APS360 Final Project Report
A Web-Based Hand-Writing Equation Solver

Group 10:

Binyu Li

Jiaxing Li

Ruijie Xu

Haoran Zhang

2021.12.08

Word Count: 2492

Primary Model Colab

https://colab.research.google.com/drive/1-o5PpD1SoBU5fomLkIeJ6qNSSqoKRHZg?usp=sharing

Baseline Model Colab

https://colab.research.google.com/drive/1XQKfSbdfqxDqU5vqp_NUceCcSqQtxm-2?usp=sharing

Project Github

https://github.com/leojiaxingli/APS360-Project

Final Report Google Docs

https://docs.google.com/document/d/1p4HWiu1_6kP_Y7SMgms4R_E6nsgaD6m8oDAIGg0pIBI/edit?usp=sharing

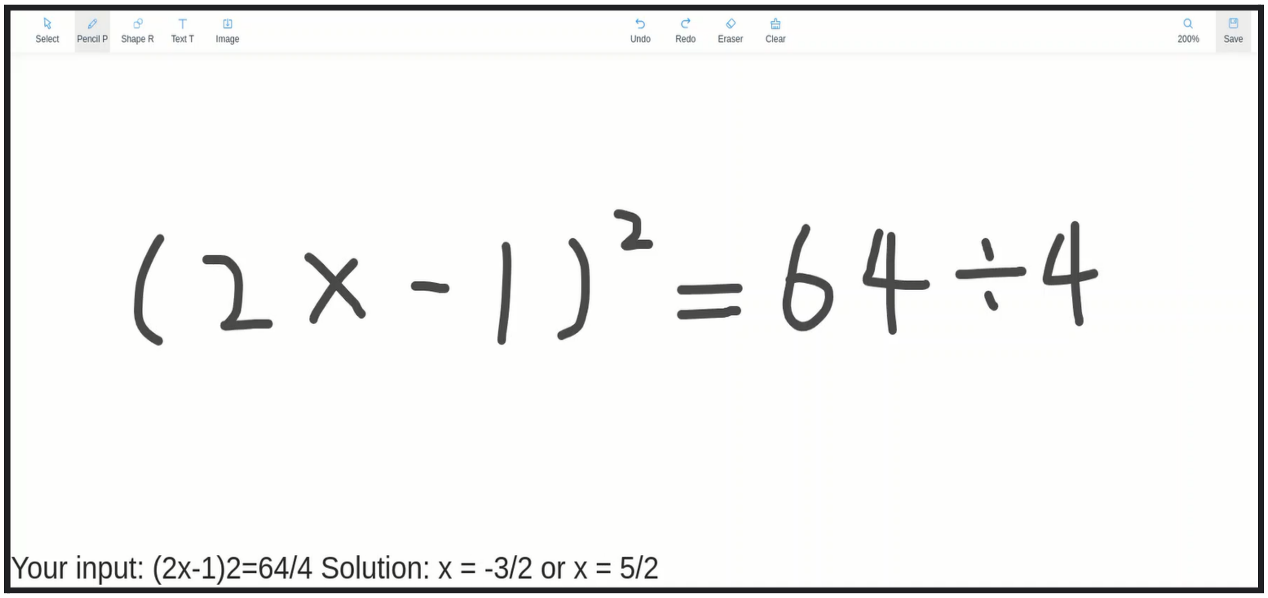
1.0 Introduction

An inevitable part of academic studies is solving mathematical equations. The effort of entering handwritten mathematical equations into a calculator is frustrating to most students [1]. It prolongs the process of learning and causes typing errors, leading to potentially miscalculated results.

Our project involves designing a web-based application that provides a whiteboard for users to input their handwritten equations, recognizes their handwritten equations through a machine-learning model, and outputs the text form of the equation and its solutions. The application serves as a time-saver, reduces calculation errors, and can be accessed anywhere with internet connection.

The major task of our project is to recognize handwritten mathematical symbols inputted by the user. The correct application of a neural network model can yield high accuracy, precision, and recall rate. In addition, there are existing datasets, such as MNIST, that have an abundant amount of well-labeled, handwritten digits training data available [2], which can be used in our proposed model to generalize the process of mathematical symbols recognition. Therefore, machine learning is a suitable approach to accommodate handwritten equations solver application.

2.0 Illustration



Your input: (2x-1)2=64/4 Solution: x = -3/2 or x = 5/2

Figure 2.1 Project Web Application Interface

3.0 Background & Related Work

The first paper guides us with a CNN model design. A subproblem of handwritten mathematical equations recognition is number recognition. The article proposes a Convolutional Neural Network (CNN) that yields a higher precision rate of 99.45% compared to 99.3% in traditional methods [3]. The CNN also performs well in English letter recognition, which is another subproblem of mathematical equations recognition due to trigonometric functions (sin, cos) and variables (x, y).

