CIS 581: Computer Vision, Fall 2023
Instructor: Professor Jianbo Shi

Final Project: Smart Scanners for Cards and Documents

Ivy Luxen Xie

Final Report (12/05)

Contents

1 Abstract

In this project, I implemented "Smart Scanners for Cards and Documents". I will explore the open-ended track 6 not covered by other topics taught in CIS 581, by designing two smart scanners upon which several tasks could be performed. One scanner is built for scanning bank cards and displaying card number and type, and the other scanner is used to scan standardized test answer sheets and grade the answers. In addition, they analyze the information on the cards and documents and pass the correct results accordingly.

2 Introduction

Scanning real-world documents and cards to identify and obtain relevant information is one of the most practical utilizations of computer vision technology, and it is also the main purpose behind this entire project. Specifically, I designed two smart scanners by implementing a template matching Optical Character Recognition (OCR) to recognize the type of a bank card along with the 16 debit card digits and Optical Mark Recognition (OMR) system that facilitated our ability of capturing human-marked documents and automatically analyzing the results. I will need Numpy, imutils and OpenCV library to accomplish this project.

I will use template matching as a form of OCR to help create a solution to automatically recognize bank cards and extract the associated bank card digits from scanned images. I chose to use the OCR-A font, a standard font created specifically to aid Optical Character Recognition algorithms, along with MICR which stands for Magnetic Ink Character Recognition code designed for easy OCR. To accomplish this, I will need to apply a few image-processing operations, including thresholding, computing gradient magnitude representations, morphological operations, and contour extraction. I plan to use sampled bank card images for this project, so I have gathered a few example images of debit cards. These debit cards are not real and for demonstration purposes only. As for the test answer scanner, A dataset of filled-in standardized test answer (STA) sheets is essential for this project. It allows for training a model to accurately recognize and grade the responses, making it useful for automating multiple-choice test evaluation. I am going to combine the techniques from many CIS 581 topics, including building a document reader, contour sorting, and perspective transforms. Using the knowledge gained from previous CIS 581 projects, I'll be able to make quick work of this STA sheet scanner and test grader.

3 Bank Card Scanner

3.1 Methods and Implementation

3.1.1 Capture Information on Card Surface

- ✓ Define a dictionary that maps the first digit of a bank card to its card type.
- ✓ Use the reference OCR-A image and convert it to grayscale, threshold it to make the digits appear as white on a black background. After that, invert it to get black digits on a white background. Find contours of the digits and initialize a dictionary to map digit names to the region of interest.

OCR-A Reference

Inverted OCR Image

- ✓ Detect the location of the bank card in the image.
  - ➢ Load the reference OCR image and test sheet from disk, convert both to grayscale, threshold, and invert OCR image, and blur the test sheet to find edges.
    - cv2.imread, cv2.cvtColor & cv2.threshold
    - cv2.cvtColor & cv2.GaussianBlur & cv2.Canny
  - ➢ Find contours in the OCR image and edge map of test sheet, sort them, and

initialize a dictionary to map digit name to the ROI and the contour that corresponds to the test sheet.
    - cv2.findContours, imutils.grab_contours & imutils.contours.sort_contours

- ✓ Localize the four groupings of four digits, pertaining to the sixteen digits on the card and extract the digits from our reference image and associate them with their corresponding digit name.

