EN.600.461 Computer Vision
Final Project
Recognizing and Translating Text from Document Images

Joon Hyuck (James) Choi
Senior Undergraduate
The Johns Hopkins University
3400 N Charles Street
Baltimore, MD 21218, USA
jchoi100@jhu.edu

Abstract

We explore optical character recognition (OCR) in photos of typed and handwritten documents. We first explore basic preprocessing of photos of documents using `OpenCV` [1] *for blurring, thresholding, and denoising. We then discuss the use of* `tesseract-ocr` [2] *to perform OCR. We discuss the incorporation of the Google Cloud Translation API* [3] *to translate the OCR results into different languages. We finally discuss template matching using signatures of three U.S. presidents and the Johns Hopkins University (JHU) logo on document images.*

1 Introduction

This work achieved the following main goals stated in the original proposal: 1) Given a photo of a document, convert it into a clean scanned version; 2) Take the scanned version and perform OCR. Optionally, this work took the OCR output and translated the text into different languages using the Google Cloud Translation API. This work also experimented with template matching using signatures of three U.S presidents and the JHU logo in input documents. Lastly, this work performed CNN training on the MNIST handwritten digit dataset as an experiment. In this document, we discsuss 2. Dataset, 3. Methods, 4. Results, 5. Discussion, and 6. Conclusion.

[1] http://opencv.org/
[2] https://github.com/tesseract-ocr
[3] https://cloud.google.com/translate/

2 Dataset

2.1 Typed Documents

We took photos of the final project guidelines for this course, the author's final project proposal, JHU's Lav Note, and four aesop's fables, which the author typed and printed in different font types and sizes. A sample input image is shown in Figure 2.

2.2 Handwritten Documents

We collected photos of letters written by three U.S. presidents: Barack Obama, George W. Bush, and Bill Clinton. We also wrote portions of the aesop's fables mentioned above by hand with varying neatness and photographed them under different settings. Figure 3 shows one of our reasonably neat images `hand_cat.png`, and Figure 4 shows a sample letter written by Preisdent Obama, `obama2.png`.

2.3 Templates

For the template mathcing experiments, we collected images of the JHU logo in several sizes and signatures of the three U.S. presidents listed above. We used images from **2.1** and **2.2** as input images.

3 Methods

In this section, we discuss the methods we took and external libraries used for each stage of our work.

3.1 Image Preprocessing

We implemented our program in Python using `OpenCV`. In order to feed the OCR algorithm clean input to achieve best performance, we used `cv2.medianBlur` to smoothen the input image

Figure 1: Sample images from MNIST

with an aperture size of 5. Then, we performed binary thresholding on the smoothened image using `cv2.adaptiveThreshold` with *adaptiveMethod= ADAPTIVE_THRESH_GAUSSIAN_C*, *threshold = binary*, *blockSize=5x5*, and *C=2*. The adaptive method we selected uses the weighted sum of (*(blockSize* x *blockSize) neighborhood of pixel (x, y)* $-$ *C*) as its threshold value.

Finally, we performed denoising on the thresholded image to remove noise and make the output clean. We used `cv2.fastNlMeansDenoising` with *templateWindowSize=7*, *searchWindowSize = 21*, and *h=7*. Parameters were chosen empirically.

3.2 OCR

In order to perform OCR on the preprocessed inputs, we used the `tesseract-ocr` library via its Python wrapper `pytesseract`. Raw output from `pytesseract`'s `image_to_string` method was used to pass into Python's file writer and Google Cloud Translation API.

As a side experiment, we used the `Keras` [4] library to train the MNIST [5] dataset of 70,000 handwritten digits on three different convolutional neural networks (CNN). We referred to code available online [6] to build the three CNNs. Refer to Figure 1 for some sample images in the MNIST dataset.

3.3 Text Translation

We made use of Google Cloud's Translation API in order to translate the OCR processed document into several different languages based on user command line input. We referred to external code [7] to make API calls and modified the details to fit our purpose. Moreover, we needed to carefully deal with the translated output characters in `utf-8` when writing them in .txt since they were not simple ASCII.

