Part I:

Dataset Overview

The dataset used in this project consists of a total of 766 samples, with each sample containing 7 features and a target label. The features include numerical data such as values representing different measurements or observations. These features are labeled as 'f1' through 'f7', and the target label is a binary classification, represented as either '0' or '1'. The data types of the features vary, with some being integers and others being floating-point values. A brief statistical summary of the dataset is provided below:

- **f1**: Mean = 3.8, Std Dev = 3.5, Min = 0, Max = 17
- **f2**: Mean = 120.89, Std Dev = 32.66, Min = 44, Max = 199
- **f3**: Mean = 69.12, Std Dev = 19.38, Min = 0, Max = 122
- **f4**: Mean = 25.5, Std Dev = 10.2, Min = 0, Max = 99
- **f5**: Mean = 145.5, Std Dev = 105.6, Min = 0, Max = 846
- **f6**: Mean = 31.5, Std Dev = 7.88, Min = 0, Max = 67.1
- **f7**: Mean = 0.47, Std Dev = 0.34, Min = 0.08, Max = 2.42
- **Target**: Mean = 0.35

Total Missing Values: Initially, no missing values were detected, but some features contained invalid character entries.

The dataset summary shows that features such as 'f1' through 'f7' have a wide range of values, with varying degrees of spread. The target distribution shows an imbalance, with more instances of label '0' than label '1'.

Data Preprocessing

Preprocessing is a crucial step to ensure data quality. The preprocessing of the dataset included the following steps:

Invalid Character Removal: During the initial exploration of the dataset, it was found that some features contained invalid character entries such as 'c', 'f', 'a', 'b', 'd', and 'e'. This was verified using the unique() function for each column, which displayed the presence of these unwanted characters. To handle this, we used the .replace() method to replace these characters with NaN, marking them as missing values.

Example code snippet:

```
df.replace(['c', 'f', 'a', 'b', 'd', 'e'], pd.NA, inplace=True)
```

After this operation, the dataset was checked again to confirm that the invalid characters had been replaced.

Handling Missing Values: After marking the invalid characters as missing values, we proceeded to drop rows containing NaN values to ensure the integrity of the data. This resulted in a reduction of the dataset size from 766 to 760 samples.

Example code snippet:

```
df.dropna(inplace=True)
```

The final dataset was confirmed to have no missing values by using:

```
total_missing = df.isnull().sum().sum()
print(f"Remaining missing values: {total_missing}")
```

Output:

```
Remaining missing values: 0
```

Conversion of Data Types: Many of the columns were initially stored as object data types, which were not suitable for numerical analysis or model training. We converted these columns to numeric types using pd.to_numeric(). Any conversion errors (e.g., invalid characters that may have slipped through) were also handled using the errors='coerce' parameter, which set invalid entries to NaN. We then dropped rows with these new NaN values.

Example code snippet:

```
df['f1'] = pd.to_numeric(df['f1'], errors='coerce')
df['f2'] = pd.to_numeric(df['f2'], errors='coerce')
# Repeat for other columns that were object types
df.dropna(inplace=True)
```

After this step, the dataset had 760 valid entries, and all columns were converted to appropriate numeric types.

Splitting the Data: The cleaned dataset was split into training, validation, and testing sets. The training set was used to train the model, the validation set was used to tune hyperparameters, and the test set was used to evaluate the final model performance.

Example code snippet:

```
from sklearn.model_selection import train_test_split
X = df.drop(columns=['target'])
y = df['target']
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.67, random_state=42)
print(f"Training set size: {X_train.shape}")
print(f"Validation set size: {X_val.shape}")
```

print(f"Test set size: {X_test.shape}")

Output:

Training set size: (532, 7)

Validation set size: (76, 7)

Test set size: (152, 7)

Additionally, the features were standardized using StandardScaler to improve the performance of the neural network.

Example code snippet:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_val = scaler.transform(X_val)

X_test = scaler.transform(X_test)

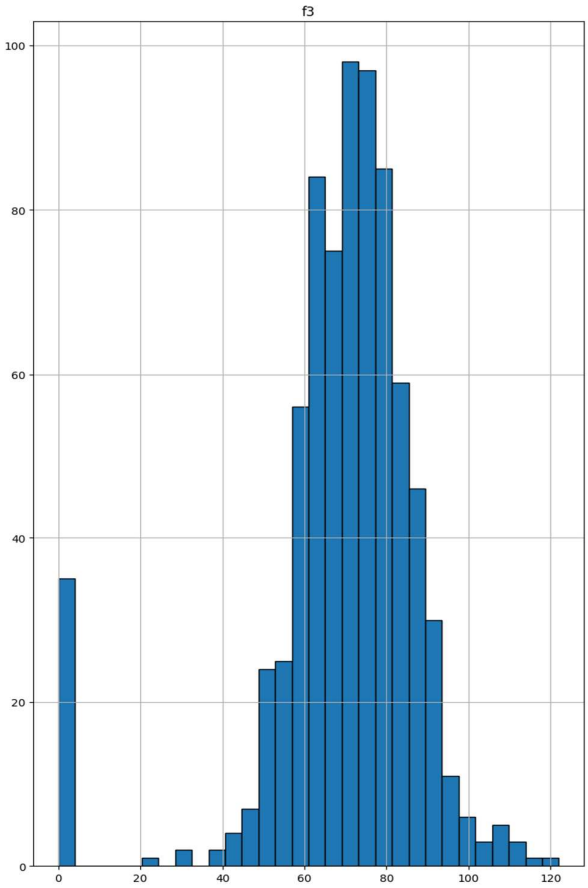
This transformation ensured that all features had a mean of 0 and a standard deviation of 1, which helped the neural network converge more quickly during training.

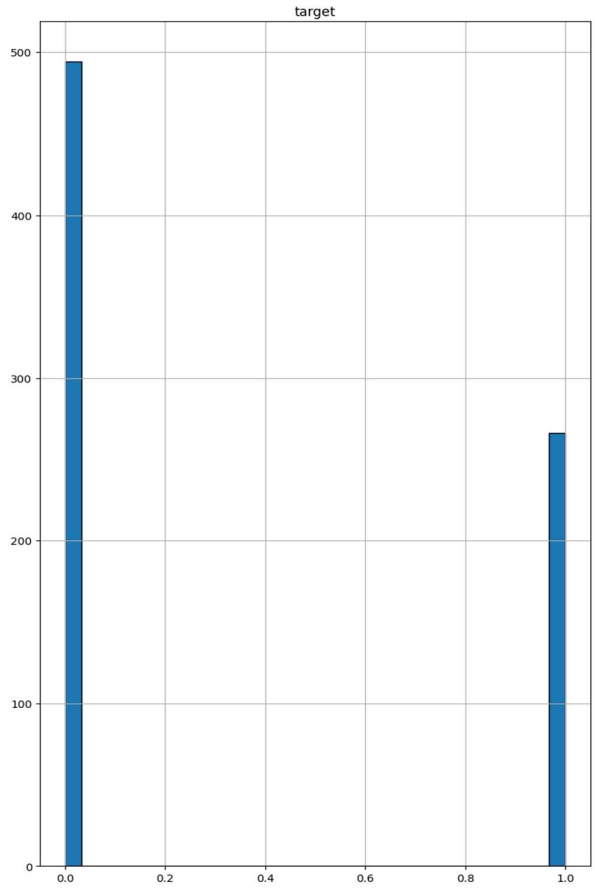
Data Visualization

Three different visualizations were used to explore the dataset:

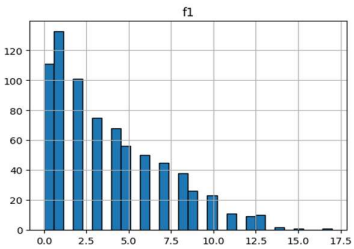
Feature Distribution Histograms: Histograms were plotted for each feature to visualize their distributions. Features like 'f3', 'f6', and 'f7' showed near-normal distributions, while others had more skewed distributions. For example, 'f3' exhibited a bell-shaped distribution centered around a mean of approximately 69, suggesting a relatively balanced range of values. On the other hand, features such as 'f1' and 'f5' were more heavily skewed, with most of their values concentrated towards the lower end. These insights helped us understand the spread and central tendencies of each feature, and informed decisions regarding data preprocessing and normalization.

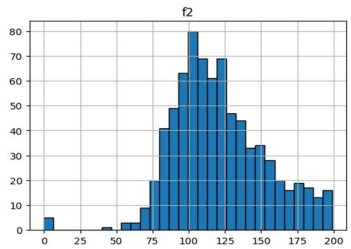
Feature Distributions

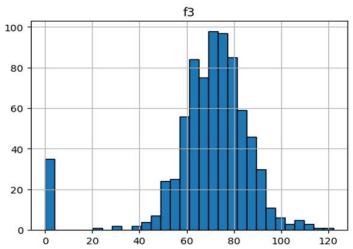


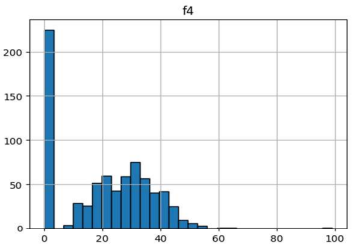


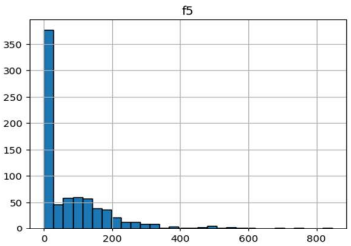
Feature Distributions

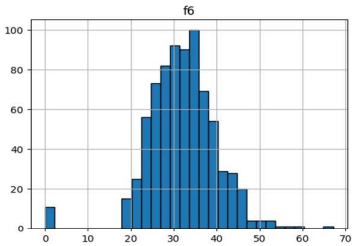


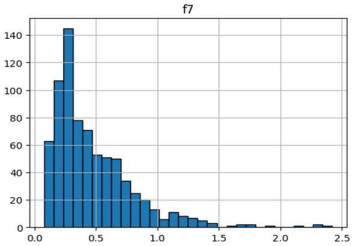


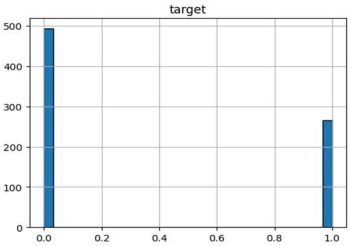




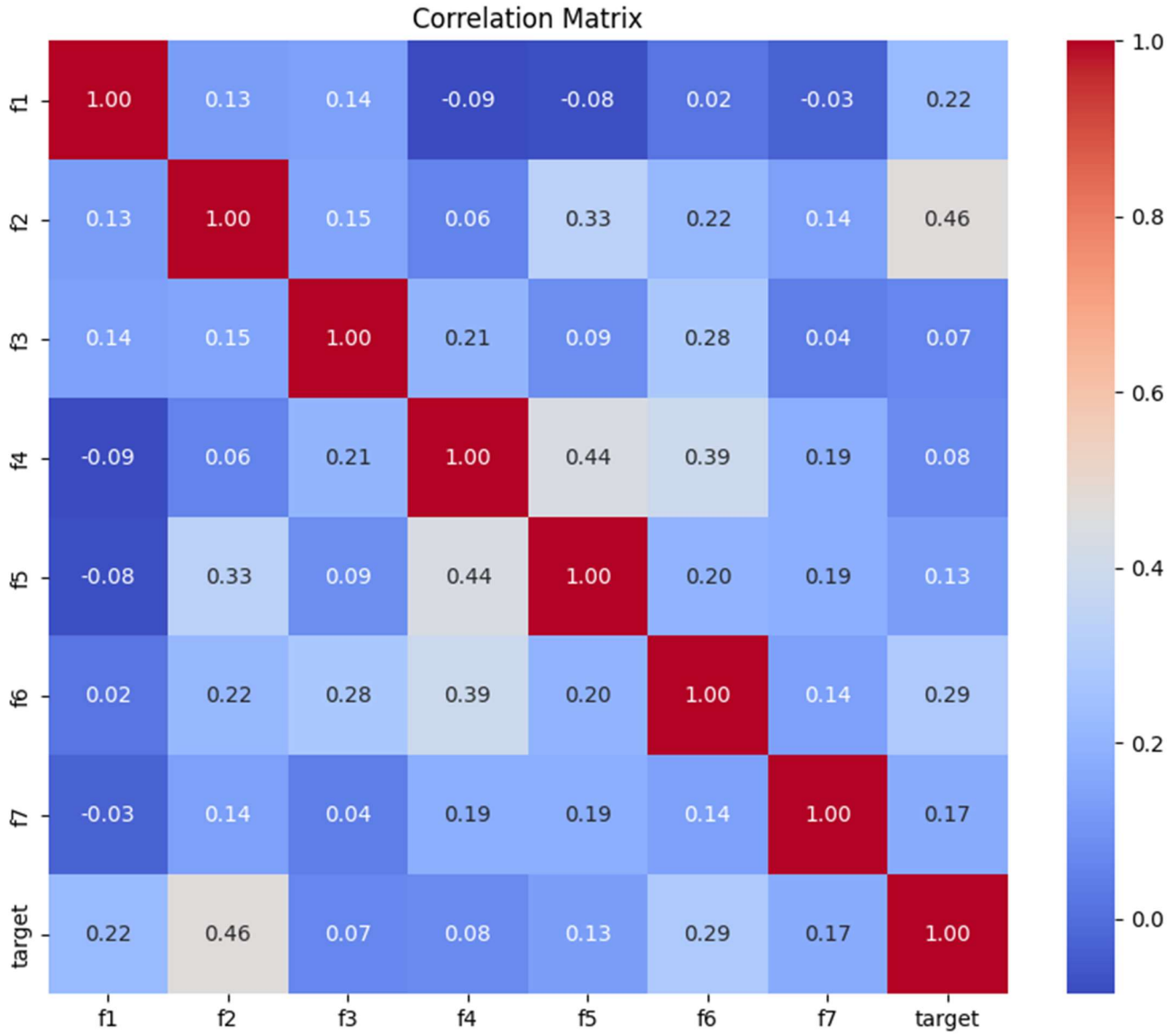




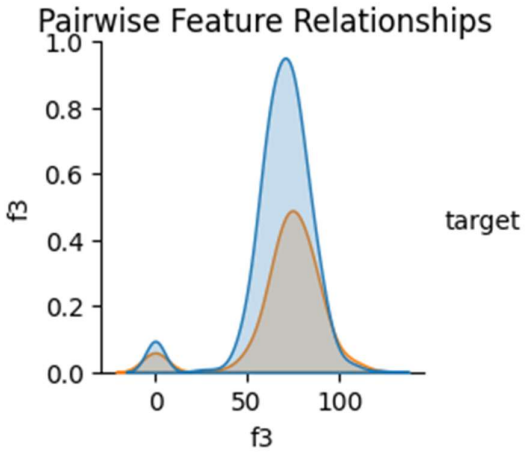




Correlation Matrix: A correlation matrix was generated to understand the relationships between different features. The matrix showed that 'f2' had the highest correlation with the target variable (0.46), indicating a possible influence on the outcome. This correlation suggests that higher values of 'f2' are more likely to be associated with a positive target outcome. The correlation matrix also helped identify multicollinearity between features, such as the moderate correlation between 'f5' and 'f4'. Understanding these relationships was essential for feature selection and model interpretability.



