Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks

Catherine Lu
Computer Science, Stanford University
cglu@stanford.edu

Karanveer Mohan
Computer Science, Stanford University
kvmohan@stanford.edu

Abstract

We further investigate the problem of recognizing handwritten mathematical expressions, which we also chose for our CS221 final project [3]. Being able to change handwritten expressions into LaTeX has applications for consumers and academics. While large amounts of work have been done for digit and character recognition [2] [10], much less progress has been made surrounding handwritten expression recognition. To the best of our knowledge, no papers have been published applying Convolutional Neural Networks (CNNs) to the task of handwritten expression recognition. We build an end-to-end system using our best CNN model to go from strokes to symbols to a LaTeX expression. We also compare our results to other systems; experimental evaluation suggests that CNNs are a powerful tool for handwritten mathematical expression recognition.

1. Introduction

Mathematical typesetting systems and editors such as LaTeX are widely used for formatting mathematical expressions. They produce well-formatted and professional output. However, they are slower to use than handwriting and may have a steep learning curve for new users. A system that recognizes handwritten mathematical expressions and turns them into a machine-readable format such as LaTeX would allow a user to benefit from the best of both approaches, but building such a system is difficult. There are many different mathematical symbols used, and there often are ambiguities in symbol location and layout in handwritten expressions [4].

The 2013 Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) was won by Vision Objects (now named MyScript), with 60% accuracy at the expression level; second place only achieved 23% accuracy. Vision Objects is a privately held company with proprietary technology [5], so while public-domain research has the potential to reach much higher accuracies, it is currently unclear how this can be achieved. The poor results in CROHME reflect the general state of handwritten expression recognition public research. Further, to the best of our knowledge, no papers have been published applying Convolutional Neural Networks (CNNs) to the task of handwritten expression recognition.

Our project investigates the problem of recognizing handwritten mathematical expressions, which we also chose for our CS221 final project [3]. Our primary contribution is in creating an end-to-end system using a well-trained CNN model to go from strokes to symbols to a LaTeX expression.

2. Background & Related Work

2.1. Handwriting Recognition

Much work has been published to date on recognizing handwritten numbers and English words [6]. Character and digit recognition are very well-studied problems; the MNIST dataset is often used as a dataset to try out new machine learning models because it is so widely used [10] [4]. In the research community, handwriting recognition has two different forms: offline recognition, where the writing is done in a non-digital medium such as paper, and online recognition, where the writing is done in a digital medium that records pen-tip movements such as a tablet. Naturally, online recognition accuracy is higher than offline recognition accuracy, since additional information is provided for online about how the writing is done. Nevertheless, offline handwriting recognition is used in large-scale real-world systems such as interpreting handwritten postal addresses or monetary values on bank checks [6].

Significantly less work has been done on handwritten mathematical expression recognition. Prior to CROHME, the relatively small number of math recognition research was done without benchmark data sets, standard encodings, or evaluation tools; this made progress slow and collaboration difficult for the community [4]. The CROHME competition, organized since 2011, seeks to make it much easier to get started working on handwritten math recognition and

meaningfully compare systems [4].

2.2. Convolutional Neural Networks

Our approach relies heavily on CNNs, which are widely used for a variety of vision recognition problems. Many papers document ways of achieving better results when training/evaluating CNNs, including using slant correction on images [7], elastic distortions to increase the size of the training data set and robustness of the model [9], and extracting additional features from images as inputs to the CNN [7] [8] [9]. We will use some of these methods in our approach to the problem.

3. Approach

Our pipeline has five distinct phases: (1) data set enrichment, (2) image segmentation, (3) data extraction, (4) character-level classification, and (5) expression-level classification.

