Using Deep Learning to Recognize Handwritten Mathematical Expressions

CAS 771 Project

Jean Lucas Ferreira

April 2018

Abstract

Classification of handwritten mathematical text is a challenging problem to take on, due to the large number of scenarios that must be considered. There currently exists a few applications that focus on solving this problem. Each application approaches the problem from a different point of view, and has its own output goals. The goal of this project is to detect offline handwritten mathematical expressions and convert them to LaTeX. We use a convolutional neural network to help with symbol classification and apply a heuristic to detect nested operations such as square roots. The proposed solution achieves high accuracy when trained with a small set of mathematical symbols.

1 Introduction

Having the ability to efficiently transform a page of handwritten notes into a digital medium is useful in a variety of different use cases. Such as digitizing the works of old research into an online library to allow indexing and searching with ease, or being able to provide real-time feedback while you solve certain mathematical problems by hand.

There are two main branches in the handwritten text recognition problem, offline recognition, and online recognition. Systems implemented with offline recognition use static images as input, and the only information available is each pixel's color intensity and position within the image. In online recognition, systems are given two input dimensions, position and timing, it knows the position of each pixel relative to the entire image, as well as the order in which each pixel was drawn.

Furthermore, the act of text recognition can be broken down into the following subtasks: (i) image segmentation, (ii) symbol recognition, and (iii) structural analysis [15]. Image segmentation allows each symbol in the image to be extracted into its own image and be processed as desired. Symbol recognition is deployed to classify each segmented symbol into its respective class. Lastly, structural analysis allows us to construct a logically and syntactically correct output once all the symbols are have been classified. There are a variety of different algorithms to use for each of these steps, choosing the correct algorithm is a challenging process, and is dependent on the project's use case and scope.

The purpose of this project is to recognize offline handwritten mathematical expressions and transform it to a digitized medium such as LaTeX. This is accomplished by applying deep learning techniques for symbol recognition, using a bounding-box algorithm for image segmentation, and a heuristic for structural analysis. The scope of valid symbols include binary operators, whole numbers and square roots.

In section 2 the related works will be discussed. Section 3 provides a system overview. Section 4 goes over implementation details. The experiment approach is given in Section 5. Finally, a discussion and conclusion are followed in sections 6 and 7.

2 Related Works

Inspiration for this project initially came from the web application *Detexify* [6], which allows users to draw a math symbol and get the corresponding LaTeX keyword for that symbol. Similar works for full expressions have been applied in *Mathpix* [1] and *Photomath* [2]. *Mathpix* converts images of equations into a LaTeX string, and *Photomath* solves hand written mathematical problems in real-time.

Due to the increase number of publicly available data, it has become more viable to use deep learning approaches for image classification in this field of work.

In [12], the authors used a CNN with the LeNet architecture, since it is a well established architecture for handwritten digit recognition. Their training data was combined from three different sources: the MNIST database of handwritten digits, the Chars74K for English letters, and write-math for mathematical symbols, for a total for 101 classes. For image segmentation, they used the Efficient Graph-Based Image Segmentation algorithm [4]. Their approached was able to achieve 81.5% accuracy.

The authors in [15] put their focus on the the symbol recognition step. For this task, they used BLSTM-RNN, a combination of bi-directional RNN with LSTM. They make use of the CROHME 2013 dataset [5] (a dataset of online handwritten mathematical expressions), by converting each symbol to its offline equivalent. They proceed to use a variety of offline features for evaluating image classification accuracy. Some of these features include Polar features, FKI features, and an offline feature used by the Pattern Recognition and Human Languages Technologies group. Their accuracy results ranged from 80% to 87% for the Top 1 scores.

Recognition of online handwritten expression can also follow similar ideas in image segmentation and classification as seen in [8]. Their approach used a heuristic for image segmentation based on intersecting traces, and a CNN for classification. They argue that CNNs are an effective approach in this problem, and can outperform other approaches such as SVM.

3 Background & System Overview

A simplified set of states for this system is given in Figure 1. The proceeding sub-sections will highlight the main goal of each step. Implantation details for each step will be given in Section 4.

