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

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

Abstract

This work explores optical character recognition (OCR) in photos of printed and handwritten documents. It first explores basic preprocessing of photos of documents using `OpenCV` [1] functionalities for blurring, thresholding, and denoising. It next discusses the use of `tesseract-ocr` [2] to perform OCR. We then discuss the incorporation of the Google Cloud Translation API [3] to translate the OCR results to different languages. If finally discusses template matching using the Johns Hopkins University (JHU) logo.

1 Introduction

This work achieved the following main goals stated in the original project 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 the JHU logo in order to determine whether the input document is an official JHU document or not.

2 Methods

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

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

2.1 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 with an aperture size of 5. Then, we passed the smoothened image through `cv2.adaptiveThreshold` with *adaptiveMethod*= `ADAPTIVE_THRESH_GAUSSIAN_C`, *threshold = binary*, *blockSize=5x5*, and *C=2*. The adaptive method we chose uses the weighted sum of the *blockSize* x *blockSize neighborhood of pixel (x, y) - C* as its threshold value. Parameter values were chosen empirically.

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*. Numbers here were also chosen empirically. The motivation behind denoising in our work was that document scans usually have many random specs here and there. 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 near by (e.g. confuse an l with an i).

2.2 OCR

In order to perform OCR on the cleaned inputs, we used the `tesseract-ocr` library through its python wrapper `pytesseract`. We used the raw output from `pytesseract`'s `image_to_string` method 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). Results will be discussed in section 3.3. We referred to code online [6] to build the three CNNs.

2.3 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 code posted online [7] and modified the details to fit our purpose.

2.4 Template Matching

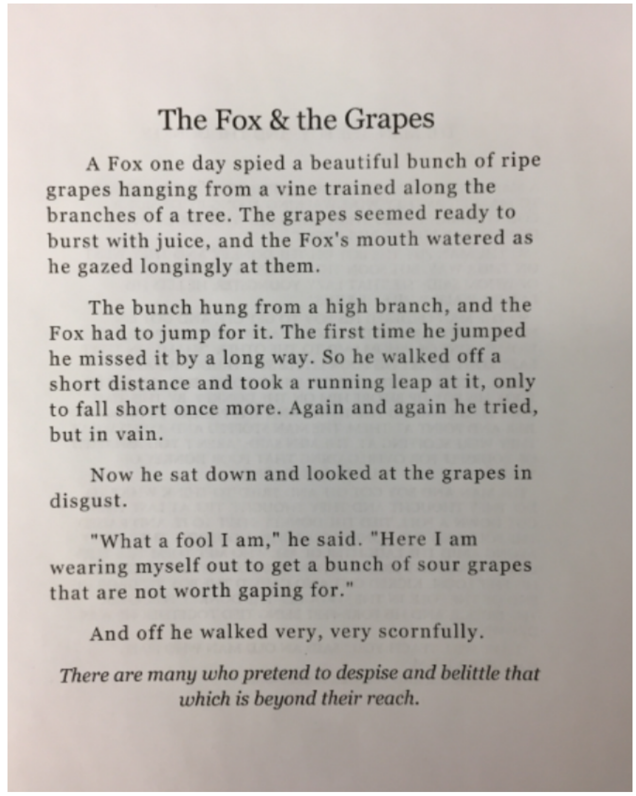
Aside from OCR, which was the main focus of this project, we experimented with template matching in images. Specifically, we used the JHU logo to determine whether the input document was an official JHU document or not. We naively assumed that official JHU documents contained a JHU logo for our experiment's sake. We used `cv2.matchTemplate` with our input document 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 of various sizes, we determined that the input document contained a JHU logo in it. The matching method and determinant value of 0.51 were chosen empirically.

3 Results

Here, we qualitatively discuss OCR results on typed and handwritten input documents.

3.1 Typed Documents

We ran our code on five document images with different font types and font sizes. The images were `cv_proj_description.png`,



The Fox & the Grapes

A Fox one day spied a beautiful bunch of ripe grapes hanging from a vine trained along the branches of a tree. The grapes seemed ready to burst with juice, and the Fox's mouth watered as he gazed longingly at them.

The bunch hung from a high branch, and the Fox had to jump for it. The first time he jumped he missed it by a long way. So he walked off a short distance and took a running leap at it, only to fall short once more. Again and again he tried, but in vain.

Now he sat down and looked at the grapes in disgust.

"What a fool I am," he said. "Here I am wearing myself out to get a bunch of sour grapes that are not worth gaping for."

