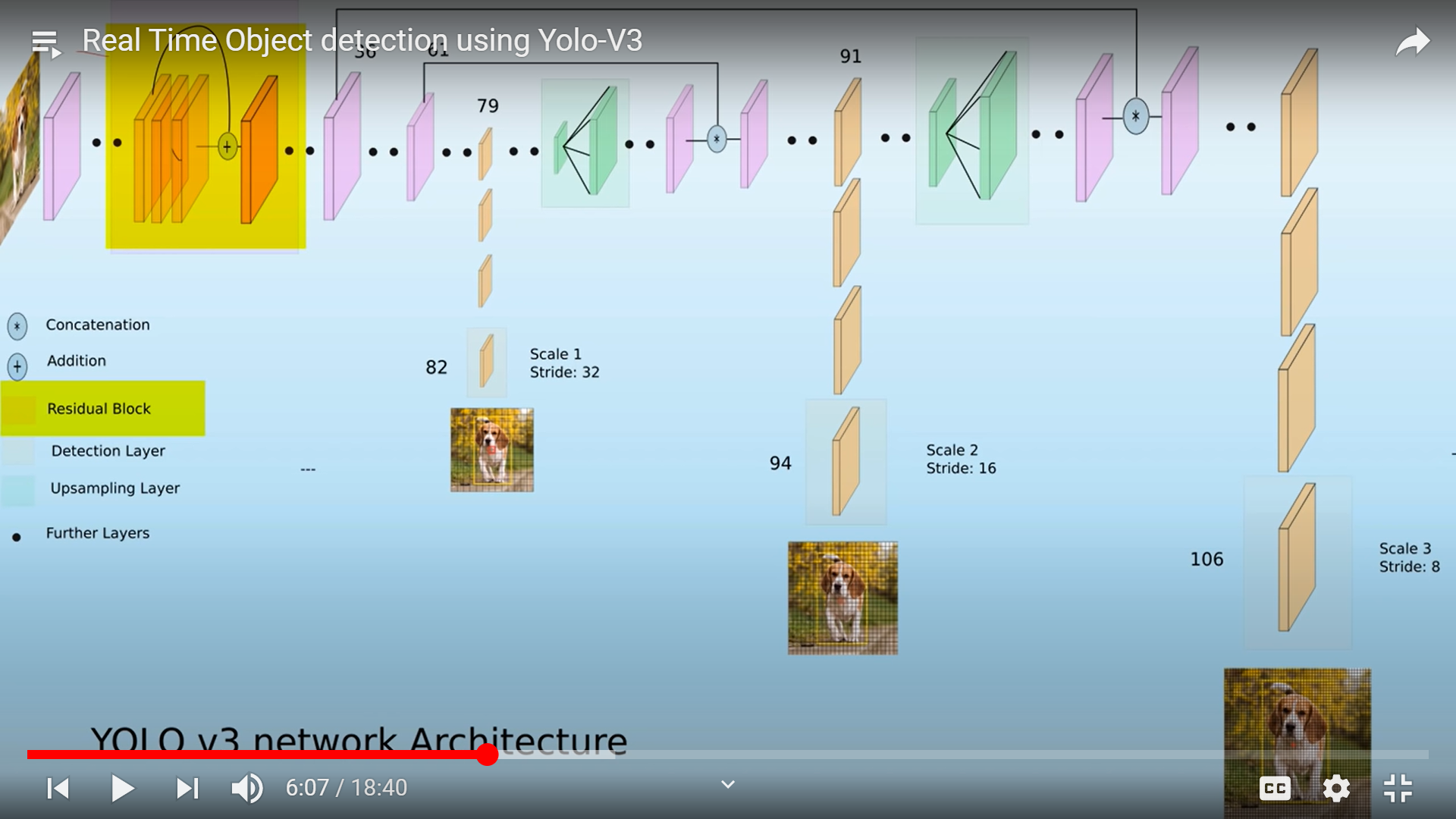
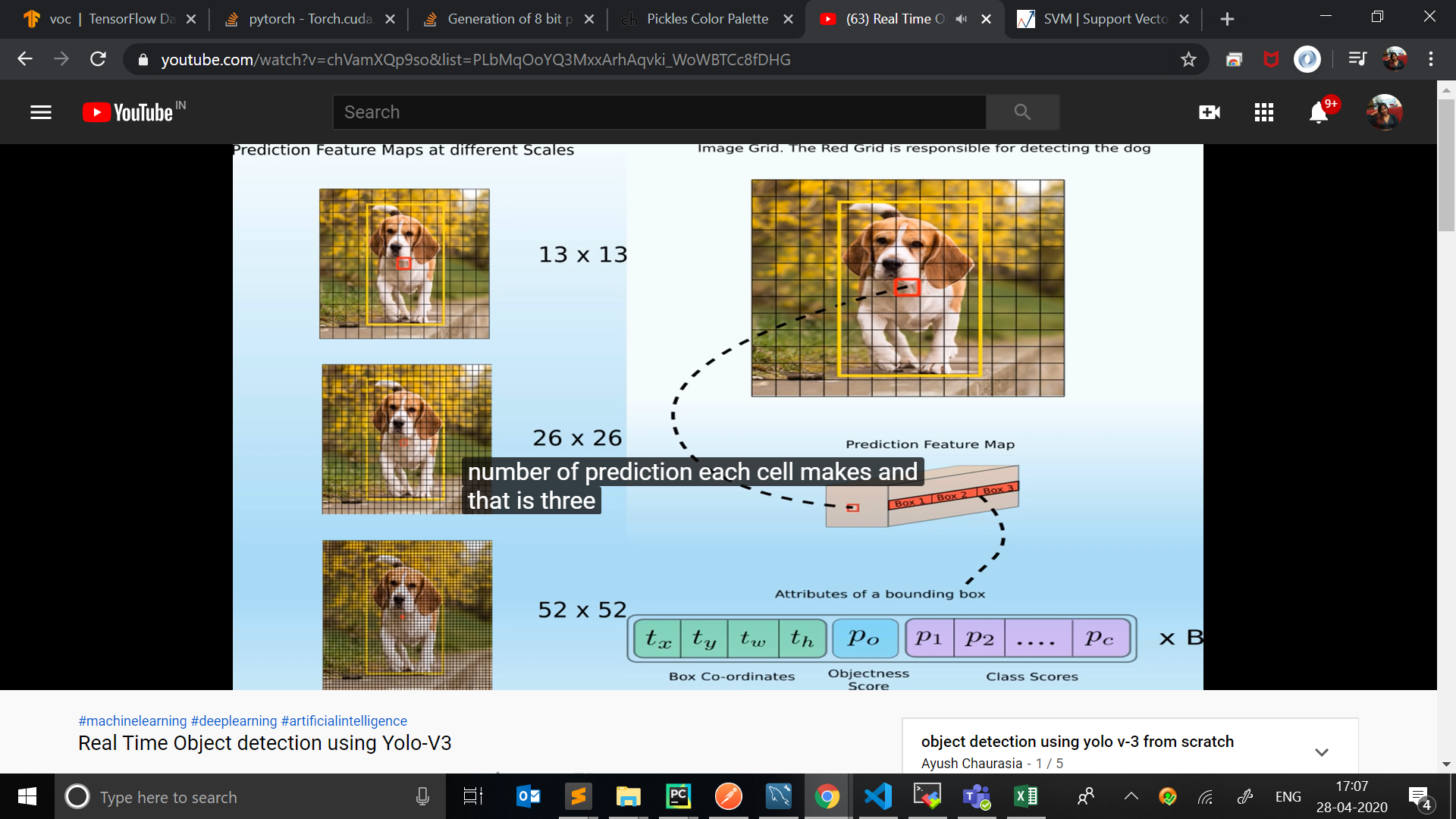
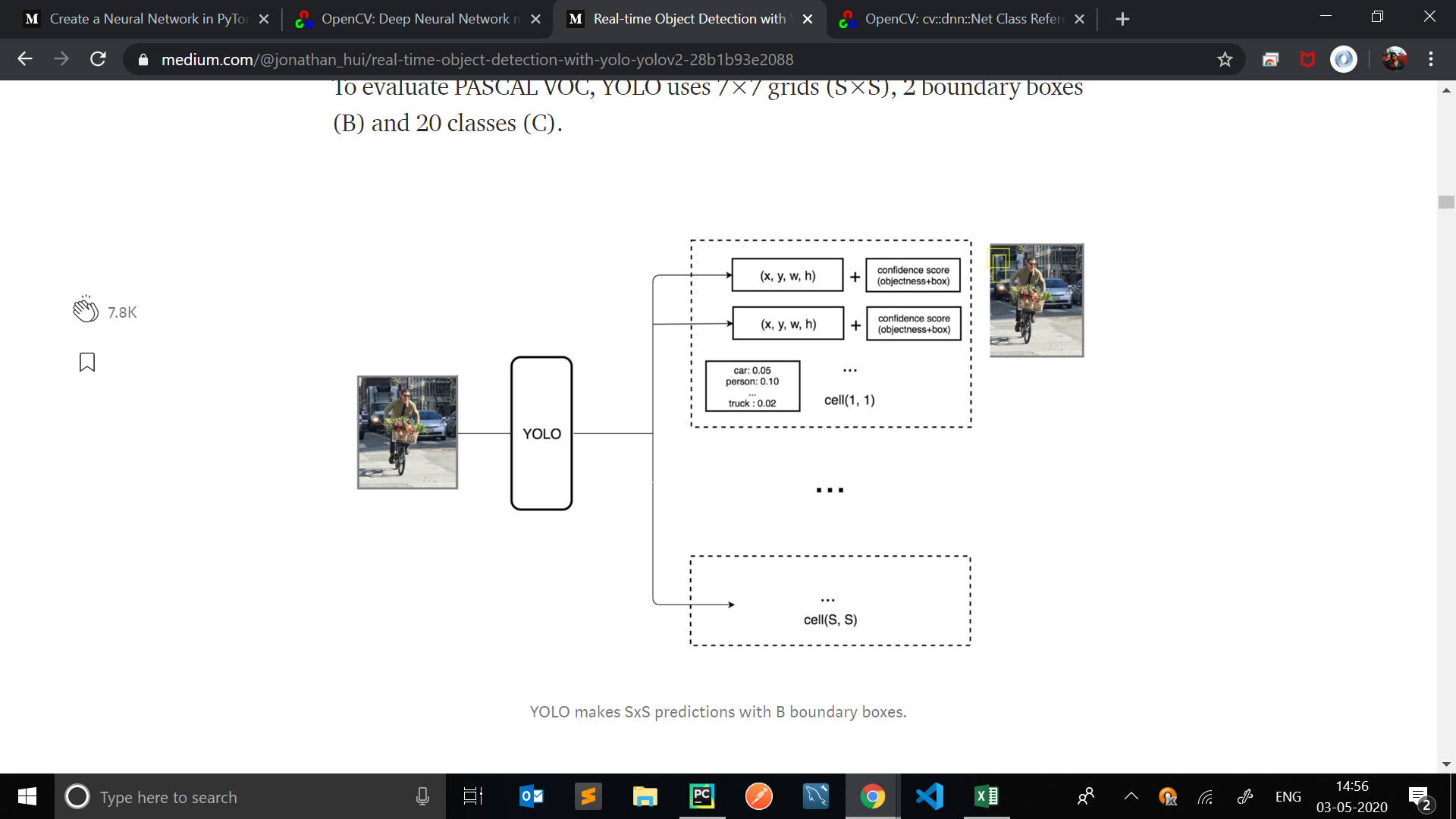
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **REAL TIME OBJECT DETECTION (YOLO notes)** | **Description** | | | |
|  |  |  |  |  |
| **YOLO** | Based on darknet(neural network) | darknet is written in C and CUDA |  |  |
|  | Pytorch Network |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Network Architechture** | Concatenation | \* |  |  |
|  | Addition | "+" |  |  |
|  | Layers | Residual block(read RES NETS) |  |  |
|  |  | Detection Layer | Detection with 3 strides(32,16,8) | 3 scales |
|  |  | Upsampling Layer |  |  |
|  |  | Further Layers |  |  |
|  |  |  |  |  |
| **Features** | 3 scale detection |  |  |  |
|  | image is divided into multiple grids |  |  |  |
|  | each grid makes 3 predictions using 3 anchor boxes |  |  |  |
|  | 3 bounding boxes for each grid |  |  |  |
|  | the grid containing an object is a final box |  |  |  |
|  | anchor\_box = predefined sized box i.e height and width of the box is defined |  |  |  |
|  | 9 anchor boxes are predefined |  |  |  |
|  | chosen using K-means clustering |  |  |  |
|  | A cell is selected if centre of an object falls in receptive field of cell |  |  |  |
|  | A cell makes 3 predictions |  |  |  |
|  |  |  |  |  |
| **Output of the network** | Output dimension at each cell | (#prediction per cell) x (5 + number of objects detected) | 3 x (5 + 80) | 80=cocoa dataset with 80 objects(probability of 80 objs) |
|  |  | 5 = centre of object(x,y), height, width, detection confidence |  |  |
