Detect.py

```python
# YOLOv5 🚀 by Ultralytics, GPL-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/Zgi9g1ksQHc'  #
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 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 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.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, smart_inference_mode


import firebase_admin
from firebase_admin import credentials, storage
from firebase_admin import messaging
import os
import cv2
import time

# Initialize Firebase SDK
cred = credentials.Certificate('cheating-detector-2e34d-firebase-adminsdk-m32u4-
b45b1f4ed3.json')
firebase_admin.initialize_app(cred, {
        'storageBucket': 'cheating-detector-2e34d.appspot.com'
    })

def detect_and_notify_cheating():
    # # Initialize Firebase SDK
    # cred = credentials.Certificate('cheating-detector-2e34d-firebase-adminsdk-m32u4-
b45b1f4ed3.json')
    # firebase_admin.initialize_app(cred, {
    #     'storageBucket': 'cheating-detector-2e34d.appspot.com'
    # })

    bucket = storage.bucket()

    # # Function to save images
    # def save_images():
    #     capC = cv2.VideoCapture(0)
    #     for i in range(2):
    #         time.sleep(1)
    #         ret, img = capC.read()
    #         path = 'images/Cheating' + str(i) + '.jpg'
    #         cv2.imwrite('images/Cheating' + str(i) + '.jpg', img)
    #     capC.release()

    # save_images()
```

```python
    # Upload files
    folder_path = "images"
    files = os.listdir(folder_path)

    for file in files:
        if file.endswith(".jpg") or file.endswith(".png"):  # upload only image files
            file_path = os.path.join(folder_path, file)
            blob = bucket.blob(file)
            blob.upload_from_filename(file_path)
            print(files)
            print(file_path)
            print('File {} uploaded to {}.'.format(file_path, file))

    # Define notification payload
    notification = messaging.Notification(
        title='Cheating Detected',
        body='Click on the notification to see the cheater',
        image="https://static.vecteezy.com/system/resources/thumbnails/007/637/364/small_2x/n
o-smartphone-black-silhouette-ban-icon-telephone-cellphone-forbidden-pictogram-no-use-mobile-
phone-red-stop-symbol-not-allowed-smart-phone-sign-cellphone-prohibited-isolated-
illustration-vector.jpg"
    )

    # Define notification payload
    message = messaging.Message(
        notification=notification,
        topic='cheating',
    )

    # Send notification
    response = messaging.send(message)

    # Print notification ID
    print('Successfully sent cheating notification to teacher:', response)


save_dir1="images"

def save_image(image, save_dir1, image_name, detection_num):
    path = 'images/Cheating' + str(detection_num) + '.jpg'
    cv2.imwrite('images/Cheating' + str(detection_num) + '.jpg', image)
    # save_path = save_dir1 / f"detection_{detection_num}_{image_name}"
    # cv2.imwrite(str(save_path), image)
    print(f"Image saved: {path}")



@smart_inference_mode()
def run(
        weights=ROOT / 'yolov5s.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_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
):
    num_detections = 0
    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
```

```python
    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(), Profile(), Profile())
    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

        # Inference
        with dt[1]:
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize
else False
            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)

        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f'{i}: '
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            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):
                save_image(im0, save_dir1, p, num_detections)
                print("Cheating Detected")
                num_detections += 1
                if num_detections>=6:
                    num_detections=0
                    print(num_detections)
                    print("Sending Cheating Alert")
                    detect_and_notify_cheating()
                    # Call the function to save the image
                # save_image(im0, save_dir, p, num_detections)
                # Call the function when cheatig detected
                #

                # 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[:, 5].unique():
                    n = (det[:, 5] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string







                # 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(f'{txt_path}.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '¥n')

                    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)  #
allow window resize (Linux)
                    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)
                            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}{'' if len(det) else '(no detections), '}{dt[1].dt * 1E3:.1f}ms")

    # Print results
    t = tuple(x.t / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape
{(1, 3, *imgsz)}' % t)
    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[0])  # update model (to fix SourceChangeWarning)


def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.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-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: --classes 0,
or --classes 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')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    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))
```

Export.py

```
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license
"""
Export a YOLOv5 PyTorch model to other formats. TensorFlow exports authored by
https://github.com/zldrobit

Format                      | `export.py --include`         | Model
---                         | ---                           | ---
PyTorch                     | -                             | yolov5s.pt
TorchScript                 | `torchscript`                 | yolov5s.torchscript
ONNX                        | `onnx`                        | yolov5s.onnx
OpenVINO                    | `openvino`                    | yolov5s_openvino_model/
TensorRT                    | `engine`                      | yolov5s.engine
CoreML                      | `coreml`                      | yolov5s.mlmodel
TensorFlow SavedModel       | `saved_model`                 | yolov5s_saved_model/
TensorFlow GraphDef         | `pb`                          | yolov5s.pb
TensorFlow Lite             | `tflite`                      | yolov5s.tflite
TensorFlow Edge TPU         | `edgetpu`                     | yolov5s_edgetpu.tflite
TensorFlow.js               | `tfjs`                        | yolov5s_web_model/
PaddlePaddle                | `paddle`                      | yolov5s_paddle_model/

Requirements:
    $ pip install -r requirements.txt coremltools onnx onnx-simplifier onnxruntime openvino-
dev tensorflow-cpu  # CPU
    $ pip install -r requirements.txt coremltools onnx onnx-simplifier onnxruntime-gpu
openvino-dev tensorflow  # GPU

Usage:
    $ python export.py --weights yolov5s.pt --include torchscript onnx openvino engine coreml
tflite ...
