INTRODUCTION TO MACHINE LEARNING

ASSIGNMENT 2

Submitted to: Dr. Alina Vereshchaka

CSE 574 - INTRODUCTION TO MACHINE LEARNING

March 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.

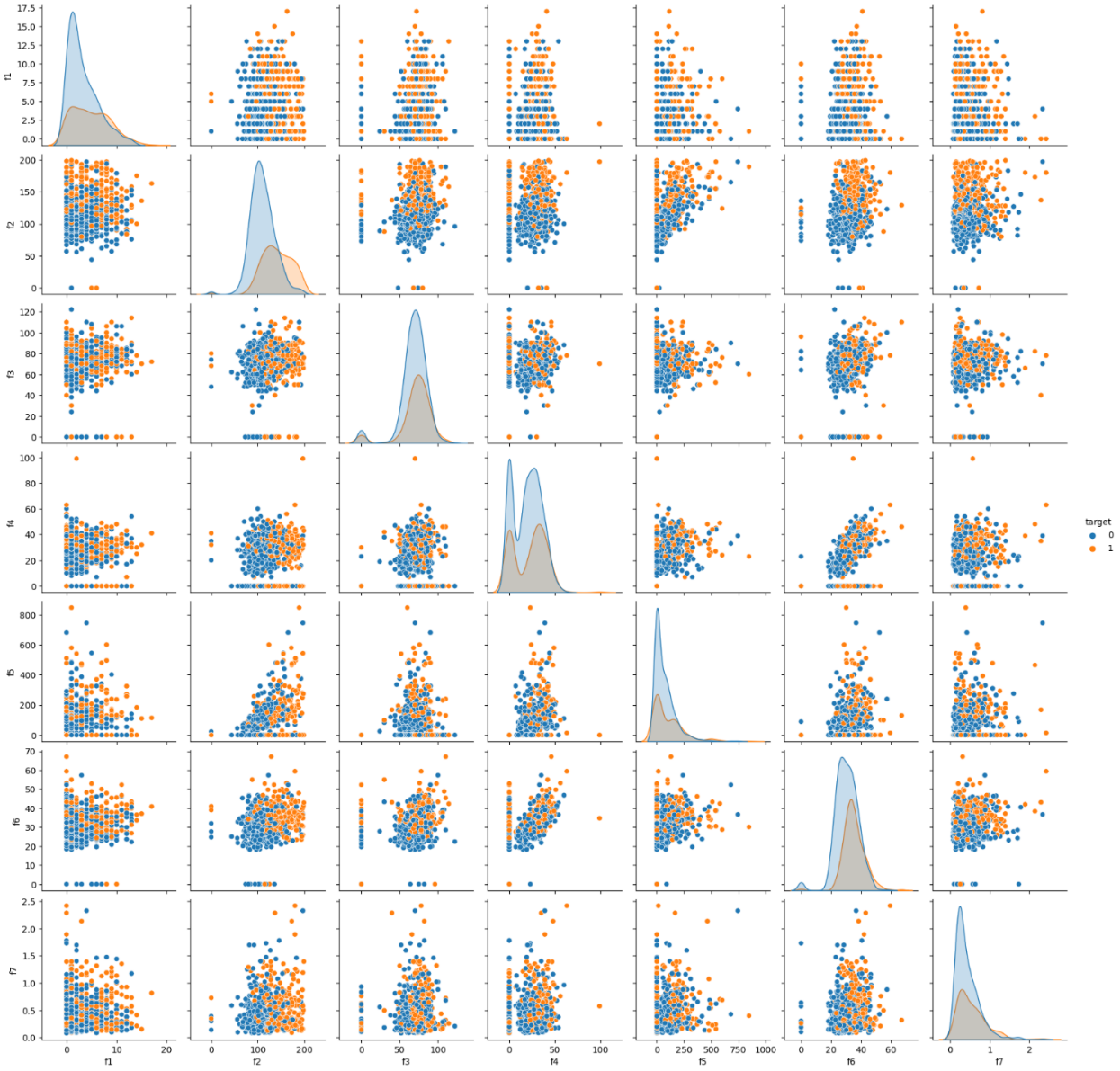
In [7]:
```
# main statistics about the entries of the data set
df.describe()
```

Out[7]:

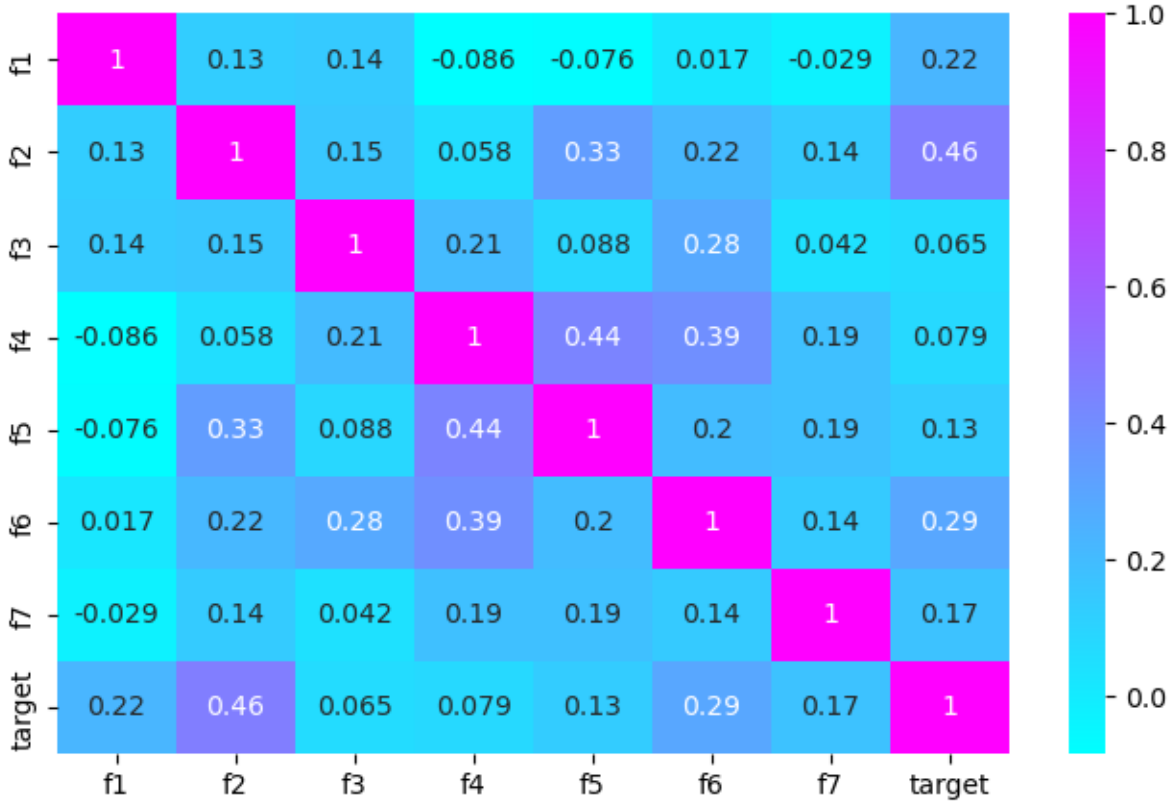
	f1	f2	f3	f4	f5	f6	f7	target
count	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000
