https://drive.google.com/drive/folders/1YunTPWbJuRDbeQHBbe0qU23xpu9QIDQH?usp=sharing

In [3]:

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import torch.optim as optim
import pickle
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings("ignore")
import sys
with open('sentence_matrices_test.pkl', 'rb') as file:
        sentence_matrices_test = pickle.load(file)
with open('tfidf_vectorizer_Ytest.pkl', 'rb') as file:
    y_test = pickle.load(file)
tensor_test = torch.tensor(sentence_matrices_test, dtype=torch.float32)
labels_tensor_test = torch.tensor(y_test, dtype=torch.long)
size = len(tensor_test[0])
class CustomDatasetDNN(Dataset):
    def __init__(self, tfidf, labels):
        self.tfidf = tfidf
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return self.tfidf[idx], self.labels[idx]
class CustomDatasetCNN(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype = torch.float32).unsqueeze(1)
        self.y = torch.tensor(y, dtype = torch.long)
    def __len__(self):
        return len(self.y)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
class CustomDatasetLSTM(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype = torch.float32)
        self.y = torch.tensor(y, dtype = torch.long)
    def __len__(self):
        return len(self.y)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers):
        super(NeuralNetwork, self).__init__()
        layers = []
        for _ in range(num_layers):
            if len(layers) == 0:
                layers.append(nn.Linear(input_size, hidden_size))
                # layers.append(nn.BatchNorm1d(hidden_size))
                # layers.append(nn.Dropout(dropout))
                layers.append(nn.ReLU())
            else:
                layers.append(nn.Linear(hidden_size, hidden_size))
                # layers.append(nn.BatchNorm1d(hidden_size))
                # layers.append(nn.Dropout(dropout))
                layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_size, output_size))
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
import torch
import torch.nn as nn

class Network(nn.Module):
    def __init__(self, input_channel=1, output_dim=2):
        super(Network, self).__init__()
        self.conv_1 = nn.Conv2d(input_channel, 32, (7, 7), stride=5)
        self.activation_1 = nn.ReLU()

        self.conv_2 = nn.Conv2d(32,64, (5, 5), stride=5)
        self.activation_2 = nn.ReLU()

        self.calculate_conv_sizes()

        self.flatten = nn.Flatten()
        self.dropout = nn.Dropout(0.5)
        self.linear_3 = nn.Linear(64* self.new_height * self.new_width, o
        self.activation_3 = nn.Sigmoid()

    def calculate_conv_sizes(self):
        with torch.no_grad():
            dummy_input = torch.randn(1, 1, size, 100)
            conv1_out = self.conv_1(dummy_input)
            conv1_out = self.activation_1(conv1_out)

            conv2_out = self.conv_2(conv1_out)
            conv2_out = self.activation_2(conv2_out)

            self.new_height, self.new_width = conv2_out.size(2), conv2_ou

    def forward(self, x):
        out = self.conv_1(x)
        out = self.activation_1(out)
        # out = self.batch_norm_1(out)
        # out = self.pool1(out)

        out = self.conv_2(out)
        out = self.activation_2(out)
        # out = self.batch_norm_2(out)
        # out = self.pool2(out)

        out = self.flatten(out)
        # out = self.dropout(out)
        out = self.linear_3(out)
        out = self.activation_3(out)
        return out
class LSTM(nn.Module):
    def __init__(self,embed_dim):
        super().__init__()
        self.lstm = nn.LSTM(embed_dim,64,num_layers=1,bidirectional=True,
        self.fc = nn.Linear(128,2)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        x,(hidden_state,cell_state)=self.lstm(x)
        hidden=torch.cat((hidden_state[-2,:,:], hidden_state[-1,:,:]), di
        x=self.fc(hidden)
        x=self.sigmoid(x)
        return x
def callDNN(X_test, Y_test):

    with open(X_test, 'rb') as file:
        tfidf_vectorizer_test = pickle.load(file)
    with open(Y_test, 'rb') as file:
        y_test = pickle.load(file)
    with open('DNN_Model.pkl', 'rb') as file:
        final_model = pickle.load(file)
    tfidf_tensor_test = torch.tensor(tfidf_vectorizer_test.todense(), dty
    labels_tensor_test = torch.tensor(y_test, dtype=torch.long)
    dataset = CustomDatasetDNN(tfidf_tensor_test, labels_tensor_test)
    test_data_loader = DataLoader(dataset, batch_size=32, shuffle=True)
    correct = 0
    total = 0
    y_pred = []
    y_test = []
    for x, y in test_data_loader:
        # x = x.to(device)
        with torch.no_grad():
            yp = final_model(x)
        yp = torch.argmax(yp.cpu(), dim = 1)
        y_pred += yp
        y_test += y
        correct += (yp == y).sum()
        total += len(y)
    print("---------------DNN Model Accuracy:---------------")
    print(f"Accuracy on Test Data {(correct * 100 / total):.2f}")
    # Generate classification report
    print(classification_report(y_test, y_pred))

