# CSE 574 Introduction to Machine Learning

# Assignment 2

# Building Neural Networks and CNNs

## Part I: Building a Basic NN

1. **Provide a brief overview of your dataset (e.g. type of data, number of samples, and features. Include key statistics (e.g. mean, standard deviation, number of missing values for each feature).**

The dataset which is used for the analysis contains a total of **766 samples** and **8 features** labelled from f1 to f7, along with a target variable. This target variable is a binary class variable, having two categories: 0 and 1. Some of the key statistics of the dataset are as follows:

```
Main Statistics of the dataset:
                f3      target
count  766.000000  766.000000
mean    69.118799    0.349869
std     19.376901    0.477240
min      0.000000    0.000000
25%     62.500000    0.000000
50%     72.000000    0.000000
75%     80.000000    1.000000
max    122.000000    1.000000
```

The mean value of f3 is 69.12 with a standard deviation of 19.38.
The target variable has a mean value of 0.35 and a standard deviation of 0.48

Furthermore, df.info() did report the following output, from which we can understand that f3 and the target variable are numeric (int64), and the rest of the features, namely f1, f2, f4, f5, f6, and f7 are of type object. Therefore, they are not numeric but can take some mixture of types and would need further

processing:

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   f1      766 non-null    object
 1   f2      766 non-null    object
 2   f3      766 non-null    int64
 3   f4      766 non-null    object
 4   f5      766 non-null    object
 5   f6      766 non-null    object
 6   f7      766 non-null    object
 7   target  766 non-null    int64
dtypes: int64(2), object(6)
```

**Missing Values:** There is no missing values in this dataset as it was shown by the following output of the df.isnull().sum():

```
Missing Values Count:
f1        0
f2        0
f3        0
f4        0
f5        0
f6        0
f7        0
target    0
dtype: int64
```

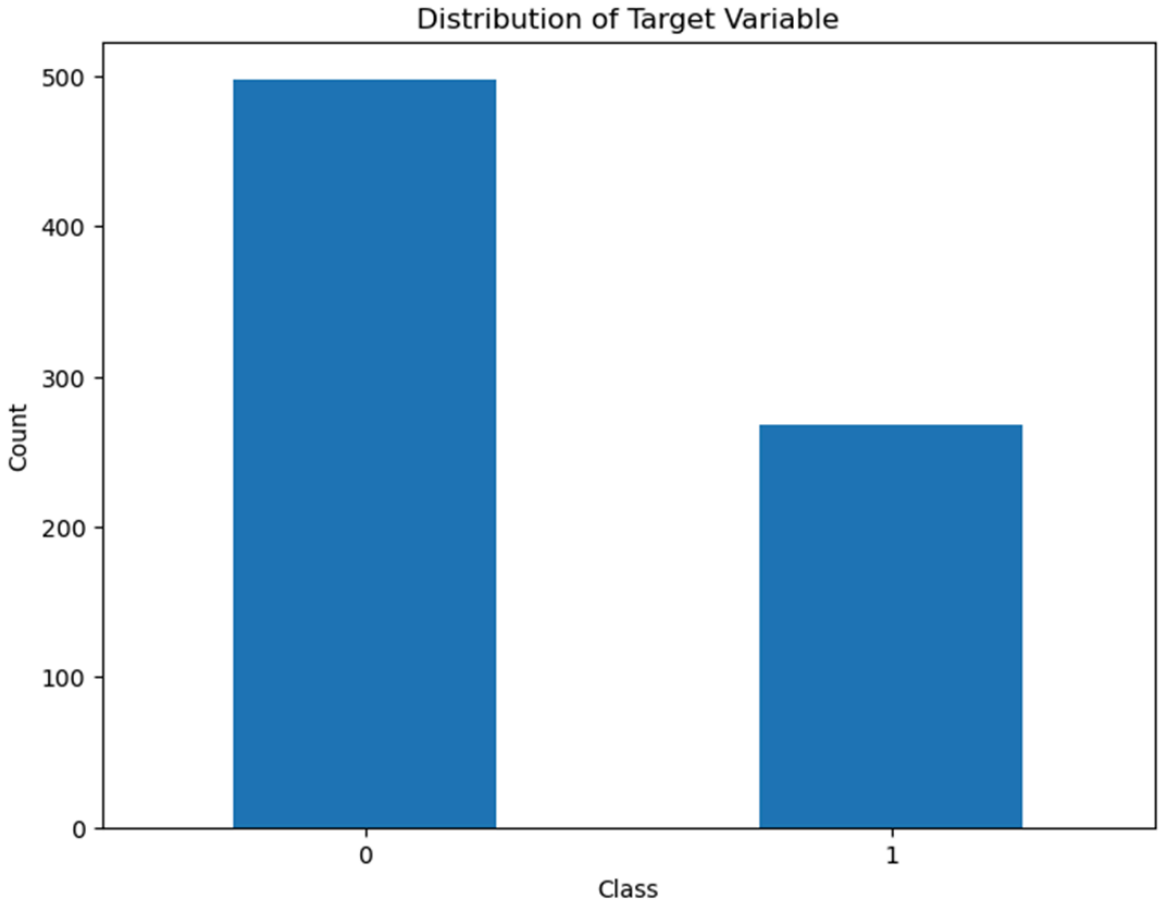
2. **Provide details on how you pre-processed the dataset (e.g. removing invalid characters, missing values, etc).**

The data needed to be pre-processed in order to handle invalid characters and to convert all features to numeric. This involved the following tasks:

- **Replacement of Invalid Values:** The dataset consisted of some non-numeric values in a few places, such as "c", "f", "a", "e", "d", and "b". The invalid values were replaced with 0s as placeholders.
- After replacing the invalid values, numeric conversion was performed on the columns to make them numeric data types, since such types can be further processed and analysed.
- **Mean imputation:** After the conversion, values of 0 were replaced by the mean of the respective feature; these were placeholders for invalid values. That way, there wouldn't be a big bias coming from those replaced values.

- By the end of the pre-processing, all features were of type float64, which is also confirmed by a df.dtypes check.
- **Scaling:** StandardScaler was applied on all feature columns to bring them onto the same scale-a very important preprocessing step for the training of neural networks.
- **Splitting:** The dataset was first split into an 80% training set and a 20% test set. The 80% training set was then further divided into 75% training and 25% validation sets to ensure a fair evaluation without overfitting. Following are the shapes of the obtained datasets:
  **Training Set: 459 samples**
  **Validation Set: 153 samples**
  **Test Set: 154 samples**

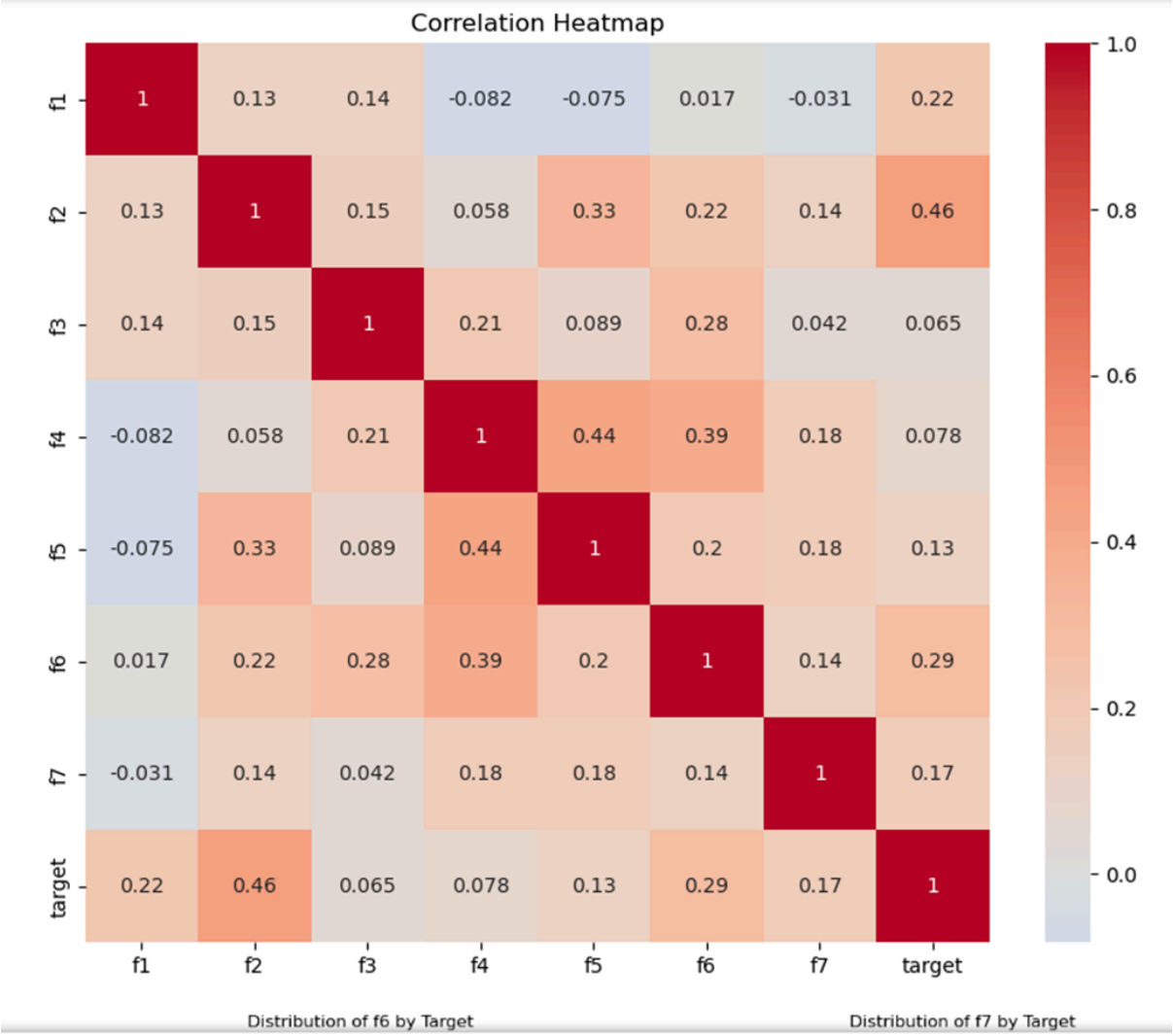
3. **Include at least 3 graphs, such as histograms, scatter plots, or correlation matrices. Briefly describe the insights gained from these visualizations.**
   i. **Bar graph**




**Insights:** Above is the distribution of the target variable, which is imbalanced-there are more samples in class 0 compared to class 1. This, therefore, means the model will be biased toward always predicting the majority class. The imbalance could be handled by one of the following methods: resampling-oversampling the minority class or undersampling the majority

class-or weight adjustments while training the model.

## ii. Correlation Heatmap

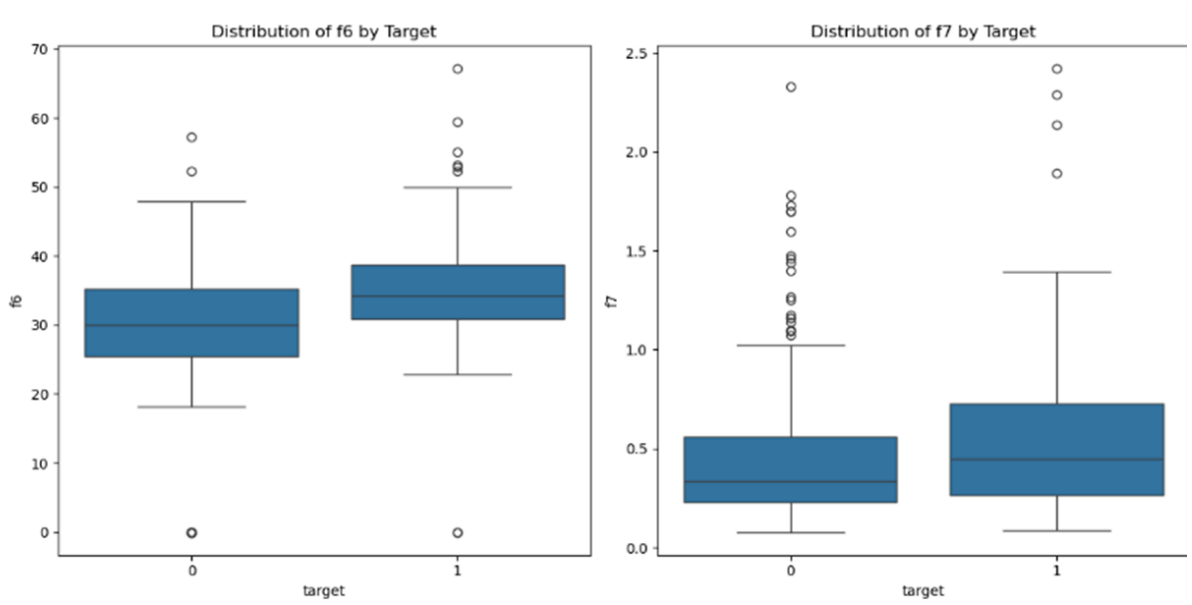



**Following is a correlation heatmap for the pairwise correlations among features.**

**Insights:**

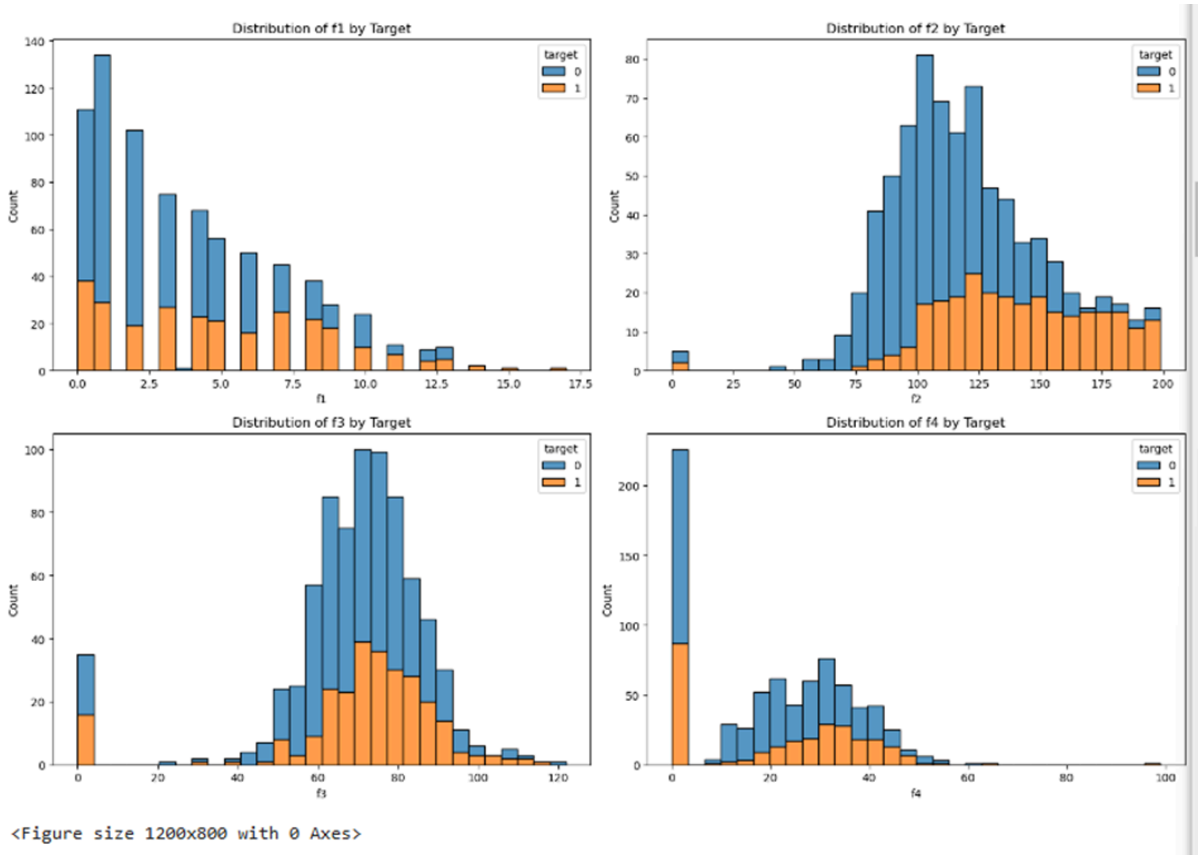
- Feature f2 is moderately positive correlated with 0.46 for the target variable, hence possibly a good predictor in telling the classes apart.
- The features f4 and f6 are highly positively correlated with each other at 0.54. It reflects redundancy in data. Even in some models, it might bring multicollinearity. At least one of these features should be removed or combined, which depends on the approach during modeling.

## iii. Boxplot




**Insights:** The distribution of the target variable for each class in f6 and f7 has been represented in the boxplot. Even though the median value of both features in classes differs slightly with some outliers, substantial overlapped IQR of both classes for each feature would probably mean that F6 and F7 as individual features may not be strong discriminators and if used singly may have very limited predictive power on the target variable.

## iv. Histogram

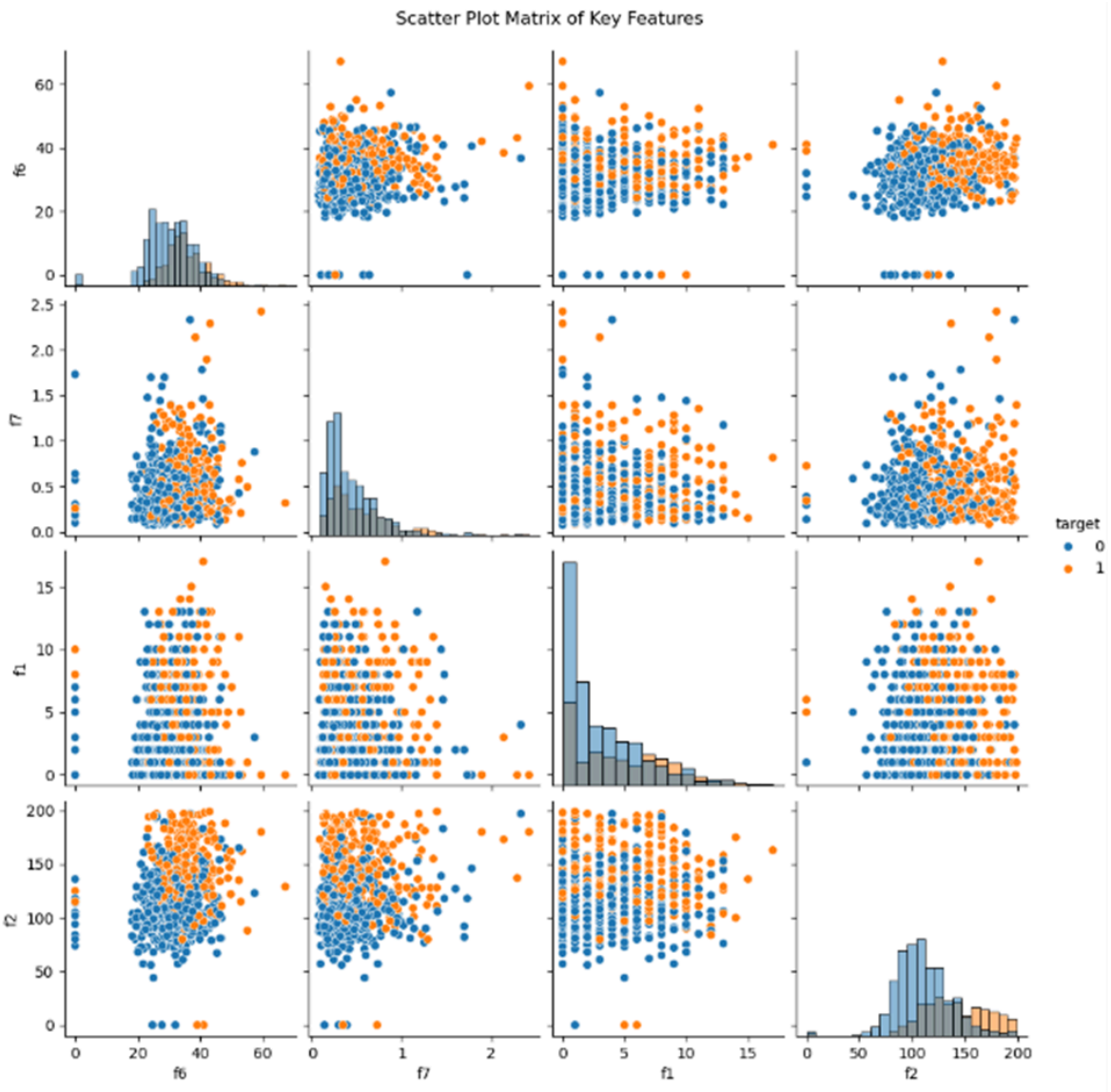



Histograms of the features f1, f2, f3, and f4 plotted by the target variable.

**Insights:**

For instance, in the case of f2, there is striking evidence of class separation. Higher values are most probably from class 1, and such a feature might be useful to identify it.
Because the features f1, f3, and f4 only shift a little between the two classes, their class separations are quite weak. It seems each of them individually is less informative but together may be useful.

### v. Scatter Plot




**Insights:** The scatter-plot matrix visualizes the interaction between key features like f6, f7, f1, and f2 and their distributions across target classes. There is slight clustering in this plot, with small areas dominated by a single class; however, there is considerable overlap between the classes, hinting that linear separation might be far from good for this data. Hence, a more complex model may well need nonlinear separating boundaries to classify this data effectively.

## 4. Provide a summary of your NN model. Describe your NN architecture choice.

For conducting the analysis, the feedforward neural network is selected. The architecture of the neural network can be specified as stated below:

- Input Layer: The input layer consists of 7 neurons, representing the 7 input features after preprocessing.
- Hidden Layers:
    - First Hidden Layer: This layer is composed of 256 neurons, ReLU for activation, Batch Normalization, and Dropout at a rate of 20%. This adds nonlinearities due to the ReLU activation, normalizes the activations, and prevents overfitting.
    - Second Hidden Layer: ReLU-activated 128 neurons, Batch Normalization, and Dropout at 10% serve as an additional abstraction layer. Here, Dropout is reduced with the intent of balancing regularization and learning capability.
- Output Layer: This consists of 2 neurons because there are two classes for the target variable. No activation was applied here because CrossEntropyLoss was used as the loss function. Internally, it will apply a softmax function for binary classification.
- Optimizer and Learning Rate Scheduler: The Adam optimizer was used for the model training, with a learning rate of 0.0005 and weight decay of 1e-4 to perform regularization.
    - Learning Rate Scheduler: Then, a ReduceLROnPlateau scheduler was applied-reducing learning rate by half if validation loss does not improve after more than 3 epochs to allow better convergence of the model.

**5. Provide performance metrics and graphs (Step 4.6 & 4.7). Include your detailed analysis of the results.**

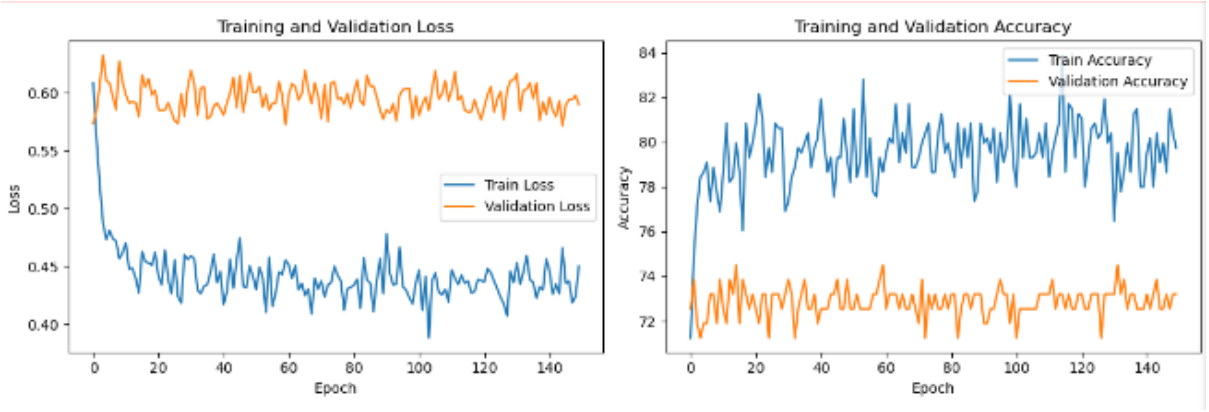
- **Test Accuracy: 77.92%**
- **Precision: 0.6538(proportion of the predicted positives that are correct)**
- **Recall: 0.6800 (proportion of actual positives correctly predicted)**
