Automatic Contact Importer from Business Cards for Android

Piyush Sharma
Electrical Engineering
Stanford University
Stanford, CA
spiyush@stanford.edu

Kacyn Fujii
Electrical Engineering
Stanford University
Stanford, CA
khfujii@stanford.edu

Abstract—This report describes the development of a standalone business card reader for Android smartphones. We use image processing techniques to preprocess the card’s image, which might have been taken from a difficult perspective, unfavorable lighting conditions, or partial occlusions. The text in the preprocessed image is then extracted using optical character recognition (OCR) and parsed to isolate the name, phone number, and email address of the contact to be automatically imported in the user’s address book. The app comprises of a single Java class linked with Tesseract and OpenCV libraries. The preprocessing is performed in C++ and called using Java Native Interface (JNI) provided by Android. The system was tested with business cards of varying appearances in a wide range of backgrounds and occlusions by the user holding it in his/her hand. Our standalone Android app performs well in these test scenarios.

I. INTRODUCTION

Recently, smartphones have become extremely popular in the general population — as of August 2013, 145 million people in the United States own smartphones [1]. Of these, Android phones make up 51.6% of the smartphone market share [1] and have increasingly powerful processors and cameras. This allows for the creation of standalone Android applications that can capture and process images. One such application is a business card scanner which imports contact information into the phone’s address book. There have been dedicated devices to scan and digitize physical business cards in the past. But given the popularity and power of smartphones, there is no need for a dedicated card scanner. Hence, we aim to use image processing on the Android mobile platform to accomplish this task. With such an app, scanning a business card should be as simple as taking a picture (Fig. 1).

Fig. 1. Importing contacts from business cards should be as easy as taking a picture.

There has been an interest in developing a mobile business card scanner in the past, which has led to multiple commercial apps on Google Play and the Apple Store. Additionally, EE368 students have also worked on a similar problem [2]. We aim to improve the performance of business card scanners by making preprocessing more robust, e.g. to the color of the card and background, occlusions by the user’s hand, and reflections from flash. We also aim to implement the end-to-end system on Android in a standalone application.

II. SYSTEM OVERVIEW

This section provides an overview of the system and the paper. The implementation details on Android mobile platform are discussed thereafter.

A. Pipeline

Fig. 2 is a flowchart of the application pipeline demonstrating the output after each step. Once the user captures an image with the camera, there are three major stages in the pipeline:

1) Preprocessing: The image captured from camera is preprocessed to prepare it for text recognition. Section III talks about the three steps in preprocessing — rectification, text bounding box detection, and binarization.

2) Optical Character Recognition (OCR): This stage recognizes individual text characters in the provided image. Section IV discusses this in detail.

3) Importing Android Contacts: The recognized text is parsed to extract name, email address, and phone number from the business card. The user is then prompted to add this contact to the phone’s address book as discussed in Section V.

B. Implementation

The end-to-end system was implemented on Android 3.0. The application consists of a single Java class, which links to OpenCV 2.4.6 [3] and Tesseract 3.02 [12] libraries. This class creates a user interface to capture the input image, run OCR, and add the extracted contact information to the phone’s address book (Section V). The Java class calls OpenCV code written in C++ using the Java Native Interface (JNI) to do the required preprocessing as discussed in Section III.
III. PREPROCESSING

Preprocessing is key to achieving reliable text recognition results with the Tesseract engine due to unideal image capturing conditions. The image captured with a handheld camera could have perspective distortions, which makes it difficult to recognize characters. Additionally, contrast between the characters and background could be low, which further toughens the task of text recognition. This section discusses our approach to handle these cases in the following subsections.

A. Rectification

OCR performs best if it is provided a frontal view of the business card. Generating this view from the input image is referred to as rectification in this paper. If the four corners of the card can be located, a homography can be computed, which warps the input image to get a frontal view of the card. Fig. 2 illustrates the steps involved in obtaining the four corners of the card and the homography thereafter. Other than warping, each step is performed on a downscaled version of the image and corresponds to a block in Fig. 2.

1) Find Edges: Find edges in the grayscale version of the input image using the Canny edge detector [6].

