A Lightweight CNN for Multi-Class Classification of Handwritten Digits and Mathematical Symbols

Nicholas Abisha[1], **Tita Putri Redytadevi**[2], **Sri Nurdiati**[3], **Elis Khatizah**[4], **Mohamad Khoirun Najib***[5]
E-mail : nicholasabisha@apps.ipb.ac.id[1], titaputriredytadevi @apps.ipb.ac.id[2], nurdiati@apps.ipb.ac.id[3], elis_khatizah@apps.ipb.ac.id[4], mkhoirun@apps.ipb.ac.id[5]*
**Corresponding author*

Received 18 June 2025; Revised 17 July 2025; Accepted 23 July 2025

Abstract - Recognizing handwritten digits and mathematical symbols remains a nontrivial challenge due to handwriting variability and visual similarity among classes. While deep learning, particularly Convolutional Neural Networks (CNNs), has significantly advanced handwriting recognition, many existing solutions rely on deep, resource-intensive architectures. This study aims to develop a lightweight and efficient CNN model for multi-class classification of handwritten digits and mathematical symbols, with an emphasis on deployability in resource-constrained environments such as educational platforms and embedded systems. The proposed model, implemented in Julia using the Flux.jl library, features a compact architecture with only two convolutional layers and approximately 55,000 trainable parameters significantly smaller than typical deep CNNs. Trained and evaluated on a publicly available dataset of over 10,000 grayscale 28×28-pixel images across 19 symbol classes, the model achieves a test accuracy of 91.8% while maintaining low computational demands. This work contributes to the development of practical handwritten mathematical expression recognition systems and demonstrates the feasibility of using Julia for developing lightweight deep learning applications.

Keywords - Digits, Mathematical Symbol, Classification, CNN

1. INTRODUCTION

In an era marked by the rapid growth of internet access, digital academic resources such as journals and scientific papers increasingly incorporate mathematical expressions that require precise formatting using markup languages like LaTeX or MathML [1]. To support editing, publishing, and educational processes, there is growing demand for systems that can automatically convert handwritten mathematical notation into machine-readable formats.

However, recognizing handwritten mathematical symbols and expressions remains a complex problem. Variations in handwriting such as inconsistent size, spacing, and orientation make accurate detection and classification difficult [2,3]. These challenges are magnified in mathematical notation, which is two-dimensional in nature and includes structures like fractions, superscripts, and nested expressions that are not found in standard text [4]. As a result, handwritten mathematical expression recognition (HMER) has emerged as an active research area spanning computer vision, artificial intelligence, and pattern recognition.

To address these challenges, many recent studies have leveraged the power of deep learning. Convolutional Neural Networks (CNNs), in particular, have shown significant success due to their ability to extract location-invariant features while remaining computationally efficient compared to traditional deep neural networks [5]. CNNs have become the dominant approach in handwriting recognition, especially for digits and symbols, thanks to their resilience to geometric distortions such as rotation and scaling.

A prominent example is the DIGITNET model introduced by Kusetogullari et al. [6], which was trained on the historical DIDA dataset consisting of Swedish handwritten documents from the 19th century. DIGITNET uses a YOLO-based detection scheme and combines predictions from multiple CNNs using a voting mechanism, demonstrating superior accuracy even in challenging, noisy inputs. The system was explicitly designed to recognize digit strings written in diverse styles, reflecting the growing interest in robust solutions for real-world handwriting data.

While deeper architectures such as ResNets and encoder-decoder networks have achieved strong performance on benchmarks like ImageNet, they often require substantial computational resources. These constraints pose challenges for deployment in real-time or resource-limited environments. Furthermore, recent work has shown that even benchmark-driven gains may not always reflect true generalization [7], reinforcing the need for efficient yet robust alternatives. In the domain of HMER, encoder-decoder frameworks with attention mechanisms have achieved promising results, treating the problem as an image-to-sequence translation task into LaTeX format. However, models such as SAN (Syntax-Aware Network) have highlighted the need for better syntactic understanding by explicitly modeling the grammatical structure of mathematical expressions [8]. Even so, attention-based models still struggle with interpretability, especially since most do not segment symbols explicitly, making post-processing and user interaction more difficult [9].

To address these limitations, many researchers adopt a bottom-up approach that focuses first on the classification of individual handwritten symbols, including digits, operators, and variables, as a foundational step toward full HMER pipelines. Recognizing individual symbols accurately is critical because they form the atomic units of more complex expressions.

Therefore, this study proposes a lightweight convolutional neural network (CNN) for multi-class classification of handwritten digits and mathematical symbols. Implemented in Julia using the Flux.jl framework, the model is designed to be compact, with approximately 55000 parameters, and suitable for environments with limited computational resources. Unlike many existing models, the architecture emphasizes simplicity and interpretability while still achieving competitive performance. The model was evaluated on a publicly available dataset of over 10,000 grayscale images representing 19 symbol classes, and achieved a peak accuracy of 91.8%. This work contributes to the growing body of efficient deep learning systems for mathematical handwriting recognition, with practical implications for educational tools to support the teaching–learning process in various educational contexts, such as handwriting recognition [10,11], while also showcasing the viability of Julia and Flux.jl for developing deep learning models in scientific computing contexts.

