# Classification of Mathematic and Scientific Symbols

**Alec Situ** aasitu@gmail.com
University Hill Secondary School

**Editor:** Alec Situ

## Abstract

Mathematical Expressions Recognition (MER) has become increasingly popular in the modern world and developing better models to classify such symbols can be very useful in the world. Ever since CROHME competition had launched, the topic has spread throughout the machine learning world. There are many models for MER, but they aren't accurate and efficient enough or don't perform well in large datasets. Although novel approaches are not attempted, this paper attempts to enhance current models and also incorporate some scientific symbols for classification. The models are trained and validated on the HASYv2 dataset and have better results than most other models using similar classifying techniques. The images were preprocessed with noise reduction techniques and color binarization. The models use deeply connected convolutional neural networks, which includes symbol segmentation and symbol recognition. However, the models weren't particularly complex because short and efficient models are highly favorable. Ultimately, the accuracies were 79.16% and 80.46% for the two models and the time per epoch was 190s and 350s, respectively.

**Keywords:** Handwritten recognition, Symbol Recognition, Image Recognition, Performance Evaluation, Error Correction, Mathematical Expression Recognition, Convolution Neural Networks, Neural Networks, Symbol Segmentation

## 1. Introduction

Understanding handwritten works is indispensable in modern day society as we write everyday. However, in large batches of writing, human reading becomes dull and inefficient. Therefore, it becomes important to use computers to read mass writings. In particular, mathematical and scientific symbols are of the most unexplored areas of computer vision compared to word texts, as they are complex. As such, it is difficult to create good models to accurately classify the symbols [3]. By using computers, marking widespread math/science tests, computerizing notes, scanning work, etc, becomes significantly easier. Today, computers are becoming increasingly common in recognizing and scanning texts. However, current models don't work well enough to be able to consistently scan large amounts of math symbols in efficient manners.

Over the past decade or so, Mathematical Expressions Recognition has become increasingly popular, especially since the competition CROHME was launched. There were many attempts to classify different symbols. However, many of which were self-created datasets with few symbols and few images [5] (this one has roughly 10000 images, and not a lot of symbols). As a result, these types of datasets were relatively easily to get 95%+ accuracy with these datasets. The primary problem with these datasets is the lack of variety and

complexity of the symbols. So, they aren't practical. Other attempts at classifying complex datasets, however, led to low accuracies [6, 7]. Therefore, this paper will attempt to apply machine learning algorithms to complex dataset without sacrificing high accuracies.

## 2. Method

### 2.1 Dataset

The dataset was the HASYv2 dataset from kaggle.com, which is derived from the HWRT dataset [4]. It includes 168,233 images and each image contains one symbol. There are a total of 369 distinct symbol classes in the dataset. Each symbol class contains one symbol from a pool of English alphabets, Greek alphabet, mathematical symbols, and other scientific symbols. Each image has a size of 32 pixels by 32 pixels. As such, this dataset allows for sufficient analysis of a wide range of common symbols, which contributes to the ultimate goal: to create a model applicable to real world scenarios.

Once the dataset was loaded, preprocessing was performed. The dataset included a csv file for the path of the image, symbol class, and LaTeX symbol for each image in Figure 1.

| | path | symbol_id | latex |
|---|---|---|---|
| 0 | hasy-data/v2-00000.png | 31 | A |
| 1 | hasy-data/v2-00001.png | 31 | A |
| 2 | hasy-data/v2-00002.png | 31 | A |
| 3 | hasy-data/v2-00003.png | 31 | A |
| 4 | hasy-data/v2-00004.png | 31 | A |
| ... | ... | ... | ... |
| 168228 | hasy-data/v2-168228.png | 1400 | \guillemotleft |
| 168229 | hasy-data/v2-168229.png | 1400 | \guillemotleft |
| 168230 | hasy-data/v2-168230.png | 1400 | \guillemotleft |
| 168231 | hasy-data/v2-168231.png | 1400 | \guillemotleft |
| 168232 | hasy-data/v2-168232.png | 1400 | \guillemotleft |

