COMPSYS 302 - Python Project

Team 10 - Rachel Nataatmadja

ID: 980468406, UPI: rnat697

Table of Contents

Table of Contents 2

Introduction 3

Planning 3

- Summary of Literature Review 3
- Project Setup 5
- Gantt Chart 5

Design Software Architecture 6

- Purpose of the System 6
- Database and Models Used 7
- System Diagram 8
- Component Explanation 9
  - Dataset class 9
  - LeNet5 and SpinalNet classes 9
  - TrainOrTestModel class 9
  - savedModel class 9
  - Main Menu class - Image Processing functions 9
  - ImageViewerGUI class 9
- Benefit of Architecture 9

Results 10

Conclusion 11

Introduction

This project is to design and implement a user interface and its back-end where users can carry out the machine learning tasks with a dataset of handwritten English characters and digits. I was assigned to the back-end portion of this project, where I had to implement the associated machine learning and dataset tasks and connect them to the front-end. The tasks included downloading the dataset from the internet, format dataset to view, train and test, training and testing the dataset using the AI models LeNet5 or SpinalNet and predicting a character or digit based on what the user has drawn. For this project, I have teamed up with Shou Miyamoto and our team, Team 10, has developed a tool that recognises handwritten digits and English letters using python.

My UPI is rnat697 and my email is rnat697@aucklanduni.ac.nz for further contact

Planning

Summary of Literature Review

The EMNIST dataset is a dataset of handwritten English characters and digits converted into a grayscale 28x28 pixel image and was derived from the NIST Special Database 19[1]. This dataset is an extension to the MNIST dataset and has performed similar image processing. The dataset contains 6 different splits of the dataset, ByClass, ByMerge, Balanced, Letters, Digits and MNIST. Each split contains varying amounts of images ranging from 70,000 to 814,255 images[1].The MNIST dataset is a dataset of handwritten digits images with a grayscale 28x28 pixel format. Since EMNIST was recently developed in 2017, there isn't a large amount of research when compared to its predecessor, MNIST, which was introduced in 1998[2]. However, EMNIST is still being used researchers to recognise handwritten English characters and digits because of its similarity to MNIST and allows them to give neural networks a more challenging benchmark[2]. In our project, we'll be using EMNIST because we want to recognise both English characters and digits. Further explanation of this will be under the Database and Models Used section.

Our project has a feature that allows users to draw a character or digit on a drawing canvas and send it for prediction with a saved trained model. I would need to utilise the image

[1] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373
[2] Baldominos, A., Saez, Y., & Isasi, P. (2019). A Survey of Handwritten Character Recognition with MNIST and EMNIST. Applied Sciences, 9(15), 3169. https://doi.org/10.3390/app9153169

conversion process that is referred to in the “EMNIST: an extension of MNIST to handwritten letters” paper.

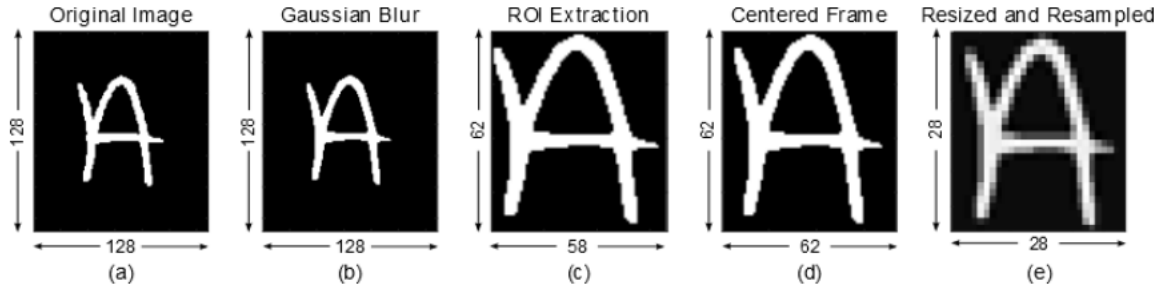


Figure 1: A diagram of image conversion process used in EMNIST dataset[1]

According to Cohen et al, I would need to convert the original image down to 128x128 pixels shown in (a). In section (b), the edges of the image need to be softened so I need to apply a Gaussian filter with σ = 1. Section (c) indicates I need to extract the Region of Interest (ROI) of the image and apply a 2 pixel padding. This allows the character or digit to fill up the entire image. Then the image is centred in section (d) to create the square aspect ratio. Finally, the image is downsized to 28x28 pixels. Throughout this conversion, I assumed the original image would need to be in a similar colour scale as the EMNIST dataset images. I would apply grayscale on to the image so that it would match the a black background and white drawing of the EMNIST images.

