Mobile Image-based Solver for Handwritten Equations

Scott Harvey
Department of Management Science and Engineering
Stanford University
Palo Alto, California

Will Harvey
Department of Computer Science
Stanford University
Palo Alto, California

***Abstract*—In this paper we demonstrate an iOS mobile image-based equation solver using the built-in camera. The application is capable of solving expressions that include digits, the simple arithmetic operators (addition, subtraction, multiplication and division including both slashes and large line division), parentheses, and exponentials. All computation is done locally on the iPhone without the need to relay data to a server.**

I. Introduction

A major source of frustration when dealing with complex calculations is the time and effort cost of entering equations into a calculator or computer. When doing computational work, an individual must go through the laborious and time-consuming process of entering equations into a calculator or scientific computing program even after writing it all out by hand. Also, many potential users of these resources are unfamiliar with the syntax of these programs and cannot take advantage of the more advanced functions. Mobile apps such as 'MyScript' have proven the consumer desire for automated mobile calculators. However, these apps require the user to enter information on the screen instead of on handwritten paper and do not support more rigorous uses.

We limited the initial scope of the application to arithmetic expressions and avoided more complex operators in order to focus on and optimize the core functionality of acquiring quality characters, correct classification and equation structure reconstruction.

II. System Overview

The approach for acquring and solving an equation from an image is outlined below:

1) *Capture image*
2) *Convert to grayscale and blur*
3) *Binarize using adaptive thresholding*
4) *Segment characters*
5) *Normalize characters to dataset*
6) *Classify characters using SVM and heuristics*
7) *Construct equation and error check*
8) *Prompt user for confirmation or editting*
9) *Solve using Wolfram Alpha and display results*

Using the OpenCV library, the application is able to perform all of the steps except solving the equation locally on the mobile device. Even with relatively high recognition, it is important to provide the user the ability to edit and confirm the equation before it is evaluated.

III. Image Capture

When the application runs, the camera is locked into landscape view for image acquisition. A difficulty for the application is the automatic focus for the iPhone. The user needs to wait until the mobile device focuses and the equation becomes clear before capture. A blurry image will prevent the rest of the system from determining the written expression. The application takes the image in landscape view.

IV. Preprocessing and Normalizing

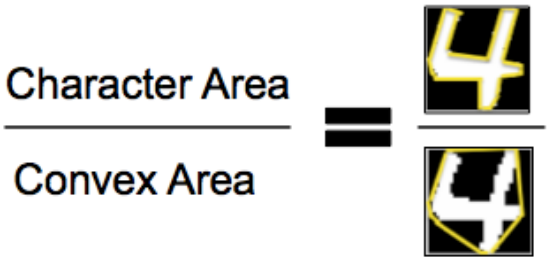
The objective of preprocessing is to take the captured image and segment the characters. The image is converted to grayscale and blurred to reduce lines on lined paper as well as noise. The image is then binarized using adaptive thresholding with different values for lined paper and plain white paper. This binarized image is then inverted and segmented using the findContours OpenCV function. To remove noise, only contours that have either a width or height of 10 pixels are kept. From these valid contours, any contours that are completely bounded by the bounding boxes of other valid contours are removed. This process ensures that inner contours (such as the circle inside a 6) are not classified.

After the characters are segmented, they need to be normalized to the dataset of characters. The dataset characters have a size of 28x28 with a stroke width of either 5 or 6 pixels. The objective of character normalization is to make any character with any stroke width match as much as possible to the dataset. To normalize, the characters are dilated to ensure a continuous stroke, stretched to 28x28 pixels and then thinned using the Zhang Suen algorithm. The thinned image is then dilated using a structuring element with a radius of 2. The acquired image now has the same stroke width of 5 or 6 as the dataset images.

Figure 1: (From left to right) Segmented character stretched to 28x28, thinned character using Zhang Suen algorithm, character dilated with structuring element with radius 2.

V. CHARACTER CLASSIFICATION

Polynomial support vector machine classification was chosen as the primary classifier of characters because of its high success on the MNIST digit dataset and its easy implementation in OpenCV. When characters are classified using a SVM, the features are simply the 784 pixels of 28x28 pixel image. To further improve the SVM, a simple decision tree is used with an SVM classifier at two of the bottom nodes. The first branch in the tree is based on the convexity of the character. The convexity is defined as the character area divided by its convex area.



If the convexity factor is high enough (>=0.74), the character is considered convex and will be classified to either '1' '/' '*' '.' '-'. The convex character it is classified to is determined by a nested decision tree based upon the moments of the character (ie '1' will have a large ν_{02} and '–' will have a large ν_{20}). '*' and '.' are classified based on their y-position to the next character. The cutoff point is one third of the height of the next character. Below that value it is classified as a '.' and above, it is classified as a '*'. For simplicity, users can only write '1' as a line and '*' as a dot. Removing these 'convex' characters from the digit classifer reduced misclassification of oddly shaped digits.

