```
import numpy as np
import pandas as pd
# import file utilities
import os
import glob
import random

# import charting
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, ArtistAnimation
%matplotlib inline

from IPython.display import HTML

# import computer vision
import cv2


from google.colab import drive
drive.mount('/content/drive')
```

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

```
realdata = "/content/drive/MyDrive/celebs/real"
fakedata = "/content/drive/MyDrive/celebs/fake"


def save_frames_from_video(video_path, output_folder):
    # Open the video file
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Error opening video file: {video_path}")
        return

    # Create the output folder if it doesn't exist
    os.makedirs(output_folder, exist_ok=True)

    # Read and save frames from the video
    frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Save the frame as an image
        frame_path = os.path.join(output_folder, f"frame_{frame_count}.jpg")
        cv2.imwrite(frame_path, frame)

        frame_count += 1

    # Release the video capture object
    cap.release()

# Output folders for saving frames
real_output_folder = "real_frames"
fake_output_folder = "fake_frames"

# Save frames from one video in realdata folder
save_frames_from_video(os.path.join(realdata, os.listdir(realdata)[0]), real_output_folder)
# Save frames from one video in fakedata folder
save_frames_from_video(os.path.join(fakedata, os.listdir(fakedata)[0]), fake_output_folder)


import os
import shutil
from sklearn.model_selection import train_test_split

# Define your input folders
real_folder = "real_frames"
fake_folder = "fake_frames"
combined_folder = "combined_data"

# Create the combined folder if it doesn't exist
if not os.path.exists(combined_folder):
    os.makedirs(combined_folder)
```

```
# Move real images to the combined folder
for filename in os.listdir(real_folder):
    src_path = os.path.join(real_folder, filename)
    dst_path = os.path.join(combined_folder, filename)
    shutil.copy(src_path, dst_path)

# Move fake images to the combined folder
for filename in os.listdir(fake_folder):
    src_path = os.path.join(fake_folder, filename)
    dst_path = os.path.join(combined_folder, filename)
    shutil.copy(src_path, dst_path)

# Split the combined data into train and test sets
all_images = os.listdir(combined_folder)
train_images, test_images = train_test_split(all_images, test_size=0.2, random_state=42)

# Create train and test folders
train_folder = "train_data"
test_folder = "test_data"
os.makedirs(train_folder, exist_ok=True)
os.makedirs(test_folder, exist_ok=True)

# Move train images
for filename in train_images:
    src_path = os.path.join(combined_folder, filename)
    dst_path = os.path.join(train_folder, filename)
    shutil.copy(src_path, dst_path)

# Move test images
for filename in test_images:
    src_path = os.path.join(combined_folder, filename)
    dst_path = os.path.join(test_folder, filename)
    shutil.copy(src_path, dst_path)

print("Data split successfully!")
```

```
Data split successfully!
```

```
import os
import shutil
import random

# Define your input folders
real_folder = "real_frames"
fake_folder = "fake_frames"

# Define output folders
train_folder = "train"
test_folder = "test"

# Create output directories
os.makedirs(os.path.join(train_folder, "real"), exist_ok=True)
os.makedirs(os.path.join(train_folder, "fake"), exist_ok=True)
os.makedirs(os.path.join(test_folder, "real"), exist_ok=True)
os.makedirs(os.path.join(test_folder, "fake"), exist_ok=True)

# Function to split files into train and test sets
def split_data(source_folder, train_subfolder, test_subfolder, test_size=0.2):
    files = os.listdir(source_folder)
    random.shuffle(files)  # Shuffle files to ensure randomness
    split_index = int(len(files) * (1 - test_size))  # Calculate index for train-test split

    train_files = files[:split_index]
    test_files = files[split_index:]

    # Move files to the respective folders
    for file in train_files:
        shutil.copy(os.path.join(source_folder, file), os.path.join(train_subfolder, file))

    for file in test_files:
        shutil.copy(os.path.join(source_folder, file), os.path.join(test_subfolder, file))

# Split the real and fake frames
split_data(real_folder, os.path.join(train_folder, "real"), os.path.join(test_folder, "real"))
split_data(fake_folder, os.path.join(train_folder, "fake"), os.path.join(test_folder, "fake"))
```

```
print("Data organization completed!")
```

```
Data organization completed!
