COMPSYS 302 PROJECT 1

HANDWRITTEN DIGIT RECOGNITION USING MNIST DATASET

Andy (Sangwoo) Kweon

Group_01 / UPI: 408682449
skwe902@aucklanduni.ac.nz

1. Introduction

Team VA

We are group_01 and our group name is Team VA. The members of this project are Andy Kweon (myself) and Vishnu Hu (my partner) and for the past few weeks we have been undertaking the handwritten digit recognition project using the MNIST dataset.

Right: Vishnu Hu and Left: Andy Kweon (me)

2. Planning

2.1. Literature Review

In recent years, many advancements in the image classification using convolutional neural networks have been made. Some of the existing works that I have found during research are listed below:

Nitish Srivastava et al. [1] has tested a model based on convolutional nets and dropouts and won the ILSVRC-2012 competition. The study also shows that the best methods based on standard vision features get a top-5 error rate of 25% whereas convolutional nets with dropouts achieve a test error of 16%. The study also has tested the effectiveness of dropouts in MNIST dataset, the error was reduced up to 0.79% by finetuning dropouts.

Rahul Chauhan et al. [2] has used and tested a CNN model using MNIST dataset that showed an accuracy of 99.6%. The model consisted of 3 convolutional layers, each followed with ReLU activation function, max pooling, and dropout layers. The training was done with batch size of 128 and number of epochs 10.

Alex Krizhevsky et al. [3] have trained a deep convolutional neural network to classify 1.2 million images in the ImageNet LSVRC-2010 contest, and showed that a large, deep convolutional neural network can achieve high results using purely supervised learning.

Aman Dureja [4] has tested different activation functions for classifying cat/dog dataset using convolutional neural networks and found that ReLU function gave the best performance compared to Tanh, Selu, PRelu, Elu. A CNN model with 3 ReLU hidden layers gave a validation loss of 0.3912 and validation accuracy of 0.8320 at 25thepoch.

Aanchal Jain et al. [5] has compared the results of sigmoid, linear and tanh activation functions on MNIST dataset, where on binary cross-entropy error (10 epochs) the sigmoid function showed an accuracy of 99.65% at 1390 seconds of completion time.

2.2. Setup for the Project

The development was done using my own laptop, with CUDA enabled to accelerate the training process. The specifications of my machine are as follows:

- CPU: Intel® Core™ i7-8750H
- GPU: NVIDIA GTX 1060
- Memory (RAM): 16 GB

The operating system used was Windows 10 and the entire project was written in Python 3.8.5. Miniconda and pip, were used to install the needed modules and packages. The IDE for this project was Microsoft Visual Studio Code, as it was a simple and easy IDE to use. Github was also set up using Github Classroom and any changes made to the code were committed and uploaded using Github for easy collaboration between me and my partner.

2.3. Schedules and Roles of the Project

With the code and report due on April 29th, and the demo due on the 23rd of April, me and my teammate, Vishnu Hu have decided to split the roles into two main parts. Vishnu took charge of the GUI framework development and my role was to research and find the best deep learning model and integrate the deep learning model into the GUI so it could be trained using the MNIST dataset. The gantt chart for the project can be seen in fig.1 below.

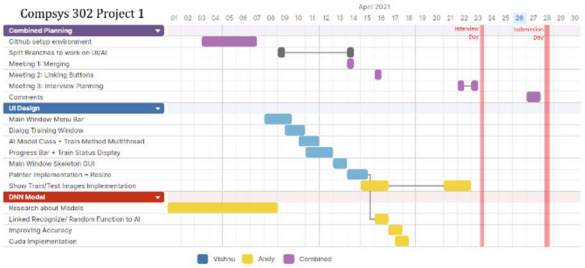

Fig. 1, Gantt Chart of the project

While during the project we sometimes did work related to each other's roles (eg. I would work on viewing the MNIST images on the GUI) but for the most part, my main role was to find and integrate the model into the GUI that Vishnu developed.

3. Software Architecture

3.1. Purpose of the System

Recently, artificial intelligence (AI) has been integrating more and more into our daily lives. From medical diagnosis to speech recognition, AI is the new frontier of engineering, and more companies and even governments are interested in developing this technology to utilize it to the fullest. Through this project, we aim to create a handwritten number recognition system that could detect what number a person has drawn with reliable accuracy. We also aim to provide a user-friendly user interface along with intuitive controls so a user can input their own handwriting and the trained model can detect what number has been drawn.

