**Solving Handwritten Mathematical Equations**

Jimin Heo, Jasmine Son, Hazel Yu

**Introduction**

The objective of this project is to digitize handwritten mathematical equations and solve them. Given an image of a handwritten mathematical expression, the model should segment the expression into individual symbols and evaluate the mathematical expression that results from it. In order to simplify the scope of this project, we limited the expressions to simple equations with addition, subtraction, multiplication, and division operators, as well as evaluating only whole numbers.

**Methods**

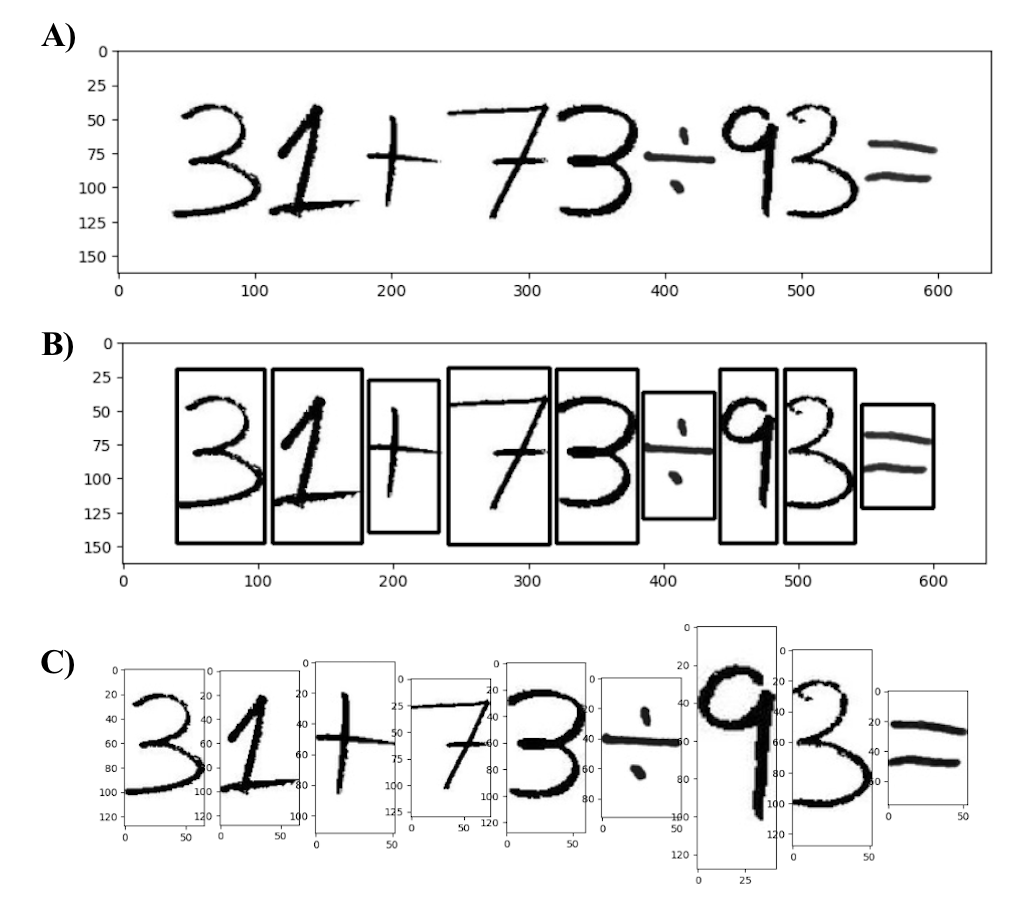
*Data*

We use the Handwritten Math Symbols dataset from Kaggle for our training set of handwritten numbers and mathematical symbols[1]. The data contains over 10000 images of digits, and mathematical operators and symbols (+,-,✕,÷, equals), with a total of 15 classes. The images are sorted into folders with the name of the number or symbol. We use the name of the folder as the class label of the image for training. We split the data into 60% train and 40% validation. We transformed the images by converting them to grayscale, resizing them to 32 x 32 pixels, and converted them into PyTorch tensors.

