```
import json
from sklearn.model_selection import train_test_split

#Reading all the signatures and loading them into final_data
with open('file_70_8_all_cords.txt', 'r') as file:
    file_contents = file.read()
    final_data = json.loads(file_contents)

test_size = 0.3

train_data = {}
test_data = {}

for user in final_data:
    # Split Genuine signatures into training and testing sets
    gen_train, gen_test = train_test_split(final_data[user][1][0], test_size=test_size, random_state=42, stratify

    # Split Forgery signatures into training and testing sets
    for_train, for_test = train_test_split(final_data[user][0][0], test_size=test_size, random_state=42, stratify

    # Combine the training and testing sets for each signature type
    train_signs = [for_train, gen_train]
    test_signs = [for_test, gen_test]

    # Add the user's training and testing data to the appropriate dictionaries
    train_data[user] = train_signs
    test_data[user] = test_signs
```

```
pip install fastdtw
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting fastdtw
  Downloading fastdtw-0.3.4.tar.gz (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.4/133.4 kB 14.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from fastdtw) (1.22.4)
Building wheels for collected packages: fastdtw
  Building wheel for fastdtw (setup.py) ... done
  Created wheel for fastdtw: filename=fastdtw-0.3.4-cp310-cp310-linux_x86_64.whl size=517900 sha256=e519dff
  Stored in directory: /root/.cache/pip/wheels/73/c8/f7/c25448dab74c3acf4848bc25d513c736bb93910277e1528ef4
Successfully built fastdtw
Installing collected packages: fastdtw
Successfully installed fastdtw-0.3.4
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Dropout, LSTM, Bidirectional, Concatenate, TimeDistributed, Res
from tensorflow.keras.preprocessing.sequence import pad_sequences
from fastdtw import fastdtw

# Calculate DTW distance between two signatures
def dtw_distance(signature1, signature2):
    distance, _ = fastdtw(signature1, signature2)
    return distance

# Prepare data for training and testing
def prepare_data(data,max_seq_length):
    X = []
    y = []
```

```
    for user, signatures in data.items():
      forgeries, genuine = signatures
      # Pair each forgery with each genuine signature for comparison
      for forgery in forgeries:
          for gen_signature in genuine:
              dtw_dist = dtw_distance(forgery, gen_signature)
              X.append((dtw_dist, forgery, gen_signature))
              y.append(0)

      # Pair each pair of genuine signatures for comparison
      for i in range(len(genuine)):
          for j in range(i + 1, len(genuine)):
              dtw_dist = dtw_distance(genuine[i], genuine[j])
              X.append((dtw_dist, genuine[i], genuine[j]))
              y.append(1)

    X_padded = []

    # Pad sequences to ensure equal length
    for dtw_dist, sig1, sig2 in X:
        sig1_padded = pad_sequences([sig1], maxlen=max_seq_length, padding='post', dtype='float32')[0]
        sig2_padded = pad_sequences([sig2], maxlen=max_seq_length, padding='post', dtype='float32')[0]
        X_padded.append((dtw_dist, sig1_padded, sig2_padded))

    return X_padded, np.array(y)

# Determine maximum sequence length
max_seq_length_train = max([len(signature) for user_signatures in train_data.values() for signature_set in user_s
max_seq_length_test = max([len(signature) for user_signatures in test_data.values() for signature_set in user_sig
max_seq_length = max(max_seq_length_train, max_seq_length_test)

X_train, y_train = prepare_data(train_data, max_seq_length)
X_test, y_test = prepare_data(test_data, max_seq_length)
max_seq_length = X_train[0][1].shape[0]


# Building TARNN model
def build_model(input_shape_dtw, input_shape_sig):
    """
    Build the TARNN (Temporal Attention-based Recurrent Neural Network) model.

    Args:
    - input_shape_dtw: Shape of the input DTW distance.
    - input_shape_sig: Shape of the input signature.

    Returns:
    - model: Compiled TARNN model.