Through this simple project, we hope to raise awareness about the importance of AI to people as it could be used to demonstrate the power/limitations of AI and how it could be used in our daily lives. Furthermore, in the future, this tool could be modified to detect letters as well as numbers, which could be used as a text-to-speech tool and a handwriting recogniser, which could be used in the field of education.

3.2. Database and Model

The database used to train and test the model was the original MNIST dataset, consisting of 60,000 testing images and 10,000 training images. The MNIST dataset (shown in fig. 2) comes in image/label pairs, with the images being 28 by 28 pixels. For our project, the MNIST dataset was downloaded from the MNIST database [6].

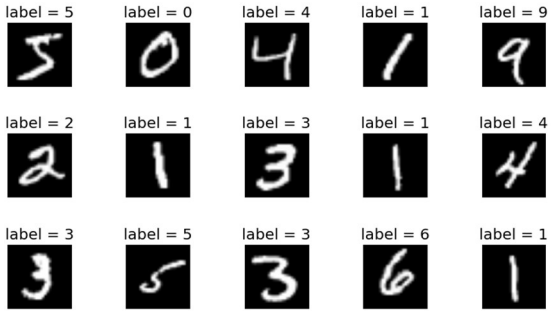

Fig. 2, MNIST Dataset

As the one overseeing everything related to the deep learning model, I had originally thought about looking to add our own handwriting to the dataset to improve accuracy. However, due to the time constraints, I decided to utilize only the MNIST dataset and to refine the model as much as possible to provide good accuracy.

The first model that I tested was the code given in the lab (Net1), which utilized just linear transformations and using ReLU as activation function. The accuracy of the model was not great, giving an accuracy of 9761 correctly classified images out of 10,000 testing images at 20 epoch, 32 batch size, and showed signs of overtraining from around 14 epoch. (shown in fig. 3)

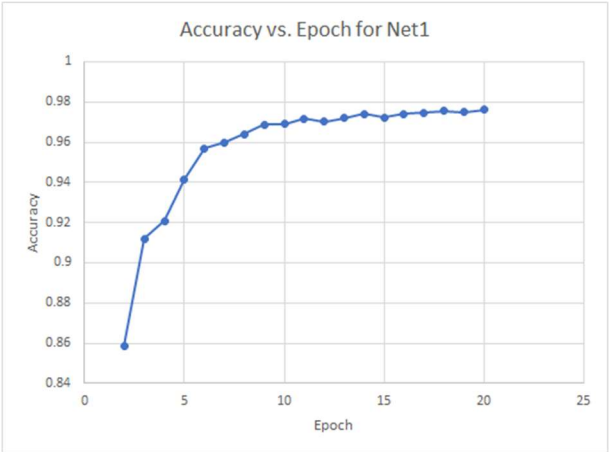

Fig. 3, Training results of Net1 (32 batch size)

With our initial model not providing great results, I have decided to research and find different models for handwritten digit classification. CNN in particular caught my attention, as previous researches done [2,3] have pointed out that CNN models are widely used in machine learning, especially in image recognition, and it has been also used in MNIST images with great success [2]. The first CNN model that I found [7] (Net2) featured 2D convolutional layers, pooling, dropouts and ReLU activation function (shown in fig. 5).

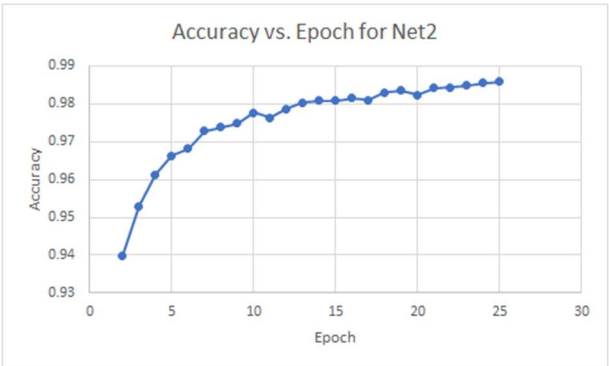

Fig. 4, Training results of Net2 (32 batch size)

