

# Pytorch Computer Vision Cookbook

2021210088 허지혜

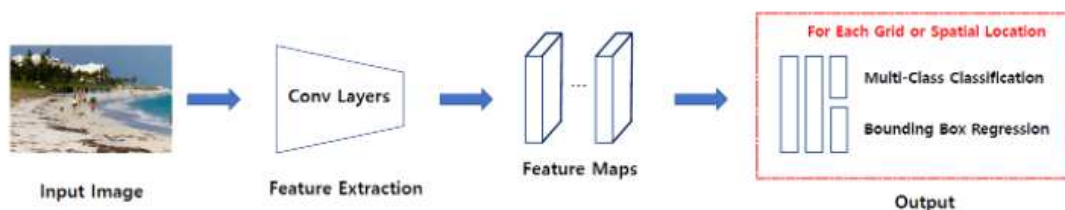
## Chapter5. Multi-Object Detection

Object Detection은 이미지에서 존재하는 물체를 찾고 분류를 하는 과정이다. 식별된 Object는 이미지 안에서 bounding box로 보여진다. 위 분야에서는 크게 두 가지 방법에 의해 Object Detection을 수행할 수 있다.

<https://dacon.io/forum/405806?page=1&dtype=recent>

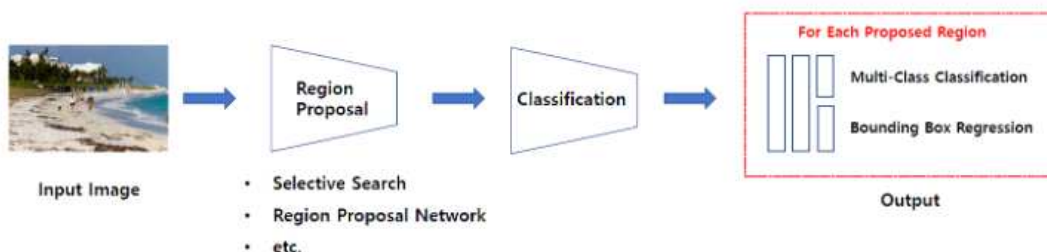
- **One Stage Detector** : regression/classification-based

ex) YOLO, SSD 모형 등



- **Two Stage Detector** : region proposal-based

ex) R-CNN 계열 모형 등



이번 장에서는, One stage Detector의 대표 주자인 YOLO 모형에 대하여 공부를 할 것이다. YOLO는 You Once Look Once의 약자로 현재 v5까지 모형이 나와있다. 여기서 YOLOv3는 이전 버전인 YOLO, YOLO9000보다 빠르고 정확도가 좋다. YOLOv3를 Pytorch를 가지고 구현을 해보자.

### 1. Create dataset

YOLOv3를 실습할 데이터는 COCO 데이터이다. COCO dataset은 Object Detection, Segmentation, Keypoint Detection등의 Computer Vision 분야의 task를 목적으로 만들어진 dataset이다.

COCO 데이터셋 다운로드 : <https://cocodataset.org/#home>

우리가 사용할 데이터는 COCO2014 데이터이다. COCO2014 데이터의 구성은 다음과 같다.

- COCO\_TRAIN2014 : 82,783장
- COCO\_VAL2014 : 40,504장

공식 홈페이지에서 데이터를 다운받아도 되지만 시간이 오래 걸리므로 데이터 다운 방법에 의거하여 다운을 받았다. (데이터 다운 방법.hwp 파일 확인)

후에 모형 가중치를 위해서 darknet 폴더도 함께 다운 받는다.

다운 받은 폴더는 다음과 같다.

