**YOLOv5**

**Python notebook for whole explanation of model**

**For detection we use Detect.py and to run that we use Run.py**

**Detect.py:**

YOLOv5 detection script has been adapted to add functionality for saving detection results in a CSV format. Here's a brief explanation of how this script works:

1. Imports and Initial Setup: The script imports necessary libraries, sets up paths, and defines utility functions.
2. Argument Parsing: The parse\_opt function parses command-line arguments to set configurations such as weights, source, confidence threshold, etc.
3. Main Detection Function (run): This function handles the main detection logic:
   * Loading the Model: Loads the specified YOLOv5 model.
   * Data Loading: Loads images or video streams for detection.
   * Inference and NMS: Performs inference and applies non-maximum suppression to filter out overlapping bounding boxes.
   * Saving Results: Annotates and saves images or videos with detections, and optionally writes results to a CSV file.
4. CSV Writing: A helper function write\_to\_csv is defined to append detection results to a CSV file.
5. Running the Script: The script's entry point checks requirements and starts the detection process using the provided options.

Key Additions for CSV Saving:

* Command-line Argument: Added --save-csv to control whether to save results in a CSV format.
* CSV Writing Logic: Inside the run function, write\_to\_csv is called to append each detection result to a CSV file.

Here is the complete code:

# YOLOv5 🚀 by Ultralytics, AGPL-3.0 license

"""

Run YOLOv5 detection inference on images, videos, directories, globs, YouTube, webcam, streams, etc.

Usage - sources:

$ python detect.py --weights yolov5s.pt --source 0 # webcam

img.jpg # image

vid.mp4 # video

screen # screenshot

path/ # directory

list.txt # list of images

list.streams # list of streams

'path/\*.jpg' # glob

'https://youtu.be/LNwODJXcvt4' # YouTube

'rtsp://example.com/media.mp4' # RTSP, RTMP, HTTP stream

Usage - formats:

$ python detect.py --weights yolov5s.pt # PyTorch

yolov5s.torchscript # TorchScript

yolov5s.onnx # ONNX Runtime or OpenCV DNN with --dnn

yolov5s\_openvino\_model # OpenVINO

yolov5s.engine # TensorRT

yolov5s.mlmodel # CoreML (macOS-only)

yolov5s\_saved\_model # TensorFlow SavedModel

yolov5s.pb # TensorFlow GraphDef

yolov5s.tflite # TensorFlow Lite

yolov5s\_edgetpu.tflite # TensorFlow Edge TPU

yolov5s\_paddle\_model # PaddlePaddle

"""

import argparse

import csv

import os

import platform

import sys

from pathlib import Path

import torch

FILE = Path(\_\_file\_\_).resolve()

ROOT = FILE.parents[0] # YOLOv5 root directory

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 ultralytics.utils.plotting import Annotator, colors, save\_one\_box

from models.common import DetectMultiBackend

from utils.dataloaders import IMG\_FORMATS, VID\_FORMATS, LoadImages, LoadScreenshots, LoadStreams

from utils.general import (

LOGGER,

Profile,

check\_file,

check\_img\_size,

check\_imshow,

check\_requirements,

colorstr,

cv2,

increment\_path,

non\_max\_suppression,

print\_args,

scale\_boxes,

strip\_optimizer,

xyxy2xywh,

)

from utils.torch\_utils import select\_device, smart\_inference\_mode

@smart\_inference\_mode()

def run(

weights=ROOT / "best.pt", # model path or triton URL

source=ROOT / "data/images", # file/dir/URL/glob/screen/0(webcam)

data=ROOT / "data/coco128.yaml", # dataset.yaml path

imgsz=(640, 640), # inference size (height, width)

conf\_thres=0.25, # confidence threshold

iou\_thres=0.45, # NMS IOU threshold

max\_det=1000, # maximum detections per image

device="", # 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\_csv=False, # save results in CSV format

save\_conf=False, # save confidences in --save-txt labels

save\_crop=False, # save cropped prediction boxes

nosave=False, # do not save images/videos

classes=None, # 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

vid\_stride=1, # video frame-rate stride

):

source = str(source)

save\_img = not nosave and not source.endswith(".txt") # save inference images

is\_file = Path(source).suffix[1:] in (IMG\_FORMATS + VID\_FORMATS)

is\_url = source.lower().startswith(("rtsp://", "rtmp://", "http://", "https://"))

webcam = source.isnumeric() or source.endswith(".streams") or (is\_url and not is\_file)

screenshot = source.lower().startswith("screen")

if is\_url and is\_file:

source = check\_file(source) # download

# Directories

save\_dir = increment\_path(Path(project) / name, exist\_ok=exist\_ok) # increment run

(save\_dir / "labels" if save\_txt else save\_dir).mkdir(parents=True, exist\_ok=True) # make dir