Convolution Neural Networks (CNN) are neural networks that are primarily used to classify and recognise images. CNNs will be particularly useful for our project because we need to recognise images of handwritten English characters and digits compared to other neural networks such as Recurrent Neural Networks (RNN). RNN is more useful for time related data such as speech detection.

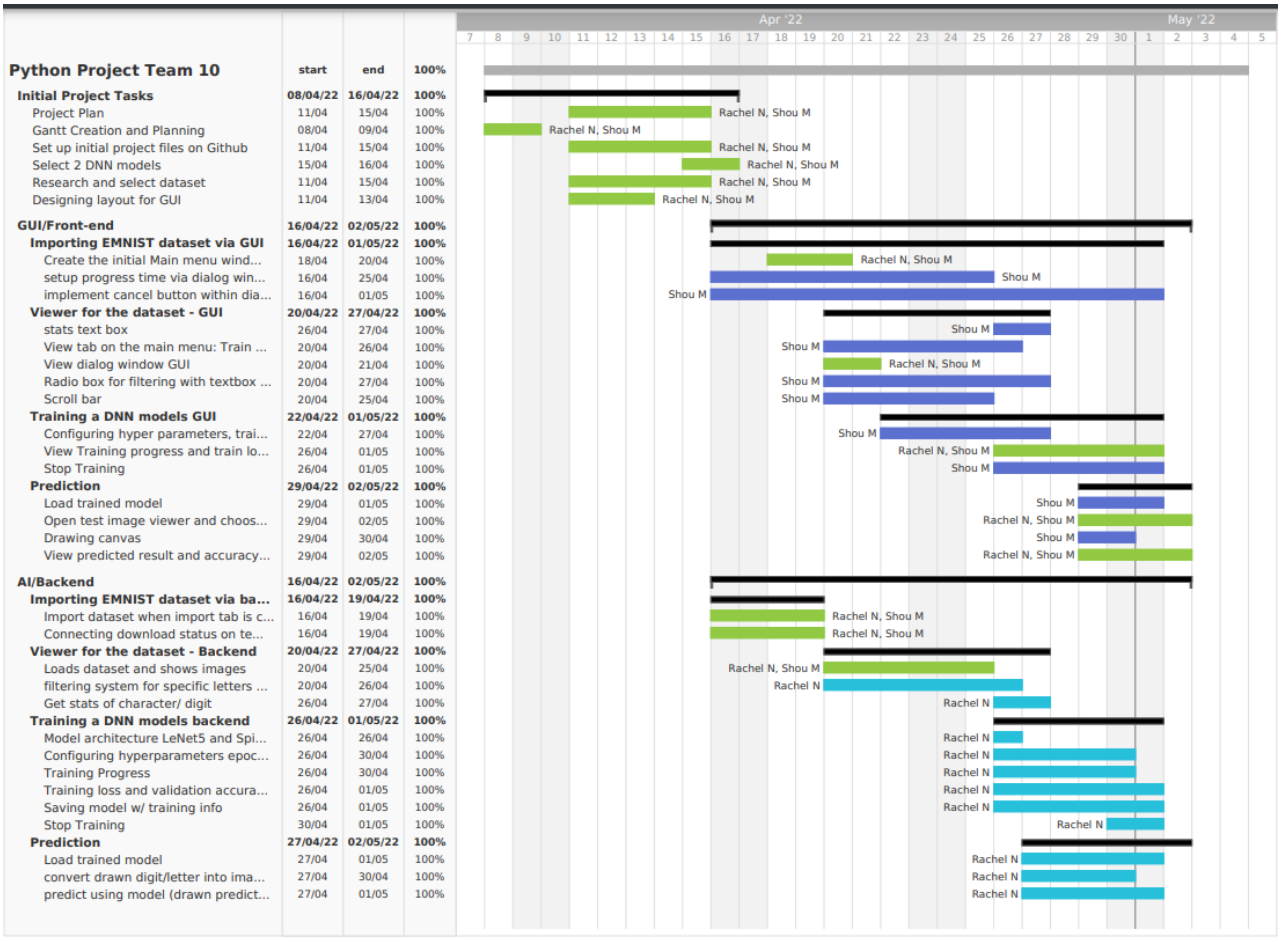
CNN models that can be used for this project are LeNet, AlexNet, VGGNet, GoogleNet, CapsNet, VGG-5 with SpinalNet and SpinalNet CNN. One paper found that out of AlexNet, VGGNet, GoogleNet and CapsNet, CapsNet was the best overall performer with an accuracy of 99.7% when trained with the entire balanced EMNIST dataset[3]. AlexNet was also considered to be a good baseline for the other three models with 83.3% accuracy when trained on 50% of the EMNIST dataset[3]. Anuradha et al also took note that VGGNet takes a long time to train which resulted in the lowest accuracy of 81.5% with 50% of the dataset. On another paper, VGG-5 with a SpinalNet as the fully connected layer has the highest average accuracy of 90.73% when using the balanced EMNIST dataset[4]. While SpinalNet CNN has an average accuracy of 82.57% when using the same dataset[4]. Further explanation of which models I selected will be in Database and Models Used section.

