Artefact

Thesis is done using Python Programming language as it supports multiple libraries in machine learning. I have used Syder from Anaconda for coding.

1. Python Code Description

1.1 Library Dependencies

1.1.1 from __future__ import division

from __future__ import division, will change the / operator to mean true division throughout the module. A command line option will enable run-time warnings for classic division applied to int or long arguments.

1.1.2 import time

This module provides various time-related functions

1.1.3 import torch
import torch.nn as nn
from torch.autograd import Variable

Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep learning, and uses the scripting language LuaJIT, and an underlying C implementation.

1.1.4 import numpy as np

NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

1.1.5 import cv2

OpenCV-Python is a library of Python bindings designed to solve computer vision problems

1.1.6 from util import *

from DNModel import net as Darknet
from img_process import inp_to_image, custom_resize

We are importing the files which is in the folder in all the above 3 lines of code

1.1.7 import pandas as pd

It presents a diverse range of utilities, ranging from parsing multiple file formats to converting an entire data table into a NumPy matrix array.

1.1.8 import random

The random module, which contains a variety of things to do with random number generation. Among these is the random() function, which generates random numbers between 0 and 1

1.1.9 import pickle as pkl

First, import pickle to use it, then we define an example dictionary, which is a Python object. Next, we open a file (note that we open to write bytes in Python 3+), then we use pickle. dump() to put the dict into opened file, then close. Use pickle.

1.1.10 import argparse

argparse combines the argument definitions into "groups." By default, it uses two groups, with one for options and another for required position-based arguments. import argparse parser = argparse. ArgumentParser(description='Short sample app') parser.

1.1.11 import math

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math.

1.2 Function to calculate the distance of object from the camera lens

```
def dist_calculator(startX,startY,endX,endY,box_width,box_height,img_w,img_h):
   x_3,y_3 = startX, endY - (box_height/7) # top left of the triangle

   #assumption: camera is rasied above the ground so considering 90% of the height of the image height

   x_1,y_1 = img_w/2,0.9*img_h # bottom of the triangle
   x_2,y_2 = endX , endY - (box_height/7) # top right of the triangle

   #find the angle between bottom and right point

   angle_x1_x2 = math.degrees(math.atan2(x_1-x_2,y_1-y_2))

   #find the angle between bottom and left point

   angle_x1_x3 = math.degrees(math.atan2(x_1-x_3,y_1-y_3))
```

```
    angle_right = 90 + angle_x1_x2
    angle_left = 90 - angle_x1_x3

    #total angle of view for the car from bottom center point of the image.

    total_angle = angle_right + angle_left

    # Bench length assumed to be 2 metersin millimeters. This value can automated, based on the type of bench used.

    bench_length = 2000.0


    #distance to object = (size of object) x (1°/angular size in degrees) x 57
    #Refer the link for more understadnign on the formula mentioned above - https://www.cfa.harvard.edu/webscope/activities/pdfs/measureSize.PDF

    distance = (bench_length*(1.0/total_angle)*57) / 1000
    print(total_angle)
    print(distance)
    return total_angle,distance



    #Prepare image for inputting to the neural network.
    #Perform tranpose and return Tensor


def prepare_input(img, inp_dim):

    orig_im = img
    dim = orig_im.shape[1], orig_im.shape[0]
    img = (custom_resize(orig_im, (inp_dim, inp_dim)))
    img_ = img[:,:,::-1].transpose((2,0,1)).copy()
    img_ = torch.from_numpy(img_).float().div(255.0).unsqueeze(0)
    return img_, orig_im, dim


#to give the bounding boxes to the object

def write(x, img,dimension_out):
    c1 = tuple(x[1:3].int())
    c2 = tuple(x[3:5].int())
    cls = int(x[-1])
    if dimension_out<=1:
      label = "{0}".format(classes[cls])
      if label != 'car':
        label='Car not detected'
        str1='--'
        str2='--'

      else:
        label='Car detected'
        height, width, ch = img.shape
        x_center=int(x[0])*width
        y_center=int(x[2])*height
        width_box=int(x[1])*width
        height_box=int(x[3])*height
        x1=int(x_center-width_box * 0.5) # Start X coordinate
        y1=int(y_center-height_box * 0.5)# Start Y coordinate
        x2=int(x_center+width_box * 0.5)# End X coordinate
        y2=int(y_center+height_box * 0.5)# End y coordinate
        speed,distance = dist_calculator(x1,y1,x2,y2,width_box,height_box,width,height)
        str1= str(np.max(2-distance))
        str2= str(np.abs(np.max(speed)))
    if dimension_out>1:
     label = "{0}".format(classes[cls])
     if label != 'car':
        label='Car not detected'
        str1='--'
        str2='--'

     else:
        label='Car detected'
        height, width, ch = img.shape
        x_center=int(x[0])*width
        y_center=int(x[2])*height
        width_box=int(x[1])*width
        height_box=int(x[3])*height
        x1=int(x_center-width_box * 0.5) # Start X coordinate
        y1=int(y_center-height_box * 0.5)# Start Y coordinate
        x2=int(x_center+width_box * 0.5)# End X coordinate
        y2=int(y_center+height_box * 0.5)# End y coordinate
```