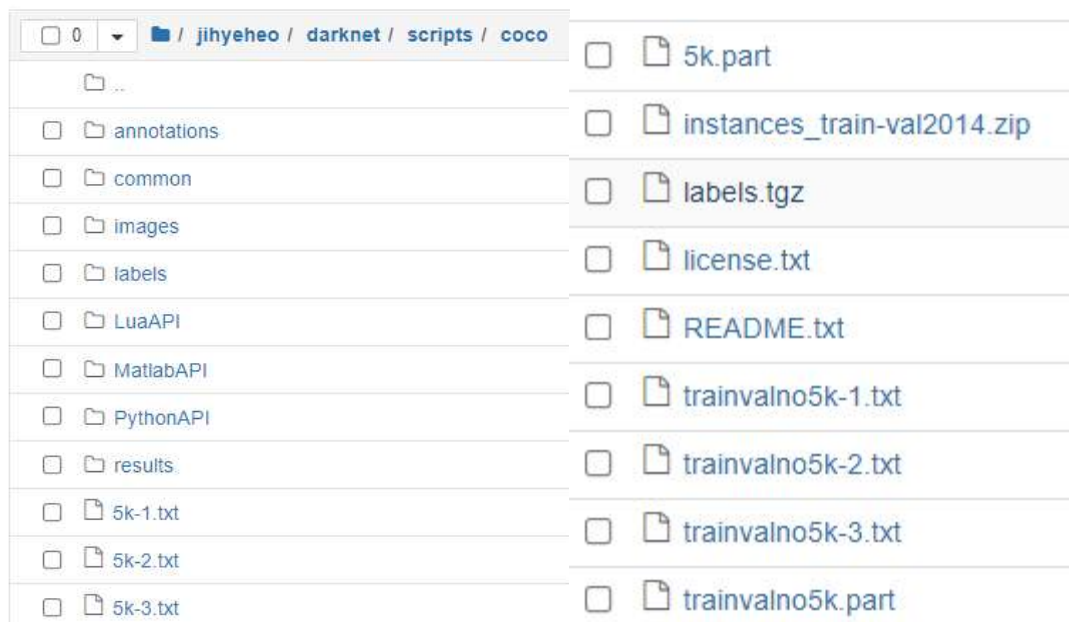


그림. darknet 폴더 안에 다운받은 coco

그림. darknet 폴더 안에 다운받은 coco

다운 받은 COCO 데이터 파일의 일부를 가져와서 labels를 살펴보면 다음과 같다.

```
*COCO_val2014_0000000000042 - Windows 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)
16 0.606688 0.341381 0.544156 0.510000
```

그림. coco\_val2014\_42의 좌표값 및 class

첫 번째 숫자는 object의 ID이고 네 가지 숫자인 [xc, yc, w, h]는 각각 다음을 나타낸다.

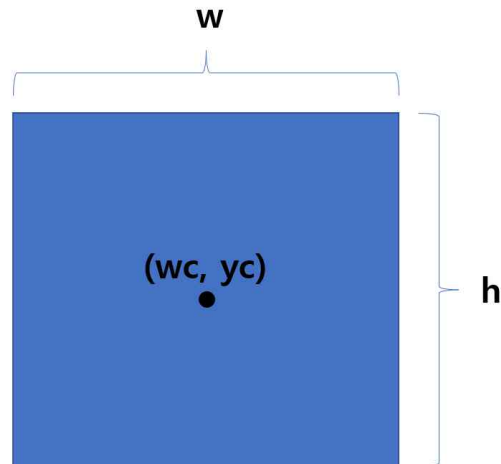


그림. Bounding Box

- $(w_c, y_c)$  : bounding box 좌표값의 중심값
- $(w, h)$  : bounding box의 높이와 너비

darknet 폴더 안에 다운받은 coco를 살펴보면 5k-3.txt 파일과 trainvalno5k-3.txt 파일이 있다. 각각의 파일은 image 경로가 담긴 list 목록이다.

- trainvalno5k-3.txt : 모델을 train 하기 위해 필요한 image 경로 list이다. 여기에는 train2014와 val2014를 모두 포함시킨다. 5000개의 image를 제외하고.
- 5k-3.txt : 제외시킨 5000개의 validation image

마지막으로, COCO dataset의 label을 다운로드 받으면 데이터 준비는 끝이다. COCO dataset의 label은 80개의 class로 나뉘는데 나중에 다시 얘기해 볼 것이다.

## Creating a custom COCO dataset

위 단계에서는 앞서 다운받은 COCO dataset에 대하여, training과 validation dataset을 만들고 Dataloaders로 정의한 후 몇 개의 샘플 이미지를 살펴볼 것이다.

