Camera Based Equation Solver for Android Devices

Aman Sikka
Department of Electrical Engineering
Stanford University
Stanford, California 94305
Email: asikka@stanford.edu

Benny Wu
Department of Applied Physics
Stanford University
Stanford, California 94305
Email: bennywu@stanford.edu

***Abstract*—In this paper we demonstrate an Android based mobile application equation solver that takes a camera image of an equation and displays the solution. The program is capable of solving simple arithmetic equations (addition, subtraction, and multiplication) and systems of linear equations of up to two variables written by hand or in computer font.**

I. Introduction

Mathematical equations are a ubiquitous part of student life. Generally, one has to make use of a calculator or computer to solve complex equations. Depending on the assistive device, the equations needs to be entered in a specific format. Our goal is to develop a mobile based application that bridges the gap between technology and the traditional pen and paper approach in an intuitive manner. The user can capture a camera image of a mathematical expression - both in computer generated font or in handwritten text, and a solution will be calculated and displayed on the mobile device.

Initially, we limit the scope of the task to solving simple arithmetic equations, quadratic equations, and systems of linear equations of up to two variables. Mobile applications for solving such equations do exist. However, these apps are essentially templates for pre-defined equations where users enter numbers in static text boxes. While the constrained interface ensures robust capture of the input, it is cumbersome for the user, who must click multiple times to input an equation. A more intuitive approach would be for the user to snap a picture of the written expression.

II. System Overview

The general approach for solving an equation contained in an image is as follows:

1) Capture image.
2) Binarize image.
3) Segment expressions.
4) Correct perspective distortion.
5) Recognize text.
6) Prompt for user confirmation.
7) Solve equation.

Due to limitations in the text detection accuracy, a prompt for confirmation from the user was added between steps 5 and 7 to ensure that the correct equation is passed to the solver. The overall system consists of two subsystems. The first is an Android application that captures the input image, displays the detected equation, prompts for confirmation, and displays the solution. The second is a server running the image processing, text recognition, and equation solving algorithms. The overall system flow is shown in Figure 1.

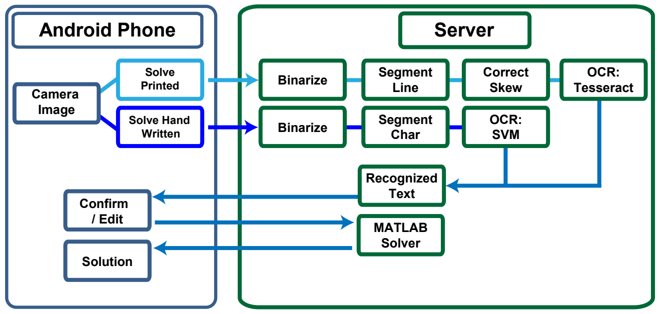

Fig. 1. Overall system flow showing processes running on the Android platform and on the server.

III. Image Capture

Within the mobile application, the user is given two options - "Solve Printed" or "Solve Hand Written" which capture an image and initiate two different PHP scripts on the server. The image is downsized to 640 x 480 pixels from its original size to speed up transmission to the server.

IV. Binarization and Line Segmentation

Binarization and segmentation are crucial steps in identifying the regions of interests. Otsu's adaptive thresholding was initially used. However, the processing time was fairly slow and the binarization was unsatisfactory for certain lighting conditions. Instead, the image is binarized using Maximally Stable Extremal Regions (MSER) from the VL_FEAT toolbox [2]. Regions that are either too small or too large are excluded from the candidate list. Regions in the binarized image are labeled, and their centroids and bounding boxes calculated.

Individual lines (a single equation or expression) are segmented out by plotting the y-centroids of the regions in a histogram with a bin size of 20 pixels, as shown in Figure 2(b). The average y-centroid of each line is assumed to be the maximum of the well-separated peaks. The midpoint between each line is then calculated and used to segment the binarized image into images of individual lines.

