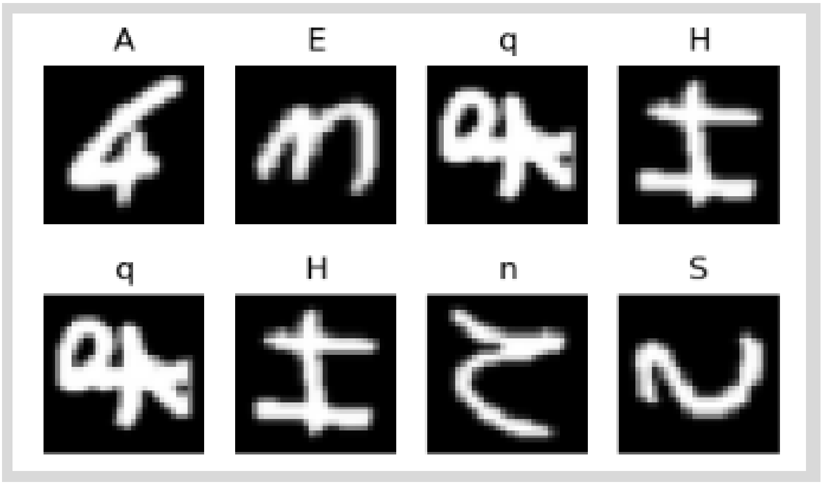
IMPLEMENTING IMAGE CLASSIFICATION ALGORITHMS

By Chiamaka Jibuaku (201670698) and Frank Enendu (201607044)

Introduction of the dataset

The dataset used for this project is the EMNIST (Extended MNIST) dataset. The EMNIST Balanced dataset is a collection of handwritten character images developed as an extension of the MNIST dataset. It includes over 131,000 images of handwritten characters, including uppercase and lowercase letters, digits, and some special characters, divided into 6 different splits for training and testing purposes. This dataset is balanced, meaning that each class has an equal number of samples, which helps to prevent bias in model training. The EMNIST Balanced dataset is available in PyTorch and can be used for tasks such as character recognition, OCR, and handwriting recognition.

Figure 1: Image visualization of the EMNIST dataset



We implemented a function, **plot_samples** to visualize the EMNIST dataset in PyTorch. The function takes a PyTorch dataset object containing images and labels, and a list of label names corresponding to the labels in the dataset. It creates a 2x2 grid of subplots and displays some sample images with their corresponding label names as the title in grayscale. The pixel values of the image are normalized to be between -1 and 1 before being displayed. The function is helpful in understanding the dataset and checking if the images are loaded correctly before training a model on them.

Structure of Multi Layer Perceptron Implemented

We developed the Multilayer Perceptron (MLP), a feed-forward neural network, utilising the PyTorch libraries for implementing MLP for image classification. The MLP is a neural network model used for image classification tasks. The architecture of the MLP consists of 3 fully connected layers, where each layer is connected to the previous layer. The number of nodes in each hidden layer is defined by the 256 hidden sizes, and the number of output classes is defined by the num_classes parameter which is 47. The activation parameter specifies the activation function used for each layer, which can be 'relu', 'sigmoid', or 'tanh'. The batch_norm parameter determines whether batch normalization is applied after each fully connected layer, and the dropout parameter sets the dropout rate to be applied after each fully connected layer. The regularization parameter specifies the regularization strength to prevent overfitting. The MLP model is implemented as a PyTorch module and takes an input tensor of shape (batch_size=250, input_size=28*28). The output tensor has a shape of (batch_size=250, num_classes=47).

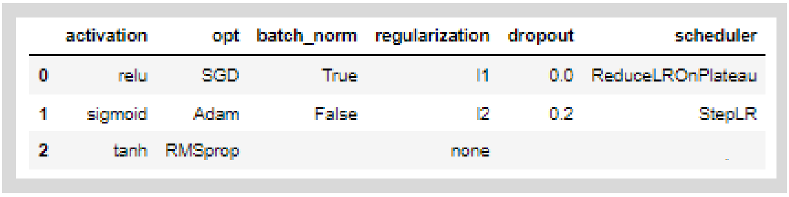
Structure of Convolutional Neural Network Implemented

We built the Convolutional Neural Network (CNN), a feed-forward neural network using the nn.module imported from the Pytorch library. The above code shows the implementation of a Convolutional Neural Network (CNN) for image classification. The architecture consists of two convolutional layers followed by two fully connected layers. The input is an image with a size of input_size, and the output is a classification into num_classes categories. The activation function, batch normalization, dropout, and regularization can be specified by the user. The first convolutional layer has 1 input channel, 6 output channels, a kernel size of 5, a stride of 1, and a padding of 2. The second convolutional layer has 6 input channels, 16 output channels, a kernel size of 5, a stride of 1, and no padding. The first fully connected layer has 120 output features, the second has 84 output features, and the last layer has 47 output features. The architecture uses max-pooling with a kernel size of 2 and stride of 2 after each convolutional layer. The ReLU, sigmoid, or hyperbolic tangent (tanh) activation functions can be used. Batch normalization is applied after each convolutional layer if batch_norm is set to True. Dropout regularization can be applied to the second fully connected layer if dropout is set to a value between 0 and 1. L1 and L2 regularization can be applied to the model if regularization is set to True.

Model Design Rationale for MLP

***Kind of hyperparameters or techniques do you choose to tune and explore*:** To find the optimal configuration for our model, we experimented with six different techniques. We varied three activation functions ('relu', 'sigmoid', or 'tanh') and three optimizers ('SGD', 'Adam', or 'RMSprop') to update the weights during training. Additionally, we tested the impact of applying batch normalization in the hidden layer, with the option to set the 'batch_norm' hyperparameter to either True or False. To further fine-tune the model, we explored the effects of regularization, which can be set to 'l1' with a coefficient of 0.001 (L1 regularization), 'l2' with a coefficient of 0.01 (L2 regularization), or 'none' with a coefficient of 0.00 (no regularization). We also tested two different dropout rates (0.0 and 0.2), which control the probability of dropping out a neuron in the hidden layer during training. Finally, we used a learning rate of 0.01 and compared two different learning rate schedulers ('ReduceLROnPlateau' and 'StepLR') for the 'scheduler' hyperparameter.

Table 1: Hyperparameters explored for the MLP



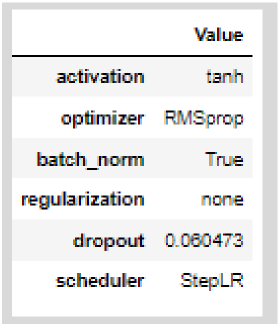
	activation	opt	batch_norm	regularization	dropout	scheduler
0	relu	SGD	True	l1	0.0	ReduceLROnPlateau
1	sigmoid	Adam	False	l2	0.2	StepLR
2	tanh	RMSprop		none		

We implemented a random search function to randomly combine hyperparameters for the multilayer perceptron model. It defines a range of values for each hyperparameter and loops through the number of trials specified, sampling a set of hyperparameters at random for each trial. For each set of hyperparameters, it tests the model's accuracy using a specified dataset and stores the highest accuracy achieved and the corresponding hyperparameters. The function returns the best hyperparameters, the highest accuracy achieved, and the total training time in seconds. This approach was useful in identifying the best combination of hyperparameters for a model without having to exhaustively search through all possible combinations. You can also choose the amount of trials and this optimizes the computing time.

***Reason for choice of hyperparameters or techniques*:** These set of hyperparameters as seen above were chosen over other numerious hyperparameters based on their effectiveness for the MLPs models, such as the use of ReLU, sigmoid, and tanh activation functions for non-linear transformations and mitigating the vanishing gradient problem, SGD, Adam, and RMSprop optimizers for faster convergence and handling noisy data, and 'ReduceLROnPlateau' and 'StepLR' adaptive learning rate schedulers for dynamically adjusting the learning rate to help models converge faster and avoid getting stuck in local minima.

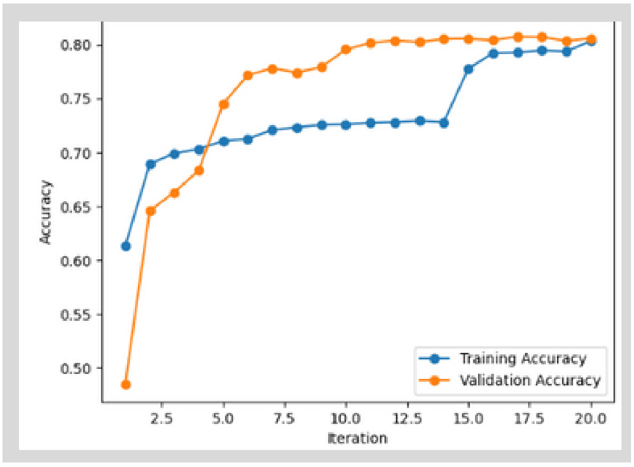
***How technique affects model's performance and why this technique boosts or decrease the performance of the models*:** The performance of a machine learning model is influenced by various techniques, including hyperparameter tuning. In this case, a set of hyperparameters used was able to improve the accuracy which was initially 86.6% for our base model to 89.5%. Using our random search function we implemented, we found the best hyperparameter combination as seen in the dataframe. This combination produced the highest accuracy after 3 trials among all the tested combinations. It is possible to obtain a better combination if the trial time is increased, but for lesser computation time, we selected 3

