Real Time Hand Gesture Recognition for Mathematical Calculations

Sajal Korde[1], Shruti Bhargava[2], Snehil Sharma[3], Amit Rathi[4], Sanjay Garg[5]
Jaypee University of Engineering and Technology
Raghogarh, Distt. - Guna – Madhya Pradesh
[[1]sajalkorde01@gmail.com, [2] shrutibhargava2004@gmail.com, [3]ssnehil06@gmail.com, [4]amitrathi@gmail.com, [5]sanjaygarg@gmail.com (Corr. Author)]

Abstract: With advancements in computer vision and gesture recognition, contactless and intuitive natural human-computer interaction are increasingly viable - and crucial. Air-writing is a deduction to this type of interaction; it refers to the technique where users write characters or symbols in the air via finger movements, enabling a hygienic and innate form of interaction without any physical devices. In this paper, we present an air-writing calculator; it allows users to perform basic arithmetic calculations by drawing arithmetic expressions in the air. Our air-writing calculator tracks, processes, and interprets real time hand gestures with a video camera. The air-writing calculator works by converting the user's air-writing gestures into mathematical expressions by tracking the fingertips movement, which can assist the user to verify whether the output is correct. As the user completes an air-writing gesture, the location of the index fingertip's trajectory will be captured as an image. The image contains the computed trajectory of the finger, and will be classified using a Convolutional Neural Network (CNN). We achieved 99.28% classification accuracy using a dataset of hand-written digits and symbols. Once the eye-writing gesture is recognized, it is parsed, validated, evaluated, and displayed on the screen. The user can now interact and conduct calculations all without contact or the use of a physical device.

Keywords: Air-writing, Hand Gesture Recognition, Computer Vision, Convolutional Nueral Network, Mathematical Expression Recognition, AI calculator.

1 Introduction

Conventional input techniques like keyboards or touch screens are problematic for writing complex mathematical expressions. In mobile or space-constrained environments, traditional typing input for mathematical symbols, like square roots, fractions, and special characters, is slow and cumbersome. This is a frustrating limitation for those in educational and academic contexts such as students, educators, engineers, and researchers who must regularly enter mathematical formulas quickly and accurately. Air-writing is a desirable, alternative option. Air-writing allows the user to draw characters into the air using simple hand gestures, while often allowing for a potentially more natural, contactless way to work with their digital devices.
Drawing on recent developments in computer vision and deep learning, real-time identification of hand gestures has recently become both logic and reliable. Technologies such as MediaPipe offer the ability to use a common webcam to detect hand and finger movement with high confidence. At the same time, Convolutional Neural Networks (CNNs) are now very good at recognizing and classifying handwritten digits and mathematical symbols.
This paper presents a real-time air-writing calculator that integrates the aforementioned technologies. The system records hand motion from a webcam, traces fingertip movement using MediaPipe, and produces an image of the tracked motion. The image is passed to a CNN model that has been trained to classify a dataset of 23 different mathematical symbols including digits and operators. When the system recognizes a symbol, it will append it to a running expression. When the user indicates that they completed the expression, the system parses it, evaluates it, and displayed the result.

The key contributions of this project are: (1) a reliable and real-time hand tracking and gesture recognition system, (2) a CNN model trained on a dataset of handwritten digits and custom math symbols that achieves high classification accuracy, and (3) a fully functional air-writing calculator that evaluates expressions on the fly. This system creates a touchless, user-friendly interface that can be especially helpful in educational settings or for users with limited mobility. It also lays the groundwork for future systems that use hand gestures for broader forms of input and control.

2 Related Work

Certain developments are seen in the field of hand gesture recognition for mathematical calculation in recent years. These systems aim to bridge the gap between human intention and digital interpretation, offering alternatives to traditional input tools like keyboards or styluses.

One approach in this domain centres around the use of convolutional neural networks to interpret hand-drawn numerals in real time. In a study by Jabde et al. [1], the authors introduced a model capable of recognizing numeric gestures in both Devanagari and English. By combining palm detection with fingertip tracking, the system offered high accuracy while requiring only standard video input from a webcam. This minimized hardware dependency makes the approach particularly attractive for wider applications, including educational technology.

Mukherjee et al. [2] focused their efforts on improving the reliability of finger motion recognition in video sequences. They designed a system that begins with detecting the pose of the writing hand, then tracks the fingertip using a curvature-based function. The authors also proposed a velocity threshold method to determine when a user starts or stops writing. Their results demonstrated not only strong recognition performance but also the system's potential for real-time use.

Another notable contribution comes from Roy et al. [3], who built a framework that recognizes numerals formed by unistroke gestures. The method uses a coloured marker for easier tracking via basic video processing. Their system showed high accuracy across several languages and emphasized the benefits of applying transfer learning to adapt existing models to new datasets with minimal training.

Wu et al. [4] tackled the challenge of recognizing complex character sets, particularly those used in logographic scripts like Chinese. Their work introduced a large-scale dataset recorded using everyday cameras, along with a recognition model that extracts motion patterns and structural features of the characters. Their system demonstrated notable improvements over previous models, particularly in handling characters with multiple strokes and varying visual cues.

