Spring 2019

Context-based Multi-stage Offline Handwritten Mathematical Symbol Recognition using Deep Learning

Sui Kun Guan
San Jose State University

Context-based Multi-stage Offline Handwritten Mathematical Symbol Recognition using Deep Learning

A Master Project Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Sui Kun Guan

December, 2018

The Designated Thesis Committee Approves the Thesis Titled

Context-based Multi-stage Offline Handwritten Mathematical Symbol Recognition using Deep Learning

by

Sui Kun Guan

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE SAN JOSE STATE UNIVERSITY

December 2018

Dr. Teng Moh Department of Computer Science

Dr. Melody Moh Department of Computer Science

Dr. Robert Chun Department of Computer Science

Abstract

We propose a multi-stage machine learning (ML) architecture to improve the accuracy of offline handwritten mathematical symbol recognition. In the first stage, we train and assemble multiple deep convolutional neural networks to classify isolated mathematical symbols. However, certain ambiguous symbols are hard to classify without the context information of the mathematical expressions where the symbols belong. In the second stage, we train a deep convolutional neural network that further classifies the ambiguous symbols based on the context information of the symbols. To further improve the classification accuracy, in the third stage, we develop a set of rules to classify the ambiguity or otherwise the syntax of the mathematical expressions will be violated. We evaluate the proposed method by using the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) dataset. The proposed method results the state-of-the-art accuracy of 94.04%, which is 1.62% improvement compared with the previous single-stage approach.

Keywords: Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME). Context-based Offline Handwritten Mathematical Recognition. Convolutional Neural Network (CNN). HAndwritten SYmbols (HASYv2). Machine Learning (ML). Multi-Column Deep Neural Network (MCDNN).

Acknowledgments

I would like to thank my project advisor Dr. Teng Moh for his guidance and support throughout the entire project. His vast knowledge in machine learning has provided valuable suggestions for my breakthroughs in many challenges. I would like to thank my committee members Dr. Melody Moh and Dr. Robert Chun for their support and feedback on my project. I would like to thank Dr. Teng Moh, Dr. Melody Moh, and Dr. Robert Chun for teaching me the fundamental and advanced topics in modern computer science. And finally, I would like to thank my families and especially my wife Rishuo Lin for their support throughout my entire Master program.

Table of Contents

List of Figures

List of Tables

1. Introduction

Deep learning has achieved great success in many areas. From single perceptron to very deep neural network, the rapid technology advancement in computation power has driven the success of deep learning. Recently, many challenging tasks, from computer vision to neutral language processing, most state-of-the-art solutions are based on deep learning. The success of deep learning in these areas has demonstrated its powerful capability. Particularly, convolutional neural network has achieved great success in computer vision, including handwritten symbol recognition. For example, machine recognition of the MNIST dataset of handwritten digits has achieved great success with deep learning. The top 2 error rates of the MNIST dataset are only 0.21% and 0.23% [19, 6]. On the other hand, handwritten mathematical symbol recognition is a much harder task in machine intelligence. Not only there are many mathematical symbol classes, but also some symbols can be very similar in shapes and thus harder to classify. For instance, the symbols "l" and "1" are very hard to distinguish due to handwriting styles. Therefore, the general approach to the problem of handwritten mathematical symbol recognition is to design a better regularized classifier. However, isolated mathematical symbol classifiers all face the difficulties in distinguishing the ambiguous symbols, e.g., "s" and "S", and this challenge may only be resolved by using context information – the mathematical expressions where the symbols are appeared [1, 8]. In order to utilize the classification results and further improve the classification accuracy, we must combine the isolated symbol recognition with context information. We refer this combined approach as the context-based classification. In this work, we focus on context-based classification for offline handwritten mathematical symbol recognition using deep learning.

We summarize our main contributions below. In the first stage of isolated mathematical symbol recognition, we extend and improve the current state-of-the-art approach by using ensemble of multiple deep neural network classifiers. Also, compared to previous approaches, our first stage classifier is constructed through a more generic

and deterministic method. In the second stage of context-based recognition, we propose a scheme to encode the context information and train a deep convolutional neural network to classify ambiguous symbols using the encoding scheme. In the third stage, we develop an algorithm to further improve the classification accuracy by enforcing a set of rules that mathematical symbols must be obeyed or otherwise the resulting mathematical expressions will be invalid.

The rest of the paper is organized as follow. Section 2 surveys the related backgrounds in deep learning methodologies. Section 3 surveys the related works in handwritten mathematical symbol recognition. Section 4 presents the proposed deep neural network based multi-stage architecture for context-based classification. Section 5 elaborates on the background of the dataset evaluated in this paper. Section 6 demonstrates the results and comparisons with previous works. And finally, we conclude the paper in section 7.

2. Related Works in Deep Learning and Multi-Column Deep Neural Network

In offline handwritten mathematical symbol recognition, assuming a "perfect" function exists such that it can classify all the handwritten mathematical symbol images correctly, then the goal is to approximate this function from an observable set of inputs and outputs. Under the universal approximation theorem [10], given the proper parameters and conditions, a neural network can approximate a wide range complicated function. Therefore, neural network can be an effective technique for classification problems. Furthermore, through the aid of backpropagation and gradient descent algorithms [15], many modern hardwares, such as GPU, can be used to train/identify the close-optimal parameters of a neural network very efficiently based on the available training samples. Many recent researches have shown that very deep neural network (DNN) is very effective on approximating certain complex functions. On the other hand, using very deep neural network on image classification will significantly increase the

complexity of the neural network, such as the number of learnable parameters, and thus, make it very hard to train. To address these challenges, convolutional neural network was proposed [11] and recent researches have demonstrated its effectiveness on improving neural network for handwritten symbol image classification [6, 9, 14, 19].

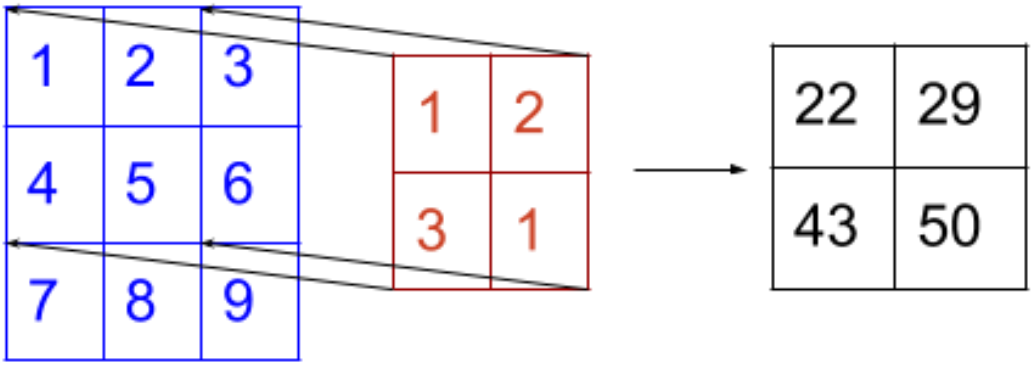

Figure 1: A 2D convolution operation on 3x3 image with 2x2 kernel (stride 1) and the resulting 2x2 output image. The resulting top left pixel value 22 = 1x1 + 2x2 + 3*4 + 1*5.

