Detection of objects in images using YoloV5

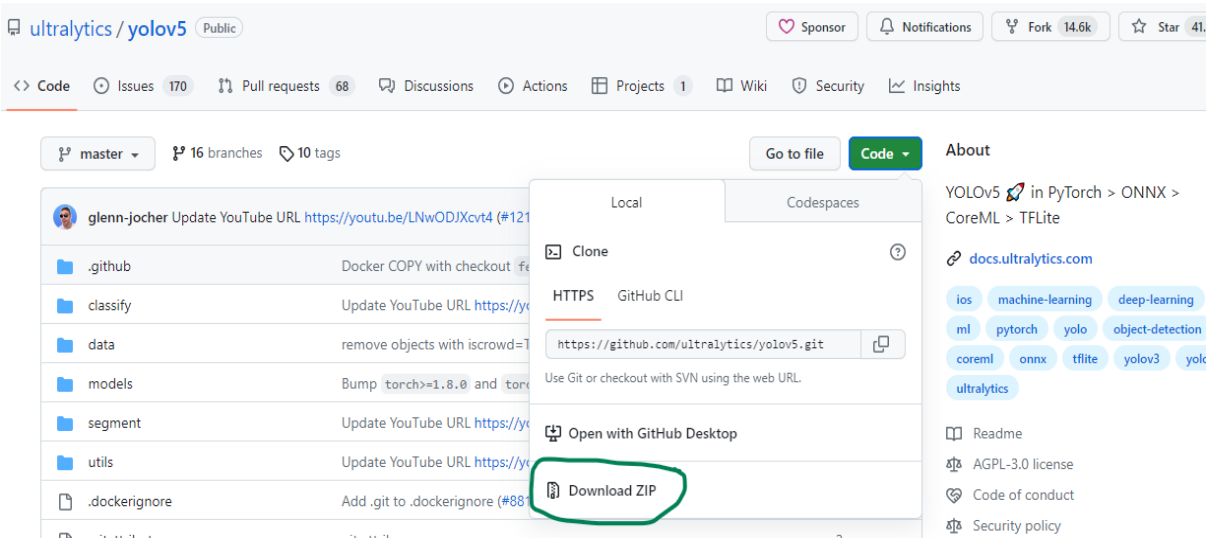
You look only once (YOLO) is an object detection model. It is a fast, easy-to-use model. YOLOv5s is a pre-trained object detection model capable of detecting up to 80 classes. I will be using the YOLOV5s model and modifying the "detect.py" file for my particular needs. The “detect.py” files accept a trained model and an image or video as input.

In this document, we detect objects in an image through Yolov's model, print the detected objects **Numpy arrays** and their **labels,** and also show the **annotated image** in the Open Computer Vision Python frame. So let's start.

The very first step is to ensure that Python is installed on your computer. To verify it, write the following command `python --version` or simply `python,` in the common prompt. The output of these commands will indicate the installed version of Python.

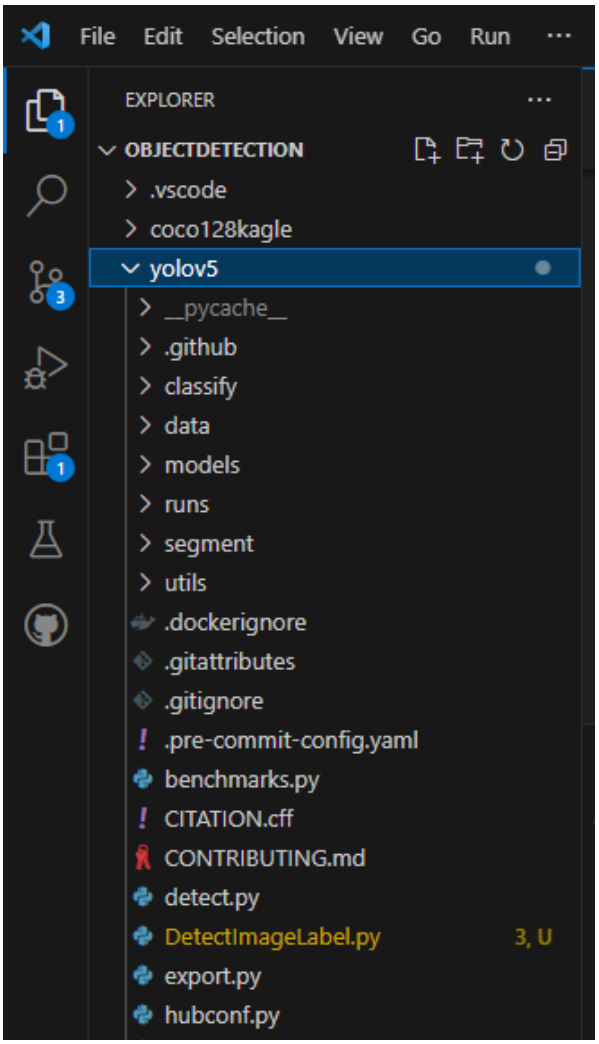
After confirming the Python installation, the next step is to clone/download Yolov5s model from the official repository. Open this link and download the project as follows:

By: Amber Riaz



Once you have this model, open it in Vscode or any other IDE you are using and create a python file named **`DetectImageLabel.py`** or whatever you want to name it.

The file structure looks like this.



The content of the file should be the following.

```
#This code snippet detects the objects from the given image path and
print the label of the detected
#also show the image with bounding boxes drawn on it.
import os
import sys
from pathlib import Path
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

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
from matplotlib import pyplot as plt
from PIL import Image
from scipy.spatial import distance as dist
import imutils
from threading import Thread
import playsound
import threading
import pandas as pd
import numpy
```

```
from datetime import datetime
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('cars1.jpg', img_size=imgsz, stride=stride,
auto=pt)
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  # pixels convert from 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()  # normalised 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 box to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if
hide_conf else f'{names[c]} {conf:.2f}')
                        print(label)
                        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

# Load and preprocess the image
image_path = 'cars1.jpg'
img = cv2.imread(image_path)  # Load the image using OpenCV
img = np.ascontiguousarray(img)  # Ensure contiguous memory layout
img_size = 640  # You can adjust the image size based on your model
img = cv2.resize(img, (img_size, img_size))  # Resize the image

print("These are the detected object")
pred_img = custom_infer(img0 = img)[0]
print(pred_img,"Predicted image") # print array of predicted image
pixels
cv2.imshow("frame", pred_img)
cv2.waitKey(0) # This line will cause the program to wait until a key
is pressed before closing the window.
```

This function (`def custom_infer()`) is an *inference function* for object detection using a pre-trained YOLOv5 model.

The function takes an input image (`'img0'`) and performs object detection using a YOLOv5 model. It uses various parameters to control the inference process, including confidence thresholds, class filtering, and visualization options.

The function processes the input image and returns the image (`'im0'`) with bounding boxes drawn around detected objects, along with a list of predictions (`'pred'`) containing information about the detected objects, including their coordinates, class labels, and confidence scores.
Save the file and run it to perform object detection.

OUTPUT:


```
PS C:\Users\HP\Desktop\IntrnshipProjects\ObjectDetection> python -u "c:\Us
op\IntrnshipProjects\ObjectDetection\yolov5\DetectImageLabel.py"
YOLOv5  v7.0-210-gdd10481 Python-3.10.6 torch-2.0.1+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
These are the detected object
car 0.38
car 0.42
car 0.43
truck 0.44
car 0.51
car 0.54
car 0.55
car 0.56
car 0.72
car 0.79
airplane 0.82
[[[184 145 106]
  [184 145 106]
  [184 145 106]
  ...
  [172 141 110]
  [172 141 110]
  [172 141 110]]

 [[184 145 106]
  [184 145 106]
  [184 145 106]
  ...
  [173 142 111]
  [173 142 111]
  [173 142 111]]

 [[185 145 107]
  [185 145 107]
  [185 145 107]
```

```
  ...

 [[ 53  76  51]
  [ 80 103  78]
  [ 80 103  78]
  ...
  [ 65 121 106]
  [ 50 105  91]
  [ 46 103  88]]

 [[107 134 108]
  [ 95 122  96]
  [ 67  94  68]
  ...
  [ 58 109  95]
  [ 60 110  98]
  [ 61 114 100]]

 [[ 95 125  99]
  [ 76 106  80]
  [ 47  74  48]
  ...
  [ 57 104  94]
  [ 73 121 112]
  [ 82 131 121]]] Predicted image
```