Precision Redefined: CNN and Bounding Boxes in Mathematical Equation Solving

Edward Wu[#1], Shivashankar Hanagondahalli Chandrashekaraiah[*2], Wasey Mulla[#3], Sreeman Tummala[#4]

[#]*Department of Computer Science, The University of Texas at Dallas*
800 W Campbell Rd, Richardson, TX 75080 USA

[1]eww170000@utdallas.edu

[2]sxh220091@utdallas.edu

[3]wam180001@utdallas.edu

[4]sxt190060@utdallas.edu

Abstract— **This paper explores the integration of Optical Character Recognition (OCR) in solving mathematical equations, aiming to bridge the gap between physical and digital mathematics. The study assesses the challenges and advancements in OCR algorithms for accurately recognizing and interpreting mathematical symbols, variables, and operators. By reviewing existing methodologies and discussing the implications in educational settings, the research emphasizes the potential of OCR to facilitate the digitization of handwritten mathematical content. Through case studies and experimental results, the paper advocates for continued OCR development tailored to mathematical notation, offering a seamless transition from traditional pen-and-paper practices to enhanced digital workflows in education, research, and industry.**

I. Introduction and background work

In the dynamic landscape of Optical Character Recognition (OCR), the synthesis of Convolutional Neural Networks (CNN) and bounding box methodologies represents a paradigm shift, poised to redefine mathematical equation solving. This paper delves into the synergy of OCR and CNN-based bounding boxes, introducing a refined framework tailored for the intricate demands of recognizing mathematical notation. Leveraging CNNs for feature extraction and bounding boxes for precise symbol localization, our research aims to elevate the accuracy and resilience of OCR in equation solving. This introduction sets the stage for a nuanced exploration of the interplay between OCR, CNN, and bounding boxes, offering insights into their collective potential to advance the digitization of mathematical content. Through a judicious examination of methodologies and experimental validations, this paper contributes to the discourse on integrating cutting-edge technologies into mathematical equations, with broad implications for education, research, and practical applications.

II. Approach

A. Preprocessing Data

CNNs have a more intensive requirement for preprocessing as it is a more complex form of data as you are dealing with images instead of regular data. However, even with this there are good libraries to preprocess the images, such as the pillow library. After this the next step in preprocessing goes to bounding boxes. Bounding boxes serve as fundamental tools in the analysis of images containing mathematical symbols and numbers, acting as rectangular frames that encapsulate specific regions of interest identified through contour detection. Once these bounding boxes are generated, they effectively label and isolate individual symbols or numbers within the image. To enhance visibility, the system enlarges these bounding boxes, creating visually distinctive representations depicted as green rectangles in the visual output, offering clear demarcation of recognized elements.

The importance of these bounding boxes extends to their role in training a Convolutional Neural Network (CNN), functioning as labeled regions of interest (ROIs) used as training samples. This enables the network to learn and comprehend the unique features of mathematical symbols and numbers, forming a crucial component in solving basic math questions based on image inputs.

Implementing bounding boxes involved several key steps in the image processing pipeline. Initially, an image designated for processing is loaded and goes through an essential transformation such as conversion to grayscale to simplify subsequent operations focused on intensity variations for symbol detection. Otsu's thresholding method was then applied to convert the grayscale image into a binary representation, separating foreground elements from the background and aiding contour detection.

Contour detection, a pivotal phase, identifies specific shapes or regions within the binary image, outlining distinct areas representing symbols or numbers. These contours facilitated the creation of bounding boxes that encapsulated the identified regions. Ensuring optimal coverage and visibility, these bounding boxes were expanded systematically around each symbol or number within the image, offering a clear visual representation for subsequent analysis.

The expansion of the bounding boxes was fine-tuned via the 'expansion_value' parameter, allowing controlled enlargement without compromising the integrity of enclosed elements. The delineated bounding boxes, presented as green rectangles overlaid on the input image, served as a visual indicator of the system's identification of mathematical elements within the image.