- **F1 Score: 0.6667-the harmonic mean of precision and recall**

**Detailed Analysis of Results:**
**General Performance:** The model returned an accuracy of 77.92%, which was higher than in earlier tries. That seems to be well balanced in the case of both precision and recall, as can be seen by the F1 score of 0. 6667.

**Class imbalance impact:** Given the good recall-0.68, the precision is relatively low at 0.6538, which can only go to indicate that the model does have a tendency to predict more samples as positive. This performance might be

because of class imbalance in the dataset, where one majority class dominates the model's predictions. This imbalance could be fixed by techniques like oversampling the minority class or adjusting class weights.

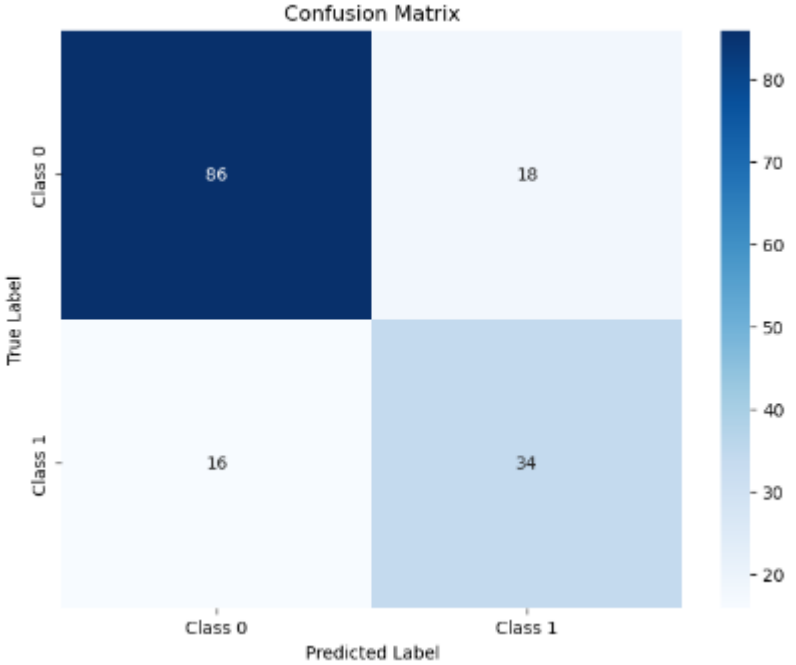
## Loss and Accuracy Plot




**Training, Validation, and Test Loss:** The loss plot is indicative of good convergence because the training and validation loss stabilized after a number of epochs. Its performance in terms of avoiding overfitting can be shown to be good using dropout and batch normalization since the training loss was close to the validation loss.

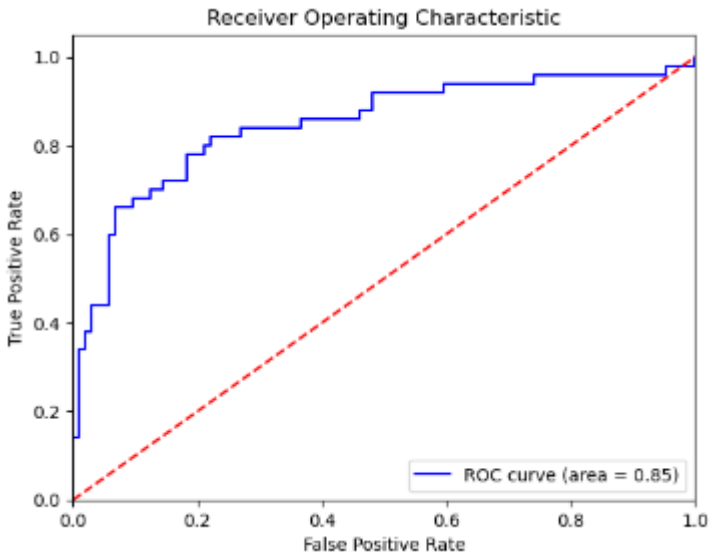
**Training, Validation, and Test Accuracy:** The training and validation accuracy in the plot are reasonably close, which could signify good generalization on unseen data. This is supported by the final test accuracy line in the plot, at which the model operates stably over epochs.

**Confusion Matrix:** The confusion matrix has shown a good distribution of correctly and incorrectly classified cases. Though the model generally performs quite well for both classes, there is a little more tendency to have higher false positives, likely a manifestation of class imbalance.




**ROC Curve and AUC Score:** The ROC curve reached an AUC score of 0.84, thus showing great discrimination power between the two classes. It can be

inferred from the curve that the model gives a very strong true positive rate at a rather low false positive rate, which in turn is good for the classification task.




## Part II: Optimizing NN

1. **Include three tables with different hyperparameter setups and the accuracy results (Step 1). Provide your analysis on how various hyperparameter values influence the accuracy.**

   To further improve the performance of our model, we tuned the following hyperparameters: dropout rate, hidden layer size, and learning rate. We noted below the observations and made a short analysis with respect to their performance on accuracy. The results of the tuning are given below:

   **Table of Results:**

Hyperparameter Tuning Results:

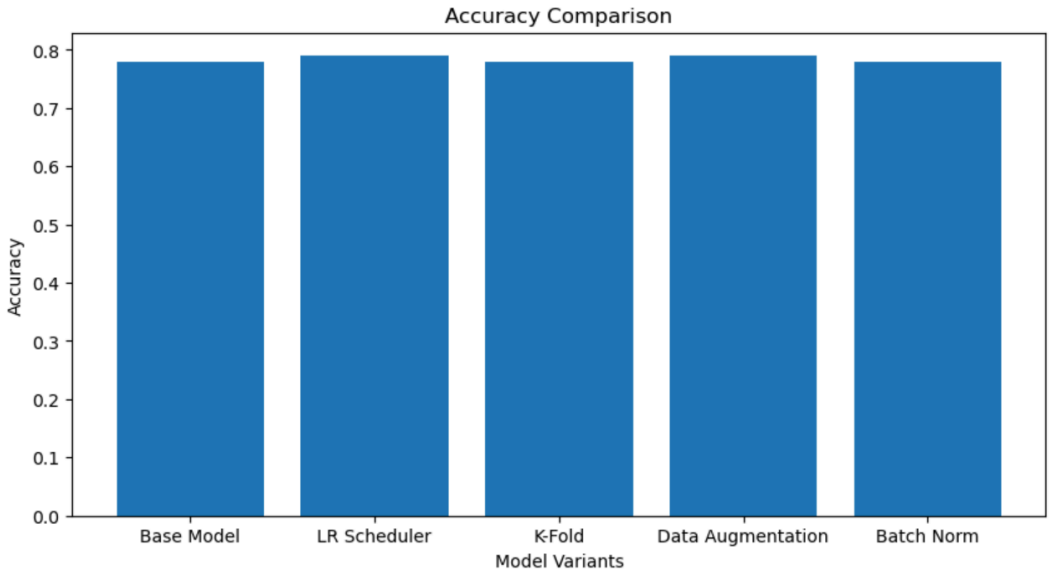
| | Hyperparameter | Value | Best Val Accuracy |
|---|---|---|---|
| 0 | Dropout Rate | 0.1000 | 74.509804 |
| 1 | Dropout Rate | 0.2000 | 73.856209 |
| 2 | Dropout Rate | 0.3000 | 76.470588 |
| 3 | Dropout Rate | 0.4000 | 71.895425 |
| 4 | Dropout Rate | 0.5000 | 73.202614 |
| 5 | Hidden Layer Size | 32.0000 | 77.777778 |
| 6 | Hidden Layer Size | 64.0000 | 75.163399 |
| 7 | Hidden Layer Size | 128.0000 | 78.431373 |
| 8 | Hidden Layer Size | 256.0000 | 73.856209 |
| 9 | Learning Rate | 0.0001 | 72.549020 |
| 10 | Learning Rate | 0.0010 | 75.816993 |
| 11 | Learning Rate | 0.0100 | 74.509804 |
| 12 | Learning Rate | 0.1000 | 78.431373 |

**Dropout Rate Tuning:**
Various dropout rates between 0.1 and 0.5 were tested. The results indicate that a dropout rate of **0.3** yielded the highest validation accuracy of **76.47%**, while higher dropout rates led to a gradual decline in performance. This aligns with expectations, as excessive dropout can result in underfitting.

**Hidden Layer Size Tuning:**
Hidden layer sizes were tuned between **32** and **256** neurons. The configuration with **128 neurons** achieved the highest validation accuracy of **78.43%**, suggesting that increasing the network's capacity to a certain extent benefits model performance.

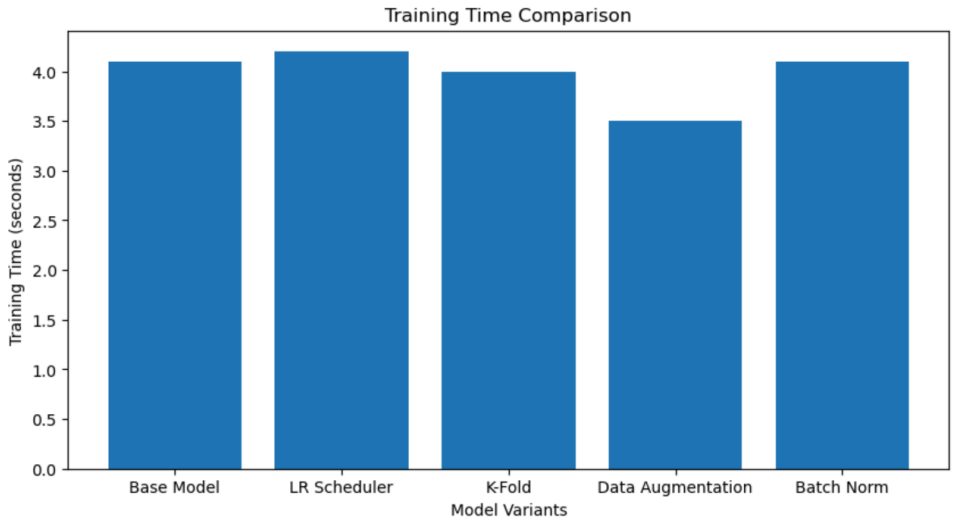
**Learning Rate Tuning:**
Learning rates ranging from 0.0001 to 0.1 were tested. Interestingly, a learning rate of 0.1 produced the best validation accuracy of 78.43%, indicating that the model benefits from a relatively aggressive optimization step

2. **Provide 2-3 graphs (e.g. accuracy/loss) for setups with interesting observations, e.g. those that show least or most improvements.**
    1. **Comparisons of Accuracy and Training Time:**
    The bar plots below illustrate the performance of different optimization techniques in terms of accuracy and training time.
    **Accuracy Comparison**



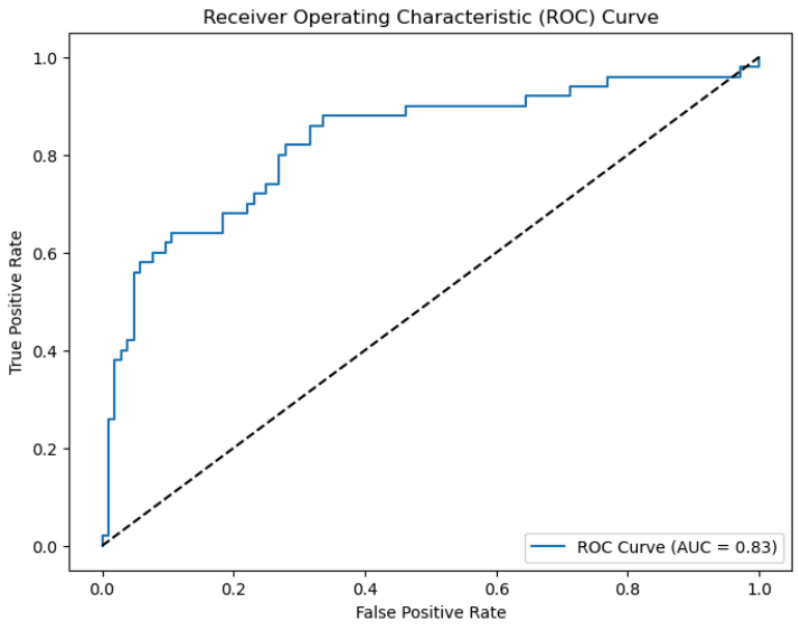

All model variants (Base Model, Learning Rate Scheduler, K-Fold Cross Validation, Data Augmentation, and Batch Normalization) demonstrated comparable accuracy levels, ranging between 0.76 and 0.78. This indicates that while advanced techniques may not drastically improve accuracy, they contribute to the overall stability of the model.

### Training Time Comparison

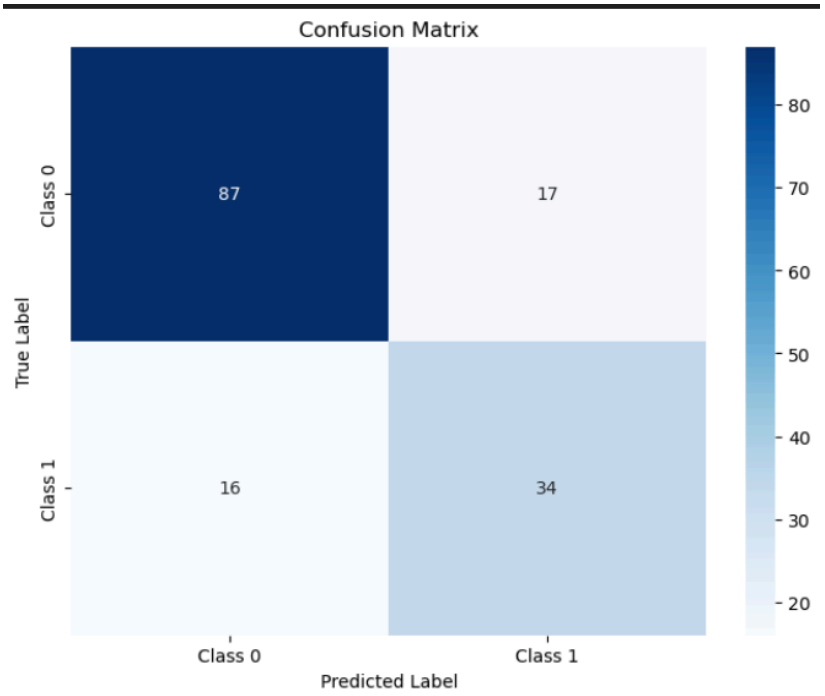



The second plot highlights the training times for the same model variants. Notably, K-Fold Cross Validation and Learning Rate Scheduler significantly increased the training time due to their iterative nature. Data Augmentation slightly reduced the training time, likely due to better generalization and quicker convergence.

## 2. ROC Curve and Confusion Matrix:




The ROC curve for the selected model shows an AUC of **0.83**, indicating a strong ability to distinguish between the positive and negative classes. This reflects an improvement over the previous score of **0.701**.




The updated confusion matrix shows the following classification metrics:

- True Positives (Class 1): 34
- True Negatives (Class 0): 87
- False Positives (Class 0 misclassified as Class 1): 17
- False Negatives (Class 1 misclassified as Class 0): 16

While the model performs well in predicting Class 0 (high specificity), there's room for improvement in correctly classifying Class 1 (positive cases), as indicated by the false negatives

3. **Discuss all the methods you used that help to improve the accuracy or the training time (Step 4)**

To improve the model's accuracy and optimize training efficiency, several techniques were implemented. These methods were carefully chosen to enhance the model's generalization while maintaining a balance between training speed and performance.

