2021 5th International Conference on Electrical Engineering and Information & Communication Technology (ICEEICT)
Military Institute of Science and Technology (MIST), Dhaka-1216, Bangladesh

Recognition of Basic Handwritten Math Symbols Using Convolutional Neural Network with Data Augmentation

Rafi Ibn Sultan
Department of Computer Science and Engineering,
Varendra University
Rajshahi, Bangladesh
rafi.ruet13@gmail.com

Md. Nahid Hasan
Department of Computer Science and Engineering,
Varendra University
Rajshahi, Bangladesh
nahid12cse@gmail.com

Mohammad Kasedullah
Department of Computer Science and Engineering,
Varendra University
Rajshahi, Bangladesh
kasid.raj@gmail.com

***Abstract*—In the ever-updating digital world, automatic handwritten math symbols classification (HMC) plays many vital roles in the advancement of computer-aided systems. It is the main foundation of perfecting one of the most challenging tasks out there: recognizing handwritten mathematical formulas. As with the other similar automated handwritten characters classifications tasks, HMC also faces various difficulties while attempting to correctly classify images. As people tend to have distinct types of handwriting styles and unique ways to write symbols, a simple character may have infinite versions of itself. In our research, we focused on the classification of such images of numerous handwritten mathematical symbols. For this classification, we have developed a convolutional neural network (CNN) model and worked with three different datasets to test our model's efficiency. We introduced different data augmentation techniques to construct various versions of the already available images. This created a virtual mimicry of people's tendency to write the same character in many styles. Our CNN model of 11 layers (6 were convolutional layers) worked to classify 16 classes (each denoting a mathematical symbol or digit) and had an accuracy of 98.71%, 99.01%, and 99.85% respectively on three publicly available datasets. To our knowledge, our model performed better than every other research work in this field. Considering this remarkable success, we are bent on working further on this and creating a fully working app that would eventually be able to automatically classify handwritten mathematical formulas.**

Keywords— Handwritten math symbols recognition; Classification; CNN; Deep Convolutional Neural Network; Data augmentation; Pre-processing; Optimization; Multiple Datasets

I. INTRODUCTION

Handwritten symbol recognition (HMC) is a subset of the vast field of handwriting character recognition. This is particularly important for scientific and educational purposes as these symbols are used very commonly in various mathematical formulas and equations. Researchers have been working on automatic recognition of mathematical notation for about half a century now [1]. Due to the popularization of modern touch-based or pen-based electronics where people can easily scribble mathematical formulas, this has been researched actively by scholars of the field [2]. Despite so many years of ongoing research, existing systems are still far off from perfect. This work focuses on addressing a subset of automatic mathematical notation recognition by tackling the issue of mathematical symbol recognition.

This research focuses on offline recognition [3] of characters where two-dimensional images of handwritten character images are loaded for the recognition process. The final goal of this research work is to create a working app that would be able to recognize a handwritten mathematical formula from scanning or capturing an image containing such formulas. Typically, a handwritten mathematical formula consists of a sequence of characters that are to be segmented and recognized before deciding on the formula. In this paper, we focused on the more challenging task of the two domains, classifying the segmented basic handwritten math symbols.

In any handwritten character recognition, the task of automatic recognition gets complex due to the inherent structural similarities between the characters and the numerous appearances and forms each one can take [4]. In the case of mathematical characters, this is no different. There are roughly over 300 classes of symbols. At times many of those symbols are distinguished from one another by a simple dot or a bar. Any classification algorithm can get confused due to this structural similarity. In this paper, we have worked with 16 basic symbols: the ten digits (0-9), the mathematical signs (addition: +, subtraction: -, multiplication: ×, division: ÷), the decimal sign (.) and the equal sign (=). A Convolutional Neural Network (CNN) is developed as the classifier to classify these above-mentioned 16 classes of symbols.

The model has been successful in recognizing math symbol images from one of the 16 labels. The model has outperformed the performance of the recent research in this field. Implementing this on a larger scale will be a great step towards greater success.

