CSE 574 - Intro To Machine Learning Assignment 2

	Contributions	Parts
minazmeh	50%	Part 1, Part 2, Part 3,Part 4 , Report
aramired	50%	Part 1, Part 2, Part 3,Part 4 , Report

PART I

1. Overview of the dataset:

a. Type of data: f1 to f7 are features columns representing numeric values. Target indicates the outcome of each row 0 and 1

b. Number of samples: 766

c. Number of features: 8

d. Key statistics:

	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

2. Preprocessing:

a. Handling invalid entries:

Non numeric entries in a dataframe ar detected by iterating through each column and identifying invalid values. We convert the entries from object type to numeric. Any values that can not be converted are coerced into Nan and these NaN values are identified as invalid entries. These then are added to the invalid_entries dictionary.

```
# figure out the non numeric entries in our df
invalid_entries = {}

for col in df.columns:
    if df[col].dtype == 'object':
        invalid_values = df[col][pd.to_numeric(df[col], errors='coerce').isna()].unique()
        if len(invalid_values) > 0:
            invalid_entries[col] = invalid_values

print("Non-numeric entries detected:")
print(invalid_entries)
```

```
Non-numeric entries detected:
{'f1': array(['c'], dtype=object), 'f2': array(['f'], dtype=object), 'f4': array(['a'], dtype=object), 'f5': array(['b'], dtype=object), 'f6': a
```

```
# replace them with NA
df = df.replace(['a', 'b', 'c', 'd', 'e', 'f'], pd.NA)
```

b. Checking if target column needs pre-processing:

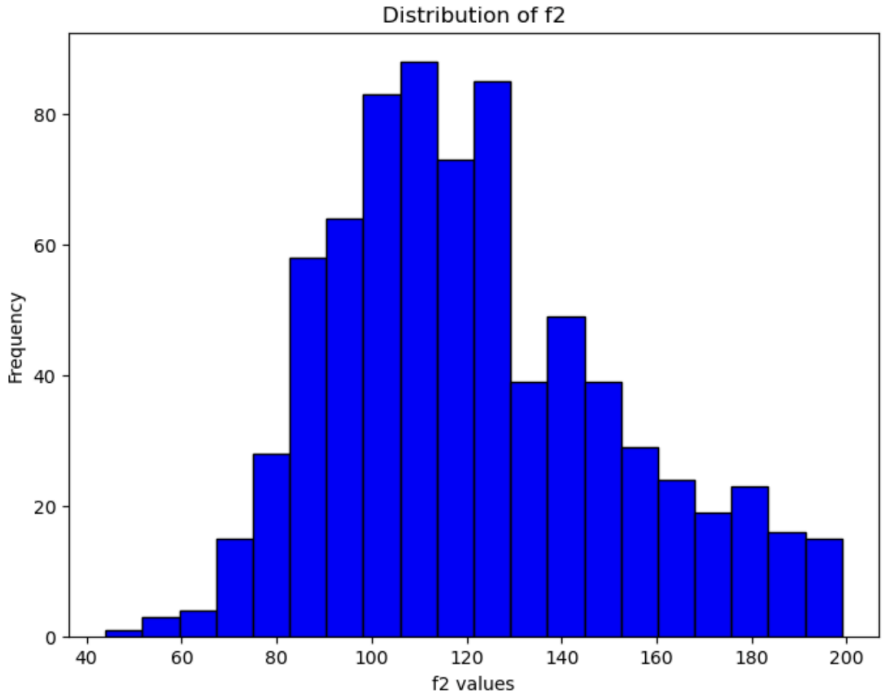
```
    # do we needd to preprocess our target for categorical cleaning?

    print(f"Unique values in the target", df['target'].unique())
```

```
Unique values in the target [1 0]
```

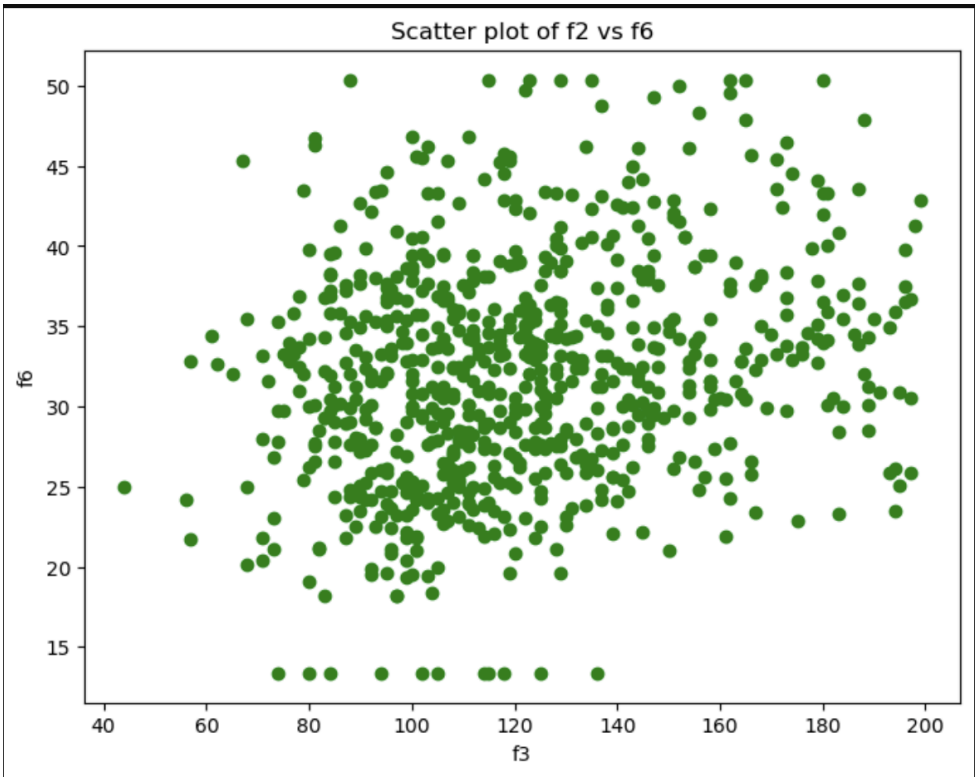
3. Visualizations:

a.



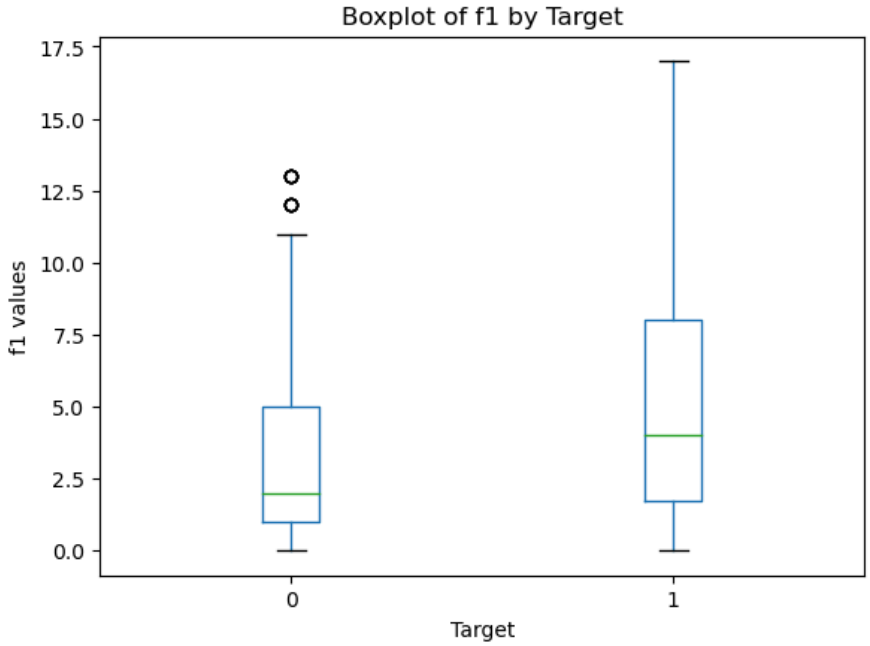
histogram shows the distribution of the values in column f2. The data is approximately normally distributed, with the majority of values concentrated between 75 and 150. The peak frequency occurs around the value of 100, indicating that this range is the most common for f2.

b.



The scatter plot shows the relationship between the features f3 (x-axis) and f6 (y-axis). The points are distributed somewhat densely in the middle range, but the data appears spread out without a clear linear trend, suggesting a weak or non-linear correlation between these two features.

c.



The box plot illustrates the distribution of f1 values for two target classes (0 and 1) For Target = 0, the median is lower, and there are some outliers, while Target = 1 shows a wider spread with a higher median and fewer outliers.

4. **summary of your NN model:**

Input Layer : receives the input features. X_train.shape[1]
Hidden Layers:

1. Layer 1 : we have a fully connected layer fc1 with 64 neurons, activated by Relu function and it takes our input features as input.
2. LAyer 2 : WE have another fully connected fc2 with 64 neurons this is also activated by Relu function.
3. Dropout Layer : we have added a drop out layer with 0.3 rate, as a regularization to prevent overfitting.

OutPut Layer :

We have a fully connected layer fc3 with a single output neuron, activated using sigmoid function for the binary classification.

Why have we chosen this architecture?

Two hidden layers:

With 64 neurons are used to allow the network to learn more complex patterns in the data while maintaining computational efficiency.

ReLU activation :

functions are chosen to introduce non-linearity which is essential for capturing complex relationships.

Sigmoid activation :

In the output layer we have used sigmoid activation because the task is binary classification.

Dropout (0.3)

Is incorporated to mitigate overfitting this ensures the model generalizes well on unseen data.

5. Our Performance Metric and detail analysis:

Test Accuracy : 0.7807 which is 78% which shows that our model performs well in predicting the binary outcomes.

Precision : 0.7692 which indicates that of all the positive predictions made, around 76.92% were actually correct.

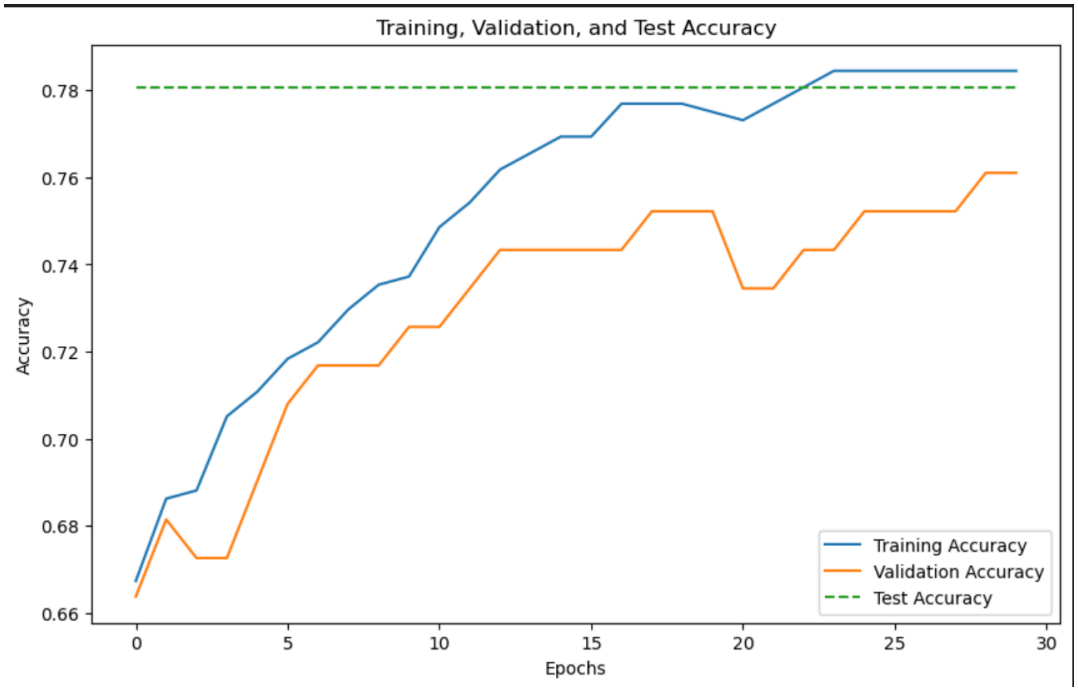
Recall: **0.5128** suggests that out of all the actual positives, the model correctly identified around 51.28%. This low recall indicates that the model missed a substantial number of true positives.

F1-Score: **0.6154** is the harmonic mean of precision and recall, which balances the two metrics. The score indicates a moderate performance, with room for improvement, especially in recall.

Total Training Time: The model training was completed in 0.26 seconds, demonstrating an efficient training process, possibly due to the the small dataset.

Analysis on Performance:

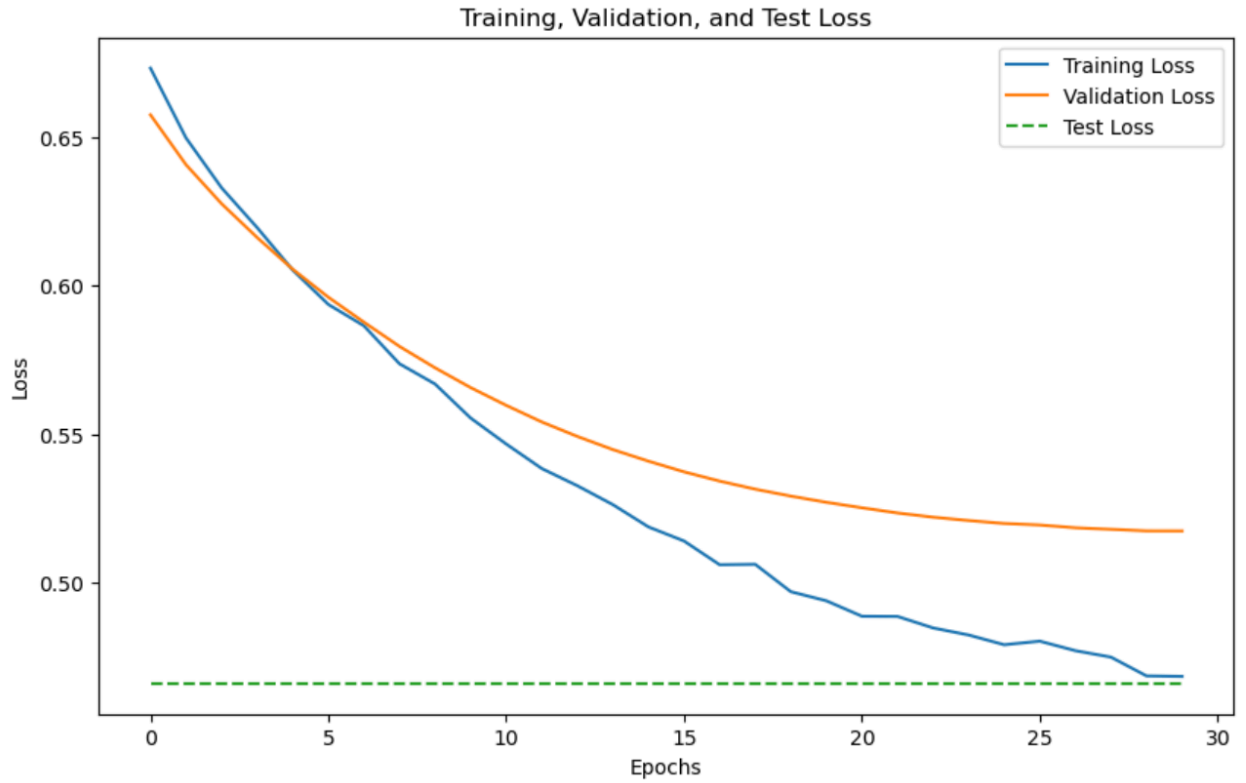
Training and Validation Accuracy:



Both training and validation accuracy steadily improve as the epoch increases overtime. Training accuracy shows consistent improvement, reaching around **0.78** after 30 epochs.
Validation accuracy reaches around **0.75**

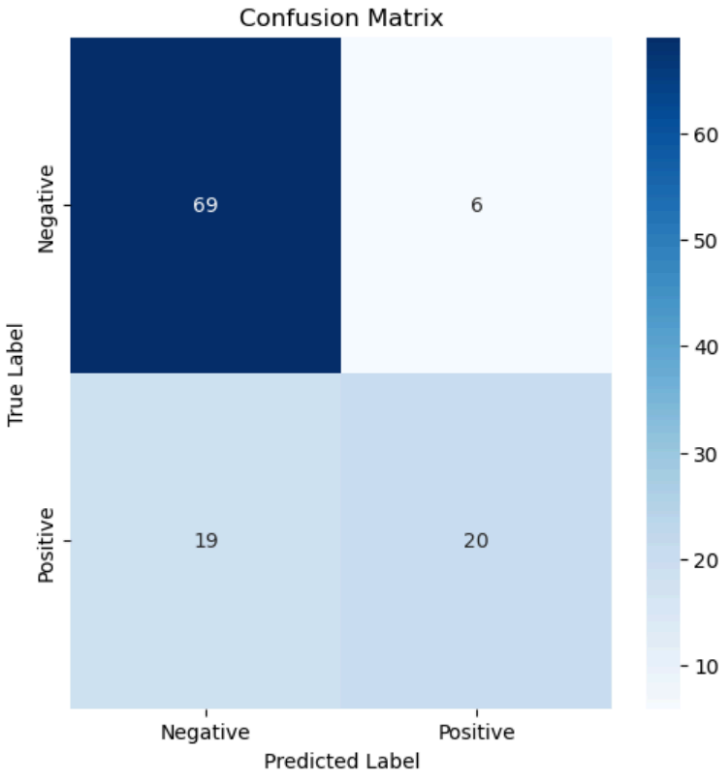
This means that the model generalizes reasonably well on unseen data.

Training and Validation Loss:



The training loss consistently decreases with more epochs this indicating that the model is learning.
The validation loss also decreases but more slowly compared to the training loss.

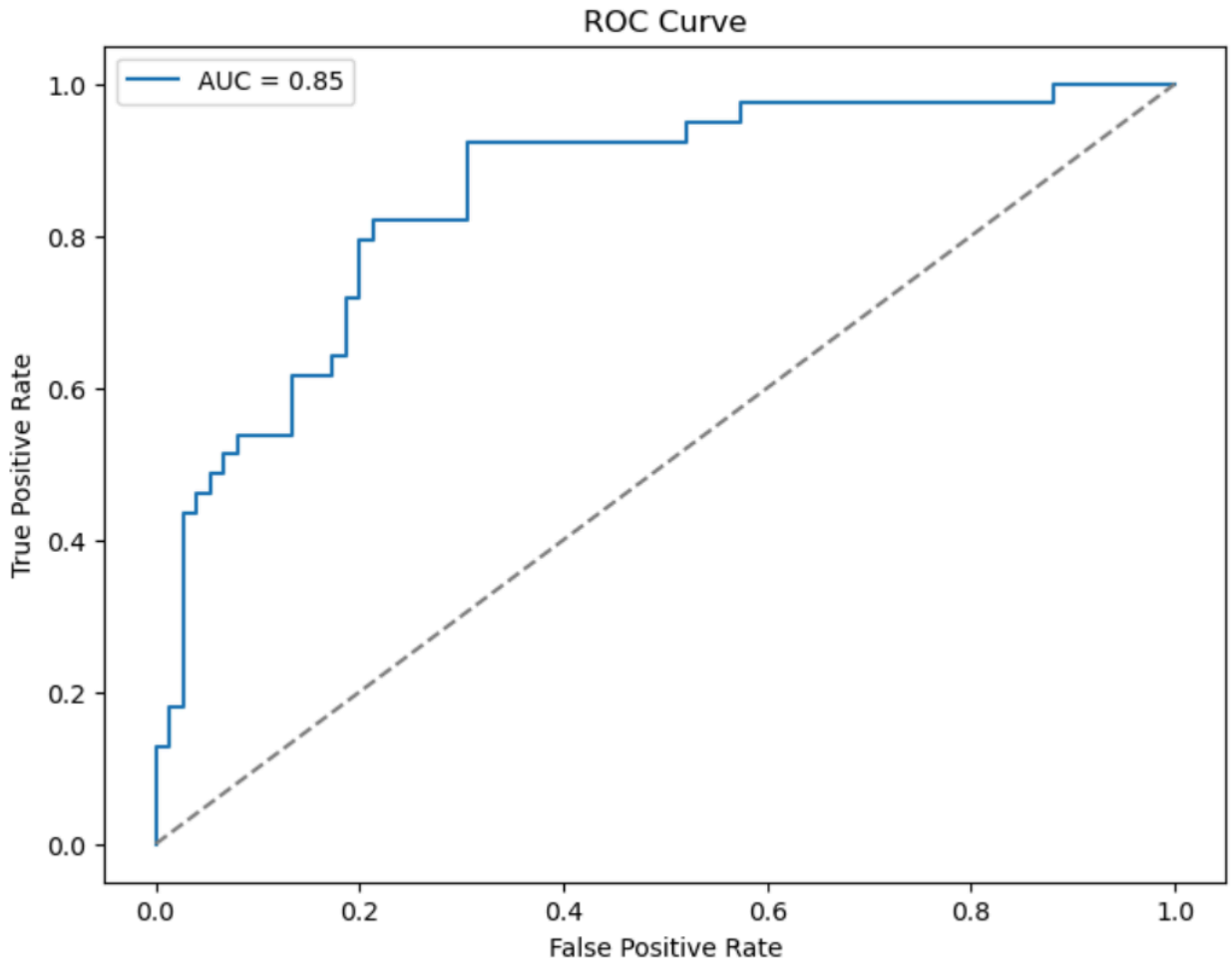
Confusion Matrix:



The confusion matrix shows that the model performs well in identifying negative labels (69 true negatives, only 6 false positives) but struggles somewhat with positive labels where 19 instances were misclassified as negative while only 20 were correctly identified.

This highlights an issue with recall, as the model is missing many true positive cases.

ROC Curve:



The ROC curve shows an AUC (Area Under the Curve) score of 0.85, which is a strong indicator of the model's ability to distinguish between the positive and negative classes

PART II

Q 1.

The original model made in part 1 had the following hyper parameters:

1. Neuron size: 64
2. Layers : 3
3. Activation : Relu
4. Dropout rate : 0.3
5. Learning rate : 0.0001
6. Optimizer : RMSProp
7. Batch Size : 32

We changed the hyperparameters in following way:

Setup 1 : Changed Dropout Rates:

	Neuron size	Layers	Activation	Drop out	Learning rate	Optimizer	BatchSize	Accuracy
Changed Dropout 0.4	64	3	Relu	0.4	0.0001	RMSProp	32	75.44
Changed Dropout 0.1	64	3	elu	0.1	0.0001	RMSProp	32	75.44
Changed Dropout 1	64	3	Relu	1	0.0001	RMSProp	32	77.19

Setup 2 : Changed Activation Function:

	Neuron size	Layers	Activation	Drop out	Learning rate	Optimizer	BatchSize	Accuracy
ELU	64	3	elu	0.3	0.0001	RMSProp	32	78.07
TANH	64	3	tanh	0.3	0.0001	RMSProp	32	78.07
LEAKYRELU	64	3	leakyrelu	0.3	0.0001	RMSProp	32	77.19

Setup 1 : Changed Optimizer:

	Neuron size	Layers	Activation	Drop out	Learning rate	Optimizer	BatchSize	Accuracy
Adam	64	3	Relu	0.4	0.0001	Adam	32	77.19
AdamW	64	3	elu	0.1	0.0001	AdamW	32	78.07
NAdam	64	3	Relu	1	0.0001	NAdam	32	77.19

We have chosen our base model the one with Changed Activation function TANH which shows accuracy of 78.95

Analysis:

1. Dropout Rate:

- In Setup 1, increasing the dropout rate from 0.4 to 1.0 improved the accuracy from 75.44% to 77.19%.
- However, the model with a 0.1 dropout rate gave the same accuracy as the 0.4 dropout rate (both 75.44%).
- This suggests that a very high dropout rate (1.0) forces the network to generalize better, but lower rates might not significantly impact accuracy.

2. Activation Functions:

- In Setup 2, the model with the TANH activation function yielded the highest accuracy of 78.07%, equating ELU (78.07%) and outperforming Leaky RELU (77.19%).

- This suggests that TANH is better suited for this problem, likely due to its symmetric output range,

3. Optimizer Changes:

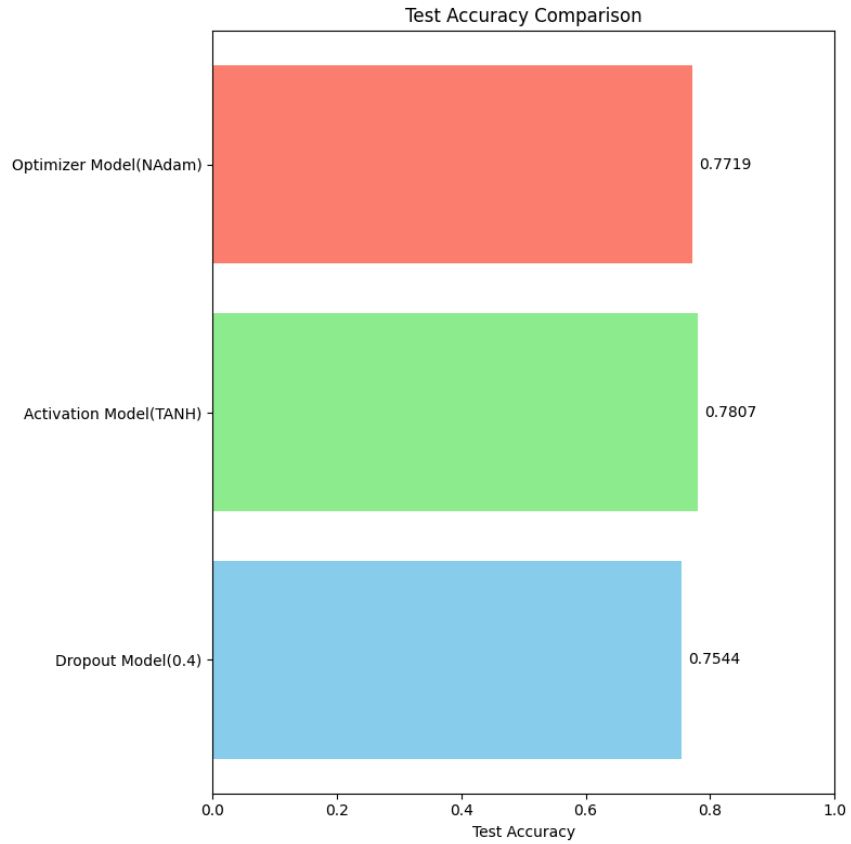
- Switching from RMSProp to AdamW resulted in the highest accuracy of 78.07% compared to the standard Adam or NAdam optimizer
- AdamW incorporates weight decay, which can improve regularization, explaining its slightly better performance.

Final Analysis:

The TANH activation function combined with RMSProp optimizer led to the best performance (78.07% accuracy), this shows that activation functions significantly influence model performance probably due to gradient behaviours.

Q. 2

TEST ACCURACY : Comparing the best setups of three variations for (hyper parameters)



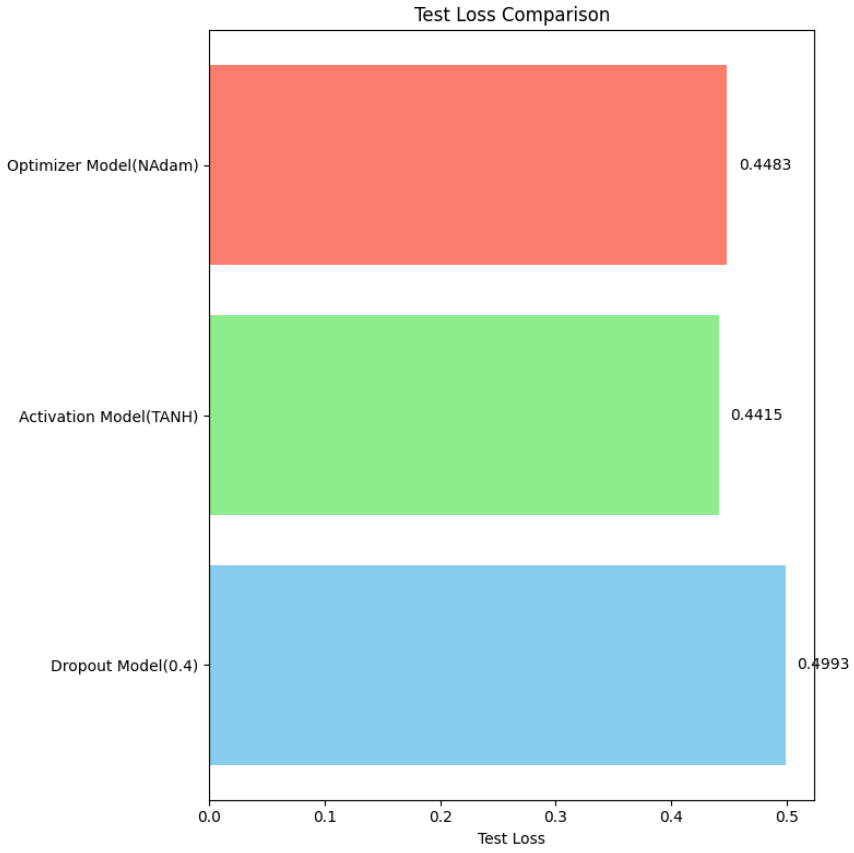
1. Best Test Accuracy Comparison

- **Dropout Model**: 77.19%
- **Activation Model**: 78.07%
- **Optimizer Model**: 75.44%

Observations:

- The changed activation model with TANH gave the best performance with highest accuracy of 78.95% and least loss 44.15%.
- The changed drop out rate model gave best performance for drop out rate = 1
- The optimizer gave best performance for NAdam.

TEST LOSSES : Comparing the three variations



2. Test Loss Comparison

- **Dropout Model**: 0.4493
- **Activation Model**: 0.4415
- **Optimizer Model**: 0.4483

Observations:

- The **Activation Model** showed the lowest test loss of **0.4415**, indicating better generalization performance.
- The **Optimizer Model** showed a moderately lower loss of **0.4883** compared to the Dropout Model but performed worse than the Activation Model.
- The **Dropout Model**, despite having similar accuracy to the Optimizer Model, had the highest loss (**0.4493**), suggesting that while it performs well on accuracy, it could still benefit from further tuning to reduce overfitting or improve generalization.

Analysis

- **Best Accuracy**: The **Activation model** provided the best accuracy (78.95%). This suggests that both changing the activation function can significantly impact model performance.
- **Best Loss**: The **Activation Model** showed the best performance in terms of loss (0.4415), implying that this setup had a more stable optimization process with better convergence.

Q.3 methods you used that help to improve the accuracy:

1. **Earlystopping:**

   We have used earlystopping to prevent overfitting. The network has 2 hiddn layers with ReLU activation functions and finally a sigmoid. A dropout layer is added into the training process to aid with regularization. Earlystopping monitors the validation loss, and if it doesnot increase
   for a specified number of epochs, the training is stopped early to prevent
   overfitting.

   Training loop calculates both training and validation loss and the accuracy for each epoch. Validation loss monitoring is done after each epoch. After the earlystopping is triggered, model reverts the best weights with least validation loss. Accuracy, precision, recall and F1 score are calculated on this set.
   Test accuracy achieved:
2. **K-fold:**

   The dataset is divided into five folds. A distinct subset of the data from each fold is used to train the model, and the remaining data is used for validation. A new model is

initialized at the beginning of each fold to guarantee separate training, and each fold has its own training and validation loop.

For every fold, the optimizer, loss function, and model are reinitialized. validation loss and accuracy are computed and recorded after the training loss and accuracy for every epoch. Following the processing of each fold, the average performance metrics—test accuracy, validation accuracy, training accuracy, and training loss—are computed for each fold.

The test accuracy is calculated for each fold once the model has been assessed on a different test set. By reducing the possibility of overfitting or bias brought on by arbitrary data splits, this method offers a reliable estimate of the model's performance.

3. **Learning rate scheduler**

   We have used a ReduceLROnPlateau learning rate scheduler, which dynamically adjusts the learning rate based on the validation loss. If the validation loss does not improve for a set number of epochs (patience = 3), the scheduler reduces the learning rate by a factor of 0.1. This ensures that the optimizer can explore smaller steps and potentially reach a better convergence. During training, the model's performance is evaluated on a validation set, and the best model is saved.

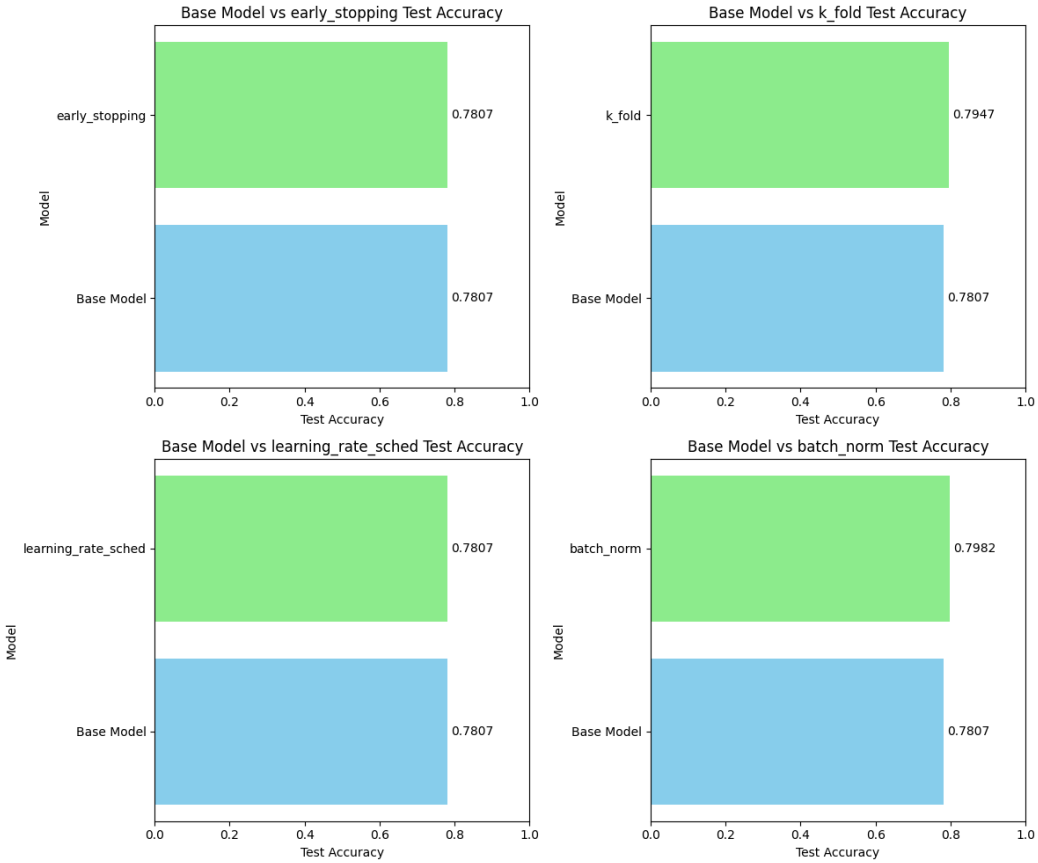
   After training, the best model is loaded for testing, where it is evaluated on a separate test set. Metrics such as test accuracy, precision, recall, and F1 score are calculated to assess the model's performance.

4. **Batch normalization**

   The model architecture includes two hidden layers, each followed by a batch normalization layer and a ReLU activation function. Batch normalization is used to normalize the outputs of the layers, improving training stability and speeding up convergence. Dropout is added to prevent overfitting, and a sigmoid activation function is applied to the final layer for binary classification.

   The model is trained over 30 epochs, with training and validation losses and accuracies computed for each epoch. The best-performing model, based on validation accuracy, is saved during the training process. Once training is complete, the best model is loaded and tested on a separate test dataset, where performance metrics such as accuracy, precision, recall, and F1 score are computed to evaluate its effectiveness. The inclusion of batch normalization helps in faster and more stable training, leading to potentially better model performance.

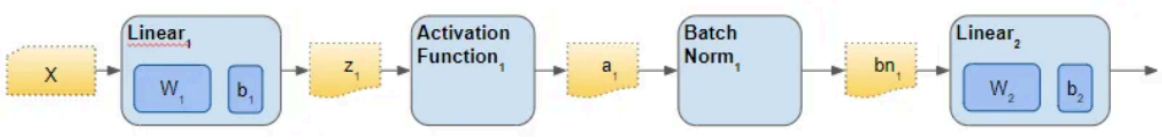
Q.4. best model:



Comparing the test accuracy of the base model with activation model changed to tanh with the earlystopping, k-fold, learning rate scheduler, batch normalization, we can see that batch normalization returns the best accuracy.

batch normalization:

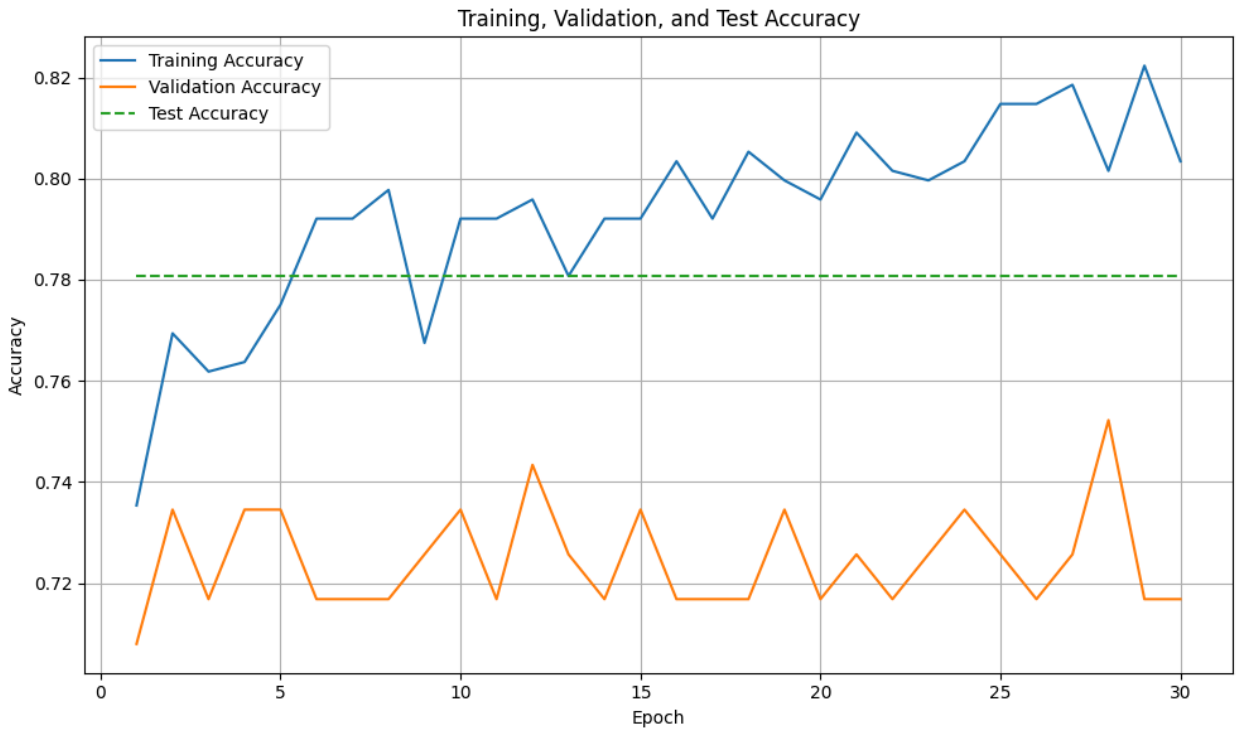
Batch Norm is just another network layer that gets inserted between a hidden layer and the next hidden layer. Its job is to take the outputs from the first hidden layer and normalize them before passing them on as the input of the next hidden layer. By normalizing the inputs, learning is stabilized which allows for faster convergence and better generalization.



The Batch Norm layer normalizes activations from Layer 1 before they reach layer 2 (Image by Author)

- fc1 maps the input features to the first hidden layer, fc2 connects the first hidden layer to the second, and fc3 outputs the result for binary classification (single output).

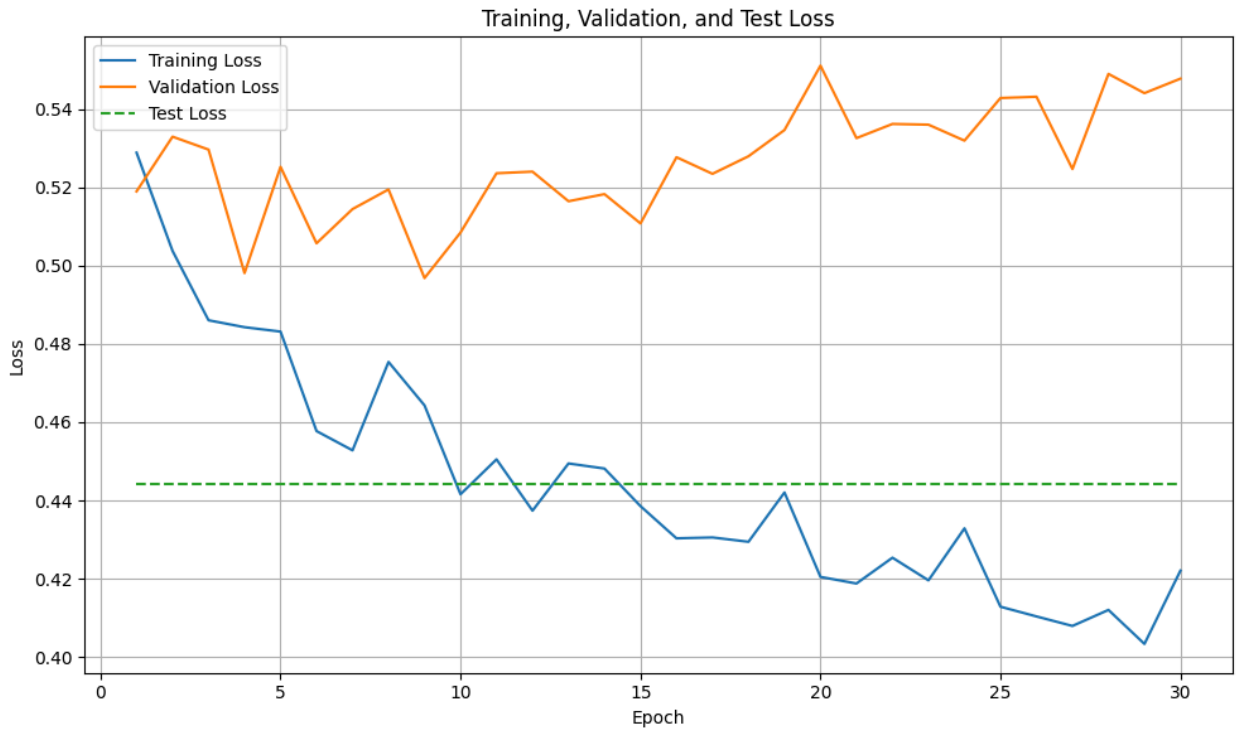
- bn1 normalizes the output of the first hidden layer, and bn2 normalizes the output of the second hidden layer.
- **self.dropout**: This regularization technique randomly drops units during training to prevent overfitting. The dropout probability is set to 0.4
- **Forward pass:** 3 layers.
    - First Layer: The input x is passed through fc1, followed by bn1 for normalization, and then the ReLU activation
    - Second Layer: The output of the first layer goes through fc2, followed by bn2 for normalization, and then another ReLU activation is applied.
    - Output Layer: The output of the second layer is passed through fc3 and then through a sigmoid activation function, which squashes the output to the range [0, 1] for binary classification.
- **Optimization**: The RMSprop optimizer is used with a learning rate of 0.001
- **Loss Function**: The binary cross-entropy loss (BCELoss) is used since this is a binary classification problem.
- **Training Mode**: The model is set to training mode (net_bn.train()), which ensures that dropout and batch normalization layers behave accordingly
- **Batch Processing**: For each batch, the inputs are passed through the model, the loss is computed, and the gradients are backpropagated.
- **Validation:** (net_bn.eval()), the model is set to evaluation mode, which ensures that dropout is turned off and batch normalization uses running statistics (computed during training) rather than batch-specific statistics.
- torch.no_grad() is used to save memory and improve speed as gradients are not needed
- After each epoch, if the validation accuracy is higher than the previous best, the model's parameters are saved.



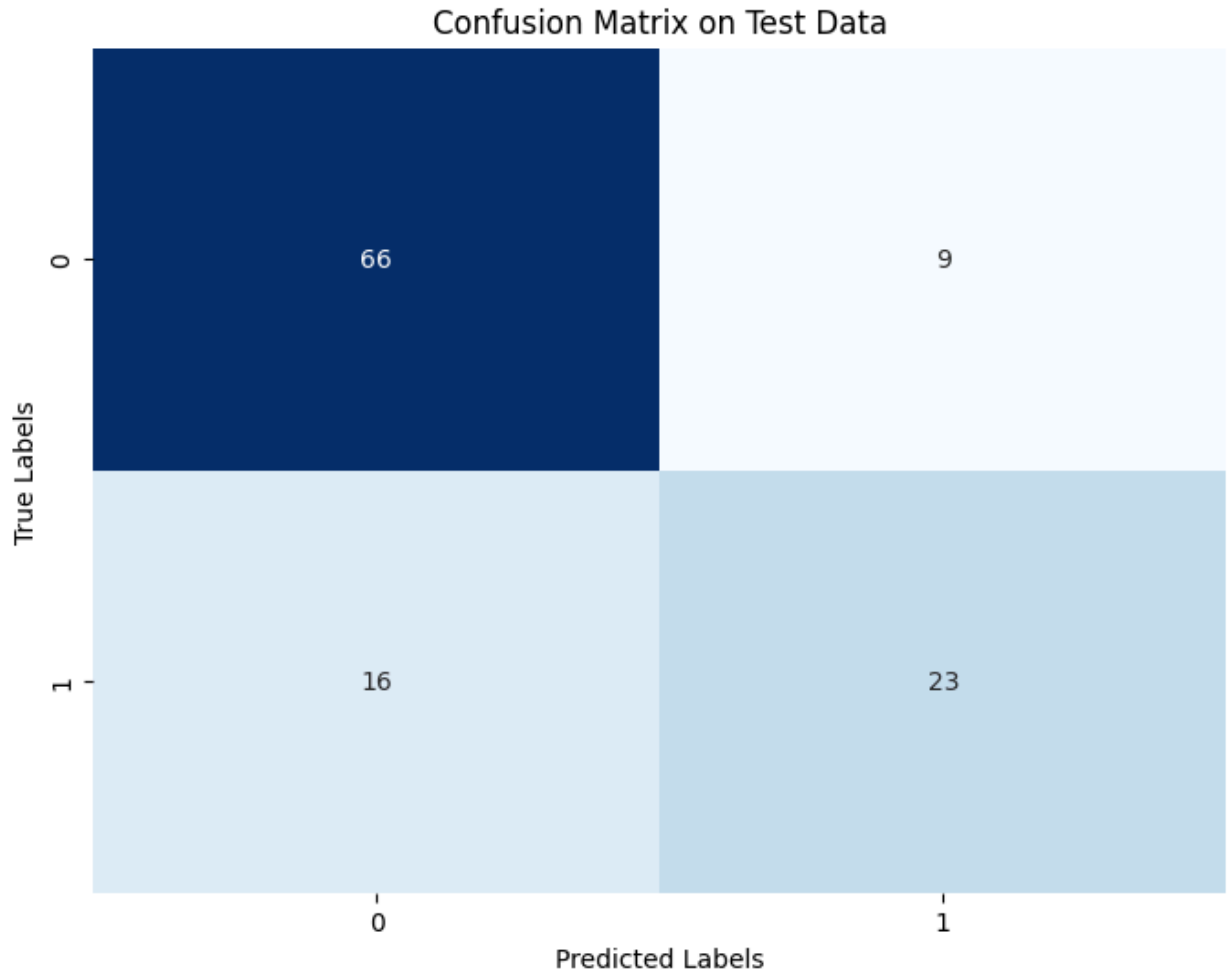
Analysis on Training,Validation and Test accuracies:

The training accuracy improves steadily, reaching close to 84%, while the validation accuracy fluctuates more significantly.

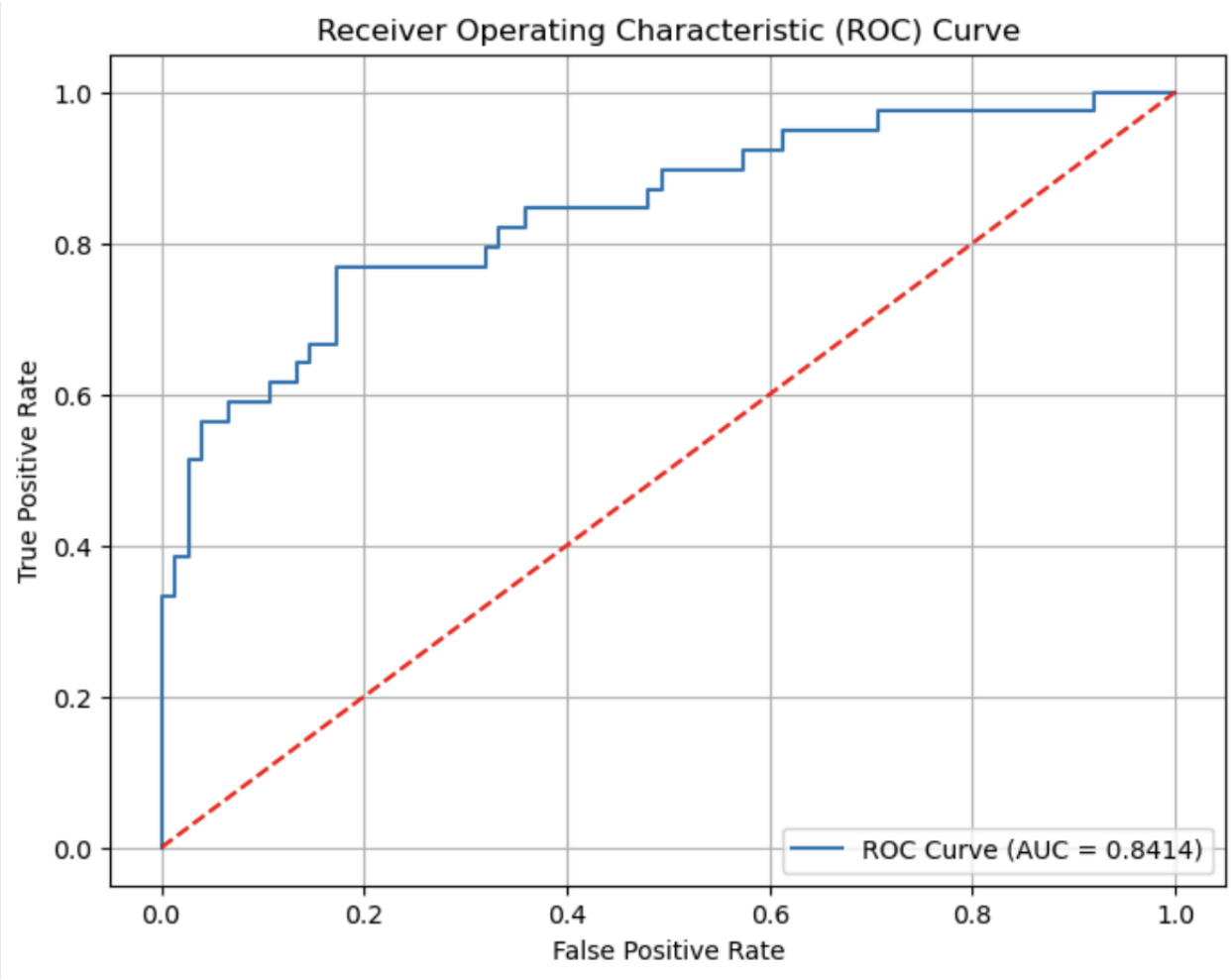
Test accuracy remains constant at around 81%, indicating stable performance on unseen data with batch normalization.



Analysis on Training, Validation and Test losses:
Training loss decreases smoothly, showing continuous improvement as the epochs progress, suggesting good learning from the model.
Validation loss fluctuates but stays higher than training loss, highlighting potential overfitting, while test loss remains around 0.45, indicating decent generalization.



Analysis on Confusion matrix:
The confusion matrix shows the model performs well on classifying the majority class with only 5 false negatives.
However, it misclassifies 16 instances from the minority class, indicating room for improvement in handling imbalanced data.



Analysis on ROC curve:

he ROC curve shows a reasonably strong classifier with an AUC of 0.8414, indicating that the model performs well in distinguishing between the classes.

The curve suggests the model can achieve a good balance between the true positive rate and the false positive rate.

PART III

Note: The confusion matrix & ROC Curve is annotated with labels from Class 0 to Class 36, where Classes 0 to 9 correspond to the digits 0 to 9, and Classes 10 to 36 represent the alphabetic characters A through Z.

1. ***Provide a brief overview of your dataset (e.g. type of data, number of samples, and features. Include key statistics***

   Answer:

   Dataset Overview:

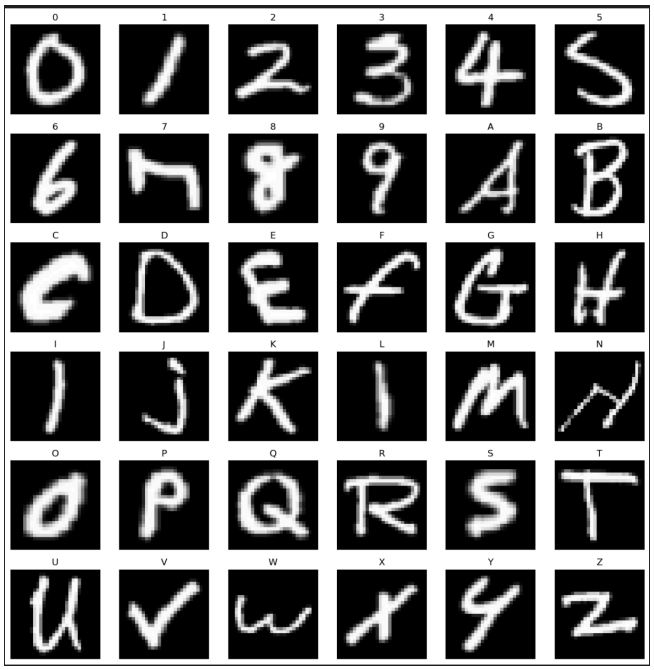
   - Type of Data: Image dataset
   - Number of Samples: 100,800 images
   - Data Source: Stored in a directory called `cnn_dataset`
   - Number of Classes: 36
   - The dataset includes labels for digits (`0-9`) and uppercase English letters (`A-Z`), excluding lowercase letters

   Image Properties:

   - Resolution: 28x28 pixels
   - Grayscale: Each image has been converted to grayscale with a single color channel
   - Normalization: Images are normalized with a mean of 0.5 and a standard deviation of 0.5

   Transforms Applied:

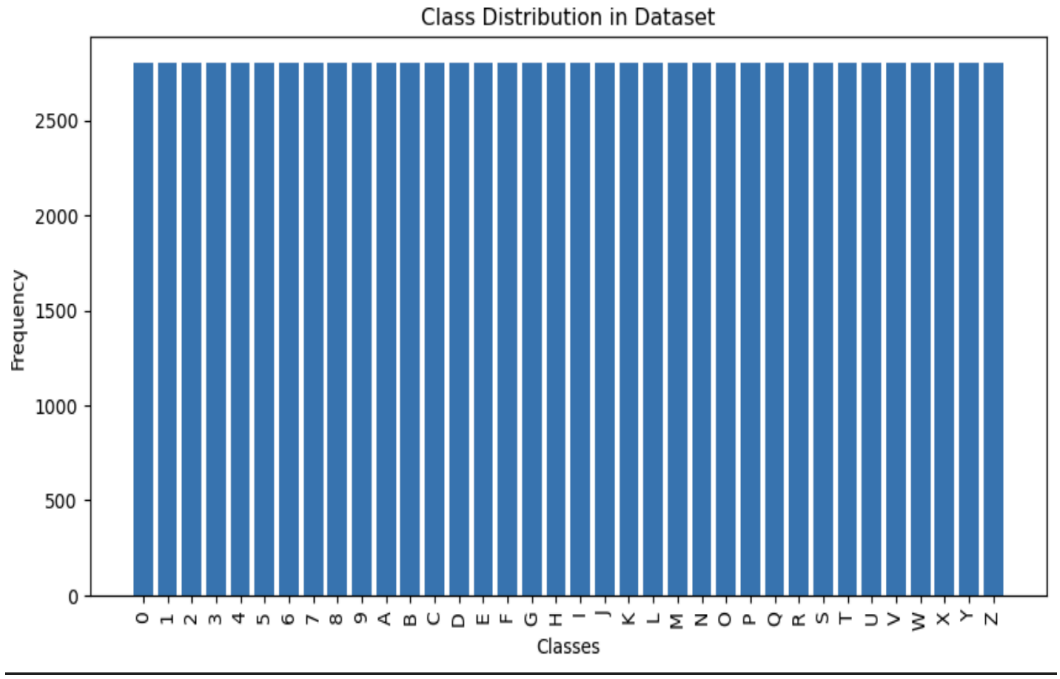
   - Resizing to 28x28 pixels (Just to ensure onceagain)
   - Converting to grayscale
   - Converting to tensor format
   - Normalizing pixel values



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

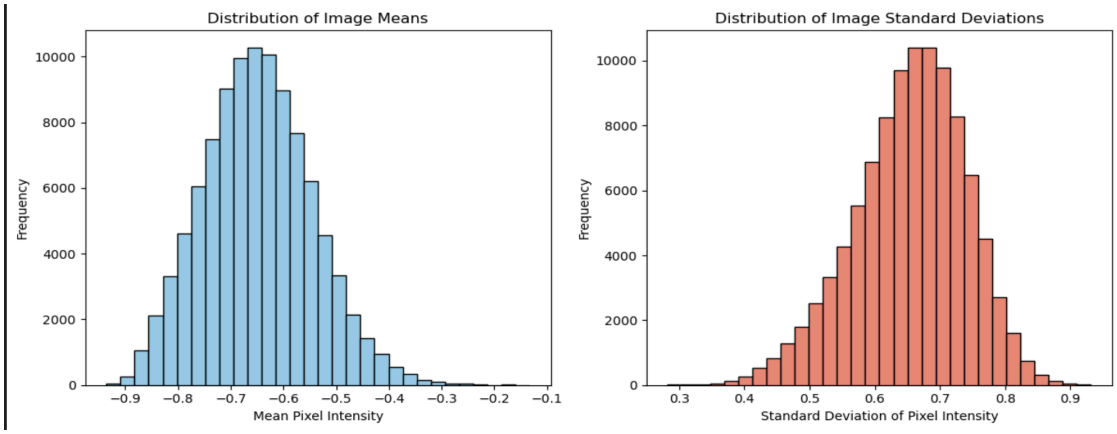
Answer:

<u>Class Distibution Histogram :</u>



This bar chart shows the frequency of each class label in the dataset. Each class, representing a digit (0-9) or letter (A-Z), has approximately the same frequency. Thus our dataset is well-balanced across all classes.

