Name: Santiago, John Loyd C.
Course and Section: CPE 019 - CPE32S3
Date of Submission: April 2, 2024
Instructor: Engr. Roman Richard

LINK: https://colab.research.google.com/drive/1JYmPdjaU2CLBNzCBfSmQMIDZxTPb3x2P?usp=sharing

Activity 1.2 : Training Neural Networks

Objective(s):

This activity aims to demonstrate how to train neural networks using keras

Intended Learning Outcomes (ILOs):

- Demonstrate how to build and train neural networks
- Demonstrate how to evaluate and plot the model using training and validation loss

Resources:

- Jupyter Notebook

CI Pima Diabetes Dataset

- pima-indians-diabetes.csv

Procedures

Load the necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_auc_score, roc_curve, accuracy_score
from sklearn.ensemble import RandomForestClassifier

import seaborn as sns

%matplotlib inline
```

```
## Import Keras objects for Deep Learning
from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop
```

Load the dataset

```
filepath = "pima-indians-diabetes.csv"
names = ["times_pregnant", "glucose_tolerance_test", "blood_pressure", "skin_thickness", "insulin",
         "bmi", "pedigree_function", "age", "has_diabetes"]
diabetes_df = pd.read_csv(filepath, names=names)
```

Check the top 5 samples of the data

```
print(diabetes_df.shape)
diabetes_df.sample(5)
```

(768, 9)

	times_pregnant	glucose_tolerance_test	blood_pressure	skin_thickness	insulin	bmi	pedigree_function	age	has_diabetes
145	0	102	75	23	0	0.0	0.572	21	0
238	9	164	84	21	0	30.8	0.831	32	1
376	0	98	82	15	84	25.2	0.299	22	0
533	6	91	0	0	0	29.8	0.501	31	0
396	3	96	56	34	115	24.7	0.944	39	0

```
diabetes_df.dtypes
```

```
times_pregnant              int64
glucose_tolerance_test      int64
blood_pressure              int64
skin_thickness              int64
insulin                     int64
bmi                       float64
pedigree_function         float64
age                         int64
has_diabetes                int64
dtype: object
```

```
X = diabetes_df.iloc[:, :-1].values
y = diabetes_df["has_diabetes"].values
```

Split the data to Train, and Test (75%, 25%)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=11111)
```

```
np.mean(y), np.mean(1-y)
```

(0.3489583333333333, 0.6510416666666666)

Build a single hidden layer neural network using 12 nodes. Use the sequential model with single layer network and input shape to 8.

Normalize the data

```
normalizer = StandardScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)
```

Define the model:

- Input size is 8-dimensional
- 1 hidden layer, 12 hidden nodes, sigmoid activation
- Final layer with one node and sigmoid activation (standard for binary classification)

```
model  = Sequential([
    Dense(12, input_shape=(8,), activation="relu"),
    Dense(1, activation="sigmoid")
])
```

View the model summary

```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_26 (Dense)            (None, 12)                108

 dense_27 (Dense)            (None, 1)                 13

=================================================================
Total params: 121 (484.00 Byte)
Trainable params: 121 (484.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Train the model

- Compile the model with optimizer, loss function and metrics

- Use the fit function to return the run history.

```
model.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
run_hist_1 = model.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test), epochs=200)
```

```
Epoch 172/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4371 - accuracy: 0.7847 - val_loss: 0.5088 - val_accuracy: 0.7500
Epoch 173/200
18/18 [==============================] - 0s 3ms/step - loss: 0.4370 - accuracy: 0.7865 - val_loss: 0.5088 - val_accuracy: 0.7500
Epoch 174/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4369 - accuracy: 0.7812 - val_loss: 0.5088 - val_accuracy: 0.7500
Epoch 175/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4368 - accuracy: 0.7830 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 176/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4367 - accuracy: 0.7865 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 177/200
18/18 [==============================] - 0s 3ms/step - loss: 0.4366 - accuracy: 0.7847 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 178/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4365 - accuracy: 0.7847 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 179/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4364 - accuracy: 0.7830 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 180/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4362 - accuracy: 0.7865 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 181/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4363 - accuracy: 0.7847 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 182/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4362 - accuracy: 0.7830 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 183/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4360 - accuracy: 0.7847 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 184/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4359 - accuracy: 0.7865 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 185/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4358 - accuracy: 0.7882 - val_loss: 0.5088 - val_accuracy: 0.7552
Epoch 186/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4357 - accuracy: 0.7882 - val_loss: 0.5089 - val_accuracy: 0.7552
Epoch 187/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4356 - accuracy: 0.7865 - val_loss: 0.5089 - val_accuracy: 0.7552
Epoch 188/200
18/18 [==============================] - 0s 3ms/step - loss: 0.4356 - accuracy: 0.7882 - val_loss: 0.5089 - val_accuracy: 0.7552
Epoch 189/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4354 - accuracy: 0.7865 - val_loss: 0.5090 - val_accuracy: 0.7552
Epoch 190/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4354 - accuracy: 0.7847 - val_loss: 0.5090 - val_accuracy: 0.7552
Epoch 191/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4353 - accuracy: 0.7865 - val_loss: 0.5090 - val_accuracy: 0.7552
Epoch 192/200
18/18 [==============================] - 0s 3ms/step - loss: 0.4351 - accuracy: 0.7865 - val_loss: 0.5091 - val_accuracy: 0.7552
Epoch 193/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4351 - accuracy: 0.7865 - val_loss: 0.5091 - val_accuracy: 0.7552
Epoch 194/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4350 - accuracy: 0.7847 - val_loss: 0.5091 - val_accuracy: 0.7552
Epoch 195/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4349 - accuracy: 0.7847 - val_loss: 0.5091 - val_accuracy: 0.7552
Epoch 196/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4348 - accuracy: 0.7882 - val_loss: 0.5092 - val_accuracy: 0.7552
Epoch 197/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4347 - accuracy: 0.7865 - val_loss: 0.5092 - val_accuracy: 0.7552
Epoch 198/200
18/18 [==============================] - 0s 3ms/step - loss: 0.4346 - accuracy: 0.7882 - val_loss: 0.5093 - val_accuracy: 0.7552
Epoch 199/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4346 - accuracy: 0.7899 - val_loss: 0.5093 - val_accuracy: 0.7552
Epoch 200/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4346 - accuracy: 0.7882 - val_loss: 0.5093 - val_accuracy: 0.7552
```

