Automated Handwritten Math Solver

Group 18 - Project Report
ECS 171 Machine Learning S23

Teaden Gurung, Faizan Haque, Brian Tom, Gordon Tze

Github repository:
https://github.com/tsepten/mathExpressionSolver

Code execution:
https://youtu.be/Dm57jEr4Frk

Department of Computer Science
University of California, Davis
May 30, 2023

Introduction and background

Mathematics equations and expressions provide a structured approach to problem solving and decision making. Simple calculations are integrated in society on a daily basis. These calculations may be simple, such as addition, subtraction, multiplication, and division, but they play a crucial role in day to day interactions. Take for example, a quick exchange between a cashier and customer, where the total cost is added up from each of the products. These math equations and more advanced ones foster critical thinking, logical reasoning, and analytical skills, helping us to understand and interpret data, make predictions, and solve everyday problems.

The main focus of the project is on solving handwritten mathematical problems. Through image detection, image classification, symbol parsing, and equation solving, our model solves simple math problems including simple addition, subtraction, multiplication, division, linear equations, and can ideally be extended to math involving polynomial equations, exponents, roots, integrals, and more. Our model works by parsing through the image to predict the written equation. When detecting the image, it must consider the symbol representation, segmentation between each symbol, classification of each segmented symbol, expression parsing. Then, the parsed equation is put into an equation solver and the final result is outputted. Here is an outline on the process:

- Image segmentation: The model first processes images as necessary and segments the image into different components (numbers) so that each segment can then be fed into the image classifier.
- Image classifier model: The image classifier model is trained on the combined Modified National Institute of Standards and Technology database (MNIST) and symbols dataset. This image classifier should be able to convert a single image into its symbol form.
- Symbol parser: The symbol parser reads each symbol outputted from the image classifier and turns it into a formal equation that can be solved in the equation solver. It should additionally do equation validation to ensure that it can be solved. Else it will return that the equation or expression cannot be solved.
- Equation solver: This solver will read the equation given from the symbol parser and output a result. This part could use an external library.

Proposed methodology

The hand-written math solver is a type of multi-nominal classification problem where each number and symbol is put into its own class. After discussing strategies, we decided that the project could be divided into four key steps: (1) **Image segmentation**, (2) **Image classification**, (3) **Expression parsing** (4) and **Expression solving** shown in figure 1.

Figure 1: High-level Methodology of Project

Firstly, image segmentation involves initial image processing to produce a grayscale blurred image, and then Otsu's method is utilized for image thresholding, a technique to optimize class variance. Noise reduction is achieved by this process, with each image labeled and segmented for subsequent analysis. Contours detection is used to segment the labeled images based on image curves, and border detection identifies image edges and surrounding padding and is additionally used for expression parsing for square roots.

Considering prior work in this field, we selected a Convolutional Neural Network (CNN) for model training, typically employed for computer vision tasks including handwriting recognition. CNN's proficiency in low-level feature detection (edges, textures) is evidently successful in learning of higher level features for identifying symbols and digits in handwritten expressions. We explain more in the literature review.

Our CNN model will classify segmented images into digits or symbols. Key hidden layers are the convolutional and max pooling layers, with learning mainly occurring in the convolutional layer when testing data is inputted. Max pooling layer reduces node count within layers, and the last hidden layer is the Rectified Linear Unit (ReLU).

As for post-identification, the symbols are parsed and processed into something that is solvable. This step would involve understanding square roots and brackets and their relationship to nested elements. We then lastly input our expression into the symbolic computation library, SymPy, which is suited for solving parsed mathematical expressions.

Literature and Article review

Recall, we seek to build a four step process. The most difficult portions are image processing and image segmentation. Both of these areas are popular areas in machine learning that have much literature work done on them. We read through numerous articles and papers to understand how we can implement our own image processing method and segmentation model.

Yann LeCun et al. made a significant contribution to the field of computer vision with their groundbreaking paper on Convolutional Neural Networks (CNNs). Their work revolutionized

the entire field by introducing a powerful and effective approach to visual recognition tasks. The paper laid the foundation for the widespread adoption of CNNs and sparked numerous advancements in computer vision, leading to significant breakthroughs in image classification, object detection, and image segmentation.

