INTRODUCTION TO MACHINE LEARNING

ASSIGNMENT 2

Submitted to: Dr. Alina Vereshchaka

CSE 574 - INTRODUCTION TO MACHINE LEARNING

April 2023

Jeevalkant Dandona (50485395)
Nitish Kumar (50485470)

University at Buffalo
Graduate School of Education

TABLE OF CONTENTS

Part-1

Nature of the Dataset

The dataset has 766 rows and 8 columns, last 6 rows contains some alphabetical values because of which they have been dropped and the dtype is changed from object to float64.

The main statistics are described as follows.

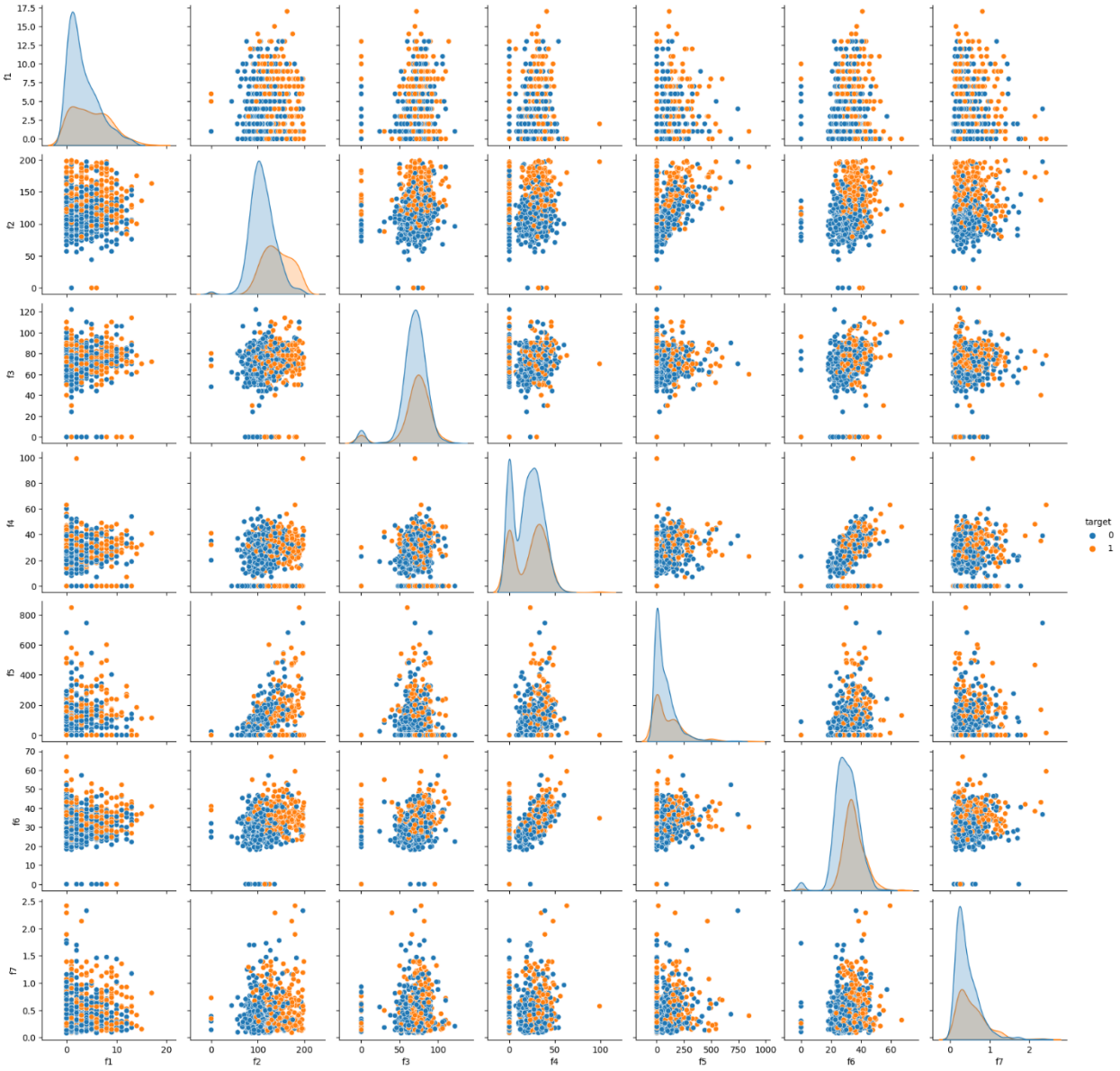
```
In [7]: # main statistics about the entries of the data set
        df.describe()
```

Out[7]:

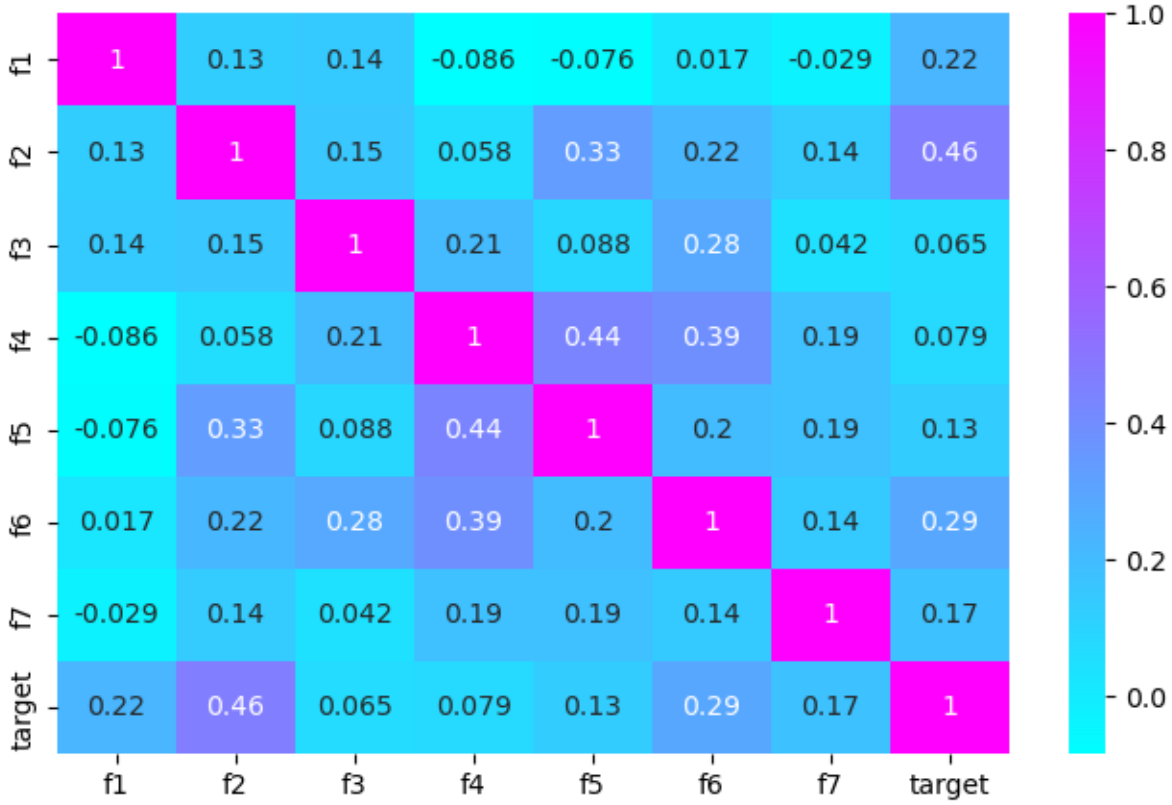
	f1	f2	f3	f4	f5	f6	f7	target
count	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000
mean	3.834211	120.969737	69.119737	20.507895	80.234211	31.998684	0.473250	0.350000
std	3.364762	32.023301	19.446088	15.958029	115.581444	7.899724	0.332277	0.477284
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	0.000000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.300000	0.243750	0.000000
50%	3.000000	117.000000	72.000000	23.000000	36.000000	32.000000	0.375500	0.000000
75%	6.000000	141.000000	80.000000	32.000000	128.250000	36.600000	0.627500	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	1.000000

Visualizations

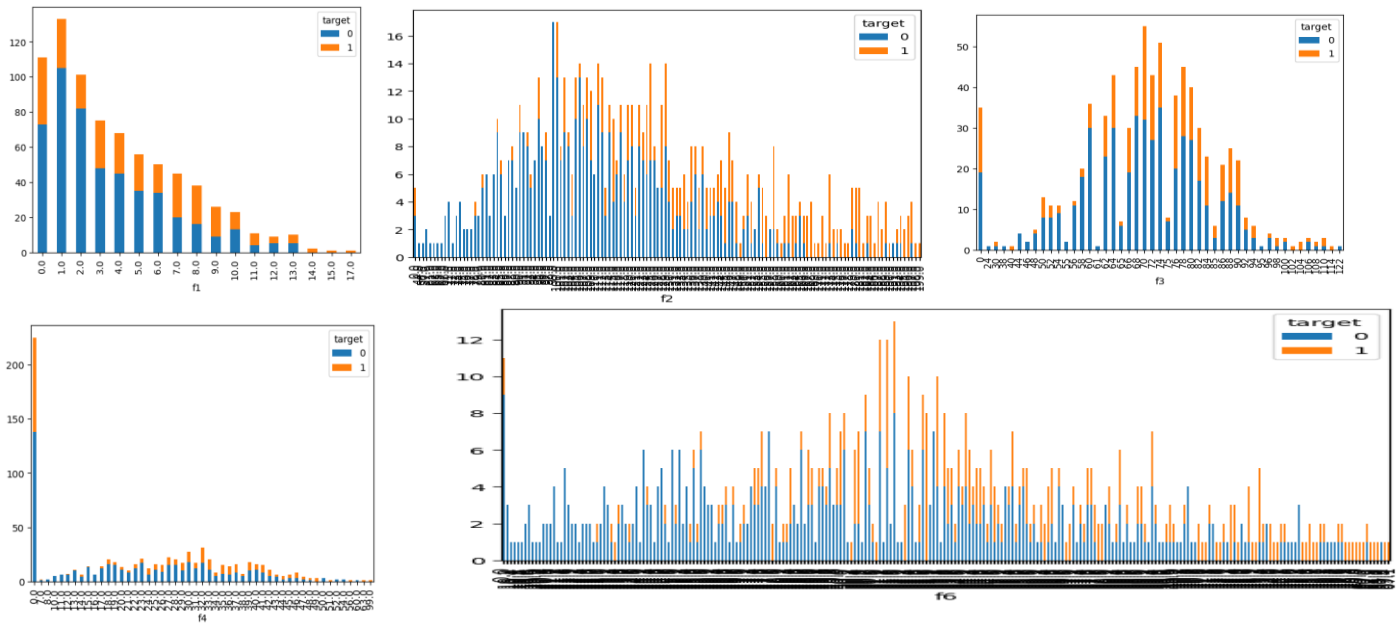
1. **Pair plot** visualizes the given data to find the relationship between the features where the variables can be continuous or categorical. Here the hue is chosen as the target variable.



2. **Heatmap,** this shows the co-relation matrix plotted on the heatmap and shows the relation between features how are the related.



3. **Stacked Bar,** the charts are grouped by the target variable.