```

```
Inference:
    $ 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

TensorFlow.js:
    $ cd .. && git clone https://github.com/zldrobit/tfjs-yolov5-example.git && cd tfjs-
yolov5-example
    $ npm install
    $ ln -s ../../yolov5/yolov5s_web_model public/yolov5s_web_model
    $ npm start
"""

import argparse
import contextlib
import json
import os
import platform
import re
import subprocess
import sys
import time
import warnings
from pathlib import Path

import pandas as pd
import torch
from torch.utils.mobile_optimizer import optimize_for_mobile

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
if platform.system() != 'Windows':
    ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from models.experimental import attempt_load
from models.yolo import ClassificationModel, Detect, DetectionModel, SegmentationModel
from utils.dataloaders import LoadImages
from utils.general import (LOGGER, Profile, check_dataset, check_img_size,
check_requirements, check_version,
                           check_yaml, colorstr, file_size, get_default_args, print_args,
url2file, yaml_save)
from utils.torch_utils import select_device, smart_inference_mode

MACOS = platform.system() == 'Darwin'  # macOS environment


def export_formats():
    # YOLOv5 export formats
    x = [
        ['PyTorch', '-', '.pt', True, True],
        ['TorchScript', 'torchscript', '.torchscript', True, True],
        ['ONNX', 'onnx', '.onnx', True, True],
        ['OpenVINO', 'openvino', '_openvino_model', True, False],
        ['TensorRT', 'engine', '.engine', False, True],
        ['CoreML', 'coreml', '.mlmodel', True, False],
        ['TensorFlow SavedModel', 'saved_model', '_saved_model', True, True],
        ['TensorFlow GraphDef', 'pb', '.pb', True, True],
        ['TensorFlow Lite', 'tflite', '.tflite', True, False],
        ['TensorFlow Edge TPU', 'edgetpu', '_edgetpu.tflite', False, False],
        ['TensorFlow.js', 'tfjs', '_web_model', False, False],
        ['PaddlePaddle', 'paddle', '_paddle_model', True, True],]
    return pd.DataFrame(x, columns=['Format', 'Argument', 'Suffix', 'CPU', 'GPU'])


def try_export(inner_func):
    # YOLOv5 export decorator, i..e @try_export
    inner_args = get_default_args(inner_func)

    def outer_func(*args, **kwargs):
        prefix = inner_args['prefix']
        try:
            with Profile() as dt:
                f, model = inner_func(*args, **kwargs)
            LOGGER.info(f'{prefix} export success ✅ {dt.t:.1f}s, saved as {f}
({file_size(f):.1f} MB)')
            return f, model
        except Exception as e:
            LOGGER.info(f'{prefix} export failure ❌ {dt.t:.1f}s: {e}')
            return None, None

    return outer_func


@try_export
def export_torchscript(model, im, file, optimize, prefix=colorstr('TorchScript:')):
    # YOLOv5 TorchScript model export
    LOGGER.info(f'¥n{prefix} starting export with torch {torch.__version__}...')
    f = file.with_suffix('.torchscript')
    ts = torch.jit.trace(model, im, strict=False)
    d = {'shape': im.shape, 'stride': int(max(model.stride)), 'names': model.names}
    extra_files = {'config.txt': json.dumps(d)}  # torch._C.ExtraFilesMap()
    if optimize:  # https://pytorch.org/tutorials/recipes/mobile_interpreter.html
        optimize_for_mobile(ts)._save_for_lite_interpreter(str(f), _extra_files=extra_files)
    else:
        ts.save(str(f), _extra_files=extra_files)
    return f, None


@try_export
def export_onnx(model, im, file, opset, dynamic, simplify, prefix=colorstr('ONNX:')):
    # YOLOv5 ONNX export
    check_requirements('onnx>=1.12.0')
    import onnx

    LOGGER.info(f'¥n{prefix} starting export with onnx {onnx.__version__}...')
