DSC 43C8 – Final Project

Courtney Hodge and Tara Williams

Agenda

Evaluating Reproducibility of Deep Learning Models

Model Comparisons and Results

PHATE, t-SNE, PCA + t-SNE Visualizations for Credit Analytics

Applying CNN, LSTM, and GRU Classification to Credit Analytics

Model Comparisons and Results

Part A: Evaluate the Reproducibility of deep learning

Investigating mnist, emodb, and kolod datasets

Datasets & CLR

CORRECT LEARNING REPRODUCIBILITY (CLR)

- The ratio that a model correctly does classification after n runs. Here we can define each run has 100 epochs. CLR can be represented as the average classification ratios of n runs

EMODB

- A well-known dataset in the field of speech emotion recognition, consisting of 535 short phrases with 7 different emotional tones (N,A,B,D,An, H, S).

KOLOD

- A publicly available dataset of single-cell RNA sequencing (scRNA-seq) data. It has 704 samples across 3 classes: embryonic stem cells, neural progenitor cells, and neural crest cells. (0,1,2)

MNIST

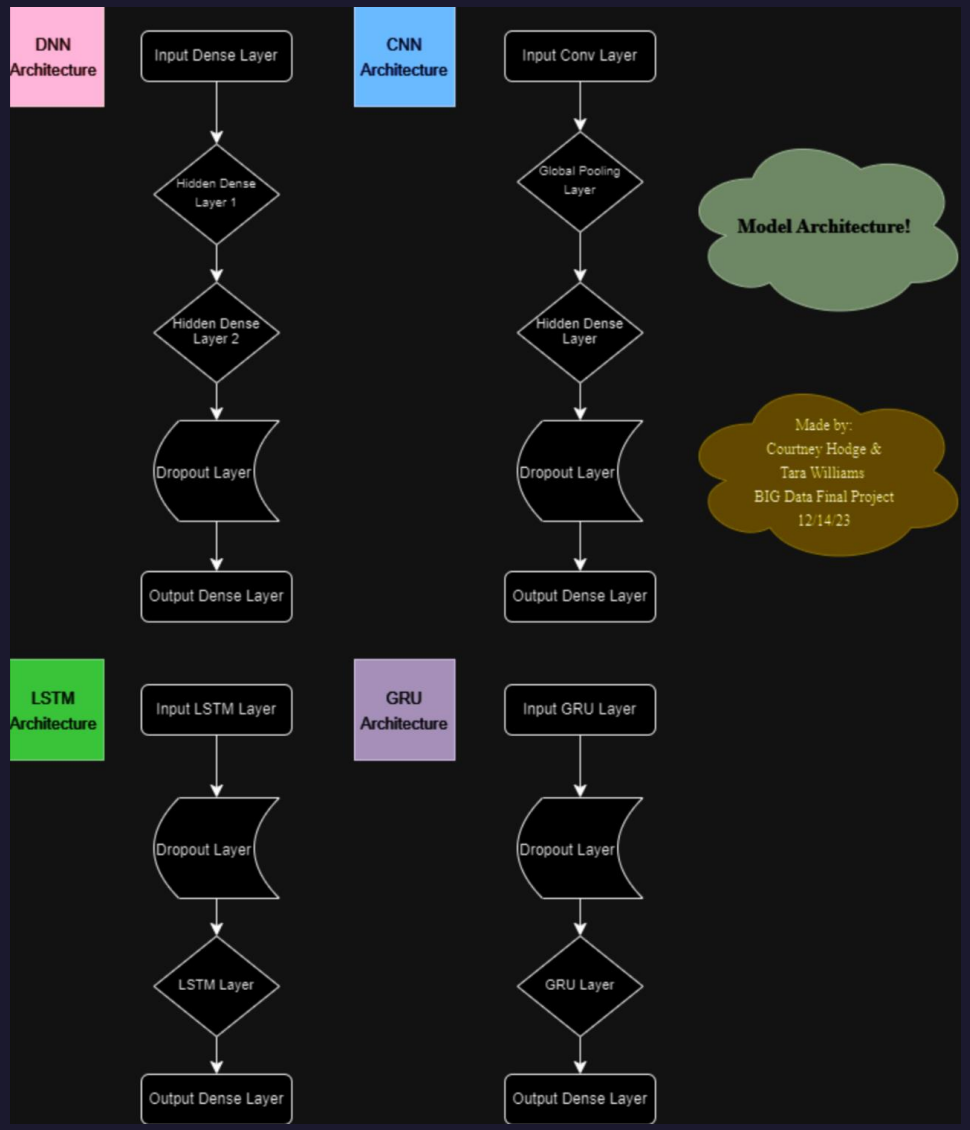
- As seen in project 2, it is a dataset comprised of images of handwritten digits (0-9) that is commonly used for machine learning models.

Initial Steps

```
########################
#####Load the Data#####
########################
emodb = pd.read_csv('/content/drive/MyDrive/Baylor F23 - S24/1-Fall/BIG data/EMODB.csv')

#mnist
from keras.datasets import mnist

#kolod
kolod = pd.read_csv('/content/drive/MyDrive/Baylor F23 - S24/1-Fall/BIG data/Kolod.csv')
```

A) Use DNN, CNN, LSTM, GRU models

Model's Architecture



B) Run each model - EMODB

EMODB

> Everything should work

```
[59] ############
     ####emodb###
     ############
     n = 5
     emodb_DNN_accuracy = 0
     for i in range(5):
       emodb_DNN_accuracy += classification_measures(emodb_DNN_model, emodb_x_train, emodb_x_test, emodb_y_train, emodb_y_test)
     emodb_DNN_CLR = emodb_DNN_accuracy/n

     emodb_CNN_accuracy = 0
     for i in range(5):
       emodb_CNN_accuracy += classification_measures(emodb_CNN_model, emodb_x_train, emodb_x_test, emodb_y_train, emodb_y_test)
     emodb_CNN_CLR = emodb_CNN_accuracy/n

     emodb_LSTM_accuracy = 0
     for i in range(5):
       emodb_LSTM_accuracy += classification_measures(emodb_LSTM_model, emodb_x_train, emodb_x_test, emodb_y_train, emodb_y_test)
     emodb_LSTM_CLR = emodb_LSTM_accuracy/n

     emodb_GRU_accuracy = 0
     for i in range(5):
       emodb_GRU_accuracy += classification_measures(emodb_GRU_model, emodb_x_train, emodb_x_test, emodb_y_train, emodb_y_test)
     emodb_GRU_CLR = emodb_GRU_accuracy/n

     print("Emodb DNN CLR: ", emodb_DNN_CLR)
     print("Emodb CNN CLR: ",emodb_CNN_CLR)
     print("Emodb LSTM CLR: ",emodb_LSTM_CLR)
     print("Emodb GRU CLR: ",emodb_GRU_CLR)
```

B) Run each model - MNIST

Mnist

Everything should work

```
]  ############
   #--mnist---#
   ############

   mnist_DNN_model = DNN("mnist", x_train)
   mnist_CNN_model = CNN("mnist",x_train)
   mnist_LSTM_model = LSTM("mnist",x_train)
   mnist_GRU_model = GRU("mnist",x_train)

   mnist_DNN_accuracy = 0
   for i in range(5):
     mnist_DNN_accuracy += mnist_class_measures(mnist_DNN_model, x_train, x_test, y_train, y_test, y_train_label)
   mnist_DNN_CLR = mnist_DNN_accuracy/n

   mnist_CNN_accuracy = 0
   for i in range(5):
     mnist_CNN_accuracy += mnist_class_measures(mnist_CNN_model, x_train, x_test, y_train, y_test, y_train_label)
   mnist_CNN_CLR = mnist_CNN_accuracy/n

   mnist_LSTM_accuracy = 0
   for i in range(5):
     mnist_LSTM_accuracy += mnist_class_measures(mnist_LSTM_model, x_train, x_test, y_train, y_test, y_train_label)
   mnist_LSTM_CLR = mnist_LSTM_accuracy/n

   mnist_GRU_accuracy = 0
   for i in range(5):
     mnist_GRU_accuracy += mnist_class_measures(mnist_GRU_model, x_train, x_test, y_train, y_test, y_train_label)
   mnist_GRU_CLR = mnist_GRU_accuracy/n

   print("mnist DNN CLR: ",mnist_DNN_CLR)
   print("mnist DNN CLR: ",mnist_CNN_CLR)
   print("mnist DNN CLR: ",mnist_LSTM_CLR)
   print("mnist DNN CLR: ",mnist_GRU_CLR)
```

B) Run each model – KOLOD

KOLOD

DNN and CNN used to work, now nothing does

LSTM

GRU

DNN

CNN

```
5] ############
   #--kolod---#
   # ############
   # n= 5

   # kolod_DNN_model = DNN("kolod", x_train)
   # kolod_CNN_model = CNN("kolod",x_train)
   # kolod_LSTM_model = LSTM("kolod",x_train)
   # kolod_GRU_model = GRU("kolod",x_train)

   # kolod_DNN_accuracy = 0
   # for i in range(5):
   #   kolod_DNN_accuracy += classification_measures(kolod_DNN_model, kolod_x_train, kolod_x_test, kolod_y_train, kolod_y_test)
   # kolod_DNN_CLR = kolod_DNN_accuracy/n


   # kolod_CNN_accuracy = 0
   # for i in range(5):
   #   kolod_CNN_accuracy += classification_measures(kolod_CNN_model, kolod_x_train, kolod_x_test, kolod_y_train, kolod_y_test)
   # kolod_CNN_CLR = kolod_CNN_accuracy/n


   # # #got stuck on lstm
   # # kolod_LSTM_accuracy = 0
   # # for i in range(5):
   # #  kolod_LSTM_accuracy += classification_measures(kolod_LSTM_model, kolod_x_train, kolod_x_test, kolod_y_train, kolod_y_test)
   # # kolod_LSTM_CLR = kolod_LSTM_accuracy/n


   # # kolod_GRU_accuracy = 0
   # # for i in range(5):
   # #  kolod_GRU_accuracy += classification_measures(kolod_GRU_model, kolod_x_train, kolod_x_test, kolod_y_train, kolod_y_test)
   # # kolod_GRU_CLR = kolod_GRU_accuracy/n

   # print("Kolod DNN CLR: ", kolod_DNN_CLR)
   # print("Kolod CNN CLR: ", kolod_CNN_CLR)
   # print("Kolod LSTM CLR: ", kolod_LSTM_CLR)
   # print("Kolod GRU CLR: ", kolod_GRU_CLR)
```

Results

EMODB

```
#----------#
#D-index: 0.0199
#----------#

Emodb DNN CLR:  0.07476635514018691
Emodb CNN CLR:  0.07476635514018691
Emodb LSTM CLR:  0.07476635514018691
Emodb GRU CLR:  0.07476635514018691
```

MNIST

```
#----------#
#D-index: 0.0151
#----------#

mnist DNN CLR:  0.37383177570093457
mnist DNN CLR:  0.37383177570093457
mnist DNN CLR:  0.37383177570093457
mnist DNN CLR:  0.37383177570093457
```