```
## Like we did for the Random Forest, we generate two kinds of predictions
#  One is a hard decision, the other is a probabilitistic score.

y_pred_class_nn_1 = (model.predict(X_test_norm)> 0.5).astype('int32')
y_pred_prob_nn_1 = model.predict(X_test_norm)
```

```
6/6 [==============================] - 0s 2ms/step
6/6 [==============================] - 0s 2ms/step
```

```
# Let's check out the outputs to get a feel for how keras apis work.
y_pred_class_nn_1[:10]
```

```
array([[1],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0]], dtype=int32)
```

```
y_pred_prob_nn_1[:10]
```

```
array([[0.6086611 ],
       [0.74070066],
       [0.3285517 ],
       [0.2226287 ],
       [0.15997736],
       [0.49751613],
       [0.02409566],
       [0.43399602],
       [0.9366918 ],
       [0.22799575]], dtype=float32)
```

Create the plot_roc function

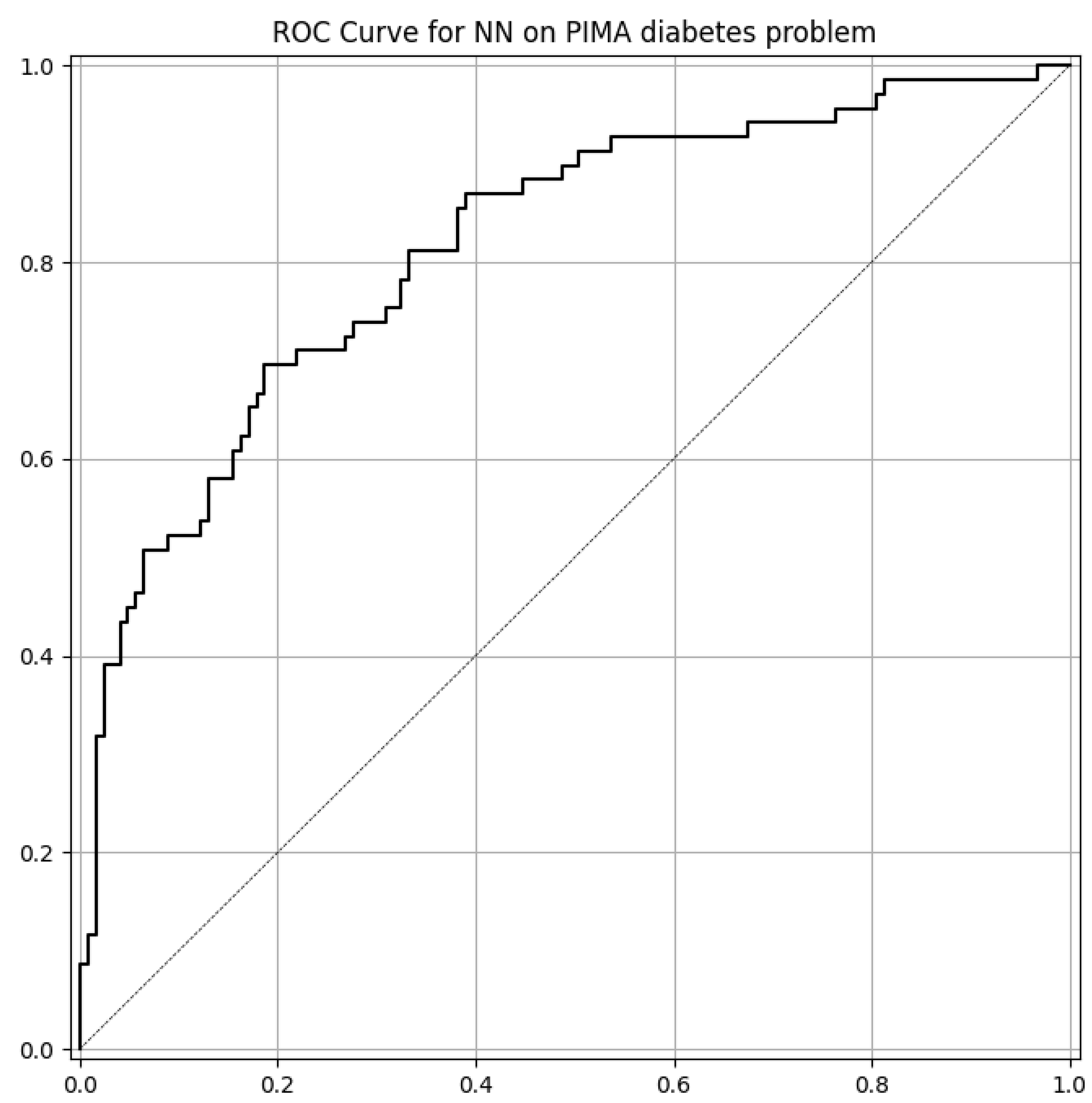
```
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA diabetes problem'.format(model_name),
           xlim=[-0.01, 1.01], ylim=[-0.01, 1.01])
```

Evaluate the model performance and plot the ROC CURVE

```
print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class_nn_1)))
print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob_nn_1)))

plot_roc(y_test, y_pred_prob_nn_1, 'NN')
```

```
accuracy is 0.755
roc-auc is 0.816
```



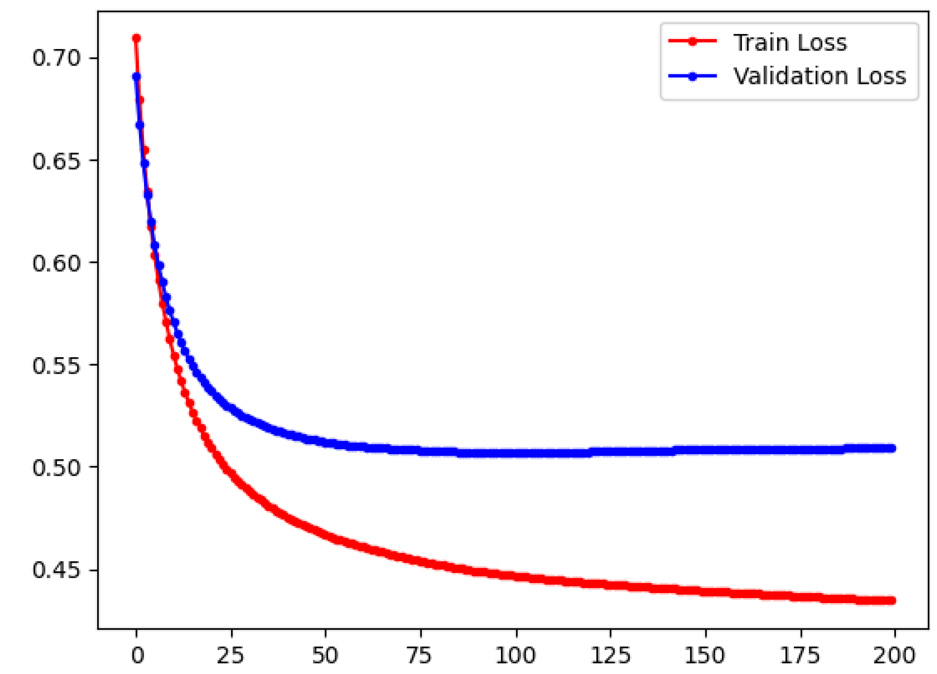
Plot the training loss and the validation loss over the different epochs and see how it looks

```
run_hist_1.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
fig, ax = plt.subplots()
ax.plot(run_hist_1.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(run_hist_1.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7e43c6487250>
```



What is your interpretation about the result of the train and validation loss?

type your answer here

The graph shows that the data is not process and trained efficiently as it shows in the high loss value and low accuracy and also in the ROC-AUC score. its still a positive score but it didn't achieve the satisfactory performance in terms of training and prediction task

Supplementary Activity

- Build a model with two hidden layers, each with 6 nodes
- Use the "relu" activation function for the hidden layers, and "sigmoid" for the final layer
- Use a learning rate of .003 and train for 1500 epochs
- Graph the trajectory of the loss functions, accuracy on both train and test set
- Plot the roc curve for the predictions
- Use different learning rates, numbers of epochs, and network structures.
- Plot the results of training and validation loss using different learning rates, number of epocgs and network structures
- Interpret your result