In contrast to the aforementioned real-time systems, Nazemi et al. [5] addressed symbol recognition in static images. Their focus was on mathematical notation, where visual similarities between symbols often pose classification difficulties. Through a combination of segmentation techniques and deep learning models, the authors were able to achieve solid performance across a wide range of symbols. While this study dealt with offline inputs, its insights are valuable for any system dealing with complex character sets.

Together, these works reflect the progress being made toward more intuitive and accessible interaction systems. Whether through real-time gesture recognition or offline symbol analysis, the research underscores a broader movement toward interpreting human expression in digital contexts with greater nuance and accuracy.

3 Proposed Method

The proposed system enables real-time recognition and evaluation of mathematical expressions written in the air using hand gestures, leveraging a combination of computer vision and deep learning. The process begins with video input captured through a webcam. The system uses the MediaPipe Hands module for real-time detection and tracking of hand landmarks [6]. When the index finger is raised, its tip coordinates are used to draw a continuous trace on a virtual canvas, effectively allowing the user to "write" in mid-air.

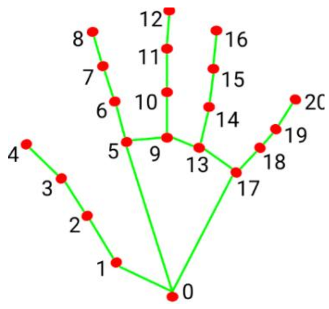

Figure 1: 21 Hand Landmarks

Preprocessing follows once the gesture is completed. Specific gestures serve distinct purposes raising only index finger allows user to draw (Figure 2 (A)), raising all five fingers clears the canvas and current expression (Figure 2(B)), raising the index and pinky fingers triggers the prediction phase (Figure 2(C)), four fingers up (excluding thumb) signal the evaluation of the expression (Figure 2(D)), and a raised pinky alone removes the last entered symbol (Figure 2(E)). The drawn image is cropped, thresholded, resized to 28×28 pixels while maintaining aspect ratio, cantered on a black canvas, and normalized to match the format of the training dataset.

This pre-processed image is then passed to a Convolutional Neural Network (CNN), which classifies it into one of 23 possible symbols (digits 0–9 and 13 mathematical operators/functions). The output symbol is appended to the current expression. Upon receiving the evaluation gesture, the entire expression is parsed and computed using Python's expression evaluation libraries. Results are displayed live on the screen, providing immediate feedback to the user. This interactive gesture-based interface allows for seamless air-writing and evaluation of mathematical expressions without physical contact or external tools.

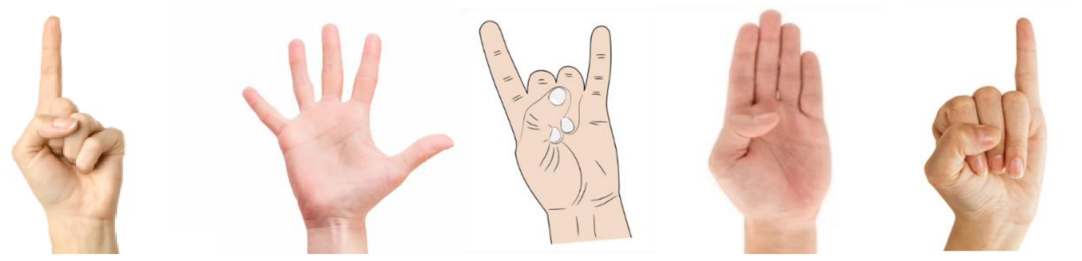

Figure 2: Hand Gestures

3.1 Pre-Processing

The classification model developed in this work relies on two key sources of data: the MNIST dataset for handwritten digits and a custom-curated dataset containing mathematical operators and special symbols. To ensure uniformity and compatibility with the neural network input requirements, all images underwent systematic preprocessing.

3.1.1 MNIST Dataset

The MNIST dataset which can be accessed via keras.datasets.mnist, contains grayscale images of handwritten digits ranging from 0 to 9. Each image is 28×28 pixels in size. The preprocessing of this dataset involved:

- **Normalization**: In order to scale all pixel values from [0,255] to [0,1], they were divided by 255.0. The model performs better and converges more quickly during training because to this standardization.

- **Reshaping**: Convolutional neural networks require input data with a channel dimension, therefore each image was reshaped from (28, 28) to (28, 28, 1), where the final dimension denotes a single grayscale channel.

3.1.2 Custom Operator and Symbol Dataset

To extend the calculator's functionality beyond numerical digits, a custom dataset of 13 operator and function symbols (+, −, *, /, !, (,), [,], {, }, pi, sqrt) was created and standardized to match the MNIST format. Each image was first converted to grayscale and processed using binary inverse thresholding to enhance symbol visibility against a dark background. The region containing the symbol was then cropped using bounding box extraction and resized proportionally to fit within a 20×20 area. To maintain consistency, the resized symbol was cantered on a 28×28 black canvas, replicating MNIST's layout. All images were normalized to a [0, 1] pixel intensity range and organized into class - labelled folders for training and testing. Invalid or unreadable images were removed during preprocessing. This uniform preprocessing pipeline ensured consistent input across all 23 symbol classes—digits and operators—resulting in a clean, balanced dataset optimized for CNN-based classification.