This work contributes to the field by demonstrating that a compact CNN model with only 55K parameters, developed in Julia, can achieve strong symbol-level classification for HMER. This fills a practical gap between high-accuracy models and deployable systems in resource-constrained educational settings

2. RESEARCH METHOD

This study adopts an experimental computational approach to develop and evaluate a convolutional neural network (CNN) model for classifying handwritten digits and mathematical symbols. The process involves several stages, including data preprocessing, model construction, training, and evaluation.

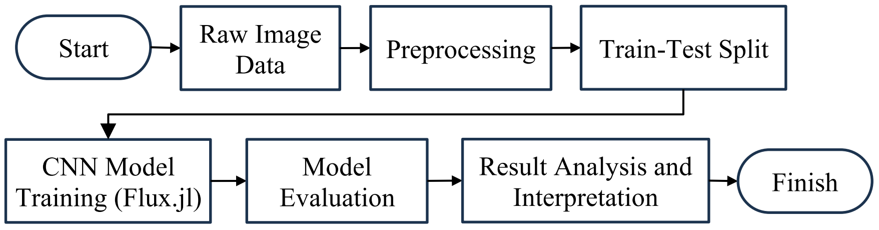

Figure 1. Samples of digits and mathematical symbols from dataset

The model is implemented in the Julia programming language using the Flux.jl library, which is optimized for numerical and deep learning tasks. The dataset used consists of grayscale images of handwritten symbols collected from an open-source repository, which are preprocessed and split into training and testing subsets. Each stage of the research method is detailed in the following subsections.

2.1. Data Loading and Preprocessing

The initial step in this study involves loading and preprocessing the handwritten symbol image data to prepare it for model training. The dataset used in this study is a publicly available collection titled "Handwritten Math Symbols" by Sagyam Thapa, hosted on Kaggle [12]. The dataset consists of grayscale images of 19 symbols representing digits (0-9) and various mathematical symbols (e.g., $+$, $-$, $=$, $\div$, x, y, z). These images are organized into subfolders named after their respective classes.

Table 1. Class Distribution by Symbol and Codename in the Dataset

Symbol	Codename	Number of Images	Symbol	Codename	Number of Images
0	0	595	$+$	add	596
1	1	562	.	dec	624
2	2	433	$\div$	div	618
3	3	541	$=$	eq	634
4	4	526	$\times$	mul	577
5	5	433	$-$	sub	655
6	6	581	x	x	452
7	7	533	y	y	399
8	8	554	z	z	212
9	9	546			

Each image, regardless of its original resolution, is first converted to grayscale if it is not already, and then resized to a uniform dimension of 28×28 pixels to ensure consistency in input shape for the CNN model. This resizing is performed using the Images.imresize function. Following this, the pixel values are normalized and reshaped into tensors with dimensions suitable for input into a convolutional neural network specifically, a shape of (28, 28, 1) to represent height, width, and the single color channel.

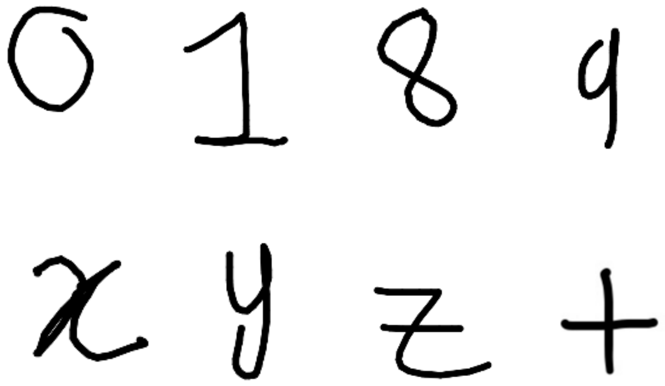

Figure 2. Samples of digits and mathematical symbols from dataset

Alongside image preprocessing, the corresponding labels for each image are collected and encoded using one-hot encoding to facilitate multi-class classification. The entire dataset is then stacked into a four-dimensional array representing the batch of input images and their associated categorical labels. This preprocessing pipeline ensures that the data is clean, uniformly formatted, and ready for the subsequent training and evaluation phases.

2.2. Dataset Splitting (Train-Test Stratification)

After preprocessing, the dataset is divided into two subsets: training data and testing data. This division follows a stratified sampling approach to ensure that each class is proportionally represented in both subsets, maintaining the overall class distribution.