KOLOD

```
# print("Kolod DNN CLR: ", kolod_DNN_CLR)
# print("Kolod CNN CLR: ", kolod_CNN_CLR)
# print("Kolod LSTM CLR: ", kolod_LSTM_CLR)
# print("Kolod GRU CLR: ", kolod_GRU_CLR)
```

1C) Do PCA for each dataset - EMODB

Emodb PCA

Everything should work

```
#apply pca
pca = PCA(n_components = 10)
emodb_x_train_pca = pca.fit_transform(emodb_x_train)
emodb_x_test_pca = pca.transform(emodb_x_test)

#get models
emodb_DNN_PCA_model = DNN("emodb", emodb_x_test_pca)
emodb_CNN_PCA_model = CNN("emodb", emodb_x_test_pca)
emodb_LSTM_PCA_model = LSTM("emodb", emodb_x_test_pca)
emodb_GRU_PCA_model = GRU("emodb", emodb_x_test_pca)

#get CLRs
emodb_DNN_PCA_accuracy = 0
for i in range(5):
  emodb_DNN_PCA_accuracy += classification_measures(emodb_DNN_PCA_model, emodb_x_train_pca,
                                                    emodb_x_test_pca, emodb_y_train, emodb_y_test)
emodb_DNN_PCA_CLR = emodb_DNN_PCA_accuracy/n

emodb_CNN_PCA_accuracy = 0
for i in range(5):
  emodb_CNN_PCA_accuracy += classification_measures(emodb_CNN_PCA_model, emodb_x_train_pca,
                                                    emodb_x_test_pca, emodb_y_train, emodb_y_test)
emodb_CNN_PCA_CLR = emodb_CNN_PCA_accuracy/n

emodb_LSTM_PCA_accuracy = 0
for i in range(5):
  emodb_LSTM_PCA_accuracy += classification_measures(emodb_LSTM_PCA_model, emodb_x_train_pca,
                                                     emodb_x_test_pca, emodb_y_train, emodb_y_test)
emodb_LSTM_PCA_CLR = emodb_LSTM_PCA_accuracy/n

emodb_GRU_PCA_accuracy = 0
for i in range(5):
  emodb_GRU_PCA_accuracy += classification_measures(emodb_GRU_PCA_model, emodb_x_train_pca,
                                                    emodb_x_test_pca, emodb_y_train, emodb_y_test)
emodb_GRU_PCA_CLR = emodb_GRU_PCA_accuracy/n

print("Emodb DNN CLR: ", emodb_DNN_PCA_CLR)
print("Emodb CNN CLR: ", emodb_CNN_PCA_CLR)
print("Emodb LSTM CLR: ", emodb_LSTM_PCA_CLR)
print("Emodb GRU CLR: ", emodb_GRU_PCA_CLR)
```

1C) Do PCA for each dataset - MNIST

Mnist PCA

Everything should work

```
] #dnn
  def DNN_with_PCA():
    my_epochs = 10
    # Load MNIST data
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    # Flatten the images
    x_train = x_train.reshape(x_train.shape[0], -1)
    x_test = x_test.reshape(x_test.shape[0], -1)
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

    # Apply PCA for dimensionality reduction
    pca = PCA(n_components=100)  # Adjust number of components as needed
    x_train_pca = pca.fit_transform(x_train)
    x_test_pca = pca.transform(x_test)

    # Create a simple DNN model
    model = Sequential()
    model.add(Dense(128, input_dim=x_train_pca.shape[1], activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(10, activation='softmax'))  # Output layer, 10 classes

    # Compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Train the model
    model.fit(x_train_pca, y_train, epochs=my_epochs, batch_size=128, validation_data=(x_test_pca, y_test))

    # Evaluate the model
    y_pred_prob = model.predict(x_test_pca)
    y_pred = y_pred_prob.argmax(axis=1)

    # Calculate accuracy
    accuracy = accuracy_score(y_test.argmax(axis=1), y_pred)

    # Calculate D-index
    d_index = 1 - accuracy
    print("#---------#")
    print("d_index:", d_index)
    print("#---------#")


    return accuracy

  # Call the function
  accuracy = DNN_with_PCA()
  print(f"Accuracy with PCA and DNN: {accuracy}")
```

1C) Do PCA for each dataset – MNIST cont.

```python
def CNN_with_PCA():
    # Load MNIST data
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    # Reshape and normalize the data
    x_train = x_train.reshape(x_train.shape[0], -1)
    x_test = x_test.reshape(x_test.shape[0], -1)
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

    # Apply PCA for dimensionality reduction on flattened images
    pca = PCA(n_components=100)
    x_train_flatten = x_train.reshape(x_train.shape[0], -1)
    x_test_flatten = x_test.reshape(x_test.shape[0], -1)
    x_train_pca = pca.fit_transform(x_train_flatten)
    x_test_pca = pca.transform(x_test_flatten)

    # Reshape PCA-transformed data back to images
    x_train_pca = x_train_pca.reshape(-1, 10, 10, 1)
    x_test_pca = x_test_pca.reshape(-1, 10, 10, 1)

    # Create a CNN model
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(10, 10, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))  # Output layer, 10 classes

    # Compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Train the model
    model.fit(x_train_pca, y_train, epochs=5, batch_size=128, validation_data=(x_test_pca, y_test))

    # Evaluate the model
    y_pred_prob = model.predict(x_test_pca)
    y_pred = np.argmax(y_pred_prob, axis=1)

    # Calculate accuracy
    accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)

    # Calculate D-index
    d_index = 1 - accuracy

    return accuracy, d_index

# Call the function
accuracy, d_index = CNN_with_PCA()
print(f"Accuracy with PCA and CNN: {accuracy}")
print(f"D-index with PCA and CNN: {d_index}")
```

Results - PCA

EMODB

```
#----------#
#D-index: 0.0199
#----------#

Emodb DNN CLR:  0.07476635514018691
Emodb CNN CLR:  0.07476635514018691
Emodb LSTM CLR:  0.07476635514018691
Emodb GRU CLR:  0.07476635514018691
```

MNIST

What did not work:

- CNN
- LSTM
- GRU

```
#---------#
d_index: 0.050699999999999997
#---------#
Accuracy with PCA and DNN: 0.9493
```

```
Accuracy with PCA and CNN: 0.922
D-index with PCA and CNN: 0.07799999999999996
```

KOLOD

Total Issues with:

- DNN
- CNN
- LSTM
- GRU

2C) Do TSNE for each dataset - EMODB

Emodb tSNE

CNN

LSTM

GRU

Everything else should work

```
#do TSNE
tsne = TSNE()  # Adjust n_components as needed
emodb_tsne  = tsne.fit_transform(emodb_data)

#split data again
x_train, x_test, y_train, y_test = train_test_split(emodb_tsne, emodb_label, test_size=0.2, random_state=42)

#get model
emodb_DNN_TSNE_model = DNN_TSNE(emodb_tsne)

#get CLRs
accuracy = 0
for i in range(5):
  accuracy = classification_measures(emodb_DNN_TSNE_model, x_train, x_test, y_train, y_test)
emodb_DNN_TSNE_CLR = accuracy/n


print("Emodb DNN CLR: ", emodb_DNN_TSNE_CLR)
```

2C) Do TSNE for each dataset - MNIST

```
[75] (x_train_data, y_train_label), (x_test_data, y_test_label) = mnist.load_data()

    x_train = x_train_data.reshape(y_train_label.shape[0], x_train_data.shape[1], x_train_data.shape[2], 1)
    x_test = x_test_data.reshape(y_test_label.shape[0], x_test_data.shape[1], x_test_data.shape[2], 1)

    y_train = to_categorical(y_train_label)
    y_test = to_categorical(y_test_label)

    x_train_flat = x_train.reshape(-1, 28 * 28)
    x_test_flat = x_test.reshape(-1, 28 * 28)

    #perform tsne on mnist
    tsne = TSNE()
    train_tsne = tsne.fit_transform(x_train_flat)
    test_tsne = tsne.fit_transform(x_test_flat)
```

```
#TSNE
model = Sequential()

layer_1_node_number, layer_2_node_number, layer_3_node_number = 200, 100, 50

model.add(Dense(layer_1_node_number,activation = 'relu', input_shape = (train_tsne.shape[1],)))

model.add(Dense(layer_2_node_number, activation = 'relu'))

model.add(Dense(layer_3_node_number, activation = 'relu'))

dense_node_number = 10

model.add(Dense(dense_node_number, activation = 'softmax'))

my_epochs = 10

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics= ['accuracy'])

history = model.fit(train_tsne, y_train, epochs = my_epochs, batch_size = 500, validation_data= (test_tsne, y_test))

#calculate the classificaiton measures
#in order to find these, I have to fit a classifier on PCA,
#so I will be using Support Vector Machines, svm
from sklearn.svm import SVC
#from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score
```

2C) Do TSNE for each dataset - MNIST cont

```
#SVM
svm = SVC()

svm.fit(train_tsne, y_train_label)

pred_labels = svm.predict(test_tsne)


#clf = GradientBoostingClassifier()
#clf.fit(train_tsne, y_train_label)
#pred_labels = clf.predict(test_tsne)


#calculating metrics for multiclass classification via 'macro'
conf_matrix = confusion_matrix(y_test_label, pred_labels)

precision = precision_score(y_test_label, pred_labels, average = 'macro')

f1 = f1_score(y_test_label, pred_labels, average = 'macro')

accuracy = accuracy_score(y_test_label, pred_labels)

recall = recall_score(y_test_label, pred_labels, average = 'macro')

d_index = 2 * (precision * recall)/(precision + recall)

#calculate specificity
TN = conf_matrix[0][0]
FP = conf_matrix[0][1:].sum()
FN = conf_matrix[1:, 0].sum()
TP = conf_matrix[1:, 1:].sum()

specificity = TN / (TN + FP)

#displaying the metrics
print("Confusion Matrix: \n", conf_matrix)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"D-index: {d_index:.4f}")
print(f"Specificity: {specificity:.4f}")
```

Results - TSNE

EMODB

What did not work:

- CNN
- LSTM
- GRU

```
#----------#
#D-index: 0.0199
#----------#

Emodb DNN CLR:  0.0149532710280373 82
```

MNIST

What did not work:

- CNN
- LSTM
- GRU

```
Accuracy: 0.1580
Precision: 0.1491
Recall: 0.1629
F1 Score: 0.1483
D-index: 0.1557
Specificity: 0.0316
```

KOLOD

Total Issues with:

- DNN
- CNN
- LSTM
- GRU

D) Increase models to 7 layers - EMODB

EMODB 7 Layers

GRU

Everything else should work

```
#get models
emodb_DNN_7_model = DNN_7_layers("emodb", emodb_x_train)
emodb_CNN_7_model = CNN_7_layers("emodb", emodb_x_train)
emodb_LSTM_7_model = LSTM_7_layers("emodb", emodb_x_train)
emodb_GRU_7_model = GRU_7_layers("emodb", emodb_x_train)

n = 5
accuracy = 0
for i in range(5):
  accuracy += classification_measures(emodb_DNN_7_model, emodb_x_train,
                                      emodb_x_test, emodb_y_train, emodb_y_test)
emodb_DNN_7_CLR = accuracy/n

accuracy = 0
for i in range(5):
  accuracy += classification_measures(emodb_CNN_7_model, emodb_x_train,
                                      emodb_x_test, emodb_y_train, emodb_y_test)
emodb_CNN_7_CLR = accuracy/n

accuracy = 0
for i in range(5):
  accuracy += classification_measures(emodb_LSTM_7_model, emodb_x_train, e
                                      modb_x_test, emodb_y_train, emodb_y_test)
emodb_LSTM_7_CLR = accuracy/n

accuracy = 0
for i in range(5):
  accuracy += classification_measures(emodb_GRU_7_model, emodb_x_train,
                                      emodb_x_test, emodb_y_train, emodb_y_test)
emodb_GRU_7_CLR = accuracy/n

print("DNN CLR: ", emodb_DNN_7_CLR)
print("CNN CLR: ", emodb_CNN_7_CLR)
print("LSTM CLR: ", emodb_LSTM_7_CLR)
print("GRU CLR: ", emodb_GRU_7_model)
```

D) Increase models to 7 layers - MNIST

MNIST 7 Layers

Everything should work

```
#reload data
(x_train, y_train_label), (x_test, y_test_label) = mnist.load_data()

x_train_data = x_train.reshape(y_train_label.shape[0], x_train.shape[1], x_train.shape[2], 1)
x_test_data = x_test.reshape(y_test_label.shape[0], x_test.shape[1], x_test.shape[2], 1)

y_train = to_categorical(y_train_label)
y_test = to_categorical(y_test_label)

#get models
mnist_DNN_7_model = DNN_7_layers("mnist", x_train_data)
mnist_CNN_7_model = CNN_7_layers("mnist", x_train_data)
mnist_LSTM_7_model = LSTM_7_layers("mnist", x_train_data)
mnist_GRU_7_model = GRU_7_layers("mnist", x_train_data)

#get CLRs
n = 5
accuracy = 0
for i in range(5):
  accuracy += mnist_classification_measures(mnist_DNN_7_model, x_train_data,
                                            x_test_data, y_train, y_test, y_test_label)
mnist_DNN_7_model = accuracy/n

accuracy = 0
for i in range(5):
  accuracy += mnist_classification_measures(mnist_CNN_7_model, x_train_data,
                                            x_test_data, y_train, y_test, y_test_label)
mnist_CNN_7_model = accuracy/n

accuracy = 0
for i in range(5):
  accuracy += mnist_classification_measures(mnist_LSTM_7_model, x_train_data,
                                            x_test_data, y_train, y_test, y_test_label)
mnist_LSTM_7_model = accuracy/n

accuracy = 0
for i in range(5):
  accuracy += GRU_7_layers_classification(mnist_GRU_7_model, x_train_data,
                                          y_train, x_test_data, y_test)
mnist_GRU_7_model = accuracy/n
```

D) Increase models to 7 layers - KOLOD

KOLOD 7 Layers

GRU

LSTM

CNN

DNN

```
#get models
kolod_DNN_7_model = DNN_7_layers("kolod", kolod_x_train)
kolod_CNN_7_model = CNN_7_layers("kolod", kolod_x_train)
# kolod_LSTM_7_model = LSTM_7_layers("kolod", kolod_x_train)
# kolod_GRU_7_model = GRU_7_layers("kolod", kolod_x_train)

n = 5
accuracy = 0
for i in range(5):
  accuracy += classification_measures(kolod_DNN_7_model, kolod_x_train,
                                      kolod_x_test, kolod_y_train, kolod_y_test)
kolod_DNN_7_CLR = accuracy/n

# accuracy = 0
# for i in range(5):
#   accuracy += classification_measures(kolod_CNN_7_model, emodb_x_train,
#                                       emodb_x_test, emodb_y_train, emodb_y_test)
# kolod_CNN_7_CLR = accuracy/n

# accuracy = 0
# for i in range(5):
#   accuracy += classification_measures(kolod_LSTM_7_model, emodb_x_train,
#                                       emodb_x_test, emodb_y_train, emodb_y_test)
# kolod_LSTM_7_CLR = accuracy/n

# accuracy = 0
# for i in range(5):
#   accuracy += classification_measures(kolod_GRU_7_model, emodb_x_train,
#                                       emodb_x_test, emodb_y_train, emodb_y_test)
# kolod_GRU_7_CLR = accuracy/n

print("DNN CLR: ", kolod_DNN_7_CLR)
#print("CNN CLR: ", kolod_CNN_7_CLR)
# print("LSTM CLR: ", kolod_LSTM_7_CLR)
# print("GRU CLR: ", kolod_GRU_7_CLR)
```

Results

EMODB

What did not work:

- GRU

```
#----------#
#D-index: 0.0199
#----------#

DNN CLR:  0.07476635514018691
CNN CLR:  0.07476635514018691
LSTM CLR:  0.07476635514018691
GRU CLR:  <keras.src.engine.sequential.Sequential object at 0x7b28ece1fcd0>
```

MNIST

```
Accuracy: 0.9853
D-index: 0.014700000000000046
DNN:  0.1135
CNN:  0.9836199999999999
LSTM:  0.8827999999999999
GRU:  0.9846199999999999
```

KOLOD

```
#----------#
#D-index: nan
#----------#

DNN CLR:  0.0
```

D) Do PCA before DNN, CNN, and LSTM

This is just the same thing as Part C!

This concludes the first half

Part B: Deep Learning with Credit Risk Analytics

Visualizing PHATE, t-SNE, and PCA + t-SNE Embedded data for Credit Risk data

Applying CNN, LSTM, and GRU Classification

Introduction to Deep Learning Models

CONVOLUTIONAL NEURAL NETWORKS (CNN)

- A feed-forward neural network, ideal for image, audio, and speech data classification; the issue of vanishing and exploding gradients are prevented through weights present in the model's algorithm

LONG SHORT-TERM MEMORY (LSTM)

- A recurrent neural network (RNN) designed to deal with the issue of vanishing gradients in traditional RNNs, capable of long-term dependencies

GATED RECURRENT UNIT (GRU)

- A RNN similar to LSTM due to its use of gates to control the flow of information; its gating mechanisms are used to selectively update the hidden state of the network at each time step

Overview of Resampling Methods

RANDOMOVERSAMPLER()

- The most naïve strategy to generate new samples, generates new samples by randomly sampling with replacement the currently available samples

SYNTHETIC MINORITY OVERSAMPLING TECHNIQUE

- One of the most commonly used oversampling methods, aims to balance class distribution by randomly increasing minority class examples by replicating them

ADAPTIVE SYNTHETIC (ADASYN)

- Similar to SMOTE but generates a different number of samples depending on an estimate of the local distribution of the class to be oversampled

SMOTEENN()

- Oversampling using SMOTE, cleaning using Edited Nearest Neighbors (ENN)

Overview of Resampling Methods

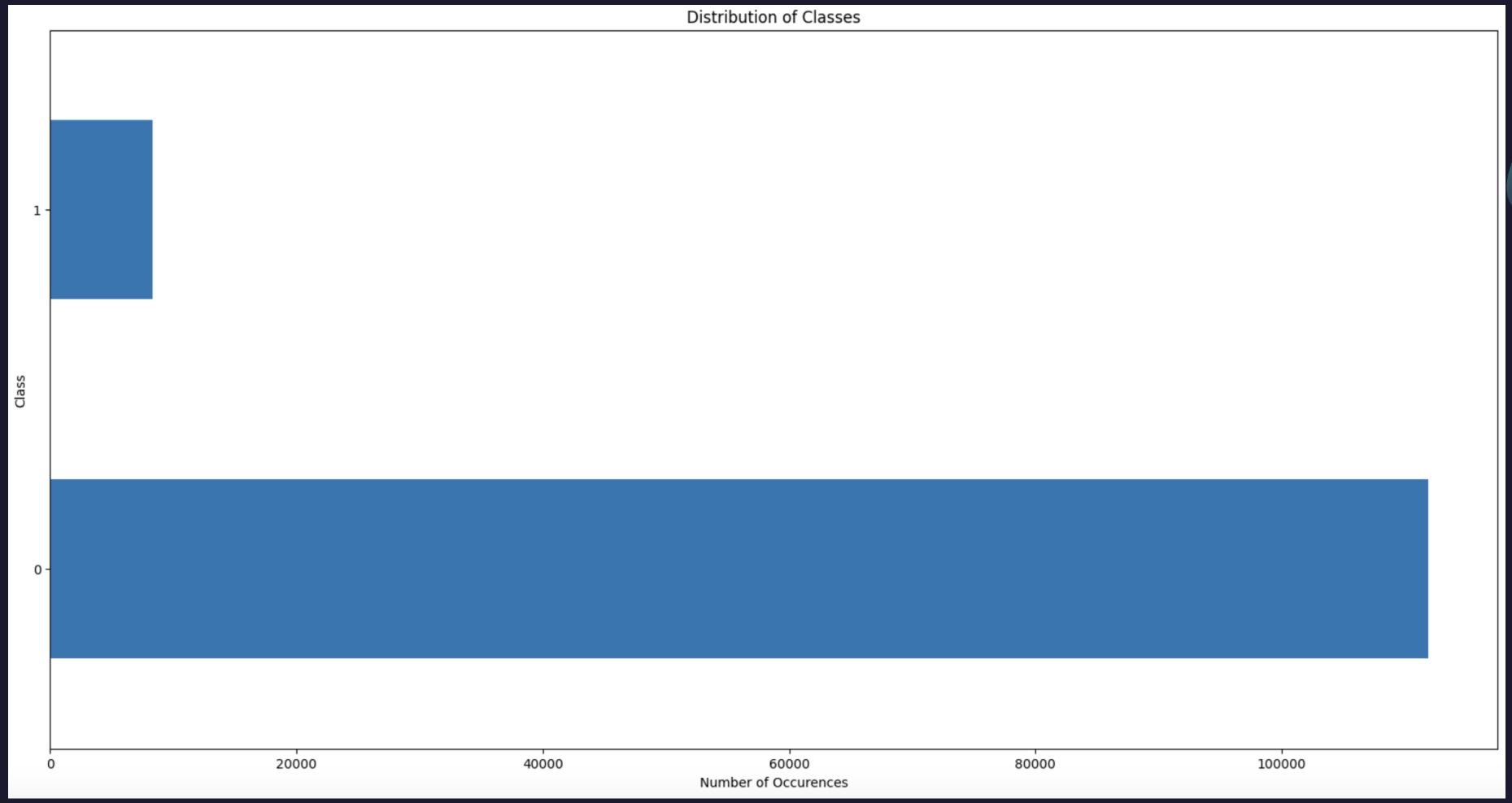
TOMEKLINKS()