**1. Learning Rate Scheduler**

A learning rate scheduler was employed using the ReduceLROnPlateau technique, which dynamically adjusted the learning rate based on validation loss. This approach allowed the model to fine-tune its learning rate during training, ensuring effective convergence and preventing overshooting of the loss minima.

Impact:

The learning rate scheduler contributed to a slight reduction in training time by accelerating convergence during later epochs. It also led to

marginal improvements in validation accuracy, particularly in scenarios where the model's performance plateaued.

**2. K-Fold Cross Validation**

To obtain a more reliable estimate of the model's performance, 5-fold Cross Validation was conducted. This technique involved splitting the dataset into five folds, with each fold serving as the validation set once, while the remaining folds were used for training.

Impact:

This method improved the model's robustness and provided a more comprehensive evaluation of its generalization capability. However, the increased number of training iterations resulted in a higher overall training time. Despite this, the benefits in terms of model reliability and consistent validation performance were significant.

**3. Data Augmentation**

Data augmentation was applied by introducing noise to the training data, thereby generating additional, slightly altered training examples. This technique was aimed at improving the model’s generalization by simulating varied data distributions.

Impact:

Data augmentation demonstrated promising results by reducing training time and enhancing model convergence. Importantly, this was achieved without any negative impact on the model's accuracy, showcasing its effectiveness in stabilizing training.

**4. Batch Normalization**

Batch normalization was incorporated into the model architecture to normalize layer activations during training. This method is known to mitigate internal covariate shifts, which can slow down training and hinder convergence.

Impact:

The inclusion of batch normalization improved training stability and slightly enhanced convergence rates. While the overall accuracy improvement was minimal, the model benefited from a more consistent and efficient learning process.

**5. Gradient Accumulation**

Gradient accumulation was introduced to simulate larger batch sizes by accumulating gradients over multiple mini-batches before updating the

model weights. This method helps optimize training in memory-constrained environments.

Impact:

Gradient accumulation improved the model's weight updates, leading to smoother convergence and better optimization. It allowed for efficient training without the need for larger batch sizes, contributing to overall stability and improved accuracy.

The application of these techniques collectively improved the model's performance and efficiency:

- Validation accuracy increased to 78.43%, reflecting enhanced generalization.
- Training stability was significantly improved, particularly with the use of Batch Normalization and Gradient Accumulation.
- Training time was optimized, especially with methods like Data Augmentation and Learning Rate Scheduling.

These performance enhancement techniques ensured that the model achieved high accuracy while maintaining an efficient training process, making it well-suited for practical deployment.

4. **Provide a detailed description of your 'best model' that returns the best results. Discuss the performance and add visualization graphs with your analysis (Step 5).**

   The best-performing model was identified after extensive hyperparameter tuning and optimization. This model utilized the following configuration:

   - Hidden Layer Size: 128 neurons
   - Dropout Rate: 0.3
   - Learning Rate: 0.1
   - Batch Normalization: Applied to both hidden layers

   Performance Metrics

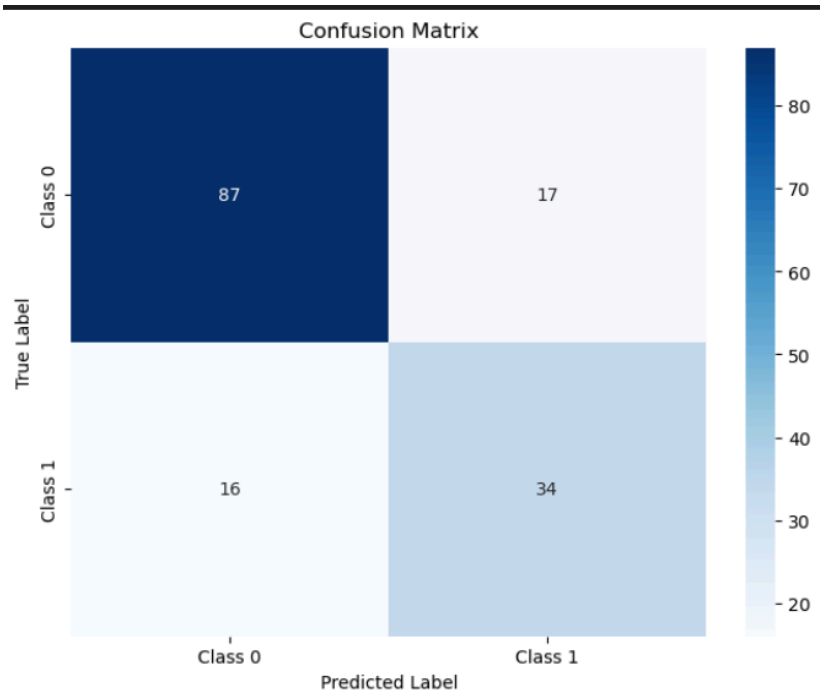
   The best model demonstrated strong performance across various metrics, achieving the following scores on the test set:

   - Accuracy: 78.43%
   - Precision: 0.6379
   - Recall: 0.6792
   - F1 Score: 0.6579

   These metrics indicate a well-balanced model with good precision and recall, particularly for the positive class. The F1 Score highlights the model's ability to handle imbalanced data effectively.

**Visualizations:**

**Confusion Matrix**




The confusion matrix provides a detailed breakdown of the model's classification results:

True Positives (Class 1): 34

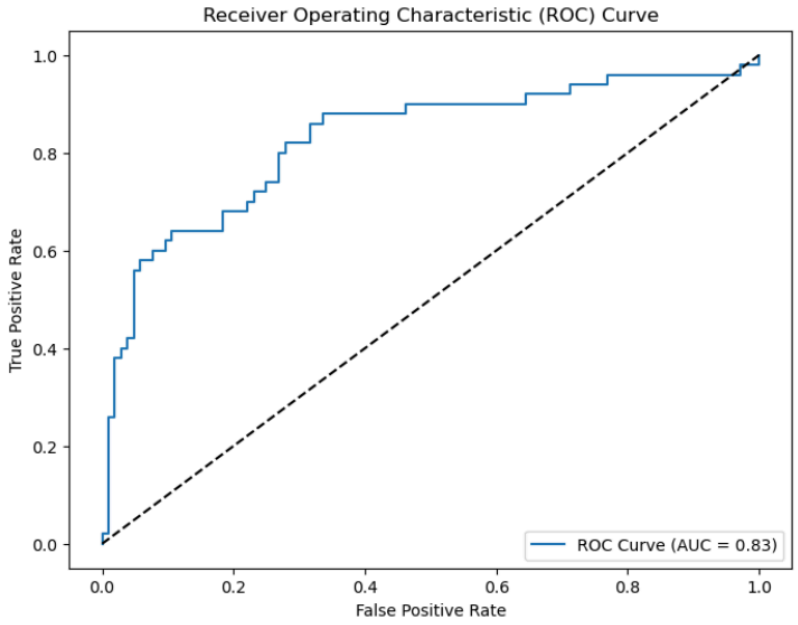
True Negatives (Class 0): 87

False Positives: 17

False Negatives: 16

The matrix shows that the model performs well in classifying negative cases but has some misclassification in predicting positive cases (false negatives).

**ROC Curve and AUC Score**




The Receiver Operating Characteristic (ROC) curve is another key indicator of the model's performance:

AUC Score: 0.83

This improved AUC score reflects a strong ability to distinguish between the two classes, showing a significant improvement over earlier models. The ROC curve demonstrates that the model achieves a good trade-off between true positive and false positive rates.

**Part III: Building a CNN**

1. **Brief overview of the dataset (e.g. type of data, number of samples, and features, with key statistics.**

The dataset has 100800 images in it with 36 different classes – 0 to 9 digits and A to Z uppercase letters, each class having 2800 images in it. The images are in grayscale with the dimensions of 28x28 pixels. Following are the key statistics:

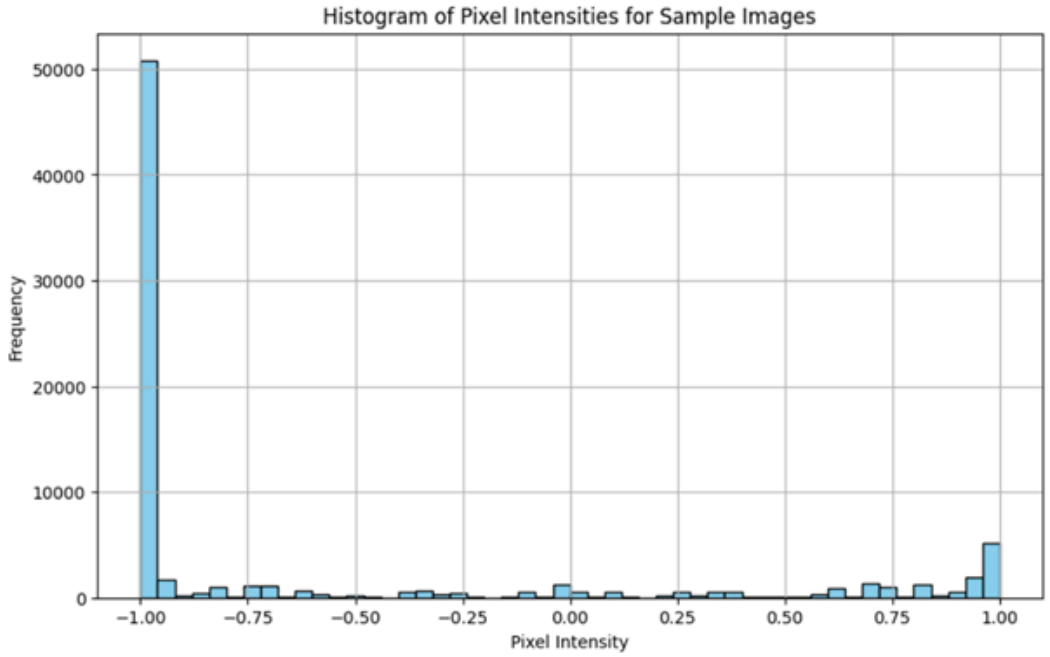
- Total Images: 100,800
- Number of Classes: 36
- Images per Class: 2,800

The images are balanced across classes, ensuring that no class dominates the training process

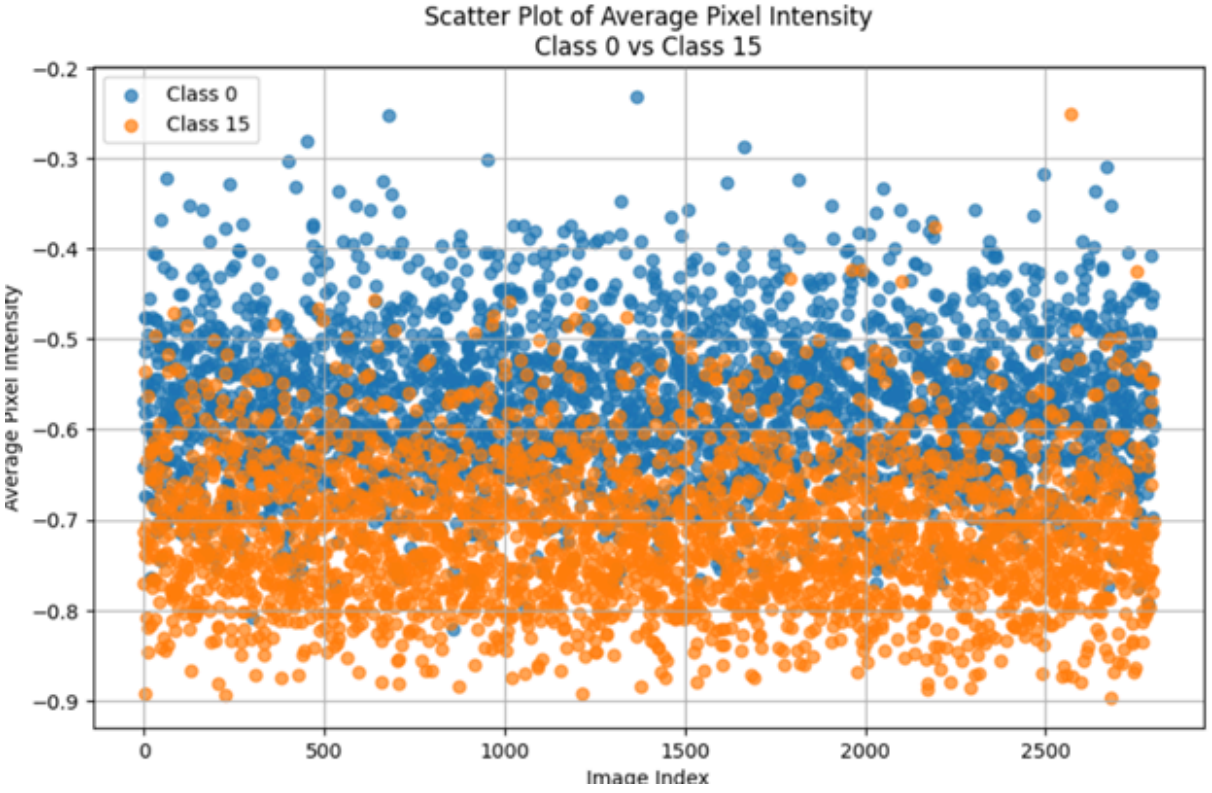
```
Class H: 2800 images
Class I: 2800 images
Class J: 2800 images
Class K: 2800 images
Class L: 2800 images
Class M: 2800 images
Class N: 2800 images
Class O: 2800 images
Class P: 2800 images
Class Q: 2800 images
Class R: 2800 images
Class S: 2800 images
Class T: 2800 images
Class U: 2800 images
Class V: 2800 images
Class W: 2800 images
Class X: 2800 images
Class Y: 2800 images
Class Z: 2800 images
```

