Recognition of Handwritten Mathematical Expressions Using Systems of Convolutional Neural Networks

Tate Rowney[1], Alexander I. Iliev[2]

[1] The Bay School of San Francisco, United States
taterowney@gmail.com

[2] Institute of Mathematics and Informatics,
Bulgarian Academy of Sciences, Bulgaria
ailiev@berkeley.edu

Abstract

Accurate recognition of handwritten mathematical expressions has proven difficult due to their two-dimensional structure. Various machine-learning techniques have previously been employed to transcribe handwritten math, including approaches based on convolutional neural networks (CNNs) and larger encoder/decoder-based models. In this work, we explore a CNN-based method for transcribing handwritten math expressions into the typesetting language known as $\LaTeX$. This approach utilizes machine learning not only for classifying individual characters but also for extracting individual characters from handwritten inputs and determining what forms of two-dimensionality exist within the expression. This approach achieves significant reliability when recognizing common mathematical expressions.

Keywords: Handwritten mathematical expression recognition, convolutional neural networks, deep learning

ACM Computing Classification System 2012: Computing methodologies → Artificial intelligence

Mathematics Subject Classification 2020: 68T45 (Computer science → Artificial intelligence → Machine vision and scene understanding)

1 Introduction

Machine learning — the use of a variety of statistical and computational methods to help a computer stochastically discover patterns in inputs — is not a new concept. As early as 1943, McCulloch and Pitts proposed that complex thought could be modeled using a system of mathematically simulated "Artificial Neurons" [1], creating what is now known as an artificial neural network. However, progress in machine learning relatively stagnated for many decades and was plagued by unsatisfactory results compared to other methods of implementing artificial intelligence. The exploration of deep learning, the use of multiple connected "layers" of artificial neurons, started to regain momentum in the 1980s with the application of backpropagation, a simple calculus-based algorithm to tune neurons to reduce error and fit patterns in data. Breakthroughs in deep learning strategies such as Recurrent Neural Networks and Convolutional Neural Networks, as well as the ease with which widely available Graphics Processing Units (GPUs) can expedite and parallelize the training of neural networks [2], have caused deep learning to once again reside at the forefront of AI.

Machine learning has shown particular aptitude in problems of computer image recognition. In 1998, LeCun et al. introduced a machine-learning architecture to recognize handwritten numerical digits with an accuracy surpassing many traditional methods, using what is known as a

Convolutional Neural Network (CNN) [3]. This paper helped to popularize the use of CNNs in image recognition and signal processing, where they are broadly applicable. CNNs, like traditional neural networks, are composed of layers of artificial neurons that translate and linearly scale inputs (effectively a high-dimensional affine transformation). CNNs also contain "convolutional layers," in which an n-dimensional array of input values is combined with an n-dimensional array of adjustable weights (called the "kernel") in a discrete analog of mathematical convolution (Figure 1). This strategy is used in classical programs to blur, sharpen, recolor, or adjust images and, when tuned by machine learning algorithms, can significantly improve artificial neural networks' ability to discover patterns in image data.