2) Find Edge Intersections: Apply the probabilistic Hough line transform [7],[8] to find line segments in the detected edges. We dilate the edge-map obtained from Canny using a $3 \times 3$ rectangular structuring element before this step to make the edges stronger. To find the intersections of these detected edges, line segments which touch the image boundary were rejected at this stage. We assume that the business card of interest does not intersect with the image boundaries. Also, since the edges of a rectified business card intersect at an angle of $90^\circ$, we assume that the card edges in the input image would intersect at angles in the range of $45^\circ$ to $135^\circ$ given the possibility of perspective transformation. All other edge intersections are rejected.

Additionally, the edges of the card intersect at their end-points. Thus, any point of intersection for which the maximum of end-point-distance-ratios for the point the two contributing lines was greater than 0.5 was rejected. End-point-distance-ratio for a line and a point is defined as the minimum of the point’s distance to the two end-points of the line, normalized by the line’s length. This test ensures that line-segments which intersect close to their end-points (both inside and beyond) are included, while others are rejected. Fig. 3 demonstrates the criterion. The case on the left has a low end-point-distance-ratio and hence the point “X” is a possible corner of the card. The case on the right has a very high end-point-distance-ratio and needs to be rejected.

---

1Although the rectification algorithm explained here is developed and illustrated for business cards, it can be applied to any rectangular object, e.g. family photographs, documents, and even whiteboards. As long as the object is the unrivaled biggest quadrilateral in the image, the algorithms should be able to rectify it, since no assumptions regarding color or texture of the business card were made other than the aspect ratio.
Fig. 4a shows all possible points of intersection (without any rejection criteria above), and Fig. 4b shows those which pass the above criteria. It clearly helps to reject those points early which do not correspond to the card’s corners.

We search for four points which give a quadrilateral of the biggest area, while also minimizing distortion. We model distortion as the sum of differences of lengths of the quadrilateral’s opposite edges. In other words, we try to find the biggest quadrilateral as close as possible to a parallelogram. Fig. 6 demonstrates the criteria — the distortion term in the cost function would force the function to select quadrilateral ABCD over AECD, even though the latter has larger area.

3) Find the Largest Quadrilateral: Next, we need to locate the points amongst these candidates which correspond to the actual corners of the card. For this, we first take the convex hull of these points with an assumption that the card is the biggest quadrilateral in the image. This typically brings down the number of candidate points from more than 100 to around 10. This allows us to do a more expensive test to reject possible candidates on the edges or neighborhood of the card. We decided not to use the Determinant of Hessian (DoH) because it would reject an actual corner of the card if it happens to be covered by user’s finger. Fig. 5 illustrates such a scenario where the bottom left corner of the card is under the user’s finger, and a high DoH criteria would discard this crucial corner. In contrast, our approach works well for this card image.

We compute a homography matching these corners to \((0,0), (W-1,0), (W-1,H-1)\) and \((0,H-1)\) in a clockwise order starting from top-left. Here, W and H are the respective width and height of the rectified business card image. Assuming a standard width to height ratio of 7:4, and considering the size of the input image, we set \(W = 1400\) and \(H = 800\).

B. Text Bounding Boxes

The grayscale version of the rectified image is used to compute local intensity variance to locate the possible bounding boxes for text segments. In a business card, text is often located in groups and multiple locations. To compute the variance at a pixel, we consider a neighborhood window of 35 x 35 centered at that pixel. Variance can be computed as \(E[X^2] - E[X]^2\), where \(X\) is a random variable representing the pixel values in the neighborhood, and \(E[\cdot]\) is the expected value operator. We compute \(E[X]\) by applying a box filter to the grayscale rectified image. \(E[X^2]\) is computed by applying the same box filter to the image obtained by squaring all pixel values.

A threshold of 100 was applied to the variance image. All locations with variance \(\geq 100\) were classified as text regions. Contours and bounding-boxes of these regions were then found. Rectification imperfections often lead to high variance at the borders of the image. Very large, very small, and too narrow boxes were rejected.