|  |  | detection confidence = P(cell contains objects) |  |  |
|  |  |  |  |  |
|  | Output at each scale | 3 x 85 x number of cells |  |  |
|  |  | say Grid dimension( 8x8,16x16, 32x32) |  |  |
|  |  | For 16x16 grid distribution | 3x85x16x16 |  |
|  |  |  | 768x85(checking 85 coordinates for bounding boxes) |  |
|  |  |  |  |  |
| **Predicting the output** | Predicting the bounding boxes | bx,by,bw,bh |  |  |
|  |  | bx and by is passed to sigmoid function | 0 or 1 |  |
|  |  | bw and bh found by anchor boxes |  |  |
|  | Concatenate outputs at various scales |  |  |  |
|  |  |  |  |  |
|  | Non max suppression algorithm (NMS) | get rid of multiple detections of same object |  |  |
|  |  | best bounding box |  |  |
|  |  |  |  |  |
| **IMPLEMENTATION** |  |  |  |  |
|  |  |  |  |  |
| **cfg file** | configuration file of various yolo networks | available on yolo site |  |  |
|  |  | contains layers |  |  |
|  |  | info of layers |  |  |
|  |  | type of layer [convolutional] and properties |  |  |
| **data** | coco.names(conatins names of classes) |  |  |  |
| **yolov3.weights** | pretrained weights on yolo site |  |  |  |
|  |  |  |  |  |
| **DNModel / DNN** | Darknet model |  |  |  |
|  | constructs cfg | take config file |  |  |
|  |  | [ : start of a layer |  |  |
|  |  | entity and value |  |  |
|  |  | append network blocks |  |  |
|  |  | dictionary of networks |  |  |
|  |  |  |  |  |
|  | builds network | takes network blocks |  |  |
|  |  | modules created from pytorch |  |  |
|  |  | channels = 3 (rgb) |  |  |
