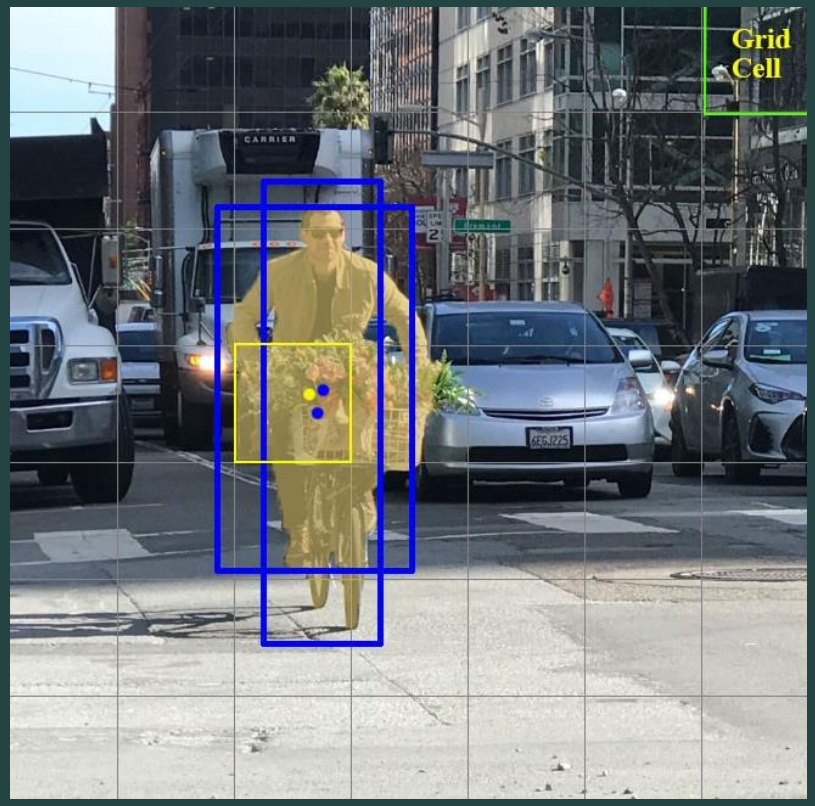
# YOLOV5를 이용한 상품탐지 및 총합 계산

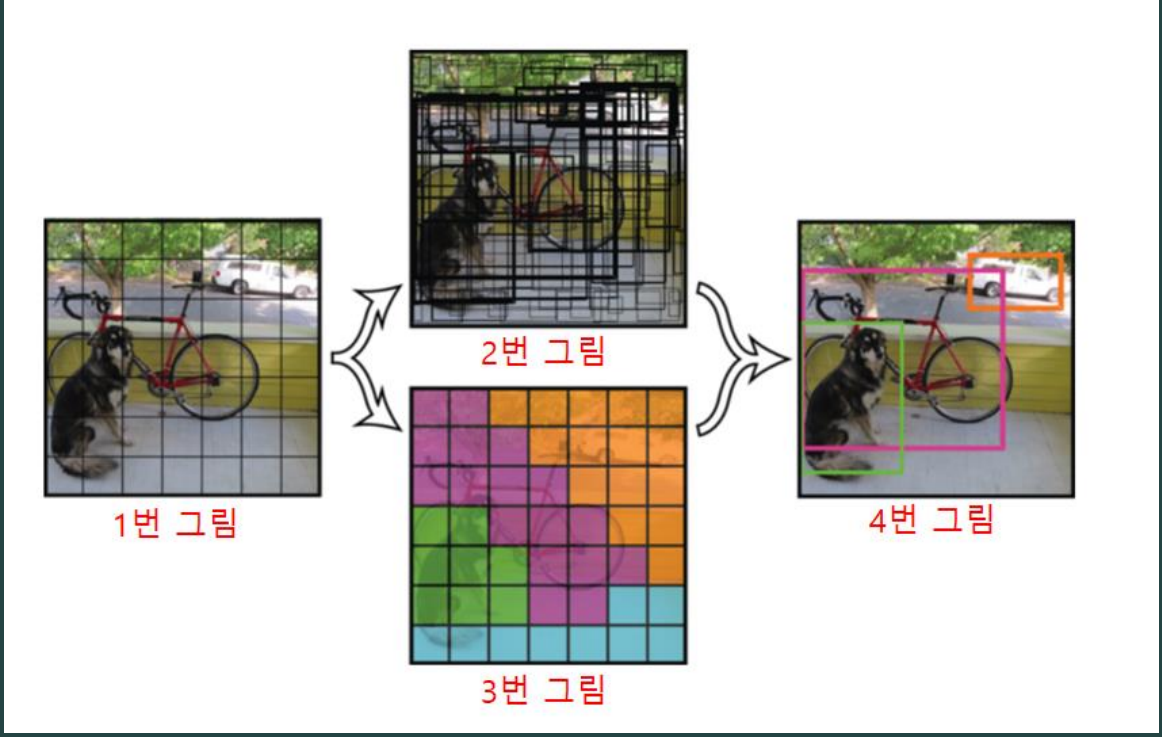
떠든 사람 : 피피티 공유 중

# INDEX

# YOLO (You Only Look Once)모델 원리



# YOLO (You Only Look Once)모델 원리



# YOLO (You Only Look Once)모델 원리

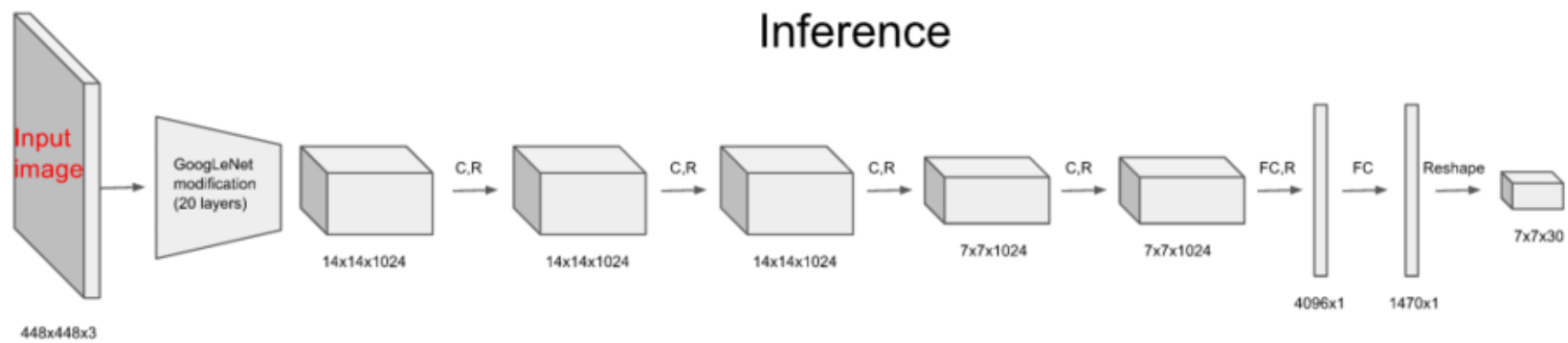
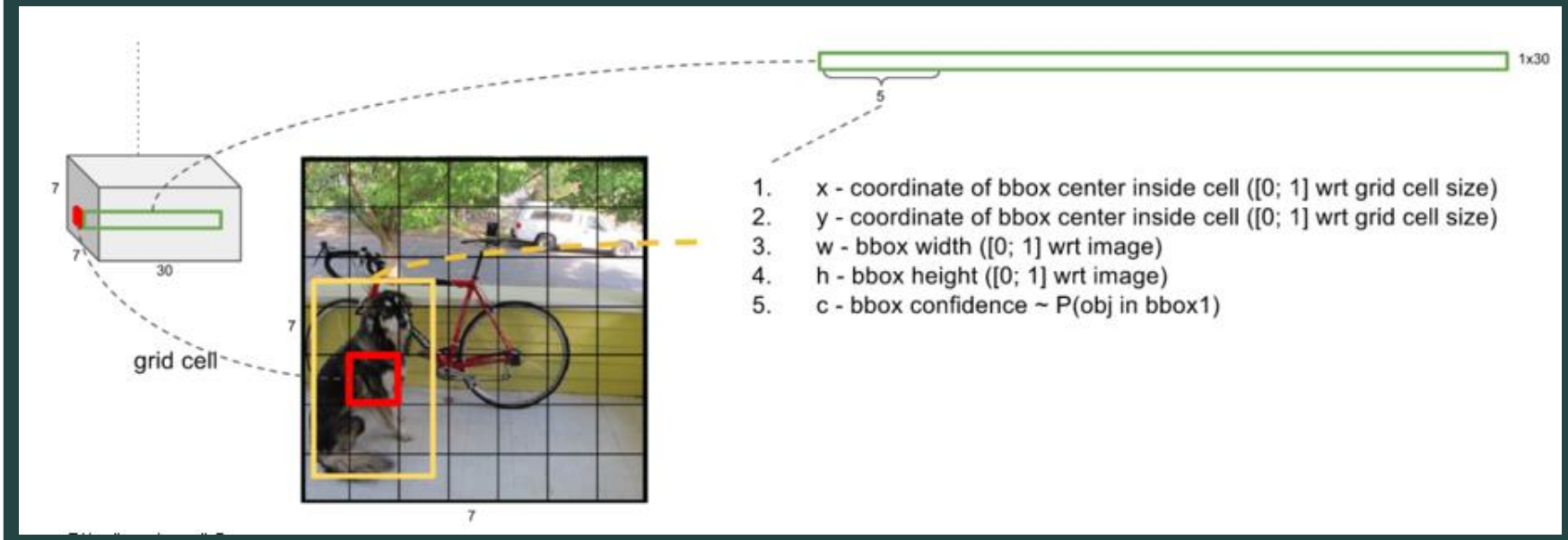
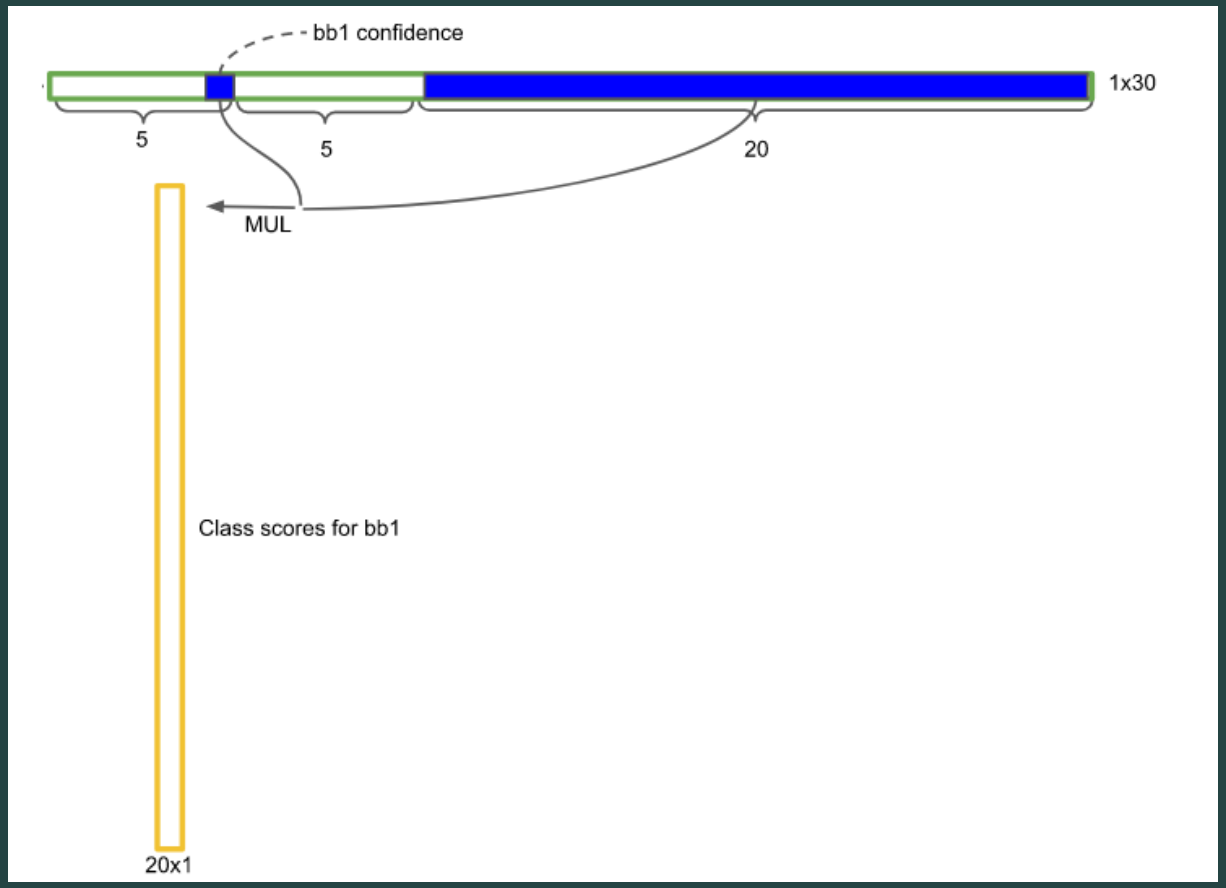