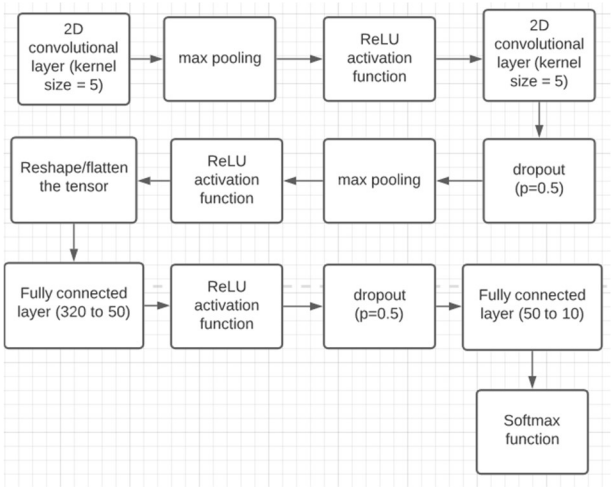

Fig. 5, Architecture of Net2

Overall, Net2 showed promising results, achieving 9823 correct images out of 10,000, and was able to get an accuracy of 0.9859 at 25 epoch, 32 batch size. (shown in fig.4)

However, when I tried testing the Net2 with my own handwriting, the model seemed to struggle with recognizing 9's and 3's, where it would detect 9 as a 7 (shown in fig. 6 Left) and 3 as a 2 (shown in fig. 6 Right) if I drew it in a certain way.

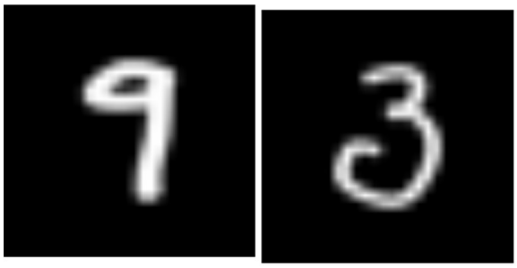

Fig. 6, Left: my own drawing of 9 into the GUI scaled down to 28x28 pixel input, Right: my own drawing of 3 (also scaled)

This was alarming, as it suggested that the model was overfitting, and the model was learning the statistical noise in the training data. This meant that when given a new previously unseen data, the model would have a poor performance [8]. Because the aim of this project was to get the model to recognize people's own handwritten numbers, I further researched and tried to find solutions to this problem. A few of the options I found and considered were:

- Reduce the epoch [9]
- Use dropouts [1]
- Find a different model
- Data Augmentation [9]

In the end, due to time constraints, I decided to find a different model and compare it with Net2, and try tweaking the model that seemed to have a better performance. The next model that I found (Net3) was also a CNN model, found online [10]. As shown in fig. 7, the model features three 2D convolutional layers, each with a kernel size of 5. ReLU was used as an activation function, and the model features multiple dropouts to avoid overfitting.

