Convolutional Neural Network for Solving Mathematical Expressions

Kudashev Matvei, Chayakova Ayauly, Quyi Le, Marsaoui Maram, and Obeid Abeer.

Universite Grenoble Alpes

January 30, 2026

ABSTRACT

Aims. Building two neural networks, a Convolutional Neural Network and a Fully Connected Neural Network, from scratch to identify mathematical expressions and equations and solve them
Methods. Several neural networks were developed using c++, following a low-level approach, independent of high-level libraries. Model parameters were then tweaked in order to find the best setting for handwritten digit and mathematical symbol recognition.
Results. The best accuracy reached among the developed models was at 98.6%
Conclusions. The convolutional neural network performed relatively well in digit recognition, with an accuracy rate comparative to state-of-the-art rates, outperforming the fully connected network in all measured metrics, including lower confusion between symbols.

Key words. Convolutional Neural Networks – Handwriting Recognition – Fully Connected Neural Networks – Handwritten Mathematical Expressions

1. Introduction

This project implements a neural network from scratch in C++ to recognize handwritten digits and arithmetic symbols, allowing for the segmentation and solving of whole arithmetic expressions and single-variable polynomials. The low-level implementation serves two purposes: an educational purpose, allowing a better understanding of the inner workings of neural networks, training, and backpropagation; and allowing deeper customization and optimization of the network settings than could be achieved by using built-in libraries.

Offline recognition of handwritten expressions has numerous applications, most notably in educational technology, particularly in improving the accessibility of education for schoolchildren with mobility challenges [18], and in developing learning tools for students. Popular applications of handwritten digit recognition can also be found in fields ranging from archiving (preserving historical handwritten documents) to postal services (sorting letters by handwritten postal codes) [1].

Previous approaches to the task of handwriting recognition used classical machine learning algorithms, which, compared to even rudimentary neural networks (multilayer perceptron), showed lower accuracy[16].

Given that handwritten character recognition is strongly affected by variations in individual writing styles, convolutional neural networks are a natural choice, as they are capable of learning robust nonlinear features directly from raw input data [16]. In addition, such networks, particularly convolutional architectures, have proven effective for recognizing handwritten characters, even in high-dimensional images with minimal preprocessing [11].

In addition to character-level variability, handwritten arithmetic expression recognition presents further challenges related to symbol segmentation and spatial layout, which must be addressed to enable reliable expression interpretation. As a result of this, we also aim to investigate the advantage of convolutional neural networks over fully connected neural networks through the implementation and testing of two models.

This report first reviews relevant literature on handwritten character and expression recognition, highlighting the strengths and limitations of prior approaches. It then describes the architecture of the proposed system, including the segmentation and classification stages, followed by the training methodology. Experimental results are presented, and the performance of the system is evaluated using standard metrics. Finally, limitations are discussed, along with directions for future improvements.

2. Problem Overview and Literature Review

A significant aspect of this work focuses on handwritten digit recognition, a task that has been extensively studied in the literature, ranging from early methods based on hand-crafted features and classical machine learning algorithms to modern approaches leveraging convolutional neural networks and deep learning.

The literature distinguishes between on-line and off-line character recognition. The former deals with the live reception of pen strokes and interactive prediction as text is being written, while the latter focuses on the recognition of pre-written text [15]. The present work falls within the off-line recognition paradigm, operating on fully written mathematical expressions.

Handwritten mathematical expression recognition is commonly decomposed into multiple subproblems. As noted in prior work, "most recognition systems consider mathematical expression recognition as a set of three subproblems: segmentation, recognition, and structural analysis and interpretation" [2]. In this project, the focus is placed on the recognition component, assuming a simplified input structure.

Early approaches to character recognition relied on rule-based systems [14] and classical machine learning algorithms such as Support Vector Machines and K-Nearest Neighbors [8]. The performance of these methods was limited by their reliance on linear decision boundaries. While such models can be extended to capture nonlinear relationships by applying them to a transformed input $\phi(x)$, early approaches to defining this mapping required extensive manual feature engineering. Neural networks provided a more flexible alternative by learning these nonlinear transformations directly from data, significantly reducing the need for hand-crafted features [4].

