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 8, 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 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 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 original 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 resulting in a data set with 300.000 image samples.

2.1.2 Preprocessing the Data

The optimal image size to train the machine learning model 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 as well as adding a border of 20% on each side. Compare Figure 1 and Figure 2.

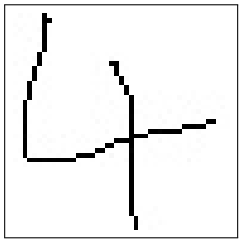

Figure 1 Raw Data

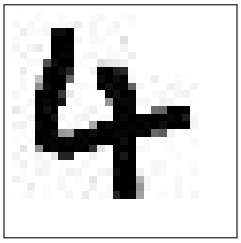

Figure 2 Processed Data

2.2 Real Data

Before handing the real-world data to the trained machine learning model the 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 [3]. 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). After the convolution the image array gets substituted by 50-min, where min is the minimal pixel 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.

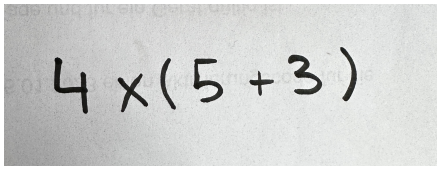

Figure 3 Pre-Thresholding

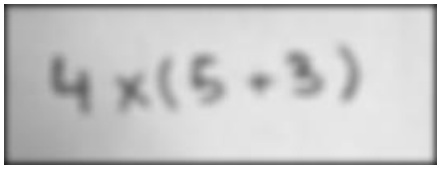

Figure 4 After-Convolution

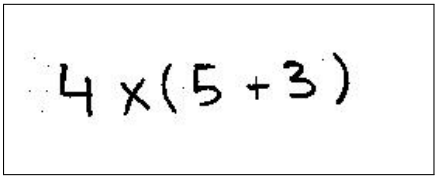

Figure 5 After-Thresholding

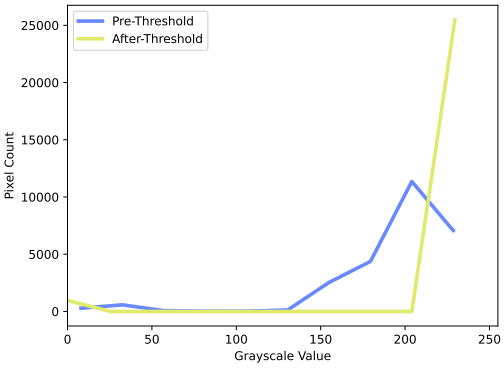

Figure 6 Pixel Distribution Histogram

Figure 7 Individual Characters after Scanning

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 first to find the starting and ending column of each 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 to eliminate potential characters with a threshold dimension shape of min(shape)/max(shape) < 0.1. 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. The final result can be seen in Figure 7.

3 Model

3.1 Theoretical background

For this classification problem a Convolutional Neural Network is being employed. A CNN is a specific type of Deep Neural Network (DNN) that is based on supervised learning as the correct labels of classes are always available for the CNN. 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. 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 [6]. Also, CNNs often use pooling layers like Max or Average Pool-

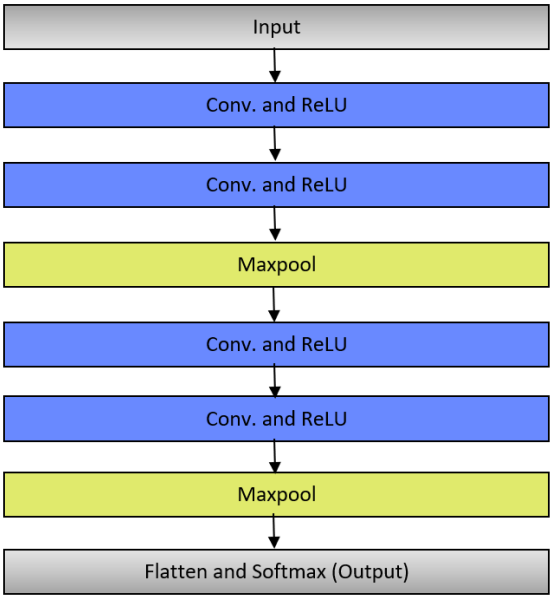

Figure 8 Architecture of the CNN

ing in order to reduce the dimensionality of the feature maps, which further reduces the computational cost and helps to avoid overfitting. 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 8] As only black and white images are passed to the network, the input size (channel)was set to one instead of three which would have been used 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 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 three, four and five which results in thirty-two (7x7x1) tensors for layer five. As the Softmax function requires one-dimensional input, the three-dimensional tensors are flattened into a (1568) array. Finally, the SoftMax function evaluates a probability for each output class.

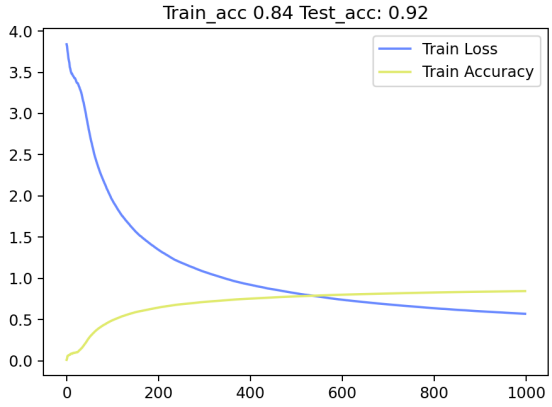

Figure 9 Train Loss and Train Accuracy

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. Due to its low computational cost and fast convergence the adaptive moment estimation Adam [7] 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 0.86 and Test accuracy of 0.92 after training only on a thousand images, as shown in figure **??**.
After three epochs of the whole training set that contains over 200,000 images and a test set of over 100,000 images the Network reaches a test accuracy of 96 percent, which is already quite sufficient for the project as even for the authors some images are hard to identify.
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. For example, a classic Data Augmentation like rotating the images or modifying their brightness [8] 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. Standard Data Augmentation techniques were tried in the project but were not able to increase the performance of the Network but in fact only decreased it.

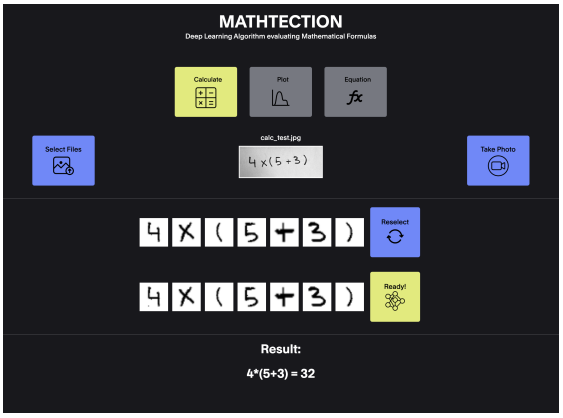

Figure 10 Plotly Dash

4 Performance and Results

4.1 Dashboard

To make easy handling possible a Plotly [2] Dashboard is being used (Figure 10). The dashboard design is kept minimal to maximize the user experience. The first step for a user is to select a functionality, calculation, plotting, equation. Afterwards you can upload a photo of the handwritten formula or even take a photo with the webcam. Then the submitted image gets processed accordingly to the above sectionsReal Data, the individual detected characters get displayed, and the user can make a reselection in the case that 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 Performance and results

Can be better

6 Outlook and Improvements

As the Network already has a good test accuracy on the given test set further improvements with a more complex Network or by increasing the size of the hidden layers might increase the performance of the Network on the test set but would not significantly change the performance of the Network when classifying real handwritten symbols. Instead, training on a real human generated dataset of handwritten symbols would be a better option.

Another possible improvement could be to further develop the threshold and scanning process, to reduce the amount of misidentified characters. In future the entered 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] 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).

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

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

[6] 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).

[7] 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).

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