Mobile Lottery Ticket Recognition Using Android Phone

Ashish Gupta, Thomas Zou
Department of Electrical Engineering
Stanford University
Stanford, CA
thomaszou85@gmail.com, ashish1.gupta@gmail.com

Abstract— we explore an image processing algorithm for detection of lottery ticket numbers using the android phone platform. American spends billions of dollars on lotteries every year. When there are huge jackpots, many buy multiple tickets with multiple numbers. Manually checking each ticket is cumbersome while checking the ticket at the store removes the possibility of being anonymous. First we will discuss the MATLAB implementation of the image processing algorithm and report on its challenges. Second we will give an overview of what was implemented on the android phone. Finally we will summarize the challenges and possibilities for future work.

Keywords-component; Image Processing; Image Segementation; OCR; Android

I. INTRODUCTION

Our goal is to use the Android phone to capture images of the lottery ticket and perform date and numbers extraction. The date and number extracting must be robust against rotation, lighting, scaling, and background variation. Once the date and numbers are extracted, we utilize an Optical Character Recognition (OCR) engine to translate the images into text format. Once we have the date and numbers, then we use the Android phone to query the lottery website to retrieve the winning numbers. We then compare the winning numbers versus the detected numbers to determine whether the ticket is a winner.

A. MATLAB

We implement the date and number extraction using MATLAB. MATLAB has many useful image processing functions built in. Some of these include region labeling, Canny edge detection, Hough Transform, and image dilation and erosion.

B. Tesseract

We use Tesseract for our OCR engine. Tesseract is arguably the best free open source OCR engine available. It was originally developed by HP Labs between 1985 and 1995 but now has been taken over by Google. We rely on Tesseract to be able to accurately recognize a uniform font size of digits and alphanumeric characters.

II. SYSTEM IMPLEMENTATION

A. Overview

Our original intent was to implement the image processing algorithm entirely using OpenCV and have it run entirely on the Android phone. OpenCV stands for “Open Source Computer Vision” and includes a very rich set of image processing libraries we could integrate into the Android development system on the phone. We were unable to implement this system since we did not realize that OpenCV did not have a region labeling function integrated. We did investigate the CVBlobs libraries, but it was decided that it was too risky to implement our algorithm in OpenCV on the phone.

The final implementation uses the Android phone to capture the image and perform user interaction and feedback. We implement our image processing algorithm in MATLAB on a server. The communication between the phone and server is achieved thru HTTP calls using PHP scripts.

B. Android Phone

The Android phone captures an image and then sends the image to the server running MATLAB using a HTTP call to a PHP script. When the result from the server returns, the phone parses the date and numbers and sends a HTTP request to the lottery website (Super Lotto or Mega Millions) to retrieve the winning numbers based on the recognized date. It then compares and matches the numbers and displays the image on the phone, showing the matching numbers in red, and the unmatched numbers in green.

We also implemented a demo mode where the user can input the winning numbers instead of fetching them from the lottery websites. We used this demo mode extensively in our testing.
C. **Server Side**

Once the server running MATLAB recognizes that it has received a file from the Android phone, it starts processing the image. MATLAB outputs 7 images which then are sent to the Tesseract OCR engine. The output of the Tesseract is returned back to the phone thru the HTTP connection.

### III. IMAGE PROCESSING PIPELINE

The image processing pipeline is responsible for cleanly extracting the date and lottery numbers from the image of the lottery ticket, so they can be sent to the Tesseract OCR engine for identification. A sample image of a lottery ticket taken with an Android phone is shown in Figure 1. As can be seen from the figure, lottery tickets contain watermarks, and other security features (thin curved lines) that can occur anywhere on the ticket. These security features make clean extraction of the date and numbers a challenging task. The sample ticket contains 10 rows of numbers, but lottery tickets can contain anywhere from 1 to 10 rows of numbers per ticket. The image processing pipeline is responsible for determining how many rows of numbers are on the ticket, and extracting all of them.

![Figure 1. Sample Lottery Ticket](image1.jpg)

It can be seen that the sample image has some perspective distortion, and is also slightly rotated. Variations in scale, perspective distortion, background color/texture, and lighting conditions all pose challenges when trying to cleanly extract the date and numbers from the ticket. To reduce the complexity, we constrained the problem by requiring the image to fully contain the barcode, numbers and date, and requiring the scale, rotation, perspective distortion and lighting to be as good as can be reasonably expected to be achieved by a user of a mobile phone. No constraints have been put on the type of background the ticket is placed on.

Figure 2 shows the stages in our image processing pipeline, and the following sub-sections describe the details of each stage, showing the results of processing the above sample ticket at each stage. Our current implementation of the image processing pipeline uses MATLAB; however, our longer term goal is to port the MATLAB algorithm to OpenCV and Java, so we can run it natively on an Android phone.

![Figure 2. Image Processing Pipeline](image2.png)

#### A. **Binarization**

The binarization stage of the algorithm converts the image from a color (RGB) image to a binary image where each pixel is represented by a value of 1 (white) or 0 (black). The goal of the binarization algorithm is to remove as much of the clutter (watermarks, security lines) as possible, while fully retaining the date, columns of numbers, and barcode.

