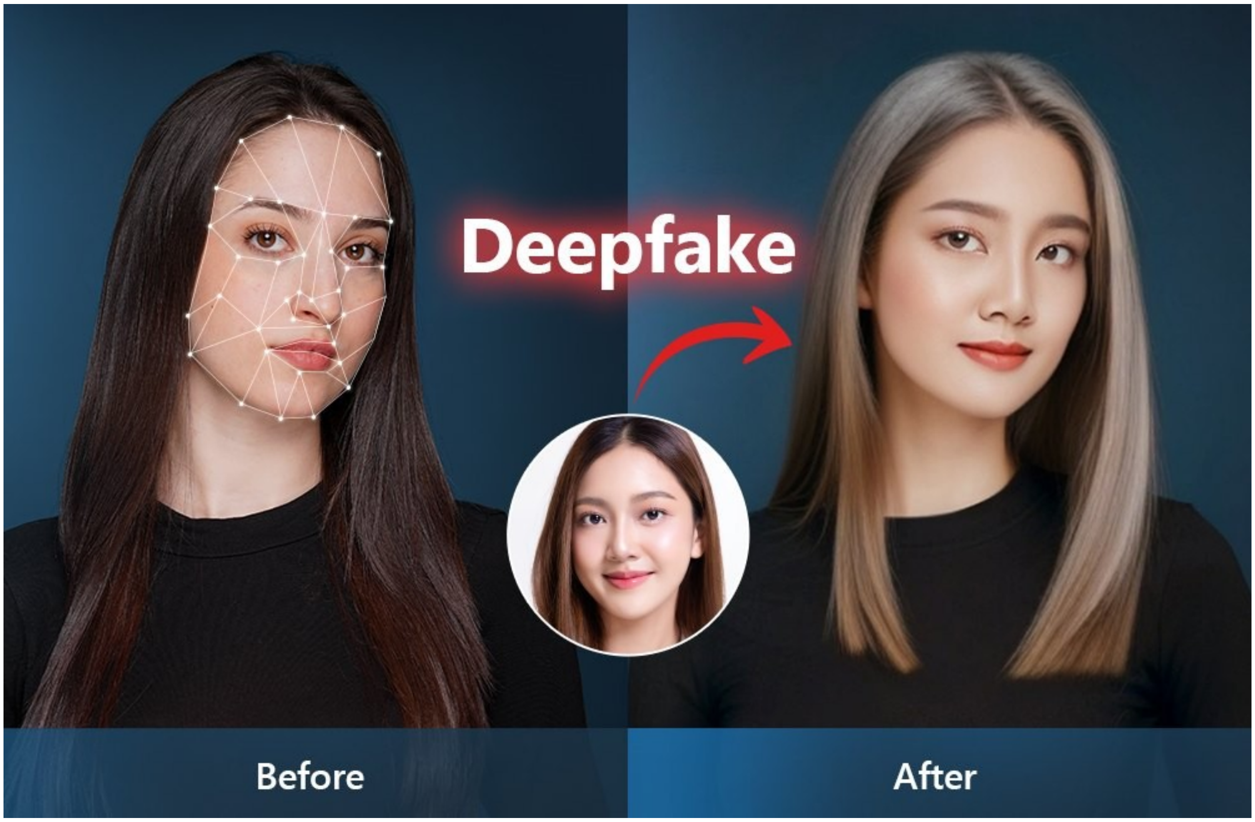
Deep Fake Classification using InceptionV3 (pre-trained on ImageNet) with a unique Fourier Attention layer