If the convexity factor is low (<0.74), the character is considered non-convex and reaches another node in the character decision tree. The character is searched for an inner contour. This node was introduced to primarily handle '8' being misclassified with '2' and '3'. The node has the effect of seperating out '8' from the classifier. If an inner contour is not found, the character is sent to a SVM classifier with all of the remaining characters excluding '8'. If an inner contour is found it is sent to a classifier with '0' '9' '8' '6' '4'.

After the characters are classified, the result is sent to a classification error checking stage. This stage uses structural heuristics to correct the most common misclassifications such as (4 vs +) and (5 vs 3). To correct between 4 and +, the number of pixels in the top left corner are counted. If not enough pixels are filled, the character is classified as a plus and vice versa. To correct between 5 and 3, the ν_{11} of the top third of the character is calculated. A negative moment suggests a 3 and a positive moment suggests a 5.

Figure 3: Example of classification error checking between + and 4.

Figure 4: The ν_{11} of a 3 and 5 used for classification error checking.

Other similar heuristics are used for additional classification error checking.

VI. EQUATION CONSTRUCTION

The equation structure is initially determined by sorting all of the characters by their x-coordinate centroid. The equation is then sorted to account for flat division lines and fractions. This process first loops through all classified characters and examines all characters that were classified as a '-' since these reflect possible division bars. The process then determines which of these are valid division bars by searching for characters below and above each possible division bar. Once a division bar is identified, the process then extracts all characters directly above and adds a '[' and ']' around them. Then, a '/' character is inserted and the characters below the bar are extracted and similarily wrapped in '[' and ']' characters. In this sense, the algorithm has locally sorted the characters directly above and below this division bar. This process continues over all division bars, locally sorting the characters directly above and below. Since the relative positions are maintained, this process sorts multiple tiered fractions (for instance a fraction in the denominator of another fraction) in a recursive manner.

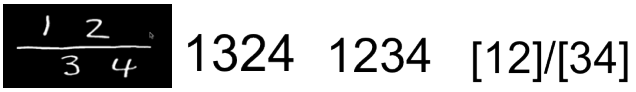


Figure 5: (left to right) The process of equation construction for flat division symbols.

The next step of equation construction is the insertion of exponents. The sorted equation is processed sequentially, character by character. If the bottom of the current character's bounding box is above the centroid of the previous character, a '^(' is inserted between the characters. When the current character's centroid is below the bottom of the previous character, a ')' is inserted. In order to deal with multiple layers

of exponents, the algorithm maintains a sorted list of the centroids of the characters at the beginning of each exponential layer. This list allows the algorithm to check if multiple layers of exponents are being dropped and thus requires multipled ‘)’ characters to be inserted.

This process also examines the classification of the current character and the previous character and avoids inserting exponents when there are logical incongruities. For example, after a decimal point the requirements for an exponent would often be filled but this would not be a valid equation so the algorithm does not attempt to add an exponent after a classified decimal point. Similarily, the algorithm does not try to raise operators as exponents (with the exception of a minus which could be acting as a negative sign).

VII. Dataset Construction

The MNIST dataset was used for the digit dataset. These digits were modified by finding a tight bounding box around the digit and then adding padding equally on the two sides of the smaller dimension. This image was then interpolated to a 28x28 square. It was chosen to not stretch these images to preserve the integrity and variability of the dataset.

The MfrDB mathematical symbol database was used for the additional symbols ‘(‘ ‘)’ ‘+’. These symbols were in the InkML format which is a series of x, y-coordinate samples for strokes of a stylus. This dataset has many other mathematical symbols that could be used to further expand the functionality of the mobile application. The MfrDB symbol set had to be converted into a comparable format to the MNIST dataset. The first step was to add the x, y-coordinates to an appropriate matrix. Adjacent samples that had the same x or y coordinate would be connected. A more robust version of this algorithm would connect each adjacent sample. These images were then sent through a similar process as the preprocessing normalization for acquired images. The images were dilated to connect pixels, stretched to 28x28, thinned and dilated again to achieve the same stroke width as the MNIST dataset.

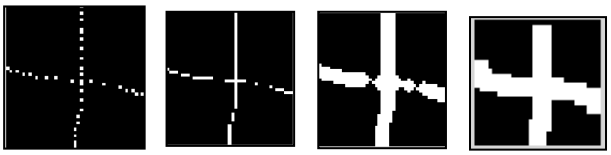


Figure 6: MfrDB conversion process (left to right). Original x,y-coordinates sampled, connect adjacent pixels with same x or y-coordinate, dilate to fill gaps, stretch to 28x28, thin, dilate.

About a third of the data from the MfrDB was removed for each character. This filtering process was based on the height to width ratio of the sample as well as the total number of samples. Extremely thin or wide images did not translate through the conversion process as well as images with few x,y-coordinate samples.