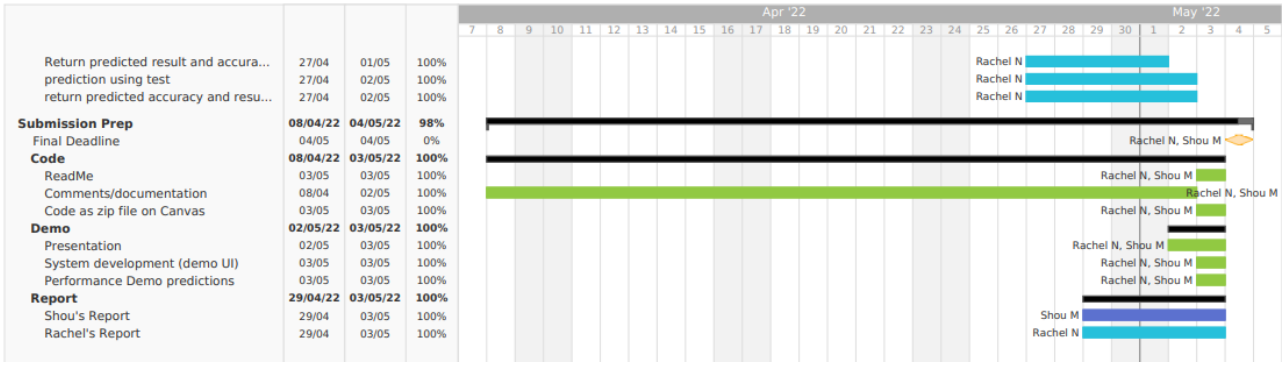
[3] R. Anuradha, N. Saranya, M. Priyadharsini and G. D. Kumar, "Assessment of Extended MNIST (EMNIST) dataset using Capsule Networks," 2019 International Conference on Intelligent Sustainable Systems (ICISS), 2019, pp. 263-266, doi: 10.1109/ISS1.2019.8908006.
[4] H M Dipu Kabir, Moloud Abdar. Seyed Mohammad Jafar Jalali. Abbas Khosravi. Amir F Atiya. Saeid Nahavandi.; Dipti Srinivasan. (2022) “SpinalNet: Deep Neural Network with Gradual Input,” https://arxiv.org/pdf/2007.03347.pdf

Project Setup

We have used Anaconda with Python 3.8 as the environment of our project. Our required libraries that need to be installed with our program are scikit-image, PyQt5, PyTorch and torchvision. We used scikit-image library to help with manipulating the image drawn by the user into a suitable format for the machine learning models. The PyQt5 library which we use to create our Graphical User Interface (GUI). The PyTorch and Torchvision libraries are used to create the machine learning framework. Instructions for installation of this environment and any packages that are required to run our project are in the README file on our Github repository. We also used Visual Studio Code to develop our project. Our university course, COMPSYS 302, recommends us to use Anaconda and Visual Studio Code during our development.

Gantt Chart





To see the pdf version of the chart click here.

