

# **SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT, NAGPUR.**

***Project report for***

***For Course: - Deep Learning-I VI sem B.Tech (ECS)***

***Academic Year (2024-25)***

***Department of***

## **ELECTRONICS AND COMPUTER SCIENCE**

***by***

***Rishabhdev Makode - A1 / 13***

***Sarvesh Shirulkar - A1 / 15***

**Guide:**

**Prof. Vikas Gupta**

**Mini Project Report**

Handwritten Equation Solver using CNN

**Introduction**

In this project, we explore the development and implementation of a Convolutional Neural Network (CNN) for the task of image classification. CNNs are a class of deep learning models that have proven remarkably effective in recognizing patterns within image data, making them particularly well-suited for tasks such as handwritten digit recognition, facial detection, and general object classification.

The goal of this mini project is to design a CNN that can take in grayscale images of size 32x32 pixels and accurately classify them into one of 14 possible categories.

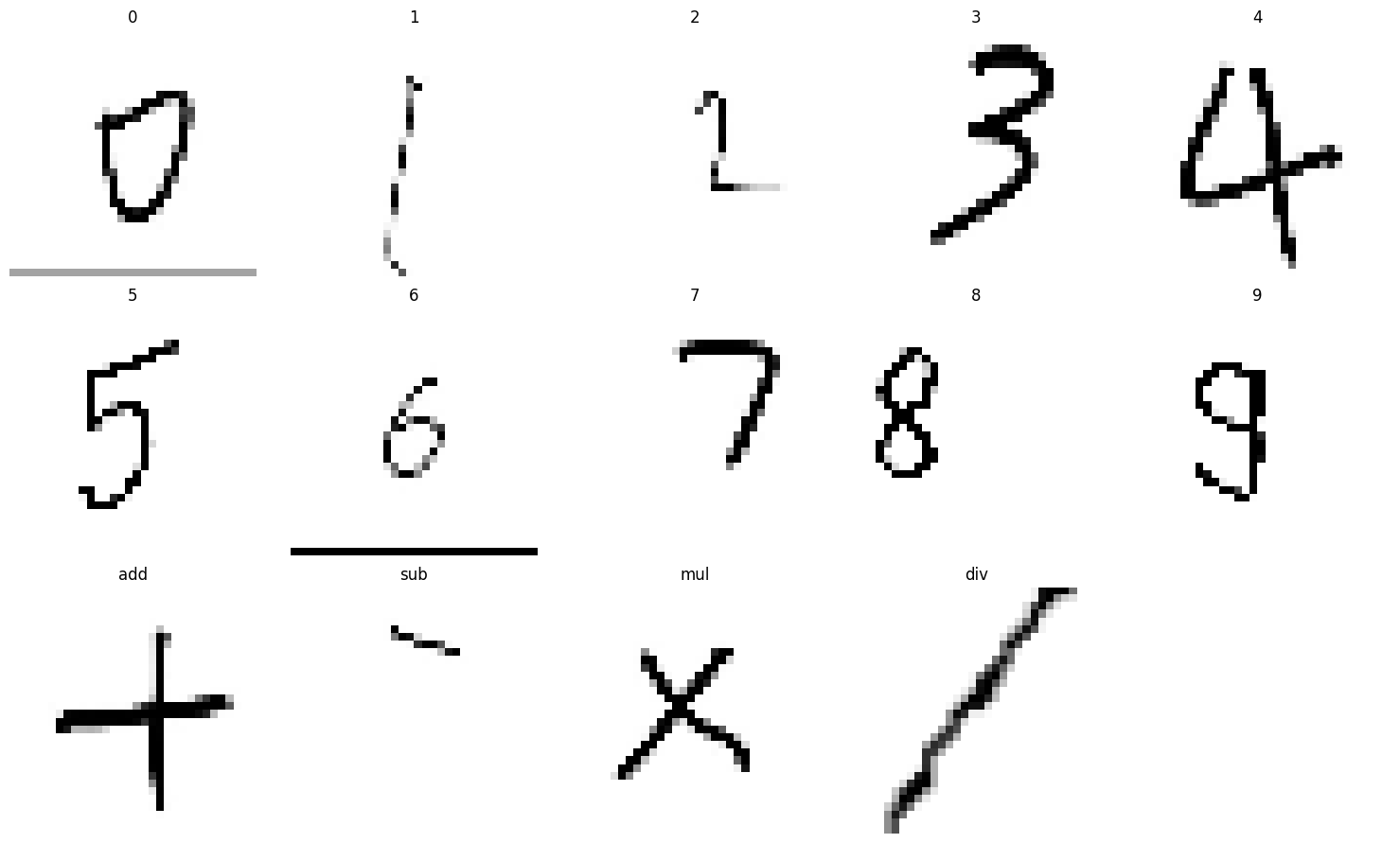
**Problem Statement:**

The goal of this project is to build a system that can recognize and solve simple handwritten mathematical equations using a Convolutional Neural Network (CNN). The model classifies individual handwritten symbols and digits, reconstructs the equation, and outputs the solution.

**Dataset:**

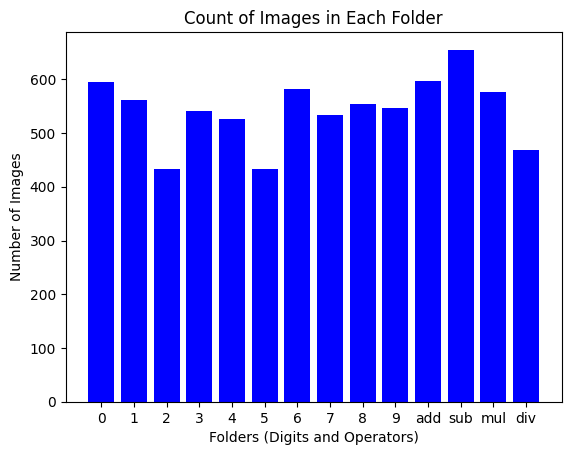
The dataset used in this project was sourced from an online repository and downloaded using the link: <https://cainvas-static.s3.amazonaws.com/media/user_data/Yuvnish17/data.zip>

This dataset contains total 7600 images.



**Data Preprocessing**

The dataset contains 7,600 images across 14 classes: 10 digits (0–9) and 4 mathematical operation symbols. The distribution of each class can be seen in the graph below.



On average, each class contains around 500 images.

**Methodology:**

**Step 1:**

We read every image in grayscale mode and resized each to a uniform size of 32x32 pixels. These images were then flattened and normalized by scaling the pixel values to a range of 0 to 1, which helps the neural network converge faster and more effectively during training. Each image was tagged with its corresponding label (inferred from its folder name), and the dataset was shuffled to ensure that the training and testing sets were diverse and not biased toward any specific class order. This processed data was then stored in a CSV file for convenience and future reproducibility.

