Deploying YoloV5

The majority of tutorials I have come across only explain how to train YOLOV5 and generate bounding boxes on custom images or videos using the “detect.py” script. However, to use YOLOV5 for deployment, one needs a script that can load a trained YOLOV5 model and make decisions based on the class detected from its output. In this article, I will be using the YOLOV5s model and modifying the “detect.py” file for custom use.

The “detect.py” files accepts a trained model and image or video as an input. There are many other arguments like conf values and others but we will focus on model and input image/video only. We will write Webcam.py file for live object detection using webcam. Lets start

First of all you will have to clone/download Yolov5 from gtihub repo. Then install all the required libraries. The scope of which is out of this article.

Once you have trained model create a python file named as webcam.py or whatever you want to name it. The content of file should be the following.

```
import os
import sys
from pathlib import Path
import time
import numpy as np
import cv2
import torch
import torch.backends.cudnn as cudnn
import io
import base64
import datetime


ROOT = '/home/rcai/Desktop/yolov5'
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative



import argparse
import os
import sys
from pathlib import Path

import torch

from models.common import DetectMultiBackend

import utils
from utils.augmentations import letterbox
from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages,
LoadStreams
from utils.general import (LOGGER, check_file, check_img_size,
check_imshow, check_requirements, colorstr,
                           increment_path, non_max_suppression,
print_args, scale_boxes, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, time_sync
import cv2
from PIL import Image
import time


from scipy.spatial import distance as dist

import argparse
import imutils
import time
#import dlib
import time
from threading import Thread
import math
import cv2
import playsound
import numpy as np
import threading


import cv2
import numpy as np
import pandas as pd
import csv
import numpy
from datetime import datetime
```

```
import math



from imutils.video import VideoStream
from imutils import face_utils


device = select_device('cpu')#Set 0 if you have GPU
model = DetectMultiBackend('yolov5s.pt', device=device, dnn=False,
data='data/coco128.yaml')
model.classes = [0, 2]
stride, names, pt, jit, onnx, engine = model.stride, model.names,
model.pt, model.jit, model.onnx, model.engine
imgsz = check_img_size((640, 640), s=stride)  # check image size


dataset = LoadImages('./me.jpg', img_size=imgsz, stride=stride, auto=pt)



def draw_rect(image,points):
    x1=int(points[0])
    y1=int(points[1])
    x2=int(points[2])
    y2=int(points[3])
    midpoint=(int((x2+x1)/2),int((y2+y1)/2))
    print(midpoint)
    #print("Hi")
    cv2.rectangle(image, (x1,y1), (x2,y2), color = (255, 90, 90),
thickness=4)
    cv2.circle(image, midpoint, radius=9, color=(0, 33, 45), thickness=-1)
    y_mid=int(y2+y1/2)
    return image, y_mid

def yolo(img):
    img0=img.copy()
    img = letterbox(img0, 640, stride=stride, auto=True)[0]
    img = img.transpose((2, 0, 1))[::-1]  # HWC to CHW, BGR to RGB
    img = np.ascontiguousarray(img)
    im = torch.from_numpy(img).to(device)
    im = im.float() # uint8 to fp16/32
    im /= 255  # 0 - 255 to 0.0 - 1.0
    if len(im.shape) == 3:
        im = im[None]  # expand for batch dim
    dt = [0.0,0.0,0.0]
    pred = model(im, augment=False, visualize=False)
    seen = 0
    pred = non_max_suppression(pred, conf_thres=0.45, iou_thres=0.45,
classes=[0,1,2,3,4,6] , max_det=1000)
    det=pred[0]
    det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], img0.shape).round()
    prediction=pred[0].cpu().numpy()
    for i in range(prediction.shape[0]):
        imag,mid=draw_rect(img0,prediction[i,:])
    return imag,mid
```

```
def custom_infer(img0,
                 weights='./best.pt',  # model.pt path(s),
        data='data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.35,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=[0,1,2,3,4,6,8,10,12],  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
        model=model):



    img = letterbox(img0, 640, stride=stride, auto=True)[0]

    # Convert
    img = img.transpose((2, 0, 1))[::-1]  # HWC to CHW, BGR to RGB
    img = np.ascontiguousarray(img)
    im = torch.from_numpy(img).to(device)
    im = im.float() # uint8 to fp16/32
    im /= 255  # 0 - 255 to 0.0 - 1.0
    if len(im.shape) == 3:
        im = im[None]  # expand for batch dim
    dt = [0.0,0.0,0.0]
    pred = model(im, augment=augment, visualize=visualize)
    seen = 0
    if 1<2:

        # NMS
        pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            p, im0, frame = 'webcam.jpg', img0.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            imc = im0.copy() if save_crop else im0  # for save_crop
            annotator = Annotator(im0, line_width=line_thickness,
example=str(names))
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4],
im0.shape).round()

                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class

                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls,
*xywh)  # label format
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line +
'\n')

                    if 1<2:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if
hide_conf else f'{names[c]} {conf:.2f}')
                        annotator.box_label(xyxy, label, color=colors(c,
True))
                        if save_crop:
                            save_one_box(xyxy, imc, file=save_dir /
'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

            # Stream results
            im0 = annotator.result()
    return im0,pred

from matplotlib import pyplot as plt

def my_resize(img):
    scale_percent = 10 # percent of original size
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)

    # resize image
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
    return resized



# In[8]:
```

```
# import the opencv library
import cv2
import time
from datetime import datetime

print("Im before while Loop")
window_name = "window"
vid = cv2.VideoCapture(0)
#cv2.namedWindow(window_name, cv2.WND_PROP_FULLSCREEN)
#cv2.setWindowProperty(window_name, cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)

print("Im before while Loop 2" )

start=time.time()
while(True):
    print("Im inside the loop" )
    ret, frame = vid.read()
    print("Frame is ",type(frame))
    print("Frame shape ",frame.shape)




    # Image returned by yolo with classes
    pred_img = custom_infer(img0 = frame)[0]
    print(pred_img,"Predicted image")
    #detected classes returned by Yolo
    detected_classes=custom_infer(img0 = frame)[1]
    detected_classes
    classses=detected_classes[:][0].cpu().numpy()[:,-1]

    #cv2.imshow('ceh', pred_img)
    cv2.imshow("frame", pred_img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
vid.release()
cv2.destroyAllWindows()
```