    f = file.with_suffix('.onnx')

    output_names = ['output0', 'output1'] if isinstance(model, SegmentationModel) else
['output0']
    if dynamic:
        dynamic = {'images': {0: 'batch', 2: 'height', 3: 'width'}}  # shape(1,3,640,640)
        if isinstance(model, SegmentationModel):
            dynamic['output0'] = {0: 'batch', 1: 'anchors'}  # shape(1,25200,85)
            dynamic['output1'] = {0: 'batch', 2: 'mask_height', 3: 'mask_width'}  #
shape(1,32,160,160)
        elif isinstance(model, DetectionModel):
            dynamic['output0'] = {0: 'batch', 1: 'anchors'}  # shape(1,25200,85)

    torch.onnx.export(
        model.cpu() if dynamic else model,  # --dynamic only compatible with cpu
        im.cpu() if dynamic else im,
        f,
        verbose=False,
        opset_version=opset,
        do_constant_folding=True,  # WARNING: DNN inference with torch>=1.12 may require
do_constant_folding=False
        input_names=['images'],
        output_names=output_names,
        dynamic_axes=dynamic or None)

    # Checks
    model_onnx = onnx.load(f)  # load onnx model
    onnx.checker.check_model(model_onnx)  # check onnx model

    # Metadata
    d = {'stride': int(max(model.stride)), 'names': model.names}
    for k, v in d.items():
        meta = model_onnx.metadata_props.add()
        meta.key, meta.value = k, str(v)
    onnx.save(model_onnx, f)

    # Simplify
    if simplify:
        try:
            cuda = torch.cuda.is_available()
            check_requirements(('onnxruntime-gpu' if cuda else 'onnxruntime', 'onnx-
simplifier>=0.4.1'))
            import onnxsim

            LOGGER.info(f'{prefix} simplifying with onnx-simplifier
{onnxsim.__version__}...')
            model_onnx, check = onnxsim.simplify(model_onnx)
            assert check, 'assert check failed'
            onnx.save(model_onnx, f)
        except Exception as e:
            LOGGER.info(f'{prefix} simplifier failure: {e}')
    return f, model_onnx


@try_export
def export_openvino(file, metadata, half, prefix=colorstr('OpenVINO:')):
    # YOLOv5 OpenVINO export
    check_requirements('openvino-dev')  # requires openvino-dev:
https://pypi.org/project/openvino-dev/
    import openvino.inference_engine as ie

    LOGGER.info(f'¥n{prefix} starting export with openvino {ie.__version__}...')
    f = str(file).replace('.pt', f'_openvino_model{os.sep}')

    args = [
        'mo',
        '--input_model',
        str(file.with_suffix('.onnx')),
        '--output_dir',
        f,
        '--data_type',
        ('FP16' if half else 'FP32'),]
    subprocess.run(args, check=True, env=os.environ)  # export
    yaml_save(Path(f) / file.with_suffix('.yaml').name, metadata)  # add metadata.yaml
    return f, None


@try_export
def export_paddle(model, im, file, metadata, prefix=colorstr('PaddlePaddle:')):
    # YOLOv5 Paddle export
    check_requirements(('paddlepaddle', 'x2paddle'))
    import x2paddle
    from x2paddle.convert import pytorch2paddle

    LOGGER.info(f'¥n{prefix} starting export with X2Paddle {x2paddle.__version__}...')
    f = str(file).replace('.pt', f'_paddle_model{os.sep}')

    pytorch2paddle(module=model, save_dir=f, jit_type='trace', input_examples=[im])  # export
    yaml_save(Path(f) / file.with_suffix('.yaml').name, metadata)  # add metadata.yaml
    return f, None


@try_export
def export_coreml(model, im, file, int8, half, prefix=colorstr('CoreML:')):
    # YOLOv5 CoreML export
    check_requirements('coremltools')
    import coremltools as ct

    LOGGER.info(f'¥n{prefix} starting export with coremltools {ct.__version__}...')