Empirical studies have consistently demonstrated the effectiveness of neural network-based approaches. [9] reported that convolutional neural networks outperformed all other tested methods for handwritten digit recognition across all evaluated performance metrics. Similarly, in a comparative evaluation of multiple classifiers, [16] found that a multilayer perceptron achieved the highest accuracy among the approaches considered. Advances in computational resources further enabled the practical success of deep neural networks, with [10] demonstrating a dramatic improvement in state-of-the-art performance on large-scale image recognition tasks.

Beyond isolated character classification, neural networks have also been shown to model more complex outputs. Goodfellow demonstrated that neural networks can be trained to output entire sequences of characters transcribed from an image, rather than identifying individual symbols in isolation [5].

Building on these findings, the present work does not aim to propose a novel recognition architecture, but rather to implement and evaluate neural network models from first principles. By developing both a fully connected neural network and a convolutional neural network from scratch, this project enables a direct comparison between architectures that do and do not explicitly exploit spatial structure. This approach provides insight into the practical trade-offs between model complexity, performance, and implementation effort in the context of handwritten mathematical symbol recognition.

Finally, a key distinction between linear models and neural networks lies in the nonconvexity introduced by nonlinear activation functions. As a result, neural networks are typically trained using iterative, gradient-based optimization methods rather than closed-form solutions or convex optimization techniques with global convergence guarantees [4].

3. Methodology

3.1. Datasets

Three datasets were used to train and evaluate the proposed models.

For handwritten digit recognition, the MNIST dataset was employed. MNIST is a widely used benchmark dataset for digit recognition and has been described by Geoffrey Hinton as "the drosophila of machine learning"[1]. The dataset consists of grayscale images of handwritten digits from 0 to 9, each paired with a corresponding label. It contains 60,000 training images and 10,000 test images, evenly distributed across the ten digit classes. Due to its controlled nature and standardized format, MNIST provides a suitable baseline for evaluating recognition performance [4].

To extend the recognition task beyond digits, two additional datasets were used for handwritten mathematical symbol recognition. The first dataset [17] contains labeled images of common arithmetic symbols, while the second dataset [13], a modified version of the CROHME dataset [12] was used to supplement training data for less frequently occurring symbols, specifically the dot and backslash.

Combining these datasets, a final classification set of 18 classes was constructed, consisting of the digits 0–9, parentheses "(" and ")", and the symbols $\{=, *, /, -, +, x\}$. For most classes, 8,000 samples were allocated to the training set and 2,000 samples to the test set; fewer samples were available for the $*$ and / symbols due to dataset limitations. This combined dataset enabled robust training across a diverse set of handwritten symbols.

In addition to the training and test sets, a separate validation dataset was assembled using handwritten symbols collected from individuals not represented in the training data. This validation set was used to assess how the models generalized on unseen data. While these datasets provide substantial variability in handwriting styles, they do not fully capture the diversity of real-world handwritten mathematical expressions.

3.2. Preprocessing

The preprocessing procedure differed for the data used to train the convolutional neural network and the fully connected neural network. The main difference being the final size chosen for the training images. Starting with the preparation for the convolutional neural network, minimal preprocessing was sufficient for the images prior to training due to the controlled nature of the datasets and the inherent robustness of convolutional neural networks to small spatial variations. Convolutional models have been shown

[1] The drosophila is a small fruit fly used extensively in genetic research due to its large chromosomes, numerous variants, and rapid reproduction rate.

to effectively learn hierarchical feature representations directly from raw pixel data, reducing the need for extensive hand-crafted preprocessing steps.