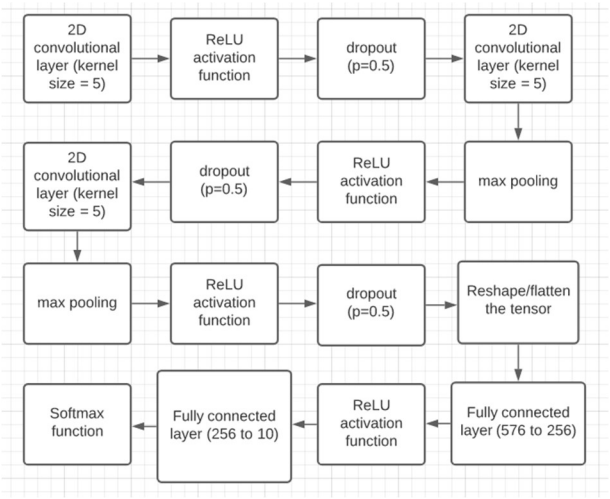

Fig.7, Architecture of Net3

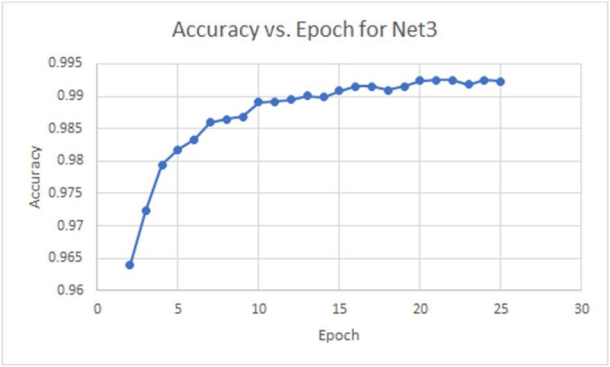

Fig. 8, Training results of Net3 (32 batch size)

The performance of Net 3 was even better than Net2, getting 9923 images correct out of 10,000 at 25 epoch, and showed a higher accuracy at all different epochs (shown in fig.8). Therefore, I decided to try and tweak the Net3, so it could reach the maximum performance.

After research, one of the easiest parameters to change was the batch size, and I tried to see how it would affect the performance of the Net3. After lowering it to 16, the result seemed promising, showing a slight improvement of 9928 correct images out of 10,000 at 25 epoch. (shown in fig. 9) After testing the same images as fig. 6, the model was able to detect the fig. 6 Left as 3 and was 99% confident. However, it still struggled with the number 9, and showed a higher probability that it was a 7 rather than a 9.

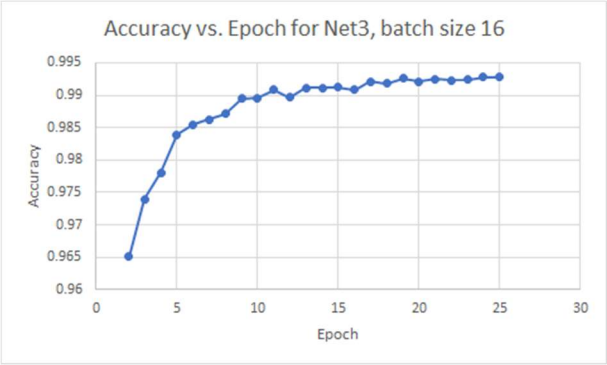

Fig. 9, Training results of Net3 with 16 batch size

Compared to the initial model given in the lab, the CNN models provided a visible advantage, at the expense of increased training time. However, the performance advantage was very visible, and after looking at the data and comparing the models, I decided on the final model as Net3 with 19 epoch and 16 batch size. This gave me an accuracy of 9930 correct images out of 10,000, with an average loss of 0.026 (fig.10). The model took 3 minutes to train on CUDA.

```
Training Model... Epoch size: 19
==========================
Test set: Average loss: 0.0260, Accuracy: 9930/10000 (99%)
```

Fig. 10 Training results after 19 epoch, 16 batch size

With the models taking longer to train, the utilization of CUDA in NVIDIA graphics cards was essential, which sped up the training process by a considerable amount of time compared with training on CPU. I would recommend anyone training the model to use a NVIDIA GPU and enabling CUDA to speed up the training process.

3.3. System Architecture and Components

The system consists of 5 python files. main.py (which contains the code for the main window of the GUI and imports the other files) AnotherWindow.py (which contains the code for training/testing of the model) Net3.py (which is the deep learning model) and TrainingImages.py and Testing Images.py (which contains the code to show and cycle through the MNIST dataset). The Architecture of the system can be shown in fig.11 below.