def callCNN(X_test, Y_test):
    with open(X_test, 'rb') as file:
        sentence_matrices_test = pickle.load(file)
    with open(Y_test, 'rb') as file:
        y_test = pickle.load(file)
    with open('CNN_Model.pkl', 'rb') as file:
        final_model = pickle.load(file)
    tensor_test = torch.tensor(sentence_matrices_test, dtype=torch.float3
    labels_tensor_test = torch.tensor(y_test, dtype=torch.long)
    dataset = CustomDatasetCNN(tensor_test, labels_tensor_test)
    test_data_loader = DataLoader(dataset, batch_size=32, shuffle=True)
    correct = 0
    total = 0
    y_pred = []
    y_test = []
    for x, y in test_data_loader:
        with torch.no_grad():
            yp = final_model(x)
        yp = torch.argmax(yp.cpu(), dim = 1)
        y_pred += yp
        y_test += y
        correct += (yp == y).sum()
        total += len(y)
    #     print(yp)
    #     print(y)
    print("---------------CNN Model Accuracy:---------------")
    print(f"Accuracy on test Data {(correct * 100 / total):.2f}")
    print(classification_report(y_test, y_pred))

def callLSTM(X_test, Y_test):
    with open(X_test, 'rb') as file:
        sentence_matrices_test = pickle.load(file)
    with open(Y_test, 'rb') as file:
        y_test = pickle.load(file)
    tensor_test = torch.tensor(sentence_matrices_test, dtype=torch.float3
    labels_tensor_test = torch.tensor(y_test, dtype=torch.long)
    dataset = CustomDatasetLSTM(tensor_test, labels_tensor_test)
    test_data_loader = DataLoader(dataset, batch_size=10, shuffle=True)
    with open('LSTM_Model.pkl', 'rb') as file:
        final_model = pickle.load(file)
    correct = 0
    total = 0
    y_pred = []
    y_test = []
    for x, y in test_data_loader:
        # x = x.to(device)
        with torch.no_grad():
            yp = final_model(x)
        # print(yp)
        yp = torch.argmax(yp.cpu(), dim = 1)
        y_pred += yp
        y_test += y
        correct += (yp == y).sum()
        # print(yp)
        # print(y)
        total += len(y)
    print("---------------LSTM Model Accuracy:---------------")
    print(f"Accuracy on test Data {(correct * 100 / total):.2f}")
    print(classification_report(y_test, y_pred))

# if len(sys.argv) != 4:
#     print("Usage: python3 Model_Name(DNN,CNN,LSTM) X_testFile.pkl Y_tes
#     sys.exit(1)

# modelname = sys.argv[1]
# X_test = sys.argv[2]
# Y_test = sys.argv[3]
# modelname = 'DNN'
# X_test = 'tfidf_matrix_new_test.pkl'
# Y_test = 'y_new_test.pkl'
# if modelname == "DNN":
#     callDNN(X_test,Y_test)
# elif modelname =="CNN":
#     callCNN(X_test,Y_test)
# elif modelname == "LSTM":
#     callLSTM(X_test,Y_test)
# else:
#     print("Incorrect Model Name (Enter DNN, CNN or LSTM)")
callDNN('tfidf_vectorizer_Xtest.pkl','tfidf_vectorizer_Ytest.pkl')
```

```
callCNN('sentence_matrices_test.pkl','tfidf_vectorizer_Ytest.pkl')
callLSTM('sentence_matrices_test.pkl','tfidf_vectorizer_Ytest.pkl')
```

```
---------------DNN Model Accuracy:---------------
Accuracy on Test Data 94.72
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       503
           1       0.94      0.96      0.95       557

    accuracy                           0.95      1060
   macro avg       0.95      0.95      0.95      1060
weighted avg       0.95      0.95      0.95      1060

---------------CNN Model Accuracy:---------------
Accuracy on test Data 92.45
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       503
           1       0.92      0.93      0.93       557

    accuracy                           0.92      1060
   macro avg       0.92      0.92      0.92      1060
weighted avg       0.92      0.92      0.92      1060

---------------LSTM Model Accuracy:---------------
Accuracy on test Data 91.23
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       503
           1       0.94      0.89      0.91       557

    accuracy                           0.91      1060
   macro avg       0.91      0.91      0.91      1060
weighted avg       0.91      0.91      0.91      1060
```

In [ ]:

In [ ]: