

OpenCV

What is OpenCv

OpenCV, short for "Open Source Computer Vision Library," is an open-source computer vision and machine learning software library. It provides a wide range of tools and functions for tasks related to computer vision, image processing, and machine learning. OpenCV is written in C++ and can also be used with Python and other programming languages through various bindings.

Here are some key features and functionalities of OpenCV:

1. **Image Processing:** OpenCV includes a comprehensive set of functions for image manipulation, filtering, enhancement, and transformation. It can handle various image formats and provides tools for resizing, cropping, and color manipulation.
2. **Computer Vision:** OpenCV offers a rich set of computer vision algorithms, including object detection, feature extraction, image stitching, optical flow analysis, and camera calibration. These tools are often used in applications like object tracking, facial recognition, and augmented reality.
3. **Machine Learning:** OpenCV integrates with machine learning libraries like TensorFlow and PyTorch, allowing users to build and train machine learning models for tasks such as image classification, object detection, and image segmentation.
4. **Real-time Processing:** OpenCV is optimized for real-time image and video processing, making it suitable for applications like video surveillance, robotics, and autonomous vehicles.
5. **Cross-Platform:** OpenCV is designed to work on various platforms, including Windows, macOS, Linux, iOS, and Android, making it versatile for a wide range of applications.
6. **Open-Source:** OpenCV is open-source software, which means it is freely available for both personal and commercial use. This has contributed to its widespread adoption and community support.
7. **Rich Documentation:** OpenCV provides extensive documentation and a large user community, making it accessible for developers and researchers.

Install OpenCv

Windows

Step 1:

Install the last version of Python: <https://www.python.org/downloads/>



To check if the Python is installed properly: open the terminal and write this below

```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 10.0.19042.546]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\jas>python --version
Python 3.8.5
```

Step 2: Install OpenCV by writing this command in the terminal (`pip install opencv-contrib-python`)

```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 10.0.19042.546]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\jas>python --version
Python 3.8.5

C:\Users\jas>pip install opencv-contrib-python
Requirement already satisfied: opencv-contrib-python in c:\users\jas\appdata\local\programs\python\python38\lib\site-packages (4.4.0.42)
Requirement already satisfied: numpy>=1.17.3 in c:\users\jas\appdata\local\programs\python\python38\lib\site-packages (from opencv-contrib-python) (1.18.5)

C:\Users\jas>
```

then need to install other packages to run OpenCV smoothly (`pip install caer`)

```
C:\Users\jas>pip install caer
Requirement already satisfied: caer in c:\users\jas\appdata\local\programs\python\python38\lib\site-packages (1.7.4)
Requirement already satisfied: numpy in c:\users\jas\appdata\local\programs\python\python38\lib\site-packages (from caer) (1.18.5)
Requirement already satisfied: opencv-contrib-python in c:\users\jas\appdata\local\programs\python\python38\lib\site-packages (from caer) (4.4.0.42)
Requirement already satisfied: h5py in c:\users\jas\appdata\local\programs\python\python38\lib\site-packages (from caer) (2.10.0)
Requirement already satisfied: six in c:\users\jas\appdata\local\programs\python\python38\lib\site-packages (from h5py->caer) (1.12.0)

C:\Users\jas>
```

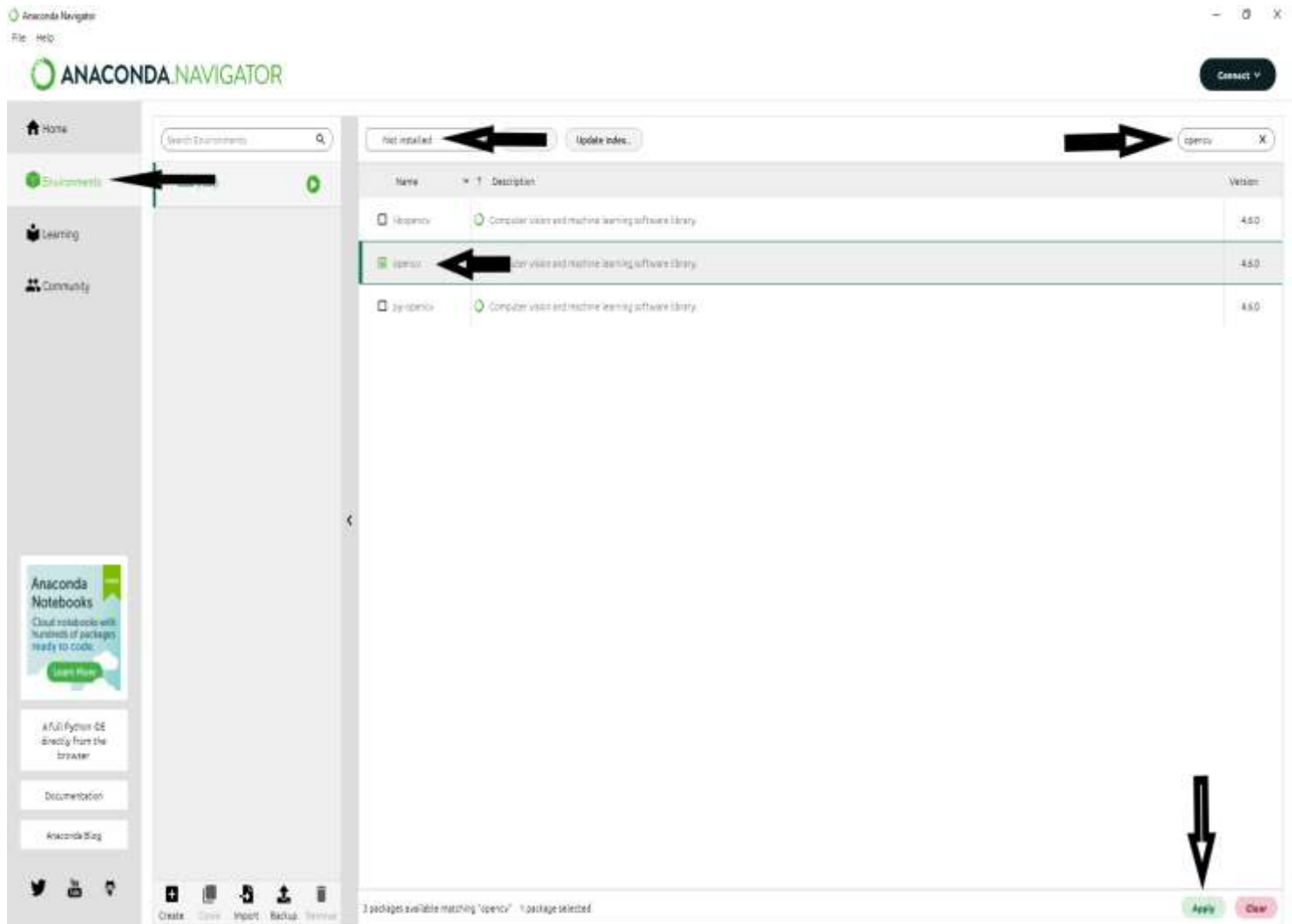
Done with installation.