```

```
Start coding or generate with AI.
```

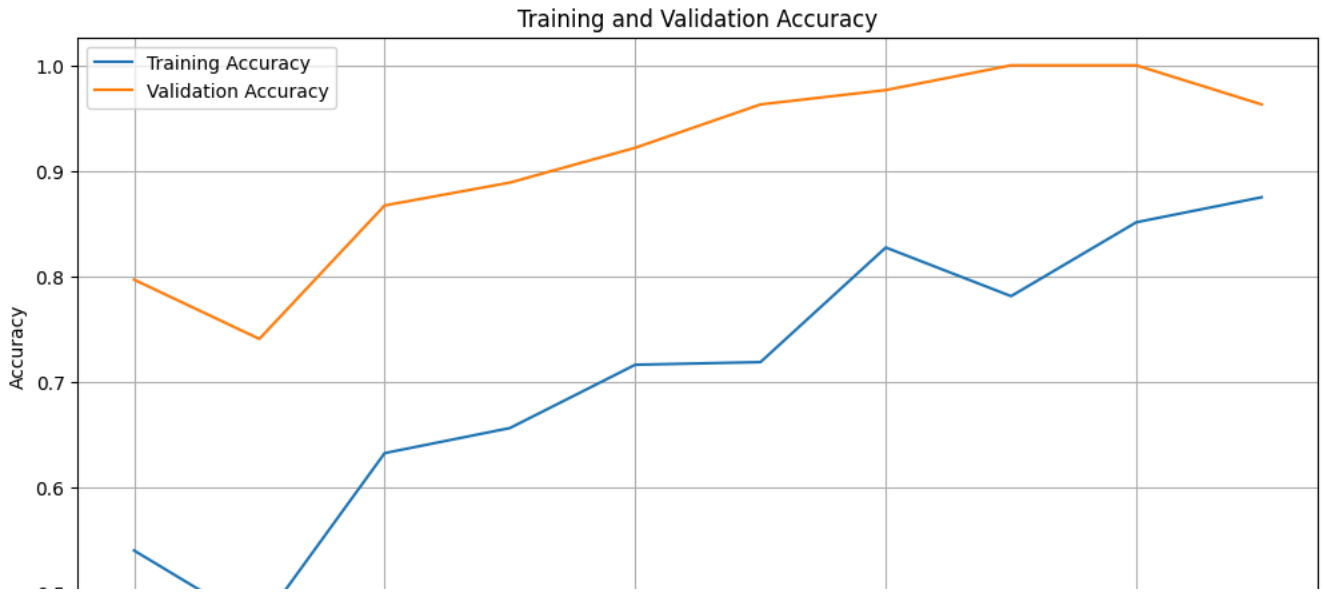
```
import os
import shutil
import random
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import cv2

def save_frames_from_video(video_path, output_folder):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Error opening video file: {video_path}")
        return

    os.makedirs(output_folder, exist_ok=True)

    frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame_path = os.path.join(output_folder, f"frame_{frame_count}.jpg")
        cv2.imwrite(frame_path, frame)
        frame_count += 1

    cap.release()

def split_data(source_folder, train_subfolder, test_subfolder, test_size=0.2):
    files = os.listdir(source_folder)
    random.shuffle(files)
    split_index = int(len(files) * (1 - test_size))

    train_files = files[:split_index]
    test_files = files[split_index:]

    for file in train_files:
        shutil.copy(os.path.join(source_folder, file), os.path.join(train_subfolder, file))
    for file in test_files:
        shutil.copy(os.path.join(source_folder, file), os.path.join(test_subfolder, file))

class DeepfakeDetector:
    def __init__(self, input_shape=(224, 224, 3), learning_rate=0.00001):
        self.input_shape = input_shape
        self.learning_rate = learning_rate
        self.model = self._build_model()

    def _build_model(self):
       base_model = MobileNetV2(input_shape=self.input_shape, include_top=False, weights='imagenet')
       base_model.trainable = False
       x = base_model.output
       x = GlobalAveragePooling2D()(x)
       x = Dense(256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
       x = Dropout(0.5)(x)  # Increased dropout
       predictions = Dense(1, activation='sigmoid')(x)
       model = Model(inputs=base_model.input, outputs=predictions)
       model.compile(optimizer=Adam(learning_rate=self.learning_rate), loss='binary_crossentropy', metrics=['accuracy'])
       return model


    def train(self, train_dir, valid_dir, batch_size=32, epochs=10):
        train_datagen = ImageDataGenerator(
            rescale=1./255,
            rotation_range=40,
```

```
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            brightness_range=[0.8, 1.2],
            horizontal_flip=True,
            fill_mode='nearest',
            channel_shift_range=20.0
        )

        valid_datagen = ImageDataGenerator(rescale=1./255)

        train_generator = train_datagen.flow_from_directory(train_folder, target_size=(224, 224),
                                                            batch_size=32, class_mode='binary')

        valid_generator = valid_datagen.flow_from_directory(test_folder, target_size=(224, 224),
                                                            batch_size=32, class_mode='binary')


        # Callbacks

        early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
        checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)
        lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-4 * 0.1**(epoch // 5))

        try:
            history = self.model.fit(train_generator,
                                     steps_per_epoch=train_generator.samples // batch_size,
                                     validation_data=valid_generator,
                                     validation_steps=valid_generator.samples // batch_size,