The second paper helps us with application interface design. It illustrates a mobile application that focuses on providing a user-friendly interface that reduces the learning curve for professional mathematical tools [4]. The application also includes a few additional features such as graph plotting and simultaneous equation solving. The cross-validation percentage accuracy on the Kaggle math symbol dataset is 99.2% with reduced symbols.

4.0 Data Processing

A handwritten mathematical equation can be broken down into three categories: digits (0-9), lowercase English letters (a-z), and symbols (+, -, =, etc.). Digits and letters are collected from the EMNIST dataset [5], and symbols are collected from a Kaggle dataset [6]. All image sizes are 28 by 28.

There were two data pre-processing steps required for the letters dataset. First, since the letters split option from EMNIST mixed uppercase and lowercase letters together to the same label, we filtered lowercase letters by selecting from the balanced split option. Second, we rotated each image 90 degrees to the right and performed a horizontal flip to ensure each letter stayed at the upright position.

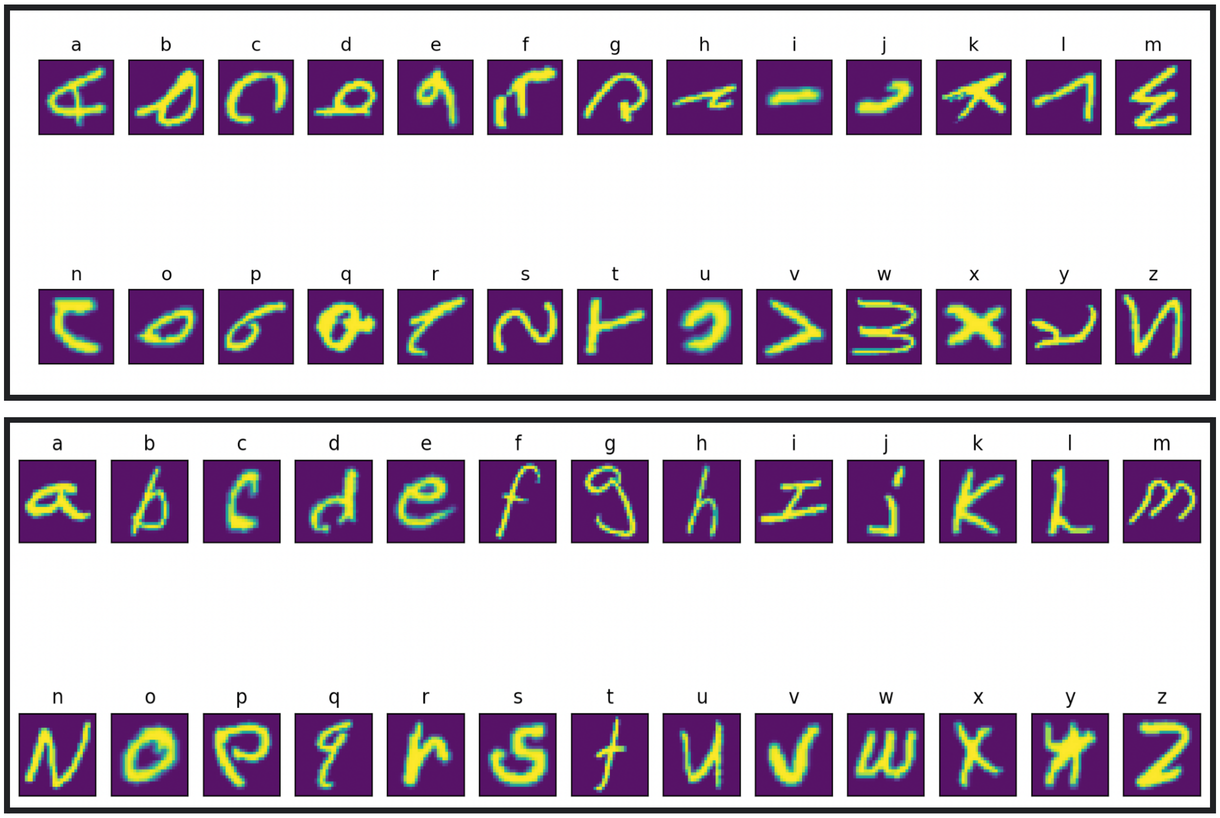


Figure 4.1 Raw Letters Data (Top) Pre-processed Letters Data (Bottom)

For symbols data pre-processing since all samples from EMNIST are white on black and gray-scaled but raw data from Kaggle dataset are black on white, we inverted all symbol data color, transformed them to gray-scale, and resized them to 28 by 28 to match EMNIST standard.