While the bounding box implementation effectively recognized most symbols and numbers, it encountered a minor issue with identifying the "=" sign accurately. However, this limitation was resolved during the CNN training phase, where the network learned to correctly recognize and comprehend the "=" symbol, contributing to the system's improved accuracy in comprehending mathematical elements in image-based inputs.

B. Convolution Neural Network

In our Optical Character Recognition (OCR) task, we leverage a comprehensive set of techniques, including a fully connected Deep Neural Network with Rectified Linear Unit (RELU) and Softmax activation functions, as well as a Cross Entropy loss function. Our approach employs Convolutional Neural Networks (CNNs) with specific focus on 10 numerical digits and fundamental arithmetic signs (addition, subtraction, multiplication, division). This tailored feature set not only enhances character recognition but also equips our OCR system with the adaptability and efficiency required for recognizing characters and symbols within mathematical expressions.

The versatility of our OCR system extends to parsing and interpreting mathematical expressions, with the network's adeptness at handling variations in font, size, and orientation. By capitalizing on the hierarchical feature extraction and classification capabilities of CNNs, our system accurately processes and interprets equations. This adaptability positions our OCR system as a potential tool for automating the extraction and computation of mathematical expressions, essentially functioning as a calculator. The integration of CNNs in OCR showcases the practical utility and versatility of our approach, offering broader applications in mathematical problem-solving beyond conventional character recognition.

III. Results and Analysis

In our study, we utilized the "Handwritten math symbol dataset" for training and evaluating our Convolutional Neural Network (CNN) model. It is important to note that the class distribution within the dataset is not balanced, prompting the adoption of data augmentation techniques to address this imbalance and improve model generalization.

The dataset comprises 82 distinct classes, encompassing a diverse range of mathematical symbols and characters. Notably, the main classes include digits (0 to 9), characters (both lowercase and some uppercase letters), and fundamental math operators such as addition (+), subtraction (-), division (/), multiplication (*), and the equality sign (=). Additionally, the dataset incorporates basic Greek alphabet symbols, including alpha, beta, gamma, mu, sigma, phi, and theta, along with English alphanumeric symbols. It encompasses a comprehensive set of math operators, set operators, and basic predefined math functions such as log, limits, cos, sin, and tan.

Given the grayscale nature of the image dataset, each image is represented by 2025 pixels, serving as the features for our CNN model. The recognition and classification of this diverse set of symbols and characters pose a unique challenge, and our exploration involves training the model to effectively differentiate among the 82 classes while addressing the class imbalance through data augmentation techniques shown in Figure 1.

Furthermore, the input shapes for our CNN model are as follows:
- X_train shape: (62908, 1, 45, 45)
- X_valid shape: (6990, 1, 45, 45)
- X_test shape: (17475, 1, 45, 45)

These shapes represent the number of images, number of image channels, image height, and image width, respectively. The results obtained from this study will shed light on the effectiveness of our CNN approach in handling complex mathematical symbol recognition tasks within this diverse dataset.

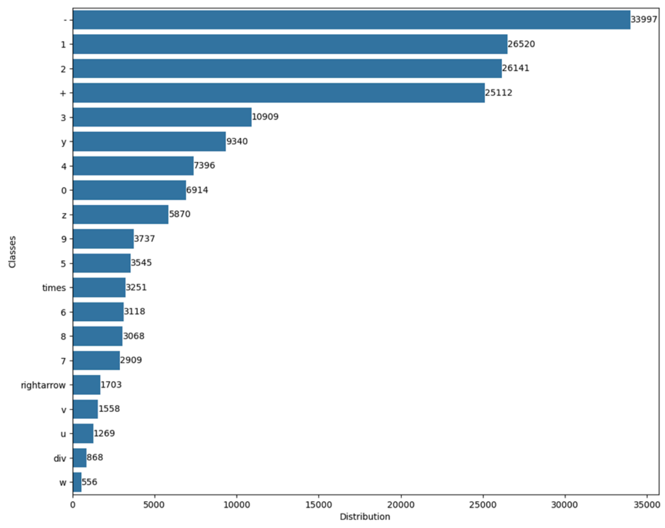


Fig. 1 An image of the class distribution.