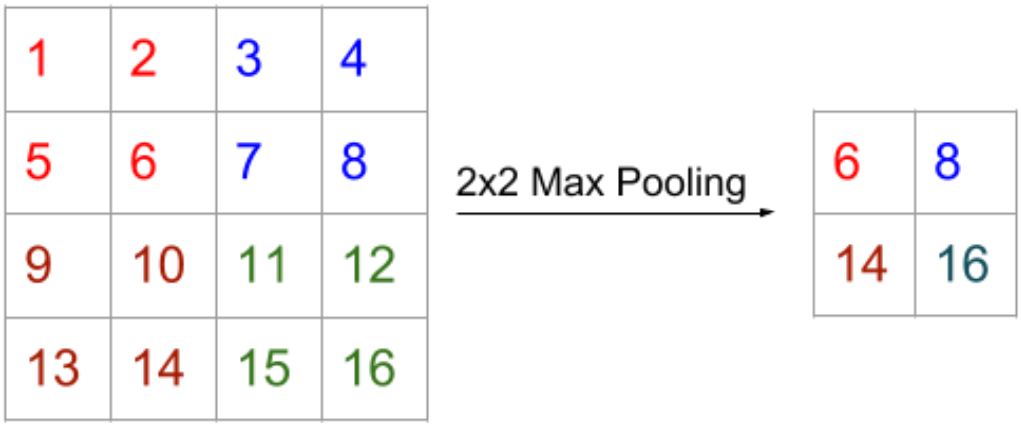

Figure 2: A 2D max pooling operation on 4x4 image with 2x2 kernel (stride 2) and the resulting 2x2 output image. The resulting top left pixel value 6 = max(1, 2, 5, 6).

Convolutional neural network extends the ordinary artificial neural network in three important properties: local receptive field, shared weights, and sub-sampling [11]. Local receptive field defines the kernel/filter size that will be convolutionally applied to the input image. The shared weights property identifies that the same kernel weights to be

used across the input image. Figure 1 illustrates the process of a 2x2 kernel applied to a 3x3 input image. The resulting image is a 2x2 feature map. Notice that the 2x2 kernel weights are shared and applied across all 2x2 regions of the input image. The kernel weights will be updated through the back-propagated error signals so that the results of the convolution will extract the most relevant features for subsequent layers. A typical convolution neural network will consist of multiple convolutional layers. The lower convolutional layers help extract simple low-level features from the image, while the higher convolutional layers construct high-level features through these low-level features. Sub-sampling layers can be inserted between these convolutional layers to further reduce the dimensionality of the intermediate output feature maps, as well as help reduce the non-trivial variances from the inputs [14]. Examples of the most common sub-sampling layers include max pooling and average pooling. An example of a max pooling operation is shown in Figure 2.

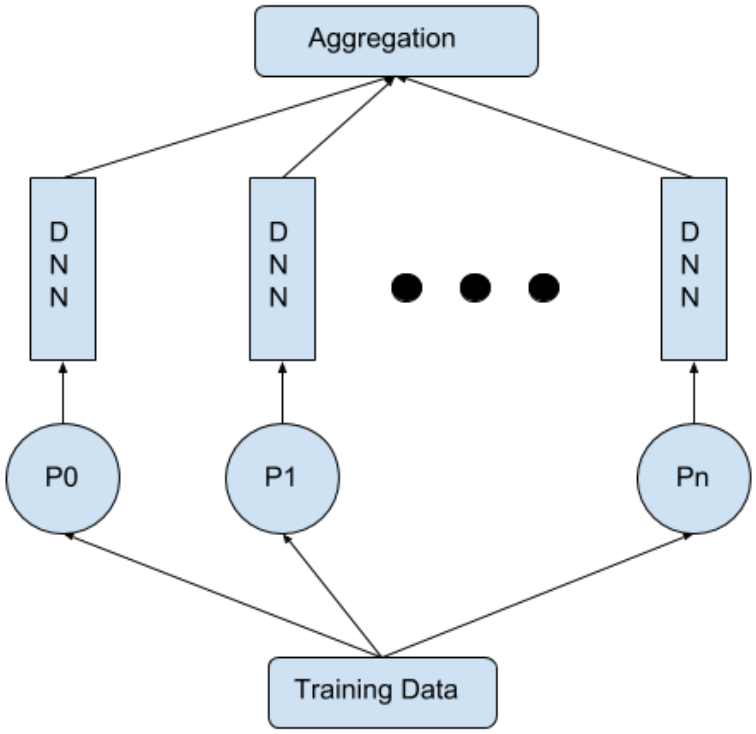

Figure 3: An example of a MCDNN architecture. Training data is expanded by different preprocessing (P0, ..., Pn). Results from trained DNNs are aggregated to produce the final result.

A frequently used technique to construct a better regularized DNN classifier is by assembling multiple DNN classifiers. A Multi-Column Deep Neural Network (MCDNN) architecture can be utilized to achieve this purpose for symbol recognition [6]. Figure 3 illustrates an example of such MCDNN architecture. First, training data is expanded through different preprocessing techniques. Second, for each expanded training dataset, it can be used to train one or more DNN classifier(s). Finally, during the inferencing/test step, the input data is classified by all trained DNN classifiers, and their results will be aggregated to produce the final classification result. This MCDNN architecture tends to be a better regularized classifier than each individual DNN classifier because of the concepts similar to bagging [4]. From the view of the bagging technique, in the overall MCDNN network, training data is expanded through preprocessing. These expanded training data is then split to train multiple individual DNN networks. Experiments have shown that the final result from combining these individual DNNs can often achieve better accuracy [4, 6].

3. Background and Related Works in Handwritten Mathematical Symbol Recognition

In online handwritten mathematical symbol recognition, the pen trace information of the handwritten symbol is provided. The overall shape of the symbol can be obtained by re-plotting and connecting the pen trace information to a digital image. In offline handwritten mathematical symbol recognition, there is no pen trace information. Only the final shape of the symbol image is provided. Thus, offline symbol recognition is traditionally considered as a harder problem.

For online handwritten mathematical symbol recognition, there are many studies conducted and great results are obtained. The “Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME)” currently is the most well-known dataset used in the researches [12, 13]. Davila et al. observed that the training and test data samples per symbol class is highly imbalanced. They used Perlin Noise (PN) to

expand and balance the training data by creating new distorted online training samples. They extracted a feature vector of 102 values that consists of global feature, crossing feature, 2D fuzzy histogram of points, and fuzzy histograms of orientations for each sample. Their best performance was obtained through the use of Support Vector Machine (SVM), and they achieved the accuracy of 85.89% in CROHME 2013 test dataset [8]. Álvaro et al. further enhanced this work by combining both online and offline features for recognizing online mathematical symbols. They generated a sequence of online and offline features for each symbol, and then used Bidirectional Long Short Term Memory (BLSTM) to classify the symbols. They evaluated multiple offline features and the best performance was obtained by combining the online 7-time based features and the offline features used by Pattern Recognition and Human Language Technologies (PRHLT) [1]. They achieved 87.1% and 91.24% accuracy in CROHME 2013 and 2014 test set respectively [1, 13]. Moreover, Dai et al. further extended this work through the ensemble of two classifiers, Deep Maxout Convolutional Network (DMCN) and BLSTM [7]. DMCN operated directly on the offline image generated from the online pen traces. BLSTM used a sequence of online and offline features. They used 6-time based online feature, and gradient direction offline features based on the 8-chain code direction. Their methods improved the accuracy of CROHME 2013 and 2014 test set to 87.35% and 91.28% respectively [7]. Lastly, MyScript resulted the current state-of-the-art accuracy of 92.81% for online mathematical symbol recognition in CROHME 2016 test set [12]. However, they used a large private dataset and their methodology remained unpublished [20].

On the other hand, there are fewer studies that focus only on offline handwritten mathematical symbol recognition. One reason is due to the lack of large offline mathematical symbol datasets. Therefore, most studies in offline mathematical symbol recognition conduct experiments through the usage of the CROHME dataset by first converting the online symbols to offline images. Ramadhan et al. was the first research that focused only on offline mathematical symbol recognition. They used a deep convolutional neural network, which had 2 5x5 convolutional layers with max pooling, and followed by a single MultiLayer Perceptron (MLP) layer with softmax [14]. The best

performance they achieved was 87.72% accuracy in CROHME 2014 test set. L. Dong and Liu further enhanced the work by expanding the training dataset using elastic distortion and random rotation. They also enhanced the deep learning model by using 4 3x3 convolutional layers with max pooling along with other advanced regularization techniques, such as, dropout, batch normalization, and global average pooling [9]. Their model resulted the best performance of 91.82% and 92.42% in CROHME 2014 and 2016 test set. This was the best accuracy so far for offline handwritten mathematical symbol recognition to the best of our knowledge.