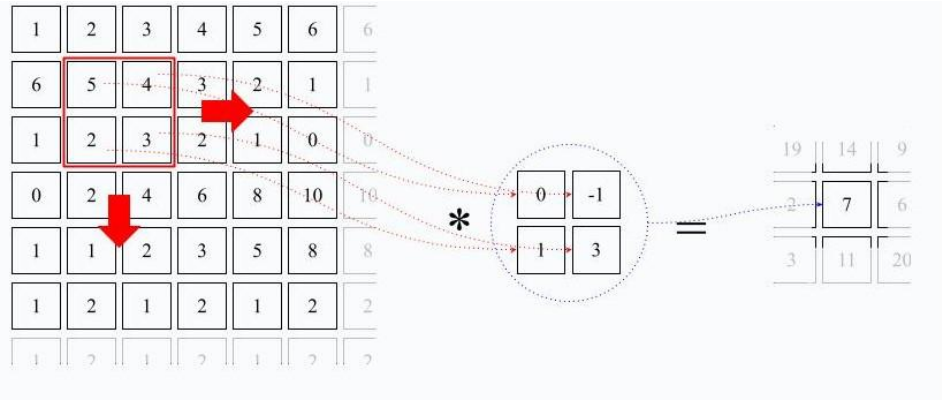


Fig. 1: Convolution of a 2D input with 2-by-2 kernel

CNN-based deep learning has achieved nearly superhuman results in transcription of handwritten text [3, 4]. However, since CNN-based classifiers sort inputs into a discrete number of classes, they are generally trained to recognize only one character at a time (as with [3 - 5]), requiring longer inputs to be split into smaller images containing a single character each. This becomes a challenge in problems such as transcribing handwritten mathematical expressions into a typesetting language: variation in characters' size and location (as with exponents), as well as overlap between characters (such as the contents of a square root), make separating individual characters much more complicated than in standard, evenly-spaced handwritten text. Previous implementations have utilized classical methods for character separation, such as by combining all contiguous curves into a single character [5]. This method provides varying levels of accuracy, including the erroneous combination of different characters that happen to overlap (as in Figure 2). Some approaches have bypassed this problem entirely by utilizing encoder-based models with variable output lengths, such as the transformer architecture. These models are

significantly more complex than a standard CNN classifier, using considerably more parameters and requiring upwards of 150 training epochs to achieve accuracy [6].

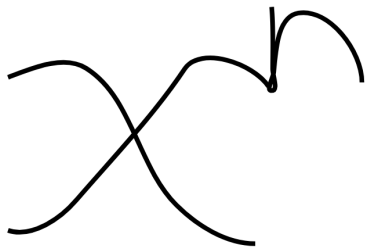
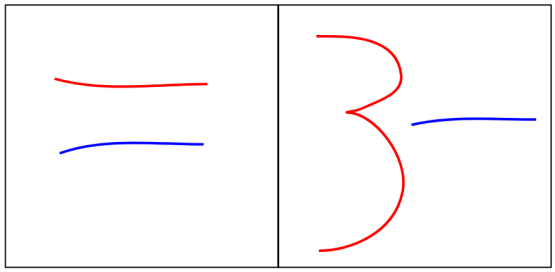


Fig. 2: A single connected curve forming two different characters (x^n).

2 Strategy

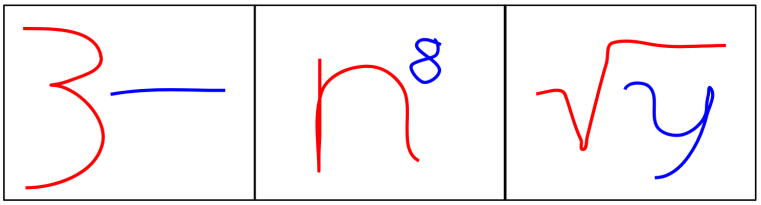
To mitigate these challenges, we propose a new strategy to recognize and transcribe handwritten expressions into the $\LaTeX$ typesetting language. This strategy implements online character recognition, requiring data on the positions of and the order in which the user draws each distinct curve that composes each character. It utilizes three separate convolutional neural networks: the first of these, a model to detect stroke combination, is trained to predict whether two strokes belong within the same character, using images formed by combining two subsequent strokes, as well as a binary feature indicating whether these strokes are part of a single $\LaTeX$ symbol (Figure 3). The second, a special formatting predictor, determines what kind of "two-dimensionality" is present between two adjacent characters; more specifically, whether the two characters lie next to each other or form an exponent, fraction, etc., equivalent to the presence of "{" and "}" in the $\LaTeX$ output. This model is trained on cropped, scaled images of adjacent characters, as well as a label indicating what (if any) type of nonstandard formatting is present (Figure 4). Finally, a character classifier is trained on images of individual characters, mapping each to its $\LaTeX$ equivalent (Figure 5).
Thus, full-length handwritten mathematical expressions can be analyzed by feeding each distinct curve drawn by the user, as well as the curve drawn immediately after, into the stroke

combination predictor to determine which strokes are part of a single character (a low predicted probability indicates that curves before and after the one considered should be separated). Subsequently, each group of curves can be individually classified into a LaTeX character. Appropriate curly braces can be added to the LaTeX output based on the results of the special formatting prediction model: by testing a character combined with others located incrementally longer distances from it until they are determined to be "in a row," the locations of opening and closing curly braces can be determined.