Table 1. Details of Preprocessing for Digit and Symbol Recognition

Parameter	**Description**	**Value**
Rescaling	Normalizes pixel intensities to the range [0, 1]	1/255
Grayscale Conversion	Converts RGB images to grayscale	Enabled
Thresholding	Applies binary inverse thresholding to highlight symbols	Threshold = 50
Cropping	Extracts bounding box around symbol pixels	Enabled
Resize	Resizes symbol to fit 20×20 while maintaining aspect ratio	Dynamic
Canvas Standardization	Centres resized symbol on 28×28 black background	(28, 28)
Image Format	Standardizes input shape for CNN	(28, 28, 1)

3.2 Deep Learning Architecture

The core of the proposed system's recognition capability lies in a custom-designed Convolutional Neural Network (CNN) [7]. It is optimized for classifying handwritten digits and mathematical symbols. The CNN is used for identifying 23 distinct classes, which include the digits 0–9 and various operators and functions such as addition (+), subtraction (−), multiplication (×), division (÷), factorial (!), brackets of different types i.e. parentheses (), square brackets [], curly braces {}, as well as the mathematical constants and functions π and √. All input images are standardized to a 28×28 pixel grayscale format and normalized to the [0, 1] range to match the style of the MNIST dataset which ensures uniformity in input data distribution and promoting better generalization during training.
The CNN structure begins with a convolutional layer that has 32, 3×3 filters which extract low-level features from the image, such as edges and simple curves. This is then followed by an activation function that is a ReLU (rectified linear unit) [8]. The ReLU function gives our model non-linearity, allowing the model to learn complicated patterns. Next, we have what is called a 2×2 max pooling layer, which eliminates redundant features to down sample the feature maps. The max pooling in CNNs down samples to reduce the complexity of computations, achieves translation invariance, and helps eliminate the risk of overfitting. This is done again for a second convolutional layer which also has 64 filters, followed again by a ReLU activation and max pooling. These filters are deeper with more complex

weights to extract more complicated patterns and hierarchy of structures in the images, such as the intersections or corners of one symbol vs. another.

The new feature maps are flattened into one single long vector and put into a fully connected dense layer with 128 neurons. The dense layer merges the features provided by the individual convolutional layers; these features are also combined to be classified during interpretation. Also, to help prevent overfitting and improve generalization on unseen dataset a dropout layer is added post dense layer with a dropout of 0.3. A dropout of 0.3 means that 30% of the neurons are randomly switched off at the training step and thus requires the model to learn more robust and independent features.
The final output layer consists of 23 neurons (one for each class) using the softmax activation function to output a probability distribution across all possible classes [9]. The class with the highest probability was selected as the predicted label. The model is trained using the Adam optimizer and Categorical cross-entropy loss function. Adam optimizer is known for its adaptive learning capabilities and Categorical cross-entropy is standard for multi-class classification tasks. During training, the model demonstrated strong convergence and achieved a test accuracy of 99.28%, showcasing its high reliability in symbol recognition.
In real-time operation, this trained CNN processes user-drawn expressions captured via webcam. After each drawing is completed (as detected by specific hand gestures), the drawn image is pre-processed and fed into the CNN. The CNN outputs the predicted symbol, which is appended to the current mathematical expression. Once the expression is complete and the evaluation gesture is recognized, the system parses and computes the expression's result. This tight integration between visual recognition and symbolic computation makes the CNN a critical component in enabling a seamless and interactive air-writing calculator.
The final output layer consists of 23 neurons (one for each class) using the softmax activation function to output a probability distribution across all possible classes. The class with the highest probability was selected as the predicted label. The model is trained using the Adam optimizer and Categorical cross-entropy loss function. Adam optimizer is known for its adaptive learning capabilities and Categorical cross-entropy is standard for multi-class classification tasks. During training, the model demonstrated strong convergence and achieved a test accuracy of 99.28%, showcasing its high reliability in symbol recognition.
In real-time operation, this trained CNN processes user-drawn expressions captured via webcam. After each drawing is completed (as detected by specific hand gestures), the drawn image is pre-processed and fed into the CNN. The CNN outputs the predicted symbol, which is appended to the current mathematical expression. Once the expression is complete and the evaluation gesture is recognized, the system parses and computes the expression's result. This tight integration between visual recognition and symbolic computation makes the CNN a critical component in enabling a seamless and interactive air-writing calculator.

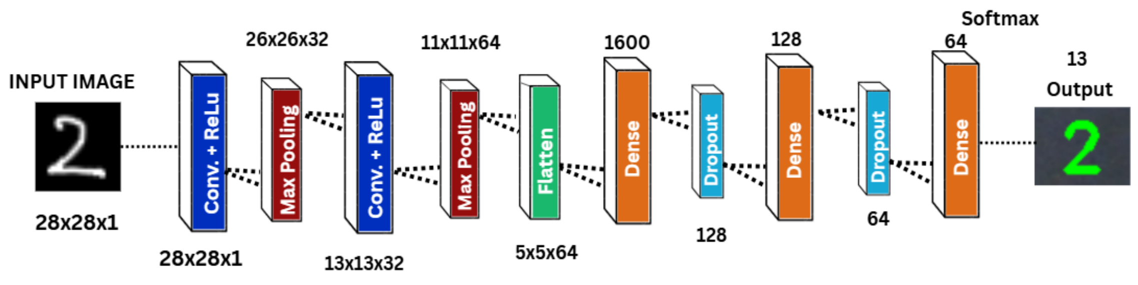

Figure 3: Proposed CNN Model Architecture

4 Experimental Results

The experimental evaluation of the proposed air-writing calculator system was carried out on a laptop equipped with an 11th Gen Intel(R) Core (TM) i5-1155G7 processor running at 2.50 GHz and 16 GB of RAM. The system was tested in both offline training and real-time execution environments to evaluate its effectiveness in symbol recognition and expression solving. This section presents the observed performance and outcomes based on these configurations.