- 패키지 불러오기

```
from torch.utils.data import Dataset
import torchvision.transforms.functional as TF
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw, ImageFont
from torchvision.transforms.functional import to_pil_image
import random
%matplotlib inline

import torch
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(torch.__version__)
```

1.9.0+cu11

- CocoDataset class 정의하기

\_\_init\_\_ function에서는 3가지 input가 필요하다.

- path2listFile : image 목록을 포함하는 text 파일의 위치를 나타내는 문자열
- transform : resizing, tensor 바꾸기 등의 다양한 transformations
- trans\_params : target image size와 같은 transformation 파라미터를

정의하는 파이썬 dictionary

```
class CocoDataset(Dataset):
    def __init__(self, path2listFile, transform=None, trans_params=None):
        # path2listFile을 r(read)로 읽어온다.
        with open(path2listFile, "r") as file:
            self.path2imgs = file.readlines() # 줄마다 읽어온다.

        # label 리스트를 추출한다.
        # 각각의 path들을 images -> labels , png.jpg -> txt change
        self.path2labels = [path.replace("images", "labels").replace(".png", ".txt").
            replace(".jpg", ".txt") for path in self.path2imgs]

        self.trans_params = trans_params
        self.transform = transform
```

```

def __len__(self):
    return len(self.path2imgs)

def __getitem__(self, index):
    # rstrip() : 오른쪽 공백 제거
    # index % len(self.path2imgs) => 나눈 수의 나머지
    path2img = self.path2imgs[index % len(self.path2imgs)].rstrip()
    # 이미지를 RGB로 바꾼다.
    img = Image.open(path2img).convert('RGB')
    path2label = self.path2labels[index % len(self.path2imgs)].rstrip()

    labels= None
    if os.path.exists(path2label):
        # labels 파일 값은 [objectID, wc, yc, w, h]로 구성되어 있다.
        labels = np.loadtxt(path2label).reshape(-1, 5)

    if self.transform:
        img, labels = self.transform(img, labels, self.trans_params)

    return img, labels, path2img

```

CocoDataset을 class를 가지고 img, labels, path2img를 얻는다.  
이를 예시를 통해 더 자세히 알아보자.

- training 데이터에 대해 CocoDataset 만들어주기

```

root_data="./data/coco"
path2trainList=os.path.join(root_data, "trainvalno5k-3.txt")
coco_train = CocoDataset(path2trainList)
print(len(coco_train))

```

```
117264
```

**\*질문\***

trainvalno5k-3.txt 파일 같은 경우 train 데이터(82783개)와 validation 데이터(5000개 제외, 35504개)의 목록이 나와야 하는데 왜 117264개가 나오는지 궁금합니다.

CocoDataset class가 돌아가는 프로세스를 training 데이터에 대하여 설명하면 다음과 같다.

1) trainvalno5k.txt는 다음과 같이 그림 파일의 경로가 '\t'로 분리되어 있다.