Design Software Architecture

Purpose of the System

Artificial intelligence is being increasingly ingrained into our daily lives, especially during the pandemic where researchers were helped by AI to quickly develop COVID-19 treatments[5]. As more AI tools and models are being adopted by companies, it has been more apparent that machine learning will become a core technical skill.

Since the pandemic, the world has become more reliant on technology. I've seen most students would opt to use an Ipad or tablet to write their notes over paper notes. When using digital handwritten note taking, there are technologies that use AI to recognise handwritten words on a tablet and convert it to text. I've found this application of machine learning to recognise handwritten characters and digits to be interesting. For this project, we have developed a simple handwritten character and digit recogniser tool.

Some uses for using machine learning to recognise handwritten character and digit are:

- Patient prescriptions digitization in healthcare[6]
- Preservation of historical texts and let those texts be able to be identified[6]
- Handwriting to text on a touchscreen to increase productivity

Our goal for this project is to be able to correctly recognise handwritten characters and digits based on given test images and when the user has drawn a character or digit. It is also our goal to become familiar with software design, GUI programming and basic machine learning skills while developing this project.

---

[5] Enterprise.nxt staff. (2021). The rise of artificial intelligence and machine learning. Hewlett Packard Enterprise.https://www.hpe.com/us/en/insights/articles/the-rise-of-artificial-intelligence-and-machine-learning-2108.html#:~:text=From%20advancing%20medicine%20to%20optimizing,artificial%20intelligence%20and%20machine%20learning.

[6] Matcha,ACN. (2022).How to easily do Handwriting Recognition using Machine Learning. NanoNets https://nanonets.com/blog/handwritten-character-recognition/

Database and Models Used

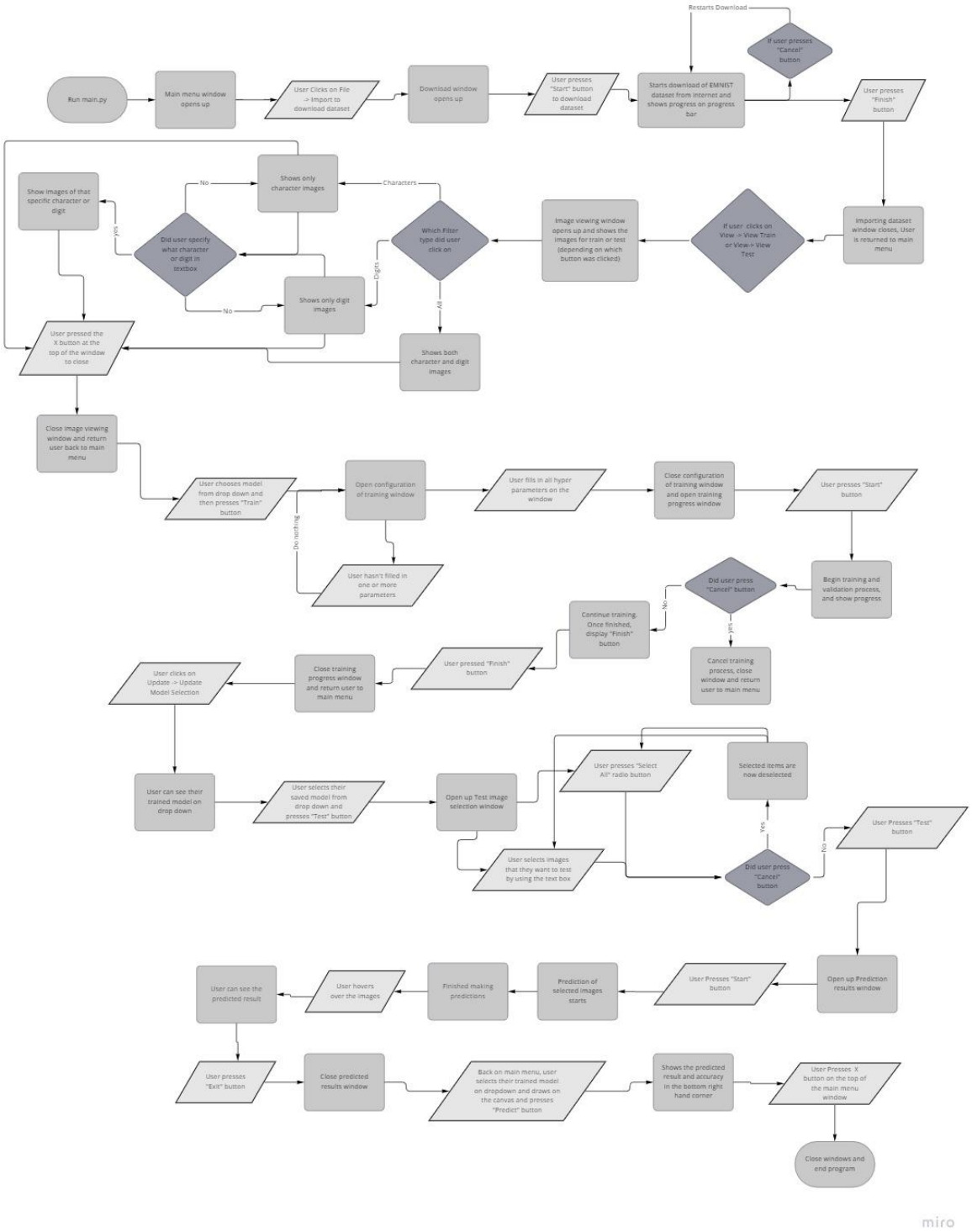
As per my section of the Summary of Literature Review, I used the EMNIST dataset for this project from this link: https://www.nist.gov/itl/products-and-services/emnist-dataset . I decided to use the existing EMNIST ByClass dataset split because it contained all digits from digits from 0 to 9 as well as uppercase and lowercase letters from a to z. This means there are 62 classes for the model to classify and 814,255 dataset images. I assumed that our project would need to classify 0 to 9, A to Z and a to z individually, hence I used the ByClass split instead of all the other splits.

