CS-33: Machine Learning with Python

Unit -5 : Computer Vision with OpenCV

P-1 face and eyes detect

```
#import libraries
import cv2

#classifier for face detection
face_detector =
cv2.CascadeClassifier('haarcascades\haarcascade_front
alface_default.xml')

#classifier for eye detection
eye_detector =
cv2.CascadeClassifier('haarcascades\haarcascade_eye.x
ml')

#read image
img1 = cv2.imread('data\img.jpg')
img = cv2.resize(img1, (1300, 900))
#convert image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#detecting faces
faces = face_detector.detectMultiScale(gray, 1.3, 5)

#drawing bounding boxes on faces
for (x, y, w, h) in faces:
    img = cv2.rectangle(img, (x, y), (x + w, y + h),
(255, 0, 0), 2)
    roi_gray = gray[y:y + h, x:x + w]
    roi_color = img[y:y + h, x:x + w]
```

```
#detecting eyes for a particular faces
    eyes = eye_detector.detectMultiScale(roi_gray) #
Moved inside the face detection loop
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(roi_color, (ex, ey), (ex + ew,
ey + eh), (0, 255, 0), 2)

print("Found {0} faces!".format(len(faces)))

cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

P-2 car-detection object

```
import time
import numpy as np
import cv2
# Create our body classifier
car_classifier =
cv2.CascadeClassifier('haarcascades\haarcascade_car.x
ml')
# Initiate video capture for video file
cap = cv2.VideoCapture('data\cars.avi')
# Loop once video is successfully loaded
while cap.isOpened():

    time.sleep(.05)
    # Read first frame
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Pass frame to our car classifier
```

```
    cars = car_classifier.detectMultiScale(gray, 1.1,
2)

    # Extract bounding boxes for any bodies
identified
    for (x,y,w,h) in cars:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,
255, 255), 2)
        cv2.imshow('Cars', frame)
    if cv2.waitKey(1) == 13: #13 is the Enter Key
        break
cap.release()
cv2.destroyAllWindows()
```

P-3 Tracking object(track Human)

```
import cv2
import numpy as np
# Load Haar Cascade for tracking
track_cascade =
cv2.CascadeClassifier("haarcascades\haarcascade_fullb
ody.xml")

# Initialize the video capture (use 0 for webcam or a
video file path)
cap = cv2.VideoCapture("data\walking.avi")

# Initialize tracking variables
tracking_window = None
roi_hist = None
term_crit = (cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 1)

while True:
    ret, frame = cap.read()
```

```
    if not ret:
        break

    # Convert to grayscale for tracking
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # track human (you can adjust the scaleFactor,
minNeighbors, etc. for better results)
    tracks = track_cascade.detectMultiScale(gray,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # If no tracking is active, detect human and
initialize tracking
    if tracking_window is None and len(tracks) > 0:
        x, y, w, h = tracks[0]

        # Set the tracking window for MeanShift
        tracking_window = (x, y, w, h)

        # Create a mask for the region of interest
(ROI)
        roi = frame[y:y+h, x:x+w]
        roi_gray = cv2.cvtColor(roi,
cv2.COLOR_BGR2GRAY)
        roi_hist = cv2.calcHist([roi_gray], [0],
None, [256], [0, 256])

        # Normalize the histogram
        cv2.normalize(roi_hist, roi_hist, 0, 255,
cv2.NORM_MINMAX)
    # If a track is being tracked, apply MeanShift
    if tracking_window is not None:
        # Use MeanShift to track the human on the
histogram backprojection
```

```
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv], [0],
roi_hist, [0, 256], 1)

        # Apply MeanShift to find the new location
        ret, tracking_window = cv2.meanShift(dst,
tracking_window, term_crit)

        # Draw the tracked human
        x, y, w, h = tracking_window
        cv2.rectangle(frame, (x, y), (x + w, y + h),
(0, 255, 0), 2)

    # Display the frame
    cv2.imshow('Human Tracking with MeanShift',
frame)

    # Exit on pressing 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close the windows
cap.release()
cv2.destroyAllWindows()
```

P-4 Detect face

```
import cv2
faceCascade =
cv2.CascadeClassifier("haarcascades\haarcascade_front
alface_default.xml")

video_capture = cv2.VideoCapture(r"E:\6-Machine
Learning with Python\u-5(practical)\ch-5\data\2.mp4")
```

```
while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
    )

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,
255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

P-5 Detect human

```
import numpy as np
import cv2
# Create our body classifier
body_classifier =
cv2.CascadeClassifier('haarcascades\haarcascade_fullb
ody.xml')
```

```
# Initiate video capture for video file
cap = cv2.VideoCapture('data\walking.avi')
# Loop once video is successfully loaded
while cap.isOpened():

    # Read first frame
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Pass frame to our body classifier
    bodies = body_classifier.detectMultiScale(gray,
1.1, 3)

    # Extract bounding boxes for any bodies
identified
    for (x,y,w,h) in bodies:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,
255, 255), 2)
        cv2.imshow('image', frame)
    if cv2.waitKey(1) == 13: #13 is the Enter Key
        break
cap.release()
cv2.destroyAllWindows()
```

