Computer Vision

LAB 6 & LAB 7

Submitted By:

Gaurav Agarwal	BT17CSE001
Abhibha Gupta	BT17CSE020
Thogaru Himabindu	BT17CSE056

Submitted to:

Dr. Mayur Parate

Task:

DEVELOP FEATURE ALGORITHM (HOG)

&

ADD FEATURE INFORMATION ALONG WITH ITS MOTION

TO TRACK AN OBJECT IN VIDEO SEQUENCE

भारतीय सूचना प्रौद्योगिकी संस्थान, नागपुर

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, NAGPUR

(An Institution of National Importance by Act of Parliament)

BSNL RTTC, Near TV Tower, Besides Balaji Temple, Seminary Hills, Nagpur-440006

LAB 6

DEVELOP FEATURE ALGORITHM (HOG)

```
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
class Hog_descriptor():
   def __init__(self, img, cell_size=16, bin_size=8):
       self.img = img
       self.img = np.sqrt(img / float(np.max(img)))
       self.img = self.img * 255
       self.cell_size = cell_size
       self.bin_size = bin_size
       self.angle_unit = 360 / self.bin_size

   def extract(self):
       height, width = self.img.shape
       gradient_magnitude, gradient_angle = self.global_gradient()
       gradient_magnitude = abs(gradient_magnitude)
       cell_gradient_vector = np.zeros((int(height / self.cell_size),
int(width / self.cell_size), self.bin_size))
       for i in range(cell_gradient_vector.shape[0]):
           for j in range(cell_gradient_vector.shape[1]):
               cell_magnitude = gradient_magnitude[i *
                                self.cell_size:(i + 1) * self.cell_size,
                                j * self.cell_size:(j + 1) *
                                 self.cell_size]
               cell_angle = gradient_angle[i * self.cell_size:(i + 1) *
                                         self.cell_size,
                              j * self.cell_size:(j + 1) * self.cell_size]
               cell_gradient_vector[i][j] =
                             self.cell_gradient(cell_magnitude,cell_angle)

       hog_image = self.render_gradient(np.zeros([height, width]),
                                    cell_gradient_vector)
       hog_vector = []
       for i in range(cell_gradient_vector.shape[0] - 1):
           for j in range(cell_gradient_vector.shape[1] - 1):
```

```python
            block_vector = []
            block_vector.extend(cell_gradient_vector[i][j])
            block_vector.extend(cell_gradient_vector[i][j + 1])
            block_vector.extend(cell_gradient_vector[i + 1][j])
            block_vector.extend(cell_gradient_vector[i + 1][j + 1])
            mag = lambda vector: math.sqrt(sum(i ** 2 for i in vector))
            magnitude = mag(block_vector)
            if magnitude != 0:
                normalize =
                          lambda block_vector,
                            magnitude: [element / magnitude
                             for element in block_vector]
                block_vector = normalize(block_vector, magnitude)
            hog_vector.append(block_vector)
    return hog_vector, hog_image

def global_gradient(self):
    gradient_values_x = cv2.Sobel(self.img, cv2.CV_64F, 1, 0, ksize=5)
    gradient_values_y = cv2.Sobel(self.img, cv2.CV_64F, 0, 1, ksize=5)
    gradient_magnitude = cv2.addWeighted(gradient_values_x, 0.5,
                                         gradient_values_y, 0.5, 0)
    gradient_angle = cv2.phase(gradient_values_x, gradient_values_y,
                                         angleInDegrees=True)
    return gradient_magnitude, gradient_angle

def cell_gradient(self, cell_magnitude, cell_angle):
    orientation_centers = [0] * self.bin_size
    for i in range(cell_magnitude.shape[0]):
        for j in range(cell_magnitude.shape[1]):
            gradient_strength = cell_magnitude[i][j]
            gradient_angle = cell_angle[i][j]
            min_angle, max_angle, mod =
                               self.get_closest_bins(gradient_angle)
            orientation_centers[min_angle] +=
                          (gradient_strength*(1-(mod / self.angle_unit)))
            orientation_centers[max_angle] +=
                            (gradient_strength * (mod / self.angle_unit))
    return orientation_centers
```

```
    def get_closest_bins(self, gradient_angle):
        idx = int(gradient_angle / self.angle_unit)
        mod = gradient_angle % self.angle_unit
        if idx == self.bin_size:
            return idx - 1, (idx) % self.bin_size, mod
        return idx, (idx + 1) % self.bin_size, mod

    def render_gradient(self, image, cell_gradient):
        cell_width = self.cell_size / 2
        max_mag = np.array(cell_gradient).max()
        for x in range(cell_gradient.shape[0]):
            for y in range(cell_gradient.shape[1]):
                cell_grad = cell_gradient[x][y]
                cell_grad /= max_mag
                angle = 0
                angle_gap = self.angle_unit
                for magnitude in cell_grad:
                    angle_radian = math.radians(angle)
                    x1 = int(x * self.cell_size + magnitude * cell_width *
                             math.cos(angle_radian))
                    y1 = int(y * self.cell_size + magnitude * cell_width *
                             math.sin(angle_radian))
                    x2 = int(x * self.cell_size - magnitude * cell_width *
                             math.cos(angle_radian))
                    y2 = int(y * self.cell_size - magnitude * cell_width *
                             math.sin(angle_radian))
                    cv2.line(image, (y1, x1), (y2, x2), int(255 *
                             math.sqrt(magnitude)))
                    angle += angle_gap
        return image

img = cv2.imread('car.png', cv2.IMREAD_GRAYSCALE)
hog = Hog_descriptor(img, cell_size=16, bin_size=16)
vector, image = hog.extract()
from google.colab.patches import cv2_imshow
cv2_imshow(image)
```