4. Proposed Architecture

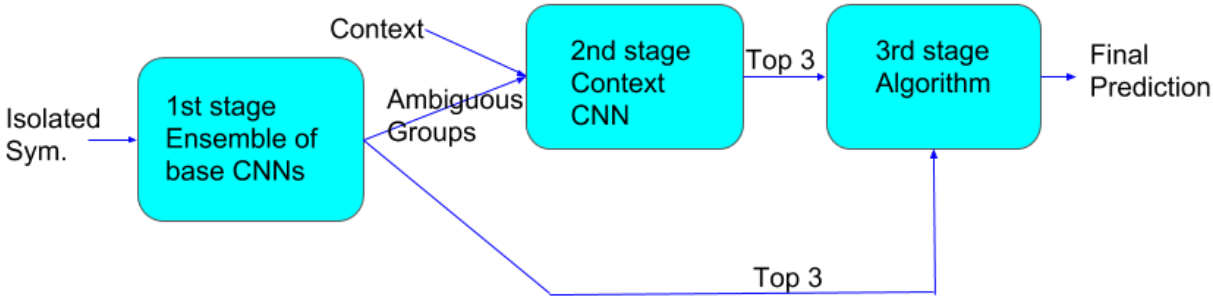

Figure 4: The overall architecture for multi-stage context-based recognition system.

In offline handwritten mathematical symbol recognition, we assume that both the isolated symbol and its corresponding mathematical expression images are inputs to our recognition system. This overall architecture consists of three main stages. The first stage uses an enhanced version of the existing state-of-the-art CNN model for isolated symbol classification. This classifier serves as the base classifier of the whole architecture that recognizes the symbols only based on its isolated symbol information. If the predicted symbol class does not belong to one of the ambiguous groups, then its top 1 and top 3 predictions will be used directly in the third stage for the final prediction. Otherwise, this symbol will be fed to the second stage classifier where its context information is also incorporated. The top 1 and top 3 predictions from the second stage are used in the third stage instead. We identify two ambiguous groups based on the classification error in the

first stage. In the second stage of context-based classification, we train and utilize a similar CNN model in the first stage to classify only the ambiguous symbols [17]. Finally, we collect all the top 1 and top 3 classification results from either the first or the second stage, and then we develop an algorithm to select the best prediction based on a set of rules. This algorithm is used as the last stage in the overall architecture. Figure 4 illustrates the overall multi-stage context-based recognition architecture.

4.1 1st Stage: Isolated Symbol Classification

In the first stage of isolated symbol classification, it consists of three steps: training data expansion and balancing, model training, and ensemble of models. In the first step of data expansion and balancing, we expand the training data using elastic distortion and random rotation [16], and balance the data so that each symbol class contains at least k number of samples, where k is a parameter of choice. In the second step of model training, we train a deep convolutional neural network using the expanded training samples and validate on the validation set. In the last step of ensemble of models, we create multiple models by repeating the first two steps, and finally, and the final classifier is the ensemble of these individual models.

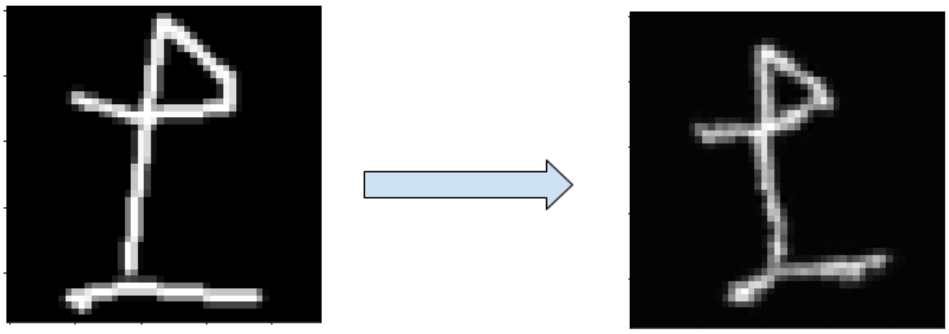

Figure 5: A “plus-minus” symbol after elastic distortion and rotation of 12 degree.

During the first step of training data expansion, we expand the training dataset by using elastic distortion and random rotation [16]. The amount of augmented data to generate per symbol class is controlled by a parameter k. This parameter k will be used in the last step. If the total number of training samples for a symbol class is less than k

thousands (minimum threshold), we randomly sample a data from that symbol class, and then apply the image augmentation of elastic distortion and random rotation to create a new distorted sample for that symbol class until the total number of samples for that symbol class equals to this minimum threshold. Otherwise, there is no image augmentation for that symbol class. This is a very similar balancing method used in [8]. The parameters chosen for the elastic distortion are: 11x11 gaussian kernel with standard deviation (sigma) of 5 and elasticity coefficient (alpha) of 12. The elasticity coefficient is used to control the among of the distortion [16]. The random rotation is between -25 to 25 degree. Figure 5 shows an example of image after elastic distortion and rotation.

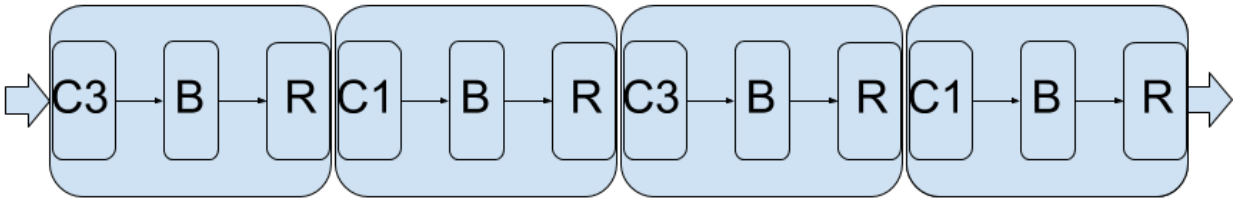

Figure 6: A convolutional block (CB) consists of 4 3-layer-blocks with convolutional layer and batch normalization (BN) and ReLU activation. (C3 = 3x3 Conv. C1 = 1x1 Conv. B = BN. R = ReLU).

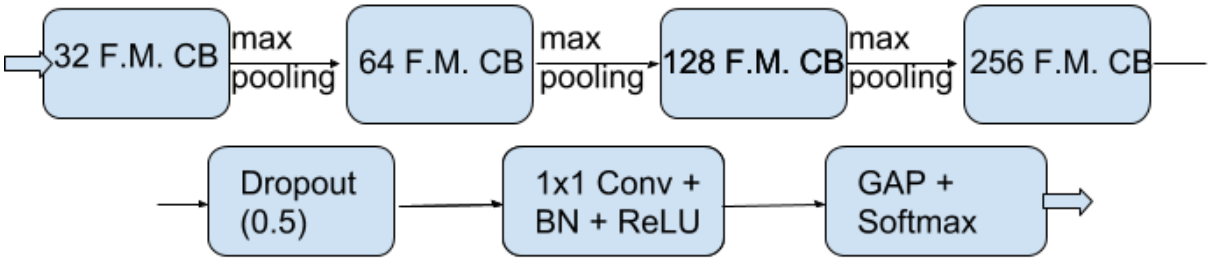

Figure 7: An end-to-end deep learning model with 4 convolutional blocks (CB) and global average pooling (GAP). Each CB doubles number of feature maps (FM) from the previous layer.