3.1. Data set

Before it is possible to explain our approach, it is necessary to talk about the data we are provided. Our data comes from the CROHME competition, which provides train and test datasets freely available for research purposes:

	Symbols (S)	Expressions (E)	S/E
Train	16970	2259	7.5
Test	8195	836	9.8

There are 75 different mathematical symbols present in the datasets. The data is given in Ink Markup Language (InkML) format, which is a specific XML type that is specifically used to describe digital writing [1]. Each expression is represented in a separate `<ink>` element. Within an expression, each stroke is then represented in a separate `<trace>` element; the contents of the `<trace>` element are X-Y coordinates representing the places at which the pen-tip is over time.

```
<ink>
<trace>-238 -91, -238 -91, -238 -92,
-238 -92, -238 -92</trace>
<trace>-235 -103, -235 -103, -235 -103,
-234 -103</trace>
<trace>-218 -88, -218 -88, -219 -90,
-220 -89, -220 -89, -220 -89, -220
-89</trace>
<trace>-198 -93, -198 -95, -198 -96,
-198 -97, -198 -97, -198 -98</trace>
</ink>
```

A sample of InkML

3.2. Data set enrichment

We increase the size of our training data set by adding distortions to the original dataset. This is done through a method suggested in [9]: to use an interpolation scheme to slightly displace each of the pixel values in an image; thus, for each data point we get from the trace, we randomly select one of -1, 0, 1 to add to each of the coordinates. We increase our training dataset to twice the size by adding a distorted version of each expression.

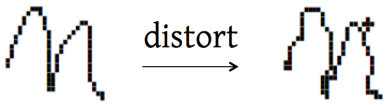

Figure 1.

3.3. Image Segmentation

Though we are given trace information from the dataset, it becomes necessary to determine which traces comprise a complete symbol for eventual symbol classification. We use the straightforward heuristic that intersecting traces are part of the same symbol. There are two cases in which the heuristic fails. In the first case, it may fail if there is overlap between two symbols. In the second case, symbols which generally do not contain overlap, such as $=$, will fail almost all the time. To account for this, we add an additional step to find symbols with non-intersecting strokes e.g. $=$, $\log$, $\sin$, $\cos$, etc. The additional step considers additional strokes even if they are non-overlapping. If our best classifier determines that the strokes considered constitute one of these symbols with probability of at least 0.7, we segment these strokes out as that symbol.

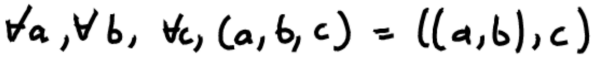

Figure 2. This expression would be perfectly segmented.

3.4. Data extraction

We transform the InkML data into a form that our models can handle. We change the InkML data comprising individual mathematical symbols into normalized images represented as pixel arrays. Then, we blur the pixels and include the number of strokes as an additional feature. Pixel blurring is done by changing the pixel information from 0s and 1s to a distribution with mean 1 on the pixel and noise with 0.2 decreasing information in the surrounding pixels for every pixel distance away (defined by euclidean distance). For the CNN models, including the number of strokes is done only in the last affine layer.

The pixel arrays are normalized to size 24x24 pixels for the classifiers, except for the 4-layer CNN which takes in a pixel array of size 32x32.

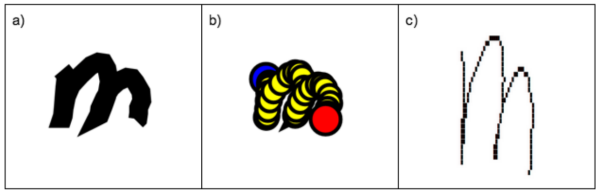

Figure 3. a) The symbol b) As InkML c) As pixel array

3.5. Character-Level classification

We train a classifier to output one of 75 symbol classes. Our baseline classifier is the SVM classifier trained previously for CS221 [3], which saw 87% accuracy for the test and train datasets.

Though we believe that CNNs will work much better, we train fully-connected neural networks (NNs) as another baseline of comparison.

Finally, we spend most of our efforts in training good CNNs for the character classification task. This is done by training multiple models with different architectures and parameters in an effort to find the best trained CNN for this task.

3.6. Expression-level classification

Performing symbol-level classification alone leaves out much of the important contextual information that a mathematical expression provides. For instance, $x = 1$ is a valid mathematical expression, and it is more likely to be what was written as opposed to $x =<$.