|  |  | track number of filters |  |  |
|  |  | Sequential layer |  |  |
|  |  | filter + pad + kernel + stride to buiild convolutional network |  |  |
|  |  | bn: batch normalization |  |  |
|  |  | bias : True |  |  |
|  |  | activation: leaky relu |  |  |
|  |  | add all modules |  |  |
|  |  | if upsample layer then mode bilinear model |  |  |
|  |  | if route layer (concates or adds) | "-4 : concat to 4th previous layer" |  |
|  |  |  | "-4,36: concat to 36th layer" |  |
|  |  | dummy layer | route and add module |  |
|  |  | filtertracker | track no of filters |  |
|  |  | yolo layer | detector layer |  |
|  |  |  | extracts anchors(9) |  |
|  |  |  | mask: 3 detections(0,1,2) |  |
|  |  | detector layer |  |  |
|  | net | constructs all networks |  |  |
|  |  | forward network |  |  |
|  |  | written\_output |  |  |
|  |  | shortcut layer |  |  |
|  |  | 4D to 2D for concatenation |  |  |
|  |  | int | no detection |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **.py** | **darknet** | <https://pjreddie.com/darknet/> |  |  |
|  | **config\_file** | yolov3.cfg | <https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg> |  |
|  | **weights\_file** | yolov3.weights | <https://pjreddie.com/darknet/yolo/> |  |
|  | **labels** | coco.names | <https://github.com/pjreddie/darknet/blob/master/data/coco.names> |  |
|  | **darknet = cv2.dnn.readNetFromDarknet(config\_file, weights\_file)** | Reads a network model stored in Darknet model files. | it returns a network (NET object) ready to do forward propogation |  |