In the model training step, we create a deep convolutional neural network as the classifier. The fundamental building block of our deep convolutional neural network is illustrated in Figure 6. It consists of 4 convolutional layers followed by a 2x2 max pooling. The first and second convolutional layers are the same as the third and fourth

convolutional layers. The first convolutional layer uses a 3x3 kernel with n numbers of feature maps (F.M.), where n is a parameter of choice. The second convolutional layer uses a 1x1 kernel with also n number of feature maps. The reason of the usage of the 1x1 convolutional layer is to increase the depth of the network, and thus, it can provide more representational power to the network. The full convolutional neural network model is created by chaining 4 of these fundamental convolutional blocks (each with different choice of n), and then followed by a global average pooling (GAP) with softmax. Figure 7 presents the architecture of the full deep learning model. Notice that the number of feature maps are doubled after each 2x2 max pooling. The first block uses n = 32, the second block uses n = 64, etc. The loss function is the standard categorical cross entropy loss. The full model is trained using gradient descent with Adam optimizer in Keras with Tensorflow backend. Model weights initialization and other training parameters (e.g., learning rate) are the default parameters given in Keras documentation [5].

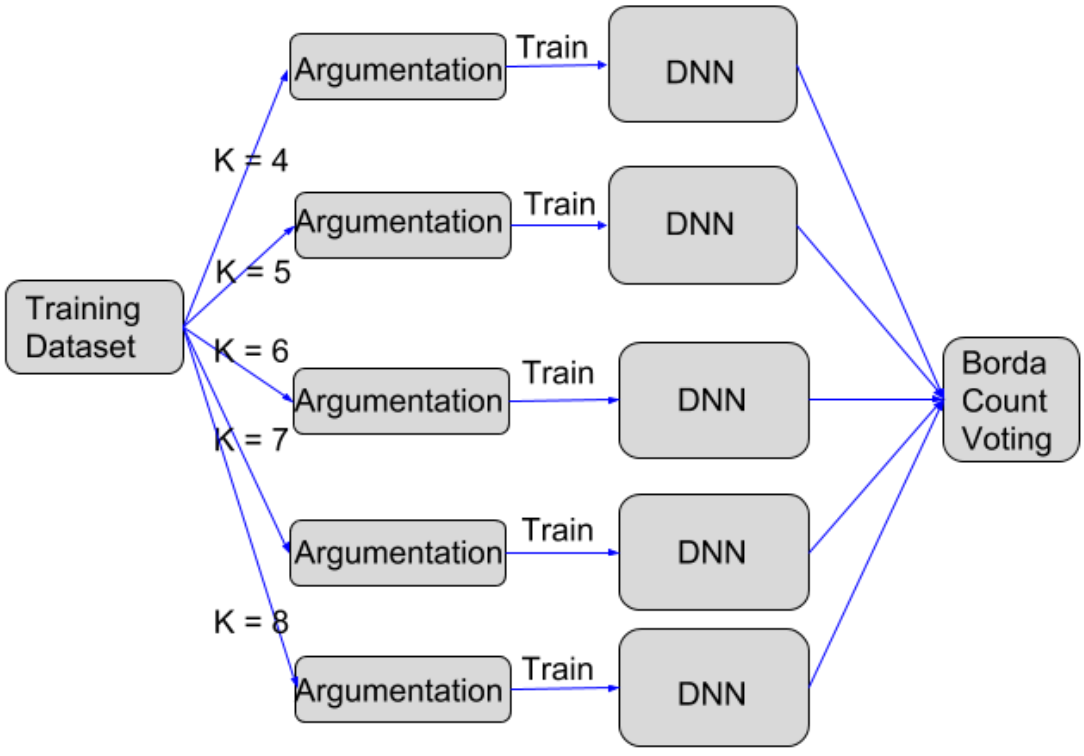

Figure 8: The MCDNN architecture is used in the experiments.

In the last step of model assembling, we train multiple models by using different amount of training samples through different expansion parameters, k = 4, 5, 6, 7, and 8 thousand respectively. Each time we train a new model, we regenerate the training samples directly from the base training set, that is, when we expand the samples with k=5, we do not use the previously expanded train samples of k=4 because this can create double distorted sample from an already distorted sample. Then, the final prediction is made by combining the outputs of these models. This ensemble method is similar to the concept of bagging [4], and the overall structure forms an MCDNN architecture [6]. Figure 8 shows the overview of this MCDNN.

For each expanded training samples, we train an instance of the CNN model with 20 epochs because after approximately 10 epochs, there is no further accuracy increase to the validation set. We then select two trained model instances that give the two highest accuracies for the validation set. Thus, we have a total of 10 trained models. The top 1 and top 3 predictions are made by combining all 10 models using the novel Borda Count voting [3].

4.2 2nd Stage: Context-based Classification

In the second stage of context-based classification, since we are only classifying ambiguous symbols, for each ambiguous symbol group, we train a deep CNN to classify that symbol group. Although there could be more ambiguous groups, we use only the two most significant ambiguous groups in our dataset during the experiments. Since we use two ambiguous groups, there are total of two CNNs in this stage. The first group is x-like group, containing “X”, “x”, and “\times”. The second group is 1-like group, containing “1”, “|”, “)”, and “/”. Even though the definition of the ambiguous group could be data dependent, this context-based classification approach can be generalized to any dataset once such ambiguity is identified. In our experiments, these two ambiguous groups are identified by analyzing the top errors from the first stage due to symbol ambiguity [9].