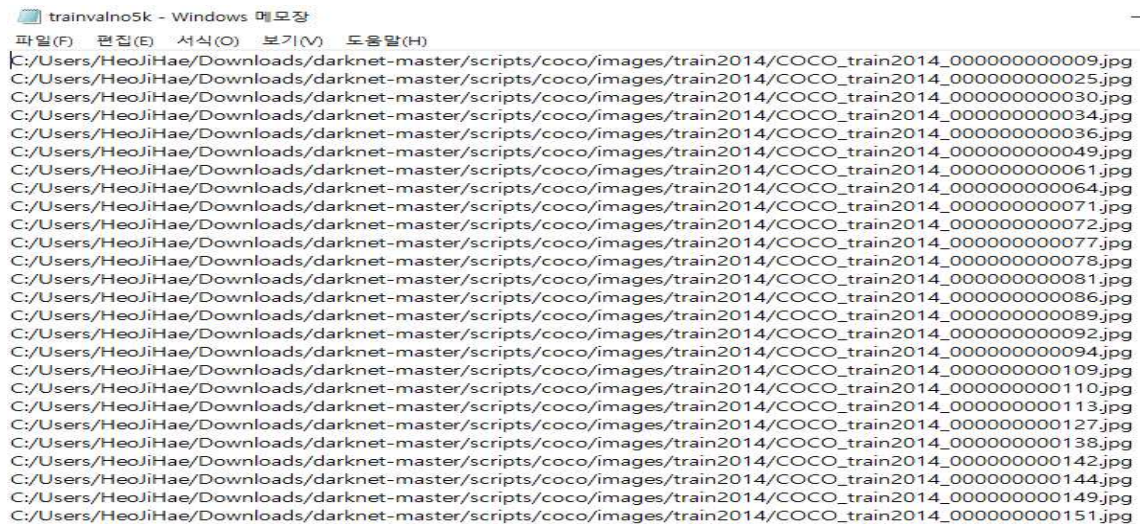


그림. trainvalno5k.txt 파일 내용

2) 위 파일의 내용을

C:/Users/HeoJiHae/Downloads/darknet-master/scripts/coco/images/train2014/COCO\_train2014\_000000000009.jpg

C:/Users/HeoJiHae/Downloads/darknet-master/scripts/coco/images/train2014/COCO\_train2014\_000000000025.jpg

한 줄씩 불러온 후, 위 경로의 images를 labels로, jpg.png를 txt로 변경한다.

C:/Users/HeoJiHae/Downloads/darknet-master/scripts/coco/labels/train2014/COCO\_train2014\_000000000009.txt

C:/Users/HeoJiHae/Downloads/darknet-master/scripts/coco/labels/train2014/COCO\_train2014\_000000000025.txt

3) 만약에 txt 파일이 존재하면 읽어와서 reshape 해라.

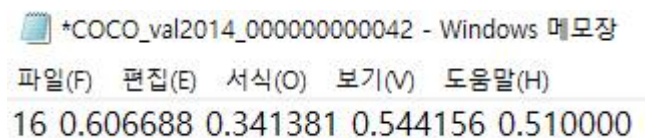


그림. coco\_val2014\_42의 좌표값 및 class

이를 16

0.606688

0.341381

0.544156

0.510000

이렇게 reshape 하라는 의미이다.

- coco\_train 예제

```
img, labels, path2img = coco_train[1] # 숫자 부분을 바꾸어 데이터를 변경
print("image size:", img.size, type(img))
print("labels shape:", labels.shape, type(labels))
print("labels \n", labels)
```

```
image size: (640, 426) <class 'PIL.Image.Image'>
labels shape: (2, 5) <class 'numpy.ndarray'>
labels
[[23.      0.770336  0.489695  0.335891  0.697559]
 [23.      0.185977  0.901608  0.206297  0.129554]]
```

반복 작업을 validation 데이터에도 해주겠다.

- validation 데이터에 대한 Cocodataset 만들어주기

```
path2valList=os.path.join(root_data, "5k.txt")
coco_val = CocoDataset(path2valList, transform=None, trans_params=None)
print(len(coco_val))
```

5000

- coco\_val 예제

```
img, labels, path2img = coco_val[7]
print("image size:", img.size, type(img))
print("labels shape:", labels.shape, type(labels))
print("labels \n", labels)
```

```
image size: (640, 427) <class 'PIL.Image.Image'>
labels shape: (3, 5) <class 'numpy.ndarray'>
labels
[[20.      0.539742  0.521429  0.758641  0.957143]
 [20.      0.403469  0.470714  0.641656  0.695948]
 [20.      0.853039  0.493279  0.293922  0.982061]]
```

- COCO 데이터 label

```
path2cocoNames="./data/coco.names"
fp = open(path2cocoNames, "r")
coco_names = fp.read().split("\n")[:-1] # 뒤에 공백이 하나 있어서 빼준다.
print("number of classese:", len(coco_names))
```

```
print(coco_names)
```

```
number of classes: 80
```

```
['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat',  
'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat',  
'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack',  
'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',  
'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',  
'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',  
'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake',  
'chair', 'sofa', 'pottedplant', 'bed', 'diningtable', 'toilet', 'tvmonitor', 'laptop',  
'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink',  
'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier',  
'toothbrush']
```



```
coco.names - Windows 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
person  
bicycle  
car  
motorbike  
aeroplane  
bus  
train  
truck  
boat  
traffic light  
fire hydrant  
stop sign  
parking meter  
bench  
bird  
cat  
dog  
horse  
sheep  
cow  
elephant  
bear  
zebra  
giraffe  
backpack
```

그림. coco.names 파일 안에 있는 label 목록

COCO 데이터셋은 80가지의 Object로 이루어져 있다.