|  | **layer\_names = darknet.getLayerNames()** | Get the name of all layers of the network. |  |  |
|  | **[layer\_names[i[0] - 1] for i in darknet.getUnconnectedOutLayers()]** | Get the index of the output layers. | to obtain indexes of the unconnected output layers in order to find out how far function forward() must run through the network |  |
|  |  |  |  |  |
|  | **vid = cv2.VideoCapture(video\_file)** | capture video file |  |  |
|  | **vid = cv2.VideoCapture(0)** | capture video from webcam |  |  |
|  | **(ret, frame) = vid.read()** | read each frame by frame | ret : returns true or false |  |
|  | **frame.shape()** | (720, 1280, 3) | height, width, channels |  |
|  | **cv2.dnn.blobFromImage(frame,scalefactor = 1 / 255.0, size = (416, 416), swapRB=True, crop=False)** | Creates 4-dimensional blob from image. Optionally resizes and crops image from center, subtract mean values, scales values by scalefactor, swap Blue and Red channels. | 4-dimensional Mat with NCHW dimensions order. |  |
|  |  |  | N: batch size |  |
|  |  |  | C: channel |  |
|  |  |  | H: height |  |
|  |  |  | W: width |  |
|  | **darknet.setInput(blob)** | Sets the new input value for the network. | blob | A new blob. Should have CV\_32F or CV\_8U depth. |
|  |  |  | name | A name of input layer. |
|  |  |  | scalefactor | An optional normalization scale. |
|  |  |  | mean | An optional mean subtraction values. |
|  |  |  |  |  |
|  |  | If scale or mean values are specified, a final input blob is computed as: | input(n,c,h,w)=scalefactor×(blob(n,c,h,w)−meanc) |  |
|  |  |  |  |  |