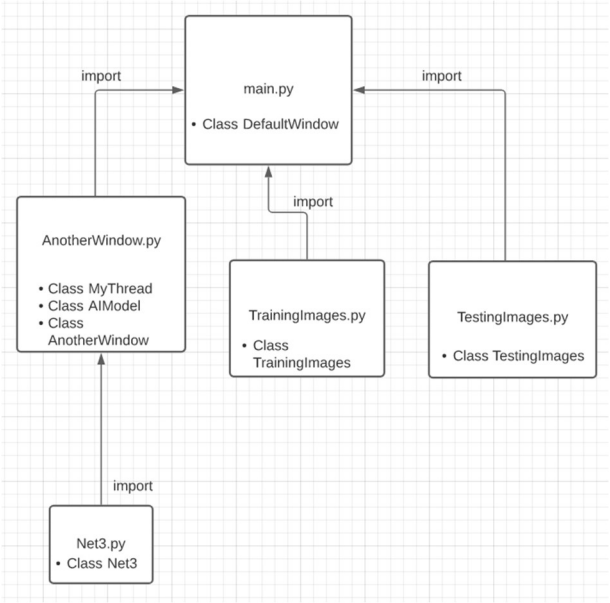

Fig. 11, System Architecture

We have tried to apply a modular approach to the system, where each file is a different module and can be removed/replaced/altered easily. This is especially true for Net3.py, which contains the deep learning model. People can create a different model and easily replace Net3.py, which means that once a more advanced/recent model comes out, we can easily update the code.

We used an object-oriented style, utilizing classes and functions to create this system. The most significant class is the class DefaultWindow in main.py. the class DefaultWindow contains the functions for the GUI of the main window, and imports the 3 other modules AnotherWindow.py, TrainingImages.py and TestingImages.py which all open up separate windows.

Once main.py is run, the code opens up the handwritten digit recognizer, shown in fig. 12. The left-center area next to the buttons is a large canvas, which can be drawn on by click+drag. On the right-hand side, we have the four buttons (shown in fig.13). The “Clear” button clears the canvas if the user makes a mistake while drawing. The “Random” button selects a random image from the MNIST training dataset, displays the image on the Image Viewer (shown in fig. 14) and tests the model, and finally, the “About Model” displays a short information about the model.

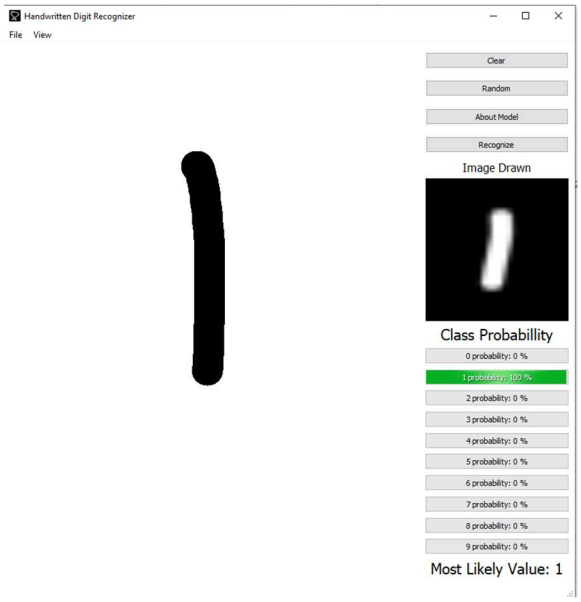

Fig. 12, Main Window of the Handwritten Digit Recogniser

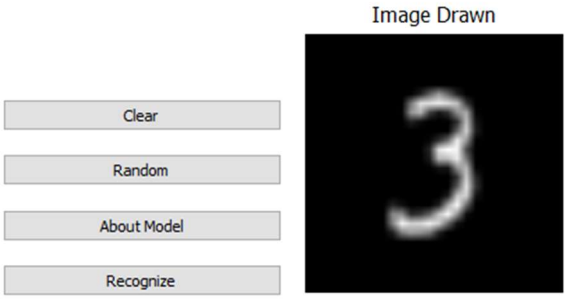

Fig. 13, Four Clickable Buttons (left) and Fig. 14, Drawn Image Viewer showing a random MNIST image (right)

Once the image has been drawn on the canvas, the image can be recognized by the deep learning model once the user clicks the "Recognise" button. The drawn image is then shown on the Image Viewer and the probabilities of each number are shown on the Class Probability (shown in fig. 15)

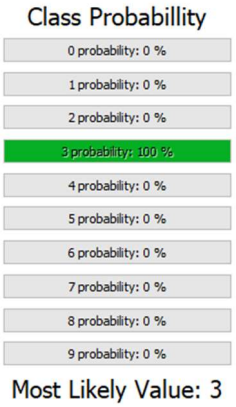

Fig. 15, Class Probability showing the most likely value and the probabilities of each number (result from the MNIST image in fig.14)

From the main window, the user can train the model by clicking "File" > "Train Model", which will call the AnotherWindow.py. (shown in fig. 16 left) The user can view the training/testing MNIST images by clicking on "View" > "View Training/Testing Images" (shown in fig 16 right).