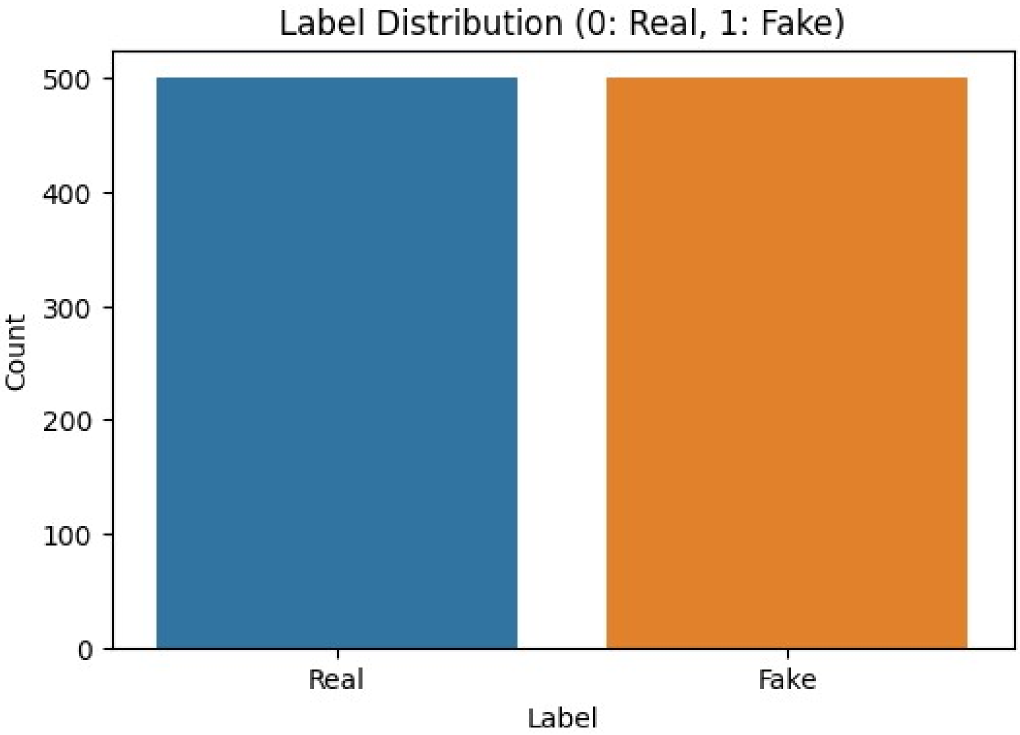
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, classification_report
import warnings
warnings.filterwarnings('ignore')

os.environ['TF_FORCE_UNIFIED_MEMORY'] = '1'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

tf.random.set_seed(42)
np.random.seed(42)

base_path = '/kaggle/input/deepfake-videos-dataset/'

def load_and_extract_frames(csv_path, frames_per_video=100):
    try:
        df = pd.read_csv(csv_path)
        print("Loaded actual CSV file.")
        print("CSV Columns:", df.columns.tolist())
        print("Sample Data:\n", df.head())

        def infer_label(row):
            if row['deepfake'].startswith('deepfake/'):
                return 1
            elif row['video'].startswith('video/'):
                return 0
            else:
                raise ValueError(f"Cannot infer label for row: {row}")

        df['label'] = df.apply(infer_label, axis=1)
        print("Label Distribution in CSV:")
        print(df['label'].value_counts())

        frame_data = {'filename': [], 'label': []}
        output_dir = '/kaggle/working/frames/'
        os.makedirs(output_dir, exist_ok=True)

        for _, row in df.iterrows():
            video_id = row['id']
            for video_type, label in [('deepfake', 1), ('video', 0)]:
                video_path = os.path.join(base_path, row[video_type])

                if not os.path.exists(video_path):
                    video_path_alt = video_path.replace('.mov',
'.MOV') if '.mov' in video_path else video_path.replace('.MOV',
'.mov')
                    if os.path.exists(video_path_alt):
                        video_path = video_path_alt
                    else:
                        print(f"Cannot find video: {video_path} or
{video_path_alt}")
                        continue

                try:
                    cap = cv2.VideoCapture(video_path)
                    if not cap.isOpened():
                        print(f"Cannot open video: {video_path}")
                        continue

                    total_frames = 
int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
                    if total_frames < 1:
                        print(f"Empty video: {video_path}")
                        cap.release()
                        continue

                    step = max(1, total_frames // frames_per_video)

                    frame_count = 0
                    frame_idx = 0
                    while frame_count < frames_per_video:
                        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
                        ret, frame = cap.read()
                        if not ret:
                            print(f"Failed to read frame {frame_idx} 
from {video_path}")
                            break

                        frame_filename = os.path.join(output_dir, 
f"{video_id}_{video_type}_frame_{frame_count}.jpg")
                        cv2.imwrite(frame_filename, frame)
                        if not os.path.exists(frame_filename):
                            print(f"Failed to save frame: 
{frame_filename}")
                            break

                        frame_data['filename'].append(frame_filename)
                        frame_data['label'].append(label)

                        frame_count += 1
                        frame_idx += step

                    cap.release()
                    print(f"Extracted {frame_count} frames from 
{video_path} (label: {label})")
                except Exception as e:
                    print(f"Error processing video {video_path}: 
{str(e)}")

        frame_df = pd.DataFrame(frame_data)
        if frame_df.empty:
            raise ValueError("No frames extracted from videos.")