Our first attempt was to convert the image to grayscale, and then use a locally adaptive threshold to convert to a binary image; however, we were not able to remove the clutter of the watermarks and security features with this method. Next, we attempted to remove the clutter by employing a color thresholding scheme to make all the non-black regions of the image white. We investigated color thresholding in the YCbCr, HSV, and RGB color spaces, but we were not able to find consistent ranges that worked across varying lighting conditions. During the investigation of color thresholding, we noticed that the ‘R’ (Red) component of the RGB image was free of most of the clutter, but still contained all the information of interest (date, columns of numbers, and barcode). Figure 3 shows an image of the ‘R’ component for the sample lottery ticket.

![Figure 3. Red Component Picture](image3.jpg)
After identifying this ‘clean’ starting point, the next step is to convert the ‘R’ component of the original image to a binary image. We experimented with different methods including a global threshold using Otsu’s method, locally adaptive thresholding based on Otsu’s method, thresholding based on the median value of the pixels in the neighborhood region of each pixel, and thresholding based on the mean value of the pixels in the neighborhood region of each pixel. We obtained the best results by comparing each pixel against the mean value of the 50 pixels surrounding it, and setting the output pixel to 0 when the pixel value is lower than 90% of the mean.

Figure 4 shows the result of applying this mean filter thresholding scheme on the ‘R’ component of the sample lottery ticket.

As can be seen from Figure 4, there is noise in the binary image. As a final step in the binarization stage, we inverted the thresholded image, and applied region labeling, and filtering by region area to remove small area regions. Figure 5 shows the result of applying small region removal to the sample lottery ticket.

B. Barcode Corner Detection

After binarization, the next stage in the algorithm is to detect the barcode. The location and size of the barcode are later used to perform perspective distortion correction, rotation, and scaling of the image. Instead of using the barcode for these operations, we first tried to use the four corners of the lottery ticket, but we discovered that distinguishing the four corners from the background was very difficult when the ticket was put on a yellow or orange background. Since we wanted to keep the algorithm background invariant, we decided to detect the 4 corners of the barcode instead of the 4 corners of the ticket.

To detect the barcode, we dilated the binarized image from Stage 1, with a structuring element sized to join all the individual barcode elements together. The result of dilating the sample lottery ticket is shown in figure 6.

As can be seen from Figure 6, the result of applying small region removal to the sample lottery ticket.
with the largest Major Axis Length are removed, leaving only
the barcode remaining. The result of barcode detection for the
sample lottery ticket is shown in figure 7.

![Figure 7. Barcode Extracted Image](image)

---

Once we obtained an image with only the barcode in it, we
identified the coordinates of the 4 corners of the barcode by
performing Canny Edge detection, followed by a Hough
transform. The 4 peaks of the Hough transform are taken to be
the 4 lines of the quadrilateral that maps to the barcode, and the
intersection of these 4 lines is computed to determine the 4
corners of the barcode. If the 4 corners of the barcode cannot
be found, the algorithm gives up at this point, and asks the user
to take another image.

---

C. Perspective Distortion Correction, Rotation, and Scaling

As stated above, one of the goals of the algorithm is to be
invariant to reasonable perspective, rotation and scale. Stage 3
of the algorithm performs perspective distortion correction, and
normalizes the scale and rotation of the image. The 4 corners
of the barcode detected in stage 2, are fed into a projective
transform, along with 4 outputs coordinates. The 4 output
coordinates are found by using the detected bottom left corner
of the barcode for one of the coordinates, and calculating the
other 3 by using the height and width of the normalized
barcode we are trying to achieve.

The output of the projective transform on the barcode of the
sample lottery ticket is shown in figure 8.

![Figure 8. Transformed Barcode Image](image)

---

The same transform is also applied to the ‘R’ channel of the
original image from Stage 1, and the binarized image from
Stage 1. All three of these transformed images are required in
later stages of the algorithm. The transformed ‘R’ channel and
binarized image for the sample lottery ticket are shown in
figure 9.

![Figure 9. Transformed Raw Images](image)

Unfortunately, using the barcode for the projective
transform proved to not be as robust as using the 4 corners of
the lottery ticket. If the ticket is slightly bent, or there is some
remaining noise in the image which gets dilated into the
barcode during the barcode detection stage, the projective
transform is not perfect, and this causes problems during stage
4 of our algorithm which relies on an almost perfect
transformation. Improvement of this weakness in the algorithm
is discussed in the Future Work section of this document.

After the projective transformation, the final step of this
stage is to rotate the 3 transformed images to orient the barcode
at the bottom of the image. The required rotation angle (90, -
90, or 180) is determined by performing region labeling and
checking the orientation of the barcode, and the coordinates of
its centroid.
D. Locate Date and Columns of Numbers

One of the outputs of stage 3 is a transformed, rotated, and scaled version of the binarized image from stage 1. From stage 3, we also know the location and size of the barcode in this image. For all of the lottery tickets, the location of the date relative to the barcode is fixed, so we determined constant values for the location of the date relative to the location of the barcode experimentally. In a similar manner, the horizontal starts and ends of each column of numbers are determined based on the location and size of the barcode. The heights of the columns of numbers is determined by dilating the transformed version of the binarized output from stage 1, and finding the median height of the bounding box of all regions with bottom corners in a constant region relative to the barcode location. The final column of numbers is treated as a special case. Since the “Mega” text is very close the first row of numbers, it is extracted along with the column of numbers by using the maximum region height instead of the median height that is used for the other columns.

