In [1]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
import collections
import time
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
```

In [3]:
```
def set_seeds(seed):
    np.random.seed(seed)
    random.seed(seed)
    tf.random.set_seed(seed)

# Call this function before creating and training your model
set_seeds(42)
```

In [4]:
```
# CONSTANT
LEARNING_RATE_VAL = 0.0001
BATCH_SIZE_VAL = 32
VALIDATION_SPLIT_VAL = 0.2
EPOCH_NUMBER = 50
```

In [5]:
```
early_stopping = EarlyStopping(
    monitor='val_loss',       # Metric to monitor
    patience=1,              # Number of epochs to wait for improvement
    restore_best_weights=True # Restore model weights from the epoch with the best value
)
```

load dataset

In [6]:
```
def load_np_array(file_name):
    X_array = np.load('data/X_' + file_name + '_array.npy')
    y_array = np.load('data/y_' + file_name + '_array.npy')
    return X_array, y_array

X_train_fall, y_train_fall = load_np_array("train_fall")
X_train_notfall, y_train_notfall = load_np_array("train_notfall")
X_test_fall, y_test_fall = load_np_array("test_fall")
X_test_notfall, y_test_notfall = load_np_array("test_notfall")
```

In [7]:
```
print(X_train_fall.shape)
print(y_train_fall.shape)
print(X_train_notfall.shape)
print(y_train_notfall.shape)
```

```
(2912, 40, 3)
(2912,)
(2912, 40, 3)
(2912,)
```

In [8]:
```
print(X_test_fall.shape)
print(y_test_fall.shape)
print(X_test_notfall.shape)
print(y_test_notfall.shape)
```

```
(1456, 40, 3)
(1456,)
(1456, 40, 3)
(1456,)
```

In [9]:
```
y_test_fall
```

Out[9]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [10]:
```
# Combine fall and non-fall data for training
X_train = np.concatenate((X_train_fall, X_train_notfall), axis=0)
y_train = np.concatenate((y_train_fall, y_train_notfall), axis=0)
```

```
# Combine fall and non-fall data for testing
X_test = np.concatenate((X_test_fall, X_test_notfall), axis=0)
y_test = np.concatenate((y_test_fall, y_test_notfall), axis=0)

# Shuffle the training data
train_indices = np.arange(X_train.shape[0])
np.random.shuffle(train_indices)
X_train = X_train[train_indices]
y_train = y_train[train_indices]

# Shuffle the testing data
test_indices = np.arange(X_test.shape[0])
np.random.shuffle(test_indices)
X_test = X_test[test_indices]
y_test = y_test[test_indices]

print("Combined training data shape:", X_train.shape, y_train.shape)
print("Combined testing data shape:", X_test.shape, y_test.shape)
```

```
Combined training data shape: (5824, 40, 3) (5824,)
Combined testing data shape: (2912, 40, 3) (2912,)
```

In [11]:

```
# Define the GRU model
model = Sequential()
model.add(GRU(256, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=LEARNING_RATE_VAL), loss='binary_crossentropy', metrics=['accura
```

```
2024-08-05 00:05:57.469329: I metal_plugin/src/device/metal_device.cc:1154] Metal device set to: Apple M1 P
ro
2024-08-05 00:05:57.469349: I metal_plugin/src/device/metal_device.cc:296] systemMemory: 16.00 GB
2024-08-05 00:05:57.469354: I metal_plugin/src/device/metal_device.cc:313] maxCacheSize: 5.33 GB
2024-08-05 00:05:57.469367: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:3
05] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built
with NUMA support.
2024-08-05 00:05:57.469377: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:2
71] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical P
luggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
/opt/miniconda3/envs/tensorflow/lib/python3.10/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