We use a Handwritten Math Equation Image Generator found on Github[2]. The repository uses images of handwritten numbers and math operators and generates images of random equations. We edited some of the code to generate the images we needed. The format of the equations are first number, operator, second number, operator, third number, and an equal sign. The equation images were used as our test data. For each equation, we created bounding boxes for each number or symbol and then extracted them and stored them as individual images. Figure 1 shows our process of extracting numbers and symbols from an image of an equation.

*Bounding Box*

To create the bounding box, we first converted the image into grayscale and then changed it to binary image data. Since the numbers and symbols are handwritten, the strokes were not thick enough, so we dilated the image. However, this caused the symbols to be too close to each other, where the bounding box could not separate the individual symbols. We then transformed the images to have enough space between the symbols. With these two changes the images could be contoured more clearly. After that using (x, y) coordinates we cut the images into individual symbols, and input them for testing.



**Figure 1. A)** Image of Equation. **B)** Bounding boxes of all numbers/symbols. **C)** Extracted numbers/symbols.

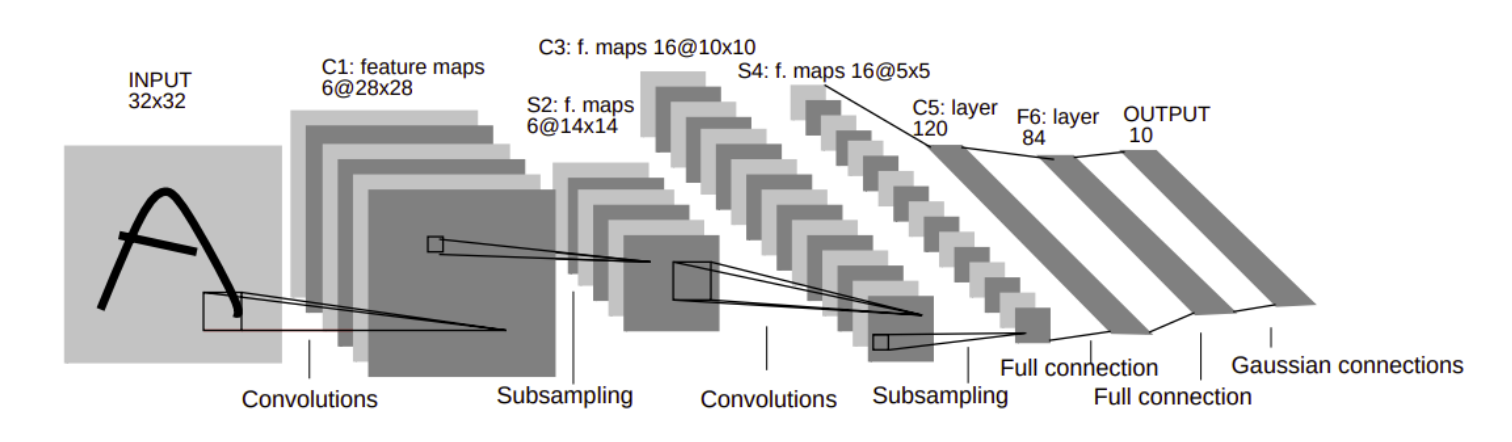
*Generalizing Testing Data*

Our goal was to generalize our model so that we can input any handwritten equation. However, in the beginning, testing with data that varied from the training data produced inaccurate results.

Therefore, we tried several ways to generalize the input. The most significant change was to resize the subimages obtained from the bounding box. As shown in figure 1C, the extracted images are not uniform. When we directly resized the image to 32\*32, the images were distorted. To solve this issue, we preserved the image proportions and added padding when resizing them.

