딥 러닝을 통해 영상에서 차를 검출하여 주차 공간 관리 시스템 구축

Establishing a Parking Space Management System by Detecting Cars in the Images through Deep Learning

송 윤 경, 이 건 호, 김 영 근*
(Yoon-kyoung Song[1] , Keon-ho Lee[1], Young-Keun Kim[1*])
[1]School of Mechanical & Control Engineering, Handong Global University

Abstract: This paper deals with building the real-time parking space management system through image processing using deep learning. YOLOv5s model was used to detect the vehicles in the images. The detection model detected vehicles with a performance of 0.014 second frame time and 98.91% of accuracy. This showed that it is possible to establish an accurate and high-speed real-time parking management system with a huge amount of CCTV data.

Keywords: Deep-Learning, image processes, parking management system, yolo v5

I. 서론

요즘 사회의 지속적인 문제 중 하나는 주차 공간의 부재이다. 포항시의 육거리와 죽도시장 근교에도 주차 공간이 부족하여 교통이 혼잡해지는 문제가 발생한다. 또한 이 곳에는 공영 주차장이외에도 여러 사설 주차장들이 많아 주차가 가능한 공간의 여부를 확인하기 어렵다. 행정기관에서는 이를 해결하기 위해 각 주차장의 빈 주차공간을 거리의 전광판에 안내하여 해결하고자 한다.

주차 가능 공간을 전광판에 안내하기 위해서는 먼저 각 주차장의 주차 가능 공간을 파악해야 한다. 주차 공간을 파악하기 위해 현재 사용되는 방법에는 초음파 센서 등을 이용하여 주차 공간에 주차된 차량을 감지하는 방법이 있다. 하지만 이는 주차공간 천장에 무선 초음파 센서를 주차 공간 하나 하나에 설치해야 하기 때문에 공간에 따라 설치의 제약이 있을 수 있고, 그러한 시스템을 구축하는데 많은 비용이 사용된다. 그래서 포항시의 CCTV를 사용하여, 딥 러닝을 통해 주차장의 실시간 주차 가능 공간과 주차한 차의 수를 확인하는 시스템을 구축하고자 한다. 이렇게 주차 가능한 공간을 센 후에는 이를 전송하여 전광판에 띄우는 것이 목적이지만 현재는 하나의 영상을 가지고 확인을 하기 때문에 영상에 직접 주차 가능 공간과 주차장에 들어온 차의 수를 세어 표시하려 한다.

본 연구에서는 CCTV 영상에서 딥 러닝을 통해 자동차를 찾고 그 수를 세어 주차장에 잔여 주차 공간의 수를 화면에 띄우는 것이 목적이다. 실제로 이를 사용하여 주차 공간 부족의 문제를 해결하기 위해서는 방대한 양의 CCTV 데이터를 처리하고 실시간으로 이를 전송하여 전광판 띄워야 한다. 이러한 실시간 데이터 처리 및 전송을 위해선 빠른 데이터 처리 속도가 필수적이다. 따라서 수준 높은 정확도와 빠른 데이터 처리를 딥러닝 설계의 중점으로 두고 프로젝트를 진행하여 범용성을 확대하고자 한다.

II. 실험 환경

이번 연구에서 Object Detection을 위해 구축한 딥러닝 환경은 다음과 같다.

- Nvidia GeForce GTX 1660 Super
- Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
- 16GB RAM
- Yolov5- Yolov5x, Yolov5s
- PyTorch 1.7.1