In [12]:

```python
# Train the model
history = model.fit(X_train, y_train, epochs=EPOCH_NUMBER, batch_size=BATCH_SIZE_VAL, validation_split=VAL
```

```
Epoch 1/50
```

```
2024-08-05 00:05:57.956200: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117] P
lugin optimizer for device_type GPU is enabled.
```

```
146/146 ━━━━━━━━━━━━━━━━━━━━ 4s 19ms/step - accuracy: 0.7160 - loss: 0.6324 - val_accuracy: 0.8120 - val_lo
ss: 0.4931
Epoch 2/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.7886 - loss: 0.5089 - val_accuracy: 0.8155 - val_lo
ss: 0.4465
Epoch 3/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.8353 - loss: 0.4120 - val_accuracy: 0.8670 - val_lo
ss: 0.3320
Epoch 4/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.8911 - loss: 0.2961 - val_accuracy: 0.8944 - val_lo
ss: 0.2878
Epoch 5/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.9142 - loss: 0.2485 - val_accuracy: 0.9039 - val_lo
ss: 0.2484
Epoch 6/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.9249 - loss: 0.2193 - val_accuracy: 0.9099 - val_lo
ss: 0.2297
Epoch 7/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9313 - loss: 0.2057 - val_accuracy: 0.9227 - val_lo
ss: 0.2075
Epoch 8/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9335 - loss: 0.1891 - val_accuracy: 0.9253 - val_lo
ss: 0.2051
Epoch 9/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9355 - loss: 0.1777 - val_accuracy: 0.9348 - val_lo
ss: 0.1816
Epoch 10/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9501 - loss: 0.1645 - val_accuracy: 0.9348 - val_lo
ss: 0.1665
Epoch 11/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9521 - loss: 0.1544 - val_accuracy: 0.9399 - val_lo
ss: 0.1557
Epoch 12/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9576 - loss: 0.1435 - val_accuracy: 0.9468 - val_lo
ss: 0.1361
Epoch 13/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9595 - loss: 0.1322 - val_accuracy: 0.9485 - val_lo
ss: 0.1347
Epoch 14/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9644 - loss: 0.1257 - val_accuracy: 0.9519 - val_lo
ss: 0.1268
Epoch 15/50
```

```
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9675 - loss: 0.1144 - val_accuracy: 0.9511 - val_lo
ss: 0.1251
Epoch 16/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9685 - loss: 0.1159 - val_accuracy: 0.9528 - val_lo
ss: 0.1199
Epoch 17/50
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.9693 - loss: 0.1059 - val_accuracy: 0.9519 - val_lo
ss: 0.1205
```

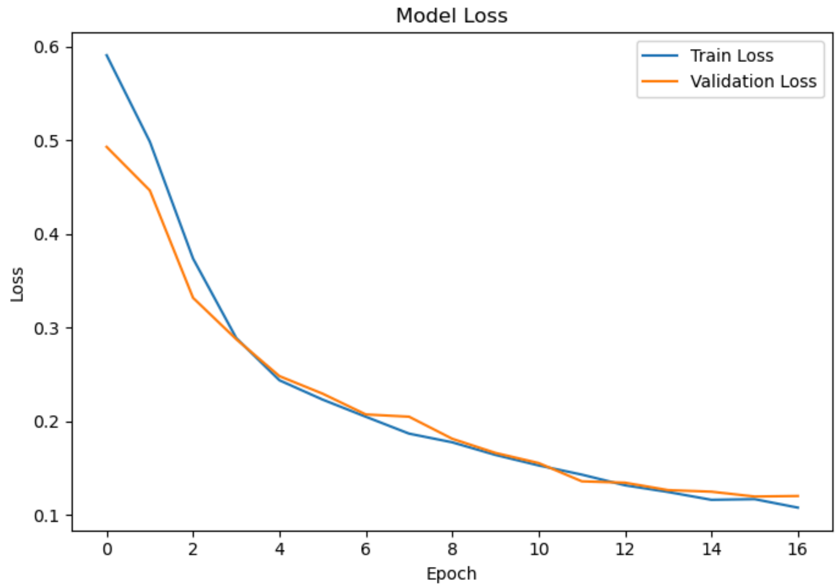
In [13]:

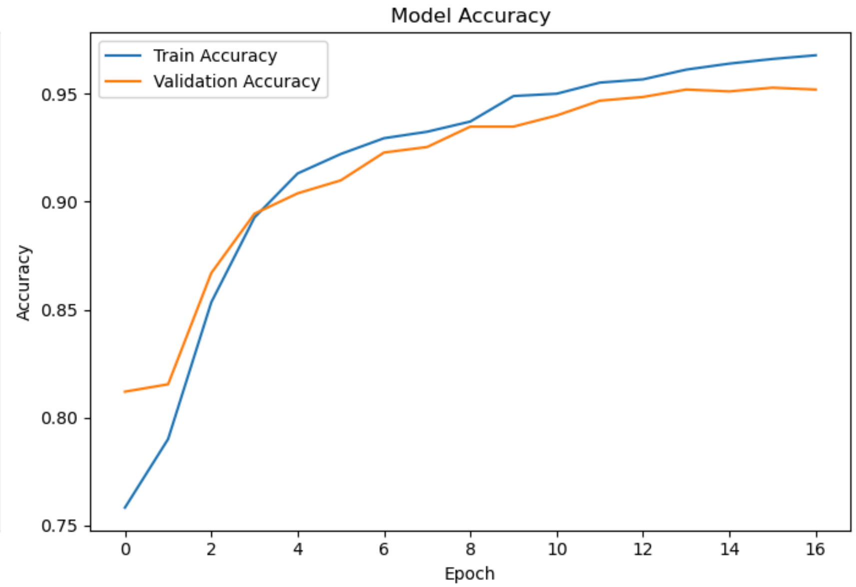
```
# Plot training & validation loss values
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')

# Plot training & validation accuracy values
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```





In [14]:

```
# Predict on the test set
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Accuracy: 0.9543
Precision: 0.9522
Recall: 0.9567
F1 Score: 0.9544
```

In [15]:

```
# Generate and print classification report
report = classification_report(y_test, y_pred, target_names=['Not Fall', 'Fall'])
print("\nClassification Report:\n", report)
```

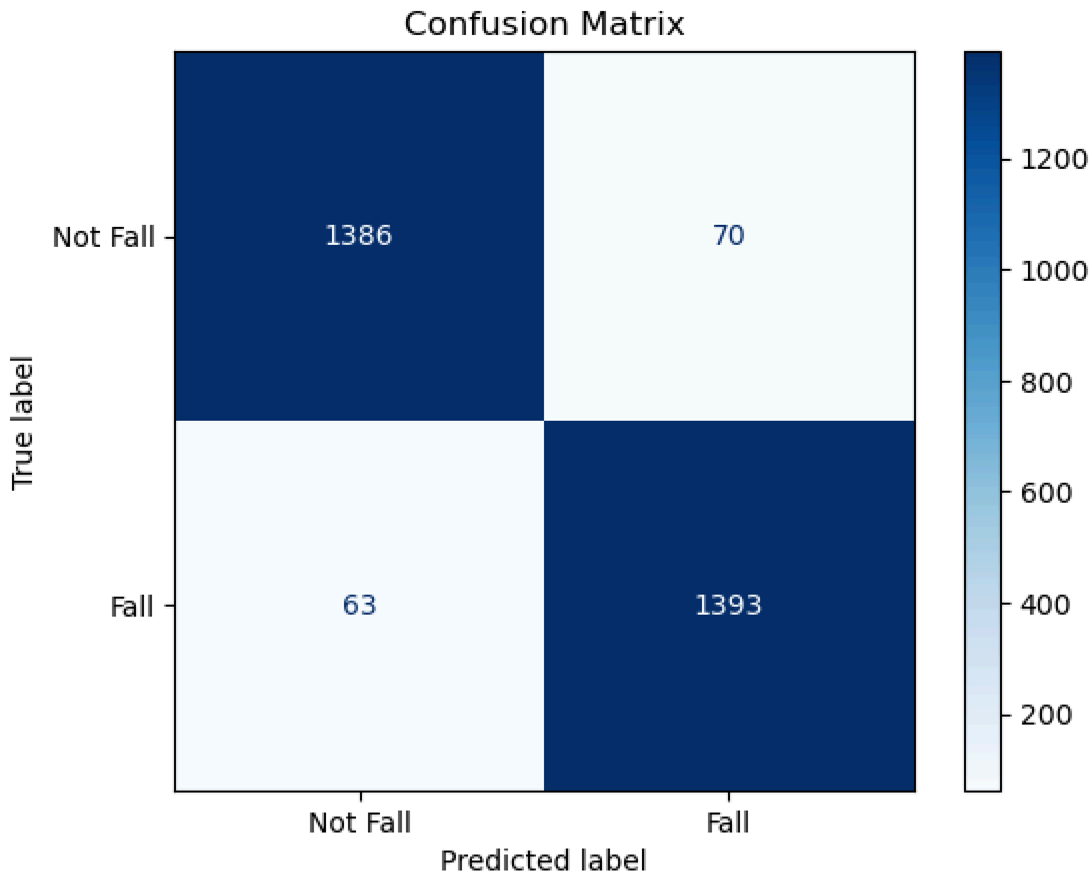
```
Classification Report:
              precision    recall  f1-score   support

