```
import numpy as np
import os
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense,
Flatten
import cv2
import matplotlib.pyplot as plt
f_train_path = r"C:\Users\Sandeep\Desktop\ads1\train_1"
f_train = []
for filename in os.listdir(f_train_path):
    if filename.endswith(".jpg") or filename.endswith(".jpeg") or
filename.endswith(".webp"):
        image_path = os.path.join(f_train_path, filename)
        img = cv2.imread(image_path)
        if img is not None:
           f_train.append(img)
        else:
            print(f"Could not load {image_path}")
if  not f_train:
    print("f_train not successfull")
else:
    print("f_train  successfull")
f_test_path = r"C:\Users\Sandeep\Desktop\ads1\test_1"
f_test = []
for filename in os.listdir(f_test_path):
    if filename.endswith(".jpg") or filename.endswith(".jpeg") or
filename.endswith(".webp"):
        image_path = os.path.join(f_test_path, filename)
        img = cv2.imread(image_path)
        if img is not None:
           f_test.append(img)
        else:
            print(f"Could not load {image_path}")
if not f_test:
    print("f_test not successfull")
else:
    print("f_test successfull")
nf_train_path = r"C:\Users\Sandeep\Desktop\ads1\train"
nf_train = []
for filename in os.listdir(nf_train_path):
    if filename.endswith(".jpg") or filename.endswith(".jpeg")  or
filename.endswith(".webp"):
        image_path = os.path.join(nf_train_path, filename)
        img = cv2.imread(image_path)
        if img is not None:
           nf_train.append(img)
        else:
            print(f"Could not load {image_path}")
```

```
if not  nf_train:
    print("nf_train not successfull")
else:
    print("nf_train successfull")
nf_test_path = r"C:\Users\Sandeep\Desktop\ads1\test"
nf_test = []
for filename in os.listdir(nf_test_path):
    if filename.endswith(".jpg") or filename.endswith(".jpeg") or
filename.endswith(".webp"):
        image_path = os.path.join(nf_test_path, filename)
        img = cv2.imread(image_path)
        if img is not None:
           nf_test.append(img)
        else:
            print(f"Could not load {image_path}")
if not nf_test:
     print("nf_test not successfull")
else:
    print("nf_test successfull")
```

```
f_train  successfull
f_test successfull
nf_train successfull
nf_test successfull
```

```
f_train=np.array(f_train)
f_test=np.array(f_test)
nf_train=np.array(nf_train)
nf_test=np.array(nf_test)
if isinstance(f_train, np.ndarray):
    print("f_train is a NumPy array.")
else:
    print("f_train is not a NumPy array.")
if isinstance(f_test, np.ndarray):
    print("f_test is a NumPy array.")
else:
    print("f_test is not a NumPy array.")
if isinstance(nf_train, np.ndarray):
    print("nf_train is a NumPy array.")
else:
    print("nf_train is not a NumPy array.")
if isinstance(nf_test, np.ndarray):
    print("nf_test is a NumPy array.")
else:
    print("nf_test is not a NumPy array.")
```

```
f_train is a NumPy array.
f_test is a NumPy array.
nf_train is a NumPy array.
nf_test is a NumPy array.
```

```
import cv2
import numpy as np
target_height = 256
target_width = 256
resized_images = []
for img in f_train:
    resized_img = cv2.resize(img, (target_width, target_height))
    resized_images.append(resized_img)
f_train = np.array(resized_images)

print("shape of f_train : ", f_train.shape)
print("shape of nf_train : ", nf_train.shape)
print("shape of f_test : ", f_test.shape)
print("shape of nf_test : ", nf_test.shape)
```

```
shape of f_train :  (304, 256, 256, 3)
shape of nf_train :  (308, 256, 256, 3)
shape of f_test :  (66, 256, 256, 3)
shape of nf_test :  (66, 256, 256, 3)
```

```
# Convert the loaded images to numpy arrays
f_train = np.array(f_train)
f_test = np.array(f_test)
nf_train = np.array(nf_train)
nf_test = np.array(nf_test)

# Prepare the data and labels
X_train = np.concatenate((f_train, nf_train), axis=0)
X_test = np.concatenate((f_test, nf_test), axis=0)
y_train = np.concatenate((np.ones(len(f_train)),
np.zeros(len(nf_train))), axis=0)
y_test = np.concatenate((np.ones(len(f_test)),
np.zeros(len(nf_test))), axis=0)

# Normalize pixel values to be between 0 and 1
X_train = X_train / 255.0
X_test = X_test / 255.0

# Create and train the model
model = Sequential([
    Conv2D(16, (3,3), activation='relu',
input_shape=X_train.shape[1:]),
    MaxPooling2D(2,2),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, validation_data=(X_test,
y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'Test accuracy: {test_acc}')

Epoch 1/10
20/20 [==============================] - 70s 2s/step - loss: 1.6042 -
accuracy: 0.5997 - val_loss: 0.4630 - val_accuracy: 0.7803
Epoch 2/10
20/20 [==============================] - 38s 2s/step - loss: 0.4712 -
accuracy: 0.7827 - val_loss: 0.3482 - val_accuracy: 0.8561
Epoch 3/10
20/20 [==============================] - 30s 1s/step - loss: 0.3095 -
accuracy: 0.8595 - val_loss: 0.3194 - val_accuracy: 0.9015
Epoch 4/10
20/20 [==============================] - 29s 1s/step - loss: 0.2457 -
accuracy: 0.8954 - val_loss: 0.2636 - val_accuracy: 0.8864
Epoch 5/10
20/20 [==============================] - 29s 1s/step - loss: 0.1865 -
accuracy: 0.9297 - val_loss: 0.2496 - val_accuracy: 0.8939
Epoch 6/10
20/20 [==============================] - 30s 1s/step - loss: 0.1835 -
accuracy: 0.9281 - val_loss: 0.3048 - val_accuracy: 0.8864
Epoch 7/10
20/20 [==============================] - 29s 1s/step - loss: 0.1663 -
accuracy: 0.9330 - val_loss: 0.2610 - val_accuracy: 0.8864
Epoch 8/10
20/20 [==============================] - 30s 2s/step - loss: 0.1160 -
accuracy: 0.9624 - val_loss: 0.4704 - val_accuracy: 0.8939
Epoch 9/10
20/20 [==============================] - 30s 1s/step - loss: 0.1278 -
accuracy: 0.9575 - val_loss: 0.3162 - val_accuracy: 0.9015
Epoch 10/10
20/20 [==============================] - 29s 1s/step - loss: 0.0780 -
accuracy: 0.9673 - val_loss: 0.3863 - val_accuracy: 0.8864
5/5 - 1s - loss: 0.3863 - accuracy: 0.8864 - 1s/epoch - 267ms/step
Test accuracy: 0.8863636255264282
```