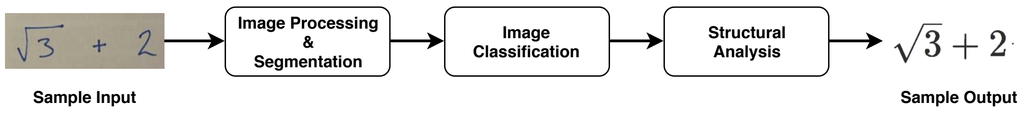

Figure 1: Simplified System Overview

3.1 Image Processing and Segmentation

Image segmentation is the process of extracting components from an image based on some desired criteria. In this scenario, it is the process of extracting symbols, numbers and letters from the input image, and be able to process each accordingly. There are many challenges involved in segmentation, in this case, such challenges comprise of symbols made up of multiple components (ie: $=, \leq$), or when two separate symbols are touching (as often seen in cursive handwriting). Different approaches may be used to deal with such situations. Due to time constraints, this project considers only single component symbols and non-touching symbols.

After segmentation, it is often desired to process these images in order to conform to a certain size or some requirements. This can include, for example, image re-scaling, re-sizing, centering, and blurring.

Figure 2 provides an example of segmenting multiple symbols from an image, and processing each segment to be centered, and of equal dimensions.

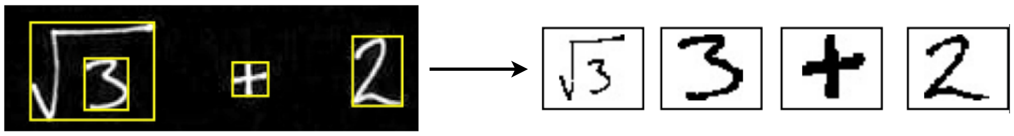

Figure 2: Image segmentation and post-processing example

3.2 Symbol Classification

Symbol classification allows us to match a segmented image into one of the possible output classes. As discussed in the Section 2, there are many approaches to this step, the most popular being deep learning techniques using Convolutional Neural Networks (CNN) [12, 8], Recurrent Neural Networks (RNN) [15], Long Short-Term Memory (LSTM) [15], Hidden Markov Models [8], and Support Vector Machines (SVM) [11].

In this project, a CNN model is deployed for the task of symbol classification, since the input to this model will be images of uniform size, and the expected output will be one of the possible output classes.

3.3 Structural Analysis

This step is applied in order to organize all classes that were classified in the previous step, into a logical output. A simple data structure can be used to preserve the left-to-right order of operation, and provide a way to handle nested operations due to brackets and square roots. Figure 3, shows that a tree data structure may be used to deal with these requirements.

Thus, once all symbols have been classified, the output can be generated by a tree traversing algorithm.

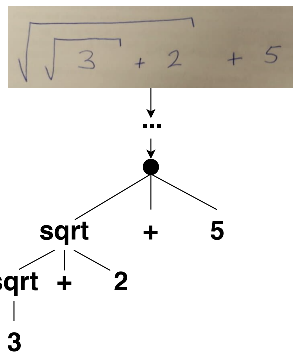

Figure 3: Tree data structure for Structural Analysis

3.4 Training Data

The HASYv2 dataset [13] is a free dataset of various handwritten digits, letters, and mathematical symbols and operators. The format of the images is similar to the MNIST dataset [7] of handwritten digits. The HASYv2 dataset consists of 168233 images of 369 classes, and each image is labelled by its corresponding LaTeX keyword. A set of example images in given in figure 4.

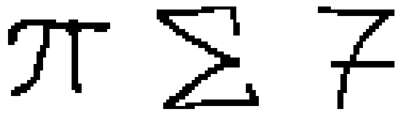

Figure 4: Three sample images from HASYv2 Dataset

4 Implementation

This project is implemented through the use of Keras [3] (with Tensorflow), and MATLAB's Neural Network and Image Processing Processing Toolboxes [10, 9]. Figure 5 gives an overview of the work-flow between each component.