We've generated four models with different architecture.

A. Model 1 (Flatten1, Dense1, Softmax, Cross Entropy)

In our first model, we've tried using ReLu in every layer, Softmax as the last activation layer. Here's a table and training accuracy indicated in Figure 2 and Figure 3.

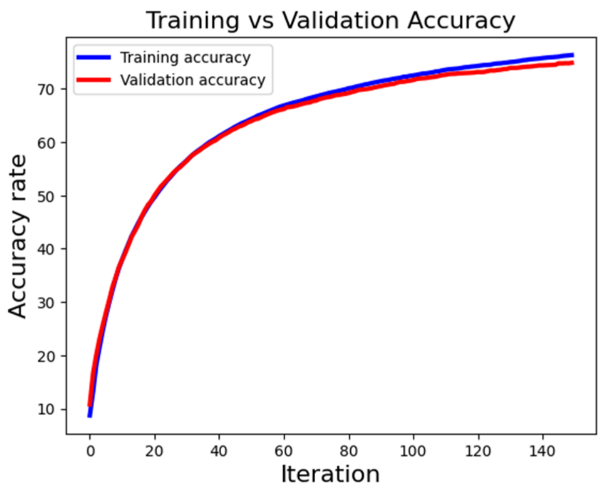


Fig. 2 A graph resembling the training accuracy and validation accuracy of Model 1.

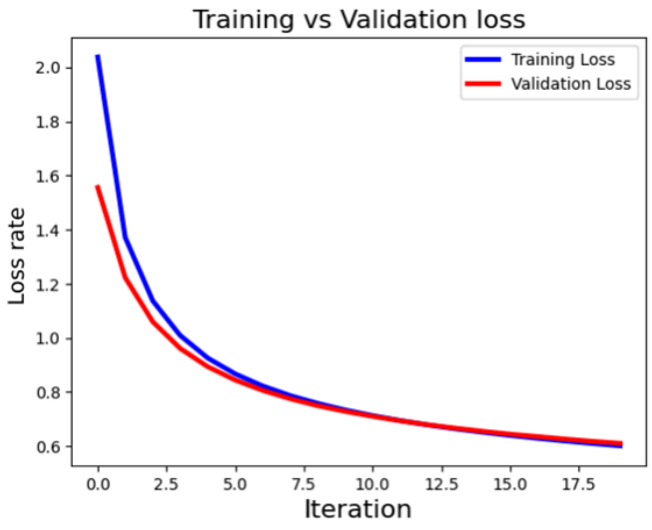


Fig. 3 A graph resembling the Training vs Validation Loss.

The training and validation metrics for Model 1.1 and Model 1.2 showcase the impact of different factors on model convergence. For Model 1.1, trained over 150 epochs with a batch size of 16 and a learning rate of 0.01, the final results reveal a train accuracy of 76.34% and validation accuracy of 74.89%, with corresponding losses of 2.14 and 2.36, respectively. The test accuracy and loss stand at 75.07% and 2.34.

In contrast, Model 1.2, initialized with Glorot activation, demonstrates faster convergence over 20 epochs with the same batch size and learning rate. The concluding metrics include a train accuracy of 84.88% and validation accuracy of 85.01%, accompanied by train and validation losses of 0.60 and 0.61. The test accuracy and loss for Model 1.2 are 84.02% and 0.62, respectively.

Comparing the two models, it becomes evident that the choice of weight initialization significantly influences convergence. While Model 1.1 exhibits a slower convergence rate, plateauing at 74% accuracy after 100 epochs, Model 1.2, with Glorot initialization, achieves a remarkable 84% validation and testing accuracy within just 20 epochs. This emphasizes the efficacy of Glorot/Xavier initialization in enhancing model performance, especially evident in the context of a single-layer neural network like Model 1.2.

