# Handwritten Mathematical Formula Detection using Deep Learning

**Niklas Schweiger ✉ and Jannik Obenhoff ✉**

Department of Electrical and Computer Engineering, Technical University of Munich, Arcisstr. 21, 80333 Munich, Germany

✉ niklas.schweiger@tum.de
✉ jannik.obenhoff@tum.de

March 7, 2023

**Abstract** — In this analysis project, a data set [1] containing numbers, mathematical symbols and letters has been processed in order to train a deep neural network (DNN) that serves as a mathematical formula classifier model. The final model is applied to real handwritten formulas over a plotly dashboard to demonstrate the model performance in the real-world use case. In addition, there will be an outlook on future improvements and application scenarios.

## 1 Introduction

Handwritten formula detection is a challenging problem in the field of pattern recognition and image processing. Despite the recent advancements in machine learning algorithms, different styles, sizes and character shapes make it difficult to find a perfect detection model. In this paper, we present a novel approach to handwritten formula detection using convolutional neural networks (CNNs), image processing, and feature extraction to achieve great results in real world-scenarios.

## 2 Data

This section describes the essential preprocessing that has had to be done to the raw data set [1] before handing it over to the Deep Neural Network, as well as the scanning process that was made to the real-world data using various methods like adaptive thresholding.

### 2.1 Training Data

#### 2.1.1 Understanding and Cleaning the Data

The training data set contains over 370.000 image samples, all in the 45x45 jpg file format. The first step was to reduce the data to the necessary characters needed for the implemented functionalities. Therefore, characters like the sum formula and more have been removed, limiting the data set to 300.000 image samples.

#### 2.1.2 Preprocessing the Data

In order to get the best results possible we figured out that the optimal image size would be 28x28 pixels. After scaling down the samples and first tests, the results were not as desired. The next preprocessing step was to widen the line thickness of each character. After some back and forth the optimal line thickness was 6 pixels for the 45x45 jpg samples and 4 pixels for the down sampled 28x28 images.

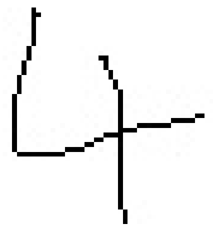
**Figure 1** Raw Data

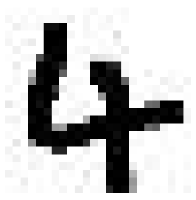
**Figure 2** Processed Data

### 2.2 Real Data

Just as with the training data there has had to be done preprocessing before handing over the real world data to the Machine Learning Model.

#### 2.2.1 Adaptive Thresholding

To eliminate distracting background noise an adaptive threshold is applied to the image. The first steps are two convolutions with a gaussian kernel with side length 7 (Figure 4). After the convolution the image array gets substituted by 50-minimum value of the original image. The last step is to threshold the original image with the obtained threshold image. The result is an image only containing black and white pixels as seen in Figure 5.

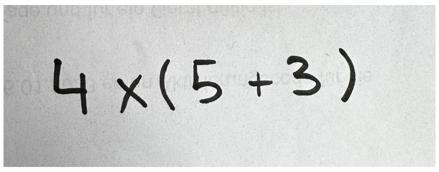
**Figure 3** Pre-Thresholding

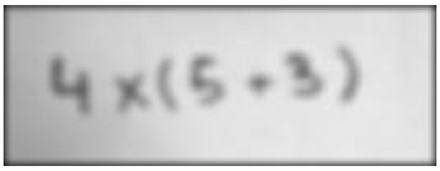
**Figure 4** After-Convolution

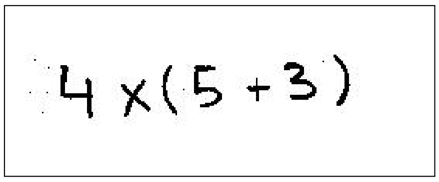
**Figure 5** After-Thresholding

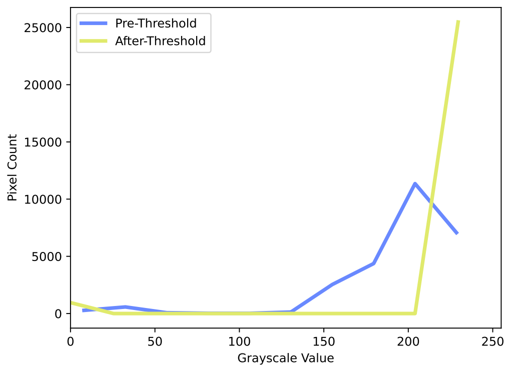

**Figure 6** Pixel Distribution Histogram

#### 2.2.2 Scanning and Sampling

The next step is to scan the image to obtain all individual characters and feed it to the model. The image array, only containing black and white values, gets scanned by columns to find the starting and ending column of a character. To eliminate possible noise, charters only containing one or two columns are being deleted. After column search, height search is applied to find the character baseline and height. Every character column gets reduced in size to the maximal height and minimal baseline found in height search. The second last step is again to eliminate noise, characters with a threshold dimension shape of min(shape)/max(shape) < 0.1 will get deleted. The final step is to add an artificial border and to resize every character to a 28x28 image to perfectly fit the training data see Figure 7.