Figure 1: CSV file for the dataset

Firstly, the images were transformed into Numpy arrays. Noise reduction techniques were employed to remove or correct some of the faulty images/data. This includes the fixation of images from 3D Numpy arrays to 2D Numpy arrays. Then, the images were converted into simple black and white pictures (binarization) for its better contrast and high computing efficiency. In the case of Numpy arrays, the colors above a certain threshold will be black and below that threshold will be white. This method is particularly useful as the images in the dataset are relatively simple with clear background as the thresholding method could reduce the quality of images in more complicated cases.

### 2.2 Models

The preprocessed images were split so that 80% is used for training the model and the rest is used for testing. This allows for the model to test in separate, random data like in the real world. All of the models tested were Convolution Neural Network (CNN) models, as analyzing previous models shown that CNN gave the best results for deep learning [1, 2]. In particular, the models are drawn from the TensorFlow Keras package, using four of the layers from the package: Conv2D, MaxPool2D, Flatten, and Dense. The number of Conv2D and MaxPool2d layers varied with different models, but the number of Conv2D always equals the number of MaxPool2D. For most models in general, including the model I chose, there was 1 Flatten layer. Finally, I decided to use only 1 Dense layer.

However, for the Dense layer, the activation function was Softmax. For each of the Conv2D layers, the padding was always "same" for the Conv2D layers. The loss was calculated through categorical cross entropy and the optimizer was Adam, a stochastic gradient descent method. Finally, the models were trained on 7 epochs, as any more didn't increase the accuracy. Occasionally, if there is overfitting, the training would end early by employing the EarlyStop function from Tensorflow Keras.

The first of the models tested was focused mainly on the efficiency and the simplicity of the model. There were only 2 Conv2D and 2 MaxPool2D layers, and the layers weren't particularly complex. For this model, the sigmoid function was used as the activation function:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

The sigmoid function.

The architecture of the first model is illustrated in table 1:

Table 1: Model 1 and its construction. Includes layer type, output shape, and parameters of each layer.

| **Layer Type** | **Output Shape** | **Param** | |
|---|---|---|---|
| Conv2D | (None, 32, 32, 32) | 64 | |
| MaxPool2D | (None, 16, 16, 32) | 0 | |
| Conv2D | (None, 16, 16, 32) | 9248 | |
| MaxPool2D | (None, 8, 5, 32) | 0 | Accuracy: 79.16% |
| Conv2D | (None, 8, 5, 64) | 18496 | |
| MaxPool2D | (None, 4, 2, 64) | 0 | |
| Flatten | (None, 512) | 0 | |
| Dense | (None, 369) | 189297 | |

Another model used was a more complex one, which had a total of 4 Conv2D and 4 MaxPool2D layers. This time, the activation function was ReLU, but as said previously, the activation functions don't make too much of a difference.

$$R(x) = \max(0, x) \tag{2}$$

The ReLU function.

The architecture of this model is in table 2:

Table 2: Model 2 and its construction. Includes layer type, output shape, and parameters of each layer.

| Layer Type | Output Shape | Param | |
|---|---|---|---|
| Conv2D | (None, 32, 32, 32) | 320 | |
| MaxPool2D | (None, 16, 16, 32) | 0 | |
| Conv2D | (None, 16, 16, 64) | 18496 | |
| MaxPool2D | (None, 8, 8, 64) | 0 | |
| Conv2D | (None, 8, 8, 64) | 36928 | Accuracy: 80.46% |
| MaxPool2D | (None, 4, 4, 64) | 0 | |
| Conv2D | (None, 4, 4, 128) | 73856 | |
| MaxPool2D | (None, 2, 2, 128) | 0 | |
| Flatten | (None, 512) | 0 | |
| Dense | (None, 369) | 189297 | |

Obviously, the models were measured by accuracy, but time was also considered.