Preprocessing

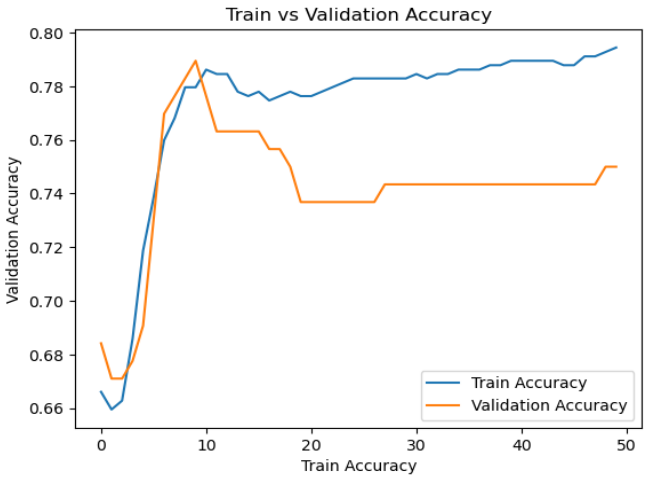
StandardScaler is used in the preprocessing Standardization is a process of scaling the input features so that they have zero mean and unit variance.
The main use of Standard Scaler is to bring all the features to the same scale so that they can be easily compared and analyzed

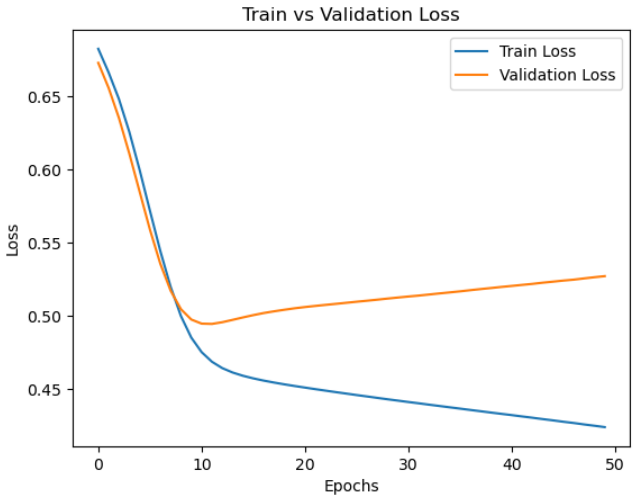
Neural Network Architecture

```
#Neural Network Architecture

class BinaryClassifier(nn.Module):
    def __init__(self):
        super(BinaryClassifier, self).__init__()
        self.hidden_layer1 = nn.Linear(7, 128)
        self.hidden_layer2 = nn.Linear(128, 128)
        self.hidden_layer3 = nn.Linear(128, 128)
        self.output_layer = nn.Linear(128, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.hidden_layer1(x))
        x = self.relu(self.hidden_layer2(x))
        x = self.relu(self.hidden_layer3(x))
        x = self.sigmoid(self.output_layer(x))
        return x
```

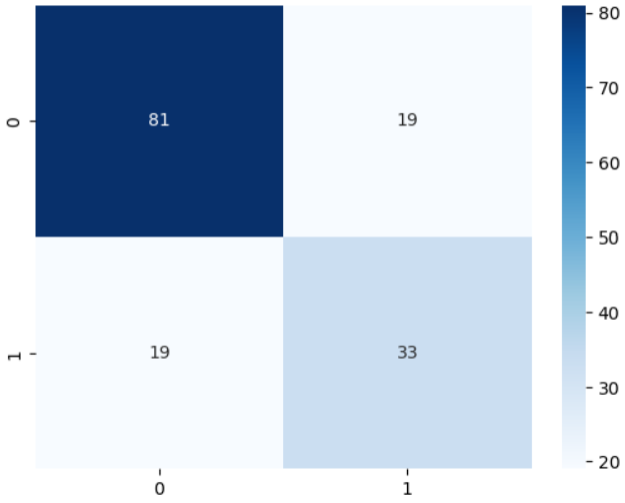
```python
import torch.nn.functional as F
model=BinaryClassifier()
lr=0.0001
def train(model, X_train, y_train, X_test, y_test, epochs=10, lr=0.001, batch_size=32,optimizer = optim.Adam(model.parameters(),
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCELoss()
    train_loss_list, valid_loss_list, train_acc_list, valid_acc_list = [], [], [], []



    for epoch in range(epochs):
        train_loss = 0.0
        valid_loss = 0.0
        train_acc = 0.0
        valid_acc = 0.0
        model.train()

        for i in range(0, len(X_train), batch_size):
            batch_X = X_train[i:i+batch_size]
            batch_y = y_train[i:i+batch_size]
            optimizer.zero_grad()
            output = model(batch_X)
            loss = criterion(output, batch_y.unsqueeze(1).float())
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * len(batch_X)
            train_acc += ((output > 0.5).int() == batch_y.unsqueeze(1)).sum().item()

        model.eval()
        with torch.no_grad():
            output = model(X_test)
            valid_loss = criterion(output, y_test.unsqueeze(1).float()).item() * len(X_test)
            valid_acc = ((output > 0.5).int() == y_test.unsqueeze(1)).sum().item()


        train_loss /= len(X_train)
        valid_loss /= len(X_test)
        train_acc /= len(X_train)
        valid_acc /= len(X_test)
        train_loss_list.append(train_loss)
        valid_loss_list.append(valid_loss)
        train_acc_list.append(train_acc)
        valid_acc_list.append(valid_acc)


        #print(f'Epoch: {epoch+1}/{epochs}, Train Loss: {train_loss:.4f}, Valid Loss: {valid_loss:.4f}, Train Accuracy: {train_ac

    return train_loss_list, valid_loss_list, train_acc_list, valid_acc_list
```

PLOTS





Accuracy: 0.7500



Part-2

All Neural Network Setups

Dropout

	Setup-1	Accuracy	Setup-2	Accuracy	Setup-3	Accuracy
Dropout	0.1	0.7566	0.2	0.7434	0.5	0.74
Optimizer	Adam		Adam		Adam	
Activation Function	ReLu		ReLu		ReLu	
Initializer	-		-		-	

Optimizer

	Setup-1	Accuracy	Setup-2	Accuracy	Setup-3	Accuracy
Dropout	-	0.7566	-	0.7566	-	0.7434
Optimizer	Adam		SGD		RMSprop	
Activation Function	ReLu		ReLu		ReLu	
Initializer	-		-		-	

Initializer

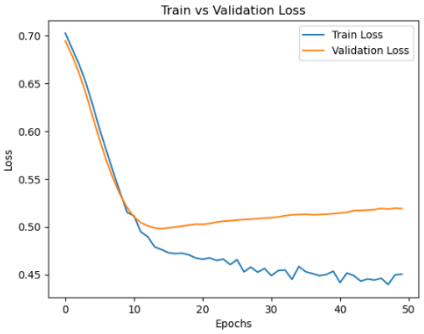
	Setup-1	Accuracy	Setup-2	Accuracy	Setup-3	Accuracy
Dropout	-	0.7566	-	0.7039	-	0.75
Optimizer	Adam		Adam		Adam	
Activation Function	ReLu		ReLu		ReLu	
Initializer	Xavier uniform		Kaiming Uniform		Othogonal	

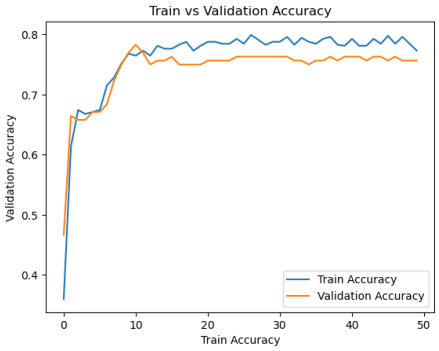
Activation Function

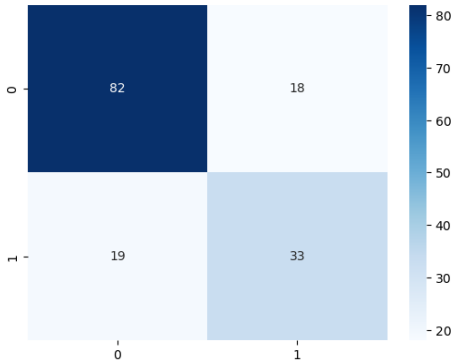
	Setup-1	Accuracy	Setup-2	Accuracy	Setup-3		Accuracy
Dropout	-	0.7632	-	0.7368	-		0.7566
Optimizer	Adam		Adam		Adam		
Activation Function	ReLu		tanh		Sigmoid		
Initializer	-		-		-		

Dropout Plots

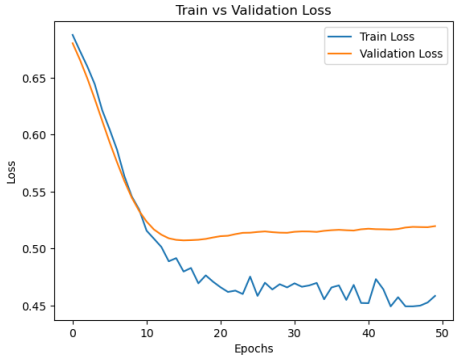
for 0.1

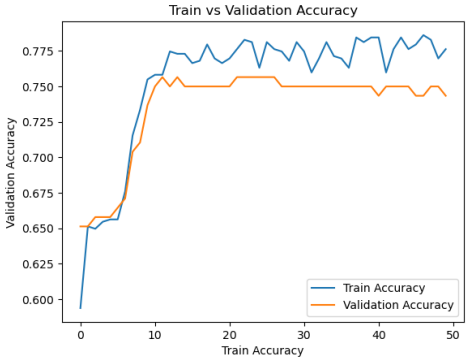


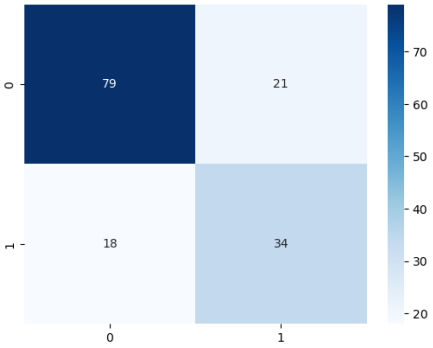




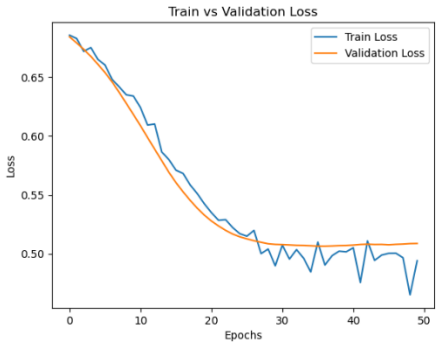
for 0.2

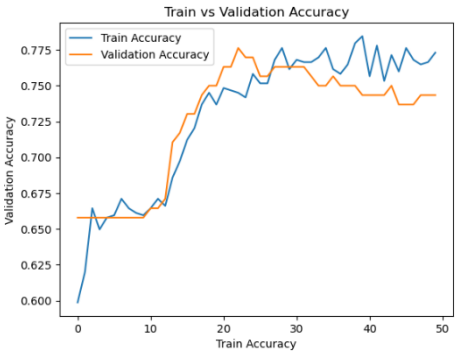


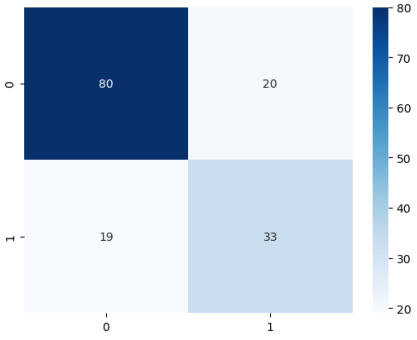




for 0.5

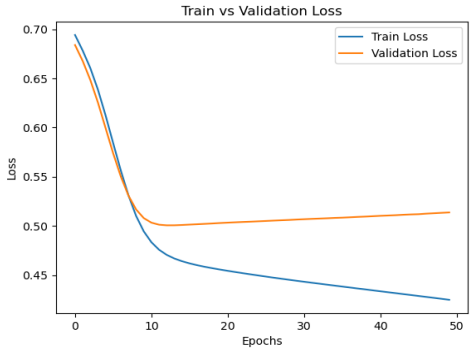


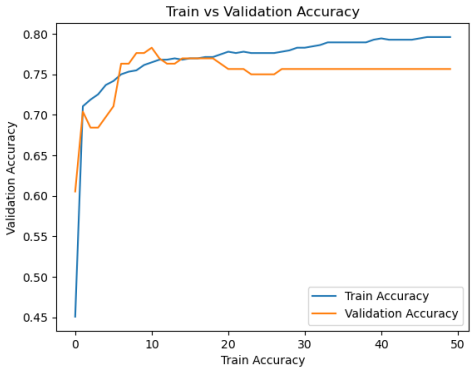


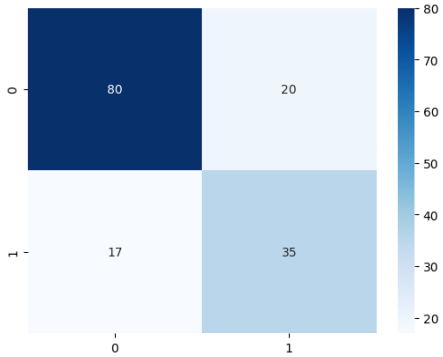


Optimizer Plots

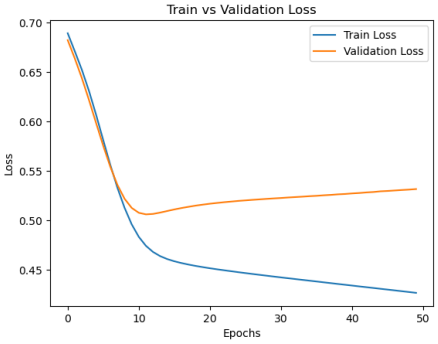
for Adam

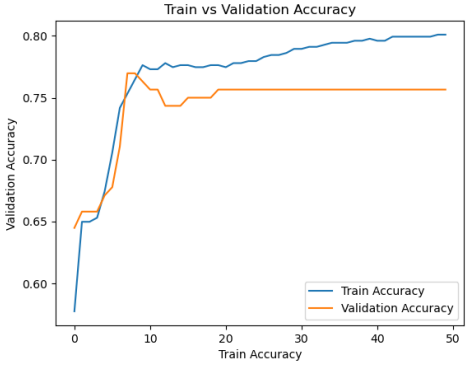


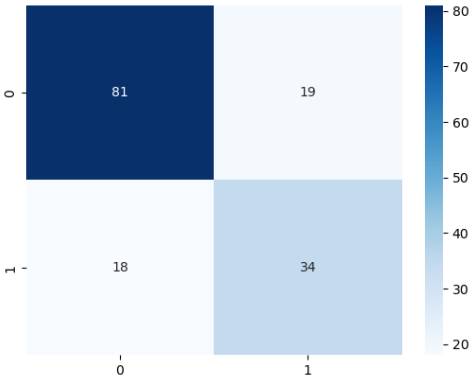




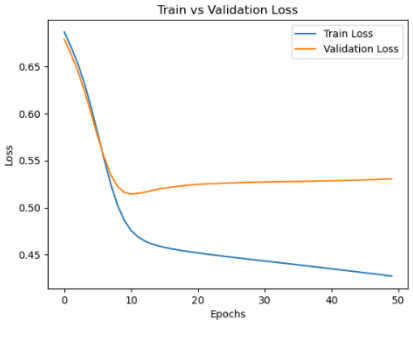
for SGD

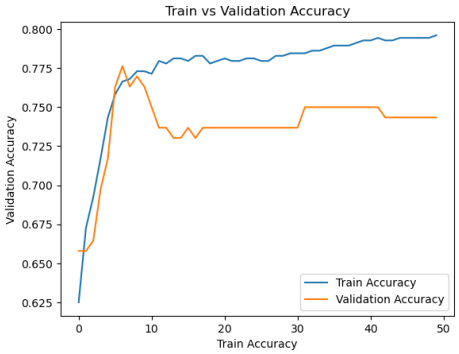


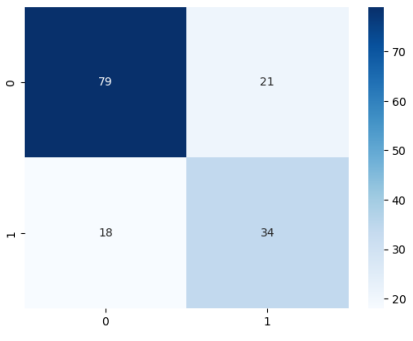




for RMSprop

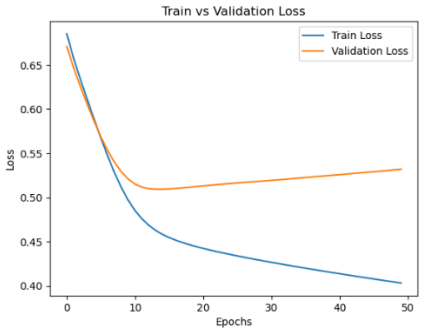


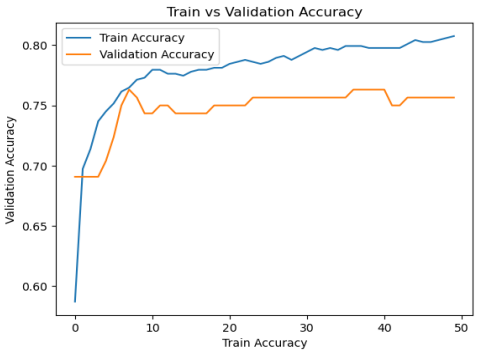


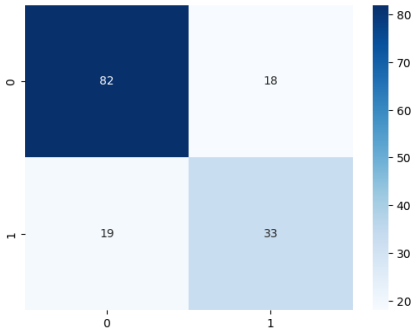


Initializer Plots

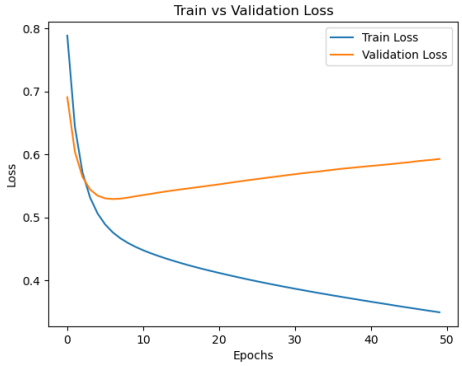
for Xavier Uniform

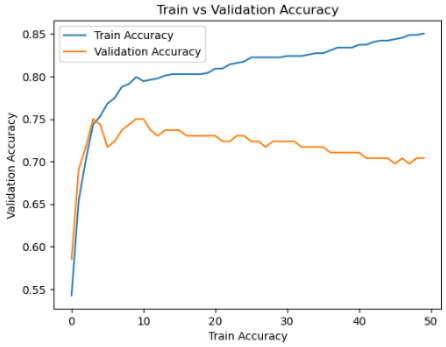


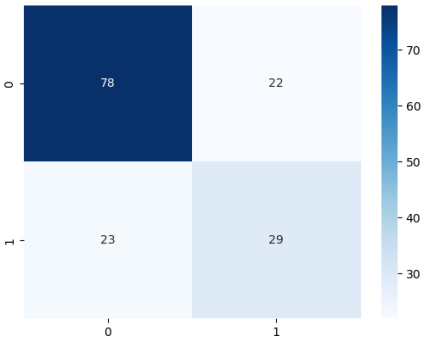




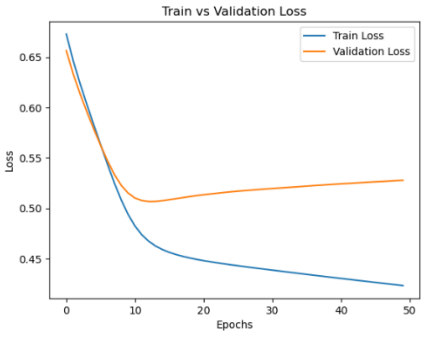
for Kaiming Uniform

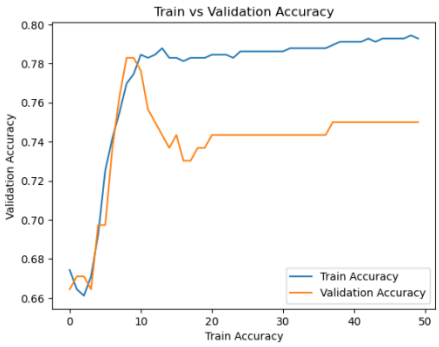


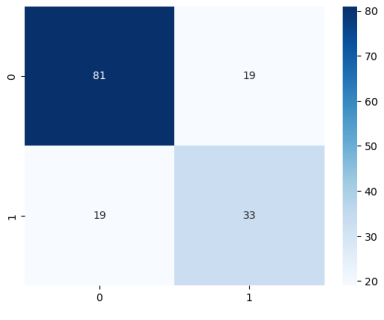




for Orthogonal

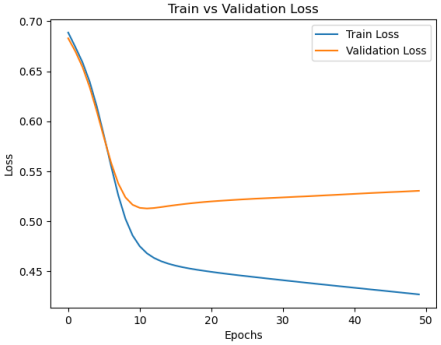


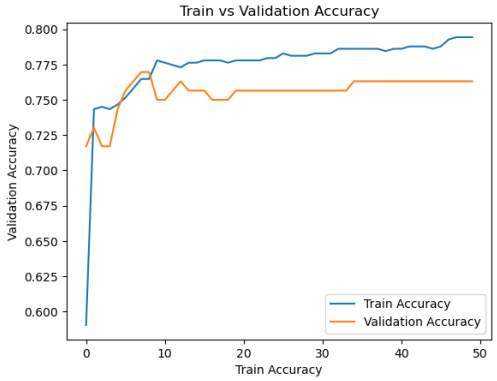


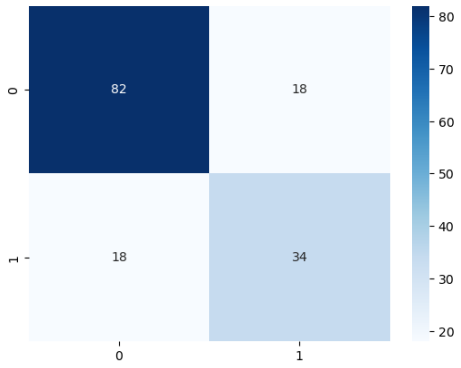


Activation Function Plots

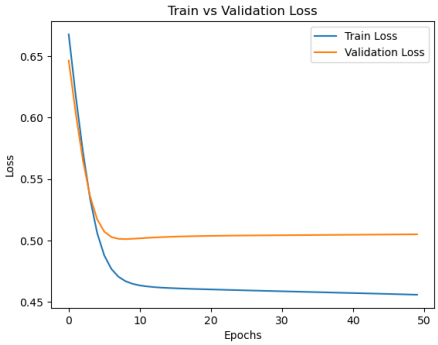
for ReLu

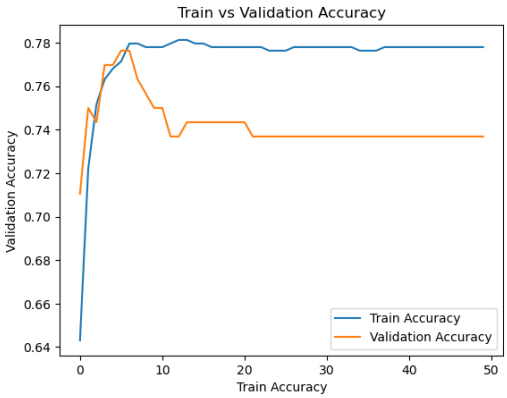


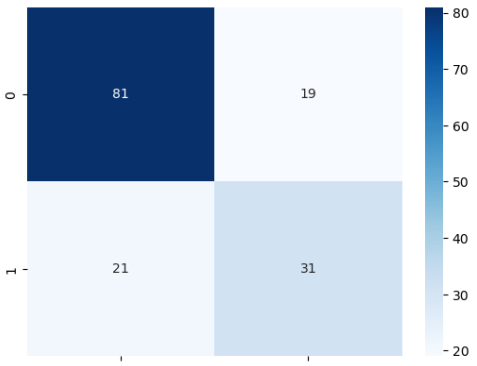




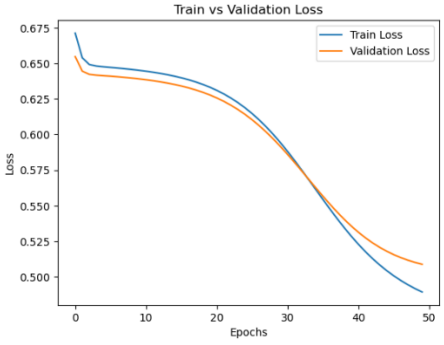
for tanh

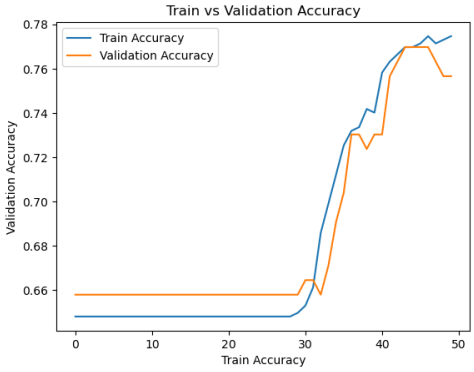


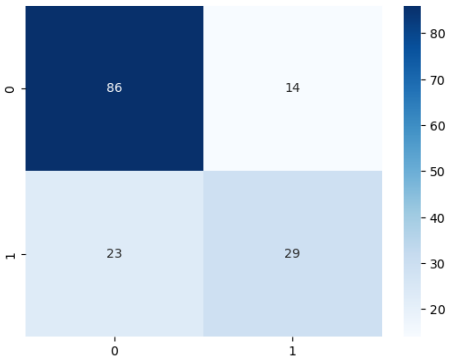




for Sigmoid







Analysis of Neural Network Setups

12 Different neural network setups were created with 4 different hyperparameter's namely, Dropout, Optimizer, Activation Function and Initializer.

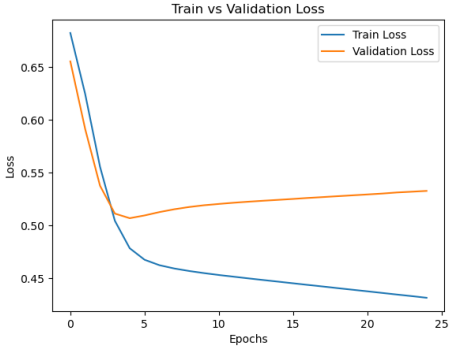