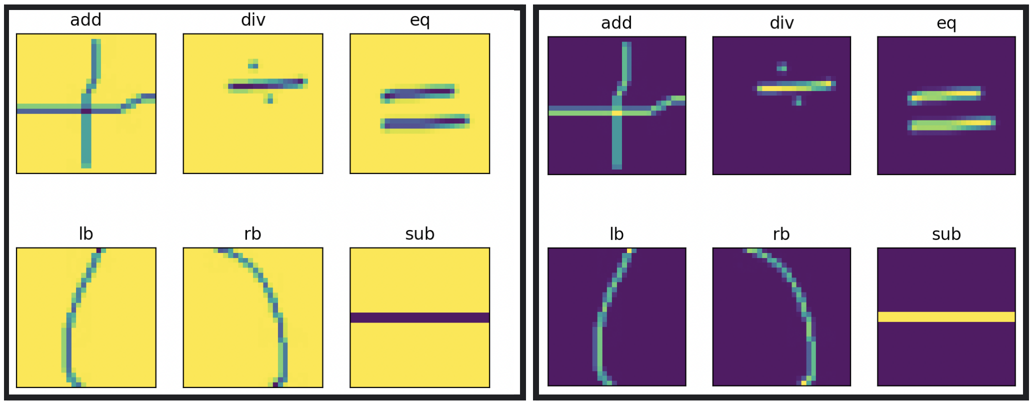


Figure 4.2 Raw Symbol Data (Left) Pre-processed Symbol Data (Right)

As shown in **Figure 4.3**, the initial data sample distribution is imbalanced. Therefore, we applied data augmentation on all three categories with details in **Table 4.4**. As a result, we have 10,000 data available for model training with training size of 7,000, validation size of 1,500 and testing size of 1,500.