<u>Distribution of Image Mean and Standard Deviation :</u>



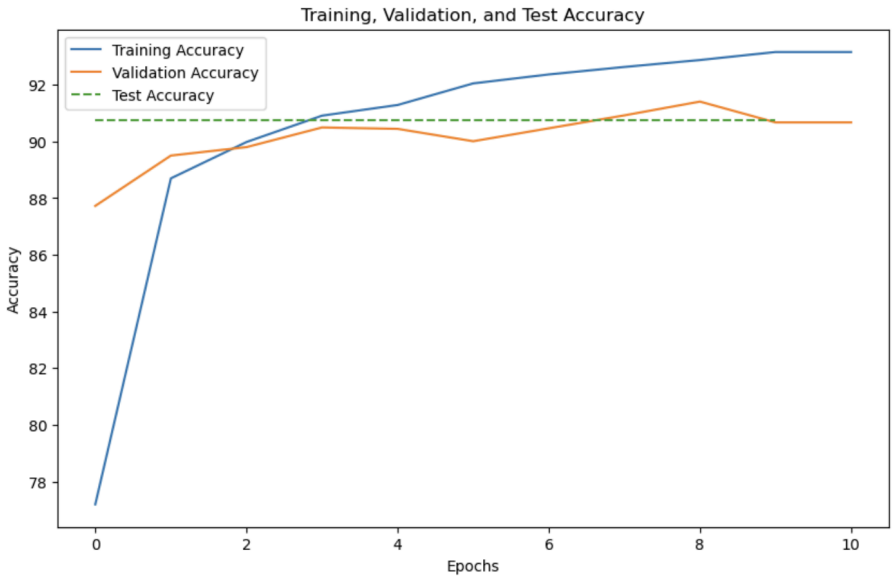
The histogram of mean pixel intensities across images is centered around -0.6, indicating that most images have a normalized average pixel intensity near this value.

The histogram for standard deviations shows a peak around 0.6-0.7, indicating that pixel intensity variations within images are relatively consistent.

This means that the normalization preprocessing which we had applied (mean of 0.5, standard deviation of 0.5) has resulted in a dataset where the images are uniformly scaled.

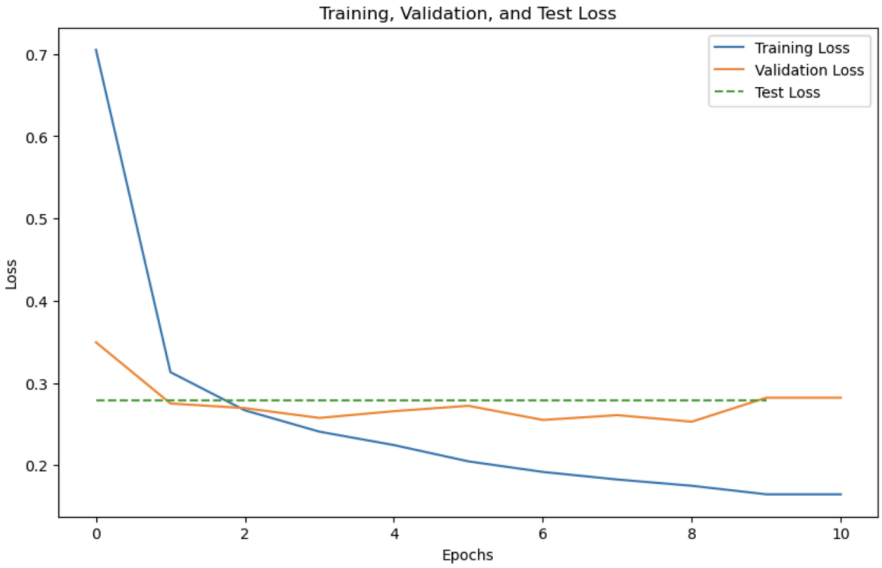
Now we will move forward on reporting the confusion matrix, accuracy and losses for our CNN Basic model.

<u>Training Validation Test accuracy:</u>



The training accuracy continues to increase with each epoch, indicating that the model is learning well from the training data. The validation and test accuracy reach a plateau around 90-92%, which may indicate the model's generalization limit on this data. Also we have test accuracy of 90.77%

Training Validation Test Loss:



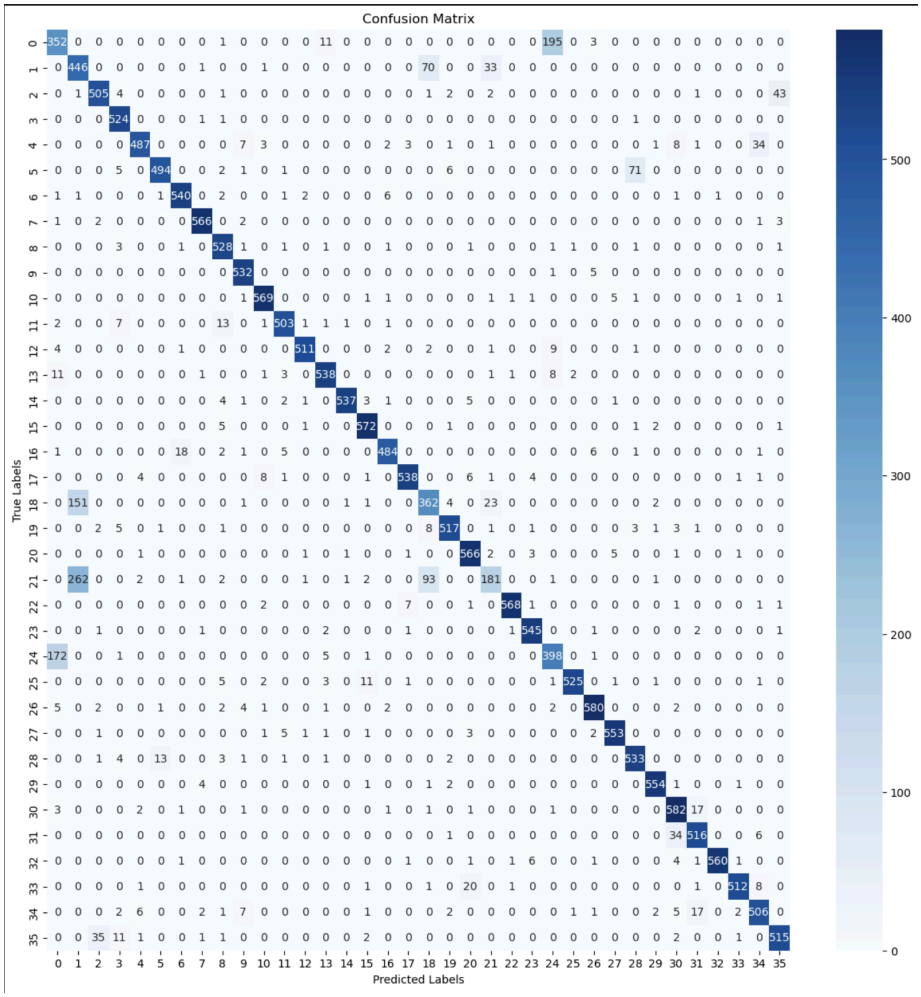
The training loss decreases steadily, while the validation loss plateaus and remains relatively stable. This pattern aligns with the accuracy plot, where the model reaches a consistent level of performance on validation data. The stable test loss suggests that the model has not overfitted and performs consistently on unseen data

Confusion Matrix : (Image on next page)

The confusion matrix visualizes the model's performance on a subset of test data. Each cell represents the number of times the model predicted a particular class when the actual class was something else.

The diagonal cells (where the predicted label matches the true label) have the highest values, indicating that the model generally performs well in correctly classifying the characters. However, some misclassifications are visible, with certain characters being confused with others. For instance, the digits and similar-looking letters (like "O" and

"0") might have occasional confusion. This information can be used to refine the model or data preprocessing to reduce these specific misclassifications.



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

Answer:

Our best model is Learning Rate Scheduler with Accuracy of 91.86% and Loss of 0.2315

CNN Model with Learning Rate Scheduler:

It has achieved optimal results, showing strong generalization and effective training progression.

CNN Architecture

Convolutional Layers:
The model contains four convolutional layers with progressively increasing filter sizes (32, 64, 128, and 256). This helps in gradually learning both low-level and high-level features.

Pooling Layers:
Each convolutional layer is followed by a max-pooling layer (kernel size of 2) to reduce the spatial dimensions, allowing the model to capture more abstract patterns and reduce computation.

Fully Connected Layers:
After flattening the final feature map, a fully connected layer with 512 neurons is used for feature integration before the final output layer.

Dropout:
Dropout with a probability of 0.5 is applied after the first fully connected layer to prevent overfitting by randomly disabling 50% of neurons during training.

Activation Function:
ReLU is used for all hidden layers to ensure computational efficiency and address the vanishing gradient problem, while the final layer uses CrossEntropyLoss which internally applies Softmax for classification.

Learning rate scheduler - IMPROVEMENT #3

```
model = CNN_basic(num_classes=36).to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()

step_size = 5
gamma = 0.1
scheduler = lr_scheduler.StepLR(optimizer, step_size=step_size, gamma=gamma)

num_epochs = 10
scheduler_metrics = {
    "train_loss": [],
    "train_accuracy": [],
    "val_loss": [],
    "val_accuracy": [],
    "learning_rate": [],
    "training_time": None,
    "test_precision":0,
    "test_recall":0,
    "test_f1":0
}
```

Learning Rate Scheduler

- A LR scheduler is used with a step_size of 5 epochs and a gamma of 0.1, reducing the learning rate by 90% after every 5 epochs.
- This gradual decrease in learning rate helped stabilize the training process, allowing the model to converge better, especially in later epochs.
- This model represents an improvement to our training process by incorporating a learning rate scheduler to enhance performance. Specifically, the scheduler is used to reduce the learning rate by a factor of 0.1 every 5 epochs, which helps the model converge more effectively by gradually refining the learning process
- The model setup also includes tracking key metrics like accuracy, loss, precision, recall, and F1 score, allowing a comprehensive assessment of performance improvements.

scheduler_metrics


```
{'train_loss': [0.6840380695821584,
  0.3121818471649292,
  0.2639347040453286,
  0.2385642931459713,
  0.221423581232429 08,
  0.16352320670176396,
  0.14817367203986223,
  0.14049161990879885,
  0.13552667331440224,
  0.12922434044236 88],
 'train_accuracy': [77.53826530612245,
  88.60827664399093,
  90.07369614512471,
  90.90844671201815,
  91.37471655328798,
  93.34325396825396,
  93.8265306122449,
  94.19642857142857,
  94.26020408163265,
  94.54648526077098],
 'val_loss': [0.31011221279637724,
  0.2769590037627311,
  0.2797604820019 0823,
  0.2596478870358726,
  0.2590451161789743,
  0.2290279128341169 4,
  0.2340821653863863 2,
  0.2354190241356816 4,
  0.2401850360881867 3,
  0.2441125826356 06],
 'val_accuracy': [88.21428571428571,
  89.8015873015873,
  89.84126984126983,
  90.23809523809524,
  90.64484126984127,
  91.67658730158 73,
  91.65674603174604,
  91.56746031746032,
  91.69642857142857,
  91.70634920634 92],
 'learning_rate': [0.001,
  0.001,
  0.001,
  0.001,
  0.0001,
  0.0001,
  0.0001,
  0.0001,
  0.0001],
 'training_time': 711.4237658977509,
 'test_precision': 0.9195518105647802,
 'test_recall': 0.9185515873015873,
 'test_f1': 0.9181000365488121,
 'test_loss': 0.23145050203043316,
 'test_accuracy': 91.85515873015873}
```

With a test accuracy of **91.86%**, this model achieves the highest accuracy among all the compared models. This high test accuracy is indicative of the model's ability to correctly classify data on unseen samples, largely attributed to the stability and adaptability provided by the scheduler.

Test precision, recall, and F1 scores for this model are 0.9196, 0.9186, and 0.9181, respectively. These high scores demonstrate that the model is not only accurate but also balanced in its ability to correctly identify positive and negative samples, minimizing both false positives and false negatives.

Training and Performance

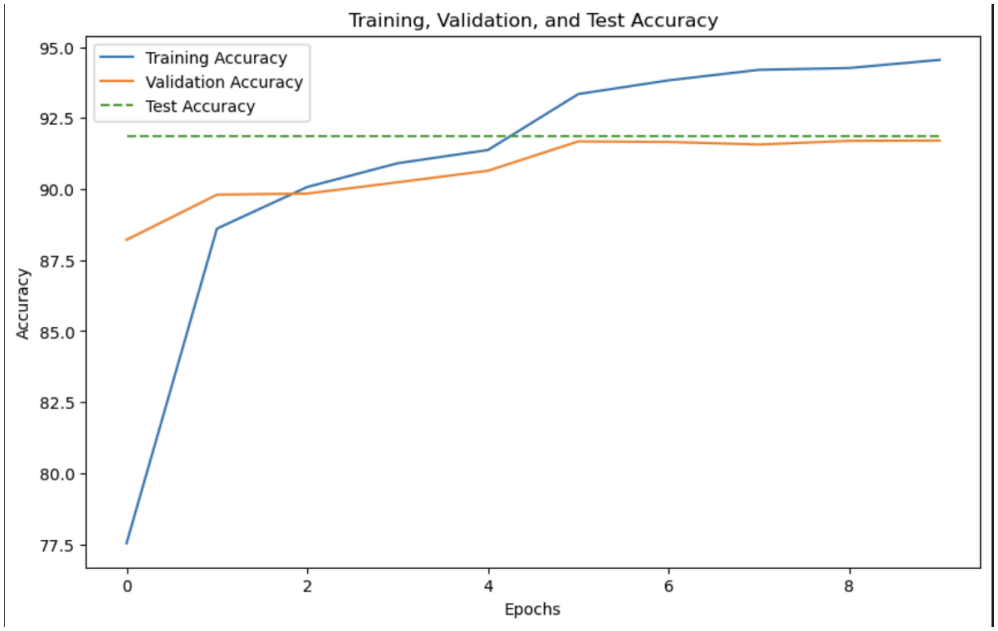
- Initial Learning Rate: Set at 0.001.
- Training Time: Approximately 711.42 seconds across 10 epochs.
- Performance on Validation Set: The model achieved a validation accuracy of approximately 91.71%.
- Test Performance: After training, the model recorded a test accuracy of 91.86%, with a test loss of 0.2315.

Choice Justification

The addition of a learning rate scheduler proved crucial in achieving the best results, as it enabled the model to adjust its learning dynamically, facilitating better convergence and improved generalization. The Learning Rate Scheduler outperforms other models in both test accuracy and test loss, demonstrating superior generalization. By adapting the learning rate, the model achieves a balance between rapid learning and stable fine-tuning, leading to better test results. This model is the most reliable choice when optimizing for accuracy and minimizing loss on unseen data, making it the best performer among the options tested.