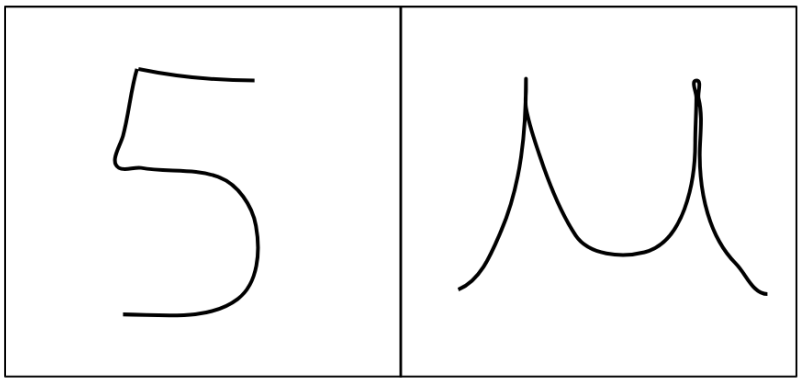
Combine (1) Don't Combine (0)

Fig. 3: Example training data for stroke combination model



No nonstandard formatting | Nonstandard formatting: exponent | Nonstandard formatting: square root

Fig. 4: Example training data for nonstandard formatting classification model



5 \mu

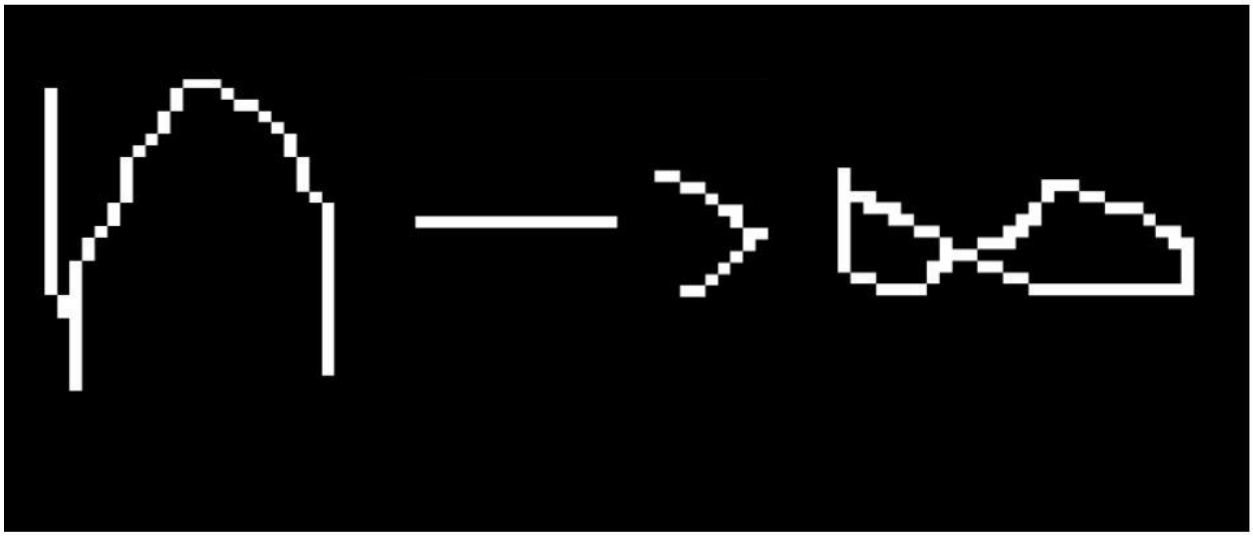
Fig. 5: Example training data for character classification model