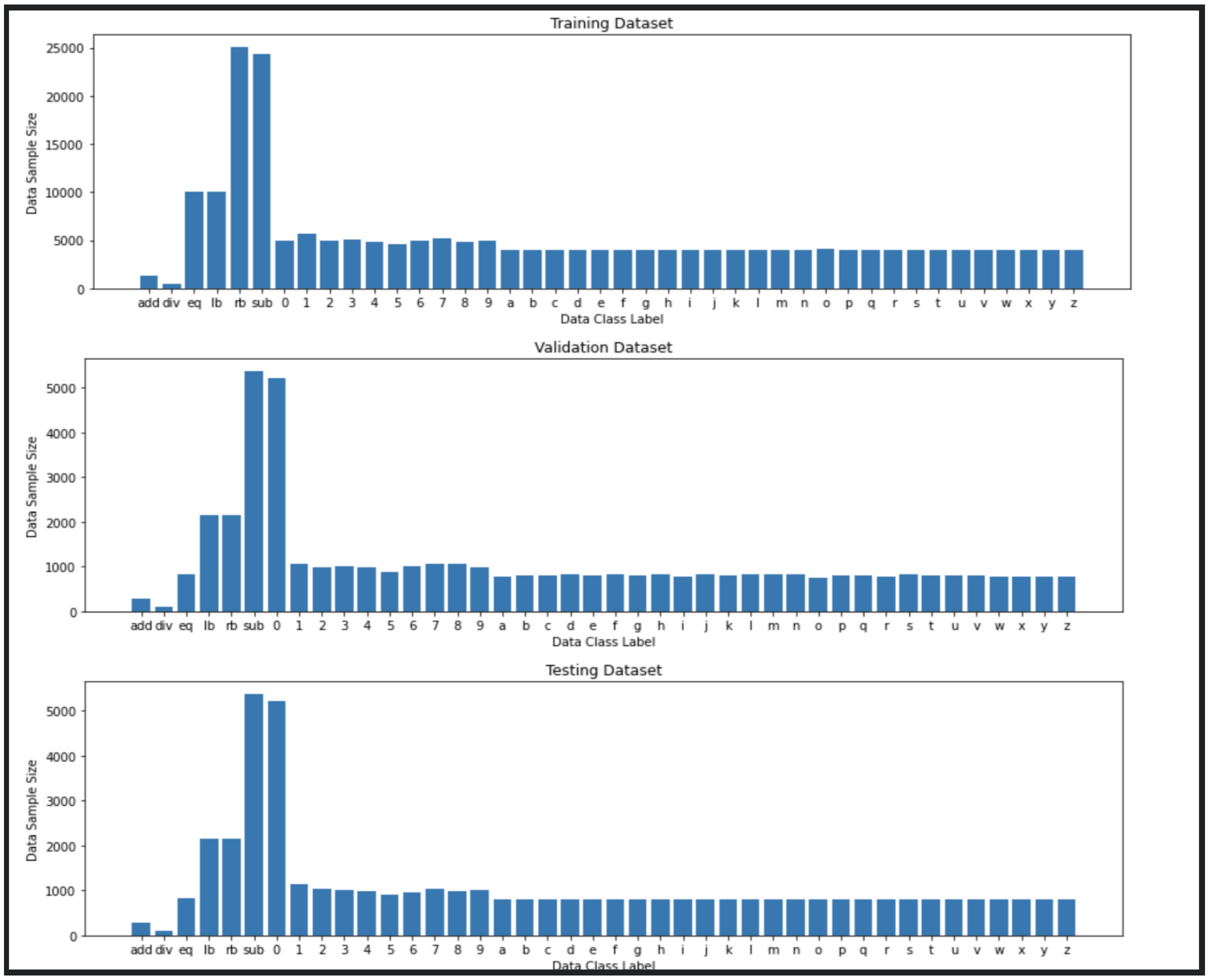


Figure 4.3 Initial Data Sample Distribution

Data Augmentation	**Data Size (# per class)**
Digits and Letters	
Translate in x, y direction with 0.12 = 28*0.12 pixels	4,000
Rotation -10 to 10 degrees	3,000
Rotation -15 to 15 degrees	3,000
Symbols +, -,=, ÷	
Rotation -15 to 15 degrees	4,000
Horizontal Flip	3,000
Vertical Flip	3,000

Table 4.4 Data Augmentation Summary

5.0 Architecture

After experimenting with many different models and different learning methods, the team chose CNN as the final model to tackle the task of handwritten equation recognition. In **Figure 5.1**, The model, in particular, has 2 convolutional layers with max-pooling between, together with two fully connected layers in the end to classify features extracted into results that we want. The first layer in particular has 1 input channel, 6 output channels, a 5x5 kernel size, and a padding of 2 pixels with a stride of 1. Input images are all in grayscale so there's only one input channel.

The main changes made to the model during training were the addition of paddings and the lowering of capacity in the fully connected layers. Initially, the team didn't choose to have any padding in the convolutional layers. The initial model was doing really well on the validation and test datasets, but the team discovered an issue with the model when testing it with new human input. The model was having trouble recognizing the number 9 correctly when they're written in full size. The top half of the loop was too close to the top edge of the image, causing the first convolutional layer to miss it due to the lack of padding, resulting in the misrecognition of the number as 7. The fully connected layers also had more capacity initially, with 120 as the number of out_features in the first fully connected layer. The model was trained over many epochs and it was exerting behavior of overfitting on training data, causing the model to perform poorly on unseen data. After realizing this, the team experimented with a different number of out_features and chose 80 in the end. With this change in place, the model's accuracy became more consistent across training, validation, and test data, as we can see in **Figures 7.1** and **7.2** comparatively.

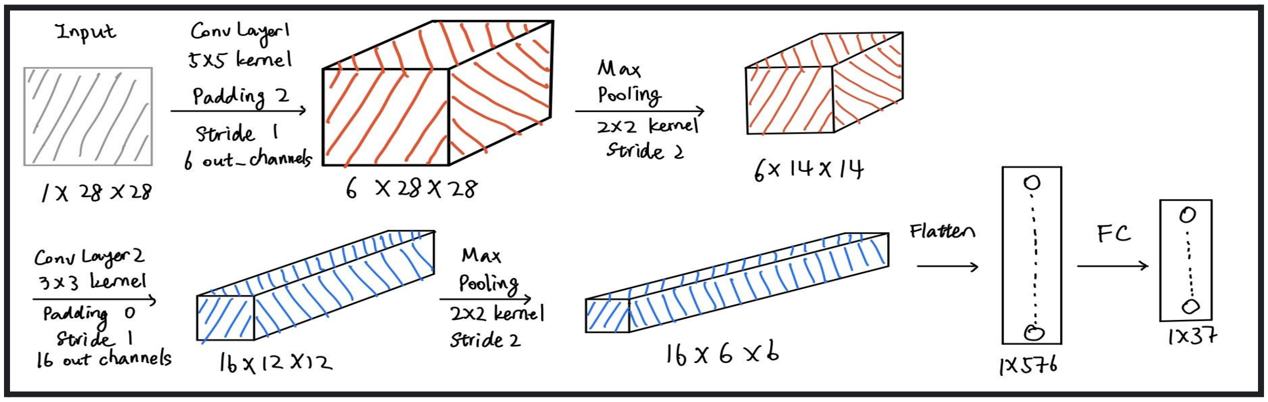


Figure 5.1 CNN Model Architecture

6.0 Baseline Model

The baseline model used to compare with our neural network is Support Vector Machines (SVM) [7]. SVM is a machine learning algorithm without utilizing neural network models. The SVM implementation from sklearn library is used to build the baseline model. After tunings on the different SVM implementations provided by the library with the initial dataset, sklearn.svm.SVC is determined to be the best-performing implementation so it's accepted for the baseline model. The SVM is utilized with the following parameters.