```
x = diabetes_df.iloc[:, :-1].values
y = diabetes_df["has_diabetes"].values
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=11111)
```

```
normalizer = StandardScaler()
x_train_n = normalizer.fit_transform(x_train)
x_test_n = normalizer.transform(x_test)
```

```
#Build a model with two hidden layers, each with 6 nodes
#Use the "relu" activation function for the hidden layers, and "sigmoid" for the final layer
#Use a learning rate of .003 and train for 1500 epochs
model = Sequential([
      Dense(6, input_shape=(8,), activation="relu"),
      Dense(6, activation = 'relu'),
      Dense(1, activation="sigmoid")])
model.compile(SGD(learning_rate = 0.003),"binary_crossentropy", metrics = ['accuracy'])
model_fit = model.fit(x_train_n, y_train, validation_data = (x_test_n, y_test),epochs = 1500)
```

```
Epoch 1472/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4123 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1473/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4123 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1474/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4123 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1475/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4123 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1476/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4122 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1477/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4122 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1478/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4122 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1479/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4123 - accuracy: 0.8073 - val_loss: 0.5191 - val_accuracy: 0.7708
Epoch 1480/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.4122 - accuracy: 0.8073 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1481/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4122 - accuracy: 0.8056 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1482/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4121 - accuracy: 0.8073 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1483/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4121 - accuracy: 0.8073 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1484/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.4121 - accuracy: 0.8073 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1485/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4121 - accuracy: 0.8073 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1486/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4121 - accuracy: 0.8073 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1487/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4120 - accuracy: 0.8056 - val_loss: 0.5192 - val_accuracy: 0.7708
Epoch 1488/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4120 - accuracy: 0.8056 - val_loss: 0.5193 - val_accuracy: 0.7708
Epoch 1489/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4120 - accuracy: 0.8090 - val_loss: 0.5193 - val_accuracy: 0.7708
Epoch 1490/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4120 - accuracy: 0.8073 - val_loss: 0.5193 - val_accuracy: 0.7708
Epoch 1491/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.4120 - accuracy: 0.8073 - val_loss: 0.5193 - val_accuracy: 0.7708
Epoch 1492/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4120 - accuracy: 0.8073 - val_loss: 0.5194 - val_accuracy: 0.7708
Epoch 1493/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4119 - accuracy: 0.8090 - val_loss: 0.5194 - val_accuracy: 0.7760
Epoch 1494/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4119 - accuracy: 0.8073 - val_loss: 0.5194 - val_accuracy: 0.7708
Epoch 1495/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4118 - accuracy: 0.8073 - val_loss: 0.5194 - val_accuracy: 0.7708
Epoch 1496/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.4118 - accuracy: 0.8073 - val_loss: 0.5194 - val_accuracy: 0.7708
Epoch 1497/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.4118 - accuracy: 0.8073 - val_loss: 0.5194 - val_accuracy: 0.7708
Epoch 1498/1500
18/18 [==============================] - 0s 7ms/step - loss: 0.4118 - accuracy: 0.8056 - val_loss: 0.5194 - val_accuracy: 0.7708
Epoch 1499/1500
18/18 [==============================] - 0s 7ms/step - loss: 0.4118 - accuracy: 0.8073 - val_loss: 0.5195 - val_accuracy: 0.7708
Epoch 1500/1500
18/18 [==============================] - 0s 9ms/step - loss: 0.4117 - accuracy: 0.8073 - val_loss: 0.5195 - val_accuracy: 0.7760
```

```
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_34 (Dense)            (None, 6)                 54