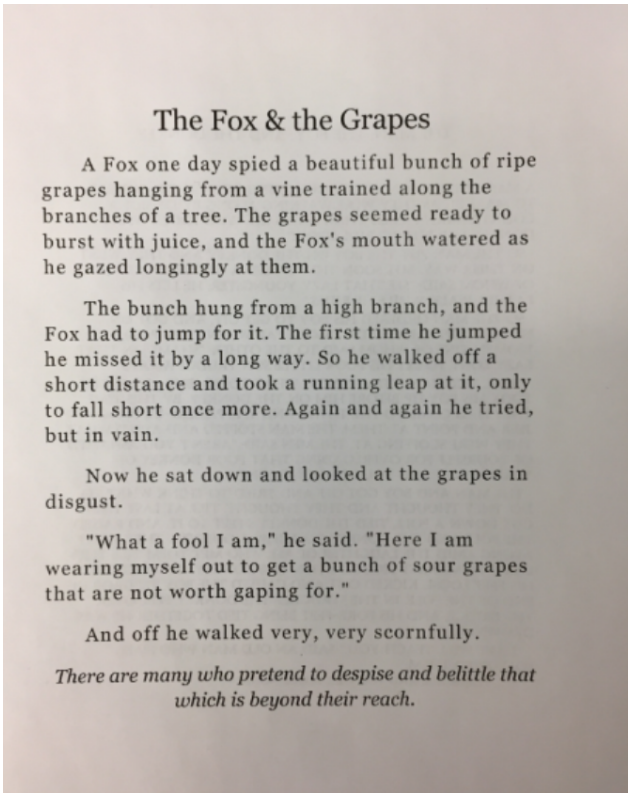

Figure 2: Sample typed document `aesop_fox.png`

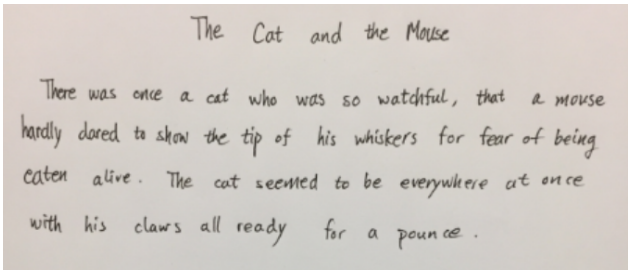

Figure 3: Reasonably handwritten doc `hand_cat.png`

3.4 Template Matching

Aside from OCR, which was the main focus of this project, we also experimented with template matching in images. We tested with various targets. First,

[4] https://keras.io/

[5] http://yann.lecun.com/exdb/mnist/

[6] http://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

[7] http://github.com/mouuff/mtranslate

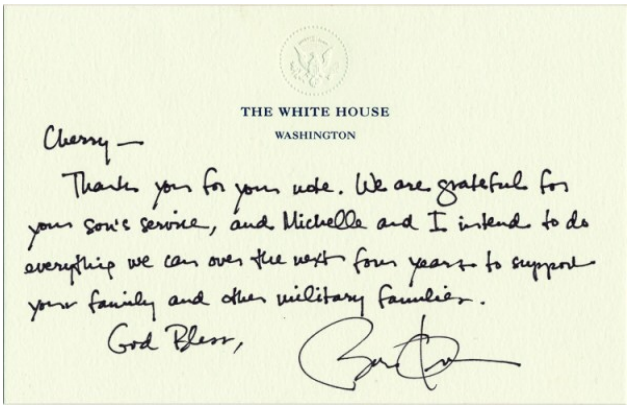

Figure 4: Less neatly handwritten doc `obama2.png`

we targeted to identify signatures of U.S. Presidents Barack Obama, George W. Bush, and Bill Clinton in documents. Our dataset consisted of handwritten letters written by the presidents as input images and their respective signature images as templates.

Next, we experimented with the JHU logo to determine whether an input document was an official JHU document. We simply assumed that official JHU documents contained a JHU logo for experimental purposes. We used `cv2.matchTemplate` with our input documents as the source image, multiple JHU logo images (each with different sizes since `cv2.matchTemplate` is scale sensitive) as templates, and matching method `cv2.TM_CCOEFF_NORMED`. If the normazlied `TM_CCOEFF` value came out to be greater than 0.51 for any of the input JHU logo templates, we determined that the input document contained a JHU logo in it. The matching method and determinant value of 0.51 were chosen empirically.

4 Results

Please see **Appendix A** for experiment run outputs on the three sample images `aesop_fox.png`, `hand_cat.png`, and `obama2.png`.

4.1 OCR on Typed Documents

We ran our code on seven document images with different font types and font sizes. The images were `cv_proj_description.png`, `cv_jchoi_proposal.png`, `aesop_fox.png`, `lav_notes1.png`, `aesop_miller.png`, `aesop_goose.png`, and `aesop_cat.png`. With the exception of `aesop_goose.png`, performance on all documents were nearly perfect. Potential reasons for failure with this particular image are discussed in **5.2**.