*Convolutional Neural Network*

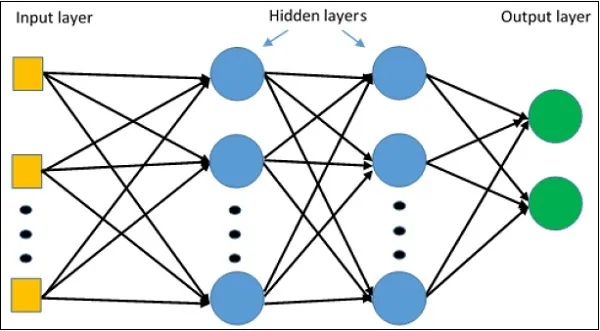
To train the dataset of numbers and math operations, we used a Convolutional Neural Network. We used the LeNet-5 architecture[3] (Figure 2), which has the form CONV - ReLU - POOL - CONV - ReLU - POOL - CONV - ReLU - FC - ReLU - FC. The convolutional layers use a 5x5 kernel with stride 1 and the pooling was done using average pooling over a 2x2 window. We used the ReLU activation function for each layer. The output is the probabilities that that image will belong to each class.



**Figure 2.** LeNet Architecture (from LeCun 1998).

*Multilayer Perceptron*

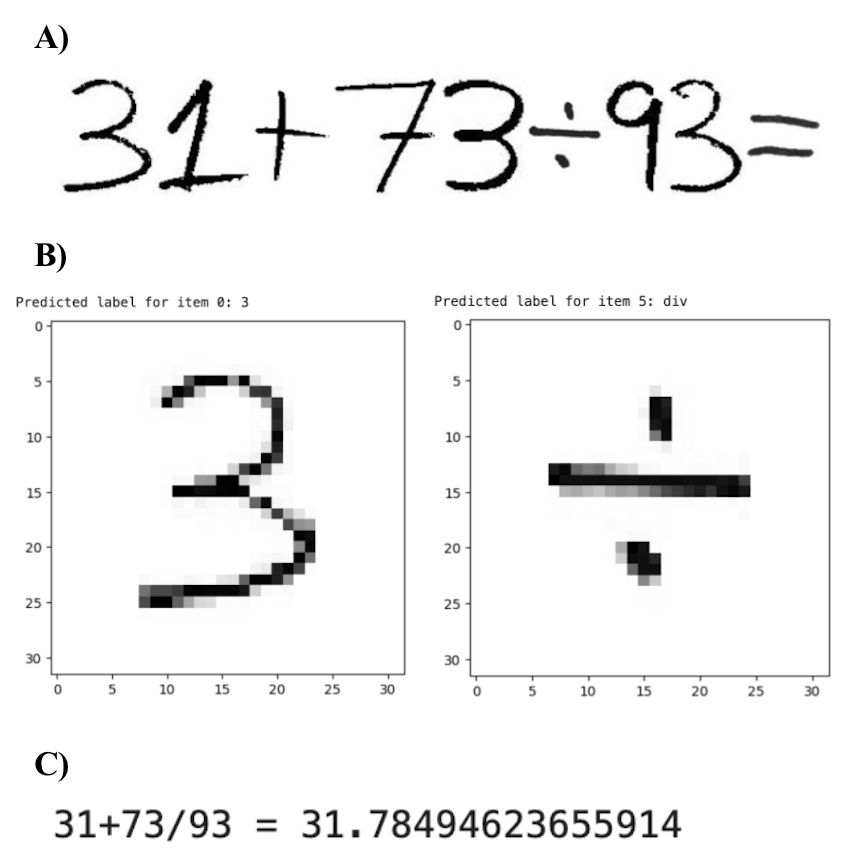
Our second method was to use a Multilayer Perceptron (MLP)[4] for training. We use 3 hidden layers, along with an input and output layer. The input layer has 1024 nodes, the hidden layers have 256, 128, and 64 nodes respectively, and the output layer has 15 (number of classes) nodes. Our first MLP model used a ReLU activation function, which introduces nonlinearity to the model, so that it can learn complex patterns. The model wasn’t performing well, so we added batch normalization and dropout regularization. For our second MLP model, after each hidden linear layer, there is a batch normalization layer to help stabilize and speed up the training process. After the batch normalization layer, there is a ReLU activation function and after that, there is a dropout layer to help reduce overfitting. The input of MLP is reshaped, then passed through the hidden layers. The output layer applies the Softmax function to the output of the hidden layers to obtain the class probabilities. We use Cross Entropy loss as our criterion and Adam optimization criterion.



**Figure 3.** Multilayer Perceptron Architecture (from Turner 2022)

*Expression Evaluation*

The function to evaluate the expression takes in a list of the predicted symbols, and creates a mathematical expression. Figure 4 shows the full process from input to output.



**Figure 4. A)** Input image of handwritten equation. **B)** Example of the predicted labels for each symbol. **C)** The final output and calculation made from the input.

**Results**

*CNN Results*

To evaluate our model, we measure the number of correctly classified images divided by the total number of images we fed through the model. The LeNet model achieves 94% train accuracy and 88% valid accuracy. There is some slight overfitting, which we hypothesize is due to a smaller dataset. Figure 5 shows the training and valid loss and accuracy over epochs.

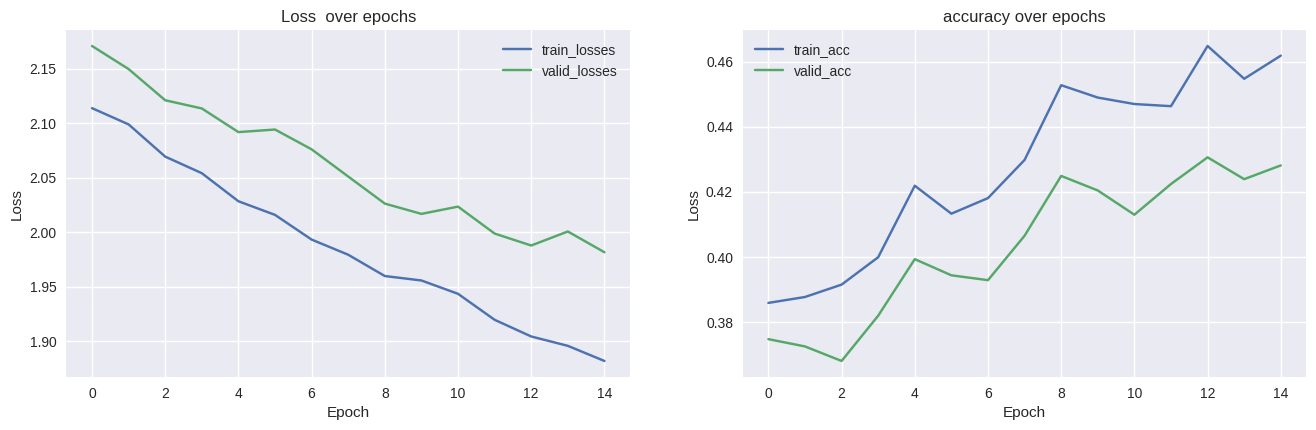


**Figure 5.** LeNet Training & Validation Loss and Accuracy over epochs

*MLP Results*

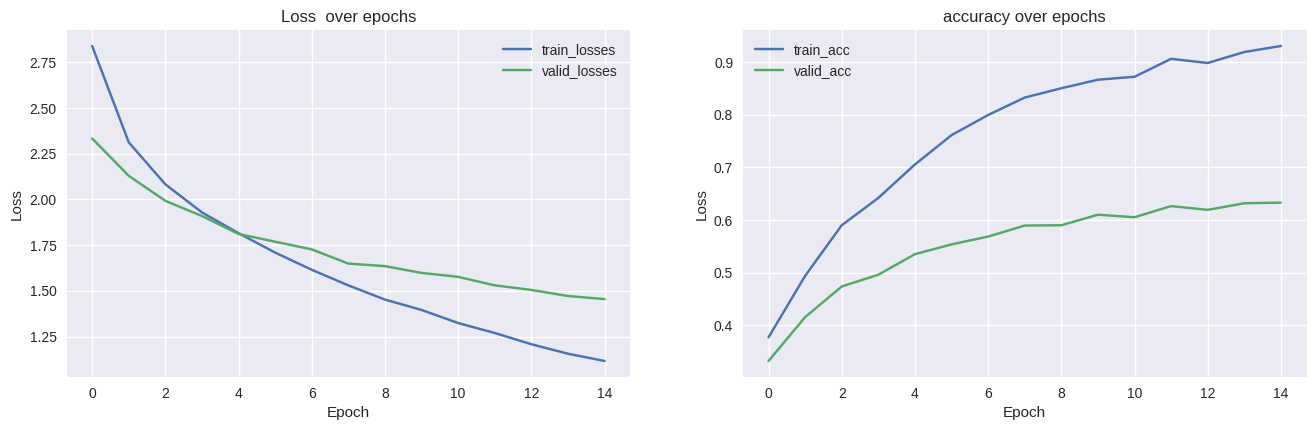
Similar to CNN, we evaluate our model by measuring the number of correctly classified images divided by the total number of images.

The first MLP model with no batch normalization or dropout achieves 46% train accuracy and 42% valid accuracy. While the train and valid accuracies are not too different, the accuracies are still quite low. Figure 6 shows the training and valid loss and accuracy over epochs.



**Figure 6.** Multilayer Perceptron 1 Training & Validation Loss and Accuracy over epochs

The second MLP model achieves 93% train accuracy and 63% valid accuracy. The model overfits the data a lot. We hypothesize that the reason for our results is because our dataset is quite small. MLP has worked quite well for MNIST data, but that dataset has 60,000 images for training while we only had about 6000 images. Figure 7 shows the training and valid loss and accuracy over epochs. We also tested to see how the dropout rate affects the performance. Table 1 shows the results of this. A lower dropout rate has higher training accuracy, but low valid accuracy.