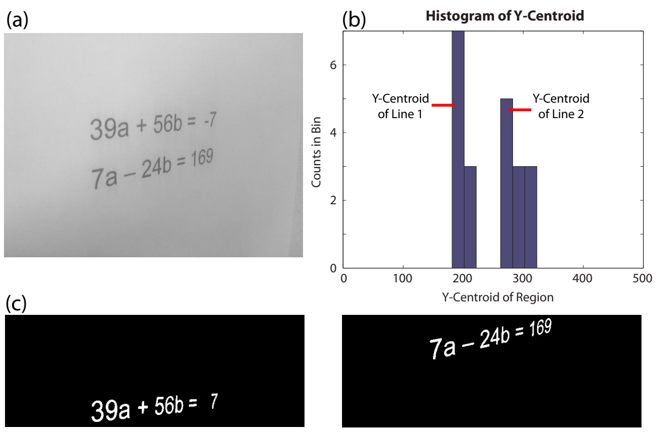

Fig. 2. (a) Camera image taken by Android phone. (b) After binarization with MSER and region labeling, a histogram of the y-centroid of the regions is plotted. Regions from separate expressions form distinct peaks. (c) Using the histogram results, the binarized image is segmented into separate expressions.

V. Text Recognition

Text recognition is divided into two separate classes - computer printed expressions and hand written expressions. Computer printed expressions are processed with Tesseract [3], a free Optical Character Recognition (OCR) engine originally developed by HP Labs and now maintained by Google. Handwritten expressions are processes with a Support-Vector Machine (SVM) prediction model.

In order to improve detection acuracy the number of characters in the database are limited to digits (0 - 9), two variables (a, b), and a subset of mathematical operators (+,-, x, and =). These characters are sufficient for composing both quadratic equations and the majority of a system of linear equations in two variables. The dictionary can be easily expanded to include more characters in the future.

A. *Computer Text Recognition*

Recognizing computer generated text is considerably easier than recognizing handwritten text. Due to the predictable structure (font style) of the characters, a template matching algorithm can be used. The free OCR engine Tesseract was chosen for printed text recognition.

As demonstrated in [4], the accuracy of optical text-recognition algorithms improves significantly if perspective distortion is first corrected. Following a similar method to [4], the topline and baseline of the regions in the segmented expression are found. The segmented line image is shown in 3(a). The RANSAC MATLAB Toolbox from the Graphics and Media Lab of Lomonosov Moscow State University [5] is used to compute RANdom SAmple Consensus (RANSAC) fits to the topline and baseline as shown in 3(b). A bounding box for the distorted text is found, and the four corner points identified. Using the leading edge of this bounding box, the desired, corrected bounding box is then calculated, as shown in 3(c). A projective affine transformation that maps the four corner points of the distorted box to the desired box is then used to correct for the perspective distortion. The resulting image after correction, as shown in 3(d), is then passed to the Tesseract OCR engine.

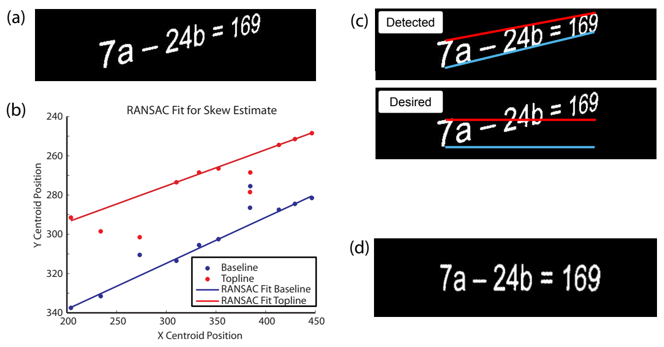

Fig. 3. (a) Segmented line image showing perspective distortion. (b) RANSAC fits to the topline and the baseline. (c) Top: The detected, distorted bounding box. Bottom: The desired, corrected bounding box. (d) Corrected image after projective affine transformation is applied to the distorted text.

B. *Handwritten Text Recognition*

Though Tesseract performed well for printed text, with a detection rate better than 85% for fonts within its database. However, its detection rates for handwritten text is below 50%, likely due to size variations in the writing and a lack of matching fonts in its database. A machine learning algorithm based on support vector machines (SVM) was applied instead.

SVM is a supervised learning method that analyzes data and recognizes patterns. It is often used for classification and regression analysis. The prediction model was created using libSVM, a set of tools for training and modeling SVM developed at National Taiwan University [6].

