**SMART VEHICLES &HELMETS :**

**Introduction :**

This work detects if the rider on the bike is wearing helmet or not . It is cable of detecting different standards and sizes of helmets from different vendors (half helmets are discarded as its not a safe option) . The helmet comes with cameras to detect motion around the rider . Say a car is approaching from behind , the cameras will see it, an algorithm will interpret it , alerting danger to the rider.

**Preperations :**

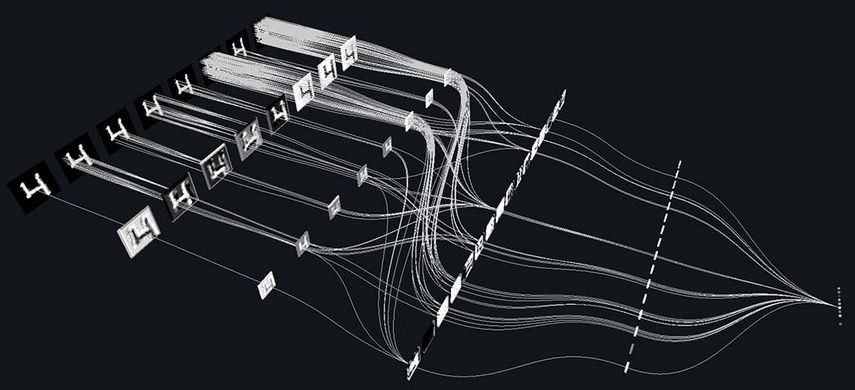
Installing dependencies : Following things are needed to execute the code we will be writing.

* Python3
* Numpy
* Opencv( sudo pip3 install opencv-python)
* pip3 .

**Image Classification :**

Object detection is actually a two-part process, image classification and then image localization. Image classification is determining what the objects in the image are, like a car or a person, while image localization is providing the specific location of these objects, as seen by the bounding boxes above.

To perform image classification, a convolutional neural network is trained to recognize various objects, like traffic lights and pedestrians. A convolutional neural network performs convolution operations on images in order to classify them.



# YOLO — You only look once, real time object detection :

YOLO (You Only Look Once) is a method / way to do object detection. It is the algorithm /strategy behind how the code is going to detect objects in the image.

Training your own custom object detection :

For training our custom object detection model, we will need a lot of images of objects which we’re going to train. nearly a few thousand. more number of images means more accuracy. I will suggest to collect data from real scenario rather downloading from google. collecting real images(in case of images) is really helpful. it’s like physics problems we performed in our schools. we use to calculate for ideal conditions. but in real life it doesn’t happen. we feel a lot of outside forces that affect the quality of machine. so, collect as much as possible images from real scenario.

Data-set preparation

For data preparation we need to use some tool to mark object in the image. YOLO have it’s own format for training data.

Yolo format is:

<object-class> <x> <y> <width> <height>

<x\_center> = ((X\_end + X\_start) / 2) / image\_width

<y\_center> = ((Y\_end + Y\_start) / 2) / image\_height

<width> = (X\_end - X\_start) / image\_width

<height> = (Y\_end - Y\_start) / image\_height

if you are using some opensource data-set, you need to convert it into this format.otherwise use this tool for marking the objects in the image. https://github.com/AlexeyAB/Yolo\_mark this is a nice tool to mark images for YOLO. this tool will create a text file for each image and the file will contain the co-ordinate of objects in the image. there is a nice explanation for it in the given link.

Setting up platform for training data :

Now we are done with data-set and next is to set the platform for training data. clone the repository from this link. https://github.com/AlexeyAB/darknet . after downloading this file move to darknet folder and use the make command to make it. there are a lot of parameters in Makefile like CPU, GPU,CUDNN,opencv etc. if you’re cloud server to train your model; you can make opencv 0.

after that, make a copy of yolov3.cfg file and make the modification in the following lines.

change the batch size to 64 and subdivision to 8. if you get any memory error in future try to adjust these values according to your hardware configuration.

change the number of classes in line 610, 696 and 783. number of classes is no of objects you’re going to train.

change the size of filter in line number 603, 689 and 776.

we define filters in YOLOv3 as, filter= (classes + 5) x 3.

create a file obj.names which contain the names of the objects (classes) we’re going to train. each class in a new line and put it in the directory build\darknet\x64\data\

create a file obj.data which will contain following lines:

classes= 2

train = data/train.txt

valid = data/test.txt

names = data/obj.names

backup = backup/

now put all image-files of your objects in the directory build\darknet\x64\data\obj\

create a file train.txt with the help of linux a command . move to terminal and navigate till data folder . from outside of data folder type

ls data/obj/\*.jpg >train.txt

it will create a text file with name of all .jpg images with path from data as data/obj/abc.jpg.

Download pre-trained weights for the convolutional layers (154 MB): https://pjreddie.com/media/files/darknet53.conv.74 and it put to the directory build\darknet\x64 .

To train on Linux use command: ./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74

on windows machine use darknet.exe instead of ./darknet.

if all went well the training will start. you can see detailed information on training like when to stop how much iteration shall i train etc at https://github.com/AlexeyAB/darknet .

training on cloud:

If you don’t own a good GPU machine, you have to train your data on cloud .matrix calculation is almost 15 times faster on GPU then CPU. and that’s all we do during training. there are a lot of cloud services which provide some free credit. register there,setup CUDA and you can start training . once done , download the model from backup folder.

you can follow this link https://medium.com/devoops-and-universe/installation-of-cuda-toolkit-on-linux-54765a3e3c7d for setting up GPU.

using the model to detect the object.

I have trained a model for helmet detection. you an download the model from my github link: https://github.com/BlcaKHat/yolov3-Helmet-Detection. the script written is simple and fast. you can set it up easily within few minutes.

Opencv implementation :

The darknet implementation for detecting objects takes a lot of time to detect the object. I have implemented a simple opencv code for it.it is much faster than darknet. you can also use it for finding specific classes and finding the co-ordinates of detected objects. just visit the link and download the necessary files . https://github.com/BlcaKHat/yolov3-Helmet-Detection

this way, you can create your model for detecting different types of objects.



# About JeVois :

JeVois is the world's smallest TensorFlow-enabled deep learning edge AI smart camera for IoT, computer vision, home automation and robotics!

JeVois runs deep neural networks that can recognize 1000 different types of objects in real time.

Insert a microSD card loaded with the provided open-source computer vision algorithms (including OpenCV 4.1.0, TensorFlow, Caffe, Darknet, and many others).