# Name: Gullas Rainer L.

# Section: BSCPE32S3

# Date Perfomred: 04/02/2024

# Date Submitted: 04/02/2024

# Instructor: Engr. Roman Richard


## Activity 6.2 : 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

```
from google.colab import files
files.upload()
```

### 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 | b |
|---|---|---|---|---|---|---|
| **536** | 0 | 105 | 90 | 0 | 0 | 29 |
| **674** | 8 | 91 | 82 | 0 | 0 | 35 |
| **589** | 0 | 73 | 0 | 0 | 0 | 21 |
| **27** | 1 | 97 | 66 | 15 | 140 | 23 |
| **184** | 4 | 141 | 74 | 0 | 0 | 27 |

```
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)

```

```
Epoch 163/200
18/18 [==============================] - 0s 3ms/step - loss: 0.4398 - accuracy: 0.7795 - val_loss: 0.4925 - val_accuracy: 0.7552
Epoch 164/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4397 - accuracy: 0.7778 - val_loss: 0.4925 - val_accuracy: 0.7552
Epoch 165/200
18/18 [==============================] - 0s 4ms/step - loss: 0.4396 - accuracy: 0.7778 - val_loss: 0.4925 - val_accuracy: 0.7552
Epoch 166/200
```

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

```
6/6 [==============================] - 0s 4ms/step
6/6 [==============================] - 0s 5ms/step
```

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

```
array([[0.05151687],
       [0.33694816],
       [0.3720943 ],
       [0.22370118],
       [0.5326897 ],
       [0.34477696],
       [0.318967  ],
       [0.3697302 ],
       [0.05910133],
       [0.24207284]], dtype=float32)
```

```
y_pred_prob_nn_1[:10]
```

```
array([[0.05151687],
       [0.33694816],
       [0.3720943 ],
       [0.22370118],
       [0.5326897 ],
       [0.34477696],
       [0.318967  ],
       [0.3697302 ],
       [0.05910133],
       [0.24207284]], 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')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-54-1f0a10ee7606> in <cell line: 1>()
----> 1 print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class_nn_1)))
      2 print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob_nn_1)))
      3
      4 plot_roc(y_test, y_pred_prob_nn_1, 'NN')

                          2 frames
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py in
_check_targets(y_true, y_pred)
     93
     94     if len(y_type) > 1:
---> 95         raise ValueError(
     96             "Classification metrics can't handle a mix of {0} and {1}
targets".format(
     97                 type_true, type_pred

ValueError: Classification metrics can't handle a mix of binary and continuous targets
```

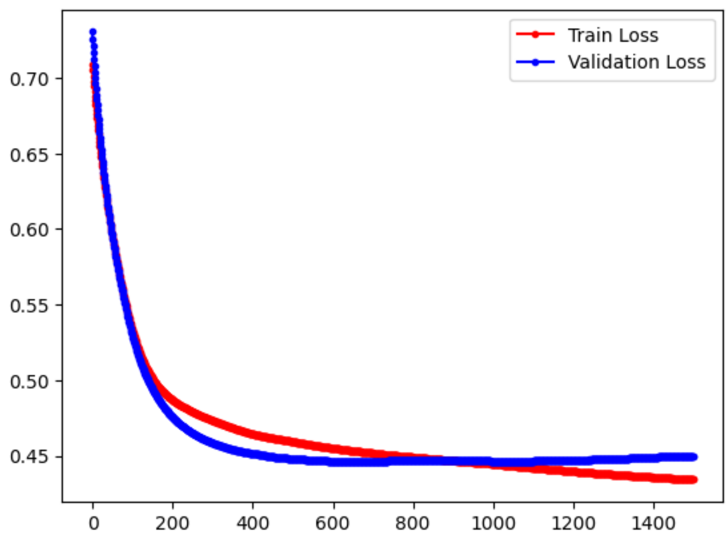
Plot the training loss and the validation loss over the different epochs and see how it looks

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

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

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

```
<matplotlib.legend.Legend at 0x7bc849ed6290>
```




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

base on the result of the result of the train and validation loss since its in a downward trend and doing some research on what is the meaning of this it is said that if it is in a downward trend is that the model is learning and improving its performance over epochs. This is a desirable behavior during the training phase of a machine learning model.