In order to create a training dataset, character images must first be converted into vector form. After line segmentation, region labels are used to determine the bounding box for individual characters. A small amount of padding is added to the border, as shown in Figure 4(a). The segmented character is downsampled to 32x32 pixels and further divided into 64 4x4 regions, with region 1 at the top left and region 64 at the bottom right. The count in each region is the determined vector value, as shown in Figure 4(b). This conversion thus results in a 64 dimensional vector for each character image.

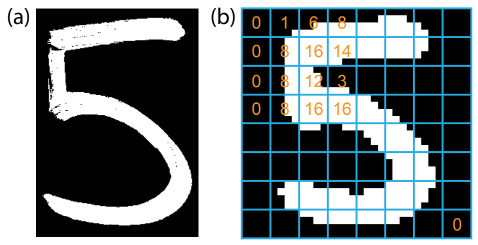

Fig. 4. (a) Segmented character from input image. (b) Character downsampled to 32 x 32 pixels. The image is then further divided into 64 4x4 regions, with region 1 at the top left and region 64 at the bottom right. The count in each region is the vector value.

Initially, the Optical Recognition of Handwritten Digits Data Set from the Machine Learning Repository at the University of California at Irvine [7] was used as the training set. The 64 dimensional vector conversion of this data set was kindly provided by Sam Tsai. After training libSVM, a test of our handwritten digits resulted in a low prediction rate of between 20-30%. We thus decided to create a new training data set based on our own handwriting. A simple application for taking a camera image of a single character, segmenting and downsampling the character, calculating the corresponding 64 dimensional vector, and writing it to an output text file, was created to generate a training dataset. Approximately 20 entries for each character in the dictionary (10 from each author) was entered as a training data set.

In hindsight, the most likely cause for the discrepancy between the training data in [7] and our handwritten digits is the use of the bounding box for downsampling the individual characters. The downsampled image as shown in Figure 4(b) has an aspect ratio of nearly one. On the other hand, characters from [7] appear to have been written in a predetermined box, preserving the nearly 1:2 (width:height) aspect ratio of most digits. The inclusion of this much larger dataset would likely have increased the robustness of the algorithm in detecting handwritten text from individuals not within the training data set.

VI. Experimental Results

The detection rate of individual characters in the dictionary dictates the overall probability of correctly identifying an equation. For testing, two sets of equations were used, one printed from the computer and the other handwritten, resulting in approximately 20 instances of each character within the dictionary. The detection rate for printed text using Tesseract and for handwritten text using SVM are shown in Figure 5(a) and 5(b) respectively. Except for the first entry, characters with detection rates within a 5% window have been grouped together

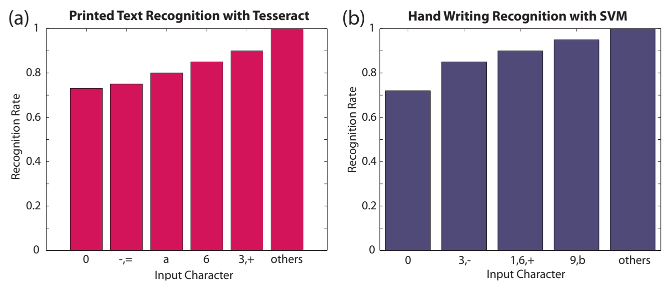

Fig. 5. (a) Accuracy of printed text recognition with Tesseract. (b) Accuracy of handwriting recognition with SVM prediction model.

A. *Tesseract OCR*

By limiting the number of alphabets to 'a','b', and 'x', the spell-check function of Tesseract is essentially circumvented. Segmenting and detecting an entire line of text was actually more accurate than segmenting and detecting individual characters.

Tesseract performs well for fonts that exist in its database, with an accuracy in the range of 85-90%. The notable exceptions were the characters '0',' -', '=', and 'a'. The character '0' was at times mistakenly identified as '()' (prior to the removal of '('and ')' from the dictionary). The character 'a' was at times mistakenly identified as '3'or '8'. The characters '-' and '=' were at times completely absent from the detection, likely due to the down-sampling of the camera image to 640x480 pixels. This problem was not observed for well-focused images of equations written in font sizes larger than 16. In fact, varying font sizes are not a factor if the image remains in focus and fills most of the camera's viewfinder.

