```python
import os
import urllib.request
import bz2
import dlib
import numpy as np
from imutils import face_utils
from scipy.spatial import distance
import cv2
import tempfile
import datetime
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow


url = "http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2"
compressed_path = "shape_predictor_68_face_landmarks.dat.bz2"
extracted_path = "shape_predictor_68_face_landmarks.dat"

# Download the compressed file
if not os.path.exists(compressed_path):
    print("Downloading the landmark model...")
    urllib.request.urlretrieve(url, compressed_path)
    print("Download complete.")
else:
    print("File already downloaded.")

# Extract the .bz2 file
if not os.path.exists(extracted_path):
    print("Extracting the model...")
    with bz2.BZ2File(compressed_path, "rb") as f_in:
        with open(extracted_path, "wb") as f_out:
            f_out.write(f_in.read())
    print("Extraction complete.")
else:
    print("Model already extracted.")

# Confirm path
print(f"Model is ready to use at: {os.path.abspath(extracted_path)}")

predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

detector = dlib.get_frontal_face_detector()
predictor_path = "shape_predictor_68_face_landmarks.dat"  # Download this from Dlib
predictor = dlib.shape_predictor(predictor_path)

# Lip landmark indices (outer + inner lips)
LIP_IDX = list(range(48, 61))

LIP_POINTS = list(range(48, 61))
```

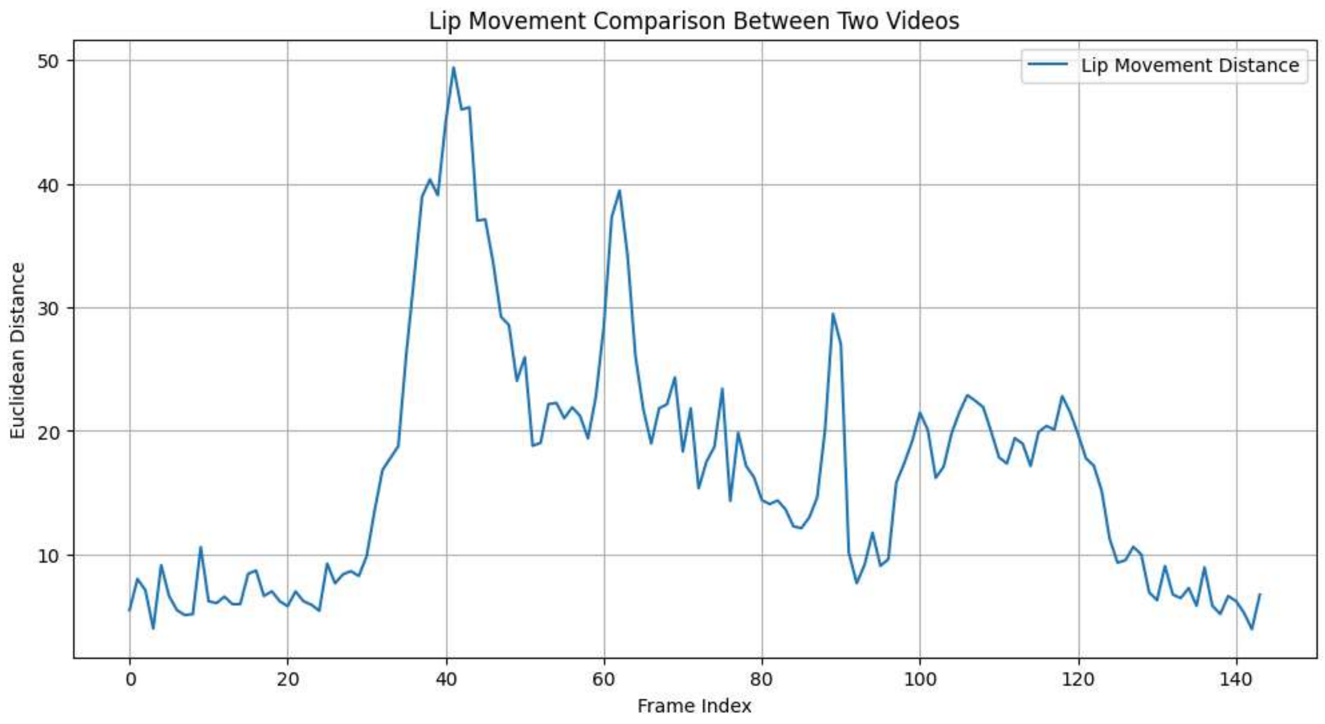
```
Downloading the landmark model...
Download complete.
Extracting the model...
Extraction complete.