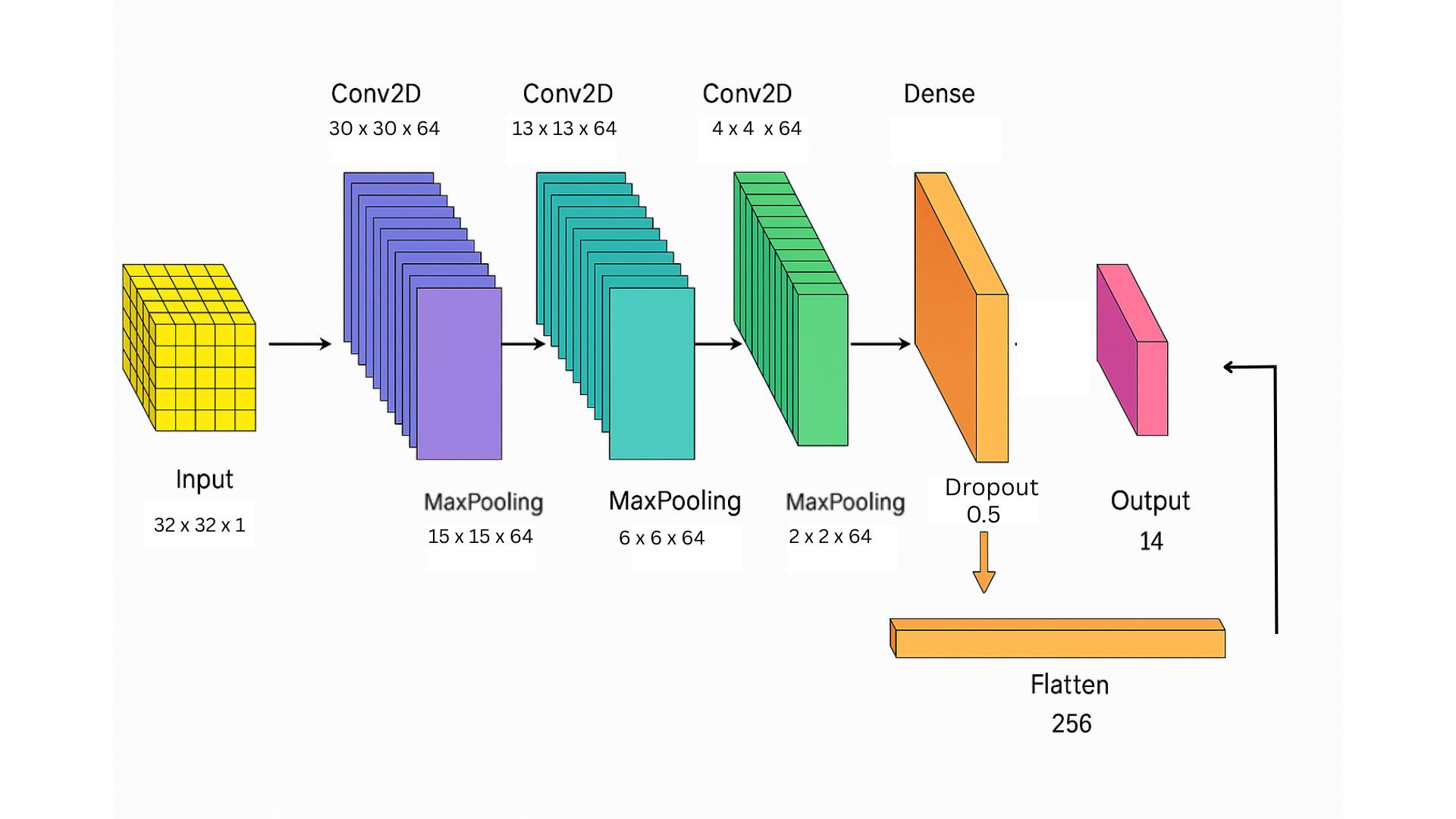
**Step 2:**

Next, we divided the dataset into training and testing sets using an 80-20 split. This allowed us to evaluate the model’s performance on unseen data after training.

In this project, we chose not to use a separate validation set and instead focused on training and testing splits only. This decision was primarily based on the manageable size of the dataset and the simplicity of the classification task at hand. With a total of 7,600 images, we opted for an 80-20 train-test split, ensuring that the model had sufficient data to learn from while still being evaluated on a reasonable portion of unseen data