978-1-6654-9522-6/21/$31.00 ©2021 IEEE

II. Related Works

In this section, we skim through the recent research works that have been done on handwritten math symbols classification following various approaches. The most recent work to date would be Ayeb et al. [5] using the transfer learning technique on various pre-trained CNNs to achieve 83.68% accuracy on a comparatively large dataset of 101 class labels. Working with the same dataset, Nguyen et al. [6] proposed a bidirectional recurrent neural network for segmenting and classifying, resulting in a final accuracy of 92.30% in classification. Like the previous work [6], Nazemi et al. [7] approached to solve this issue by utilizing two pre-trained CNN models LeNet and SqueezNet, reaching a final accuracy of 90% while evaluating the model on the test dataset. Dong et al. [8] constructed a novel CNN model named HMS-VGGNet for this classification task. The model earned a maximum of 92.42% Top-1 accuracy on the datasets they chose for testing.

III. Proposed Methodology

The rudimentary stages of our proposed method for this research work are illustrated here in fig. 1. These phases shown in the figure will be explained in the following sections of the paper in due time.

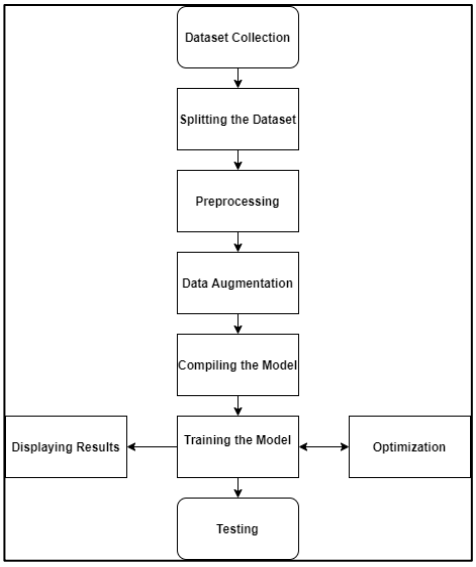


Fig. 1. Proposed methodology

IV. Data Source

A. Sample Collection

In this work, we have used three different public datasets of handwritten math symbols. To identify them throughout the paper, we will be naming them Dataset I [9], Dataset II [10], and Dataset III [11] respectively. Each of the datasets contained a significant number of images where each image belonging to a particular handwritten symbol class.

Dataset I contained a little over 9000 images, each attributing to one of the 16 class labels of handwritten math symbols that we discussed earlier. Each class contained about 500 image samples with a resolution of 400x400 pixels. Dataset II consisted of a total of 8,567 images of the 16 class labels, common to the previous dataset. These images had a resolution of 155x155 pixels contained in JPEG format. Dataset III is significantly larger than the previous two datasets. It had over 100000 images of 82 different labels. Each image was in JPEG format with a 45x45 pixels resolution.

Dataset III was initially extracted and parsed into this isolated image dataset from a well-known public dataset: CHROME 2014 [12]. To correlate with the other two datasets, we have decided to work with only 15 labels (that were mutually inclusive of the other datasets) from the available 82 labels. Although Dataset III had a vast number of classes, 15 of them were common to the previously mentioned 16 classes. The class label Decimal (.) was unavailable here, so we decided not to include it in there.

To have a more understanding of the datasets, one random image from each of the classes available from each dataset is illustrated in fig. 2.

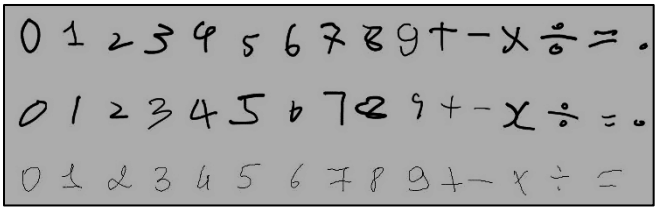


Fig. 2. Random images from each label of every dataset.

B. Datasets Splitting & Preprocessing

At the initial state, the datasets were not ready to be fed through the mode. We had to split each of the datasets into three parts: training, validation, and test sections. The working method of the proposed research was to use training and validation datasets to train the model. Later, the weights gained during training were saved. In the next step, a separate dataset i.e. the test dataset was used to evaluate our model's performance. Therefore, using different sets of images for compiling and evaluating made sure that no bias was involved in the entire process and the result we showed.

Dataset I was split into three sections: 80%, 15%, and 5% of the total images. These three parts were selected to be train, validation, and test datasets, respectively. Dataset II was already split into train and validation portions when we collected it. We further split the training dataset and took 8% of its training images to make the test dataset. For Dataset III, we took the 15 labels that we discussed earlier from the available 82 labels and ignored the rest. Again, this was also split into 80%, 15%, and 5% for the three major parts, respectively. The summary of the three datasets (after the split) is illustrated more graphically in fig. 3.