mean	3.834211	120.969737	69.119737	20.507895	80.234211	31.998684	0.473250	0.350000
std	3.364762	32.023301	19.446088	15.958029	115.581444	7.899724	0.332277	0.477284
min	0.000000	0.000000	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

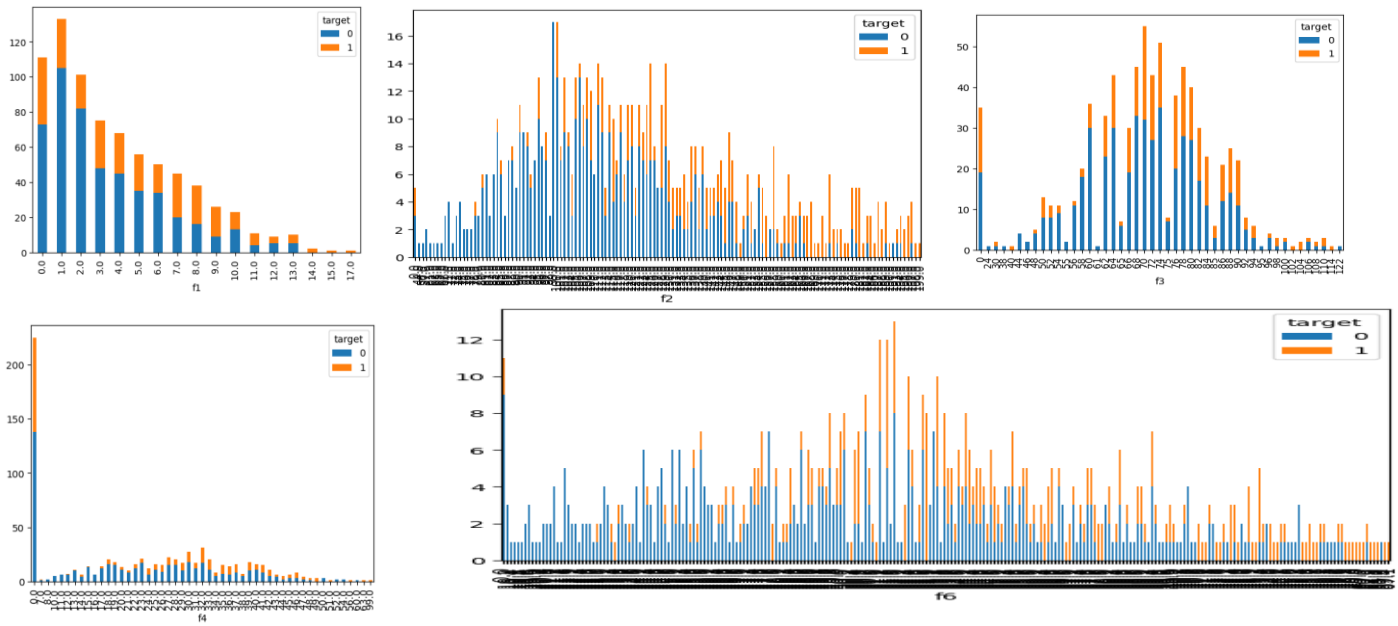
1. **Pair plot** visualizes the given data to find the relationship between the features where the variables can be continuous or categorical. Here the hue is chosen as the target variable.