Subsets of these data were used to construct the two SVMs used in the classification. These subsets were run through a program that formatted them as row vectors into a matrix. A matrix of the same height was created as filled with class labels, where the label of row i matched the class of the image corresponding to the row vector of the first matrix. These matrices were then used to train the openCV SVM objects. Because this training was extremely demanding both in terms of time and memory usage, the trained SVM objects were serialized as XML files and then included in the iPhone application. This allowed the application to quickly read in the trained SVM objects and compute the classification without having to relay data to a server and not exhaust the limited memory available to such a mobile device during training.

VIII. Results

The application was tested on 19 well written equations from several individual writers. The equation consisted of over 17 characters including all digits, four operators, parentheses, decimal, exponents, and large divisions.

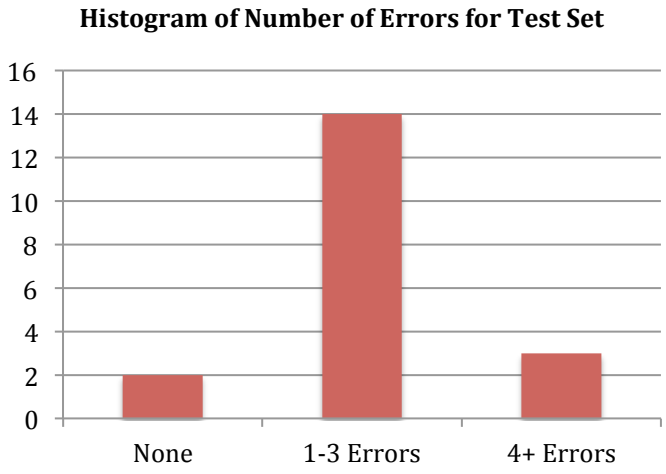


Figure 7: Histogram of number of errors from sample test equations.

A histogram method for the results was chosen because it parallels how a user of the application would evaluate its effectiveness. The number errors represents how much time the user will have to spend correcting the equation before it can be submitted for evaluation.

IX. Further Improvements

There are a number of improvements to the application that would improve classification accuracy as well as robustness and yield a more desirable user experience. The most prominent challenge to the application’s robustness is currently the lack of reliable thresholding. The ability to take pictures in different levels of luminosity and on different types of paper (bold lined, faded lined, graph, blank) demands a level of dynamic thresholding that the current algorithm fails to meet. Consequently, the current application is optimized for average lighting on faded-lined paper and thus only yields reliable

results in this context. A more advanced algorithm would possibly use color to detect the blue and red lines and remove them, allowing the subsequent thresholding to be more relaxed and not compromise the text in any lighting.

Another improvement that would significantly improve user experience would be the ability to detect the presence of a valid equation while in the camera view. This would allow users to only take a picture when there is a valid image that can be classified.

The user experience would be further improved by making the algorithm invariant to rotation. Currently, if the captured image is significatly rotated, each individual character will most likely be misclassified. By using an equation line detection algorithm, the application could detect rotation and re-orient the image to improve classification.

Another improvement that would yield a more desirable user experience and improve current classification methods would be a more intelligent segmenting algorithm. Currently, the application simply detects all the individual contours and only accepts non-bounded contours that fulfill certain size requirements. While this method is effective when taking an image of a single, clean image, it leads to poor classifications when the paper is not completely clean or parts of other equations or handwriting is present on the borders of the image. Ideally, a future algorithm would be able to detect the region and contours of the desired equation and classify only them.

Another improvement to the application would be the inclusion of more special characters (e, log, ln, x, y, =, derivitaves, integrals, etc.). The application would really become an effective tool with the ability to classify these characters and construct the appropriate syntax. Entering these complex equations into a calculator or another program, especially with calculus (i.e. the inclusion of integrals and derivatives), is a tedious and even difficult process because it often requires non-trivial formatting and syntax. This application could potentially eliminate this problem by translating the equation into the appropriate syntax for the user.

Misclassifying non-convex characters as convex could be improved, especially for characters with a thick stroke width. Currently a dilation of 1 pixel is applied to the image before segmentation to ensure that a character is not broken apart. When digits have an inner contour and a thick stroke width, this dilation can close the digit and remove the inner contour. This process creates a a thick, convex character that can be misclassified to a '1'. While one of the necessary conditions for convexity is that it has no inner contours, the dilation can remove the inner contours. A more robust implementation would check both the dilated character and the original character before dilation for inner contours before classifying the character as convex.

X. ACKNOWLEDGEMENTS

We would like to thank Professor Bernd Girod for excellent digital image processing lectures and a great sense of humor as well as the teaching assistants David Chen and Matt Yu for their help. Sam Tsai was a valuable mentor and resource specifically for this project and we really appreciate his willingness to meet with us.

XI. REFERENCES

[1] https://github.com/aptogo/OpenCVForiPhone/blob/master/OpenCVClient/UIImage%2BOpenCV.mm

[2] http://opencv-code.com/quick-tips/implementation-of-thinning-algorithm-in-opencv/

[3] http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

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

[5] http://mfr.felk.cvut.cz/Database_download.html

Appendix- Individual Work Breakdown

Scott Harvey	Acquisition, conversion and normalization of datasets
Will Harvey	iOS programming implementation
Both	Algorithms, testing, research