    f = file.with_suffix('.mlmodel')

    ts = torch.jit.trace(model, im, strict=False)  # TorchScript model
    ct_model = ct.convert(ts, inputs=[ct.ImageType('image', shape=im.shape, scale=1 / 255,
bias=[0, 0, 0])])
    bits, mode = (8, 'kmeans_lut') if int8 else (16, 'linear') if half else (32, None)
    if bits < 32:
        if MACOS:  # quantization only supported on macOS
            with warnings.catch_warnings():
                warnings.filterwarnings('ignore', category=DeprecationWarning)  # suppress
numpy==1.20 float warning
                ct_model =
ct.models.neural_network.quantization_utils.quantize_weights(ct_model, bits, mode)
        else:
            print(f'{prefix} quantization only supported on macOS, skipping...')
    ct_model.save(f)
    return f, ct_model


@try_export
def export_engine(model, im, file, half, dynamic, simplify, workspace=4, verbose=False,
prefix=colorstr('TensorRT:')):
    # YOLOv5 TensorRT export https://developer.nvidia.com/tensorrt
    assert im.device.type != 'cpu', 'export running on CPU but must be on GPU, i.e. `python
export.py --device 0`'
    try:
        import tensorrt as trt
    except Exception:
        if platform.system() == 'Linux':
            check_requirements('nvidia-tensorrt', cmds='-U --index-url
https://pypi.ngc.nvidia.com')
        import tensorrt as trt
    if trt.__version__[0] == '7':  # TensorRT 7 handling
https://github.com/ultralytics/yolov5/issues/6012
        grid = model.model[-1].anchor_grid
        model.model[-1].anchor_grid = [a[..., :1, :1, :] for a in grid]
        export_onnx(model, im, file, 12, dynamic, simplify)  # opset 12
        model.model[-1].anchor_grid = grid
    else:  # TensorRT >= 8
        check_version(trt.__version__, '8.0.0', hard=True)  # require tensorrt>=8.0.0
        export_onnx(model, im, file, 12, dynamic, simplify)  # opset 12
    onnx = file.with_suffix('.onnx')

    LOGGER.info(f'¥n{prefix} starting export with TensorRT {trt.__version__}...')
    assert onnx.exists(), f'failed to export ONNX file: {onnx}'
    f = file.with_suffix('.engine')  # TensorRT engine file
    logger = trt.Logger(trt.Logger.INFO)
    if verbose:
        logger.min_severity = trt.Logger.Severity.VERBOSE

    builder = trt.Builder(logger)
    config = builder.create_builder_config()
    config.max_workspace_size = workspace * 1 << 30
    # config.set_memory_pool_limit(trt.MemoryPoolType.WORKSPACE, workspace << 30)  # fix TRT
8.4 deprecation notice

    flag = (1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
    network = builder.create_network(flag)
    parser = trt.OnnxParser(network, logger)
    if not parser.parse_from_file(str(onnx)):
        raise RuntimeError(f'failed to load ONNX file: {onnx}')

    inputs = [network.get_input(i) for i in range(network.num_inputs)]
    outputs = [network.get_output(i) for i in range(network.num_outputs)]
    for inp in inputs:
        LOGGER.info(f'{prefix} input "{inp.name}" with shape{inp.shape} {inp.dtype}')
    for out in outputs:
        LOGGER.info(f'{prefix} output "{out.name}" with shape{out.shape} {out.dtype}')

    if dynamic:
        if im.shape[0] <= 1:
            LOGGER.warning(f'{prefix} WARNING ⚠ --dynamic model requires maximum --batch-
size argument')
        profile = builder.create_optimization_profile()
        for inp in inputs:
            profile.set_shape(inp.name, (1, *im.shape[1:]), (max(1, im.shape[0] // 2),
*im.shape[1:]), im.shape)
        config.add_optimization_profile(profile)

    LOGGER.info(f'{prefix} building FP{16 if builder.platform_has_fast_fp16 and half else 32}
engine as {f}')
    if builder.platform_has_fast_fp16 and half:
        config.set_flag(trt.BuilderFlag.FP16)
    with builder.build_engine(network, config) as engine, open(f, 'wb') as t:
        t.write(engine.serialize())
    return f, None


@try_export
def export_saved_model(model,
                       im,
                       file,
                       dynamic,
                       tf_nms=False,
                       agnostic_nms=False,
                       topk_per_class=100,
                       topk_all=100,
                       iou_thres=0.45,
                       conf_thres=0.25,
                       keras=False,
                       prefix=colorstr('TensorFlow SavedModel:')):
    # YOLOv5 TensorFlow SavedModel export
    try:
        import tensorflow as tf
    except Exception:
        check_requirements(f"tensorflow{'' if torch.cuda.is_available() else '-macos' if
MACOS else '-cpu'}")
        import tensorflow as tf
    from tensorflow.python.framework.convert_to_constants import
convert_variables_to_constants_v2

    from models.tf import TFModel

    LOGGER.info(f'¥n{prefix} starting export with tensorflow {tf.__version__}...')