3 Motivation

This strategy presents numerous benefits over current CNN-based handwritten equation recognition models, mainly due to its use of deep learning to differentiate between characters. This offers greater potential for adaptability and accuracy compared to classical approaches. Additionally, it eliminates the need to hard-code different recognition algorithms for different types of two-dimensionality in expressions: helping the model recognize more complex $\LaTeX$ tokens is simply a matter of adding to the training set.
Even though this strategy cannot adjust its predictions based on the context of other nearby characters, it uses a far simpler architecture than most context-interpreting encoder/decoder models, likely requiring fewer parameters. Furthermore, this strategy can begin to predict the corresponding $\LaTeX$ of user input before a user has finished writing. It only compares curves near each other, allowing it to begin segmenting and recognizing characters the user has just written. This can save computation time that would otherwise occur after the expression was fully written.

4 Results

An implementation of this strategy was trained using Python's TensorFlow library on the CROHME 2019 dataset. This dataset contains approximately 11,000 handwritten mathematical expressions composed of a total of 98 distinct characters (Figure 6). Expressions are recorded as a series of curves, each composed of a variable number of x and y coordinates, representing (in order) the points the user's pen has passed through. It also contains information on which curves combine to form any given character, as well as $\LaTeX$ and MathML representations of the entire expression.



n \rightarrow \inf

Fig. 6: Format (after cleaning and rendering) of CROHME dataset expressions

The character classification model was trained by combining each character's strokes together, converting them to a bitmap representation using Bresenham's Line Algorithm [7], and scaling down to a 28x28x1 black-and-white image. Each of the 201,114 images, along with an integer representing the characters they depict, was fed into a convolutional neural network (described below) with a final 98-unit softmax layer and compiled using the sparse categorical cross-entropy loss function.

The stroke combination model was trained by combining pairs of consecutive strokes from each expression, converting them to an image as described above, and labeling them based on whether they come from the same character. The 493,440 examples were fed into the convolutional neural network below with a sigmoid-activated final layer and compiled using a binary cross-entropy loss function.

The character nesting model was trained by combining the strokes from a single character and all characters following it, which were either nested or directly following the set of nested characters. Each pair of strokes was converted into an image as described above, which was then separated into two different arrays, each containing one of the characters. These 172,357 examples were labeled according to which type of nesting they represented (no nesting, superscript, subscript, square root, and the top/bottom of a fraction). These were fed into the convolutional neural network below with a final softmax-activated layer and compiled using the sparse categorical cross-entropy loss function.

All of the above models utilized a similar structure, pictured in Figure 7. Three 2-dimensional convolutional layers with a kernel size of 3x3 were used, each followed by a batch normalization operation and a ReLU activation function. The first convolutional layer was initialized with 32 input channels, with the number of channels doubling each subsequent layer. A max pooling layer with a 2x2 pool size was placed after the final convolutional layer to prevent overfitting. Following convolution, the output was flattened, fed through 2 dense sigmoid-activated layers with 256 units each, a 30% dropout layer, and a final dense layer with properties dependent on the network in question. All networks were optimized using the Adam optimizer with a learning rate of 0.001 and trained for 20 epochs with a batch size of 32.