Install openvcl with anaconda

Install anaconda

Download anaconda : <https://www.anaconda.com/download>

First method : open anaconda navigator



1-then select environment

2-search about opencv

3-chose not install list

4-search of opencv

5-select apply

Second method : using anaconda command prompt

1- open the prompt

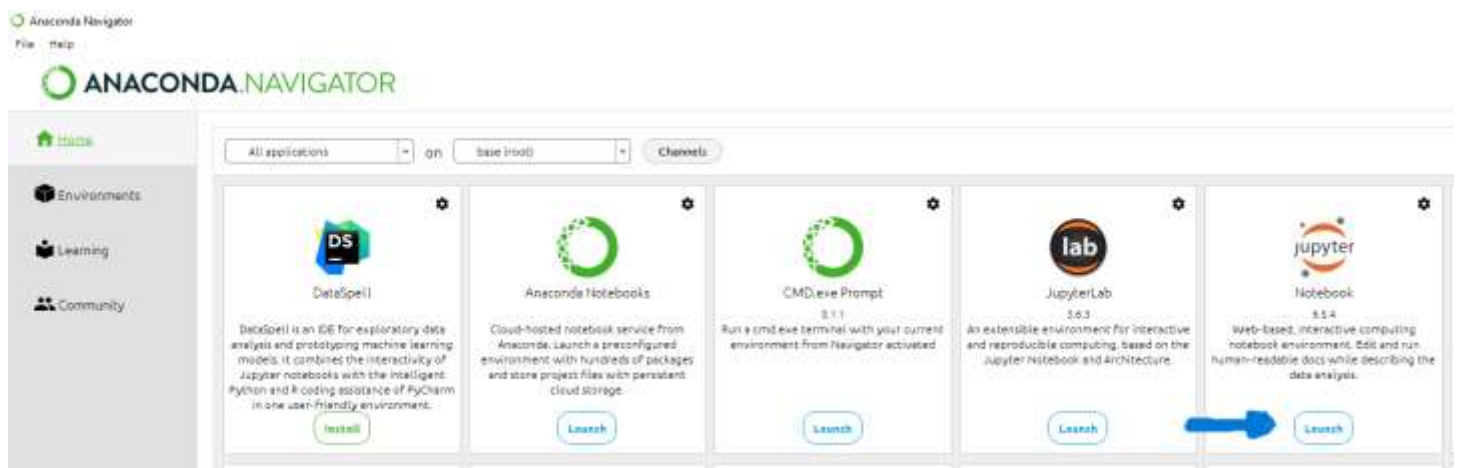
2- write command (**pip install opencv-contrib-python**)

```
Anaconda Prompt
(base) C:\Users\USER> pip install opencv-contrib-python
Collecting opencv-contrib-python
  Obtaining dependency information for opencv-contrib-python from https://files.pythonhosted.org/packages/05/33/5a6436146bda09c69decc456cfb54f41d52fbcf558fe91e6df7bdde6cce0/opencv_contrib_python-4.8.0.76-cp37-abi3-win_amd64.whl.metadata
  Using cached opencv_contrib_python-4.8.0.76-cp37-abi3-win_amd64.whl.metadata (20 kB)
Requirement already satisfied: numpy>=1.21.2 in c:\users\user\anaconda3\lib\site-packages (from opencv-contrib-python) (1.24.3)
Using cached opencv_contrib_python-4.8.0.76-cp37-abi3-win_amd64.whl (44.8 MB)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.8.0.76

(base) C:\Users\USER>
```

third method : using jupyter notebook

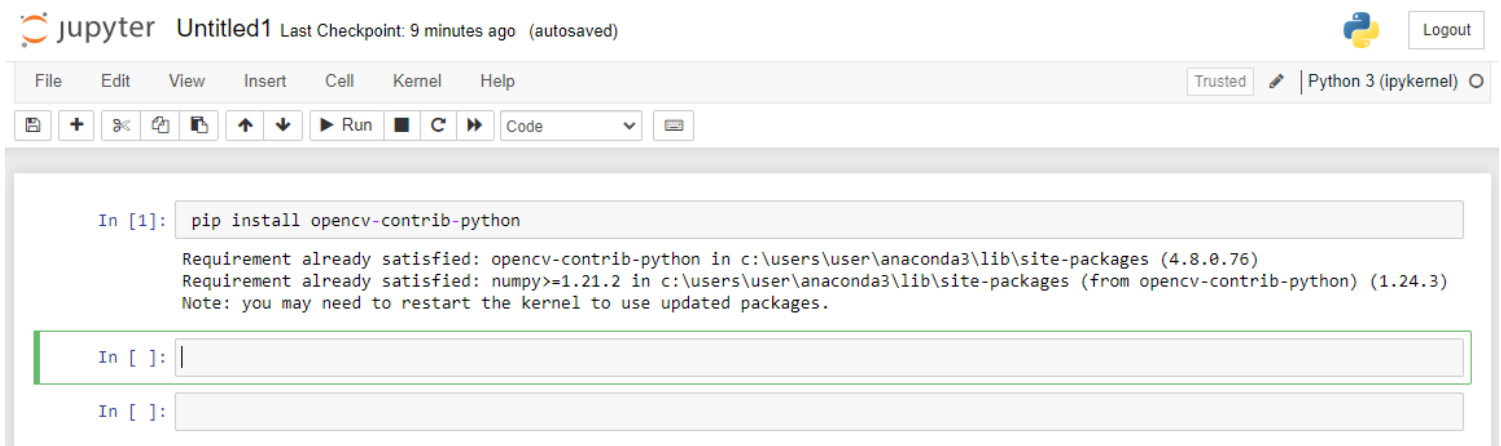
1- lunch jupyter notebook from anaconda navigator



2- then open new python file editor

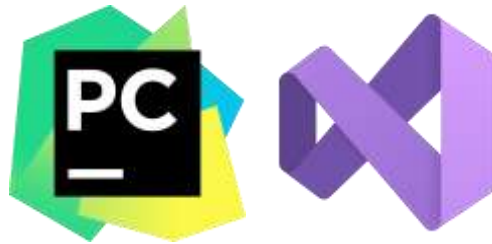


3- write command (**pip install opencv-contrib-python**)



Exercise one: Reading images

Step 1: Create a python file in Visualstudio or PyCharm Community



Step 2 : Write a code to read the image

```
# Import the OpenCV library and alias it as 'cv' for easier use.
import cv2 as cv

# Load an image from the file 'image.jpg' located in the 'photos' directory.
# The 'imread' function reads an image and stores it in the 'img' variable.
img = cv.imread('photos/image.jpg')

# Display the loaded image in a window with the title 'photo'.
# The 'imshow' function is used for this purpose.
cv.imshow('photo', img)

# Wait indefinitely for a key press event. This keeps the image window open
# until a key is pressed. The argument '0' means that it will wait forever.
cv.waitKey(0)

# After a key is pressed, the window will close, and the program will terminate.
```