그림) 네티워크 구조

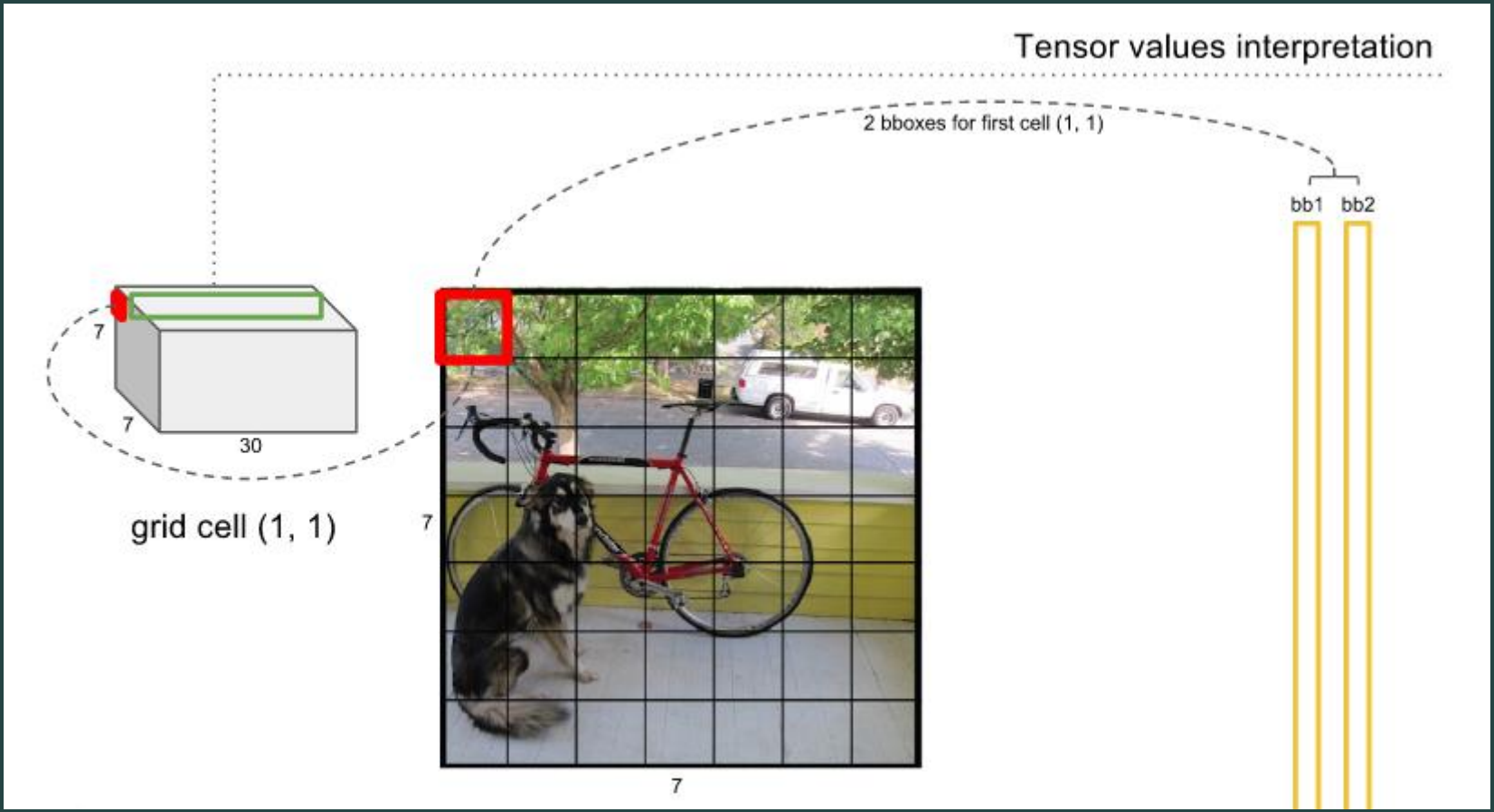
# YOLO (You Only Look Once)모델 원리



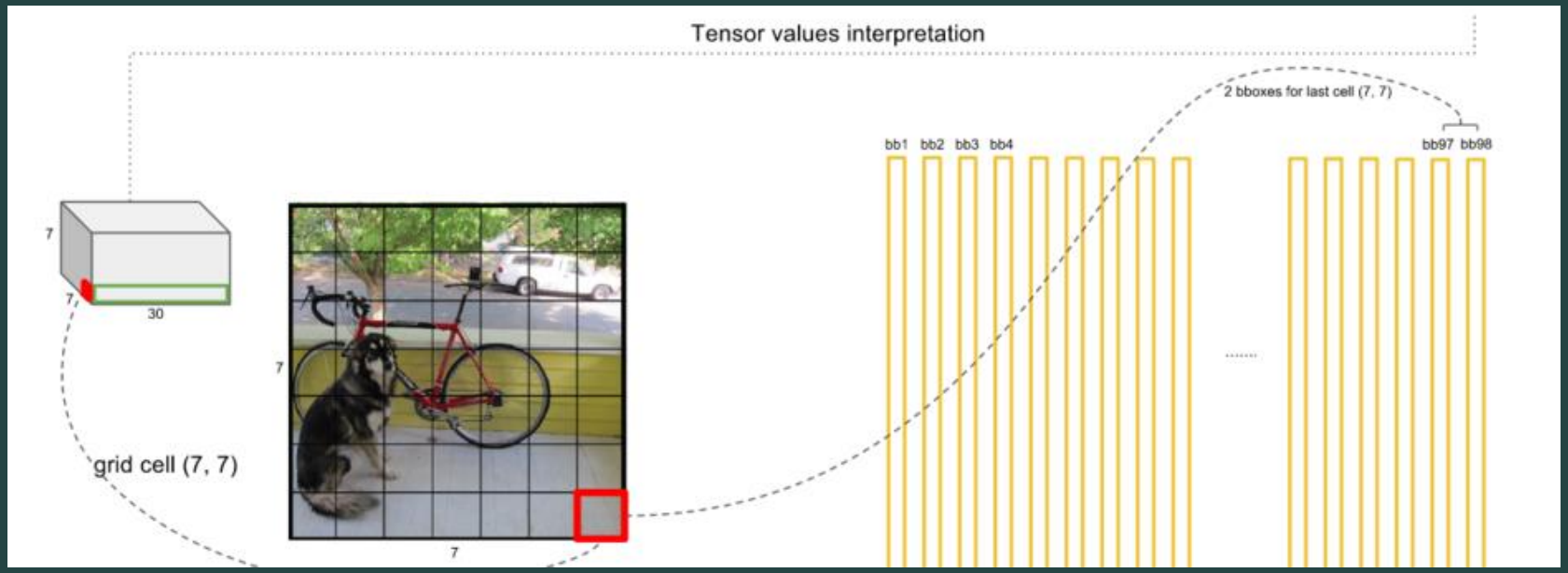
# YOLO (You Only Look Once)모델 원리



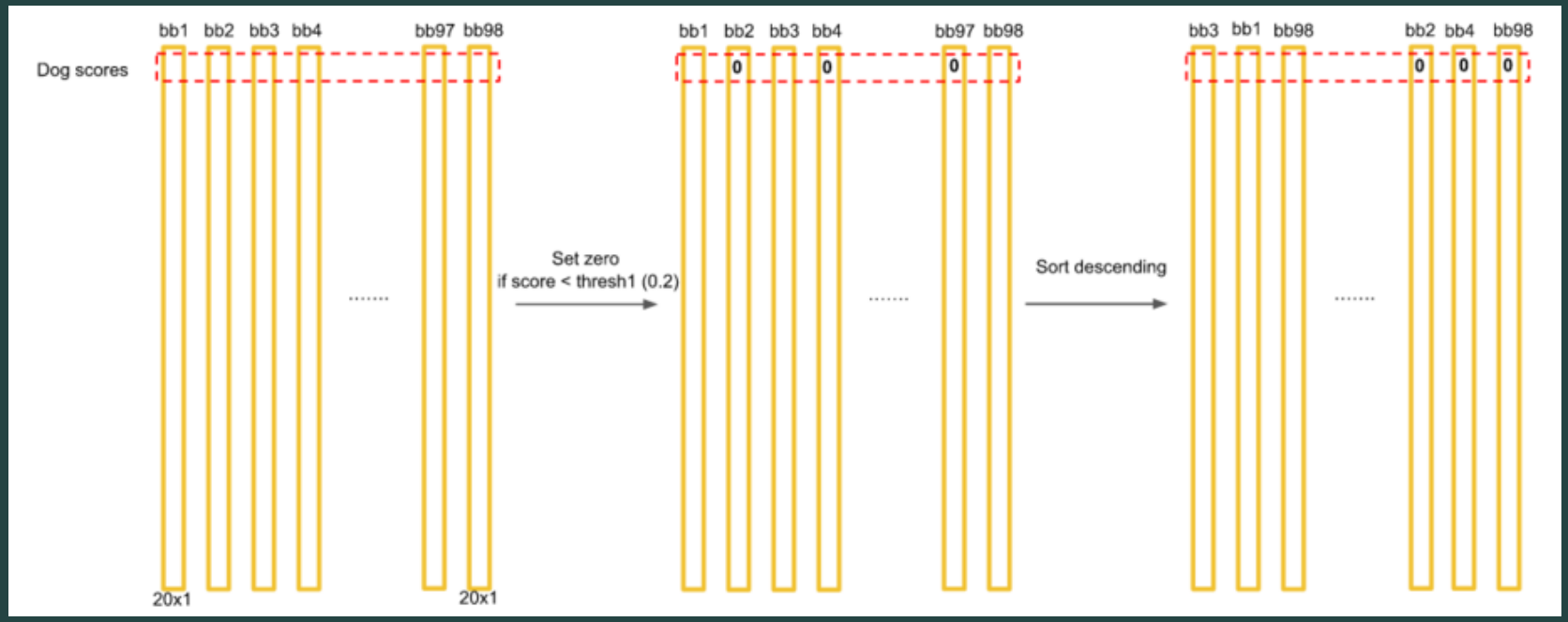
# YOLO (You Only Look Once)모델 원리



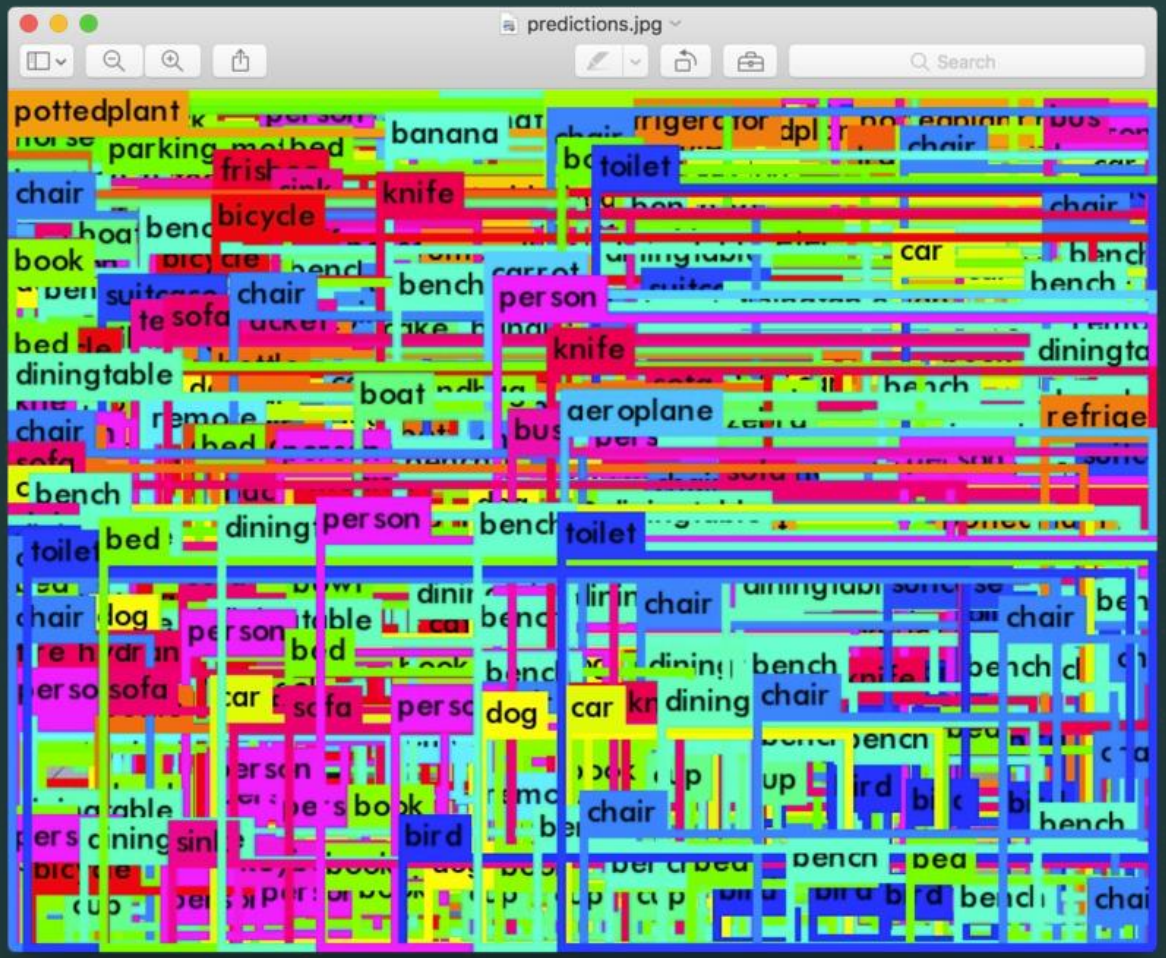
# YOLO (You Only Look Once)모델 원리



# YOLO (You Only Look Once)모델 원리

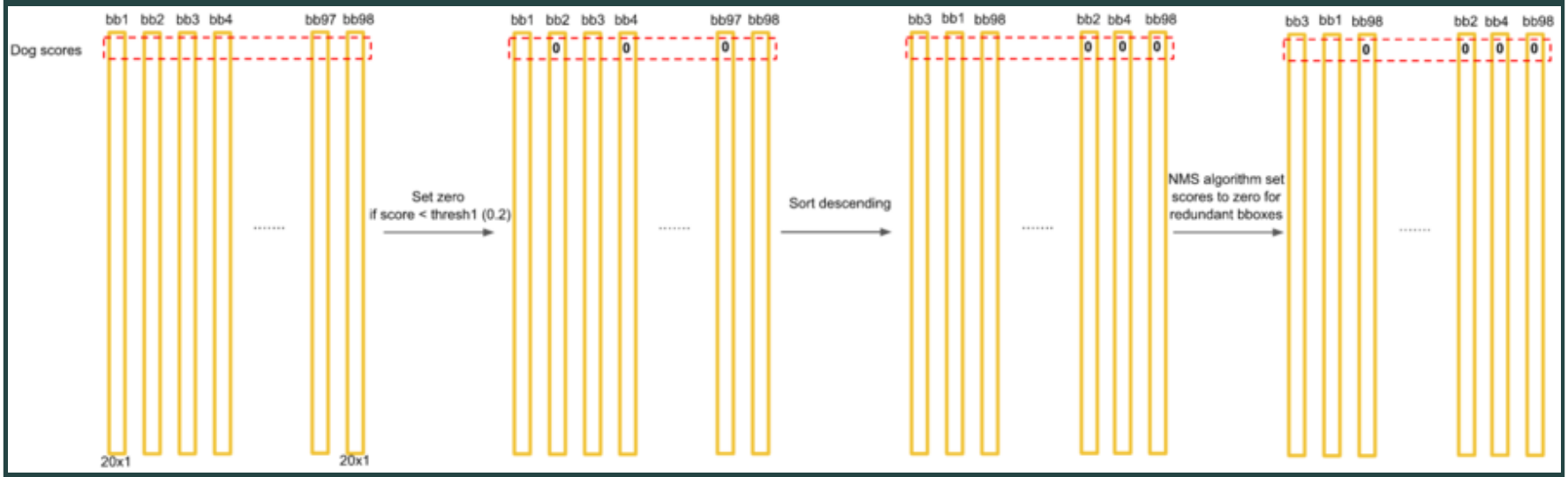


# YOLO (You Only Look Once)모델 원리



# YOLO (You Only Look Once)모델 원리

## NMS(Nom-maxmar-suppression) 비-최대값 억제




# NMS
# (Nom-maxmal-suppression)
# 비-최대값 억제

# YOLO (You Only Look Once)모델 원리

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

coordinate loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2$$

box loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3)$$

class loss

# YOLO (You Only Look Once) 장점

1. 매우 빠르다. Titan X GPU 사용 시 45프레임
FAST YOLO의 경우 150프레임을 기록한다.

2. 기존 R-CNN과 달리 YOLO는 train시간 내내 전체 이미지로 학습하므로
큰 문맥을 볼 수 있다.
즉 백그라운드 문제의 오류를 절반으로 줄인다.

3. 객체의 일반화된 표현을 찾아 학습한다.
따라서 새로운 도메인이나 예기치 않은 입력에 잘못 예측하지 않는다.

# YOLO (You Only Look Once) 단점

1. Faster R-CNN보다 정확도가 떨어진다.

2. 다크넷을 이용한 특징추출을 함으로 불편하다.
다크넷은 c++로 작성되었다.

# YOLO (You Only Look Once) 버전 별 특징

1. YOLOV2 (YOLO 9000)

   1-1. Batch Normalization 적용 : 모든 컨볼루션 레이어에 배치 정규화를 추가

   1-2. High Resolution Classifier : 고해상도 이미지로
   classification network를 먼저 학습시킨 후 Object detection에 fine tuning