B. Model 2 (Dense1, Relu, Dense2, Softmax, Cross Entropy)

In our second model, we decided to add a second dense layer of ReLu activation as well as Softmax. Here's our results in Figure 4 and Figure 5.

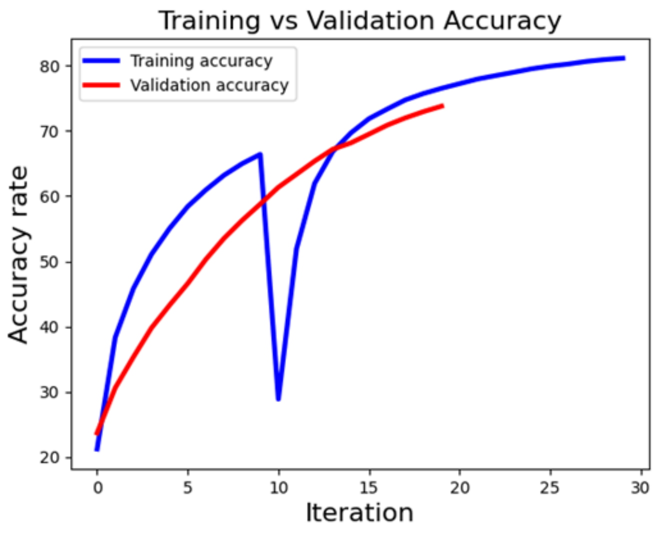


Fig.4 A graph resembling the training accuracy and validation accuracy of Model 2.

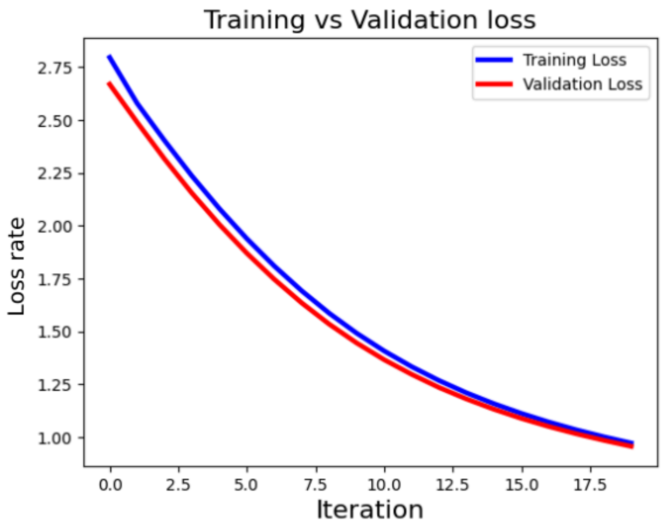


Fig.5 A graph resembling the Training vs Validation Loss of Model 2.

The model (Model 2.1) encountered a local minima issue, impeding its progress, but the problem was mitigated when Glorot initialization was applied to a similar model (Model 2.2). Interestingly, despite both models sharing the same hyperparameters, Model 1.2, featuring a single dense layer, exhibited superior metrics compared to Model 2.2, which incorporated two dense layers. This suggests that the model architecture, specifically the number of dense layers, played a crucial role in determining the performance, with Model 1.2 outperforming Model 2.2 in terms of the specified metrics.

Model 2.1 had a train loss of 2.32 and a validation loss of 2.37 after 20 epochs, using a batch size of 16 and a learning rate of 0.01. The train accuracy at the end of 20 epochs was 48.56%, and the validation accuracy was 48.77%. The corresponding train and validation losses were 1.94 and 1.96, respectively.

