```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Activity 1.2 : Training Neural Networks

Name: Jared Miguel F. Roque
Course & Section: CPE 313 - CPE32S8

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 = "/content/drive/MyDrive/Hands-on Activity 2.2 Training Neural Networks/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
674	8	91	82	0	0	35.6	0.587	68	0
35	4	103	60	33	192	24.0	0.966	33	0
475	0	137	84	27	0	27.3	0.231	59	0
248	9	124	70	33	402	35.4	0.282	34	0
192	7	159	66	0	0	30.4	0.383	36	1

```
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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 12)                108       
                                                                 
 dense_1 (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)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
Epoch 1/200
18/18 [==============================] - 1s 18ms/step - loss: 0.9330 - accuracy: 0.3229 - val_loss: 0.8766 - val_accuracy: 0.3698
Epoch 2/200
18/18 [==============================] - 0s 4ms/step - loss: 0.8649 - accuracy: 0.3351 - val_loss: 0.8212 - val_accuracy: 0.3802
Epoch 3/200
18/18 [==============================] - 0s 4ms/step - loss: 0.8126 - accuracy: 0.3559 - val_loss: 0.7787 - val_accuracy: 0.3958
Epoch 4/200
18/18 [==============================] - 0s 5ms/step - loss: 0.7717 - accuracy: 0.3837 - val_loss: 0.7452 - val_accuracy: 0.4583
Epoch 5/200
18/18 [==============================] - 0s 4ms/step - loss: 0.7394 - accuracy: 0.4219 - val_loss: 0.7184 - val_accuracy: 0.4740
Epoch 6/200
18/18 [==============================] - 0s 5ms/step - loss: 0.7132 - accuracy: 0.4670 - val_loss: 0.6968 - val_accuracy: 0.5000
Epoch 7/200
18/18 [==============================] - 0s 5ms/step - loss: 0.6915 - accuracy: 0.5451 - val_loss: 0.6793 - val_accuracy: 0.5781
Epoch 8/200
18/18 [==============================] - 0s 5ms/step - loss: 0.6736 - accuracy: 0.6111 - val_loss: 0.6645 - val_accuracy: 0.6354
Epoch 9/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6583 - accuracy: 0.6667 - val_loss: 0.6519 - val_accuracy: 0.6875
Epoch 10/200
18/18 [==============================] - 0s 5ms/step - loss: 0.6450 - accuracy: 0.6892 - val_loss: 0.6408 - val_accuracy: 0.7083
Epoch 11/200
18/18 [==============================] - 0s 5ms/step - loss: 0.6331 - accuracy: 0.7049 - val_loss: 0.6310 - val_accuracy: 0.7188
Epoch 12/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6224 - accuracy: 0.7222 - val_loss: 0.6223 - val_accuracy: 0.7188
Epoch 13/200
18/18 [==============================] - 0s 5ms/step - loss: 0.6129 - accuracy: 0.7257 - val_loss: 0.6144 - val_accuracy: 0.7188
Epoch 14/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6044 - accuracy: 0.7292 - val_loss: 0.6073 - val_accuracy: 0.7240
Epoch 15/200
18/18 [==============================] - 0s 5ms/step - loss: 0.5967 - accuracy: 0.7378 - val_loss: 0.6007 - val_accuracy: 0.7292
Epoch 16/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5895 - accuracy: 0.7344 - val_loss: 0.5946 - val_accuracy: 0.7292
Epoch 17/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5830 - accuracy: 0.7413 - val_loss: 0.5891 - val_accuracy: 0.7292
Epoch 18/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5770 - accuracy: 0.7431 - val_loss: 0.5839 - val_accuracy: 0.7292
Epoch 19/200
18/18 [==============================] - 0s 5ms/step - loss: 0.5715 - accuracy: 0.7413 - val_loss: 0.5791 - val_accuracy: 0.7240
Epoch 20/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5663 - accuracy: 0.7483 - val_loss: 0.5747 - val_accuracy: 0.7292
Epoch 21/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5615 - accuracy: 0.7517 - val_loss: 0.5705 - val_accuracy: 0.7396
Epoch 22/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5569 - accuracy: 0.7535 - val_loss: 0.5665 - val_accuracy: 0.7448
Epoch 23/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5526 - accuracy: 0.7552 - val_loss: 0.5628 - val_accuracy: 0.7448
Epoch 24/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5485 - accuracy: 0.7552 - val_loss: 0.5593 - val_accuracy: 0.7604
Epoch 25/200
18/18 [==============================] - 0s 3ms/step - loss: 0.5446 - accuracy: 0.7604 - val_loss: 0.5561 - val_accuracy: 0.7604
Epoch 26/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5410 - accuracy: 0.7622 - val_loss: 0.5530 - val_accuracy: 0.7604
Epoch 27/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5375 - accuracy: 0.7639 - val_loss: 0.5502 - val_accuracy: 0.7552
Epoch 28/200
18/18 [==============================] - 0s 5ms/step - loss: 0.5342 - accuracy: 0.7639 - val_loss: 0.5475 - val_accuracy: 0.7552
Epoch 29/200
```

```
## 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(int)
y_pred_prob_nn_1 = model.predict(X_test_norm)
```

```
6/6 [==============================] - 0s 3ms/step
6/6 [==============================] - 0s 3ms/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],
       [1],
       [0],
       [0],
       [1],
       [0]])
```

```
y_pred_prob_nn_1[:10]
```

```
array([[0.5787629 ],
       [0.5951423 ],
       [0.30613333],
       [0.15462056],
       [0.15982619],
       [0.51102567],
       [0.02212371],
       [0.26453504],
       [0.95351917],
       [0.30655366]], dtype=float32)
```

Create the plot_roc function

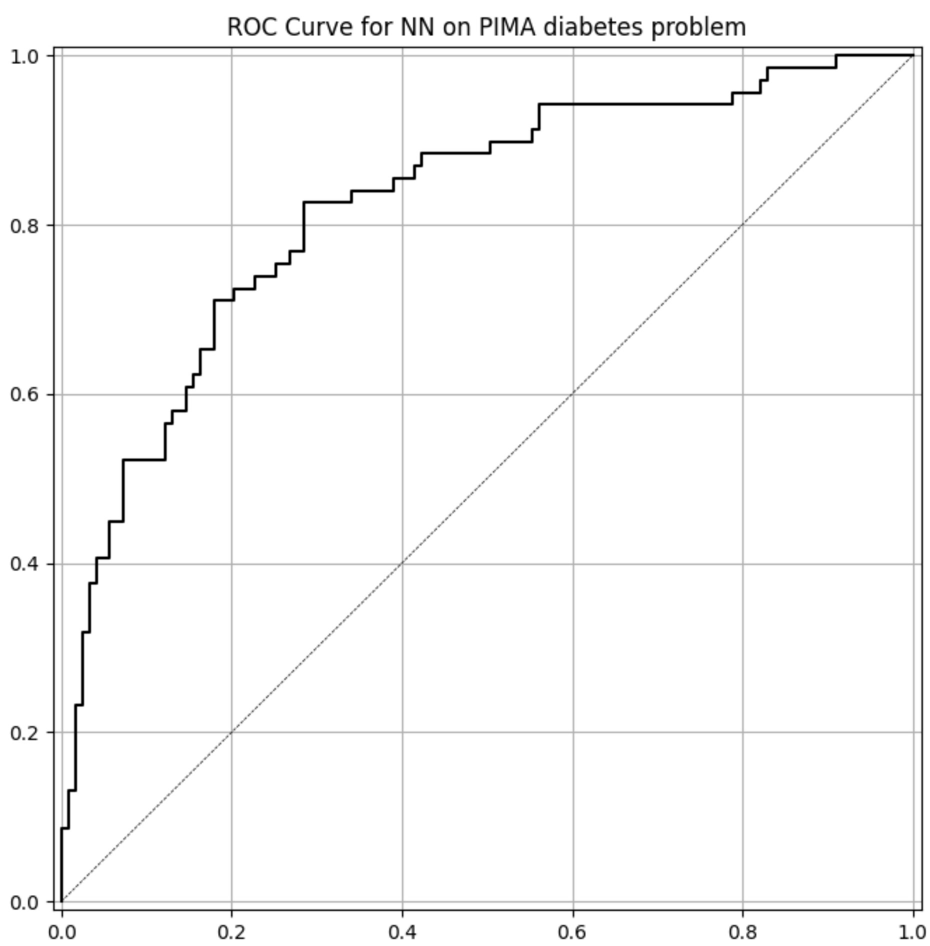
```
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA 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 0.823
```



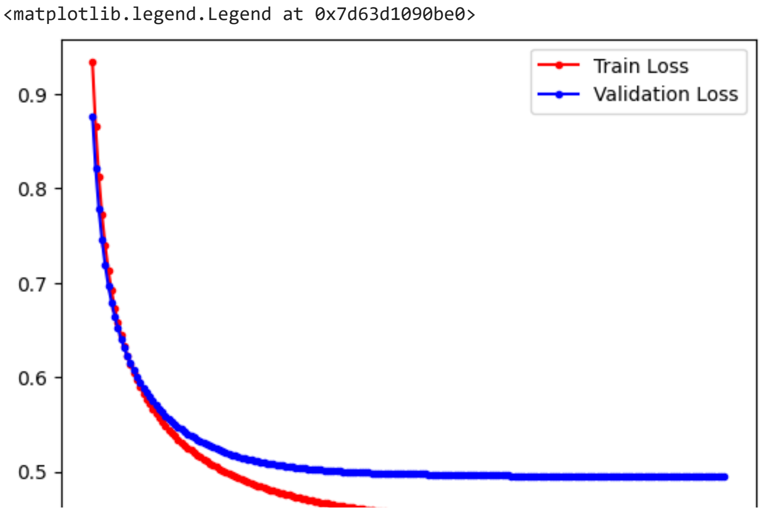
Plot the training loss and the validation loss over the different epochs and see how it looks

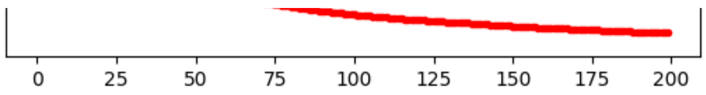
```
run_hist_1.history.keys()
```

```
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 0x7d63d1090be0>
```





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

- The plot shown above tells about the train and validation loss of the used model. The train loss above shows that the trained predicted output has a better result compared to the predicted output without training.

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

```
df = pd.read_csv('/content/drive/MyDrive/Hands-on Activity 2.2 Training Neural Networks/diabetes.csv')
```

```
print(df.shape)
df
```

```
(768, 9)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

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

```
model = Sequential([
    Dense(6, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

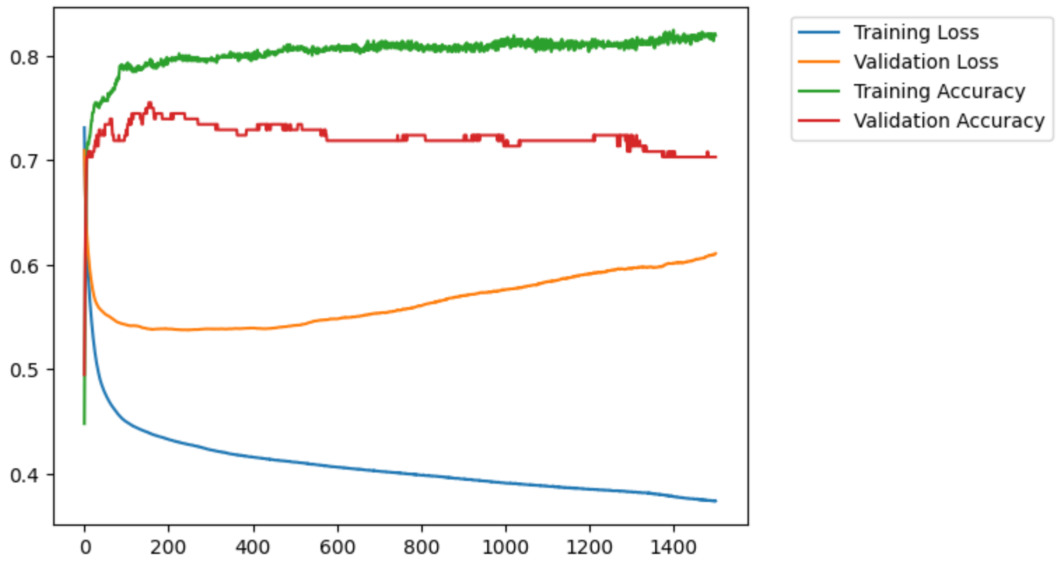
```
model.compile(SGD(lr = .003), 'binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test), epochs=1500)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
Epoch 1/1500
18/18 [==============================] - 1s 20ms/step - loss: 0.7312 - accuracy: 0.4479 - val_loss: 0.7094 - val_accuracy: 0.4948
Epoch 2/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.6982 - accuracy: 0.5243 - val_loss: 0.6862 - val_accuracy: 0.5625
Epoch 3/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.6738 - accuracy: 0.5920 - val_loss: 0.6693 - val_accuracy: 0.6094
Epoch 4/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.6545 - accuracy: 0.6354 - val_loss: 0.6560 - val_accuracy: 0.6406
Epoch 5/1500
18/18 [==============================] - 0s 6ms/step - loss: 0.6390 - accuracy: 0.6580 - val_loss: 0.6451 - val_accuracy: 0.6823
Epoch 6/1500
18/18 [==============================] - 0s 7ms/step - loss: 0.6263 - accuracy: 0.6823 - val_loss: 0.6360 - val_accuracy: 0.6979
Epoch 7/1500
18/18 [==============================] - 0s 7ms/step - loss: 0.6153 - accuracy: 0.6997 - val_loss: 0.6285 - val_accuracy: 0.7031
Epoch 8/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.6060 - accuracy: 0.7135 - val_loss: 0.6223 - val_accuracy: 0.7031
Epoch 9/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.5979 - accuracy: 0.7170 - val_loss: 0.6167 - val_accuracy: 0.7031
Epoch 10/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5906 - accuracy: 0.7153 - val_loss: 0.6118 - val_accuracy: 0.7083
Epoch 11/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5840 - accuracy: 0.7153 - val_loss: 0.6073 - val_accuracy: 0.7083