4.Provide performance metrics and graphs (Step 7 & 8). Include your detailed analysis of the results.

Train, Validation and Test Accuracy for Learning Rate Scheduler:



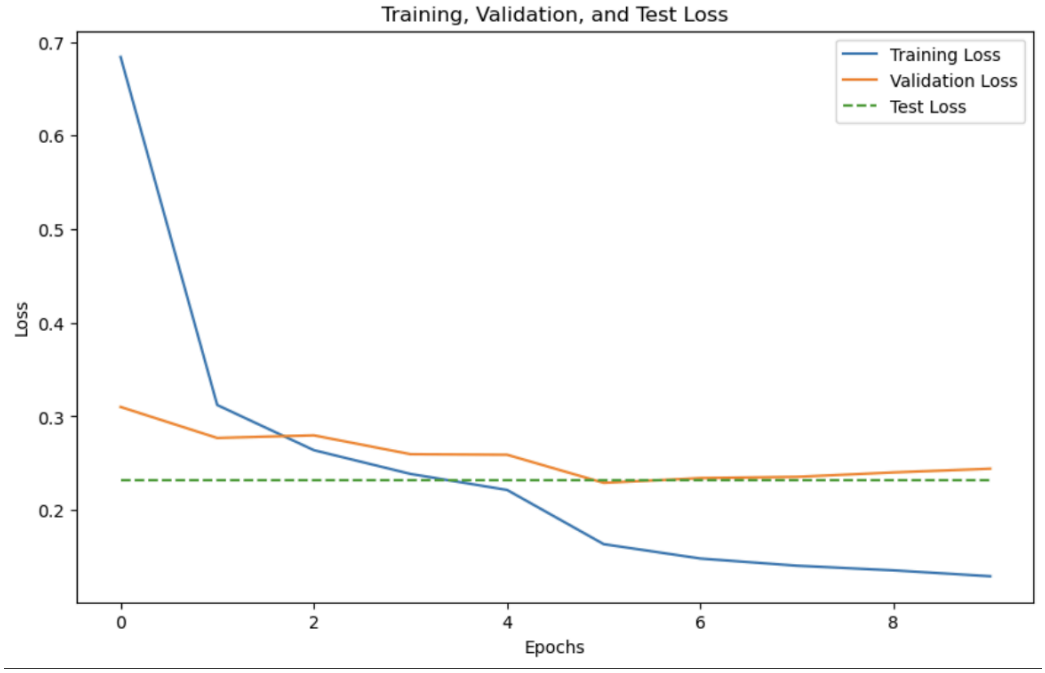
Training Accuracy (blue line): This line increases steadily over the epochs, starting from around 77.5% and reaching approximately 95% by the end. This indicates that the model is learning effectively on the training set, with no signs of underfitting as the accuracy continues to rise.

Validation Accuracy (orange line): The validation accuracy begins around 87.5% and shows gradual improvement, leveling off around 91.5% by the final epoch. The stability of this line indicates that the model generalizes well without significant overfitting, as it maintains a similar trend without sharp drops.

Test Accuracy (green dashed line): The test accuracy is a constant around 91.86%, close to the validation accuracy. This alignment with validation accuracy suggests that the model's performance on unseen data is consistent and reliable.

Reaching high accuracy on both the validation and test sets, while the learning rate scheduler may have contributed to the steady accuracy improvements over epochs without overfitting.

Train, Validation and Test Loss for Learning Rate Scheduler:

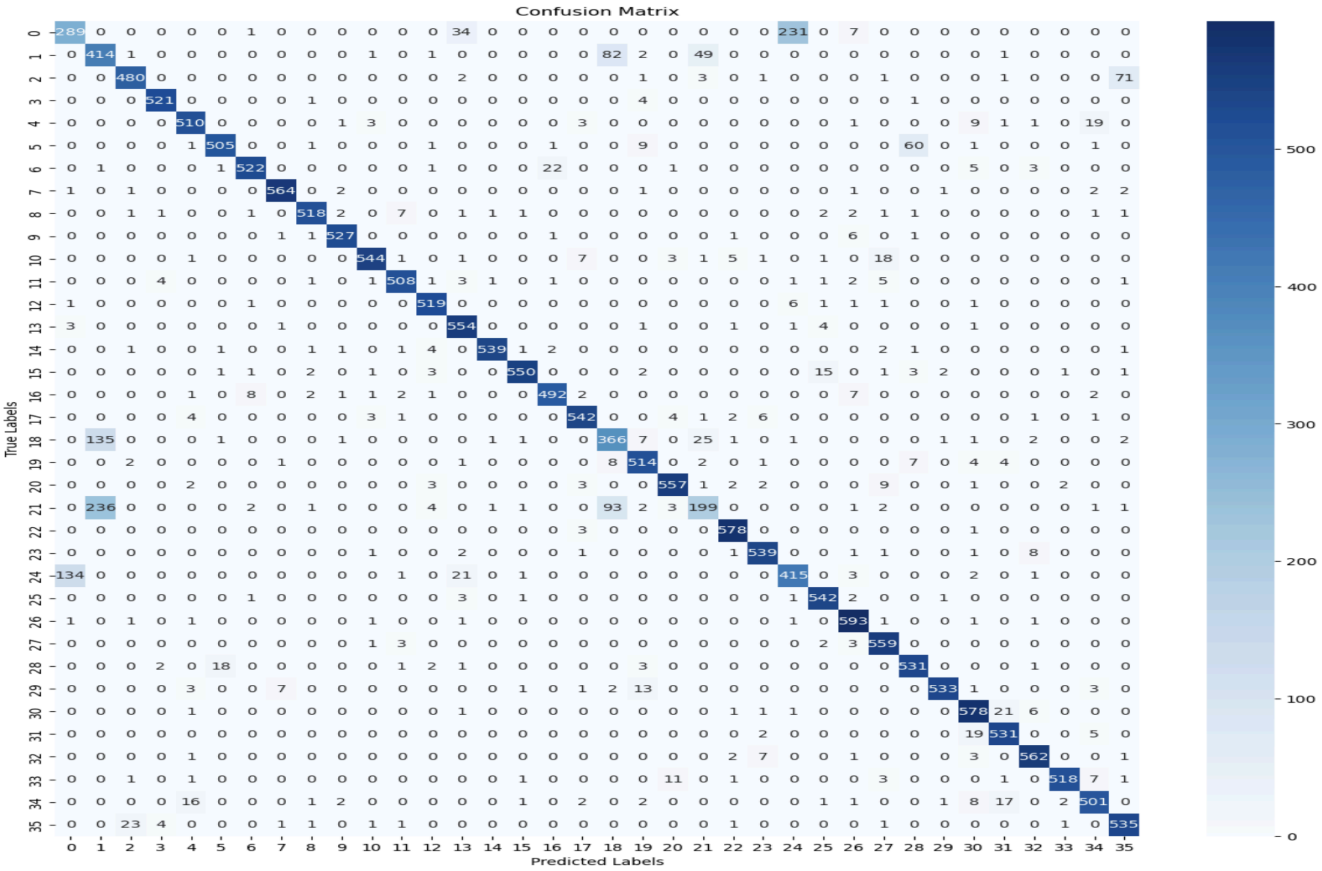


Training Loss (blue line): The training loss decreases sharply in the initial epochs and continues to decline steadily, indicating effective learning and reduction in error on the training data as the model optimizes.

Validation Loss (orange line): The validation loss initially decreases, then stabilizes around the mid-point of training, showing only minor fluctuations. This suggests that the model generalizes well to unseen data without significant overfitting, as the validation loss remains relatively stable despite the continuous decline in training loss.

Test Loss (green dashed line): The test loss is almost constant throughout training and remains close to the validation loss, suggesting that the model's performance on completely unseen data is similar to that on the validation set, further confirming strong generalization.

Confusion Matrix:



In confusion matrix, darker cells along the diagonal indicate a higher number of correctly classified samples for each class.

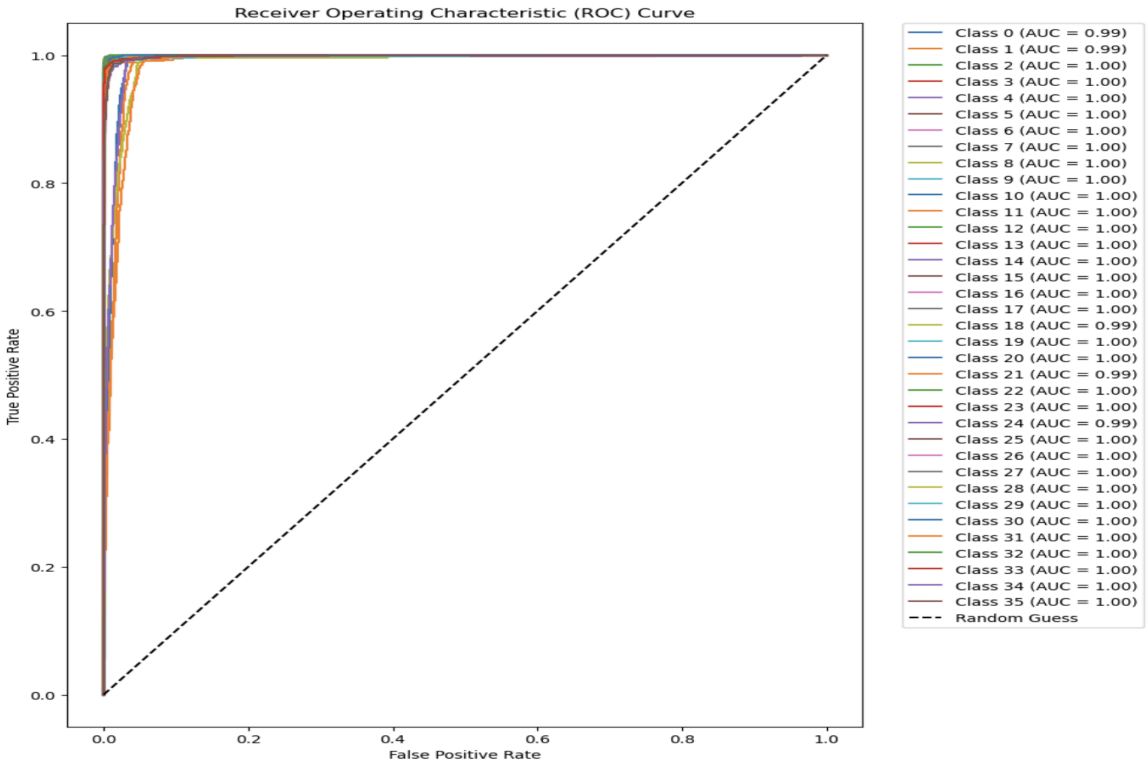
Most values are concentrated along the diagonal, suggesting that the model generally performs well and classifies most samples correctly for each class.

There are some off-diagonal entries, indicating instances where the model predicted the wrong class. For example:

- Class 0 has misclassifications in classes 13 , 24, 26 with noticeable errors in class 24.

Some classes have significantly higher numbers along the diagonal (e.g., classes with counts over 500), indicating higher accuracy, whereas other classes with lower counts along the diagonal may have poorer performance.

ROC :



- The ROC curve plot for each class in this multi-class classification task highlights the model's exceptional performance, with nearly all classes achieving high AUC scores of either 0.99 or 1.00.

- The curves are densely clustered near the top-left corner, indicating that the model maintains high sensitivity and specificity across the board, with minimal variation between classes. This tight clustering and overlap of the ROC curves signify a consistently strong classification capability, as the model can reliably distinguish between different classes without significant misclassification.

- While a few classes, such as Class 0, Class 1, Class 18, Class 22, and Class 23, show slight deviations with AUC scores just below 1.00, these minor differences do not detract from the overall robust performance.
- The dashed diagonal line represents random guessing (AUC = 0.5), and all class curves are well above this baseline, affirming that the model performs substantially better than chance.

Comparison with Basic CNN and improvements:

Model	Test Loss	Test Accuracy
Basic CNN	0.278	90.76
Early Stopping	0.283	91.19
Kfold	0.255	90.52
Learning Rate Scheduler	0.231	91.855 which is (91.86)

Basic CNN Model: The Basic CNN, without any learning rate adjustments, achieves a test loss of 0.279 and test accuracy of 90.76%.

While the model performs well, it does not achieve the same test accuracy as the scheduler model, likely due to the lack of refined learning rate control, which can limit fine-tuning, especially during later training stages.

Early Stopping Model: The Early Stopping model has a slightly lower test loss at 0.283 and an accuracy of 91.20%, which is close to the scheduler model but does not surpass it.

Early stopping is beneficial for avoiding overfitting, but it may cut training short before the model reaches its full potential, as seen here.

K-Fold Cross-Validation Model: The K-Fold model achieves an average test loss of 0.267 and test accuracy around 90.05%, making it consistent across folds but still slightly less accurate than the scheduler model.

This approach's strength lies in verifying robustness across different data splits, though it sacrifices some accuracy and efficiency compared to the learning rate scheduler. The lower accuracy on this could be the factor that we have used less number of epoch and folds.

PART IV

Note: The confusion matrix & ROC Curve is annotated with labels from Class 0 to Class 35, where Classes 0 to 9 correspond to the digits 0 to 9, and Classes 10 to 35 represent the alphabetic characters A through Z.

Question 1: Provide the model details of your CNN.

The CNN model in part 4 resembles the VGG13 configuration with Local Response Normalization (LRN) in layer 1 as proposed. The model has 11 weight layers arranged in a structural manner across multiple **blocks**. Each block captures different levels of hierarchies of features. The architecture employs a series of small 3×3 convolutional filters throughout the network. Using smaller convolutional filters allows for deeper architectures without a substantial increase in parameters.

In the first block, CNN has two convolutional layers with 64 **filters**. These filters detect the basic patterns in the input. As we go deeper in the network, the depth of layers increases sequentially. The second block has two convolutional layers with 128 filters, followed by a third block with two layers using 256 filters. In the final two blocks, the model utilizes two convolutional layers with 512 filters each. This enables the model to learn more complex representations.

The **LRN** layer applied in the first block, after the first convolutional layer. LRN operates by normalizing the responses of neurons within a local region and thereby allowing for better contrast between neuron activities. It also helps prevent overfitting during training.

The **max-pooling** layers are included after the first 3 blocks. These use a kernel size of 2x2 with a stride of 2 to reduce the spatial dimension of the corresponding output feature maps. This helps reduce computational overhead.

At last, three **fully connected layers** that bring together the features learned in the earlier convolutional layers. The first two layers have 4096 neurons each, which helps the model understand more complex relationships in the data. The final fully connected layer has 1000 neurons, one for each output class. These layers are essential for interpreting the patterns learned from the previous layers and making the final classification decisions.

