Semester Project- Crowd Size Detector

IT 101 - Cornerstone: IT

Drew Wilson

Date 11/24/2020

# References

* <https://circuitdigest.com/microcontroller-projects/crowd-size-estimation-using-opencv-and-raspberry-pi>
* <https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/>
* <https://docs.opencv.org/master/d5/d33/structcv_1_1HOGDescriptor.html>
* <https://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/>
* <https://projects.raspberrypi.org/en/projects/getting-started-with-picamera>

# Commented Python Code

#imports the libraries that we will use

import cv2 #openCV- object recognition software

import imutils #image utilities

from imutils.object\_detection import non\_max\_suppression #the function that merges overlapping boxes that identify the same object.

import numpy as np

import requests

import time

import base64

from matplotlib import pyplot as plt #used to create the image window

from urllib.request import urlopen #opens a URL

channel\_id = 1160920 ## PUT CHANNEL ID HERE

WRITE\_API = 'BIATVNBSES1RKSDF' ## PUT YOUR WRITE KEY HERE

#this line concatenates the start of the url we use to update Thingspeak

#it uses the .format method to insert our API writing permission key in the {}

BASE\_URL = "https://api.thingspeak.com/update?api\_key={}".format(WRITE\_API)

#sets variable named hog to a pretrained model of OpenCV for people detection

hog = cv2.HOGDescriptor() #HOG- histogram oriented object descriptor

#inside of this call returns the classifier trained for people detection

#outside calls the Support Vector Machine detector. it uses a linear model to classify objects

hog.setSVMDetector(cv2.HOGDescriptor\_getDefaultPeopleDetector())

prevResult = [0] #keeps track of the result before the most recent one, to be used to help decide when to update the site later

## In[3]:

def detector(image):#what we called frame in record is now called image in detector

clone = image.copy() #copies image again. why?

#puts same data into rects and weights variable- weights is not used though?

#for detectMultiScale- first parameter is image, second is a tuple that determines the size of the sliding window

#the sliding window goes over a portion of the image, sends it to the linear SVM, then moves it's width minus padding to the right and repeats until finished with the entire image. the SVM communicates whether or not the contents of that box are a person's body

#scale controls resizing image. has implications for smoothing details. larger than 1 means that it enlarges the picture, making it run faster but less accurately

#changing scale is why we need to use non\_max\_suppression later

#in short, creates array of the x/y locations of parts of the image it found bodies in

rects, weights = hog.detectMultiScale(image, winStride=(4, 4), padding=(8, 8), scale=1.05)

#rects is an array / list of data. this for loop just loops through each set of data stored in rects

for (x, y, w, h) in rects:

#places a rectangle around the image where ever the detectMultiScale gave it coordinates for- i.e. where it found a body

#first arg is image, then tuples for the opposite corners, a tuple for the color of the rectangle, and then thickness as an int

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

#shorthand form of a for loop that makes an array storing the locations of all of the rectangles placed in the image

rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])

#this line deals with overlapping boxes- given the size of the sliding window, it is possible for the body to be placed in several rectangles, only one of which is necessary

#passes locations that it found the rectangles. if they overlap by more than 70%, they are counted as only one box

result = non\_max\_suppression(rects, probs=None, overlapThresh=0.7)

return result

#sample\_time tells the program how often to upload the number of people to ThingSpeak in seconds

def record(sample\_time=15):

print("camera initialized")

#camera is the object that contains the commands for the physical camera

#cv2.VideoCapture tells us that we will be taking information from the camera as it is passed through openCV

camera = cv2.VideoCapture(0) #0 is the id of the first camrea hooked up to a pi

init = time.time() #captures current time into init variable

## ubidots sample limit

if sample\_time < 3:

sample\_time = 1

#infinite while loop. will only stop if the program ends by a keyboard interrupt or closing the terminal

while(True):

print("cap frames")

#sets both variables ret and frame to the image that is currently in the camera's view

ret, frame = camera.read()

#frame becomes a resized version of itself

#template parameters- .resize(image to resize, width=new width of frame)

#the min(400, frame.shape[1]) tells the program to use the smaller value- either 400 or the current width

frame = imutils.resize(frame, width=min(400, frame.shape[1]))

#.copy assigns the value being passed to what frame currently is, but will not change when frame updates, like it would with the assignment operator =

result = detector(frame.copy())#result becomes the return value of the detector function, with the argument frame passed

#result1 is length of the result returned by the detector function

result1 = len(result) #result contains x/y values of locations of faces. the number of entries in the list (the length) shows the number of people found

print (result1)#prints how many people were found

#this does the same thing as the matching for loop in the detector function, but this allows the information to be used within the scope of this function, which is necessary for the rectangles on the image to be displayed.

for (xA, yA, xB, yB) in result:

cv2.rectangle(frame, (xA, yA), (xB, yB), (0, 255, 0), 2)

#next two lines show a popup of the image captured on the screen

#i comment them out because I don't need to see the image on my screen, so the program can just keep looping

#shows image

plt.imshow(frame)

#stores either True or False internally.

#if True, the window is showing and crowdSize.py stops after ths line. while this program is stopped, matplotlib is running its own main function to control the actions the user can take with the image that pops up (scrolling, reset, close, etc.)

#will change to False when the window closes, which allows crowdSize.py to continue running

#if set to false, it will not stop the program from running behind the image screen

plt.show(block=False)

plt.pause(2)

plt.close()

# this block sends results to ThingSpeak

# if statement is executed if it has been more than sample\_time since we last updated the site

if (time.time() - init >= sample\_time) or (prevResult[len(prevResult) - 1] != result1):

prevResult.append(result1) #updates the most recent # of people to previous result

#uses concatenation to append the green string to our base url

#note the .format again, putting in result1, which is the number of people detected in the image, into the url

thingspeakHttp = BASE\_URL + "&field1={}".format(result1)

print(thingspeakHttp) #just prints URL to terminal

conn = urlopen(thingspeakHttp) #opens the site. the url contains information that updates the data for us when entered

print("result sent")

init = time.time() #resets the last time we updated the site to now

#these lines only execute when the while loop exits (when the program ends)

camera.release() #lets go of connection to camera

cv2.destroyAllWindows() #if any windows that display the current images are up when the program is exit, they will close

## In[7]:

def main():

record()

## In[8]:

#this line means that if this file is running code for itself (not being run by another program)

#if this file is used as a library for another project, nothing will execute without being called in the other program

if \_\_name\_\_ == '\_\_main\_\_':

main()