One of the reasons we chose CNN as the algorithm for this work is that it needs very little preprocessing of data compared to other classification techniques available [13]. After the datasets were split into desired slices of our choice, a little bit of preprocessing was needed to be done. First, all the images from the three datasets were resized into 50x50 pixels. We finalized these dimensions after tweaking and trying with various resolutions.

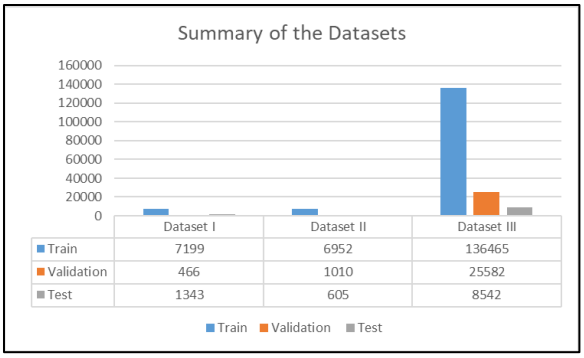


Fig. 3. Summary of the datasets

Picking a reasonable resolution number for the images without compromising the results was one of the main prerequisites before the training. Images of lower resolution tend to possess lower details. As a result, some key features could have been missed while extracting if we had resized them too low. Again, if we had taken comparatively larger images, it may have guaranteed more features, but it would have also made the training process very slow. After much consideration and different tryouts, we picked this resolution (50x50) as we believed this would not only be large enough to provide enough details but also not be too large that the training would take an indefinite time.

After the resizing was done, we constructed matrices from each of the train, validation, and test portions of the datasets. Next, we normalized the images by converting the pixel values in the range between 0 and 1 by dividing each pixel value by the number 255 (which is the highest value a pixel can have). Then each matrix was converted to numerous one-dimensional arrays where each array represented the values of a particular image. The class label of each image was also present at the end of those arrays to be used while training the data. After that, (x_train & y_train) arrays were created that represented the training dataset, where the values of the images were put in the rows of x_train. The array y_train denoted the class label values of the images sequentially. Similarly, (x_val & y_val) and (x_test & y_test) were created to represent validation and test datasets respectively. From these arrays, (x_train & y_train) and (x_val & y_val) were used for training the model and (x_test & y_test) were used for the final evaluation of the model's performance.

C. Data Augmentation

Data augmentation technique is the process of creating artificial data by modifying the available ones. This modification may include different versions of shifting, rotating, scaling, flipping, etc. Although CNN is generally invariant to such modifications while training, such data augmentation techniques make the model more robust during feature extraction [14]. As handwriting differs from person to person, a single symbol is typically written in numerous styles of choices. The fundamental features that distinguish one symbol from another are naturally the same for a particular symbol. However, they vary in styles and strokes from person to person. This type of variation is in a way similar to the principles of data augmentation. With this thought in mind, we used different data augmentation techniques such as 30-degree rotation, 20% zoom, different flips, ZCA whitening, etc to our already available data. With augmentation, we found the accuracy to be increased by a significant amount compared to the data without augmentation. Therefore, we decided to keep these augmentation techniques to create diversity in the available data.

V. The CNN Model

A. Building the Model

CNN has been the forerunner algorithm in different image classification tasks for quite a while now [15]. CNN's unique ability to learn features automatically from a little preprocessed data makes it an in-demand algorithm in computer vision challenges [9]. CNN is an advanced version of the ever-popular Artificial Neural Network (ANN) which incorporates the concept of convolutions to extract features. Just like ANN, CNN also mimics the human brain's "learning abilities" while training the model to classify images. A typical CNN consists of multiple layers such as an input layer, convolutional layers, max pooling layers, fully connected layers, etc. All these layers work up towards the final output layer to classify the inputted image to a particular class [4].