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

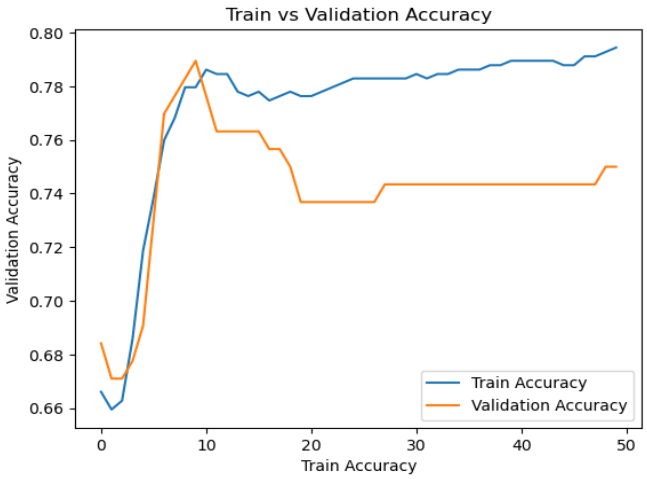
StandardScaler is used in the preprocessing Standardization is a process of scaling the input features so that they have zero mean and unit variance.
The main use of Standard Scaler is to bring all the features to the same scale so that they can be easily compared and analyzed

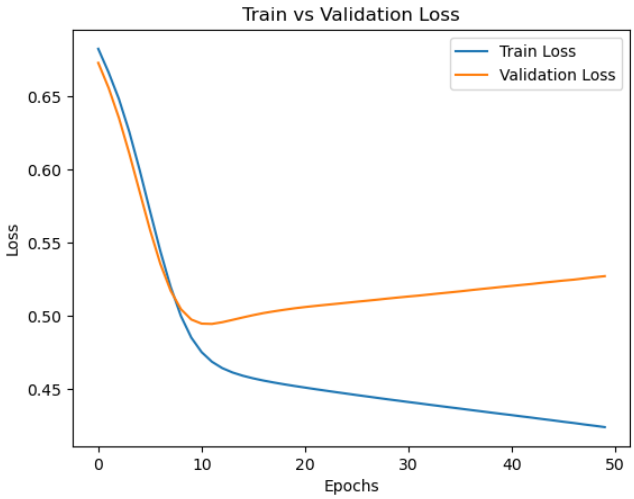
Neural Network Architecture

```
#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
```

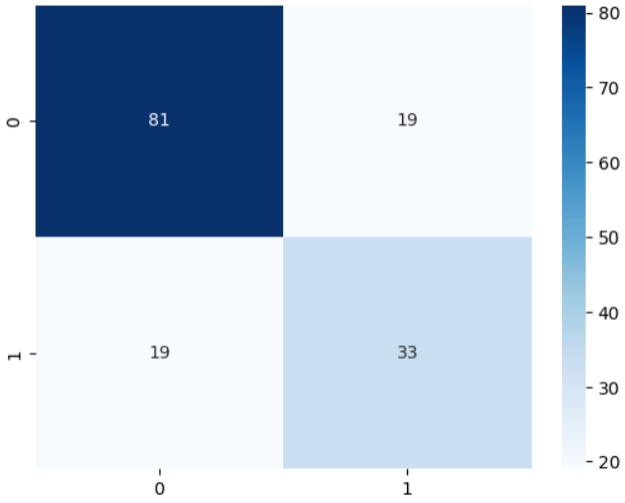
```python
import torch.nn.functional as F
model=BinaryClassifier()
lr=0.0001
def train(model, X_train, y_train, X_test, y_test, epochs=10, lr=0.001, batch_size=32,optimizer = optim.Adam(model.parameters(),