   1-3. FCNN : 기존 Fully connected Layer를 전부 Convolution Layer로 대체

2. YOLOV3

   2-1. 소프트맥스를 없애고 모든 클래스에 대해 binary cross entropy loss로 변경

   2-2. Darknet-19에서 Darknet-59로 변경

   2-3. 3개의 스케일에 대해 Prediction을 하여 작은 물체를 탐지 못하는 단점을 보완

# YOLO (You Only Look Once) 버전 별 특징

3. YOLOV4

1-1. Bag of Freebies 기법 도입

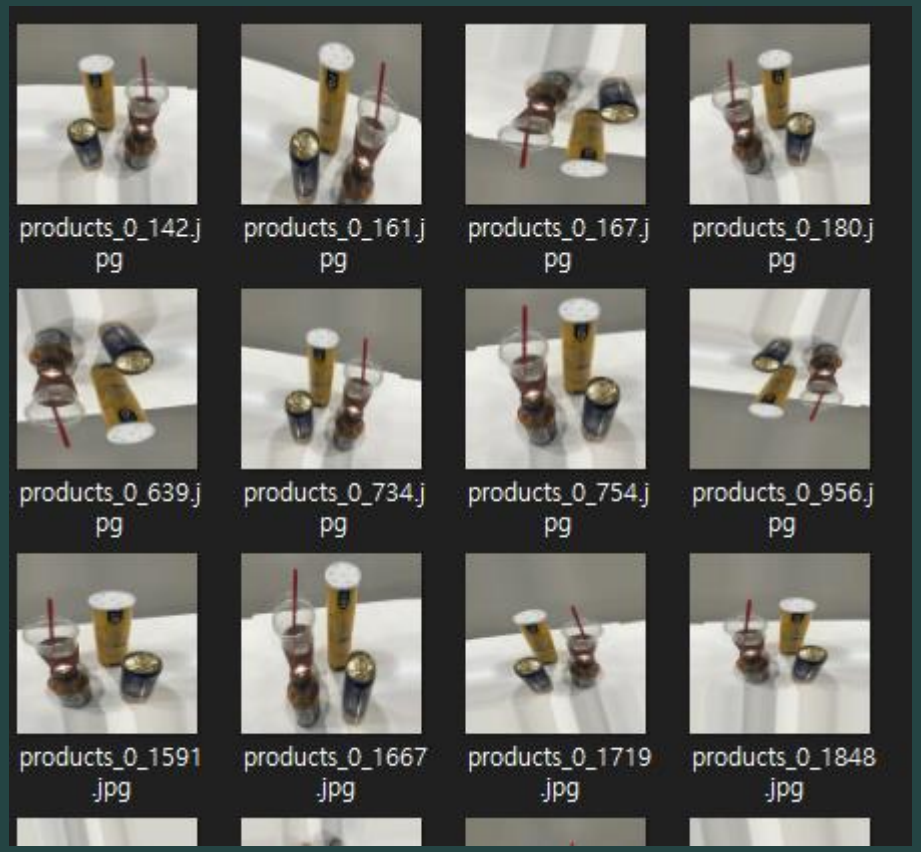
1-1-1. data augmentation : image의 일부 영역에 box를 생성하고 해당 영역을 0 ~ 255의 랜덤한 값으로 채우는 Random erase, 0으로 채우는 Cutout, 두 image와 label을 alpha blendin하는 Mixup, CutOut과 MixUp을 응용한 CutMix, Style-transfer GAN 등의 기법을 사용했다.

4. YOLOV5

1. 기존 YOLOV4에서 Darknet기반 프레임을 PyTorch 기반 프레임으로 변경 이로 인해 사용자 편의성이 증가.