A novel CNN architecture was constructed to train the data. In this step, the built model was loaded from the disk for training purposes. The CNN model that was developed for this research had a total of 6 convolutional layers and a max pooling layer after every 2 convolutional layers. There was also a fully connected layer just before the output layer. The model is depicted in fig. 4 for further illustration. The layers in the proposed CNN model are described below:

1. **Convolutional Layer 1 & 2:** As the name suggests, Convolutional layers are the core parts of any CNN model. It works by placing filters over an array of image pixels. The resultant values are feature maps that are fed into the next layer. If x denotes the inputs, w is the filter weights, and b is the bias term then the convolutional equation would be:

$$y = w^T x + b \quad (1)$$

This first convolutional layer took images of shape 50x50x3. For the convolution purpose, 64 filters (size of 3x3) were used in this layer. An activation function i.e. ReLU (Rectified Linear Unit) was used to introduce non-linearity to the model. ReLU activation function is solely responsible for converting the resultant values after convoluting into outputs for the next layer. The ReLU activation function can be expressed as the following function:

$$f(x) = \max(0, x) \quad (2)$$

The feature map generated in this layer had the shape of 50x50x64, where 64 denotes the number of filters used. The "same padding" technique was used so that the output shape remained the same. For initializing a bias term, the default value 'zeroes" was selected. The second convolutional

layer was identical to the first convolutional layer with the same setup.

2. **Max Pooling Layer 1:** Max pooling layers are responsible for reducing the sample size of a particular feature map i.e. outputs from the previous layer. This layer consequently is responsible for reducing the number of parameters and avoiding the overfitting problem while training the data. The first max pooling layer (pool size = 2x2) reduced the shape to exactly half (25x25x64). A dropout [12] of the score (0.5) was introduced at the end of the layer. Dropout randomly sets a subset of the neurons to zero in a particular layout. Therefore, only a subsection of the original network was used and aided in reducing the overfitting problem as well.
3. **Convolutional Layer 3 & 4:** Same as the first two convolutional layers, both layers also had the same setup except for the number of filters. The third and fourth convolutional layers had 128 filters for convolution. In a similar fashion to the previous layers, this step converted the current input shape to 25x25x128.
4. **Max Pooling Layer 2:** After two convolutional layers a max pooling layer was introduced throughout the model. This max pooling layer also had the identical setup as the max pooling layer before and reduced the current output to a size that was exactly half (12x12x128).
5. **Convolutional Layer 5 & 6:** The fifth and sixth convolutional layers continued to use the same setup as previous convolution layers of the model. The filters were again increased here into 256. The output shape was then turned into 12x12x256.
6. **Max Pooling Layer 3:** This max pooling layer was a carbon copy of the previous pooling layers resulting in the new output shape into 6x6x256.
7. **Fully Connected Layer:** The final layer before the output layer was a fully connected layer of 128 neurons. We flattened the previous layer's output to create a single long feature vector which then subsequently passed to the fully connected layer.
8. **Output Layer:** An activation function i.e. Softmax Function was used in this layer that calculated the net output of the 16 neurons (or 15 for the third dataset). From the probability score generated for each of the nodes, the highest value of a class would determine the image to be belonging to that class label.

The total learnable parameters of the model are 2,327,248. The summary of the model is given in table 1.

B. Compiling the Model and Optimization

To facilitate both the training and compiling processes of the model, we took the help of some optimization techniques. Instead of the filters having random weights in each convolutional layer, the Xavier Initialization [17] method was used to initialize the weights at first. Using this method, it was possible to take the weights from a uniform distribution rather than a random distribution of data samples.

Moreover, while compiling the model we utilized the Adam optimization [18] technique to provide an adaptive learning rate throughout the training process. The learning rate oversees the convergence rate of updating the weights of the filters. Instead of having a fixed global learning rate (the classical stochastic gradient descent), Adam provides an adaptive learning rate that keeps updating carefully considering the moving average of the gradient. This ensures individual learning rates are provided for different parameters of the model. As a result, we were released from the burden of experimenting with different learning rates before finalizing one.