The sample lottery ticket with the locations of the date, and columns of numbers identified is shown in figure 10. The left and right boundaries of the columns of numbers are shown in blue, while the top and bottom of the columns are shown in red. The top and bottom of the “Mega” column are shown in yellow, and the date region is marked in green.

E. Extract Columns of Numbers and Date

The output of stage 4 is 7 sets of bounding box coordinates (1 for the date, 1 for each of the 6 columns of numbers). The transformed original “R” channel of the image is cropped with each of these 7 bounding boxes to extract the date and columns of numbers. The extracted images are once again binarized using the same ‘mean filter thresholding’ that was used in stage 1.

The “Mega” is removed from the last column of numbers by doing a horizontal dilation and using region labeling to identify and remove the region with the centroid nearest the top of the column.

As a final step, the binarized date image, and 6 binarized number images are post processed before they are sent to Tesseract. A lot of experimentation and research was done to try to determine how to post process the data to improve Tesseract accuracy, however, there is very little information and documentation available for Tesseract. Downscaling by a factor of 4, and applying a Gaussian filter gave us the best results with Tesseract.

The 7 final images for the sample lottery ticket are shown in Figure 11.

IV. EXPERIMENTAL RESULTS

We processed 49 images taken under reasonable lighting, scaling, and perspective conditions with the MATLAB image processing code, and then ran the outputs through the Tesseract OCR. The barcode detection failed for 2 tickets (4%), and the projective transform resulted in slanted text for 5 of the tickets (10%). For the remaining 42 tickets, the percentage of correctly detected numbers is 89%, and the percentage of correctly detected dates is 82%. For all of these cases where the barcode detection succeeded, and the text was not slanted after the projective transform, the mismatches in the numbers and date are due to Tesseract not being able to reliably recognize reasonably ‘clean’ images. The most common recognition problems were distinguishing between ‘5’ and ‘6’, distinguishing between ‘8’ and ‘9’, and distinguishing between ‘2’ and ‘Z’.

V. CONCLUSIONS

We have successfully demonstrated that given reasonable lighting conditions, reasonable image size, and perspective, it is possible to create an application on a mobile phone to compare lottery ticket numbers against the winning numbers. With some improvements in robustness and performance, this application should be more convenient and user friendly than the existing similar apps which require the user to align each
row of lottery ticket numbers inside a rectangle, one row at a time.

VI. FUTURE WORK

The current image processing algorithm has two major weaknesses that we would like to improve. The first is that using the detected barcode for perspective correction is not completely reliable. In situations where the ticket is bent, or some noise gets dilated into the barcode during barcode detection due to uneven lighting conditions, the detected quadrilateral mapping to the barcode may not be perfect, resulting in the date and columns of numbers being slanted after a projective transformation. To improve this weakness, we plan to analyze the consistency between the text flow of the rows of numbers and the detected barcode lines, and use this to improve the transform result. In addition, rather than assuming that the text is perfectly aligned after the transform, we plan to search across each row of pixels to determine the start and end of each column. This will allow us to still correctly extract the columns of numbers even if they are slanted. The second major weakness is that Tesseract does not provide reliable results unless the image given to Tesseract is very clean. The images we are feeding to Tesseract often have missing pixels, which result in incorrect outputs. Since the font type and font size are fixed, and the alphabet size of a lottery ticket is small, we plan to try using template matching as a replacement to Tesseract.

In addition to accuracy, the performance of the algorithm also needs to be improved. The region labeling step of stage 1 can take a long time when there are uneven lighting conditions or textured backgrounds due to many very small (single pixel) regions after thresholding. We need to experiment with methods to remove this noise prior to region labeling to improve performance.

Finally, the current implementation is using a server client model where the Android App makes a call to a server running MATLAB to execute the image processing pipeline. We would like to port the MATLAB code to OpenCV and Java so it can run natively on the phone.

ACKNOWLEDGMENT

The authors would like to acknowledge Sam Tsai for all his advice and help on Tesseract, David Chen for his help on everything, and Professor Bernd Girod for his continued support through our project.

CODE SOURCES

We used the line calculation MATLAB code from spring 2010 EE 368 project “Business Card Recognition”.

We leverage the JAVA, PHP, and PYTHON code from EE368 Android tutorials as the base of our system code.

REFERENCES


APPENDIX

Ashish Gupta was responsible for researching Tesseract and the image processing algorithm. He was also the primary person to implement the algorithm in MATLAB.

Thomas Zou was responsible for implementation of the overall system. This includes Android application development and the server to phone communication. He also helped out in various stages in the MATLAB programming, and brainstorming to make improve the algorithm robustness.

We both contributed to this report, poster session, and many late night hours testing our code.