Epoch 12/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5779 - accuracy: 0.7188 - val_loss: 0.6030 - val_accuracy: 0.7083
Epoch 13/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5719 - accuracy: 0.7222 - val_loss: 0.5989 - val_accuracy: 0.7083
Epoch 14/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.5663 - accuracy: 0.7222 - val_loss: 0.5953 - val_accuracy: 0.7031
Epoch 15/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5609 - accuracy: 0.7257 - val_loss: 0.5919 - val_accuracy: 0.7083
Epoch 16/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5558 - accuracy: 0.7309 - val_loss: 0.5886 - val_accuracy: 0.7031
Epoch 17/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.5507 - accuracy: 0.7344 - val_loss: 0.5858 - val_accuracy: 0.7031
Epoch 18/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5462 - accuracy: 0.7378 - val_loss: 0.5830 - val_accuracy: 0.7031
Epoch 19/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.5418 - accuracy: 0.7378 - val_loss: 0.5804 - val_accuracy: 0.7031
Epoch 20/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5376 - accuracy: 0.7448 - val_loss: 0.5780 - val_accuracy: 0.7031
Epoch 21/1500
18/18 [==============================] - 0s 5ms/step - loss: 0.5337 - accuracy: 0.7448 - val_loss: 0.5756 - val_accuracy: 0.7083
Epoch 22/1500
```

```
18/18 [==============================] - 0s 4ms/step - loss: 0.5297 - accuracy: 0.7465 - val_loss: 0.5735 - val_accuracy: 0.7083
Epoch 23/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5261 - accuracy: 0.7465 - val_loss: 0.5716 - val_accuracy: 0.7083
Epoch 24/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5228 - accuracy: 0.7517 - val_loss: 0.5698 - val_accuracy: 0.7083
Epoch 25/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5194 - accuracy: 0.7535 - val_loss: 0.5682 - val_accuracy: 0.7135
Epoch 26/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5165 - accuracy: 0.7535 - val_loss: 0.5668 - val_accuracy: 0.7135
Epoch 27/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5136 - accuracy: 0.7552 - val_loss: 0.5656 - val_accuracy: 0.7188
Epoch 28/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5111 - accuracy: 0.7552 - val_loss: 0.5646 - val_accuracy: 0.7135
Epoch 29/1500
```

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7d63d1015330>
```