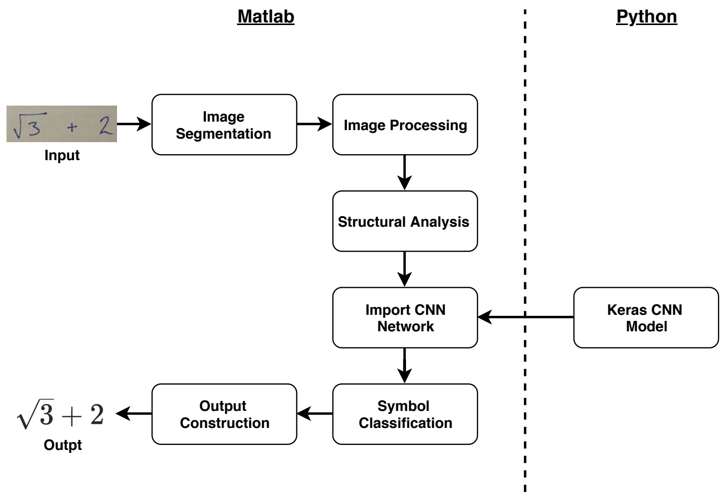

Figure 5: Work flow from input to output

4.1 Image Segmentation & Processing

Each symbol is segmented from the input image by using a bounding box approach. This creates a minimum enclosing rectangle around each region of the image. In this context, region refers to a single shape found in the image. For example, if an image contains the = sign, then two bounding boxes will be generated for the top and bottoms bars.

MATLAB's Image Processing Toolbox provides a set of features that easily allows the identification of these bounding boxes.When this is applied to an image, it provides a list containing the top left coordinates, width and height of each bounding box. This allows us to crop individual images for each regions. The left side of Figure 2 shows a sample output from MATLAB.

After cropping each bounding box, the resulting crop is as big as the region it encloses. Therefore, image processing is necessary in order to make each crop uniform to the specified requirements. In this case, the requirements for each cropped image is: white background and black regions, 32 pixels high, 32 pixels wide, and regions should be centered with equal padding on all sides. A sample output, after segmentation, can be seen on the right side of Figure 2. As we can see, the number three appears twice in different circumstances, since it is enclosed in two different bounding boxes. This will be handled during structural analysis.

4.2 Convolutional Neural Network

The initial CNN architecture was implemented by the Keras [3] team, and it was designed for the MNIST [7] dataset. Since the image data used for this project closely resembles the image data used in MNIST, it was a good starting point.

Figure 6 shows the CNN architecture after tweaking the original model in order to accommodate with this project's dataset.

As seen from the last layer of the CNN architecture, there is a total of 22 possible output classes. This set includes:

- digits 0 through 9
- characters: a b c
- operators: $+$ $-$ $\times$ $/$
- symbols: π α β $<$ $>$

Discussion on accuracy, training size, and testing size will be discussed in section 5.

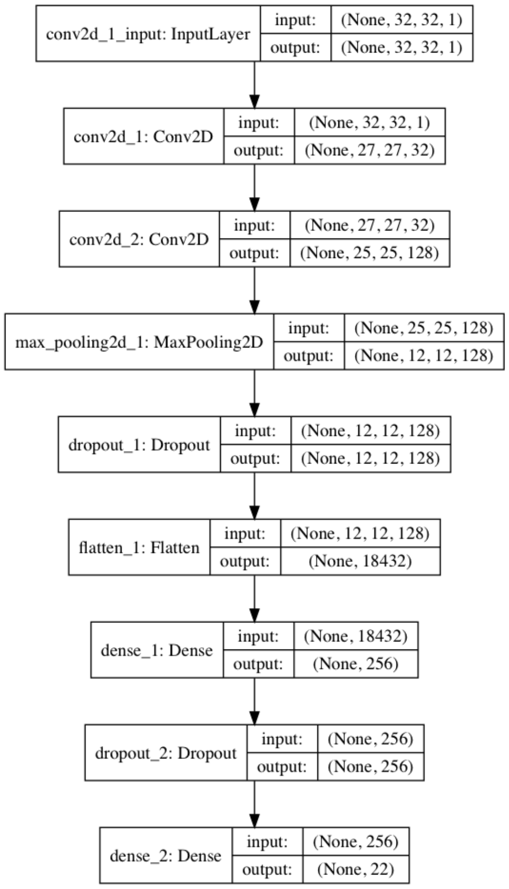

Figure 6: CNN Architecture

4.3 Structural Analysis & Symbol Classification

Once each symbol is segmented and processed, they are saved into a tree data structure in a way that preserves the order of operations from left-to-right, and appropriate nesting. An example of this can be seen in Figure 7.