Kernel	**Degree**	**Gamma**	**Tol**	**C**	**Nu**	**Coef0**
Linear	3	Auto	1e-3	1.0	0	0
Shrinking	**Probability**	**Class Weight**	**Epsilon**	**Cache Size**	**Max Iteration**	
True	False	None	0	200	-1	

Table 6.1: SVM Parameters

7.0 Quantitative Results

The team has decided to use training loss/accuracy, validation loss/accuracy, and test accuracy as the 5 main measurements. Accuracy and loss are computed using the training and validation datasets at the end of every training epoch, while testing using the separate test dataset was performed once after the completion of the entire training process. As shown in **Figure 7.1**, illustrates the training (Blue) and validation (Orange) loss/accuracy curve of the model during training. Validation accuracy peaked at epoch 53 (95.581%) with the specific hyperparameter settings, so the team picked model parameters at that particular epoch as the best model. After training, the model parameters were loaded and measured against the test dataset, resulting in a test accuracy of 94.086%.

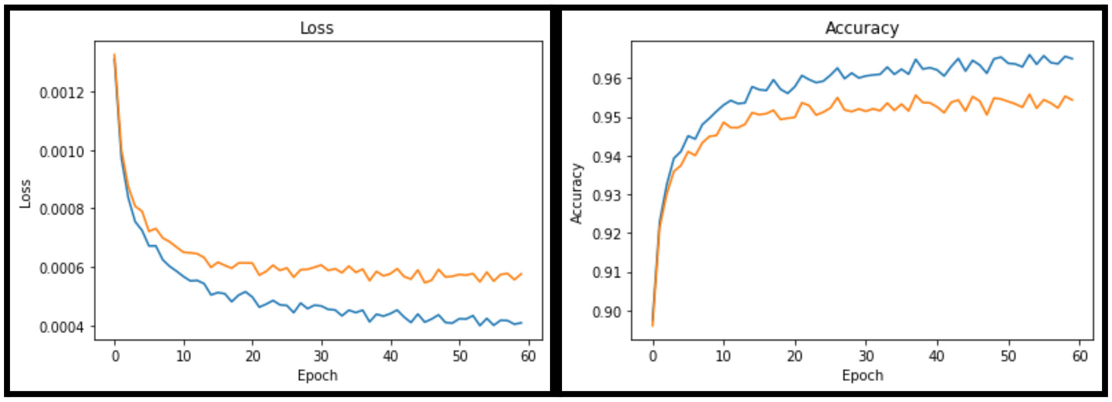


Figure 7.1 Loss (Left) Accuracy (Right)

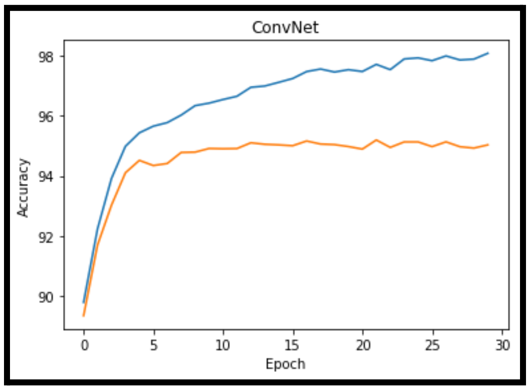


Figure 7.2 Accuracy Curve of Model with High Capacity

8.0 Qualitative Results

In general, the model does very well recognizing new data written by humans, but it sometimes has trouble distinguishing letters/numbers/symbols that naturally look alike.

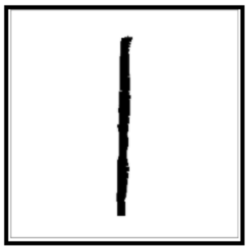
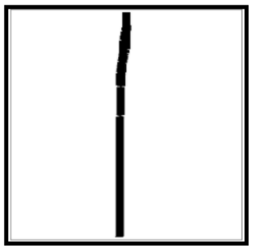
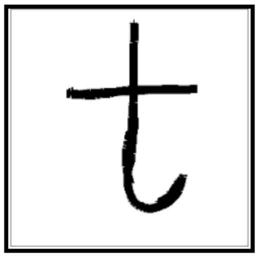
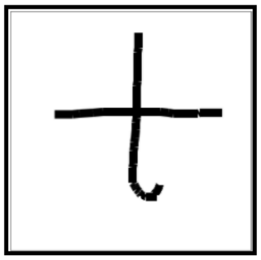
Input Image	**True Label / Prediction Result**	**Prediction Tensor (Label / Accuracy)**	**Explanation**
	**1** / correct	**1** / 0.9889	This is a perfectly written number 1, the model correctly classified the input with a high level of confidence.
	**1** / correct	**1 /** 0.55194 **(** / 0.4221	Although the model correctly identified it as 1, it did so with low confidence. This is due to the majority of the data collected for parenthesis spans all the way from the top edge to the bottom and their similarity in shape.
	**t** / correct	**t** / 0.9789	This is a perfectly written letter t, the model correctly classified the input with a high level of confidence.
	**t** / incorrect (+).	**+** / 0.5568 **t** / 0.4208	This letter is written more sloppily when compared to the previous one. The hook of the letter is written much smaller, so the model confused it with a plus sign.