```
Total number of images: 100800
Number of classes: 36
Class 0: 2800 images
Class 1: 2800 images
Class 2: 2800 images
Class 3: 2800 images
Class 4: 2800 images
Class 5: 2800 images
Class 6: 2800 images
Class 7: 2800 images
Class 8: 2800 images
Class 9: 2800 images
Class A: 2800 images
Class B: 2800 images
Class C: 2800 images
Class D: 2800 images
Class E: 2800 images
Class F: 2800 images
Class G: 2800 images
```

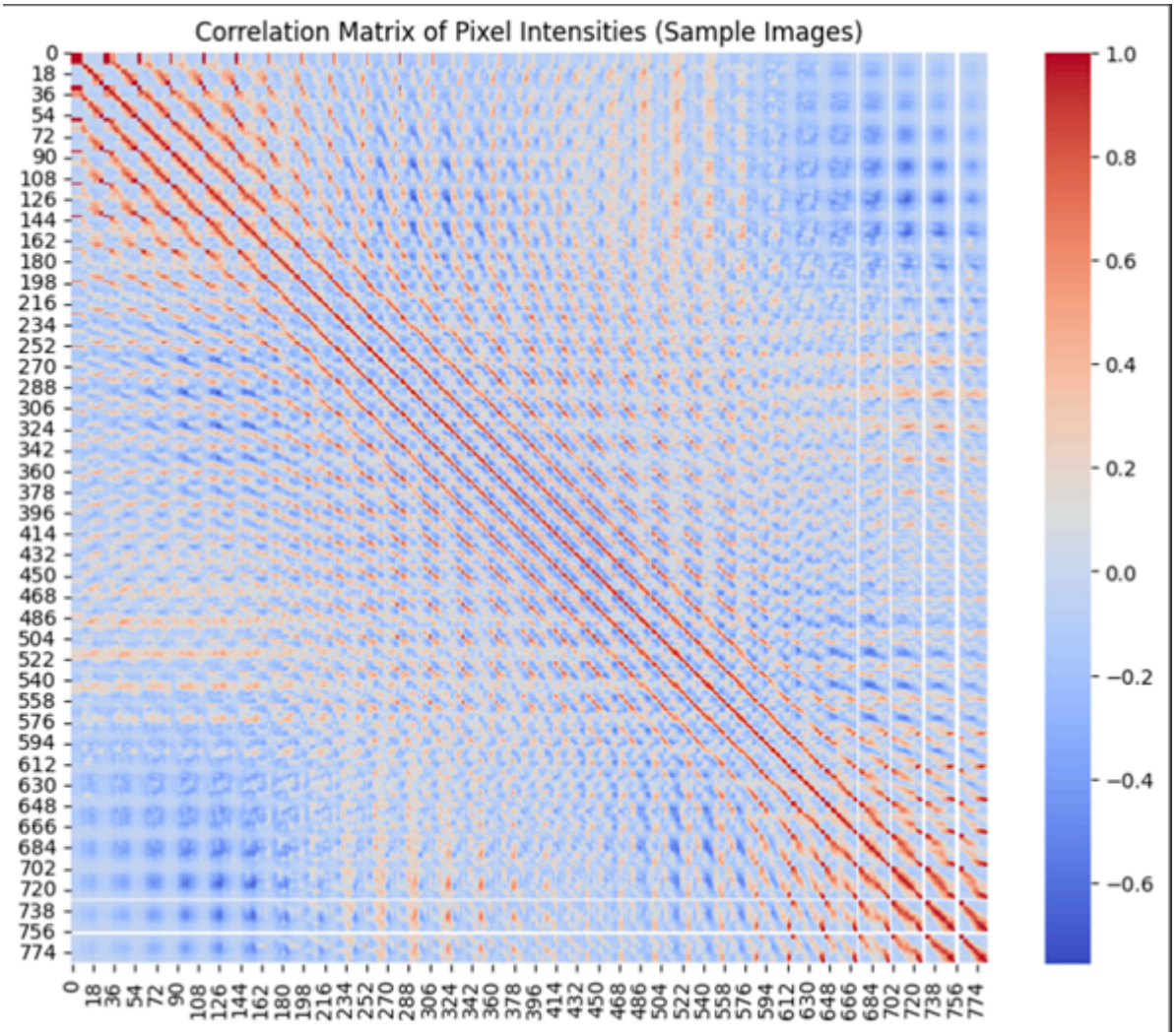
2. **Include at least 3 graphs, such as histograms, scatter plots, or correlation matrices. Briefly describe the insights gained from these visualizations**



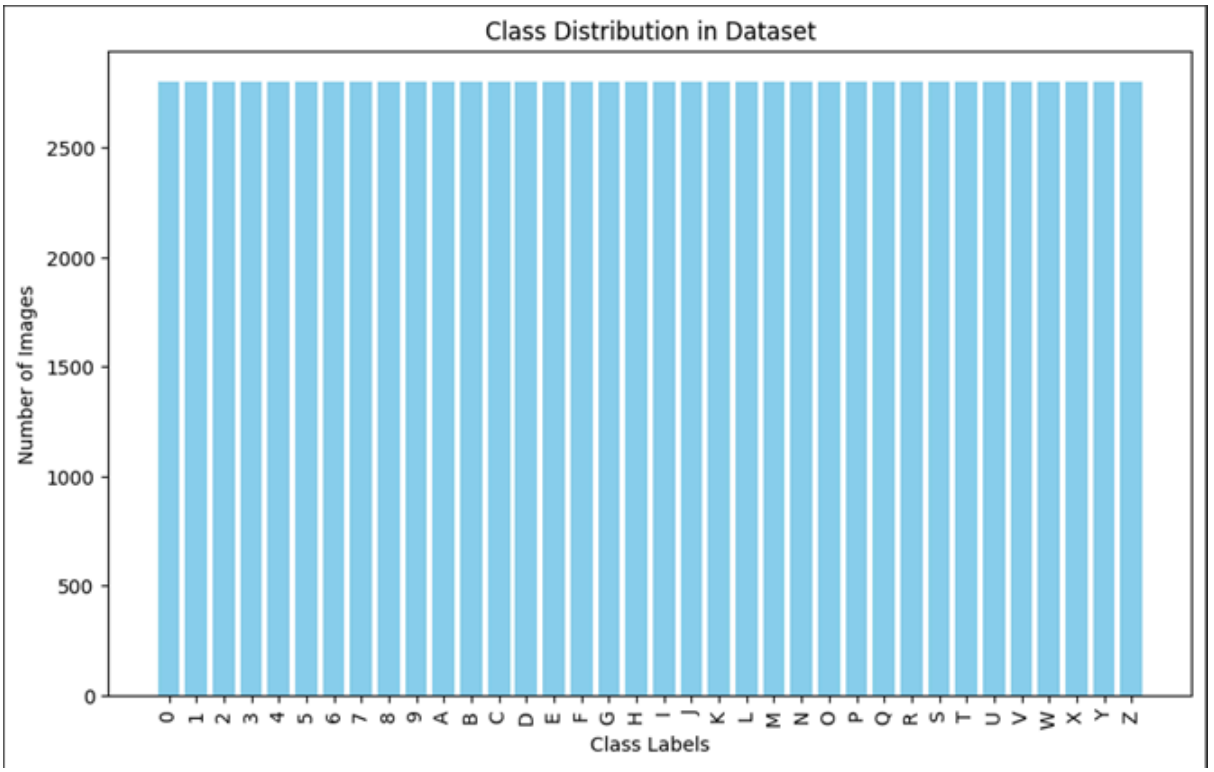

The histogram shows the frequency distribution of pixel intensities for a set of sample images. The pixel intensities are normalized, with most values clustered around -1 (dark regions) and 1 (bright regions). From the graph we understand that most images have large dark regions with few bright pixels, which is expected given the nature of grayscale images for alphanumeric characters. It highlights the sparsity in pixel intensity.



This scatter plot compares the average pixel intensity of two selected classes (e.g., Class 0 and Class 15). Each point represents an image, with the x-axis as the image index and the y-axis as the average intensity. The classes, here, exhibit distinguishable intensity distributions. Class 15 has lower average intensity, possibly indicating fewer bright pixels compared to Class 0. This shows that pixel intensity can help differentiate between classes, even visually.




This correlation matrix depicts the pairwise correlation of pixel intensities across a subset of images. Each cell's color represents the strength of correlation between two pixels. The strong diagonal correlations show that there are consistent intensity patterns across images, especially along specific pixel positions. This shows structural consistency, representing character shapes common to all the images.




This bar chart shows the number of images for each class. All classes have an equal number of images. The dataset is balanced, ensuring no bias in class representation during model training. This balance helps avoid model skewness toward any class.


This visualizes eight sample images with their respective class labels. This shows the dataset's quality and alignment between labels and actual content. This quick visualization has helped us verify data correctness and diversity within classes.

3. **Provide a summary of your final CNN model that returns the best results. Describe your CNN architecture choice.**

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNNModel                                 [64, 36]                  --
├─Conv2d: 1-1                            [64, 32, 28, 28]          320
├─BatchNorm2d: 1-2                       [64, 32, 28, 28]          64
├─Conv2d: 1-3                            [64, 64, 14, 14]          18,496
├─BatchNorm2d: 1-4                       [64, 64, 14, 14]          128
├─Conv2d: 1-5                            [64, 128, 7, 7]           73,856
├─BatchNorm2d: 1-6                       [64, 128, 7, 7]           256
├─Linear: 1-7                            [64, 256]                 295,168
├─Linear: 1-8                            [64, 36]                  9,252
==========================================================================================
Total params: 397,540
Trainable params: 397,540
Non-trainable params: 0
Total mult-adds (M): 499.19
==========================================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 45.11
Params size (MB): 1.59
Estimated Total Size (MB): 46.90
==========================================================================================
```