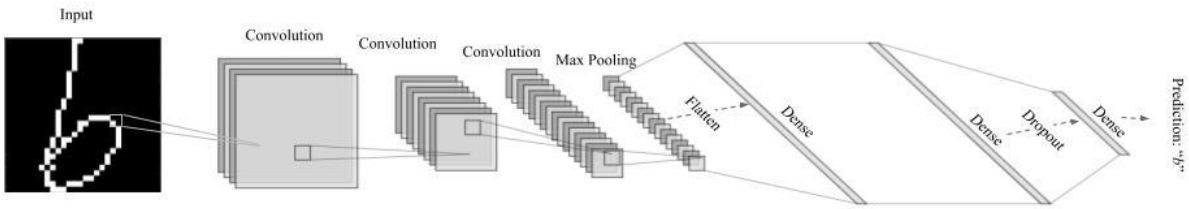


Fig. 7: Model structure repurposed for each of the three CNNs

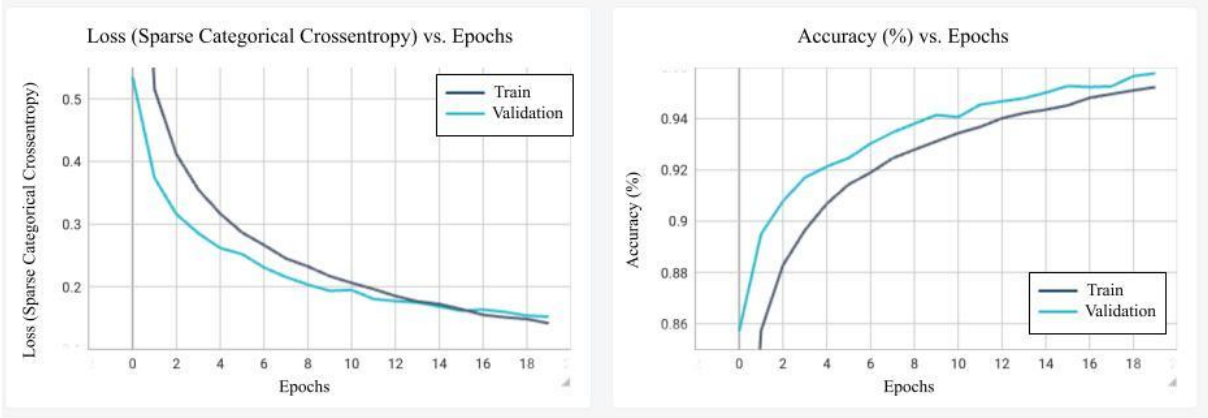


Fig. 7: Loss and accuracy plots for character classification model

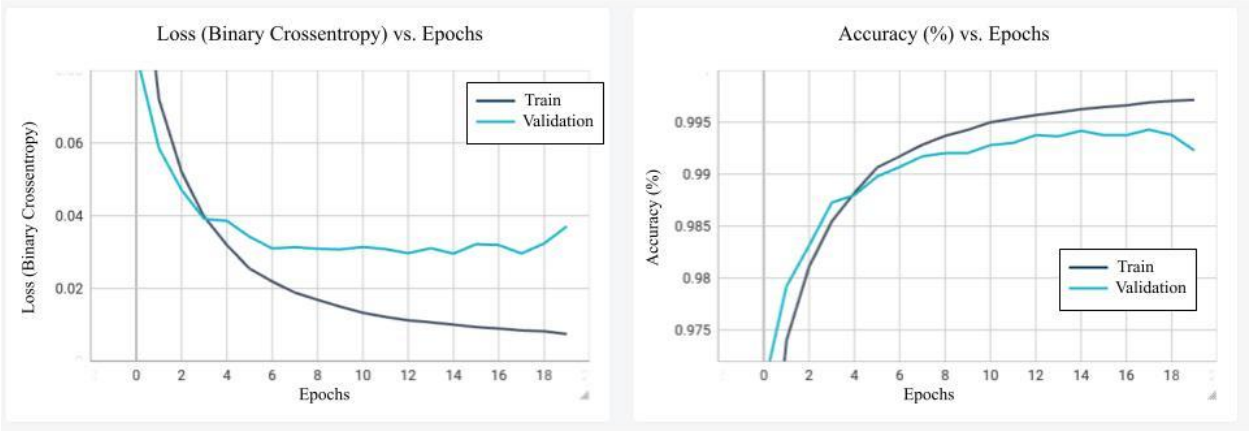


Fig. 8: Loss and accuracy plots for stroke combination model

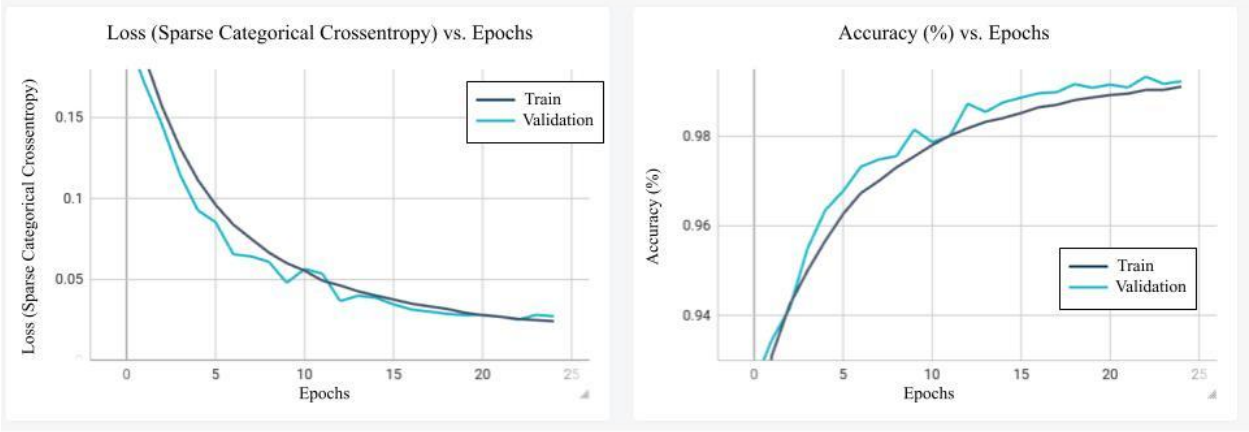


Fig. 9: Loss and accuracy plots for nonstandard formatting classification model

Model Name	**Final Layer Size**	**Model Accuracy**	**Trainable Parameters**
Character Classification	96	95.77%	314,912
Stroke Combination	1	99.42%	290,497
Nonstandard Formatting Classification	6	99.33%	423,110

Fig. 10: Size of final layer, mean accuracy on data outside of training set, and number of trainable parameters for each of the three models

Calculations were performed on a machine running Ubuntu 22.04, Python 3.9, and TensorFlow 2.12, equipped with an Intel® i9 CPU and NVIDIA® 40-series GPU. Python's Matplotlib, OpenCV, and Numpy libraries were used during the cleaning and preprocessing of the dataset. When tested on a set of 50 common mathematical expressions, the program as a whole was able to successfully recognize individual characters with approximately 84% accuracy and correctly identify full expressions with approximately 58% accuracy. The program performed better on shorter inline and exponential equations but became somewhat unreliable when working with characters composed of many separate lines.

5 Conclusion

