Assignment – 2

Contribution Table

Name	Parts	Contribution
Ruthvik Vasantha Kumar	1 & 2 & 3 & 4 & bonus	50%
Shishir Hebbar	1 & 2 & 3 & 4 & bonus	50%

Part – 1:

1. Overview of the dataset:

	f1	f2	f3	f4	f5	f6	f7	target
0	6	148	72	35	0	33.6	0.627	1
1	1	85	66	29	0	26.6	0.351	0
2	8	183	64	0	0	23.3	0.672	1
3	1	89	66	23	94	28.1	0.167	0
4	0	137	40	35	168	43.1	2.288	1
...	...	...	...	...	...	...	...	...
761	9	89	62	0	0	22.5	e	0
762	10	101	76	48	180	d	0.171	0
763	2	122	70	27	b	36.8	0.34	0
764	c	121	72	23	112	26.2	0.245	0
765	1	126	60	a	0	30.1	0.349	1

766 rows × 8 columns

Shape: 766 rows and 8 columns

Dataset statistics before cleaning:

	f1	f2	f3	f4	f5	f6	f7	target
count	766	766	766.000000	766	766	766	766	766.000000
unique	18	137	NaN	52	186	249	517	NaN
top	1	100	NaN	0	0	32	0.258	NaN
freq	134	17	NaN	226	372	13	6	NaN
mean	NaN	NaN	69.118799	NaN	NaN	NaN	NaN	0.349869
std	NaN	NaN	19.376901	NaN	NaN	NaN	NaN	0.477240
min	NaN	NaN	0.000000	NaN	NaN	NaN	NaN	0.000000
25%	NaN	NaN	62.500000	NaN	NaN	NaN	NaN	0.000000
50%	NaN	NaN	72.000000	NaN	NaN	NaN	NaN	0.000000
75%	NaN	NaN	80.000000	NaN	NaN	NaN	NaN	1.000000
max	NaN	NaN	122.000000	NaN	NaN	NaN	NaN	1.000000

Dataset statistics after cleaning:

	f1	f2	f3	f4	f5	f6	f7	target
count	766.000000	766.000000	766.000000	766.000000	766.000000	766.000000	766.000000	766.000000
mean	3.849673	120.909804	69.118799	20.542484	80.091503	31.998170	0.472128	0.349869
std	3.371490	31.927057	19.376901	15.950080	115.298950	7.893111	0.331328	0.477240
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	0.000000
25%	1.000000	99.000000	62.500000	0.000000	0.000000	27.300000	0.244000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	36.000000	32.000000	0.374500	0.000000
75%	6.000000	140.000000	80.000000	32.000000	127.750000	36.600000	0.625500	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	1.000000

```
f1          float64
f2          float64
f3            int64
f4          float64
f5          float64
f6          float64
f7          float64
target        int64
dtype: object
```

2. Dataset preprocessing steps:

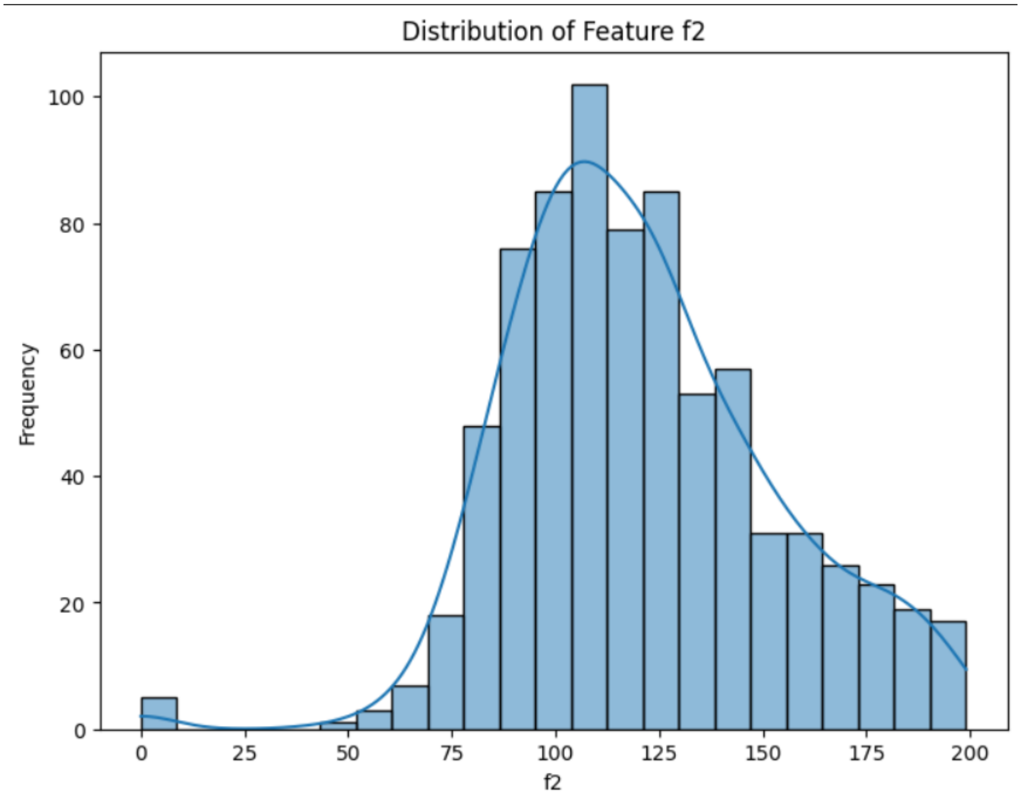
- Handling Invalid Characters in Numeric Columns:
    - Step: A loop is run over each column in the dataset (df.columns), and the unique values in each column are checked.
    - Observation: Some numeric columns contain non-numeric values such as alphabets.
    - Action Taken:
        - The pd.to_numeric() function is applied to each column with the argument errors='coerce'. This converts non-numeric values to NaN (missing values).
        - After converting invalid values to NaN, the missing values are filled with the mean of the respective column using df[col].fillna(df[col].mean(), inplace=True).

- Data Normalization:
  - Step: The dataset is normalized to ensure that all features (except the target column) have values on the same scale.
  - Action Taken:
    - A StandardScaler() is used to normalize all columns except the target column.
    - The target column ('target') is dropped using df.drop('target', axis=1) to create df1 (the features dataset).
    - The remaining features are scaled using scaler.fit_transform(df1[column]), and the 'target' column is added back to df1.
- Data Splitting and Oversampling:
  - Step: To handle class imbalance, Random Oversampling is used.
  - Action Taken:
    - The feature matrix X is separated from the target labels Y by dropping the 'target' column.
    - The target labels are extracted into Y = df['target'].
    - An oversampler (RandomOverSampler) is applied to balance the class distribution in X and Y.
- Train-Test-Validation Splitting:
  - Step: The dataset is split into training, validation, and test sets in a two-step process:
    1. First, the dataset is split into 85% training and 15% testing sets using train_test_split.
    2. The training data is further split into 82.5% training and 17.5% validation using another train_test_split.
  - Result: The resulting splits are:
    - X_train, X_val, X_test: Feature matrices for the training, validation, and test sets.
    - y_train, y_val, y_test: Corresponding labels for the training, validation, and test sets.
- Conversion to PyTorch Tensors:

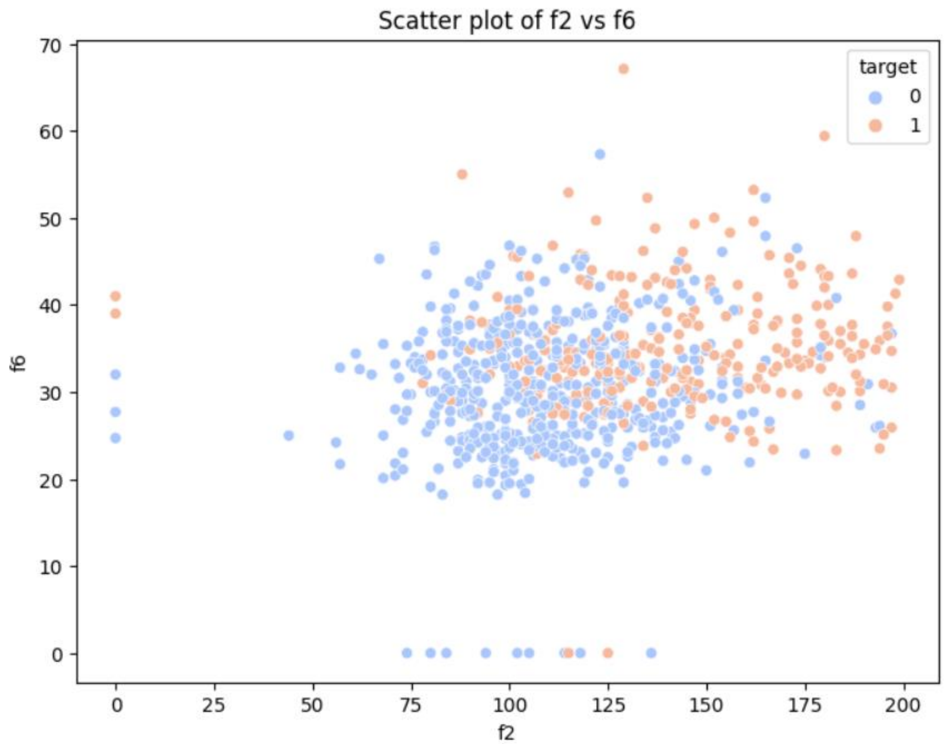
- Step: The data is converted into PyTorch tensors for model training.
- Action Taken:
  - The feature matrices and target labels are converted to PyTorch tensors using torch.tensor().
  - Data type torch.float32 is specified for numerical consistency.

- Dataset Information:
  - Size of Tensors:
    - The sizes of the training, validation, and test sets are printed using tensor.shape to verify the correct splits:
      - X_train_tensor: Shape of training features
      - Y_train_tensor: Shape of training labels
      - X_val_tensor: Shape of validation features
      - Y_val_tensor: Shape of validation labels
      - X_test_tensor: Shape of test features
      - Y_test_tensor: Shape of test labels

3. Graphs:



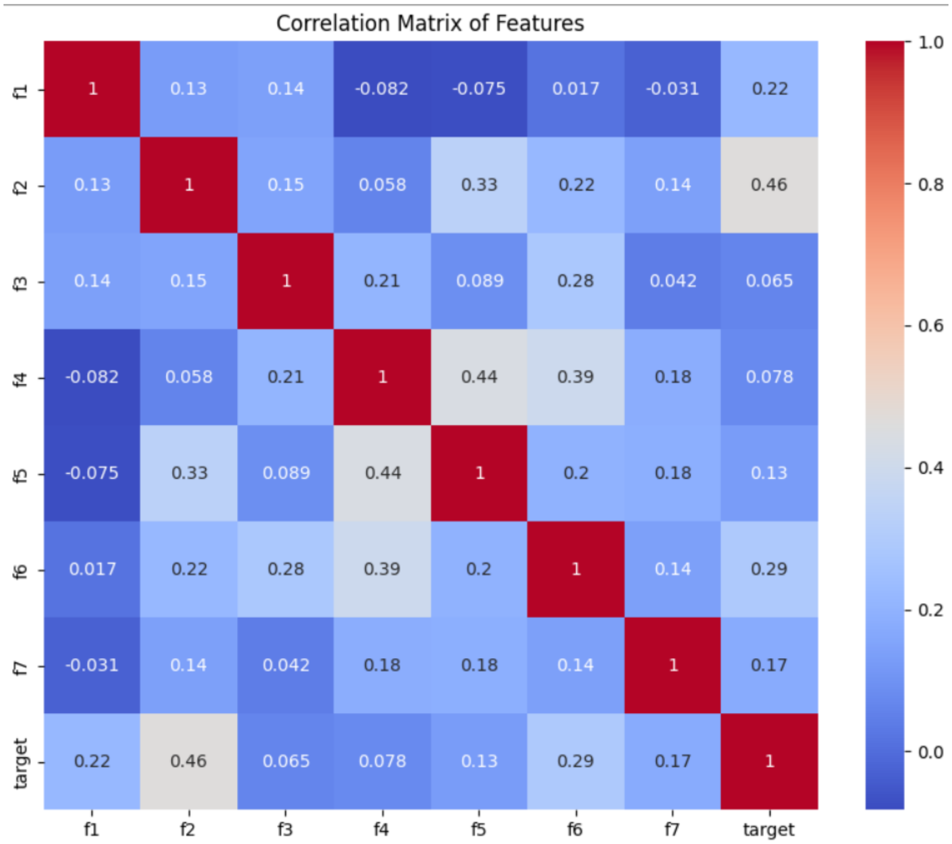
- The distribution of f2 is right-skewed, with the majority of values falling between 75 and 150.
- There is a clear peak (mode) around the value of 100, suggesting that most of the observations for f2 are concentrated around this range.
- The distribution tapers off significantly as the values approach 0 and beyond 150.
- This could indicate that f2 is a feature with a somewhat normal-like distribution but with a skew towards lower values.