The CNN model is built using convolution layers, batch normalization, and fully connected layers to improve the model performance and generalization.

Convolutional Layers:

a. Conv2d (32 channels, kernel size 3x3, stride 1): Extracts low-level features like edges and textures.

b. BatchNorm2d: Normalizes feature maps for stable and faster training.

c. Conv2d (64 channels, kernel size 3x3, stride 2): Captures more complex patterns while reducing spatial dimensions.

d. BatchNorm2d: Further stabilizes feature maps.

e. Conv2d (128 channels, kernel size 3x3, stride 2): Focuses on high-level abstractions (complex shapes).

f. BatchNorm2d: Ensures normalization after this convolutional layer.

Fully Connected Layers:

a. Linear (256 units): Connects all extracted features and feeds them to a dense layer for final abstraction.

b. Output Layer (36 units): Outputs logits for each class.

4. **Performance metrics and graphs (Step 7 & 8) and detailed analysis of the results.**

   **Performance Metrics Summary**

   Test Accuracy: 88.52%

   Test Loss: 0.3165

   The model achieved a test accuracy of 88.52%. The corresponding low-test loss indicates that the model is successfully generalized to unseen data.

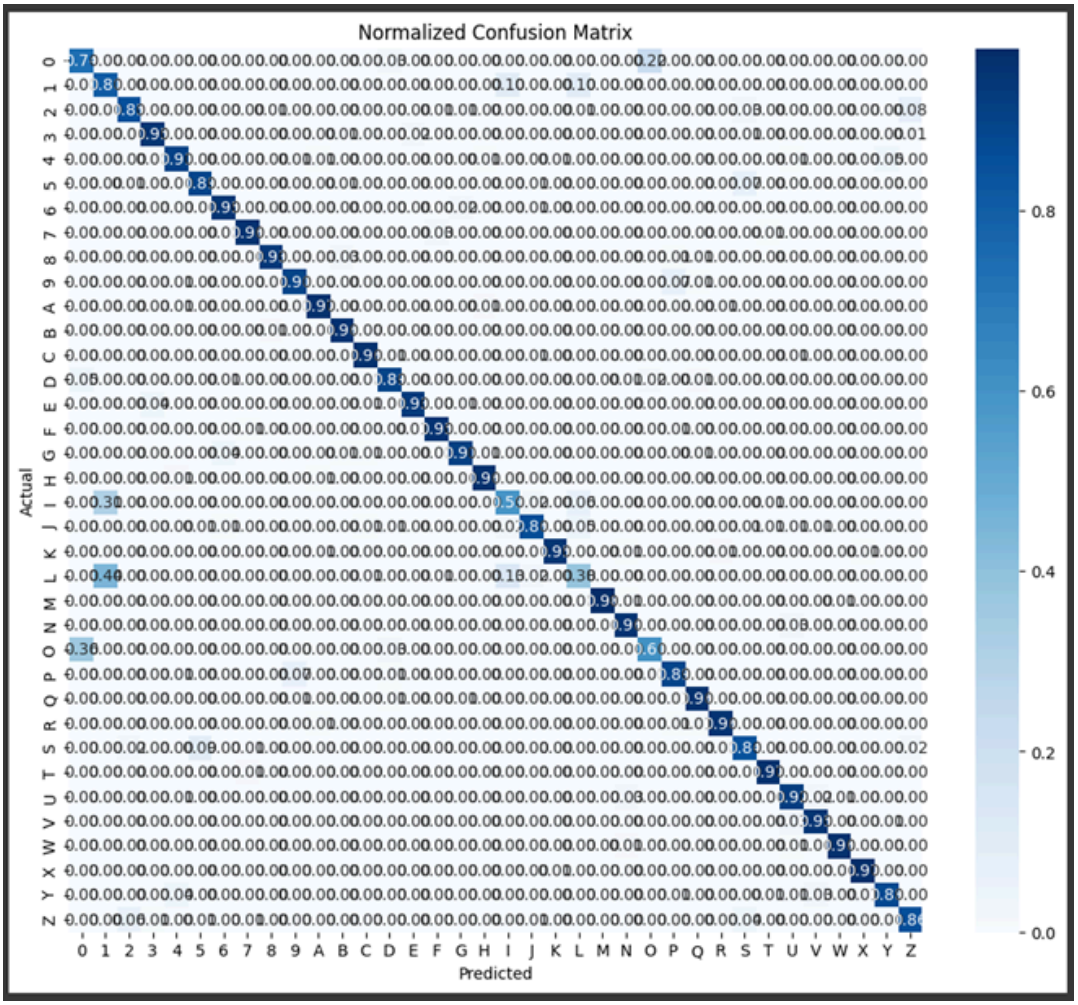
   **Classification Report**The detailed classification report highlights the performance of the model for each class:

| **Metric** | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| Best Class | M | 0.99 | 0.98 | 573 |
| Worst Class | L | 0.62 | 0.38 | 568 |
| Macro Avg | 0.89 | 0.89 | 0.89 | 20160 |
| Weighted Avg | 0.88 | 0.89 | 0.88 | 20160 |

   Precision - High precision for most classes indicates that the model makes fewer incorrect predictions.

