”. Vietnamese accents are also the barriers for the traditional recognition. For example, the dot character “.” can be ambiguously mistaken among various cases: the dot in the letter “i”, dot in Vietnamese low constricted accent, and the “ending sentence” dot. There are also more concerned situations which are easily confused by the system and thus take the results into inaccuracies.

2.3.2 Using machine learning method

Mostly, the best results in text recognition currently come from deep learning models. In this project, we experiment with two models: CRNN using traditional CTC Loss and Transformer OCR.

CRNN with CTC loss is a widely used model for text and speech recognition. At the time of writing, we have not been able to bring this model to operate our program, because... At the beginning, we succeeded in training it on the handwritten Vietnamese address data set provided in here, with Word Error Rate reached 0.2544 (Details see this notebook). We were planning to extend to training with printed text to unify handwritten text and printed text in a same model. However, the number of images in the printed-text dataset is much larger than the handwritten-text dataset. Besides, we find that the model is quite slow due to the use of VGG backbone with more than 22 million parameters and the use of LSTM network also limits the parallel computation of the model. Therefore, we decided to study other architectures to replace the VGG backbone (see Chapter 3) in order to decrease the training time. However, for some reason it is not working and we are still debugging it.

Therefore, to complete a product, we are temporarily using the pre-trained Transformer OCR model from VietOCR for our Text recognition stage.

Chapter 3

Theoretical foundation

In this chapter, we will present the theoretical background of the techniques used in our system. That is the knowledge that has not been taught in the IT4343E - Computer Vision class. Among them, there are techniques that we have not yet implemented in our system, such as CRNN model with CTC Loss.

3.1 Deconvolution

In this section, we will present the image deblur technique used in our system. However, before we dive straight into the it, we need to recall some properties of the Fourier Transform.

There is a very important relationship between convolution and the Fourier transform, referred to as the convolution theorem (Table 3.1).

It turns out, this convolution theorem can be effectively used to deblur an image, whether it is the result of a motion blur or out-of-focus blur. For example with motion blur, we might be holding our camera, and when we press there's a little bit of shake. And what happens is that our ideal image is being convolved by a motion blur function (Figure 3.1). The problem is that, given a captured image g, and the **point spread function** h, we need to find f.

Finding f can easily be solved by applying the convolution theorem if we know h. One way have it is right in our sensor in our camera called *inertial measurement*

Table 3.1: Convolution Theorem

Spatial domain	**Frequency domain**
$g(x) = f(x) * h(x)$ **convolution**	$G(u) = F(u)H(u)$ **multiplication**
$g(x) = f(x)h(x)$ **multiplication**	$G(u) = F(u) * H(u)$ **convolution**

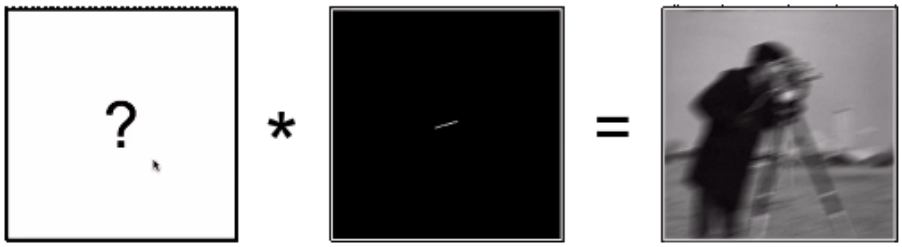

Figure 3.1: Camera shake and deconvolution. From left to right: $f(x, y)$, $h(x, y)$, $g(x, y)$ where $f(x, y) * h(x, y) = g(x, y)$.