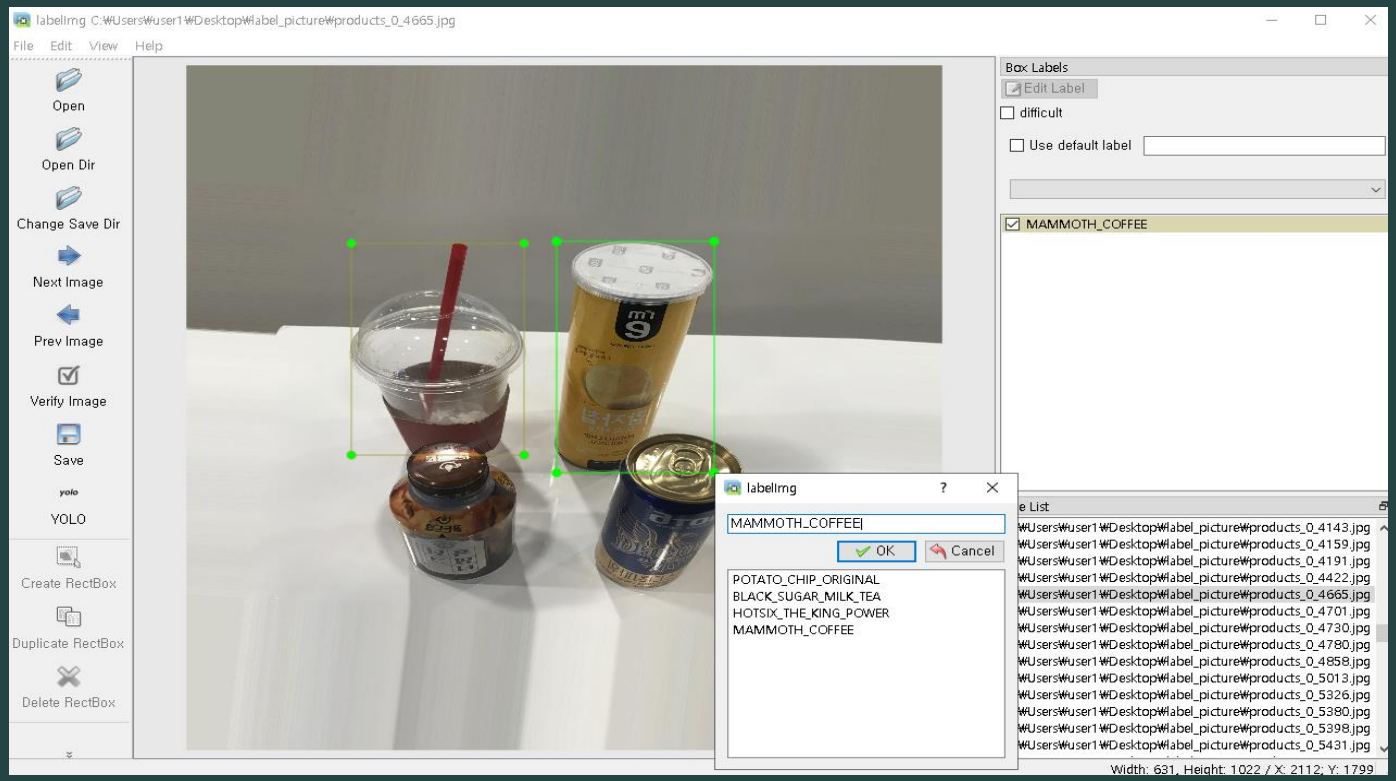
# YOLOV5 학습 데이터

데이터 수를 늘리기 위해 ImageDataGenerator를 사용
총 210개의 데이터셋



# YOLOV5 학습 데이터

LabelImg.py를 이용하여 라벨링 작업을 하였음.
Class, x_val, y_val, w_val, h_val 생성 – yolo포멧



# YOLOV5 학습 데이터

LabelImg.py를 이용하여 라벨링 작업을 하였음.
Class, x_val, y_val, w_val, h_val 생성 – yolo포멧

products_0_142.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
2 0.383929 0.668485 0.199735 0.289352
1 0.689153 0.680390 0.179233 0.252315
3 0.721561 0.381283 0.246693 0.395503
0 0.446098 0.382606 0.210979 0.428571
```

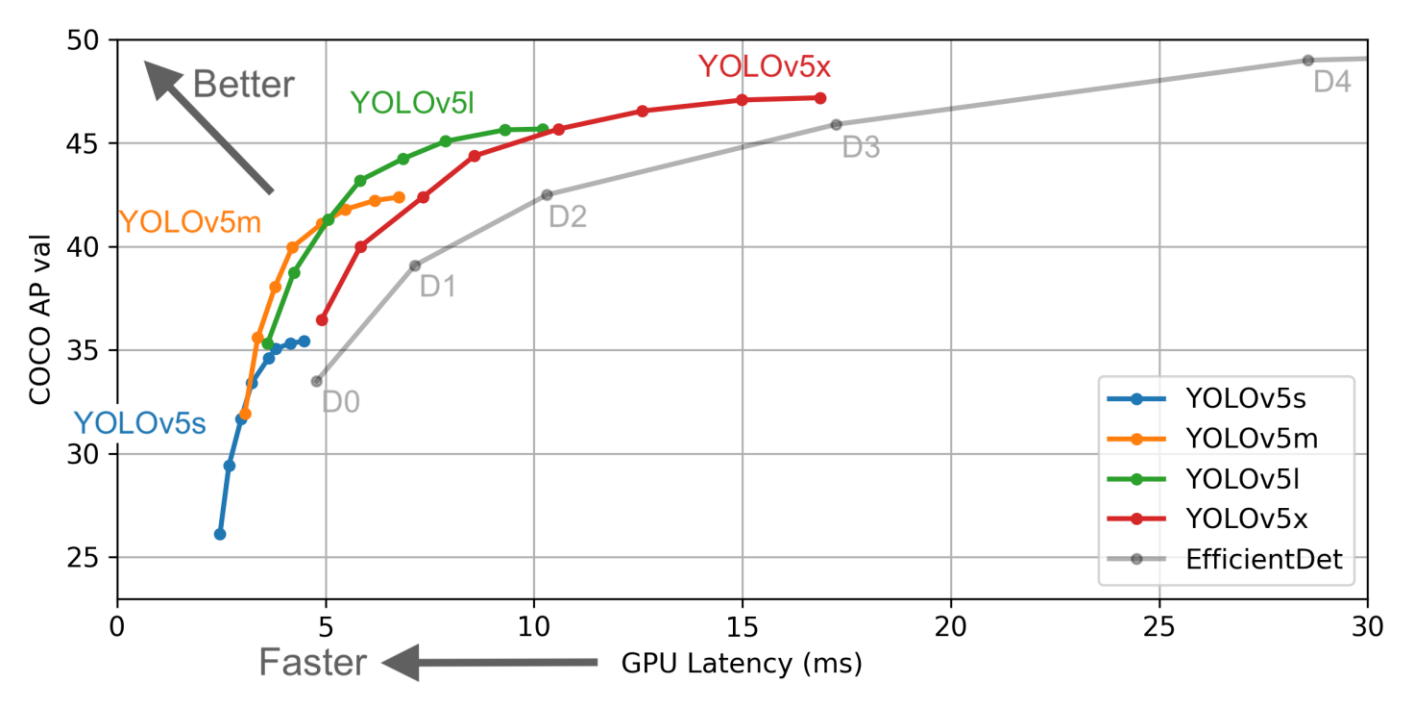
# YOLOV5 Annotation 설정

| Name | Kind | Size |
|---|---|---|
| coco | Folder | -- |
| images | Folder | -- |
| train2017 | Folder | -- |
| val2017 | Folder | -- |
| labels | Folder | -- |
| train2017 | Folder | -- |
| val2017 | Folder | -- |
