In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

Establishing connection between Colab and drive as zip file is stored in drive.The below code opens the zip file in read mode and extracts the data and dumps it into "newdata" folder.

In [2]:
```
#Import the libraries
import zipfile
import os

zip_ref = zipfile.ZipFile('/content/drive/MyDrive/date.zip', 'r') #Opens th
zip_ref.extractall('/newdata') #Extracts the files into the newdata folder
zip_ref.close()
```

Checking whether the data has been loaded.

In [3]:
```
len(os.listdir('/newdata/LCC_FASD/LCC_FASD_training/real/'))
```

Out[3]:
```
1223
```

Building a basic cnn model and performing hyper-parameter tuning with different combinations and building the model with best parameters.

In [5]:
```
import os
import random
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, D
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix
import numpy as np

# Set the path to the training data
train_data_dir = "/newdata/LCC_FASD/LCC_FASD_training/"

# Split data into train, validation, and test sets
train_split = 0.8  # 80% for training
validation_split = 0.1  # 10% for validation
test_split = 0.1  # 10% for testing

# Create separate directories for training, validation, and test sets
train_dir = os.path.join(train_data_dir, 'train')
validation_dir = os.path.join(train_data_dir, 'validation')
test_dir = os.path.join(train_data_dir, 'test')

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

# Move the images to their respective directories
for class_name in ['spoof', 'real']:
    class_folder = os.path.join(train_data_dir, class_name)
    image_list = os.listdir(class_folder)
    random.shuffle(image_list)

    train_size = int(train_split * len(image_list))
    validation_size = int(validation_split * len(image_list))

    train_images = image_list[:train_size]
    validation_images = image_list[train_size:train_size + validation_size]
    test_images = image_list[train_size + validation_size:]

    for img_name in train_images:
        src = os.path.join(class_folder, img_name)
        dst = os.path.join(train_dir, class_name, img_name)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        os.rename(src, dst)

    for img_name in validation_images:
        src = os.path.join(class_folder, img_name)
        dst = os.path.join(validation_dir, class_name, img_name)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        os.rename(src, dst)

    for img_name in test_images:
        src = os.path.join(class_folder, img_name)
        dst = os.path.join(test_dir, class_name, img_name)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        os.rename(src, dst)


batch_size = 32
target_size = (224, 224)
```

```
# Create data generators for training, validation, and testing sets
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary'
)

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

# Defining function to build the Basic CNN model with tunable hyperparamete
def build_basic_cnn_model(learning_rate=0.001, conv1_filters=32, conv1_kern
    model = Sequential()

    model.add(Conv2D(filters=conv1_filters,
                     kernel_size=conv1_kernel,
                     activation='relu',
                     input_shape=(224, 224, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(units=dense_units, activation='relu'))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

# Manually perform hyperparameter tuning for Basic CNN
best_accuracy_basic = 0.0
best_params_basic = {}

learning_rates_basic = [0.001, 0.0001]
```

```
conv1_filters_values_basic = [32]
conv1_kernel_values_basic = [3, 5]
dense_units_values_basic = [128, 256]



for lr in learning_rates_basic:
    for filters in conv1_filters_values_basic:
        for kernel in conv1_kernel_values_basic:
            for units in dense_units_values_basic:
                model = build_basic_cnn_model(learning_rate=lr, conv1_filte
                history = model.fit(train_generator, epochs=10, validation_
                test_loss, test_accuracy = model.evaluate(test_generator)
                print("**************************************************

                if test_accuracy > best_accuracy_basic:
                    best_accuracy_basic = test_accuracy
                    best_params_basic = {'learning_rate': lr, 'conv1_filter

# Build the best Basic CNN model with the best hyperparameters
print("Building and training the cnn model with best hyperparameters")
best_basic_cnn_model = build_basic_cnn_model(**best_params_basic)

# Train the best Basic CNN model
best_history_basic = best_basic_cnn_model.fit(train_generator, epochs=10, v

# Evaluate and print confusion matrix for the best Basic CNN model
def evaluate_model(model, test_generator):
    test_loss, test_accuracy = model.evaluate(test_generator)
    print("Test Accuracy:", test_accuracy)

def print_confusion_matrix(model, test_generator):
    predictions = model.predict(test_generator)
    predicted_labels = np.round(predictions)
    true_labels = test_generator.classes
    cm = confusion_matrix(true_labels, predicted_labels)
    print("Confusion Matrix:")
    print(cm)

evaluate_model(best_basic_cnn_model, test_generator)
print_confusion_matrix(best_basic_cnn_model, test_generator)
```

```
Found 6638 images belonging to 2 classes.
Found 829 images belonging to 2 classes.
Found 832 images belonging to 2 classes.
Epoch 1/10
208/208 [==============================] - 175s 836ms/step - loss: 0.94
81 - accuracy: 0.8510 - val_loss: 0.3599 - val_accuracy: 0.8540
Epoch 2/10
208/208 [==============================] - 174s 837ms/step - loss: 0.31
35 - accuracy: 0.8762 - val_loss: 0.3719 - val_accuracy: 0.8673
Epoch 3/10
208/208 [==============================] - 174s 838ms/step - loss: 0.30
52 - accuracy: 0.8775 - val_loss: 0.3095 - val_accuracy: 0.8770
Epoch 4/10
208/208 [==============================] - 174s 838ms/step - loss: 0.28
07 - accuracy: 0.8876 - val_loss: 0.3211 - val_accuracy: 0.8830
Epoch 5/10
208/208 [==============================] - 174s 837ms/step - loss: 0.26
87 - accuracy: 0.8924 - val_loss: 0.3240 - val_accuracy: 0.8806
Epoch 6/10
```