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

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

```
y_pred_class_nn_1[:10]
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [1],
       [1],
       [0]])
```

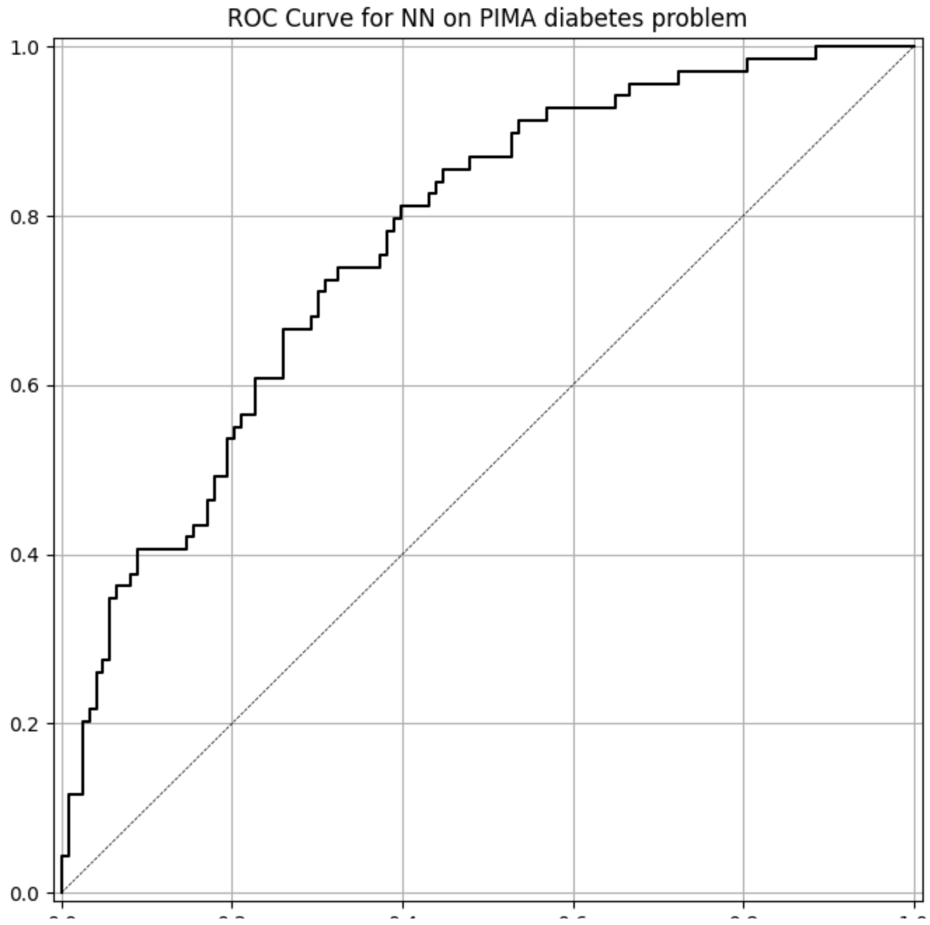
```
y_pred_prob_nn_1[:10]
```

```
array([[4.1216114e-01],
       [6.7286544e-02],
       [1.9415682e-02],
       [3.6249506e-01],
       [2.2468674e-01],
       [6.7347896e-01],
       [4.6577599e-05],
       [7.3379940e-01],
       [7.8834391e-01],
       [4.6811068e-01]], dtype=float32)
```

```
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 0.703
roc-auc is 0.770
```



0.0 0.2 0.4 0.6 0.8 1.0

Conclusion

- This activity is all about building and training neural networks using models with different activation activation functions such as relu and sigmoid. We were tasked to evaluate and plot the models using metrics such as training, validation loss and the ROC curve. In conclusion, this activity helped us in understanding how basic training and testing of models in neural networks using relu and sigmoid activation function with different epochs and learning rates to get the loss functions and accuracy of the training and validation set.

https://colab.research.google.com/drive/1JI-Xa7IqjBtp_XfxMCwVQzxUgME_O06u?usp=sharing