It is important to note that each node in this tree represents a segmented image. At this point, the segments have not been classified by the CNN model.

The generation of this data structure uses a heuristic approach to generate appropriate square root nesting. Once we are given the set of bounding boxes found in the image we identify square roots by detecting whether a bounding box contains one or more bounding boxes inside. This behaviour is seen in Figure 7.

Therefore, whenever a node in the tree that has one or more child nodes they are automatically considered to be a square root, thus, these intermediate parent nodes are not fed into the CNN model for symbol classification.

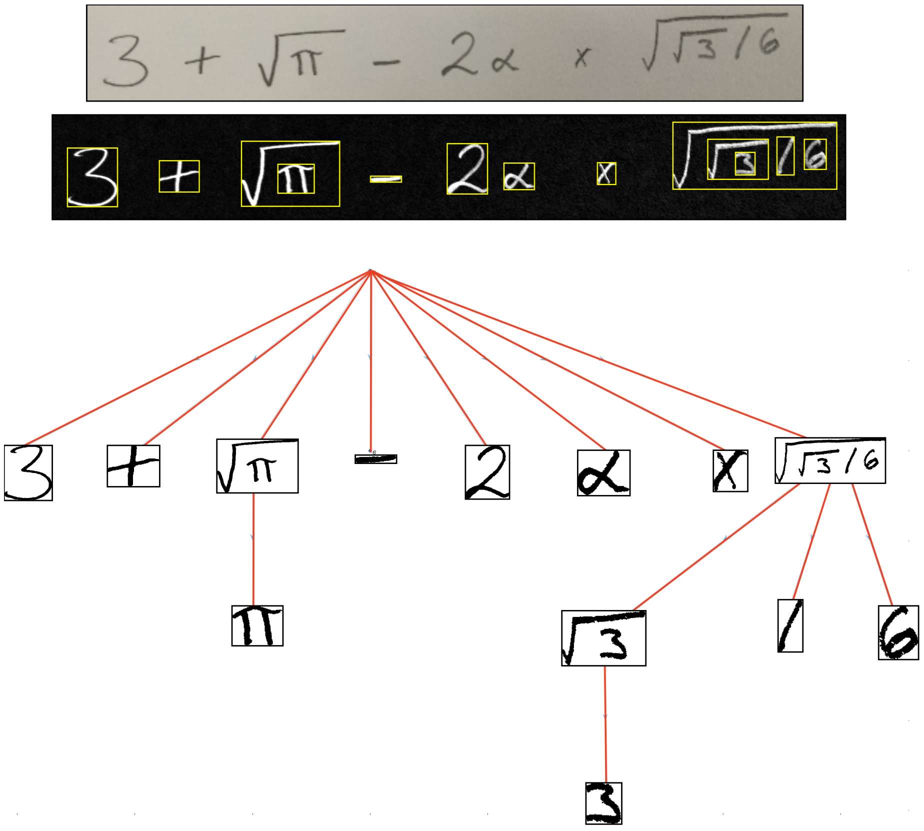
Figure 7: Sample input represented by tree-like data structure

All other nodes that are visited during the traversal algorithm (Listing 1) are fed into the CNN to get the corresponding output class. The output provided by the CNN can be used to fetch the corresponding LaTeX keyword for the given symbol.

The above example in Figure 7 will produce the following output after the traversal:

```
3 + begin-sqrt \pi end-sqrt - 2 \alpha \times begin-sqrt begin-sqrt 3 end-sqrt  / 6 end-sqrt
```

Which can be parsed to the appropriate LaTeX equation:

$$3 + \sqrt{\pi} - 2\alpha \times \sqrt{\sqrt{3}/6}$$

Listing 1: Tree Traversal Algorithm

```
function traverse(graph, root, node, output):
  child_list = succesors(graph, node)

  if isempty(child_list):
    symbol = cnn.classify(node.image)
    output.append(symbol)

  else
    if node != root:
      output.append('begin-sqrt')
    end

    for child in child_list:
      traverse(graph, root, child, output)
    end

    if node != root:
      output.append('end-sqrt')
    end
  end
end
```

5 Experiments

When trained under a large set of output classes, and does not take into consideration uniform sample sizes, the CNN behaves poorly. It announces it has high accuracy, but when tested with during experimentation many symbols are detected wrongfully.