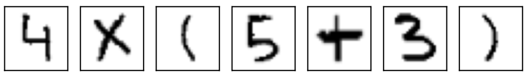
**Figure 7** Individual Characters after Scanning

**Figure 8** Architecture of the CNN

## 3 Model

### 3.1 Theoretical background

Based on the (general and language-agnostic) best practices introduced in the section above, we implement measures for a C++ software library with bindings for the Python language in this section.

### 3.2 Architecture

The model that has been used for this project is a convolutional deep neural network with a total of six hidden layers. As only black and white images are passed to the network, the input size is one which connects to ten neurons as illustrated in figure 8. In the first layer each neuron performs a convolution of the input tensor (28 x 28 x 1) with a kernel size of three and stride and padding of one. After the convolution the ReLU function is used as activation. In the second layer again a convolution and ReLU activation is performed and after that a MaxPool layer with stride and padding of two which results in a (14 x 14 x 1) tensor at each neuron. The same two layers are then again used for layers three and four which results in a (7 x 7 x 10) tensor for the whole layer five. The three-dimensional tensors are then flatted to an (490) array which is necessary because the following SoftMax function requires a one-dimensional input. Finally, the SoftMax function evaluates a probability for each output class.

### 3.3 Training

In order to train the network Cross Entropy Loss was used as the loss function as it works very well with probabilistic methods in the neural network such as the sigmoid function and the SoftMax and for classification problems in general. For the optimization function Adam with a learning rate of 0.001 is used as it generally works well on CNNs because of its low computational cost and fast convergence. Even though the Network is rather small with only a few layers and no data augmentation, it performs well on the given dataset and reaches a Training accuracy of 0.86 and Test accuracy of 0.92 as shown in figure.after training only on a thousand images. Data augmentation often has the ability to significantly improve the performance of a network but in this case, it doesn't help very much, because for one our dataset is already very big with over 300,000 images and the detected images have a very specific format and coloring. For example, a classic Data Augmentation like rotating the images or increasing and decreasing their brightness wouldn't have much use because the symbols will usually always be in the correct rotation and are preprocessed in a way that they have only pixels with two grayscale values for black and white.

## 4 Performance and Results

### 4.1 Dashboard

To make easy handling possible a Plotly Dashboard [2] is being used. The first step for a user is to select one out of three functionalities, calculation, plotting, equation. After that it is possible to upload a photo of the handwritten formula or even take a photo with the webcam. After the threshold and scanning processReal Data the individual characters are displayed and the user can make a reselection if noise or other undesired parts have been detected. The final selection gets hand over to the model to classify the characters. Depending on the selected functionality the result is being displayed.

## 5 Outlook and Improvements

More training data. Improve model. Another possible improvement could be to further develop the threshold and scanning process, to reduce the amount of misidentified characters. In future the entered

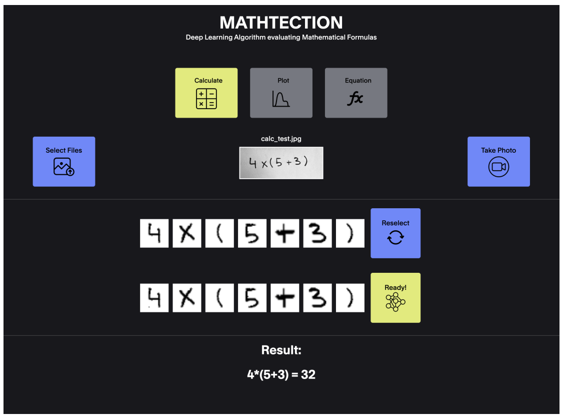

**Figure 9** Plotly Dash

real data could be collected in order to extend the training data set for the Neural Network.

## References

[1] Xai Nano, *Kaggle Data set: Handwritten math symbols dataset* https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols (last accessed: 20.02.2023).

[2] Plotly https://plotly.com/dash/ (last accessed: 07.03.2023).

[3] A. Hunt and H. Thomas, *The Pragmatic Programmer: From Journeyman to Master* (Addison-Wesley, Boston, 1999), 1st ed.

[4] A. Prlić and J. B. Procter, "Ten simple rules for the open development of scientific software," PLoS Comput. Biol. **8**, 1–3 (2012).

[5] L.Alzubaidi, J.Zhang, A.Humaidi, A. Al-Dajuaili, Y. Duan, O. Al-Shamma, J. Santamaria, M.Fadhel, M. Al-Amidie, L. Farhan, *]Review of deep learning: concepts, CNN architectures, challenges, applications, future directions* https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8 (last accessed)