Initially, we were going to use AlexNet and VGG for our CNN models based on the statistics stated in the Summary of Literature Review section. However, when I implemented the model architecture for both AlexNet and VGG, they both took a lot more computing power than I expected. It also took a long time to train the EMNIST dataset with a train ratio of 70%, epoch number of 5 and a batch size of 130. Due to this, I selected two simpler models, LeNet5 and SpinalNet. Both these models are able to train under the same conditions in significantly less time than AlexNet and VGG.

According to the "SpinalNet: Deep Neural Network with Gradual Input", the model architecture is proposed to split each layer into three splits, the input split, the intermediate split and the output split[4]. By doing so, it lowers the number of incoming weights compared to traditional DNNs which can reduce errors with lower computational costs[4].
The architecture of LeNet5 has 7 layers, 3 of them are convolutional layers, 2 are subsampling layers and the last 2 are fully connected layers[7].

[7] Richmond Alake. (2020). "Understanding and Implementing LeNet-5 CNN Architecture (Deep Learning)." Towards Data Science https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342

System Diagram



Here's the link to the pdf version or the miro board.
We also created a GUI layout on Google Slides for reference while developing this project.

Component Explanation

Here are some explanations for some of the components I have worked on.

Dataset class

This class handles the download of the EMNIST dataset and data loading of train, validate and test sets. There is also a function that converts the test and train sets into numpy arrays to be able to display on a pixel map on the image viewing GUI.

LeNet5 and SpinalNet classes

These classes house their respective model architectures. LeNet5's model architecture was modified from here and SpinalNet's model architecture was modified from here.

TrainOrTestModel class

This class handles the training, testing and prediction of the CNN models. The training function trains the model with the train and validate datasets. The testing function predicts the results of an array of selected images from the test set using a previously trained model. While in the prediction function, the previously trained model is also used but it is only predicting the result of a single image of the drawing that the user has drawn on the canvas.

savedModel class

This class stores the trained model's associated values such as the model used to train (either LeNet5 or SpinalNet), the epoch number, the name of the trained model, the train set ratio and the file path of the trained model .pt file. The functions in this class are used to extract the values when it is needed for the testing or predicting with the trained model.

Main Menu class - Image Processing functions

Inside the main menu class, there are functions for image processing for processing the image of the drawn character or digit made by the user. This image processing follows the steps indicated in Summary of Literature Review so that the image is in a grayscale format of 28x28 pixels.

ImageViewerGUI class

Inside this class, I created the filter for showing the images of only characters, only digits and only a specific character or digit. I also helped out with trying to load the images on the GUI.