    f = str(file).replace('.pt', '_saved_model')
    batch_size, ch, *imgsz = list(im.shape)  # BCHW

    tf_model = TFModel(cfg=model.yaml, model=model, nc=model.nc, imgsz=imgsz)
    im = tf.zeros((batch_size, *imgsz, ch))  # BHWC order for TensorFlow
    _ = tf_model.predict(im, tf_nms, agnostic_nms, topk_per_class, topk_all, iou_thres,
conf_thres)
    inputs = tf.keras.Input(shape=(*imgsz, ch), batch_size=None if dynamic else batch_size)
    outputs = tf_model.predict(inputs, tf_nms, agnostic_nms, topk_per_class, topk_all,
iou_thres, conf_thres)
    keras_model = tf.keras.Model(inputs=inputs, outputs=outputs)
    keras_model.trainable = False
    keras_model.summary()
    if keras:
        keras_model.save(f, save_format='tf')
    else:
        spec = tf.TensorSpec(keras_model.inputs[0].shape, keras_model.inputs[0].dtype)
        m = tf.function(lambda x: keras_model(x))  # full model
        m = m.get_concrete_function(spec)
        frozen_func = convert_variables_to_constants_v2(m)
        tfm = tf.Module()
        tfm.__call__ = tf.function(lambda x: frozen_func(x)[:4] if tf_nms else
frozen_func(x), [spec])
        tfm.__call__(im)
        tf.saved_model.save(tfm,
                            f,
                            options=tf.saved_model.SaveOptions(experimental_custom_gradients=
False) if check_version(
                                tf.__version__, '2.6') else tf.saved_model.SaveOptions())
    return f, keras_model


@try_export
def export_pb(keras_model, file, prefix=colorstr('TensorFlow GraphDef:')):
    # YOLOv5 TensorFlow GraphDef *.pb export
https://github.com/leimao/Frozen_Graph_TensorFlow
    import tensorflow as tf
    from tensorflow.python.framework.convert_to_constants import
convert_variables_to_constants_v2

    LOGGER.info(f'¥n{prefix} starting export with tensorflow {tf.__version__}...')
    f = file.with_suffix('.pb')

    m = tf.function(lambda x: keras_model(x))  # full model
    m = m.get_concrete_function(tf.TensorSpec(keras_model.inputs[0].shape,
keras_model.inputs[0].dtype))
    frozen_func = convert_variables_to_constants_v2(m)
    frozen_func.graph.as_graph_def()
    tf.io.write_graph(graph_or_graph_def=frozen_func.graph, logdir=str(f.parent),
name=f.name, as_text=False)
    return f, None


@try_export
def export_tflite(keras_model, im, file, int8, data, nms, agnostic_nms,
prefix=colorstr('TensorFlow Lite:')):
    # YOLOv5 TensorFlow Lite export
    import tensorflow as tf

    LOGGER.info(f'¥n{prefix} starting export with tensorflow {tf.__version__}...')
    batch_size, ch, *imgsz = list(im.shape)  # BCHW
    f = str(file).replace('.pt', '-fp16.tflite')

    converter = tf.lite.TFLiteConverter.from_keras_model(keras_model)
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS]
    converter.target_spec.supported_types = [tf.float16]
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    if int8:
        from models.tf import representative_dataset_gen
        dataset = LoadImages(check_dataset(check_yaml(data))['train'], img_size=imgsz,
auto=False)
        converter.representative_dataset = lambda: representative_dataset_gen(dataset,
ncalib=100)
        converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
        converter.target_spec.supported_types = []
        converter.inference_input_type = tf.uint8  # or tf.int8
        converter.inference_output_type = tf.uint8  # or tf.int8
        converter.experimental_new_quantizer = True
        f = str(file).replace('.pt', '-int8.tflite')
    if nms or agnostic_nms:
        converter.target_spec.supported_ops.append(tf.lite.OpsSet.SELECT_TF_OPS)

    tflite_model = converter.convert()
    open(f, 'wb').write(tflite_model)
    return f, None


@try_export
def export_edgetpu(file, prefix=colorstr('Edge TPU:')):
    # YOLOv5 Edge TPU export https://coral.ai/docs/edgetpu/models-intro/
    cmd = 'edgetpu_compiler --version'
    help_url = 'https://coral.ai/docs/edgetpu/compiler/'
    assert platform.system() == 'Linux', f'export only supported on Linux. See {help_url}'
    if subprocess.run(f'{cmd} >/dev/null', shell=True).returncode != 0:
        LOGGER.info(f'¥n{prefix} export requires Edge TPU compiler. Attempting install from
{help_url}')
        sudo = subprocess.run('sudo --version >/dev/null', shell=True).returncode == 0  #
sudo installed on system
        for c in (
                'curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key
add -',
                'echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" |
sudo tee /etc/apt/sources.list.d/coral-edgetpu.list',
                'sudo apt-get update', 'sudo apt-get install edgetpu-compiler'):
            subprocess.run(c if sudo else c.replace('sudo ', ''), shell=True, check=True)
    ver = subprocess.run(cmd, shell=True, capture_output=True,
check=True).stdout.decode().split()[-1]

    LOGGER.info(f'¥n{prefix} starting export with Edge TPU compiler {ver}...')