All three datasets underwent the same preprocessing pipeline to ensure consistency. All images were resized to a fixed resolution of 44×44 pixels, while preserving their original aspect ratios to avoid geometric distortion. The images were then converted to grayscale and binarized such that the foreground characters appeared as white pixels on a black background. Finally, pixel values were normalized to the range [0,1] by dividing by 255. This preprocessing strategy was chosen to standardize input dimensions while maintaining the essential structural characteristics of handwritten symbols, enabling the convolutional network to focus on learning discriminative features relevant to character recognition.

As for the fully connected neural network, the same preprocessing steps were followed for the training data (conversion to greyscale, pixel value normalization, and resizing). However, the final size of the images was 28x28 pixels.

3.2.1. Input Assumptions and Simplifications

The system assumes that handwritten expressions are provided in a single horizontal line and that symbols are sufficiently separated to allow reliable segmentation. While minor variations in spacing are tolerated, cases involving severe character overlap, disconnected strokes within a single symbol, or multi-line expressions are not explicitly handled. These assumptions simplify the segmentation and recognition pipeline and are discussed further in the limitations section.

3.3. Model Architecture

3.3.1. Convolutional Neural Network

The main inspiration behind the design choices in the convolutional neural network was the LeNet-5 . We adopt a similar overall structure because, in previous studies [11], it has been shown that this architecture copes well with the task of character recognition., while modifying certain components, most notably the activation functions. The network takes as input grayscale images resized to a resolution of 44×44, with pixel intensities normalized to the range $[0, 1]$ by division by 255.

First convolutional block. The first layer is a convolutional layer with 6 learnable filters of size 5×5, applied with stride $S = 1$ and with zero-padding ($P = 2$). Formally, this layer can be denoted as

$$\text{Conv}(6, 5 \times 5, S = 1, P = 2).$$

The spatial size of the output feature maps is computed as

$$\text{Out} = \frac{N + 2P - K}{S} + 1,$$

where N is the input size, K the kernel size, P the padding, and S the stride. Substituting the values for this layer yields an output of size 44×44, resulting in a tensor of dimension $6 \times 44 \times 44$.

The filter weights are initialized using Kaiming initialization, which is designed to preserve the variance of activations and gradients across layers, thereby improving training stability and convergence when using ReLU-based activation functions as next layer[7]. This layer contains $5 \times 5 \times 6 + 6 = 156$ trainable parameters, including biases. During training, the filters learn to detect low-level visual features such as edges and strokes.

In a convolutional layer with padding, the input is first extended by adding p zero-valued pixels around the border (zero-padding) to form X_p. Then each filter slides over X_p with stride s; at every spatial position, it computes a dot product of the local $C \times K_h \times K_w$ input patch and the filter weights, and adds a bias:

$$Y[n, k, y, x] = \sum_{c} \sum_{i=0}^{K_h-1} \sum_{j=0}^{K_w-1} W[k, c, i, j]\ X_p[n, c,\ ys + i,\ xs + j] + b[k].$$

Padding increases the effective input size and helps control the output spatial resolution, preserving border information.

In the backward pass of a convolutional layer, given the gradient from next layer $g = \frac{\partial L}{\partial Y}$, we compute gradients w.r.t. bias, weights, and the padded input. The bias gradient is the sum over all spatial locations:

$$\frac{\partial L}{\partial b[k]} = \sum_{n,y,x} g[n, k, y, x].$$

The weight gradient is obtained by correlating G with the corresponding input patches from the padded input X_p:

$$\frac{\partial L}{\partial W[k, c, i, j]} = \sum_{n,y,x} g[n, k, y, x]\ X_p[n, c,\ ys + i,\ xs + j].$$

The input gradient is computed by distributing G back through the transposed convolution with the same stride/padding conventions:

$$\frac{\partial L}{\partial X_p[n, c, u, v]} = \sum_{k} \sum_{i,j} G[n, k, y, x]\ W[k, c, i, j]$$

where $u = ys + i,\ v = xs + j$.

Finally, the gradient w.r.t. the original input X is obtained by removing the padded border from $\frac{\partial L}{\partial X_p}$.