                                     epochs=epochs,
                                     callbacks=[early_stopping, checkpoint],
                                     verbose=1)
        except Exception as e:
            print(f"Error during training: {e}")

        return history



    def predict(self, image):
        if len(image.shape) == 3:
            image = np.expand_dims(image, axis=0)
        return self.model.predict(image)

    def save_model(self, path):
        self.model.save(path)

    @classmethod
    def load_model(cls, path):
        model = tf.keras.models.load_model(path)
        detector = cls()
        detector.model = model
        return detector


def extract_faces_from_frame(frame, face_detector):
    faces = face_detector.detect_faces(frame)  # Assuming face_detector is implemented elsewhere
    if len(faces) > 0:
        x, y, w, h = faces[0]['box']
        face = frame[y:y+h, x:x+w]
        x = Dropout(0.5)(x)
        face = cv2.resize(face, (224, 224))
        return face
    return None

def extract_frames(video_path, face_detector, num_frames=30):
    cap = cv2.VideoCapture(video_path)
    frames = []
    frame_count = 0

    while cap.isOpened() and frame_count < num_frames:
        ret, frame = cap.read()
        if not ret:
            break

        face = extract_faces_from_frame(frame, face_detector)
```

```
            if face is not None:
                frames.append(face)
                frame_count += 1

        cap.release()
        return frames



detector = DeepfakeDetector()
history=detector.train(os.path.join(train_folder), os.path.join(test_folder), batch_size=32, epochs=10)
```

```
Found 617 images belonging to 2 classes.
Found 155 images belonging to 2 classes.
Epoch 1/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 54s 2s/step - accuracy: 0.5503 - loss: 5.1279 - val_accuracy: 0.7969 - val_loss: 4.8155
Epoch 2/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 5s 211ms/step - accuracy: 0.4688 - loss: 5.1498 - val_accuracy: 0.7407 - val_loss: 4.8412
Epoch 3/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 70s 2s/step - accuracy: 0.6245 - loss: 4.9625 - val_accuracy: 0.8672 - val_loss: 4.6713
Epoch 4/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 158ms/step - accuracy: 0.6562 - loss: 4.8876 - val_accuracy: 0.8889 - val_loss: 4.6781
Epoch 5/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.6894 - loss: 4.7949 - val_accuracy: 0.9219 - val_loss: 4.5627
Epoch 6/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.7188 - loss: 4.7737 - val_accuracy: 0.9630 - val_loss: 4.5289
Epoch 7/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 74s 2s/step - accuracy: 0.8259 - loss: 4.6361 - val_accuracy: 0.9766 - val_loss: 4.4740
Epoch 8/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 149ms/step - accuracy: 0.7812 - loss: 4.7009 - val_accuracy: 1.0000 - val_loss: 4.4401
Epoch 9/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.8244 - loss: 4.5725 - val_accuracy: 1.0000 - val_loss: 4.3990
Epoch 10/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 91ms/step - accuracy: 0.8750 - loss: 4.5150 - val_accuracy: 0.9630 - val_loss: 4.3935
```

```
import matplotlib.pyplot as plt

# Assuming 'history' is the variable storing the training history

# Plotting training and validation accuracy
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
1.0
0.9
0.8
0.7
0.6