Test Cases and Examples

The following Test cases are included:

- Record and Display a video from Webcam
- Detect faces from an image
- Detect Facial Landmarks from a detected image
    - Facial landmarks can be eyes, nose, lip, chin etc
- Encode a Detected and Landmarked face
- Compare two encoded faces
- Serialize and Deserialize data from JSON files
- Mail Images from Python using SMTP and TLS Security

Record and Display a Video

```
import cv2
from datetime import datetime

# Set video parameters
frame_width = 640
frame_height = 480
fps = 15.0

# Get current date and time
current_datetime = datetime.now().strftime("%Y%m%d-%H%M%S")

# Specify custom directory for output
output_directory = "output/"

# Generate output filename with custom directory
output_file = f"{output_directory}/output_{current_datetime}.mp4"

# Capture frames from input source (e.g., webcam)
cap = cv2.VideoCapture(0)



# Create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_file, fourcc, fps, (frame_width, frame_height))

while True:
    # Read frame from the video capture
    ret, frame = cap.read()

    if not ret:
        break

    # Flip the frame horizontally
    flipped_frame = cv2.flip(frame, 1)
```

```
    # Write the flipped frame to the video file
    # out.write(flipped_frame)

    # Display the flipped frame (optional)
    cv2.imshow('Flipped Frame', flipped_frame)

    # Stop recording if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
# out.release()
cap.release()
cv2.destroyAllWindows()
```

Detect Faces from an image

```
import dlib
import cv2

# Load the image
path = input("Enter the path of the image: ")
image = cv2.imread(path)

# resize the image
small_image = cv2.resize(image, (0, 0), fx=0.50, fy=0.50)

# Initialize the face detector
detector = dlib.get_frontal_face_detector()

# Detect all faces in the image
face_rects = detector(small_image, 0)

# Draw each rectangle on the image
for rect in face_rects:
    cv2.rectangle(small_image, (rect.left(),rect.top()),
(rect.right(),rect.bottom()), (255, 0, 0), 2)

# Display the image with detected faces
cv2.imshow('Detected Faces', small_image)

# display the object types

print("small_image: ", type(small_image))
print("detector: ", type(detector))
print("face_rects: ", type(face_rects))

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Detect Facial Landmarks (eye, nose, lips etc) from an image

```
# import required packages
import dlib
import cv2
import face_recognition_models

# note: OpenCV uses BGR color scheme by default to process images,
# whereas dlib uses RGB, so they are converted as needed to process faces properly

# import the image
img = cv2.imread('example.jpg')

print("img type: ", type(img))

# resize the image to half of it's size
img_resized = cv2.resize(img, (0, 0), fx=0.25, fy=0.25)

print("img_resized type: ", type(img_resized))

# convert image from BGR to RGB
img_rgb = cv2.cvtColor(img_resized, cv2.COLOR_BGR2RGB)

print("img_rgb type: ", type(img_rgb))

# initialize face detector
detector = dlib.get_frontal_face_detector()

# detect faces from an image
faces = detector(img_rgb, 1)


# Draw rectangles on the image
for face in faces:
    cv2.rectangle(img_rgb, (face.left(), face.top()), (face.right(),
face.bottom()), (0, 255, 0), 2)


# initialize the 68 point face predictor
predictor_model = face_recognition_models.pose_predictor_model_location()
predictor = dlib.shape_predictor(predictor_model)


# detect face landmarks
landmarks = []
for face in faces:
    face_landmarks = predictor(img_rgb, face)
    landmarks.append(face_landmarks.parts())
```

```
# this code is for multi-face prediction
for landmark in landmarks:
    for point in landmark:
        x = point.x
        y = point.y
        # place the landmark coordinate in the image using OpenCV
        cv2.circle(img_rgb, (x, y), 1, (0, 0, 255), -1)

# print no. of detected faces in the image
text = "Detected faces: " + str(len(faces))
# set font parameters
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
font_scale = 0.7
font_color = (0, 0, 0)
font_thickness = 1

# set image coordinates
img_width, _, _ = img_rgb.shape
text_size, _ = cv2.getTextSize(text, font, font_scale, font_thickness)
text_width, _ = text_size
text_position = (10, img_width - 10)

cv2.rectangle(img_rgb, (2, img_width - 30), (text_width + 15, img_width), (255,
255, 255), thickness=cv2.FILLED)
cv2.putText(img_rgb, text, text_position, font, font_scale, font_color,
font_thickness)

# convert RGB image back to BGR
img_bgr = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)

# display the image in the window
cv2.imshow('Face Landmark', img_bgr)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Encode a Detected Face

```
# import required packages
import dlib
import cv2
import face_recognition_models
# import Tests.FaceLandmarks      # face landmark detection is already tested in
this file

# load the images
img = cv2.imread('example.jpg')

# resize the image
img_resized = cv2.resize(img, (0, 0), fx=0.50, fy=0.50)

# convert from BGR to RGB
```

```
img_rgb = cv2.cvtColor(img_resized, cv2.COLOR_BGR2RGB)

# load face detector
detector = dlib.get_frontal_face_detector()

# detect faces in the image
faces = detector(img_rgb, 1)

# load landmark predictor
predictor_model = face_recognition_models.pose_predictor_model_location()
predictor = dlib.shape_predictor(predictor_model)

# predict face landmarks
landmarks = []
for face in faces:
    face_landmarks = predictor(img_rgb, face)
    # face_landmarks.parts() is useful if we want to draw the points(facial
landmarks) on the face image
    # the encoder needs the whole object_detection object (without the points
list), so the whole object is added as a landmark element
    landmarks.append(face_landmarks)

# load face encoder
face_recognition_model = face_recognition_models.face_recognition_model_location()
face_encoder = dlib.face_recognition_model_v1(face_recognition_model)

# encode the faces
encodings = []
for landmark in landmarks:
    print(type(landmark))
    encoded_face = face_encoder.compute_face_descriptor(img_rgb, landmark, 1)
    encodings.append(encoded_face)
```

Compare Two Encoded Faces

```
# import necessary packages
import dlib
import cv2
import numpy as np
import face_recognition_models

# load and resize two images
img1 = cv2.imread('compare2.jpg')
img2 = cv2.imread('compare22.jpg')

# img1_resized = cv2.resize(img1, (0, 0), fx=0.5, fy=0.5)
# img2_resized = cv2.resize(img2, (0, 0), fx=0.5, fy=0.5)

# change the image from BGR to RGB
img1_rgb = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
```

```
img2_rgb = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

# load face detector and detect images
detector = dlib.get_frontal_face_detector()
faces1 = detector(img1_rgb, 1)
faces2 = detector(img2_rgb, 1)

# find landmarks
predictor_model = face_recognition_models.pose_predictor_model_location()
predictor = dlib.shape_predictor(predictor_model)

landmarks1 = []
landmarks2 = []

for face in faces1:
    face_landmarks = predictor(img1_rgb, face)
    landmarks1.append(face_landmarks)

for face in faces2:
    face_landmarks = predictor(img2_rgb, face)
    landmarks2.append(face_landmarks)

# encode the faces
encoder_model = face_recognition_models.face_recognition_model_location()
encoder = dlib.face_recognition_model_v1(encoder_model)

encoding1 = []
encoding2 = []

encoded_face1_np = []
encoded_face2_np = []

for landmark in landmarks1:
    encoded_face1 = encoder.compute_face_descriptor(img1_rgb, landmark, 1)
    encoded_face1_np = np.array(encoded_face1)

for landmark in landmarks2:
    encoded_face2 = encoder.compute_face_descriptor(img2_rgb, landmark, 1)
    encoded_face2_np = np.array(encoded_face2)

# compare the faces
same_face = False
tolerance = 0.5

# compare = encoding1 - encoding2
# axis = 0  because there is only one face to compare with
compare_float = np.linalg.norm(encoded_face1_np - encoded_face2_np)
if compare_float <= tolerance:
    same_face = True

# print no. of detected faces in the image
text = "Same Face: " + str(same_face)
print(text)
# set font parameters
font = cv2.FONT_HERSHEY_DUPLEX
```

```
font_scale = 0.6
font_color = (0, 0, 0)
font_thickness = 1

# set image coordinates
img1_width, _, _ = img1_rgb.shape


# put if faces matches
for face in faces1:
    cv2.rectangle(img1_rgb, (face.left(), face.top()), (face.right(),
face.bottom()), (0, 255, 0), 2)
    cv2.rectangle(img1_rgb, (face.left(), face.bottom()), (face.right(),
face.bottom() + 50), (0, 255, 0), cv2.FILLED)
    cv2.putText(img1_rgb, text, (face.left() + 5, face.bottom() + 35), font,
font_scale, font_color, font_thickness)

# convert RGB image back to BGR
img_bgr = cv2.cvtColor(img1_rgb, cv2.COLOR_RGB2BGR)

# display the image in the window
cv2.namedWindow("Face Landmark", cv2.WINDOW_KEEPRATIO)
cv2.imshow('Face Landmark', img_bgr)
if cv2.waitKey(0) == ord('q'):
    cv2.destroyAllWindows()
```

Serialize and Deserialize data from JSON file

```
import os
import json

# data is stored in the json file as list of dictionaries

# take data as user input
data = {}

data["Name"] = input("Enter Name: ")
data["Course"] = input("Enter Course: ")
data["Sem"] = int(input("Enter Semester: "))
data["Roll"] = int(input("Enter Roll No: "))

# for key, value in data.items():
#     print(key,": ",value)

# dump/seralize a data in a json file

path = "Tests/json_data/test.json"

# open json file pointers
# fp(json file as append mode)
# fp_truncate(json file as append binary mode)
```

```
if os.path.exists(path):
    fp = open(path, "a")
    fp_truncate = open(path, "ab")
    print("file opened")
else:
    print(path+" doesn't exist")

# check if the file is empty
# this block is to manage the basic syntax of the json file
if os.stat(path).st_size == 0:
    print("File is empty. dumping...")
    fp.write("[\n")

else:
    print("File isn't empty. dumping...")
    fp_truncate.seek(-1, 2)
    fp_truncate.truncate()
    fp.write(",\n")

# write/serialize data in json file
json.dump(data, fp, indent=4, separators=(',', ':'))
fp.write("\n]")
print("Student data dumped to "+path)

# close opened files
fp.close()
fp_truncate.close()

# read/deserialize data from a json file
fp_read = open(path, "r")
# list of dictionaries
des_data = json.load(fp_read)
print("Student dataset read successfully")

# #print the deserialized data
for data in des_data:
    for key, value in data.items():
        print(key,": ",value)
```

Mail Images from Python

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.image import MIMEImage


def send_email(sender_email, sender_password, receiver_email, subject, message,
image_path):
    # Create a multipart message
    msg = MIMEMultipart()
```

```
    msg["From"] = sender_email
    msg["To"] = receiver_email
    msg["Subject"] = subject

    # Add the message body
    msg.attach(MIMEText(message, "plain"))

    # Load and attach the image
    with open(image_path, "rb") as fp:
        img = MIMEImage(fp.read())
    img.add_header("Content-Disposition", "attachment", filename="img.jpg")
    msg.attach(img)

    # Establish an SMTP connection
    with smtplib.SMTP("smtp.gmail.com", 587) as server:
        server.starttls()
        print("Logging in : (" + sender_email + ")...")
        server.login(sender_email, sender_password)
        print("Logged in to server.")
        print("Sending mail to: " + receiver_email + "...")
        server.sendmail(sender_email, receiver_email, msg.as_string())
        print("Mail sent successfully.")


# Example usage
sender_email = "example@gmail.com"
sender_password = "password"
receiver_email = "example2@gmail.com"
subject = "Sending an image via email"
message = "Hello, I'm sending you an image."
image_path = "compare2.jpg"

send_email(sender_email, sender_password, receiver_email, subject, message,
image_path)
```