- An under-sampling technique developed by Ivan Tomek, used to find samples from the majority class having the lowest Euclidean distance with the minority class data and then removing it

SMOTETOMEK()

- Oversampling using SMOTE and cleaning using the aforementioned TomekLinks technique

Overview of Credit Risk Data



LSTM & GRU Helper Functions

```python
def apply_lstm(training_data, training_labels, testing_data, testing_labels):

  # Reshaping data for LSTM
  num_features = training_data.shape[1]

  model = Sequential()
  model.add(LSTM(100, activation = "relu", input_shape = (num_features, 1)))

  # Output layer = 1, for two classes represented in data (classes 0 and 1)
  model.add(Dense(1, activation = "sigmoid"))

  # Compile model
  model.compile(optimizer = "adam", loss = "binary_crossentropy")

  # Train Model
  history = model.fit(training_data, training_labels, epochs = 50, validation_split = 0.2)

  # Collect training and testing loss history for model
  lstm_training_losses = history.history["loss"]
  lstm_testing_losses = history.history["val_loss"]

  # Make predictions for classification metrics
  predicted_labels = model.predict(testing_data)

  return lstm_training_losses, lstm_testing_losses, predicted_labels
```

```python
def apply_gru(training_data, training_labels, testing_data, testing_labels):

  num_features = training_data.shape[1]

  model = Sequential()

  model.add(GRU(32, activation = "relu", input_shape = (num_features, 1)))

  # Output layer - 1 for two classes represented in the data
  model.add(Dense(1, activation = "sigmoid"))

  # Compile model
  model.compile(optimizer = "adam", loss = "binary_crossentropy")

  # Train model
  history = model.fit(training_data, training_labels, epochs = 50, validation_split = 0.2)

  # Collect training and testing loss history for model
  gru_training_losses = history.history["loss"]
  gru_testing_losses = history.history["val_loss"]


  # Make predictions for classification metrics
  predicted_labels = model.predict(testing_data)

  return gru_training_losses, gru_testing_losses, predicted_labels
```

CNN Helper Function

```python
def apply_cnn(training_data, training_labels, testing_data, testing_labels):

  # to_categorical
  training_labels = to_categorical(training_labels)
  testing_labels = to_categorical(testing_labels)

  # Assuming your data has 10 features
  input_shape = (10, 1)

  # Build model
  model = Sequential()

  # Add first convolutional layer
  model.add(Conv1D(32, kernel_size = 3, activation = 'relu', input_shape = input_shape))
  model.add(MaxPooling1D(pool_size = 2))

  # Flatten layer
  model.add(Flatten())

  model.add(Dense(64, activation='relu'))

  # Output layer - for binary classification use 2 and sigmoid activation
  model.add(Dense(2, activation='sigmoid'))

  # Compile the model - binary_crossentropy for binary data
  model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

  history = model.fit(training_data, training_labels, epochs = 50, batch_size = 32, validation_split = 0.2)

  # Obtain Training and Validation Losses from model history
  cnn_training_losses = history.history["loss"]
  cnn_testing_losses = history.history["val_loss"]

  # Make predictions for classification metrics
  predicted_labels = model.predict(testing_data)

  return cnn_training_losses, cnn_testing_losses, predicted_labels
```

Classification Measures Helper Function

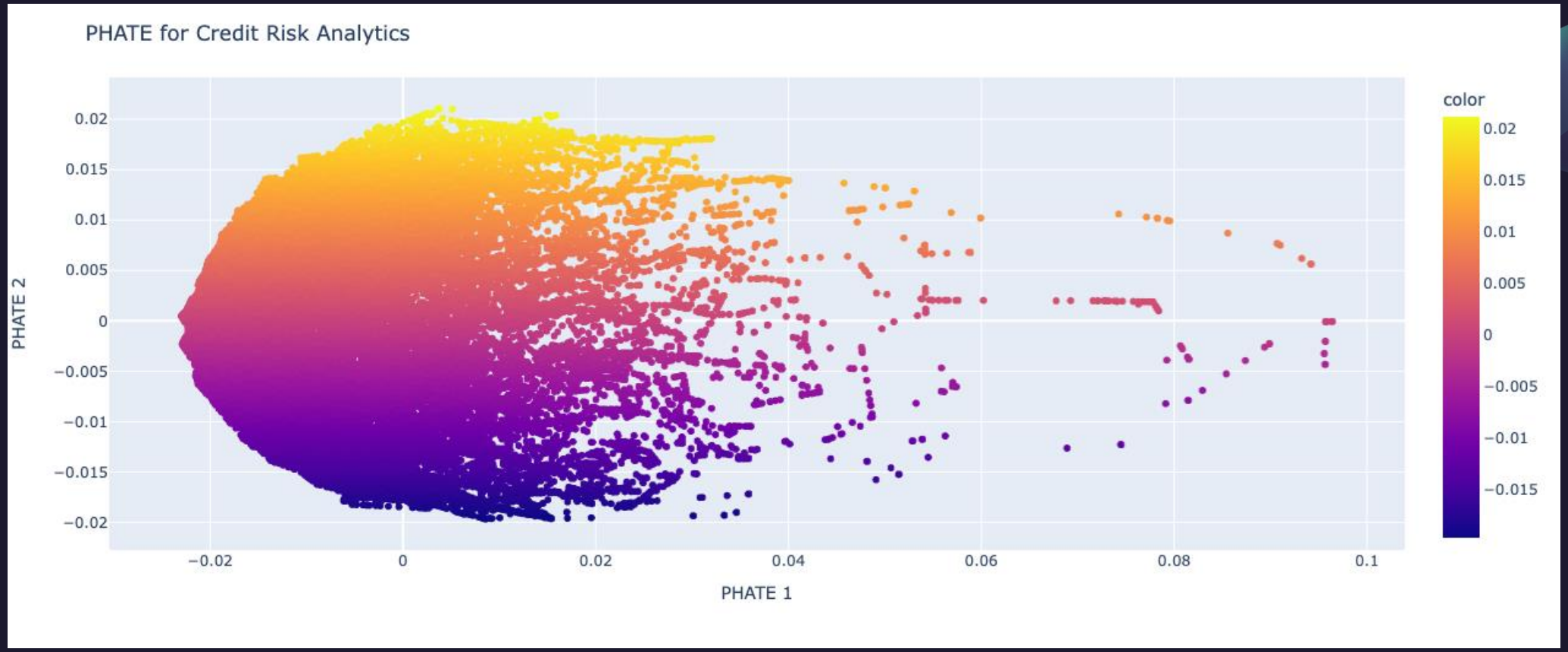
```python
# Helper Function for Calculating Classification Measures

def compute_measure(true_label, predicted_label):
  t_idx = (true_label == predicted_label) # correctly predicted
  f_idx = np.logical_not(t_idx) # incorrectly predicted
  p_idx = (true_label>0) # positive
  n_idx = np.logical_not(p_idx) # negative
  tp = np.sum(np.logical_and(t_idx, p_idx)) # TP
  tn = np.sum(np.logical_and(t_idx, n_idx)) # TN
  fp = np.sum(n_idx) - tn
  fn = np.sum(p_idx) - tp
  tp_fp_tn_fn_list = []
  tp_fp_tn_fn_list.append(tp)
  tp_fp_tn_fn_list.append(fp)
  tp_fp_tn_fn_list.append(tn)
  tp_fp_tn_fn_list.append(fn)
  tp_fp_tn_fn_list = np.array(tp_fp_tn_fn_list)
  tp = tp_fp_tn_fn_list[0]
  fp = tp_fp_tn_fn_list[1]
  tn = tp_fp_tn_fn_list[2]
  fn = tp_fp_tn_fn_list[3]

  with np.errstate(divide= 'ignore'):
    sen = (1.0 * tp)/(tp + fn)
  with np.errstate(divide= 'ignore'):
    spec = (1.0 * tn)/(tn + fp)
  with np.errstate(divide= 'ignore'):
    ppr = (1.0*tp)/(tp+fp)
  with np.errstate(divide= 'ignore'):
    npr = (1.0*tn)/(tn+fn)
  with np.errstate(divide= 'ignore'):
    f1=tp/(tp+0.5*(fp+fn))

  acc = (tp+tn)*1.0/(tp+fp+tn+fn)
  d = np.log2(1 + acc) + np.log2(1 + (sen+spec)/2)
  ans = []
  ans.append(acc)
  ans.append(sen)
  ans.append(spec)
  ans.append(ppr)
  ans.append(npr)
  ans.append(f1)
  ans.append(d)

  return ans
```

```python
# Printing Classification Report

def model_metrics(true,pred):
  ans = compute_measure(testing_labels, np.argmax(predicted_labels, axis = 1))
  print("Accuracy is {0:4f}".format(ans[0]))
  print("Sensitivity is {0:4f}".format(ans[1]))
  print("Specificity is {0:4f}".format(ans[2]))
  print("Precision is {0:4f}".format(ans[3]))
  print("Negative Prediction ratio is {0:3f}".format(ans[4]))
  print("F1-score is {0:3f}".format(ans[5]))
  print("Diagnostic index is {0:4f}".format(ans[6]))
  print("\n")
```