In all the tuning models the basic architecture, epochs, batch size and learning rate was kept the same to see the difference in impact on the accuracy of the models.

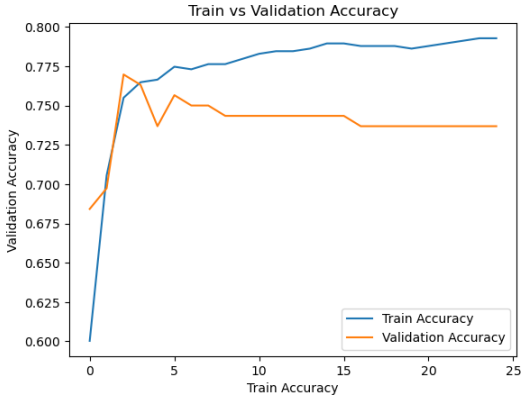
So the ReLu Activation function model performed the best and is chosen as our base model for the further analysis.

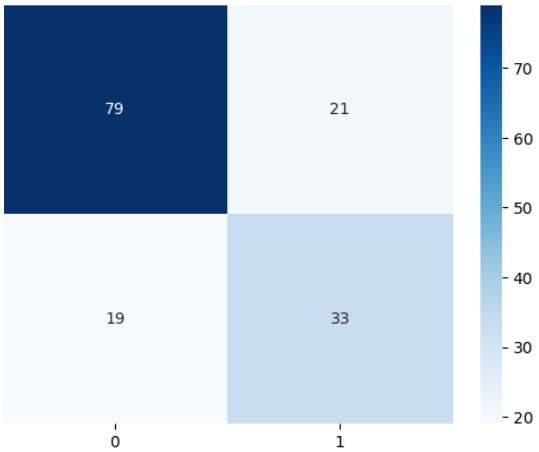
Methods used to improve the accuracy of the Model

Early Stopping

Early stopping works by monitoring the performance of a model on a validation set during the training process, and stopping the training when the performance on the validation set starts to deteriorate.

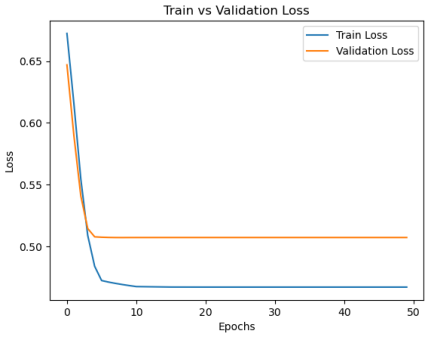


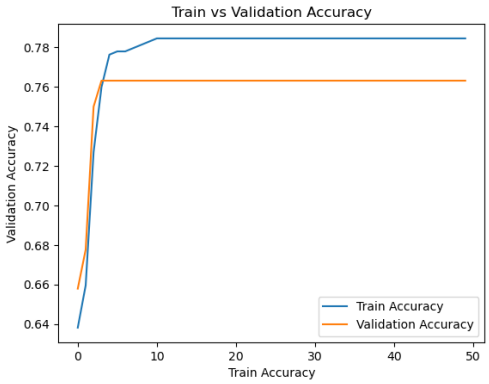


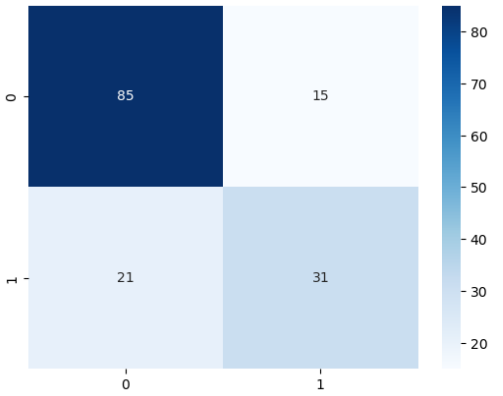


Learning Rate Scheduler

It is a technique used in machine learning to dynamically adjust the learning rate during training. The learning rate is a hyperparameter that controls the step size at each iteration of the optimization algorithm. It determines how quickly or slowly the model learns from the data.

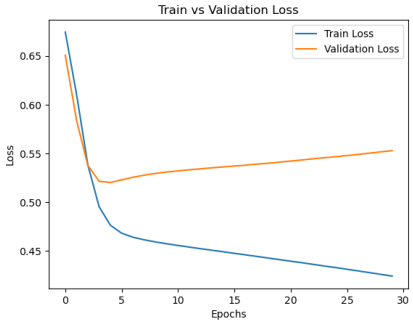


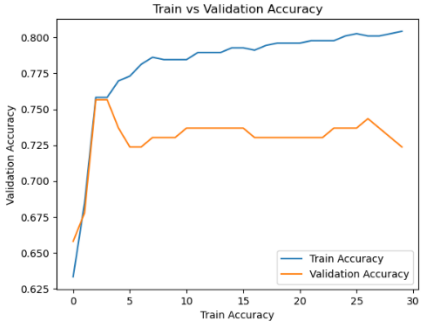


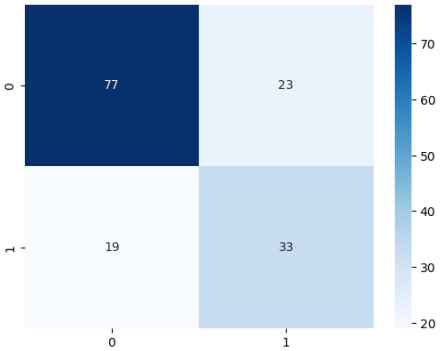


Gradient Clipping

Gradient clipping is a technique used in machine learning to prevent the exploding gradient problem, which can occur during the training of deep neural networks. The exploding gradient problem occurs when the gradients of the loss function with respect to the model parameters become very large, making it difficult for the optimization algorithm to converge.

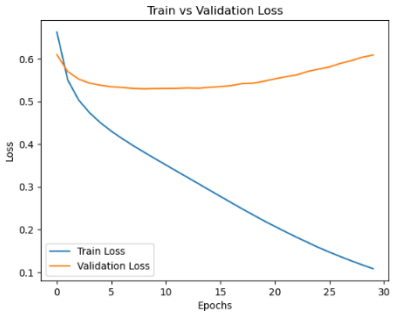


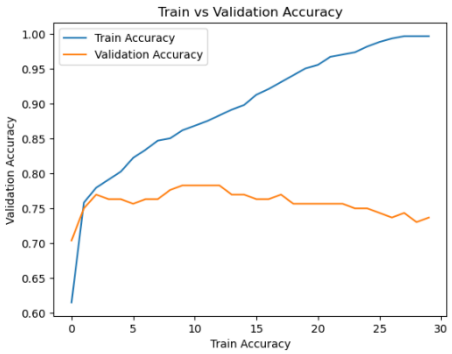


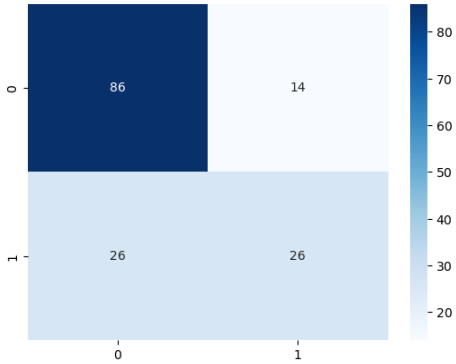


Batch Normalization

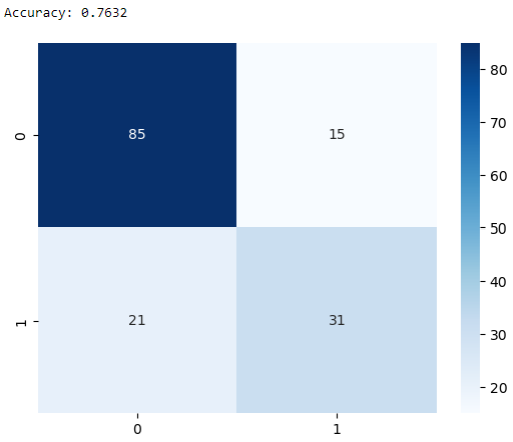
Deep learning uses batch normalization as a method to enhance the performance and stability of neural networks. Each layer's input values are standardized to have a zero mean and unit variance before being scaled and shifted using learnable parameters.