Printing the best params

In [9]:
```
print(best_params_basic)
```

```
{'learning_rate': 0.0001, 'conv1_filters': 32, 'conv1_kernel': 5, 'dense_u
nits': 128}
```

Building a Cnn model with best parameters obtained above with dropout

In [8]:
```
# Defining function to build the CNN model with Dropout and tunable hyperpa
def build_cnn_with_dropout(learning_rate=0.0001, conv1_filters=32, conv1_ke
    model = Sequential()

    model.add(Conv2D(filters=conv1_filters,
                     kernel_size=conv1_kernel,
                     activation='relu',
                     input_shape=(224, 224, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(units=dense_units, activation='relu'))

    # Adding Dropout layer
    model.add(Dropout(rate=dropout_rate))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

# Dropout values to try
dropout_rates = [0.1, 0.3, 0.5]

# Best hyperparameters from previous tuning
best_params = {'learning_rate': 0.0001, 'conv1_filters': 32, 'conv1_kernel'

# List to store models
models_with_dropout = []

# Build models with different dropout rates
for dropout_rate in dropout_rates:
    model = build_cnn_with_dropout(dropout_rate=dropout_rate, **best_params
    models_with_dropout.append(model)

# Train, evaluate, and print confusion matrix for models
for i, model in enumerate(models_with_dropout):
    print(f"Training Model with Dropout Rate: {dropout_rates[i]}")
    history = model.fit(train_generator, epochs=10, validation_data=validat
    test_loss, test_accuracy = model.evaluate(test_generator)
    print(f"Test Accuracy for Dropout Rate {dropout_rates[i]}:", test_accur

    # Calculate and print confusion matrix
    print_confusion_matrix(model, test_generator)
    print("\n")
```

```
Training Model with Dropout Rate: 0.1
Epoch 1/10
208/208 [==============================] - 176s 839ms/step - loss: 0.42
37 - accuracy: 0.8435 - val_loss: 0.3480 - val_accuracy: 0.8745
Epoch 2/10
208/208 [==============================] - 174s 834ms/step - loss: 0.30
23 - accuracy: 0.8775 - val_loss: 0.3060 - val_accuracy: 0.8745
Epoch 3/10
208/208 [==============================] - 173s 834ms/step - loss: 0.29
62 - accuracy: 0.8822 - val_loss: 0.3286 - val_accuracy: 0.8745
Epoch 4/10
208/208 [==============================] - 174s 836ms/step - loss: 0.27
31 - accuracy: 0.8891 - val_loss: 0.2944 - val_accuracy: 0.8914
Epoch 5/10
208/208 [==============================] - 170s 818ms/step - loss: 0.26
89 - accuracy: 0.8923 - val_loss: 0.2830 - val_accuracy: 0.8963
Epoch 6/10
208/208 [==============================] - 170s 820ms/step - loss: 0.26
55 - accuracy: 0.8938 - val_loss: 0.2722 - val_accuracy: 0.9035
```