## 3. Result

The models performed as follows on the testing data:

Table 3: Results for the two models.

| | Training Accuracy | Validation Accuracy | Time |
|---|---|---|---|
| **Model 1** | 82.12% | 79.16% | 190s |
| **Model 2** | 84.37% | 80.46% | 350s |

Although there are often one or two more Dense layers in other models, my testing showed that including the extra Dense layers led to significantly worse accuracies. This is mainly due to the existence of 369 categories, making it difficult to add more Dense layers. Moreover, many different activation functions were tested, but the results remained similar between different activation functions. Although the 2 proposed models have different activation functions, the change is unnoticeable. From Table 3, also note that the training accuracy was higher than the validation accuracy in both models. This means that although there is a little bit overfitting, it wouldn't cause too many problems.

## 4. Discussion

Considering there were 369 different categories with over 160,000 images, the results were good. Moreover, the low image resolution (Figure 2) conjunction with the fact that many symbols were similar (Figure 3) made the classification even harder. Although there was a difference in the accuracies between, it was only 1.3%. Despite being 1.3% less accurate, model 1 took almost half the time of model 2, which is very significantly because one of the primary focuses of this paper is to discover more efficient models. Compared to previous studies [3], model C improved the accuracy by 4%. While it doesn't sound like a lot, 4% of 160,000 images is 6400 images. In the case of real exams, this would be a huge increase in correctness.

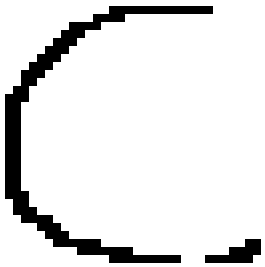

Figure 2: An image of the letter C from the dataset

Figure 3: Two sample images of the letter E and $\sum$

While the results are promising and better than before, it is far from useful. 80%, despite being a good accuracy for this type of test, is not nearly accurate enough for the real world.

There are also many limitations of this study which was not addressed. For instance, because not all of the dataset (160,000 images) were checked, there might have been errors in some of the images, such as a symbol on a wrinkled paper. Although some error checking was performed, it is not enough to guarantee the dataset is flawless. Therefore, in future works, eliminating all these errors would be ideal. Furthermore, testing different architecture types for the models would be useful because the two models presented in the paper are very similar in construction. Finally, as the CSV file already includes the LaTeX for each symbol/image (Figure 1), it would be useful to translate the results into LaTeX as it becomes more readable.

## References

[1] F.C.F. Marques et al. "Recognition of Simple Handwritten Polynomials Using Segmentation with Fractional Calculus and Convolutional Neural Networks". In: (2019), pp. 245–250. DOI: 10.1109/BRACIS.2019.00051.

[2] Irwansyah Ramadhan, Bedy Purnama, and Said Al Faraby. "Convolutional neural networks applied to handwritten mathematical symbols classification". In: *2016 4th International Conference on Information and Communication Technology (ICoICT)*. 2016, pp. 1–4. DOI: 10.1109/ICoICT.2016.7571941.

[3] Kukreja Sakshi and Ahuja. "Recognition And Classification Of Mathematical Expressions Using Machine Learning And Deep Learning Methods". In: *ICRITO* 9 (2021).

[4] Martin Thoma. *HASYV2*. URL: https://www.kaggle.com/guru001/hasyv2. (accessed: 12.07.2021).

[5] Zi-long WAN et al. "Recognition of Printed Mathematical Formula Symbols Based on Convolutional Neural Network". In: *DEStech Transactions on Computer Science and Engineering* ica (2019).

[6] Hongyu Wang and Guangcun Shan. "Recognizing handwritten mathematical expressions as LaTex sequences using a multiscale robust neural network". In: *arXiv preprint arXiv:2003.00817* (2020).

[7] Jianshu Zhang, Jun Du, and Lirong Dai. "Track, attend, and parse (tap): An end-to-end framework for online handwritten mathematical expression recognition". In: *IEEE Transactions on Multimedia* 21.1 (2018), pp. 221–233.