Table 2: Best Hyperparameter combination for MLP



	Value
activation	tanh
optimizer	RMSprop
batch_norm	True
regularization	none
dropout	0.060473
scheduler	StepLR

Figure 2: Elbow Graph of Training and Validation Accuracy



Reporting overfitting or underfitting issues and how I deal with it? One of the most critical optimization challenges in building a model is avoiding overfitting. Therefore, to create a model capable of generalizing well on external data, we divided our training dataset into training and validation sets. To determine the optimal number of epochs for our model, we implemented early stopping technique using the elbow graph method. At the beginning of the training process, both the training and validation accuracies increased steadily, which was an indication that the model was learning from the data. However, after a certain point (specifically, at epoch 20), the validation accuracy stopped increasing even though the training accuracy continued to increase.

Model Evaluation and Result MLP

Explaining the training loss, testing loss and testing accuracy of MLP: The below graphs show the trend of the test accuracy, and the the training accuracies for the best hyperparameter combination over 20 epochs. The growth rate of the test and train accuracies are similar, growing rapidly for fewer epochs but tends to reduce from arround epoch 15. The asumption is that the accuracy will stabilise at some point.

Figure 3: Test Accuracy/Iteration Plot for MLP

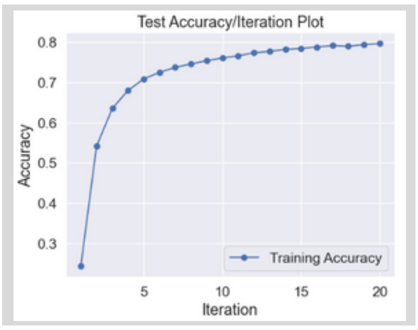


Figure 4: Train Accuracy/Iteration Plot for MLP

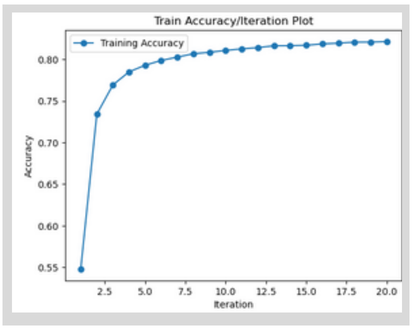
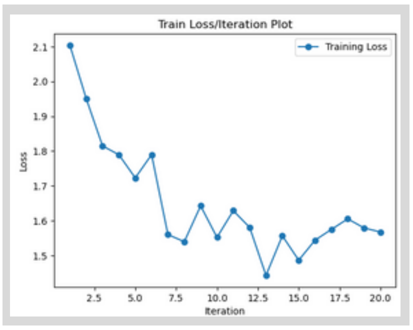
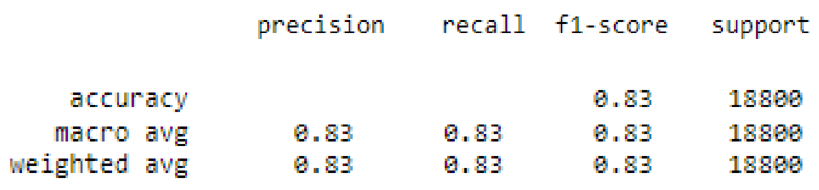


Figure 5: Train Loss/Iteration Plot for MLP



Describing and explaining the predicted results of your models.: The MLP model perform very well on the test dataset at 83% average precision, recall and F1-score respectively for all the class labels Also looking at the first 10 prediction, we can see that our model performed well, correctly predicting the true label of the images.