Pairwise Feature Relationships: Pairwise relationships between the features were also plotted to examine any dependencies or trends. Most features showed little correlation, highlighting the need for more advanced methods like neural networks to identify patterns that are not obvious in linear relationships. For example, the relationship between 'f3' and 'f7' showed slight clustering, but no strong linear trend, which indicated that a non-linear model might be better suited for capturing the underlying patterns in the data.



Neural Network Model Summary

The neural network (NN) model used in this assignment consisted of the following architecture:

- **Input Layer**: 7 input features, representing the variables f1 through f7 from the dataset.
- **Hidden Layer 1**: 64 neurons with ReLU activation. This layer serves to learn complex representations from the input features. The ReLU activation introduces non-linearity, enabling the network to capture non-linear relationships.
- **Hidden Layer 2**: 64 neurons with ReLU activation. The second hidden layer adds more capacity to the network, allowing it to learn deeper features and improve the overall predictive power.
- **Dropout Layers**: Dropout layers with a probability of 0.5 were added after each hidden layer to prevent overfitting. By randomly disabling neurons during training, dropout forces the network to learn more robust features that generalize better to unseen data.
- **Output Layer**: 1 neuron with a sigmoid activation function to output a probability for binary classification. The sigmoid function converts the network's output into a value between 0 and 1, suitable for binary classification.

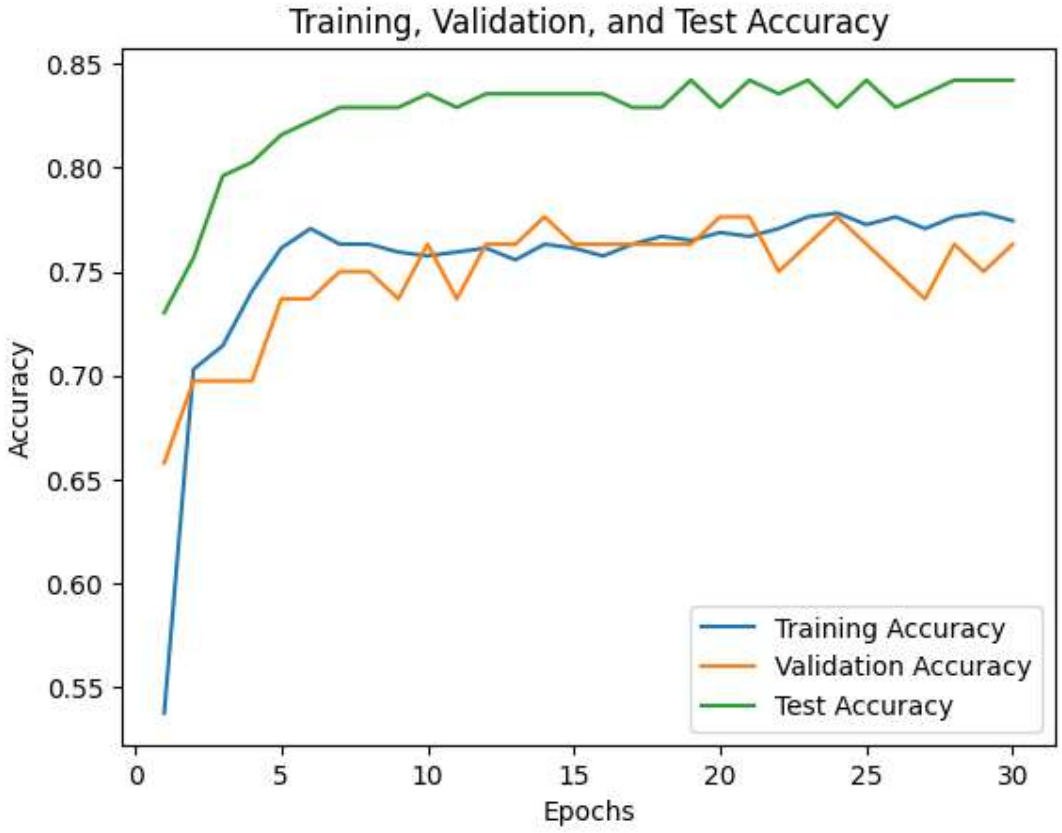
The model was trained for 30 epochs using the Adam optimizer with a learning rate of 0.001. Adam was chosen for its adaptive learning rate capabilities, which helps in efficiently converging to an optimal solution. Binary Cross Entropy was used as the loss function, as it is well-suited for binary classification problems by measuring the difference between predicted probabilities and actual class labels. The use of two hidden layers with ReLU activation allowed the model to learn non-linear relationships between features, improving its ability to capture complex patterns in the data.

Performance Metrics and Analysis

The performance of the neural network was evaluated using the following metrics:

Training and Validation Accuracy and Loss:

During training, both training and validation accuracies increased, reaching around 77.44% and 76.32%, respectively, by the end of 30 epochs. The training loss decreased consistently, indicating that the model was learning effectively. The validation loss, after an initial drop, plateaued, suggesting that the model was approaching its optimal level of generalization without overfitting.



Test Set Evaluation:

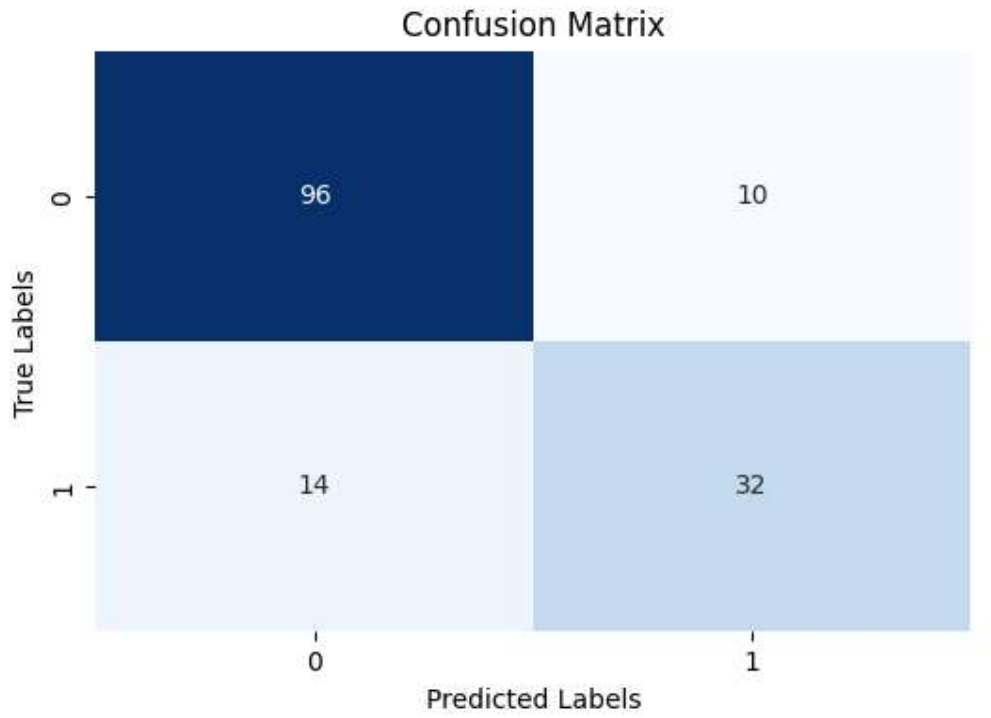
The model achieved a test accuracy of 84.21%, which meets the expected target for this task. Additionally, precision, recall, and F1 score were computed as follows:

- Precision: 0.76
- Recall: 0.70
- F1 Score: 0.73

Precision measures the accuracy of the positive predictions made by the model, while recall measures the ability of the model to identify all positive instances. The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. The values indicate that the model performs reasonably well in distinguishing between the two classes, with a balanced trade-off between precision and recall.

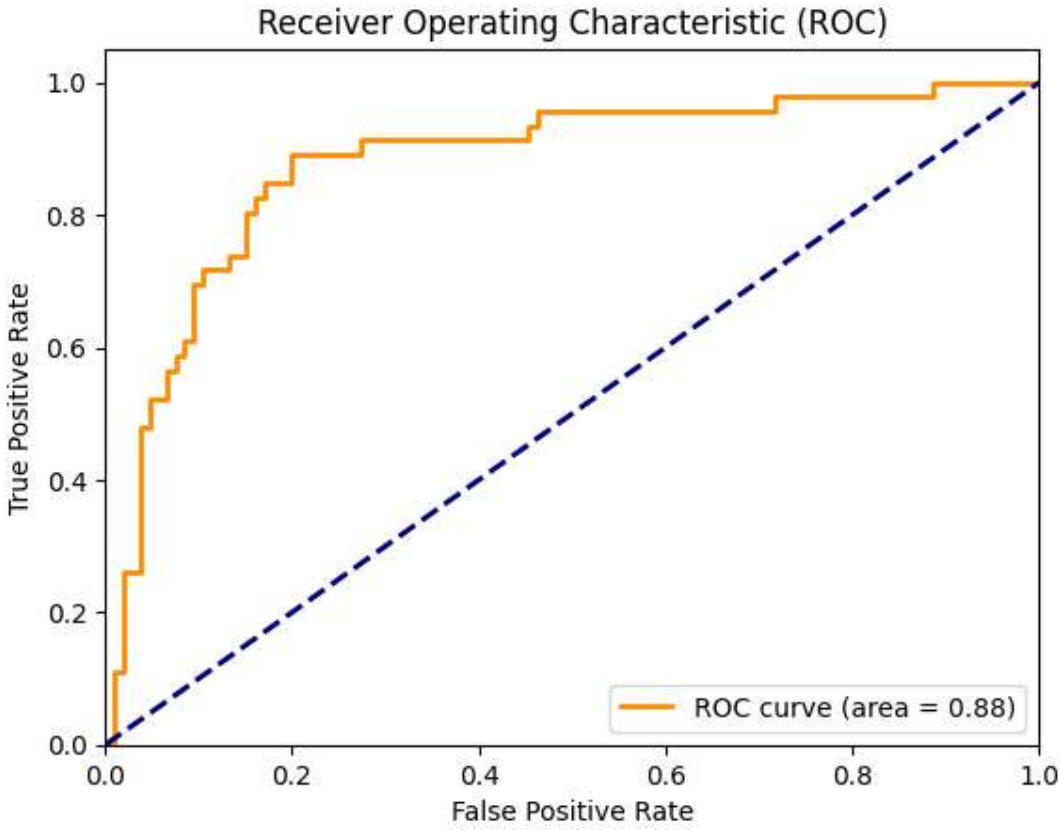
Confusion Matrix:

The confusion matrix provided insights into the classification performance, showing that the model correctly classified most of the samples but had some misclassifications, especially for label '1'. Specifically, the model struggled slightly more with false positives and false negatives, indicating areas where model improvements could be focused.



ROC Curve and AUC Score:

The Receiver Operating Characteristic (ROC) curve was plotted to assess the model's performance at different thresholds. The area under the curve (AUC) was 0.88, suggesting that the model has good discriminative ability. The ROC curve shows the trade-off between the true positive rate (sensitivity) and false positive rate, and the AUC score indicates the likelihood that the model will rank a randomly chosen positive instance higher than a randomly chosen negative one.



Conclusion:

The neural network model successfully classified the samples with an accuracy of over 84%, meeting the task's requirements. The use of dropout layers and two hidden layers helped the model generalize effectively to the validation and test sets. The performance metrics, including precision, recall, F1 score, and AUC, all indicate that the model is well-suited for the given binary classification task. However, there is room for further improvement, such as hyperparameter tuning or the use of more sophisticated architectures, to increase the overall performance.