## 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=50000)
```

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

```
(0.3489583333333333, 0.6510416666666666)
```

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

```
model  = Sequential([
    Dense(6, input_shape=(8,), activation="relu"),
    Dense(6, activation="relu"),
    Dense(1, activation="sigmoid")
])
```

```
model.summary()
```

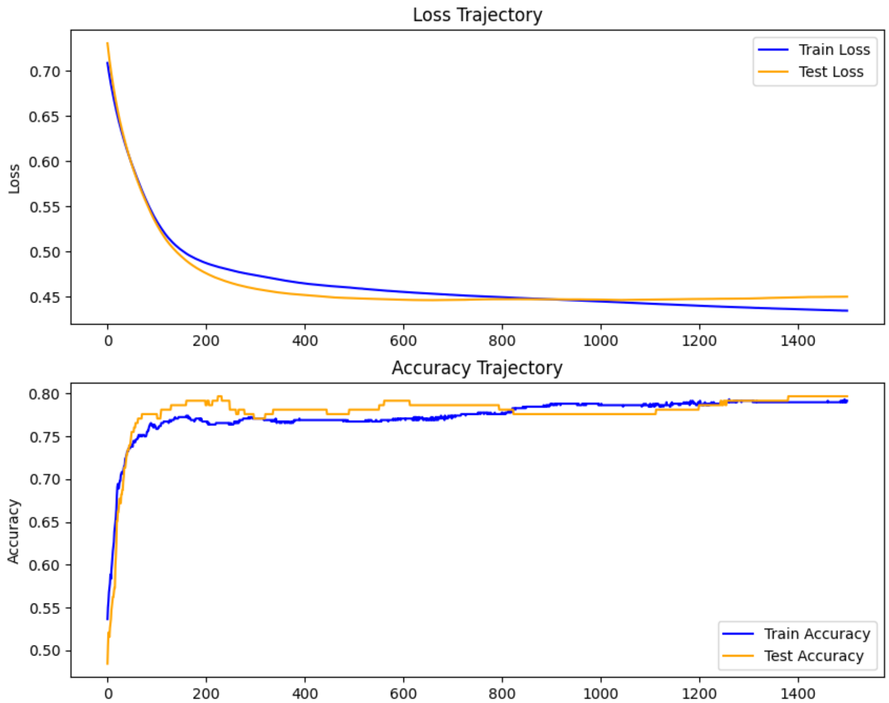
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 6)                 54

 dense_7 (Dense)             (None, 6)                 42

 dense_8 (Dense)             (None, 2)                 14

=================================================================
Total params: 110 (440.00 Byte)
Trainable params: 110 (440.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(SGD(learning_rate=0.003), "binary_crossentropy", metrics=["accuracy"])

run_hist_1 = model.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test), epochs=1500)
```

```
Epoch 100/1500
18/18 [==============================] - 0s 3ms/step - loss: 0.5342 - accuracy: 0.7587 - val_loss: 0.5305 - val_accuracy: 0.7760
Epoch 101/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5333 - accuracy: 0.7587 - val_loss: 0.5295 - val_accuracy: 0.7760
Epoch 102/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5324 - accuracy: 0.7587 - val_loss: 0.5286 - val_accuracy: 0.7708
Epoch 103/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5315 - accuracy: 0.7587 - val_loss: 0.5276 - val_accuracy: 0.7708
Epoch 104/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5306 - accuracy: 0.7604 - val_loss: 0.5266 - val_accuracy: 0.7708
Epoch 105/1500
18/18 [==============================] - 0s 3ms/step - loss: 0.5297 - accuracy: 0.7604 - val_loss: 0.5256 - val_accuracy: 0.7708
Epoch 106/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5289 - accuracy: 0.7622 - val_loss: 0.5247 - val_accuracy: 0.7708
Epoch 107/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5280 - accuracy: 0.7622 - val_loss: 0.5238 - val_accuracy: 0.7708
Epoch 108/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5272 - accuracy: 0.7622 - val_loss: 0.5229 - val_accuracy: 0.7708
Epoch 109/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5264 - accuracy: 0.7622 - val_loss: 0.5220 - val_accuracy: 0.7760
Epoch 110/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5256 - accuracy: 0.7656 - val_loss: 0.5211 - val_accuracy: 0.7812
Epoch 111/1500
18/18 [==============================] - 0s 3ms/step - loss: 0.5248 - accuracy: 0.7656 - val_loss: 0.5202 - val_accuracy: 0.7812
Epoch 112/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5239 - accuracy: 0.7656 - val_loss: 0.5193 - val_accuracy: 0.7812
Epoch 113/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5231 - accuracy: 0.7656 - val_loss: 0.5185 - val_accuracy: 0.7812
Epoch 114/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5223 - accuracy: 0.7656 - val_loss: 0.5176 - val_accuracy: 0.7812
Epoch 115/1500
18/18 [==============================] - 0s 4ms/step - loss: 0.5215 - accuracy: 0.7674 - val_loss: 0.5168 - val_accuracy: 0.7812
Epoch 116/1500
```