On the other hand, for Model 2.2, which also ran for 20 epochs with a batch size of 16 and a learning rate of 0.01, the train accuracy at the end was 73.23%, and the validation accuracy was 73.81%. The train loss and validation loss for Model 2.2 were 0.97% and 0.96%, respectively. Additionally, the test accuracy and test loss for Model 2.2 were recorded as 73.45% and 0.96%, respectively. These results highlight the impact of model architecture on performance, emphasizing the intricate interplay of architecture and initialization techniques in neural network training.

C. Model 3 (Conv1, Relu, Maxpool, Flatten1, Dense1, Softmax, Cross Entropy)

In our third model, we've decided to add a convolutional layer and maxpool before flattening and adding a softmax dense layer. The results can be seen below in Figure 6 and Figure 7.

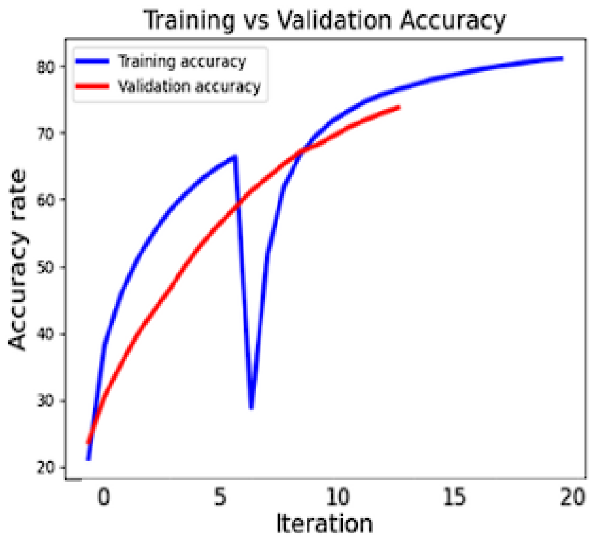


Fig. 6 A graph resembling the training accuracy and validation accuracy of Model 2.

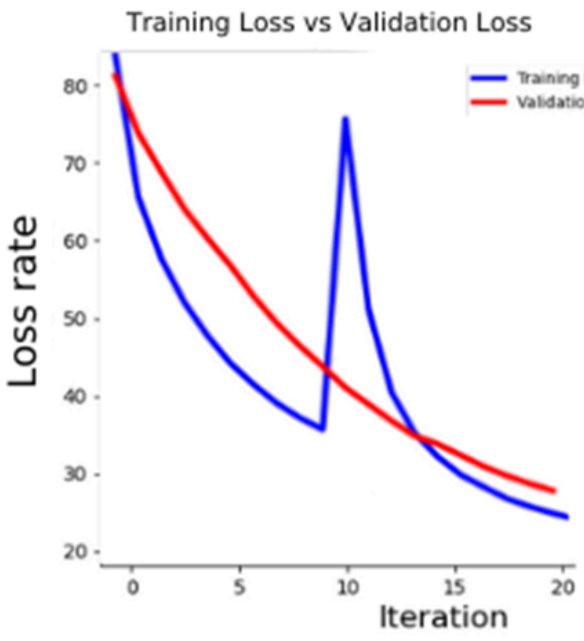


Fig. 7 Training vs Validation Loss for Model 3.

Model 3, utilizing convolutional operations, exhibited prolonged training times compared to other models. Despite achieving metrics similar to Model 1.2, it demonstrated a longer time per epoch, indicative of heightened computational complexity. Notably, an abrupt loss spike at the 11th epoch necessitates further investigation.

For Model 3 with 10 epochs, batch size 64, and a learning rate of 0.01, the 11th epoch's accuracy was 66.42% (training) and 67.18% (validation) with corresponding losses of 1.3163 and 1.2807. The test accuracy reached 68%.

Extending the analysis to 20 epochs maintained the batch size and learning rate, yielding improved metrics. The 30th epoch showed a training accuracy of 81.17%, validation accuracy of 81.40%, and losses of 0.7258 (training) and 0.7287 (validation). The test accuracy at 20 epochs was 81%. The prolonged training duration and loss spike at the 11th epoch highlight the need for deeper investigation into Model 3's unique challenges.