3.1.2 Select and Analyze Embedded Results

- ✓ Resize input image and convert it to grayscale, and structure a rectangular kernel. Apply a Whitehat morphological operator to find light regions against the dark background to isolate card numbers for computing the Scharr gradient of the Whitehat image, then scale the rest back into the [0, 255] range.
- ✓ Apply a closing operation using our kernel to help close gaps between card number digits, then apply Otsu's thresholding method to binarize the image. Next, find contours in the threshold image, then populate a list with digit locations. Since bank cards use a fixed size font with 4 groups of 4 digits, we can prune potential contours based on the aspect ratio by computing the coordinates of bounding boxes of the contours to derive the aspect ratio. Then we sort the digit locations and initialize the list of classified digits.
- ✓ Recognize the type of card (i.e., Visa, MasterCard, American Express, etc.). Apply template matching to each digit by comparing it to the OCR-A font to obtain our digit classification. Examine the first digit of the credit card number to determine the issuing company via various image processing techniques, including morphological operations, thresholding, and contour extraction.
    - ➢ Compute a Scharr gradient of the Whitehat image of bank card in the x direction to isolate the digits. After computing the absolute value of each element in the gradient array, we take some steps to scale the values into the [0, 255] range, then we convert the gradient array to a uint8 which has a range of [0, 255]
    - ➢ Do a closing operation to close the gaps. Then we perform an Otsu and binary threshold of the gradient image of bank card, followed by another closing operation.
- ✓ Lastly, we extract the group region of interest of 4 digits from the grayscale image, then apply thresholding to segment the digits from the background of bank cards. Then we compute the bounding box of the individual digit, extract the digit, and resize it to have the same fixed size as the reference OCR-A image. We can loop over the reference digit name and region of interest to apply correlation-based template matching, obtain the score, and update the template matching scores list. The classification for the digit region of interest will be the reference digit name with the largest template matching score. The last step is to draw the digit classification around the groups and to display the card information to the screen.
- ✓ Output the card number and card type to our terminal and display the output image to our screen.

3.2 Design Choices and Results

I will have OpenCV, Numpy, Matplotlib, Argparse and the most recent version of Imutils installed on my system, along with a helper function named imageutils to find the center of the image and display the output. My set of convenience functions make performing basic image processing operations easier, and will make a template matching system for the OCR-A font, commonly found on the front of bank cards.

- ➢ Download and install imutils 0.5.4 which contains series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much easier with OpenCV and Python.

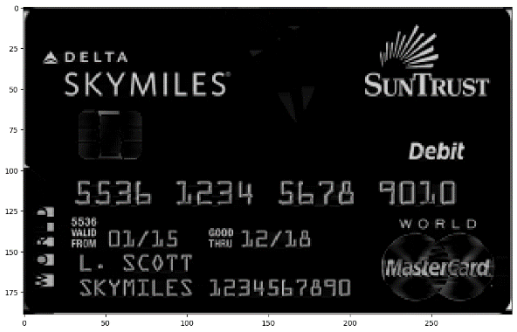
Bank card types, such as American Express, Visa, etc., can be identified by examining the first digit in the 16 digit card number. I will draw a rectangle around each group and view the bank card number on the image in red text. For the third and final block for this loop, I will draw a 5-pixel padded rectangle around the group followed by drawing the text on the screen.

- **Template Matching OCR**:

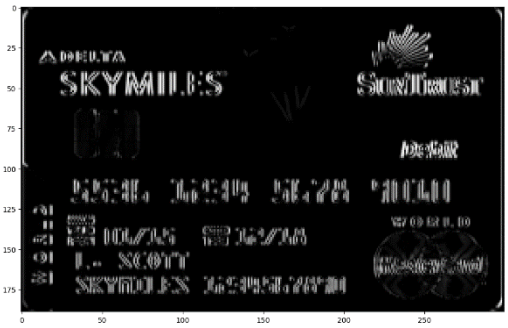
  Template matching is a technique based on the geometric coordinates of regions of interest (ROI) OCR means optical character recognition. We can OCR specific zones and get the output text fast and directly insert that data into any database.

  First, we load the reference image followed by converting it to grayscale and thresholding/inverting it. In each of these operations we store or overwrite OCR reference image. It needs to ensure every digit is resized to a fixed size to apply template matching for digit recognition. It continues by initializing a couple structuring kernels. You can think of a kernel as a small matrix which we slide across the image to do convolution operations such as blurring, sharpening, edge detection, or other image processing operations. To apply OCR to recognize the sixteen digits on the bank card, we need to devise our own custom solution to OCR bank cards. Extract each of these four groupings followed by segmenting each of the sixteen numbers individually. Recognize each of the sixteen card digits by using template matching and the OCR-A font.