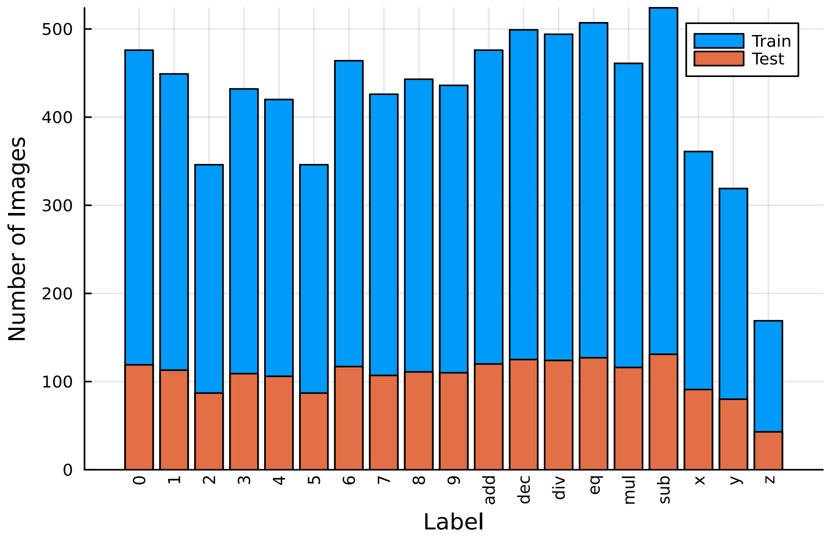

Figure 3. Train and test data distribution

As shown in Figure 3, 80% of the data is allocated for training and 20% for testing. For each class, the available image indices are shuffled randomly, then split into training and testing portions. This method helps prevent class imbalance issues that could bias the model during training or evaluation. The resulting training set is used to train the CNN model, while the testing set is held out for evaluating the model's ability to generalize to unseen data. This separation is crucial for assessing the real-world performance and robustness of the model.

2.3. Model Architecture Design

The classification model implemented in this study is a custom-designed Convolutional Neural Network (CNN) tailored for recognizing grayscale images of handwritten digits and mathematical symbols. This architecture was constructed using the Flux.jl deep learning library in the Julia programming language, offering flexibility and efficiency for model development and training.

The model accepts input images of size 28×28 pixels with a single color channel. The first layer is a 2D convolutional layer consisting of 8 filters of size 3×3, followed by a ReLU activation function and a 2×2 max-pooling operation that reduces the spatial resolution to 14×14. This is followed by a second convolutional layer with 16 filters of size 3×3, again followed by ReLU activation and a 2×2 max-pooling layer, which further reduces the feature map to a 6×6 spatial resolution.

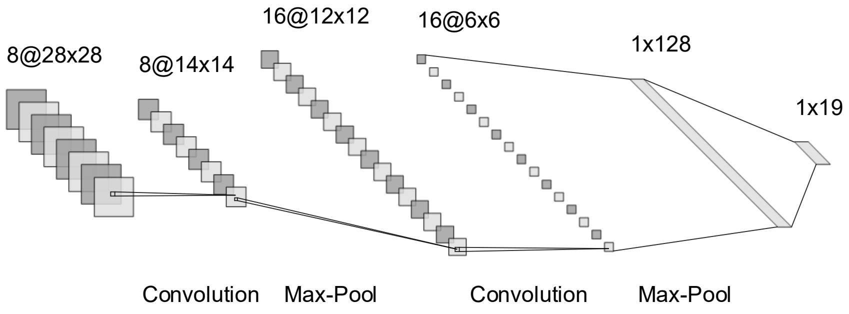

Figure 4. Model architecture

After these convolutional and pooling layers, the output feature maps are flattened into a one-dimensional vector of length 400 and passed to a fully connected layer with 128 units, using ReLU activation. The final output layer is a dense layer with 19 units, corresponding to the number of target classes, and employs the softmax activation function to produce a probability distribution over the classes. A visualization of the complete model architecture is shown in Figure 4, illustrating how the input image is transformed through each layer of the network from raw pixels to class probabilities.

The model was trained using the Adam optimizer with a learning rate of 0.001 and a batch size of 1 (online gradient updates). The loss function used was categorical cross-entropy, and training was run for 30 epochs. Online training was selected to simplify implementation and observe per-sample learning behavior, although it is generally slower than mini-batch approaches. No dropout or explicit regularization was applied.

Compared to widely used deep CNN architectures in handwriting recognition such as LeNet-5, AlexNet, or VGGNet, this model significantly reduces complexity by using only two convolutional layers with small filter sizes (3×3), minimal depth (8 and 16 filters), and no dropout, batch normalization, or residual blocks. The network has approximately 55000 trainable parameters, making it orders of magnitude smaller than typical CNNs, which often exceed millions of parameters. The proposed architecture is therefore considered lightweight due to its:

- Shallow depth (only two convolutional layers)
- Low parameter count (~55K)
- Small model size (easily trainable on CPU)
- Fast training time (on modest hardware)

This design makes the model highly suitable for deployment in resource-constrained environments such as educational platforms, mobile applications, and embedded systems without the need for specialized GPUs or cloud computing infrastructure.