        print("Frame Label Distribution:")
        print(frame_df['label'].value_counts())
        return frame_df
```

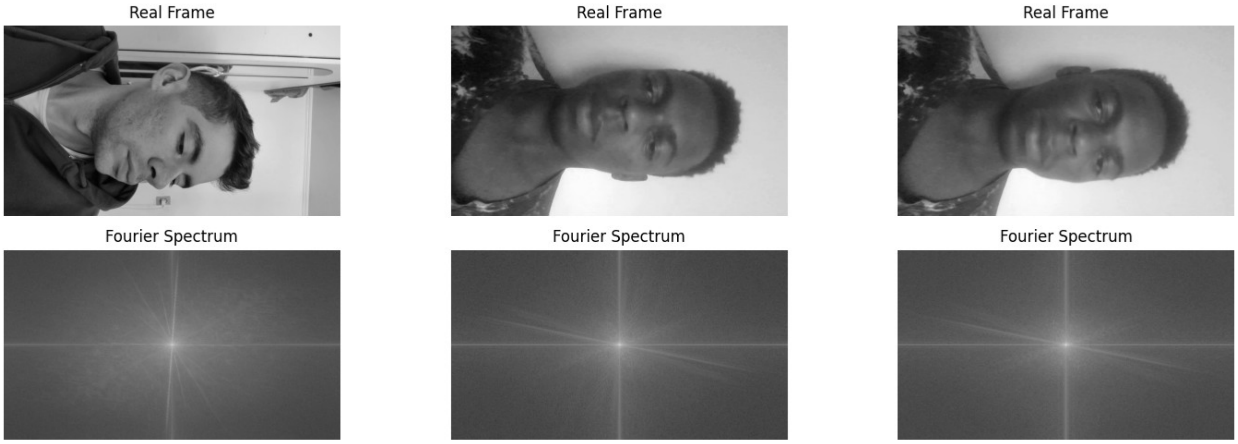
```
    except Exception as e:
        print(f"Error loading CSV or extracting frames: {str(e)}.
Using simulated data.")
        data = {
            'filename': [f"/kaggle/working/frames/{i}_frame_{j}.jpg"
for i in range(1, 6) for j in range(100)],
            'label': [0 if i % 2 == 0 else 1 for i in range(1, 6) for
_ in range(100)]
        }
        return pd.DataFrame(data)

def perform_eda(df):
    print("Frame Dataset Info:")
    print(df.info())
    print("\nLabel Distribution:")
    print(df['label'].value_counts())

    plt.figure(figsize=(6, 4))
    sns.countplot(x='label', data=df)
    plt.title('Label Distribution (0: Real, 1: Fake)')
    plt.xlabel('Label')
    plt.ylabel('Count')
    plt.xticks([0, 1], ['Real', 'Fake'])
    plt.show()

def visualize_images_and_fourier(df, num_samples=3):
    sample_df = df.sample(n=min(num_samples, len(df)),
random_state=42)

    plt.figure(figsize=(15, 5))
    for i, row in enumerate(sample_df.itertuples()):
        img_path = row.filename
        try:
            img = Image.open(img_path).convert('L')
            img_array = np.array(img, dtype=float)

            f_transform = np.fft.fft2(img_array)
            f_transform_shifted = np.fft.fftshift(f_transform)
            magnitude_spectrum = np.log(np.abs(f_transform_shifted) +
1)

            plt.subplot(2, num_samples, i + 1)
            plt.imshow(img_array, cmap='gray')
            plt.title(f"{'Fake' if row.label else 'Real'} Frame")
            plt.axis('off')

            plt.subplot(2, num_samples, i + 1 + num_samples)
            plt.imshow(magnitude_spectrum, cmap='gray')
            plt.title('Fourier Spectrum')
            plt.axis('off')
        except FileNotFoundError:
            print(f"Frame not found: {img_path}")
    plt.tight_layout()
    plt.show()

class FourierAttention(layers.Layer):
    def __init__(self, **kwargs):
        super(FourierAttention, self).__init__(**kwargs)

    def call(self, inputs):
        fft = tf.signal.fft2d(tf.cast(inputs, tf.complex64))
        fft_shifted = tf.signal.fftshift(fft)
        magnitude = tf.abs(fft_shifted)
        attention = tf.reduce_mean(magnitude, axis=[1, 2],
keepdims=True)
        attention = tf.nn.softmax(attention, axis=-1)
        return inputs * attention

def create_novel_model(input_shape=(224, 224, 3), num_classes=2):
    base_model = InceptionV3(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False

    inputs = layers.Input(shape=input_shape)
    x = base_model(inputs, training=False)
    x = FourierAttention()(x)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs, outputs)
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