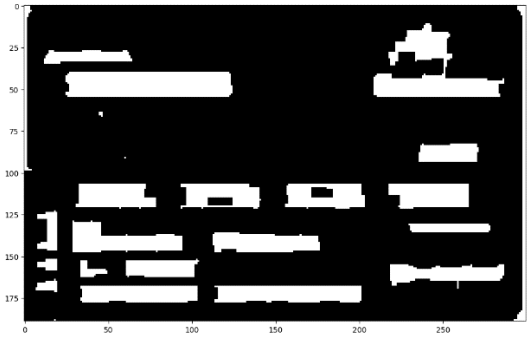
  OpenCV has a handy function called cv2.matchTemplate in which we supply two images: one being the template and the other being the input image. The goal of applying cv2.matchTemplate to these two images is to determine how similar they are. In this case we supply the reference image and the ROI from the bank card containing a candidate digit. Using these two images we call the template matching function and store the result. Next, we extract the score from the `result` and append it to our scores list. Using the scores (one for each digit 0-9), we take the maximum score which should be our correctly identified digit.



Whitehat Operation



Scharr Gradient



Otsu's Thresholding

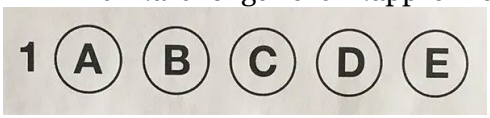
Final Output

4 Standardized Test Answer Scanner

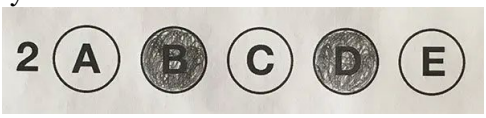
4.1 Methods and Implementation

4.1.1 Capture Information on Document Surface

- ✓ Detect the test region in an image.
    - ➢ Loop over the sorted contours of the test sheet and populate an array with all digits. If our approximated contour has four points, then we have found the paper.
        - cv2.arcLength & cv2.approxPolyDP



Empty Questions



Filled-in Questions

- ✓ Apply a perspective transform to extract the top-down, birds-eye-view of the exam. Extract the set of answers choices from the perspective transformed exam.
    - ➢ Load the input image, resize it and convert it to grayscale. Structure a kernel and apply a Whitehat morphological operator to find light regions against a dark background to extract the card number.
    - ➢ Apply a four-point perspective transform to both the original and the grayscale images to get a bird's eye view of the test sheet, and threshold the warped sheet for binarization.
        - imultils.perspective.four_point_transform
- ✓ Sort the questions/answers into rows. Determine the filled-in answer for each row.
- ✓ Lookup the correct answer in our answer key to determine if the user was correct in their choice. Repeat all questions in the exam.

4.1.2 Select and Analyze Human-marked Results

- ✓ Define correct answers, load the image, convert it to grayscale and blur it. Find contours in the edge map to initialize contours that correspond to the document. Apply perspective transform to both the original image and grayscale image to obtain a top-down birds eye view of the document.
- ✓ Apply Otsu's thresholding method to binarize the warped paper. Next, we find contours in the thresholded image and initialize the list of contours that correspond to the questions.
- ✓ Find contours in the thresholded test sheet and initialize the list of contours that correspond to questions. Sort the question contours top to bottom to find correct answers.
- ✓ We will then loop over the questions in batches of the number of possible answers. To loop over question alternatives, we need to construct a mask that reveals only the current answer for the question and count the number of non-zero pixels in the masked area. If the number is larger than the previous total non-zero pixels, it is one of the filled answers.
- ✓ Sort the contours for the current questions and initialize the index of the bubbled answers. Construct a mask for the current answer, apply the mask to the thresholded image, and count the number of non-zero pixels in the bubble area. Any answer that has a larger number of total non-zero pixels should be a filled-in answer. Then we draw the outline of the correct answer on the test.
- ✓ Then we will draw the outline of the correct answer on the test. Finally, we calculate and print the score based on how many correct answers are filled in.