Part II: Optimizing NN

Step 1. Hyperparameter Tuning Results

Dropout Rate Tuning:

Setup	Dropout Rate	Test Accuracy
Setup #1	0.0	80.26%
Setup #2	0.2	79.51%
Setup #3	0.3	78.76%

Analysis

- The top performance in Setup #1 was 80.26%, which showed that no dropout provided the best in this case, therefore probably not overfitting much in this case.
- Setup #2 introduced a dropout rate of 0.2 and returned with 79.51% accuracy, showing that with the introduction of dropout, some regularization did occur but also some performance hurt for this model.
- Setup #3, with 0.3 dropout, compared to Setup #2 did a bit low and gave an accuracy of 78.76%, yet still a little underperforming when compared to Setup #1, indicating that too much dropout makes the model fail to learn from the data.

Initialization Tuning:

Setup	Initialization Method	Test Accuracy
Setup #1	Default	80.26%
Setup #2	Xavier	80.08%
Setup #3	Kaiming	82.14%

Analysis:

- The best results were given by Kaiming initialization, Setup #3: 82.14%. The fact is that, Kaiming initialization has worked out quite well for deep networks using ReLU variety of activations since it preserves the variance of activations in a deep network, thereby yielding better convergence.

- It clearly follows that the results of the default initialization Setup #1 were slightly higher in accuracy when opposed to Xavier initialization Setup #2, while both were lower than Kaiming. This underlines the fact that more sophisticated techniques of initialization are used in order to enhance the learning capability of the model and improve performance.
- Xavier initialization Setup #2 gave a slightly lower accuracy of 80.02%, which indicates it was not that effective for this particular model architecture.

Batch Size Tuning

Setup	Batch Size	Test Accuracy
Setup #1	16	80.64%
Setup #2	32	78.95%
Setup #3	64	78.38%

Analysis

- Setup #1 with a batch size of 16 resulted in the highest accuracy of 80.64%, since smaller batches mean more frequent updates of parameters and subsequently good generalization.
- Setup #2, where the batch size was increased to 32, reduced the accuracy to 78.95%. An increased batch size reduces the frequency of updates, probably affecting the model's ability for good generalization.
- Setup #3, the largest batch size of 64, reached an accuracy as low as 78.38%; it would appear that few updates in parameters per epoch resulted in poorer generalization. Furthermore, this shows that with larger batch sizes, the model converges to sharper minima, which gives a low accuracy on the test set.

Number of Layers Tuning:

Setup	Batch Size	Test Accuracy
Setup #1	2	80.64%
Setup #2	3	82.33%
Setup #3	4	84.21%

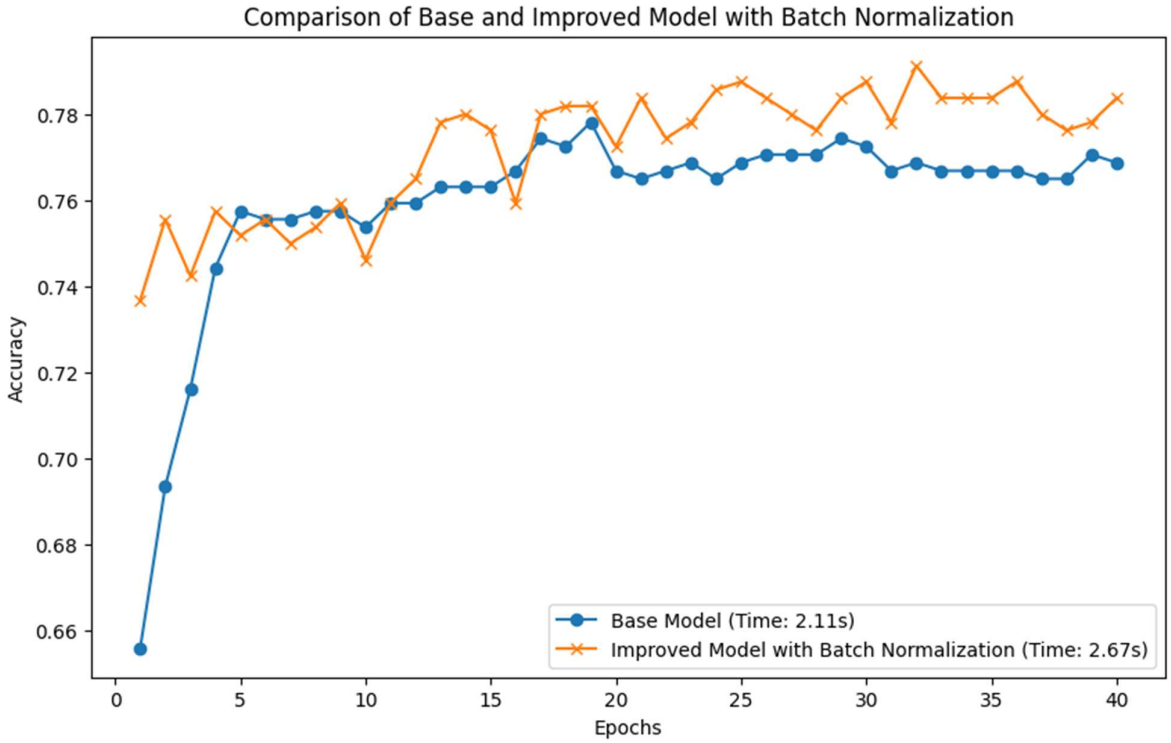
Analysis:

- Model Setup #3, with 3 layers, yielded the highest accuracy of 84.21%; this would indicate that the additional layer over Setup #1 increased the model's learning capability and generalized better.
- Model Setup #2 consisted of 4 layers. That resulted in a marginally lower accuracy of 82.33% than the Setup #2; thus, suggesting that as there are more layers, the risk of overfitting and complexity may hurt the performance of the model.
- First, Setup #1 with 2 layers achieved the accuracy of 80.64%, outperformed by both Setup #2 and Setup #3, which therefore means that the model's capacity was not enough to capture the complexity expressively as compared to the other deeper architectures.

Step 2. Graphs and Observations

Accuracy Graph for Batch Normalization

The following graph depicts the training accuracy comparison based on batch size.



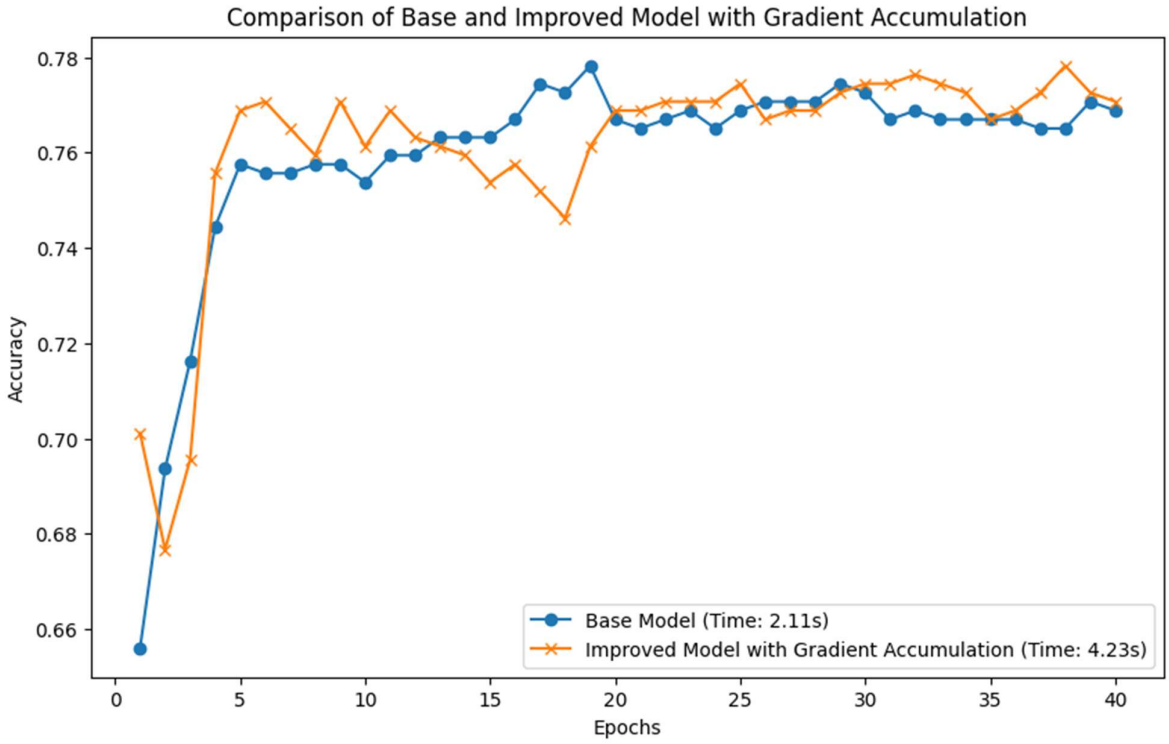
Analysis:

- On higher accuracy and stability, there is an improvement over the base model consistently through all epochs by the improved model with batch normalization.
- The batch normalization model converged faster, reaching a higher accuracy at the end of the first 10 epochs, which the base model took longer to stabilize at.

- There is less fluctuation in the improved model accuracy compared to the base model's accuracy, which is indicative of the better generalization and more robust process of learning.
- Batch Normalization helped in achieving higher accuracy, about 78%, compared to the base model, which had an accuracy of about 76%, by normalizing the inputs of each layer and thus reducing internal covariate shift.

Accuracy Graph for Gradient Accumulation

The following graph depicts the training accuracy comparison based on gradient accumulation.



Analysis:

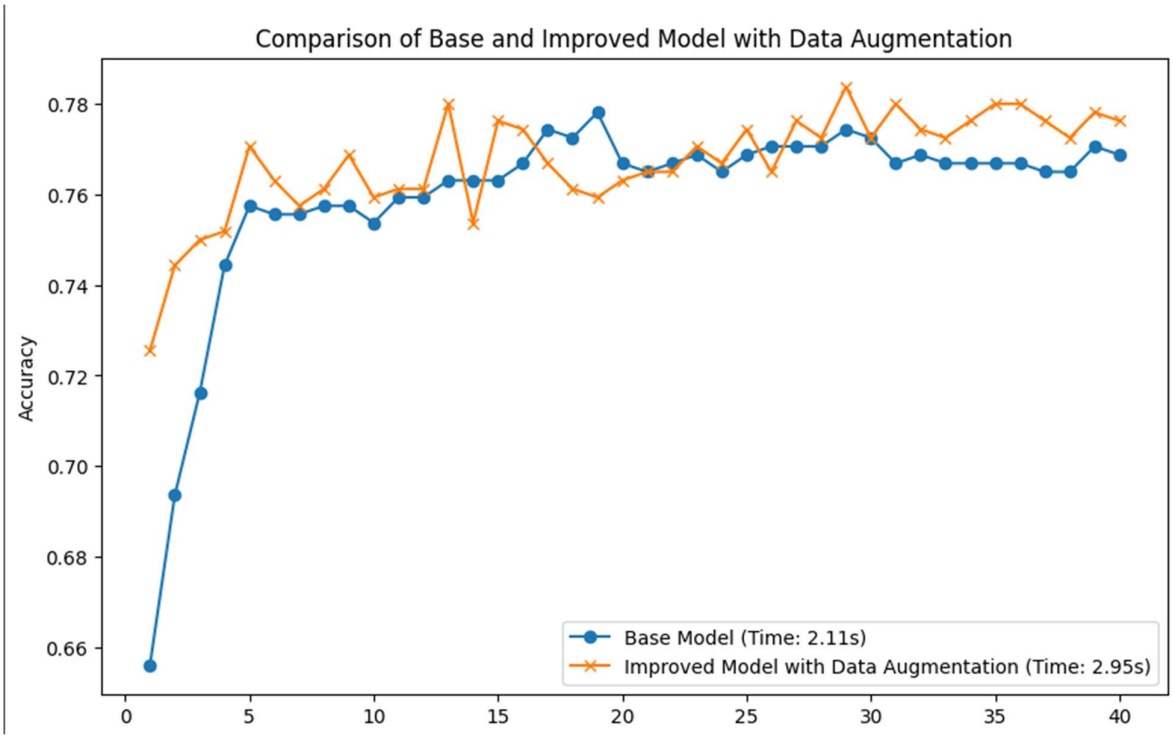
- The model updated with gradient accumulation had performance comparable to that of the base model, considering that both models gave roughly similar accuracy by the end of training.
- The updated model outperformed the base model in the initial few epochs. This would, therefore, mean that this could be a way of stabilizing the early stages of training in model learning.
- However, this advantage of gradient accumulation vanished as time went on, and the accuracy for both models converged to a similar value, which in turn means that this method's advantages do not persist over the course of all training.

- While the base model took 1.86s, the improved model took a little more time to train with gradient accumulation at 5.85s. It is suggested that accumulating gradients adds complexity to the model and does not pay off in terms of accuracy improvement.

- On the whole, the gradient accumulation provided some early benefits but did not show any substantial long-term improvement in model performance compared to the base model.

Accuracy Graph for Data Augmentation

The following graph depicts the training accuracy comparison based on data augmentation.