C. Binarization

Each bounding box is thresholded independently using Otsu’s method. This is a form of adaptive Otsu thresholding where the local regions are well defined text boxes. A comparison with adaptive Otsu’s method using tiling gave similar binarization results.

A distinction was made for binarization between cards with dark text in bright background and vice versa. In the former case, segments of text regions were binarized by inverse thresholding (and vice versa) and copied at corresponding places in a binary image initialized with zeros. A card was classified to be in the former case if more than half pixels were binarized to 255 using a global Otsu’s method.
Input images with their corresponding binarized image and the text locations marked in red boxes are shown in Fig. 10, 12, 14 in Section VII.

IV. OPTICAL CHARACTER RECOGNITION

A. Tesseract

Tesseract is an open-source OCR engine developed by HP Labs and currently maintained by Google. It is the leading open-source OCR engine and closely follows commercial OCR products in terms of accuracy. The Tesseract library can be linked from an Android app [5].

The Tesseract algorithm assumes its input is a binary image and works in two steps: preprocessing and recognition [12]. Tesseract preprocessing involves finding the outline of the text, identifying the text lines, and separating each character in a word. The recognition step passes the words it identifies to an adaptive classifier that it then uses for the subsequent text in the document. Thus, it should perform better on text further down in the page. This recognition step is run on the entire document for a second pass, since the classifier did not have as much training data for the earlier words in the document [12].

B. Business Card Text Recognition

Our Android application uses the tess-two library, which uses a fork of the Tesseract Tools for Android. The tess-two library works with version 3.03 of Tesseract and contains the native code within the "jni" folder of the library. To use the tess-two library in our own Android application, we imported the tess-two library and placed the necessary training files in the "assets" folder of our app. Our app then copies the files to the phone's SD card to use as necessary. The training files can be found online [10].

Lastly, the tess-two library has to be compiled before it could be used in Android. For a Windows machine, this is done by going to the "jni" folder of the tess-two library within Command Prompt and calling ndk-build. To call ndk-build for Windows, you must specify the path to the Android NDK (For example: "C:/android-ndk-r9b/ndk-build"). For a Mac, the command "ndk-build" will work once you are in the tess-two folder. After this step, OCR is ready for use by the main Android app. The steps are also illustrated in [12].

Instead of running OCR on the entire image, we ran OCR on the identified text locations indicated by the bounding boxes as described in Section III-B. The rectified image was then cropped using the bounding boxes and OCR was run on the smaller images. The resulting text was then concatenated as the final output of OCR.

V. IMPORTING ANDROID CONTACTS

A. Extracting contact information

Once the text from the business card is extracted using Tesseract OCR, a parser using regular expressions in Java is applied to the text in order to find the pertinent contact information. Specifically, the parser isolates the contact's name, email address, and phone number. The parser code is a modified version of the code available in the Eyes-Free project [4] and uses Java classes Pattern and Matcher. The Pattern class [9], which contains a compiled regular expression, is used in conjunction with the Matcher [9] class, which contains the output of applying the regular expression to an input. Java uses a subset of Perl 5 regular expression syntax [11].

For name matching, the regular expression searches for words that begin with a capital letter. It can accommodate up to three names (first, middle, last) and will also accept initials. Hyphenated last names are also acceptable. For example, it will recognize Alec M. Fletcher-Jones as a name. The phone matching regular expression recognizes phone numbers in the forms: (123) 456-7890, 1234567890, 123.456.7890, and 123-456-7890, for example. It will not accept strings with the wrong number of digits. Finally, the email matching regular expression looks for the "@" symbol and ".com" or similar expressions. It will accept any two or three character string after the ".", so ".edu" or ".ca" are also acceptable, for example.

B. Adding Contact Information

Once the contact's name, email address, and phone number are extracted, the user is then prompted to choose if they want to save the contact information to their phone's address book (Figure 7).

If the user selects "No", the app will continue to display the rectified, binarized input. The app can then be restarted so the user can scan another business card.

If they select "Yes", an Android intent is started to let the user add this information to their contacts list. The name, email address, and phone number fields found from the parser are pre-filled for the user, as shown in Figure 9.