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCELoss()
    train_loss_list,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

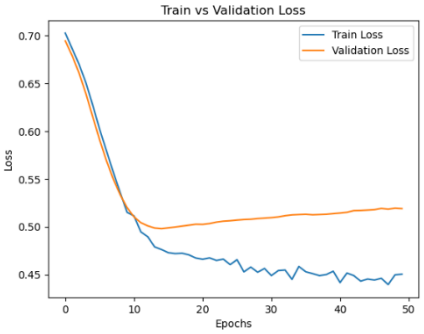
	Setup-1	Accuracy	Setup-2	Accuracy	Setup-3	Accuracy
Dropout	-	0.7566	-	0.7039	-	0.75
Optimizer	Adam		Adam		Adam	
Activation Function	ReLu		ReLu		ReLu	
Initializer	Xavier uniform		Kaiming Uniform		Othogonal	

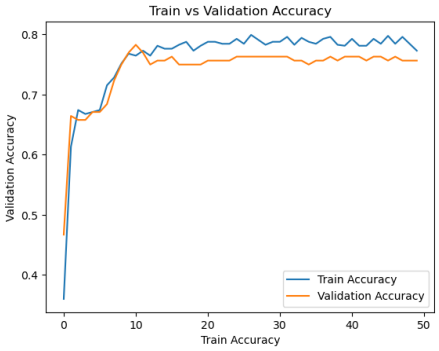
Activation Function

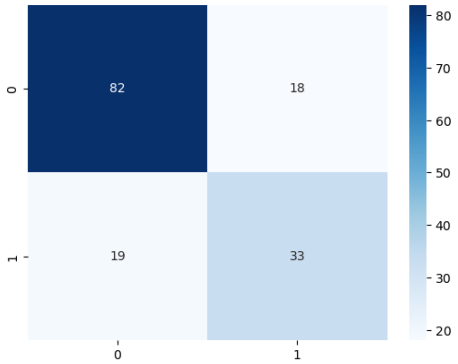
	Setup-1	Accuracy	Setup-2	Accuracy	Setup-3		Accuracy
Dropout	-	0.7632	-	0.7368	-		0.7566
Optimizer	Adam		Adam		Adam		
Activation Function	ReLu		tanh		Sigmoid		
Initializer	-		-		-		

Dropout Plots

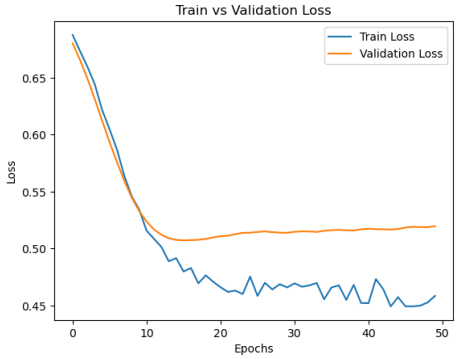