우리는 위 coco.names를 사용하여 Object 이름을 표시하는데 사용할 것이다.

- coco\_train과 coco\_val 데이터 일부 시각화 전, 시각화 bbox 정의

```
1) 정규화된 bounding box의 scale을 변경하여 원래 이미지 크기로 돌리는 함수 정의  
def rescale_bbox(bb,W,H):
```



```
x,y,w,h=bb
return [x*W, y*H, w*W, h*H]
```

```
# 랜덤 컬러 생성 : [0 , 255) 범위에서 (80,3) array 생성
COLORS = np.random.randint(0, 255, size=(80, 3),dtype="uint8")
```

```
# 이미지 폰트 지정, 기존에 책에 나와있는 글꼴은 오류가 뜬다.
fnt = ImageFont.truetype('/root/pyscripts/arial.ttf', 16)
```

2) image에 bounding box 표기 함수 정의

```
# img = (3*H*W)
```

```
# targets = bounding box 좌표
```

```
def show_img_bbox(img,targets):
    if torch.is_tensor(img):
        img=to_pil_image(img)
    if torch.is_tensor(targets):
        targets=targets.numpy()[:,1:]
```

```
W, H=img.size
draw = ImageDraw.Draw(img)
```

```
for tg in targets:
    id_=int(tg[0]) # 제일 처음 숫자
    bbox=tg[1:] # 뒤에 4가지 숫자
    bbox=rescale_bbox(bbox,W,H)
    xc,yc,w,h=bbox
```

```
color = [int(c) for c in COLORS[id_]]
name=coco_names[id_]
```

```
# color는 random tuples의 배열 형태
draw.rectangle(((xc-w/2, yc-h/2), (xc+w/2, yc+h/2)),outline=tuple(color)
,width=3)
```

```
# 왼쪽 위에 text로 물체 종류 표기
# (255,255,255,0)=(red, green, blue, alpha) alpha는 투명도(0~1)
draw.text((xc-w/2,yc-h/2),name, font=fnt, fill=(255,255,255,0))
```