Table 8.1 Quantitative Analysis of Mode on Selected Real-World Data

9.0 Evaluate Model on New Data

To evaluate the model on new data, testing data is withheld from the collected dataset. Real-world data collected from students are also used to evaluate the model.

9.1 Test Dataset

From data processing, 1,500 EMNIST [5] data is withheld for testing. Additionally, data augmentation is applied to generate extra test data: 500 with center crop, 500 with rotation angles, 500 with zoom in/out. Total test data size is 3,000.

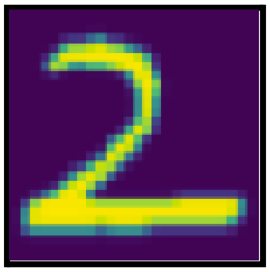
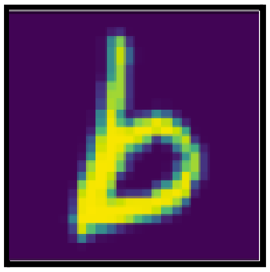
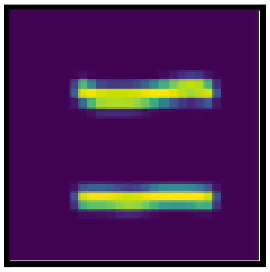
Input Image	**Category**	**True Label / Prediction Result**	**Prediction Tensor (Label / Accuracy)**	**Explanation**
	Number	**2** / correct	**2** / 0.961	This is a perfectly written number **2**, the model correctly classified the input with a high level of confidence.
	Letter	**b** / Incorrect (**6**).	**b** / 0.392 **6** / 0.489	**b** and **6** look very similar even to human eyes. The low resolution of the image also has effects on making the prediction uncertain.
	Symbol	= / correct	= / 0.993	Equal sign is very unique looking compared to the other data in the dataset. The model correctly classified the input with a high level of confidence.

Table 9.1 Test Dataset Analysis

9.2 Real World Data

The team also acquires data from real people. 15 students volunteered to participate in the data collection. Each student hand-writes 10 samples of the character set which the model is designed to recognize. The input is collected with the same writing board the project used to keep a consistent format. This yields 150 real-world new data for the team to evaluate the model. Note that the team members are excluded from this data collection to avoid bias. The raw data is adjusted to have thinner and thicker strokes to add more variations. This adds up to a total of 450 samples. Similar to the test data, data augmentation is applied to generate extra data: 150 with center crop, 150 with rotation angles, 150 with zoom in/out. Total test data size is 900.

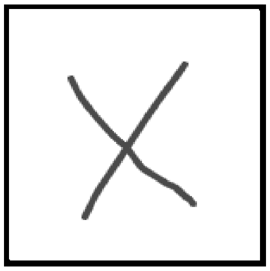
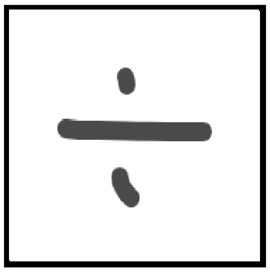
Input Image	**Category**	**True Label / Prediction Result**	**Prediction Tensor (Label / Accuracy)**	**Explanation**
	Number	**9** / correct	**9 / 0.339** **q / 0.218**	The training is done with EMNIST where **9** and **g** look very similar.
	Letter	**X** / correct	**x / 0.961**	Although the stroke is thinner, the uniqueness of the letter **x** makes the model give a confident prediction.
	Symbol	÷ / correct	÷ / 0.945	Division sign is very unique looking similar to the equal sign.

Table 9.2 Real World Data Analysis

10.0 Discussion

After a substantial amount of time was dedicated to the implementation of the model and data processing, our model is now able to achieve an accuracy of over 95% on the validation dataset. Compared to the baseline model we are using, there is a substantial increase in model accuracy, as our baseline model could only achieve 89% validation accuracy. On the other hand, in real-time testing, the model also yields promising results: it is able to identify and differentiate between similar characters, such as the number seven and the number nine, or the number one and a bracket.

The result induced from the project gives some interesting points that were unexpected. Through multiple trials of testing on our frontend whiteboard, the team has identified that the thickness of the drawing pen would have an impact on the consistency of results. For example, if the model is trained with a thicker pen, it will have a hard time testing data presented or drawn with a thinner pen. The team has therefore learned how important it is to have a training dataset that consists of large varieties of data augmentations so that the model is able to adapt to any form of new data. In addition, the team has initially used the whole alphabet list, numbers, and symbols as the nodes in the output layer, which means a probability would be computed on each one of them. This may be inefficient and affect accuracy in our specific project as many alphabet characters will rarely if not never show up in an equation. Therefore, the team, ultimately, eliminated some of the alphabets and only left the ones that are useful in an equation. Consequently, the model includes fewer categories to predict, which minimizes any confusion for the model, thereby increasing its accuracy.

Upon completion of the project, the team realized that machine learning is unable to work in all scenarios. At the start of the project, the team thought we could use machine learning to classify all the inputs into three categories, namely symbols, letters, and digits. However, the model's accuracy turned out to be low or non-functioning at all, as there isn't a specific way to distinguish between letters and symbols. In conclusion, although machine learning is useful and powerful, in certain circumstances, it is necessary to alternate the direction of solving the problem to gain full advantage of machine learning.

11.0 Ethical Considerations

The main source of ethical considerations of the project comes from the training data. The training process naturally involves the collection of handwriting from different individuals. Researchers at the University College London have developed a program that is capable of mimicking individuals' handwriting by studying their various characteristics (style, irregularities, spacing habit, etc) [8]. This poses potential risks to identity theft from the forgery of legal documents and signatures. Due to this reason, the team collected all training data from anonymous sources so individual privacy is protected.

12.0 Project Difficulty

The primary difficulty of this project stems from three sectors: data preprocessing, data augmentation, and different models. Given the difficulty of the project, the model is able to achieve an accuracy of over 95%, and in real-world testing on our front-end, the model possessed the ability to differentiate between similar characters such as one and a left bracket.

For data processing, the difficulties were pre-processing of raw letters and symbols dataset and data augmentation to balance all class sample sizes. As mentioned in **4.0 Data Processing**, the raw letters data from EMNIST and symbol data from the Kaggle math symbol dataset had issues that needed to be pre-processed such as inverting color and rotating to an upright position before training. Initially, we tried to train with imbalanced dataset size, and the model yielded a misleadingly high accuracy of 98.74%. We then exclusively tested for equal and divide signs since they each only have 600 samples compared to 25,000 samples for add sign; the accuracy was 20.57%. The team realized the importance of training on balanced data size, therefore, multiple data augmentations were performed in detail to ensure there were 10,000 samples per class as summarized in **Table 4.1**.

Image segmenter: For each character to be recognized by the model, image segmentation is performed to split the input equation into pieces. OpenCV is used to find the contours of each character. To correctly segment a non-connected character such as an equal sign (=), morphological transformations are performed. A cross-shaped kernel with dimension (3,27) is used to perform the transform. As shown in the figures below, the original image is transformed so that non-connected characters can be grouped to generate a single contour and segmented for the model to recognize.

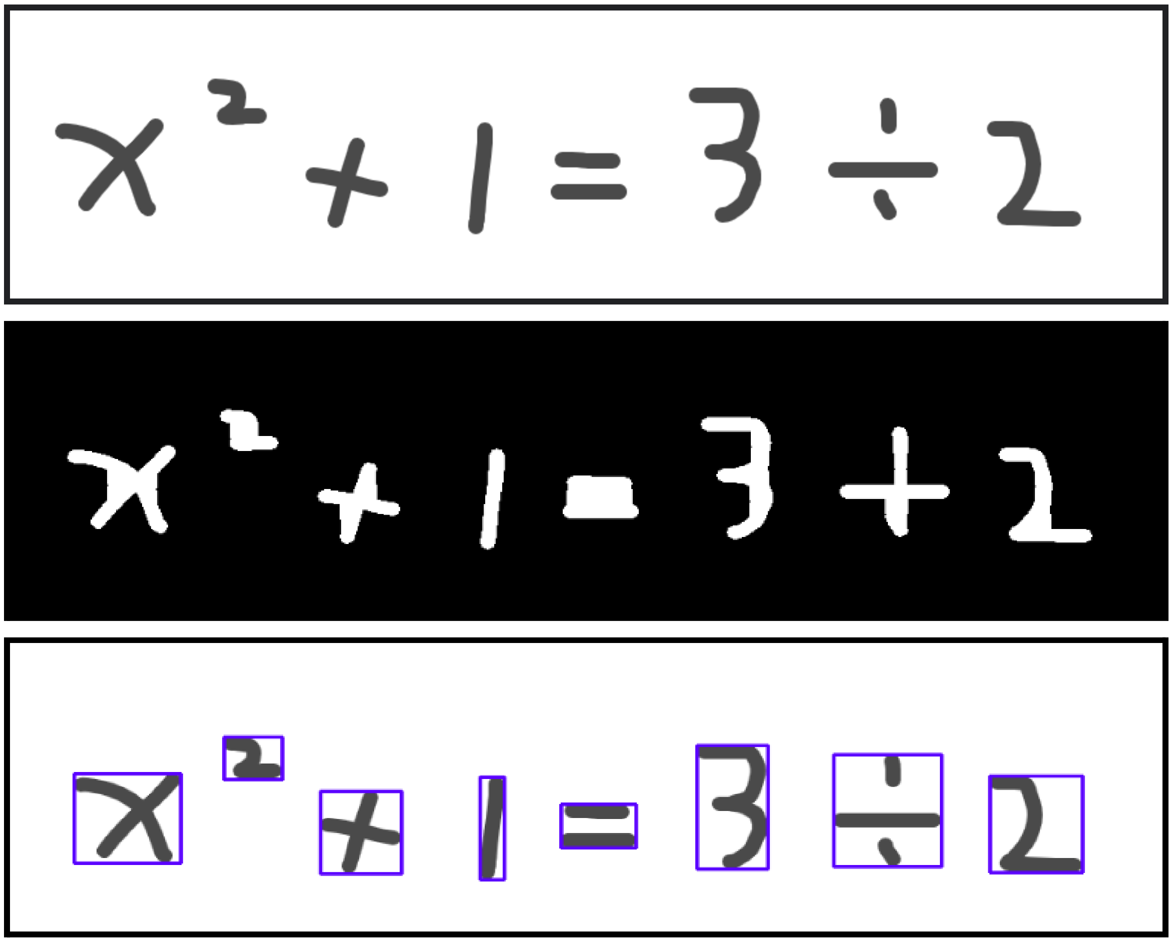


Figure 12.1 Original Image (Top) Image after Morphological Transformation (Middle) Image after Segmentation (Bottom)

Apart from the CNN architecture that was selected in the end, the team also considered and experimented with transfer learning with many other models that are pre-trained under the torchvision.models module. The team tried VGG16 [9] and GoogleNet [10] in particular. Despite their promising results in the field of image recognition, the models performed poorly on our task. Training the classification layers took a very long time and the accuracy was suboptimal. We think that these models' immense complexity and their focus on recognizing RGB images are the cause for their poor performance on our tasks.

13. Reference

[1] K. Chang, "Why mathematicians hate that viral equation," The New York Times, 06-Aug-2019. [Online]. Available: https://www.nytimes.com/2019/08/06/science/math-equation-pemdas.html.[Accessed: 08-Oct-2021].

[2] "The mnist database," MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 05-Oct-2021].

[3] Y. A. Nanehkaran, D. Zhang, S. Salimi, J. Chen, Y. Tian, and N. Al-Nabhan, "Analysis and comparison of machine learning classifiers and deep neural networks techniques for recognition of Farsi handwritten digits," The Journal of Supercomputing, 28-Jul-2020. [Online]. Available: https://link.springer.com/article/10.1007%2Fs11227-020-03388-7. [Accessed: 05-Oct-2021].

[4] S. B. K.S., V. Bhat and A. S. Krishnan, "SolveIt: An Application for Automated Recognition and Processing of Handwritten Mathematical Equations," 2018 4th International Conference for Convergence in Technology (I2CT), 2018, pp. 1-8, doi: 10.1109/I2CT42659.2018.9058273. [Accessed: 05-Oct-2021].

[5] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending mnist to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017. [Accessed: 1-Nov-2021]

[6] Xai Nano, "Handwritten Math Symbols Dataset (Version 2)," 15-Jan-2017, [Online]. Available: https://www.kaggle.com/xainano/handwrittenmathsymbols. [Accessed: 1-Nov-2021]

[7] "SVM Vs Neural Network," *Baeldung, Gabriele De Luca,* 25-Aug-2021. [Online]. Available: https://www.baeldung.com/cs/svm-vs-neural-network [Accessed: 08-Oct-2021].

[8] "This program can mimic your handwriting with shocking accuracy-what could go wrong?," The Daily Dot, 26-May-2021. [Online]. Available: https://www.dailydot.com/debug/handwriting-algorithm-privacy-security/.[Accessed: 05-Oct-2021].

[9] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," The International Conference on Learning Representations (ICLR 2015), 10-Apr-2015. [Online]. Available: arXiv:1409.1556v6. [Accessed: 15-Nov-2021]

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision*

and Pattern Recognition (CVPR), 2015. [Online]. Available: https://arxiv.org/abs/1409.4842. [Accessed: 15-Nov-2021]