D. Model 4 (Conv1, Relu1, Maxpool1, Conv2, Relu2, Maxpool2, Flatten1, Dense1, Softmax, Cross Entropy)

In our fourth model, we've added 2 convolutional networks with both ReLu activation functions before flattening with a dense layer. Our results are in Figure 8 and Figure 9.

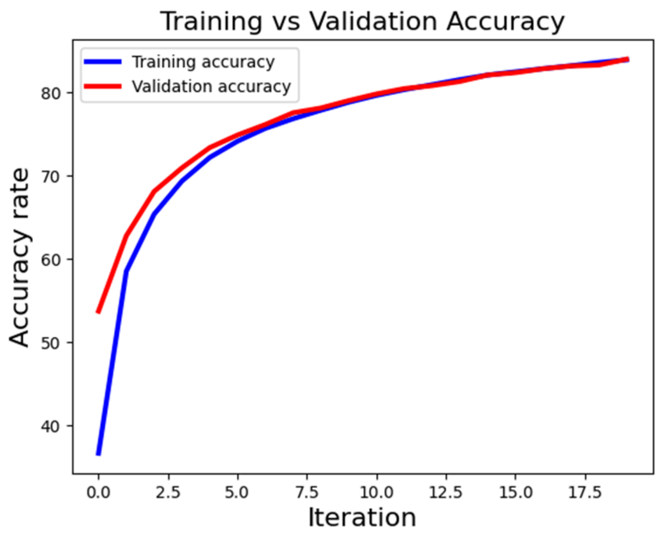


Fig. 8 A graph representing the Training vs Validation Accuracy for Model 4.

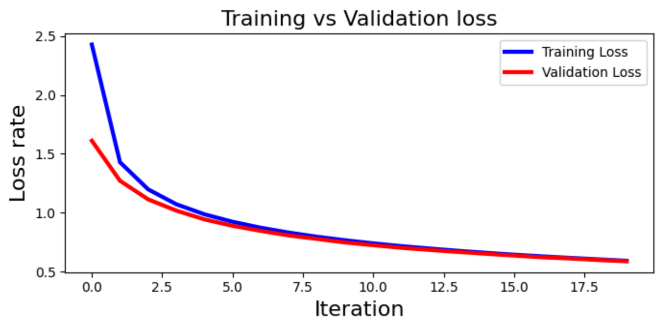


Fig. 9 Training vs Validation Loss for Model 4.

Model 4, while achieving an overall accuracy of 83%, exhibited a noteworthy disparity in performance between test data and real-world images, suggesting potential preprocessing issues. This discrepancy raises concerns about overfitting, as the model appears less effective in real-world scenarios. To address this, there is a crucial need for additional work in dataset preparation, especially in augmenting images for classes with limited representation. The detailed classification report further emphasizes variations in precision, recall, and F1-score across different classes, with a particular focus on classes with lower support values, indicating fewer instances. These findings underscore the importance of refining preprocessing steps and augmenting data to enhance Model 4's adaptability to diverse and challenging real-world conditions.

IV. Conclusion and Future Work

In conclusion, the integration of Convolutional Neural Networks (CNNs) with bounding boxes represents a pivotal advancement in object detection, significantly enhancing localization precision across diverse applications. While this collaboration proves instrumental in tasks such as autonomous vehicles and facial recognition, challenges arise in the real-world performance of the best model, indicating potential issues in preprocessing and the risk of overfitting. Addressing these concerns requires focused efforts on dataset preparation, including image augmentation for classes with limited samples, to enhance generalization in complex real-world scenarios.