 dense_35 (Dense)            (None, 6)                 42

 dense_36 (Dense)            (None, 1)                 7

=================================================================
Total params: 103 (412.00 Byte)
Trainable params: 103 (412.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
y_pred_class = (model.predict(x_test_n) > 0.5).astype('int32')
y_pred_prob = model.predict(x_test_n)
```

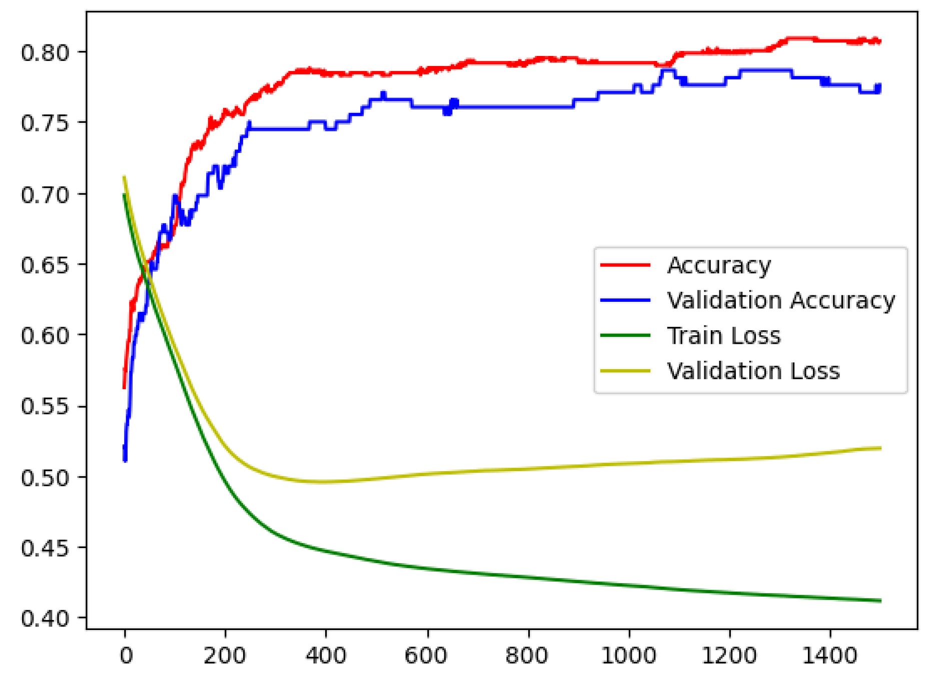
```
6/6 [==============================] - 0s 2ms/step
6/6 [==============================] - 0s 3ms/step
```

```
accuracies = accuracy_score(y_test,y_pred_class)
accuracies
```

```
0.7760416666666666
```

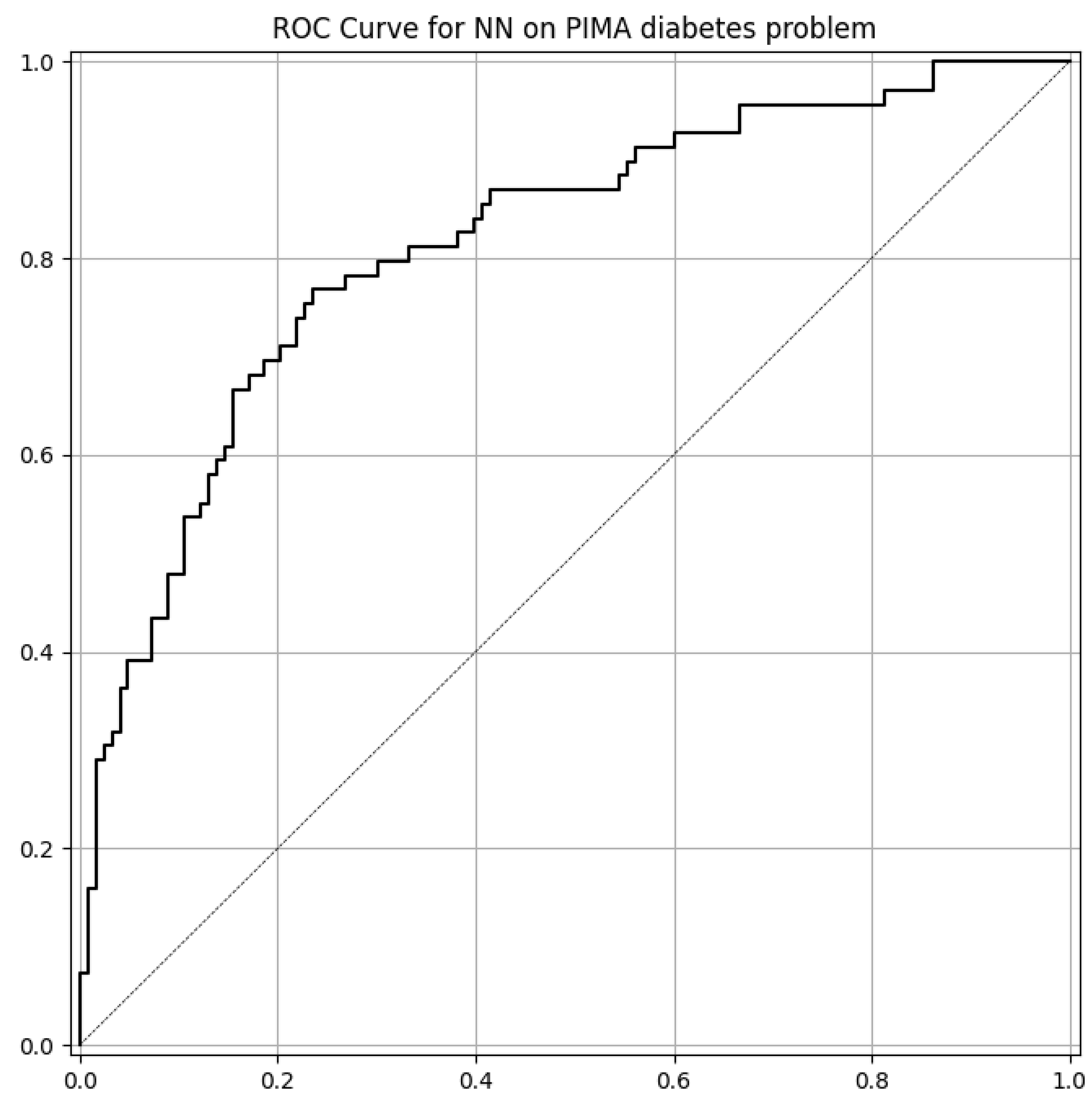
```
#Graph the trajectory of the loss functions, accuracy on both train and test set
fig, ax = plt.subplots()
ax.plot(model_fit.history["accuracy"],'r', label="Accuracy")
ax.plot(model_fit.history["val_accuracy"],'b',label="Validation Accuracy")
ax.plot(model_fit.history["loss"],'g', label="Train Loss")
ax.plot(model_fit.history["val_loss"],'y', label="Validation Loss")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7e43c067d060>
```