    Not Fall       0.96      0.95      0.95      1456
        Fall       0.95      0.96      0.95      1456

    accuracy                           0.95      2912
   macro avg       0.95      0.95      0.95      2912
weighted avg       0.95      0.95      0.95      2912
```

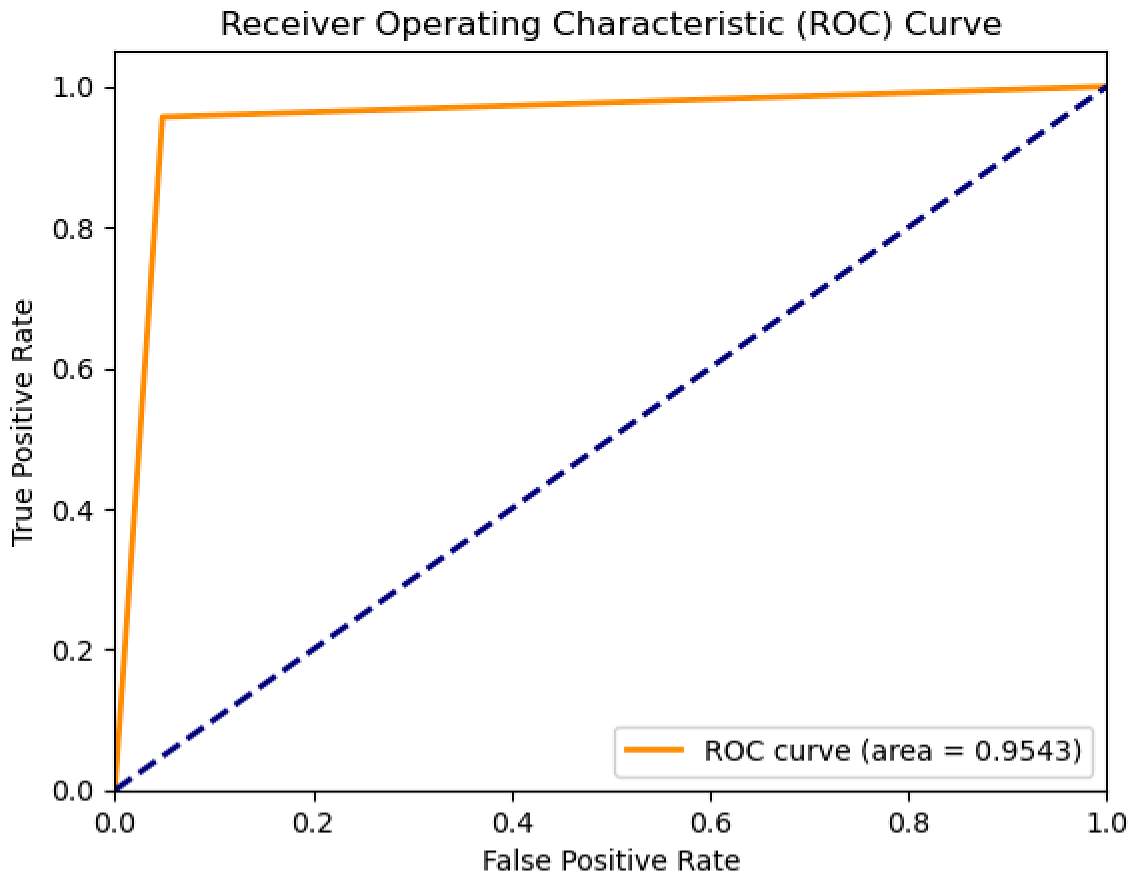
In [16]:

```
# Compute and plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Not Fall', 'Fall'])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```



In [17]:
```
# Plot ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.4f)' % 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()
```


Receiver Operating Characteristic (ROC) Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.9543)

```
In [ ]:
```