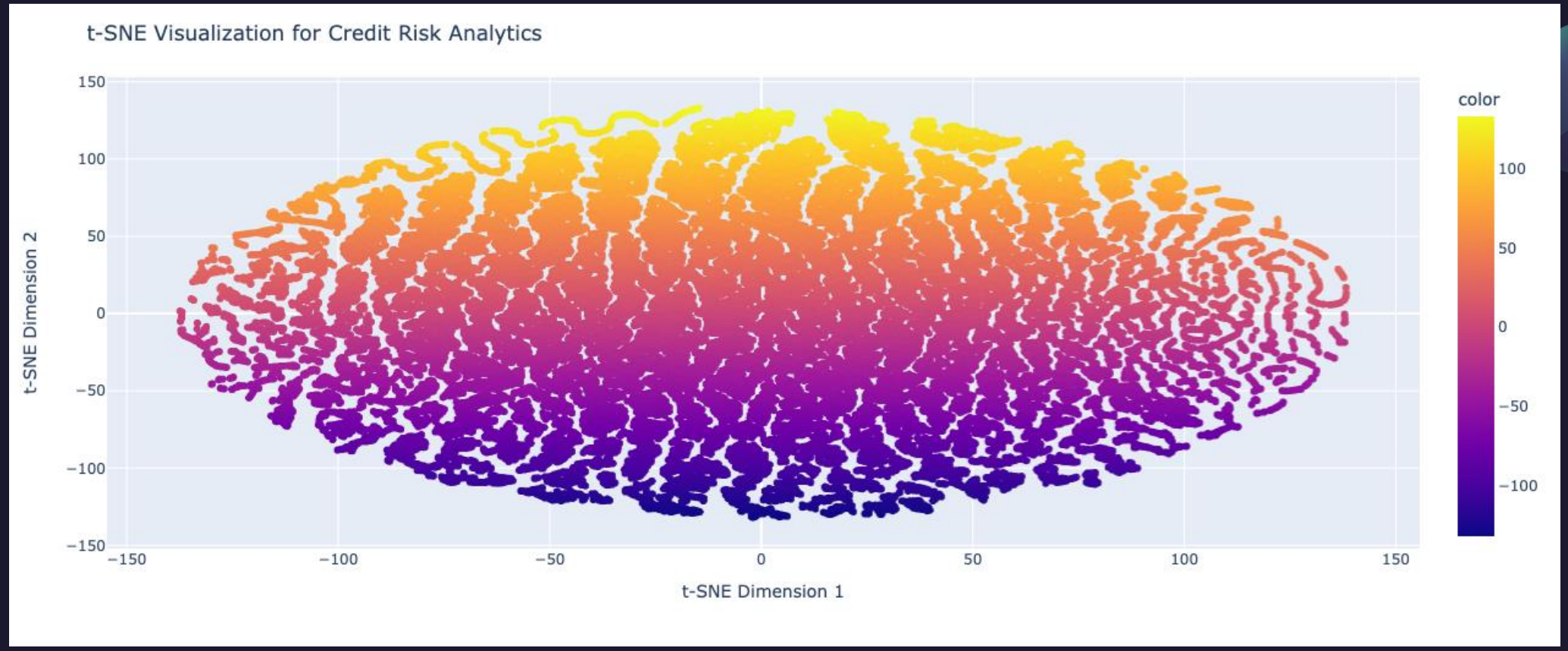
Training and Testing Loss Visualization Helper Function

```
[ ]  # Helper Function to display Training and Validation Losses
     def visualize_losses(model_type, train_losses, test_losses):

       fig = plt.figure(figsize = (20,10))
       plt.plot(train_losses, color = "#253C78", label = 'Training Loss')
       plt.plot(test_losses, color = "#D36582", linestyle = "dashed", linewidth = 2, label = 'Test Loss')
       plt.title("Training and Testing Loss Curves under Adam Solver: " + model_type)
       plt.xlabel('Epochs')
       plt.ylabel('Loss')
       plt.legend()
       plt.grid('on')
       plt.show()
```