Learning rate parameter *worked best on the base model and gave the accuracy of 76.32%*



Part-3

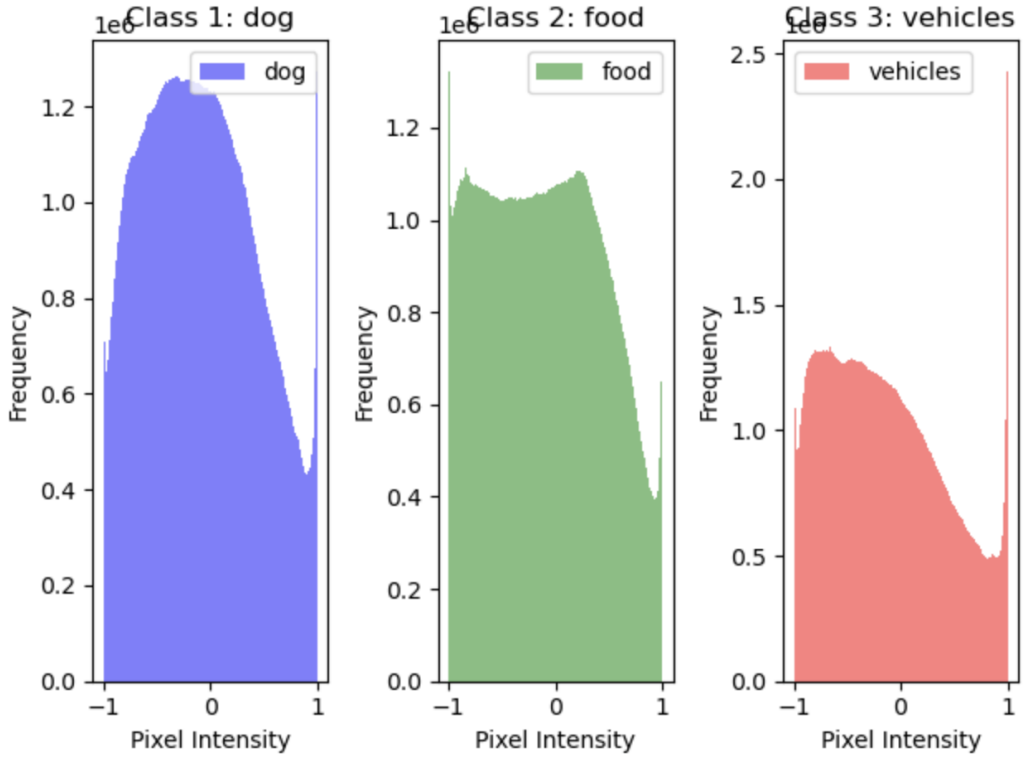
1. **Provide brief details about the nature of your dataset. What is it about? What type of data are we encountering? How many entries and variables does the dataset comprise? Provide the main statistics about the entries of the dataset. Provide at least 3 visualization graphs with short description for each graph.**

    - The given dataset comprises of 30,000 images of 64x64 pixels.
    - The images are labelled into three classes – 1. Dogs 2. Vehicles 3. Food.
    - There are 10,000 images of each class separated into three folders.
    - The format of images is .jpg
    - The dataset is not split into train and test.
    - The source of this dataset is UBlearns.
    - The images are RGB images, and they have 3 channels.

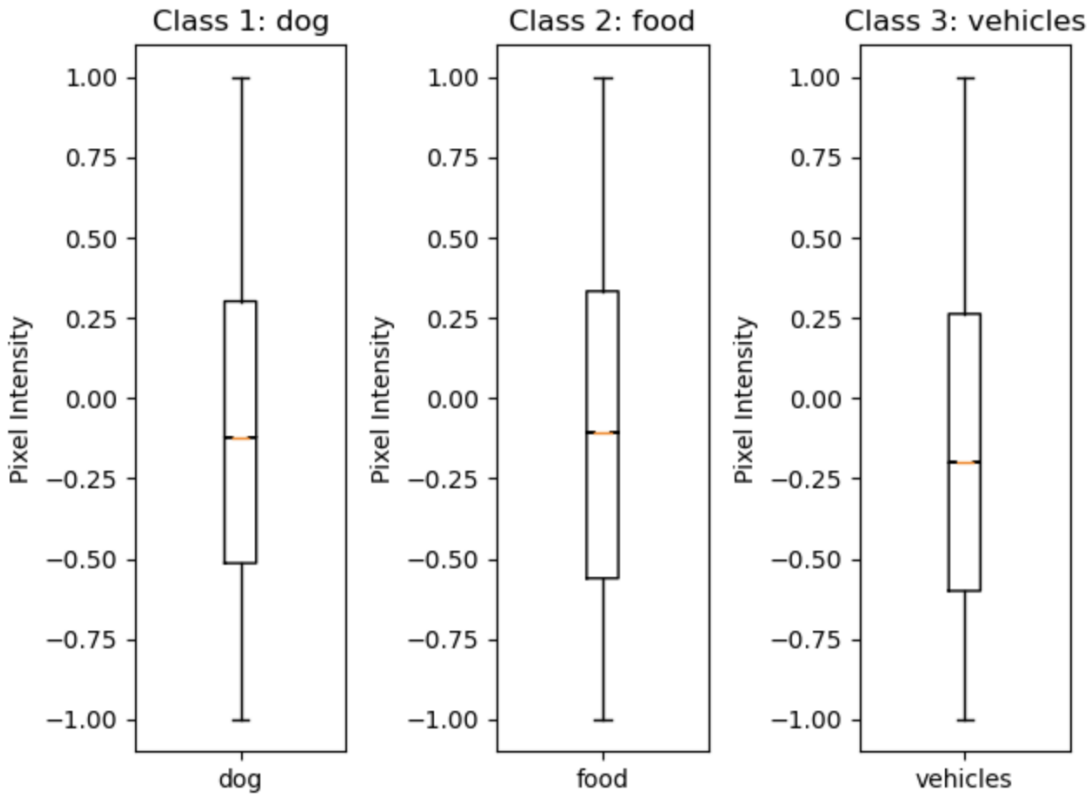
Plots to describe the dataset:

a) Intensity Histogram.

We can clearly see the difference between the intensity distribution of the three classes of images.

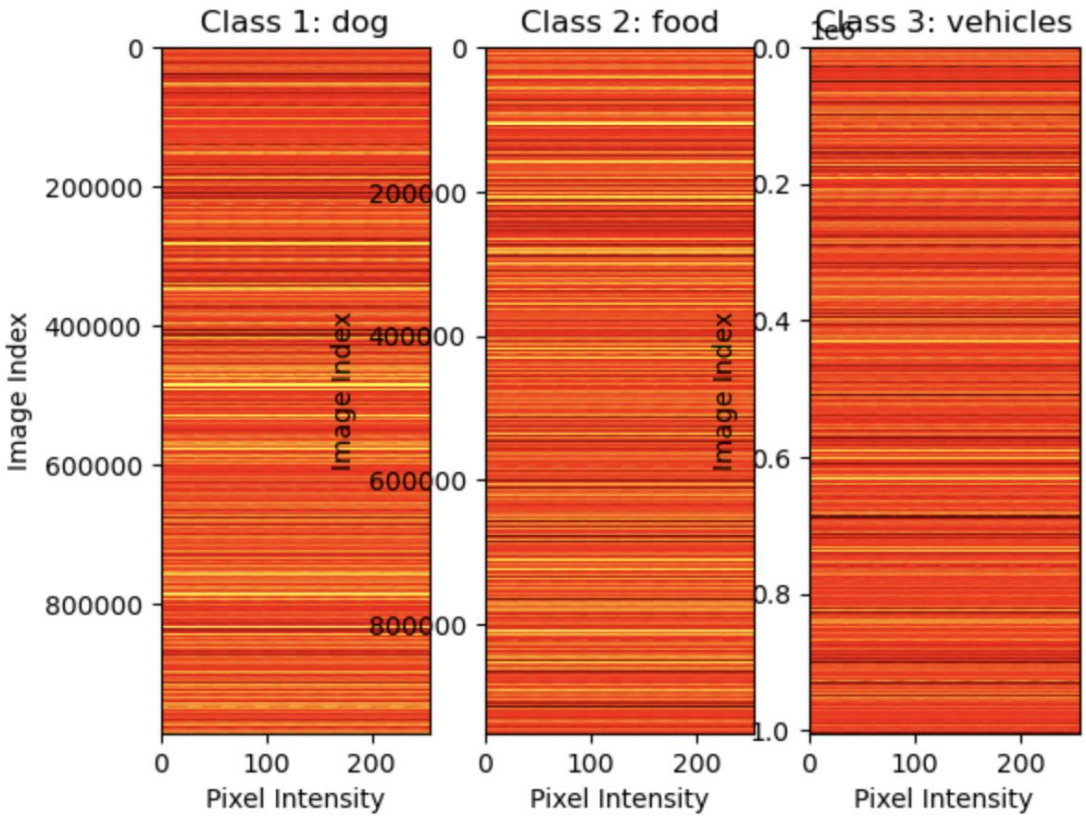


b) Boxplots



We can clearly see the difference between the median intensities and other quartiles.

c) Heatmap



There is a marked difference between the heatmaps of the three classes.

2. **Provide details about your implemented model (AlexNet). Discuss your results. Provide graphs that compares test and training accuracy on the same plot.**

The implemented model is a convolutional neural network (CNN) called AlexNet, which was proposed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in their 2012 paper "ImageNet Classification with Deep Convolutional Neural Networks". AlexNet was one of the pioneering models that demonstrated the effectiveness of deep neural networks for image classification tasks.

The model architecture consists of two main parts: the features module and the classifier module.

1. Features Module:

- The features module starts with a convolutional layer with 3 input channels (representing RGB images), 96 output channels, a kernel size of 11, a stride of 4, and padding of 2.
- ReLU activation is applied in-place (inplace=True) after the first convolutional layer.
- A max pooling layer with a kernel size of 3 and a stride of 2 is applied after the first convolutional layer.
- The second convolutional layer has 96 input channels (from the previous layer), 256 output channels, a kernel size of 5, and padding of 2.
- ReLU activation is applied after the second convolutional layer.
- Another max pooling layer with a kernel size of 3 and a stride of 2 is applied after the second convolutional layer.
- The third, fourth, and fifth convolutional layers have 256, 384, and 384 output channels, respectively, with kernel sizes of 3 and padding of 1.
- ReLU activation is applied after each of these convolutional layers.
- The last max pooling layer with a kernel size of 3 and a stride of 2 is applied after the fifth convolutional layer.

2. Classifier Module:

- The classifier module consists of three fully connected (linear) layers.
- Dropout with a probability of 0.5 is applied before and after the first fully connected layer.
- The first fully connected layer has an input size of 256 * 6 * 6 (calculated based on the input size of the feature map) and an output size of 4096.
- ReLU activation is applied after the first fully connected layer.
- Dropout with a probability of 0.5 is applied before the second fully connected layer.
- The second fully connected layer has an input size of 4096 and an output size of 4096.
- ReLU activation is applied after the second fully connected layer.
- The last fully connected layer has an input size of 4096 (from the previous layer) and an output size equal to the number of classes specified during model instantiation (default is 1000 for ImageNet classification, but in the given code it is set to 3).
- No activation function is applied after the last fully connected layer.

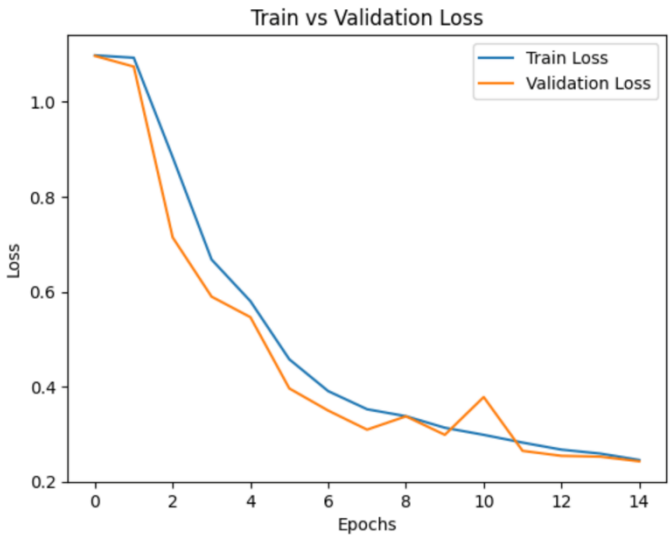
The **forward** method defines how the input data flows through the model, passing through the features module and then the classifier module, and finally returning the output for classification.

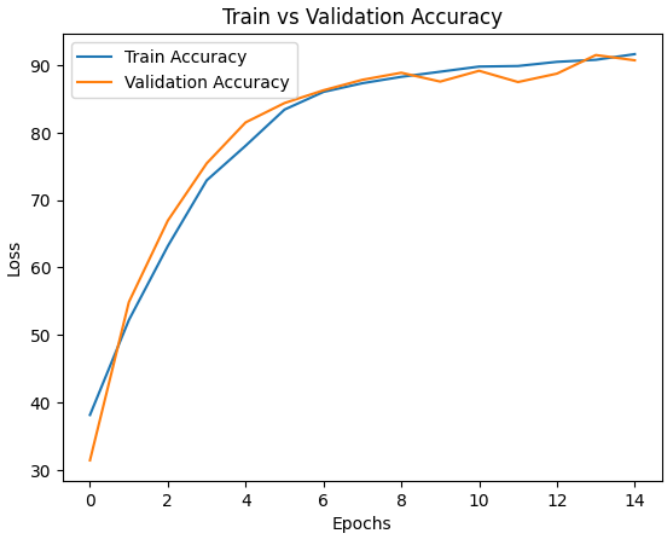
```
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(96, 256, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2)
        )
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.5),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

# Create an instance of AlexNet
model = AlexNet(num_classes=3)

# Print the model architecture
print(model)
```

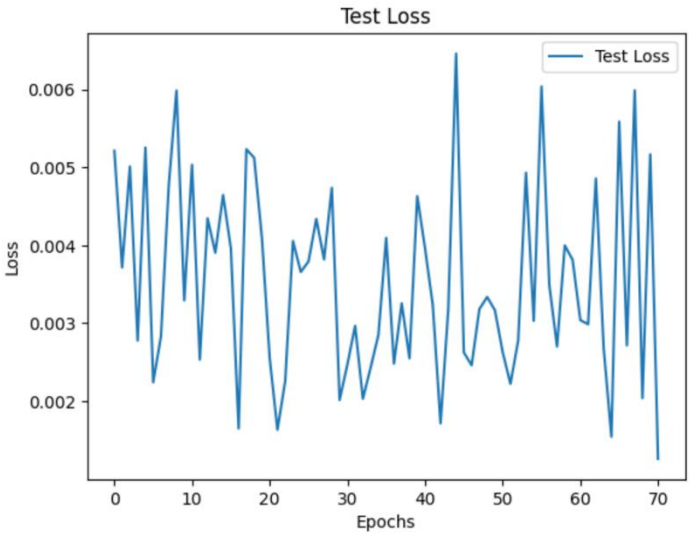
Results:

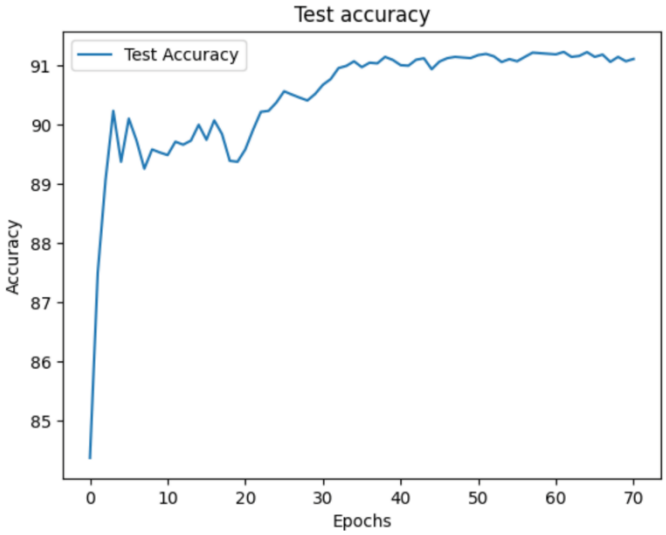
A validation accuracy of 90.93% was achieved at 15 epochs. Final test loss was 0.0013 and final accuracy on test dataset was 91.11

Refer to the following plots:







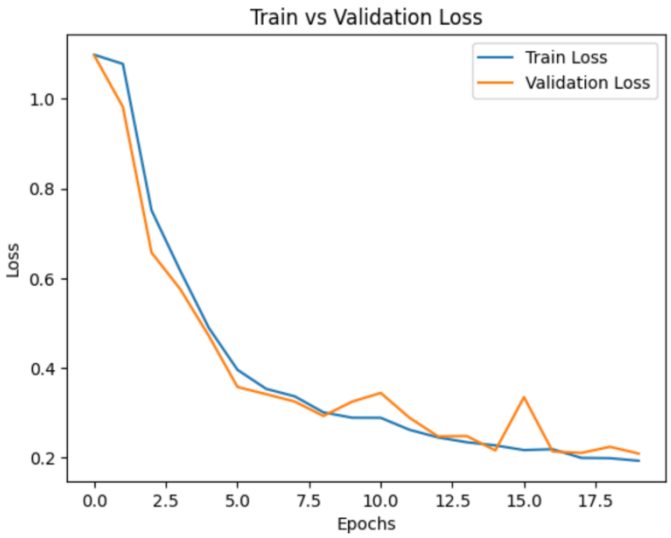


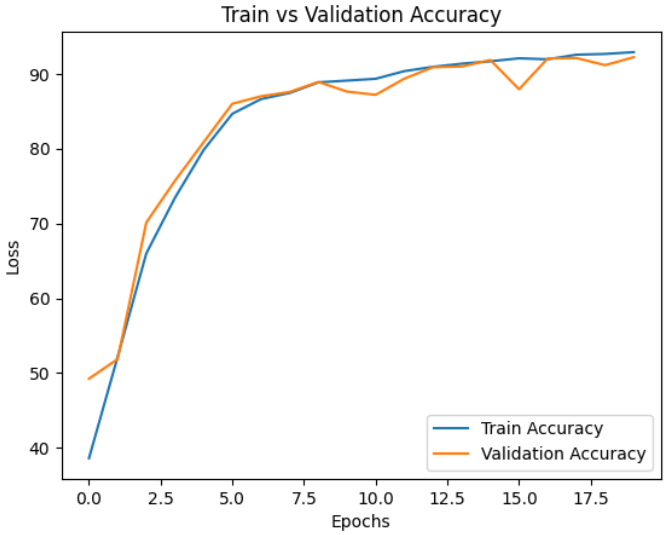
3. Discuss how you improve AlexNet, what methods and tools you have tried and how that helped to improve the training accuracy and training time. Provide graphs that compares test and training accuracy on the same plot.

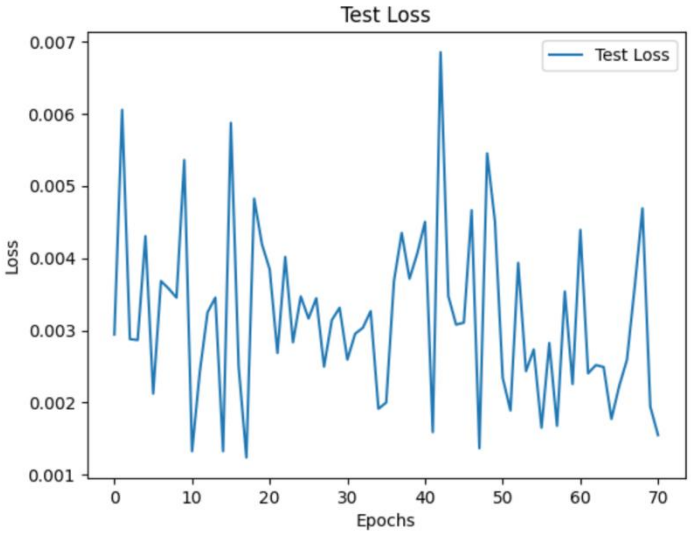
The accuracy was improved by reducing the dropout value and increasing the number of epochs. Also, early stopping was used. Refer to the following plots:

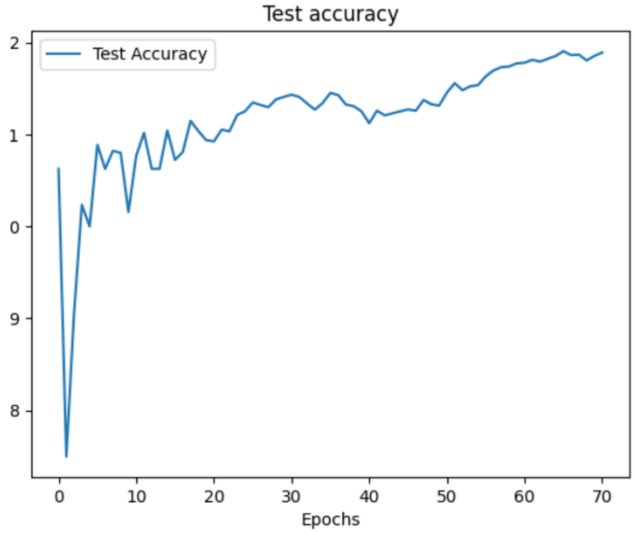
Training time was 12m 20 sec for 20 epochs from 13min 30 sec for 15 epochs

```
class AlexNetImp(nn.Module):
    def __init__(self, num_classes=1000):
        super(AlexNetImp, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(96, 256, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2)
        )
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.35),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.35),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

# Create an instance of AlexNet
model = AlexNet(num_classes=3)

# Print the model architecture
print(model)
```