| README.txt | Plain Text | 831 bytes |
| LICENSE | TextEdit | 35 KB |
| coco128 | Folder | -- |
| yolov5 | Folder | -- |
| data | Folder | -- |
| inference | Folder | -- |
| models | Folder | -- |
| study | Folder | -- |
| utils | Folder | -- |
| weights | Folder | -- |
| tutorial.ipynb | Document | 3.3 MB |
| README.md | Markdo...ument | 8 KB |
| requirements.txt | Plain Text | 871 bytes |
| detect.py | Python Source | 7 KB |
| test.py | Python Source | 12 KB |
| train.py | Python Source | 21 KB |
| Dockerfile | TextEdit | 2 KB |
| LICENSE | TextEdit | 35 KB |

Train = 80%
Val = 20%

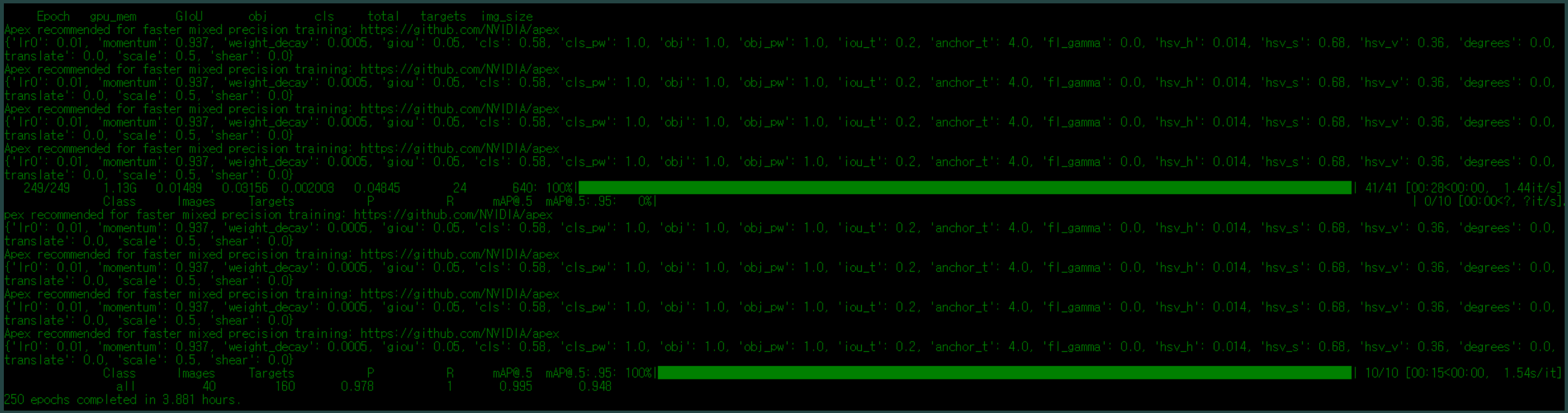
이하 세부 dataset 경로 설정
및
학습사용모델 설정
본 학습에선
YOLOV5S를 사용함

# YOLOV5 모델 별 성능 지표



# YOLOV5 학습 결과

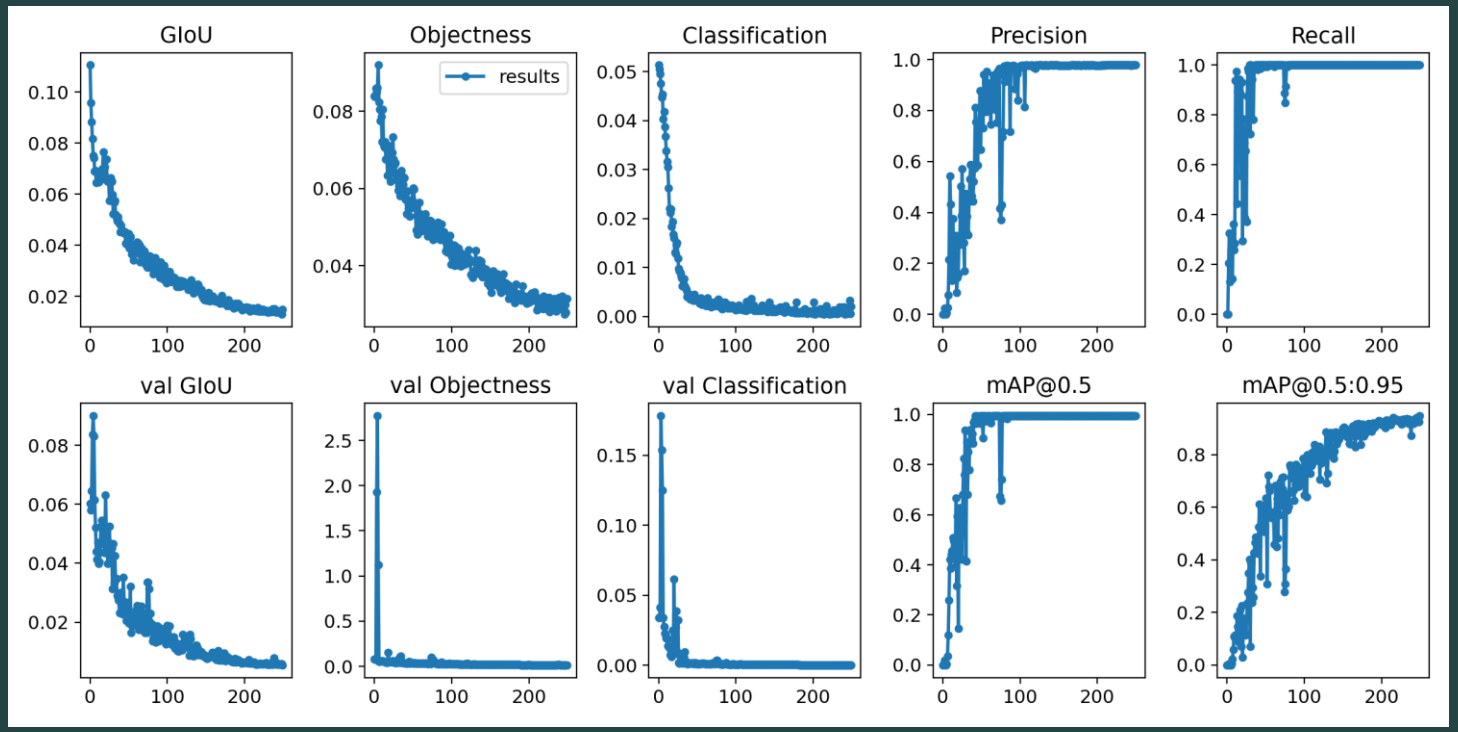
1차학습 – GPU GTX 750Ti 사용 (CPU학습도 가능)
Batch = 4
Epoch = 250