Exercise two: reading a video

code for reading a video

```
# Import the OpenCV library and alias it as 'cv' for easier use.
import cv2 as cv

# Create a VideoCapture object, which is used to capture video frames from a file.
# In this case, it opens and reads frames from 'video.mp4'.
capture = cv.VideoCapture('video.mp4')

# Create an infinite loop to continuously read and display video frames.
while True:
    # Read the next frame from the video capture.
    # 'isTrue' will be True if a frame was successfully read, and 'frame' will contain the frame data.
    isTrue, frame = capture.read()

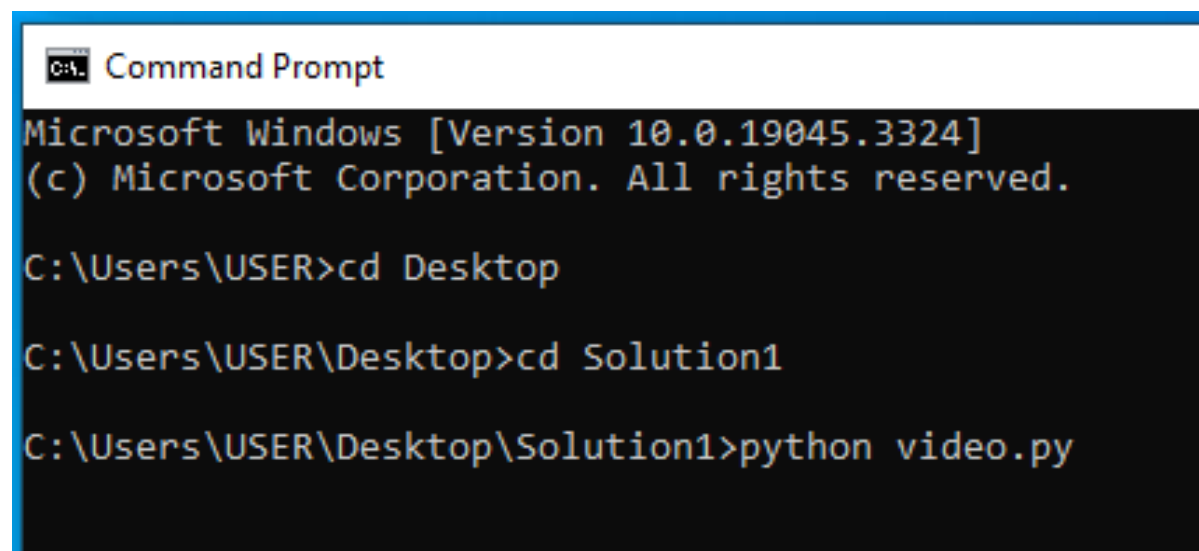
    # Display the current frame in a window titled 'video1'.
    cv.imshow('video1', frame)

    # Wait for a key press event for 20 milliseconds and check if the pressed key is 'd'.
    # If the 'd' key is pressed, break out of the loop and exit.
    if cv.waitKey(20) & 0xFF == ord('d'):
        break

# Release the video capture object to free up resources.
capture.release()

# Close all OpenCV windows.
cv.destroyAllWindows()
```

Step 3 : Run the file in the Visualstudio or PyCharm Community or open terminal find the path for the python file then write the code below



```
C:\> Command Prompt

Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

C:\Users\USER>cd Desktop

C:\Users\USER\Desktop>cd Solution1

C:\Users\USER\Desktop\Solution1>python video.py
```

Image transformations

Exercise three : Resizing

Step :1 for Resizing and Rescaling Frames for images will use the code below

```
# Import the OpenCV library and alias it as 'cv' for easier use.
import cv2 as cv

# Load an image from the file '12.jpg'.
img = cv.imread('12.jpg')

# Display the original image in a window titled 'normal'.
cv.imshow('normal', img)

# Define a function called 'rescaleFrame' that takes a 'frame' and an optional 'scale' parameter.
def rescaleFrame(frame, scale=0.75):
    # Calculate the new dimensions for the frame based on the specified 'scale'.
    width = int(frame.shape[1] * scale)
    height = int(frame.shape[0] * scale)
    dimensions = (width, height)

    # Resize the frame using the calculated dimensions and specify the interpolation method as 'cv.INTER_AREA'.
    resized_frame = cv.resize(frame, dimensions, interpolation=cv.INTER_AREA)

    # Return the resized frame.
    return resized_frame

# Call the 'rescaleFrame' function to resize the 'img' using the default scale of 0.75.
resized_image = rescaleFrame(img)

# Display the resized image in a window titled 'resized_image'.
cv.imshow('resized_image', resized_image)

# Wait indefinitely for a key press event. This keeps the image window open until a key is pressed.
cv.waitKey(0)

# After a key is pressed, the window will close, and the program will terminate.
```

Exercise four: For resizing the video will use the code below

```
# Import the OpenCV library and alias it as 'cv' for easier use.
import cv2 as cv
# Define a function called 'rescaleFrame' that takes a 'frame' and an optional 'scale' parameter.
def rescaleFrame(frame, scale=0.75):
    # Calculate the new dimensions for the frame based on the specified 'scale'.
    width = int(frame.shape[1] * scale)
    height = int(frame.shape[0] * scale)
    dimensions = (width, height)
    # Resize the frame using the calculated dimensions and specify the interpolation method as 'cv.INTER_AREA'.
    resized_frame = cv.resize(frame, dimensions, interpolation=cv.INTER_AREA)
    # Return the resized frame.
    return resized_frame
# Create a VideoCapture object to capture video frames from 'video.mp4'.
capture = cv.VideoCapture('video.mp4')
# Create an infinite loop to continuously read and display video frames.
while True:
    # Read the next frame from the video capture.
    # 'isTrue' will be True if a frame was successfully read, and 'frame' will contain the frame data.
    isTrue, frame = capture.read()
    # Resize the frame using the 'rescaleFrame' function.
    frame_resized = rescaleFrame(frame)
    # Display the original frame in a window titled 'video1'.
    cv.imshow('video1', frame)
    # Display the resized frame in a window titled 'video resized'.
    cv.imshow('video resized', frame_resized)
    # Wait for a key press event for 20 milliseconds and check if the pressed key is 'd'.
    # If the 'd' key is pressed, break out of the loop and exit.
    if cv.waitKey(20) & 0xFF == ord('d'):
        break

# Release the video capture object to free up resources.
capture.release()
# Close all OpenCV windows.
cv.destroyAllWindows()
```

Step 2 : Run the file in the Visualstudio or PyCharm Community or open terminal then find the path for the python file then write the code below

```
Command Prompt

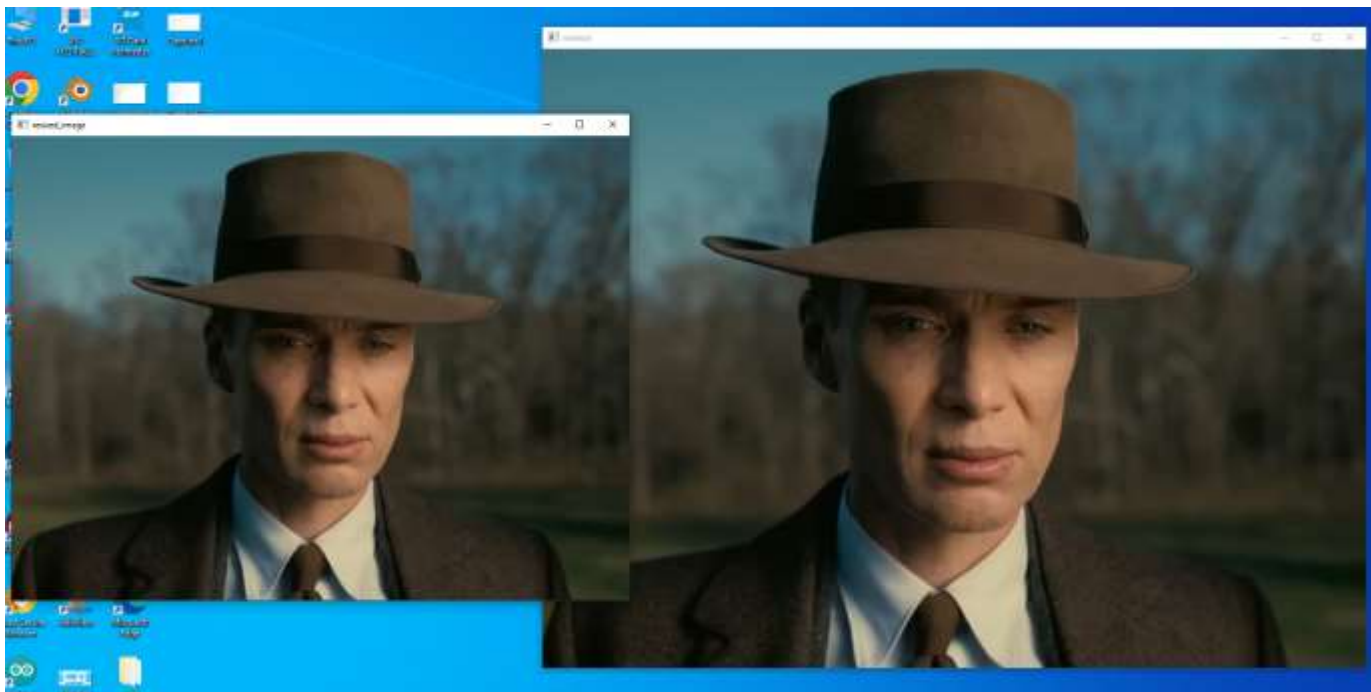
Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

C:\Users\USER>cd Desktop

C:\Users\USER\Desktop>cd Solution1

C:\Users\USER\Desktop\Solution1>python video.py
```