Lokhande et al. developed a CNN model to achieve 98.46% accuracy in recognizing handwritten digits and mathematical symbols. Their workflow involved preprocessing the image into grayscale, segmenting each line and character, and training the model using MNIST dataset, which includes handwritten digits 0-9 and three mathematical symbols. However, they aim to enhance the model's ability to identify connected digits or symbols that cannot be properly segmented.

Similarly, Shinde et al. addressed the task of training a model to solve handwritten math equations. Their dense model achieved 98% accuracy in solving simple equations, while the Recurrent Neural Network (RNN) model achieved 96% accuracy. However, the CNN model struggled with predicting complex equations, with only around 85% accuracy. Predicting polynomial equations posed a greater challenge compared to simple arithmetic equations. To further enhance our expression solver's capabilities, we could consider the utilization of RNNs as a potential improvement.

In addition to research through academic papers, we extensively explored various documentation sites and articles. Specifically, we examined multiple implementations of image processing projects utilizing OpenCV's libraries. This enabled us to develop a comprehensive understanding of how we could apply and customize image processing techniques to our own project. Additionally, we delved into learning how to utilize data augmentation techniques with the help of Keras libraries. These efforts broadened our knowledge and provided valuable insights into enhancing our project's capabilities.

Dataset Description and Exploratory Data Analysis (EDA)

The HASYv2 data set contains over 150,000 different images of handwritten mathematical symbols. It contains 168,233 instances of 369 classes and each mathematical symbol is 32 pixels by 32 pixels. This data set will be used to train the model to recognize individual numbers and symbols

After dropping some of the unneeded symbols, we digits from 0 to 9 and arithmetic operators, and the square root symbol. We additionally wanted to create brackets to perform nested operations and hence we created the left and right bracket manually of around 60-100 samples. The distribution of the remaining dataset of numbers and symbols are shown in figure below. Our motivation for choosing this dataset lies in its inclusion of mathematical symbols.

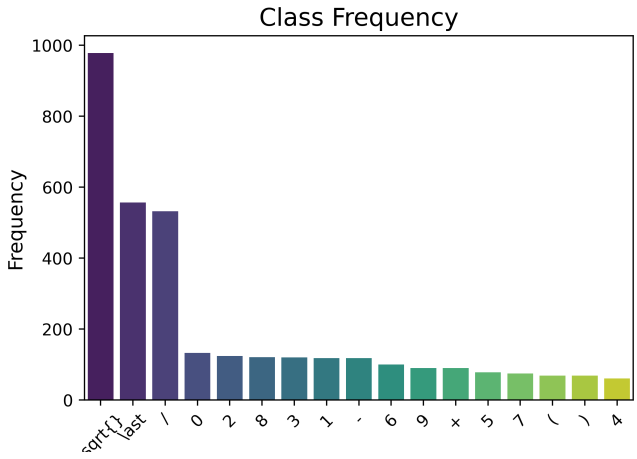

Figure 2: Distribution of number labels in HASYv2 Subset

We observed and understood that this bias in samples for certain classes could lead to a biassed model and thus we attempted to reduce sample size in the square roots and increase sample sizes of numbers through data augmentation. Lastly we performed more EDA by seeking to understand the average of the images shown in figure. This combined with figure 3 gave us an understanding of which numbers may be more likely to be confused with others.