```
     Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
   249/249     1.13G   0.01489   0.03156  0.002003   0.04845        24       640: 100%|██████████████████████████████████████████████████████████████████████████████| 41/41 [00:28<00:00,  1.44it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:   0%|                                                                                          | 0/10 [00:00<?, ?it/s].
pex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
{'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0,
translate': 0.0, 'scale': 0.5, 'shear': 0.0}
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100%|██████████████████████████████████████████████████████████████████████| 10/10 [00:15<00:00,  1.54s/it]
                 all          40         160       0.978           1       0.995       0.948
250 epochs completed in 3.881 hours.
```

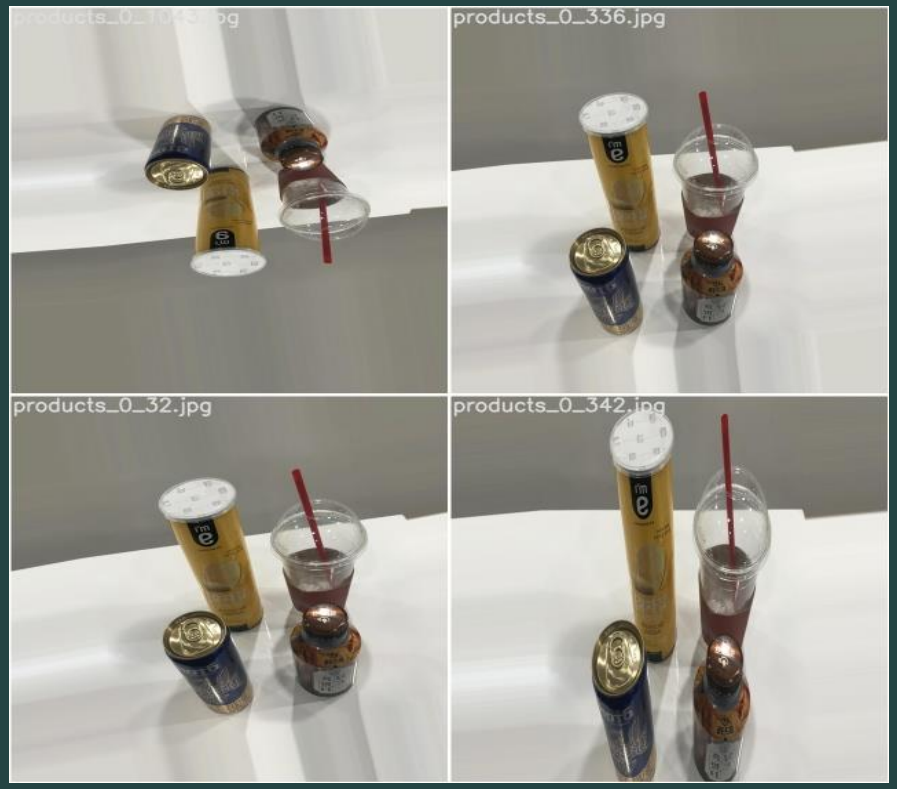
# YOLOV5 학습 결과

1차학습 – GPU GTX 750Ti 사용 (CPU학습도 가능)



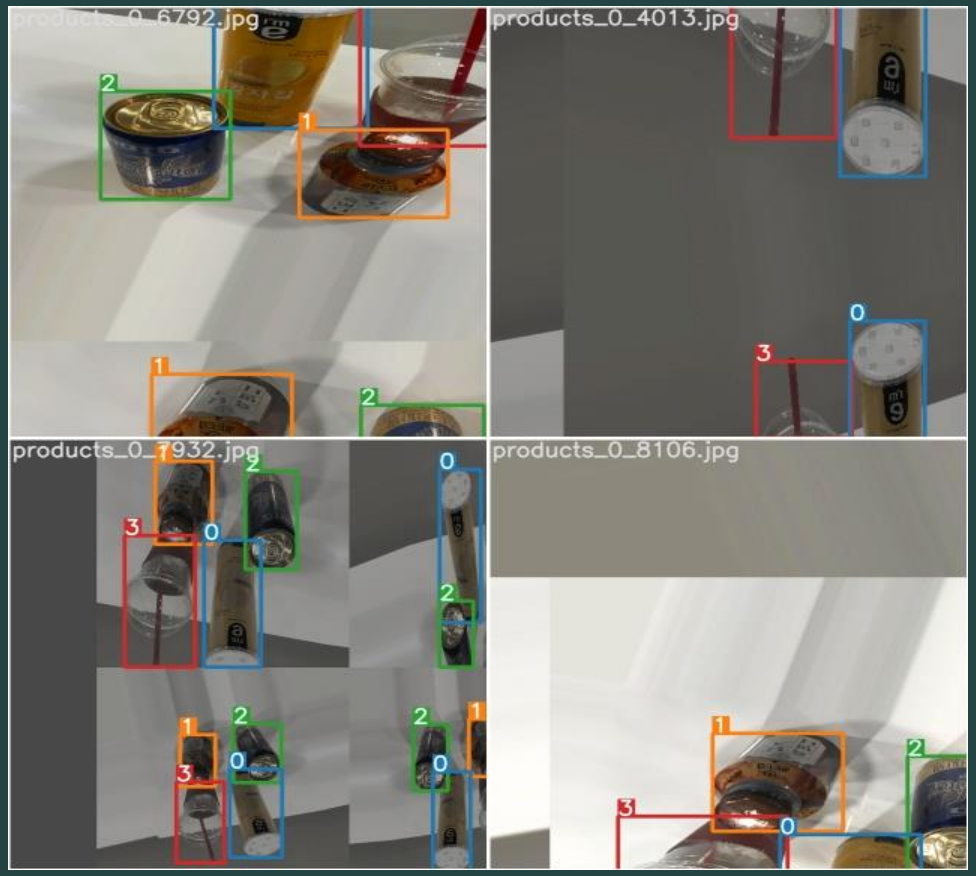
# YOLOV5 학습 결과

1차 학습 epoch-batch 별 학습과정 – epoch = 1



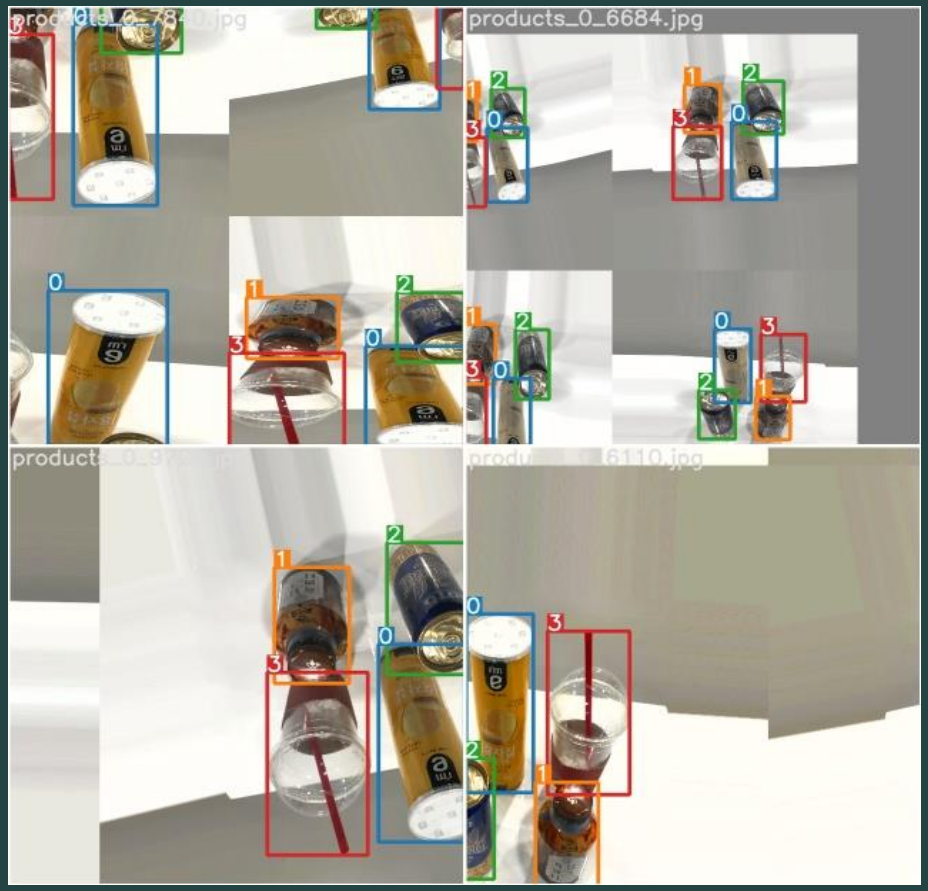
# YOLOV5 학습 결과

1차 학습 epoch-batch 별 학습과정 – epoch = 2



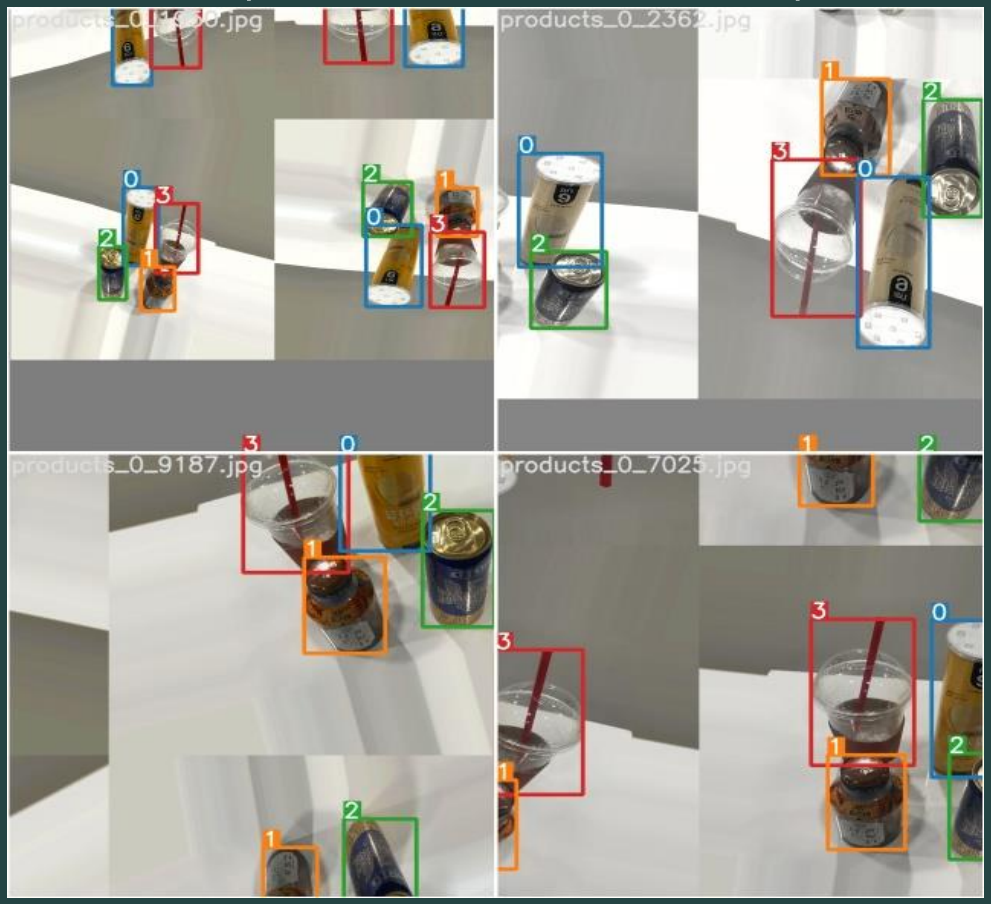
# YOLOV5 학습 결과

1차 학습 epoch-batch 별 학습과정 – epoch = 3



# YOLOV5 학습 결과

1차 학습 epoch-batch 별 학습과정 – epoch = 4



# 동영상

detect2.py
Dockerfile
hubconf.py
labels.png
last.pt
LICENSE
README.md
requirements.txt
results.png
results.txt
test.py
test_batch0_gt.jpg
test_batch0_pred.jpg
train.py
train_batch0.jpg
train_batch1.jpg
train_batch2.jpg