2.4. Model Training

The training phase is conducted using the training subset produced through stratified splitting. The model is trained to minimize cross-entropy loss, a standard loss function for multi-class classification problems. The Adam optimizer is employed to update the model's parameters, offering adaptive learning rates that accelerate convergence and improve stability.

The training process is run for a fixed number of epochs, where one epoch represents a full pass through the entire training dataset. In each epoch, the model processes mini-batches of input images, computes the output predictions, and adjusts weights using backpropagation based on the computed loss. The model's performance is monitored after each epoch by evaluating its accuracy on the test set, allowing for early identification of overfitting or underfitting.

The training loop is implemented manually using Julia and Flux.jl primitives, offering full transparency and control over the process. During training, both the training loss and the test accuracy are recorded for each epoch to visualize learning progression over time. These metrics serve as the basis for model selection in the next stage. The training loss and test accuracy were recorded at each epoch, and their progression is discussed further in Section 4, along with visualizations that illustrate the model's learning behavior over time.

During training, the model's performance was evaluated after each epoch using the test dataset to monitor its generalization ability. The key evaluation metric used for model selection was test accuracy, measured at the end of each epoch. In addition, the training loss was tracked to identify signs of convergence or overfitting. After training was complete, the model that achieved the highest test accuracy was selected as the final model for further evaluation. This model was saved to disk using Julia's BSON.jl package, which stores the model parameters and structure for later reuse. The saved model was then reloaded for detailed testing and computation of evaluation metrics such as precision, recall, F1 score, and confusion matrix analysis. This evaluation and selection strategy ensures that the final results are based on the model with the best generalization performance, rather than one that simply performed well on the training data.

2.5. Performance Metrics and Confusion Matrix Analysis

To evaluate the classification performance of the model, a set of standard metrics derived from the confusion matrix were used. These metrics provide detailed insight into how well the model distinguishes each class, beyond overall accuracy. Evaluation was performed on the test dataset using the following per-class metrics:

- True Positive (TP): Correctly predicted instances of a specific class.
- False Positive (FP): Instances incorrectly predicted as a class but belonging to another.
- False Negative (FN): Instances of a class that were missed (predicted as something else).
- True Negative (TN): All other instances correctly predicted as not belonging to the class.

From these, the following metrics were computed for each class:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \quad (1)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \quad (2)$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN+FP}} \quad (3)$$

$$\text{F1} = \frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision+Recall}} \quad (4)$$

Precision measures how many of the model's positive predictions were actually correct, while recall focuses on how many of the actual positive cases the model successfully identified. Specificity looks at how well the model avoids false alarms by correctly identifying negative cases. Finally, the F1 score combines precision and recall into a single measure that balances both, making it especially useful when dealing with imbalanced class distributions.

To summarize performance across multiple classes, this study uses three common averaging strategies: macro, micro, and weighted averages. Let M_i represent the metric (such as precision, recall, or F1 score) for class i, and let n_i be the number of true instances (support) in class i. The macro average is calculated as the unweighted mean of the metric across all C classes:

$$\text{Macro-Average} = \frac{1}{C} \times \sum_{i=1}^{C} M_i \quad (5)$$

This treats each class equally, regardless of size. In contrast, the micro average aggregates all true positives, false positives, and false negatives across classes before computing the metric. For example, micro precision is defined as:

$$\text{Micro-Precision} = \frac{\sum \text{TP}_i}{\sum(\text{TP}_i+\text{FP}_i)} \quad (6)$$

This approach gives more weight to larger classes, making it suitable when class imbalance is present. Lastly, the weighted average computes the per-class metric as usual but averages them according to their class support:

$$\text{Weighted-Average} = \sum_{i=1}^{C} \frac{n_i}{N} \times M_i \qquad (7)$$

where $N = \sum_{i=1}^{c} n_i$ is the total number of instances. This provides a balance between treating all classes equally and reflecting the actual class distribution in the dataset.

3. RESULTS AND DISCUSSION

3.1. Model Performance Overview

The custom CNN model was trained on the preprocessed dataset of handwritten digits and mathematical symbols using the training set, with performance evaluated on the test set after each epoch. Out of all training epochs, the best performance was observed at epoch 17, where the model achieved a test accuracy of 91.8%. At this point, the training loss had significantly decreased from its initial value, indicating successful learning and convergence.

The final selected model, saved using the BSON format, is the one that achieved the highest test accuracy across all epochs. This model was subsequently reloaded for detailed evaluation using per-class metrics and confusion matrix analysis. Overall, the performance of the model indicates that the network was able to effectively learn the distinguishing features of most symbols in the dataset.