If they select "Yes", an Android intent is started to let the user add this information to their contacts list. The name, email address, and phone number fields found from the parser are pre-filled for the user, as shown in Figure 9.

The user has the option to modify any of the contact information before saving it to the address book. A message will pop up saying "Contact Saved" after the contact has been saved successfully. A video demonstration of this process is available [13].
VI. RESULTS

In this section, we present the output for a few tough input images. Please note that for all the cases, the input to the OCR engine is each of the individual bounding boxes containing text.

Fig. 10 and Fig. 11 correspond to a case in which the card is black in color and placed in a brighter background. This is the opposite of the case illustrated in Section III. Additionally, Fig. 12 and Fig. 13 illustrate a case where there are strong reflections on the card. Finally, Fig. 14 and Fig. 15 correspond to a case where the card is occluded by a hand and of similar color to the background.

For all three test cases, our algorithm is successfully able to rectify and binarize the image. The regions of text are also successfully identified. Even with difficult conditions, our app extracts useful name, phone, and email contact information with accuracy.

VII. CONCLUSION

An Android app to capture and process business card images was developed. The standalone Android app uses image processing techniques to rectify and binarize the input image. The Tesseract OCR engine is used to recognize characters within text regions determined by our algorithm. The raw OCR output is then parsed to extract out contact information including name, phone number, and email address. Additionally, the Android app prompts the user to save the extracted contact information to the address book.

It was observed that the developed algorithms are robust to difficult conditions like uneven illumination, reflection, cards held and partially occluded by the user’s hand. It was further noted that the performance of Tesseract OCR engine is highly dependent on the quality of the preprocessed image.

The deliverable of this work is a standalone Android app that successfully scans business cards.

VIII. FUTURE WORK

It was observed that the performance of Tesseract, and hence the entire system, is highly dependent on pre-processing. The developed rectification algorithm works reasonably well, but fails in presence of significant background clutter. This is because some lines in the background could confuse the largest quadrilateral finder of Section III-A3. The cost function described therein could be improved, for example by taking into account the color information of the scene. Also, statistical information of common words and characters appearing on business cards can be modelled to improve the parser for extracting contact information from OCR results.

IX. ACKNOWLEDGMENTS

We would like to thank David Chen for his help and encouragement throughout the project. We sincerely appreciate the time and insights he shared with us. We would also like to thank Sam Tsai for his help with Tesseract. Finally, thanks to Prof. Bernd Girod for the great class. The lessons learned in EE368 were key in making this project a success.

REFERENCES


X. APPENDIX

The work presented in this report and the accompanying source code are the results of joint work of the following two students. This appendix outlines which portions of the work were completed individually and as a team.

A. Piyush Sharma

Piyush developed and implemented the pre-processing stage of the pipeline. This involves rectification, text-box recognition, and binarization. Most of the code was written in C++ using OpenCV and the Android app used JNI to call the native C++ code.

B. Kacyn Fujii

Kacyn developed a prototype app to run the Tesseract OCR engine. She also completed the task of parsing the OCR output to extract the contact information and further add it to the phone’s address book.
Fig. 10. Input and Rectified Binarized images for set 1. Please note that card is darker than the background. This is the opposite of the case illustrated in Section III.

![Input Image](image1.png)  ![Rectified with Bounding Boxes](image2.png)

Fig. 11. The output of OCR engine and parser to extract contact information. Note that the input image does not have a phone number.

![Input Image](image3.png)  ![Rectified with Bounding Boxes](image4.png)

Fig. 12. Input and Rectified Binarized images for set 2. Please note that there are strong reflections on the card.

![Input Image](image5.png)  ![Rectified with Bounding Boxes](image6.png)

Fig. 13. The output of OCR engine and parser to extract contact information.

![Input Image](image7.png)  ![Rectified with Bounding Boxes](image8.png)

Fig. 14. Input and Rectified Binarized images for set 3. Please note that the card is held in the user’s hand and is of similar color to the background.
C. Joint Efforts

Most of the brainstorming and debugging of the integrated app was done as a team. Additionally, the poster, report, and demo video were created together.