```
#Plot the roc curve for the predictions
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA diabetes problem'.format(model_name),
          xlim=[-0.01, 1.01], ylim=[-0.01, 1.01])
    print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class)))
    print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob)))
plot_roc(y_test, y_pred_prob, 'NN')
```

```
accuracy is 0.776
roc-auc is 0.815
```




```
#Use different learning rates, numbers of epochs, and network structures.
model = Sequential([
      Dense(12, input_shape=(8,), activation="relu"),
      Dense(8, activation = 'relu'),
      Dense(12, activation = 'relu'),
      Dense(8, activation = 'softmax'),
      Dense(8, activation = 'tanh'),
      Dense(1, activation="sigmoid") ])
model.compile(SGD(learning_rate = 0.01),"binary_crossentropy", metrics = ['accuracy'])
model_fit = model.fit(x_train_n, y_train, validation_data = (x_test_n, y_test), epochs = 500)
```

```
Epoch 472/500
18/18 [==============================] - 0s 7ms/step - loss: 0.3995 - accuracy: 0.8212 - val_loss: 0.5159 - val_accuracy: 0.7604
Epoch 473/500
18/18 [==============================] - 0s 7ms/step - loss: 0.3993 - accuracy: 0.8177 - val_loss: 0.5156 - val_accuracy: 0.7708
Epoch 474/500
18/18 [==============================] - 0s 7ms/step - loss: 0.3990 - accuracy: 0.8212 - val_loss: 0.5161 - val_accuracy: 0.7604
Epoch 475/500
18/18 [==============================] - 0s 7ms/step - loss: 0.3992 - accuracy: 0.8194 - val_loss: 0.5160 - val_accuracy: 0.7604
Epoch 476/500
18/18 [==============================] - 0s 9ms/step - loss: 0.3987 - accuracy: 0.8177 - val_loss: 0.5167 - val_accuracy: 0.7656
Epoch 477/500
18/18 [==============================] - 0s 6ms/step - loss: 0.3994 - accuracy: 0.8160 - val_loss: 0.5170 - val_accuracy: 0.7656
Epoch 478/500
18/18 [==============================] - 0s 7ms/step - loss: 0.3987 - accuracy: 0.8194 - val_loss: 0.5169 - val_accuracy: 0.7604
Epoch 479/500
18/18 [==============================] - 0s 7ms/step - loss: 0.3985 - accuracy: 0.8194 - val_loss: 0.5173 - val_accuracy: 0.7656
Epoch 480/500
18/18 [==============================] - 0s 7ms/step - loss: 0.3981 - accuracy: 0.8229 - val_loss: 0.5172 - val_accuracy: 0.7656
Epoch 481/500
18/18 [==============================] - 0s 6ms/step - loss: 0.3983 - accuracy: 0.8212 - val_loss: 0.5172 - val_accuracy: 0.7604
Epoch 482/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3978 - accuracy: 0.8212 - val_loss: 0.5172 - val_accuracy: 0.7604
Epoch 483/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3975 - accuracy: 0.8212 - val_loss: 0.5179 - val_accuracy: 0.7656
Epoch 484/500
18/18 [==============================] - 0s 6ms/step - loss: 0.3974 - accuracy: 0.8177 - val_loss: 0.5176 - val_accuracy: 0.7656
Epoch 485/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3975 - accuracy: 0.8177 - val_loss: 0.5175 - val_accuracy: 0.7708
Epoch 486/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3970 - accuracy: 0.8212 - val_loss: 0.5185 - val_accuracy: 0.7656
Epoch 487/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3969 - accuracy: 0.8160 - val_loss: 0.5181 - val_accuracy: 0.7708
Epoch 488/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3973 - accuracy: 0.8194 - val_loss: 0.5188 - val_accuracy: 0.7656
Epoch 489/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3967 - accuracy: 0.8177 - val_loss: 0.5186 - val_accuracy: 0.7656
Epoch 490/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3965 - accuracy: 0.8194 - val_loss: 0.5186 - val_accuracy: 0.7708
Epoch 491/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3969 - accuracy: 0.8177 - val_loss: 0.5187 - val_accuracy: 0.7760
Epoch 492/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3957 - accuracy: 0.8229 - val_loss: 0.5192 - val_accuracy: 0.7656
Epoch 493/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3958 - accuracy: 0.8194 - val_loss: 0.5190 - val_accuracy: 0.7708
Epoch 494/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3958 - accuracy: 0.8229 - val_loss: 0.5189 - val_accuracy: 0.7760
Epoch 495/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3958 - accuracy: 0.8212 - val_loss: 0.5195 - val_accuracy: 0.7708
Epoch 496/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3954 - accuracy: 0.8212 - val_loss: 0.5197 - val_accuracy: 0.7708
Epoch 497/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3957 - accuracy: 0.8194 - val_loss: 0.5200 - val_accuracy: 0.7656
Epoch 498/500
18/18 [==============================] - 0s 5ms/step - loss: 0.3958 - accuracy: 0.8160 - val_loss: 0.5198 - val_accuracy: 0.7708
Epoch 499/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3950 - accuracy: 0.8142 - val_loss: 0.5199 - val_accuracy: 0.7760
Epoch 500/500
18/18 [==============================] - 0s 4ms/step - loss: 0.3944 - accuracy: 0.8247 - val_loss: 0.5198 - val_accuracy: 0.7760
```