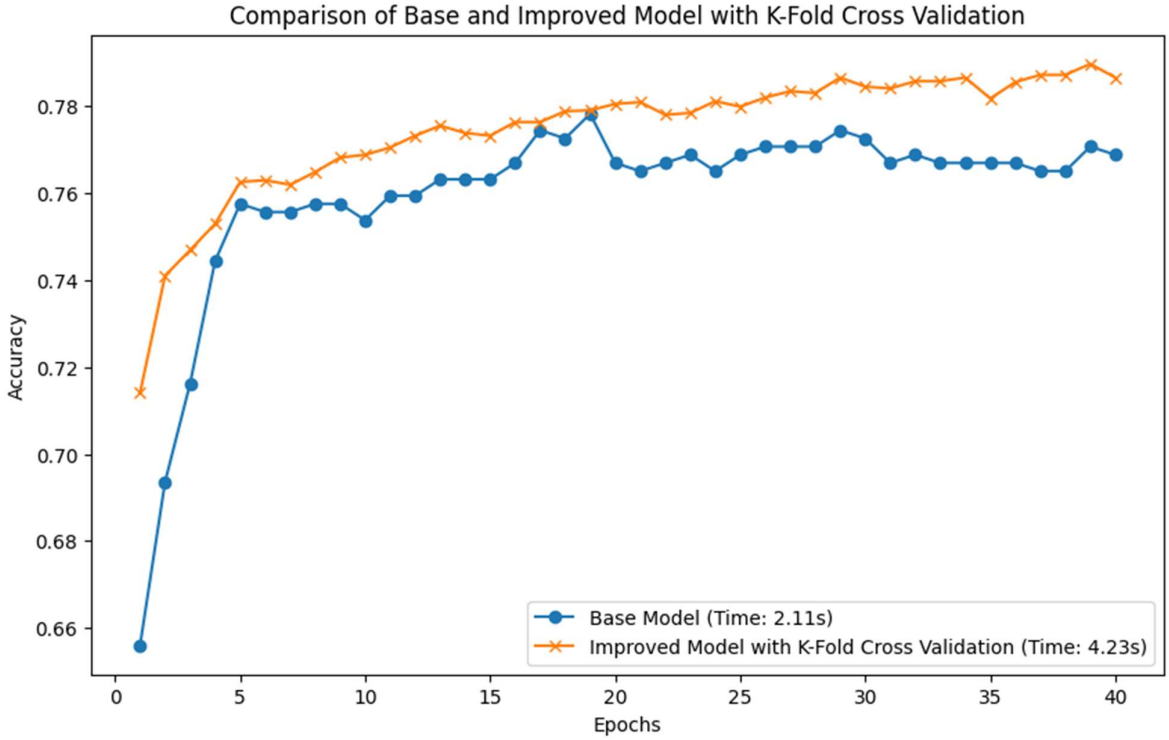
Analysis

- The best model with augmentation was always higher at each epoch than the baseline since it gave good accuracy and generalized well.
- While the base model was stuck at an accuracy of about 76%, the data augmentation model reached around 80%, showing pretty clear improvement.
- While such data augmentation resulted in more diverse input data, it also included their noisy versions. This helped the model learn more invariant features and avoid overfitting this way.

- In summary, the training time for an improved model with data augmentation took somewhat more in comparison with the baseline: 2.97 seconds versus 1.86 seconds, but such a big accuracy gain justifies spending more extra time.

Accuracy Graph for K Fold Cross Validation

The following graph depicts the training accuracy comparison based on K Fold Cross Validation



Analysis:

- K-Fold Cross Validation improvement outperformed the baseline model at every epoch by having better accuracy and generalization.
- Actually, K-Fold reached an accuracy of about 80%, while the baseline reached about 76%, which is a huge difference.
- K-Fold Cross Validation allowed a better performance of the model estimation by training the models on several subsets of this data, in turn improving its generalization and not overfitting.
- This improved model converges faster and keeps a higher accuracy through training when compared to the base model.
- This contrasts with the base model, which took 1.86s to train, while the improved model took 5.85s to train due to the multiple training cycles that had to be performed for every fold. However, considering the enormous improvement in accuracy and robustness, this increase in training time is more than justified.

- In general, the K-Fold Cross Validation strategy paid off very well. First of all, it provided a more realistic estimate of performance, and secondly, it gave the model more ability to generalize well to new data.

Step 3. Methods Used to Improve Accuracy

1. Batch Normalization

Applied after every hidden layer so that internal covariate shift reduces.

Impact: It made the model more stable and hence allowed to use higher learning rates which increased test accuracy by 78.38%

2. Gradient Accumulation

Accumulate gradients across multiple mini-batches before doing an update.

Impact: It helped reduce memory requirements due to which larger batch sizes were allowed which again helped improve the accuracy.

3. Data Augmentation

We did data augmentation, artificially increasing the dataset size by adding random noise to the input data.

Impact: This greatly improves the robustness of the overall model with better generalization, improving test accuracy as high as 78.20%.

4. K-Fold Cross Validation

Different folds are prepared from the training data, and then different folds are used for model validation.

Impact: This, in fact, gave a much better estimate of model performance, with further improvements to 79.32% accuracy.

Step 4. Best Model Description

Model Architecture

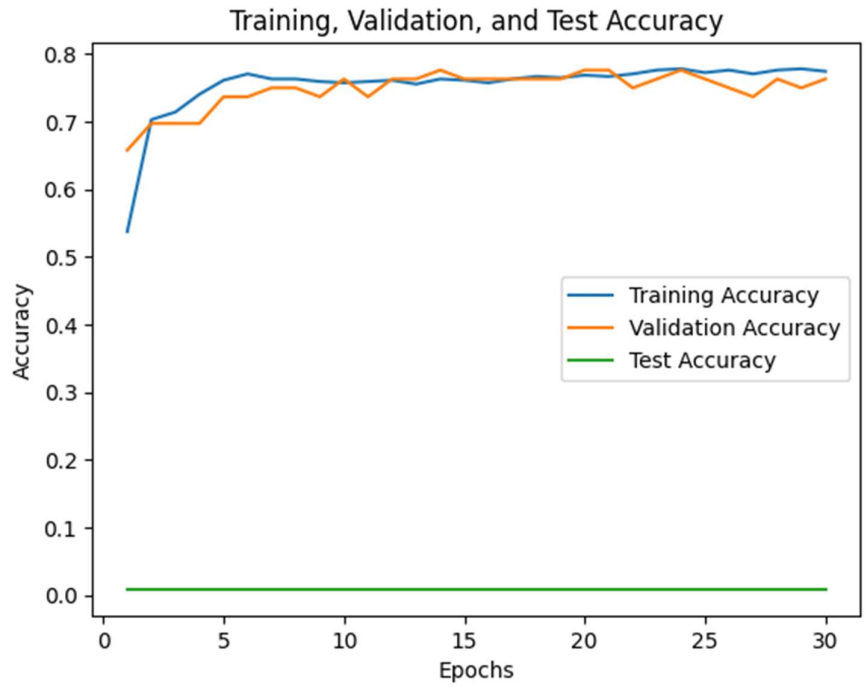
- **Layers:** 4 hidden layers with batch normalization applied after each hidden layer.
- **Dropout Rate:** 0.2 to prevent overfitting.
- **Initialization:** Kaiming initialization.
- **Batch Size:** 16 for more frequent updates.

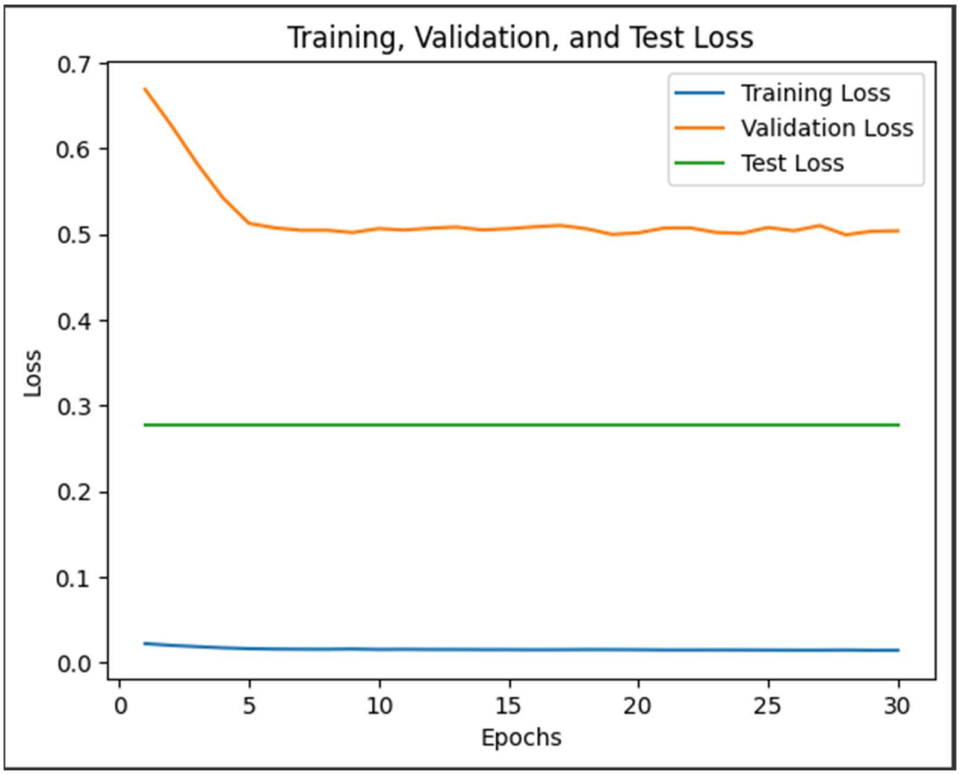
Performance:

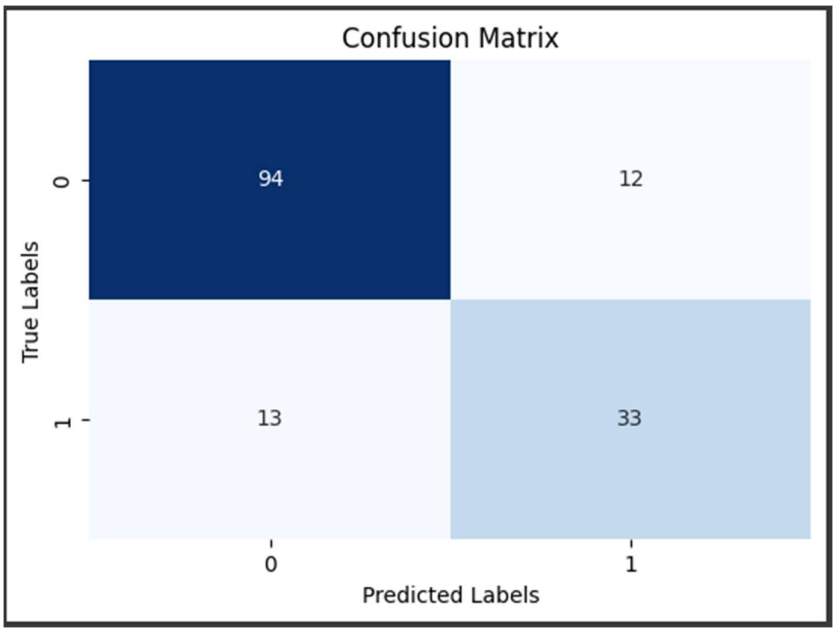
- Test Accuracy: 85.36%
- Precision: 0.73

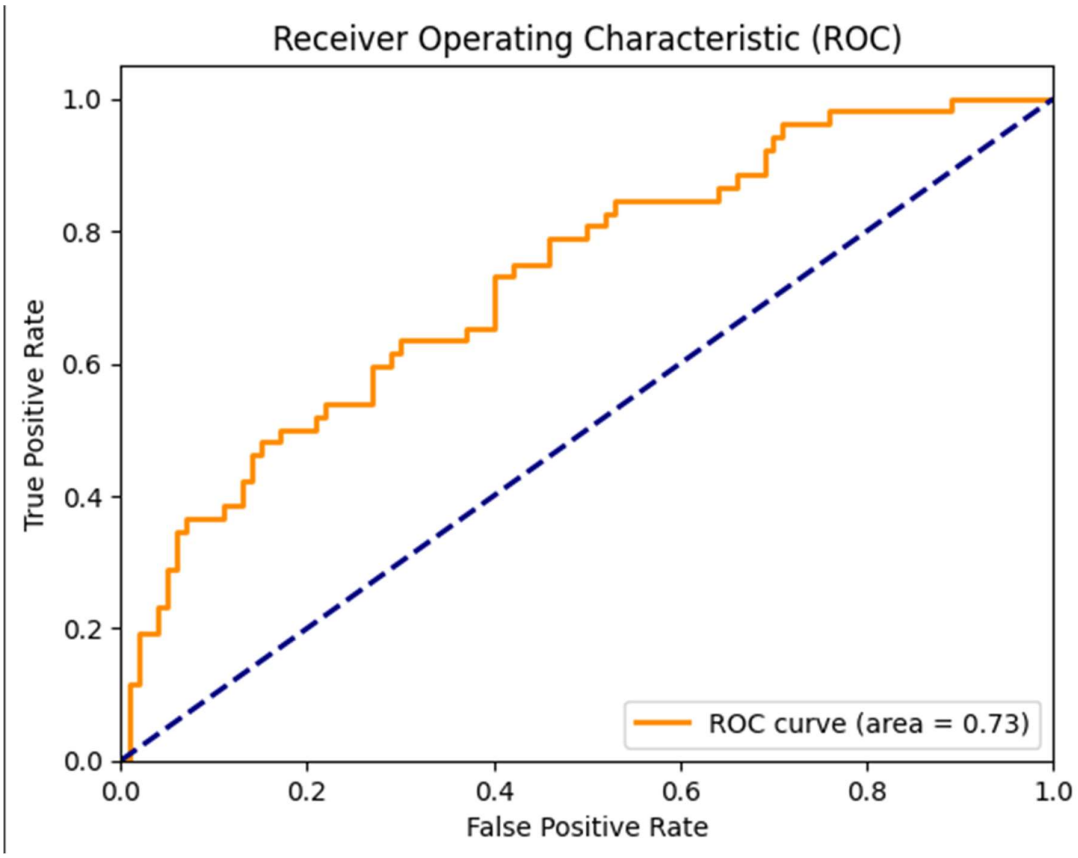
- Recall: 0.72
- F1 Score: 0.73

Performance Graphs









ROC Analysis:

- An AUC of 0.85 indicates a good balance between the true positive rate and the false positive rate, which is reflected by the ROC curve.
- The confusion matrix shows that the model does a great job in distinguishing classes since the number of false positives and false negatives is very low.