for 0.1

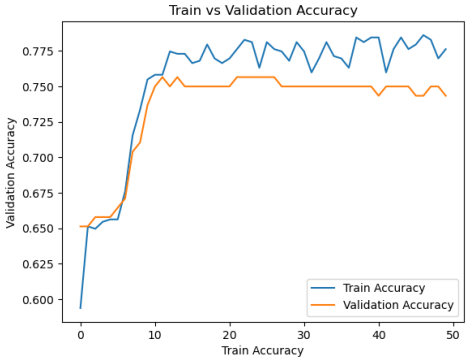


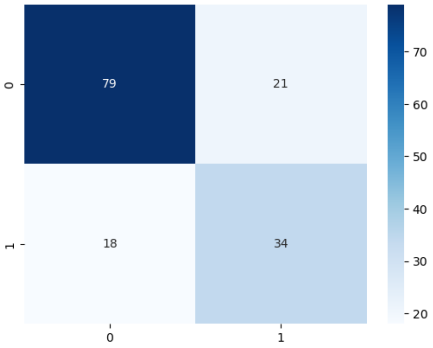




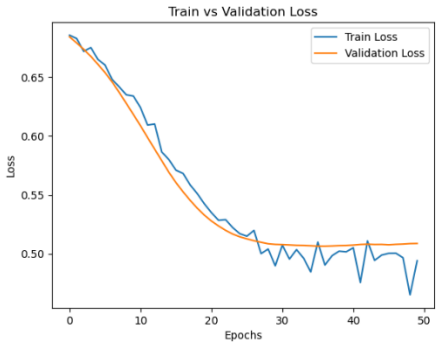
for 0.2

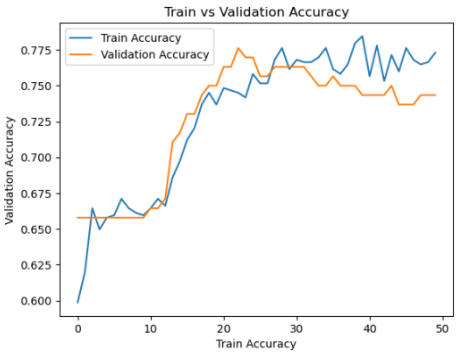


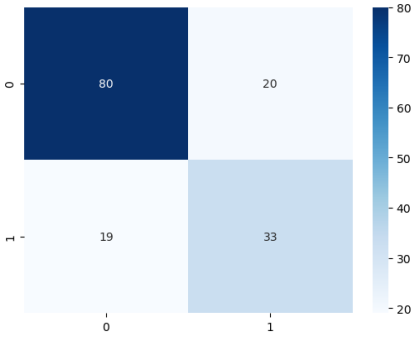




for 0.5

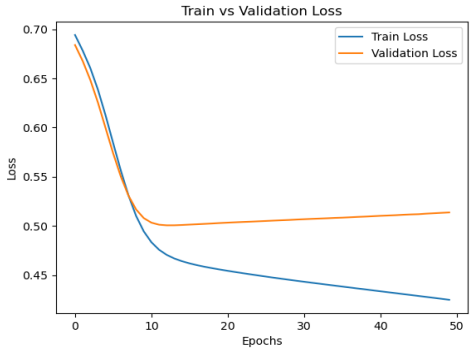


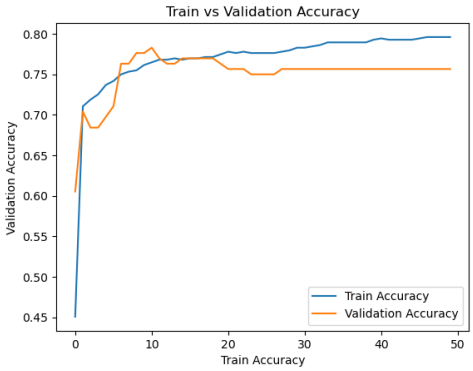


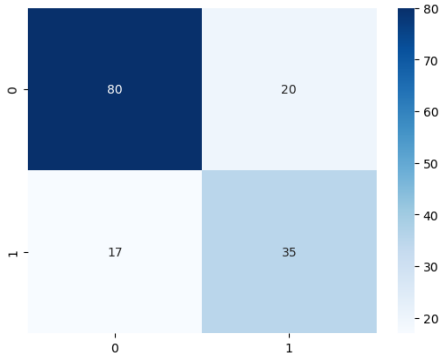


Optimizer Plots

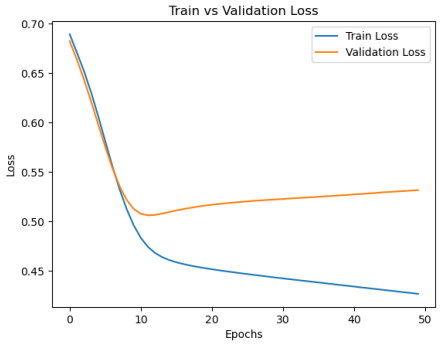
for Adam

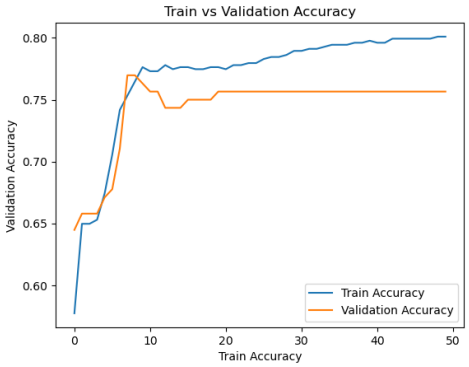


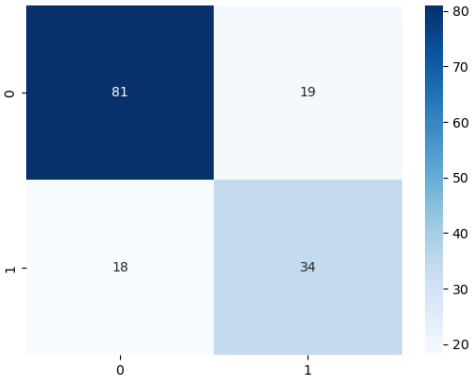




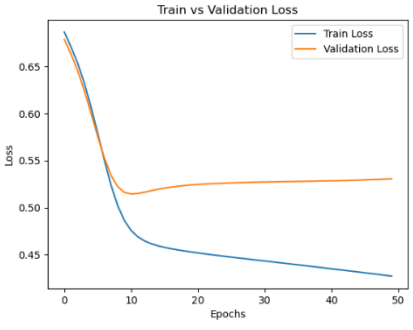
for SGD

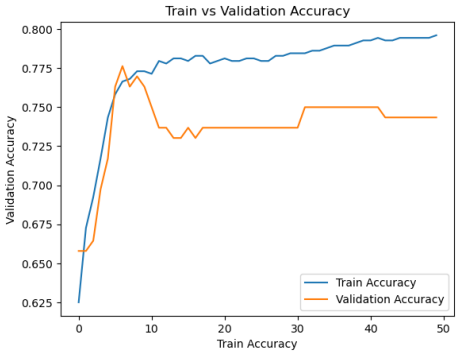


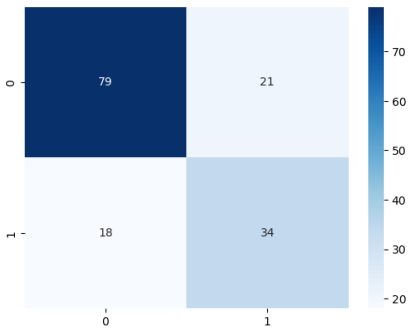




for RMSprop

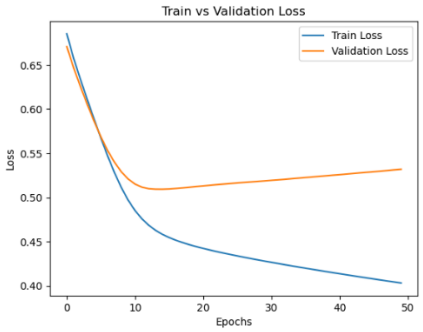