TABLE I. SUMMARY OF EACH LAYER OF THE PROPOSED MODEL

Layer	Output Shape	No. of Parameters
Convolution 1	(None, 50, 50, 64)	1792
Convolution 2	(None, 50, 50, 64)	36928
Max Pool 1	(None, 25, 25, 64)	0
Convolution 3	(None, 25, 25, 128)	73856
Convolution 4	(None, 25, 25, 128)	147584
Max Pool 2	(None, 12, 12, 128)	0
Convolution 5	(None, 12, 12, 256)	295168
Convolution 6	(None, 12, 12, 256)	590080
Max Pool 3	(None, 6, 6, 256)	0
Fully Connected Layer	(None, 128)	295040
Output	(None, 16)	2064

The goal of a CNN model is to update the weights in a way that will minimize the loss during the training process. To calculate the loss, we selected the sparse categorical cross-entropy loss function. Before choosing the loss function, we had to analyze the data in our hands. The output classes were mutual, meaning an image could belong solely to one specific class. For this type of data distribution, the cross-entropy loss function works in a swift manner with great precision. If S denotes samples and C denotes classes, then the function describing the sparse categorical cross-entropy loss function would be:

$$-\frac{1}{N}\sum_{s\in S}\sum_{s\in C} 1_{s\in C}\log_p(s \in C) \quad (3)$$

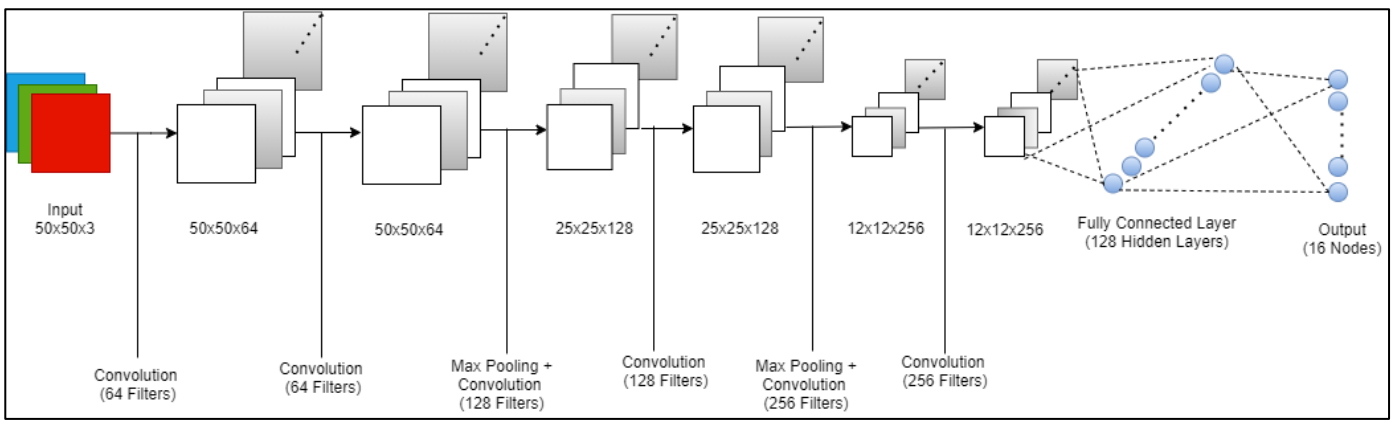


Fig. 4. An illustration of the structure of our proposed CNN architecture

C. *Training the Model*

In the training process, mini-batches of size 32 were used. An initial epoch number of 100 was used to train the model. However, the concept of early stopping was applied, where if the accuracy did not improve after 10 epochs, then the training procedure would have stopped. While training, the three respective datasets were stopped after epoch 31, epoch 29, and epoch 42 respectively. For having better control of the learning rate, we also implemented the ReduceLROnPlateau class [19]. It reduces the learning rate if the validation loss does not improve in a couple of epochs. In this model, the current learning rate decayed after 3 epochs by a factor of 0.5 (the minimum learning rate it could have reached was 0.00001). After the training was completed, both the model and the weights were saved on the disk for further use.

VI. EXPERIMENTAL ANALYSIS

The implemented CNN model was trained on an NVIDIA GeForce GTX 1650 (8GB GDDR6 memory) with system RAM of 8GB. It implemented the Keras API on top of Tensorflow (CUDA toolkit 11.0.221, cuDNN v7.6.5, and Python 3.8).

The continuous update of the training accuracy and the validation accuracy of these datasets are illustrated in fig. 5. Summary of the model's performance based on the accuracy of the three datasets are detailed in table 2. Analyzing the table, we can conclude that we achieved an average accuracy of 99.19%. Fig. 6 illustrates the progression of loss decreasing during the training and validation processes with respect to epochs.

Further, we also analyzed the performance of the model based on F1-accuracy. The three respective F1 accuracies of the three datasets are 0.99, 0.99, 1.00 and the average F1 accuracy we achieved was 0.99. A comparison between the proposed model and the related works is depicted in table 3.