Model is ready to use at: /content/shape_predictor_68_face_landmarks.dat
```

```python
def extract_lip_movements(video_path):
    cap = cv2.VideoCapture(video_path)
    lip_movements = []

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        rects = detector(gray, 0)

        for rect in rects:
            shape = predictor(gray, rect)
            shape = face_utils.shape_to_np(shape)
            lips = shape[LIP_IDX]
            lips_centered = lips - lips.mean(axis=0)
            lip_movements.append(lips_centered.flatten())  # Flatten to 1D for comparison
        cap.release()
        return np.array(lip_movements)

    def compare_lip_movements(movements1, movements2):
        # Pad shorter sequence
        min_len = min(len(movements1), len(movements2))
        movements1 = movements1[:min_len]
        movements2 = movements2[:min_len]

        # Use DTW, cosine similarity or Euclidean distance
        distances = [distance.euclidean(a, b) for a, b in zip(movements1, movements2)]
        return np.mean(distances)

    # Record video using OpenCV and save to a temporary file
    def record_video_from_webcam(duration, fps=20):
        cap = cv2.VideoCapture(0)
        width = int(cap.get(3))
        height = int(cap.get(4))

        # Create a temporary file to store the video
        temp_video = tempfile.NamedTemporaryFile(suffix=".mp4", delete=False)
        temp_filename = temp_video.name

        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(temp_filename, fourcc, fps, (width, height))

        num_frames = duration * fps
        print(f"Recording {duration} seconds of video...")

        for _ in range(num_frames):
            ret, frame = cap.read()
            if not ret:
                break
            out.write(frame)
            cv2.imshow('Recording...', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        print(f"Saved to {temp_filename}")
        cap.release()
        out.release()
        cv2.destroyAllWindows()

        return temp_filename

    def videoLength(path):
      data = cv2.VideoCapture(str(path))
      frames = data.get(cv2.CAP_PROP_FRAME_COUNT)
      fps = data.get(cv2.CAP_PROP_FPS)
      seconds = round(frames / fps)
      video_time = datetime.timedelta(seconds=seconds)
      return seconds


    def extract_lip_sequence(video_path):
        cap = cv2.VideoCapture(video_path)
        sequence = []

        while True:
            ret, frame = cap.read()
            if not ret:
                break
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = detector(gray)
            if len(faces) == 0:
                continue

            shape = predictor(gray, faces[0])
            shape = face_utils.shape_to_np(shape)
            lips = shape[LIP_IDX]
            lips_centered = lips - lips.mean(axis=0)
            sequence.append(lips_centered.flatten())

        cap.release()
        return np.array(sequence)
```

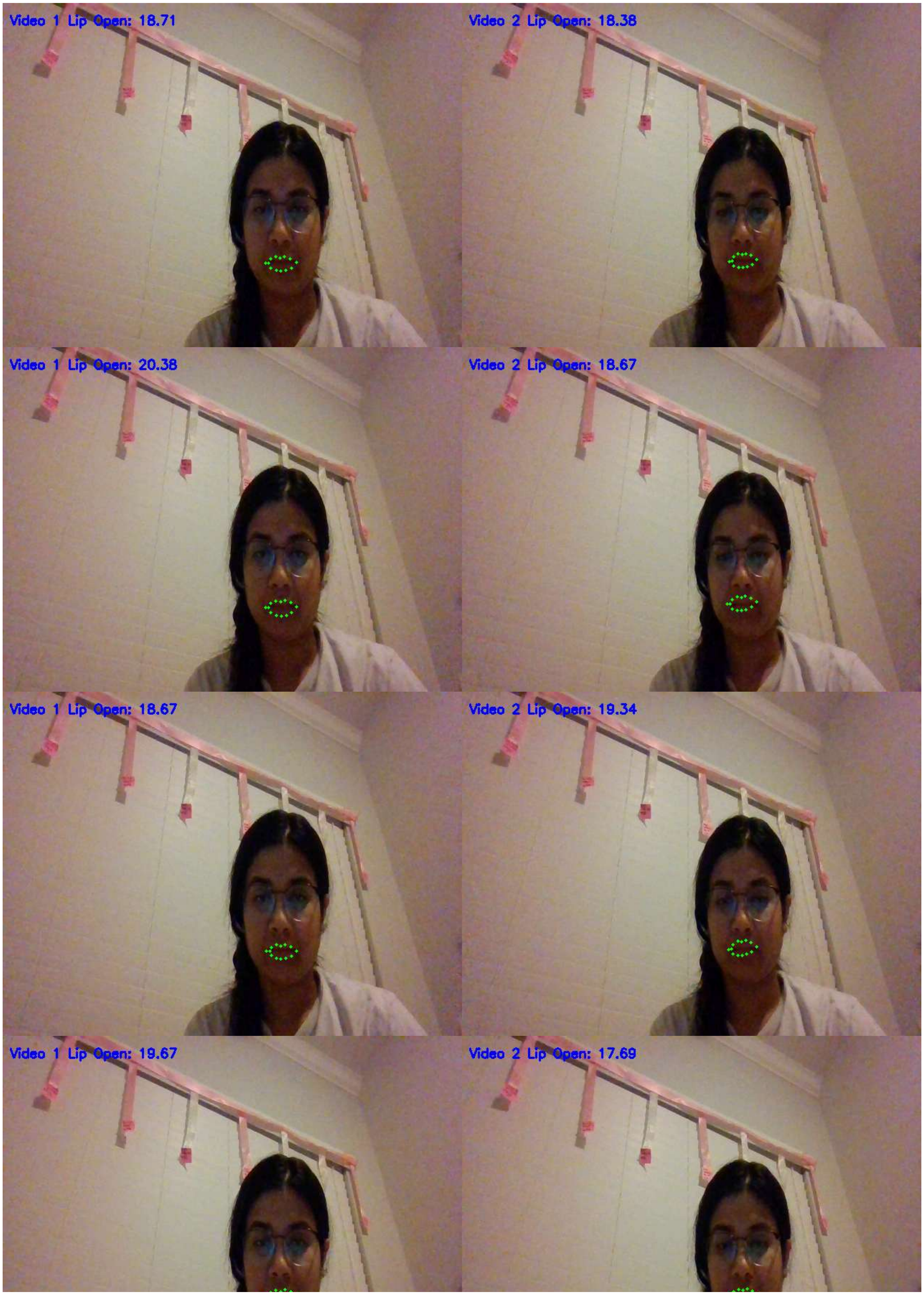
```
def get_lip_distance(landmarks):
    top_lip = np.mean([landmarks[i] for i in [50, 51, 52]], axis=0)