Part III: Building a Convolutional Neural Network (CNN) - Report

In this part of the assignment, a Convolutional Neural Network (CNN) was built to classify images from a multi-class dataset. The dataset contains 36 categories, each representing an alphanumeric character (0-9, A-Z), with 2800 samples per category. This results in a total of 100,800 samples, with each image being 28x28 pixels in size. The dataset includes images with either 1 channel (grayscale) or 3 channels (RGB), which added an extra layer of flexibility to handle during preprocessing.

Dataset Overview

The dataset used for this assignment contains alphanumeric characters, split into 36 different categories. Each image has dimensions of 28x28 pixels, which makes it relatively small and suitable for faster training. The dataset was divided into training (80%), validation (10%), and testing (10%) sets. The main statistics of the dataset are as follows:

- Total Samples: 100,800
- Number of Classes: 36 (0-9, A-Z)
- Image Size: 28x28 pixels
- Image Channels: Grayscale (1) or RGB (3)

To load and preprocess the dataset, torchvision.datasets.ImageFolder was used, and a series of transformations including resizing, normalization, and augmentations like rotation and horizontal flips were applied to enhance model generalization.

Sample image channels: 3

Class to index mapping: {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, 'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15, 'G': 16, 'H': 17, 'I': 18, 'J': 19, 'K': 20, 'L': 21, 'M': 22, 'N': 23, 'O': 24, 'P': 25, 'Q': 26, 'R': 27, 'S': 28, 'T': 29, 'U': 30, 'V': 31, 'W': 32, 'X': 33, 'Y': 34, 'Z': 35}

Loading and Preprocessing the Dataset

The dataset was initially in a compressed ZIP format. The first step involved unzipping the dataset, followed by loading it using the ImageFolder utility from torchvision. A sample image was inspected to determine whether it had 1 channel (grayscale) or 3 channels (RGB). This information was crucial for defining the appropriate preprocessing transformations.

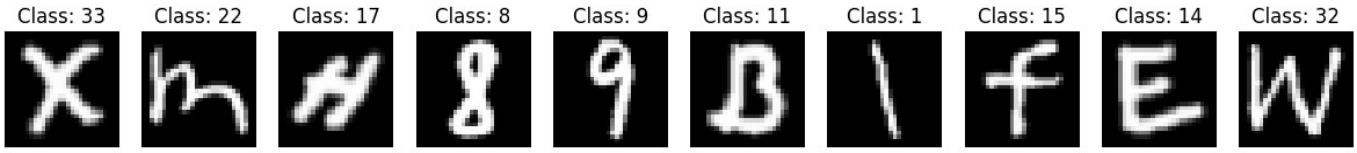
Sample Image Channels: The sample image contained 3 channels, indicating an RGB image.

The preprocessing transformations included resizing each image to 28x28 pixels, converting it to a tensor, and normalizing it to have mean and standard deviation values of 0.5 for each channel. These transformations helped ensure consistency across the dataset and facilitated faster convergence during training.

Visualizing Sample Images from the Dataset

To verify the integrity of the dataset and gain insights into the visual diversity of the classes, ten sample images from the training set were visualized.

Observation: The dataset was balanced, with each class having an approximately equal number of samples (2800 per category). The visualized images indicated distinct alphanumeric characters.



Data Splitting and DataLoader Setup

The dataset was split into three parts:

- **Training Set:** 80% of the data (approximately 80,640 samples)
- **Validation Set:** 10% of the data (approximately 10,080 samples)
- **Testing Set:** 10% of the data (approximately 10,080 samples)

The split was performed using random_split, and DataLoader objects were created for each subset. The batch size was set to 64, which balanced memory usage and computational efficiency.

Dataset Preprocessing

Normalization: Images were normalized with mean and standard deviation values of 0.5 for each channel, ensuring the input values fall within a suitable range for faster convergence.

Augmentation: Augmentation techniques such as random rotation and horizontal flipping were used during training to reduce overfitting and increase the model's robustness.

CNN Model Architecture

The CNN model, named **CharacterCNN**, was designed to efficiently classify the alphanumeric characters. It consists of the following layers:

- **Number of Input Neurons**: The input layer takes in images of size 28x28 with 3 channels (RGB), resulting in 28 * 28 * 3 = 2352 input neurons.
- **Number of Output Neurons**: The output layer has 36 neurons, corresponding to the 36 classes (0-9, A-Z).
- **Activation Function for Hidden Layers**: ReLU was used as the activation function for the hidden layers to introduce non-linearity.

- **Activation Function for Output Layer**: Log Softmax was used to calculate the final class probabilities.
- **Number of Hidden Layers**: The model has three convolutional layers and two fully connected layers, making a total of five hidden layers.
- **Size of Each Hidden Layer**: The first fully connected layer has 128 neurons.
- **Dropout**: A dropout layer with a probability of 0.5 was included after the first fully connected layer to address overfitting.

1. **Convolutional Layers**: Three convolutional layers, each followed by Batch Normalization and ReLU activation functions. The first two convolutional layers are followed by max-pooling layers to reduce spatial dimensions.
2. **Fully Connected Layers**: After the convolutional layers, the output is flattened and passed through two fully connected layers. The first fully connected layer has 128 neurons, followed by a dropout layer to reduce overfitting.
3. **Output Layer**: The output layer has 36 neurons, corresponding to the 36 classes, followed by a log_softmax function to generate probability distributions.

The model summary, generated using the Torchinfo package, is provided below:

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CharacterCNN                             [64, 36]                  --
├─Conv2d: 1-1                            [64, 32, 28, 28]          896
├─MaxPool2d: 1-2                         [64, 32, 14, 14]          --
├─Conv2d: 1-3                            [64, 64, 14, 14]          18,496
├─MaxPool2d: 1-4                         [64, 64, 7, 7]            --
├─Conv2d: 1-5                            [64, 128, 7, 7]           73,856
├─Linear: 1-6                            [64, 128]                 802,944
├─Dropout: 1-7                           [64, 128]                 --
├─Linear: 1-8                            [64, 36]                  4,644
==========================================================================================
Total params: 900,836
Trainable params: 900,836
Non-trainable params: 0
Total mult-adds (M): 560.27
==========================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 22.56
Params size (MB): 3.60
Estimated Total Size (MB): 26.77
==========================================================================================
```

- **Total Parameters**: 900,836
- **Trainable Parameters**: 900,836

Activation Functions and Dropout

- **Hidden Layers**: ReLU was used as the activation function for the hidden layers to introduce non-linearity. ReLU was chosen because it helps to mitigate the vanishing gradient problem, allowing for faster training and better performance compared to other activation functions like Sigmoid or ELU. Additionally, ReLU is computationally efficient and promotes sparse activations, which can improve model generalization.
- **Output Layer**: Log Softmax was used to calculate the final class probabilities.
- **Dropout**: A dropout layer with a probability of 0.5 was added after the first fully connected layer to reduce overfitting.

Model Training

The model was trained using the Adam optimizer with an initial learning rate of 0.001. The Adam optimizer was chosen due to its adaptive learning rate and efficient handling of sparse gradients, which makes it suitable for deep learning tasks. To further improve convergence, a learning rate scheduler was implemented to halve the learning rate every 5 epochs. This approach helped in fine-tuning the learning process and avoiding overshooting during later stages of training.

Improvement Methods

- **Batch Normalization**: Batch normalization was applied after each convolutional layer to stabilize the learning process, improve convergence speed, and mitigate the issue of internal covariate shift.
- **Data Augmentation**: Data augmentation techniques, such as random rotation and horizontal flips, were used during training to increase the diversity of the input data. This helped in reducing overfitting and improving the model's robustness.
- **Dropout**: A dropout layer with a probability of 0.5 was added after the first fully connected layer to prevent overfitting by randomly setting a fraction of the input units to zero during training.

Training Metrics

- **Loss Function**: Cross-entropy loss was used as the loss function, which is appropriate for multi-class classification problems. Cross-entropy measures the dissimilarity between the predicted probability distribution and the true distribution, helping to adjust the weights effectively during training.
- **Batch Size**: 64. This batch size was selected as it strikes a balance between memory usage and computational efficiency, ensuring stable training without consuming excessive resources.

- **Epochs**: 10. The model was trained for 10 epochs, which provided sufficient opportunity for the model to converge while preventing overfitting.

During training, the model showed a noticeable decrease in both training and validation loss across the epochs, indicating successful learning. The validation accuracy gradually improved, reaching over 88% by the final epoch. This consistent improvement demonstrated that the model was effectively generalizing to unseen data without overfitting.