```
y_pred_class = (model.predict(x_test_n) > 0.5).astype('int32')
y_pred_prob = model.predict(x_test_n)
```

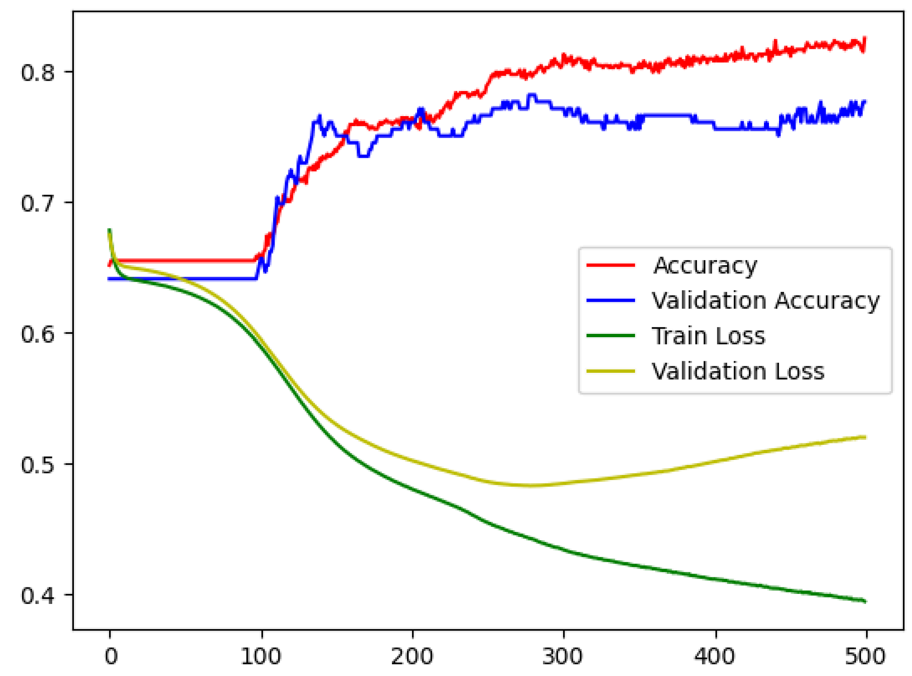
```
6/6 [==============================] - 0s 2ms/step
6/6 [==============================] - 0s 2ms/step
```

