# Handwritten Mathematical Formula Detection using Convolutional Neural Networks

**Niklas Schweiger, 03739305** ✉ **and Jannik Obenhoff, 03742939** ✉

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

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

March 16, 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.

## 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 an approach to handwritten formula detection using convolutional neural networks (CNNs), image processing, and feature extraction to achieve the best possible 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 CNN, as well as the image processing methods for the real world data.

### 2.1 Training Data

#### 2.1.1 Understanding and Cleaning the Data

The initial training dataset contains over 370,000 computer generated image samples, each in the 45x45 jpeg file format. The first step was to reduce the dataset to the characters essential for the implemented mathematical functionalities (see 4.1), thus yielding a data set of 300,000 image samples.

#### 2.1.2 Preprocessing the Data

The optimal image size to train the CNN would be 28x28 pixels, a good compromise between memory complexity and detail loss. In this case a higher resolution is not needed, as the subject of the images are rather simple. The next preprocessing step involved increasing the line thickness of each character to 6 pixels for the 45x45 jpg images and 4 pixels for the down sampled 28x28 images, as well as incorporating a white border that spans 20% on each side. Compare Figure 1 and Figure 2.

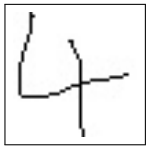

**Figure 1** Raw Data

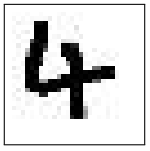

**Figure 2** Processed Data

### 2.2 Real Data

Before handing the real-world data to the CNN a submitted image needs to be transformed in order to achieve the best possible results. The raw input data is a photo of a handwritten mathematical formula, preferable on white paper and written in dark color.

#### 2.2.1 Adaptive Thresholding

To eliminate distracting background noise and to separate the desired foreground formula from the background based on the difference in pixel intensities of each region, an adaptive threshold is applied to the image [2]. For each pixel in the submitted image, a threshold has to be calculated with the following operations. The first steps are two convolutions with a gaussian kernel with side length of seven (Figure 4). Following the convolution, each

value of the image array gets subtracted by 50 minus the minimum pixel value (Min) of the original image. The last step is to threshold the original image with the obtained threshold image. The outcome of this process is an image only containing black and white pixels, as illustrated in Figure 5.

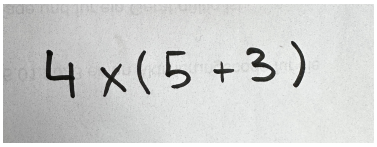


**Figure 3** Input Image

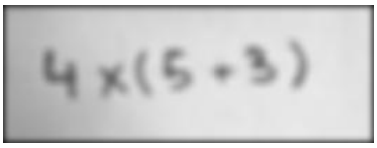


**Figure 4** After Convolution

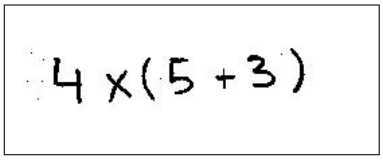


**Figure 5** After Thresholding

#### 2.2.2 Scanning and Sampling

The next step is to scan the image obtaining all individual characters. The image array, only containing black and white values, gets scanned by columns first to find the starting and ending column of each character. To eliminate possible noise, characters only containing one or two columns are being deleted. Every character column gets reduced in size to the maximal height and minimal baseline found in height search. The second last step involves the elimination of possible characters, whereby those with a threshold dimension shape of min(shape)/max(shape) < 0.1 are excluded. The final step includes appending an artificial border to each character and resizing every them to a 28x28 image to perfectly fit the dimensions of the training dataset. The final result can be seen in Figure 6.


**Figure 6** Individual Characters after Scanning

## 3 Model

### 3.1 Theoretical Background

For this classification problem a Convolutional Neural Network is being employed. As a specific type of Deep Neural Network, a CNN fits under the definition of a supervised learning model as the correct labels are always available for the Network [3, p. 9]. The main advantages of CNNs are their use of local connectivity and shared weights. CNNs use convolutional layers that are specifically designed to take advantage of the spatial relationship between adjacent pixels in an image. By using local connectivity, CNNs are able to learn features that are translation invariant, meaning they can recognize the same feature in different parts of an image. The shared weights approach reduces the number of parameters in the model by using the same weights for each part of the image, which makes the training process faster and more efficient [3, p. 13-14]. Also, CNNs often use pooling layers like Max or Average Pooling in order to reduce the dimensionality of the feature maps, which further reduces the computational cost and helps to avoid over fitting. For these reasons CNNs are very suitable for image recognition but also find applications in speech processing. In fact, well-trained CNNs have been shown to outperform humans on certain image classification tasks.