Moreover, optimizing the computational efficiency of CNNs emerges as a key priority. While forward propagation is efficiently vectorized, attention should be directed towards vectorizing code during backward propagation, especially in gradient calculations across channels. Exploring methods to expedite the training speed of CNN layers, such as leveraging Python optimization and vectorization over image channels, holds the potential to enhance overall model efficiency. Looking ahead, exciting prospects include projects focused on advanced calculators for complex problem-solving and the extension of CNN applications to language handwriting recognition, showcasing the continuous innovation in object recognition systems through versatile CNN technologies. The overall steps in our algorithm are to read in an image of a linear equation, get bounding boxes for each character, pre-process each bounding box(sub-image), load in the saved model through a pickle file, and finally get predictions of each image. The final step is to use the Python library SymPy to solve the equation. There is some preprocessing that needs to be done to convert the classes we recognize to a valid form Sympy will accept, but given an image the bounding boxes will give an array where each index corresponds to each bounding box left to right in the image. From this we will compute the result. In the future we would like to come up with a better metric to analyze error within the equations that we compute and solve, as another means of predicting the real life error of our algorithm.

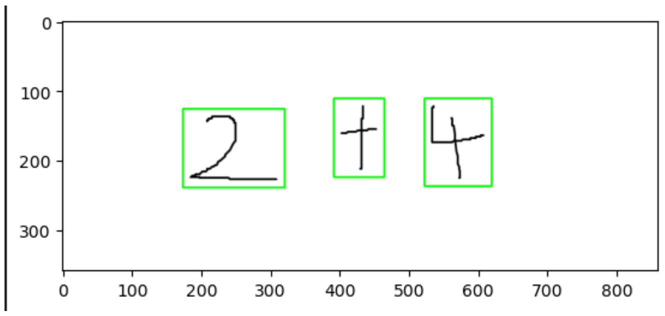
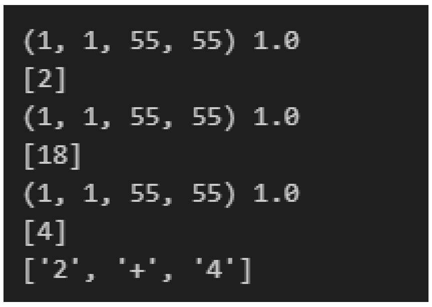
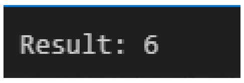


Fig. 10 Sample Equation we are testing on for the Equation solver



```
(1, 1, 55, 55) 1.0
[2]
(1, 1, 55, 55) 1.0
[18]
(1, 1, 55, 55) 1.0
[4]
['2', '+', '4']
```

Fig. 11 CNN output from the image in Fig.10



```
Result: 6
```

Fig. 12 Final Result with Sympy

References

[1] V. S. Subramanyam, "Basics of Bounding Boxes," *Analytics Vidhya*, Jan. 29, 2021. https://medium.com/analytics-vidhya/basics-of-bounding-boxes-94e583b5e16c

[2] "Softmax and Cross Entropy Loss," *www.parasdahal.com.* https://www.parasdahal.com/softmax-crossentropy

[3] "Derivation of the Gradient of the cross-entropy Loss," *jmlb.github.io.* https://jmlb.github.io/ml/2017/12/26/Calculate_Gradient_Softmax/ (accessed Nov. 26, 2023).

[4] "Softmax and Cross Entropy Gradients for Backpropagation," *www.youtube.com.* https://www.youtube.com/watch?v=5-rVLSc2XdE&ab_channel=SmartAlphaAI (accessed Nov. 26, 2023).

[5] A. Rosebrock, "Understanding weight initialization for neural networks," *PyImageSearch*, May 06, 2021. https://pyimagesearch.com/2021/05/06/understanding-weight-initialization-for-neural-networks/

[6] "Stride - Convolution in Neural Networks," *www.youtube.com.* https://www.youtube.com/watch?v=3TdBtI9dh2I (accessed Nov. 26, 2023).

[7] "CNN BackPropagation Fall 2021." Available: https://deeplearning.cs.cmu.edu/F21/document/recitation/Recitation5/CNN_Backprop_Recitation_5_F21.pdf