- Pattern Analysis:
    - The scatter plot shows that f2 and f6 are positively correlated, meaning that as f2 increases, f6 tends to increase as well.
    - There is a wide spread of points, but most data points are concentrated in a central cluster.
- Target Distribution:
    - The data points are color-coded by the target (blue for class 0, orange for class 1).

- Both classes are spread across the feature space, but class 1 (orange) tends to be slightly more frequent in the higher ranges of f2 and f6. This suggests that higher values of f2 and f6 may be more indicative of class 1.

- However, there is also significant overlap between the two classes, meaning that separating the two purely based on these features may not be straightforward, but these features could still have predictive power.



- f2 and the target: There is a moderately positive correlation (0.46) between feature f2 and the target, which suggests f2 could be useful in predicting the target.
- f6 and the target: A moderate positive correlation (0.29) exists between f6 and the target, also indicating potential predictive value.
- f1 and the target: Feature f1 has a smaller positive correlation (0.22) with the target.
- f4, f5, and the target: These features show weaker correlations with the target (0.078 and 0.13, respectively), meaning they may have a less significant influence.
- Correlations between features:

- f5 and f4 are strongly correlated (0.44), suggesting potential multicollinearity, where these features may be redundant.
- f6 also shows moderate correlations with several features like f4 (0.39) and f2 (0.22), meaning f6 shares information with these features.

- In general, the features f2, f6, and f1 appear to be the most important for predicting the target, while multicollinearity could be a concern for features f5 and f4.

4. Summary of the Neural Network (NN) Model:

1. Architecture:

    Input Layer: Takes the input data (features).

    Hidden Layers: Three fully connected layers, each of the same size, allowing the model to learn complex patterns.

    Output Layer: Produces the final predictions.

2. Activation Function:

    LeakyReLU: Applied after each layer, it introduces non-linearity while allowing small gradients for negative values to avoid "dead neurons."

3. Regularization:

    Dropout (50%): Applied after each hidden layer, randomly deactivating 50% of neurons to prevent overfitting and improve generalization.

4. Flow (Forward Pass):

    Data moves through the network in the order: Input → Hidden Layers → Output, with activation and dropout applied after each hidden layer.

5. Why this Design?

    Multiple Layers: To capture complex relationships in the data.

    LeakyReLU: Helps keep neurons "active" during training.

    Dropout: Reduces overfitting, making the model more robust.

5. Performance metrics and Graphs overview:

- Performance Metrics:

  Test Loss: 0.5323

  Test Accuracy: 0.7533

Test Precision: 0.7833

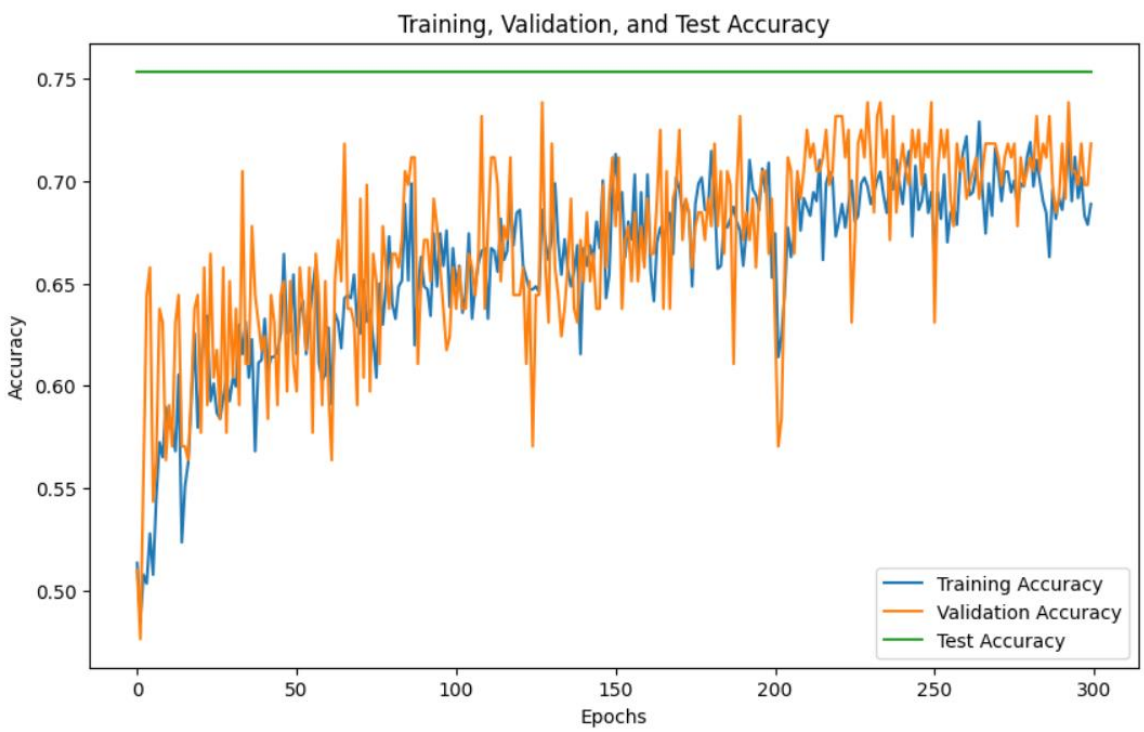
Test Recall: 0.6620

Test F1 Score: 0.7176

- The Test Precision of 0.7833 suggests that when the model predicts a positive class, it is correct 78.33% of the time.
- The Test Recall of 0.6620 shows that the model successfully identifies 66.20% of all actual positive instances.
- The F1 Score of 0.7176 balances both precision and recall, indicating a good balance between the two metrics.

- Graphs Overview:

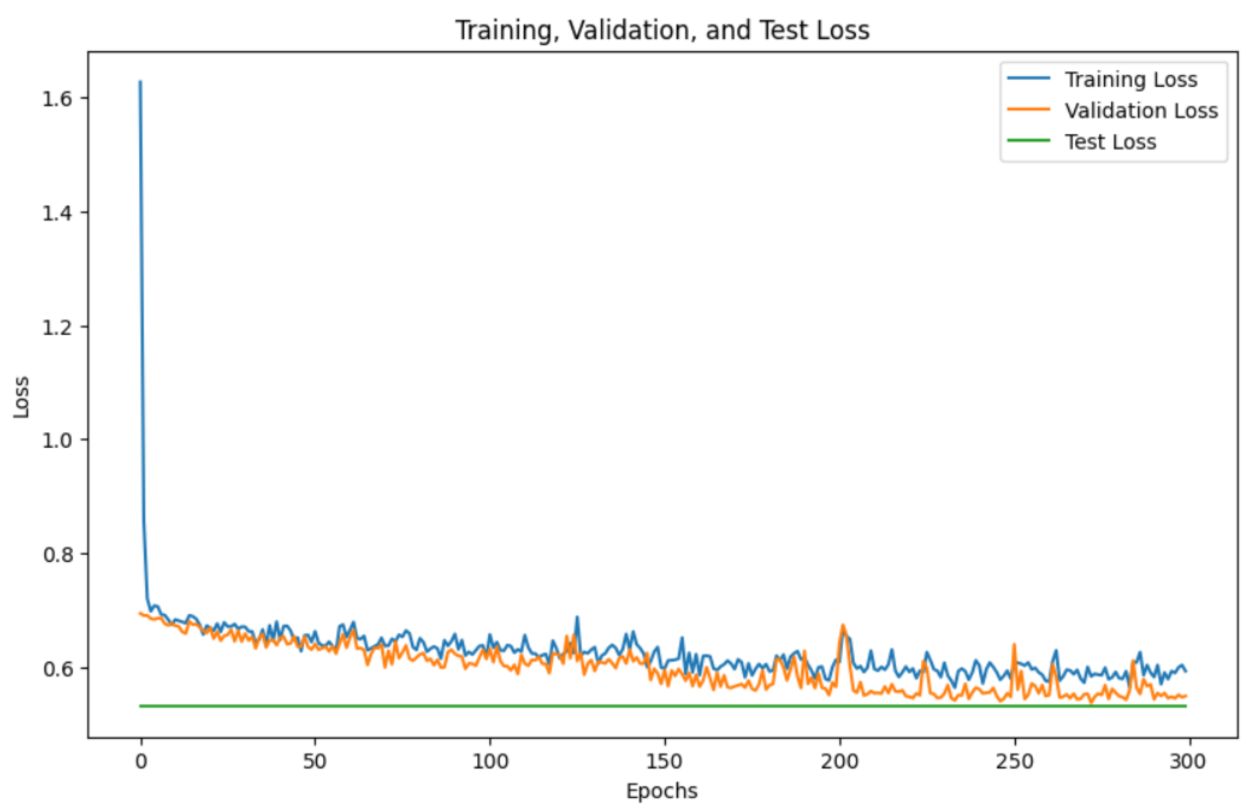
  Accuracy Graph:



  - The graph shows the Training Accuracy (blue), Validation Accuracy (orange), and Test Accuracy (green line) over 300 epochs.
  - Both training and validation accuracy exhibit fluctuations, but overall show improvement, stabilizing around 65-70% accuracy.

- The Test Accuracy line is stable at around 75.33%, indicating that the model generalizes well on unseen data.
- The model reaches a Test Accuracy of 75.33%, which suggests a decent fit to the data.
- The Training and Validation Accuracy follow similar trends, indicating that the model does not suffer from severe overfitting or underfitting. Although the validation accuracy shows some fluctuations, it remains closely aligned with the training accuracy.
- This alignment between training and validation performance suggests that the model is learning effectively without much overfitting.