**Figure 7.** Multilayer Perceptron 2 Training & Validation Loss and Accuracy over epochs

**Table 1.** Comparison of Dropout rates in MLP

| **Dropout Rate** | **Training Accuracy** | **Valid Accuracy** |
| --- | --- | --- |
| 0.1 | 91.33 | 59.77 |
| 0.3 | ​​74.63 | 55.90 |
| 0.5 | 54.26 | 46.16 |

**Conclusion**

Our first initiative was to train complex equations that included fractions, integrals, square roots, and parentheses. However, determining the spatial relationships between symbols, especially with fractions and superscripts, proved to be more challenging. Therefore, we reduced the scope of our project, and solved simpler equations with the four basic mathematical operations.

When comparing the CNN and MLP models, CNN had a much higher accuracy for our data. This is because of the translational invariance property of CNN. We learned to build CNN models and separate images into individual symbols to make it trainable. In the future, we can use x and y coordinates to train our model to handle more complex equations with fractions and square roots. We could potentially use an RNN model to do so.

**Contributions**

Jimin Heo worked on CNN(LeNet), bounding boxes, transform testing data for generalization, and the report.

Jasmine Son worked on the calculation function, MLP, data splitting, and the report.

Hazel Yu worked on MLP, generating equation images, plotting performances, and the report.

**References**

[1] Thapa, S. (n.d.). Handwritten Math Symbols. Kaggle. Retrieved from https://www.kaggle.com/datasets/sagyamthapa/handwritten-math-symbols

[2] RobinXL. (n.d.). Handwritten Math Equation Image Generator. GitHub. Retrieved from https://github.com/RobinXL/Handwritten-Math-Equation-Image-Generator

[3] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

[4] Turner, W. (Year, Month Day). Build a Model Using Multilayer Perceptron. Medium. Retrieved from https://medium.com/@willturnerau/build-a-model-using-multilayer-perceptron-791a62502387