A ReLU activation function is then applied by elements to the convolution output to introduce non-linearity. Formally, for each activation x, ReLU computes

$$\mathrm{ReLU}(x) = \max(0, x),$$

saving positive responses and setting negative values to zero. This operation increases the model's ability to identify nonlinear effects without changing the spatial resolution, and helps mitigate the effects of the disappearing gradient, thereby contributing to more stable and effective learning.

In the backward pass of ReLU, the gradient is propagated only through activations that were positive in the forward pass. For $y = \mathrm{ReLU}(x) = \max(0, x)$ and upstream gradient $g = \frac{\partial L}{\partial y}$, the input gradient is

$$\frac{\partial L}{\partial x} = g \cdot \mathbb{I}(x > 0),$$

Following this, a 2×2 max-pooling layer is applied. This operation selects the maximum value within each non-overlapping 2×2 window, reducing the spatial dimensions by a factor of two and producing an output tensor of size $6 \times 22 \times 22$.

During the backward pass, the upstream gradient $G = \frac{\partial L}{\partial Y}$ is passed only to the input element that achieved this maximum in the forward pass; all other elements in the window receive zero:

$$\frac{\partial L}{\partial X[n, c, u, v]} = \sum_{y,x} G[n, c, y, x]\ \mathbb{I}((u, v) = \underset{(i,j)}{\mathrm{argmax}}\, X[n, c, ys + i, xs + j]).$$

In our program, the mask is saved (1 - if the element, 0 - otherwise) during the forward pass and then used during the backward pass.

Second convolutional block. The second convolutional layer consists of 16 filters of size 5×5, again with stride 1 and no padding:

$$\mathrm{Conv}(16, 5 \times 5, S = 1, P = 2).$$

The input to this layer is a $6 \times 22 \times 22$ tensor, and the resulting output has spatial dimensions 22×22, yielding a tensor of size $16 \times 22 \times 22$. The number of trainable parameters in this layer is

$$16 \times 6 \times 5 \times 5 + 16 = 2416.$$

As in the previous block, the convolution is followed by a ReLU activation and a 2×2 max-pooling layer, which reduces the output to a tensor of dimension $16 \times 11 \times 11$.

Fully connected layers. After the convolutional stages, the feature maps are flattened into a one-dimensional vector of length $16 \cdot 11 \cdot 11 = 1936$. This vector is passed to a fully connected layer with 120 neurons, denoted as FC(120). Each neuron is connected to all input features, resulting in $120 \cdot (1936 + 1)$ trainable parameters, including biases. We also initialize these weights using the Kaiming algorithm. A ReLU activation is applied to the output of this layer to introduce non-linearity. Without a non-linear activation between fully connected layers, a stack of FC layers collapses into a single linear transformation: for three consecutive layers

$$h_1 = W_1 x + b_1, \quad h_2 = W_2 h_1 + b_2, \quad h_3 = W_3 h_2 + b_3,$$

the overall mapping can be rewritten as

$$h_3 = (W_3 W_2 W_1)\, x \ + \ (W_3 W_2 b_1 + W_3 b_2 + b_3),$$

which is equivalent to one FC layer with weight $W_{\text{eff}} = W_3 W_2 W_1$ and bias $b_{\text{eff}} = W_3 W_2 b_1 + W_3 b_2 + b_3$.

The network then includes a second fully connected layer with 84 neurons, followed by another ReLU activation. Finally, the output layer is a fully connected layer with 18 neurons, corresponding to the 18 target classes.

After this layer, a Softmax activation is applied to produce a probability distribution over classes. For logits from third FC layer $x \in \mathbb{R}^K$, Softmax is defined as

$$p_i = \mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \quad i = 1, \ldots, K.$$

This ensures that each output neuron represents a predicted probability for the corresponding class with $p_i \geq 0$ and $\sum_{i=1}^{K} p_i = 1$.