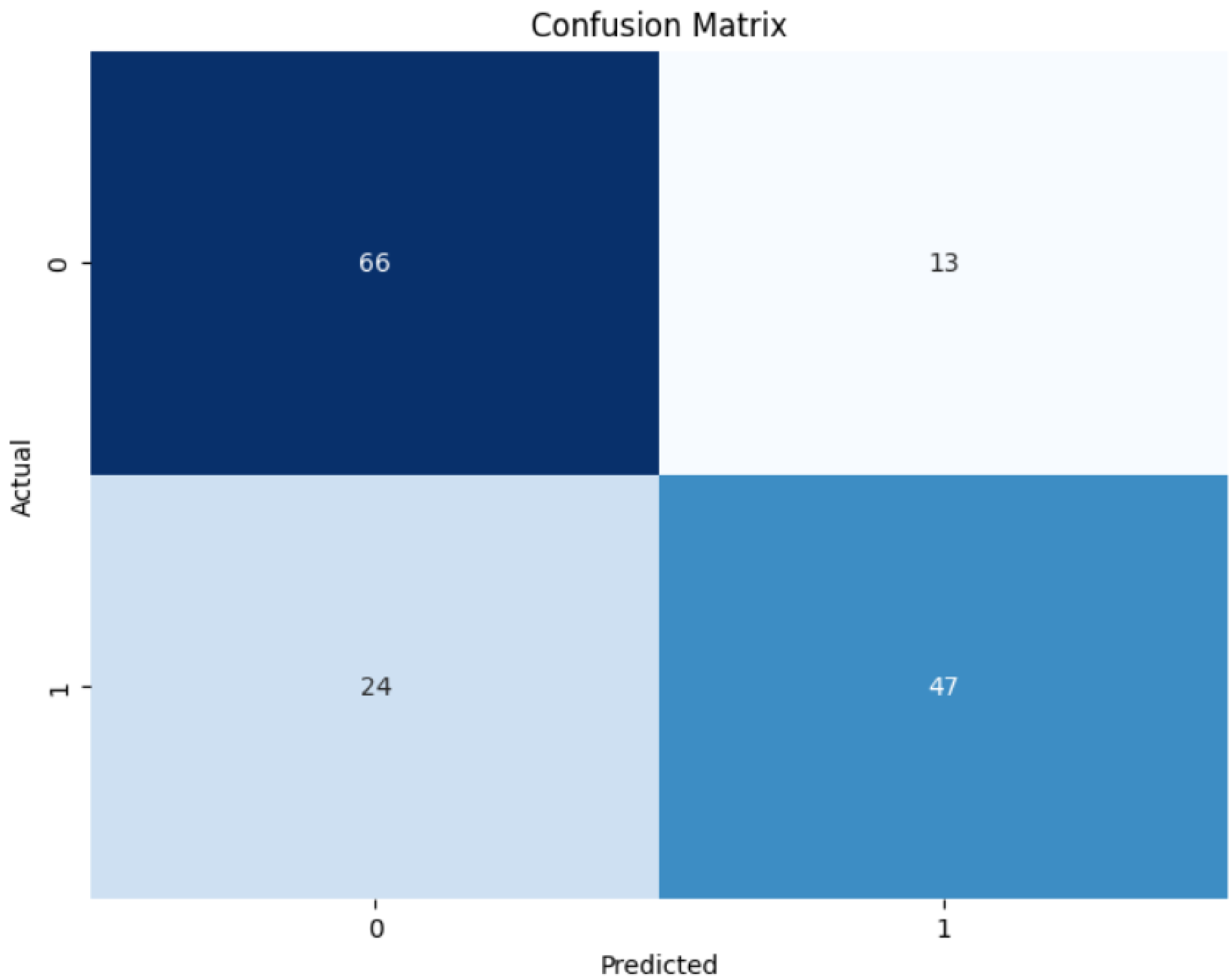
Loss Graph:



- The graph shows Training Loss (blue), Validation Loss (orange), and Test Loss (green line).

- The loss initially drops steeply during early epochs, suggesting rapid learning. It then flattens and stabilizes, with both training and validation loss converging around 0.6.
- The Test Loss is stable at 0.5323, indicating good generalization and model stability.
- The loss stabilizes for both the training and validation sets after the initial steep decline. This indicates that the model is converging and not learning much additional information from the data as the epochs increase.
- A Test Loss of 0.5323 further confirms that the model is not overfitting and is performing well on unseen data.

Confusion Matrix:



The confusion matrix shows the performance in terms of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN):

- True Negatives (TN): 66

  The model correctly predicted 66 instances as negative (class 0).
- False Positives (FP): 13

  The model incorrectly predicted 13 instances as positive (class 1) when they were actually negative (class 0).
- False Negatives (FN): 24

  The model incorrectly predicted 24 instances as negative (class 0) when they were actually positive (class 1).
- True Positives (TP): 47

  The model correctly predicted 47 instances as positive (class 1).

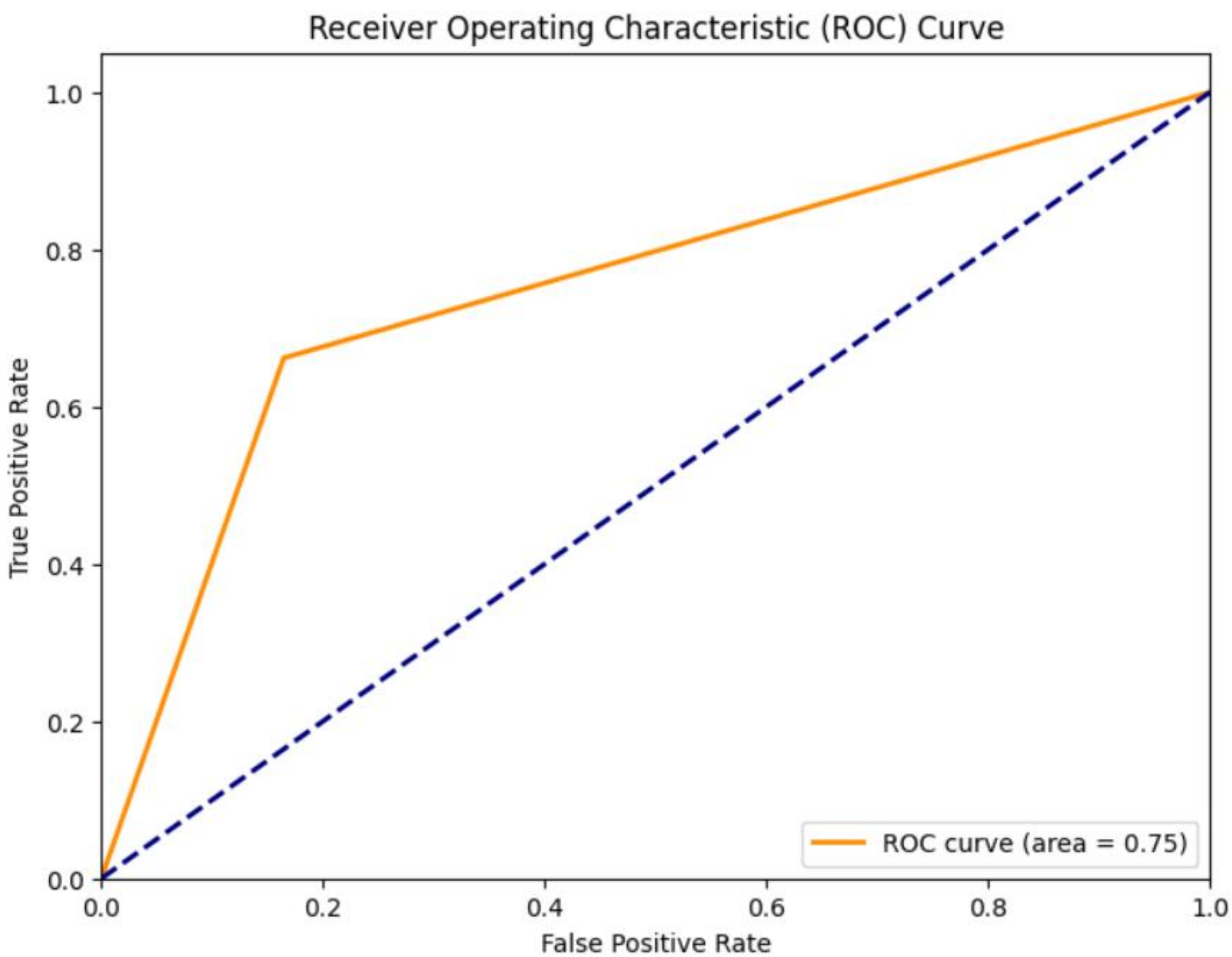
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = 0.762$$

$$precision = \frac{TP}{TP + FP} = 0.783$$

$$recall = \frac{TP}{TP + FN} = 0.662$$

$$F1\ score = 2 \times \frac{precision \times recall}{precision + recall} = 0.717$$

ROC Curve:



The ROC curve depicts the trade-off between the True Positive Rate (Recall) and the False Positive Rate (FPR), and the Area Under the Curve (AUC) is an important metric to assess model performance.

- AUC (Area Under the Curve): 0.75
- An AUC of 0.75 indicates that the model has a reasonably good ability to distinguish between positive and negative classes, though there's room for improvement. A score of 1.0 would represent perfect classification, and 0.5 would mean the model is performing no better than random guessing.

Part 2:

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

Dropout Value Tuning

	Dropout Value	Test Accuracy
Setup #1	0.3	71.33%
Setup #2	0.5	75.33%
Setup #3	0.7	65.33%

A moderate dropout of 0.5 achieves the best performance. As dropout increases beyond that (0.7), the model's performance declines, likely due to excessive randomness being introduced. This implies that a higher dropout prevents the model from learning meaningful patterns effectively.

Batch Size

	Value	Test Accuracy
Setup #1	32	72.67%
Setup #2	64	76%

Setup #3	128	76%

A larger batch size (64 or 128) offers slightly better performance compared to a smaller batch size (32). Larger batches may help the model stabilize learning with better gradient estimates, leading to higher accuracy.

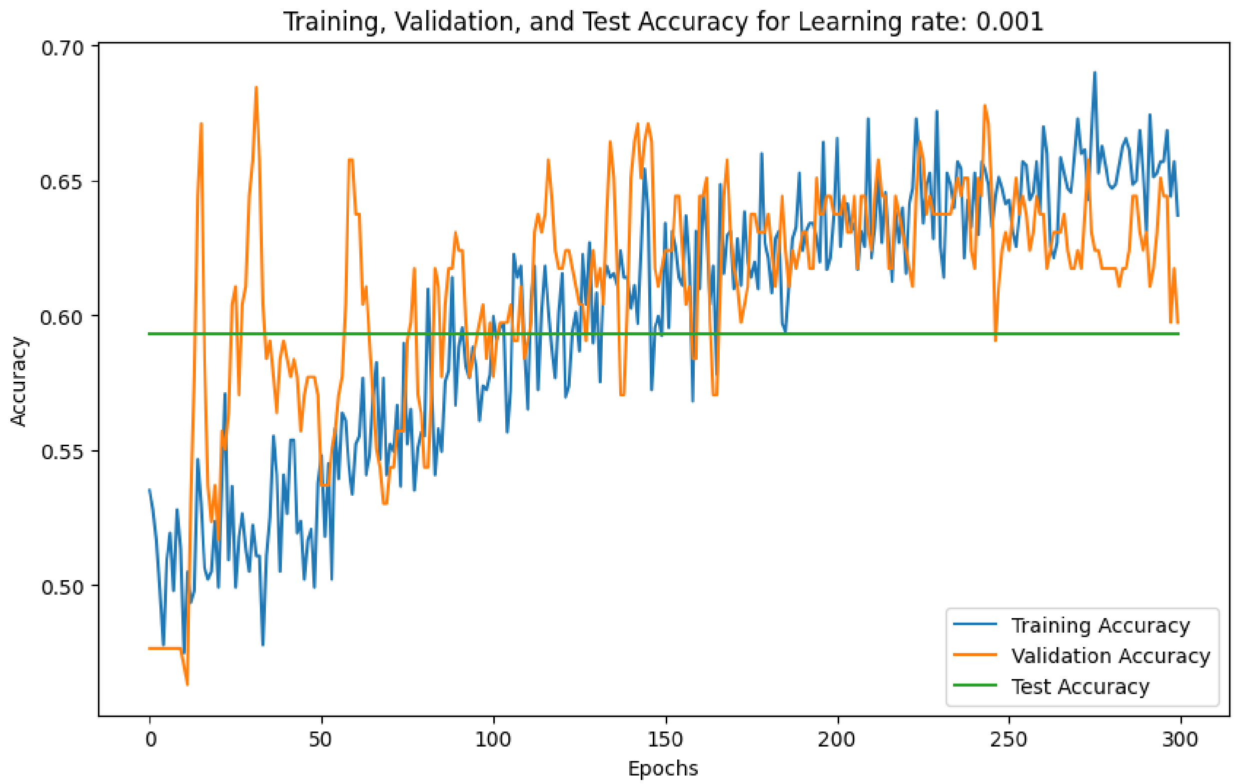
Learning Rate

	Value	Test Accuracy
Setup #1	0.001	59.33%
Setup #2	0.007	69.33%
Setup #3	0.01	72.67%

A higher learning rate, such as 0.01, leads to the best performance. A learning rate (0.001) too low results in slower convergence and underperformance. However, the learning rate must be carefully chosen as higher rates could lead to unstable training beyond certain limits.

2. Provide 2-3 graphs (e.g. accuracy/loss) for setups with interesting observations, e.g. those that show least or most improvements.

   Worst accuracy graph:

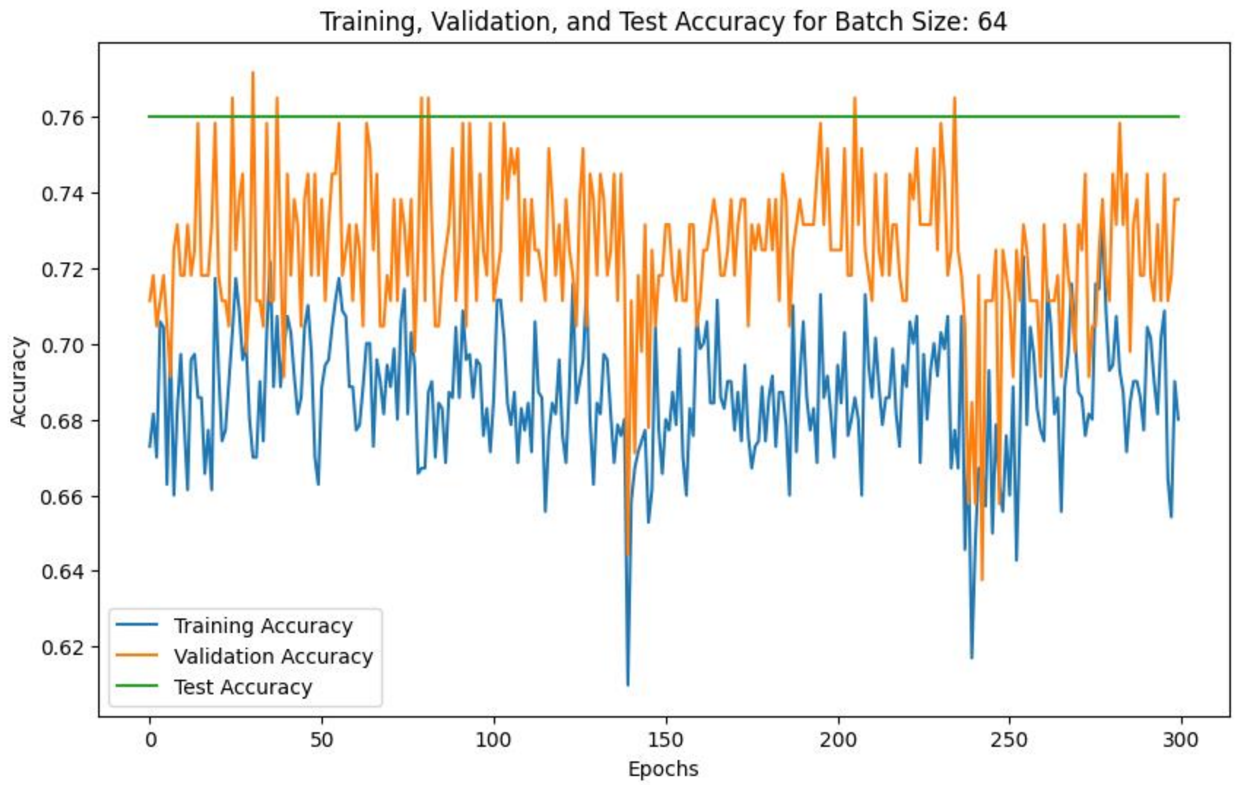


The plot shows the training, validation, and test accuracies over 300 epochs for a learning rate of 0.001.

Training-Validation Relationship: While the training and validation accuracies track each other closely after 100 epochs, the large oscillations in validation accuracy, especially early on, indicate that the model struggles with consistent generalization. This is due to the low learning rate causing slower convergence.

Overfitting Risk: The increasing gap between training accuracy and test accuracy suggests potential overfitting—where the model learns to perform well on the training data but fails to generalize to the test set. This is why the test accuracy is very low. This indicates that the learning rate is very less and the model isnt genaralizing the data well. Therefore we further increase the learning rate for further models.

Best accuracy graph:



Training and Validation Variability: The training and validation accuracies are not consistently improving and exhibit significant fluctuations, indicating that the model struggles to stabilize during training. This could be due to the batch size or a suboptimal learning rate that prevents smooth learning.

It is seen that the test accuracy is very high for this case, this is because of the batch size combined with an optimal learning rate and dropout rate. This combination is prodiucing the best results. The batch size of 64 is moderate also. In this case, we have chosen a learning rate of 0.007 which is found to be optimal. Therefore, with an optimal learning rate, the model converges properly.

3. Discuss all the methods you used that help to improve the accuracy or the training time.

1. Early Stopping:

Early stopping is meant to prevent overfitting by stopping training once the model's performance on the validation set starts to degrade. However, in this case, early stopping is triggered prematurely before the model has fully learned the patterns in the training data, leading to underfitting. Therefore the training of this model stops very early and does not improve accuracy beyond 60 percent. Thereofre we dont choose thismodel as the best one.

2. Learning Rate Scheduler:

A learning rate scheduler adjusts the learning rate during training, typically reducing it when the performance reduces. However, since the learning rate is reduced too quickly or too drastically, it is probably slowing down learning, causing the model to get stuck in a suboptimal solution. In this case, the scheduler might be reducing the learning rate too soon, preventing the model from making further progress, which explains the drop in accuracy.

3. Batch Normalization:

Batch normalization normalizes the inputs to each layer of the model, which stabilizes the learning process. It allows for faster training and enables the use of higher learning rates without risking instability. In this case, this method improves accuracy because it helps in keeping the network weights more stable during training. It also acts as a form of regularization, reducing overfitting and ensuring better generalization on the test data. This explains why it increases the accuracy in your model.

4. Gradient Accumulation:

Gradient accumulation is useful for handling smaller batch sizes by accumulating gradients over multiple steps before updating weights. However, in some cases, if the gradient updates are too delayed or if the model benefits from more frequent updates, gradient accumulation could harm performance. This explains why it decreases the accuracy in this situation. The model might benefit more from frequent, smaller updates, rather than accumulating gradients over several batches.

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

1. Model Architecture:

The model is defined as a class NN_batchnorm that inherits from nn.Module. The architecture includes:

I. Input Layer (self.fc1): Takes in the input features (number of features equals the shape of X_train_tensor).

II. Hidden Layers:

    a. The network consists of three fully connected hidden layers (fc1, fc2, and fc3), each followed by batch normalization and dropout.

    b. Batch Normalization (self.bn1, self.bn2, self.bn3): Applied after each hidden layer to normalize the activations and stabilize learning.

    c. Leaky ReLU Activation (self.relu): A non-linear activation function applied after each batch normalization layer to introduce non-linearity and help the network learn complex patterns.

    d. Dropout (self.dropout): A dropout rate of 30% is used to prevent overfitting by randomly setting a fraction of input units to zero during training, ensuring the model generalizes better.

III. Output Layer (self.fc4): A fully connected output layer that produces a single output. Since this is a binary classification problem, this output will later be passed through a sigmoid function (via torch.sigmoid during training) to make the result between 0 and 1, representing the probability of the positive class.

2. Forward Method:

The forward pass defines how data flows through the network:

I. Inputs are passed through the first layer (fc1), followed by batch normalization (bn1), Leaky ReLU, and dropout.

II. This is repeated for the second and third layers (fc2 and fc3), applying the same combination of operations (batch normalization, activation, and dropout).

III. The final layer (fc4) outputs a single value, which is the raw score for the binary classification (before applying a sigmoid function to get the probability).

3. Training Process:

The training process involves optimizing the model using binary cross-entropy loss with logits (i.e., BCEWithLogitsLoss) and the Adam optimizer.

Hyperparameters:

I. **Input Size**: Matches the number of features in the training data.
II. **Hidden Size**: Set to 64 units for each hidden layer.
III. **Output Size**: 1 (for binary classification).
IV. **Batch Size**: 16.
V. **Learning Rate**: 0.007 (this can be adjusted via learning rate schedulers).
VI. **Total Epochs**: 300.

Data Loaders:

DataLoader objects are used to create mini-batches for training (train_dataloader), validation (val_dataloader), and testing (test_dataloader).

4. Evaluation Metrics:

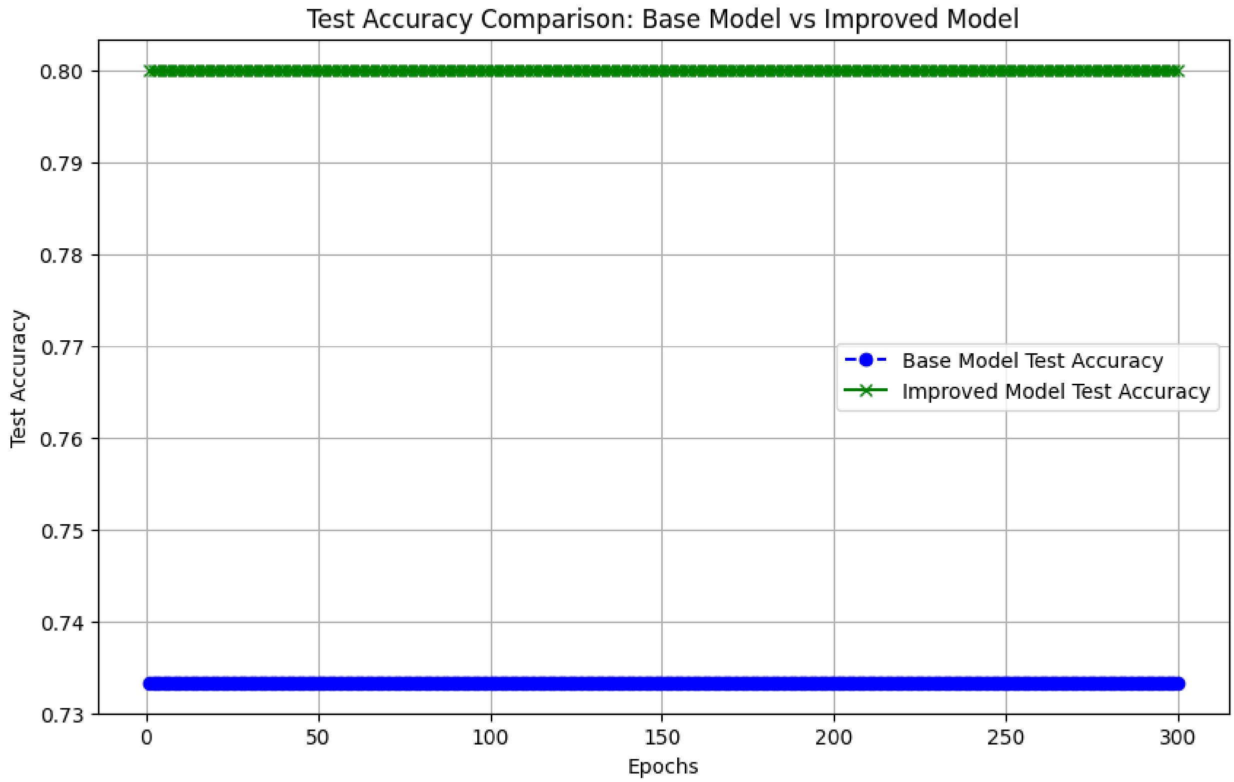
I. **Accuracy**: Proportion of correct predictions.
II. **Precision**: Measures how many of the positively predicted samples are actually positive.
III. **Recall**: Measures how many of the actual positive samples were correctly predicted.
IV. **F1 Score**: Harmonic mean of precision and recall, used to evaluate overall model performance when there's an imbalance between precision and recall.