    f = str(file).replace('.pt', '-int8_edgetpu.tflite')  # Edge TPU model
    f_tfl = str(file).replace('.pt', '-int8.tflite')  # TFLite model

    subprocess.run([
        'edgetpu_compiler',
        '-s',
        '-d',
        '-k',
        '10',
        '--out_dir',
        str(file.parent),
        f_tfl,], check=True)
    return f, None


@try_export
def export_tfjs(file, int8, prefix=colorstr('TensorFlow.js:')):
    # YOLOv5 TensorFlow.js export
    check_requirements('tensorflowjs')
    import tensorflowjs as tfjs

    LOGGER.info(f'¥n{prefix} starting export with tensorflowjs {tfjs.__version__}...')
    f = str(file).replace('.pt', '_web_model')  # js dir
    f_pb = file.with_suffix('.pb')  # *.pb path
    f_json = f'{f}/model.json'  # *.json path

    args = [
        'tensorflowjs_converter',
        '--input_format=tf_frozen_model',
        '--quantize_uint8' if int8 else '',
        '--output_node_names=Identity,Identity_1,Identity_2,Identity_3',
        str(f_pb),
        str(f),]
    subprocess.run([arg for arg in args if arg], check=True)

    json = Path(f_json).read_text()
    with open(f_json, 'w') as j:  # sort JSON Identity_* in ascending order
        subst = re.sub(
            r'{"outputs": {"Identity.?.?": {"name": "Identity.?.?"}, '
            r'"Identity.?.?": {"name": "Identity.?.?"}, '
            r'"Identity.?.?": {"name": "Identity.?.?"}, '
            r'"Identity.?.?": {"name": "Identity.?.?"}}}', r'{"outputs": {"Identity":
{"name": "Identity"}, '
            r'"Identity_1": {"name": "Identity_1"}, '
            r'"Identity_2": {"name": "Identity_2"}, '
            r'"Identity_3": {"name": "Identity_3"}}}', json)
        j.write(subst)
    return f, None


def add_tflite_metadata(file, metadata, num_outputs):
    # Add metadata to *.tflite models per
https://www.tensorflow.org/lite/models/convert/metadata
    with contextlib.suppress(ImportError):
        # check_requirements('tflite_support')
        from tflite_support import flatbuffers
        from tflite_support import metadata as _metadata
        from tflite_support import metadata_schema_py_generated as _metadata_fb
        tmp_file = Path('/tmp/meta.txt')
        with open(tmp_file, 'w') as meta_f:
            meta_f.write(str(metadata))

        model_meta = _metadata_fb.ModelMetadataT()
        label_file = _metadata_fb.AssociatedFileT()
        label_file.name = tmp_file.name
        model_meta.associatedFiles = [label_file]

        subgraph = _metadata_fb.SubGraphMetadataT()
        subgraph.inputTensorMetadata = [_metadata_fb.TensorMetadataT()]
        subgraph.outputTensorMetadata = [_metadata_fb.TensorMetadataT()] * num_outputs
        model_meta.subgraphMetadata = [subgraph]

        b = flatbuffers.Builder(0)
        b.Finish(model_meta.Pack(b), _metadata.MetadataPopulator.METADATA_FILE_IDENTIFIER)
        metadata_buf = b.Output()

        populator = _metadata.MetadataPopulator.with_model_file(file)
        populator.load_metadata_buffer(metadata_buf)
        populator.load_associated_files([str(tmp_file)])
        populator.populate()
        tmp_file.unlink()


@smart_inference_mode()
def run(
        data=ROOT / 'data/coco128.yaml',  # 'dataset.yaml path'
        weights=ROOT / 'yolov5s.pt',  # weights path
        imgsz=(640, 640),  # image (height, width)
        batch_size=1,  # batch size
        device='cpu',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        include=('torchscript', 'onnx'),  # include formats
        half=False,  # FP16 half-precision export
        inplace=False,  # set YOLOv5 Detect() inplace=True
        keras=False,  # use Keras
        optimize=False,  # TorchScript: optimize for mobile