def prepare_data(df, target_size=(224, 224), batch_size=16):
    valid_files = []
    valid_labels = []
    for _, row in df.iterrows():
        if os.path.exists(row['filename']):
            valid_files.append(row['filename'])
            valid_labels.append(row['label'])
        else:
            print(f"Skipping missing file: {row['filename']}")

    if not valid_files:
        raise ValueError("No valid frame files found for training.")
    valid_df = pd.DataFrame({'filename': valid_files, 'label':
valid_labels})
    valid_df['label'] = valid_df['label'].astype(str)

    if len(valid_df['label'].unique()) < 2:
        raise ValueError("Only one class found in the dataset. Need
both real and fake classes.")

    datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        validation_split=0.2
    )

    train_generator = datagen.flow_from_dataframe(
        valid_df,
        x_col='filename',
        y_col='label',
        target_size=target_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='training',
        shuffle=True
    )

    val_generator = datagen.flow_from_dataframe(
        valid_df,
        x_col='filename',
        y_col='label',
        target_size=target_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation',
        shuffle=False
    )

    return train_generator, val_generator

def train_model(model, train_generator, val_generator, epochs=20):
    history = model.fit(
        train_generator,
        validation_data=val_generator,
        epochs=epochs,
        verbose=1
    )
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Loss Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Accuracy Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()

    return history

def evaluate_model(model, val_generator):
    val_generator.reset()
    y_pred = model.predict(val_generator)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true = val_generator.classes

    cm = confusion_matrix(y_true, y_pred_classes)
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

    print("Classification Report:")
    print(classification_report(y_true, y_pred_classes, target_names=['Real', 'Fake']))

if __name__ == "__main__":
    try:
        os.system('rm -rf /kaggle/working/frames/*')

        csv_path = os.path.join(base_path, 'DeepFake Videos Dataset.csv')
        frame_df = load_and_extract_frames(csv_path,
frames_per_video=100)

        perform_eda(frame_df)

        visualize_images_and_fourier(frame_df)

        train_generator, val_generator = prepare_data(frame_df)

        model = create_novel_model()
        history = train_model(model, train_generator, val_generator)

        evaluate_model(model, val_generator)

    except Exception as e:
        print(f"Error: {str(e)}")
```

```
2025-07-01 10:41:51.036877: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1751366511.517008      35 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1751366511.643407      35 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

```
Loaded actual CSV file.
CSV Columns: ['id', 'deepfake', 'image', 'video']
Sample Data:
    id        deepfake         image        video
0   1  deepfake/1.mp4   image/1.jpg  video/1.mp4
1   2  deepfake/2.mp4  image/2.jpeg  video/2.mp4
2   3  deepfake/3.mp4   image/3.jpg  video/3.mp4
3   4  deepfake/4.mp4   image/4.jpg  video/4.mp4
4   5  deepfake/5.mov   image/5.jpg  video/5.mov
Label Distribution in CSV:
label
1    5
Name: count, dtype: int64
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/deepfake/1.mp4 (label: 1)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/video/1.mp4 (label: 0)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/deepfake/2.mp4 (label: 1)
Extracted 100 frames from
```

```
/kaggle/input/deepfake-videos-dataset/video/2.mp4 (label: 0)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/deepfake/3.mp4 (label: 1)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/video/3.mp4 (label: 0)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/deepfake/4.mp4 (label: 1)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/video/4.mp4 (label: 0)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/deepfake/5.mov (label: 1)
Extracted 100 frames from
/kaggle/input/deepfake-videos-dataset/video/5.MOV (label: 0)
Frame Label Distribution:
label
1    500
0    500
Name: count, dtype: int64
Frame Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   filename  1000 non-null   object
 1   label     1000 non-null   int64
dtypes: int64(1), object(1)
memory usage: 15.8+ KB
None

Label Distribution:
label
1    500
0    500
Name: count, dtype: int64
```