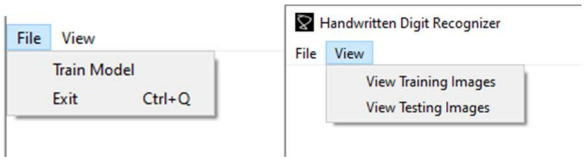

Fig. 16, Train Model (left) and View Testing/Training Images (right)

When "Train Model" is clicked, the Dialog window pops up (shown in fig. 17). Clicking "Download MNIST" will download the testing and training MNIST dataset. The "Train" button will call the deep learning model and train the model using the MNIST training images (60,000). The parameters for training (epoch, batch size etc.) can be set in class AIModel(object) inside AnotherWindow.py. The "Test" button will test the trained model using the training MNIST images (10,000) and the "Cancel" button will cancel and stop the training progress. Regardless of whether the training is completed or cancelled, the trained model is saved as trainedModel.pth file.

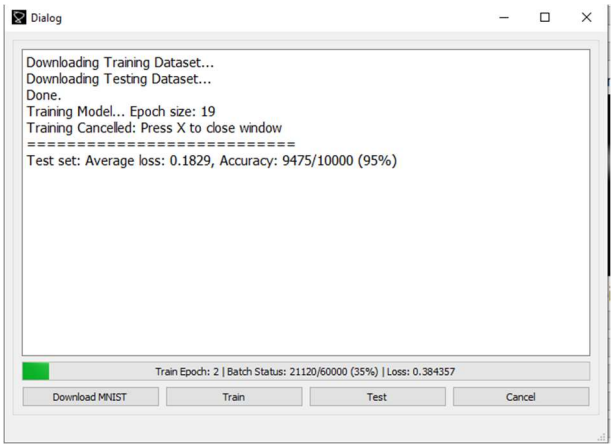

Fig. 17, Dialog Window

Once "View Training/Testing Images" is clicked, the GUI opens a new window (shown in fig. 18) and calls Training/TestingImages.py respectively. The window shows 100 random MNIST training/testing images and the next/previous 100 images can be shown by clicking "show Previous/Next". I have also implemented a filter function, where the images can be filtered by clicking in one of the check boxes on the left and clicking "Filter" (shown in fig. 19). If there are no previous images and the user clicks "Show Previous", the GUI will display an error message (shown in fig. 20).

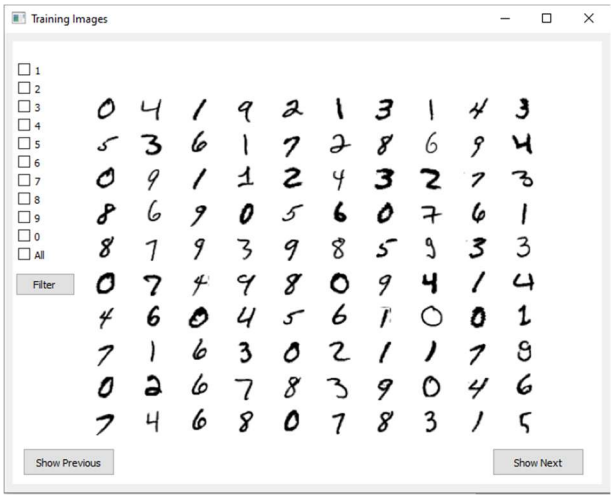

Fig. 18, 100 Random Training Images from MNIST Dataset

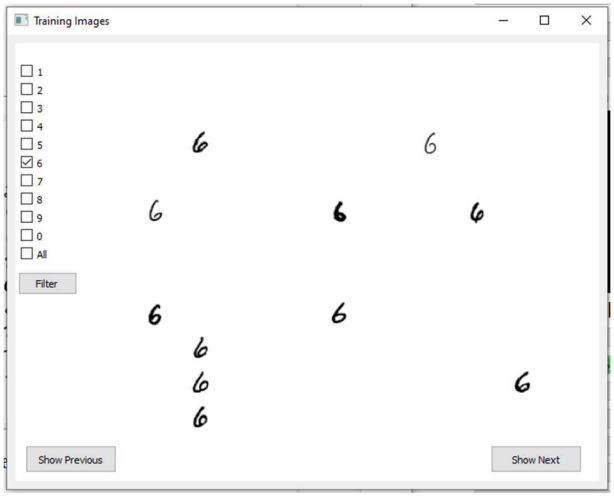

Fig. 19, Using "Filter" to Show Only the Digit 6.