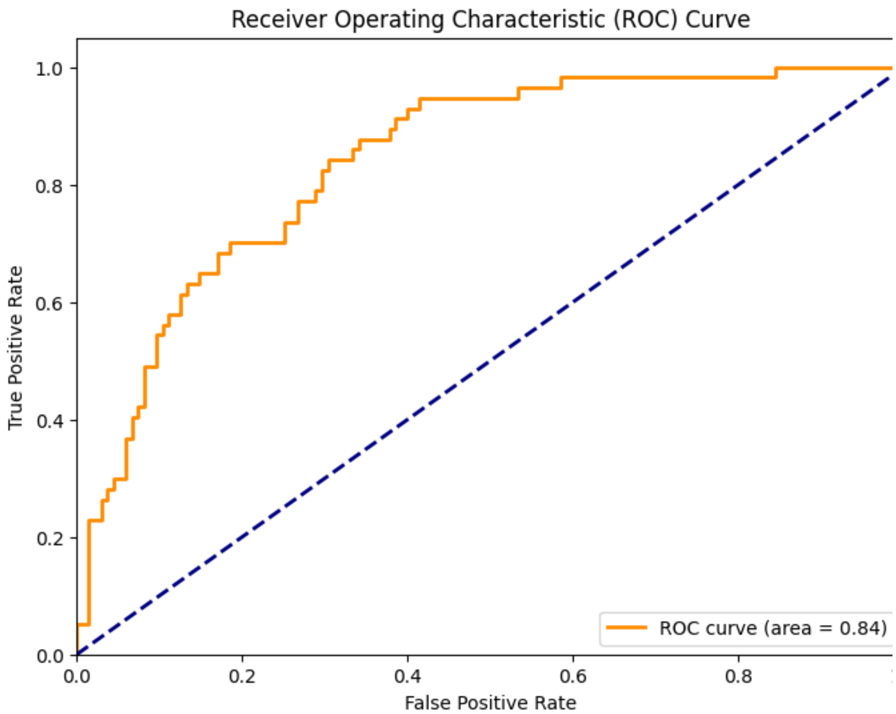
```python
train_loss = run_hist_1.history['loss']
test_loss = run_hist_1.history['val_loss']
train_accuracy = run_hist_1.history['accuracy']
test_accuracy = run_hist_1.history['val_accuracy']

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))

axes[1].plot(train_accuracy, label='Train Accuracy', color='blue')
axes[1].plot(test_accuracy, label='Test Accuracy', color='orange')
axes[1].set_title('Accuracy Trajectory')
axes[1].set_ylabel('Accuracy')
axes[1].legend()

axes[0].plot(train_loss, label='Train Loss', color='blue')
axes[0].plot(test_loss, label='Test Loss', color='orange')
axes[0].set_title('Loss Trajectory')
axes[0].set_ylabel('Loss')
axes[0].legend()
```