    Description:
    The TARNN model takes three inputs: DTW distance, input_signature_1, and input_signature_2.
    It consists of a Bidirectional LSTM layer with attention mechanism to weight the information.
    The attention weights are calculated using a TimeDistributed Dense layer followed by softmax activation.
    The weighted information is then combined using element-wise multiplication.
    The output from this layer is passed through a final LSTM layer, followed by dropout, and a dense layer
    with sigmoid activation for binary classification.

    """
    input_dtw = Input(shape=input_shape_dtw)
    input_sig1 = Input(shape=input_shape_sig)
    input_sig2 = Input(shape=input_shape_sig)
```

```
    sig1_3d = Reshape((-1, 1))(input_sig1)
    sig2_3d = Reshape((-1, 1))(input_sig2)

    inputs = Concatenate(axis=-1)([sig1_3d, sig2_3d])

    lstm = Bidirectional(LSTM(units=32, return_sequences=True))(inputs)

    # Attention mechanism for weighting information
    attn_weights = TimeDistributed(Dense(1, activation='tanh'))(lstm)
    attn_weights = tf.keras.layers.Flatten()(attn_weights)
    attn_weights = tf.keras.layers.Activation('softmax')(attn_weights)
    attn_weights = tf.keras.layers.RepeatVector(32 * 2)(attn_weights)
    attn_weights = tf.keras.layers.Permute([2, 1])(attn_weights)

    attn = Multiply()([lstm, attn_weights])
    attn = tf.keras.layers.Lambda(lambda x: tf.keras.backend.sum(x, axis=1))(attn)
    attn = Reshape((1, 32 * 2))(attn)

    lstm2 = LSTM(units=16)(attn)

    dropout = Dropout(rate=0.5)(lstm2)

    output = Dense(1, activation='sigmoid')(dropout)

    model = tf.keras.Model(inputs=[input_dtw, input_sig1, input_sig2], outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model


# Prepare the input data
X_train_dtw = np.array([x[0] for x in X_train]).reshape(-1, 1)
X_train_sig1 = np.array([x[1] for x in X_train])
X_train_sig2 = np.array([x[2] for x in X_train])

X_test_dtw = np.array([x[0] for x in X_test]).reshape(-1, 1)
X_test_sig1 = np.array([x[1] for x in X_test])
X_test_sig2 = np.array([x[2] for x in X_test])


# Build and compile the model
model = build_model(X_train_dtw.shape[1:], X_train_sig1.shape[1:])

# Train the model
history = model.fit([X_train_dtw, X_train_sig1, X_train_sig2], y_train, validation_split=0.1, epochs=30, batch_si

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate([X_test_dtw, X_test_sig1, X_test_sig2], y_test)
print("Test accuracy: ", test_accuracy)
```

```
12/12 [==============================] - 8s 647ms/step - loss: 0.5449 - accuracy: 0.8016 - val_loss: 0.514
Epoch 4/30
12/12 [==============================] - 8s 670ms/step - loss: 0.5167 - accuracy: 0.8016 - val_loss: 0.502