#Speed and distance calculation

```
        speed,distance = dist_calculator(x1,y1,x2,y2,width_box,height_box,width,height)
        str1='distance: '+ str(np.max(2-distance))
        str2= 'Speed: '+str(np.abs(np.max(speed)))

    color = random.choice(colors)
    cv2.rectangle(img, c1, c2,color, 1)
    t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_PLAIN, 1 , 1)[0]
    #c2 = c1[0] + t_size[0] + 3, c1[1] + t_size[1] + 4
    #cv2.rectangle(img, c1, c2,color, -1)
    cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), cv2.FONT_HERSHEY_PLAIN, 1,
[225,255,255], 1);
```

```
    cv2.putText(img,str1, (400,500), cv2.FONT_HERSHEY_PLAIN, 1, [225,255,255], 1)
    cv2.putText(img,str2, (400,300), cv2.FONT_HERSHEY_PLAIN, 1, [225,255,255], 1)
    #cv2.putText(img,'_', (200,200), cv2.FONT_HERSHEY_PLAIN, 1, [225,255,255],1)
    return img
```

Parse arguements to the detect module

```
def arg_parse():

    parser = argparse.ArgumentParser(description='YOLO v3 Video Detection Module')

    parser.add_argument("--video", dest = 'video', help =
                        "Video to run detection upon",
                        default = "project_video.mp4", type = str)
    parser.add_argument("--dataset", dest = "dataset", help = "Dataset on which the
network has been trained", default = "pascal")
    parser.add_argument("--confidence", dest = "confidence", help = "Object Confidence to
filter predictions", default = 0.5)
    parser.add_argument("--nms_thresh", dest = "nms_thresh", help = "NMS Threshhold",
default = 0.4)
    parser.add_argument("--cfg", dest = 'cfgfile', help =
                        "Config file",
                        default = "cfg/yolov3.cfg", type = str)
    parser.add_argument("--weights", dest = 'weightsfile', help =
                        "weightsfile",
                        default = "yolov3.weights", type = str)
    parser.add_argument("--reso", dest = 'reso', help =
                        "Input resolution of the network. Increase to increase accuracy. Decrease to
increase speed",
                        default = "128", type = str)
    return parser.parse_args()
```

#model.cuda() will push the parameters to the default device.

```
if __name__ == '__main__':
    args = arg_parse()
    confidence = float(args.confidence)
    nms_thesh = float(args.nms_thresh)
    start = 0

    CUDA = torch.cuda.is_available()

    num_classes = 80

    bbox_attrs = 5 + num_classes

    print("Loading network")
    model = Darknet(args.cfgfile)
    model.load_weights(args.weightsfile)
    print("Network loaded")
    classes = load_classes('data/coco.names')
    colors = pkl.load(open("pallete", "rb"))
    model.DNInfo["height"] = args.reso
    inp_dim = int(model.DNInfo["height"])


    if CUDA:
        model.cuda()

    model.eval()

    videofile = args.video

    cap = cv2.VideoCapture(videofile)

    assert cap.isOpened(), 'Cannot capture source'

    while cap.isOpened():

        ret, frame = cap.read()
        if ret:


            img, orig_im, dim = prepare_input(frame, inp_dim)

            im_dim = torch.FloatTensor(dim).repeat(1,2)


            if CUDA:
                im_dim = im_dim.cuda()
                img = img.cuda()

            with torch.no_grad():
                output = model(Variable(img), CUDA)
            output = write_results(output, confidence, num_classes, nms = True, nms_conf = nms_thesh)

            if type(output) == int:
                cv2.imshow("frame", orig_im)
                key = cv2.waitKey(1)
                if key & 0xFF == ord('x'):
                    break
                continue


            im_dim = im_dim.repeat(output.size(0), 1)
            scaling_factor = torch.min(inp_dim/im_dim,1)[0].view(-1,1)
            output[:,[1,3]] -= (inp_dim - scaling_factor*im_dim[:,0].view(-1,1))/2
            output[:,[2,4]] -= (inp_dim - scaling_factor*im_dim[:,1].view(-1,1))/2

            output[:,1:5] /= scaling_factor
            print(output.shape[0])
            dimension_out=output.shape[0]
            for i in range(output.shape[0]):
                output[i, [1,3]] = torch.clamp(output[i, [1,3]], 0.0, im_dim[i,0])
                output[i, [2,4]] = torch.clamp(output[i, [2,4]], 0.0, im_dim[i,1])



            list(map(lambda x: write(x, orig_im,dimension_out), output))


            cv2.imshow("frame", orig_im)
            key = cv2.waitKey(1)
            if key & 0xFF == ord('x'):
                break
        else:
            break
```

2. Python file

Realtime_output_yolo: It is primary python code used for object detection, tracking, speed and distance estimation.

3. Referred Papers

All referred papers are present in research_papers folder.