```
accuracies = accuracy_score(y_test,y_pred_class)
accuracies
```

```
0.7760416666666666
```

```
#Plot the results of training and validation loss using different learning rates, number of epocgs and network structures
fig, ax = plt.subplots()
ax.plot(model_fit.history["accuracy"],'r', label="Accuracy")
ax.plot(model_fit.history["val_accuracy"],'b',label="Validation Accuracy")
ax.plot(model_fit.history["loss"],'g', label="Train Loss")
ax.plot(model_fit.history["val_loss"],'y', label="Validation Loss")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7e43c0248a90>
```

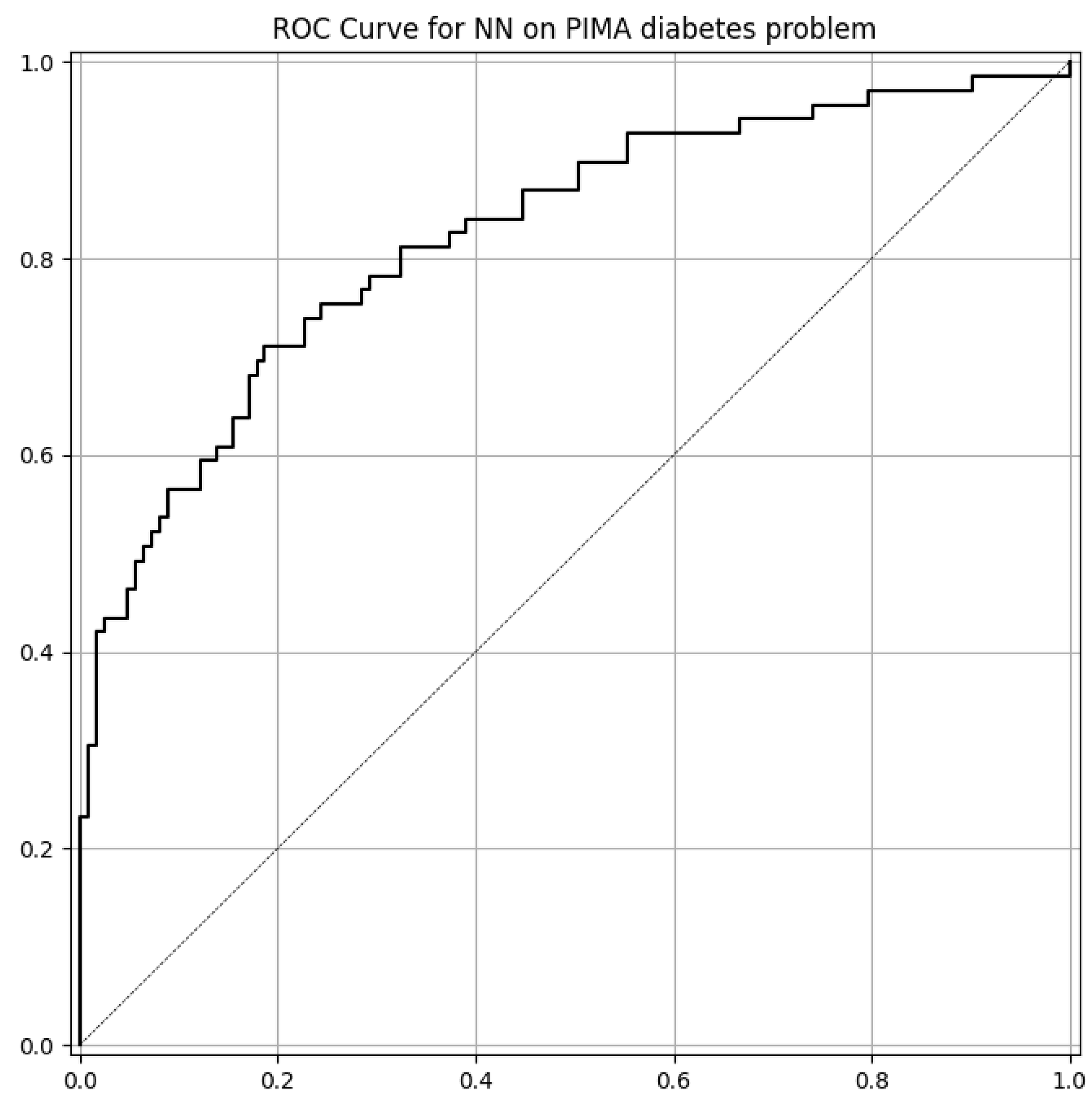



```
#Plot the results of training and validation loss using different learning rates, number of epocgs and network structures
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA diabetes problem'.format(model_name),
           xlim=[-0.01, 1.01], ylim=[-0.01, 1.01])
```

```
print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class)))
print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob)))

plot_roc(y_test, y_pred_prob, 'NN')
```

```
accuracy is 0.776
roc-auc is 0.823
```



Interpret your result

For the first graph shows that the model gives quite good training data and it's also quite efficient. Also, it shows a curve line which means that the values that were being processed and the model is learning. Also, the accuracy and roc-auc score of the model reached for 80% which shows a poor performance regarding training data and predicting its values. From all the data that were processed, the model only reached for about 77.6% in terms of accuracy as well as 81.5% when it comes to roc-auc score.

For the second graph it shows the model also have low performance just like the first model and the accuracy and validation accuracy shows a straight line in the first epochs that means that the model processed an generate the same accuracy level which is not a good indication of the performance of the model. Also, just like the first graph it also has rough curve which shows an inconsistent output value. The accuracy of the ROC-AUC score reach 80% just like the first graph so just like the first graph it also have an average performance regarding training data and predicting its values. From all the data that were processed, the model only reached for about 77.6% in terms of accuracy as well as 82.3% when it comes to ROC-AUC score.

Conclusion

I concluded in this activity that I was able to train neural networks using keras but I'm not confident that I can finish this activity without time and guides and some research but I was able to finish it because of those things. I also concluded that I was able to build and train neural networks and evaluate and plot the models using training and validation loss with this activity mind got broader and I hope that this activity can help me in the future.