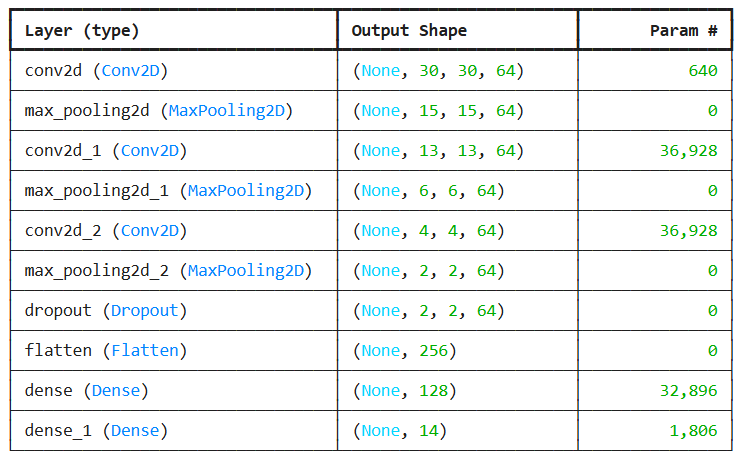
**Step 3:**

The backbone of this system is a Convolutional Neural Network (CNN) that was built using the Sequential API from Keras. The CNN was carefully designed to learn visual features from the low-resolution input images. The model starts with a convolutional layer that applies 64 filters of size 3x3 to extract local spatial features from the input image. This is followed by a max-pooling layer that reduces the spatial dimensions, helping the model focus on the most prominent features while also improving computational efficiency. This convolution-pooling combination is repeated three times to progressively extract higher-level and more abstract features from the image.



**Step 4:**

After the third set of convolution and pooling operations, a dropout layer is added. This layer randomly disables 50% of the neurons during training, which helps prevent overfitting and ensures that the model generalizes well to new data. The output from these layers is then flattened into a 1D vector, which is passed through a dense layer with 128 neurons and a ReLU activation function. Finally, the output layer uses a softmax activation to classify the input image into one of the 14 possible categories.



**Step 5:**

Once the model was trained and saved, we built a prediction pipeline to test it on new handwritten equations. The input image is first converted to grayscale and then thresholded to create a binary image, making it easier to detect individual characters. Contours are then identified to locate each symbol in the image. These contours are sorted from left to right, mimicking the natural reading order of mathematical expressions.

**Step 6:**

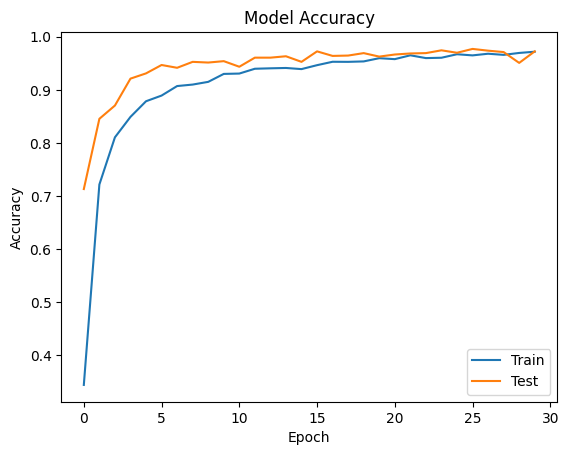
For each detected character, a bounding box is drawn, and the region of interest is cropped, resized to 32x32 pixels, and normalized just like the training data. These preprocessed character images are then passed through the trained CNN model to obtain predictions. Each predicted class label is mapped back to its corresponding digit or operator symbol.

The recognized symbols are then assembled in order to reconstruct the original handwritten equation. This reconstructed string is displayed on the image along with bounding boxes and labels for visual confirmation. Finally, the expression is evaluated using Python’s built-in **eval()** function, and the result is displayed to the user.

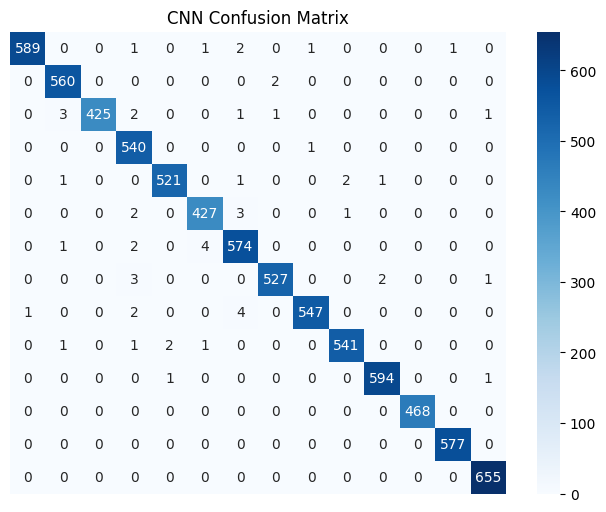
**Results:**

The performance of the CNN-based handwritten equation solver was evaluated using several visualization tools, including prediction samples, a confusion matrix, and accuracy plots.