### 3.2 Architecture

The model that has been used for this project is a CNN with a total of six layers, which structure is depicted in figure 7. Given that only monochromatic images are fed into the network, the input size, specifically the number of channels, was specified as one, as opposed to the three channels that would typically be utilized for RGB images. Each of the following hidden layers consists of thirty-two neurons. In the first layer each neuron convolves the input tensor (28x28x1) with a kernel of size three, a stride of one, and a padding of one. The Rectified Linear Unit (ReLU) function is applied as activation after the convolution. The second layer follows a

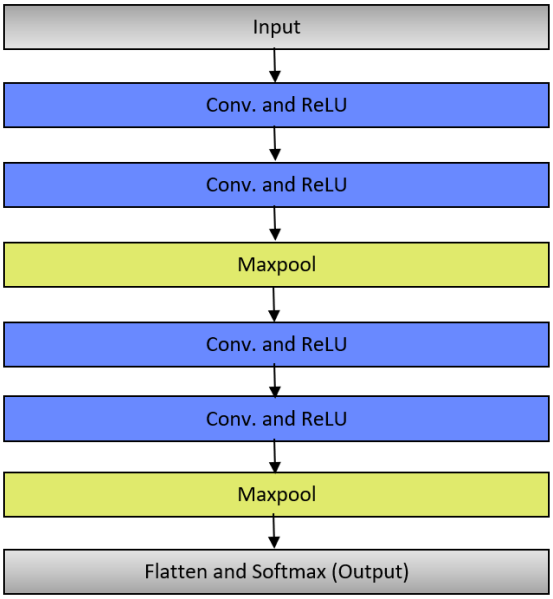


**Figure 7** Architecture of the CNN

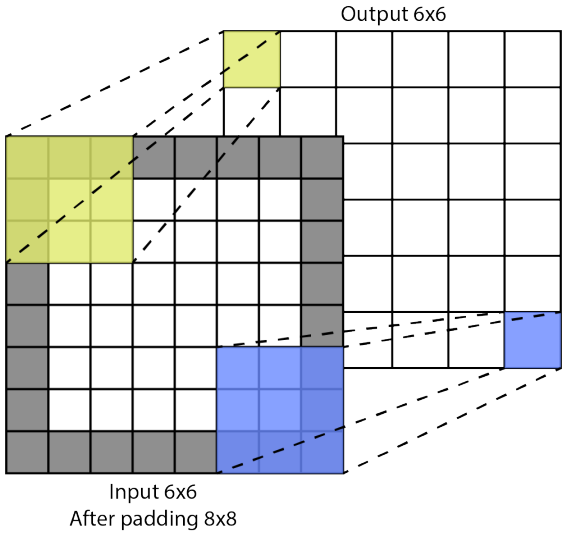


**Figure 8** Exemplary convolution with an 6x6 input, kernel size three, stride one, and padding one.

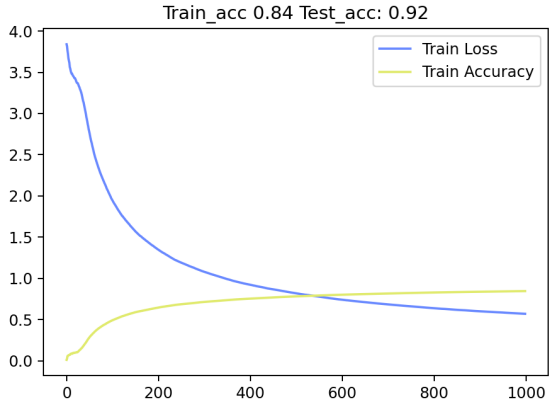


**Figure 9** Train Loss and Train Accuracy over a thousand images

similar structure, where convolution and ReLU activation are performed, followed by a MaxPool layer with stride and padding of two, resulting in a (14x14x1) tensor at each neuron. The same three-layer structure is then applied for layers four, five and six which results in thirty-two (7x7x1) tensors for layer six. As the Softmax function requires one-dimensional input, the three-dimensional tensors are flattened into an array of size (1568). Finally, the SoftMax function evaluates a probability for each output class.

## 3.3 Training

In order to train the network Cross Entropy Loss was utilized as it demonstrates exceptional efficiency with probabilistic methods such as the SoftMax function as well as in classification problems overall. Due to its low computational cost and fast convergence the adaptive moment estimation Adam [4] was used as the optimization function with a learning rate of 0.001.