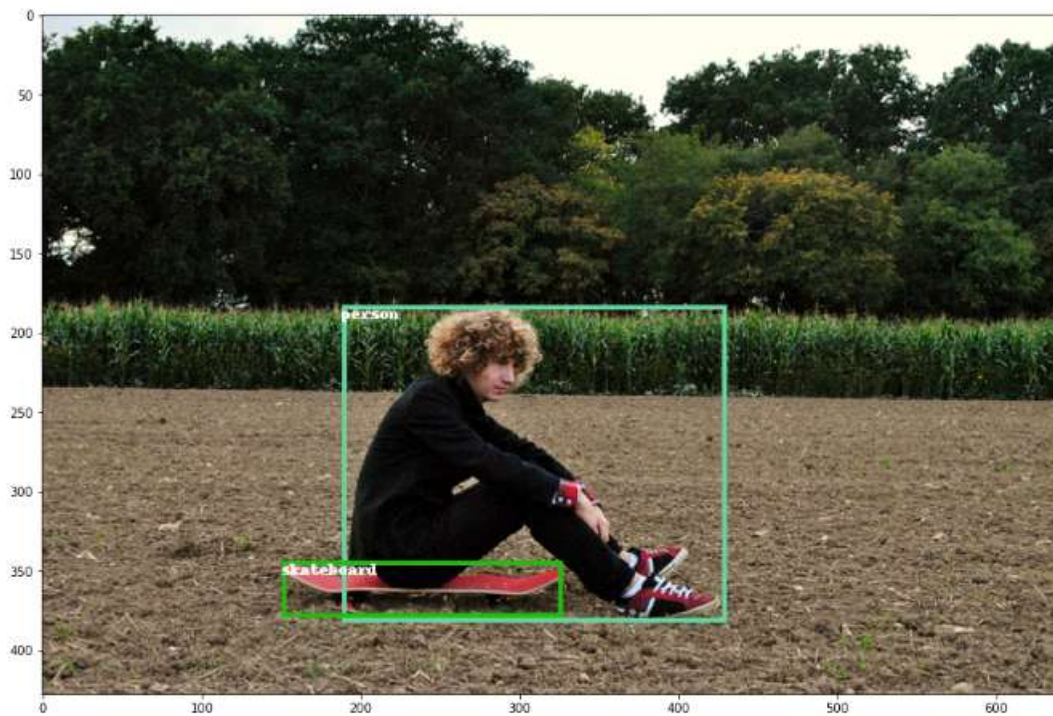
```
plt.imshow(np.array(img))
```

- coco\_train 예시 시각화

```
np.random.seed(2)
rnd_ind=np.random.randint(len(coco_train))
img, labels, path2img = coco_train[rnd_ind]
print(img.size, labels.shape)

plt.rcParams['figure.figsize'] = (20, 10)
show_img_bbox(img,labels)
```

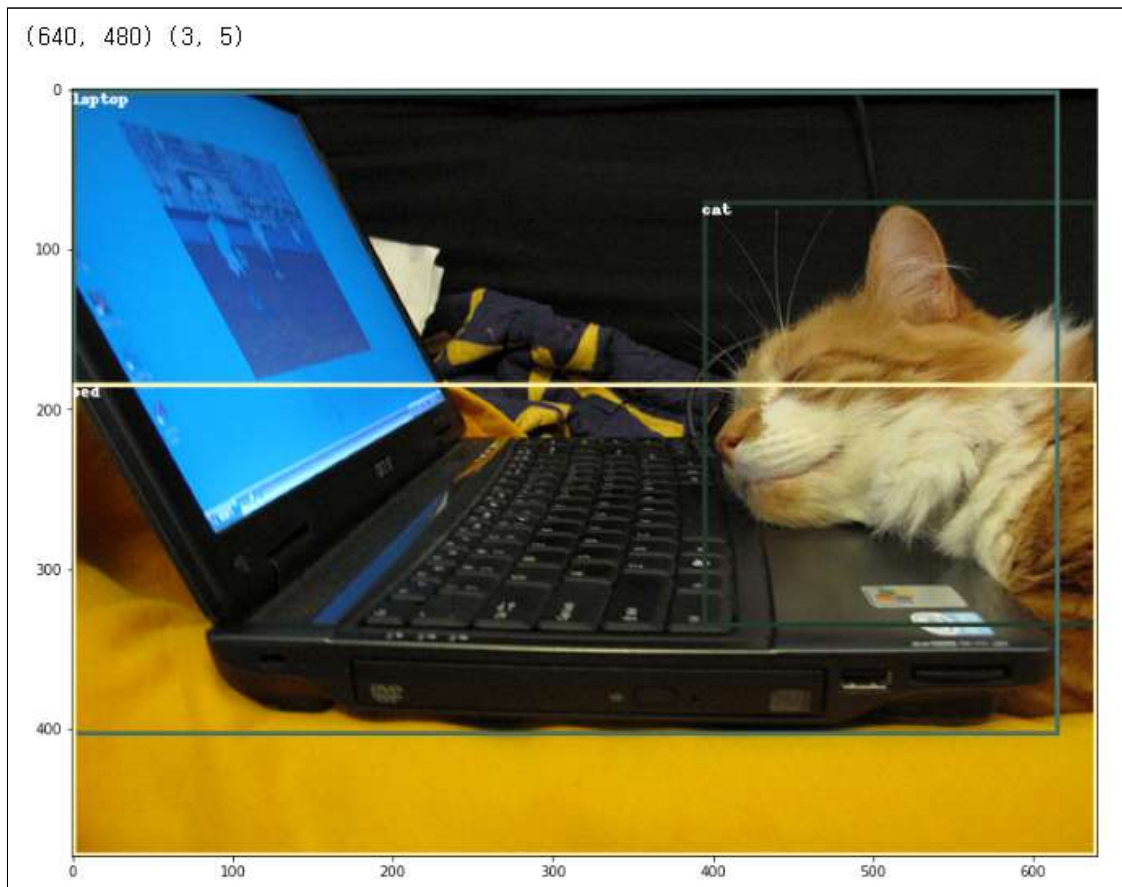
(640, 428) (2, 5)