The results demonstrate that a relatively simple CNN architecture, when properly trained and regularized, can achieve strong performance on symbol classification tasks, even in the presence of visually similar classes and moderate class imbalance.

3.2. Training Progress and Convergence

To understand how the model's performance evolved during training, the training loss and test accuracy were recorded after each epoch. These metrics provide insight into the learning behavior of the model and help assess whether it converged properly or exhibited signs of overfitting or underfitting.

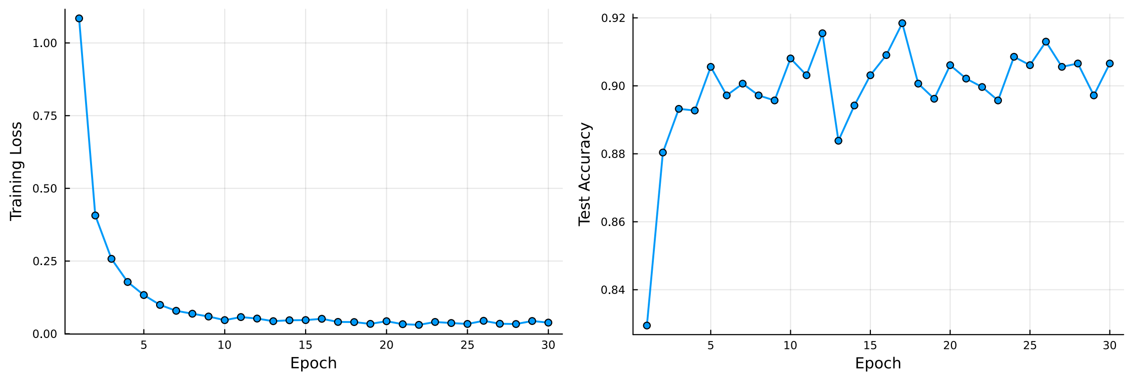

Figure 5. Cross-entropy loss and test accuracy recorded over each epoch

As shown in Figure 5, the training loss consistently decreased over time, indicating that the model was successfully minimizing the prediction error on the training data. The steep decline in early epochs reflects rapid learning, while the more gradual decline in later epochs suggests convergence toward an optimal solution. Meanwhile, test accuracy steadily improved, peaking at epoch 17 with a value of 91.8%, after which no significant improvements were observed. The absence of a major gap between training loss and test accuracy suggests that the model generalized well to unseen data, with minimal overfitting.

Overall, these trends confirm that the model reached convergence in a stable manner. The training progress indicates effective learning dynamics and supports the reliability of the selected model for further evaluation.

3.3. Confusion Matrix Analysis

To further evaluate the model's classification behavior across all symbol classes, a confusion matrix was generated based on predictions from the test dataset. The confusion matrix provides a detailed view of how often each class was correctly identified and where misclassifications occurred.

Table 2. Confusion matrix

T/P	0	1	2	3	4	5	6	7	8	9	add	dec	div	eq	mul	sub	x	y	z
0	105	2	1	0	0	0	2	1	5	1	0	0	0	0	0	0	2	0	0
1	0	107	3	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
2	2	0	81	0	0	0	0	1	0	0	0	0	1	0	0	0	2	0	0
3	0	0	3	101	0	0	2	0	1	1	0	0	0	1	0	0	0	0	0
4	1	0	0	0	93	0	3	0	0	4	0	0	0	0	0	0	3	2	0
5	1	0	0	3	0	74	5	0	1	1	0	0	0	0	0	0	1	0	1
6	2	0	0	1	2	1	103	0	4	0	0	0	1	0	0	0	2	0	1
7	0	1	0	2	0	0	0	100	0	0	3	0	0	0	0	1	0	0	0
8	1	0	0	2	0	0	9	1	95	2	0	0	0	0	0	0	0	1	0
9	2	1	0	1	0	0	0	6	2	98	0	0	0	0	0	0	0	0	0
add	0	0	0	0	1	0	1	0	0	0	113	0	1	0	0	1	3	0	0
dec	0	0	0	1	0	0	0	0	0	0	0	123	0	0	0	0	1	0	0
div	0	0	0	0	0	0	0	0	0	0	0	0	120	4	0	0	0	0	0
eq	0	0	2	0	0	0	0	0	0	0	0	0	0	121	0	3	1	0	0
mul	0	0	0	1	0	0	0	0	4	0	0	0	0	0	106	0	5	0	0
sub	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	128	0	0	0
x	0	0	0	0	1	0	0	2	0	0	2	0	1	0	0	1	84	0	0
y	0	1	1	1	3	0	0	1	0	2	0	1	1	0	0	0	4	65	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	41

The confusion matrix in provides a detailed breakdown of the model’s predictions across all 19 classes, comparing true labels (rows) to predicted labels (columns). A strong diagonal dominance is observed, indicating that the model correctly classified most instances in the dataset. The model performed consistently well on most digit classes, with very few misclassifications. For instance:

- Digit ‘0’ was predicted correctly 105 times, with only minor confusion, primarily with ‘8’ (5 cases), ‘1’ (2 cases), and ‘6’ (2 cases).
- Digit ‘1’ achieved 107 correct predictions, with a few misclassified as ‘2’ (3 times) or ‘3’ (once).
- Digit ‘4’ had 93 correct predictions, but was misclassified 4 times as ‘9’, and 3 times as ‘6’, which may be due to stylistic similarities in handwritten forms.
- Digit ‘5’ showed slightly more variability, with 74 correct predictions and misclassifications into ‘6’ (5 times), ‘3’ (3 times), and others.

Most other digits, such as ‘6’, ‘7’, ‘8’, and ‘9’, also showed strong accuracy, with only occasional misclassifications that followed predictable confusion patterns (e.g., ‘6’ mistaken for ‘4’ or ‘8’).
For mathematical operators, performance was very strong:

- ‘add’ ($+$) was predicted correctly 113 times, with only a handful of misclassifications into ‘sub’, ‘x’, ‘div’, and others.
- ‘dec’ (decimal point) had near-perfect accuracy with 123 correct classifications and 1 confused as ‘x’ and 1 confused as ‘3’.
- ‘div’ ($\div$) was predicted correctly 120 times, with 4 misclassifications as ‘eq’.
- ‘eq’ ($=$) had 121 correct predictions, with 3 being incorrectly predicted as ‘sub’ and 2 as ‘2’.
- ‘mul’ was predicted correctly 106 times with 10 misclassifications.
- ‘sub’ ($-$) was predicted correctly 128 times with 3 misclassifications.

The model also handled alphabetic variables like ‘x’, ‘y’, and ‘z’ fairly well, though with slightly higher confusion:

- 'x' had 84 correct predictions, but was confused with '7' (2 cases), '4, 'add, 'div', and 'sub' suggesting visual similarity with these digits and operators.
- 'y' had 65 correct classifications but was misclassified as '1', '2', '3', '4', '7', '9, and others indicating that 'y' may be particularly ambiguous in handwritten form.
- 'z' had 41 correct predictions with minor confusion into 'eq' (2 cases), reflecting a generally strong performance.

Overall, the confusion matrix confirms that the model performs robustly across both numeric and symbolic classes, with most errors concentrated among visually similar classes, particularly lowercase letters and digits. These patterns highlight both the strengths of the model and areas where further improvement (e.g., data augmentation or attention mechanisms) could be explored.

3.4. Per-Class Performance Metrics

To evaluate the model's ability to classify individual classes, key performance metrics, precision, recall, specificity, and F1-score were computed for each class based on the confusion matrix. These metrics provide a more nuanced view of the model's strengths and weaknesses beyond overall accuracy.

Table 3. Performance metrics per-class

Class	TP	FP	FN	TN	Precision	Recall	Specificity	F1
0	105	9	14	1895	0.921053	0.882353	0.995273	0.901288
1	107	5	6	1905	0.955357	0.946903	0.997382	0.951111
2	81	10	6	1926	0.89011	0.931034	0.994835	0.910112
3	101	13	8	1901	0.885965	0.926606	0.993208	0.90583
4	93	7	13	1910	0.93	0.877358	0.996348	0.902913
5	74	1	13	1935	0.986667	0.850575	0.999483	0.91358
6	103	23	14	1883	0.81746	0.880342	0.987933	0.847737
7	100	13	7	1903	0.884956	0.934579	0.993215	0.909091
8	95	17	16	1895	0.848214	0.855856	0.991109	0.852018
9	98	11	12	1902	0.899083	0.890909	0.99425	0.894977
add	113	5	7	1898	0.957627	0.941667	0.997373	0.94958
dec	123	3	2	1895	0.97619	0.984	0.998419	0.98008
div	120	6	4	1893	0.952381	0.967742	0.99684	0.96
eq	121	7	6	1889	0.945313	0.952756	0.996308	0.94902
mul	106	0	10	1907	1	0.913793	1	0.954955
sub	128	6	3	1886	0.955224	0.977099	0.996829	0.966038
x	84	24	7	1908	0.777778	0.923077	0.987578	0.844221
y	65	3	15	1940	0.955882	0.8125	0.998456	0.878378
z	41	2	2	1978	0.953488	0.953488	0.99899	0.953488

The model demonstrated strong and consistent performance across most classes, with precision and recall values exceeding 90% in the majority of cases. For instance, most digit classes maintained F1-scores around or above 0.90, indicating balanced performance in identifying and distinguishing these digits. The 'dec' (.) symbol achieved one of the highest F1-scores at 0.98, with precision and recall both above 95%, showing excellent detectability. Likewise, 'div' (÷) also performed with F1-score 0.96, suggesting minimal misclassification.