4.1 Dataset And Performance Metrics

Handwritten digits and operator dataset consists of 23 classes which was collected from three different sources. The handwritten digits (0-9) dataset was taken from MNIST dataset [10] which is a publicly available benchmark dataset widely used for training and testing in the field of machine learning and computer vision. The dataset for basic mathematical operators [11] like addition, subtraction, multiplication and division was taken from Kaggle. This dataset was further extended to include symbols like '(', ')', '[', ']', '{', '}', '!' and constant pi [12]. This dataset was also taken from Kaggle. The combined dataset has 128599 images which was divided into training and testing. The training dataset has 107628 images and test dataset has 20971 images. Few images of dataset are shown in Fig. highlighting the visual diversity within the dataset images.

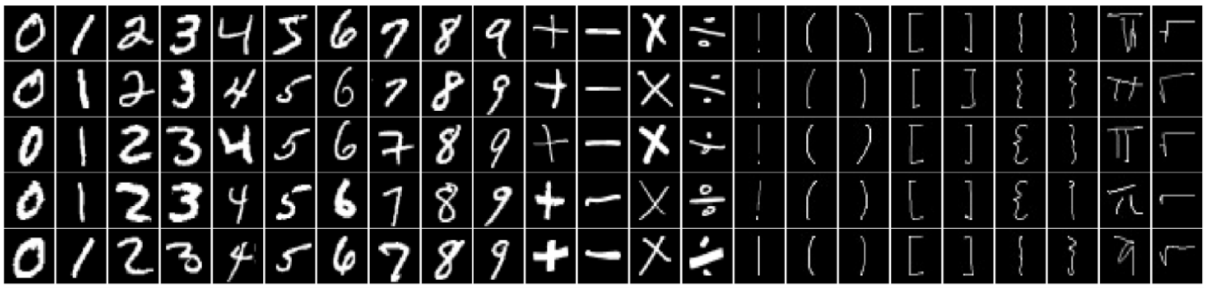

Figure 4: Sample instances of Dataset

In order to assess the model's performance, we looked at several measures, including a confusion matrix, ROC-AUC, accuracy, precision, recall, F1-score, and precision-recall curve [13][14]. Together, these metrics give us a general picture of how well the model performs in relation to various classification-related factors. Accuracy is a measure for determining the overall correctness of a model. It indicates the number of correctly predicted instances in relation to the total number of instances. The higher the figure representing accuracy score, the more correct predictions are made by the model.

Accuracy is a measure for determining the overall correctness of a model. It indicates the number of correctly predicted instances in relation to the total number of instances. The higher the figure representing accuracy score, the more correct predictions are made by the model.

$$Accuracy = \frac{(True\ Positives + True\ Negatives)}{Total\ Instances}$$

While accuracy is widely used, it may not fully reflect the performance when the class distribution is imbalanced, therefore additional metrics are considered.

Precision is how many of the predicted positive instances are actually positive. That means reducing the false positive counts. The higher the precision, the more probably the prediction for a positive result of the model is correct.

$$Precision = \frac{(True\ Positives)}{(True\ Positives + False\ Positives)}$$

Precision is particularly useful when false positives (incorrect symbol predictions) are need to be minimized.

Recall or sensitivity measures how many of the actual positives the model predicted as positive. It aims to minimize false negatives. High recall means that the model captures almost all positive examples in the dataset.

$$Recall = \frac{(True\ Positives)}{(True\ Positives + False\ Negatives)}$$

A high recall means the model correctly identifies most of the actual instances of a symbol. It is important when the cost of missing a symbol is high.

F1-score is a measure that combines both precision and recall and is thus known as a harmonic mean of the two. It is especially useful when the precision and recall are to be balanced and only a single value is required to carry both notions. The F1 score gets high when the values of both precision and recall are high

$$F1\ score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

F1-score provides a balanced metric when both false positives and false negatives are important.