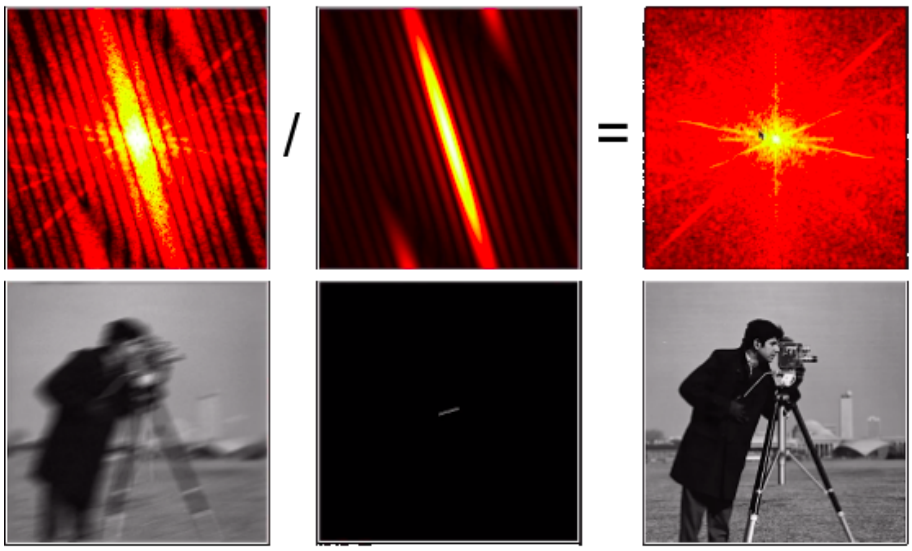

Figure 3.2: Deconvolution for blur correction

unit which will estimate the motion of the camera and use that to create a point spread function.

So the procedure is as the following: Given h and g in spatial domain, we apply Fourier transform to get H and G in frequency domain. Then we compute $F(u, v) = \frac{G(u, v)}{H(u, v)}$ and apply Inverse Fourier Transform to get f (Figure 3.2).

However, we have a problem, which is noise. Very often when we capture an image, noise might be captured for some reasons. And if we take a naive inverse Fourier transform, we get an image that is overwhelmed by the noise (Figure 3.3).

The reason is that, because the image is blurred, so the Fourier transform of h, which is H, is a low-pass filter in some sense, which mean H is nearly 0 for high frequency. At the same time, we know that noise produce many rapid changes in intensity in an image, so G is very high for high frequency. Therefore, the noise is amplified substantially, which produces the final result as in Figure 3.3.

So we need a better filter called Wiener filter: $F(u, v) = \frac{G(u, v)}{H(u, v)}[\frac{1}{1 + \frac{NSR(u,v)}{|H(u,v)|^2}}]$ where NSR is a noise-to-signal ratio, which is unknown to us, but we can choose a