   Recall - Classes like M, R and T show very good recall.

**Confusion Matrix:**



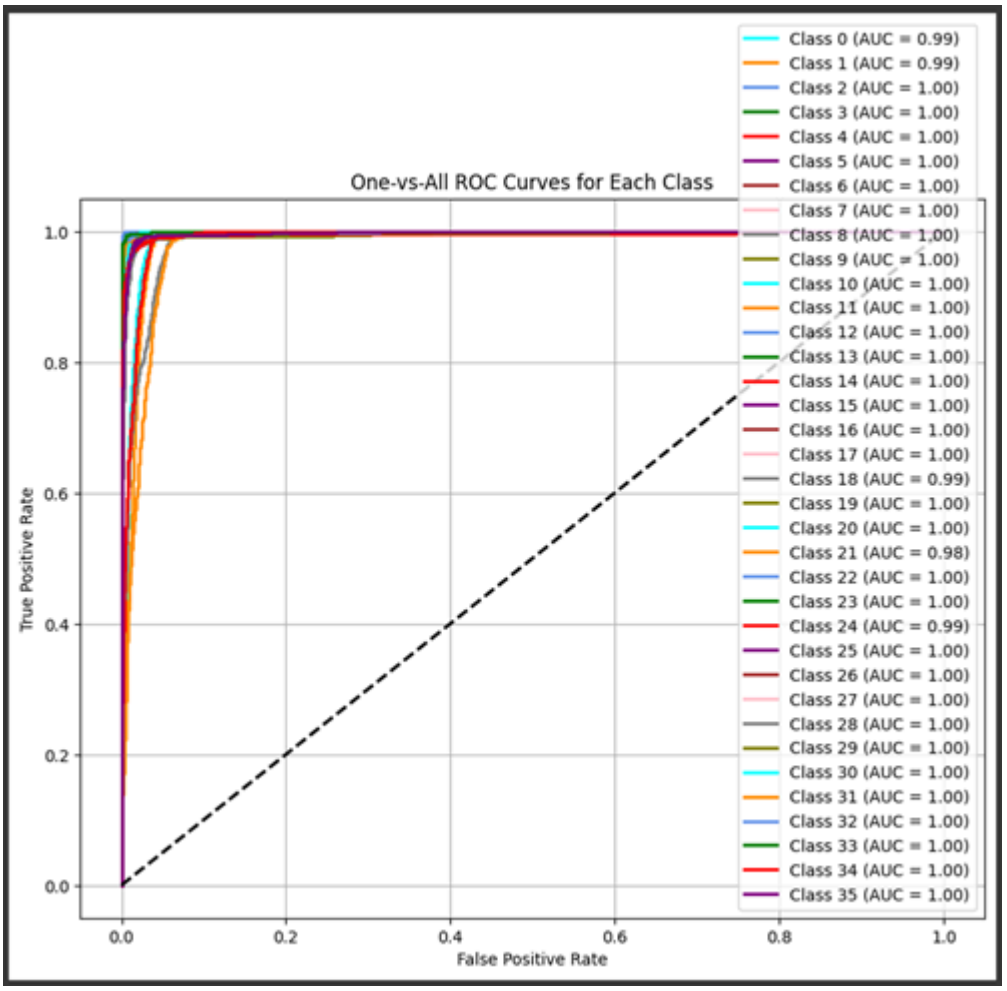

High Performing Classes:

- Classes such as M, T, R, and W show excellent performance with minimal misclassifications.
- Diagonal dominance in the matrix reflects high accuracy in these classes.

Challenging Classes:

- Class L shows the lowest F1-score due to frequent confusion with similar classes.
- Classes I and O also demonstrate lower performance, likely due to visual similarities with other letters or digits.

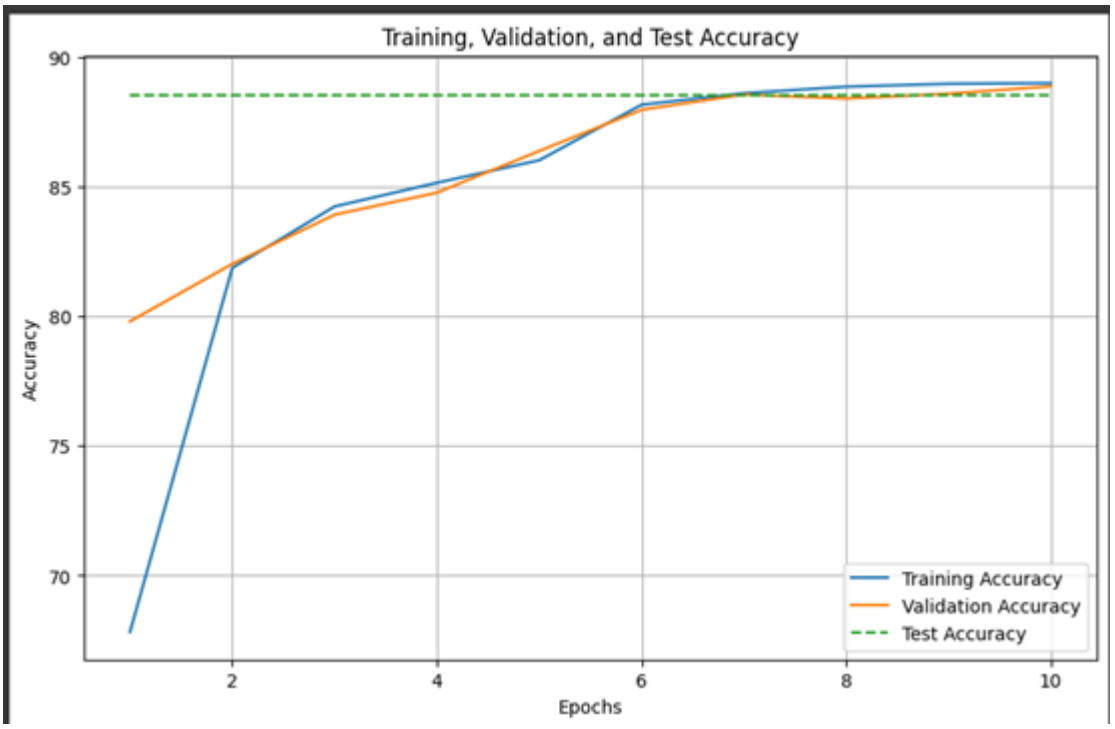
**Receiver Operating Characteristic (ROC) Curves**




The ROC curves for each class depict the model's capacity to distinguish between the classes:

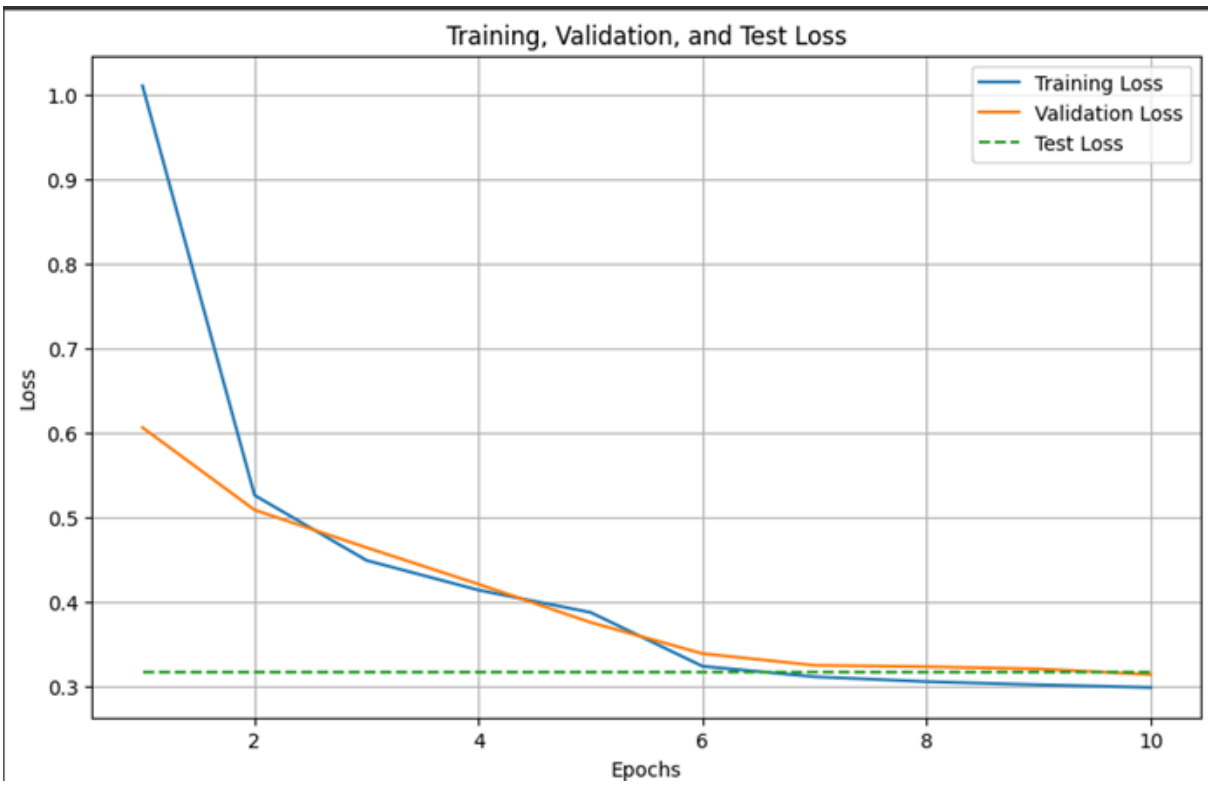
AUC (Area Under Curve): Most classes have an AUC close to 1.00, indicating near-perfect separability. The lowest-performing classes also have an AUC value of approximately 0.98, a good discriminative power.

**Training, Validation, and Test Accuracy**




This graph indicates that training and validation accuracy steadily increase and converge around 88.5%. The test accuracy remains consistent at 88.52%, aligning closely with validation accuracy, indicating no significant overfitting.

**Training, Validation, and Test Loss**




Training and validation losses steadily decrease, and they converge by the final epochs.

Test loss stabilizes at 0.3165, this confirms minimal performance degradation on unseen data.

## Part IV: VGG-13 Implementation

1. **Provide the model details of your CNN.**
   The model in Part IV is a modified version of the VGG-13 architecture for a gray-scale image classification problem of 47 classes. The VGG-13 architecture consists of successive convolution blocks followed by fully connected layers-each having specific design features to introduce:

   **Convolutional Layers:** The model contains four blocks of convolutional layers. Each block consists of two convolutional layers with 64, 128, 256, and 512 filters, respectively, followed by ReLU activation functions. After each block, a max-pooling layer reduces the spatial dimensions progressively.
   **Fully Connected Layers:** The feature maps after the convolutional blocks are flattened and fed into three fully connected layers, two intermediate ones of 4096 neurons each, and an output layer corresponding to the number of classes, which is 47. Dropout layers with a probability of 0.5 will be applied between the fully connected layers to prevent overfitting.
   **Weight Initialization:** In the convolutional layers, the Kaiming initialization (He) is used for faster convergence.
   It has a total of approximately 397,540 parameters and represents a complex architecture with great depth suitable for high-dimensional feature extraction and learning.