3.3.2. Fully Connected Neural Network

The fully connected neural network (FCNN) serves as a baseline model for comparison with the convolutional architecture. Providing a background against which to evaluate the benefits of convolutional feature extraction.

Network structure. The network follows a feedforward architecture with layer dimensions [784,64,32,C], where 784 corresponds to the flattened input image of size 28×28, the two hidden layers contain 64 and 32 neurons respectively, and C denotes the number of output classes. The use of C as a parameter allows flexibility in adapting the network to datasets with different numbers of target classes.

Activation functions. The hidden layers employ the sigmoid activation function, which introduces smooth nonlinearity and enables the network to model nonlinear decision boundaries.

The output layer uses the softmax activation function, which transforms the raw output scores into a normalized probability distribution over the C classes. This choice is well suited to multi-class classification tasks, as it enables direct interpretation of the network output as class probabilities.

Regularization. To mitigate overfitting, L2 regularization was applied to the network weights during training. This technique penalizes large weight values and encourages smoother solutions, improving generalization performance on unseen data.

Loss function and optimization. The network was trained using the categorical cross-entropy loss function, which is commonly used in multi-class classification problems and aligns naturally with the softmax output layer. Gradient-based optimization was employed to minimize the loss function during training.

3.4. Training Procedure

3.4.1. Convolutional Neural Network

The training results of the latest version of the model with three fully connected layers are as follows. The training was conducted over 10 epochs on a dataset of 130,000 images and took 1564 seconds. The learning rate stepsize of 0.001 was used for epochs 1-5, and 0.0005 for epochs 6-10. Such a step reduction is necessary in order to quickly reach the area of a good solution first, and then adjust the weights more carefully and reduce fluctuations near the minimum of the loss function.

3.4.2. Fully Connected Neural Network

Hyperparameters were selected through manual tuning, guided by prior work on handwritten character recognition and empirical experimentation. The learning rate was fixed at 0.001, which provided stable convergence during training.

The number of training epochs depended on the dataset configuration: when training on a single combined dataset, 15 epochs were sufficient, whereas training with separate training and testing folders required up to 80 epochs to achieve optimal performance. These settings resulted in the best observed trade-off between training stability and generalization.

The inference pipeline is identical for both the fully connected and convolutional neural networks. An input image containing a handwritten arithmetic expression is first passed through a segmentation stage, which decomposes the single-line expression into individual character components.

Each segmented character is then processed using the preprocessing pipeline corresponding to the selected model, resulting in grayscale images of fixed size with normalized pixel values. These images are subsequently fed into the trained neural network, which produces label predictions for each character. The resulting sequence of predicted labels serves as input for the construction of the expression tree and subsequent evaluation.

3.4.3. Expression Segmentation

While the convolutional neural network was trained exclusively on isolated characters, the final system was designed to recognize complete handwritten arithmetic expressions. To bridge this gap, a segmentation step was introduced during inference to decompose full expressions into individual character regions before classification.

Segmentation was performed using the OpenCV computer vision library [3], which was employed to detect and extract bounding boxes corresponding to individual symbols within an input image. Each segmented character was then resized and normalized using the same preprocessing pipeline described above before being passed to the neural network for prediction. By separating the tasks of segmentation and classification, the model remains focused on symbol recognition while leveraging established computer vision techniques for expression parsing.

3.4.4. Expression Parsing and Evaluation

Arithmetic expressions. The sequence of symbols predicted by the convolutional neural network is first parsed into a structured representation using operator-precedence parsing with syntax-directed tree construction [6]. This approach ensures that the resulting expression tree correctly reflects standard arithmetic precedence rules.

Parsing is performed using a double-stack algorithm that processes infix expressions directly. Operands are pushed immediately onto the operand stack, while operators are temporarily stored on an operator stack and ordered according to their precedence. Higher-precedence operators (e.g., multiplication) are applied before lower-precedence ones (e.g., addition), ensuring correct evaluation order. The output of this procedure is a binary expression tree.