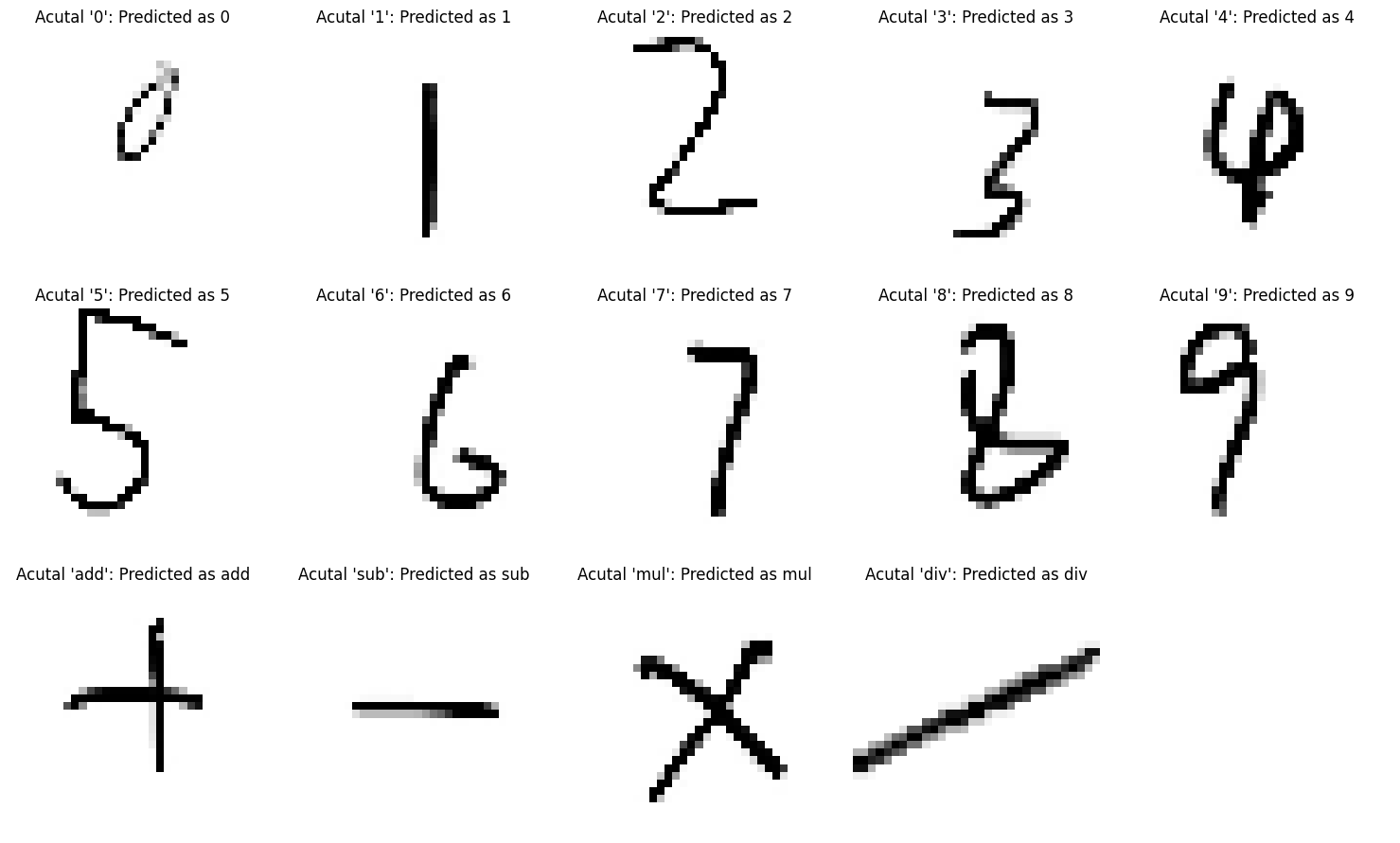
The accuracy graph shows both training and testing accuracy over 30 epochs. The model exhibits a rapid learning phase in the initial epochs, with the test accuracy quickly reaching above 90%. Over time, both training and test accuracies converge towards nearly 98–99%, indicating strong generalization without overfitting. The smooth convergence of both curves also reflects the model’s stable training dynamics.