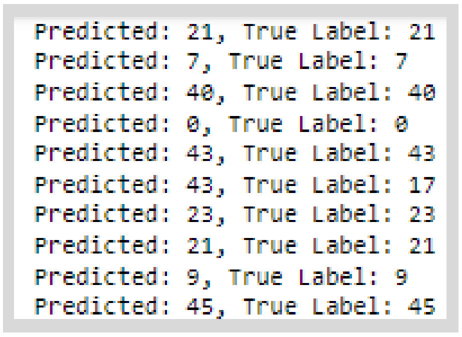
Table 3 : Evaluation metris for MLP performance



```
              precision    recall  f1-score   support

    accuracy                           0.83     18800
   macro avg       0.83      0.83      0.83     18800
weighted avg       0.83      0.83      0.83     18800
```

Figure 7: Predicted results of MLP and True Labels of the Images



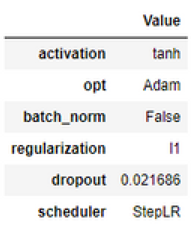
```
Predicted: 21, True Label: 21
Predicted: 7, True Label: 7
Predicted: 40, True Label: 40
Predicted: 0, True Label: 0
Predicted: 43, True Label: 43
Predicted: 43, True Label: 17
Predicted: 23, True Label: 23
Predicted: 21, True Label: 21
Predicted: 9, True Label: 9
Predicted: 45, True Label: 45
```

Model Design Rationale CNN

Kind of hyperparameters or techniques do you choose to tune and explore: Just like we implemented for the Multilayer perceptron algorithm, we explored the hyperparameters as seen in *table* 1 above.

Reason for choice of hyperparameters or techniques: The hyperparameters were selected based on their ability to improve the model's performance just like for the case of MLP. For CNN but not peculiar, the activation functions introduce non-linearity or thresholds into the model. Batch normalization normalizes the inputs and hidden layers and helped to improve the performance of the model. The dropout rate was employed as a regularization method so as to eliminate unit layers at random during training to avoid overfitting. The regularisation parameter which was L1 or L2 was employed as it applies a penalty term to the loss function, thereby stopping overfitting. These penalty terms can aid in lowering the weights' magnitude and preventing the model from overfitting the training set of data. The optimizers were chosen as it helps to adjust the weights and biases of the CNN model, so as to reduce the values of the loss function. Finally, in order to increase convergence and avoid overfitting, we added the learning rate schedulers which help to modify the optimizer's learning rate during training.

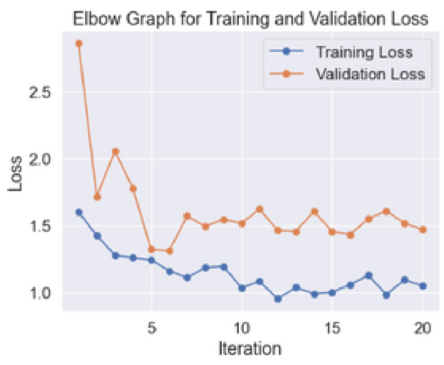
How technique affects model's performance and why this technique boosts or decrease the performance of the models: In the case of our CNN model, thesame set of hyperparameters used for the MLP model was used but this time arround was not able to improve the accuracy which reduced from its initially 87.7% for our base model to 81.8%. For the CNN, there's need to increase the amount of random search trials as the 3 random trial used could not produce the best set of hyperparameters.



	Value
activation	tanh
opt	Adam
batch_norm	False
regularization	l1
dropout	0.021686
scheduler	StepLR

Table 4: Best Hyperparameter for CNN

Figure 8: Elbow Graph for Training and Validation Loss for CNN



Reporting overfitting or underfitting issues and how I deal with it? Just like we implemented for the MLP model, we divided our training dataset into training and validation sets. To determine the optimal number of epochs for our model, we implemented early stopping technique using the elbow graph method, this time we employed the model loss. Here the trnd in the graph was not able to help dertermine early stopping as the training and validation loss has similarly consitenet trend. This however shows that as at the 20th epoch, our model was still able to generalise, therefore does not overfit or underfit.

Model Evaluation and Result CNN

Explaining the training loss, testing loss and testing accuracy of CNNs.: The below graphs show the trend of the test accuracy, and the the training accuracies for the best hyperparameter combination over 20 epochs. The growth rate of the test and train accuracies are similar, growing rapidly for fewer epochs but tends to reduce from arround epoch 15. The asumption is that the accuracy will stabilise at some point.

Figure 9: Test Accuracy/Iteration Plot for CNN

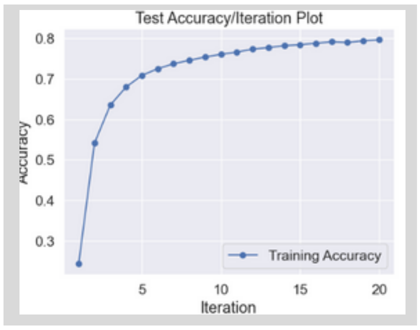


Figure 10: Train Accuracy/Iteration Plot for CNN

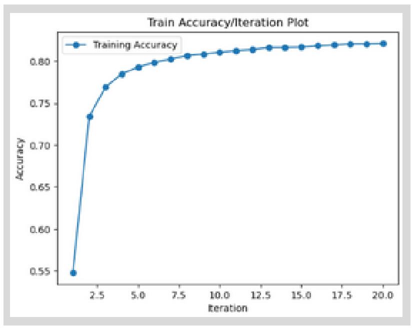
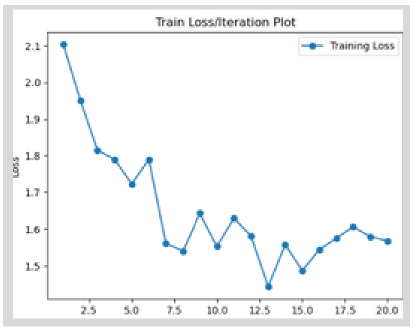
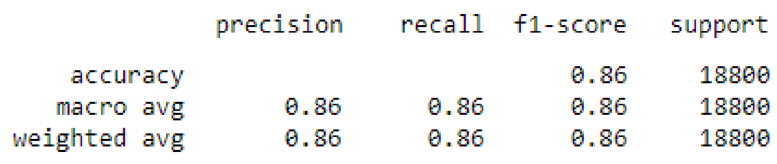


Figure 11: Train Loss/Iteration Plot for CNN



Describing and explaining the predicted results of your models.: The CNN model perform very well on the test dataset at 86% average precision, recall, and F1-score respectively for all the class labels. Also looking at the first 10 predictions, we can see that our model performed well correctly predicting the true label.

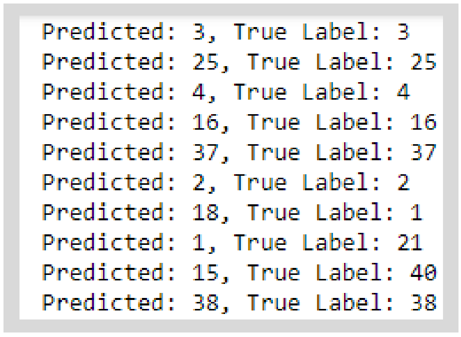
Figure 12: Evauation metrics for the CNN model performance



```
              precision    recall  f1-score   support

    accuracy                           0.86     18800
   macro avg       0.86      0.86      0.86     18800
weighted avg       0.86      0.86      0.86     18800
```

Figure 13: CNN predictions and their True Labels



```
Predicted: 3, True Label: 3
Predicted: 25, True Label: 25
Predicted: 4, True Label: 4
Predicted: 16, True Label: 16
Predicted: 37, True Label: 37
Predicted: 2, True Label: 2
Predicted: 18, True Label: 1
Predicted: 1, True Label: 21
Predicted: 15, True Label: 40
Predicted: 38, True Label: 38
```

Comparing the Performance of the two Model

***Analysis of the performance of MLP and CNNs based on my own AI knowledge*:**

Our results suggest that the CNN model performed better than the MLP on the EMNIST dataset for image classification with a training accuracy of 87.7% before hyperparameter tuning for CNN and 86.6% accuracy obtained using MLP before hyperparameter tuning. However, after hyperparameter adjustment using the best hyperparameters combination with a number of 3 random trials, which included tanh activation function, RMSprop optimizer, and StepLR as the scheduler rate, with no regularization applied, and drop out rate of 0.2 and batch-normalization were added to the hidden layers, the MLP model outperformed the CNN in terms of training accuracy. The training accuracy of the MLP increased from 86.6% to 89.5% whereas the CNN model training accuracy decreased from 87.7% to 81.8% after hyperparameter tuning. This decrease in the training accuracy of the CNN may be due to several reasons including the fact that the hyperparameter tuning for a number of 3 random trials we employed may have been too small for the CNN to converge as well as to learn the structure of the dataset. Based on our AI knowledge, MLP is a simpler neural network than CNN which is easy to train and also converges faster, although it is not able to capture complex spatial features like a CNN model, thus CNNs are often recognized as one of the best models to perform well on images classification. For the benefit of our project, we employed a number of 3 random trials during hyperparameter tuning to get the best hyperparameter combination and also 20 (iterations) epochs for our models in order to reduce the training time of our models. This may have affected the training accuracy of the CNN, thus more analysis is needed to understand the cause of the decrease in CNN's training accuracy since hyperparameter tuning is generally used to optimize the performance of a neural network model. However, when both models were tested on the test dataset, the CNN achieved an average of 86% test accuracy across all class labels, whereas the MLP achieved an average of 83% accuracy across all class labels, which is the same for other model performance evaluation metrics such as precision, recall, and F1-score. This shows that, when compared to the MLP model, the CNN model was more accurate at being generalized to be able to correctly predict the label of new or unseen images, although both models correctly determined the first 10 actual labels of the images.

***The pros and cons of MLP and CNNs and analyse why a certain model is better than another model*:**

As we have already seen from our project, MLP, and CNNs are both widely used neural network models used for varieties of machine learning problems including image predictions. However, the two networks have their respective pros and cons which make them better than each other depending on a specified context. The pros of MLP are that it is a simple network and thus easy to build and faster to train than the CNN model, as seen from the training times obtained from the two models implemented in our project, MLP had a training time of 2578 seconds (approximately 43 minutes) for 20 epochs while the training time of CNN model was 3710 seconds (approximately 62 minutes) for 20 epochs. MLP models can also be used for various problems such as classification, regression, and clustering, unlike CNN which is mostly used for image classification. Also, MLP can be used with both structured and unstructured data, and due to its simple nature, it is generally less difficult to interpret and understand. Despite these pros, the cons of MLP include that; it does not perform very well on complex images or signal processing problems. Moreover, in order to avoid overfitting which usually affects the model, it requires a large amount of training data and also may require various hyperparameters to be tuned in order to obtain good performance with MLP.

However, the pros of CNN include that; it has an excellent performance on image prediction and signal processing problems generally due to the fact that it is able to learn spatial features in images including very complex images and large datasets with high accuracy as we saw from the results of our project. Unlike MLP, CNN is less prone to overfitting. Although CNN is widely used in machine learning because of its various prons, it also has its cons, which include; that CNN is very expensive to compute, and requires a large amount of training data and time including other sophisticated hardware or resources for training the model unlike for MLP models. Due to its complexity, CNN models can be difficult to build, understand and interpret as well.

In general, deciding which of the two models is better is determined by the machine learning problems to solve and the context for instance whether there is resource constrained or not, etc. CNN models are better than MLPs for image prediction and signal processing due to their capacity to learn spatial characteristics. On the other hand, for text classification problems, MLP models are better than CNNs. Also where resources are constrained, MLPs models may be a better choice.

Conclusion

***Reflection and Project Next Step*:**

Our project satisfies the aim of this assignment which is to use neural network models to correctly predict images using the EMNIST dataset. We learnt how to build MLP neural network of at least three hidden layers and a CNN model of at least two convolutional layers. We also evaluated the importance of hyperparameters tuning in improving model performance and learnt how to tune and explore different hyperparameters for improving the performance of a neural network model, ranging from important hyperparameters such as activation functions, optimizers, learning rate, learning rate schedulers, batch normalization, regularization parameters, and dropout rates and found the best hyperparameter combination that produces the best accuracy for the MLP and CNN models. Although we were not able to try different hyperparameter combinations as due to limited time we only did for only 3 trials of hyperparameter combinations using random search, and for an epoch (iterations) of 20 for training the two models, however, we noticed the difference in the performance of the two models for image classification and found that CNN is better than MLP having an accuracy of 86% which is the same for all evaluation metrics - precision, recall and F1 score when tested on the test dataset. This corresponds to the already existing domain knowledge that CNN is a better model for image prediction than MLP.

Considering what we could improve next time, we hope to further increase the number of epochs for training the CNNs and the number of hyperparameter combination trials for the hyperparameter tests for the two models in order to improve the accuracy of the models and performance. Especially for the CNN model since CNN takes more time to converge, We hope to devote more time to training the models and other computation capacities. We also hope to explore other methods of hyperparameter exploration and tuning to improve the performance of the models and get even better accuracy. Additionally, we hope to explore other more complex datasets to train the two models and also test the models for generalization.