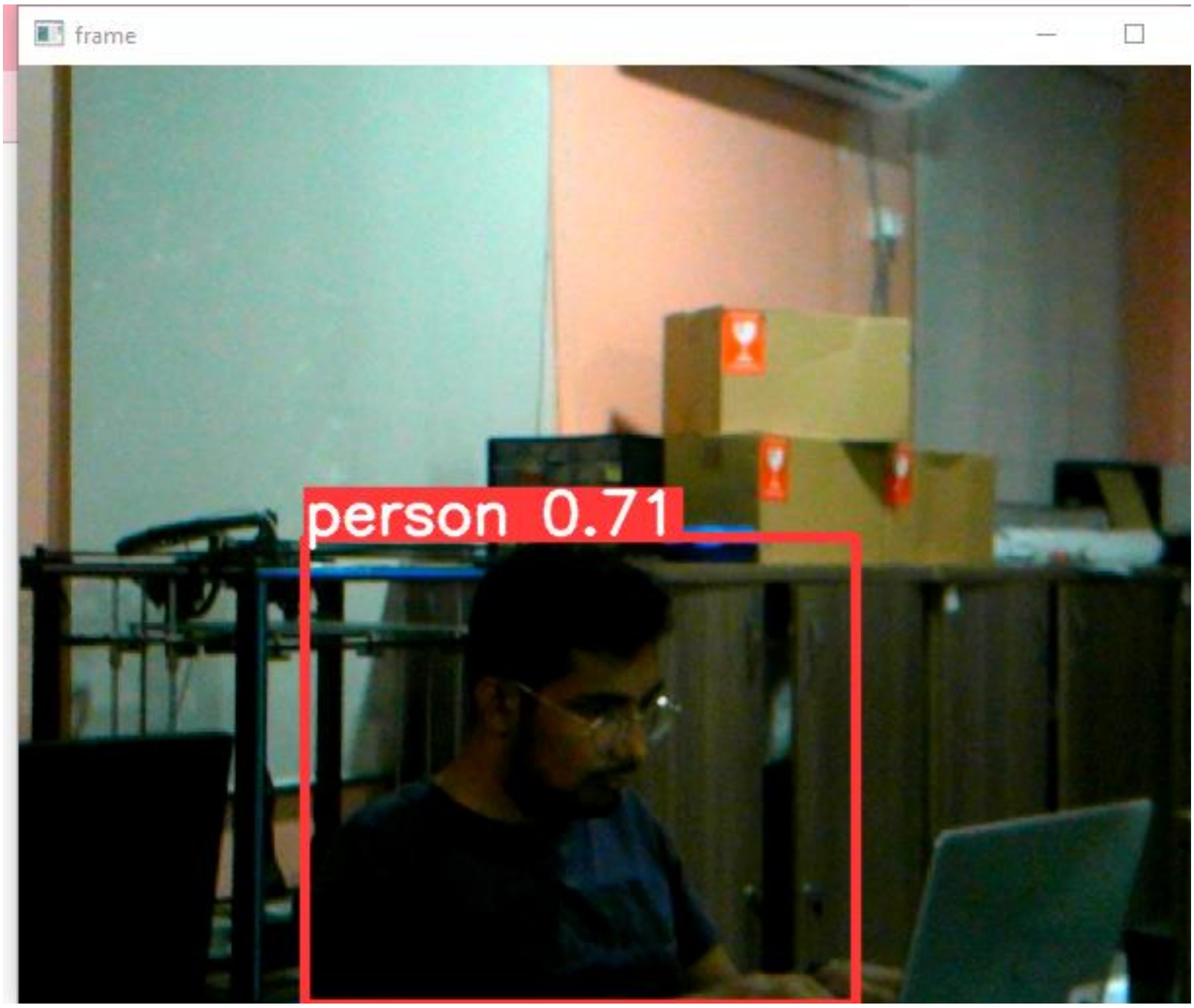
Note the function “ *custom_infer* ” will apply YOLO the trained model on image and will return image by drawing bounding boxes on it. It will also return the detected classes and their position information. We have to pass model path which we have trained and image. In the following code we are are capturing frames from web cam , feeding that frame to the trained model to perform detection.

Note we on the type of class we can write a conditional sentence to perform specific function.

```
import cv2
import time
from datetime import datetime

window_name = "window"
vid = cv2.VideoCapture(0)

print("Im before while Loop 2" )

start=time.time()
while(True):
    print("Im inside the loop" )
    ret, frame = vid.read()
    print("Frame is ",type(frame))
    print("Frame shape ",frame.shape)




    # Image returned by yolo with classes
    pred_img = custom_infer(img0 = frame)[0]
    print(pred_img,"Predicted image")
    #detected classes returned by Yolo
    detected_classes=custom_infer(img0 = frame)[1]
    detected_classes
    classses=detected_classes[:][0].cpu().numpy()[:,-1]

    #cv2.imshow('ceh', pred_img)
    cv2.imshow("frame", pred_img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
vid.release()
cv2.destroyAllWindows()
```

Save this file and run it to perform live detection.

Result can be seen as follows



live detection from frame

You can find it on my github