4.2 Design Choices and Results

As for scanning the test answers, I have written the entire answer key in plain English here:

Question #1: B
Question #2: E
Question #3: A

Question #4: D
Question #5: B

Based on whether the test taker was correct or incorrect yields which color is drawn on the exam. If the test taker is correct, I will highlight their answer in green. However, if the test taker made a mistake and marked an incorrect answer, I'll let them know by highlighting the correct answer in red.

- ✓ **Optical Mark Recognition (OMR)**:

  Optical Mark Recognition, or OMR for short, is the process of automatically analyzing human-marked documents and interpreting their results.

  We do this by digitizing all text off documents. OMR system has two main components: a template and scanned forms. The template is a non-filled document where we specify the location of the data we want to extract. The data includes some solid object at the edge of the document and human-filled fields. Then we turn the template into the scanned form with human-marking data on it. OMR is a simplified version of OCR which I will further elaborate below.

- ✓ **Contour Extraction:**

  We apply the cv2.findContours function to find the lines that correspond to the exam itself by converting input image to grayscale and blurring it to reduce high frequency noise. Then we check the number of vertices on the contour by approximating the contour, which in essence means we simplify the number of points in the contour, making it a "more basic" geometric shape. Make a check to see if our approximated contour has four points, and if it does, we assume that we have found the exam. We also apply NumPy array slicing and contour sorting to sort the current set of contours. To determine which regions of the image are answers, we first loop over each of the individual contours to compute the bounding box which also allows us to compute the aspect ratio, or more simply, the ratio of the width to the height.

- ✓ **Perspective Transformation**:

  Once we have used contours to find the outline of the exam, we can apply a perspective transform to obtain a top-down, birds-eye-view of the document. In this case, I will use my implementation of the four-point-transform function within imutils library's perspective module which orders the (x, y) coordinates of contours in a specific, reproducible manner; Apply a perspective transform to the region. We apply a perspective transform to obtain a 90-degree viewing angle of the document.

- ✓ **Otsu Segmentation/Thresholding:**

  This step starts with binarization, or the process of thresholding/segmenting the foreground from the background of the image. After applying Otsu's thresholding method, our test sheet is now a binary image. The background of the image is black, while the foreground is white. This binarization will allow us to once again apply contour extraction techniques to find each of the answers in the exam.