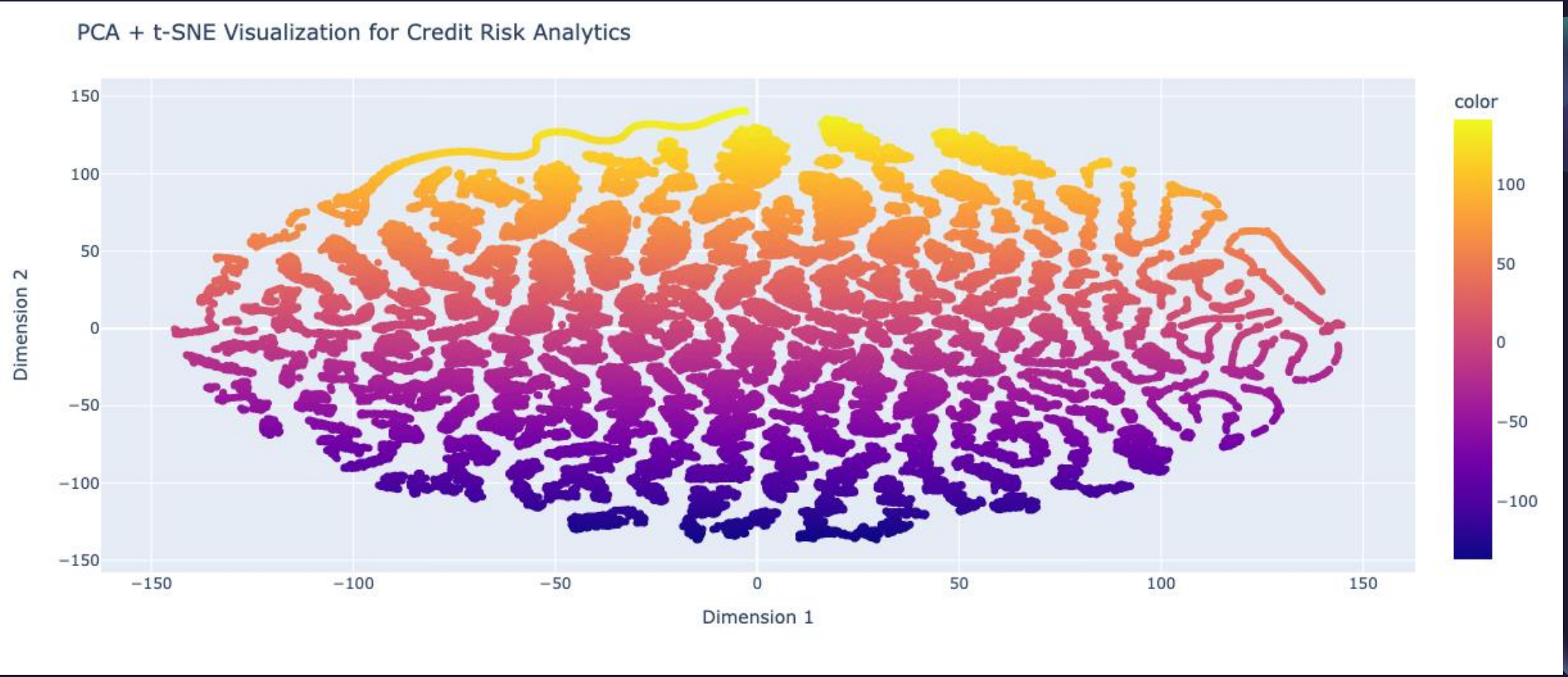
PHATE Visualization



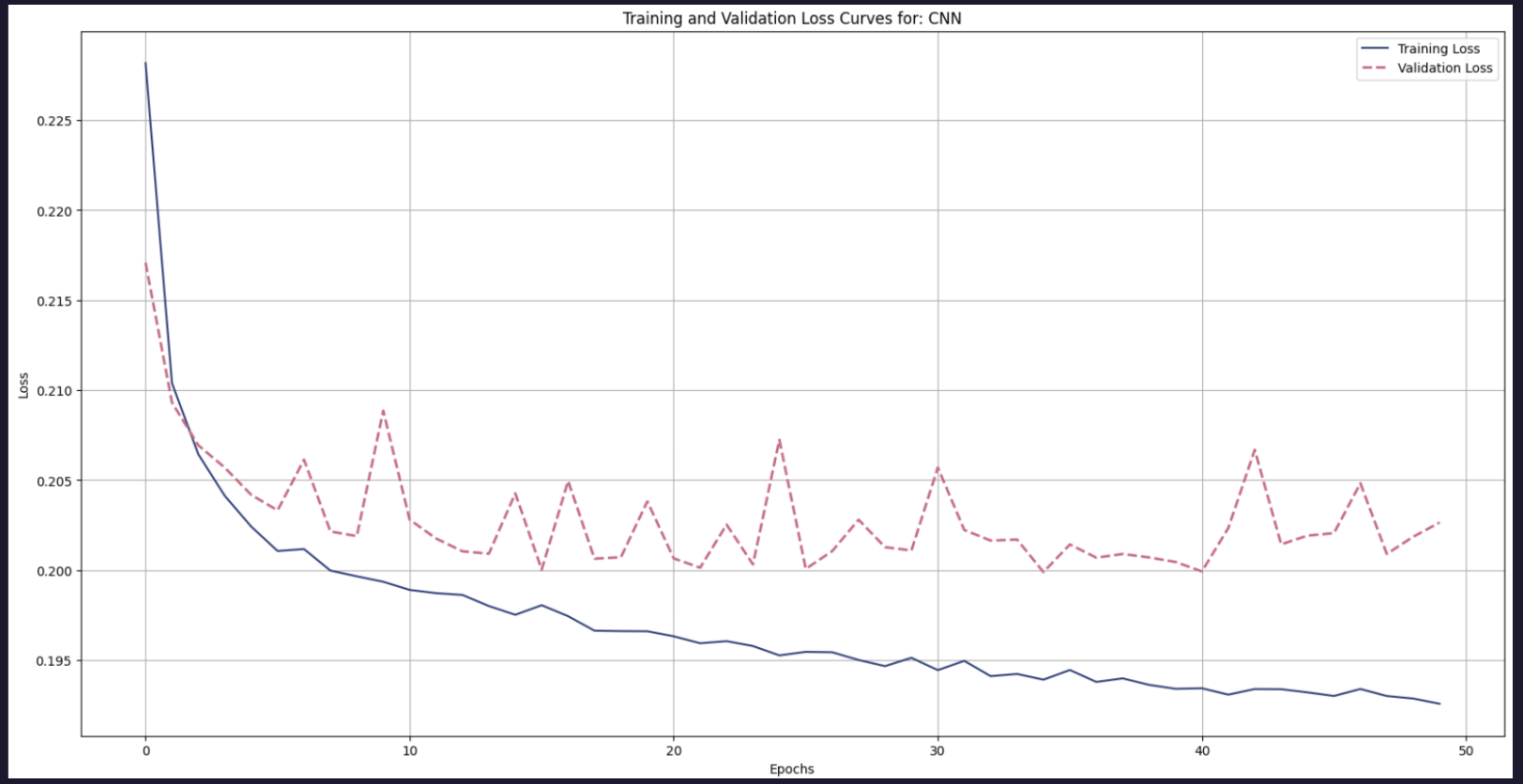
t-SNE Visualization



PCA + t-SNE Visualization

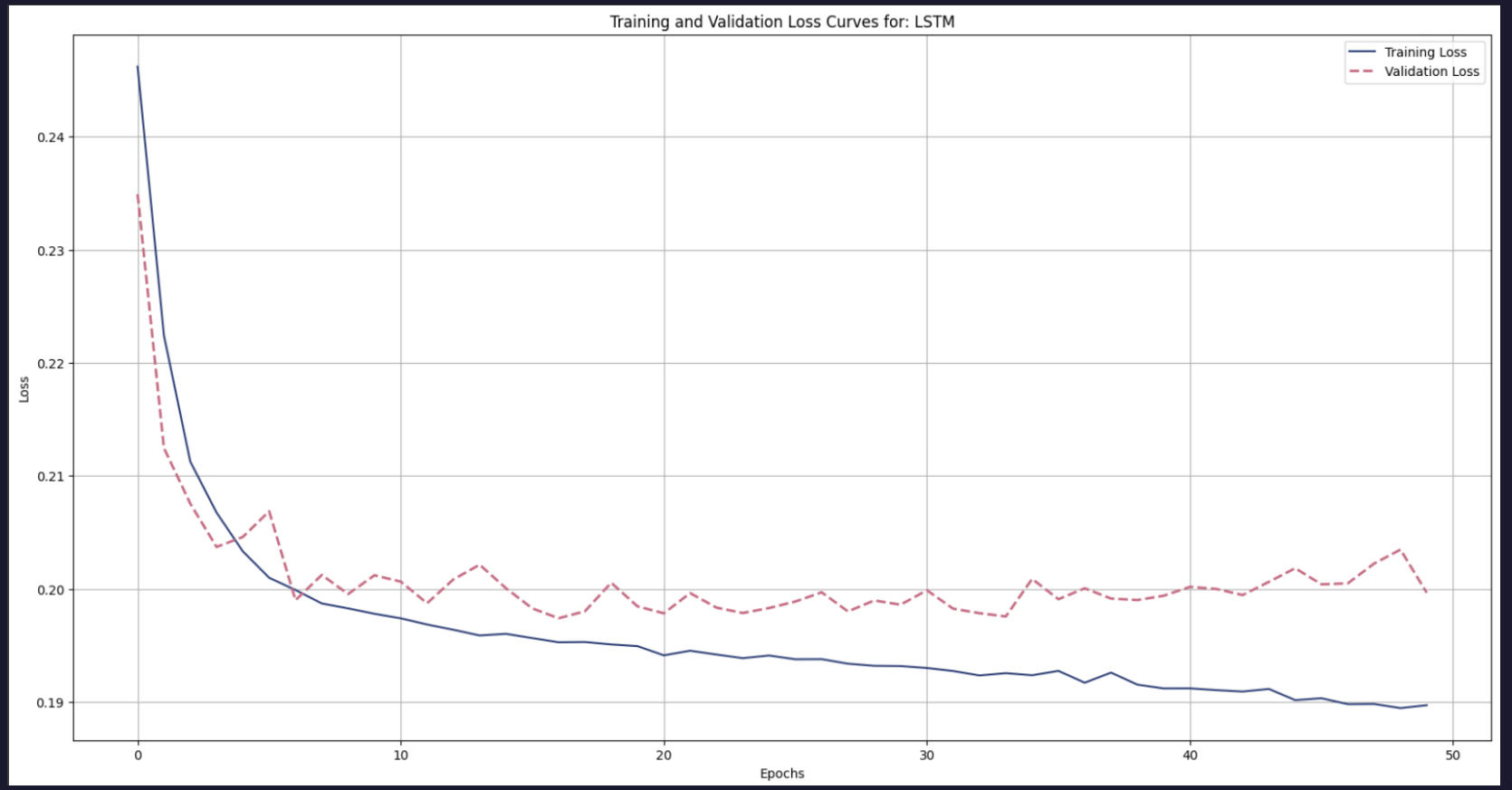


Loss History and Performance - CNN



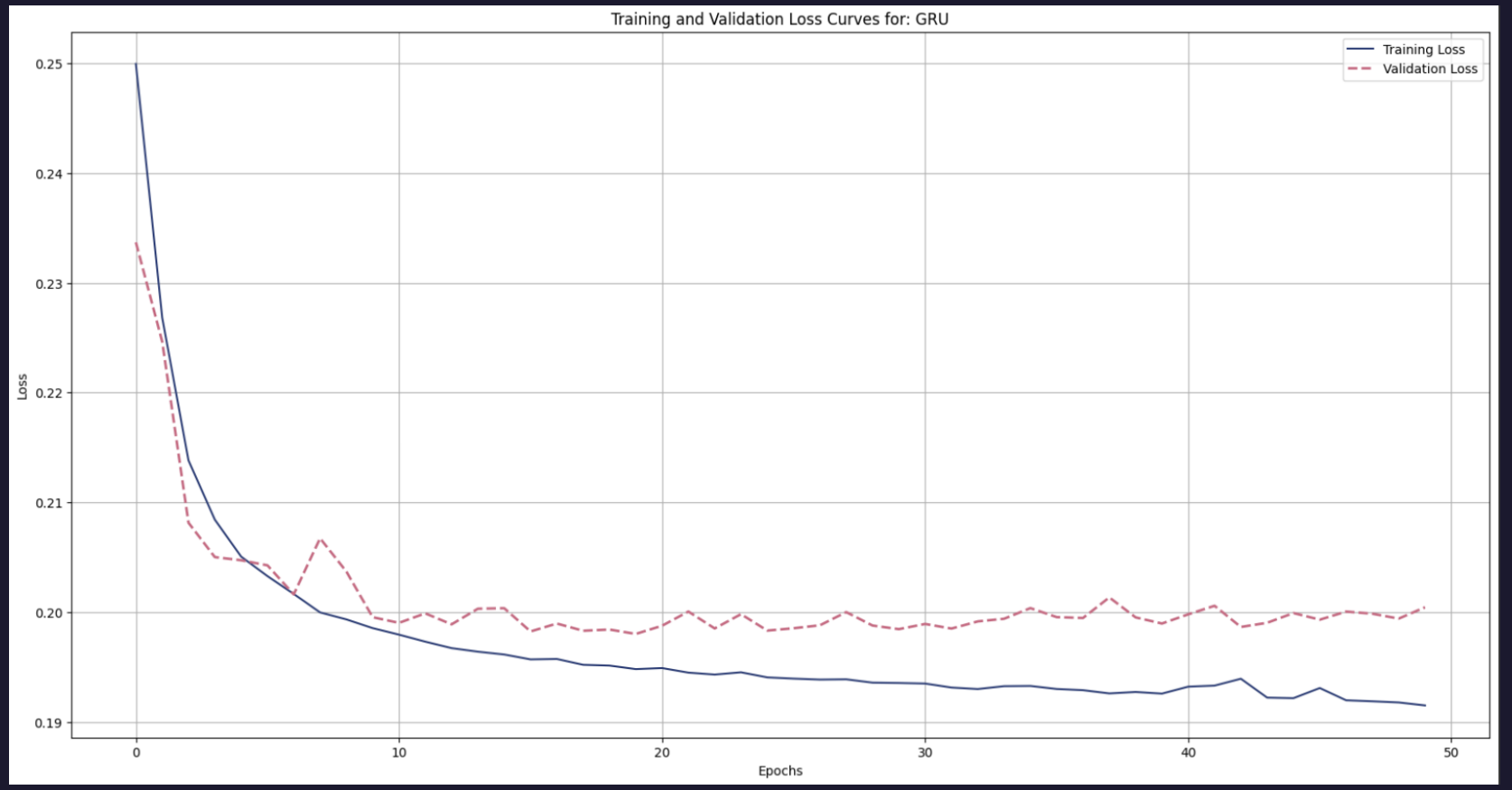

```
Accuracy is 0.931529
Sensitivity is 0.164311
Specificity is 0.989801
Precision is 0.550296
Negative Prediction ratio is 0.939738
F1-score is 0.253061
Diagnostic index is 1.606977
```

Loss History and Performance - LSTM



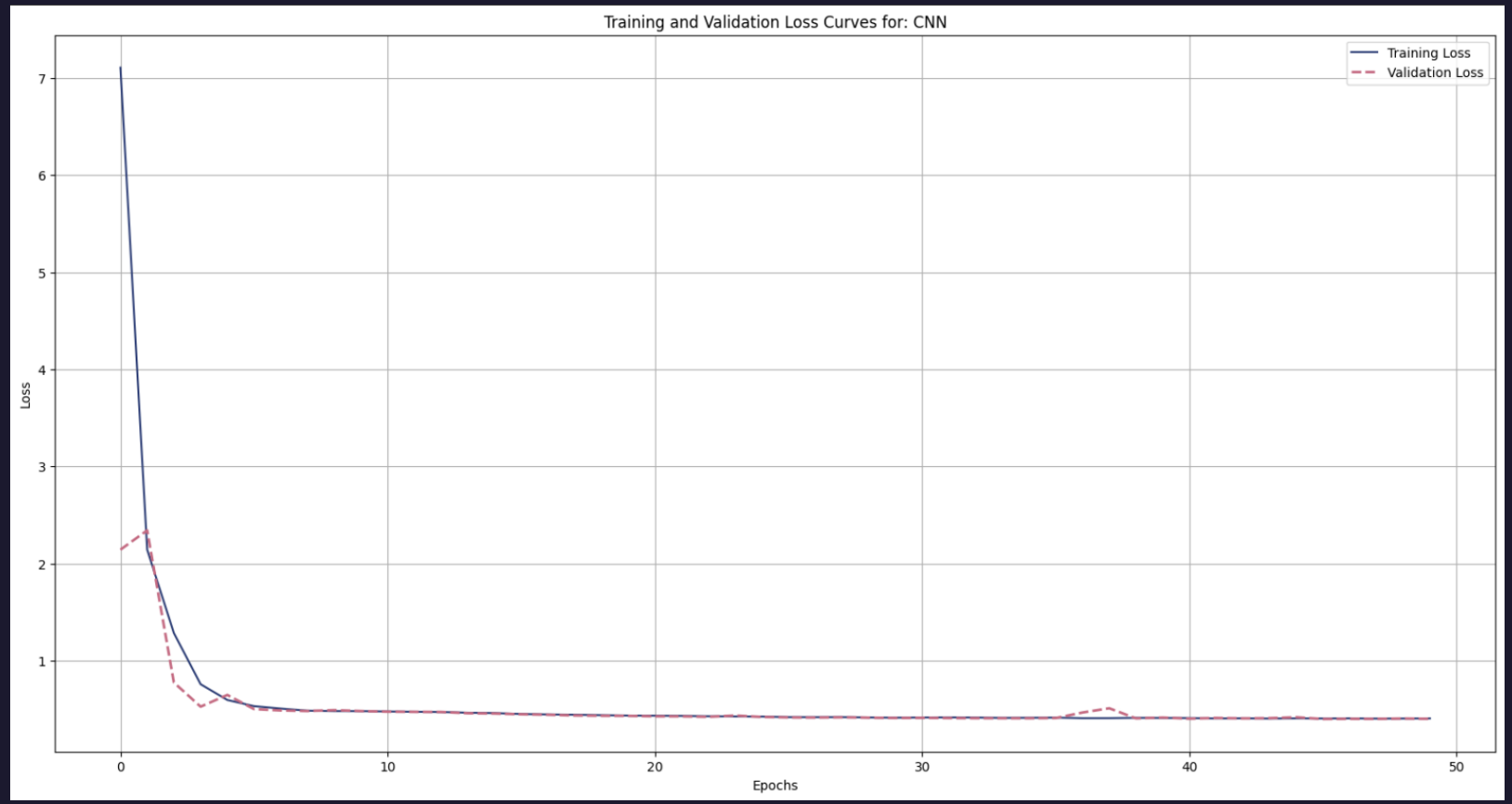

```
Accuracy is 0.929409
Sensitivity is 0.000000
Specificity is 1.000000
Precision is  nan
Negative Prediction ratio is 0.929409
F1-score is 0.000000
Diagnostic index is 1.533121
```