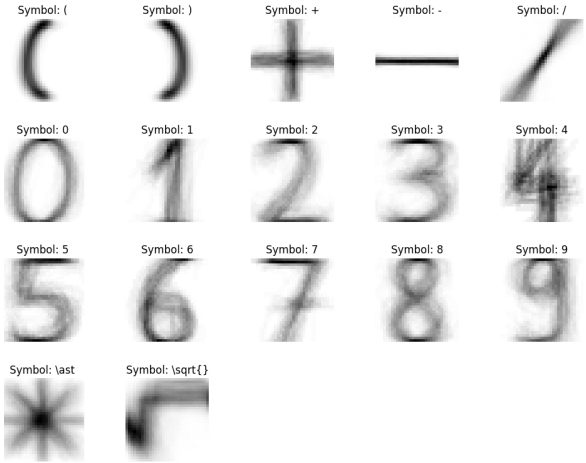

Figure 3: Mean of HASYv2 interested symbol dataset

Experimental result

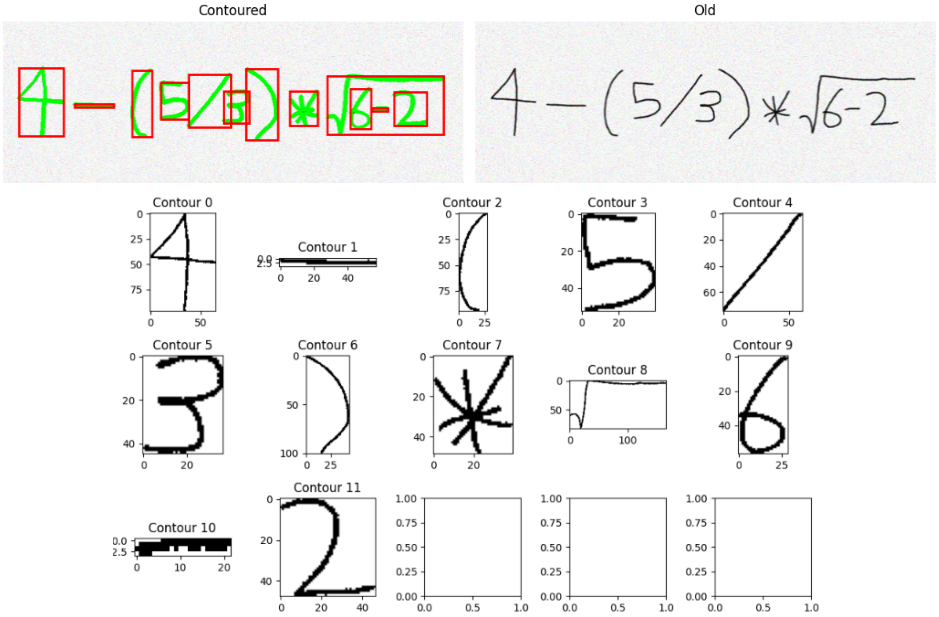

Figure 4: Contouring for images

The first task is to segment the image to be put into the CNN model that we are training next. Initially, we used traditional image thresholding methods to segment the image which failed to consistently segment the image especially with slightly noises images. Sometimes the segmented result would be correct but failed on longer expressions and variables. Our next attempt involved processing the image into black and white, utilizing Otsu's thresholding and Gaussian blurring, and lastly image contour detection. When these were combined, our image segmentation worked well. Thresholding is used to reveal the characters and finally we find the contours and get bounding box for each contour. In Figure 4, the right side labeled "Old" shows the original inputted image and the left side labeled "Contoured" shows the image segmentation result. The segmented image is separated into 11 different contours where each of them represents a mathematical symbol or number. These images will be put into the image classification model.

The second step is to build our image classification model. We attempted to build this using a very simple random forest model. Though we made some progress with it, ultimately, the model's training took longer and also did not work as well on some of our own custom inputs. Our second idea was to utilize a CNN. We played around with tons of parameters and different choices of layers for both max pooling and convolutional layers. We additionally used data augmentation to generate new samples when training our model. Our samples would be ever so slightly rotated, zoomed, and shifted. This was to account for the low sample count we had in our subset of the dataset. Additionally, to avoid overfititng, we incorporated both

dropout and early stopping with a patience of 5. Our final architecture is shown in the figure. Additionally, we show the accuracy and loss through each epoch.

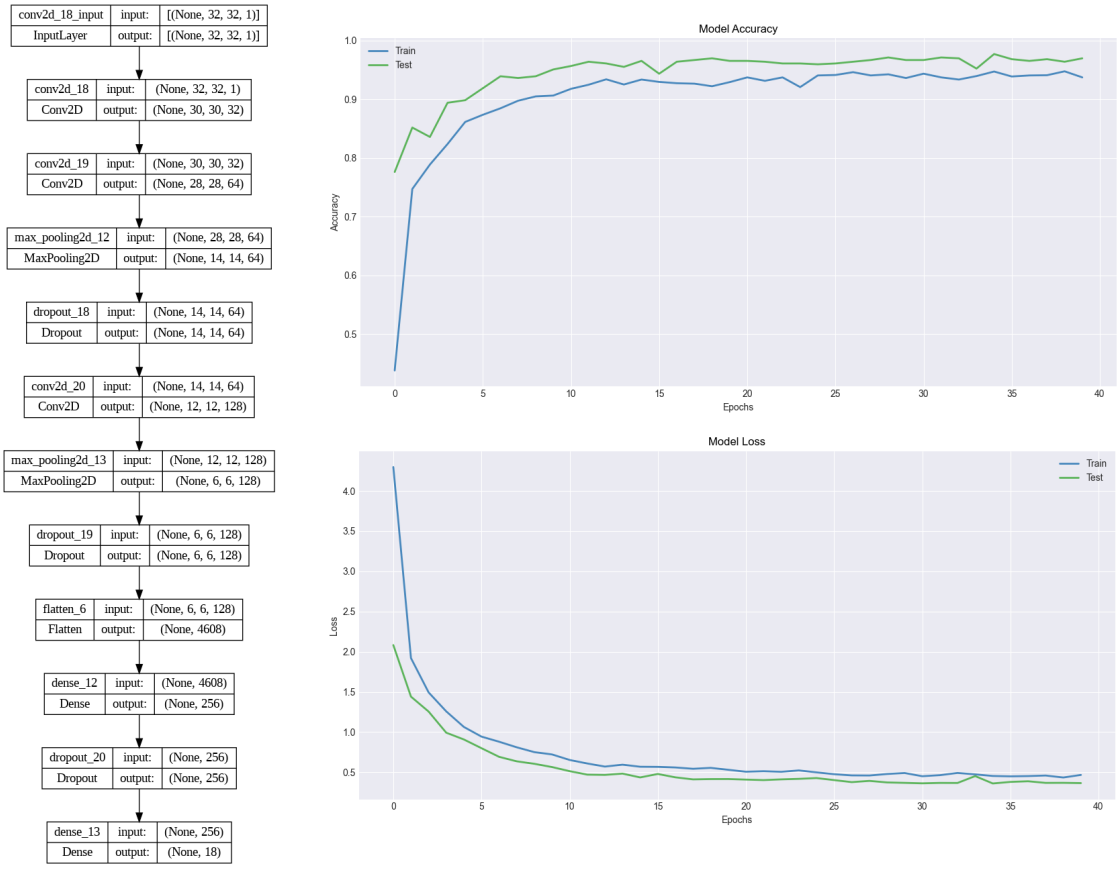

Figure 5: CNN Architecture and training process

The conv2d layer extracts local features and saves spatial patterns. It is mainly used for detecting edges and textures for our model. Max pooling 2d is used after the conv2d layer to reduce the dimensions or nodes in the next layers. It still keeps the important information in the nodes but reduces the dimension. Dropout is to prevent over fitting in the neural network when too much data is passed through the network. Flatten changes the multi dimension input into a 1 dimension vector. It maps each component in the previous layer into a 1 dimension vector and is usually to prepare the data for the next layer. Dense is a fully connected layer which connects each neuron to every other neuron in the previous layer. This layer is for classification and has an activation function to produce an output. We achieve an accuracy of around 98% on the test set and we show our model evaluation metrics below.

Classification Report for Test Set

	precision	recall	f1-score	support
(	1.00	0.87	0.93	15
)	1.00	1.00	1.00	16
+	0.89	1.00	0.94	17
-	1.00	1.00	1.00	15
/	1.00	0.97	0.99	110
0	0.96	0.96	0.96	27
1	0.93	1.00	0.96	25
2	0.96	0.96	0.96	28
3	0.88	0.95	0.91	22
4	0.94	0.88	0.91	17
5	0.93	0.82	0.87	17
6	1.00	1.00	1.00	18
7	1.00	0.95	0.97	19
8	0.95	1.00	0.98	20
9	0.87	0.93	0.90	14
\ast	1.00	0.99	1.00	120
\sqrt{}	0.99	1.00	0.99	187
accuracy			0.98	687
macro avg	0.96	0.96	0.96	687
weighted avg	0.98	0.98	0.98	687