Train vs Validation Loss
Train Loss
Validation Loss
Loss
Epochs


Train vs Validation Accuracy
Train Accuracy
Validation Accuracy
Loss
Epochs


Test Loss
Test Loss
Loss
Epochs


Test accuracy
Test Accuracy
Epochs

Part-4

1. **Provide brief details about the nature of your dataset. What is it about? What type of data are we encountering? How many entries and variables does the dataset comprise? Provide the main statistics about the entries of the dataset. Provide at least 3 visualization graphs with short description for each graph.**

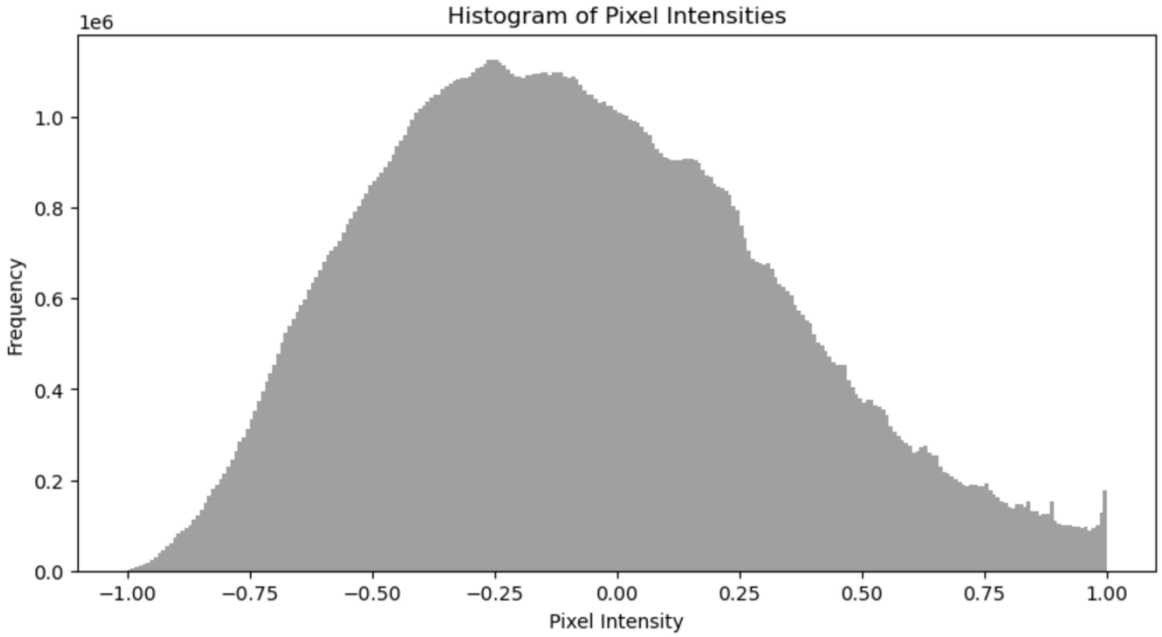
   The SVHN dataset, also known as the Street View House Numbers dataset, is a widely used computer vision dataset for digit recognition tasks. It consists of labeled images of house numbers captured from Google Street View images. The dataset is used for training and evaluating machine learning algorithms to recognize digits in natural scene images.

   The SVHN dataset contains two parts: a training set and a test set. The training set has 73257 images, while the test set has 26032 images. Each image in the dataset is 32x32 pixels in size and contains a single digit (0-9) centered in the image. The images are in RGB format, with three color channels (red, green, and blue).

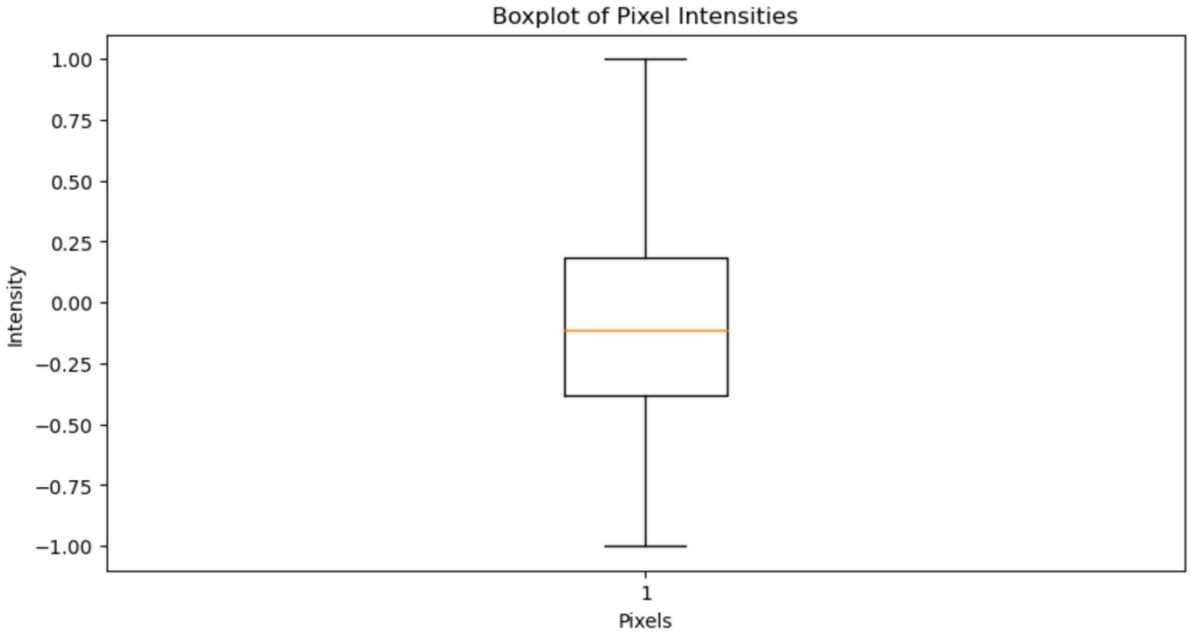
   The main statistics about the entries in the SVHN dataset are as follows:

- Training set:
  - Number of images: 73257
  - Number of digits: 604388
  - Size of images: 32x32
  - No. of channels: 3
  - Format of dataset: .mat
- Test set:
  - Number of images: 26032
  - Number of digits: 215671
  - Size of images: 32x32
  - No. of channels: 3
  - Format of dataset: .mat

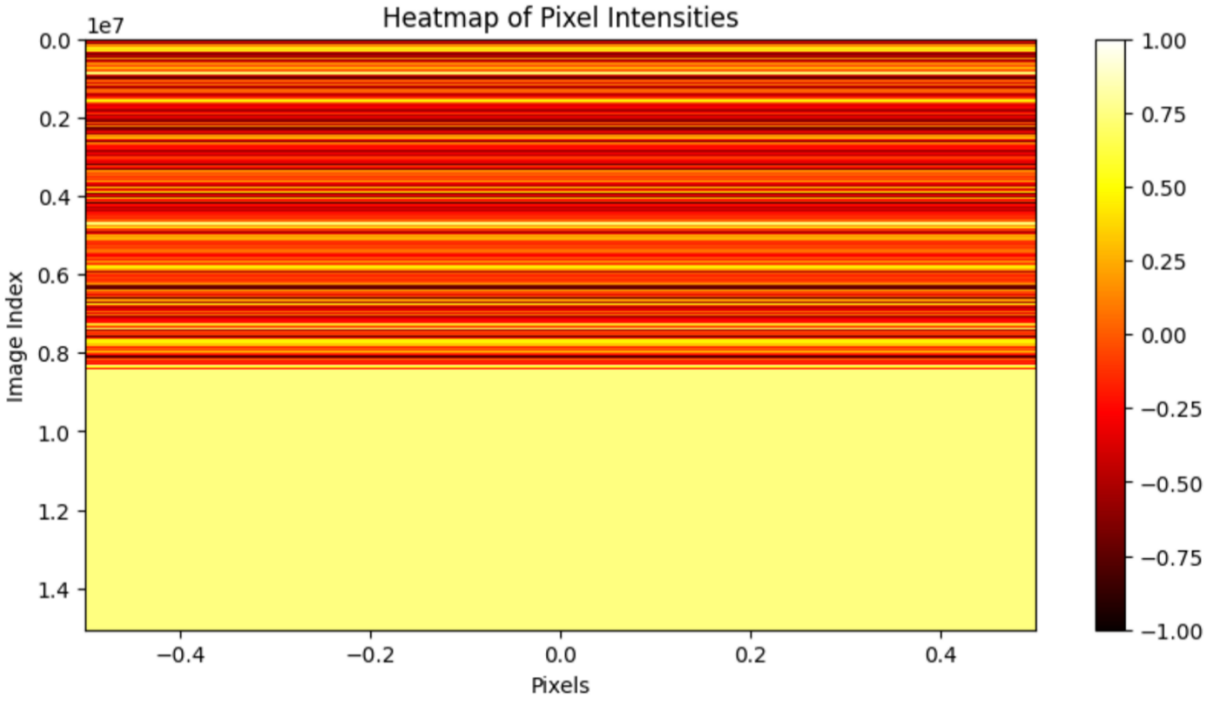
Refer to the follow visualizations:



As expected, it is very close to the normal curve.



There is not much variance in the pixel intensities indicating that the images are similar to each other to some extent.



2. **Discuss briefly how you adjusted your model from Part III for this task.**

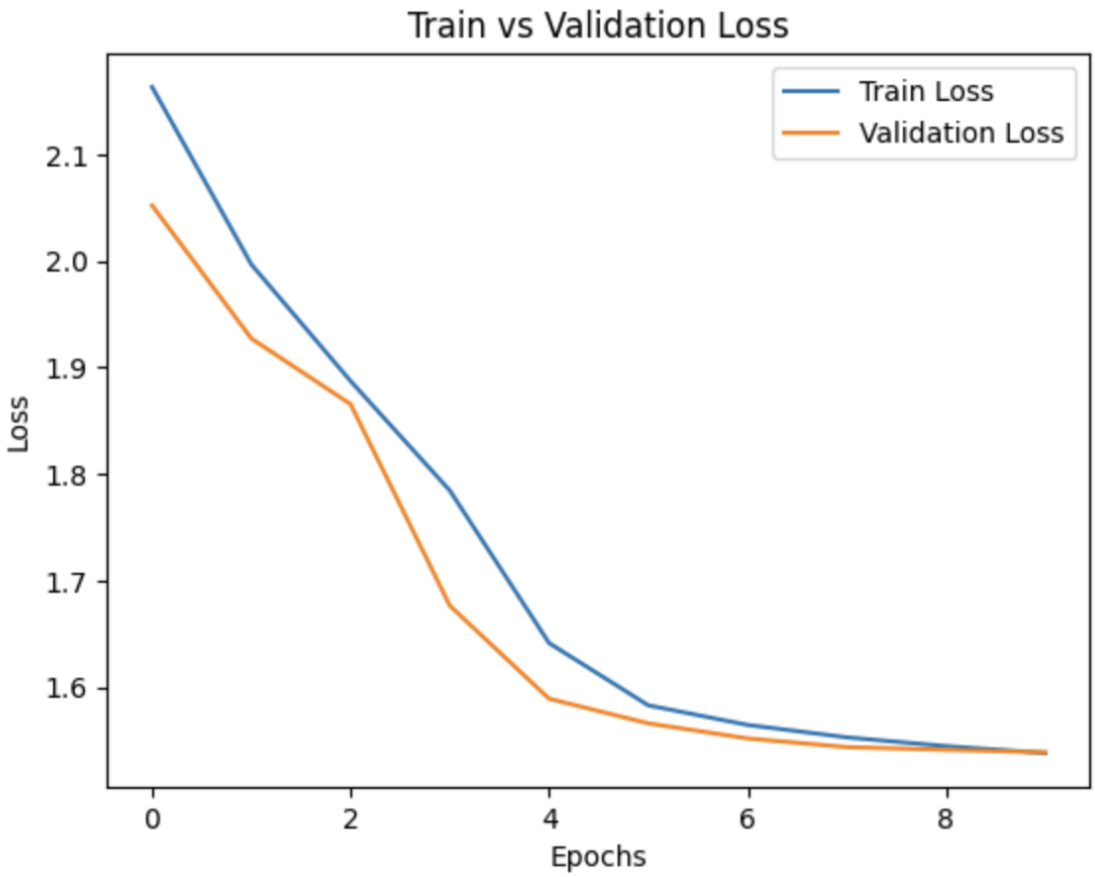
   Following modifications have been made from the part III model:

   - No. of output classes is 10 instead of 3
   - Softmax activation function is used for the output layer instead of ReLU.
   - No dropout is applied.
   - Batch normalization is applied.
   - Adaptive average pooling is used to obtain a fixed-size feature map of 6x6 before flattening and passing through the classifier.