In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions.

Calculate Histogram of gradients in 8 * 8

In this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells.

The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram.In our representation, the y-axis is 0 degrees.

Visualizing Histogram of Oriented Gradients

The HOG descriptor of an image patch is usually visualized by plotting the 9×1 normalized histograms in the 8×8 cells.

See Below Image

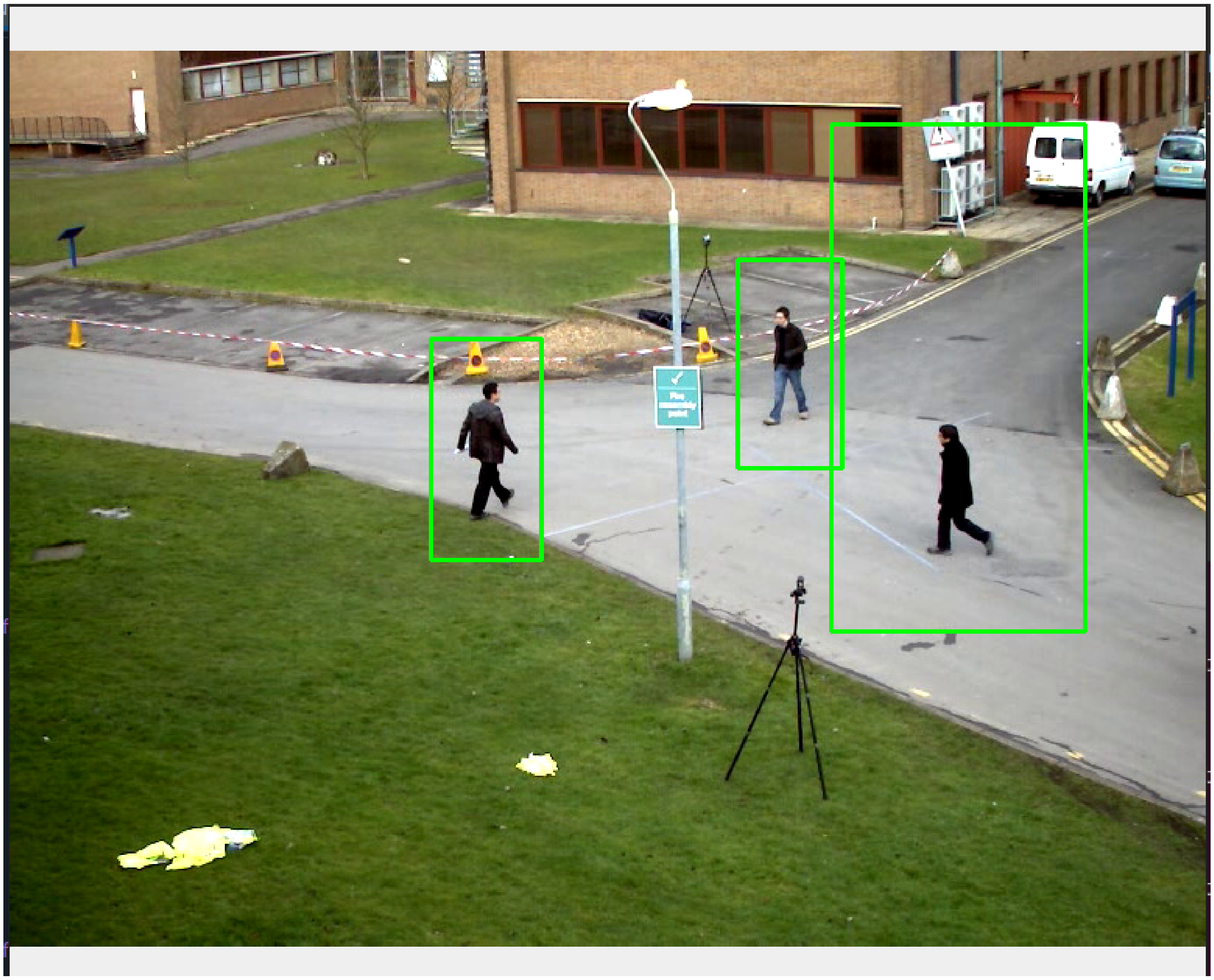
LAB 7

ADD FEATURE INFORMATION ALONG WITH ITS MOTION TO TRACK AN OBJECT IN VIDEO SEQUENCE

```
import cv2
import imutils
import argparse

def detect(frame):
    bounding_box_cordinates, weights =  HOGCV.detectMultiScale(frame, 
winStride = (4, 4), padding = (8, 8), scale = 1.03)

    for x,y,w,h in bounding_box_cordinates:
        cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)
    cv2.imshow('output', frame)

    return frame

def detectByPathVideo(path, writer):

    video = cv2.VideoCapture(path)
    check, frame = video.read()
    if check == False:
        return

    while video.isOpened():
        check, frame =  video.read()

        if check:
            frame = imutils.resize(frame , width=min(800,frame.shape[1]))
            frame = detect(frame)

            if writer is not None:
                writer.write(frame)

            key = cv2.waitKey(1)
            if key== ord('q'):
                break
        else:
            break
    video.release()
    cv2.destroyAllWindows()
```

```python
def humanDetector(args):
    video_path = args['video']

    writer = None
    if args['output'] is not None:
        writer = 
cv2.VideoWriter(args['output'],cv2.VideoWriter_fourcc(*'MJPG'), 10, 
(600,600))


    if video_path is not None:
        detectByPathVideo(video_path, writer)


def argsParser():
    arg_parse = argparse.ArgumentParser()
    arg_parse.add_argument("-v", "--video")