Despite its small size with only a few layers and no data augmentation, the Network performs well on the given dataset, achieving a Training accuracy of 0.84 and Test accuracy of 0.92 after training only on a thousand images, as shown in Figure 9.

Upon completion of three training epochs utilizing a comprehensive dataset consisting of over 200,000 images, in addition to a test dataset consisting of over 100,000 images, the network achieved a remarkable test accuracy of 96 percent. This outcome is deemed highly satisfactory for the given project, as even for the authors, the identification of certain images presented a challenge.

### 3.3.1 Data Augmentation

Data augmentation often has the ability to significantly improve the performance of a network but in this case, it was not useful, as the dataset is already very large with over 300,000 images with a specific format and coloring for the detected images. Conventional data augmentation techniques, such as image rotation or brightness modification [5], were deemed unbeneficial, given the inherent characteristics of the symbols, which are already correctly oriented and preprocessed to contain only two grayscale values of black and white.

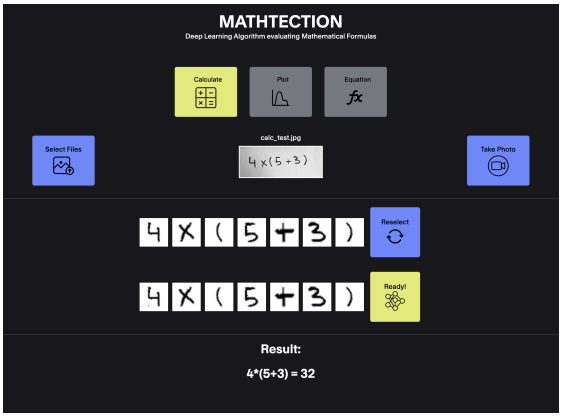


**Figure 10** Plotly Dash

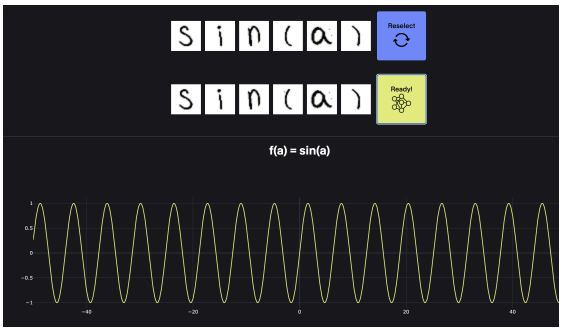


**Figure 11** Plot Functionality

# 4 Performance and Results

## 4.1 Dashboard

The final model is applied to real handwritten formulas over a Plotly Dashboard [6], with a minimalist design approach employed to optimize the user experience (Figure 10). The initial step for the user is to select a functionality:

- Basic calculations using numbers and simple mathematical operations.
- Plotting a function containing one variable (Figure 11).
- Solving mathematical equations containing variables by accessing the Wolfram Alpha Api [7].

The individual identified characters get displayed and the user can make a reselection in the event that noise or other undesired elements were detected. The final selection is handed over to the model for character classification, with the results displayed based on the selected functionality.

## 4.2 Application Performance

The evaluation was conducted under certain constraints, including limited availability of real-world data and time limitations for testing. The application demonstrated some notable strengths, including impressive computational speed across all stages of the process. However, the successful detection rate fell short of optimal performance (about 60%), indicating room for improvement for the model and the image processing.

# 5 Outlook and Improvements

Further improvements with a more complex Network might increase the performance of the CNN on the test set, but would not significantly change the performance when classifying real handwritten symbols. Instead, training on a real human generated dataset of handwritten characters would be a better option.

One approach to achieve this would be to collect the submitted formula images and utilize them to train the CNN.

In the future, additional features could be added to the application, such as support for more complex mathematical expressions, including integrals and fractions. However, this would require a better image recognition and an improved CNN as a foundation.

# References

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

[2] R. Fisher, S. Perkins, A. Walker, E. Wolfart. *Adaptive Thresholding* https://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm (last accessed: 08.03.2023).

[3] 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 07.03.2023).

[4] D. Kingma, J.Lei Ba, *ADAM: A method for stochastic optimization page 6–7* https://arxiv.org/pdf/1412.6980.pdf (last accessed 07.03.2023).

[5] J. Wang and L. Perez: *The Effectiveness of Data Augmentation in Image Classification using Deep Learning* https://arxiv.org/pdf/1712.04621.pdf (last accessed 07.03.2023)

[6] Plotly, *Dash Enterprise* https://plotly.com/dash/ (last accessed: 14.03.2023).

[7] Wolfram Research Inc., *Wolfram Alpha API* https://products.wolframalpha.com/api (last accessed: 08.03.2023).