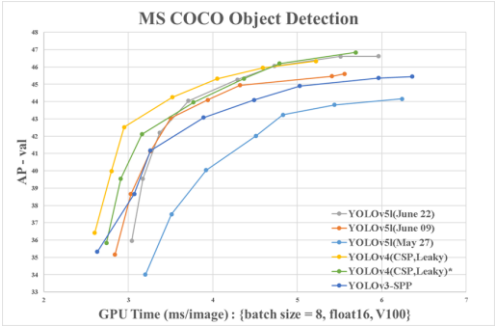

그림 1. YOLO 버전간 성능 비교
Fig 1. YOLO version comparison

딥러닝 모델로 사용한 YOLOv5 모델은 YOLO 모델 시리즈 중에 가장 최신에 나온 모델이며, PyTorch 환경에서 구동한다. Joseph Redmon의 YOLOv3 모델과 비교했을 때 YOLOv5는 FPS 및 mAP 측면에서 모두 뛰어난 성능을 발휘하는 모델이다. YOLOv5는 또한 YOLOv4 과 비교해서도 비슷한 성능에 낮은 용량 (가중치 파일의 크기) 빠른 속도를 보여 이 프로젝트와 같은 실시간 다중 영상 처리 또는 임베디드 시스템에 유리한 모델이다.

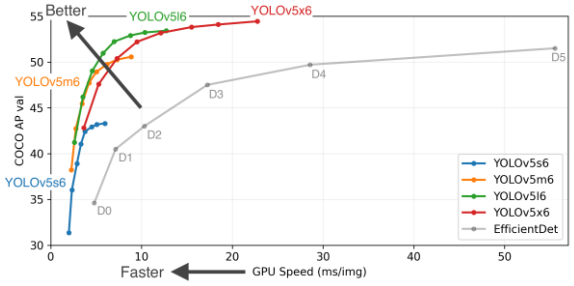

그림 2. YOLO v5의 하위 모델 비교
Fig 2. YOLO under version comparison

YOLOv5는 YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x 등의 크기별로 여러 개 모델로 나누어진다. 위의 모델들의 이름 끝에 붙은 s, m, l, x는 각각 small, medium, large, xlarge를 의미하며, s는 경량화 버전, x는 가장 느린 대신 가장 높은 정확도를 가지는 버전이다. 이번 연구에서는 YOLOv5를 사용하며, 그 세부 모델 중 YOLOv5s 와 YOLOv5x 모델의 정확도 및 빠르기를 비교 분석하여 가장 적합한 모델을 사용하고자 한다.

III. 딥러닝 영상처리 프로세스

1. 객체 검출 방식

CCTV 영상에서 검출되어야 하는 것은 자동차이다. 그 외의 다른 대상은 어떤 대상이든 알 필요가 없으므로, classes를 자동차 하나로 제한하여 자동차만 검출하도록 하였다. 2번 class가 자동차를 의미하므로, class를 2로 설정하여 자동차인 객체만 검출하였다.

2. 객체 검출을 위한 threshold 설정

객체 검출을 하기 위해서 yolo v5에서 설정하는 threshold는 confidence와 IoU(Intersection over Union)가 있다. Confidence score는 모델의 최종 레이어에서 출력되는 값으로 sigmoid 또는 softmax값으로 0~1의 값을 가진다. IoU는 예측한 박스에 실제 대상이 차지하는 비율을 나타낸 값으로 이 또한 0~1의 값을 가진다. CCTV에서 차량을 검출하고자 할 때 성능 평가 지표의 precision과 recall 중에서 recall이 더 중요하다고 판단하였다.

차량을 검출했다고 예측한 경우 중 실제로 차량이 있었는지를 보는 precision의 관점보다, 실제로 차량이 있는 경우 중에 차량을 검출한 비율을 보는 recall을 더 비중 있게 살핀 이유는, 사용자의 입장에서 차를 검출하지 못하여 주차가 불가능한 공간을 주차 공간이라 안내하는 것이 더 불편할 것이라 판단했기 때문이다. 따라서 precision과 recall의 trade-off 관계에서 recall 값이 높아지도록 파라미터를 조정하였다. IoU의 threshold가 낮을수록 recall이 높아지므로 IoU의 threshold를 낮게 설정하였다. Confidence score 또한 차량을 검출하는 것이 더 초점을 맞춰 threshold를 낮게 설정하였다. 결과적으로 두 threshold를 모두 0.3으로 설정하였다.