    arg_parse.add_argument("-o", "--output")
    args = vars(arg_parse.parse_args())
    return args

if __name__ == "__main__":
    HOGCV = cv2.HOGDescriptor()
    HOGCV.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
    args = argsParser()
    humanDetector(args)
```

OUTPUT:
BELOW MENTIONED IS ONE FRAME FROM A VIDEO SEQUENCE:



Histogram of Oriented Gradient Descriptor

HOG is a feature descriptor used in computer vision and image processing for the purpose of object detection. This is one of the most popular techniques for object detection, OpenCV has already been implemented in an efficient way to combine the HOG Descriptor algorithm with SVM.

cv2.HOGDescriptor_getDefaultPeopleDetector() calls the pre-trained model for Human detection of OpenCV and then we will feed our support vector machine with it.

Detect() method will take a frame to detect a person in it. Make a box around a person and show the frame..and return the frame with person bounded by a green box.

Everything will be done by detectMultiScale(). It returns 2-tuple.

- List containing Coordinates of bounding Box of person.
  Coordinates are in form X, Y, W, H.
  Where x,y are starting coordinates of box and w, h are width and height of box respectively.
- Confidence Value that it is a person.

DetectByPathVideo() method will be given a path to the Video. We check if the video on the provided path is found or not. At each frame we will check that it successfully reads the frame or not. At the end when the frame is not read we will end the loop.

Argparse() function simply parses and returns as a dictionary the arguments passed through the terminal to our script. There will be two arguments within the Parser:

- Video: The path to the input video
- Output: The path to the name the output video

Summary

we have learned how to create a people counter using HOG and OpenCV to generate an efficient people detector in a video