To leverage this idea, we use Hidden Markov Models (HMMs) with binary and unary potentials. HMMs allow us to determine the most likely sequence of symbols in an expression, given the probabilities of each symbol (unary potentials) and conditional probability of symbols given the symbol before it (binary potentials). The probabilities used as unary potentials are calculated using the best trained CNN model from character-level classification. Binary potentials are approximated using their empirical occurrence from the training data set with a Laplace smoothing of 1. We approximate the best expression with Gibbs sampling using a sample size of 2000. In each sample, a symbol is randomly chosen to vary; setting all others constant, the selected symbol is changed to that which is most likely is chosen. Ultimately, the sequence of symbols that is most common among the sample size of 2000 is taken as the classified mathematical expression.

As a concrete example, consider an expression with 3 symbols, which we will label as A B C. If we choose A as the symbol to vary, we calculate

$$\arg\max_s p(A = s) * p(A = s|A \text{ is start tag}) * p(B|A = s)$$

where s denotes the symbol we are choosing. Note here that since A is chosen, C doesn't affect the probability as we are only using the probability given the preceding tag i.e. binary potentials.

4. Experiments

We will now report results of image segmentation, character-level classification, and expression-level classification. Note that the results reported do not use distortions mentioned in 3.2 since we observed that distorting the images and doubling the dataset size did not provide measurable improvements in accuracy. Additional ideas related to elastic distortion are discussed in Section 5.

4.1. Character-level classification

Our first step is to train a good classifier, which will be needed for segmentation and for the HMM. As previously mentioned, our baseline SVM classifier to beat has a train and test accuracy of 87%.

4.1.1 Neural Network

We tune a 2-layer fully connected NN. The best results (with test accuracy $\geq 20\%$) are reported below with different sizes for the hidden layer (H), though note that we tuned with many more parameters. While our accuracy is fairly high considering that there are 75 symbol classes, it is nowhere near the best performance from our baseline SVM classifier.

Parameters that we tuned also include the learning rate, regularization, momentum, batch size, and learning rate decay.

Parameters	Train	Test
$H = 200$	25.3%	20.3%
$H = 500$	40.0%	30.2%
$H = 1000$	40.1%	31.4%

We observe that the accuracies for $H = 500$ and $H = 1000$ are very similar, although one has 500 hidden neurons while the other has 1000. This suggests that adding additional neurons beyond 500 does not capture anything else in the data.

4.1.2 Convolutional Neural Network

Most of our time in character-level classification has been spent training and tuning CNNs. The architecture used

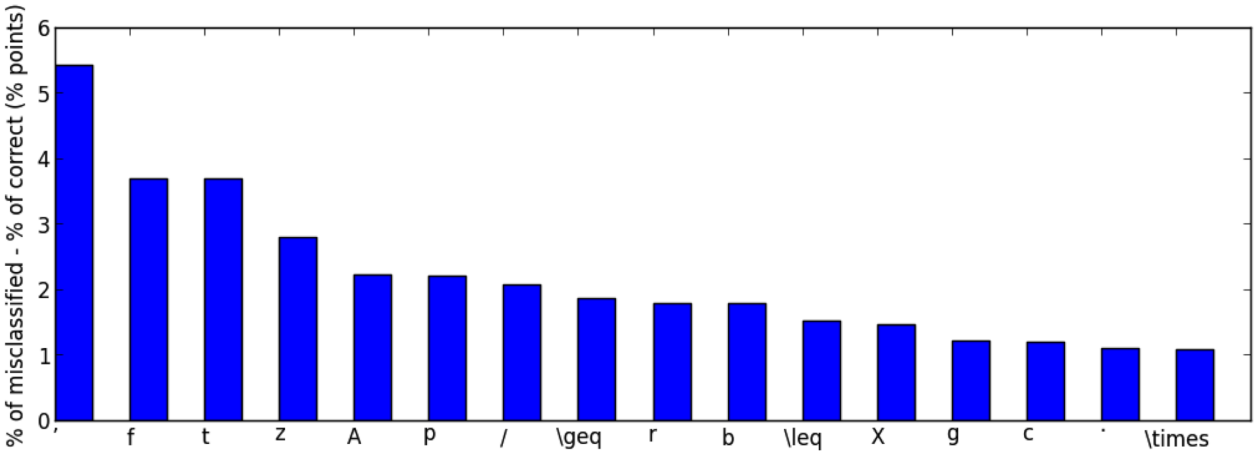

Figure 4. The most over-represented misclassified symbols.