Epoch 5/30
12/12 [==============================] - 8s 719ms/step - loss: 0.4832 - accuracy: 0.8016 - val_loss: 0.481
Epoch 6/30
12/12 [==============================] - 8s 630ms/step - loss: 0.4722 - accuracy: 0.8016 - val_loss: 0.437
```

```
12/12 [==============================] - 7s 629ms/step - loss: 0.3981 - accuracy: 0.8016 - val_loss: 0.337
Epoch 10/30
12/12 [==============================] - 8s 708ms/step - loss: 0.3541 - accuracy: 0.8043 - val_loss: 0.312
Epoch 11/30
12/12 [==============================] - 8s 712ms/step - loss: 0.3448 - accuracy: 0.8150 - val_loss: 0.309
Epoch 12/30
12/12 [==============================] - 8s 641ms/step - loss: 0.3043 - accuracy: 0.8660 - val_loss: 0.311
Epoch 13/30
12/12 [==============================] - 8s 707ms/step - loss: 0.3021 - accuracy: 0.8740 - val_loss: 0.270
Epoch 14/30
12/12 [==============================] - 8s 709ms/step - loss: 0.2927 - accuracy: 0.9008 - val_loss: 0.231
Epoch 15/30
12/12 [==============================] - 8s 638ms/step - loss: 0.2753 - accuracy: 0.9062 - val_loss: 0.205
Epoch 16/30
12/12 [==============================] - 8s 700ms/step - loss: 0.2639 - accuracy: 0.9142 - val_loss: 0.261
Epoch 17/30
12/12 [==============================] - 8s 707ms/step - loss: 0.2528 - accuracy: 0.9196 - val_loss: 0.244
Epoch 18/30
12/12 [==============================] - 8s 631ms/step - loss: 0.2543 - accuracy: 0.9062 - val_loss: 0.164
Epoch 19/30
12/12 [==============================] - 8s 703ms/step - loss: 0.2451 - accuracy: 0.9169 - val_loss: 0.236
Epoch 20/30
12/12 [==============================] - 8s 707ms/step - loss: 0.2130 - accuracy: 0.9196 - val_loss: 0.238
Epoch 21/30
12/12 [==============================] - 8s 632ms/step - loss: 0.1958 - accuracy: 0.9196 - val_loss: 0.214
Epoch 22/30
12/12 [==============================] - 8s 706ms/step - loss: 0.2231 - accuracy: 0.9196 - val_loss: 0.190
Epoch 23/30
12/12 [==============================] - 8s 693ms/step - loss: 0.2593 - accuracy: 0.8874 - val_loss: 0.174
Epoch 24/30
12/12 [==============================] - 8s 632ms/step - loss: 0.2436 - accuracy: 0.8874 - val_loss: 0.416
Epoch 25/30
12/12 [==============================] - 8s 702ms/step - loss: 0.2433 - accuracy: 0.9062 - val_loss: 0.236
Epoch 26/30
12/12 [==============================] - 8s 710ms/step - loss: 0.2008 - accuracy: 0.9249 - val_loss: 0.244
Epoch 27/30
12/12 [==============================] - 8s 646ms/step - loss: 0.2159 - accuracy: 0.9223 - val_loss: 0.261
Epoch 28/30
12/12 [==============================] - 8s 708ms/step - loss: 0.2127 - accuracy: 0.9142 - val_loss: 0.143
Epoch 29/30