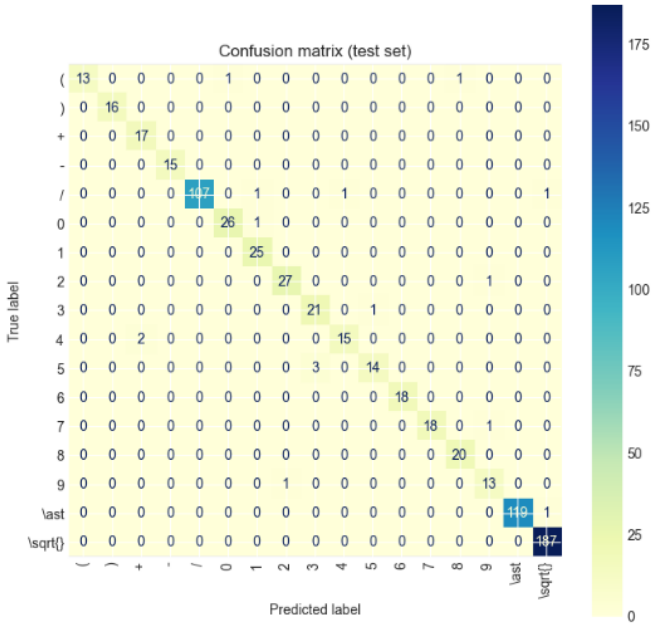

Figure 6: Model Evaluation for CNN

Finally, after all the segmented labels are classified, we need to combine all the classifications together and solve the equation. The expressions need to be parsed the solved. The CNN model classifies mathematical symbols in latex form which means that we have to convert the text back into its original symbol before we put it into the solver. Based on the bounding boxes of the segmented images, the mathematical expression (square-root or exponent) bounds all the numbers and symbols in between. We have to append parentheses in the array to correctly parse the expression or equation. Then, the mathematical expression is put into the solver using the SymPy library. After getting most of the back end running, we decided to create a simple front end using JavaScript, HTML, CSS to display the original expression, contoured expression, segmented terms, and result. We added features to upload an image of the expression or draw the equation directly into the browser.

Conclusion and discussions

In conclusion, our project aimed to develop a model for solving handwritten math problems through image detection, image classification, symbol parsing, and equation solving. We followed a four-step methodology, including image segmentation, image classification using a Convolutional Neural Network (CNN), expression parsing, and equation solving using the SymPy library.

During image segmentation, we utilized techniques such as grayscale conversion, Otsu's thresholding, and contour detection to separate the symbols and numbers in the image. The CNN model was trained on the HASYv2 dataset, which contains over 150,000 handwritten mathematical symbols. We experimented with different CNN architectures, and our final model achieved an accuracy of approximately 98% on the test set.

After classifying the symbols and numbers, we performed expression parsing to convert the symbols back into their original form. We also added support for square roots and parentheses to handle nested expressions. The parsed expression was then passed to the equation solver, which used the SymPy library to solve the mathematical equation and produce a result.

We encountered some limitations and challenges throughout the project. The model's accuracy was affected by the quality of the input image, and symbols such as the '7' and '2' or '1' and '/' were occasionally mistaken for each other. The limited sample size in our dataset for labels also posed challenges in accurately classifying certain symbols which we tried to deal with via data augmentation. Despite these limitations, our model showed promising results for simple math problems.

In the future, we plan to extend our model to handle more complex mathematical equations, such as polynomial equations, exponents, and integrals (e.g.: $2x^2 - \int_0^1 x dx = 0$). We also aim to address the challenges posed by noisy backgrounds and the presence of other objects in the image by utilizing a more advanced machine learning based segmentation model such as U-net. By continuing to refine and enhance our model, we hope to create a powerful tool for solving handwritten math problems.

References

Dataset:
HASYv2: https://www.kaggle.com/datasets/guru001/hasyv2

Literature Review:
Lecun Y., Bottou L., Bengio Y., and Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324

Lokhande M., Murudkar K., Nadaf A., Shah N. (2021, May). Handwritten Math Problem Solver Using Convolutional Neural Network. International Journal of Advanced Research in Computer and Communication Engineering

Shinde R., et al. (2002, February). Handwritten Mathematical Equation Solver. International Journal of Engineering Applied Sciences and Technology