ROC-AUC (Receiver Operating Characteristic - Area Under Curve) indicates the distinguishing ability of a model for classes. Plotting false positive rates (1-specificity) against true positive rates (sensitivity) for various thresholds is done. The model's capacity to accurately categorize positive and negative examples is measured by the area under this curve (AUC). The higher value of AUC indicates better performance. The AUC varies from 0 to 1 where the higher number represents a perfect classifier and 0.5 stands for the worst and no discrimination power.

$$ROC - AUC = Area\ under\ the\ curve\ of\ the\ ROC\ plot$$

Precision-Recall Curve comes into picture mainly when we have imbalanced class distributions. It plots precision on y-axis against recall on x-axis for various thresholds showing the trade-off between these two. In fact, compared to the ROC curve, if we talk especially about the imbalanced datasets, the precision-recall curve tends to give more insight as it emphasizes the positive class (minority).

$$Precision - Recall\ Curve = Plot\ of\ Precision\ vs\ Recall$$

Finally, we utilized a **confusion matrix** to inspect the types of errors made by the model. The confusion matrix contains the counts of true positives, false positives, true negatives, and false negatives for each class. The confusion matrix together allows us to visualize certain misclassifications and which specific symbols were compared with the model and could also be improved on future iterations along with their performance, respectively at the class level.
True Positives (TP): The number of instances where the model makes a correct prediction of the positive class.
False Positives (FP): The number of instances where the model makes an incorrect prediction of the positive class.
True Negatives (TN): The number of instances where the model accurately identifies the negative class.
False Negatives (FN): The number of instances where the model inaccurately identifies the negative class.

4.2 Evaluation Results

The model's performance was evaluated with a number of hyperparameters and performance metrics, providing a basis for how well this model generalizes to new data. Below is a comprehensive description of the model configurations, the obtained metrics, and the significance of these metrics.

4.2.1 Model Hyperparameters and Configuration:

We chose a set of hyperparameters that would optimize both the efficiency of training and the generalization of the model after training. Below is a brief description of the hyperparameters used [15].
Input Image Size: The size of the input images was resized into the size of 28×28×1 to comply with the dataset variables. This action assured that the model would learn standardized inputs that were suitable for training but did not include any irrelevant features in the images, simplifying the learning process.
Learning Rate: A learning rate of 0.001 was chosen for the Adam optimizer, known for accelerating convergence and towards a given point in a more efficient manner by allowing the particular learning rate of the model to adjust as it learnt.
Optimizer: The Adam optimizer was chosen as it can also handle sparse gradients and adjust the learning rate while training while implementing varying learning rates from batch-to-batch, which is useful when training a complex model as in this research project.
Epochs: The model accepted the maximum of 3 epochs before implementing Early Stopping. This was sufficient for the model to process the data and learn the relationships without wasting too much time.
Early Stopping Patience: Early stopping was implemented with a patience of 3 epochs which means if the validation loss improved, we'd extend training, and if it didn't, we'd stop training after 3 epochs. This allowed us to prevent overfitting and wasted computational time/resources by stopping training early once it reached an acceptable area.
Loss Function: For the loss function, we used categorical crossentropy, which is appropriate for multi-class classification problems. Categorical crossentropy attempts to reduce the distance between the predicted and actual class probabilities.
Drop-out Rate: We used a drop-out rate of 0.3 as a regularization method and to reduce overfitting. Dropout regularization works because in each epoch during training, dropout will randomly set 30% of the neurons to zero. This process provides for a better generalized understanding of the model on new data.
Activation Function: The hidden layers utilized the ReLU (Rectified Linear Unit) activation function, which has gained traction because it can avoid vanishing gradients and accelerate training speed.

Output Activation: The output layer implemented softmax to translate the raw output from the model into probabilities that summed to one, which is appropriate for multi-class classification problems.

Number of Classes: The model was designed to classify 23 different symbols, comprised of digits and mathematical operators, which defined it as multi-class classification.

Table 2. Hyperparameter Configuration (for Proposed CNN Model)

Parameter	Description	Value
Input Image Size	Shape of input images	28×28×1
Learning Rate	Initial learning rate for the optimizer	0.001
Optimizer	Optimizer used for model convergence	Adam
Epochs	Maximum number of training iterations	8
Early Stopping Patience	Number of epochs to wait for validation loss improvement before stopping	3
Loss Function	Loss function for multi-class classification	Categorical Crossentropy
Dropout Rate	Dropout rate for regularization	0.3
Activation Function	Activation used in intermediate layers	ReLU
Output Activation	Activation used in final classification layer	Softmax
Number of Classes	Total distinct gesture classes (digits + operators)	23

4.2.2 Test Dataset Performance:

The model has 99.28% accuracy on a held-out test dataset, indicating that it is strongly capable of precisely and accurately classifying hand gestures. Such a level of accuracy is valuable in real-time tasks that require speed and accuracy in the predictive process. Furthermore, with a precision of 0.9893, nearly 99% of the gestures predicted by the model will be correct, minimizing the potential for false positives. Using the best metric of recall (0.9894), the model will recognize nearly 99% of all true gestures it should recognize, minimizing the potential for missing important symbols. The F1 score, or balance of precision and recall, is 0.9894, indicating that the model offers an appropriate trade-off of the two metrics, which is important to both high-level accuracy and form recognition. The ROC-AUC score of 1.0000 indicates perfect separability of gesture classes and confirms that the model will recognize distinct gestures with high confidence. Finally, the Precision-Recall Curve score of 0.9991 confirms superior model performance since the model will maintain high precision while completely capturing relevant instances. Together, the metrics demonstrate that the model is robust, reliable, and effective for gesture recognition tasks.