for the CNNs is of the form [Conv-Relu-Pool]xN - [FC - Relu]xM - [FC] - [Softmax]. Our best test accuracy is 90%. The parameters that we report along with our values of N and M are filter size (F) and number of hidden neurons per layer (H). The number of filters used is 32 for all architectures. In the pooling step, we use a 2x2 max pool with stride of 2.

Our best results, for each architecture and after parameter tuning, are below. Parameters that we tuned also include the learning rate, regularization, momentum, batch size, and learning rate decay.

Parameters	Train	Test
$N = 1, M = 0, H = 32, F = 5$	92.3%	82.6%
$N = 2, M = 0, H = 1000, F = 5$	95.4%	83.8%
$N = 2, M = 1, H = 1000, F = 5$	96.6%	88.9%
$N = 3, M = 1, H = 500, F = 7$	98.7%	89.7%
$N = 4, M = 1, H = 500, F = 5$*	99.6%	86.7%

*Used 32x32 pixel images instead of 24x24.

Note that even a shallow CNN produces accuracies comparable to our baseline, and the CNN with $N = 3/M = 1$ beat the baseline with a test accuracy of 90%.

4.1.3 CNN Error Analysis

With our best CNN, where $N = 3$ and $M = 1$, we look at which symbols are most disproportionately misclassified. That is, we give each symbol a score $S =$ % chosen incorrectly $-$ % chosen correctly and sort them descending. Figure 4 plots the symbols that are most disproportionately misclassified. The most over-represented misclassified symbol is the comma, which makes sense since we do not include original size information from the symbol. So, it is difficult for the CNN to distinguish it from the number 1, for instance.

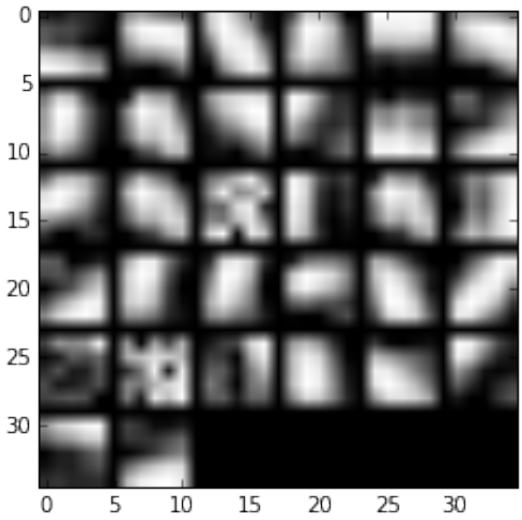

Figure 5. The visualized weights in the first hidden layer of the two-layer CNN when $N = 1$ and $M = 0$. Many of the weights appear to represent different edges.

4.2. Segmentation

Now, we perform segmentation as described previously. We compare the results of our segmentation accuracies using the heuristic + CNN against values previously determined from [3] using the heuristic alone and using the heuristic + SVM. The CNN used is our best CNN, where $N = 3$ and $M = 1$, and the SVM used is the same baseline SVM. Below are the segmentation accuracies when evaluating symbol (S)-level and expression (E)-level segmentation accuracies.