        int8=False,  # CoreML/TF INT8 quantization
        dynamic=False,  # ONNX/TF/TensorRT: dynamic axes
        simplify=False,  # ONNX: simplify model
        opset=12,  # ONNX: opset version
        verbose=False,  # TensorRT: verbose log
        workspace=4,  # TensorRT: workspace size (GB)
        nms=False,  # TF: add NMS to model
        agnostic_nms=False,  # TF: add agnostic NMS to model
        topk_per_class=100,  # TF.js NMS: topk per class to keep
        topk_all=100,  # TF.js NMS: topk for all classes to keep
        iou_thres=0.45,  # TF.js NMS: IoU threshold
        conf_thres=0.25,  # TF.js NMS: confidence threshold
):
    t = time.time()
    include = [x.lower() for x in include]  # to lowercase
    fmts = tuple(export_formats()['Argument'][1:])  # --include arguments
    flags = [x in include for x in fmts]
    assert sum(flags) == len(include), f'ERROR: Invalid --include {include}, valid --include 
arguments are {fmts}'
    jit, onnx, xml, engine, coreml, saved_model, pb, tflite, edgetpu, tfjs, paddle = flags  # 
export booleans
    file = Path(url2file(weights) if str(weights).startswith(('http:/', 'https:/')) else 
weights)  # PyTorch weights

    # Load PyTorch model
    device = select_device(device)
    if half:
        assert device.type != 'cpu' or coreml, '--half only compatible with GPU export, i.e. 
use --device 0'
        assert not dynamic, '--half not compatible with --dynamic, i.e. use either --half or 
--dynamic but not both'
    model = attempt_load(weights, device=device, inplace=True, fuse=True)  # load FP32 model

    # Checks
    imgsz *= 2 if len(imgsz) == 1 else 1  # expand
    if optimize:
        assert device.type == 'cpu', '--optimize not compatible with cuda devices, i.e. use -
-device cpu'

    # Input
    gs = int(max(model.stride))  # grid size (max stride)
    imgsz = [check_img_size(x, gs) for x in imgsz]  # verify img_size are gs-multiples
    im = torch.zeros(batch_size, 3, *imgsz).to(device)  # image size(1,3,320,192) BCHW 
iDetection

    # Update model
    model.eval()
    for k, m in model.named_modules():
        if isinstance(m, Detect):
            m.inplace = inplace
            m.dynamic = dynamic
            m.export = True

    for _ in range(2):
        y = model(im)  # dry runs
    if half and not coreml:
        im, model = im.half(), model.half()  # to FP16
    shape = tuple((y[0] if isinstance(y, tuple) else y).shape)  # model output shape
    metadata = {'stride': int(max(model.stride)), 'names': model.names}  # model metadata
    LOGGER.info(f"¥n{colorstr('PyTorch:')} starting from {file} with output shape {shape} 
({file_size(file):.1f} MB)")

    # Exports
    f = [''] * len(fmts)  # exported filenames
    warnings.filterwarnings(action='ignore', category=torch.jit.TracerWarning)  # suppress
TracerWarning
    if jit:  # TorchScript
        f[0], _ = export_torchscript(model, im, file, optimize)
    if engine:  # TensorRT required before ONNX
        f[1], _ = export_engine(model, im, file, half, dynamic, simplify, workspace, verbose)
    if onnx or xml:  # OpenVINO requires ONNX
        f[2], _ = export_onnx(model, im, file, opset, dynamic, simplify)
    if xml:  # OpenVINO
        f[3], _ = export_openvino(file, metadata, half)
    if coreml:  # CoreML
        f[4], _ = export_coreml(model, im, file, int8, half)
    if any((saved_model, pb, tflite, edgetpu, tfjs)):  # TensorFlow formats
        assert not tflite or not tfjs, 'TFLite and TF.js models must be exported separately,
please pass only one type.'
        assert not isinstance(model, ClassificationModel), 'ClassificationModel export to TF
formats not yet supported.'