5. Performance:

Test Loss: 0.4747, Test Accuracy: 0.8000, Learning Rate: 0.007, Test Accuracy: 0.8000, Test Precision: 0.7531, Test Recall: 0.8592, Test F1 Score: 0.8026

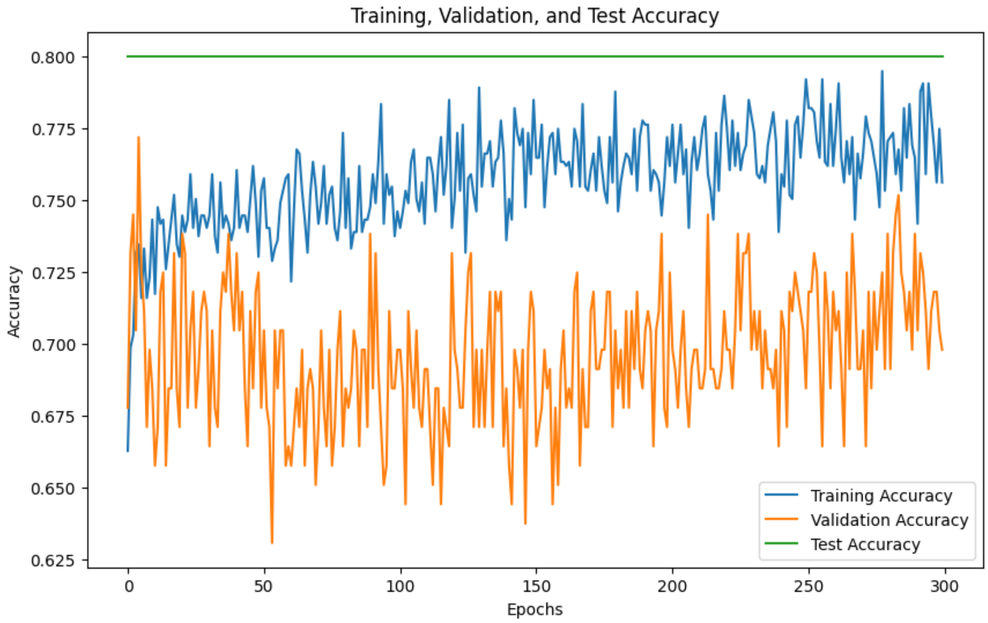
This model gives us the highest accuracy (80%). This is the model that uses a dropout rate of 0.3, batch size of 16 and learning rate of 0.007. These are the optimal parameters for this model. We see this accuracy as the highest and analyze the following graphs:

Test Accuracy Comparison:



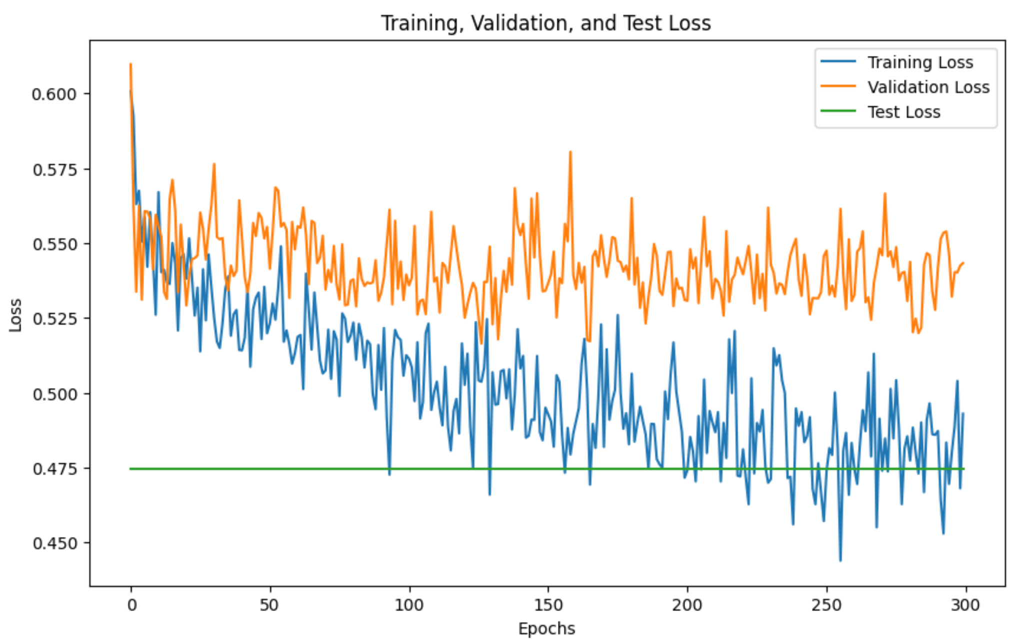
The improved model performs significantly better than the base model, with about a **7% increase** in test accuracy (from 0.73 to 0.80). This is due to the addition of batch normalization used in the best model. We have also used a couple of optimal hyperparameters here.

Training, Validation and testing accuracy for the best model:



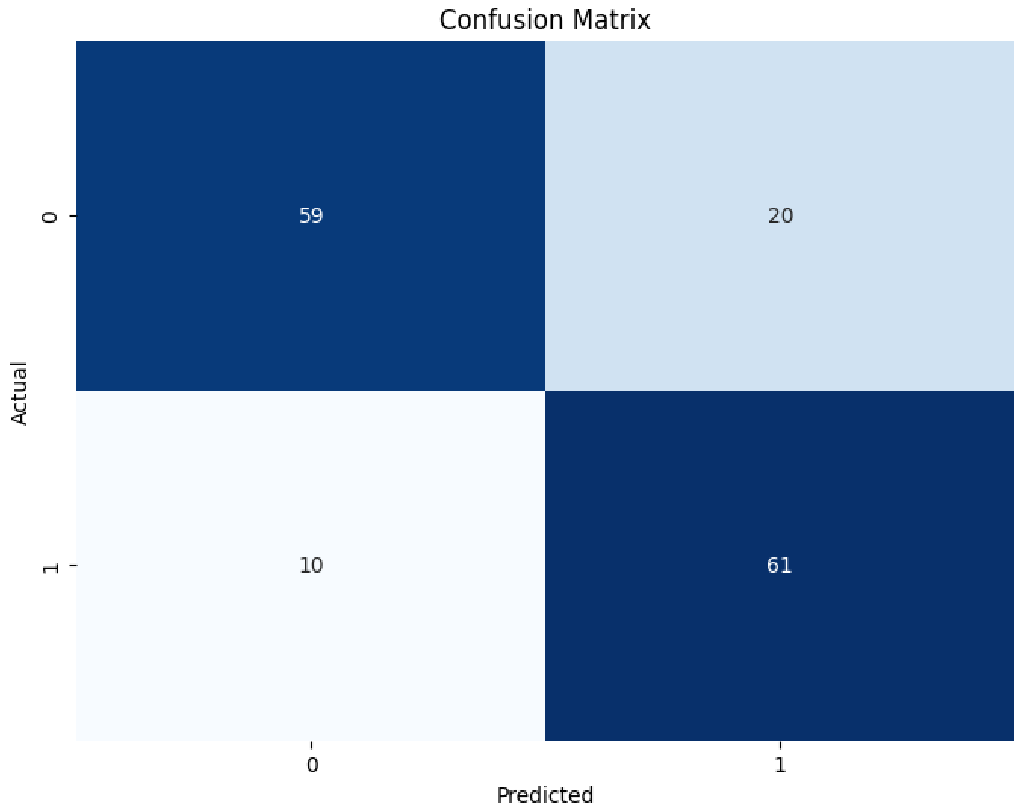
The gap between the training and validation accuracy suggests overfitting. The training accuracy increases, while validation accuracy fluctuates more wildly. The regular application of techniques like dropout and batch normalization likely helped to maintain stability in test accuracy, but validation accuracy indicates room for more regularization.

Training, validation and testing loss for the best model:



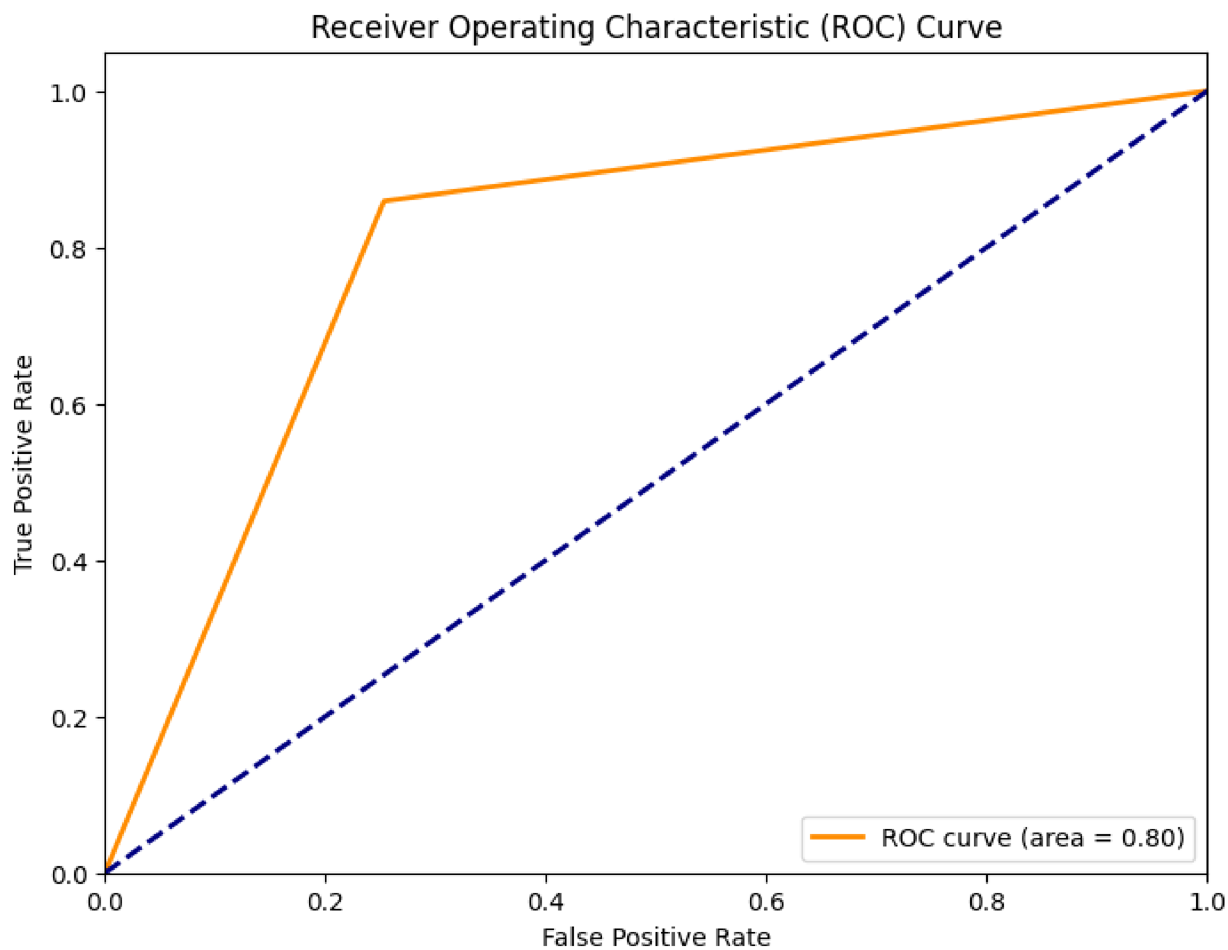
The divergence between the training and validation loss suggests overfitting. The model performs well on the training data, but its performance on validation data varies greatly, indicating that it might be memorizing the training data instead of generalizing to new data.

Confusion matrix:



I. True Negatives: The model correctly predicted class 0 for 59 instances.
II. False Positives: The model incorrectly predicted class 1 for 20 instances that were actually class 0.
III. False Negatives: The model incorrectly predicted class 0 for 10 instances that were actually class 1.
IV. True Positives: The model correctly predicted class 1 for 61 instances.

ROC Curve:



The orange curve represents the performance of your classifier. The higher it is from the diagonal, the better the model. A perfect classifier would have a point at the top left (TPR = 1, FPR = 0). The AUC value of 0.80 indicates that the model has a good balance between the true positive rate and the false positive rate. It suggests that 80% of the time, the model is able to distinguish between positive and negative instances.

Part 3

1. Overview of the dataset:

Type: Images

Training set size: 80640

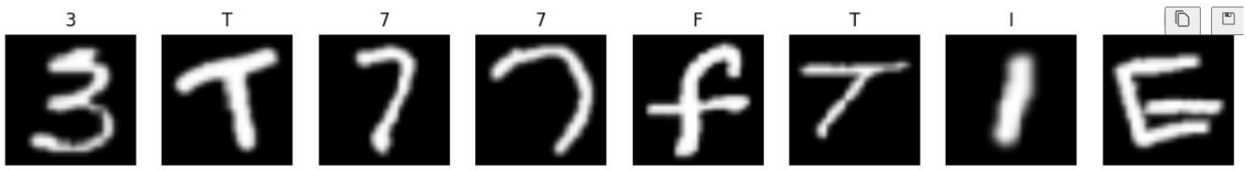
Validation set size: 10080

Test set size: 10080

Number of images: 100800

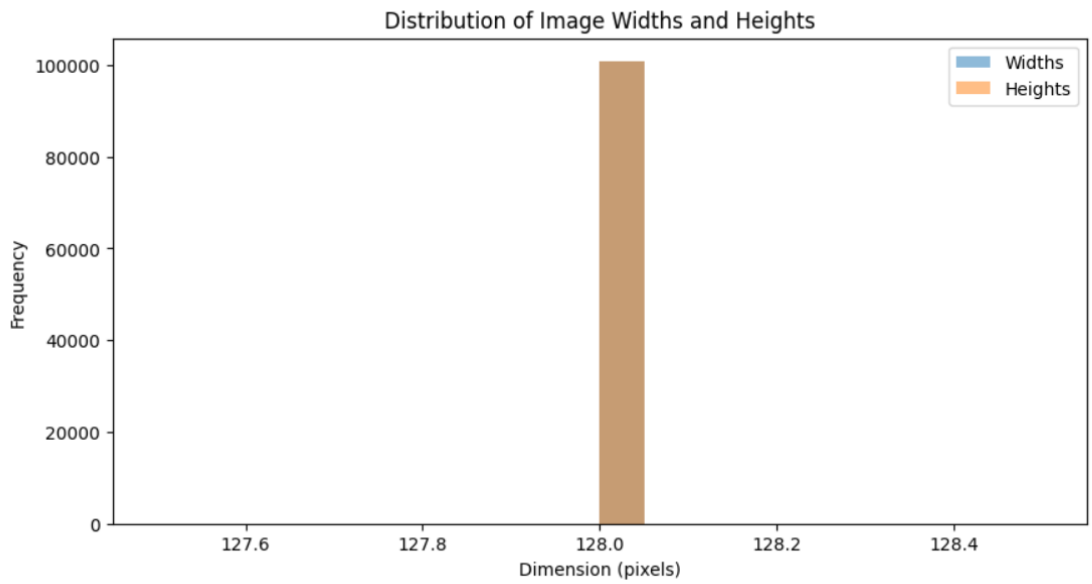
Number of classes: 36

Sample images from dataset:

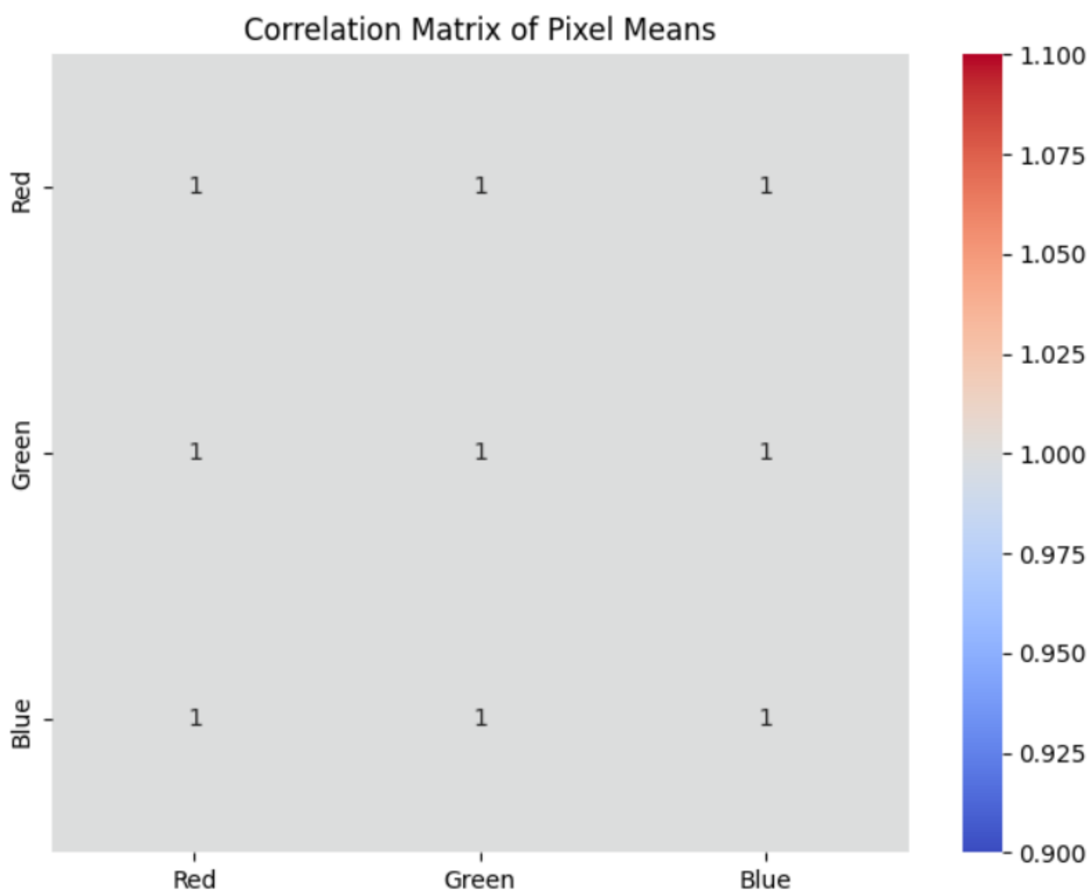


2. Graphs:

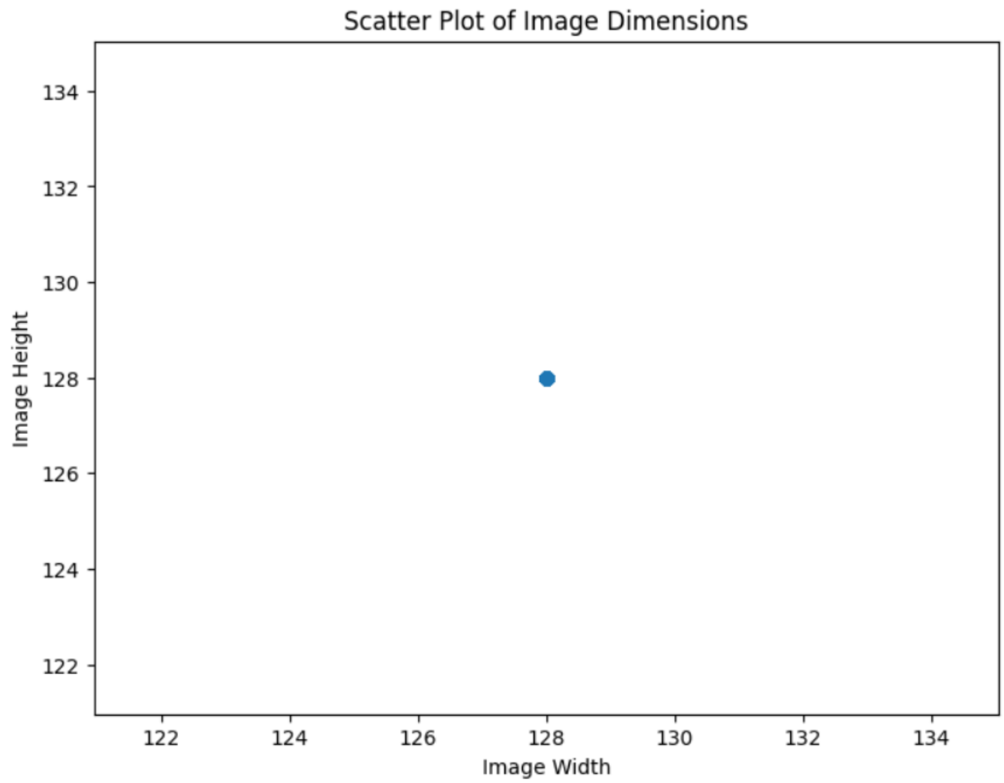
There are 2800 images in each class and there are 36 classes.



All images have a fixed dimension of 128x128 pixels as both width and height distributions are concentrated at this value.



Correlation matrix shows a perfect correlation (1) between the RGB channels indicating uniform color distribution across channels.



All images have dimensions of 128x128.