Result :



Exercise four :Rotation

Step 1 : write the code below

```
# Import the OpenCV library and alias it as 'cv' for easier use.
import cv2 as cv
import numpy as np

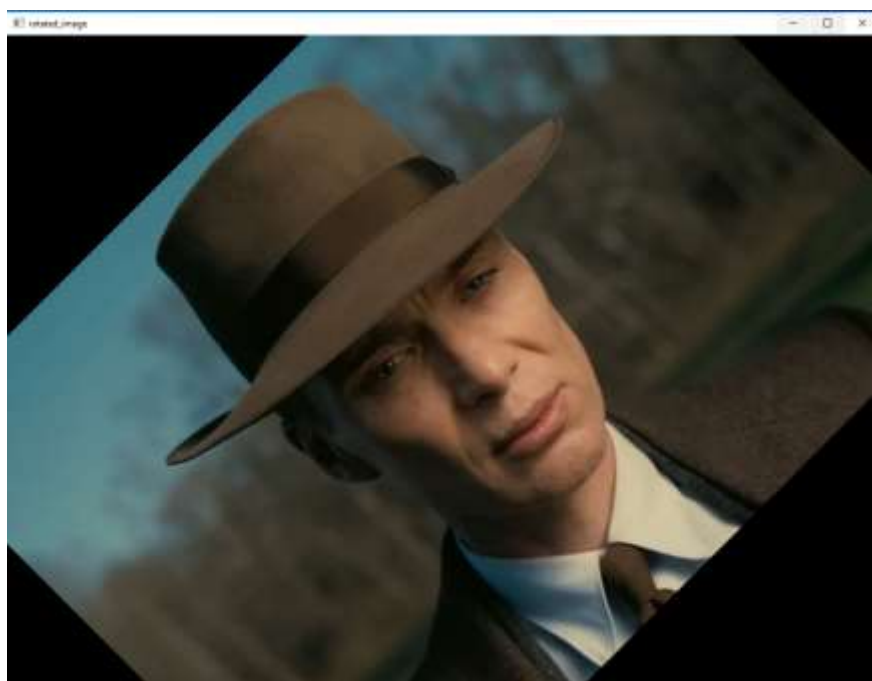
# Load an image from the file '12.jpg'.
img = cv.imread('12.jpg')
# Display the original image in a window titled 'normal'.
cv.imshow('normal', img)

# Define a function called 'rotate' that takes an 'img' (image), 'angle' (rotation angle in degrees),
# and an optional 'rotPoint' parameter (the point around which the image will be rotated, defaults to the center).
def rotate(img, angle, rotPoint=None):
    # Get the height and width of the image.
    (height, width) = img.shape[:2]
    # If 'rotPoint' is not specified, use the center of the image as the rotation point.
    if rotPoint is None:
        rotPoint = (width // 2, height // 2)
    # Create a rotation matrix ('rotMat') using 'cv.getRotationMatrix2D'.
    rotMat = cv.getRotationMatrix2D(rotPoint, angle, 1.0)
    # Define the dimensions for the output (rotated) image.
    dimensions = (width, height)
    # Apply the rotation transformation to the image using 'cv.warpAffine'.
    rotated_img = cv.warpAffine(img, rotMat, dimensions)
    # Return the rotated image.
    return rotated_img

# Call the 'rotate' function to rotate the 'img' by 45 degrees.
rotated = rotate(img, 45)
# Display the rotated image in a window titled 'rotated_image'.
cv.imshow('rotated_image', rotated)
# Wait indefinitely for a key press event. This keeps the image window open until a key is pressed.
cv.waitKey(0)
# After a key is pressed, the window will close, and the program will terminate.
```

Step 2: run the code

Result :



Exercise five : Cropping

Step 1 : write the code below

```
# Import the OpenCV library and alias it as 'cv' for easier use.
import cv2 as cv
import numpy as np
# Load an image from the file '12.jpg'.
img = cv.imread('12.jpg')
# Display the original image in a window titled 'normal'.
cv.imshow('normal', img)
# Define a region of interest (ROI) by cropping a portion of the original image.
# Here, we're selecting a rectangular region from (200,300) to (400,400).
# This creates a smaller image called 'cropped'.
cropped = img[200:400, 300:400]
# Display the cropped region in a window titled 'cropped'.
cv.imshow('cropped', cropped)
# Wait indefinitely for a key press event. This keeps the image window open until a key is pressed.
cv.waitKey(0)
# After a key is pressed, the window will close, and the program will terminate.
```

Step 2 : Run the code in the terminal

result



Haar cascades

Introduction to Haar cascades and object detection

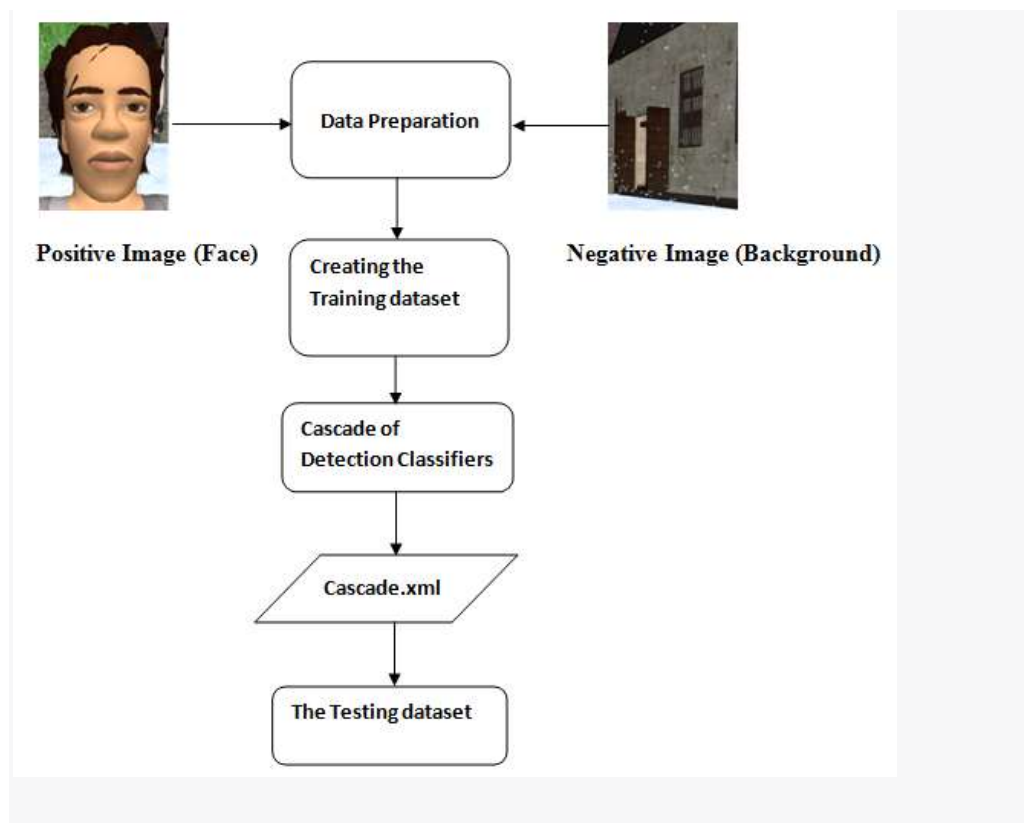
Haar cascades and object detection are important concepts in computer vision and image processing. They are used to identify and locate objects within images or video streams. Here's an introduction to these concepts:

1. Haar Cascades:

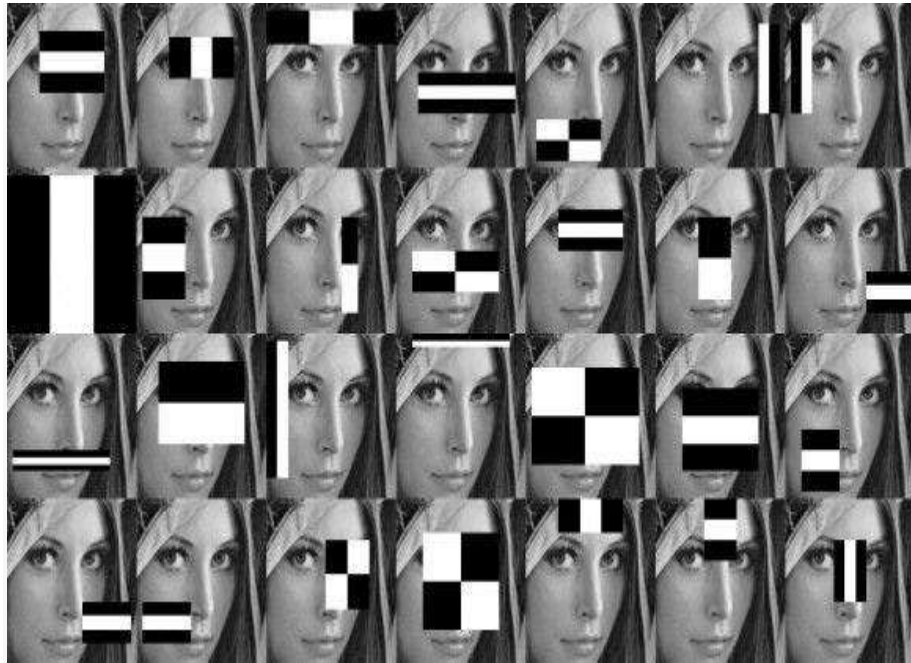
Haar cascades are a machine learning object detection method used to identify objects or patterns in images or video. They are particularly popular for their speed and efficiency in real-time applications. The term "Haar" comes from Haar wavelets, which are the basis of the feature detection technique used in this method.

How Haar Cascades Work:

- **Training:** Haar cascades are trained using positive and negative examples of the object or pattern you want to detect. Positive examples are images containing the target object, and negative examples are images without the object.



- Features: Haar cascades work by using a set of simple rectangular features that can be rapidly calculated. These features are used to distinguish between the object of interest and the background. For example, a feature might be the difference in brightness between a region of the image and an adjacent region.



- Cascade Structure: Haar cascades consist of multiple stages, where each stage contains a set of weak classifiers. Weak classifiers are simple decision rules that evaluate the features. The cascade structure allows for rapid rejection of regions of the image that are unlikely to contain the object, thus improving efficiency.

- Adaptive Learning: Haar cascades adaptively learn the most important features during training, allowing them to focus on the aspects of the object that are most discriminative.

2. Object Detection:

Object detection is a broader concept that encompasses various techniques and algorithms used to locate and identify objects within images or video. Haar cascades are just one method within the field of object detection.



How Object Detection Works:

- **Feature Extraction:** Object detection methods typically involve extracting meaningful features from the image, which can be used to identify the object. These features could include edges, textures, shapes, or more complex patterns.
- **Classifier:** A classifier is used to determine whether the extracted features represent the object of interest or not. This classifier can be a machine learning model like Haar cascades, but it can also be other algorithms like deep neural networks (e.g., Convolutional Neural Networks or CNNs).
- **Bounding Box:** Once an object is detected, object detection methods typically draw a bounding box around the object's location in the image to indicate its position.

Exercise seven: Implementing face and eye detection using Haar cascades

Step 1 : Training

1. Collecting a Dataset:

- Gather a dataset of images containing positive examples (faces and eyes) and negative examples (images without faces or eyes).
- Ensure that the positive examples are accurately annotated with bounding boxes around the faces and eyes.

2. Preparing the Dataset:

- Convert the positive and negative images to grayscale, as Haar cascades typically work with grayscale images.
- Create two text files, one listing the paths to positive images and their corresponding bounding box coordinates and another listing the paths to negative images.

this is the general step for trine the data and creating model but for learning purpose, we will use pretrained model, For our example which is eye and face we need to use two pretrained it can be found in this link below :

https://drive.google.com/drive/folders/1JinMFjFqUx5DBlrs7j8IEbD6tULupVNd?usp=s_haring

Step 2: need to put the video that we want to test, the pretrained model and python file in the same folder

```
Command Prompt
Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

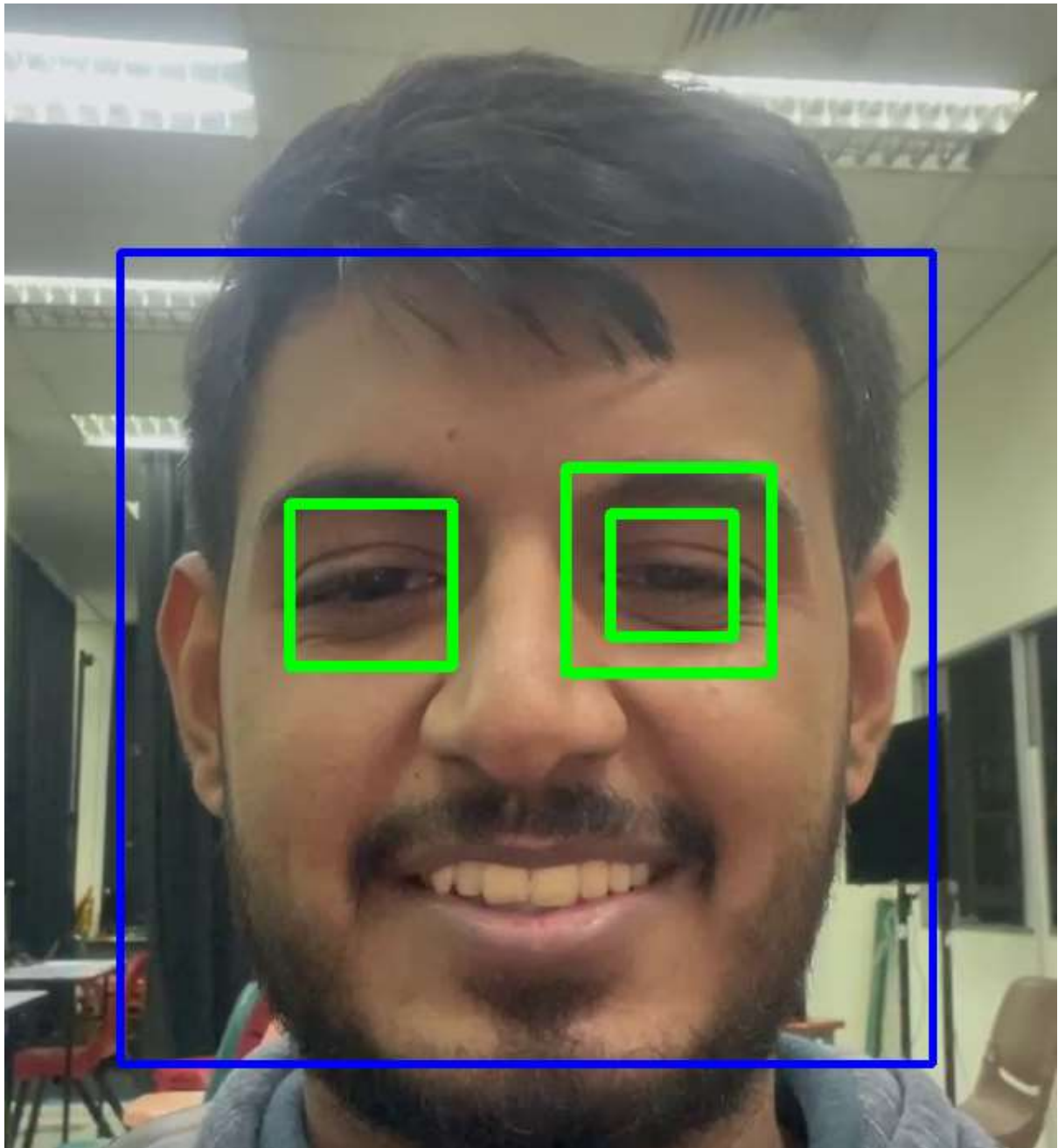
C:\Users\USER>cd Desktop\Openvc
C:\Users\USER\Desktop\Openvc>python face_eye_detection
```