Table 3. Model Performance Metrics for Test Dataset

Metric	Results with Adam Optimizer
Test Accuracy	0.9928
Precision	0.9893
Recall	0.9894
F1 Score	0.9894
ROC-AUC	1.0000
PR Curve	0.9991

4.2.3 Additional Evaluation Metrics:

Model Accuracy vs. Epochs The accuracy curve reflects how the model performance improves with each epoch. Accuracy starts from a low accuracy, and as it gets trained with the provided training data, the accuracy gradually improves. It is ideally a sharp increase in accuracy with stabilization, indicating that the model reached convergence quickly with good ideal conditions, preventing it from overfitting the training data.

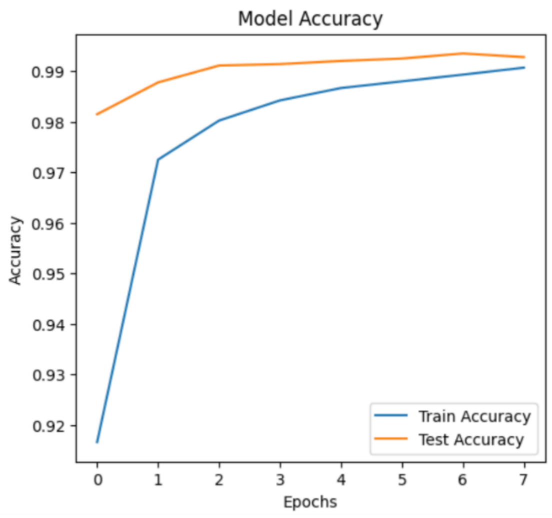

Figure 5: Accuracy Vs Epochs Graph

Model Loss (Loss vs. Epochs): The loss curve reflects the improvement on the loss function during training. With good ideal conditions, the epoch loss curve shows a steady decrease. The steep and smooth decrease ensures that the model is minimising the error, and shows that there are no unexpected changes resulting in the model not learning properly.

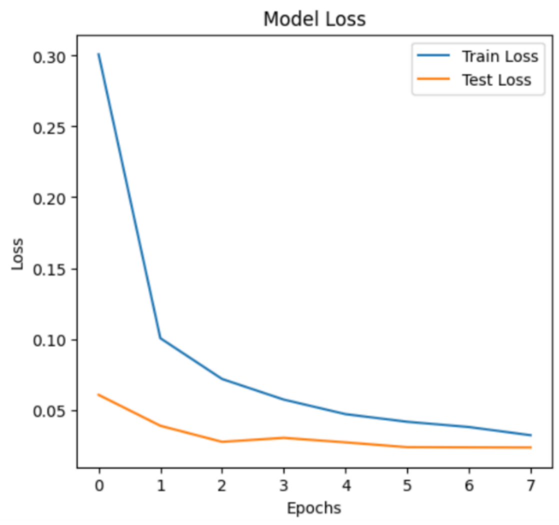

Figure 6: Loss Vs Epochs Graph

ROC-AUC Curve: The ROC curve is a visual representation of the model's ability to discriminate between classes with a curve indicating superior performance, and the area under the curve (AUC) reporting the level of performance. In this study, an AUC score of 1.0000 means that the model could discriminate between classes perfectly.

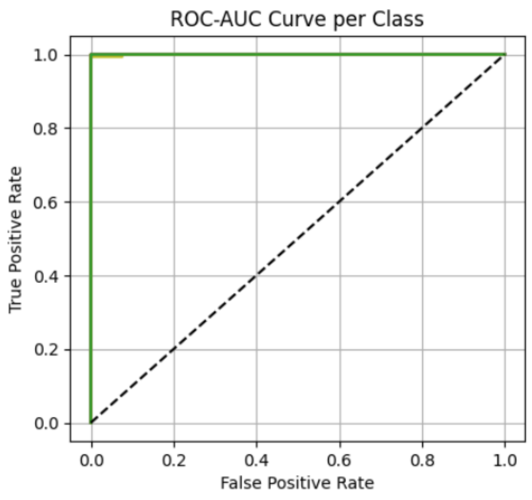

Figure 7: ROC Curve

Precision-Recall Curve: The PR curve represents the trade-off between precision and recall at different thresholds. A larger area corresponds to a model with good balance between the two, getting the relevant instances but minimizing the false positives.

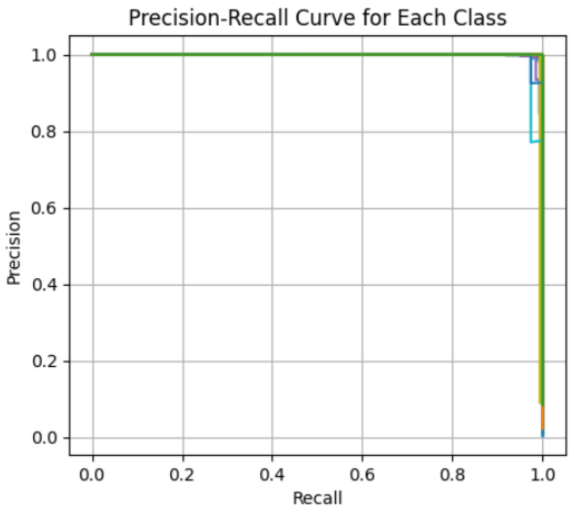

Figure 8: Precision-Recall Curve

Confusion Matrix: The confusion matrix provides another visual representation of classification performance across all 23 classes that allows further detail into how many symbols the model has truly classified correctly, as well as into the number of mis-classifications and can provide additional insights about where the model may have weaknesses in performance.