3. Summary of the best final model:

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      [32, 36]                  --
├─Conv2d: 1-1                            [32, 32, 128, 128]        896
├─MaxPool2d: 1-2                         [32, 32, 64, 64]          --
├─Conv2d: 1-3                            [32, 64, 64, 64]          18,496
├─MaxPool2d: 1-4                         [32, 64, 32, 32]          --
├─Conv2d: 1-5                            [32, 128, 32, 32]         73,856
├─MaxPool2d: 1-6                         [32, 128, 16, 16]         --
├─Conv2d: 1-7                            [32, 256, 16, 16]         295,168
├─MaxPool2d: 1-8                         [32, 256, 8, 8]           --
├─Linear: 1-9                            [32, 512]                 8,389,120
├─Dropout: 1-10                          [32, 512]                 --
├─Linear: 1-11                           [32, 36]                  18,468
==========================================================================================
Total params: 8,796,004
Trainable params: 8,796,004
Non-trainable params: 0
Total mult-adds (G): 8.00
==========================================================================================
Input size (MB): 6.29
Forward/backward pass size (MB): 251.80
Params size (MB): 35.18
Estimated Total Size (MB): 293.27
==========================================================================================
```

The final CNN model has a layered architecture consisting of convolutional, pooling, and fully connected layers optimized for classification. Here's a summary of the architecture:

Convolutional Layers:

a. Three sets of Conv2D layers followed by MaxPooling layers to progressively capture spatial hierarchies and reduce dimensionality.
b. The filters start at 128 and increase with each layer to capture more complex features, ending with 256 filters in the last Conv2D layer.

Pooling Layers:

c. MaxPooling layers after each Conv2D layer help in reducing spatial dimensions, ensuring that the model is computationally efficient and focused on the most significant features.

Fully Connected Layers:

d. A fully connected (Linear) layer with 512 units to learn high-level representations, followed by a Dropout layer to reduce overfitting.
e. The final Linear layer has 36 output units, representing the class predictions.

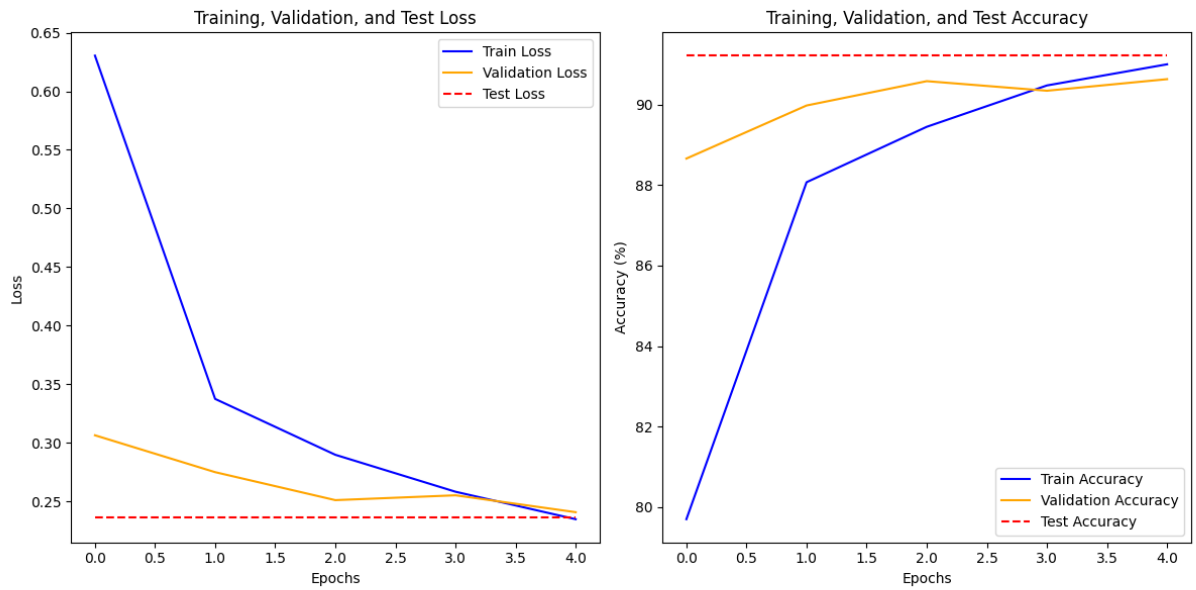
Model Parameters:

f. Total parameters: 8.8 million, with all being trainable.
g. Estimated total model size: 293.27 MB.

This CNN architecture is balanced with enough depth and feature extraction layers to achieve high accuracy while maintaining reasonable computational efficiency, suitable for classifying complex image data.

4. Performance metrics and graphs:

Best model is CNN stored in variable model which has the highest accuracy of 91.22%. We will be plotting graphs for the best model and storing the weights.



The plots show the training, validation, and test performance over five epochs:

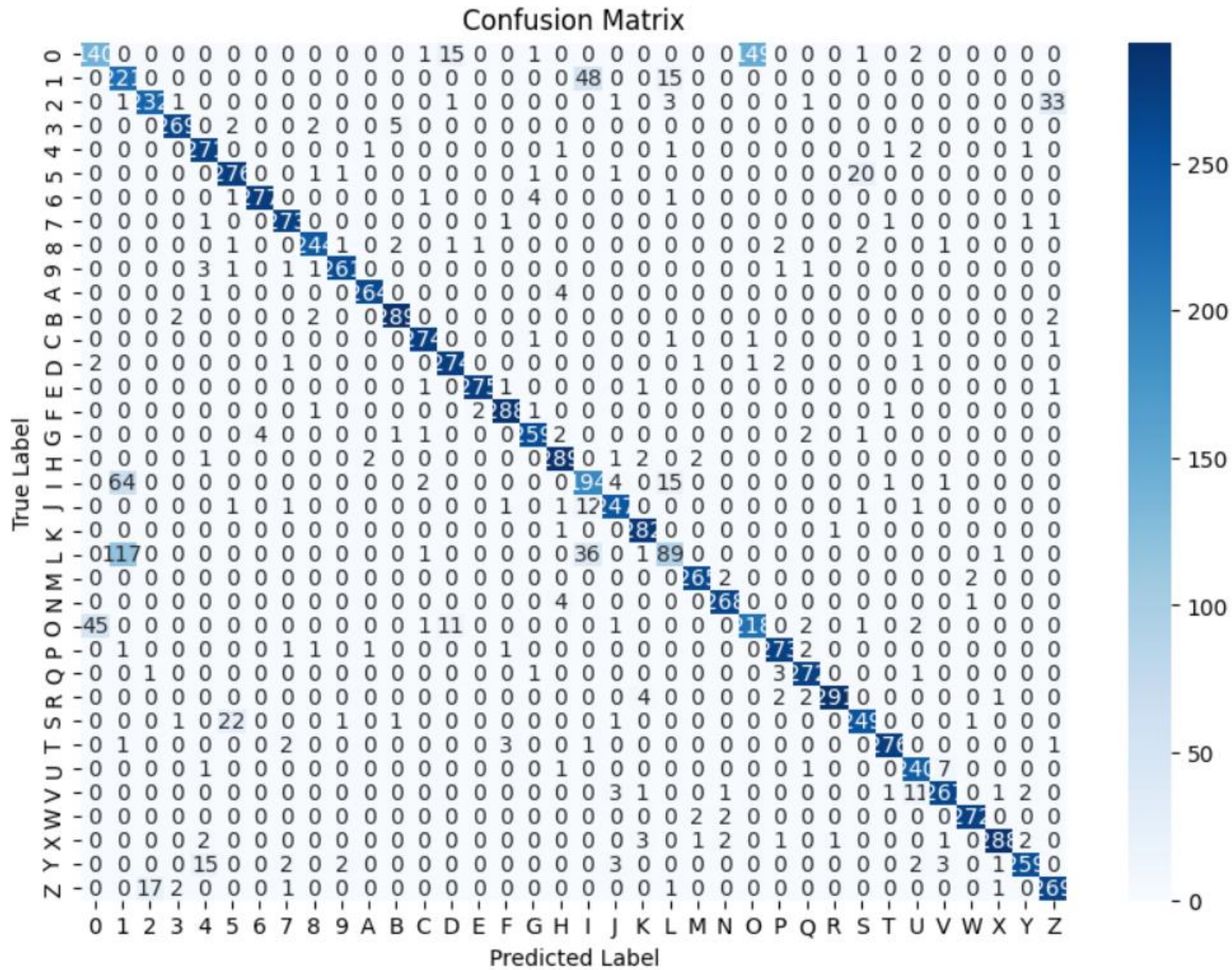
Loss Graph:

- The training loss (blue line) decreases steadily, indicating that the model is learning effectively from the training data.
- The validation loss (orange line) also decreases, suggesting that the model generalizes well to unseen data.
- The test loss (red dashed line) is consistently low and stable, showing good generalization on the test set.

Accuracy Graph:

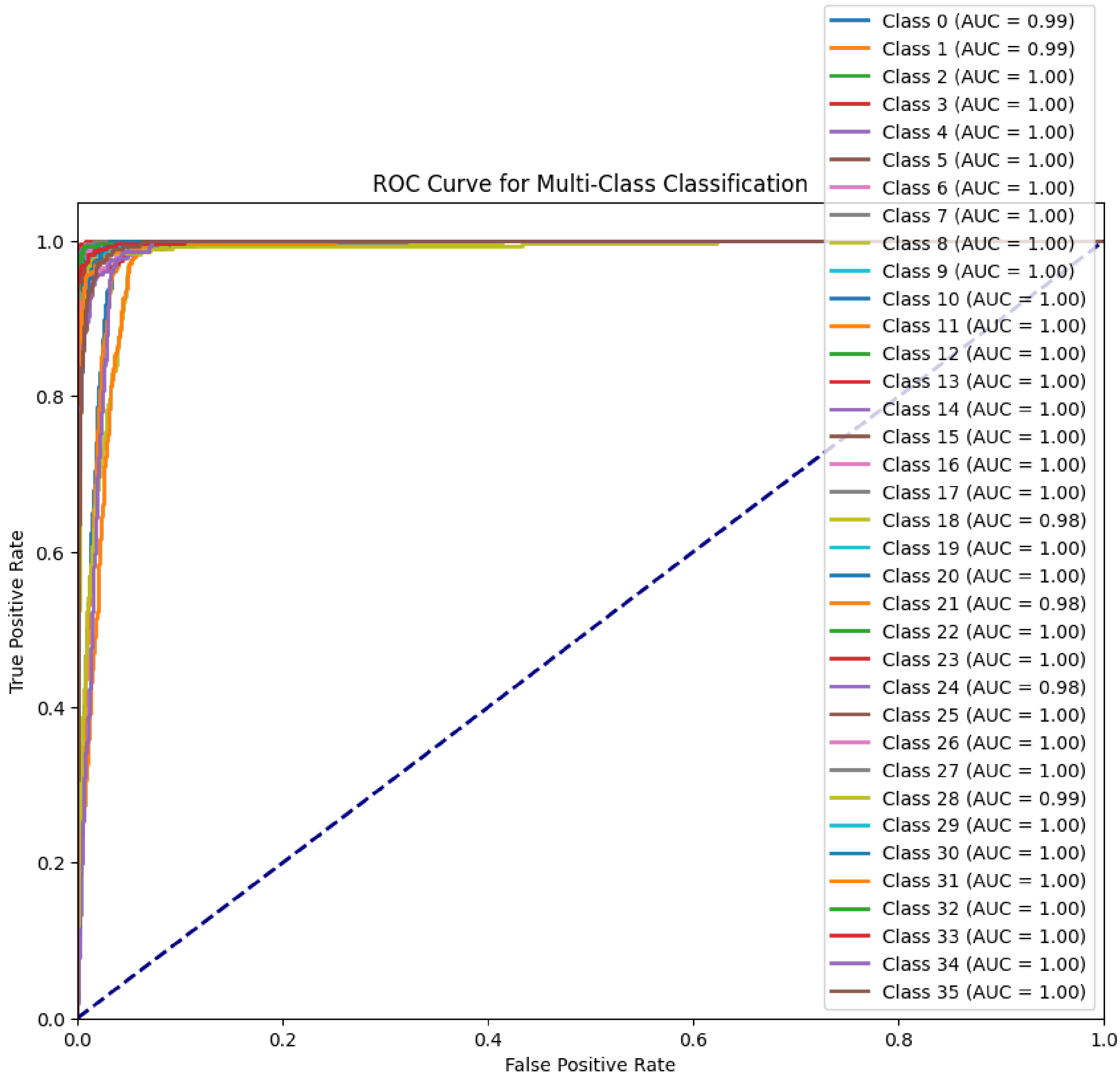
- Training accuracy (blue line) increases significantly with each epoch, showing improved learning.
- Validation accuracy (orange line) also improves and stabilizes, following a similar trend to training accuracy.
- Test accuracy (red dashed line) remains high, indicating that the model performs well on new data.

Overall, the model is effective, showing no significant overfitting or underfitting across these metrics.



This confusion matrix shows the model's performance on a multi-class classification task, likely classifying characters or digits (0-9, A-Z). Each row represents the true label, while each column shows the predicted label. The diagonal cells (highlighted with darker blue) show correct predictions for each class, with higher values indicating better accuracy. Off-diagonal values represent misclassifications.

- Most values are concentrated along the diagonal, indicating that the model correctly classified a majority of the samples for each class.
- Some classes have small misclassifications in neighboring columns, which could indicate similarities between these classes that cause confusion.
- Overall, the model seems to have high accuracy, with few errors for most classes.



- ROC Curve: Each line represents the ROC curve for a specific class, showing the trade-off between the True Positive Rate (sensitivity) and the False Positive Rate for that class. Ideally, a better-performing model will have a curve that hugs the top-left corner, indicating high sensitivity and low false positive rates.
- AUC (Area Under the Curve): The legend on the right displays the AUC value for each class. The AUC is a summary metric that gives an overall sense of how well the model distinguishes between a specific class and the rest. An AUC of 1.0 (or close to it) suggests perfect or near-perfect separation, while lower values indicate less effective separation.
- High AUC values: In this chart, most classes have AUC values close to 1.0, which indicates that the model is performing very well for each class. Some classes, like Class

0, Class 1, and Class 28, have AUC values of 0.99, which is still very good but slightly lower than 1.0.

- Diagonal Line: The dashed diagonal line represents the "no-skill" line, where the true positive rate equals the false positive rate (AUC = 0.5). A model that performs better than random chance will have curves that are above this line.

Part 4

1) Provide the model details of your CNN.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
        MaxPool2d-15          [-1, 256, 28, 28]               0
           Conv2d-16          [-1, 512, 28, 28]       1,180,160
             ReLU-17          [-1, 512, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       2,359,808
             ReLU-19          [-1, 512, 28, 28]               0
        MaxPool2d-20          [-1, 512, 14, 14]               0
           Conv2d-21          [-1, 512, 14, 14]       2,359,808
             ReLU-22          [-1, 512, 14, 14]               0
           Conv2d-23          [-1, 512, 14, 14]       2,359,808
             ReLU-24          [-1, 512, 14, 14]               0
        MaxPool2d-25            [-1, 512, 7, 7]               0
           Linear-26                 [-1, 4096]     102,764,544
             ReLU-27                 [-1, 4096]               0
          Dropout-28                 [-1, 4096]               0
           Linear-29                 [-1, 4096]      16,781,312
             ReLU-30                 [-1, 4096]               0
          Dropout-31                 [-1, 4096]               0
           Linear-32                   [-1, 36]         147,492
================================================================
Total params: 129,098,340
Trainable params: 129,098,340
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 198.68
Params size (MB): 492.47
Estimated Total Size (MB): 691.72
----------------------------------------------------------------
```

The model consists of several Conv2d layers for feature extraction, ReLU layers for activation, MaxPool2d layers for down-sampling, and Linear (fully connected) layers at the end for classification. The convolutional blocks increase the channel count (e.g., from 64 to 512) while spatial dimensions reduce through pooling (e.g., from 224x224 to 7x7). The final fully connected layers transition from a flattened 4096-dimensional output to 36, representing the class output. The model has a total of 129,098,340 trainable parameters. Memory requirements include an estimated 0.57 MB for input size, 198.68 MB for forward/backward pass activations, and 492.47 MB for parameters, with a total estimated memory usage of 691.72 MB.