        f[5], s_model = export_saved_model(model.cpu(),
                                           im,
                                           file,
                                           dynamic,
                                           tf_nms=nms or agnostic_nms or tfjs,
                                           agnostic_nms=agnostic_nms or tfjs,
                                           topk_per_class=topk_per_class,
                                           topk_all=topk_all,
                                           iou_thres=iou_thres,
                                           conf_thres=conf_thres,
                                           keras=keras)
        if pb or tfjs:  # pb prerequisite to tfjs
            f[6], _ = export_pb(s_model, file)
        if tflite or edgetpu:
            f[7], _ = export_tflite(s_model, im, file, int8 or edgetpu, data=data, nms=nms,
agnostic_nms=agnostic_nms)
            if edgetpu:
                f[8], _ = export_edgetpu(file)
            add_tflite_metadata(f[8] or f[7], metadata, num_outputs=len(s_model.outputs))
        if tfjs:
            f[9], _ = export_tfjs(file, int8)
    if paddle:  # PaddlePaddle
        f[10], _ = export_paddle(model, im, file, metadata)

    # Finish
    f = [str(x) for x in f if x]  # filter out '' and None
    if any(f):
        cls, det, seg = (isinstance(model, x) for x in (ClassificationModel, DetectionModel,
SegmentationModel))  # type
        det &= not seg  # segmentation models inherit from SegmentationModel(DetectionModel)
        dir = Path('segment' if seg else 'classify' if cls else '')
        h = '--half' if half else ''  # --half FP16 inference arg
        s = '# WARNING ⚠ ClassificationModel not yet supported for PyTorch Hub AutoShape 
inference' if cls else ¥
            '# WARNING ⚠ SegmentationModel not yet supported for PyTorch Hub AutoShape 
inference' if seg else ''
        LOGGER.info(f'¥nExport complete ({time.time() - t:.1f}s)'
                    f"¥nResults saved to {colorstr('bold', file.parent.resolve())}"
                    f"¥nDetect:          python {dir / ('detect.py' if det else 
'predict.py')} --weights {f[-1]} {h}"
                    f"¥nValidate:        python {dir / 'val.py'} --weights {f[-1]} {h}"
                    f"¥nPyTorch Hub:     model = torch.hub.load('ultralytics/yolov5', 
'custom', '{f[-1]}')  {s}"
                    f'¥nVisualize:       https://netron.app')
    return f  # return list of exported files/dirs


def parse_opt(known=False):
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', 
help='dataset.yaml path')
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', 
help='model.pt path(s)')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640, 
640], help='image (h, w)')
    parser.add_argument('--batch-size', type=int, default=1, help='batch size')
    parser.add_argument('--device', default='cpu', help='cuda device, i.e. 0 or 0,1,2,3 or 
cpu')
    parser.add_argument('--half', action='store_true', help='FP16 half-precision export')
    parser.add_argument('--inplace', action='store_true', help='set YOLOv5 Detect() 
inplace=True')
    parser.add_argument('--keras', action='store_true', help='TF: use Keras')
    parser.add_argument('--optimize', action='store_true', help='TorchScript: optimize for 
mobile')
    parser.add_argument('--int8', action='store_true', help='CoreML/TF INT8 quantization')
    parser.add_argument('--dynamic', action='store_true', help='ONNX/TF/TensorRT: dynamic 
axes')
    parser.add_argument('--simplify', action='store_true', help='ONNX: simplify model')
    parser.add_argument('--opset', type=int, default=17, help='ONNX: opset version')
    parser.add_argument('--verbose', action='store_true', help='TensorRT: verbose log')
    parser.add_argument('--workspace', type=int, default=4, help='TensorRT: workspace size 
(GB)')
    parser.add_argument('--nms', action='store_true', help='TF: add NMS to model')
    parser.add_argument('--agnostic-nms', action='store_true', help='TF: add agnostic NMS to 
model')
    parser.add_argument('--topk-per-class', type=int, default=100, help='TF.js NMS: topk per 
class to keep')
    parser.add_argument('--topk-all', type=int, default=100, help='TF.js NMS: topk for all 
classes to keep')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='TF.js NMS: IoU 
threshold')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='TF.js NMS: confidence
threshold')
    parser.add_argument(
        '--include',
        nargs='+',
        default=['torchscript'],
        help='torchscript, onnx, openvino, engine, coreml, saved_model, pb, tflite, edgetpu,
tfjs, paddle')
    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    print_args(vars(opt))
    return opt


def main(opt):
    for opt.weights in (opt.weights if isinstance(opt.weights, list) else [opt.weights]):
        run(**vars(opt))


if __name__ == '__main__':
    opt = parse_opt()
    main(opt)
```