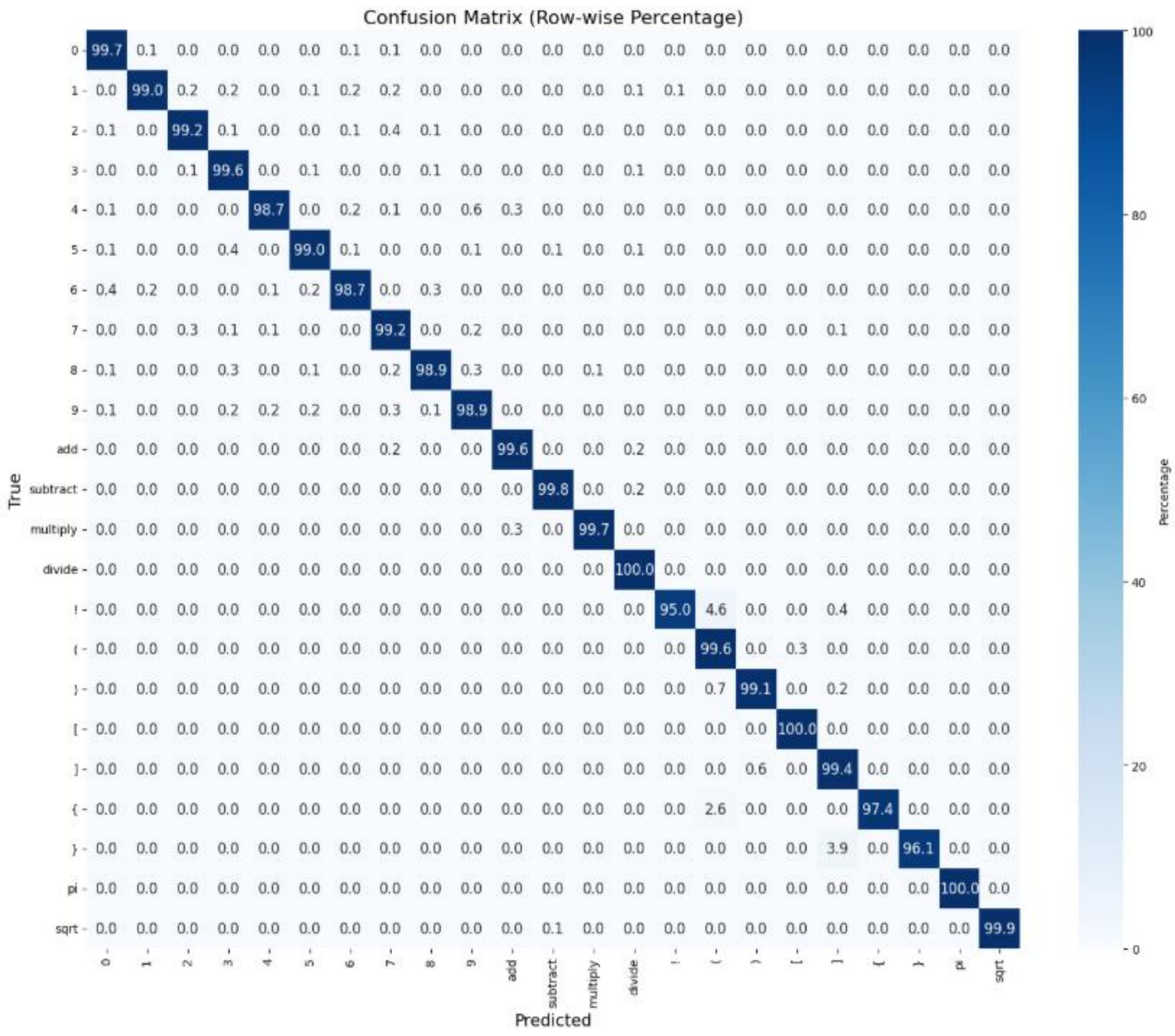

Figure 9: Confusion Matrix (Percentages)

The results demonstrate that the model, trained with the Adam optimizer, achieved excellent performance across multiple metrics, including test accuracy, precision, recall, F1-score, ROC-AUC, and precision-recall curve. The high-test accuracy reflects the model's strong ability to generalize to unseen data, while the high precision and recall indicate that the model is both reliable in its predictions and effective at capturing relevant symbols. The F1 score and ROC-AUC score further confirm the model's balanced performance, with the PR curve area suggesting that the model is well-tuned to handle any potential class imbalance. These results indicate that the model is well-suited for real-time air-writing applications where accurate gesture recognition is essential.

4.3 Result Comparison with Existing Methods

According to Nazemi et al. [5], an offline mathematical symbol character recognition system was developed based on deep learning. The study carried out segmentation via SLIC and performed classification with a modified version of the LeNet architecture as well as SqueezeNet. The system was tested on 101 classes of mathematical symbols with accuracies of up to 90% using the transfer learning approach with a pre-trained SqueezeNet model, demonstrating that this method can be effectively applied to large and complex sets of symbols.