4.2 OCR on Handwritten Documents

We applied our code on photos of handwritten documents. Performance on very neatly written documents (`hand_fox.png` and `hand_goose.png`) was nowhere close to what we saw for the typed dataset. Any reasonably neatly handwritten documents (`hand_miller.png` and `hand_cat.png`) showed even poorer performance. Moreover, our code was not able to detect a single word from document images with much less neat handwriting. Such images were `obama1.png`, `obama2.png`, `obama3.png`, `bush1.png`, `bush2.png`, `clinton1.png`, and `clinton2.png`.

4.3 Template Matching

The results for the U.S. presidents' signatures was poor. The template matching algorithm could not detect our signature templates in the majority of cases. We also used the JHU logo to determine whether the input document was an official JHU document. This experiment showed more promising results. We discuss potential reasons in **5.3**.

5 Discussion

5.1 Image Processing

For our experiments, we tested with various configurations for denoising and smoothening. Certain parameter configurations worked perfectly for some images while they produced disappointing outcome for others. We chose parameter values that performed acceptably well on all images in our dataset. The motivation behind denoising in our work was that document scans have many random specs spread throughout the image. Such noise can greatly reduce OCR accuracy because an OCR algorithm may confuse a random spec with punctuation marks or associate a spec with an actual character nearby. (e.g. confuse an l and ‘ with an i)

5.2 OCR

The reason that `aesop_goose.png` did not produce satisfactory results was that the font in the

document was relatively thick. Therefore, after the preprocessing stage, the alphabet characters in the thresholded document were hollow with just the edges remaining. Therefore, the OCR algorithm could not recognize any of the characters in this text.

The two images `cv_jchoi_proposal.png` and `lav_notes1.png` were taken under suboptimal conditions: irregular lighting, slight rotation, and/or glossy surfaces. Moreover, each of these documents had mixed font types and sizes, and the structures of the documents were more complicated than the previous inputs. We ran our code on these two document images. Despite the less optimal settings, we correctly denoised and thresholded the images and obtained similar OCR results as before.

5.3 Template Matching

Performance for experiments with U.S. presidents' signatures was poor, and we decided to experiment with a fixed, rigid template, which is easier to match. Performance on the JHU logo template experiments was better compared to that of the U.S. president signature experiments. The reason that the signature experiment is harder is that the JHU logo is fixed in terms of how it is comprised (e.g. lines, curves, color density) whereas signatures vary each time it is signed (e.g. penstroke width, curvature, ratio of one part of the signature with respect to other parts).

Another reason that the U.S. president experiment did not work well could be that the input documents (*i.e.* letters written by the presidents) were handwritten. The template matching algorithm had more variability to deal with since the entire document contained many likely matching candidates for the given signature template. This was apparent when we saw the algorithm output locations of random words within the letters as matches to the signature templates.

6 Conclusion

We explored optical character recognition on photographed documents in various settings and formats. We tested with optimal and suboptimal photography settings, experimented with documents with various fonts, and compared performance on typed and handwritten documents. For typed documents, we saw accurate results under both optimal and suboptimal settings even if they contained various font sizes and font types. However, we were not able to achieve accurate results with handwritten document photos, disregarding the neatness of the handwriting. For extremely neatly handwritten documents documents (*i.e.* nearly equal neatness as typed documents), the OCR algorithm was able to produce correct output occasionally. However, even this was not often enough to be considered reliable.

Nonetheless, on typed documents, we were able to effectively preprocess the document image and perform OCR on the cleaned data. Moreover, we leveraged the Google Cloud Translation API to translate documents into several different languages. Our final experiments on template matching also deserve future development into a project of its own.

How to Run the Code

Please refer to our GitHub repository for the source code.[8] Assuming that the user has installed all required packages and dependencies, run the following command:

$> python driver.py {path-to-image-file} {list of target languages each separated by a space}

Sample usage: *python driver.py doc7.png es fr ko*

The sample command above performs OCR on `doc7.png`, outputs the thresholded image, the text in the original input language, and the text translated in Spanish, French, and Korean.

Saved Output from Experiments

Please refer to our GitHub repo for saved output.

Acknowledgments

We thank `OpenCV` and `tesseract`, the code we referred to for constructing CNNs using `Keras`, MNIST for the handwritten digit dataset, and Google Cloud for the Translation API.

We also thank Ji Won Shin (*JHU Class of 2019*) for generously providing the author with a Macbook Pro that could run `tesseract`.

And thanks to Professor Reiter and the TAs for a great semester! Have a great winter break.

[8] https://github.com/jchoi100/computer_vision_project