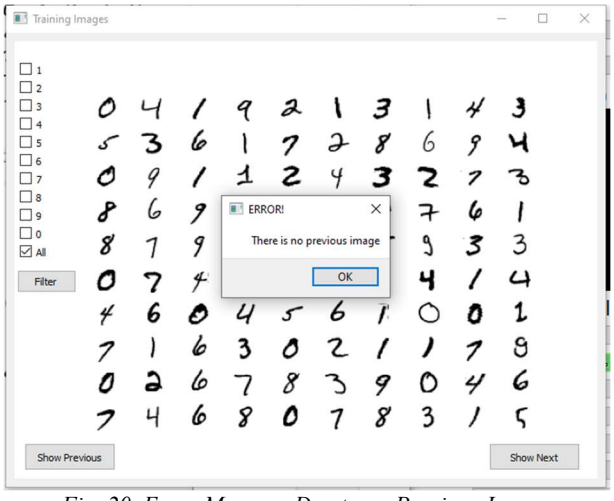

Fig. 20, Error Message Due to no Previous Image

4. Results

4.1. Testing Results Using Own Handwriting

In terms of MNIST images, as mentioned previously on section 3.2, the model showed high accuracy, getting 9930 correct images out of 10,000 testing MNIST images with an average loss of 0.026.

```
Training Model... Epoch size: 19
============================
Test set: Average loss: 0.0260, Accuracy: 9930/10000 (99%)
```

Fig. 21, Training results after 19 epoch, 16 batch size

We have also randomly picked 850 images from the MNIST dataset (some numbers had less than 1000 images) so to make the testing numbers equal, we chose 850 images from each digit and calculated the Precision, Recall and F1 Score. For each digit (box highlighted in red) shown in fig. 22 below, we accrued the predictions that our model made for each digit. Using these numbers, we calculated the precision, recall and F1 score, which showed that our model had some issues with detecting the numbers 7 and 9, shown in the graph of fig. 23. Overall, the model seemed to show good accuracy among all digits.

Predicted Value	0	1	2	3	4	5	6	7	8	9
0	847	0	0	0	0	0	2	1	0	0
1	0	847	0	0	0	0	0	3	0	0
2	1	0	847	0	0	0	0	2	0	0
3	0	0	0	846	0	3	0	0	0	1
4	0	0	0	0	844	0	1	0	0	5
5	0	0	0	5	0	843	1	1	0	0
6	3	2	0	0	1	1	843	0	0	0
7	0	1	3	0	0	0	0	845	0	1
8	0	0	1	0	0	0	0	0	847	2
9	1	0	0	0	2	2	0	1	0	844

Fig. 22 Predicted Value (column), Actual Value (row) of 850 MNIST testing images

Predicted Digit	0	1	2	3	4	5	6	7	8	9
Precision	99,41	99,65	99,53	99,41	99,65	99,29	99,53	99,06	100,00	98,94
Recall	99,65	99,65	99,65	99,53	99,29	99,18	99,18	99,41	99,65	99,29
F1 Score	99,53	99,65	99,59	99,47	99,47	99,23	99,35	99,23	99,82	99,11

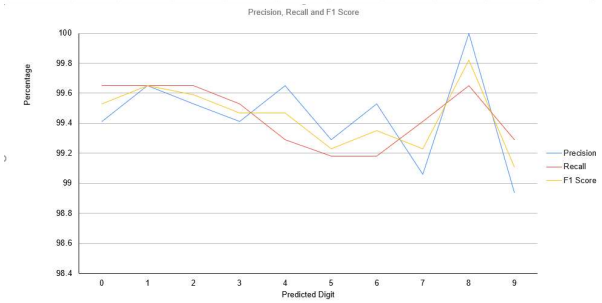

Fig. 23 Graph and Table for precision, recall and F1 score using values from fig.22