To avoid overfitting, **dropout** is applied to these fully connected layers, usually with a probability of 0.5. This means that during training, some neurons are

randomly “dropped” or ignored, encouraging the model to rely on a broader range of neurons rather than depending too much on any single one. The final layer is a **softmax layer**, which converts the output to probabilities, making it suitable for multi-class classification tasks.
ReLU (Rectified Linear Unit) activation functions are used after each layer, introducing non-linearity essential for learning complex patterns in the data.
StepLR learning rate scheduler is used in the code. Learning rate schedulers improve the training process by adjusting the learning rate over time. The learning rate is reduced as we go into the higher epochs thus leading to improved generalization and faster convergence.
By increasing the depth of the network and combining max-pooling with LRN, dropout, and ReLU, this model achieves a balance between accuracy, computational efficiency, and generalization.

The **dataset preparation** begins with defining preprocessing transformations that include resizing to **28x28 pixels**, random **horizontal flipping**, slight **rotation**, and **normalization** based on ImageNet mean and standard deviation values. The dataset, loaded using PyTorch’s ImageFolder, is split into **training (70%), validation (10%), and test (20%) sets** for balanced model evaluation. Data loaders are optimized with a **batch size of 64,** shuffling (for training), and settings like num_workers and pin_memory to enhance data loading efficiency on compatible hardware. This setup ensures a streamlined and efficient data pipeline for model training and evaluation.

After multiple training sessions, after observing the trends in accuracy - epoch, we have set the number of epochs to 6 as a compromise for accuracy and computational overhead.

Testing over 6 epoch resulted in

```
Epoch 1/6
Training: 100%|██████████| 1103/1103 [14:08<00:00,  1.30batch/s]
/opt/homebrew/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1531
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Epoch [1/6] Training Loss: 2.4380, Training Accuracy: 21.03%
Validation Loss: 1.2314, Validation Accuracy: 55.49%
Best model saved with validation accuracy: 55.49%
Epoch 2/6
Training: 100%|██████████| 1103/1103 [14:08<00:00,  1.30batch/s]
Epoch [2/6] Training Loss: 0.8048, Training Accuracy: 72.43%
Validation Loss: 0.6314, Validation Accuracy: 78.97%
Best model saved with validation accuracy: 78.97%
Epoch 3/6
Training: 100%|██████████| 1103/1103 [14:07<00:00,  1.30batch/s]
Epoch [3/6] Training Loss: 0.5105, Training Accuracy: 82.79%
Validation Loss: 0.4595, Validation Accuracy: 84.84%
Best model saved with validation accuracy: 84.84%
Epoch 4/6
Training: 100%|██████████| 1103/1103 [14:09<00:00,  1.30batch/s]
Epoch [4/6] Training Loss: 0.4333, Training Accuracy: 85.43%
Validation Loss: 0.4140, Validation Accuracy: 85.82%
Best model saved with validation accuracy: 85.82%
Epoch 5/6
Training: 100%|██████████| 1103/1103 [14:11<00:00,  1.30batch/s]
Epoch [5/6] Training Loss: 0.3884, Training Accuracy: 86.83%
Validation Loss: 0.3978, Validation Accuracy: 86.79%
Best model saved with validation accuracy: 86.79%
Epoch 6/6
Training: 100%|██████████| 1103/1103 [14:08<00:00,  1.30batch/s]
Epoch [6/6] Training Loss: 0.3021, Training Accuracy: 89.35%
Validation Loss: 0.3279, Validation Accuracy: 88.78%
Best model saved with validation accuracy: 88.78%
```

Question2: Discuss how the architecture is different from your CNN architectures defined in Part III.

Comparison with best model of Part III:

The VGG model in Part 4 is a modified VGG-A with Local Response Normalization (LRN) layers. Only the first three of its **five convolutional blocks** make use of MaxPooling. Each block employs a stack of small 3 x 3 convolution filters. The model has three fully connected layers, designed for a 36-class classification problem. The model is optimized for regularization and feature extraction with the addition of LRN layers, which reduce overfitting and improve

generalization by normalizing activations. It used Adam optimizer and StepLR learning rate scheduler.

In part 3, the best model is the one optimized with a learning rate scheduler. The scheduler is added to the standard CNN architecture three convolutional layers, followed by max-pooling layers. The design was simpler than VGG's, focusing on efficiency and optimized training through a dynamic learning rate strategy.

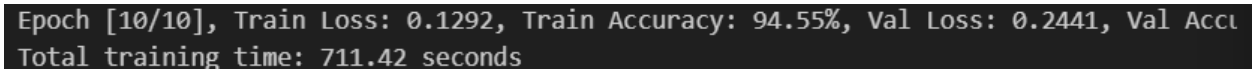
Some key differences:

- **Model complexity:**
  Part 4 VGG is significantly deeper, which enables better performance but also increases computational cost. It is a multi-layered architecture inspired by the VGG-A configuration, modified with Local Response Normalization (LRN) layers and a classifier comprising three fully connected (FC) layers. This VGG model contains five primary blocks of convolutional layers, with MaxPooling only applied in the first three blocks. Each block increases the feature map depth while maintaining a relatively compact spatial dimension. Such a design provides more comprehensive analysis of input data but is computationally more complex.

  Each epoch takes ~ 14 minutes to train!

  In contrast, Part 3's model has fewer layers, offering a faster training time but potentially less feature-capturing capacity. It includes only three convolutional layers, each followed by MaxPooling. The smaller number of convolutional layers means fewer trainable parameters, which reduces overfitting risk and makes the model faster to train.

  Each epoch takes ~ 1 minute to train!

- **Regularization and Normalization:**
  VGG in Part 4 utilizes LRN layers to stabilize training and improve

generalization capacity of the model. It normalizes response across channels leading to diverse feature learning.
Additionally, the dropout feature employed further mitigates overfitting.
Whereas Part 3's CNN model does not employ special normalization techniques and relies more on data normalization and basic regularization techniques. This simpler approach fits with its lighter architecture at the cost of robustness against overfitting.

- **Training and Optimization:**
  Both models incorporate learning rate scheduling, but VGG's stability benefits additionally from LRN and dropout layers in the fully connected layers.
  Best model of part 3 also employs a learning rate scheduler.
  Both models use Adam optimizer well-suited for models with high parameter counts, which makes learning more stable despite the deeper architecture.
- **Input Preprocessing and Data Augmentation:**
  In Part 4, the VGG model leverages data augmentation strategies like horizontal flipping and slight rotation. This approach simulates a more varied dataset, allowing the model to generalize better to new data.
  In Part 3, the CNN model preprocesses images by converting them to grayscale, which reduces the computational load while retaining essential information for character recognition tasks.
  Both models have the input resized to 28x28 pixels.

<u>Comparison with base model of Part III:</u>

In the base model of Part III, the *CNN_basic* model is a straightforward convolutional neural network with four convolutional layers, each followed by a ReLU activation and max-pooling. It starts with a layer using 32 filters, then moves up to 64 filters in the second layer, 128 in the third, and finally 256 in the fourth layer. After each of these layers, max-pooling reduces the spatial dimensions of the feature maps, which helps simplify the data and cut down on computational demands. The fully connected part of the model is simple, too, with just one hidden layer of 512 neurons, followed by a dropout layer (with a 50% probability to

prevent overfitting) and a final output layer that matches the number of classes (36).

In contrast, the Part IV model is a much more advanced setup, based on a VGG13-style architecture. It's deeper and more complex than the Part III model, with more convolutional layers and a higher variety of filter sizes, allowing it to capture richer and more detailed features at each layer. Like other VGG-inspired models, it uses small 3×3 convolutional filters across all layers, which helps keep the parameter count under control even as depth increases. This added depth allows the model to pick up on more complex patterns, which is especially useful for challenging classification tasks.

One of the big differences is that Part IV includes more sophisticated regularization and optimization techniques. For instance, it uses Local Response Normalization (LRN) after the first block of layers, batch normalization, and a learning rate scheduler. These additions help make the model more robust and improve its ability to generalize to new data. The learning rate scheduler in Part IV automatically adjusts the learning rate as training progresses, allowing the model to make smaller, more refined updates over time. In Part III, however, the model sticks to a fixed learning rate.

In short, the Part IV model builds on a more sophisticated and deeper architecture, inspired by VGG designs. With extra layers, more filters, and advanced techniques like learning rate scheduling and batch normalization, it's a more powerful model that tends to perform better than the simpler Part III model in terms of both accuracy and ability to generalize to new data.

Question3: Provide the performance metrics and graphs (Step 5) and compare them with the results obtained in Part III.

Performance metrics of VGG 13:

Final training metrics:
Training loss: 0.3021
Training accuracy: 89.35%

Validation loss: 0.3279
Validation accuracy: 88.35%

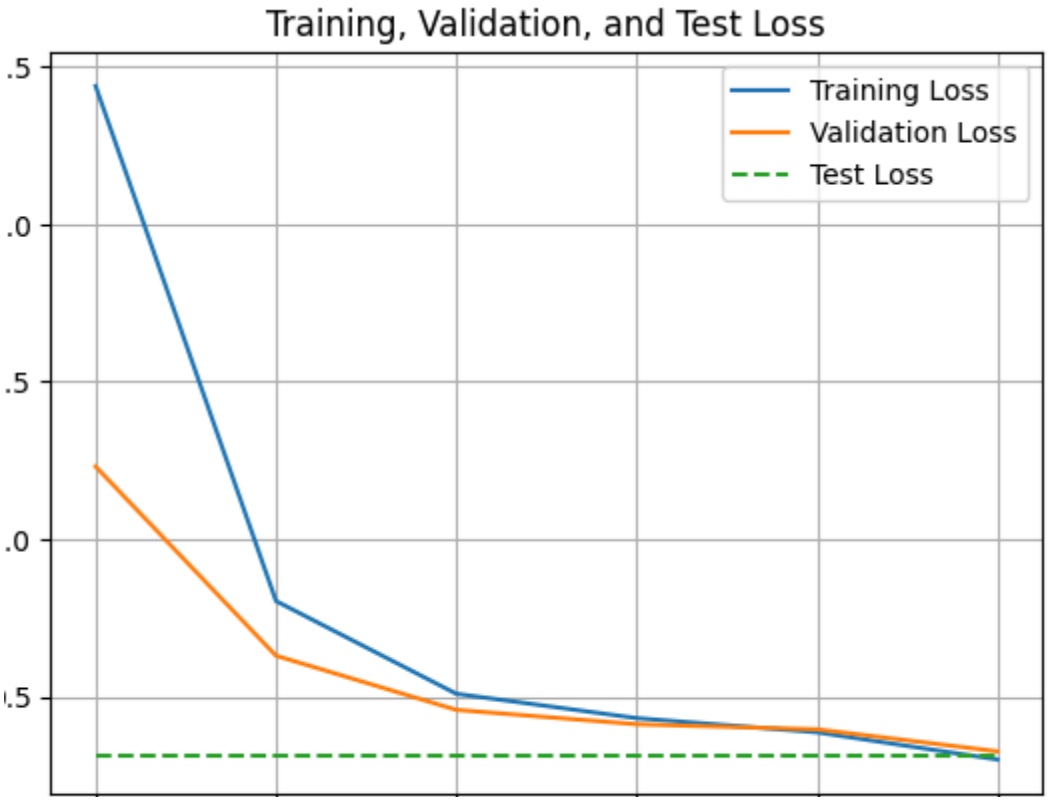
Accuracy across epochs:

- The training accuracy started lower but quickly improved, showing a steep increase in the initial epochs before gradually leveling off. This suggests that the model was effectively learning the features from the training data.

- The validation accuracy follows a similar trend, rising swiftly in early epochs and stabilizing close to the training accuracy as the model approached convergence.
- The test accuracy, represented by a constant line, shows the model's performance on unseen data, providing a benchmark that closely aligns with the final training and validation accuracies.

Similarity between validation and test accuracy indicates that the model generalizes well, with minimal overfitting.

Loss across epochs:

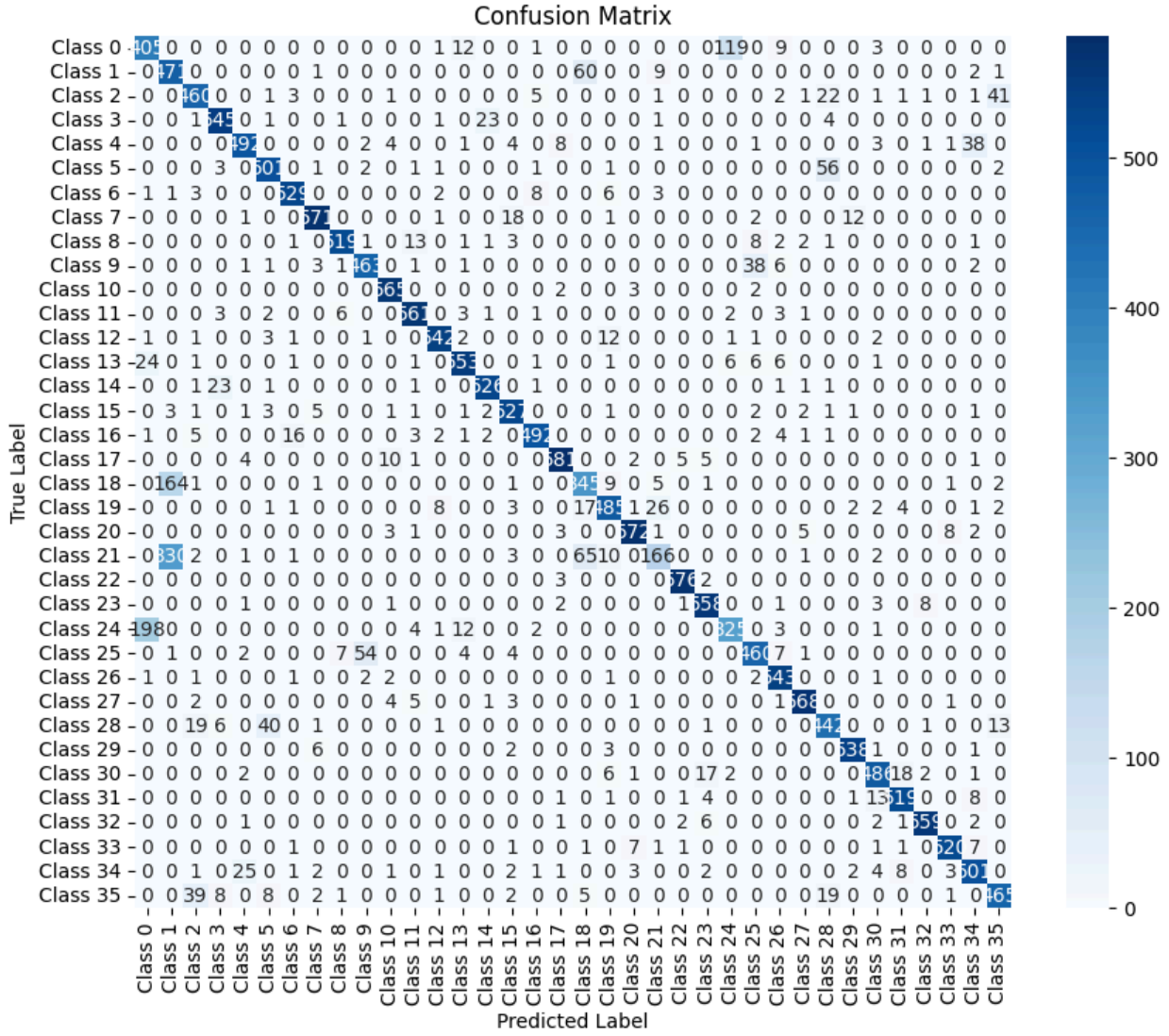


- The training loss dropped significantly in the first few epochs and then continued to decline at a slower rate, indicating that the model was learning effectively.
- The validation loss followed a similar pattern, decreasing rapidly before stabilizing, which suggests that the model maintained consistency on both training and validation sets.