```python
    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('--classes', nargs='+', type=int, help='filter by class')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    opt = parser.parse_args()
    opt.img_size = check_img_size(opt.img_size)
    print(opt)

    with torch.no_grad():
        detect()
```



if __name__ == '__main__'

detect

```
0: 512x640 Done. (0.441s)
0: 512x640 Done. (0.369s)
0: 512x640 Done. (0.405s)
0: 512x640 Done. (0.425s)
0: 512x640 Done. (0.374s)
0: 512x640 Done. (0.396s)
0: 512x640 Done. (0.451s)
0: 512x640 Done. (0.491s)
0: 512x640 Done. (0.399s)
0: 512x640 Done. (0.387s)
0: 512x640 Done. (0.399s)
0: 512x640 Done. (0.427s)

Process finished with exit code -1
```

# 동영상



```
calculator.py
detect.py
detect2.py
Dockerfile
hubconf.py
labels.png
last.pt
LICENSE
README.md
requirements.txt
results.png
results.txt
test.py
test_batch0_gt.jpg
test_batch0_pred.jpg
train.py
train_batch0.jpg
train_batch1.jpg
train_batch2.jpg
```

```
136            if save_img:
137                if dataset.mode == 'images':
138                    cv2.imwrite(save_path, im0)
139                else:
140                    if vid_path != save_path:  # new video
141                        vid_path = save_path
142                        if isinstance(vid_writer, cv2.VideoWriter):
143                            vid_writer.release()  # release previous video writer
144
145                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
146                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
147                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
148                        vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*opt.fourcc), fps, (w, h))
149                    vid_writer.write(im0)
150
151    if save_txt or save_img:
152        print('Results saved to %s' % os.getcwd() + os.sep + out)
153        if platform == 'darwin':  # MacOS
```

detect() › for path, img, im0s, vid_cap in... › for i, det in enumerate(pred) › if view_img › if cv2.waitKey(300) == ord('q') › for target in classification

detect2

```
0: 512x640 1 POTATO_CHIP_ORIGINALs, 1 BLACK_SUGAR_MILK_TEAs, 1 HOTSIX_THE_KING_POWERs, Done. (0.424s)
0: 512x640 1 HOTSIX_THE_KING_POWERs, Done. (0.381s)
0: 512x640 1 HOTSIX_THE_KING_POWERs, Done. (0.408s)
0: 512x640 1 POTATO_CHIP_ORIGINALs, 1 BLACK_SUGAR_MILK_TEAs, 1 HOTSIX_THE_KING_POWERs, Done. (0.391s)
0: 512x640 1 POTATO_CHIP_ORIGINALs, 1 BLACK_SUGAR_MILK_TEAs, 1 HOTSIX_THE_KING_POWERs, Done. (0.407s)
0: 512x640 1 POTATO_CHIP_ORIGINALs, 1 BLACK_SUGAR_MILK_TEAs, 1 HOTSIX_THE_KING_POWERs, Done. (0.372s)
Traceback (most recent call last):
  File "C:/Users/NohTaeHyun/Desktop/R_DATA_PROJECT/yolov5/detect2.py", line 180, in <module>
    detect()
  File "C:/Users/NohTaeHyun/Desktop/R_DATA_PROJECT/yolov5/detect2.py", line 131, in detect
    print("합계는 " + total +"원 입니다.")
TypeError: can only concatenate str (not "int") to str
```

# Thank You

감사합니다.