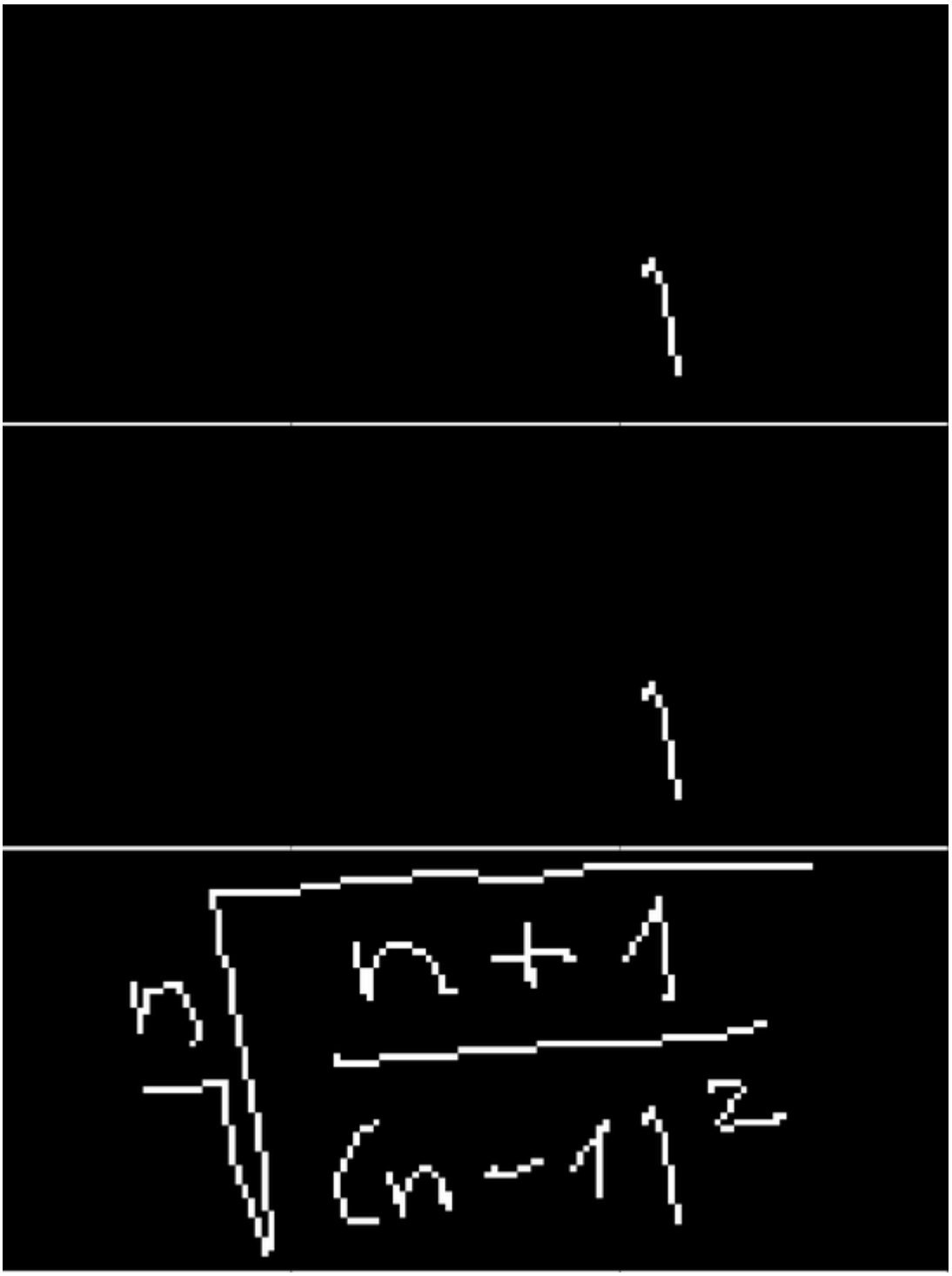

Figure 9: A 256x64 3-channel symbol with context information. Each RGB channel (from top to bottom) is illustrated separately. When viewing it as a regular RGB image, since the last channel (blue) consists of the full expression, non-targeted symbols will

appear in blue color. The first and second channels (red and green) contain only the target symbol so its color appears in white.

We encode the context information in the following way. For each ambiguous offline handwritten mathematical symbol to classify, we generate a three-channel image from the dataset. Because most mathematical expressions have width much greater than its height, we choose an image of resolution 256x64. The last channel of the image consists of the grey scale handwritten mathematical expression where the symbols belong. The first and second channels are identical grey scale images that contain only the symbol itself as other symbols in the same expression are hidden. Figure 9 illustrates an example of a such image. The training/validation data is generated by the same manner. It selects a subset from the original training/validation data, where the subset only contains the ambiguous groups.

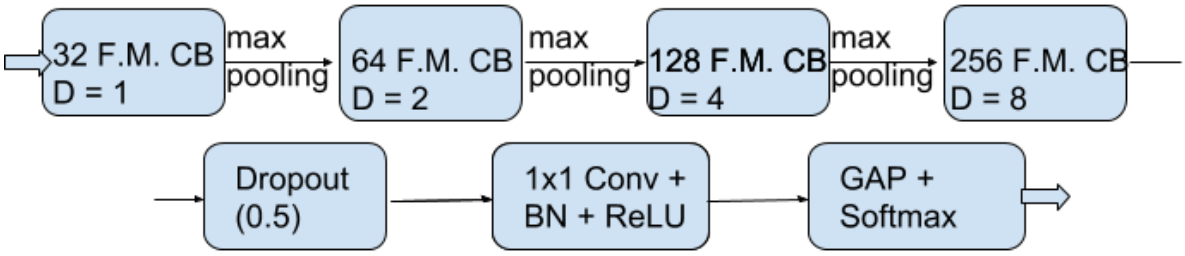

Figure 10: Dilated CNN model for context-based ambiguity classification. Dilation rate (D) doubles from the previous layer to increase the field of view.

Figure 10 shows the CNN model used in this second stage of context-based classification. This CNN model is the same as the CNN used in the first stage except the dilation rate for the purpose of increasing the receptive field on the convolutional layers. Dilated convolution is a very effective way to increase the field of view of the convolutional filter without increasing the kernel size directly [2]. Increasing the field of view of the convolutional kernel can help the neural network capture features for a large area on the input images. By definition, the regular convolutional layer has default dilation rate of 1. This CNN model doubles the dilation rate in each subsequent

convolutional block. The model is also trained with the standard categorical cross entropy loss through gradient descent with Adam optimizer in Keras with Tensorflow backend. Weights initialization and other training parameters (e.g., learning rate) are the default parameters given in Keras documentation [5].

4.3 3rd Stage: Prediction Selection Algorithm

Once the CNN models from stage 1 and stage 2 are finished training, during the test phase, the top 1 prediction from stage 1 will first be used. If the top 1 prediction for a symbol does not belong to any of the ambiguous symbol groups, then its top 1 and top 3 predictions will be used directly by the third stage. Otherwise, the top 1 and top 3 predictions from stage 2 will be used instead. The prediction selection algorithm in this stage will select the best prediction from the top 3 predictions. The inputs to the algorithm will be the following: top 3 predictions for the targeted symbol, top 1 predictions for other symbols from the same mathematical expression of the targeted symbol, and their location information related to other symbols in the expression. The location information is encoded using the bounding box of the symbols. In the dataset of our evaluation, the bounding box information can be determined by finding the starting and ending pixel positions of columns (width) and rows (height) of the symbol through scanning the symbol-only channel of the expression image (e.g., the top image in Figure 9). Then the bounding box coordinates are normalized between 0 and 1. Without loss of generality, we assume bounding box can be represented by four points (x0, y0, x1, y1), which corresponds to starting and ending positions in width and height in its mathematical expression image. The output of the algorithm is the final prediction for the targeted symbol.

The algorithm works as follow. First, the left and right neighbors of the targeted symbol are identified through the following way. For all the symbols that overlap with the targeted symbol in y-direction, all symbols that have smaller x-coordinate midpoint values than the targeted symbol's x-coordinate midpoint value are defined as the left

neighbors. Similarly, all symbols have larger x-coordinate midpoint values are defined as its right neighbors. Second, for the right neighbor that is immediately to the right of the targeted symbol, it is also identified as one of the following categories: superscript, subscript, and neither, based on their relative position. For it to be considered as a superscript or subscript, it must have smaller height compared to the height of the targeted symbol. Then, if both the length of the right symbol's portion that is above the targeted symbol and the length of the targeted symbol's portion that is below the right symbol are greater than some percentage (threshold) of the right symbol's height, it is defined as the superscript. Similarly, if both the length of the targeted symbol's portion that is above the right symbol and the length of the right symbol's portion that is below the targeted symbol are greater than some percentage (threshold) of the right symbol's height, it is defined as the subscript. The percentage of choice are 20% for both cases. We summarize the algorithms below. The detailed pseudo-code is also provided in Appendix A.

Algorithm 1: Identification of Superscript, Subscript, or Neither.

Input: Bounding boxes of two symbols in the same expression.

Procedure: If the symbol in the right appears at the top or bottom to the symbol in the left, then the right symbol is determined as a superscript or subscript respectively. Otherwise, it is neither.

Algorithm 2: Identification of the Left and Right Neighbors.

Input: Bounding boxes of the target symbol and other symbols in the same expression.

Procedure: For all other symbols that overlap vertically with the target symbol (so that we don't consider symbols separated by fraction), symbols to the left or right of the target symbol are marked as left or right neighbors respectively.

Finally, once the above neighboring information is identified for the targeted symbol, the algorithm will select the final prediction based on whether the prediction will result a meaningless expression. For example, for a targeted symbol, if the highest confident prediction from its top 3 predictions is "(", but no right neighbors have top 1 prediction as ")", then the algorithm will select the next highest confident prediction, because otherwise the resulting expression will most likely have an open parenthesis but no close parenthesis – a meaningless mathematical expression. Based on this idea, we have developed a list of rules for the algorithm to select the best prediction. Appendix B gives more details about each rule and the reasons why these rules are selected. Although there will be more general mathematical symbol rules that can be applied, we use only a few simple rules in our experiments to demonstrate the effectiveness of our overall architecture. We summarize the steps of the prediction selection algorithm below. The detailed pseudo-code is also listed in Appendix A.

Algorithm 3: Prediction Selection.

Input: Top 3 predictions of the targeted symbol. Top 1 prediction of other symbols in the same expression.

