CONVERTING MATHEMATICAL EXPRESSIONS TO LATEX

David McNeil, Chris Sadler, Zizhao Wang

Rose-Hulman Institute of Technology

1. ABSTRACT

This project consisted of taking an image of a mathematical expression and converting it into LaTeX markup code for easy printing and viewing. The main motivation for this project was to ease the conversion of a written formula to a computer friendly format. It is much easier and clearer to write out an equation on a white board or piece of paper than directly typing the equation. Many people first write out mathematical formulas by hand and then convert them to a computer friendly format. Our project attempts to cut out the need for this additional step.

Our end goal was accomplished by segmenting an image into regions each containing an individual mathematical symbol, extracting features from the given symbols, classifying each symbol with machine learning, and outputting LaTeX markup based on classification results. The minimum requirements for the program were to classify the digits 0 through 9 and the operators =, *, /, +, and -. The success of our implementation was shown by testing our classifier on a set of 228 symbols. We were able to classify these images with a 97.67 % accuracy (much better than 6.67% random chance).

Keywords— OCR, machine learning, LaTeX, mathematics

2. INTRODUCTION

The problem of interpreting handwritten characters is not a new one; people have tackled the problem from different perspectives for quite a while now. Our goal for this project was to create a system that could interpret handwritten text in images of mathematical formulas and convert them into LaTeX markup code for printing and display. It's much more convenient and efficient to write out math equations rather than typing them, and many people are more comfortable writing out equations on paper or a whiteboard than attempting to setup and use LaTeX. Not much mainstream software attempts to tackle this problem, and we thought it would be an interesting problem whose solution would be very useful.

There are a couple challenging parts to this problem. First of all, the sheer number of mathematical symbols used makes this problem quite daunting. Because of the amount of symbols in use, finding training data to use to classify different symbols is equally challenging. Handwriting varies greatly between different people, and furthermore, many people don't write the same problem or even the same symbol in the same way every time. Glare from reflective surfaces turned out to be a major complication as well.

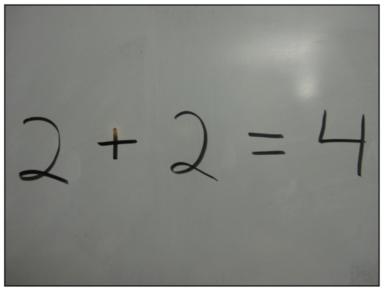

Fig. 1: How can we take this handwritten formula and convert it to a computer friendly format?

Our solution to the problem involves the use of feature vectors and multi-classification SVMs, specifically a 15-class SVM. These were used to classify images; based on a confidence value generated by each classifier in the SVM, the image was then classified based on which classifier claimed to be the most similar to the image. Our method of extracting digits from a handwritten text image was rather unique as well. Using a combination of several different image processing techniques we were able extract all of the characters from a board. ¡¡¡¡¡¡¡ HEAD

3. RESEARCH

=======

4. LITURATURE REVIEW

¿¿¿¿¿¿¿ 0d31e79823559b3ec500674ab64f763dd2a40e1c

Optical character recognition (OCR) has been the subject of much research through the past and provides a direct application of machine learning. We were able to harness the work done in the past in order to enhance the work done in our project.

Eikvil's work [1] not only provided numerous methods of feature extraction but also summarized the robustness and practicality of each algorithm. Using sensitivity to noise, distortions, style, translation, and rotation to quantify robustness and speed, complexity, and independence to measure practicality.

The paper strongly recommended using zoning and moments as features. Both of which we used in our project.

Hansen's work [2] specifically outlined implementing OCR in Matlab which provided a general model for our implementation.

"Historical Review of OCR Research and Development" [3] looks at two approaches to OCR: template matching, and structural analysis. The authors make the realization that these two approaches are overlapping and seem to be converging to a similar point. They propose a hybrid classification system using a combination of the two. We employed a similar technique by using zoning methods as well as properties of the image such as circularity or extent.

5. PROCESS

Our methodology can be broken into three steps: image segmentation, feature extraction, and classification using machine learning. At each step, we attempt to further summarize the information found in an image. Our end goal is to summarize the image of a mathematical expression down to a string representation of that same expression.

5.1. Image Segmentation

Our entire methodology relies on successfully taking a raw image and converting it to a number of consistently sized binary images. Each of these images can be thought of as a tile containing one mathematical symbol found in the original image. This process of segmentation can be divided into two steps: conversion to binary, and separation of unique symbols.