# Load model

device = select\_device(device)

model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)

stride, names, pt = model.stride, model.names, model.pt

imgsz = check\_img\_size(imgsz, s=stride) # check image size

# Dataloader

bs = 1 # batch\_size

if webcam:

view\_img = check\_imshow(warn=True)

dataset = LoadStreams(source, img\_size=imgsz, stride=stride, auto=pt, vid\_stride=vid\_stride)

bs = len(dataset)

elif screenshot:

dataset = LoadScreenshots(source, img\_size=imgsz, stride=stride, auto=pt)

else:

dataset = LoadImages(source, img\_size=imgsz, stride=stride, auto=pt, vid\_stride=vid\_stride)

vid\_path, vid\_writer = [None] \* bs, [None] \* bs

# Run inference

model.warmup(imgsz=(1 if pt or model.triton else bs, 3, \*imgsz)) # warmup

seen, windows, dt = 0, [], (Profile(device=device), Profile(device=device), Profile(device=device))

for path, im, im0s, vid\_cap, s in dataset:

with dt[0]:

im = torch.from\_numpy(im).to(model.device)

im = im.half() if model.fp16 else 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

if model.xml and im.shape[0] > 1:

ims = torch.chunk(im, im.shape[0], 0)

# Inference

with dt[1]:

visualize = increment\_path(save\_dir / Path(path).stem, mkdir=True) if visualize else False

if model.xml and im.shape[0] > 1:

pred = None

for image in ims:

if pred is None:

pred = model(image, augment=augment, visualize=visualize).unsqueeze(0)

else:

pred = torch.cat((pred, model(image, augment=augment, visualize=visualize).unsqueeze(0)), dim=0)

pred = [pred, None]

else:

pred = model(im, augment=augment, visualize=visualize)

# NMS

with dt[2]:

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

if webcam: # batch\_size >= 1

p, im0, \_ = path[i], im0s[i].copy(), dataset.count

s += f"{i}: "

else:

p, im0, \_ = path, im0s.copy(), getattr(dataset, "frame", 0)

p = Path(p) # to Path

save\_path = str(save\_dir / p.name) # im.jpg, vid.mp4, ...

txt\_path = str(save\_dir / "labels" / p.stem) + (

"" if dataset.mode == "image" else f"\_{frame}"

) # im.txt

s += "%gx%g " % im.shape[2:] # print string

gn = torch.tensor(im0.shape)[[1, 0, 1, 0]] # normalization gain whwh

imc = im0.copy() if save\_crop else im0 # for save\_crop

annotator = Annotator(im0, line\_width=line\_thickness, example=str(names))

if len(det):

det[:, :4] = scale\_boxes(im.shape[2:], det[:, :4], im0.shape).round() # rescale boxes to im0 size

for c in det[:, 5].unique():

n = (det[:, 5] == c).sum() # detections per class

s += f"{n} {names[int(c)]}{'s' \* (n > 1)}, " # add to string

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(f"{txt\_path}.txt", "a") as f:

f.write(("%g " \* len(line)).rstrip() % line + "\n")

if save\_csv: # Write to CSV

write\_to\_csv(p.name, im.shape[2:], names[int(cls)], conf, \*xyxy)

if save\_img or save\_crop or view\_img: # 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()

if view\_img:

if platform.system() == "Linux" and p not in windows:

windows.append(p)

cv2.namedWindow(str(p), cv2.WINDOW\_NORMAL | cv2.WINDOW\_KEEPRATIO)

cv2.resizeWindow(str(p), im0.shape[1], im0.shape[0])

cv2.imshow(str(p), im0)

cv2.waitKey(1) # 1 millisecond

# Save results (image with detections)

if save\_img:

if dataset.mode == "image":

cv2.imwrite(save\_path, im0)

else: # 'video' or 'stream'

if vid\_path[i] != save\_path: # new video

vid\_path[i] = save\_path

if isinstance(vid\_writer[i], cv2.VideoWriter):

vid\_writer[i].release() # release previous video writer

if vid\_cap: # video

fps = (

vid\_cap.get(cv2.CAP\_PROP\_FPS)

if vid\_cap.get(cv2.CAP\_PROP\_FPS).is\_integer()

else 30.0

)

w = int(vid\_cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

h = int(vid\_cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

else: # stream

fps, w, h = 30, im0.shape[1], im0.shape[0]

save\_path = str(

Path(save\_path).with\_suffix(".mp4")

) # force \*.mp4 suffix on results videos

vid\_writer[i] = cv2.VideoWriter(

save\_path, cv2.VideoWriter\_fourcc(\*"mp4v"), fps, (w, h)

)

vid\_writer[i].write(im0)

# Print time (inference-only)

LOGGER.info(

f"{s}Done. ({t[1] - t[0]:.3f}s)"

)