<matplotlib.legend.Legend at 0x7bc838860460>




```
from sklearn.metrics import roc_curve, auc

y_pred_proba = model.predict(X_test_norm)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

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



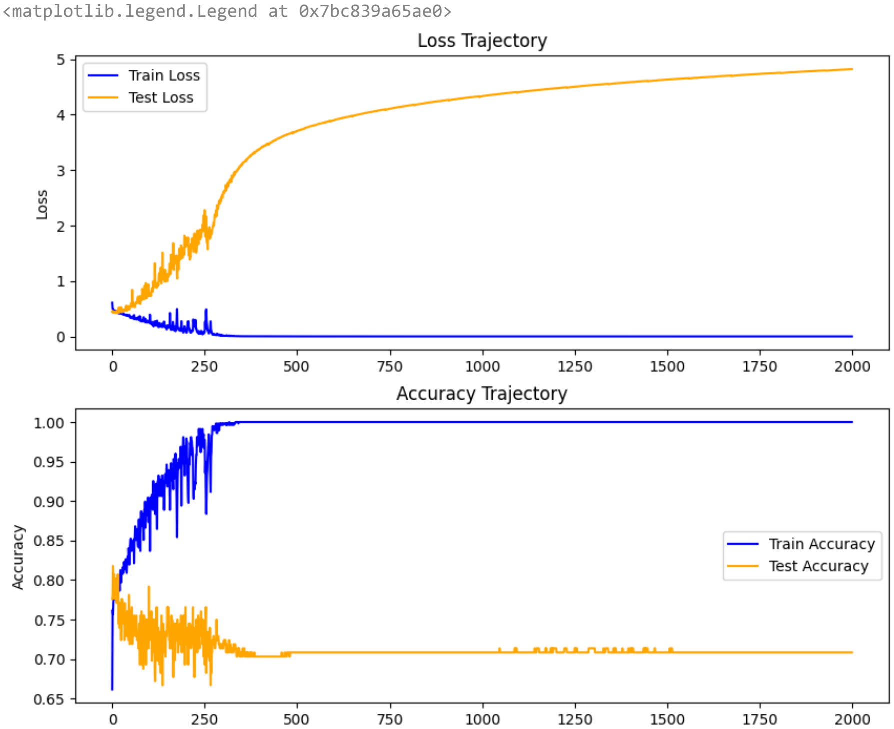

```
model  = Sequential([
    Dense(20, input_shape=(8,), activation="relu"),
    Dense(20, activation="relu"),
    Dense(1, activation="sigmoid")
])
```

```
model.compile(SGD(learning_rate=0.5), "binary_crossentropy", metrics=["accuracy"])

run_hist_1 = model.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test), epochs=2000)
```

```
Epoch 1597/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.8026e-04 - accuracy: 1.0000 - val_loss: 4.6717 - val_accuracy: 0.7083
Epoch 1598/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.8008e-04 - accuracy: 1.0000 - val_loss: 4.6723 - val_accuracy: 0.7083
Epoch 1599/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7987e-04 - accuracy: 1.0000 - val_loss: 4.6734 - val_accuracy: 0.7083
Epoch 1600/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7983e-04 - accuracy: 1.0000 - val_loss: 4.6733 - val_accuracy: 0.7083
Epoch 1601/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7963e-04 - accuracy: 1.0000 - val_loss: 4.6738 - val_accuracy: 0.7083
Epoch 1602/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7938e-04 - accuracy: 1.0000 - val_loss: 4.6742 - val_accuracy: 0.7083
Epoch 1603/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7931e-04 - accuracy: 1.0000 - val_loss: 4.6747 - val_accuracy: 0.7083
Epoch 1604/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7915e-04 - accuracy: 1.0000 - val_loss: 4.6753 - val_accuracy: 0.7083
Epoch 1605/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7893e-04 - accuracy: 1.0000 - val_loss: 4.6759 - val_accuracy: 0.7083
Epoch 1606/2000
18/18 [==============================] - 0s 5ms/step - loss: 1.7878e-04 - accuracy: 1.0000 - val_loss: 4.6758 - val_accuracy: 0.7083
Epoch 1607/2000
```

```
train_loss = run_hist_1.history['loss']
test_loss = run_hist_1.history['val_loss']
train_accuracy = run_hist_1.history['accuracy']
test_accuracy = run_hist_1.history['val_accuracy']

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))

axes[1].plot(train_accuracy, label='Train Accuracy', color='blue')
axes[1].plot(test_accuracy, label='Test Accuracy', color='orange')
axes[1].set_title('Accuracy Trajectory')
axes[1].set_ylabel('Accuracy')
axes[1].legend()

axes[0].plot(train_loss, label='Train Loss', color='blue')
axes[0].plot(test_loss, label='Test Loss', color='orange')
axes[0].set_title('Loss Trajectory')
axes[0].set_ylabel('Loss')
axes[0].legend()
```

```
<matplotlib.legend.Legend at 0x7bc839a65ae0>
```



```
y_pred_proba = model.predict(X_test_norm)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

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

Receiver Operating Characteristic (ROC) Curve