|  | **layerOutputs = darknet.forward(layer\_names)** | Runs forward pass to compute output of layer with name outputName. | returns blob for first output of specified layer. |  |
|  | **cv2.dnn.NMSBoxes(boxes, confidences, object\_confidence, nms\_threshold)** | bboxes | a set of bounding boxes to apply NMS. | Performs non maximum suppression given boxes and corresponding scores. |
|  |  | scores | a set of corresponding confidences. |  |
|  |  | score\_threshold | a threshold used to filter boxes by score. |  |
|  |  | nms\_threshold | a threshold used in non maximum suppression. |  |
|  |  | indices | the kept indices of bboxes after NMS. |  |
|  |  |  |  |  |





’

References:

|  |
| --- |
| https://www.geeksforgeeks.org/pedestrian-detection-using-opencv-python/ |
| https://pjreddie.com/darknet/yolo/ |
| <https://medium.com/analytics-vidhya/camera-calibration-with-opencv-f324679c6eb7> |
| https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/ |
| https://medium.com/diaryofawannapreneur/yolo-you-only-look-once-for-object-detection-explained-6f80ea7aaa1e |
| <https://www.learnopencv.com/camera-calibration-using-opencv/> |
| <https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088> |

<https://github.com/sichkar-valentyn/What-is-mAP-for-Objects-Detection-tasks->

Colab Notebooks:

<https://colab.research.google.com/drive/1SupQi1NHwy3CW4nBmqWNLb7S6-2EmhGJ?usp=sharing>

<https://colab.research.google.com/drive/1YtNgw9haEV8eLHcJMvwmMSfRsqWfIBPi?usp=sharing>

<https://colab.research.google.com/drive/15UucTVNZEUfgWybzaC-xjM5buq5CY85U?usp=sharing>