3. 이미지 사이즈 및 하위 모델 선정

객체를 검출할 때 이미지 픽셀 사이즈와 하위 모델을 바꿔줌에 따라 속도 및 정확도에 차이를 보인다. 이미지 사이즈는 클수록 하나의 프레임을 처리하는 속도인 Frame Time이 커진다. 이미지 사이즈는 모델의 종류와 더불어 Accuracy에 큰 영향을 미치는 파라미터이다. 일반적으로 image size는 train image size와 test image size를 같게 사용하는 것이 딥러닝 객체 검출에 있어 일반적인 방법이다. 이 프로젝트에서 사용한 YOLOv5의 train image size는 640으로 모두 동일하다. 이번 Object detection에 사용한 영상의 경우 Frame의 너비 pixel 사이즈는 1280으로, test image의 픽셀 사이즈가 1280 이다. 따라서 모든 YOLOv5의 하위 모델에서 두 이미지 사이즈를 적용하여 속도와 정확도를 비교하였다.

실험은 동일한 GPU 환경에서 이루어졌으며, YOLOv5의 모든 모델과 이미지 size에 따른 정확도를 비교 및 분석하였다. 표 1은 yolo5의 하위 모델들의 정확도와 데이터 처리 속도를 표로 정리한 것으로, 차량 인식 대수에 대한 reference ground truth value는 영상의 모든 프레임(Every Frame)에서 화면상에 차량으로 인식할 수 있는 모든 차량의 대수를 기준으로 하였으며, 주차 구역에 정차된 차량 뿐만 아니라 도로를 지나는 차량 또한 검출 대상에 포함하여 주차 가능 공간을 계산하였다.

표 1. YOLOv5의 모델과 이미지 사이즈에 따른 정확도와 Frame Time
Table 1. Accuracy and Frame Time according to YOLOv5 model and image size

모델명	Image Size	Average Frame Time	Accuracy
YOLOv5s	640	0.0149 [sec]	70.84%
	1280	0.0239 [sec]	98.909%
YOLOv5m	640	0.0252 [sec]	70.12%
	1280	0.0467 [sec]	98%
YOLOv5l	640	0.0371 [sec]	99.76%
	1280	0.0806 [sec]	97.697%
YOLOv5x	640	0.0505 [sec]	99.697%
	1280	0.1361 [sec]	95.273%

실험 결과 YOLOv5s의 경우 이미지 사이즈가 1280일 때 가장 높은 정확도를 보였으며, YOLOv5x의 경우 이미지 사이즈가 640일 때 가장 높은 정확도를 보였다. Accuracy의 경우 95% 이상의 경우엔 정확도가 어느정도 확보되었다고 보아 정확도보다 속도를 우선적으로 보았다. YOLOv5s 모델의 경우 1280 이미지 사이즈에서 98.909%의 정확도와 0.0239 sec의 평균 Frame Time을 보이고, YOLO5x 모델의 경우 640 이미지 사이즈에서 99.697%의 정확도와 0.0505 sec의 평균 Frame Time을 보인다.

따라서 이번 프로젝트에서는 정확도를 어느 정도 확보한 모델 중에 Frame Time이 가장 작은 YOLOv5s 모델의 image size 1280을 사용하였다.

5. 화면에 표시

Object Detection에 사용한 프로그램 코드 함수의 내부를 보면 각 class의 수를 반환하여 이를 콘솔 창에 출력하도록 코

드가 설정되어 있다. 이 부분에서 class가 car일 때의 수를 변수 num_car에 저장하였다. 또한 총 주차 공간이 13개임을 미리 알고 있으므로 전체 주차 공간과 검출된 차의 수, 주차 가능 공간과 처리하는데 걸린 시간을 OpenCV의 putText함수를 사용하여 영상의 왼쪽 상단에 출력하도록 하였다.