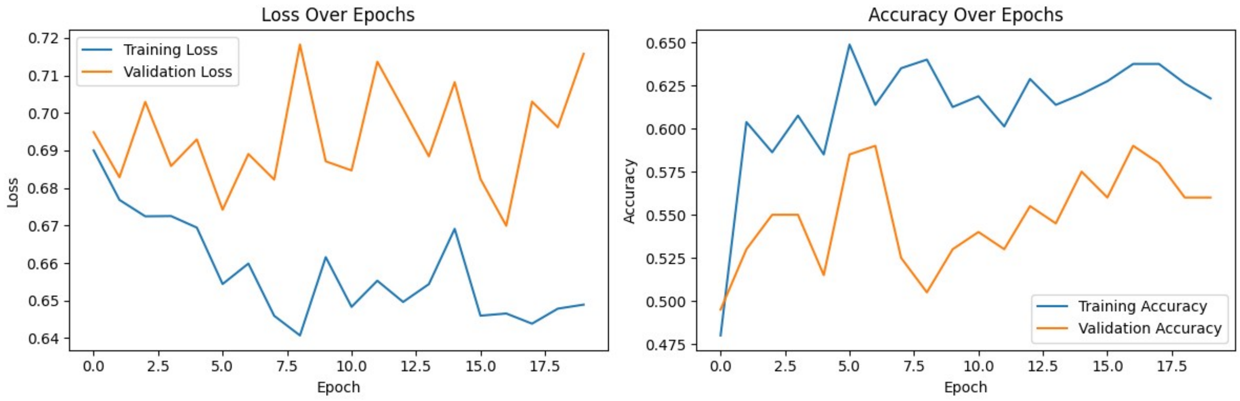




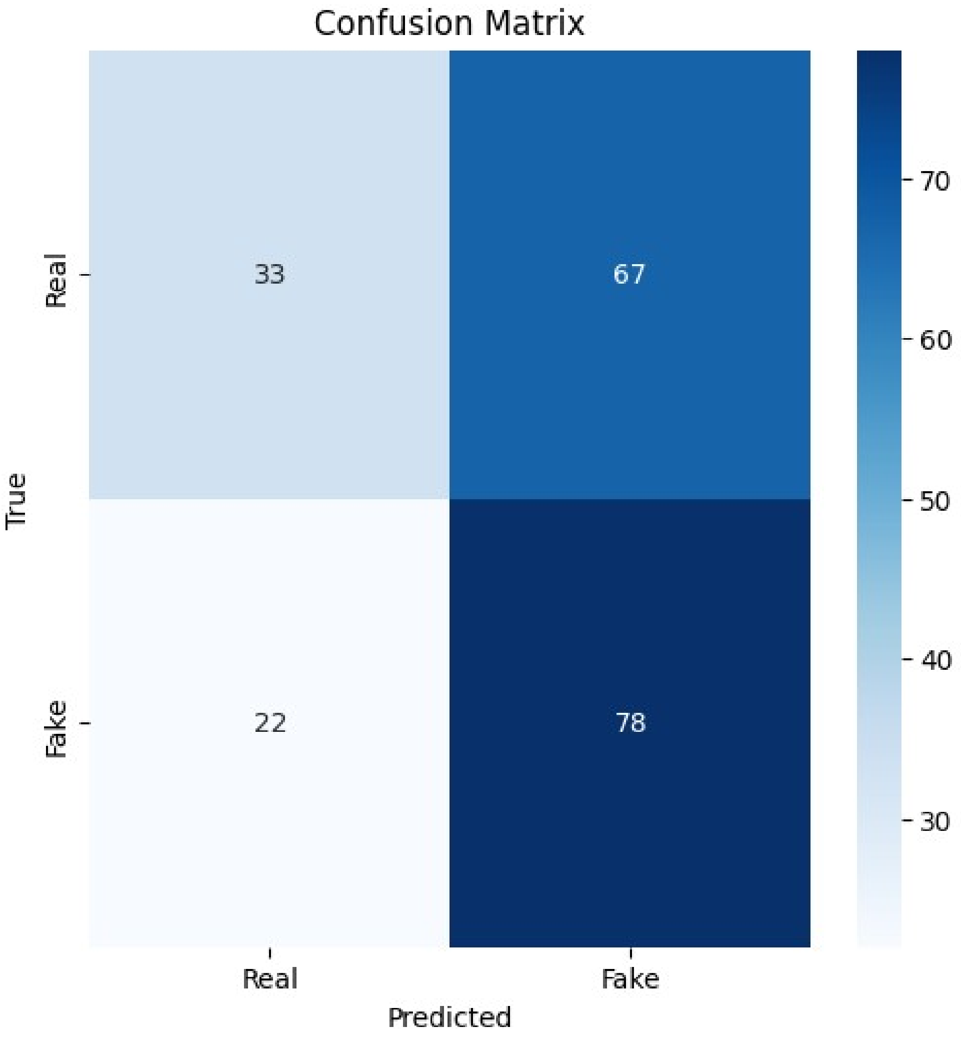
```
Found 800 validated image filenames belonging to 2 classes.
Found 200 validated image filenames belonging to 2 classes.

I0000 00:00:1751366636.305331      35 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB
memory:  -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0,
compute capability: 7.5
I0000 00:00:1751366636.305930      35 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB
memory:  -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0,
compute capability: 7.5
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/
inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Epoch 1/20

WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1751366651.818933     144 service.cc:148] XLA service
0x7829a8004860 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
I0000 00:00:1751366651.820535     144 service.cc:156]   StreamExecutor
device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1751366651.820562     144 service.cc:156]   StreamExecutor
device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1751366653.757033     144 cuda_dnn.cc:529] Loaded cuDNN
version 90300
I0000 00:00:1751366661.341720     144 device_compiler.h:188] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

50/50 ━━━━━━━━━━━━━━━━━━━━ 47s 544ms/step - accuracy: 0.4690 - loss:
0.6930 - val_accuracy: 0.4950 - val_loss: 0.6949
Epoch 2/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 302ms/step - accuracy: 0.5742 - loss:
0.6809 - val_accuracy: 0.5300 - val_loss: 0.6828
Epoch 3/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 302ms/step - accuracy: 0.5729 - loss:
0.6752 - val_accuracy: 0.5500 - val_loss: 0.7029
Epoch 4/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 301ms/step - accuracy: 0.5975 - loss:
0.6775 - val_accuracy: 0.5500 - val_loss: 0.6859
Epoch 5/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 295ms/step - accuracy: 0.5799 - loss:
0.6655 - val_accuracy: 0.5150 - val_loss: 0.6929
Epoch 6/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 298ms/step - accuracy: 0.6735 - loss:
0.6446 - val_accuracy: 0.5850 - val_loss: 0.6742
Epoch 7/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 298ms/step - accuracy: 0.6093 - loss:
0.6565 - val_accuracy: 0.5900 - val_loss: 0.6891
Epoch 8/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 295ms/step - accuracy: 0.6316 - loss:
0.6488 - val_accuracy: 0.5250 - val_loss: 0.6822
Epoch 9/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 296ms/step - accuracy: 0.6618 - loss:
0.6336 - val_accuracy: 0.5050 - val_loss: 0.7182
Epoch 10/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 292ms/step - accuracy: 0.6051 - loss:
0.6647 - val_accuracy: 0.5300 - val_loss: 0.6871
```

```
Epoch 11/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 294ms/step - accuracy: 0.6476 - loss:
0.6364 - val_accuracy: 0.5400 - val_loss: 0.6847
Epoch 12/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 299ms/step - accuracy: 0.6018 - loss:
0.6548 - val_accuracy: 0.5300 - val_loss: 0.7136
Epoch 13/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 292ms/step - accuracy: 0.6220 - loss:
0.6490 - val_accuracy: 0.5550 - val_loss: 0.7011
Epoch 14/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 294ms/step - accuracy: 0.6295 - loss:
0.6431 - val_accuracy: 0.5450 - val_loss: 0.6884
Epoch 15/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 300ms/step - accuracy: 0.5971 - loss:
0.6786 - val_accuracy: 0.5750 - val_loss: 0.7082
Epoch 16/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 294ms/step - accuracy: 0.6264 - loss:
0.6535 - val_accuracy: 0.5600 - val_loss: 0.6824
Epoch 17/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 303ms/step - accuracy: 0.6667 - loss:
0.6254 - val_accuracy: 0.5900 - val_loss: 0.6699
Epoch 18/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 292ms/step - accuracy: 0.6104 - loss:
0.6643 - val_accuracy: 0.5800 - val_loss: 0.7030
Epoch 19/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 298ms/step - accuracy: 0.6183 - loss:
0.6475 - val_accuracy: 0.5600 - val_loss: 0.6962
Epoch 20/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 291ms/step - accuracy: 0.6256 - loss:
0.6454 - val_accuracy: 0.5600 - val_loss: 0.7158
```



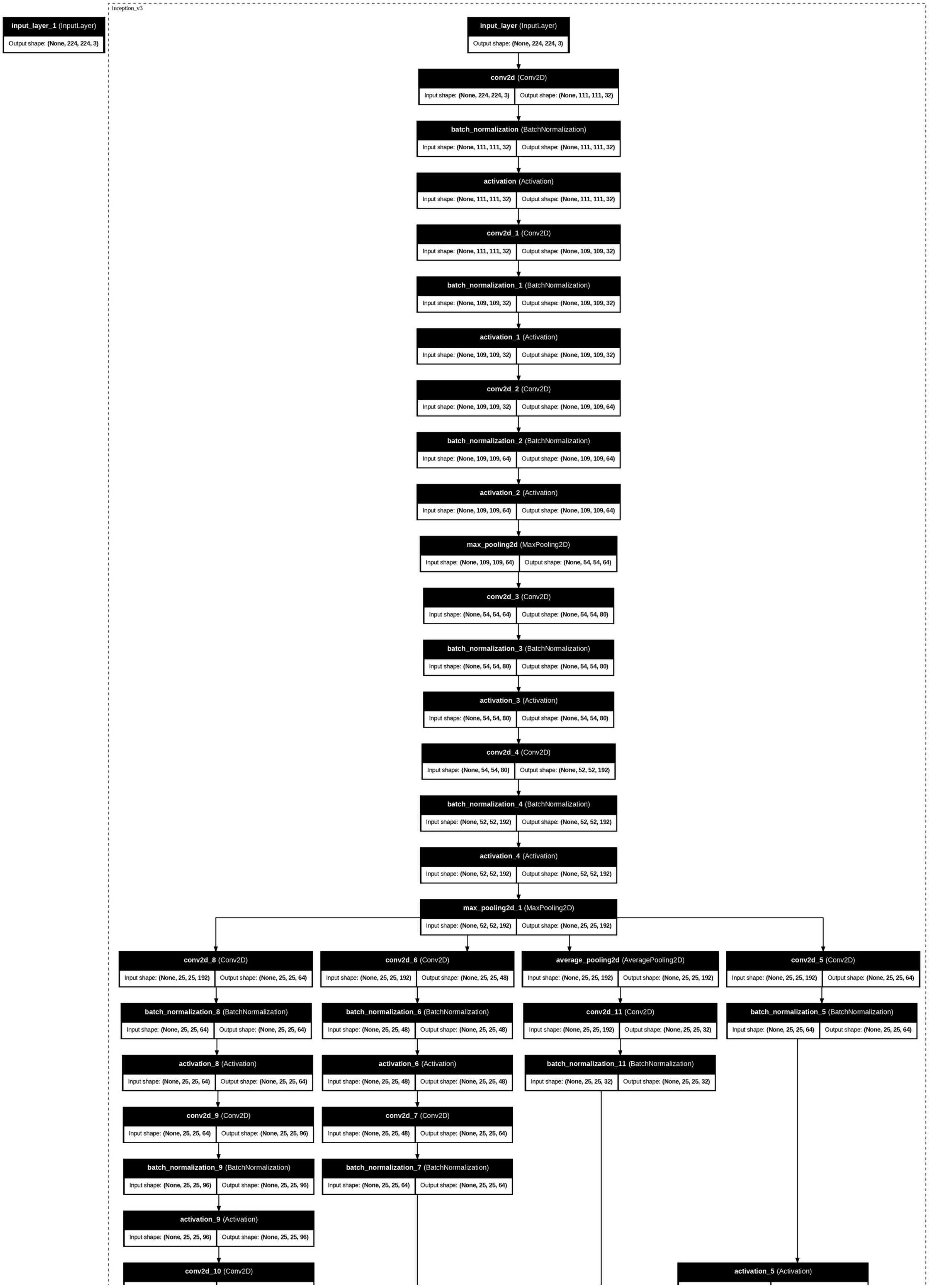
```
13/13 ━━━━━━━━━━━━━━━━━━━━ 17s 755ms/step
```



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

        Real       0.60      0.33      0.43       100
        Fake       0.54      0.78      0.64       100

    accuracy                           0.56       200
   macro avg       0.57      0.56      0.53       200
weighted avg       0.57      0.56      0.53       200


model.summary()

Model: "functional"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
```

```
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ input_layer_1 (InputLayer)           │ (None, 224, 224, 3)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ inception_v3 (Functional)            │ (None, 5, 5, 2048)          │
21,802,784 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ fourier_attention (FourierAttention) │ (None, 5, 5, 2048)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ global_average_pooling2d             │ (None, 2048)                │
0 │
│ (GlobalAveragePooling2D)             │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense (Dense)                        │ (None, 128)                 │
262,272 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout (Dropout)                    │ (None, 128)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_1 (Dense)                      │ (None, 2)                   │
258 │
└──────────────────────────────────────┴─────────────────────────────┴
─────────────────┘

 Total params: 22,590,376 (86.18 MB)

 Trainable params: 262,530 (1.00 MB)

 Non-trainable params: 21,802,784 (83.17 MB)