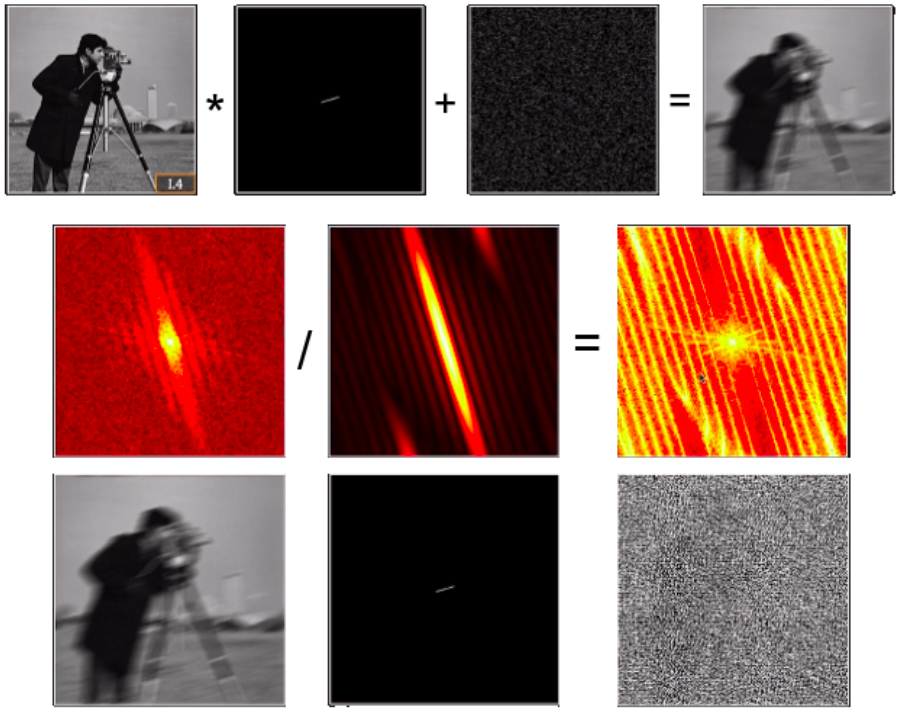

Figure 3.3: Naive deconvolution with noise. From left to right (uppermost figure): $f(x, y)$, $h(x, y)$, $\eta(x, y)$ for noise, $g(x, y)$.

value around 0.002. The result of recovering image using Wiener filter is shown in the Figure 3.4.

3.2 Cleaning

This part describes our algorithm to clean noise in document image. It's notable that our cleaning algorithm take output from corner detection step, so that we can pay full attention to the paper only.

As mentioned in Chapter 2, we observe that the text pixels are likely to sit in the left half of the histogram, and the non-text pixels are likely to sit in the right half of the histogram. The problem now lies in how to find a mechanism to push all

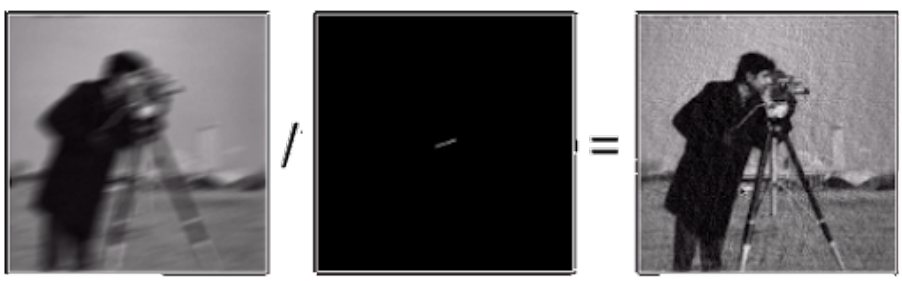

Figure 3.4: Deconvolution with noise using Wiener filter.

non-text pixels to white, and all text pixels to black (Figure 3.5).

We also observe that, for most document images that we investigate, the histogram share a common trend: the area of lower intensity values, which is corresponding to the text pixel, looks like a flat region while the area of higher intensity value, which is corresponding to the non-text pixel, looks like a mountain and has much bigger cardinality. With that in mind, we can first determine a suitable α, after which we will shift the histogram by the same amount to the left. Then, we determine β and multiply the image with it. After multiplying, the histogram is stretched so that all the pixel larger than 255 will turn white. The formula is as follow:

$$\text{image} := \text{CLIP}((\text{image} - \alpha) * \beta, 0, 255)$$

The problem is how to automatically choose suitable α and β? From our experiment, the procedure is as following:

```
val, cnt = np.unique(img, return_counts=True)
med = cnt[np.argmin(np.abs(cnt - int(np.median(cnt))))]
alpha = np.max(val[np.where(cnt == med)])

thresh = (255 * 255 + 1.2 * alpha * alpha - 255 * alpha) /
    (255 - (1 - 1.2) * alpha) - alpha
beta = 255 / thresh
```

Our algorithm it does quite well in most cases (Figure 3.6), except for some images of bad lighting variance condition. In the latter case, the user can manually change β.

3.3 Stroke Width Transform

Stroke Width Transform is not a deep learning text detector. It is based on an observation: One feature that separates text from other elements of a scene is its nearly constant stroke width. A stroke is a contiguous part of an image that forms a band of a nearly constant width (Figure 3.7).

In order to recover strokes, first Canny edge detector is applied. For each edge pixel p, we traverse along the direction of its gradient d_p until another edge pixel q is found. If d_q is roughly opposite to d_p, we assign the width $||\overrightarrow{p-q}||$ to all pixel along the traversed path

At this point, each pixel contains the width of the most likely stroke it belongs to. The next step of the algorithm is to group these pixels into letter candidates: Two neighboring pixels may be grouped together into connected component if they have similar stroke width. And if a connected component has stroke width variance too big, it will be rejected.