그림 3. 이미지에서 검출된 차 및 주차 가능 공간 수 출력

Fig 3. Output the number of cars and parking spaces detected in the image

IV. 실험 결과

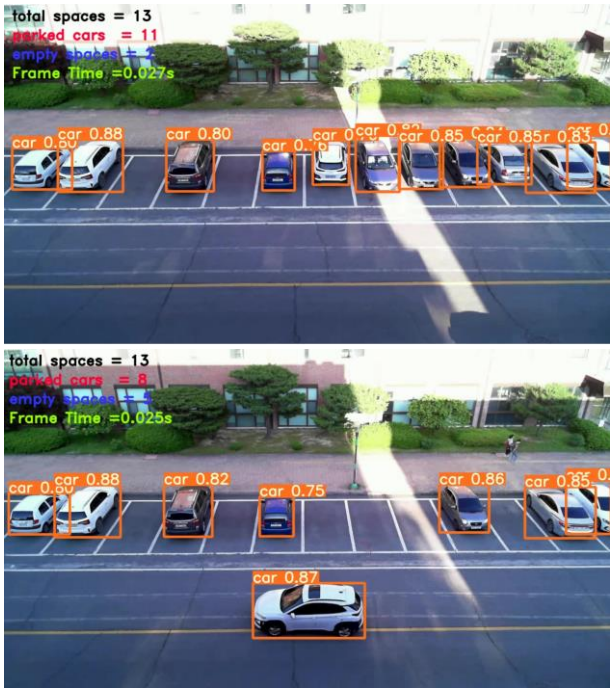

그림 4. 차량 검출 결과 영상 이미지

Fig 4. Vehicle detection result image

위의 그림 4은 같이 영상의 각 프레임에서 차량을 검출하고 검출한 차량의 대수 및 남은 자리의 개수를 출력한 이미지이다. YOLOv5s 모델을 사용하였으며 Accuracy 98.909%, 평균 Frame Time 0.0149 초의 성능을 보인다.

V. 결론

YOLOv5s 모델을 이용한 딥러닝 영상 처리를 통해 CCTV 영상에서 차량을 검출하여 잔여 주차 공간을 화면에 표시하였다. 객체 검출에 널리 사용되는 YOLO의 다양한 버전 중 성능 및 속도를 고려해 YOLOv5 로 영상 객체 검출을 하였으며, 그 하위 모델 중 정확도 및 속도 측면에서 본 연구의 목적에 가장 적합한 YOLOv5s 모델을 최종적으로 선택하여 차량을 검출하였다. 본 연구에선 포항시의 주차 공간 부족의 문제를 해결하기 위해 CCTV 영상 정보를 이용해 실시간 주차 가능 공간을 파악하는 시스템을 구축하고자 하였다. 검출 모델은 98.91%의 Accuracy와 함께 평균 Frame Time 0.024초의 성능으로 영상의 차량 탐지하였다. 이것은 기존의 초음파 센서 등의 별도 센서 및 추가 시설 없이 CCTV 영상 만으로 실시간 높은 정확도의 차량 대수 파악이 가능함을 보였으며, 이를 사용하여 방대한 양의 CCTV 데이터를 실시간으로 처리하는 주차 관리 시스템 구축할 수 있다.

참고문헌

[1] Josep nelson, “MS COCO Object Detection”. Kaggle, 2020. URL:https://www.kaggle.com/c/global-wheat detection/discussion/165414. Kaggle Community. Web. 07 June. 2021.

[2] Ultralytics. *Ultralytics Github*: *Yolov5*. May. 2020. URL: https://github.com/ultralytics/yolov5. Web. 07 June. 2021.