    bottom_lip = np.mean([landmarks[i] for i in [56, 57, 58]], axis=0)
    return np.linalg.norm(top_lip - bottom_lip)

def extract_landmarks(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = detector(gray)
    if faces:
        shape = predictor(gray, faces[0])
        coords = np.array([[p.x, p.y] for p in shape.parts()])
        return coords
    return None

def annotate_frame(frame, landmarks, label, value):
    if landmarks is not None:
        for (x, y) in landmarks[LIP_POINTS]:
            cv2.circle(frame, (x, y), 2, (0, 255, 0), -1)
    cv2.putText(frame, f"{label} Lip Open: {value:.2f}", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
    return frame


video_actual = "/content/moreLipMovement.mp4"
video_compare = "/content/lessLipMovement.mp4"

lips_actual = extract_lip_movements(video_actual)
lips_other = extract_lip_movements(video_compare)

similarity_score = compare_lip_movements(lips_actual, lips_other)
print(f"Similarity (lower is better): {similarity_score:.2f}")
```

```
Similarity (lower is better): 16.81
```

```
seq1 = extract_lip_sequence("/content/moreLipMovement.mp4")
seq2 = extract_lip_sequence("/content/lessLipMovement.mp4")
#seq2 = extract_lip_sequence("/content/v2.mp4")

# Match lengths
min_len = min(len(seq1), len(seq2))
seq1 = seq1[:min_len]
seq2 = seq2[:min_len]

# Compute frame-wise Euclidean distances
distances = [distance.euclidean(a, b) for a, b in zip(seq1, seq2)]

# Plot the comparison graph
plt.figure(figsize=(12, 6))
plt.plot(distances, label="Lip Movement Distance")
plt.xlabel("Frame Index")
plt.ylabel("Euclidean Distance")
plt.title("Lip Movement Comparison Between Two Videos")
plt.legend()
plt.grid(True)
plt.show()
```



```
# Load the two videos
cap1 = cv2.VideoCapture('/content/moreLipMovement.mp4')
cap2 = cv2.VideoCapture('/content/lessLipMovement.mp4')
#cap2 = cv2.VideoCapture(video_compare)


# Output video setup
width, height = 640, 480
fps = 20
out = cv2.VideoWriter('output_combined.avi', cv2.VideoWriter_fourcc(*'XVID'), fps, (width * 2, height))

while True:
    ret1, frame1 = cap1.read()
    ret2, frame2 = cap2.read()

    if not ret1 or not ret2:
        break

    frame1 = cv2.resize(frame1, (width, height))
    frame2 = cv2.resize(frame2, (width, height))

    lm1 = extract_landmarks(frame1)
    lm2 = extract_landmarks(frame2)

    dist1 = get_lip_distance(lm1) if lm1 is not None else 0
    dist2 = get_lip_distance(lm2) if lm2 is not None else 0

    frame1 = annotate_frame(frame1, lm1, "Video 1", dist1)
    frame2 = annotate_frame(frame2, lm2, "Video 2", dist2)

    combined = np.hstack((frame1, frame2))
    out.write(combined)

    #cv2.imshow("Lip Movement Comparison", combined)
    cv2_imshow(combined)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap1.release()
cap2.release()
out.release()
cv2.destroyAllWindows()
```


Video 1 Lip Open: 18.71
Video 2 Lip Open: 18.38
Video 1 Lip Open: 20.38
Video 2 Lip Open: 18.67
Video 1 Lip Open: 18.67
Video 2 Lip Open: 19.34
Video 1 Lip Open: 19.67
Video 2 Lip Open: 17.69