Step 3: Write the code for face and eye detection

```
# Import the OpenCV library.
import cv2
# Create CascadeClassifiers for face and eye detection using pre-trained XML files.
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye_tree_eyeglasses.xml')
# Open a video file for processing. Replace 'test1.mov' with the path to your video file.
cap = cv2.VideoCapture('test1.mov')
while cap.isOpened():
    # Read a frame from the video stream.
    _, img = cap.read()
    # Convert the frame to grayscale for face detection.
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale frame using the face_cascade.
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4)
    # Iterate over the detected faces.
    for (x, y, w, h) in faces:
        # Draw a rectangle around the detected face.
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 3)
        # Extract the region of interest (ROI) within the detected face for eye detection.
        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]
        # Detect eyes within the ROI using the eye_cascade.
        eyes = eye_cascade.detectMultiScale(roi_gray)
        # Iterate over the detected eyes.
        for (ex, ey, ew, eh) in eyes:
            # Draw a rectangle around the detected eye.
            cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 5)
    # Display the frame with detected faces and eyes.
    cv2.imshow('img', img)
    # Check for the 'q' key press to exit the loop and close the video.
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object.
cap.release()
# Close all OpenCV windows.
cv2.destroyAllWindows()
```

Step 4:Run the file in the Visaulstudio or PyCharm Community or using terminal result :



Exercise eight: Building custom object detectors for specific use cases (smile detector)

Step 1: Training

1. Collecting a Dataset:

- Gather a dataset of images containing positive examples (smiling) and negative examples (images without smiling).
- Ensure that the positive examples are accurately annotated with bounding boxes around the smile.

2. Preparing the Dataset:

- Convert the positive and negative images to grayscale, as Haar cascades typically work with grayscale images.
- Create two text files, one listing the paths to positive images and their corresponding bounding box coordinates and another listing the paths to negative images.

this is the general step for trine the data and create model but for learning purpose, we will use a pre-trained model,for our example which is smile detector we need to use two pre-trained mode it will found in this link below :

face detectors pretrained model:

https://drive.google.com/file/d/17WdZDZ1L-ehVr7vJbQ0MVomkNI5oxOcn/view?usp=drive_link

smile detectors pretrained model:

https://drive.google.com/file/d/1vJF_xAvgnbxMrO4uWdIsWBRv0npt-mVW/view?usp=drive_link

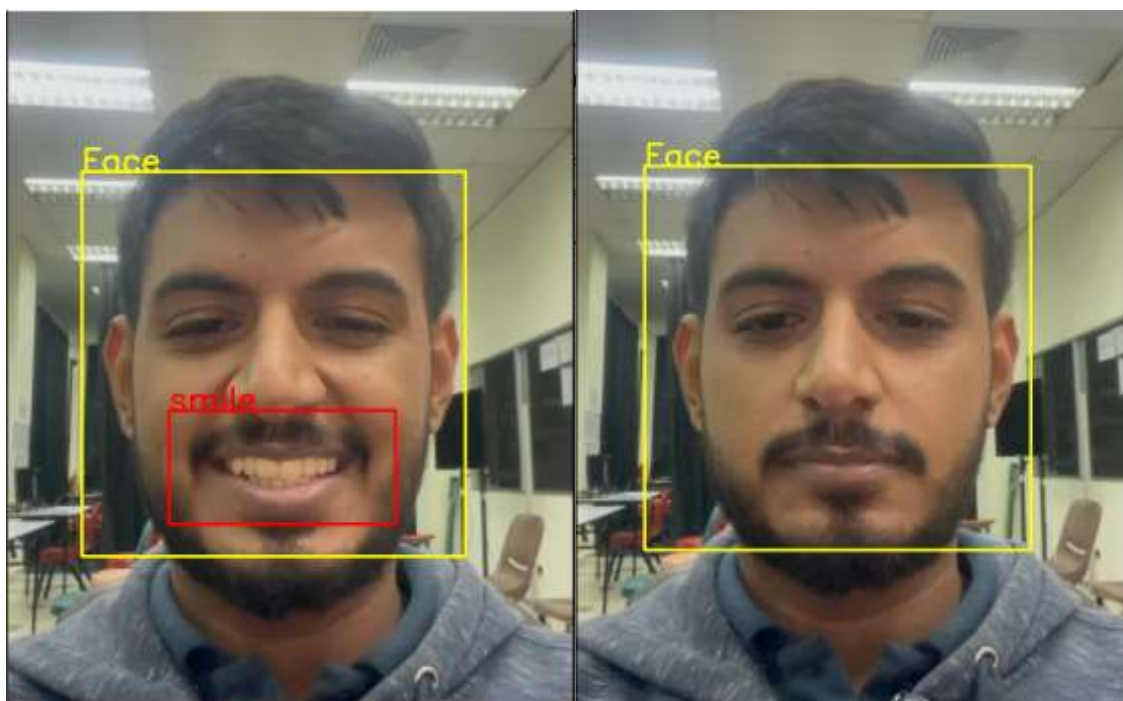
Step 2: Write the code for smile detection

the code can be found in the link below

https://drive.google.com/file/d/1UarTT19AYXggOGK7HAgi_-33R9jDnN3t/view?usp=drive_link

Step 3:Run the file in the Visaulstudio or PyCharm Community or using the terminal

result :