An air-writing recognition system developed by Mukherjee et al. [2] for the webcam video input works on solving fingertip detection, tracking, and gesture termination problems. The researchers used hand detection through Faster R-CNN, a new distance-weighted curvature entropy for fingertip tracking, and a velocity-based criterion for termination. The system evaluation worked on English

characters and digits in CNN, which resulted in a mean character recognition accuracy of 96.11% for 26 + 10 classes (letters and digits).
A framework that equips video-enabled air writing was proposed by Roy et al. [3], utilizing a webcam and colour marker for unistroke recognition. Tracking the fingertip trajectory, a CNN model was used. The model was trained on MNIST and presented with air-written samples. This system achieved 97.7% accuracy on person-independent evaluations for English numerals (10 classes), demonstrating strong generalization in real-world conditions.

Table 4. Comparing different approaches

S.No	Reference	Method	Classes	Accuracy
1	Nazemi et al. [5]	SqueezeNet	101	90.00%
2	Mukherjee et al. [2]	Faster R-CNN + CNN	36 (A–Z, 0–9)	96.11%
3	Roy et al. [3]	CNN	10 (digits)	97.70%
4	Proposed Method	Custom CNN	23	99.28%

5 Conclusion

This study brings strong to excellent findings in developing a real-time hand gesture recognition model for computing mathematical equations, using deep learning-based approaches to recognize hand gestures to identify symbols through hand gesture to contextually compute mathematical expressions. The model was developed and trained on a custom dataset of digits and operators, resulting in an excellent performance of the model, which yielded a test accuracy of 99.28%. The model has displayed its potential to be implemented in real-time settings with high accuracy. The combination of precision, recall, and F1 score all being significant numbers reflect the level of accuracy and completeness of the model by accurately detecting the vast majority of the different gestures for mathematical symbols, while generating little in the way of false positives. Even the ROC-AUC score of 1 indicates excellent discrimination of the hand gesture performances of gestures from one another as well. As the study suggests, this study's proposed system could be used in real world settings such as a interactive math calculator, assistive technology, or any computing device enacting a user interface initiated from gestures made by an end user wanting to interactively control the system. Future work could investigate how much more complex of a set of gestures could potentially be added to the dataset, or maybe with gesture-based computing as with gestures-based model, determine how well the system could be made more responsive in real time at the end user interaction instead of casual hand gestures due to cued testing. Overall, the model also shows promise towards further understanding more advanced based systems based around deep-learning based hand gesture recognition systems towards greater opportunities within human technology-based interactions.

References

[1] M. Jabde, C. H. Patil, A. D. Vibhute, and S. Mali, "An Air-Written Real-Time Multilingual Numeral String Recognition System Using Deep Convolutional Neural Networks," *Journal of Computer Science*, vol. 20, no. 12, pp. 1712–1722, 2024, doi: 10.3844/jcssp.2024.1712.1722.

[2] S. Mukherjee, A. Ahmed, D. P. Dogra, S. Kar, and P. P. Roy, "Fingertip Detection and Tracking for Recognition of Air-Writing in Videos," Sep. 2018, doi: 10.1016/j.eswa.2019.06.034.

[3] P. Roy, S. Ghosh, and U. Pal, "A CNN Based Framework for Unistroke Numeral Recognition in Air-Writing," Mar. 2023, [Online]. Available: http://arxiv.org/abs/2303.07989

[4] M. Wu, K. Huang, Y. Cai, S. Hu, Y. Zhao, and W. Wang, “Finger in Camera Speaks Everything: Unconstrained Air-Writing for Real-World,” Dec. 2024, [Online]. Available: http://arxiv.org/abs/2412.19537

[5] A. Nazemi, N. Tavakolian, D. Fitzpatrick, C. a Fernando, and C. Y. Suen, “Offline handwritten mathematical symbol recognition utilising deep learning,” Oct. 2019, [Online]. Available: http://arxiv.org/abs/1910.07395

[6] A. Suryaperdana Agoes, “Applying Hand Gesture Recognition for User Guide Application Using MediaPipe,” 2021.

[7] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *J Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00444-8.

[8] A. Fred Agarap and A. M. Fred Agarap, “Deep Learning using Rectified Linear Units (ReLU)”, doi: 10.48550/arXiv.1803.08375.

[9] M. Franke and J. Degen, “The softmax function: Properties, motivation, and interpretation *.”

[10] “MNIST Dataset,” Keras. [Online]. Available: https://keras.io/api/datasets/mnist/

[11] “Mathematics symbols data,” Kaggle. [Online]. Available: https://www.kaggle.com/datasets/amitamola/mathematics-symbols-data

[12] “Math Operators and Digits,” Kaggle. [Online]. Available: https://www.kaggle.com/datasets/patrikmesec/math-operators-and-digits

[13] A. Géron, “Hands-On Machine Learning with Scikit-Learn and TensorFlow.”

[14] J. Davis and M. Goadrich, “The relationship between precision-recall and ROC curves,” in *ACM International Conference Proceeding Series*, 2006, pp. 233–240. doi: 10.1145/1143844.1143874.

[15] R. Andonie, “Hyperparameter optimization in learning systems,” *Journal of Membrane Computing*, vol. 1, no. 4, pp. 279–291, Dec. 2019, doi: 10.1007/s41965-019-00023-0.