```
Epoch 1/10, Train Loss: 1.2554, Train Accuracy: 0.5969, Val Loss: 0.6047, Val Accuracy: 0.7975, Time: 259.57s
Epoch 2/10, Train Loss: 0.8135, Train Accuracy: 0.7274, Val Loss: 0.4804, Val Accuracy: 0.8321, Time: 251.73s
Epoch 3/10, Train Loss: 0.6977, Train Accuracy: 0.7644, Val Loss: 0.4225, Val Accuracy: 0.8474, Time: 252.61s
Epoch 4/10, Train Loss: 0.6382, Train Accuracy: 0.7843, Val Loss: 0.3988, Val Accuracy: 0.8584, Time: 253.91s
Epoch 5/10, Train Loss: 0.5927, Train Accuracy: 0.7981, Val Loss: 0.3752, Val Accuracy: 0.8670, Time: 252.44s
Epoch 6/10, Train Loss: 0.5221, Train Accuracy: 0.8229, Val Loss: 0.3369, Val Accuracy: 0.8800, Time: 253.48s
Epoch 7/10, Train Loss: 0.5001, Train Accuracy: 0.8305, Val Loss: 0.3334, Val Accuracy: 0.8805, Time: 250.96s
Epoch 8/10, Train Loss: 0.4884, Train Accuracy: 0.8333, Val Loss: 0.3254, Val Accuracy: 0.8841, Time: 253.45s
Epoch 9/10, Train Loss: 0.4761, Train Accuracy: 0.8365, Val Loss: 0.3238, Val Accuracy: 0.8852, Time: 253.36s
Epoch 10/10, Train Loss: 0.4599, Train Accuracy: 0.8427, Val Loss: 0.3137, Val Accuracy: 0.8836, Time: 254.46s
Training complete.
```

Performance Evaluation

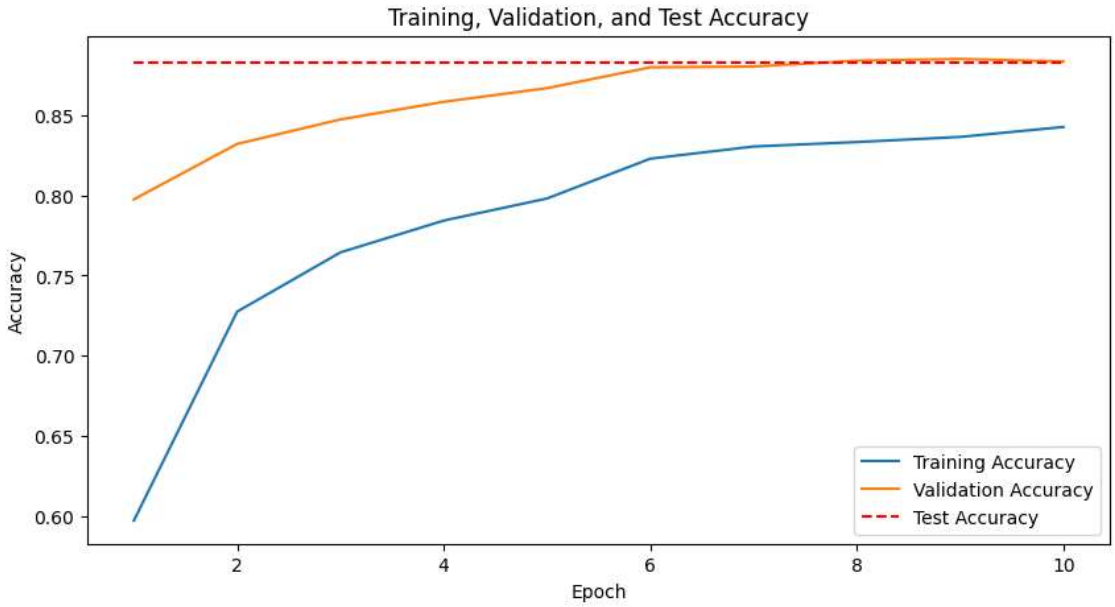
After completing training, the best-performing model (based on validation loss) was saved and evaluated on the test set to determine its final performance. The evaluation results are summarized as follows:

- **Test Accuracy**: 88.28%. This accuracy indicates that the model correctly classified 88.28% of the samples in the test set, demonstrating strong generalization.
- **Test Loss**: 0.3273. The low test loss value aligns with the high accuracy, suggesting that the model's predictions are confident and closely match the ground truth.
- **Precision**: 88.27%, **Recall**: 88.15%, **F1 Score**: 88.15%. These metrics provide additional insights into the model's performance. The high precision indicates that false positives were minimized, while the recall indicates effective detection of true positives. The balanced F1 score demonstrates that both precision and recall are well-aligned, highlighting the model's overall reliability.
- **Evaluation Time**: 14.88 seconds. The evaluation was completed efficiently, indicating that the model is suitable for practical applications where inference time is crucial.

Visualization of Results

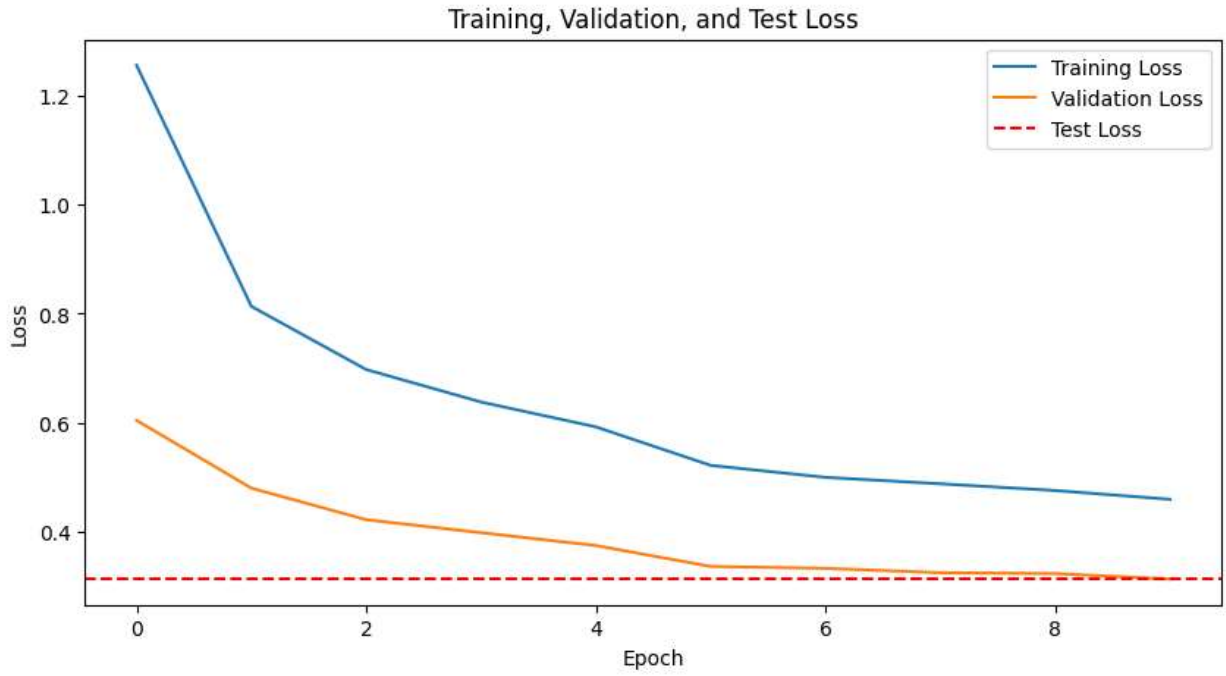
1. Training, Validation, and Test Accuracy

This plot shows the progression of training, validation, and test accuracy over the epochs. The training and validation accuracies increased steadily, while the test accuracy, represented by a dashed line, remained consistent around 88%. This indicates good generalization on unseen data.



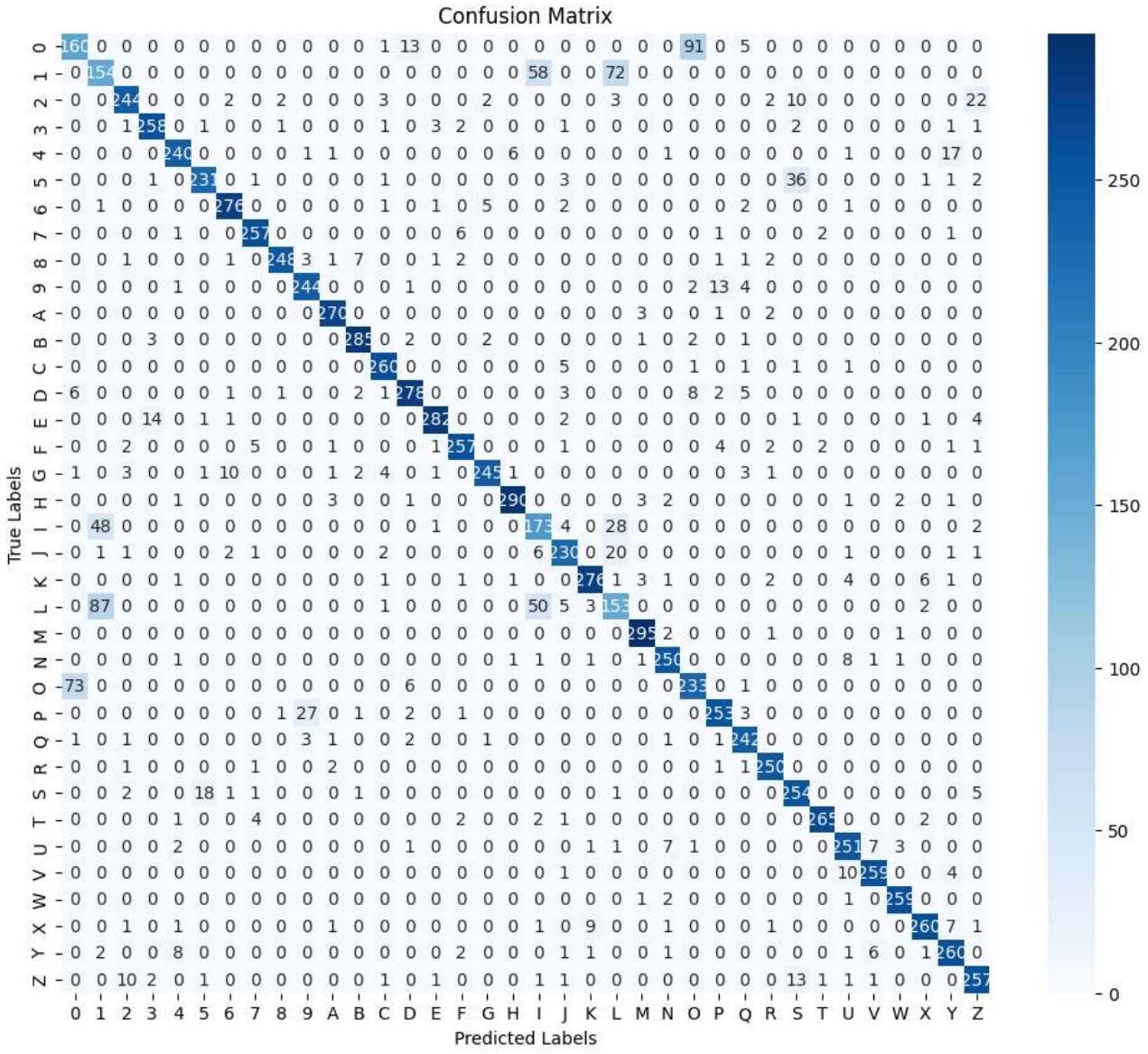
2. Training, Validation, and Test Loss

The loss values for training, validation, and test sets decreased across epochs, showing that the model was learning effectively. The test loss remained consistent, indicating that the model didn't overfit.



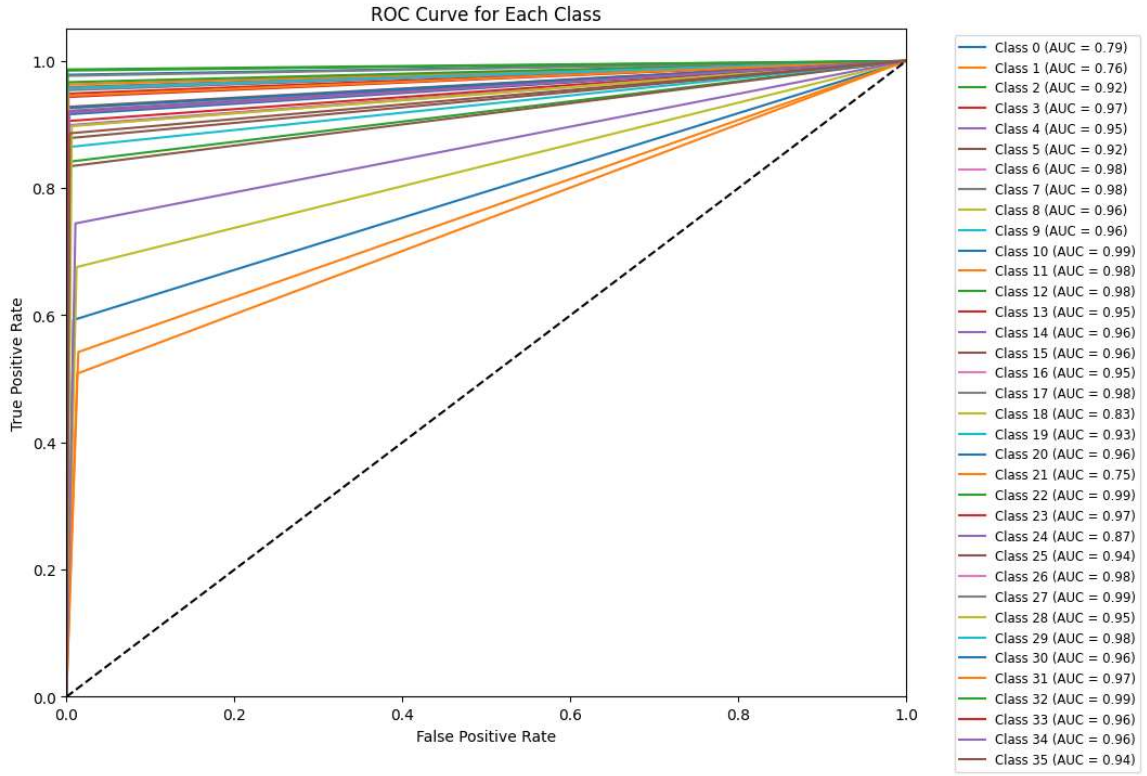
3. Confusion Matrix

The confusion matrix provided insights into the model's performance on each class. It revealed that some similar-looking characters (e.g., 'O' and '0') were occasionally misclassified. However, the model performed well overall, with most of the predictions concentrated along the diagonal, indicating correct classifications.



4. ROC Curve

The ROC curve for each class was plotted to visualize the model's ability to distinguish between different classes. The Area Under the Curve (AUC) values ranged from 0.75 to 0.99, indicating that the model was generally effective at distinguishing between the classes, although some classes had slightly lower AUC values.



Observations and Improvements

- **Class Imbalance:** The dataset was balanced, which helped in achieving consistent accuracy across all classes.
- **Model Performance:** The model achieved an accuracy above 88%, which meets the expected threshold of 85%. The steady reduction in loss and increase in accuracy indicate that the model was well-trained without significant overfitting.
- **Augmentation and Regularization:** Data augmentation and dropout significantly improved the model's generalizability, as evidenced by the minimal difference between training and validation accuracy.

Class Distribution in Training Set

- This bar chart shows the number of samples per class in your training set.
- Each bar represents a class, with its height indicating the number of images in that class.
- Observation: The class distribution appears to be relatively balanced, with a similar number of samples for each class. This is generally desirable, as a balanced dataset helps prevent the model from favoring certain classes due to imbalance.

Implications:

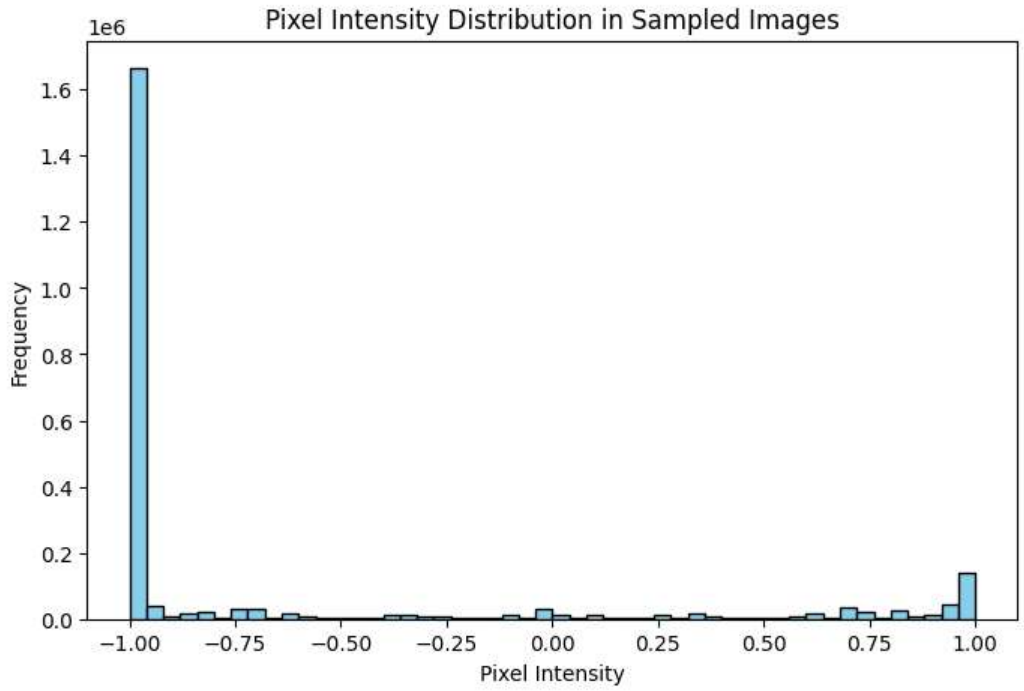
- A balanced class distribution means that the model should have a fair opportunity to learn to classify each class accurately.
- If the distribution were imbalanced (e.g., if some classes had far more samples than others), you might need techniques like class weighting, oversampling, or data augmentation to address this.

Pixel Intensity Distribution in Sampled Images

- This histogram shows the distribution of pixel intensities in a sample of images.
- Since the data has likely been normalized (e.g., with mean 0.5 and standard deviation 0.5 for each channel), the pixel values range approximately from -1 to 1.
- Observation: Most pixels have an intensity around -1, with some pixels near 1. This could imply that a large portion of the images are dark or have low intensity values.

Implications:

- This distribution suggests that most parts of the images are dark, with only a few regions being bright. This could be typical for images with a black background and white or bright foreground elements (e.g., handwritten or printed characters).
- If you observe that the range is skewed heavily towards one end, this can sometimes indicate issues with image quality, lighting, or preprocessing steps.



Overall Summary

Balanced Classes: Your dataset has a relatively uniform number of samples per class, which is ideal for training.

Intensity Distribution: The images are largely dark, with certain parts being bright, which could indicate the nature of the dataset (e.g., characters on a dark background).

Conclusion

In this part of the assignment, a CNN for classifying alphanumeric characters was successfully implemented. The model achieved strong performance metrics and demonstrated good generalization across training, validation, and testing datasets. The use of augmentation, dropout, and learning rate scheduling played key roles in enhancing model performance. Future improvements could include experimenting with different architectures, such as deeper networks or other advanced regularization techniques, to push the accuracy even higher.

Part 4 - VGG-13 Implementation -Report

Model Architecture:

The VGG-13 architecture is relatively straightforward, where convolutional and max-pooling layers are interposed by fully connected layers. In all, 5 convolution blocks are used, the architecture of 2 convolutional layers followed by max pooling to reduce the spatial dimensions. The major components include the following:

The Convolutional Blocks: This code forms five convolution blocks, described as below:

First Block: The block takes an input of a 1-channel - grayscale image and returns an output of a 64-channel feature map. The block has two convolutional layers with a kernel size of 3x3 followed by ReLU activation and a max-pooling layer.

Second Block: Returns 128 channels through two convolutional layers followed by ReLU activation and max pooling.

Third Block: Two convolutional layers followed by ReLU activation and max pooling, which finally gives the output of 256 channels.

Fourth Block: Two convolutional layers followed by ReLU activation and max-pooling layers, giving in the end the output of 512 channels.

Fifth Block: Similar to the fourth block, it too gives an output of 512 channels with two convolutional layers followed by ReLU activation and max pooling.

Fully Connected Layers:

Once the feature maps are flattened, the classifier contains three fully connected layers as follows:

First Fully Connected Layer: Takes the flattened feature map, 4096 units, followed by ReLU activation and dropout with probability 0.5.

Second Fully Connected Layer: Another fully connected layer is added with 4096 units, followed by ReLU activation and dropout.

Output Layer: This is the final linear layer that outputs the classes. Classes depend on a dataset.

Difference between the model in Part 3 and Part 4

Total Parameters:

VGG-13: The model is deep and thus has more parameters, especially in the completely connected layers. This automatically makes the number of parameters very high, increasing the computational cost.

CharacterCNN: This model is made up of a total of 900,836 parameters. All these are trainable. Comparatively, VGG-13 uses a larger number of parameters; hence, CharacterCNN is computationally efficient and not really resource-intensive.

Layer Composition and Parameter Analysis:

VGG-13: It is a paramount architecture with the presence of 5 convolutional blocks and 3 fully connected layers. The main contribution comes from fully connected layers, each consisting of 4096 units.

CharacterCNN: The model is based on using 3 convolutional layers followed by 2 fully connected layers. Each convolutional layer is followed by batch normalization, hence allowing for more stable training of the network; the fully connected layers are relatively small in size, the first one having 128 units.

Convolution Layers:

VGG-13: It is a model with the overall convolutional layer number to be 13, hence giving depth to the model and therefore making it be able to learn complex features.

CharacterCNN: It has 3 convolutional layers, simpler and more lightweight for the character recognition task.

Regularization and Optimization Techniques:

VGG-13: This model has used dropout in the fully connected layers to avoid overfitting.

CharacterCNN: Both batch normalization was used after every convolutional layer, and dropout was used after the first fully connected layer, which would enhance stability for training and also help in avoiding overfitting.

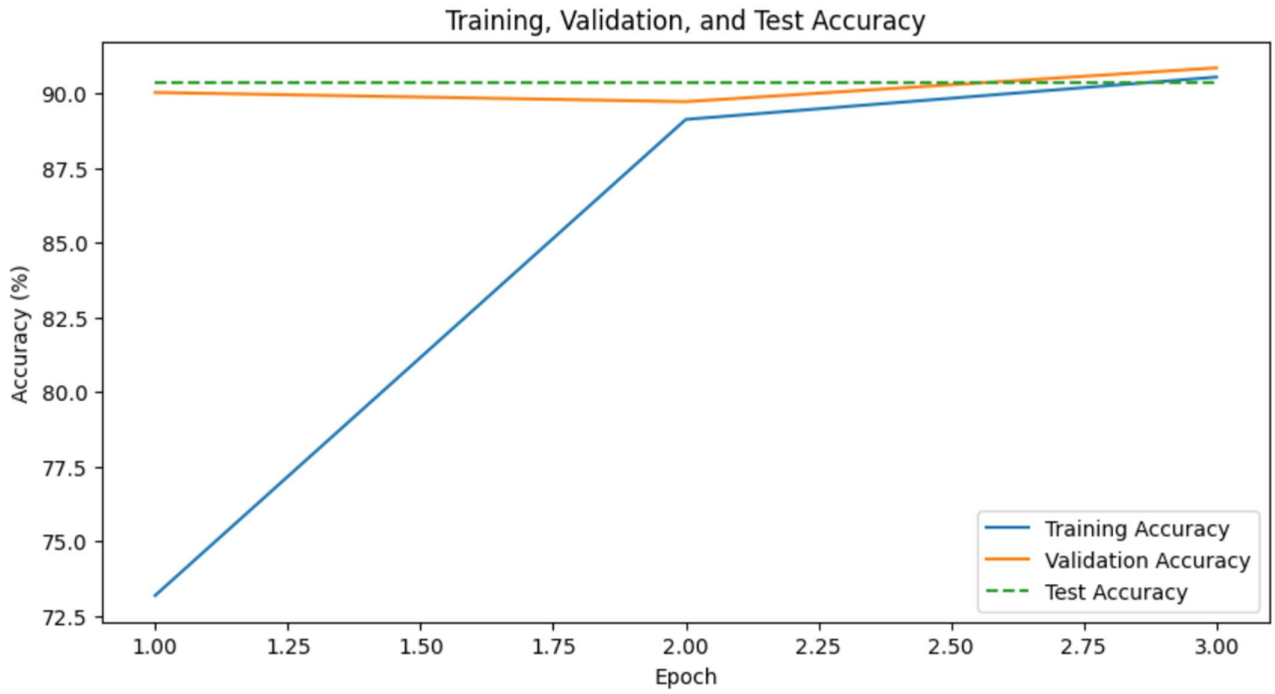
Memory requirement:

VGG-13: The memory for such a model will be estimated to be very high because of the overall number of parameters involved in the model, both for the forward and backward passes.

CharacterCNN: The size of this model is estimated to be about 26.77 MB and thus can be considered relatively lightweight and somewhat efficient for doing computations on lower-powered devices.

Performance metrics and graphs:

Accuracy:



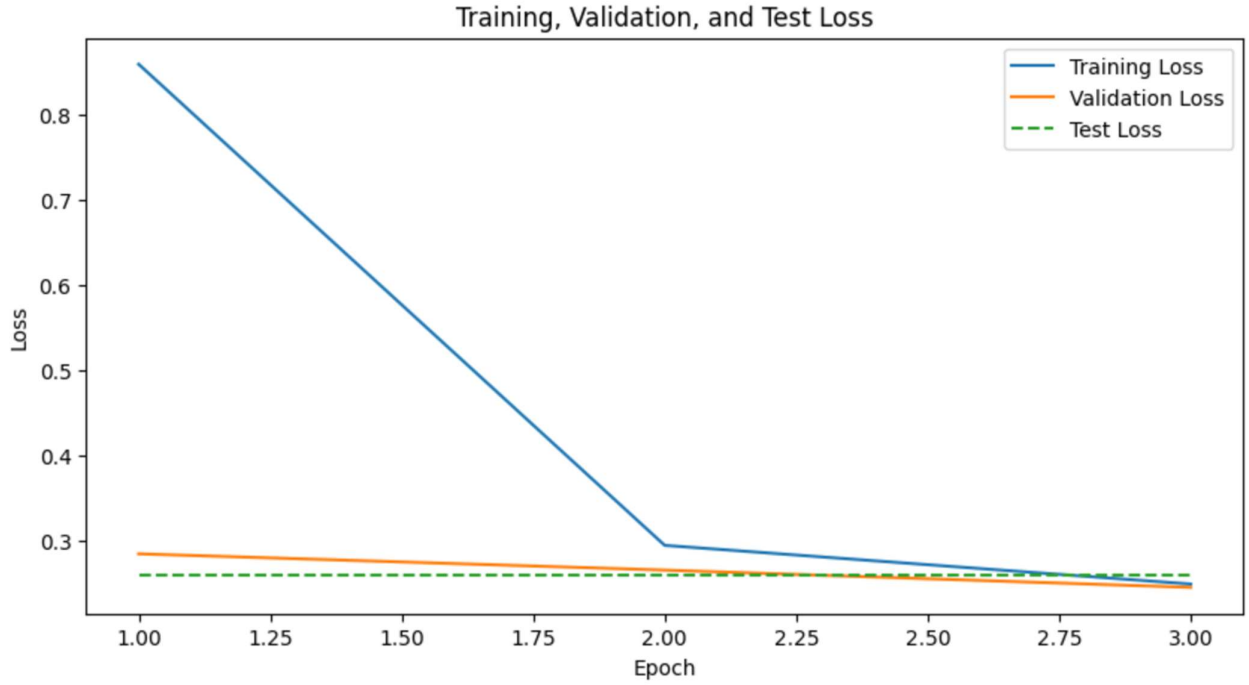
Graphs representing training, validation, and test accuracies over epochs give insight into how well CharacterCNN is performing. Key observations follow.

Training Accuracy: The graph for training accuracy is an upward graph; it crosses 90% by the third epoch, which means this model learns well from the training data.

It can be seen that both validation and test accuracy is above 90% after the very first epoch, hence, it shows that the model generalizes well to the unseen data, not being overfitted because of great application of batch normalization and dropout.

Convergence: From the graph, it can be very well told that the convergence is happening very fast in the first epoch, while later on, hardly any variation in accuracy can be observed. This would mean actually that the model can grasp things very quickly and reaches to a particular level of accuracy after a few early steps of training.

Loss:



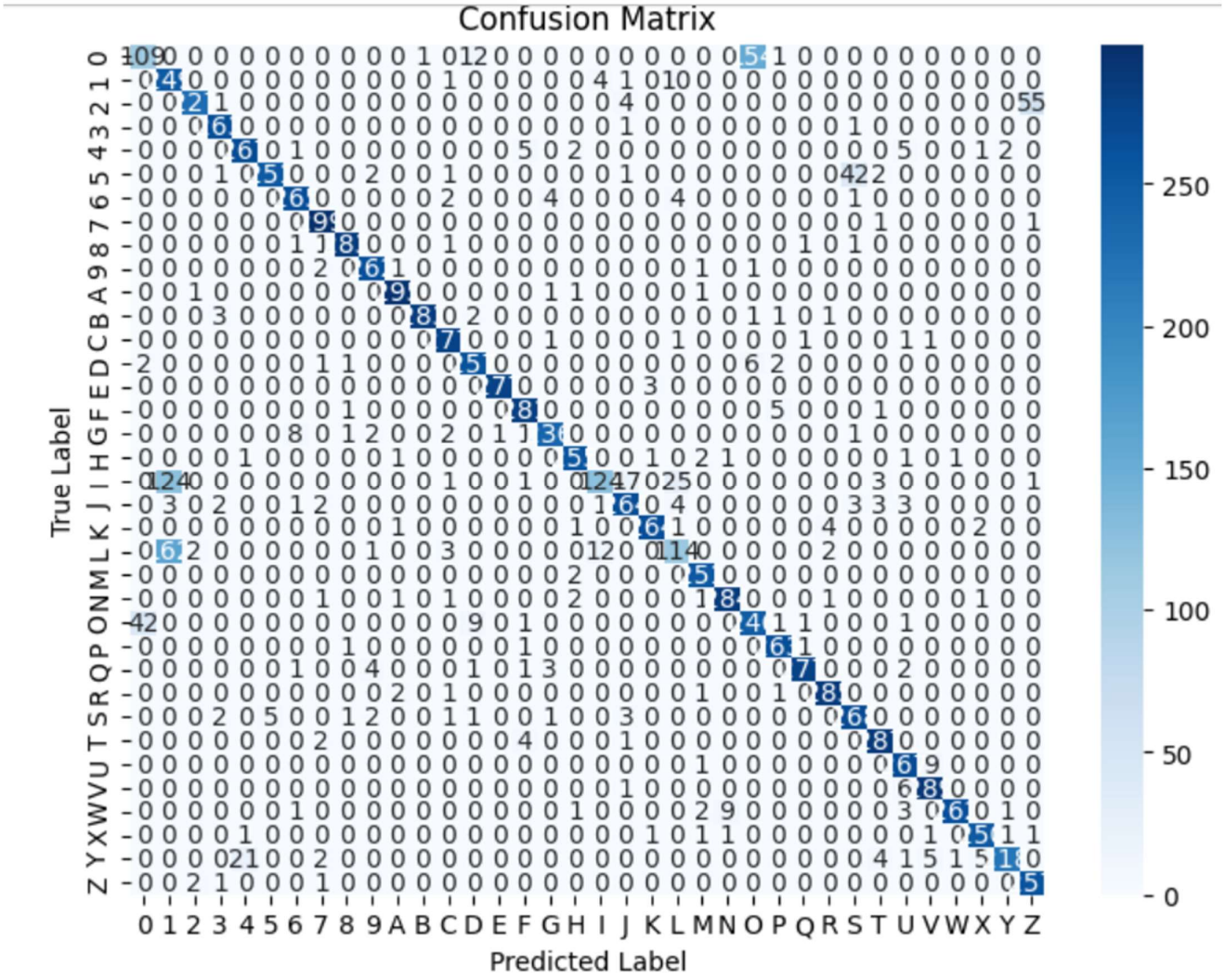
The graph for training, validation, and test loss over epochs has some important facts to tell about the learning process. From this graph, the following are observed:

Training Loss: It is observed that the training loss has remained less while considering all the epochs. This ranges from a high of about 0.9 in the first few epochs to less than 0.3 within the third epoch. It basically shows that the model is learning well from the data in terms of minimizing error.

Validation and Test Loss: The losses are mostly fairly low for validation and test purposes, standing below 0.3 since the very beginning of the training, hence not overfitting and keeping a good generalization capability.

Convergence: The training loss drops significantly in the very first epoch and keeps on falling, but with a graceful slowness. Thus, this is a good fit to data. Validation and test losses are close; this further confirms the stability of this model and the good generalization capability.

Confusion Matrix:



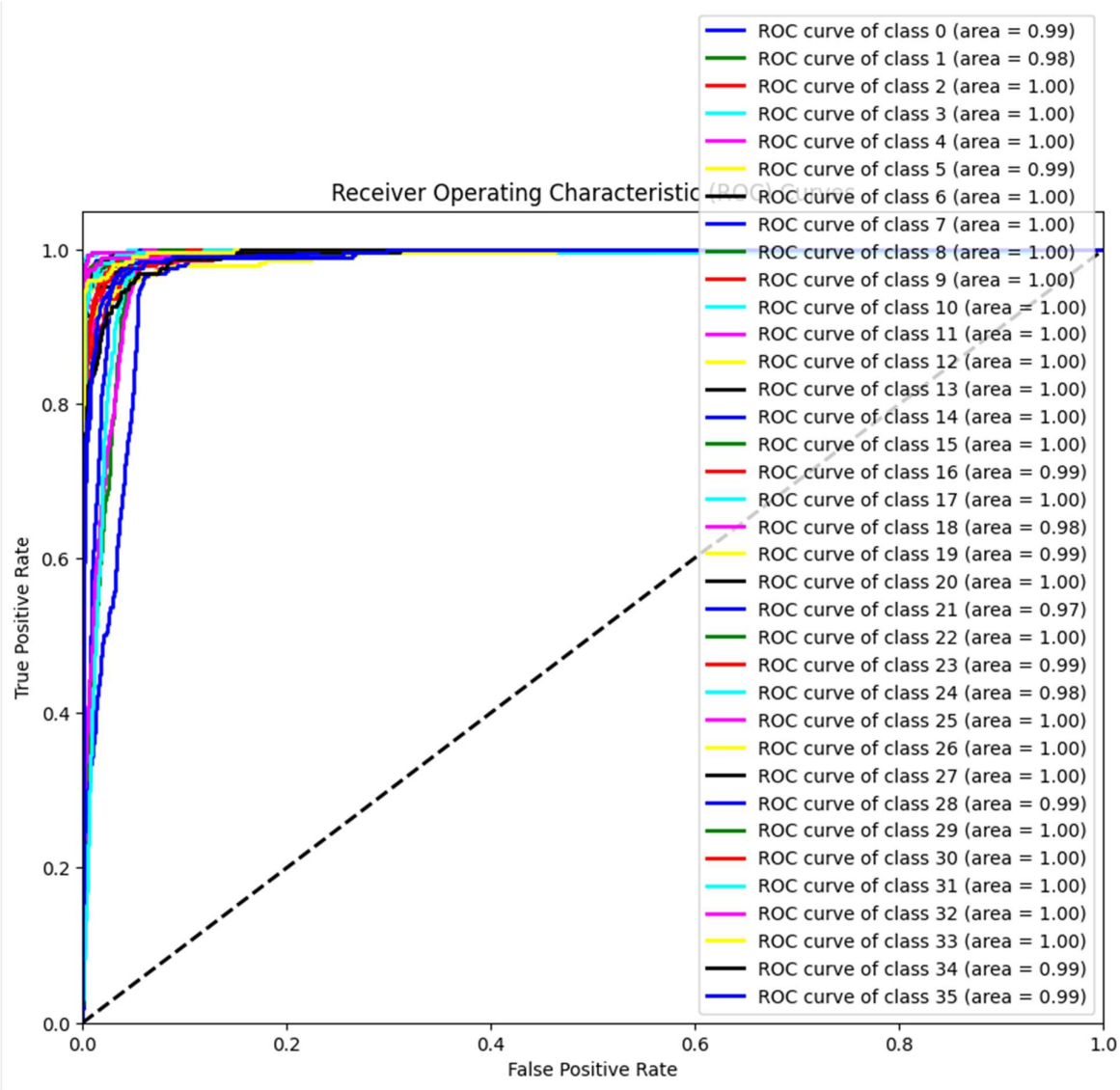
The graph for training, validation, and test loss over epochs has some important facts to tell about the learning process. From this graph, the following are observed:

Training Loss: It is observed that the training loss has remained less while considering all the epochs. This ranges from a high of about 0.9 in the first few epochs to less than 0.3 within the third epoch. It basically shows that the model is learning well from the data in terms of minimizing error.

Validation and Test Loss: The losses are mostly fairly low for validation and test purposes, standing below 0.3 since the very beginning of the training, hence not overfitting and keeping a good generalization capability.

Convergence: The training loss drops significantly in the very first epoch and keeps on falling, but with a graceful slowness. Thus, this is a good fit to data. Validation and test losses are close; this further confirms the stability of this model and the good generalization capability.

ROC Curve:



The ROC curve is meant for evaluating a model's classification performance over a multitude of classes. The following are the key observations that can be made from this ROC curve graph:

Overall performance: Following are the ROC curves for each class, giving a measure of the performance across all classes (0-9, A to Z). There is an attached AUC value corresponding to each class, which tells us about the discrimination power of the model in differentiating that positive class from all other classes.

A high value of AUC shows that most of the classes lie close to, or at, 1.0, ensuring that those classes give very good performance in classification. That means, in fact, the model is well able to predict the positive instances correctly with minimal false positives.

Slight Variations: A few classes have slightly lower AUCs, such as 0.97 to 0.99. Here, it is a bit more difficult for the model to distinguish between positive and negative instances of these