P-6 single object tracking

```
import cv2
import numpy as np
import sys

# Read video
video = cv2.VideoCapture(r"E:\6-Machine Learning with
Python\u-5(practical)\ch-5\data\ball.gif")
# Exit if video not opened
if not video.isOpened():
```

```
    print("Could not open video")
    sys.exit()

# Read first frame
ok, frame = video.read()
if not ok:
    print('Cannot read video file')
    sys.exit()

# Define an initial bounding box (you can modify this
or select it interactively)
bbox = cv2.selectROI(frame, False)

# Extract the region of interest (ROI)
roi = frame[int(bbox[1]):int(bbox[1] + bbox[3]),
int(bbox[0]):int(bbox[0] + bbox[2])]

# Convert the first frame to grayscale for optical
flow
prev_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Define the initial points for optical flow
prev_pts = np.array([[(bbox[0] + bbox[2] // 2),
(bbox[1] + bbox[3] // 2)]], dtype=np.float32)

# Start the optical flow tracking loop
while True:
    ok, frame = video.read()
    if not ok:
        break
    # Convert the current frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Calculate optical flow
```

```
    next_pts, status, error =
cv2.calcOpticalFlowPyrLK(prev_gray, gray, prev_pts,
None)

    # Get the updated position of the tracked point
    x, y = next_pts[0].ravel()

    # Draw the tracking point
    cv2.circle(frame, (int(x), int(y)), 5, (0, 255,
0), -1)

    # Display the result
    cv2.imshow("Tracking", frame)

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

    # Update the previous frame and points for the
next iteration
    prev_gray = gray.copy()
    prev_pts = next_pts
# Release the video capture object and close the
window
video.release()
cv2.destroyAllWindows()
```

P-7 track image

```
import cv2
import numpy as np
# Load the Haar Cascade classifier for face detection
face_cascade =
cv2.CascadeClassifier('haarcascades\haarcascade_front
alface_default.xml')
```

```
# Read the image
image = cv2.imread(r"E:\6-Machine Learning with
Python\u-5(practical)\ch-5\data\img.jpg")  # Change
to your image path
if image is None:
    print("Error loading image")
    exit()

# Convert the image to grayscale for face detection
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Check if a face is detected
if len(faces) == 0:
    print("No face detected in the image")
else:
    # Let's take the first detected face (you can
loop over multiple faces if needed)
    x, y, w, h = faces[0]
    roi = image[y:y+h, x:x+w]  # Region of interest
(ROI) is the detected face

    # Calculate the histogram of the face region in
HSV color space
    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv_roi, np.array((0., 60.,
32.)), np.array((180., 255., 255.)))
    roi_hist = cv2.calcHist([hsv_roi], [0], mask,
[16], [0, 180])
```

```
    roi_hist = cv2.normalize(roi_hist, roi_hist, 0,
255, cv2.NORM_MINMAX)

    # Now, simulate tracking by backprojecting the
histogram onto the whole image
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    dst = cv2.calcBackProject([hsv], [0], roi_hist,
[0, 180], 1)

    # Apply the Meanshift algorithm to "track" the
face within the image
    ret, track_window = cv2.meanShift(dst, (x, y, w,
h), (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,
10, 1))

    # Draw a rectangle around the "tracked" face in
the image
    x, y, w, h = track_window
    tracked_image = image.copy()
    cv2.rectangle(tracked_image, (x, y), (x + w, y +
h), (0, 255, 0), 2)

    # Show the resulting image with the tracked face
    cv2.imshow("Tracked Image", tracked_image)
    cv2.waitKey(0)

# Close the window after a key press
cv2.destroyAllWindows()
```

P-8 Nose detect

```
import cv2
# Load the input image
img = cv2.imread(r"data\img.JPG")
```

```
img = cv2.resize(img,(1400,800))
# Check if the image is loaded correctly
if img is None:
    print("Error: Image not found or unable to
load.")
    exit()

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

# Load the Haar cascade for face detection
face_cascade =
cv2.CascadeClassifier("haarcascades\haarcascade_front
alface_default.xml")  # Load face cascade

# Check if the cascade is loaded
if face_cascade.empty():
    print("Error: Face cascade not loaded.")
    exit()

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray,
scaleFactor=1.1, minNeighbors=7, minSize=(30, 30))

# Load the Haar cascade for nose detection
nose_cascade =
cv2.CascadeClassifier("haarcascades\haarcascade_mcs_n
ose.xml")  # Load nose cascade

# Check if the nose cascade is loaded
if nose_cascade.empty():
    print("Error: nose cascade not loaded.")
    exit()
```

```
# Loop through each face detected
for (x, y, w, h) in faces:
    # Draw a rectangle around the face
    cv2.rectangle(img, (x, y), (x + w, y + h), (0,
255, 0), 2)

    # Region of interest (ROI) for nose detection
(within the face area)
    roi_gray = gray[y + int(h / 2):y + h, x:x + w]  #
Focus on the lower half of the face
    roi_color = img[y + int(h / 2):y + h, x:x + w]

    # Detect the nose in the face region
    noses = nose_cascade.detectMultiScale(roi_gray,
scaleFactor=1.3, minNeighbors=2)

    # Loop through each detected nose and draw a
rectangle around it
    for (mx, my, mw, mh) in noses:
        # Draw a rectangle around the nose
        cv2.rectangle(roi_color, (mx, my), (mx + mw,
my + mh), (0, 0, 255), 2)


# Display the output image
cv2.imshow("Output", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

P-9 Mouth detection