This was resolved by reducing the number of output classes and applying a simple random evaluation, that limits the training sample size to 950 labels.

The final CNN configuration, training sizes and validation sizes are given below. The learning rate is based on ADADELTA [14], which uses an adaptive learning rate.

Num. Classes	Training size	Validation size	Epochs	Accuracy
22	18645	8959	15	98.9%

Symbol	**Validation size**	**Training size**
$<$	114	950
$>$	101	909
$-$	118	950
$+$	90	810
$\times$	1509	950
0	133	950
1	118	950
2	124	950
3	120	950
4	61	549
5	78	702
6	100	900
7	75	675
8	121	950
9	90	810
π	1533	950
α	2601	950
β	1131	950
a	86	774
b	57	513
c	67	603
$/$	532	950

6 Discussion

The bounding box approach has some advantages and some obvious limitations such as: (i) when a symbol is made up of two or more components, or (ii) when two symbols are touching each other. To deal with (i), it is possible to expand the structural analysis step to include a mechanism that can make probabilistic assumptions whether two components are part of the same symbol, for example a Markov chain. Unfortunately, for the second limitation (ii), a solution is not so simple. This would require that either the CNN can classify multiple symbols within one image, or that the segmentation step itself uses a machine learning approach to predict where one symbol ends and another begins.

It is important to note that although the CNN model states it's accuracy to be 98.9%, this is a little misleading. During testing on real-world data, it was observed that a lot different symbols were being classified under the same output. As an example, the equation $\sqrt{3+2} \times 8\pi + 97$ would produce an output such as $\sqrt{\alpha + 2}\alpha 8\pi + 9\alpha$. It was later discovered that this issue was caused due to an imbalanced dataset, some symbols in the dataset had 10x more sample images than others. This issue can be solved by combining more data from different data sets in order to try an balance the number of samples. Another approach, which is applied to this project, is to implement a simple random sampling step that tries to make the number of samples nearly uniform across all classes.

In this project, only 22 of the 369 available classes were used. The CNN model behaved poorly on large number of classes, in future work it would be crucial to re-design a new CNN model that can accurately classify more classes. This would also require a better random sampling algorithm to deal with the imbalanced data.

7 Conclusion

This project has shown promising results in the task of classifying handwritten mathematical expressions, by using a bounding-box approach for image segmentation, and a convolutional neural network for symbol classification. Although the work presented here was limited to only whole numbers, binary operators, square roots and one-lined equations, it serves as a useful starting point for future work.

8 Acknowledgements

I would like to thank Dr. Wenbo He for her support in the development of this project and providing guidance throughout the semester.

References

[1] Mathpix. `https://mathpix.com/`.

[2] Photomath. `https://photomath.net`.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, Sep 2004.

[5] Richard Zanibbi Utpal Garain Dae Hwan Kim et al. Harold Mouchère, Christian Viard-Gaudin. Icdar 2013 crohme: Third international competition on recognition of online handwritten mathematical expressions. 2013.

[6] Philipp Kuhl and Daniel Kirsch. Detexify. `http://detexify.kirelabs.org/classify.html`, 2009.

[7] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[8] Catherine Lu and Karanveer Mohan. Recognition of online handwritten mathematical expressions using convolutional neural networks. 2016.

[9] Matlab image processing toolbox, 2018a. The MathWorks, Natick, MA, USA.

[10] Matlab neural network toolbox, 2018a. The MathWorks, Natick, MA, USA.

[11] R. Padmapriya and S. Karpagavalli. Offline handwritten mathematical expression recognition. *International Journal of Innovative Research in Computer and Communication Engineering*, 2016.

[12] Yisu Peng and Yang Zhang. Offline mathematical expression recognition. 2016.

[13] Martin Thoma. The hasyv2 dataset. *CoRR*, abs/1701.08380, 2017.

[14] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[15] F. Álvaro, J. A. Sánchez, and J. M. Benedí. Offline features for classifying handwritten math symbols with recurrent neural networks. In *2014 22nd International Conference on Pattern Recognition*, pages 2944–2949, Aug 2014.