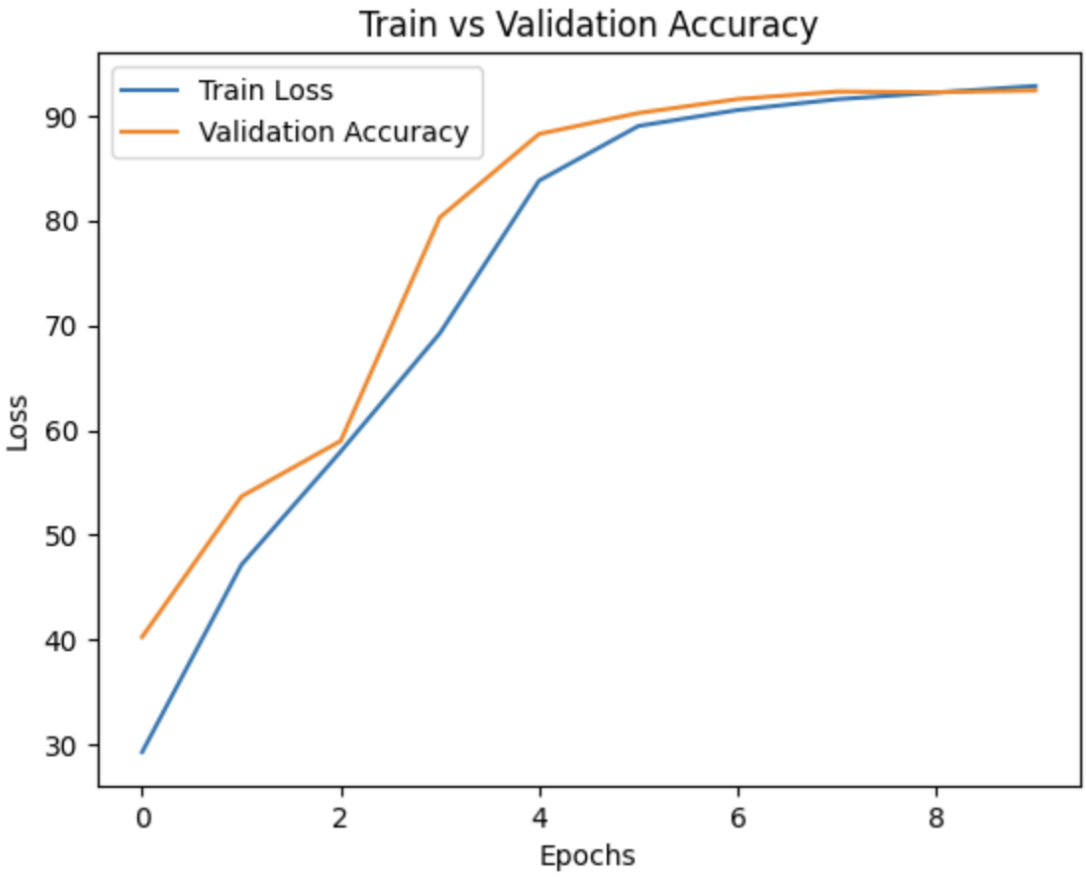
3. **Discuss data augmentation methods you tried and reason how it helped to increase the accuracy of the model.**

   The data augmentation method we used were resizing and normalization, other methods did not yield better results, so we dropped those.

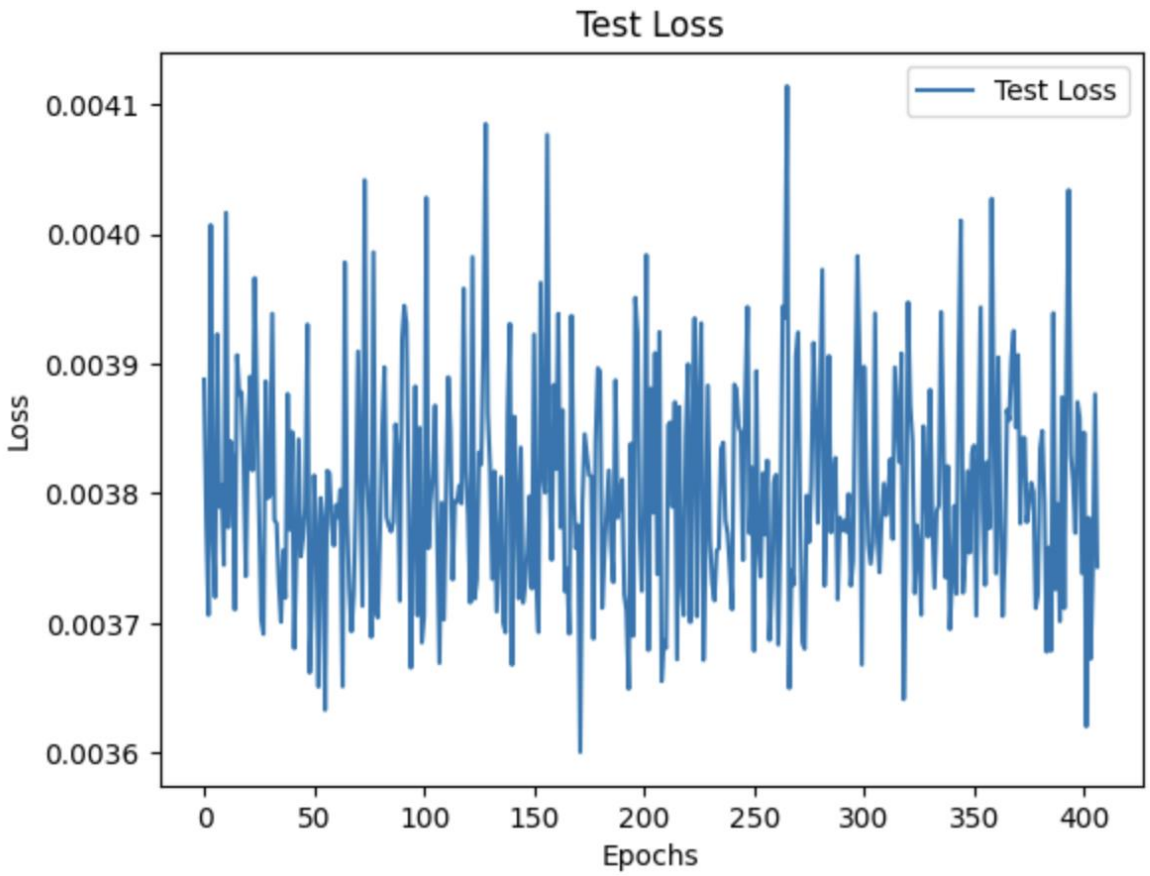
4. **Provide graphs that compares test and training accuracy on the same plot for all your setups and add a short description for each graph.**

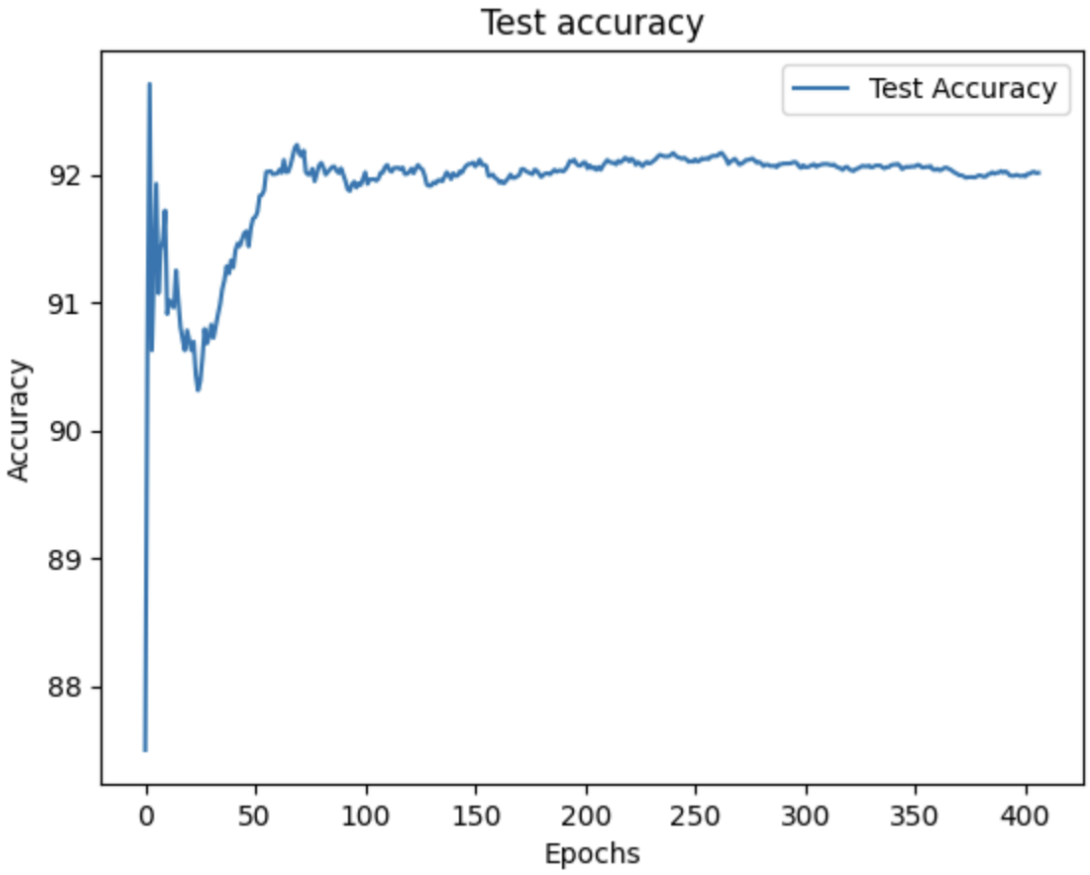


Validation loss drops more rapidly but then eventually train loss becomes lower.



Validation accuracy rises more rapidly but train accuracy eventually catches up.


Test Loss
Test Loss
0.0041
0.0040
0.0039
0.0038
0.0037
0.0036
Loss
0
50
100
150
200
250
300
350
400
Epochs



Test accuracy becomes almost constant after a few epochs.

REFERENCES

WEBSITES

- 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/tutorials/beginner/saving_loading_models.html
- https://wiki.pathmind.com/neural-network