# Print results

t = tuple(x.t / seen \* 1E3 for x in dt) # speeds per image

LOGGER.info(

f"Speed: {t[0]:.1f}ms pre-process, {t[1]:.1f}ms inference, {t[2]:.1f}ms NMS per image at shape {imgsz}"

)

if save\_txt or save\_img:

s = f"\n{len(list(save\_dir.glob('labels/\*.txt')))} labels saved to {save\_dir / 'labels'}" if save\_txt else ""

LOGGER.info(f"Results saved to {colorstr('bold', save\_dir)}{s}")

if update:

strip\_optimizer(weights) # update model (to fix SourceChangeWarning)

def write\_to\_csv(filename, img\_shape, class\_name, conf, x1, y1, x2, y2):

"""Write detection results to a CSV file."""

csv\_path = "detections.csv"

header = ["filename", "img\_width", "img\_height", "class", "confidence", "x1", "y1", "x2", "y2"]

file\_exists = os.path.isfile(csv\_path)

with open(csv\_path, mode="a", newline="") as f:

writer = csv.writer(f)

if not file\_exists:

writer.writerow(header)

writer.writerow([filename, img\_shape[1], img\_shape[0], class\_name, conf.item(), x1.item(), y1.item(), x2.item(), y2.item()])

def parse\_opt():

parser = argparse.ArgumentParser()

parser.add\_argument("--weights", nargs="+", type=str, default=ROOT / "best.pt", help="model path or triton URL")

parser.add\_argument("--source", type=str, default=ROOT / "data/images", help="file/dir/URL/glob/screen/0(webcam)")

parser.add\_argument("--data", type=str, default=ROOT / "data/coco128.yaml", help="(optional) dataset.yaml path")

parser.add\_argument("--imgsz", "--img", "--img-size", nargs="+", type=int, default=[640], help="inference size h,w")

parser.add\_argument("--conf-thres", type=float, default=0.25, help="confidence threshold")

parser.add\_argument("--iou-thres", type=float, default=0.45, help="NMS IOU threshold")

parser.add\_argument("--max-det", type=int, default=1000, help="maximum detections per image")

parser.add\_argument("--device", default="", help="cuda device, i.e. 0 or 0,1,2,3 or cpu")

parser.add\_argument("--view-img", action="store\_true", help="show results")

parser.add\_argument("--save-txt", action="store\_true", help="save results to \*.txt")

parser.add\_argument("--save-csv", action="store\_true", help="save results to CSV format")

parser.add\_argument("--save-conf", action="store\_true", help="save confidences in --save-txt labels")

parser.add\_argument("--save-crop", action="store\_true", help="save cropped prediction boxes")

parser.add\_argument("--nosave", action="store\_true", help="do not save images/videos")

parser.add\_argument("--classes", nargs="+", type=int, help="filter by class: --class 0, or --class 0 2 3")

parser.add\_argument("--agnostic-nms", action="store\_true", help="class-agnostic NMS")

parser.add\_argument("--augment", action="store\_true", help="augmented inference")

parser.add\_argument("--visualize", action="store\_true", help="visualize features")

parser.add\_argument("--update", action="store\_true", help="update all models")

parser.add\_argument("--project", default=ROOT / "runs/detect", help="save results to project/name")

parser.add\_argument("--name", default="exp", help="save results to project/name")

parser.add\_argument("--exist-ok", action="store\_true", help="existing project/name ok, do not increment")

parser.add\_argument("--line-thickness", default=3, type=int, help="bounding box thickness (pixels)")

parser.add\_argument("--hide-labels", default=False, action="store\_true", help="hide labels")

parser.add\_argument("--hide-conf", default=False, action="store\_true", help="hide confidences")

parser.add\_argument("--half", action="store\_true", help="use FP16 half-precision inference")

parser.add\_argument("--dnn", action="store\_true", help="use OpenCV DNN for ONNX inference")

opt = parser.parse\_args()

opt.imgsz \*= 2 if len(opt.imgsz) == 1 else 1 # expand

print\_args(vars(opt))

return opt

def main(opt):

check\_requirements(exclude=("tensorboard", "thop"))

run(\*\*vars(opt))

if \_\_name\_\_ == "\_\_main\_\_":

opt = parse\_opt()

main(opt)

``

**Run.py:**

1. Command Explanation:
   * python detect.py: This runs the detect.py script.
   * --weights best.pt: This specifies the weights file to use.
   * --img 416: This sets the image size to 416x416 pixels.
   * --conf 0.5: This sets the confidence threshold to 0.5.
   * --source 0: This sets the source to the first webcam (webcam input).
2. Running the Command: Using os.system in Python executes the command in a subshell. Ensure you have Python and all necessary dependencies installed and configured.

Here's how we can run it in a Python script:

import os

# Run the detect.py script with the specified arguments

os.system("python detect.py --weights best.pt --img 416 --conf 0.5 --source 0")