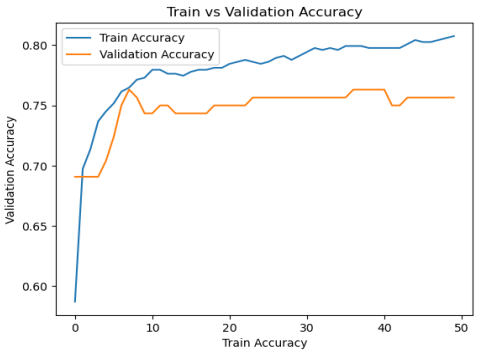


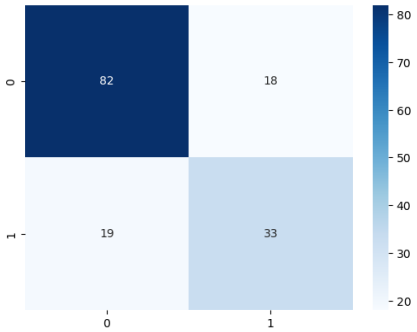


Initializer Plots

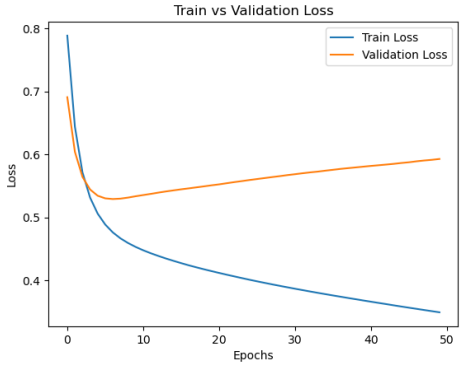
for Xavier Uniform

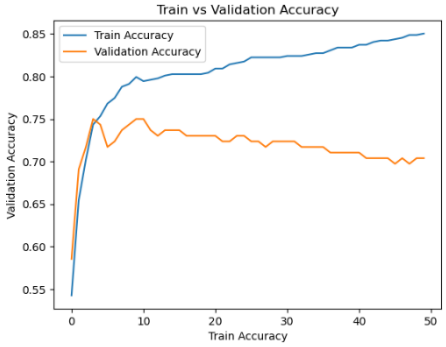


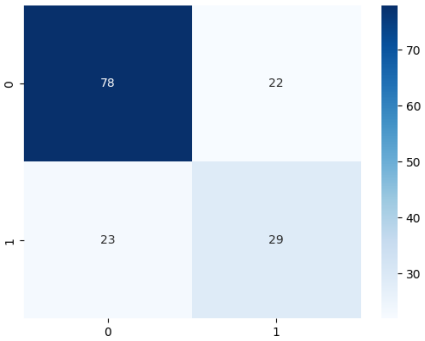




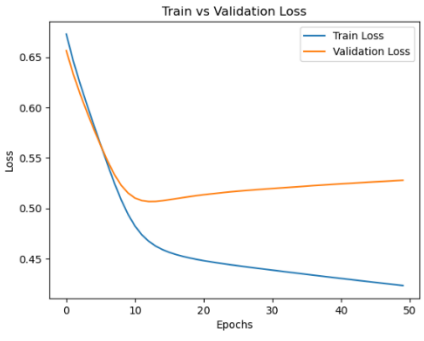
for Kaiming Uniform

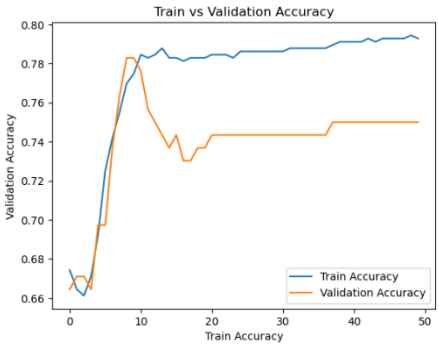


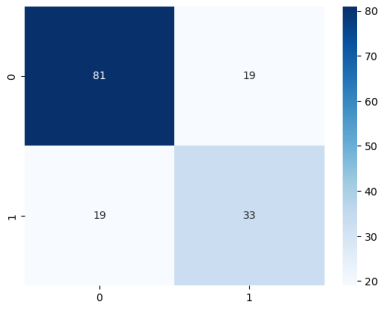




for Orthogonal

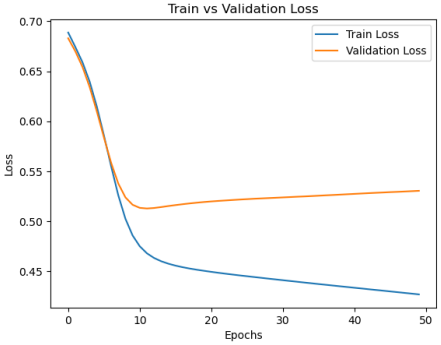


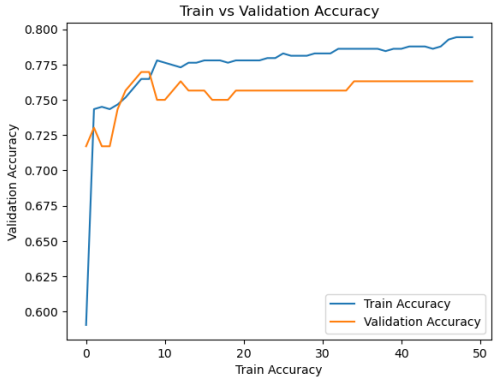


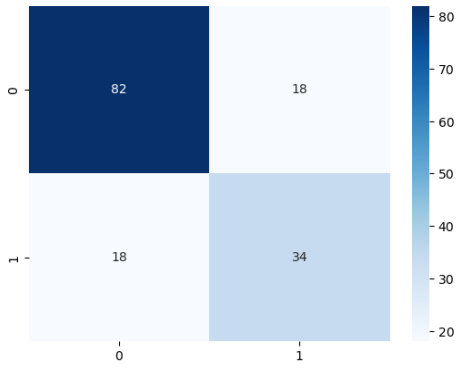


Activation Function Plots

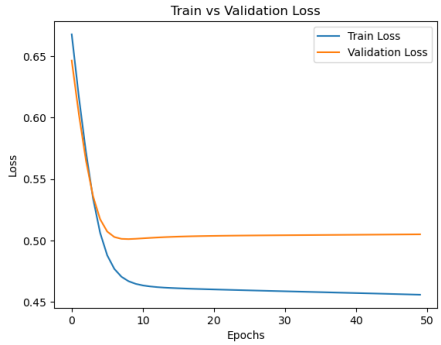
for ReLu

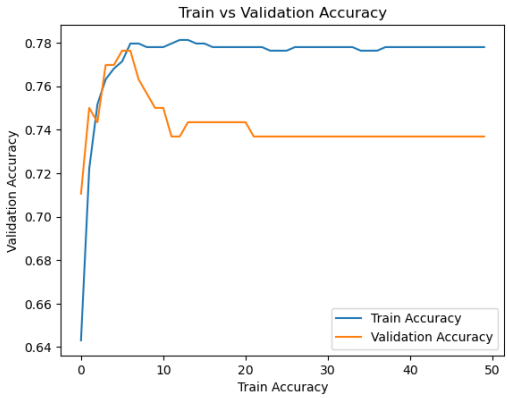


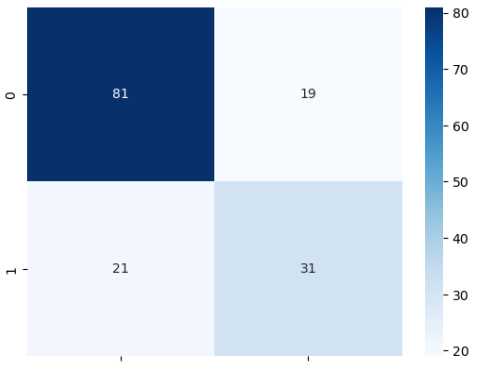




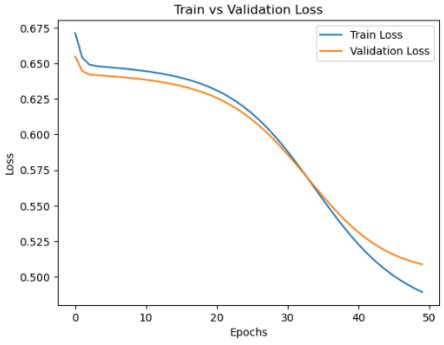