12/12 [==============================] - 8s 674ms/step - loss: 0.1990 - accuracy: 0.9249 - val_loss: 0.201
Epoch 30/30
12/12 [==============================] - 8s 632ms/step - loss: 0.1790 - accuracy: 0.9276 - val_loss: 0.183
13/13 [==============================] - 4s 279ms/step - loss: 0.1657 - accuracy: 0.9373
Test accuracy:  0.9373493790626526
```

```python
#Save the model
model.save('model_70_14_all_working.h5')
```

```python
from sklearn.metrics import roc_curve
from scipy.optimize import brentq
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

# Predict probabilities for test data
y_pred_probs_test = model.predict([X_test_dtw, X_test_sig1, X_test_sig2])

# Calculate False Acceptance Rate (FAR) and False Rejection Rate (FRR)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs_test)

far = fpr
frr = 1 - tpr

# Find the EER threshold index and value
eer_threshold_index = np.argmin(np.abs(far - frr))
```

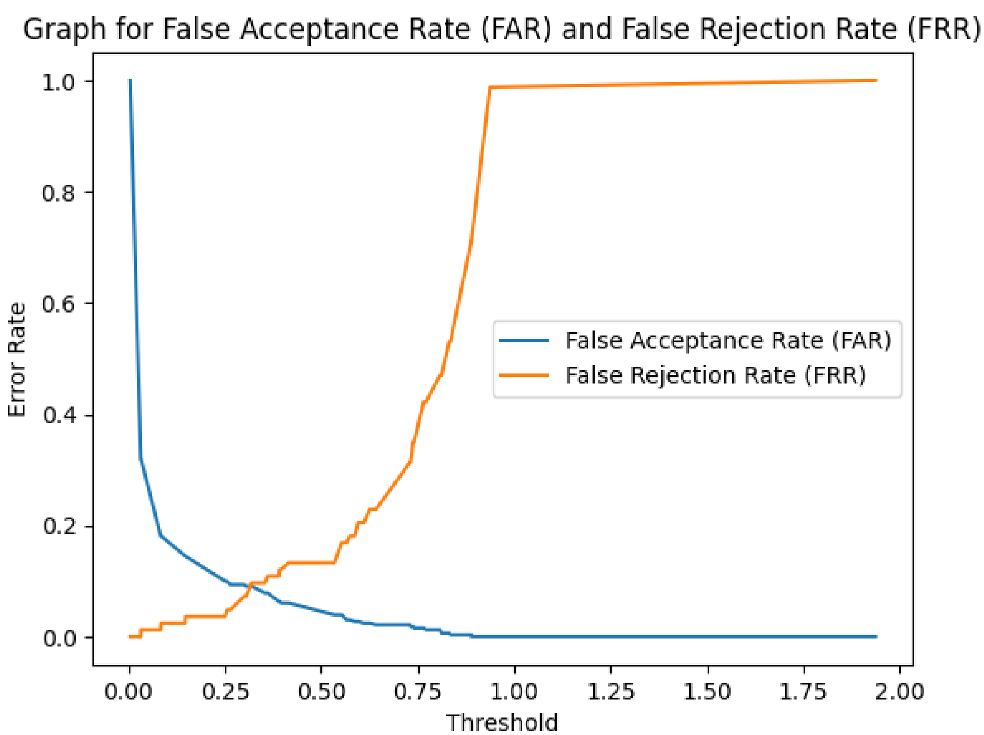
```
eer_threshold = thresholds[eer_threshold_index]

# Calculate the Equal Error Rate (EER)
eer = (far[eer_threshold_index] + frr[eer_threshold_index]) / 2
eer = brentq(lambda x: 1.0 - x - interp1d(fpr, tpr)(x), 0.0, 1.0)
print("Equal Error Rate (EER):", eer * 100)

# Plot the graph for False Acceptance Rate (FAR) and False Rejection Rate (FRR)
plt.plot(thresholds, far, label='False Acceptance Rate (FAR)')
plt.plot(thresholds, frr, label='False Rejection Rate (FRR)')

plt.xlabel('Threshold')
plt.ylabel('Error Rate')
plt.title('Graph for False Acceptance Rate (FAR) and False Rejection Rate (FRR)')
plt.legend()
plt.show()
```

```
13/13 [==============================] - 8s 621ms/step
Equal Error Rate (EER): 9.03614457828121
```




```
#Predict whether the input signatures are genuine or Forgery
def predict_genuine_or_forgery(model, signature1, signature2, max_seq_length):

    # Pad the input signatures
    sig1_padded = pad_sequences([signature1], maxlen=max_seq_length, padding='post', dtype='float32')[0]
    sig2_padded = pad_sequences([signature2], maxlen=max_seq_length, padding='post', dtype='float32')[0]

    # Calculate the DTW distance
    dtw_dist = dtw_distance(signature1, signature2)
    dtw_input = np.array([dtw_dist]).reshape(-1, 1)

    # Make a prediction using the trained model
    prediction = model.predict([dtw_input, sig1_padded[np.newaxis, ...], sig2_padded[np.newaxis, ...]])
```

```
    # Return the result: 1 for genuine, 0 for forgery
    return 1 if prediction[0][0] > 0.5 else 0

signature1 = []
signature2 = []
output = predict_genuine_or_forgery(model, signature1, signature2, max_seq_length)
if output == 1:
  print("Genuine")
else:
  print("Forgery")
```