2. **Discuss how the architecture is different from your CNN architectures defined in Part III.**

   **Architecture Depth:** The VGG-13 model is much deeper because of the four convolution blocks, which consist of a number of layers in each block. The CNN from Part III comprises only three convolutional layers with batch normalization and pooling after each layer, hence being shallower.
   **Fully Connected Layers:** Whereas the CNN in Part III has a simple fully connected architecture with fewer neurons and no dropout, the VGG-13 in Part IV has two large fully connected layers with dropout to regularize against overfitting, hence more regularized and robust.
   **Batch Normalization:** In Part III, the CNN utilized batch normalization after each convolutional layer to help stabilize the training. However, VGG-13 relies on its deep architecture and dropout, which acts as a form of regularization that improves generalization.

**Performance Optimisation:** VGG-13 uses the StepLR scheduler to decay the learning rate and early stopping to regulate overfitting. In Part III, early stopping was done, too, but without the StepLR scheduler; therefore, it was more straightforward to do learning rate adjustments. From these architectural differences, it would appear that the VGG-13 model captures finer details and generalises better but at a higher computational cost.

3. **Provide the performance metrics and graphs (Step 5) and compare them with the results obtained in Part III**

**Training Accuracy:**
Part III: Was able to achieve the best training accuracy of around 89%.
Part IV: Higher train accuracy was achieved by the VGG-13 model, peaking at around 93.09%.
**Validation Accuracy:**
Part III: Peaked at approximately 88.86%.
Part IV: Peaked at around 88.25% with early stopping triggered at epoch 16 owing to convergence in validation performance.
**Test Accuracy:**
Part III: About 88.86%
Part IV: 88.31%, close enough to the result of Part III to indicate that both architectures generalize well.
**Precision, Recall, and F1 Score :**
**Part 3:**

```
Test Accuracy: 88.52%
Test Loss: 0.3165
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.74      0.69       604
           1       0.52      0.80      0.63       570
           2       0.89      0.85      0.87       559
           3       0.94      0.95      0.94       553
           4       0.91      0.91      0.91       552
           5       0.88      0.89      0.89       597
           6       0.94      0.95      0.94       555
           7       0.93      0.96      0.94       496
           8       0.96      0.93      0.95       558
           9       0.91      0.90      0.90       558
           A       0.95      0.97      0.96       553
           B       0.94      0.96      0.95       577
           C       0.95      0.96      0.95       577
           D       0.89      0.89      0.89       571
           E       0.97      0.93      0.95       571
           F       0.94      0.95      0.95       570
           G       0.95      0.91      0.93       571
           H       0.96      0.96      0.96       522
           I       0.69      0.59      0.64       553
           J       0.91      0.86      0.88       572
           K       0.97      0.95      0.96       548
           L       0.62      0.38      0.47       568
           M       0.99      0.98      0.98       573
           N       0.91      0.96      0.93       537
           O       0.70      0.60      0.65       574
           P       0.88      0.89      0.89       550
           Q       0.93      0.96      0.95       552
           R       0.97      0.96      0.96       543
           S       0.84      0.84      0.84       577
           T       0.97      0.97      0.97       593
           U       0.89      0.92      0.91       557
           V       0.93      0.95      0.94       549
           W       0.97      0.96      0.97       539
           X       0.97      0.97      0.97       542
           Y       0.92      0.88      0.90       561
           Z       0.87      0.86      0.86       558

    accuracy                           0.89     20160
   macro avg       0.89      0.89      0.89     20160
weighted avg       0.89      0.89      0.88     20160
```

**Part 4:**

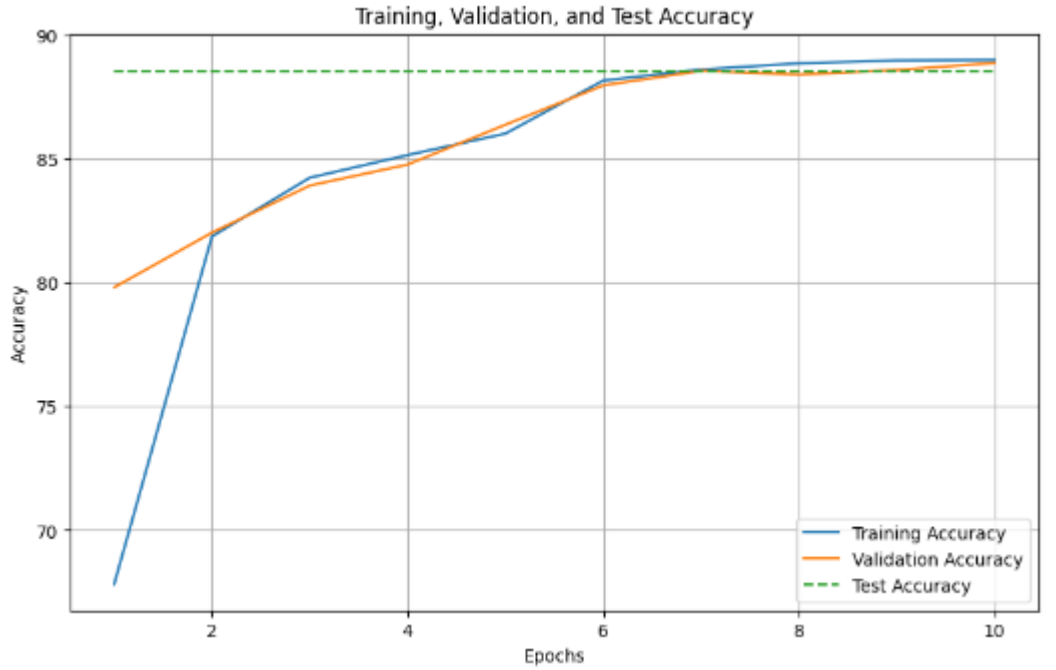
```
Test Results:
Test Accuracy: 88.31%
Precision: 0.8874
Recall: 0.8831
F1 Score: 0.8823
Training Time: 10843.75 seconds
```

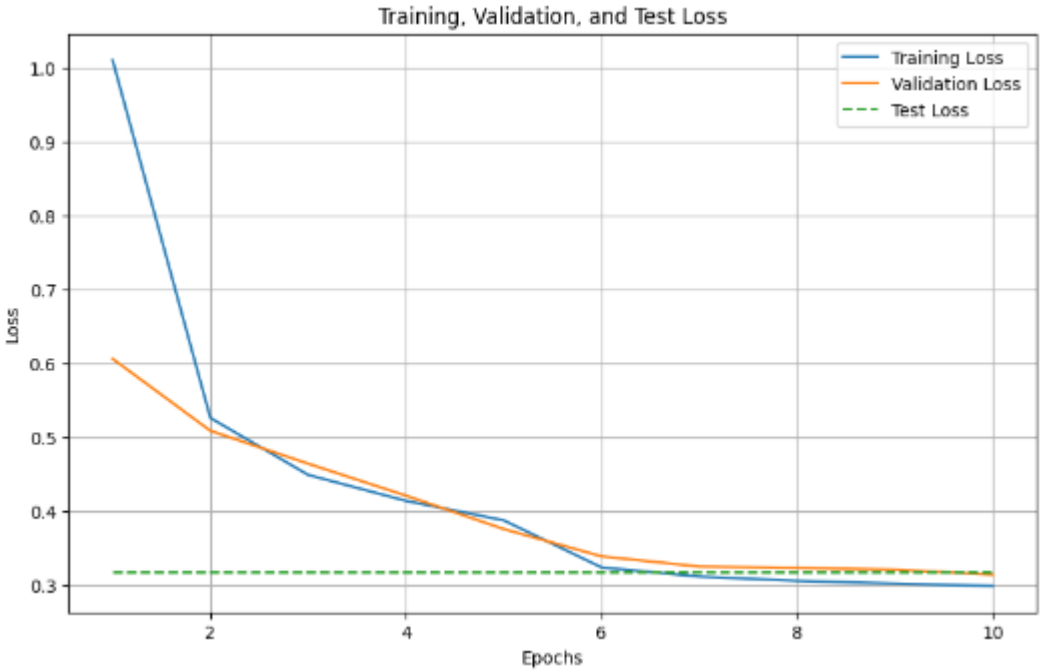
The metrics in the test set are pretty much similar in their performance for both models. This is to say that the VGG-13 architecture generalizes a bit better during training but converges to an accuracy similar to that of the CNN model from Part III.

**Graphs Comparison:**

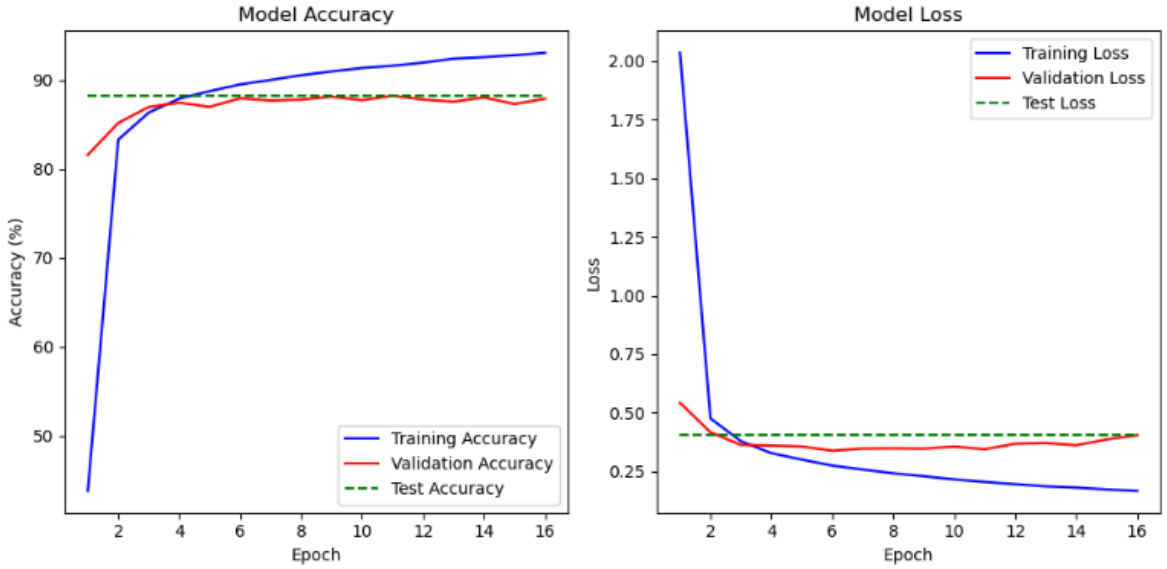
The following plots depict the performance metrics of the model that indicate the difference in their training dynamics:

**Part 3:**


Training, Validation, and Test Accuracy
Accuracy
Epochs
Training Accuracy
Validation Accuracy
Test Accuracy


Training, Validation, and Test Loss
Loss
Epochs
Training Loss
Validation Loss
Test Loss

**Part 4:**




**Model Accuracy:** Both VGG-13 models increase step by step in terms of accuracy during training. However, the VGG-13 model from Part IV achieved high accuracy after just a couple of epochs and had an earlier plateau, reflecting faster convergence. Also, its validation accuracy is less volatile, which already means a more stable learning process in Part IV.

**Loss of Model:** From both models, one can infer that there is a decrease in the initial losses of training and validation, while in Part IV, with additional dropout layers along with early stopping, the validation loss sags to a lower plateau, and it produced less overfitting when compared to Part III.

As in Part III, the AUC values, which are close to 1.0, are high for all classes from Part IV ROC curves, confirming that both models perform well in class discrimination by taking into consideration high true positive rates.