Once constructed, the expression tree can be evaluated numerically. A post-order traversal (left → right → root) of the tree yields a Reverse Polish Notation (RPN) representation, which enables efficient and unambiguous computation of the expression value.

Polynomials. For expressions corresponding to single-variable polynomials, the parsed binary expression tree is further transformed into a polynomial representation. This transformation is achieved by recursively applying the distributive property to expand products and combine like terms, resulting in an explicit polynomial form.

To compute the roots of the resulting polynomial, the Newton–Raphson method is employed. This iterative numerical technique allows for efficient root approximation but is restricted to real-valued solutions. Consequently, the current system is limited to identifying real roots of single-variable polynomials.

4. Results

The confusion matrix was built in both cases on the validation folder, which contains 100 examples for each character. This folder was assembled with the help of several people and does not contain data from the three datasets discussed above.Figure 1a shows the confusion matrix for the FNN model, and Figure 1b for the CNN model.

It can be noted that in the case of FNN, the matrix has more non-zero elements outside the diagonal than in the case of CNN. Thus, we can confirm that CNN is doing a better job of our task. It can also be noted that the model is generally good at predicting handwritten characters.

Model performance was primarily evaluated using confusion matrices, which provide detailed insight into classification behavior across classes. Overall accuracy and per-class accuracy were used as the main quantitative performance metrics.

The test accuracy of the convolutional neural network was recorded at 98.6%, compared to 89.33% for the fully connected neural network.

The convolutional neural network not only achieved higher accuracy but also scaled more effectively to a larger symbol set, enabling reliable recognition of both arithmetic expressions and polynomials in a single training cycle—something the fully connected network could not accomplish.

All the above again ties to the inherent architectural differences between the two models

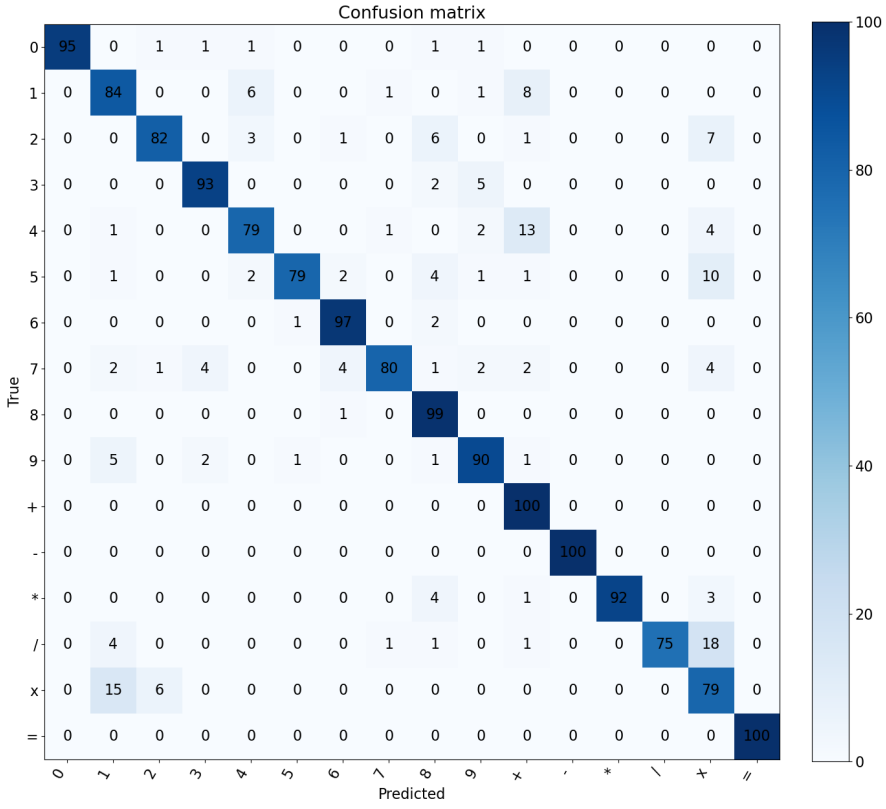
5. Limitations

Due to the nature of the time constraint on this project, some limitations are still present. One of which is the reliance on the segmentation of the expressions prior to the prediction. Other limitations include a relatively high confusion rate between similarly shaped items, particularly across classes (digits/symbols/letters). For example, a high confusion rate was found between 4 and +. In addition, the challenge of dealing with overlapping symbols was not overcome, due to its reliance on more complex image processing, which was outside the scope of this project, whose main focus was on developing a working neural network. Because the proposed models operate primarily at the recognition stage, they are sensitive to errors arising from segmentation ambiguities, disconnected symbols, or atypical spatial layouts. Figure 2 shows examples of limitations that we cannot solve with our programs.

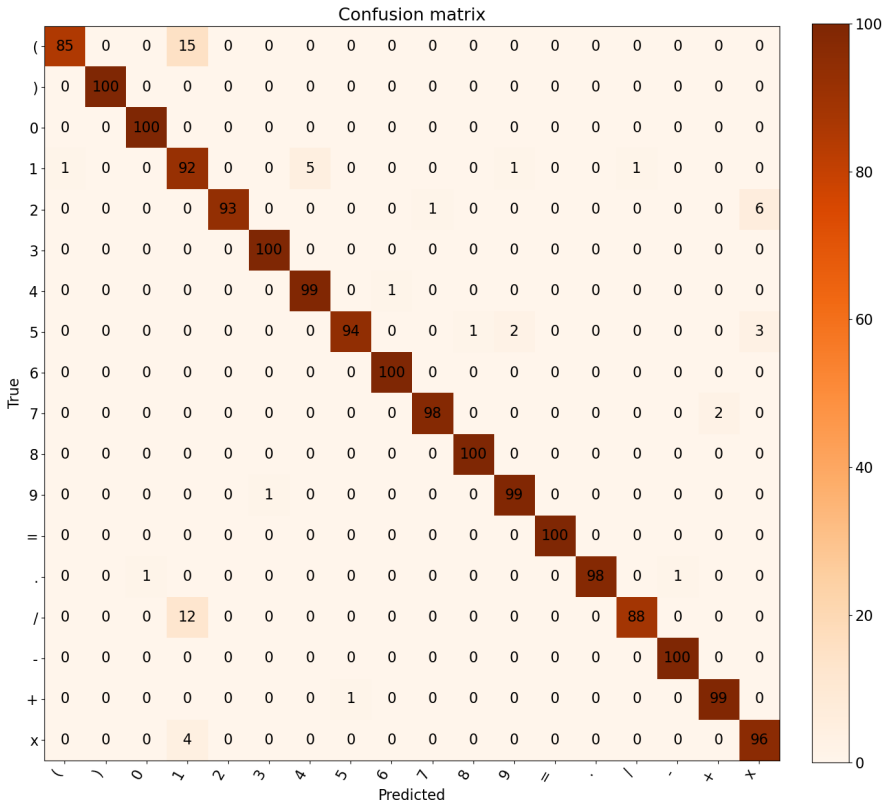
These limitations are consistent with prior observations that robust handwritten mathematical expression recognition requires coordinated segmentation and structural interpretation.

Acknowledgements.

We would like to thank M. Marek Bucki for his support throughout the project.

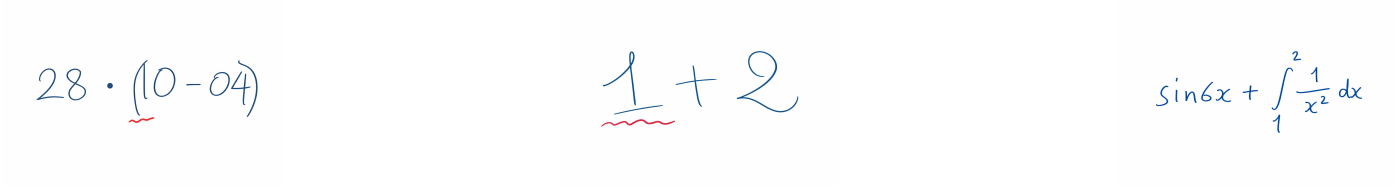


(a)



(b)

Fig. 1: Confusion matrices for (a) FCNN and (b) CNN.