Exercise nine: Human body Pose Tracking

step 1: Install all packages required

open command prompt

- cvzone

```
pip install cvzone
```

- mediapipe

```
pip install mediapipe
```

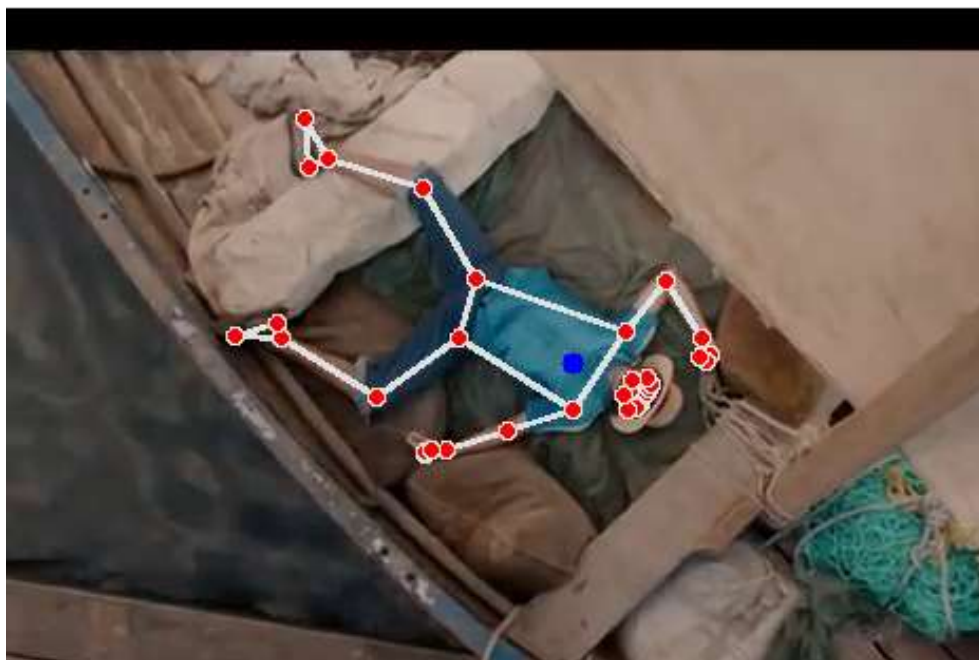
step 2 : Write the code to detect the body

the code can be found in the link below

https://drive.google.com/file/d/16qLPEoLnmQ7o1A7thpN2a0CsgCPXEdAO/view?usp=drive_link

Step 3: Run the file in the Visualstudio or PyCharm Community or using the terminal

result :



Exercise ten: Image annotation on video (logo)

Step 1: Write the code to put the logo in the video

Need to know the video size to know where to place the logo, The code in the link below

[Exercise ten.py](#)

Step 2: Run the file in the VisualStudio or PyCharm Community or using the terminal

result :

link to the video with out the logo :

https://drive.google.com/file/d/1bZanmwVzAsh_IRmdmPTLDs6tBaGMO2ta/view?usp=sharing

link to the video with the logo :

https://drive.google.com/file/d/1oy1kLCku_tl6agAY_qudFSgMUu1nrR64/view?usp=drive_link



codes

python file link :

<https://drive.google.com/drive/folders/1kr9zhATFhevDb1ZrHe3MNRb07x4ZQ5iV?usp=sharing>

Exercise one : photo reading

```
# Import the OpenCV library and alias it as 'cv' for easier
use.
import cv2 as cv

# Load an image from the file 'image.jpg' located in the
'photos' directory.
# The 'imread' function reads an image and stores it in the
'img' variable.
img = cv.imread('photos/image.jpg')

# Display the loaded image in a window with the title 'photo'.
# The 'imshow' function is used for this purpose.
cv.imshow('photo', img)

# Wait indefinitely for a key press event. This keeps the
image window open
# until a key is pressed. The argument '0' means that it will
wait forever.
cv.waitKey(0)

# After a key is pressed, the window will close, and the
program will terminate.
```

Exercise two ; video reading

```
# Import the OpenCV library and alias it as 'cv' for easier
use.
import cv2 as cv

# Create a VideoCapture object, which is used to capture video
frames from a file.
# In this case, it opens and reads frames from 'video.mp4'.
capture = cv.VideoCapture('video.mp4')
```



```

# Create an infinite loop to continuously read and display
video frames.
while True:
    # Read the next frame from the video capture.
    # 'isTrue' will be True if a frame was successfully read,
    and 'frame' will contain the frame data.
    isTrue, frame = capture.read()

    # Display the current frame in a window titled 'video1'.
    cv.imshow('video1', frame)

    # Wait for a key press event for 20 milliseconds and check
    if the pressed key is 'd'.
    # If the 'd' key is pressed, break out of the loop and
    exit.
    if cv.waitKey(20) & 0xFF == ord('d'):
        break

# Release the video capture object to free up resources.
capture.release()

# Close all OpenCV windows.
cv.destroyAllWindows()

```

Exercise three : Image Resizing

```

# Import the OpenCV library and alias it as 'cv' for easier
use.
import cv2 as cv

# Load an image from the file '12.jpg'.
img = cv.imread('12.jpg')

# Display the original image in a window titled 'normal'.
cv.imshow('normal', img)

# Define a function called 'rescaleFrame' that takes a 'frame'
and an optional 'scale' parameter.
def rescaleFrame(frame, scale=0.75):
    # Calculate the new dimensions for the frame based on the
    specified 'scale'.
    width = int(frame.shape[1] * scale)
    height = int(frame.shape[0] * scale)

```

```

        dimensions = (width, height)

        # Resize the frame using the calculated dimensions and
        specify the interpolation method as 'cv.INTER_AREA'.
        resized_frame = cv.resize(frame, dimensions,
interpolation=cv.INTER_AREA)

        # Return the resized frame.
        return resized_frame

# Call the 'rescaleFrame' function to resize the 'img' using
the default scale of 0.75.
resized_image = rescaleFrame(img)

# Display the resized image in a window titled
'resized_image'.
cv.imshow('resized_image', resized_image)

# Wait indefinitely for a key press event. This keeps the
image window open until a key is pressed.
cv.waitKey(0)

# After a key is pressed, the window will close, and the
program will terminate.

```

Exercise four ; Video Resizing

```

# Import the OpenCV library and alias it as 'cv' for easier
use.
import cv2 as cv

# Define a function called 'rescaleFrame' that takes a 'frame'
and an optional 'scale' parameter.
def rescaleFrame(frame, scale=0.75):
    # Calculate the new dimensions for the frame based on the
specified 'scale'.
    width = int(frame.shape[1] * scale)
    height = int(frame.shape[0] * scale)
    dimensions = (width, height)

    # Resize the frame using the calculated dimensions and
specify the interpolation method as 'cv.INTER_AREA'.
    resized_frame = cv.resize(frame, dimensions,
interpolation=cv.INTER_AREA)

```

```

    # Return the resized frame.
    return resized_frame

# Create a VideoCapture object to capture video frames from
'video.mp4'.
capture = cv.VideoCapture('video.mp4')

# Create an infinite loop to continuously read and display
video frames.
while True:
    # Read the next frame from the video capture.
    # 'isTrue' will be True if a frame was successfully read,
    and 'frame' will contain the frame data.
    isTrue, frame = capture.read()

    # Resize the frame using the 'rescaleFrame' function.
    frame_resized = rescaleFrame(frame)

    # Display the original frame in a window titled 'video1'.
    cv.imshow('video1', frame)

    # Display the resized frame in a window titled 'video
    resized'.
    cv.imshow('video resized', frame_resized)

    # Wait for a key press event for 20 milliseconds and check
    if the pressed key is 'd'.
    # If the 'd' key is pressed, break out of the loop and
    exit.
    if cv.waitKey(20) & 0xFF == ord('d'):
        break

# Release the video capture object to free up resources.
capture.release()

# Close all OpenCV windows.
cv.destroyAllWindows()