The symbol ‘mul’ and ‘eq’ (=) showed impressive precision and specificity values (both reaching 1.0, although ‘mul’ had slightly lower recall due to 10 false negatives. Interestingly, ‘x’ and ‘6’ exhibited the lowest F1-scores among all classes, at 0.84. These symbol were more frequently confused with digits or other variables, likely due to their visual similarity to characters such as ‘8’, especially in varied handwriting styles. In terms of trade-offs:

- ‘6’ had one of the lowest precision values (~81.7%) due to a relatively high number of false positives (23), but recall remained strong.
- ‘8’ had balanced but lower precision and recall (both ~85%), suggesting overall moderate difficulty in recognizing this symbol.
- ‘z’ achieved a perfect balance of precision, recall, and F1 (~95%), but on a much smaller sample, reflecting reliable classification when it appears.

Overall, these metrics confirm that the model is not only accurate in general but also robust across a diverse set of characters, with particularly strong performance in operators and well-defined digits. However, certain classes, especially ‘x’, ‘y’, ‘6’, and ‘8’ require further attention, possibly through expanded training data or handwriting augmentation strategies.

Table 4. Macro, micro, and weighted average performance metrics

Averaging	Precision	Recall	F1	Specificity
Macro	0.920671	0.915928	0.917075	0.995465
Micro	0.918438	0.918438	0.918438	0.995465
Weighted	0.921328	0.918438	0.918813	0.995391

To complement the per-class evaluation, aggregate metrics were computed using macro, micro, and weighted averaging strategies. As shown in Table 4, all three approaches yield very similar results, with precision, recall, and F1-scores ranging around 91% to 92%, indicating balanced performance across the dataset.

- The macro average, which treats all classes equally, yielded an F1-score of 0.917, reflecting strong performance even across minority classes.
- The micro average, which weighs each individual prediction equally, produced an F1-score of 0.918, suggesting high consistency across all predictions.
- The weighted average gave the highest overall precision at 0.921, due to the model's strength on high-support classes such as digits and common operators.

All versions of specificity are consistently above 0.995, confirming that the model rarely produces false positives across the board. These aggregated metrics reinforce the model’s robustness across diverse symbol types.

3.5. Interpretation of Results

The results demonstrate that the custom CNN model performs effectively in classifying handwritten digits and mathematical symbols, achieving a test accuracy of 91.8% and consistently strong performance across most classes. The confusion matrix, per-class metrics, and aggregate averages reveal that the model is capable of generalizing well to diverse inputs despite the natural variation in handwriting styles.

Classes such as ‘add’ (+), ‘sub’ (−), ‘dec’ (.), and most digits were recognized with high precision and recall, indicating that the model is well-suited for structured symbols with consistent visual patterns. This is further supported by high F1-scores and specificity values across these categories, confirming both strong sensitivity and low false positive rates. Additionally, the balanced macro and micro scores show that the model performs not only on dominant classes but also handles minority classes effectively.

However, certain classes, especially ‘x’, ‘y’, ‘6’, and ‘8’ showed slightly lower F1-scores. These patterns suggest that the model occasionally struggles to distinguish between characters with overlapping shapes or handwritten ambiguity.

From a practical standpoint, the model is well-suited as a symbol-level classifier in the early stage of a handwritten mathematical expression recognition system (HMER). Its strong performance on core symbols demonstrates its potential to support more complex parsing systems when paired with structural analysis or contextual models. Enhancements such as additional training data, handwriting augmentation, or incorporating attention mechanisms could improve its ability to disambiguate difficult cases.

In conclusion, the model strikes a strong balance between accuracy, generalization, and class fairness, and it provides a solid foundation for further development in mathematical handwriting recognition applications.

3.5. Comparison with Prior Work

Many recent handwriting recognition models have focused on achieving state-of-the-art accuracy using deep architectures. For instance, DIGITNET [5] report accuracies exceeding 96% but rely on attention mechanisms, and ensemble strategies. These models often exceed millions of parameters and require GPU-level hardware to train and deploy efficiently.

In contrast, the proposed model prioritizes efficiency and simplicity. With only ~55000 trainable parameters and a shallow architecture of two convolutional layers, the model achieves a respectable 91.8% accuracy on a public dataset. While not outperforming the most complex architectures in absolute accuracy, this approach offers advantages in terms of training speed, memory footprint, and feasibility on low-resource systems such as classroom PCs or embedded educational devices. This positions the model as a viable lightweight alternative for scenarios where real-time performance and computational efficiency outweigh marginal gains in accuracy.

4. CONCLUSION

This study developed and evaluated a custom Convolutional Neural Network (CNN) for classifying handwritten digits and mathematical symbols. Built using the Flux.jl library in Julia, the model was trained on grayscale images preprocessed to a standardized 28×28 format. The architecture, consisting of two convolutional-pooling layers followed by two dense layers, was designed to be compact and efficient while maintaining strong classification capability.