Despite its relative simplicity, this approach to recognizing handwritten mathematical expressions achieves considerable accuracy. One area of concern for this strategy relative to encoder/decoder-based approaches is its inability to correct errors based on context. For example, the result of "*c0s*" is likely a misinterpretation of "$\cos$." To remedy this problem, a Hidden Markov or encoder/decoder model trained on n-grams of $\LaTeX$ output could be appended to correct common errors. This model would only require a small input (the uncorrected output of the CNNs), giving it a distinct advantage in training time and likely accuracy over models that utilize the entire handwritten expression as input.

References

[1] McCulloch, W.S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133. https://doi.org/10.1007/BF02478259

[2] Oh, K., & Jung, K. (2004). GPU implementation of neural networks. Pattern Recognition 37(6), 1311-1314. https://doi.org/10.1016/j.patcog.2004.01.013

[3] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. https://doi.org/10.1109/5.726791

[4] Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. Twenty-second international joint conference on artificial intelligence.

[5] Lu, C., & Mohan, K. (2015). Recognition of online handwritten mathematical expressions using convolutional neural networks. CS231n project report Stanford.

[6] Wang, H., & Shan, G. (2020). Recognizing handwritten mathematical expressions as LaTex sequences using a multiscale robust neural network. arXiv preprint arXiv:2003.00817. https://doi.org/10.48550/arXiv.2003.00817

[7] Bresenham, J. E. (1998). Algorithm for computer control of a digital plotter. Seminal graphics: pioneering efforts that shaped the field (pp. 1-6).