And off he walked very, very scornfully.

There are many who pretend to despise and belittle that which is beyond their reach.

Figure 1: Sample typed document `aesop_fox.png`

`aesop_fox.png`, `aesop_miller.png`, `aesop_goose.png`, and `aesop_cat.png`.

With the exception of `aesop_goose.png`, performance on all documents were near-perfect. A sample image is shown in Figure 1. The reason that `aesop_goose.png` did not produce satisfactory results was that the font in the document was extremely thick and bold. Therefore, after preprocessing, 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.

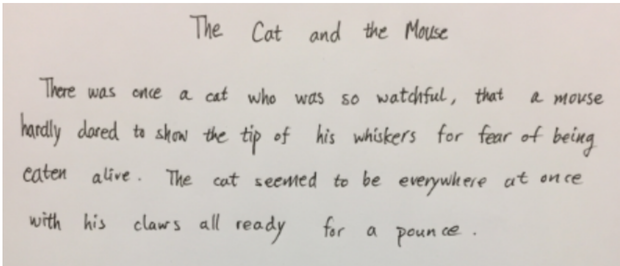
We ran our code on additional two document images that were taken under sub-optimal conditions: irregular lighting, partial skew, and/or glossy surfaces. Moreover, these documents had mixed font types and sizes, and the structure of the document was more complicated than the previous inputs. These images were `cv_jchoi_proposal.png` and `lav_notes1.png`. Despite the less optimal settings, we correctly denoised and thresholded the images and got similar results as before.

[4] https://keras.io/

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

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

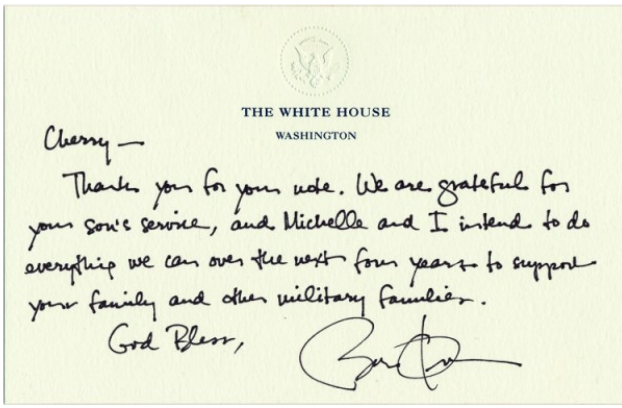
[7] http://github.com/mouuff/mtranslate



The Cat and the Mouse

There was once a cat who was so watchful, that a mouse hardly dored to show the tip of his whiskers for fear of being eaten alive. The cat seemed to be everywhere at once with his claws all ready for a pounce.

Figure 2: Reasonable handwritten doc: `hand_cat.png`



THE WHITE HOUSE
WASHINGTON

Cherry —

Thank you for your note. We are grateful for your son's service, and Michelle and I intend to do everything we can over the next four years to support your family and other military families.

God Bless,

Figure 3: Sample handwritten document: `obama1.png`

3.2 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 printed dataset. Moreover, any reasonably neat handwritten documents (`hand_miller.png` and `hand_cat.png`) showed even poorer performance. Figure 2 shows one of our reasonably neat images `hand_cat.png`. Furthermore, 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`. Figure 3 shows `obama1.png`.

4 Conclusion

We explored optical character recognition on photographed documents in various settings and formats. We tested with optimal and suboptimal photography settings, with documents using various fonts, and with printed and handwritten documents. For printed documents, we saw highly accurate results for 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 sufficiently small portions of handwritten documents (e.g. a single word or a short sentence), 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 the document into several different languages. Our short experiment with template matching also deserves future development into a project of its own.

How to Run the Code

Please extract `FINAL_PROJECT_jchoi100.zip` to view and run the code.

Assuming that the user has Python 2.5+, OpenCV, pytesseract, and tesseract installed, run the following command on the command line in the same directory as `driver.py`.

```
$python driver.py {path-to-image-file} {list of target languages separated by single space}
```

Sample usage: `$python driver.py data/doc7.png es fr`

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

Saved Output from Experiments

The output that we created by running our code on all the images in our dataset are saved under `FINAL_PROJECT_jchoi100/code/data`.

Acknowledgments

We thank the image processing and OCR libraries, the code we referred to for constructing CNNs using `Keras`, and MNIST for the digit dataset.

And thanks to Professor Reiter and the TAs!