TABLE II. SUMMARY OF EACH LAYER OF THE PROPOSED MODEL

Dataset	Training Accuracy	Validation Accuracy	Test Accuracy
Dataset I	99.41%	99.11%	98.71%
Dataset II	99.35%	99.11%	99.01%
Dataset III	99.83%	99.93%	99.85%

TABLE III. COMPARING THE MODEL PERFORMANCE WITH OTHERS

Related Work	Related Work Accuracy
Ayeb et al. [5]	83.68%
Nguyen et al. [6]	92.30%
Nazemi et al. [7]	90%
Dong et al. [8]	92.42%
Proposed CNN Model	**99.19% (average)**

Comparing the accuracy and the F1-accuracy with the recent works on handwritten math symbols classification, we can conclude that our work has surpassed them.

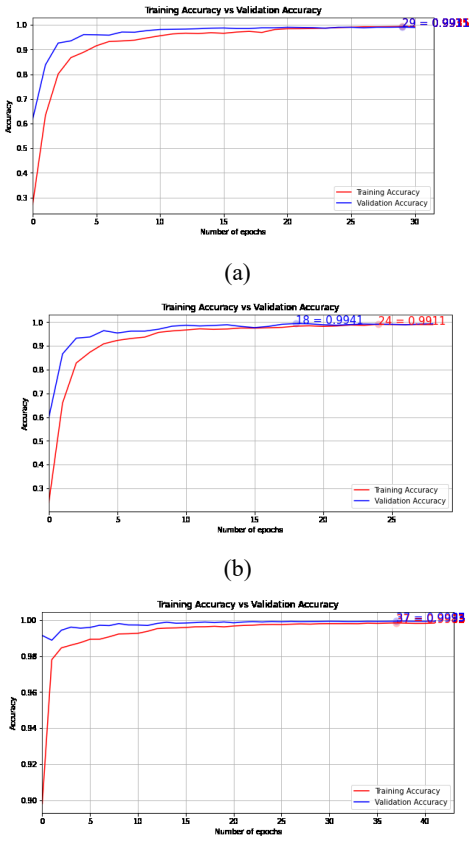
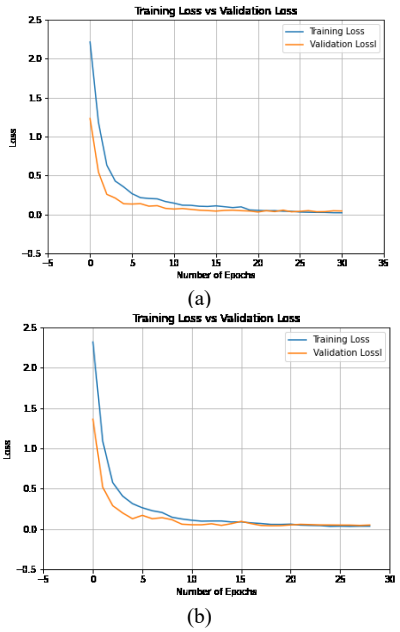
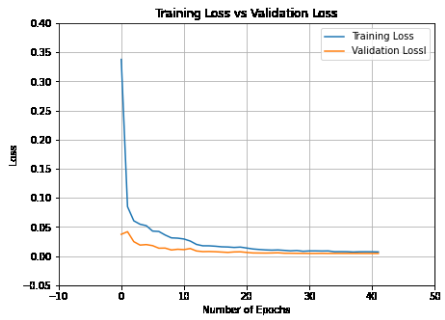


Fig. 5. Progress of training accuracy and validation accuracy of a) Dataset I b) Dataset II c) Dataset III

VII. DISCUSSION & LIMITATIONS

As per the analysis section, we can conclude that our model has performed way better than some recent works in this research field. There lies a couple of factors working behind the model's success. The first factor would be implementing data augmentation techniques to the existing data, making the model more robust and recognizing character shapes of different norms than usual.





(c)

Fig. 6. Training loss vs. validation loss of a) Dataset I b) Dataset II c) Dataset III

Also using a couple of optimization techniques such as Xavier Initialization and Adam optimization while compiling, optimized the technique of learning the weights. Finally, this particular CNN architecture that we gathered after much tweaking with various layers, filter sizes, activation functions, etc. was, in our belief the core reason behind such remarkable results.

There are some limitations in the work that we must point out. Although we worked with three different datasets to evaluate our model, the better way would have been to combine all three datasets into one big dataset to train and later test on some new data. Furthermore, this model is prepared to recognize only isolated images, and if an image contains anything other than the character itself it will face trouble in recognizing accurately. Finally, the last drawback to this model would be that it can only recognize a handful of 16 class labels of math symbols which is pretty small compared to the numerous math symbols out there.

VIII. Conclusion

To gain a clearer grasp of the proposed model's performance on classifying handwritten math symbols, we worked with three different datasets. In this way, diversity was ensured, and the model was made to be more robust in classifying. Although the model performed exceedingly well, we worked with about 16 classes, which is a small subsection of the total number of symbols out there. In our future work, we have set two objectives to accomplish. Firstly, we would like to test our model on all 82 classes of the CHROME dataset. This would ensure the recognition of virtually a complete set of all mathematical symbols. Secondly, after the completion of the first objective, we are interested in eventually moving on to classifying whole mathematical formulas instead of isolated characters and creating an app which in our humble opinion would be a great asset to the users.

References

[1] R. H. Anderson, "Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics," in *Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium*, 1967, pp. 436–459. doi: 10.1145/2402536.2402585.

[2] A. D. Le, B. Indurkhya, and M. Nakagawa, "Pattern generation strategies for improving recognition of handwritten mathematical expressions," *Pattern Recognition Letters*, vol. 128, pp. 255–262, 2019.

[3] B. Zhu and M. Nakagawa, "Online handwritten Chinese/Japanese character recognition," *Advance in Character Recognition, InTech*, pp. 51–68, 2012.

[4] M. N. Hasan, R. I. Sultan and M. Kasedullah, "An Automated System for Recognizing Isolated Handwritten Bangla Characters using Deep Convolutional Neural Network," *2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 2021, pp. 13-18, doi: 10.1109/ISCAIE51753.2021.9431799.

[5] K. K. Ayeb, Y. Meguebli, and A. K. Echi, "Deep Learning Architecture for Off-Line Recognition of Handwritten Math Symbols," *Pattern Recognition and Artificial Intelligence*, vol. 1322, p. 200, 2021.

[6] C. T. Nguyen, T.-N. Truong, H. Q. Ung, and M. Nakagawa, "Online Handwritten Mathematical Symbol Segmentation and Recognition with Bidirectional Context," in *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR),* 2020, pp. 355–360.

[7] A. Nazemi, N. Tavakolian, D. Fitzpatrick, C. Y. Suen, and others, "Offline handwritten mathematical symbol recognition utilising deep learning," *arXiv preprint arXiv:1910.07395*, 2019.

[8] L. Dong and H. Liu, "Recognition of offline handwritten mathematical symbols using convolutional neural networks," in *International Conference on Image and Graphics*, 2017, pp. 149–161.

[9] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," *Synthesis Lectures on Computer Vision*, vol. 8, no. 1, pp. 1–207, 2018.

[10] C. Zhao, (2020, May). Handwritten math symbol and digit dataset, Version 1. Retrieved April 23, 2021 from https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset.

[11] X. Nano, (2017, January). Handwritten math symbol and digit dataset, Version 2. Retrieved April 23, 2021 from https://www.kaggle.com/xainano/handwrittenmathsymbols.

[12] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain, "ICFHR2016 CROHME: Competition on recognition of online handwritten mathematical expressions," in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016, pp. 607–612.

[13] L. Eren, T. Ince, and S. Kiranyaz, "A generic intelligent bearing fault diagnosis system using compact adaptive 1D CNN classifier," *Journal of Signal Processing Systems*, vol. 91, no. 2, pp. 179–189, 2019.

[14] D.-X. Xue, R. Zhang, H. Feng, and Y.-L. Wang, "CNN-SVM for microvascular morphological type recognition with data augmentation," *Journal of medical and biological engineering*, vol. 36, no. 6, pp. 755–764, 2016.

[15] Y. LeCun, K. Kavukcuoglu and C. Farabet, "Convolutional networks and applications in vision," *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 253-256, doi: 10.1109/ISCAS.2010.5537907.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] S. Reddi, M. Zaheer, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," *Proceeding of 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018.