for tanh

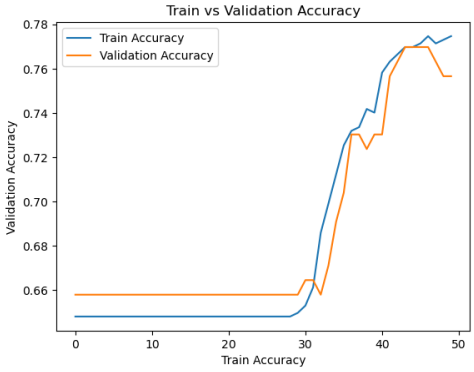


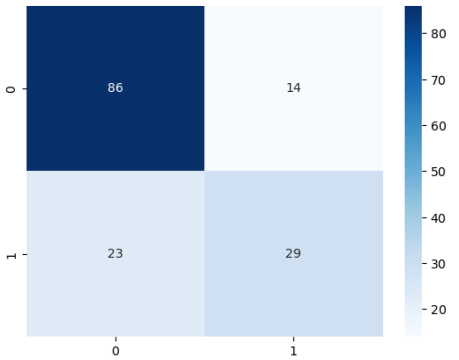




for Sigmoid







Analysis of Neural Network Setups

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

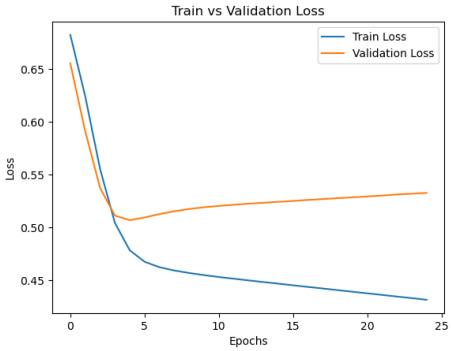
In all the tuning models the basic architecture, epochs, batch size and learning rate was kept the same to see the difference in impact on the accuracy of the models.

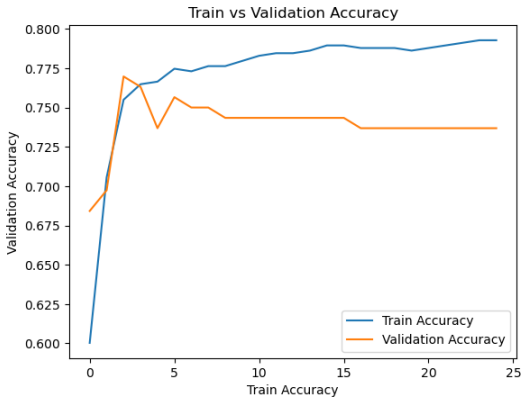
So the ReLu Activation function model performed the best and is chosen as our base model for the further analysis.

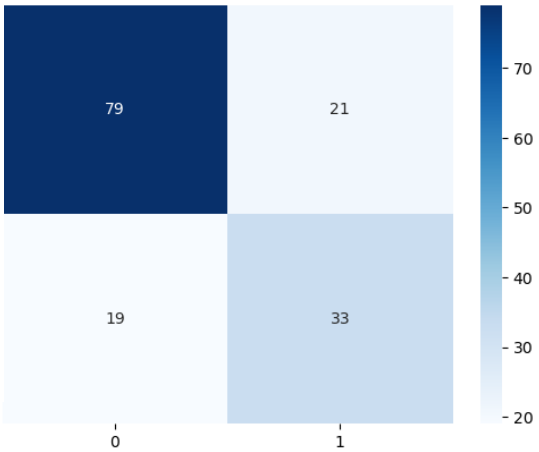
Methods used to improve the accuracy of the Model

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.