This algorithm has shown good results for multi-language text images or images with varying fonts style.

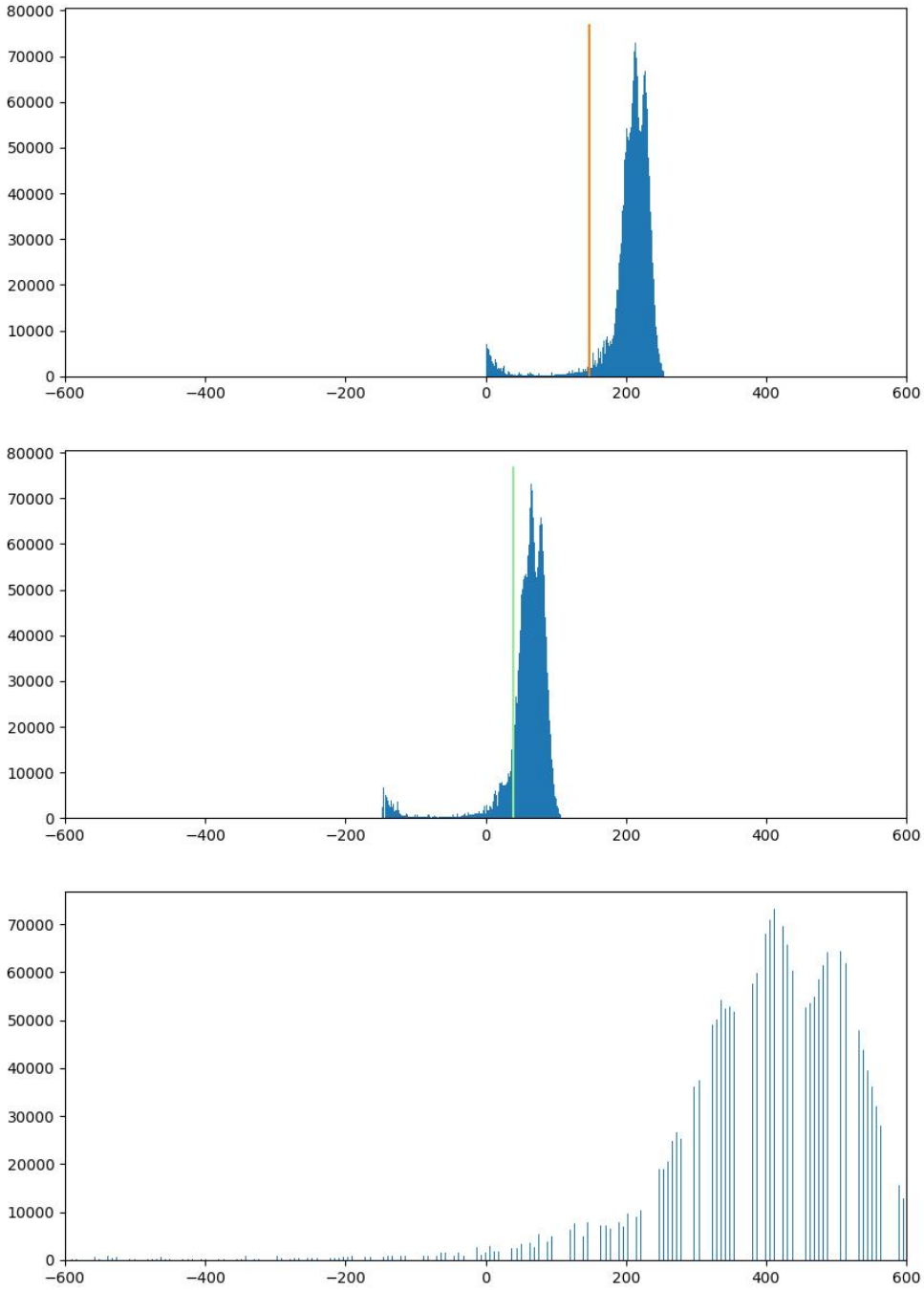

Figure 3.5: Desired histogram shifting and stretching. a) Original histogram and estimated α. b) Shifted histogram and estimated threshold to be discarded to the right. c) Stretched histogram before clipping.