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

Depth and Structure:

VGG13 is much deeper than the other two models. It features 13 layers (convolutional and fully connected) with five convolutional blocks, each containing two Conv2d layers followed by ReLU activations and MaxPooling. This deeper structure allows it to capture more complex features but requires more computational power and memory.

CNN and BatchNormalizationCNN are shallower, with only four convolutional layers, each followed by ReLU activation and MaxPooling. This simpler structure makes them computationally lighter but may limit their feature extraction capability for complex data.

Convolutional Channels:

VGG13 uses a higher number of filters in each block, increasing from 64 to 512 channels. This gradual increase in the number of channels enables it to capture fine-grained details at higher depths.

CNN and BatchNormalizationCNN have fewer channels, starting from 32 and maxing out at 256. This reduces the model's ability to capture fine-grained features compared to VGG13.

Batch Normalization:

VGG13 does not use batch normalization layers, which can lead to slower convergence and may make it more sensitive to the initial learning rate and weight initialization.

BatchNormalizationCNN includes batch normalization after each convolutional layer, which helps stabilize training, improve convergence speed, and may improve generalization. The original CNN model does not use batch normalization, which might make it more prone to internal covariate shift during training.

Fully Connected Layers:

VGG13 has three fully connected layers in the classifier block, with two large hidden layers of 4096 units each, followed by the output layer. This large classifier block adds significant parameters and enables more complex decision boundaries.

CNN and BatchNormalizationCNN use a simpler classifier with only two fully connected layers, one with 512 units and the output layer. This makes these models more lightweight but potentially less capable of complex classification tasks.

Dropout Regularization:

All models use dropout, but VGG13 uses it in both fully connected layers with the default dropout rate (likely 0.5). CNN and BatchNormalizationCNN apply dropout only after the first fully connected layer, reducing the likelihood of overfitting.

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

Training time:

The training time for the VGG-13 model was significantly longer compared to the previous CNN model. The VGG-13 model took 73 minutes and 22 seconds, which is equivalent to 4414 seconds, whereas the previous CNN model took 663.34 seconds. This results in a time difference of 3750.66 seconds, meaning the VGG-13 model required approximately 5.65 times more training time than the previous model, reflecting a 565.53% increase in training duration. This substantial increase is likely due to the VGG-13 model's deeper and more complex architecture, which demands more computational resources.

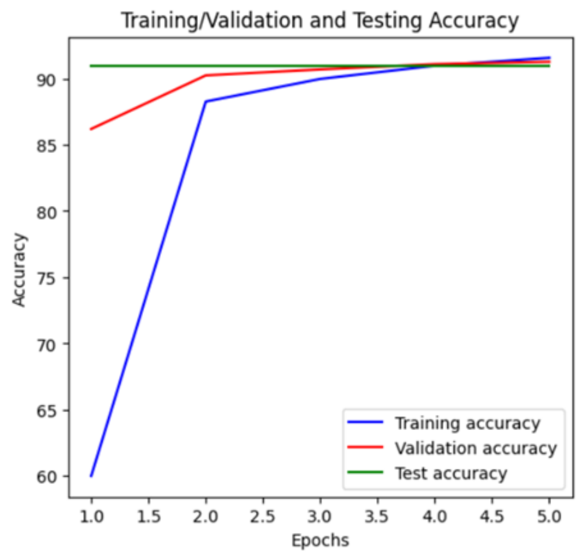
Accuracy:

In terms of accuracy, the previous CNN model performed slightly better than the VGG-13 model. The previous CNN model achieved an accuracy of 91.57%, whereas the VGG-13 model reached 90.91%. This shows a marginal difference of 0.66% in favor of the previous CNN model. Despite the VGG-13 model's deeper architecture and longer training time, it did not outperform the simpler CNN model in accuracy, suggesting that the additional layers in VGG-13 did not yield a significant improvement for this specific dataset and task.

Precision, recall and f1 score:

When comparing precision, recall, and F1 score, the previous CNN model shows a slight edge over the VGG-13 model. The VGG-13 model achieved a precision, recall, and F1 score of 0.91 across all metrics. In contrast, the previous CNN model had a precision of 0.9165, a recall of 0.9157, and an F1 score of 0.9157. These differences, though small, indicate that the previous CNN model was marginally more effective in correctly identifying relevant instances (precision), capturing all relevant instances (recall), and balancing these two aspects (F1 score). Despite the more complex architecture and increased training time of the VGG-13 model, the simpler CNN model maintained a slight advantage in overall classification performance for this dataset.

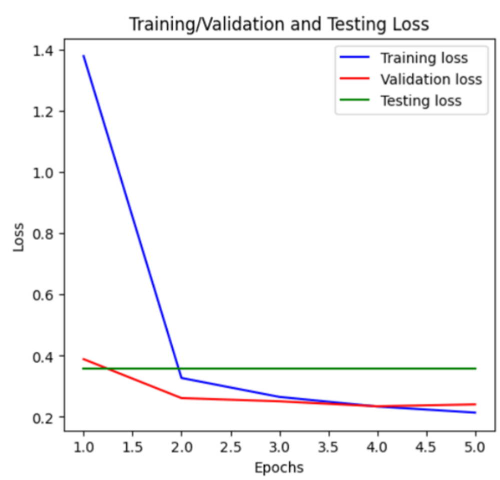
Accuracy graph:



In comparing the training accuracy graphs for the VGG-13 and previous CNN models, we observe distinct trends. The VGG-13 model, as shown in the first graph, demonstrates a rapid improvement in accuracy over the first two epochs, reaching an accuracy plateau above 90% in later epochs. This model maintains relatively stable training, validation, and testing accuracy, indicating good generalization with minimal overfitting.

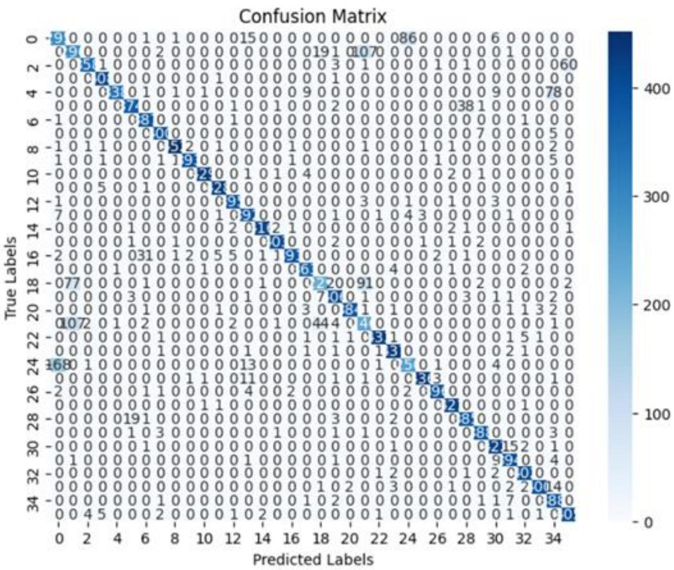
In contrast, the second graph for the previous CNN model displays a higher initial training accuracy starting point and a steady rise toward 95%. However, there is a larger discrepancy between training and validation/test accuracies, suggesting potential overfitting. The previous model achieves higher training accuracy but shows more significant divergence in generalization performance compared to VGG-13.

Loss Graph:



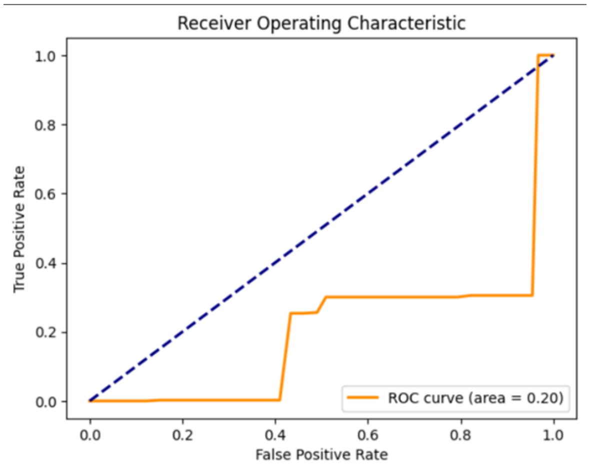
The VGG13 model shows better overall performance compared to the previous CNN model. Its training loss decreases rapidly and stabilizes at a low level, while the validation loss decreases more gradually and stabilizes just above the training loss, indicating some overfitting. The test loss remains stable, suggesting limited generalization improvement with further training. In contrast, the previous CNN model exhibits a consistently decreasing training loss but a gradually increasing validation loss, indicating significant overfitting as it struggles to generalize to validation data. The test loss for the CNN model is also higher and stable, reflecting poor generalization to unseen data. Overall, VGG13 demonstrates better learning and generalization than the previous CNN, though both models show signs of overfitting.

Confusion Matrices:



The VGG-13 model outperforms the previous CNN model in terms of classification accuracy, as seen from the confusion matrices. The VGG-13 confusion matrix shows a clearer diagonal, indicating fewer misclassifications, whereas the simpler CNN model displays more off-diagonal entries, suggesting greater confusion between certain classes. This improvement in VGG-13 can be attributed to its deeper architecture, which provides better feature extraction but requires more computational resources. Overall, VGG-13 demonstrates better accuracy and likely superior precision and recall across classes, suggesting it generalizes better to complex patterns compared to the previous CNN model.

ROC Curve:



The VGG-13 model significantly outperforms the previous CNN model in multi-class classification, as evidenced by its ROC curves and AUC scores. The VGG-13 model achieves near-perfect AUC values (close to 1.0) across most classes, indicating strong classification accuracy and effective differentiation between true positives and false positives. In contrast, the CNN model shows a much lower AUC score (0.20), suggesting weaker performance in distinguishing classes. The improvement in the VGG-13 model is likely due to its deeper architecture, which captures complex patterns, and the custom weight initialization that aids in stable and efficient training. Overall, the VGG-13 model demonstrates superior generalization and accuracy compared to the simpler CNN architecture.

References:

- https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html
- https://www.geeksforgeeks.org/imbalanced-learn-module-in-python/
- https://pytorch.org/docs/stable/tensors.html
- https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
- https://pytorch.org/docs/stable/optim.html
- https://medium.com/@aidant0001/batch-normalization-with-pytorch-959744b05325
- https://kozodoi.me/blog/20210219/gradient-accumulation
- https://www.geeksforgeeks.org/introduction-convolution-neural-network/
- https://www.tensorflow.org/tutorials/images/cnn
- https://pytorch.org/vision/main/models/generated/torchvision.models.vgg13.html
- https://www.kaggle.com/datasets/pytorch/vgg13bn