Procedure: Based on the neighboring information from Algorithm 1 and 2, if the prediction with highest confidence in the top 3 list will result a meaningless mathematical expression, then select the prediction with second highest confidence. Otherwise, select the prediction with the highest confidence.

5. Evaluation Dataset – CROHME

There two categories in handwritten mathematical symbol recognition in the highest level: offline and online. For the offline case, the input symbol is presented as a grayscale image. For the online case, the input symbol is given as a list of digital pen trajectories over times. From online symbols, we can generate its offline image by

connecting its trajectories to an image. Therefore, offline symbol recognition technique can also apply to online symbol recognition, but not vice versa.

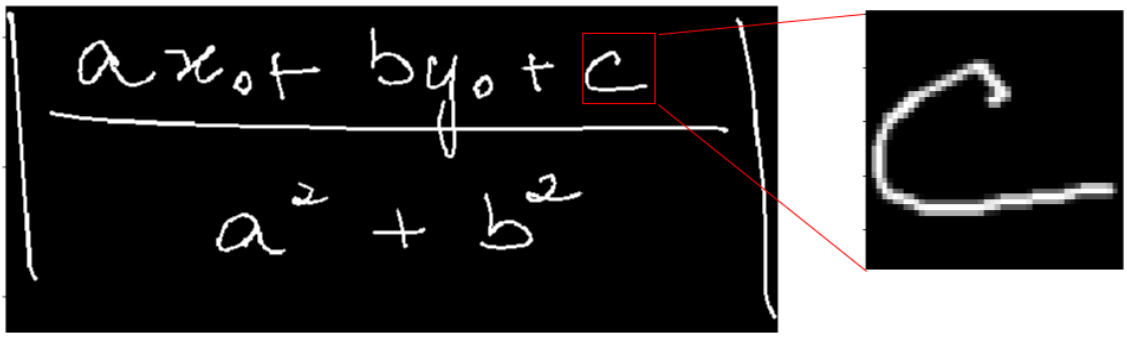

Figure 11: An example of offline handwritten mathematical expression and an extracted isolated symbol from the CROHME dataset.

During the evaluation of the architecture, the main dataset used is the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) dataset. The CROHME dataset consists of more than 10,000 online handwritten mathematical expressions [12, 13]. There are total 101 different symbol classes. This competition dataset consists of 5 different tasks: Task 1, formula recognition. Task 2a, isolated symbol recognition without invalid symbol (junk). Task 2b, isolated symbol recognition with invalid symbol. Task 3, structure recognition. And finally, task 4, matrix recognition [13]. In this research, we focus only on the offline handwritten mathematical symbol recognition, and we will use the dataset for task 2a to evaluate our methods. We extract only the valid isolated symbols from the CROHME expressions, and then convert them to the offline symbol images using the method proposed in [9], which the pen trace coordinates are normalized in a given pixel range, and then connecting the coordinates with lines using OpenCV open source library. Figure 11 illustrates an example of a handwritten mathematical expression and an extracted isolated symbol from that expression.

	Dataset	# of Isolated Symbols	# of Expressions
Train	CROHME 2013 train	85782	8834
	HASYv2 (part)	43648	-
Validation	CROHME 2013 test	6082	986

Test	CROHME 2016 test	10019	670

Table 1: Dataset Usage in Experiments

During training, we also mix the CROHME training set with a subset of the Handwritten Symbols (HASY) v2 dataset as proposed by [9]. The HASYv2 dataset consists of images of handwritten symbols from 369 different classes [18]. Among all 369 symbol classes, 85 classes are the same in the CROHME datasets. It is worth to note that the isolated mathematical symbols in CROHME 2016 test set for task 2a are the same with the CROHME 2014 test set [12]. They are both extracted from the same set of the mathematical expressions in CROHME 2014 dataset [12]. The reason that CROHME 2016 test set for task 2a contains a smaller number of isolated symbols (10019 vs.10061) than the CROHME 2013 is because CROHME 2013 contains wrongly extracted symbols. For example, an isolated "\ldots" symbol that contains 3 strokes is wrongly extracted as 3 "\ldots" symbols with 1 stroke each. Therefore, in this research, we use our own program to extract the isolated symbols from the CROHME 2014 mathematical expressions. The number of isolated symbols created matches with the CROHME 2016 test set. Table 1 summarizes the usage of the datasets in this research. The first and second stage of the proposed architecture are trained with CROHME 2013 training set plus part of the HASYv2 dataset (only the first stage training is mixed with HASYv2), and use CROHME 2013 test set as the validation set. Finally, our trained model is evaluated using the CROHME 2016 test set. We do not evaluate our model with the CROHME 2014 test set due to the above reason.

6. Evaluation and Results

Train Data Expansion with k	Best 2 Validation Accuracy	Test Accuracy
K = 4	88.66%	91.86%
	88.49%	90.68%
K = 5	88.92%	91.87%
	88.80%	91.47%
K = 6	88.84%	91.65%

	88.82%	91.65%
K = 7	88.57%	91.37%
	88.54%	91.65%
K = 8	88.56%	91.23%
	88.39%	91.65%
Combined ALL	**89.82%**	**92.58%**

Table 2: Validation and Test Accuracy of First Stage MCDNN Model with Class Balancing

The results of the MCDNN model in the first stage has been summarized in Table 2. The best two validation accuracies are presented for each iteration of training data expansion (k = 4, 5, 6, 7, and 8). We use the borda count voting method to combine all the trained models. The results indicate that the combined model gives a validation and test accuracy of 89.82% and 92.58% for the CROHME 2013 and 2016 test set. It is worth to note that the whole process of the end-to-end model training and the final model selection is solely based on only the training and validation dataset. The test dataset is only used to evaluate the accuracy of this MCDNN model. The result indicates that the overall MCDNN model can have significant improvements compared to individual CNNs.

Classifier	CROHME 2013 Test	CROHME 2016 Test	HASYv2 10-fold (min / max / avg)	Feature Used
HMS-VGGNet [9]	88.46%	92.42%	- / - / 85.05%	Offline
First Stage MCDNN	**89.82%**	**92.58%**	85.23% / 86.08% / **85.62%**	Offline

Table 3: First Stage MCDNN Accuracy Comparison

Table 3 summarizes the comparison between the first stage MCDNN model and the current state-of-the-art HMS-VGGNet model [9]. Both the first stage MCDNN classifier and the baseline HMS-VGGNet model use only the isolated offline symbols

during training. The results indicate that the first stage MCDNN already outperforms the current state-of-the-art model by 1.36% on the CROHME 2013 test set and 0.16% on the CROHME 2016 test set. We have also evaluated the first stage MCDNN model on the HASYv2 dataset using the its predefined 10-fold cross validation [18]. The expansion parameters for this dataset are k = 600, 800, 1000, and 1200. Other training parameters remain the same. So total of 8 base CNN models are combined for each fold. Comparison on the min/max/average 10-fold accuracy is also reported. The detailed validation and test accuracies on each fold are reported under Appendix C. Because the work [9] has not reported the min and max accuracies, we skip these accuracies in the table. It is important to note that, by only utilizing the first stage MCDNN classifier, our model already achieves better accuracy in both the CROHME 2016 and HASYv2 dataset compared with previous work in [9].

Classifier	CROHME 2013 Test	CROHME 2016 Test	Feature Used
MyScript [12]	-	92.81%	Online + Offline
HMS-VGGNet [9]	88.46%	92.42%	Offline
First Stage Only	89.82%	92.58%	Offline
First and Second Stages Only	90.87%	93.62%	Offline + Context
Context-based Multi-stage Architecture	**91.37%**	**94.04%**	Offline + Context

Table 4: Context-based Architecture Accuracy Comparison with Previous Works

Table 4 summarizes the results on accuracy for each stage in the overall architecture, as well as the comparison with previous works [9, 12]. The overall context-based multi-stage architecture has achieved significant accuracy improvements over previous works. The accuracy improvement between stage one and stage two shows that the additional context information can be vital for classifying the handwritten mathematical symbols. Our multi-stage architecture is one possible way to utilize this

context information. The accuracy improvement between stage two and stage three also reflects that effective analysis of mathematical expression structure may still be necessary for classifiers to achieve higher accuracy. The major accuracy improvement (compared to previous works) is due to resolving many classification errors caused by shape ambiguity. Our overall context-based multi-stage architecture has achieved the new state-of-the-art accuracy of 91.37% and 94.04% on both the CROHME 2013 and 2016 test set, respectively.

Additional Distorted Training Samples per Symbol Class	Best 2 Validation Accuracy	Test Accuracy
500	88.67%	90.49%
	88.39%	90.33%
1000	89.20%	90.43%
	88.52%	90.97%
4000	88.13%	91.00%
	88.08%	91.20%
Combined ALL	**89.00%**	**92.02%**

Table 5: Validation and Test Accuracy of MCDNN Model without Class Balancing

We also evaluate the effect on accuracy improvement due to balancing the training dataset in the first stage. In this experiment, when we expand the training data, we create the same number of additional samples for each symbol class regardless their original counts. By creating the same number of additional samples, the bias distribution in the training dataset approximately remains the same. We train 6 different models with different number of additional samples per symbol class. Similar to the class balanced approach, we expand the training set 3 times (with 500, 1000, 4000 additional samples per class using elastic distortion and random rotation). And then, for each expanded training set, we train the same CNN model with the same parameters, and pick the two trained models that give the highest validation accuracies. Table 5 summarizes the results on the accuracy of dataset expansion without class balancing. Compared with the results in Table 2, the overall accuracies for both validation and test set are 0.82% and 0.56% lower. This experiment shows that balancing the training set can have positive impacts on the overall accuracy for the CROHME 2016 dataset. Nevertheless, this experiment also

indicates that the MCDNN model can have significant improvements over individual CNN models.

7. Conclusion

In this paper, we proposed a context-based multi-stage architecture for offline handwritten mathematical symbol recognition. In the absence of context information, the first stage of the architecture can still be used as a generalized method of training a MCDNN model for isolated symbol recognition. The experiments show that this MCDNN is a powerful tool to boost the accuracy performance of individual classifiers. This MCDNN model only uses the training and validation set during the process of model training and selection. This means the MCDNN is a generic offline isolated symbol classifier without the risk of overfitting on the test set. Often context information is available because handwritten mathematical symbols are normally written in a defined mathematical expression, then the overall architecture can utilize this context information to reduce the classification error due to shape ambiguity. Experiments show that this context-based multi-stage architecture outperforms all other previous approaches, and results the state-of-the-art accuracy on both the CROHME 2013 and 2016 dataset in offline handwritten mathematical symbol recognition. Future works include researching deep learning models/architectures to better utilize the context information as well as developing more rules for better enforcing the validity of the resulting mathematical expressions.

8. References

[1] F. Álvaro, J.A. Sánchez, J.M. 2014. Benedí. Offline features for classifying handwritten math symbols with recurrent neural networks. In 22nd International Conference on Pattern Recognition, pp. 2944–2949. IEEE Press, Stockholm.

[2] S. Bai, J.Z. Kolter, V. Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271. Retrieved from https://arxiv.org/abs/1803.01271.

[3] J.-C. de Borda. Mmoire sur les lections au 31crutiny. 1781. Oxford Univ. Press for Social Sciences.

[4] L. Breiman. 1996. Bagging predictors. Mach. Learn. 24, 2 (August 1996), 123-140. DOI:http://dx.doi.org/10.1023/A:1018054314350.

[5] F. Chollet and others. 2015. Keras. Retrieved December 12, 2017 from https://keras.io/.

[6] D. Ciregan, U. Meier, J. Schmidhuber. 2012. Multi-column deep neural networks for image classification. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (CVPR '12). IEEE Computer Society, Washington, DC, USA, 3642-3649.

[7] N.H. Dai, A.D. Le, M. Nakagawa. 2015. Deep neural networks for recognizing online handwritten mathematical symbols. In 3rd IAPR Asian Conference on Pattern Recognition, pp. 121–125. IEEE Press, Kuala Lumpur.

[8] K. Davila, S. Ludi, R. Zanibbi. 2014. Using off-line features and synthetic data for on-line handwritten math symbol recognition. In Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on, IEEE, 323–328.

[9] L. Dong, H. Liu. 2017. Recognition of Offline Handwritten Mathematical Symbols Using Convolutional Neural Networks. Image and Graphics. ICIG 2017. Lecture Notes in Computer Science, vol 10666. Springer, Cham.

[10] K. Hornik. 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks, vol. 4, pp. 251-257.

[11] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. 1998. Gradient-based learning applied to document recognition. Proc. IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[12] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, U. Garain. 2016. ICFHR 2016 CROHME: Competition on Recognition of Online Handwritten Mathematical Expressions. In International Conference on Frontiers in Handwriting Recognition (ICFHR), Shenzhen, China.

[13] H. Mouchere, C. Viard-Gaudin, R. Zanibbi and U. Garain. 2014. ICFHR 2014 Competition on Recognition of On-Line Handwritten Mathematical Expressions. In 14th International Conference on Frontiers in Handwriting Recognition (ICFHR), Greece, 2014, pp. 791-796.

[14] I. Ramadhan, B. Purnama, F.S. Al. 2016. Convolutional neural networks applied to handwritten mathematical symbols classification. In 4th International Conference on Information and Communication Technology, pp. 1–4. IEEE Press, Bandung.

[15] D. E. Rumelhart, G. E. Hinton, R. J. Williams. 1986. Learning Representations by Back-Propagating Errors. Nature, vol. 323, pp. 533-536.

[16] P. Y. Simard, D. Steinkraus and J. C. Platt. 2003. Best practices for convolutional neural networks applied to visual document analysis. In Seventh International Conference on Document Analysis and Recognition, Proceedings., 2003, pp. 958-963.

[17] K. Simonyan and A. Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. Retrieved from https://arxiv.org/abs/1409.1556.

[18] M. Thoma. 2017. The HASYv2 dataset. arXiv:1701.08380. Retrieved from https://arxiv.org/abs/1701.08380.

[19] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, R. Fergus. 2013. Regularization of neural networks using dropconnect. In Proceedings of the 30th International Conference on Machine Learning (ICML'13), pp. 1058–1066.

[20] J. Zhang, J. Du, and L. Dai. 2018. Multi-scale attention with dense encoder for handwritten mathematical expression recognition. arXiv:1801.03530. Retrieved from https://arxiv.org/abs/1801.03530.

9. Appendices

Appendix A – Algorithms Pseudo-Code

This appendix summarizes the pseudo-code of the algorithms listed under Section 4.3.

Algorithm 1: Identification of Superscript, Subscript, or Neither

Input: (1) left symbol's bounding box (x_0, y_0, x_1, y_1). (2) right symbol's bounding box ($\_x_0$, $\_y_0$, $\_x_1$, $\_y_1$). (3) Threshold percentage TH.

Steps:

85. if $y_1 - y_0 <= \_y_1 - \_y_0$, **then**

return Neither

2. $\_th = TH * (\_y_1 - \_y_0)$

3. **if** $y_0 - \_y_0 >= \_th$ **and** $y_1 - \_y_1 >= \_th$, **then**

return Superscript

if $\_y_1 - y_1 >= \_th$ **and** $\_y_0 - y_0 >= \_th$, **then**

return Subscript

4. **return** Neither

Algorithm 2: Identification of the Left and Right Neighbors

Input: (1) targeted symbol's bounding box $b_{target}=(x_0, y_0, x_1, y_1)$. (2) List of bounding boxes B (of other symbols in the same mathematical expression). (3) Threshold percentage TH.