Appendix

1. Program Code

```python
import argparse
import time
from pathlib import Path

import cv2
from numpy import source
import torch
import torch.backends.cudnn as cudnn

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path, save_one_box
from utils.plots import colors, plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized

counting_result = ''

@torch.no_grad()
def detect(opt):
    source, weights, view_img, save_txt, imgsz = opt.source, opt.weights, opt.view_img, opt.save_txt, opt.img_size
    save_img = not opt.nosave and not source.endswith('.txt')  # save inference images
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    # Directories
    save_dir = increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Initialize
    set_logging()
    device = select_device(opt.device)
    half = device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model
    stride = int(model.stride.max())  # model stride
    imgsz = check_img_size(imgsz, s=stride)  # check img_size
    names = model.module.names if hasattr(model, 'module') else model.names  # get class names
    if half:
        model.half()  # to FP16

    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['model']).to(device).eval()

    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride)
    else:
        view_img = check_imshow()
        dataset = LoadImages(source, img_size=imgsz, stride=stride)

    # Run inference
    if device.type != 'cpu':
        model(torch.zeros(1, 3, imgsz, imgsz).to(device).type_as(next(model.parameters())))  # run once
    t0 = time.time()
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)

        # Inference
        t1 = time_synchronized()
        pred = model(img, augment=opt.augment)[0]

```

```
        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, opt.classes, opt.agnostic_nms,
                                   max_det=opt.max_det)
        t2 = time_synchronized()

        # Apply Classifier
        if classify:
            pred = apply_classifier(pred, modelc, img, im0s)

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

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # img.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # img.txt
            s += '%gx%g ' % img.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if opt.save_crop else im0  # for opt.save_crop
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()

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

                    if names[int(c)]=='car':
                        car_num = n
                        empty_num = 13- car_num

                # 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 opt.save_conf else (cls, *xywh)  # label format
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')


                    if save_img or opt.save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if opt.hide_labels else (names[c] if opt.hide_conf else f'{names[c]} {conf:.2f}')
                        plot_one_box(xyxy, im0, label=label, color=colors(c, True), line_thickness=opt.line_thickness)
                        if opt.save_crop:
                            save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

            # Print time (inference + NMS)
            print(f'{s}Done. ({t2 - t1:.3f}s)')

            txt_path = str(save_dir / 'counting_result')
            with open(txt_path + '.txt', 'a') as f:
                f.write(f"{frame-1}, {car_num}" + '\n')

            # Stream results
            if view_img:

                font = cv2.FONT_HERSHEY_DUPLEX
                totalspace = f"{'total spaces ='} {13}"
                parkedcars = f"{'parked cars  ='} {car_num}"
                emptyspace = f"{'empty spaces ='} {empty_num}"
                TT = f"{'Time ='} {time.time() - t0:.3f} {'s'}"
                cv2.putText(im0, totalspace, (20, 40), font, 1,(51,255,255), 2, cv2.LINE_AA) #BGR
                cv2.putText(im0, parkedcars, (20, 80), font, 1,(51,0,255), 2, cv2.LINE_AA)
                cv2.putText(im0, emptyspace, (20, 120), font, 1,(255,51,51), 2, cv2.LINE_AA)
                cv2.putText(im0, TT, (20, 160), font, 1,(0,255,153), 2, cv2.LINE_AA)
                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 != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.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 += '.mp4'
                        vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer.write(im0)

    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 ''
        print(f"Results saved to {save_dir}{s}")


    print(f'Done. ({time.time() - t0:.3f}s)')


```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='parking.mp4', help='source')  # file/folder, 0 for webcam
    parser.add_argument('--img-size', type=int, default= 1280, help='inference size (pixels)') # train and test image size
    parser.add_argument('--conf-thres', type=float, default=0.3, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.3, help='IOU threshold for NMS')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum number of 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='display 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, default=2, 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('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='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')
    opt = parser.parse_args()
    print(opt)
    check_requirements(exclude=('tensorboard', 'pycocotools', 'thop'))

    if opt.update:  # update all models (to fix SourceChangeWarning)
        for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:
            detect(opt=opt)
            strip_optimizer(opt.weights)
    else:
        detect(opt=opt)

```