Raw Code ⇒

import cv2

(img)
#Our image
img_file = "car image.jpg"

[it is just a string in python, thus when we give it to openCV, we read the image in openCV.]

(xml file)
# Our pre-trained car classifier
classifier_file = "car_detector.xml"

(imported the img)
# create opencv image
img = cv2.imread(img_file)

(image read in the opencv format)

it reads all the image data from the pixel file & then reads it into some big multi-dimensional array, so that every pixel has its own data & then everything is read in the variable (img)

Display the image with the faces spotted
cv2.imshow("ai car and pedestrian tracker", img)

name of the window in which the img will be shown

the image that shows up.

# Don't autoclose (wait here in the code & listen for a key to press)
cv2.waitKey()

it means wait until you hit a key, otherwise the image is shown only for a millisecond.

we can shuffle these two, but we're doing it later, since, we can keep the above code static (that doesn't change)

# create car classifier
car_tracker = cv2.CascadeClassifier(classifier_file)

name is cascade (cause there is a long list of HAAR cascade features we're gonna run it through)

(diff. of RGB)

# convert to grayscale (needed for haar cascade)
black_n_white = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

This steps converts coloured to black & white, so that the algorithm runs ③ times faster, since, instead of ③ colours RGB black & white: we need to focus on only (speed & accuracy increases ↑)

convert color (from RGB to black & white or Red or whatever you want)

# detect cars
cars = car_tracker.detectMultiScale(black_n_white)

once we apply this (it'll show all the places where there are cars)

(classifier object)

detect cars of any size (any scale)

So, in openCV, once you have the classifier object, we can apply it on an image.

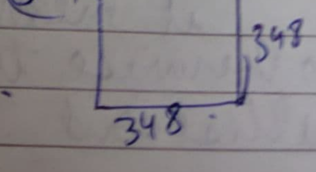
(gives the co ordinates of the rect.) - defines the square.

eg: output will come as (width) (height)

[[375 263] [348 348]]

top left pt of the rect.

height of the sq.



detect cars → we got error.

cause, our prog ~~was~~ is running on windows, but we're trying to use linux path to the cascade xml file so it was not loaded.

→ approach used:

try to replace the ~~/home/user_name/.local/lib/python3.6/~~ part with the path to the python version you really use (& ofc. check if the file is there)

You'll get this error until you get the path to the haarcascade_frontface_default.xml right.

```
# draw rectangles around the cars
for (x, y, w, h) in cars:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0,0,255), 2)
```

top left pt

bottom right

B G R

max

1

openCV allow you to draw a rectangle by using (.rectangle) command.

img

~~thickness~~

where

color (red color)

thickness (2 pixels thick)

from previous output,

eg:

a=x b=y w h

[375 263 348 348]

[700 298 175 175]]

array: In an array there are two arrays.

Now lets suppose,

```
car 1= car[0]
print(car1)
```

→ this'll give the coordinates of one car.

Now,

```
car 1 = car[0]
(x, y, w, h) = car 1.
cv2.rectangle (img, (x,y), (x+w, y+h), (0,0,255), 2)
```

(one car will be shown in a red rectangle),

for video →

```
video = cv2.Video capture('Tesla ~~dashcam~~ ~~accident~~ autopilot dashcam.mp4)
```

{ while True: → (cause we don't know for how long is the car moving)

{now what we're doing with the car image, we're to do it with each frame in the vid)

~~[illegible]~~

```
# Run forever until car stops or something
while True:

    # Read the current frame
    (read_successful, frame) = video.read()

    # safe coding
    if read_successful:
        # must convert grayscale
        grayscaled_frame = cv2.cvtColor
                (frame2.COLOR_
                 BGR2GRAY).
    else:
        break.

    # display the image with the faces spotted
    cv2.imshow(ai car detector, grayscale_
                                frame)

    # dont autoclose (wait here in the code & listen
        for a key to press)
    key = cv2.waitKey(1)
```

reads two values,
(i) if ~~read~~ was successful or not
(ii) frame which is an img from the video to check it's that or not.

tuple reads one each at a time (backend pt)

we're checking is the code valid or not (if not, it'll break out of the loop?

which will run very fast than usual cause speed clocks white

its

each frame will stay for 1 millisecond

& now grayscaled vid will be played.

Q) If different IDE is used, will speed & accuracy get impacted in output? vscode)
(opencv on)

A) No, Not at all. ~~[illegible]~~ cause only the text file is reading the img, making ~~[illegible]~~ file & doing all the tracking.

our ~~pedestrians~~ pedestrian are in yellow & cars in red.

cascade classifier::detectMultiScale →

Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

lastline of code →

```
# stop if Q key is pressed
if key == 81 or key == 113:
    break
```

→ (quit) if Q key is pressed, itll come out of the loop

← (q & Q)
(91) (113)

```
# Release the Video Capture object
video.release()
```

→ it tells the video capture obj to stop playing the vid in loop.

AI Car & Pedestrain Tracker

Main objective :- To identify cars & pedestrians, using AI&ML algorithms & python.

Kinda like:- Tesla Auto-pilot (Eg).
(helps the car to know when to stop & hence prevents accidents & promotes driver-less cars).

steps to build the app:-

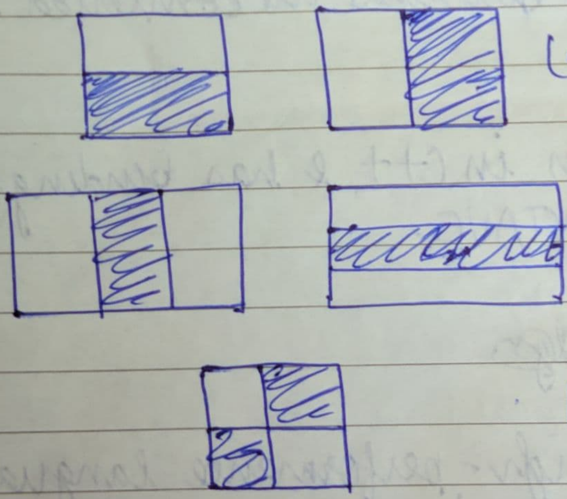
① Getting a lot of car images
② Make them all black & white
(because when its black & white, it just makes the algorithm run faster because there is less data (& hence, not worry about color data).
Also, Tesla does the same thing (It gets the vid & converts it into black & white, so that ~~do~~ decisions could be taken fastly)
③ Train the algorithm to detect cars.
(by putting them into blue/any coloured) rectangle /square boxes).

How does the computer train the algorthim?

We use 'Haar' features.
or
'Haar Cascade Classifier'

~~cascade classifier~~ is a ML object detection algorithm used to identify objects in an image or video.

Cascading classifiers are trained with several hundred "positive" sample views of a particular object (eg: cars) & arbitrary "negative" images of same size. AFTER, the classifier is trained it can be applied to a region of an image & detect the object.



(a) Edge features

(b) Line features

(c) Four-rectangle feature

We use above blocks to match with car/pedestrians. & train algo w sample code until it learns.

OpenCV → (Open Source Computer Vision Library) is an open source computer vision & ML software library.

- It was built to provide a common infrastructure for computer vision applications & to accelerate the use of ML.
- It mainly focuses on image processing, video capture and analysis including features like face detection & object detection.

- OpenCV Python is a library of python bindings to solve computer vision problems. It makes use of Numpy, which is highly optimized library for numerical operations with a MATLAB-style syntax. All openCV array structures are converted to & from Numpy arrays.

- OpenCV is written in C++ & has bindings in Python, java, MATLAB/ OCTAVE.

- ~~MATLAB is a highly~~

- MATLAB is a high-performance language for technical computing. It integrates computation, visualization & programming in an easy-to-use environment where problems & solutions are expressed in familiar mathematical notation.

- OpenCV provides a training method or pretrained models, that can be read using the [cv :: CascadeClassifier :: load] method. The pretrained models are located in OpenCV installation & can be found there.

We're not gonna train the algorithm because it'll take too much time.

As it takes a human to build a habit, ~~in the~~

A baby to learn/understand what a car & a pedestrian is; similarly, it takes a computer a long time to learn what a car is, ~~so~~ you've to give it thousands of sample images & run all the thousands of HAAR features at every location, every size & every orientation until we find the haar features that nicely match a car.

∴ Here, we are using pre-trained classifier (car classifier).

[in .xml]

(That has encapsulated all the feature in an xml file).

We're only using computer vision.
(& not lidar, i.e, camera is put at the top of the car, cause its bulky & expensive).

We've used
1) Visual Studio code (VS code)
2) Pycharm
3) openCV (Python)

RGB (Red Green Blue) → this is what all the colors are made up of.
Each pixel has a red, blue & green light, & they just mix the brightness levels of these ③ levels & they can together make any color.