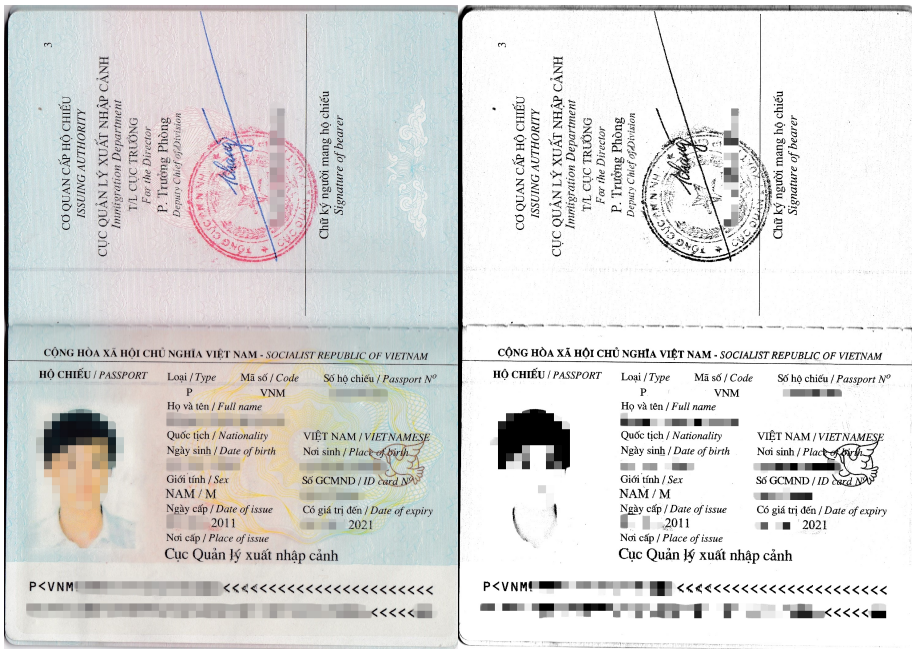

Figure 3.6: Image after automatic cleaning.

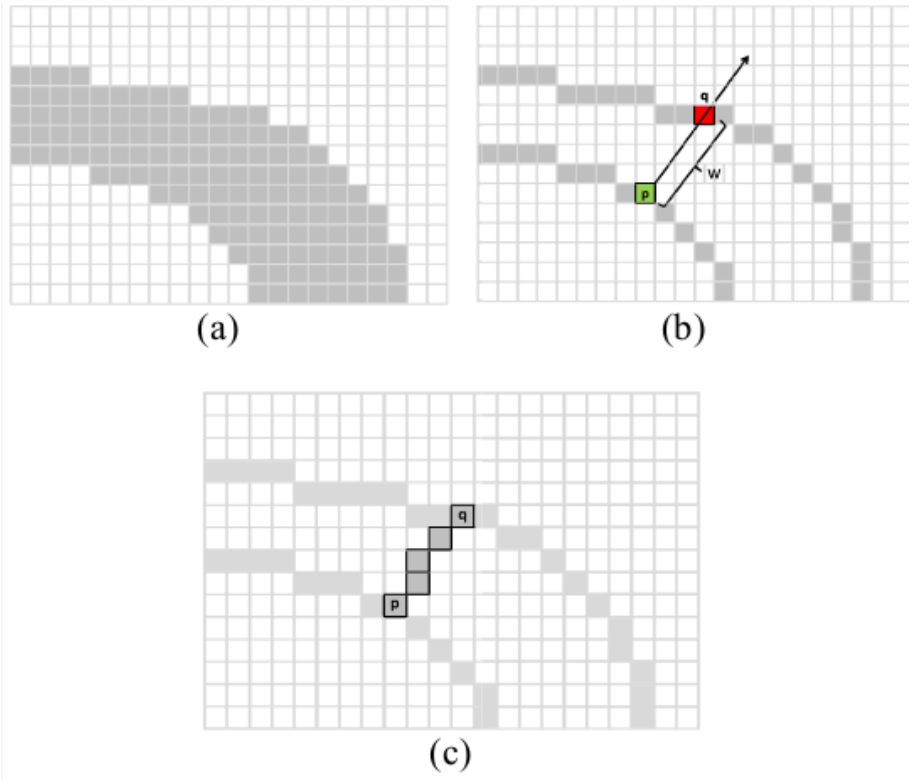

Figure 3.7: (a) A typical stroke. (b) p is a an stroke edge pixel. Searching in the direction of the gradient, leads to finding another stroke edge pixel q. (c) Each pixel along the ray is assigned by y the minimum of its current value and the found width of the stroke.

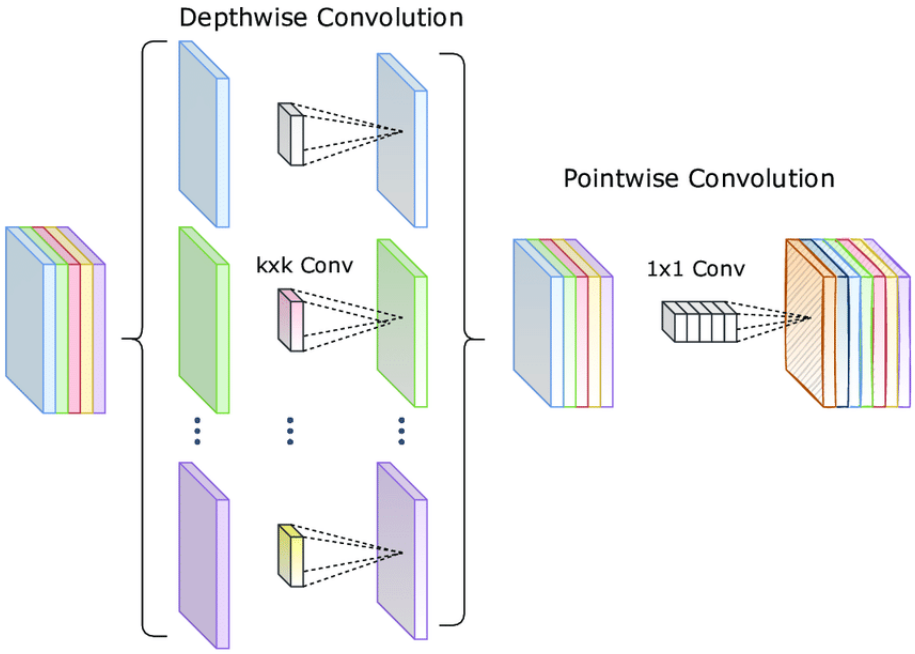

Figure 3.8: Depthwise separable convolutions

3.4 MobileNetV3 building blocks for CRNN backbone

MobileNet, since its first release with v1, has been well known for its speed and performance. It can run very well on CPUs due to careful research of its building blocks. It has less parameters, less number of operations than other architectures like ResNet or VGGNet, but maintain comparable accuracy. That's the reason why we want to change our CRNN backbone from VGG to MobileNet. Particularly, we select MobileNetV3 in order to inherit latest advances in deep learning.

MobileNetV1 introduces depthwise separable convolutions. Depthwise separable convolutions alter traditional convolution by defining 2 separate layers: depthwise convolution and 1x1 pointwise convolutions (Figure 3.8).

In general, we have the formular $\frac{\text{cost}_{\text{depwise separable}}}{\text{cost}_{\text{normal}}} = \frac{1}{n_{c'}} + \frac{1}{f^2}$ where c' is the number of output channels and f is the kernel size.

MobileNetV2 introduces the linear bottleneck and inverted residual structure. This structure is defined by a 1x1 expansion layers followed by depthwise convolutions and a 1x1 projection layer. The input and output have skip connection if they have the same number of channels (Figure 3.9).

MobileNetV3 inherits building block by MobileNetV2, in combination with squeeze and excitation from SENet, and use a modified version of Swish activation function instead of RELU. The squeeze and excitation serve as a lightweight attention modules. The idea is not to treat each of channels equally when creating the output feature maps. Instead, a single parameter to each channel is learnt so that it adds

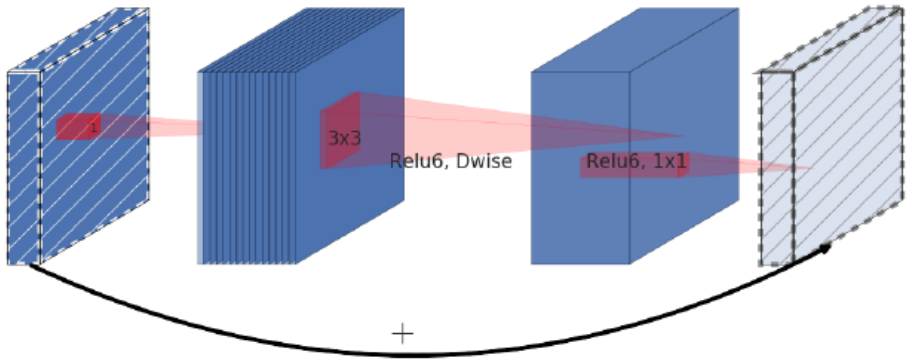

Figure 3.9: MobileNetV2 building block

a content aware mechanism to each channel. By replacing RELU with hard Swish, MobileNetV3 manages to reduce the initial set of filter from 32 filters in full 3×3 convolution to 16 filters.

In our project, we design our CRNN backbone based on MobileNetV3 blocks, except that our choice of activation function is Mish rather than Swish. The reason is that Mish is introduced after Swish, and has outperforms RELU and Swish across all the standard architectures. Unfortunately, we haven't fixing our model till the time writing yet.

Chapter 4

Result and Conclusion

Here, we demonstrate the outputs of some real-world images. For the current situation, we cannot do much in Text Recognition stage because if is totally dependent in the pre-trained model. The Transformer OCR is a language model, which is not uncommon to see the model produce text for non-text area.

Through this project, we have made a relatively complete product. Although, it still hasn't lived up to our original expectations, as there are still many parts we have not completed such as training the CRNN model or improving the speed of the system. We'll put them in the future work. We hope that in the near future, our product will be more reliable in both accuracy and speed so that it can be useful to everyone.

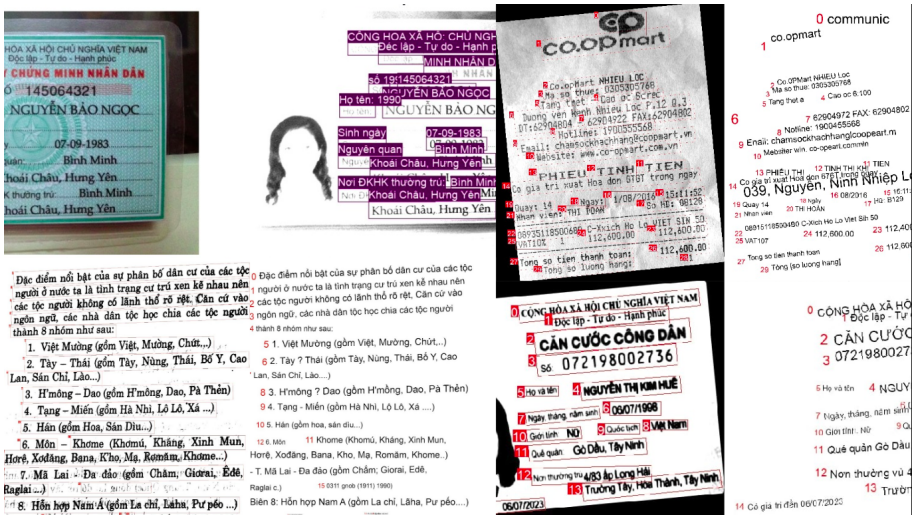

Figure 4.1: Out of some images by our system.