B. *SVM OCR*

Handwriting recognition with SVM has a detection rate of 85-90% for our handwriting. It is expected that the accuracy will be considerably lower for individuals not in the training dataset. A few outliers are noted here. The characters '0','3', and '1' are at times mistakenly identified as 'a' or '6', '5', and '7' respectively. Problems detecting the mathematical symbols '-' and '+' appear to arise from the fact that their size is quite different from other characters within the dictionary. A proper implementation of the bounding box prior to downsampling would likely correct this problem.

VII. User Confirmation and Solution

Even for a system with a single character recognition rate of 95%, the probability of detecting a set of equations with 12 characters completely correctly is only slightly above 50%. It is thus critical to provide the user with a means to confirm or to edit the recognized text. Upon confirmation, the expression is sent back to the server where it is solved.

Once the user confirms the equation, the equation is converted into a MATLAB-readable expression by inserting missing operators. For example, the equation '$2a + 5b = 6$' is converted to '$2 * a + 5 * b = 6$'and the equation '$a^2 + a = 3$' is converted to '$a^2 + a = 3$'. After parsing, MATLAB functions *eval()* and *solve()* are invoked to solve the given expression. *eval()* is used for expressions with no variables such as '123*4563+789' and *solve()* is used for a system of equations. The resulting solution is then sent back to the Android device.

VIII. Future Work

The prediction rate of the SVM model is highly dependent on the training dataset. Currently, the training dataset only contains our handwriting. Ideally, a larger training dataset from multiple individuals should be collected, to accommodate for wide ranging styles in writing. An extension would be for the system to train itself on the fly. Each time the user sends the confirmation for an equation, the corresponding 64 dimensional vectors for the detected characters can be added to the exisiting training dataset. This would allow the training dataset to grow quickly and efficiently without the need for separate training.

The second improvement would be to make the application a stand-alone device by porting the existing server code to OpenCV on the mobile device. The idea can also be extended to text recognition on a tablet device with a stylus, which would would provide a seamless workflow for the user.

IX. Conclusion

In this paper, we describe an Android-based system for capturing and solving an image containing a mathematical expression. The system consists of two parts - an Android application for capturing the camera image, and for displaying the recognized text and solution, and a Server for performing the image processing, text recognition, and equation solving algorithms. The camera image of a mathematical expression is sent from the Android phone to the server, where it is binarized with MSER. For printed text, perspective distortion is corrected for with RANSAC, then text recognition is performed with the Tesseract OCR engine. For handwritten text, a SVM prediction model trained with libSVM is used to perform text recognition. The detected text is sent back to the Android device for confirmation by the user. The confirmed equation is then solved with MATLAB on the server, and the solutions are displayed on the Android device.

Acknowledgment

We gratefully acknowledge Sam Tsai for his excellent advice and suggestions, and for kindly providing the Optical Handwritten Digits Data Set in 64 dimensional vector format.

References

[1] VLFEAT Open Source Library. Available: http://www.vlfeat.org/
[2] Tesseract Optical Character Recognition Engine. Available: http://code.google.com/p/tesseract-ocr/
[3] Sam Tsai, Huizhong Chen, David Chen, Ramakrishna Vedantham, Radek Grzeszczuk, and Bernd Girod, ”Mobile Visual Search Using Image and Text Features”, Proc. 45th Annual Asilomar Conference on Signals, Systems and Computers, November 2011.
[4] Graphics and Media Lab RANSAC Matlab Toolbox. Available: http://graphics.cs.msu.ru/en/science/research/machinelearning/ransactoolbox
[5] LIBSVM, A Library for Support Vector Machines. Chih-Chung Chang and Chih-Jen Lin, Computer Science Department, National Taiwan University. Available: http://www.csie.ntu.edu.tw/ cjlin/libsvm/
[6] Optical Recognition of Handwritten Digits Data Set. Machine Learning Repository, University of California at Irvine. Available: http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

Appendix

Aman Sikka implemented the Android graphical user interface, the communication with the server, and the server PHP scripts. Benny Wu implemented the image processing algorithms MSER, segmentation, and perspective distortion, and the MATLAB equation solver. Both authors contributed to implementing the SVM training dataset and the prediction model.