(a) (b) (c)

Fig. 2: Typical failure cases: (a) stuck symbols, (b) character gaps, (c) special math symbols.

References

[1] Savita Ahlawat et al. "Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN)". In: *Sensors* 20 (June 2020), p. 3344. DOI: 10.3390/s20123344.

[2] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. "A global learning approach for an online handwritten mathematical expression recognition system". In: *Pattern Recognition Letters* 35 (Jan. 2014), pp. 68–77. DOI: 10.1016/j.patrec.2012.10.024. (Visited on 04/23/2022).

[3] G Bradski. *The OpenCV library*. Dr. Dobb's Journal of Software Tools, 2000.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016, pp. 165–167. URL: https://www.deeplearningbook.org/.

[5] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63 (Oct. 2014), pp. 139–144. DOI: 10.1145/3422622. URL: https://dl.acm.org/doi/pdf/10.1145/3422622.

[6] Dick Grune and H Jacobs. "Parsing Techniques: A Practical Guide, 2nd edition". In: *Data Archiving and Networked Services (DANS)* (Jan. 2008). (Visited on 01/16/2026).

[7] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv preprint arXiv:1502.01852* (2015). arXiv: 1502.01852 [cs.CV]. URL: https://arxiv.org/abs/1502.01852.

[8] Kairui Jin. "Handwritten digit recognition based on classical machine learning methods". In: *2022 3rd International Conference on Electronic Communication and Artificial Intelligence (IWECAI)*. 2022, pp. 163–173. DOI: 10.1109/IWECAI55315.2022.00040.

[9] Jyoti et al. "Handwriting Recognition: Unravelling Performance Diversity and Practical Implications". In: *Procedia Computer Science* 259 (2025), pp. 1387–1397. DOI: 10.1016/j.procs.2025.04.093. (Visited on 09/09/2025).

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Communications of the ACM* 60 (May 2012), pp. 84–90.

[11] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86 (2018), pp. 2278–2324.

[12] Harold Mouchère et al. *ICFHR 2014 competition on recognition of online handwritten mathematical expressions (CROHME 2014) dataset*. 2014.

[13] Xai Nano. *Handwritten math symbols dataset*. Kaggle.com, 2017. URL: https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols/data.

[14] C.A. Perez, C.M. Held, and P.R. Mollinger. "Improved handwritten digit recognition system based on fuzzy rules and prototypes created by Euclidean distance". In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*. Vol. 4. 2000, 2715–2720 vol.4. DOI: 10.1109/ICSMC.2000.884406.

[15] R. Plamondon and S.N. Srihari. "Online and off-line handwriting recognition: a comprehensive survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), pp. 63–84. DOI: 10.1109/34.824821.

[16] S M Shamim et al. "Handwritten Digit Recognition Using Machine Learning Algorithms". In: *Indonesian Journal of Science and Technology* 3 (Apr. 2018), p. 29. DOI: 10.17509/ijost.v3i1.10795.

[17] wblachowski. *GitHub - wblachowski/bhmsds: Basic Handwritten Math Symbols Dataset*. GitHub, 2025. URL: https://github.com/wblachowski/bhmsds/tree/master?tab=MIT-1-ov-file (visited on 01/16/2026).

[18] Thi Thi Zin et al. "Handwritten Character Recognition on Android for Basic Education Using Convolutional Neural Network". In: *Electronics* 10 (Apr. 2021), p. 904. DOI: 10.3390/electronics10080904. (Visited on 05/28/2021).