Benefit of Architecture

The benefit of my architecture is the modularity of the classes. By breaking down each feature to its own class, it makes it easier to connect everything together. Testing each class on its own before integrating it to the whole software also makes error checking easier. Teamwork is much more efficient when we assign classes for each of us to do.

Results

In our tool, we can train and test CNN models, LeNet5 and SpinalNet. The models can be trained with the custom hyper parameters, the percentage of the train data to be used for training (the remaining percentage will be used to create the validation data), the batch size of the dataset, the epoch number used to iterate and a name for the trained model. Using the trained model, users can select multiple images from the test dataset and use it for prediction. This will show the predicted results of those images. Users can also use the trained model to predict their own drawn character or digit and see the result.

When SpinalNet is trained and validated with a 70% train ratio, 130 batch size and 4 epochs, it has an average loss of 1.01 and an accuracy of 78.29%.

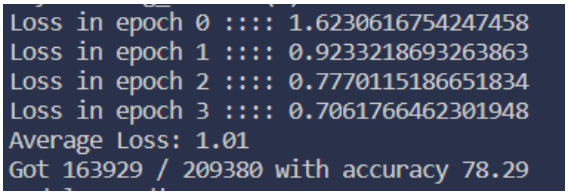


Figure 2a: SpinalNet Training and Validation Statistic

When LeNet5 is trained and validated for the same hyperparameters, it has an average loss of 0.87 and an accuracy of 81.92%.

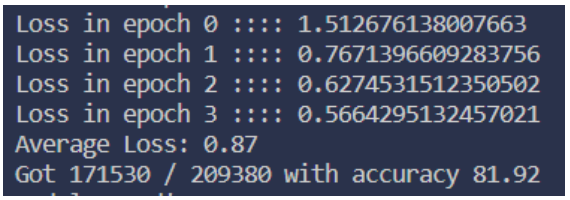


Figure 2b: LeNet5 Training and Validation Statistic

Recognition using the trained SpinalNet (70% train ratio, 130 batch size and 4 epochs) reached an accuracy of 78.48% with 10,000 images.

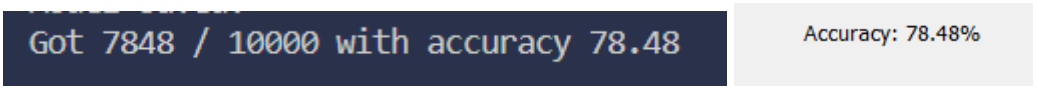


Figure 3a: Trained SpinalNet testing with 10,000 test images shown on terminal (left) and GUI (right)

Recognition using the trained LeNet5 with the same hyper parameters reached an accuracy of 81.66% with 10,000 images.

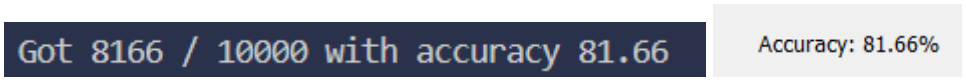


Figure 3b: Trained LeNet5 tested with 10,000 test images shown on terminal (left) and GUI (right)

With a slightly larger epoch, I estimate around 7 to 10 epochs, the accuracy could increase a bit more. However, we shouldn't use an extremely large number of epochs such as 15 or more epochs as it might end up overfitting. It will also take a long time for it to train with a large amount of epochs.

Conclusion

This project was challenging but it was an interesting experience. Our tool is able to recognise handwritten English characters and digits at least 78% of the time with hyperparameters of 70% train ratio, 130 batch size and 4 epochs. The most challenging part of the project was trying to find and understand documentation of the EMNIST dataset and various libraries such as PyTorch, Torchvision and scikit-image. It was difficult to know what part of the libraries would be useful for this project. Since Shou and I worked on the viewing images class together, we both found it difficult to optimise our image viewer to be able to load all of the images in the EMNISt dataset.

Next time, I would like to implement more complex CNNs such as AlexNet and VGG if I have better processing power or learn how to optimise it better. I would also have liked to implement a smoother way to view all 697,932 train images and 116,323 test images on the GUI by dynamically loading them. I also think that image processing functions could be in its own separate class to further support the modularity of our classes.

Here's a link to our video demo:
https://webdropoff.auckland.ac.nz/cgi-bin/pickup/c5847b63c98c98f8264839fa8e23cf37/427737