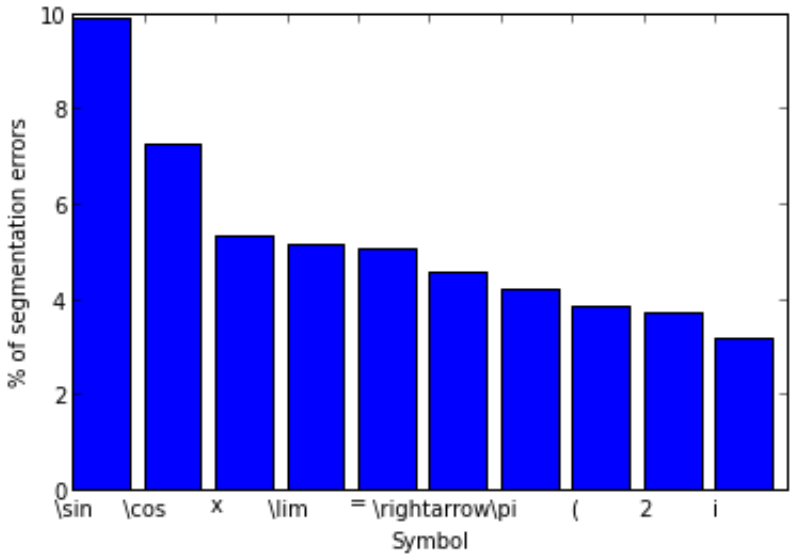

Figure 6. The top-10 symbols that contribute to segmentation error.

	S (all)	E (all)	S (*)	E (*)
Heuristic	81%	17%	89%	45%
Heuristic + SVM	87%	35%	92%	53%
Heuristic + CNN	88%	39%	93%	56%

* Ignores =, i, j, $\leq$, log, sin, cos, lim, $\geq$, $\rightarrow$, $\div$

Our heuristic + CNN slightly outperforms the heuristic + SVM model on segmentation at the symbol-level. This performance boost is compounded at the expression-level, where there is a larger increase in accuracy.

4.2.1 Heuristic + CNN Error Analysis

It is useful to determine where we are still falling short in terms of segmentation accuracy. So, we plot the top-10 symbol contributing to segmentation error (see Figure 6).

The sin and cos symbols are the top contributors to segmentation error; combined, they comprise 17%+ of all segmentation errors. They are also the two symbols that contributed the most to error for the heuristic + SVM case. Further, relaxing the threshold at which the CNN segments out a symbol with non-overlapping strokes does not improve the segmentation accuracy; while it increases the segmentation accuracy of sin and cos, for instance, the segmentation accuracies of 1-stroke symbols such as numbers and parentheses decreases. This suggests that the CNN is not very confident when classifying sin and cos compared to other symbols. One possible solution to this problem is to increase the number of sin and cos training examples to give the CNN more data for more confident classifications. Another solution is to hand-tune thresholds for different symbols, but this is less than ideal as it would be a very manual process.

Of course, in our segmentation approach we are making the big assumption that if strokes are overlapping, they belong to the same symbol. Below is a simple example where our segmentation approach fails.

Figure 7. A case where segmentation fails. The symbols A and x intersect, so segmentation will mistakenly view them as one symbol instead of two.

4.3. Expression-level classification

4.3.1 Expression-level accuracies

At this point, it's useful to determine the overall expression-level accuracies of our system.

No. of symbols wrong:	**0**	**$\leq$ 1**	**$\leq$ 2**
SVM*	35%	64%	80%
CNN*	39%	64%	83%
SVM †	16%	37%	56%
CNN †	18%	39%	57%

*With perfect (i.e. ground truth) segmentation
† With heuristic segmentation + trained model

Our CNN-based system outperforms our SVM-based system. Understandably, the expression-level accuracy is significantly lower than our symbol-level accuracy, even

with perfect segmentation. This emphasizes the importance of future work to improve the character-level classification to get much better expression-level accuracy.

Another approach to improving expression-level accuracy that we've chosen to implement is to use HMMs to take advantage of contextual information. Details are in the next section.

4.3.2 Using Hidden Markov Models

As the last step, we use HMMs to improve the expression-level overall accuracy. We discard 100 of the expressions for the burn-in period. Results are:

No. of symbols wrong:	**0**	**≤ 1**	**≤ 2**
*	41%	66%	83%
†	19%	41%	58%

*With perfect (i.e. ground truth) segmentation
† With heuristic segmentation + trained model

Expression-wide baseline SVM results were not reported in [3]. The best symbol-level accuracy obtained from previous work was 85%, down from 87% without HMMs.