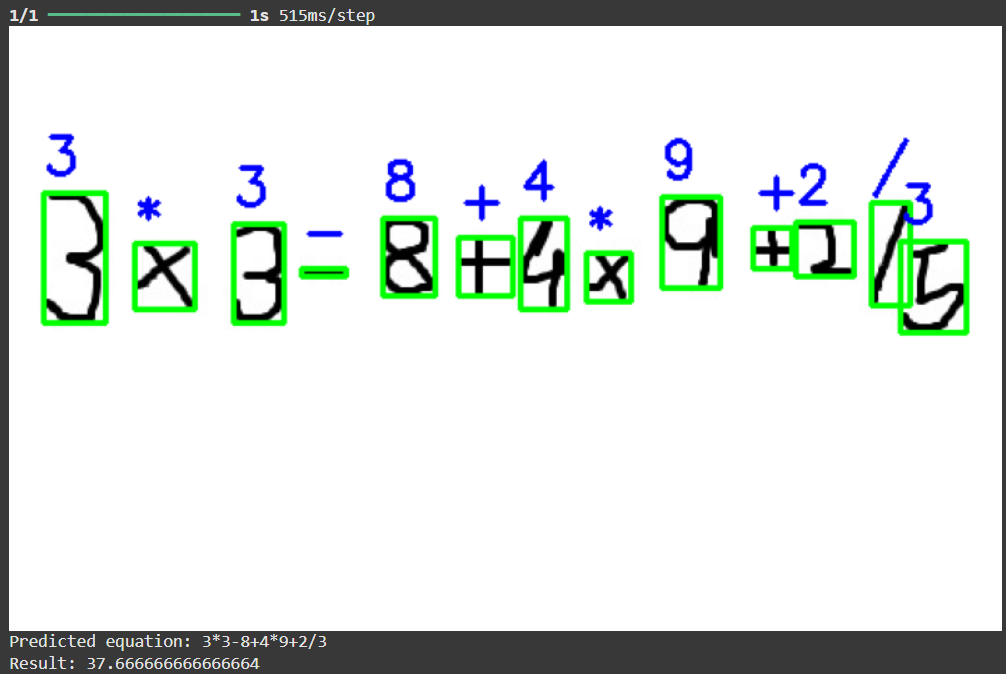
The confusion matrix provides detailed insight into the model’s classification performance across all classes. The diagonal dominance indicates that the majority of the predictions are correct. Misclassifications are minimal and appear as small off-diagonal values, confirming the model’s robustness.



The first image shows a grid of sample predictions made by the trained model on handwritten digits and arithmetic operators. Each sample is annotated with its actual label and the predicted label. The model correctly identifies all shown samples, including digits from 0 to 9 and the symbols for addition (+), subtraction (−), multiplication (×), and division (/).



**Final Output:**



**Conclusion:**

In conclusion, the handwritten equation solver utilizing deep learning techniques, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), has demonstrated promising accuracy levels. The CNN-based approach achieved an impressive accuracy of 97.57%. The CNN model excelled in accurately recognizing and classifying individual handwritten characters within the equations. Its ability to effectively capture spatial hierarchies and local patterns in the input images contributed to its high accuracy. With its deep layers and convolutional operations, the CNN was capable of learning complex features directly from the pixel values, enabling robust classification of handwritten characters.

**References:**

1. **P. P. Mali, S. R. Hase, F. K. Kolhe, V. G. Jaju and B. Sonare, "Handwritten Equations Solver Using Convolution Neural Network," *2023 4th International Conference for Emerging Technology (INCET)*, Belgaum, India, 2023, pp. 1-5, doi: 10.1109/INCET57972.2023.10170383.**

**2. Shivangi, R. K. Sah, Shreeyam, G. Florance and M. Nirmala, "CNNCalc – An Implementation of a Handwritten Mathematical Equation Solver," *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2023, pp. 638-645, doi: 10.1109/ICICCS56967.2023.10142278.**

**Contributions:**

Rishabhdev: Model Developer, Researcher.

Sarvesh : Documentation, Helped in Model Development.