The model achieved a peak test accuracy of 91.8%, supported by consistently high per-class precision, recall, and F1-scores across 19 symbol categories. Particularly high-performing classes included common digits and mathematical operators such as ‘add’ (+), ‘sub’ (−), ‘div’ (÷), and ‘dec’ (.), many of which achieved F1-scores around or above 0.95. Analysis of the confusion matrix showed that most predictions aligned along the diagonal, with relatively few misclassifications.

Despite its strengths, the model showed reduced performance on more ambiguous handwritten symbols, such as ‘x’, ‘y’, ‘6’, and ‘8’, which had slightly lower precision or were confused with visually similar characters. These challenges are typical in handwritten symbol recognition, especially when symbol forms overlap across classes.

Aggregate metrics confirmed the model's robustness, with macro, micro, and weighted F1-scores all around 0.918 to 0.921, and specificity values above 0.995, indicating a low false-positive rate. These results highlight the model's reliability and generalizability, even in the presence of class imbalance and handwriting variability.

Overall, the CNN model demonstrates strong potential as a symbol-level classifier in a handwritten mathematical expression recognition (HMER) pipeline. Its efficiency and accuracy make it suitable for integration into educational tools, input systems, or mobile applications.

Future work could explore improvements through data augmentation, context modeling, or expanding the class set to cover more mathematical notation.

REFERENCES

[1] Kusuma Putra A, Bunyamin H. “Pengenalan simbol matematika dengan metode convolutional neural network (CNN),” *Jurnal Teknik Informatika*, vol. 2, 2020.

[2] Nirmalasari D.A. “Pengenalan teks tulisan tangan menggunakan fully convolutional neural network,” Skripsi, Institut Teknologi Sepuluh Nopember, 2020.

[3] Albahli S, Nawaz M, Javed A, Irtaza A. “An improved faster-RCNN model for handwritten character recognition,” *Arabian Journal for Science and Engineering*, vol. 46, pp. 8509–8523, 2021. [Online]. Available: https://doi.org/10.1007/s13369-021-05471-4

[4] Sakshi, Kukreja V. “A retrospective study on handwritten mathematical symbols and expressions: Classification and recognition,” *Engineering Applications of Artificial Intelligence*, vol. 103, p. 104292, 2021. [Online]. Available: https://doi.org/10.1016/j.engappai.2021.104292

[5] Ahmed S.S., Mehmood Z., Awan I.A., Yousaf R.M. “A novel technique for handwritten digit recognition using deep learning,” *Journal of Sensors*, vol. 2023, Article ID 2753941. [Online]. Available: https://doi.org/10.1155/2023/2753941

[6] Kusetogullari H., Yavariabdi A., Hall J., Lavesson N. “DIGITNET: A deep handwritten digit detection and recognition method using a new historical handwritten digit dataset,” *Big Data Research*, vol. 23, p. 100182, 2021. [Online]. Available: https://doi.org/10.1016/j.bdr.2020.100182

[7] Beyer L., Hénaff O.J., Kolesnikov A., Zhai X., van den Oord A. “Are we done with ImageNet?” *arXiv preprint*, 2020. [Online]. Available: https://arxiv.org/abs/2002.12335

[8] Yuan Y., Liu X., Dikubab W., Liu H., Ji Z., Wu Z., et al. “Syntax-aware network for handwritten mathematical expression recognition,” *arXiv preprint*, n.d. [Online]. Available: (perlu penelusuran URL)

[9] Tang J.-M., Guo H.-Y., Wu J.-W., Yin F., Huang L.-L. “Offline handwritten mathematical expression recognition with graph encoder and transformer decoder,” *Pattern Recognition*, vol. 148, p. 110155, 2024. [Online]. Available: https://doi.org/10.1016/j.patcog.2023.110155

[10] Ghadekar P., Khanvilkar O., Kharat J., Joshi K., Kulkarni T., Kulkarni Y. “Automating the grading of handwritten examinations through the integration of optical character recognition and machine learning algorithms,” *Journal of Integrated Science and Technology*, vol. 13, p. 1128, 2025. [Online]. Available: https://doi.org/10.62110/sciencein.jist.2025.v13.1128

[11] Silva L.C.E., Sobrinho Á.A.C., Cordeiro T.D., Melo R.F., Bittencourt I.I., Marques L.B., et al. “Applications of convolutional neural networks in education: A systematic literature review,” *Expert Systems with Applications*, vol. 231, p. 120621, 2023. [Online]. Available: https://doi.org/10.1016/j.eswa.2023.120621

[12] Thapa S. “Handwritten Math Symbols,” *Kaggle Dataset*, 2021. [Online]. Available: https://www.kaggle.com/datasets/sagyamthapa/handwritten-math-symbols