In [10]:
```
# Defining a function to build the CNN model with L2 regularization and tun
def build_cnn_with_regularization(learning_rate=0.0001, conv1_filters=32, c
    model = Sequential()

    model.add(Conv2D(filters=conv1_filters,
                     kernel_size=conv1_kernel,
                     activation='relu',
                     kernel_regularizer=tf.keras.regularizers.l2(l2_regular
                     input_shape=(224, 224, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(units=dense_units, activation='relu'))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

# Regularization values to try
l2_regularization_values = [0.01, 0.001, 0.0001]

# Best hyperparameters from previous tuning
best_params = {'learning_rate': 0.0001, 'conv1_filters': 32, 'conv1_kernel'

# List to store models
models_with_regularization = []

# Build models with different regularization values
for l2_regularization in l2_regularization_values:
    model = build_cnn_with_regularization(l2_regularization=l2_regularizati
    models_with_regularization.append(model)

# Train, evaluate, and print confusion matrix for models
for i, model in enumerate(models_with_regularization):
    print(f"Training Model with L2 Regularization Value: {l2_regularization
    history = model.fit(train_generator, epochs=10, validation_data=validat
    test_loss, test_accuracy = model.evaluate(test_generator)
    print(f"Test Accuracy for L2 Regularization Value {l2_regularization_va

    # Calculate and print confusion matrix
    print_confusion_matrix(model, test_generator)
    print("\n")
```

```
Training Model with L2 Regularization Value: 0.01
Epoch 1/10
208/208 [==============================] - 174s 830ms/step - loss: 0.47
86 - accuracy: 0.8512 - val_loss: 0.3985 - val_accuracy: 0.8577
Epoch 2/10
208/208 [==============================] - 173s 831ms/step - loss: 0.34
51 - accuracy: 0.8792 - val_loss: 0.3649 - val_accuracy: 0.8709
Epoch 3/10
208/208 [==============================] - 176s 848ms/step - loss: 0.33
37 - accuracy: 0.8861 - val_loss: 0.3393 - val_accuracy: 0.8818
Epoch 4/10
208/208 [==============================] - 176s 845ms/step - loss: 0.31
48 - accuracy: 0.8955 - val_loss: 0.3493 - val_accuracy: 0.8842
Epoch 5/10
208/208 [==============================] - 175s 843ms/step - loss: 0.31
30 - accuracy: 0.8930 - val_loss: 0.3342 - val_accuracy: 0.8914
Epoch 6/10
208/208 [==============================] - 176s 845ms/step - loss: 0.29
55 - accuracy: 0.8998 - val_loss: 0.3241 - val_accuracy: 0.8878
```

In [11]:
```
# Defining function to build the CNN model with Dropout and L2 regularizati
def build_cnn_with_dropout_and_regularization(learning_rate=0.0001, conv1_f
    model = Sequential()

    model.add(Conv2D(filters=conv1_filters,
                     kernel_size=conv1_kernel,
                     activation='relu',
                     kernel_regularizer=tf.keras.regularizers.l2(l2_regular
                     input_shape=(224, 224, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(units=dense_units, activation='relu'))

    # Adding Dropout layer
    model.add(Dropout(rate=dropout_rate))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

# Build the CNN model with Dropout and L2 regularization
cnn_with_dropout_and_regularization = build_cnn_with_dropout_and_regulariza

# Train the model
history = cnn_with_dropout_and_regularization.fit(train_generator, epochs=1

# Evaluate the model
test_loss, test_accuracy = cnn_with_dropout_and_regularization.evaluate(tes
print("Test Accuracy:", test_accuracy)

# Calculate and print confusion matrix
predictions = cnn_with_dropout_and_regularization.predict(test_generator)
predicted_labels = np.round(predictions)
true_labels = test_generator.classes
cm = confusion_matrix(true_labels, predicted_labels)
print("Confusion Matrix:")
print(cm)
```

```
Epoch 1/10
208/208 [==============================] - 177s 843ms/step - loss: 0.3835
- accuracy: 0.8602 - val_loss: 0.3202 - val_accuracy: 0.8673
Epoch 2/10
208/208 [==============================] - 174s 835ms/step - loss: 0.3029
- accuracy: 0.8777 - val_loss: 0.3082 - val_accuracy: 0.8709
Epoch 3/10
208/208 [==============================] - 174s 835ms/step - loss: 0.2894
- accuracy: 0.8823 - val_loss: 0.3114 - val_accuracy: 0.8745
Epoch 4/10
208/208 [==============================] - 174s 835ms/step - loss: 0.2777
- accuracy: 0.8884 - val_loss: 0.2815 - val_accuracy: 0.8914
Epoch 5/10
208/208 [==============================] - 174s 838ms/step - loss: 0.2746
- accuracy: 0.8897 - val_loss: 0.2745 - val_accuracy: 0.8938
Epoch 6/10
208/208 [==============================] - 174s 837ms/step - loss: 0.2513
- accuracy: 0.8973 - val_loss: 0.2740 - val_accuracy: 0.8951
Epoch 7/10
208/208 [==============================] - 173s 833ms/step - loss: 0.2480
- accuracy: 0.8980 - val_loss: 0.2816 - val_accuracy: 0.8963
Epoch 8/10
208/208 [==============================] - 173s 832ms/step - loss: 0.2445
- accuracy: 0.9027 - val_loss: 0.2439 - val_accuracy: 0.9168
Epoch 9/10
208/208 [==============================] - 173s 832ms/step - loss: 0.2309
- accuracy: 0.9092 - val_loss: 0.2955 - val_accuracy: 0.8758
Epoch 10/10
208/208 [==============================] - 173s 834ms/step - loss: 0.2215
- accuracy: 0.9134 - val_loss: 0.2315 - val_accuracy: 0.9180
26/26 [==============================] - 12s 451ms/step - loss: 0.2055 - a
ccuracy: 0.9327
Test Accuracy: 0.932692289352417
26/26 [==============================] - 12s 445ms/step
Confusion Matrix:
[[ 69  54]
 [  2 707]]
```