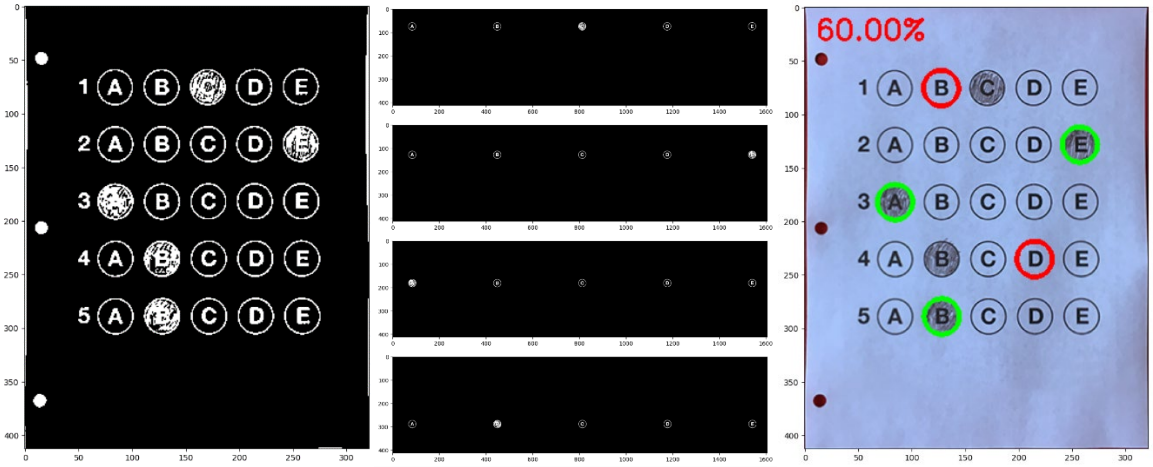
- ✓ **Canny Edge Detecting:**

  We apply the Canny edge detector cv2.canny function to find the edges/outlines of the test sheet. The edges of the document should be clearly defined, with all four vertices of the test region being present in the image. Obtaining this silhouette of the document is extremely important in our next step as we will use it as a marker to apply a perspective transform to the test sheet, obtaining a top-down, birds-eye-view of the document.

- ✓ **Morphological operations:**

  These operations apply a structuring element to an input image, creating an output image of the same size. The value of each pixel in the output image is based on a comparison of the

corresponding pixel in the input image with its neighbors. We need to construct two such kernels — one rectangular and one square. I will use the rectangular one for a Whitehat morphological operator and the square one for a closing operation, storing the result as Whitehat. The Whitehat operation reveals light regions against a dark background. Given our Whitehat image, I can compute the gradient along the x-direction. The next step in our effort to isolate the digits is to compute a Scharr gradient of the Whitehat image in the x-direction by using cv2.Sobel function. We then construct a mask for the current answer and count the number of non-zero pixels in the masked region by using cv2.morphologyEx function. The more non-zero pixels we count, the more foreground pixels there are, and therefore the answer choice with the maximum non-zero count is the index of the answer that the test taker has filled in.



Otsu's Thresholding | Contours of Filled-in Questions | Final Output

5 Related Works

- What is OMR(Optical Mark Recognition)?

https://remarksoftware.com/omr-technology/what-is-omr-optical-mark-recognition/

- How to Extract Information from Documents: Template Matching

https://aicha-fatrah.medium.com/how-to-extract-information-from-documents-template-matching-e0540ae79599

- Morphological Transformations Tutorials

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

- Contour Tracing Algorithm

https://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/intro.html

- Brief Study of Image Thresholding Algorithms

https://www.analyticsvidhya.com/blog/2022/07/a-brief-study-of-image-thresholding-algorithms/

6 Conclusion and Further Considerations

Although my project was working successfully overall with all the provided sample images and outputs correct results, there are some limitations regarding the quality and size requirements of the scannable images.

For instance, the test sheet scanner I implemented only works for test sheet in good lighting condition and position, and it cannot be too large, too small or have folded corners, otherwise the

contour detection and four-point perspective transform won't work perform well, and it might end up outputting empty sheet or incorrect grade. Moreover, only traditional bank cards issued by big banks with standard fonts and formats are scannable and return correct numbers without any issue.

To further extend my implementation, we can apply a broader algorithm to tackle each issue brought up to be able to still extract information from tilted or folded exam sheet, and obtain corresponding digits from unconventional cards issued by small banks such as BestBuy credit cards, etc. There is significant room for improvement. I have brainstormed the following potential improvements and optimizations for my scanners.

- ✓ Add more python libraries to the package and implement some self-correcting functionalities to fix any issue on the scannable images such as folded corners, poor lighting, angle issues or size issues, etc.
- ✓ To further expand the bank card scanner, we need to initialize more card types and more different OCR reference images to be able to include bank cards issued by small banks and institutions.
- ✓ The OCR template matching's bounding box computation based on fixed ratios for the standard bank cards' digits and test sheet's answers can be further extended to different ratios according to different card types to produce more accurate and diverse results.
- ✓ Make more adjustments to the Whitehat morphological operation to make it flexible enough to detect filled-in answers in any lighting condition or asymmetrical position.

The skills I learned in 581 are very useful for completing this project and future exploration in the field of computer vision. I would like to thank the TAs and Professor Shi for such a rich and interesting learning in this semester.