```
import cv2
# Load the input image
img = cv2.imread("data\img.JPG")
img = cv2.resize(img,(1400,800))
```

```
# Check if the image is loaded correctly
if img is None:
    print("Error: Image not found or unable to
load.")
    exit()

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

# Load the Haar cascade for face detection
face_cascade =
cv2.CascadeClassifier("haarcascades\haarcascade_front
alface_default.xml")  # Load face cascade

# Check if the cascade is loaded
if face_cascade.empty():
    print("Error: Face cascade not loaded.")
    exit()

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray,
scaleFactor=1.1, minNeighbors=7, minSize=(30, 30))

# Load the Haar cascade for mouth detection
mouth_cascade =
cv2.CascadeClassifier("haarcascades\haarcascade_mouth
.xml")  # Load mouth cascade

# Check if the mouth cascade is loaded
if mouth_cascade.empty():
    print("Error: Mouth cascade not loaded.")
    exit()

# Loop through each face detected
```

```
for (x, y, w, h) in faces:
    # Draw a rectangle around the face
    cv2.rectangle(img, (x, y), (x + w, y + h), (0,
255, 0), 2)

    # Region of interest (ROI) for mouth detection
(within the face area)
    roi_gray = gray[y + int(h / 2):y + h, x:x + w]  #
Focus on the lower half of the face
    roi_color = img[y + int(h / 2):y + h, x:x + w]

    # Detect the mouth in the face region
    mouths = mouth_cascade.detectMultiScale(roi_gray,
scaleFactor=1.3, minNeighbors=2)

    # Loop through each detected mouth and draw a
rectangle around it
    for (mx, my, mw, mh) in mouths:
        # Draw a rectangle around the mouth
        cv2.rectangle(roi_color, (mx, my), (mx + mw,
my + mh), (0, 0, 255), 2)

# Display the output image
cv2.imshow("Output", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

P-10 pupils detection

```
import cv2
import numpy as np

# Load the image
image = cv2.imread(r"E:\6-Machine Learning with
Python\u-5(practical)\ch-5\data\pupils\2.jpg")
```

```
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply GaussianBlur to reduce noise
blurred = cv2.GaussianBlur(gray, (15, 15), 0)

# Detect circles using Hough Circle Transform
# dp: Inverse ratio of the accumulator resolution to
the image resolution.
# minDist: Minimum distance between the centers of
the detected circles.
# param1: High threshold for the Canny edge detector.
# param2: Threshold for center detection.
# minRadius & maxRadius: Minimum and maximum radius
of circles to be detected.

circles = cv2.HoughCircles(blurred,
cv2.HOUGH_GRADIENT, dp=1.2, minDist=20, param1=50,
param2=30, minRadius=10, maxRadius=30)

# If some circles are detected
if circles is not None:
    # Convert coordinates to integer
    circles = np.round(circles[0, :]).astype("int")

    # Loop through the detected circles
    for (x, y, r) in circles:
        # Draw the circle on the image
        cv2.circle(image, (x, y), r, (0, 255, 0), 4)
 # Circle for the pupil

        # Draw the center of the circle
```

```
        cv2.circle(image, (x, y), 2, (0, 0, 255), 3)
 # Center of the pupil

# Display the result
cv2.imshow("Pupil Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

P-11 pupil detection from video

```
import numpy as np
import cv2

face_cascade =
cv2.CascadeClassifier('haarcascades\haarcascade_front
alface_default.xml')
eye_cascade =
cv2.CascadeClassifier('haarcascades\haarcascade_lefte
ye_2splits.xml')

#number signifies camera
cap = cv2.VideoCapture(r"E:\6-Machine Learning with
Python\u-5(practical)\ch-5\data\2.mp4")

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3,
5)
    eyes = eye_cascade.detectMultiScale(gray)
    for (ex,ey,ew,eh) in eyes:

cv2.rectangle(img,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
        roi_gray2 = gray[ey:ey+eh, ex:ex+ew]
```

```
        roi_color2 = img[ey:ey+eh, ex:ex+ew]
        circles =
cv2.HoughCircles(roi_gray2,cv2.HOUGH_GRADIENT,1,200,p
aram1=200,param2=1,minRadius=0,maxRadius=0)
        try:
            for i in circles[0,:]:
                # draw the outer circle

cv2.circle(roi_color2,(i[0],i[1]),i[2],(255,255,255),
2)
                print("drawing circle")
                # draw the center of the circle

cv2.circle(roi_color2,(i[0],i[1]),2,(255,255,255),3)
        except Exception as e:
            print(e)
    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```