 Optimizer params: 525,062 (2.00 MB)
```

```python
from tensorflow.keras.utils import plot_model
plot_model(
    model,
    to_file='model_architecture.png',
    show_shapes=True,
    show_layer_names=True,
    expand_nested=True
)
```



input_layer_1 (InputLayer)
Output shape: (None, 224, 224, 3)
inception_v3
input_layer (InputLayer)
Output shape: (None, 224, 224, 3)
conv2d (Conv2D)
Input shape: (None, 224, 224, 3) | Output shape: (None, 111, 111, 32)
batch_normalization (BatchNormalization)
Input shape: (None, 111, 111, 32) | Output shape: (None, 111, 111, 32)
activation (Activation)
Input shape: (None, 111, 111, 32) | Output shape: (None, 111, 111, 32)
conv2d_1 (Conv2D)
Input shape: (None, 111, 111, 32) | Output shape: (None, 109, 109, 32)
batch_normalization_1 (BatchNormalization)
Input shape: (None, 109, 109, 32) | Output shape: (None, 109, 109, 32)
activation_1 (Activation)
Input shape: (None, 109, 109, 32) | Output shape: (None, 109, 109, 32)
conv2d_2 (Conv2D)
Input shape: (None, 109, 109, 32) | Output shape: (None, 109, 109, 64)
batch_normalization_2 (BatchNormalization)
Input shape: (None, 109, 109, 64) | Output shape: (None, 109, 109, 64)
activation_2 (Activation)
Input shape: (None, 109, 109, 64) | Output shape: (None, 109, 109, 64)
max_pooling2d (MaxPooling2D)
Input shape: (None, 109, 109, 64) | Output shape: (None, 54, 54, 64)
conv2d_3 (Conv2D)
Input shape: (None, 54, 54, 64) | Output shape: (None, 54, 54, 80)
batch_normalization_3 (BatchNormalization)
Input shape: (None, 54, 54, 80) | Output shape: (None, 54, 54, 80)
activation_3 (Activation)
Input shape: (None, 54, 54, 80) | Output shape: (None, 54, 54, 80)
conv2d_4 (Conv2D)
Input shape: (None, 54, 54, 80) | Output shape: (None, 52, 52, 192)
batch_normalization_4 (BatchNormalization)
Input shape: (None, 52, 52, 192) | Output shape: (None, 52, 52, 192)
activation_4 (Activation)
Input shape: (None, 52, 52, 192) | Output shape: (None, 52, 52, 192)
max_pooling2d_1 (MaxPooling2D)
Input shape: (None, 52, 52, 192) | Output shape: (None, 25, 25, 192)
conv2d_8 (Conv2D)
Input shape: (None, 25, 25, 192) | Output shape: (None, 25, 25, 64)
conv2d_6 (Conv2D)
Input shape: (None, 25, 25, 192) | Output shape: (None, 25, 25, 48)
average_pooling2d (AveragePooling2D)
Input shape: (None, 25, 25, 192) | Output shape: (None, 25, 25, 192)
conv2d_5 (Conv2D)
Input shape: (None, 25, 25, 192) | Output shape: (None, 25, 25, 64)
batch_normalization_8 (BatchNormalization)
Input shape: (None, 25, 25, 64) | Output shape: (None, 25, 25, 64)
batch_normalization_6 (BatchNormalization)
Input shape: (None, 25, 25, 48) | Output shape: (None, 25, 25, 48)
conv2d_11 (Conv2D)
Input shape: (None, 25, 25, 192) | Output shape: (None, 25, 25, 32)
batch_normalization_5 (BatchNormalization)
Input shape: (None, 25, 25, 64) | Output shape: (None, 25, 25, 64)
activation_8 (Activation)
Input shape: (None, 25, 25, 64) | Output shape: (None, 25, 25, 64)
activation_6 (Activation)
Input shape: (None, 25, 25, 48) | Output shape: (None, 25, 25, 48)
batch_normalization_11 (BatchNormalization)
Input shape: (None, 25, 25, 32) | Output shape: (None, 25, 25, 32)
conv2d_9 (Conv2D)
Input shape: (None, 25, 25, 64) | Output shape: (None, 25, 25, 96)
conv2d_7 (Conv2D)
Input shape: (None, 25, 25, 48) | Output shape: (None, 25, 25, 64)
batch_normalization_9 (BatchNormalization)
Input shape: (None, 25, 25, 96) | Output shape: (None, 25, 25, 96)
batch_normalization_7 (BatchNormalization)
Input shape: (None, 25, 25, 64) | Output shape: (None, 25, 25, 64)
activation_9 (Activation)
Input shape: (None, 25, 25, 96) | Output shape: (None, 25, 25, 96)
conv2d_10 (Conv2D)
activation_5 (Activation)