- The test loss, shown as a constant line across epochs, aligns closely with the final training and validation losses, further confirming the model's generalization capability.

The low final values of all three losses indicate that the model reached a stable point, minimizing errors on both training and unseen data.

Confusion Matrix:

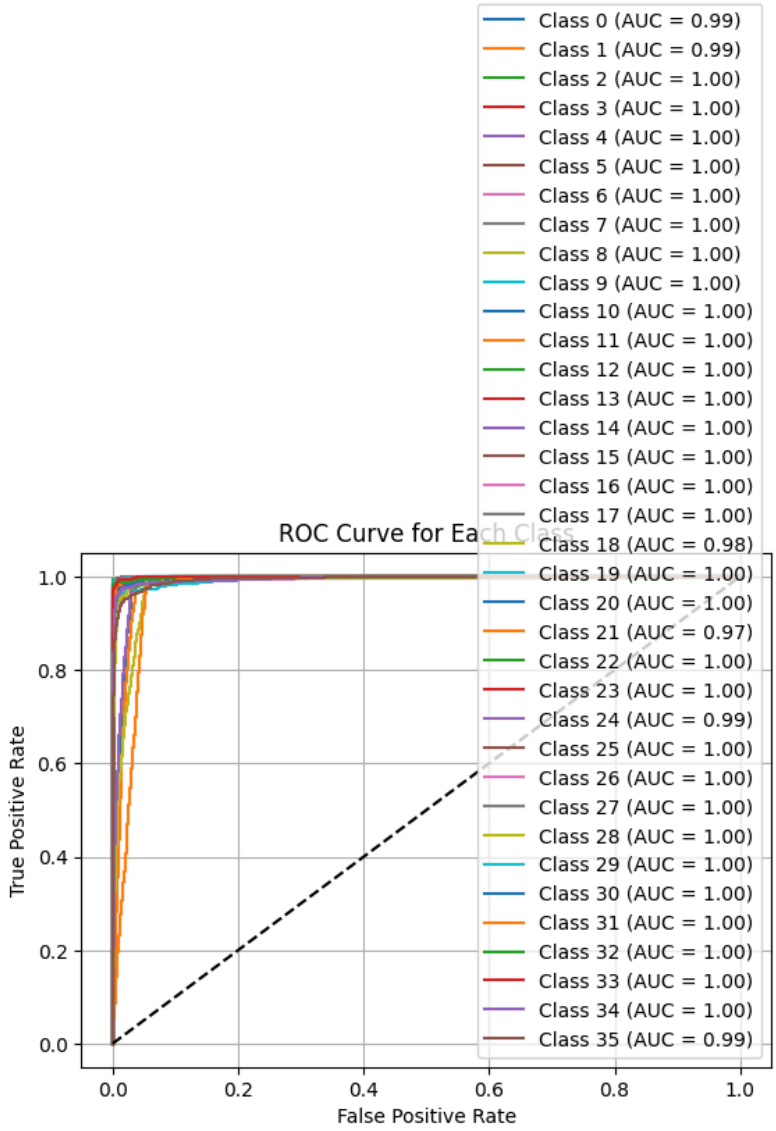


- The confusion matrix provides detailed insights into the model's classification performance across all classes.
- Each cell shows the count of predictions made for each class, with diagonal cells representing correct predictions.

- The high values along the diagonal indicate that the model successfully identified most samples correctly for each class.
- Off-diagonal values represent misclassifications, with lower values generally suggesting that misclassifications were relatively infrequent.
- This matrix highlights the model's effectiveness in distinguishing between classes, with only a few errors scattered across some classes.

The matrix reinforces the model's strong performance, particularly for classes with higher counts along the diagonal.

ROC:



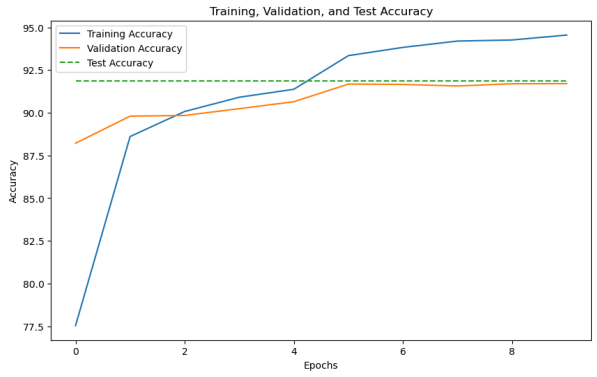
- The ROC curve shows the true positive rate versus the false positive rate for each class, with the area under the curve (AUC) indicating classification quality.
- Each class has an ROC curve, and the AUC values for most classes are close to 1.0, indicating near-perfect classification.
- The model consistently achieves high AUC values, suggesting strong sensitivity and specificity across all classes.
- For classes with AUC close to 1, the model demonstrates excellent discrimination, reliably identifying instances of that class with minimal false positives.

The overall ROC analysis confirms the model's robustness and ability to handle multi-class classification effectively.

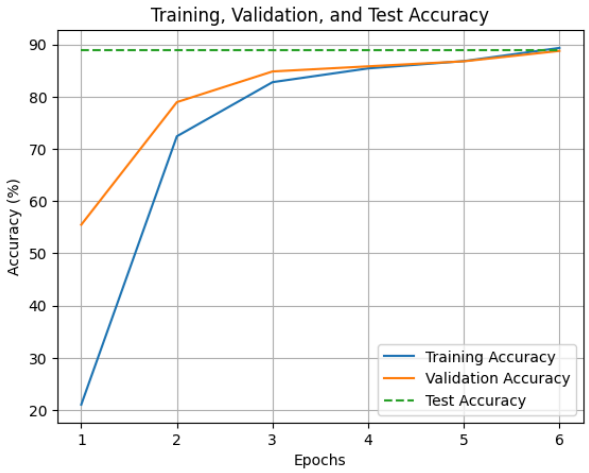
Comparison with part III: (best model)

Accuracy:

Part 3 model:



Part 4 model:

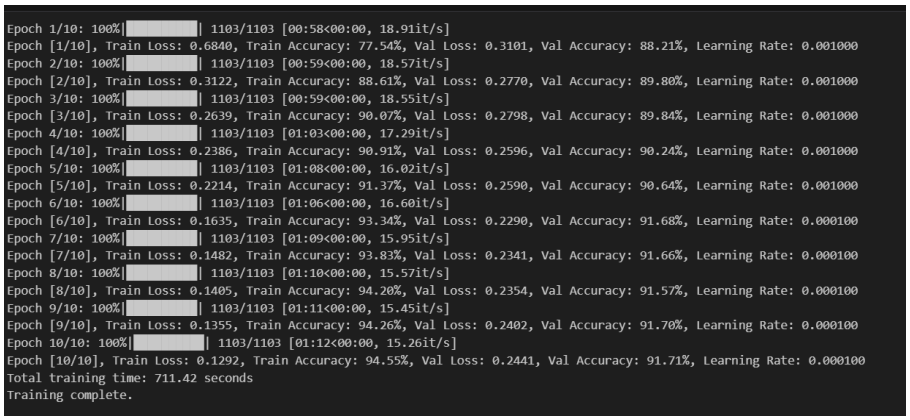


➔ In Part III, the CNN model optimized with learning rate scheduler achieved a test accuracy around 91.86%

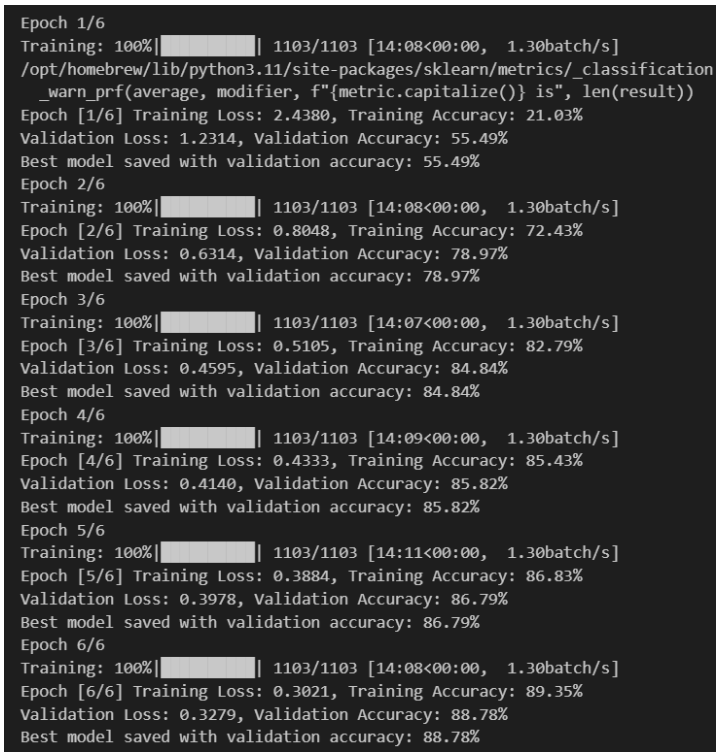
➔ In Part IV, with the addition of the learning rate scheduler and other refinements, the model reached a higher test accuracy of 88.99%.

In part 3, the model is trained over 10 epochs and for part 4 only 6. However the difference in accuracy between the two models is very small. This slight difference in accuracy suggests that CNN model with VGG architecture is the superior one. It allowed the model to better fine-tune its weights over time, leading to enhanced performance.

Part3:

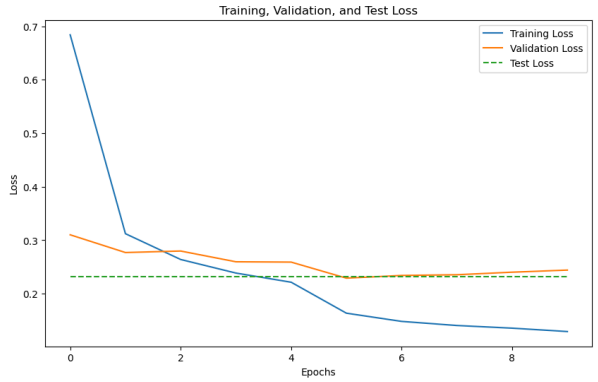


Part 4:

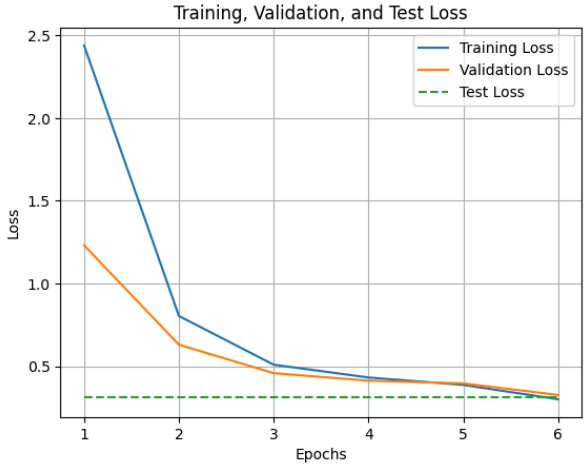


Loss:

Part 3 model:



Part 4 model:



➔ The best CNN model in Part III showed a test loss of approximately 0.278
➔ In Part IV, the test loss is 0.3147

Similar to accuracy, the model in part 4 achieves better reduction in loss in **less number epochs.** This reinstates the strength and depth of the VGG13 based CNN model compared to simple CNN with learning rate scheduler.

Confusion Matrix:

➔ In Part III, the confusion matrix indicated that while the model performed well on most classes, certain classes with similar appearances (e.g., “O” vs. “0”) faced misclassification challenges.
➔ Part IV model’s confusion matrix shows an improvement in classification accuracy across most classes, with **fewer off-diagonal values**

This suggests that the Part IV model, aided by the learning rate scheduler, batch normalization, and other enhancements, was better at distinguishing between challenging classes

ROC Curve:

➔ Part III’s ROC curve showed an AUC score around **0.85**, reflecting reasonably strong classification performance but with room for improvement

➔ In Part IV, the ROC curves for each class demonstrated near-perfect AUC values close to **1.0** for most classes, indicating very high sensitivity and specificity.

This improvement in the ROC-AUC values shows that Part IV's model achieved **better separation** between classes, leading to fewer false positives and false negatives.

REFERENCES:

https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739
https://www.w3schools.com/python/python_ml_auc_roc.asp
EarlyStopping — PyTorch-Ignite v0.5.1 Documentation
KFold — scikit-learn 1.5.2 documentation
ReduceLROnPlateau — PyTorch 2.5 documentation
https://www.geeksforgeeks.org/cross-validation-using-k-fold-with-scikit-learn/
https://www.geeksforgeeks.org/using-early-stopping-to-reduce-overfitting-in-neural-networks/
https://www.geeksforgeeks.org/what-is-batch-normalization-in-deep-learning/implementing-batch-normalization-in-pytorch
https://www.geeksforgeeks.org/understanding-pytorch-learning-rate-scheduling/
https://careerfoundry.com/en/blog/data-analytics/how-to-find-outliers/:~:text=Cap%20the%20outliers,will%20be%20set%20to%2020.
https://arxiv.org/abs/1409.1556
https://library.fiveable.me/key-terms/deep-learning-systems/local-response-normalization
https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/
https://builtin.com/machine-learning/relu-activation-function#:~:text=The%20ReLU%20activation%20function%20is%20used%20to%20introduce%20nonlinearity%20in,more%20complex%20relationships%20in%20data.
https://www.geeksforgeeks.org/auc-roc-curve/
https://stackoverflow.com/questions/50825936/confusion-matrix-on-images-in-cnn-keras
https://www.geeksforgeeks.org/how-to-plot-normal-distribution-over-histogram-in-python/
https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
https://pytorch.org/tutorials/beginner/saving_loading_models.html
https://pytorch.org/docs/stable/nn.html
https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity
https://pytorch.org/vision/main/generated/torchvision.datasets.ImageFolder.html
https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html#torch.optim.lr_scheduler.ReduceLROnPlateau
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
https://pytorch.org/ignite/generated/ignite.handlers.early_stopping.EarlyStopping.html