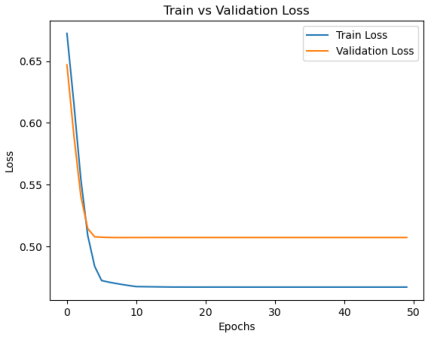


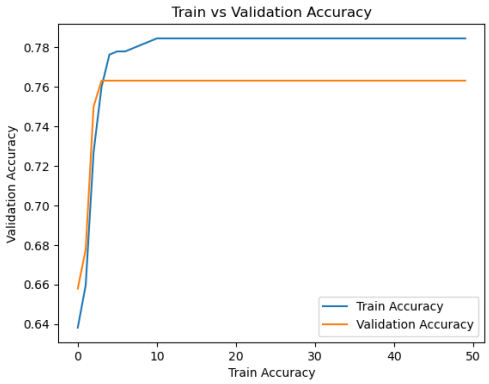


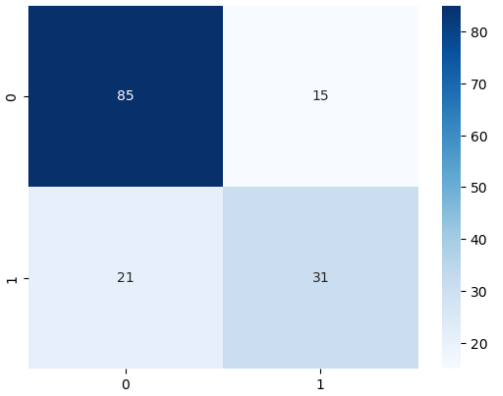


Learning Rate Scheduler

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.

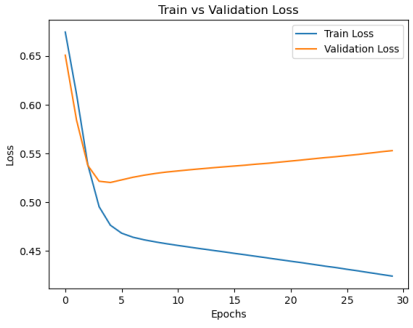


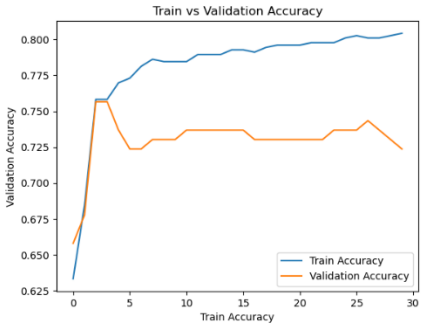


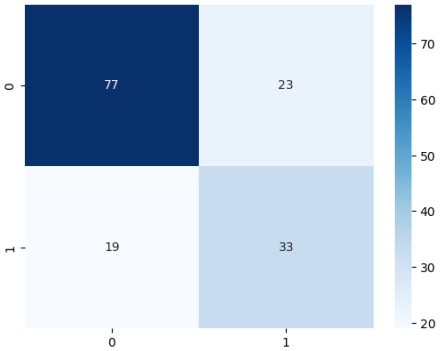


Gradient Clipping

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.

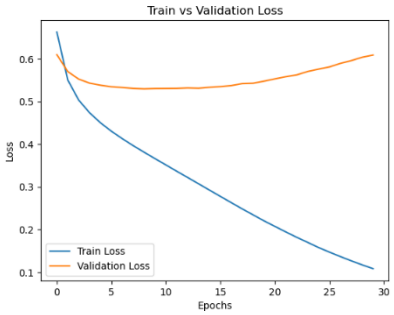


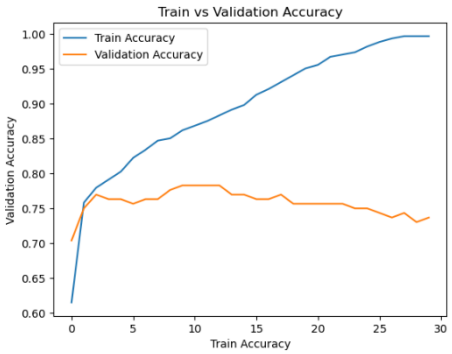


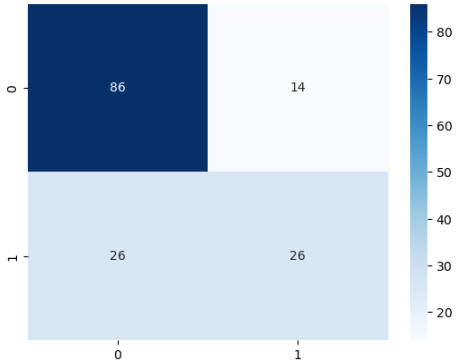


Batch Normalization

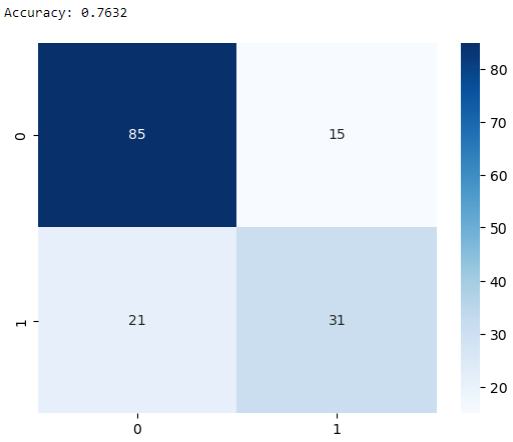
Deep learning uses batch normalization as a method to enhance the performance and stability of neural networks. Each layer's input values are standardized to have a zero mean and unit variance before being scaled and shifted using learnable parameters.







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



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