Loss History and Performance - GRU



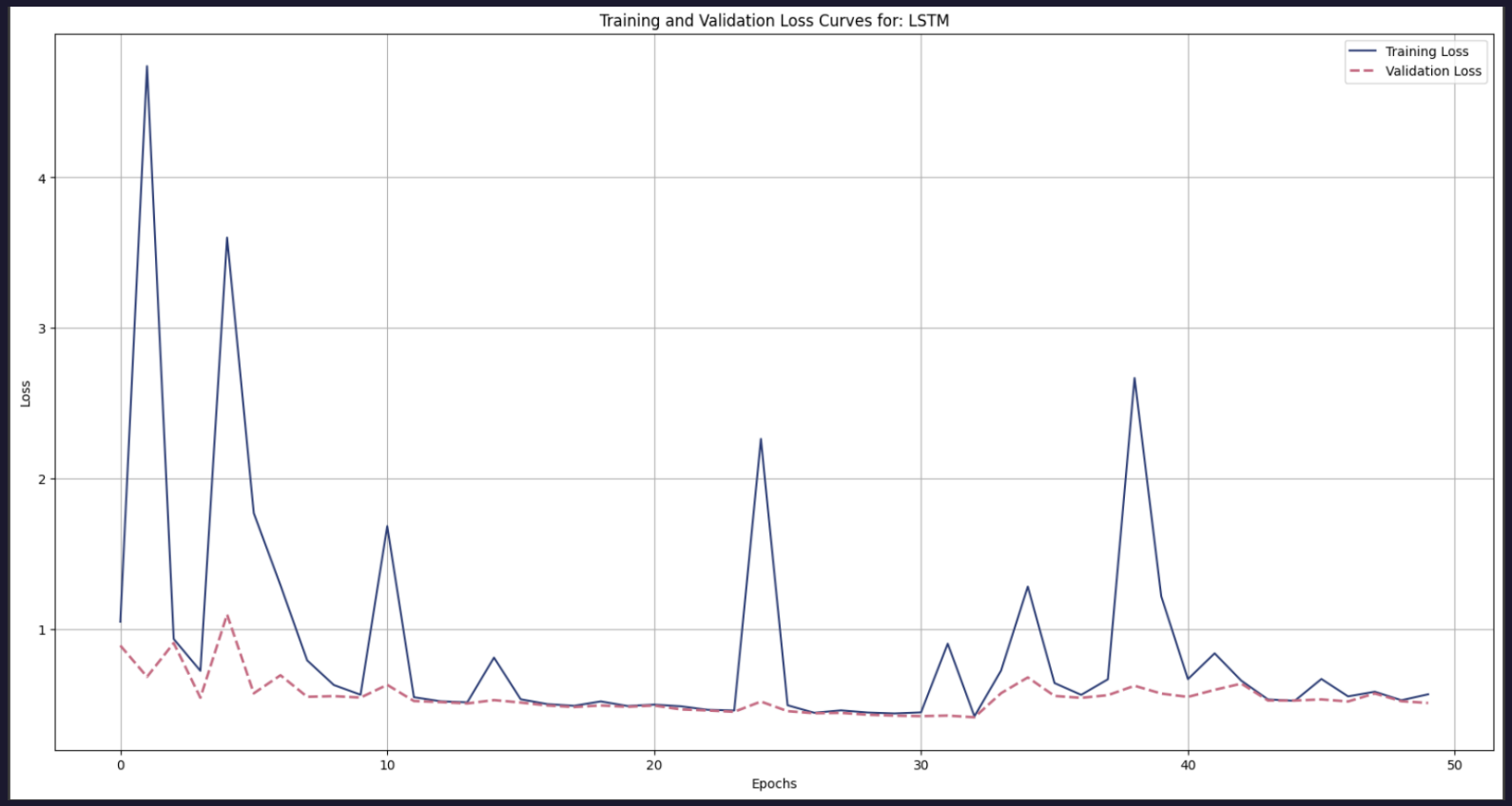

```
Accuracy is 0.929409
Sensitivity is 0.000000
Specificity is 1.000000
Precision is  nan
Negative Prediction ratio is 0.929409
F1-score is 0.000000
Diagnostic index is 1.533121
```

Loss History and Performance – CNN (with SMOTE)



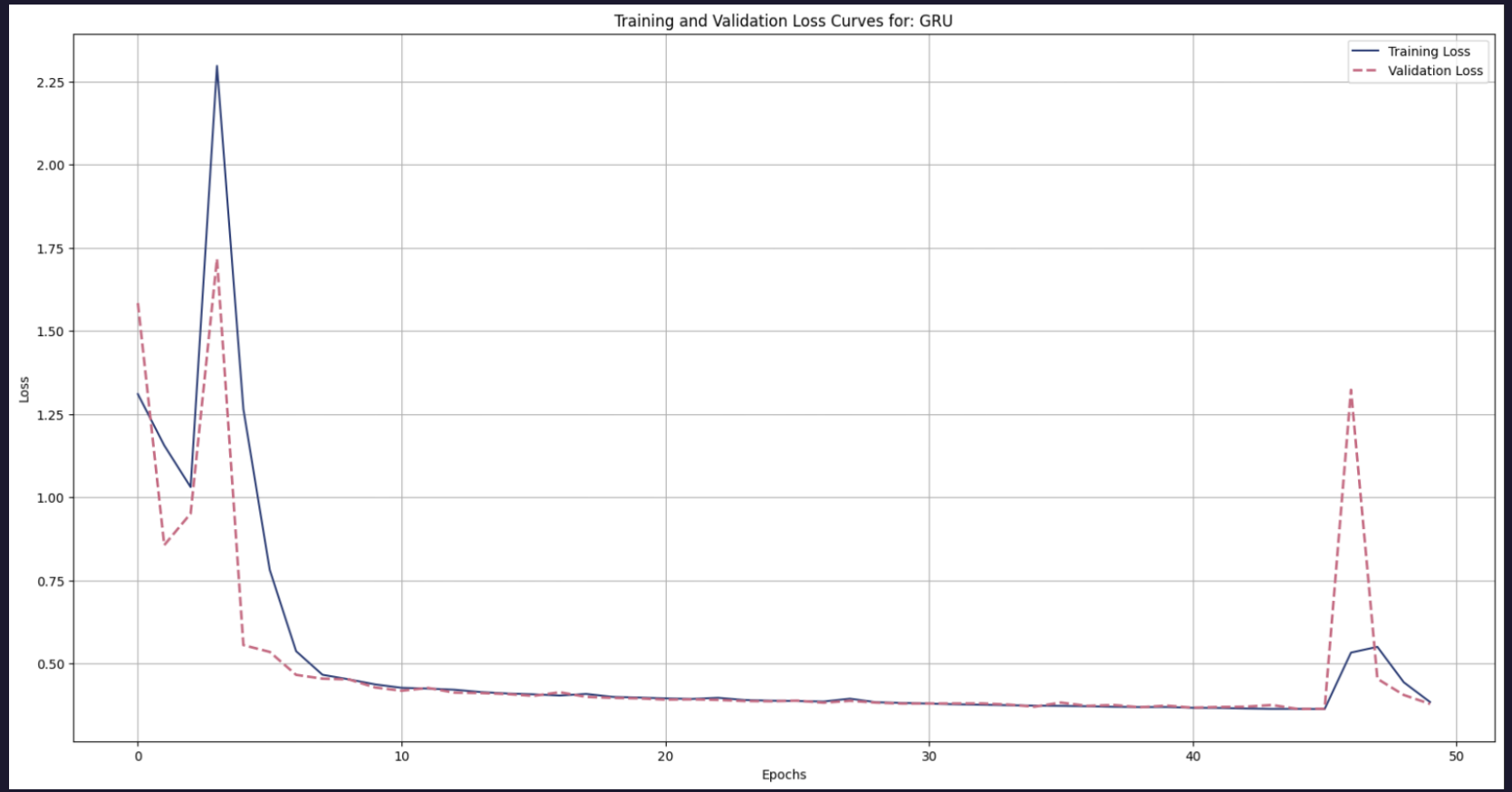

```
Accuracy is 0.809226
Sensitivity is 0.824925
Specificity is 0.793532
Precision is 0.799766
Negative Prediction ratio is 0.819297
F1-score is 0.812151
Diagnostic index is 1.710747
```

Loss History and Performance – LSTM (with SMOTE)



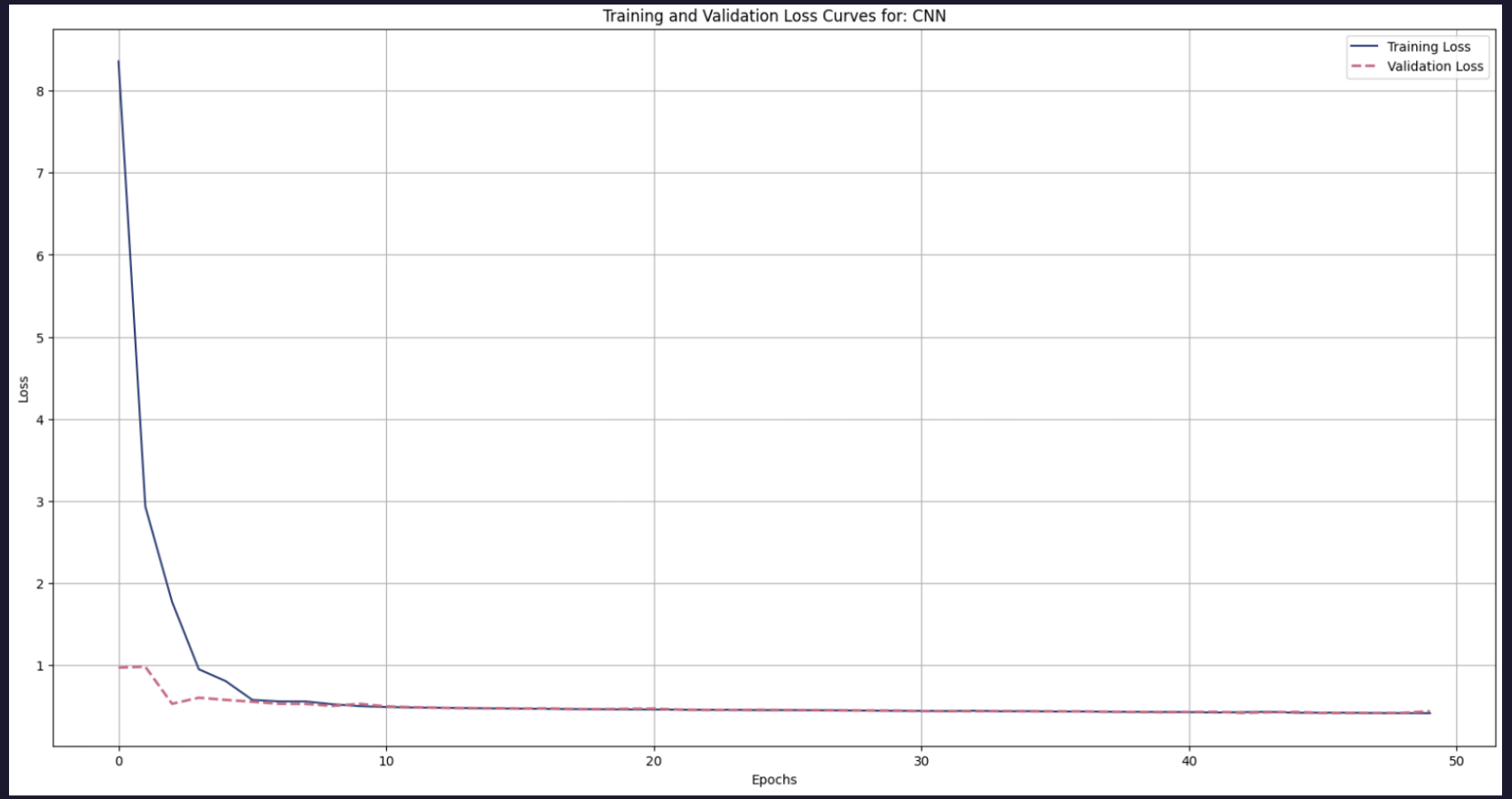

```
Accuracy is 0.500078
Sensitivity is 0.000000
Specificity is 1.000000
Precision is  nan
Negative Prediction ratio is 0.500078
F1-score is 0.000000
Diagnostic index is 1.170000
```