As the project's aim was to create a "handwritten" number detection tool, I have tested various handwritten numbers using the GUI and the trained model. Here are some of the results:

```
Predicted Digit = 9
[1.995906e-06, 8.0961345e-06, 6.877941e-05, 0.00062492
216, 0.0012420443, 0.00021869372, 7.281647e-08, 0.0030
117542, 0.00018778148, 0.994636]
Predicted Digit = 3
[0.0070055146, 7.857522e-05, 0.03504632, 0.740756, 4.2
819356e-05, 0.016959274, 0.0011020737, 6.410438e-05, 0
.18694112, 0.012004177]
Predicted Digit = 7
[0.00016587673, 1.4580728e-05, 0.00073694636, 0.001098
1212, 1.0832088e-05, 3.0996194e-05, 3.2582687e-08, 0.7
846524, 0.00023786887, 0.21305229]
```

Fig. 24, My Own Hand Drawn Digits, and the Recognition Results Using the Trained Model (respectively)

While it is not perfect, the model shows that even though it was trained on just the MNIST dataset, it could perform well on previously unseen images (my own handwriting). As shown in fig. 24, my own handwriting was recognized well, and even with confusing and difficult numbers (3 and 8, 9 and 7) the model was able to tell them apart with sizable margin. Overall, me and my teammate were very satisfied with our result, considering that the training was done with a purely MNIST dataset.

5. Conclusions / Future Work

This report has outlined the various aspects of the handwritten digit recognition project. Overall, the project and its results were satisfactory, with the model showing acceptable accuracy in handwritten digits and in MNIST dataset while having an intuitive and easy to use GUI.

In the future, more advancements in the GUI could be done, such as adding more colorful interfaces. Due to time constraints, we were not able to make the GUI pretty, but a more user-friendly and visually appealing interface could be made in the future. In terms of improving the accuracy, having a way to add our own handwriting into the training dataset and training the model could improve the accuracy further. Also, a diverse dataset (including letters) could be used to train the model, which could lead to handwritten letter detection as well in the future. Methods such as data augmentation [9] could be explored and used in the future to expand our dataset and produce maximum results from our model.

6. Acknowledgements

I would like to thank Dr. Ho Seok Ahn, JongYoon Lim and the other TAs for giving guidance and feedback during this project. I would also like to thank my partner, Vishnu Hu for doing the GUI development and getting threading to work with the progress bar.

7. References

[1] Srivastava, N., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *Journal of Machine Learning Research*, June 2014.

[2] Chauhan, R., Ghanshala, K. K., Joshi, R. C., ``Convolutional Neural Network (CNN) for Image Detection and Recognition", *ICSCCC*., December 2018.

[3] Krizhevsky, A., Sutskever, I., Hinton, G. E., “ImageNet Classification with Deep Convolutional Neural Networks”, NeurIIPS., 2012

[4] Dureja, A., “Analysis of Non-Linear Activation Functions for Classification Tasks Using Convolutional Neural Networks”, ResearchGate., October 2018

[5] Jain. A., Sharma, B. K., “Analysis of Activation Functions for Convolutional Neural Network based MNIST Handwritten Character Recognition”, International Journal of Advances Studies of Scientific Research, Volume 3, Issue 9, 2018

[6] LeCun. Yan., Cortes. C., Burges. C., “MNIST Database”, http://yann.lecun.com/exdb/mnist/

[7] Koheler. G.,“MNIST Handwritten Digit Recognition in PyTorch”,https://nextjournal.com/gkoehler/pytorch-mnist

[8] Brownlee. J., “A Gentle Introduction to Dropout for Regularizing Deep Neural Networks”, https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/, December 2018

[9] Ghoneim. S., “5 TensorFlow Techniques to Eliminate Overfitting in DNNs”, https://heartbeat.fritz.ai/5-tensorflow-techniques-to-eliminate-overfitting-in-dnns-281590cc2eb , July 2019

[10] simondele, “CNN with Pytorch for MNIST”, https://www.kaggle.com/sdelecourt/cnn-with-pytorch-for-mnist , 2019