Converting an image to black and white or binary is essential to accurately extracting individual symbols and providing consistent input for classification. Because of this, we developed a robust three step process for converting a gray image, with dark writing on a light background, to binary. We will outline each of these steps by looking at converting a sample image (Fig. 2) to binary.

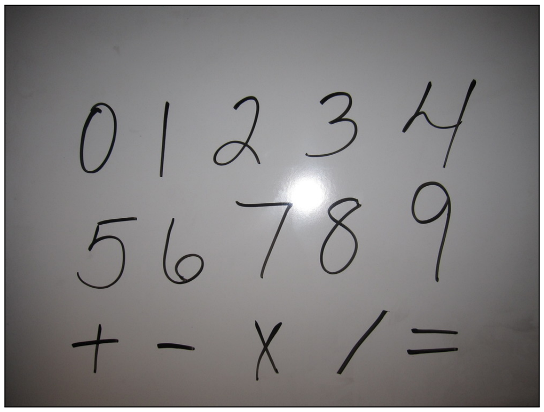

Fig. 2: Original Image

Fig. 2 presents many challenges for conversion to binary. It was clearly taken with a flash which results in a bright region in the middle of the image as well as creating a gradient to extremely dark regions in the corners.

5.1.1. Applying an Intelligent Threshold

We begin our approach by applying a dynamically calculated threshold to the original image. Pixels on either side of this threshold are cast to either black or white. The threshold is calculated using Matlab's built in `graythresh(BW)` function. `graythresh` uses Otsu's method to minimizes variance of the black and white pixel classes.

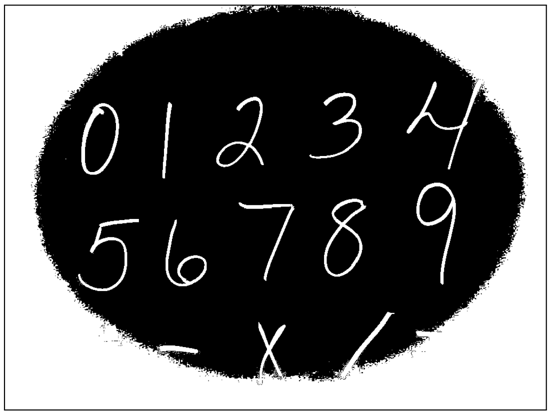

Fig. 3: Intellegent Threshold

The resulting image (Fig. 3) clearly produces poor results. The entire dark boarder is misclassified. This demonstrates that using a threshold method alone is not sufficient to properly convert the image.

5.1.2. K-means Clustering

The next step is to use unsupervised machine learning to determine the breakpoints of the two classes. This is done using the K-means algorithm in Equation 1.

$$minD = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - m_k||^2 \tag{1}$$

K is the number of clusters. In our case $K = 2$ one cluster for both white and black pixels. Each x is an individual pixel. C_k is the set of pixels in cluster k with k having a mean of m_k.

For this particular image K-means produced slightly better results than the threshold method (Fig. 4 compared to Fig. 3). However, we still misclassified a large number of exterior pixels.

Our first two methods of converting to binary have relied entirely on color. The problem with this method is that the dark boarder pixels have a very similar color to that of the written symbols. Our next method will look at changes in color in order to improve classification of symbols.

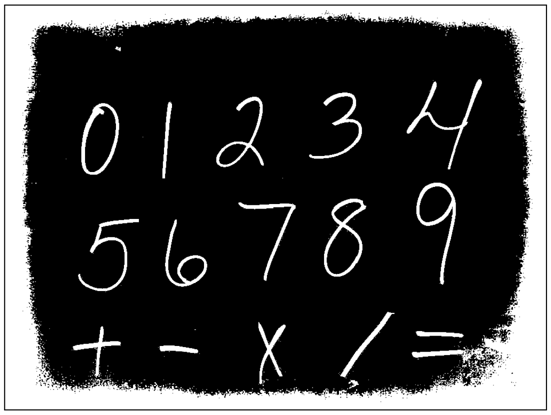

Fig. 4: K-means Classification

5.1.3. Edge Classification

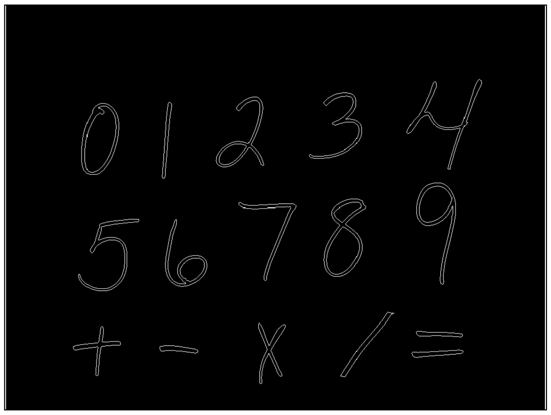

Fig. 5: Edges using sobel filter

Fig. 5 shows the results of running a sobel filter over the original image. The filter is able to perfectly distinguish the large changes in color from the whiteboard to the written symbols while ignoring the slow transition to darker color in the corners.

How can we use these edges to recreate the original symbols? A very natural thought is to simply dilate the edges until they become connected. The question now becomes how much to dilate the image.

Fig. 6 illustrates the distance of every pixel to the nearest edge pixel with white being furthest away and black actually being an edge pixel. We can then use this distance matrix to calculate the average linewidth of the symbols. This is done by extracting the local maximum from the distance matrix.

These local maximum represent approximately half of the line width. This is because these maximums occur when the distance matrix values transition from being closer to the exterior edge to the interior edge. We can then use the average of these local maximum to dilate the edge image.

The results of the edge classification seen in Fig. 8 clearly outperform either of the color based classification methods.

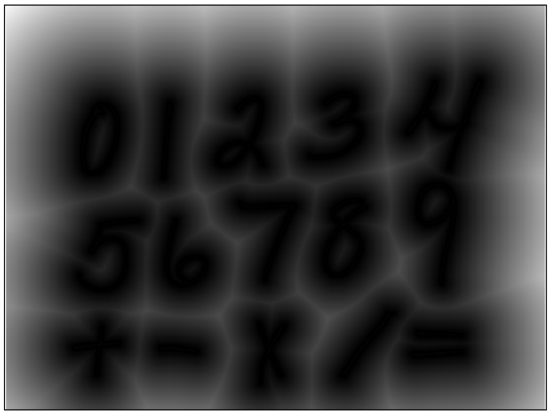

Fig. 6: Distance to Edge Pixels

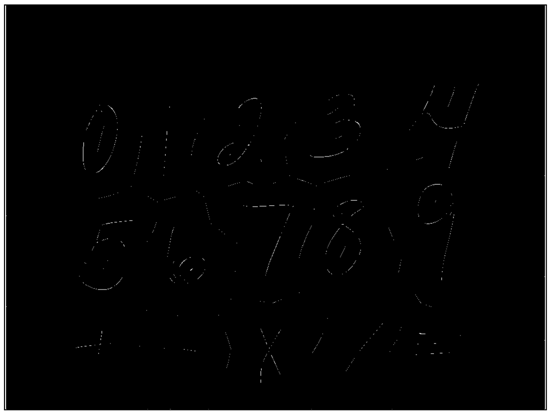

Fig. 7: Local Maximum of Distance Matrix

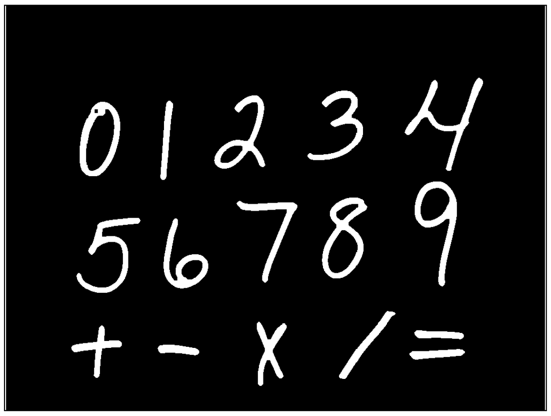

Fig. 8: Dilated Edge Image

However, the symbols lose some of their sharpness and become rounded due to the dilation. We can regain these features by using the results of converting to binary using the threshold or K-means algorithm.

We combine our results of the three methods by simply performing a logical "and" on the three resultant images. Simply "anding" the images together works because all three of the methods have a very high false positive rate and a very low false negative rate meaning they rarely do not classify the pixels that represent the written symbol correctly. The problem is they tend to over classify background pixels.

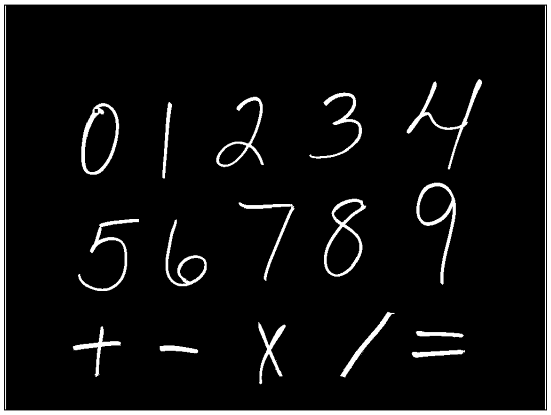

Fig. 9: Final Binary Image

Through trial and error we found that the threshold method and the K-means method each produced varied results based on image composition. Essentially, depending on the image composition one would outperform the other. Combining these two gave us an fairly accurate binary image based on color. We used this image to resharpen the detail that was lost in the edge classified image to due to dilation.

Once we have a binary image we can begin to actual segment the image into individual symbol tiles. Given our limited symbol set, the majority of this can be done through connected components analysis. If a set of pixels are all connected we simply classify this as a symbol and then extract the bounding box of that region. The only symbol this does not work for is the equals sign because it is a symbol made up of two disjoint regions. In order to handle this, we also combine symbols based on the distance of their centroids.

Finally, once we have these individual tiles we size them to 50x50 to provide a standard measure for feature extraction.

5.2. Feature Extraction

5.2.1. Circularity, Elongation, Orientation

Circularity, elongation, and orientation were used upon the recommendation of a few papers having success with these features in classifying characters and digits. Circularity is useful in our analysis because many symbols have a circular shape or at least a circular component to their shape. Elongation and orientation are useful to determine how stretched an image appears and what direction the image is rotated (if at all). This is helpful because most people's handwriting is not perfect and is likely elongated or tilted.

5.2.2. Euler Number

The Euler number corresponding to an image is defined as the number of regions minus the number of holes in the region. This is a very helpful feature to use when classifying symbols (digits and characters especially) because many have holes as part of their shape.

5.2.3. Solidity, Perimeter, Equivalent Diameter, Eccentricity

Solidity, perimeter, equivalent diameter, and eccentricity (along with a few others in this list) were taken as features mostly because they were directly pulled from the matlab `regionprops` command. All of these can be used as more identifying features about a given region that may help distinguish one group of images from another.

5.2.4. Mean, Standard Deviation, Skewness, and Kurtosis

These four features are standard statistical measures of data that provide information about the average of the data set, how far the data is spread from the average, how skewed the data is from a normal distribution, and how pointed or flat the data set is compared to a normal distribution. These were specifically run on the row and column data of the image to determine properties of the shape and relative position of objects as well as compare these features to other images.

5.2.5. Extent (Entire image vs. segments)

Extent is a measure of the percentage of an image that is filled with illuminated pixels. We used two different forms of this. Extent was first calculated on the entire image to determine what percentage was filled. This gives a relative measure of size that is homogeneous across images with different width and height but of the same character. Next, the image was divided into a three by three grid and the percentage of illuminated pixels relative to the total amount of illuminated pixels was calculated for each region. This was helpful in determining what portions of the image contained data and gave an overall idea of shape for a character.

5.3. Classification through Machine Learning

We extended Matlab's built in SVM (Support Vector Machine) for use with multi class problems. An individual classifier was trained for each symbol. Each of these SVMs was run on a symbol and their corresponding confidence rating was recorded. We then simply used the classification that was given with the highest confidence. Each of our SVMs used a radial basis kernel function (Equation 2).

$$K(x,x') = \exp(-\frac{||x-x'||^2}{2\sigma^2}) \qquad (2)$$

¡¡¡¡¡¡¡ HEAD =======

¿¿¿¿¿¿¿ 0d31e79823559b3ec500674ab64f763dd2a40e1c A large portion of the project was finding and creating training and testing data. In order to initially analyse the features we were extracting, and there success in classifying symbols, we extracted a 10,000 image training set and a 10,000 image testing set from the "MNIST handwritten digit database[1]." We were able to successfully classify these images with an accuracy of 91.3%. However, we had a very difficult time converting our handwritten symbols to the same format found in the MNIST database. This is primarily due to the extremely small size (25x25) of the symbols. Another, glaring problem with the database is that it only includes integers and not other mathematical symbols.

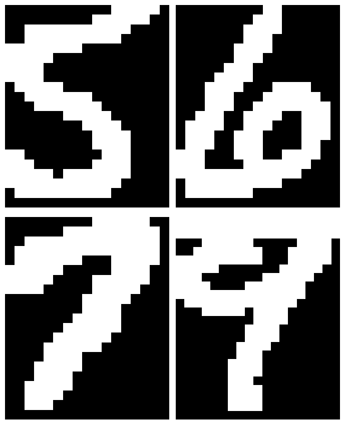

Fig. 10: Sample symbols (5, 6, 7, 9) from the MSNIST database

We devised our own set of 863 training and 228 testing images. Using this set of images we were able to classify with a 97.67% accuracy. The classifier trained using this set of images is what was ultimately used in the final product.

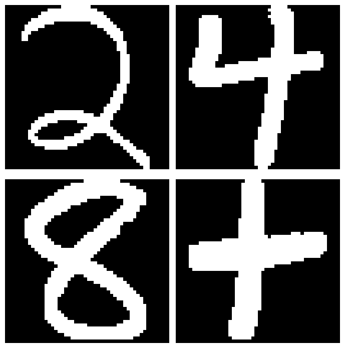

Fig. 11: Sample symbols (2, 4, 8, +) from the MSNIST database

6. CONCLUSION

Our system successfully interpreted the equations seen in the following images.

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

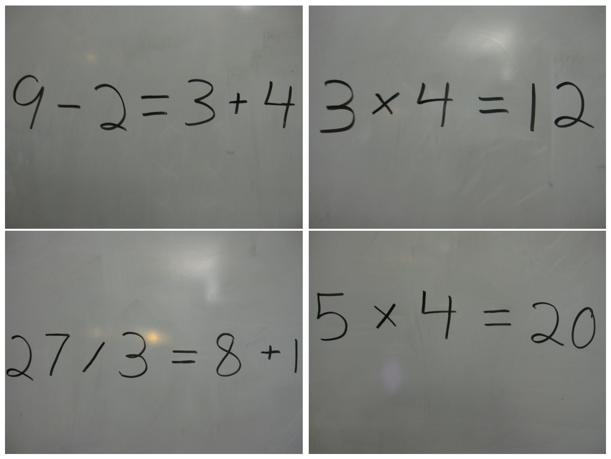

Fig. 12: Successfully interpreted images

Clearly these images are not overly contrived. They represent a variety of variably sized symbols located at different locations in the image. We had excellent success when we consistently used these large block symbols.

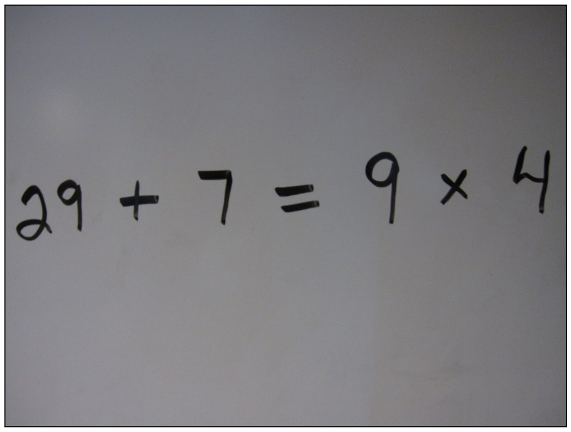

Fig. 13: Unsuccessfully classified image. Result: (5+7=9*4)

However, our system certainly has restrictions on its robustness. For instance, the image seen in Fig. 13 was misclassified because the 29 was grouped together as one symbol. This essentially made the resulting classification nonsense.

As seen increasing the robustness of image segmentation is certainly a portion of the project that needs improvement. Given two to three weeks more to work on a project this would certainly be an area of focus. We would need to devise some systematic approach that would also allow for the easy addition of new symbols. Our ultimate long term goal would be to provide support for a very large set of mathematical symbols and create an intuitive method for each user to train the classifier with their own handwriting.

7. REFERENCES

[1] Eikvil, Line. “Optical Character Recognition.” Norsk Regnesentral. `http://www.nr.no/~eikvil/OCR.pdf,December1993.`

[2] Hansen, Jesse. “A Matlab Project in Optical Character Recognition (OCR).” `http://www.ele.uri.edu/~hansenj/projects/ele585/OCR/OCR.pdf`

[3] Hunji Mori, Ching Suen, Kazuhiko Yamamoto. “Historical Review of OCR Research and Development.” IEEE. `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=156468,1992.`