Loss History and Performance – GRU (with SMOTE)



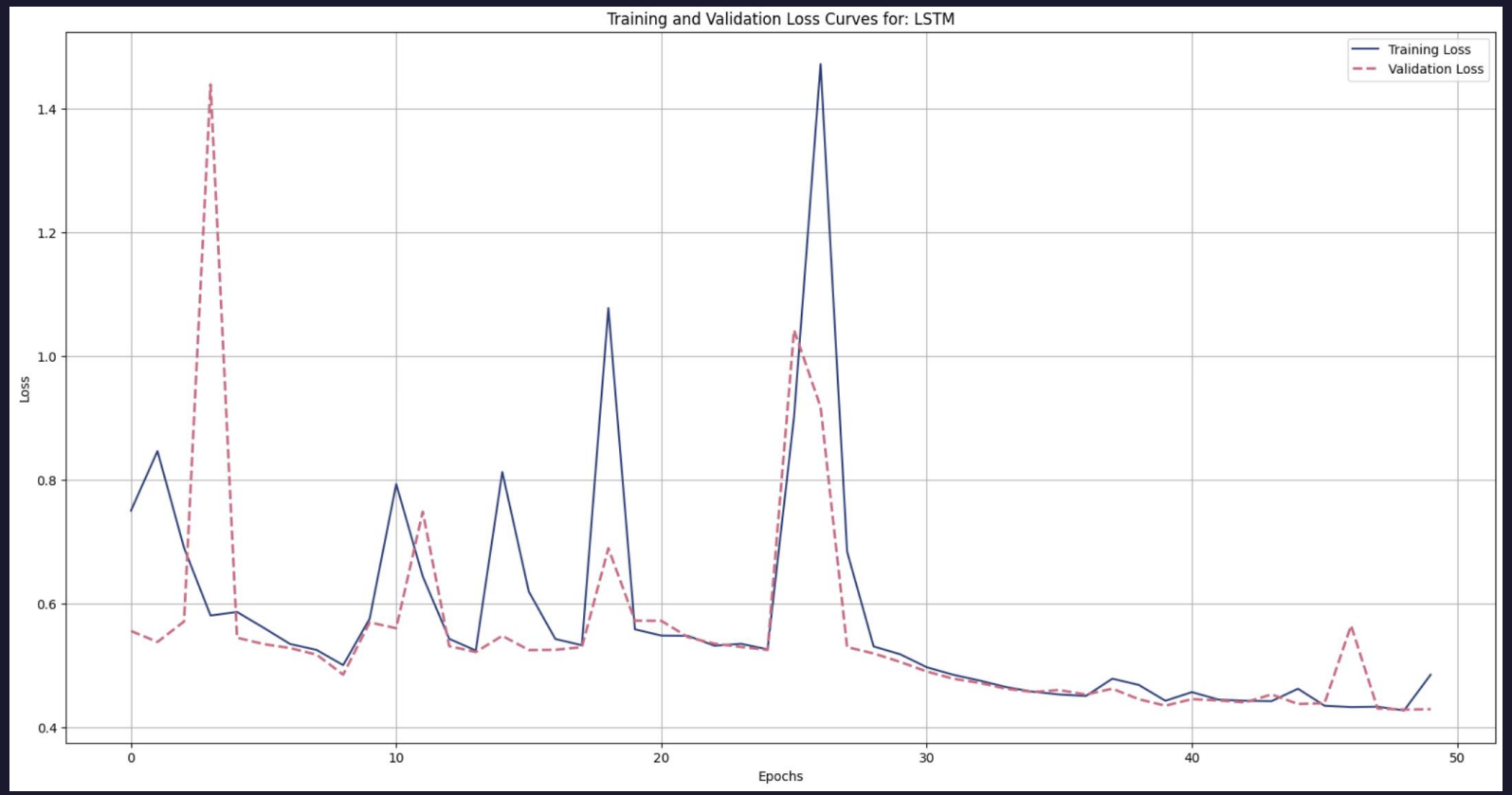

```
Accuracy is 0.500078
Sensitivity is 0.000000
Specificity is 1.000000
Precision is  nan
Negative Prediction ratio is 0.500078
F1-score is 0.000000
Diagnostic index is 1.170000
```

Loss History and Performance – CNN (with ADASYN)



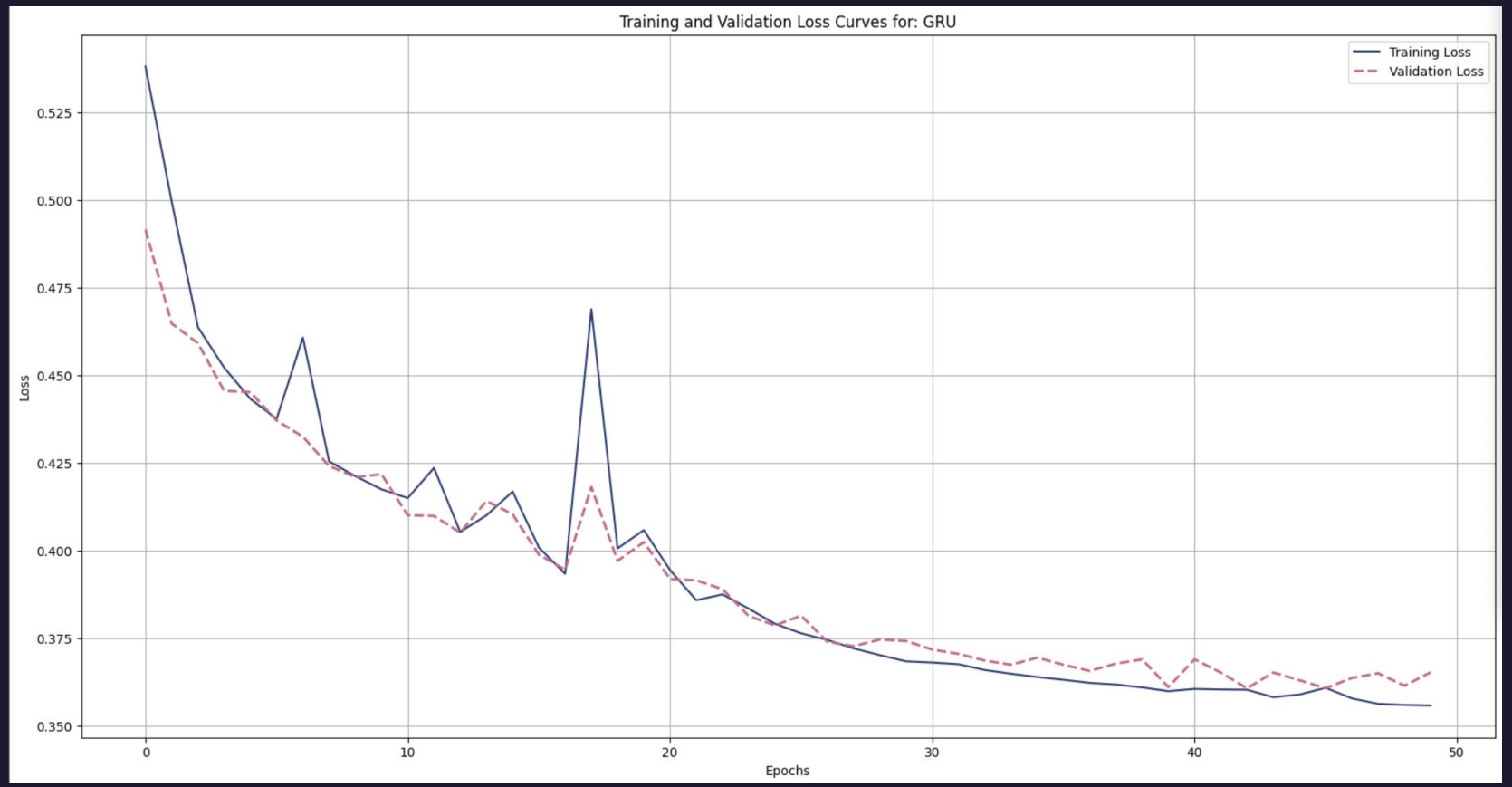

```
Accuracy is 0.794415
Sensitivity is 0.813020
Specificity is 0.775368
Precision is 0.787474
Negative Prediction ratio is 0.802003
F1-score is 0.800043
Diagnostic index is 1.686849
```

Loss History and Performance – LSTM (with ADASYN)




```
Accuracy is 0.494131
Sensitivity is 0.000000
Specificity is 1.000000
Precision is  nan
Negative Prediction ratio is 0.494131
F1-score is 0.000000
Diagnostic index is 1.164269
```

Loss History and Performance – GRU (with ADASYN)




```
Accuracy is 0.494131
Sensitivity is 0.000000
Specificity is 1.000000
Precision is  nan
Negative Prediction ratio is 0.494131
F1-score is 0.000000
Diagnostic index is 1.164269
```

Results & Conclusions

PART B

- Metrics were generally lower across all classification measures for LSTM and GRU models. This result is also shared with the models in combination with resampling methods.
- CNN, with and without resampling methods, had higher accuracy scores and higher metrics across our measured classification metrics.
- Our LSTM and GRU models had sensitivity and specificity values of 0 and 1, respectively.
- If sensitivity is 0, there are no true positives, but instead false negatives. If specificity is 1, there were no false positive predictions. Our predictions were 0 because the majority of the observations were of class 0. Therefore, our score is 1.
- The imbalance of the classes are a huge issue in making accurate predictions. Increased observations of the minority class, would be beneficial in future research as we may gain more accurate conclusions.

Thank You

Courtney Hodge, Tara Williams

Courtney_Hodge1@baylor.edu

Tara_Williams1@baylor.edu