Steps:

1. L = NotSet (left neighbor bounding box. If it is set, then it has four points. Lx_1 represents the x_1 coordinate).

2. R = NotSet (right neighbor bounding box. If it is set, then it has four points. Rx_0 represents the x_0 coordinate).

3. R_type = NotSet (Right Neighbor type: superscript, subscript, and neither)

4. _L = empty list (keep a list of left neighbors)

5. _R = empty list (keep a list of right neighbors)

6. $c = (x_0 + x_1) / 2$

7. **for** each box b = ($\_x_0$, $\_y_0$, $\_x_1$, $\_y_1$) **in** B, **do**

    if there is no overlap between (y_0, y_1) and ($\_y_0$, $\_y_1$), **then**

        continue

    $\_c = (\_x_0 + \_x_1) / 2$

    if $\_c < c$, **then**

        add b to _L

        if L is NotSet **or** $\_x_0 > Lx_1$, **then** L = b

    if $\_c > c$, **then**

        add b to _R

        if R is NotSet **or** $\_x_1 < Rx_0$, **then** R = b

8. R_type = Neither

9. **if** R is NotSet, **then return** L, R, R_type, _L, _R

11. R_type = **Algorithm1**($\underline{b_{target}}$, R, TH)

14. **return** L, R, R_type, _L, _R

Algorithm 3: Prediction Selection

Input: (1) target symbol's top3 predictions $\underline{p_0, p_1, p_2}$ (assume confident level $p_0 > p_1 > p_2$) and its bounding box $\underline{b_{target}}$. (2) Other symbols' (in the same expression) top1 prediction list P and corresponding bounding box list B. (We use the notation P[b] to denote the top 1 prediction for each b ∈B. We also define P[NotSet] = NotSet) (3) Threshold percentage TH (in our case, we use 0.2).

Steps:

1. L, R, R_type, _L, _R = **Algorithm2**(b_{target}, B, TH)

2. **if** p_0 is "\times", **then**

    if L is NotSet **or** R is NotSet, **then**

        return p_1

    if P[L] ∈Operators* (e.g., "+", "-"), **then**

        return p_1

    if P[L] is "(" or "[", **then**

        return p_1

if P[R] is “)” or “]”, **then**

return p_1

if R_type is Superscript **or** Subscript, **then**

return p_1

3. **if** p_0 is “+”, **then**

if L is NotSet **or** R is NotSet, **then**

return p_1

if P[L] ∈Operators* (e.g., “sin”, “=”), **then**

return p_1

if P[L] is “(” or “[”, **then**

return p_1

if P[R] is “)” or “]” or “=”, **then**

return p_1

if R_type is Superscript **or** Subscript, **then**

return p_1

4. **if** p_0 is “(”, **then**

for each b in _R, **do**

if P[b] is “)”, **then return** p_0

return p_1

5. **if** p_0 is “\comma”, **then**

if Algorithm1(L, b_{target}, TH) returns Superscript, **then**

return p_1

let $(x_0, y_0, x_1, y_1)=b_{target}$

if y_0 is smallest among all other symbols, **then**

return p_1

6. **if** p_0 is “g”, **then**

if L is NotSet **and** R is NotSet, **then**

return p_0

if (P[L] is “g” **or** NotSet) and (P[R] is “g” **or** NotSet), **then**

*Operators are listed under Appendix

return p_1

sort _L and _R based on $x_c = (\_x_0 + \_x_1) / 2$ for each box ($\_x_0$, $\_y_0$, $\_x_1$, $\_y_1$) in the lists:

tL = L

for each box b in descending sorted _L, **do**

if P[tL] is not ".", **then break**

tL = b

tR = R

for each box b in ascending sorted _R, **do**

if P[tR] is not ".", **then break**

tR = b

if P[tL] **and** P[tR] ∈ [numbers (0 to 9) **or** NotSet], **then**

return p_1

7. **return** p_0

Appendix B – Prediction Selection Rules

This appendix summarizes the details of the rules used in the prediction selection algorithm under Section 4.3 and Appendix A. There are five rules listed:

Rule #1: If the top 1 prediction is the "\times" operator, then change its top 1 prediction to the next highest probability in the top 3 predictions if any of the following conditions are true:

- The left symbol immediately to the left of the targeted symbol has top 1 prediction belongs to one of the following operator: '+', '-', '/', '=', '\\geq', '\\gt', '\\leq', '\\lt', '\\neq', '\\div', '\\pm', '\\in', '!', '\\rightarrow', '\\sin', '\\tan', '\\cos', '\\lim', '\\log', '\\int'. Because two operators cannot be next to each other in a mathematical expression, this must be either "x" or "X".

- The left symbol immediately to the left of the targeted symbol has top 1 prediction either "(" or "[". It is not possible to have an open bracket/parenthesis followed by a "\\times" symbol. It must be either "x" or "X".

- The right symbol immediately to the right of the targeted symbol has top 1 prediction either ")" or "]", or it is defined as a superscript or subscript. It is not possible to have superscript or subscript next to the "\\times" symbol. It must be either "x" or "X".

- There is no left or right neighbor symbols. The "\\times" symbol must have operands next to it.

Rule #2: If the top 1 prediction is the "+" operator, then change its top 1 prediction to the next highest probability in the top 3 predictions if any of the following conditions are true:

- The left symbol immediately to the left of the targeted symbol has top 1 prediction in one of the following: "'[', '(', or one of the follow operators: '/', '\sin', '\cos', '\tan', '='". This is based on the similar reasons in above as "+" is a very similar operator to "\times".

- The right symbol immediately to the right of the targeted symbol has top 1 prediction in one of the following: ")", "]", and "=", or it is defined as a superscript or subscript.

- There is no left or right neighbor symbols. The "+" symbol must have operands next to it.

Rule #3: If the top 1 prediction is "(" symbol, but there is no top 1 prediction from its right neighbors which are ")" symbol, then change this top 1 prediction to the next highest probability in the top 3 predictions.

Rule #4: If the top 1 prediction is "comma" symbol, change its prediction to the "\prime" (derivative) symbol if any of the following conditions are true:

- The targeted symbol is defined as a superscript to its immediately left symbol.
- The targeted symbol is located at the very top of the mathematical expression.

Rule #5: If the top 1 prediction is a "g" symbol, it could be ambiguous to "9". Change its prediction to "9" if any of the following conditions are true:

- If the immediately left and right symbols to the targeted symbol have top 1 predictions as one of the numbers (0 to 9). Also, if the left or right symbol is predicted as ".", then this neighbor symbol is skipped. This is most likely a number in the mathematical expression other than mixing number and the letter "g".

- If the immediately left and right symbols to the targeted symbol have top 1 predictions as letter "g", change its prediction to number "9". This is because number "999" is much more frequent than "ggg" in mathematical expressions.

Appendix C – 10-fold Cross Validation Accuracies for the HASY v2 Dataset on the First Stage MCDNN Architecture.

This appendix summarizes the accuracies for the 10-fold cross validation for the HASY v2 dataset. k=600 means that for each symbol class that has less than 600 training samples, we expand the training samples for that symbol class to 600 samples using data augmentation described in Section 4.1.

Fold	1	2	3	4	5	6	7	8	9	10	Avg
k=600	85.04%	85.15%	84.86%	84.49%	84.62%	84.60%	84.67%	84.47%	84.58%	84.89%	84.74%
k=800	85.37%	85.34%	84.79%	84.65%	84.49%	84.95%	84.60%	84.69%	84.60%	84.78%	84.83%
k=1000	85.28%	85.54%	84.75%	84.72%	84.50%	84.59%	84.76%	84.37%	84.98%	84.85%	84.83%
k=1200	85.05%	85.12%	84.76%	84.65%	84.34%	84.64%	84.33%	84.57%	84.61%	85.13%	84.72%
Combined	**86.03%**	**86.08%**	**85.56%**	**85.48%**	**85.34%**	**85.54%**	**85.66%**	**85.23%**	**85.51%**	**85.81%**	**85.62%**

Table 6: 10-Fold Cross Validation Accuracies for HASYv2