Note that using HMMs make expression-level accuracy slightly better. We then further investigate what impedes additional accuracy increases from HMMs.

The figure below highlights some of the differences in how particular symbol class accuracies increased or decreased significantly. We show the top 3 symbols with the worst improvement, and the top 3 symbols with the greatest improvement through the use of HMMs.

% accuracy:	**W/ HMM**	**W/o HMM**	**Diff**
$\geq$	35%	100%	-65%
m	10%	67%	-57%
$<$	39%	83%	-44%
∞	86%	74%	12%
π	63%	50%	13%
$\lim$	93%	0%	93%

We also want to look at the most popular symbols, and see how they are affected. Below are the top 5 most popular symbols, along with their % change with the HMM:

% accuracy:	**W/ HMM**	**Diff**	**Count**
$-$	99.9%	0.3%	922
2	96.1%	1.3%	893
1	92.5%	2.3%	729
$+$	100.0%	0.8%	683
x	94.8%	2.0%	681

Overall, using the HMM helps for symbols that are more popular. Since they appear more, their conditional probability given other symbols is higher and they are more likely to be chosen in an expression. Conversely, the HMM will penalize symbols that are less likely to appear. This trade-off ultimately slightly helps for our results overall.

5. Conclusion & Future Work

From our research, we've learned that CNNs are a strong approach to solving handwritten expression recognition. With better computing resources and more time, it may be possible to significantly increase the overall expression accuracy. Small percent increases in accuracy at the symbol level have drastic effects at the expression level, since the symbol accuracies are compounded at the expression-level. One potential way to increase symbol-level accuracy is to train deeper CNNs with smaller filter sizes. Another would be to do additional types of dataset distortions such as rotations and other affine transformations.

There are several limitations with our work. Our approach to segmentation must be refined; the 88% segmentation accuracy cuts the overall expression accuracy in half. There are many other segmentation approaches to try; one is the approach that the second-place team, from Univ. Valencia, did for CROHME 2013. Their system contains a parser which builds multiple hypotheses for segmentation, symbol classification, and structural relation. The most likely hypothesis is then chosen [4]. This is a natural extension to our work; we should include different segmentation hypotheses as input into the HMM as well.

Another limitation is that we ignore spatial information. However, spatial information to recognize fractions, summations, etc. is necessary in mathematical handwriting recognition. Again, we could try the approach done by the Univ. Valencia team, which encodes spatial information while building their hypotheses [4].

A third limitation is that we ignore size information. This is evident by our misclassified symbols graph, which shows that the comma is the most over-represented misclassified symbol. By including a normalized size into the last affine layer of the CNN, this issue may be mitigated. All of these are promising potential areas of future work.

6. Acknowledgements

Thank you to the CS231N staff for putting together such an amazing class. We've learned a ton.

References

[1] Ink markup language. http://www.w3.org/TR/InkML/.

[2] D. Cirean, U. Meier, and J. S. Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[3] C. Lu and K. Mohan. Recognition of online handwritten mathematical expressions, 2013.

[4] H. Mouchere, C. Viard-Gaudin, R. Zanibbi, U. Garain, D. H. Kim, and J. H. Kim. Icdar 2013 crohme: Third international competition on recognition of online handwritten mathematical expressions. *Conference: International Conference on Document Analysis and Recognition (ICDAR)*, 2013.

[5] MyScript. Demonstration portal. Previously Vision Objects. http://webdemo.myscript.com/.

[6] R. Plamondon and S. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and machine intelligence*, 22(1), 2000.

[7] A. Rehman and T. Saba. Neural networks for document image preprocessing: state of the art. *Artificial Intelligence Review*, 42(2):253–273, 2014.

[8] J. Shah and V. Gokani. A simple and effective optical character recognition system for digits recognition using the pixel-contour features and mathematical parameters. *(IJCSIT) International Journal of Computer Science and Information Technologies*, 5(5), 2014.

[9] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. Institute of Electrical and Electronics Engineers, Inc., August 2003.

[10] O. D. Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition-a survey. *Pattern Recognition*, 29(4):641 – 662, 1996.