```

Exercise five : Image rotation

```
# Import the OpenCV library and alias it as 'cv' for easier
use.
import cv2 as cv
import numpy as np

# Load an image from the file '12.jpg'.
img = cv.imread('12.jpg')

# Display the original image in a window titled 'normal'.
cv.imshow('normal', img)

# Define a function called 'rotate' that takes an 'img'
(image), 'angle' (rotation angle in degrees),
# and an optional 'rotPoint' parameter (the point around which
the image will be rotated, defaults to the center).
def rotate(img, angle, rotPoint=None):
    # Get the height and width of the image.
    (height, width) = img.shape[:2]

    # If 'rotPoint' is not specified, use the center of the
    image as the rotation point.
    if rotPoint is None:
        rotPoint = (width // 2, height // 2)

    # Create a rotation matrix ('rotMat') using
    'cv.getRotationMatrix2D'.
    rotMat = cv.getRotationMatrix2D(rotPoint, angle, 1.0)

    # Define the dimensions for the output (rotated) image.
    dimensions = (width, height)

    # Apply the rotation transformation to the image using
    'cv.warpAffine'.
    rotated_img = cv.warpAffine(img, rotMat, dimensions)

    # Return the rotated image.
    return rotated_img

# Call the 'rotate' function to rotate the 'img' by 45
degrees.
rotated = rotate(img, 45)
```

```
# Display the rotated image in a window titled
'rotated_image'.
cv.imshow('rotated_image', rotated)

# Wait indefinitely for a key press event. This keeps the
image window open until a key is pressed.
cv.waitKey(0)

# After a key is pressed, the window will close, and the
program will terminate.
```

Exercise six : Image cropping

```
# Import the OpenCV library and alias it as 'cv' for easier
use.
import cv2 as cv
import numpy as np

# Load an image from the file '12.jpg'.
img = cv.imread('12.jpg')

# Display the original image in a window titled 'normal'.
cv.imshow('normal', img)
# Define a region of interest (ROI) by cropping a portion of
the original image.
# Here, we're selecting a rectangular region from (200,300) to
(400,400).
# This creates a smaller image called 'cropped'.
cropped = img[200:400, 300:400]
# Display the cropped region in a window titled 'cropped'.
cv.imshow('cropped', cropped)
# Wait indefinitely for a key press event. This keeps the
image window open until a key is pressed.
cv.waitKey(0)
```

Exercise seven : face and eye detection

```
# Import the OpenCV library.
import cv2
# Create CascadeClassifiers for face and eye detection using
pre-trained XML files.
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

```

eye_cascade =
cv2.CascadeClassifier('haarcascade_eye_tree_eyeglasses.xml')
# Open a video file for processing. Replace 'test1.mov' with
the path to your video file.
cap = cv2.VideoCapture('test1.mov')
while cap.isOpened():
    # Read a frame from the video stream.
    _, img = cap.read()
    # Convert the frame to grayscale for face detection.
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale frame using the
face_cascade.
    faces = face_cascade.detectMultiScale(gray,
scaleFactor=1.1, minNeighbors=4)
    # Iterate over the detected faces.
    for (x, y, w, h) in faces:
        # Draw a rectangle around the detected face.
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0,
0), 3)
        # Extract the region of interest (ROI) within the
detected face for eye detection.
        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]
        # Detect eyes within the ROI using the eye_cascade.
        eyes = eye_cascade.detectMultiScale(roi_gray)
        # Iterate over the detected eyes.
        for (ex, ey, ew, eh) in eyes:
            # Draw a rectangle around the detected eye.
            cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey +
eh), (0, 255, 0), 5)

        # Display the frame with detected faces and eyes.
        cv2.imshow('img', img)

        # Check for the 'q' key press to exit the loop and close
the video.
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
# Release the video capture object.
cap.release()

# Close all OpenCV windows.
cv2.destroyAllWindows()

```

Exercise eight : smile detector code

```
# import required libraries
import cv2

# read input image
img = cv2.imread('sm2.png')

# convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# read haar cascade for face detection
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# read haar cascade for smile detection
smile_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')

# Detects faces in the input image
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
print('Number of detected faces:', len(faces))

# loop over all the faces detected
for (x,y,w,h) in faces:

    # draw a rectangle in a face
    cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,255), 2)
    cv2.putText(img, "Face", (x, y), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 255, 255), 2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]

    # detecting smile within the face roi
    smiles = smile_cascade.detectMultiScale(roi_gray, 1.8, 20)
    if len(smiles) > 0:
        print("smile detected")
        for (sx, sy, sw, sh) in smiles:
            cv2.rectangle(roi_color, (sx, sy), ((sx + sw), (sy +
sh)), (0, 0, 255), 2)
            cv2.putText(roi_color, "smile", (sx, sy),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
    else:
        print("smile not detected")
```

```
# Display an image in a window
cv2.imshow('Smile Image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Exercise nine

```
# Import the necessary libraries
import cv2
from cvzone.PoseModule import PoseDetector
# Create a PoseDetector object
detector = PoseDetector()

# Open a video file for reading
cap = cv2.VideoCapture('test.mp4')
# Start an infinite loop to process frames from the video
while True:
    # Read the next frame from the video
    success, img = cap.read()
    # Use the PoseDetector to find the pose in the current
    frame
    img = detector.findPose(img)
    # Use the PoseDetector to find the positions of landmarks
    and bounding box information
    lmList, bboxInfo = detector.findPosition(img,
bboxWithHands=True)
    # Display the resulting frame with the pose and landmarks
    cv2.imshow("result", img)
    # Check if the 'd' key is pressed (for quitting the loop)
    if cv2.waitKey(20) & 0xFF == ord('d'):
        break

# Release the video capture object
cap.release()

# Close all OpenCV windows
cv2.destroyAllWindows()
```


Exercise ten

```
import cv2

# Load the video
video_path = 'test.mp4'
video = cv2.VideoCapture(video_path)

# Load the logo image with transparency (PNG format is
recommended)
logo_path = 'logoh.png'
logo = cv2.imread(logo_path)

rows, cols, channels = logo.shape

# Create a VideoWriter object to save the output video
fourcc = cv2.VideoWriter_fourcc(*'mp4v') # Codec for output
video
output_path = 'output_video_with_logo.mp4'
output_video = cv2.VideoWriter(output_path, fourcc, 30,
(int(video.get(3)), int(video.get(4))))

# Process each frame in the input video
while True:
    ret, frame = video.read()
    if not ret:
        break

    # Extract the region of interest for the logo
    logoArea = frame[0:rows, 0:cols]

    # Create a mask for the logo
    img2gray = cv2.cvtColor(logo, cv2.COLOR_BGR2GRAY)
    ret, mask = cv2.threshold(img2gray, 175, 255,
cv2.THRESH_BINARY)
    mask_inv = cv2.bitwise_not(mask)

    # Apply the mask to the logo and the frame
    foreground_with_mask = cv2.bitwise_and(logo, logo,
mask=mask)
    background_with_mask = cv2.bitwise_and(logoArea, logoArea,
mask=mask_inv)
    final_img = cv2.add(foreground_with_mask,
background_with_mask)
```

```
frame[0:rows, 0:cols] = final_img

# Write the frame to the output video
output_video.write(frame)

cv2.imshow("Logo in Video", frame)

if cv2.waitKey(20) & 0xFF == ord('d'):
    break

# Release the video objects
video.release()
output_video.release()

# Close all OpenCV windows
cv2.destroyAllWindows()

# Display a message when the process is complete
print("Video with logo added saved as", output_path)
```