classes. In contrast, all classes whose AUC value exceeds 0.95 will, in general, be considered very good.

Perfect classification for a few classes: Classes 3, 4, 5, 7 among many others, have AUC values 1.0, which in fact means perfect classification for these classes, since there were no false positives nor false negatives in those instances.

Balanced Class Performance: Since most of the ROC curves are upwards and as close to the top-left corner of the graph, that indicates that the model is balanced in terms of performance across classes. The high AUC indicates that the model is well-balanced and does not seem to have high biases toward any classes.

Comparison with the performance metrics:

Accuracy:

Training Accuracy Comparison

Previous Graph: The graph previously plotted shows training accuracy starting off lower and increasing at a good pace, crossing over 90% after the third epoch, which conveyed that learning from the training dataset was effectively obtained and convergence occurred early.

New Graph: The training accuracy of this graph starts around 60%, while in about ten epochs, it goes up to approximately 82%. As can be seen, its increase is slower than in the previous graph, which means this model takes more time to converge.

Comparison of Validation Accuracy

Previous Graph: High and greater than 90% validation accuracy since the very beginning of the first epoch is a sure indication that this model was already well-generalized for unseen data right at the very beginning of its training.

New Graph: The validation accuracy starts off at around 80%, improves gradually, and reaches about 88% by the tenth epoch. Such a gradual increase indicates this model is still on its path toward generalization and that the validation performance is somewhat lagging behind compared to the earlier graph.

Comparison of Test Accuracy

Old Graph: The test accuracy was constantly above or at 90% after the first epoch, which is indicative of strong generalization with minor overfitting.

New Graph: Test accuracy for the current graph is always above 90%, just like in the previous graph. This therefore shows the robustness in the model performance on unseen data, irrespective of the difference in training dynamics.

Loss:

Comparison of Training Losses Old Loss Plot: The training loss started off higher-up to about 1.0-and within the first three epochs, it rapidly fell below 0.3. That rapid drop below 0.3-which marks the convergence point in that graph-means this model was really fast in grasping the knowledge and reducing the errors during that short run. It reached stability quickly, which indicates that early convergence happened for the model.

Current Loss Graph: In the current graph, the training loss starts off higher, at approximately 1.2, and decreases more gradually over ten epochs until it reaches about 0.35. This indicates that the model is learning much more slowly compared to the previous graph. Such slow reduction gives an indication of the more steadied and gradual learning rate the model is experiencing.

Comparison of Validation Loss

Previous Loss Graph: The validation loss in the previous graph, after the first epoch, remained constantly below 0.3, which is a good generalization at the beginning of training. The track of the validation loss also followed that of the training loss closely, showing stability of the model.

Graph of Current Loss: The graph for current loss starts from around 0.6 and smoothly keeps on decreasing toward approximately 0.3; this behavior shows that the model in this case keeps on getting better for longer compared with the previous case. Also, the closeness of the validation loss to the training loss here implies it is not overfitting much.

Comparison on Test Loss

Previous Loss Graph: The test loss, from the previous graph, remains constantly low and smooth during training, while the validation loss further indicated robustness in the performance of the model dealing with unseen data.

Graph of Current Loss: The red dashed line in the graph represents the current test loss, similar to the previous graph, staying low and stable across the different epochs. This means that not only in different stages of training, but the model generalizes well consistently.

Confusion Matrix:

High accuracy among classes. Previous Confusion Matrix: The values on the diagonal of the confusion matrix were high, thus indicating a very good classification performance for most of the alphanumeric characters. Most character classes were constantly well predicted by the model.

Current Confusion Matrix: This new confusion matrix still contains high values on the diagonal; thus, it captures well how the model keeps classifying most instances correctly. Some of these values are a little higher across the diagonals than those in the previous confusion

matrix. Thus, one might say that the current model is slightly more predictable for some classes.

Misclassification Patterns

Common Confusion Points:

'O' vs. '0': There are massive misclassifications in both confusion matrices between 'O' and '0'; this is as expected since they are quite similar in appearance. The trend continues, really showing that this remains a challenge for the model.

'I' vs. '1': Confusion in both matrices is also between 'I' and '1', showcasing difficulties in distinguishing between characters of very similar shapes.

It seems that the number of misclassifications in these common confusion points may be slightly reduced in the present confusion matrix, which perhaps underlines some improvement in distinguishing between similarly looking characters.

ROC Curve:

Previous ROC Curve: From the AUC values given on most classes of the previous ROC curve graph, one may infer a range of 0.97 to 1.0. That is to say, for most classes, classification performance was near perfect, in the sense that the model proved really good at differentiating between positive and negative instances.

ROC Curve Current: The graph for the ROC curve in the current one shows more variability; this is in its AUC values, which range from 0.75 to 1.0. Among them, the classes are Class 0 and Class 1; these classes, as shown, have relatively lower AUC values of 0.79 and 0.76, respectively. This means that, for this particular class, the model has greater difficulty distinguishing between them than it would against any other class. However, most classes still have very high AUCs of over 0.95 and are really doing well in classification.

Contribution:

Team Member	Assignment Part	Contribution %
Saroja Vuluvabeeti (50593902)	Part 1	50%
Saroja Vuluvabeeti (50593902)	Part 2	50%
Sri Sakthi Thirumagal(50592325)	Part 1	50%
Sri Sakthi Thirumagal(50592325)	Part 2	50%
Saroja Vuluvabeeti (50593902)	Part 3	50%
Sri Sakthi Thirumagal(50592325)	Part 3	50%
Saroja Vuluvabeeti (50593902)	Part 4	50%
Sri Sakthi Thirumagal(50592325)	Part 4	50%
Saroja Vuluvabeeti (50593902)	Bonus	50%
Sri Sakthi Thirumagal(50592325)	Bonus	50%

References:

1. https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
2. https://pytorch.org/docs/stable/generated/torch.nn.Module.html
3. https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html
5. https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html
7. https://pytorch.org/ignite/generated/ignite.handlers.early_stopping.EarlyStopping.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
9. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
10.