- coco\_val

```
np.random.seed(0)
rnd_ind=np.random.randint(len(coco_val))
img, labels, path2img = coco_val[rnd_ind]
print(img.size, labels.shape)

plt.rcParams['figure.figsize'] = (20, 10)
show_img_bbox(img,labels)
```



## Transforming the Data

요 단원에서는 Single-Object Detection에서도 했던 transform 함수를 정의하고 파라미터를 정의해볼 것이다. Pytorch에서는 image를 loading 할 때 torchvision.transform 함수를 사용하여 한 번에 처리를 할 수도 있고 위 단원처럼 class로 정의하고 함수로 정의하여 처리할 수도 있다.

```
class CocoDataset(Dataset):
    .....
    labels= None
    if os.path.exists(path2label):
        # labels 파일 값은 [objectID, wc, yc, w, h]로 구성되어 있다.
        labels = np.loadtxt(path2label).reshape(-1, 5)

    if self.transform:
        img, labels = self.transform(img, labels, self.trans_params)

    return img, labels, path2img
```

class CocoDataset 일부 발췌

먼저, 이미지의 size를 조절하여 정사각형으로 만드는 함수를 정의해보자.

- pad\_to\_square 함수 정의

경계를 채워 정사각형 이미지가 되도록 하기 위한 함수이다.

pad\_to\_square 함수의 필요한 input은 4가지이다.

- img : A PIL image
- boxes : (n,5)의 array형태, n개의 bounding box를 가지고 있다.
- pad\_value : 채울 값, default=0
- normalized\_labels : bounding boxes가 normalized 되었는지 나타내는 flag

```
def pad_to_square(img, boxes, pad_value=0, normalized_labels=True):
    w, h = img.size
    w_factor, h_factor = (w,h) if normalized_labels else (1, 1)

    # (h-w)의 절댓값
    dim_diff = np.abs(h - w)
    # 차원을 2로 나눈 몫
    pad1= dim_diff // 2
    pad2= dim_diff - pad1

    if h<=w:
        left, top, right, bottom= 0, pad1, 0, pad2
    else:
        left, top, right, bottom= pad1, 0, pad2, 0
    padding= (left, top, right, bottom)

    # image tensor 채우기
    img_padded = TF.pad(img, padding=padding, fill=pad_value)
    w_padded, h_padded = img_padded.size

    # bounding box 좌표값도 padding size에 맞게 바꿔야 한다.
    # 바꾸면 다음과 같은 값을 얻을 수 있다. [x1, y1, x2, y2]
    # bounding box 좌표값은 원래 xc, yc. w, h이므로 조정해준다.
    x1 = w_factor * (boxes[:, 1] - boxes[:, 3] / 2)
    y1 = h_factor * (boxes[:, 2] - boxes[:, 4] / 2)
    x2 = w_factor * (boxes[:, 1] + boxes[:, 3] / 2)
    y2 = h_factor * (boxes[:, 2] + boxes[:, 4] / 2)

    x1 += padding[0] # left
    y1 += padding[1] # top
```

```

x2 += padding[2] # right
y2 += padding[3] # bottom

boxes[:, 1] = ((x1 + x2) / 2) / w_padded
boxes[:, 2] = ((y1 + y2) / 2) / h_padded
boxes[:, 3] *= w_factor / w_padded
boxes[:, 4] *= h_factor / h_padded

return img_padded, boxes

```

- hflip 함수 정의

수평으로 뒤집는 함수이다.

```

import torchvision.transforms.functional as TF
def hflip(image, labels):
    image = TF.hflip(image)
    labels[:, 1] = 1.0 - labels[:, 1]
    return image, labels

```

- transformer 함수 정의

```

def transformer(image, labels, params):
    if params["pad2square"] is True:
        image, labels = pad_to_square(image, labels)

    image = TF.resize(image, params["target_size"])

    if random.random() < params["p_hflip"]:
        image, labels = hflip(image, labels)

    image = TF.to_tensor(image)
    targets = torch.zeros((len(labels), 6))
    targets[:, 1:] = torch.from_numpy(labels)

    return image, targets

```

transformer 함수에 보면 targets = torch.zeros((len(labels), 6))이 있다.

label의 값은 5개인데 6개의 값을 남기는 이유는 무엇일까?

-> 남은 하나는 mini-batch에서 image를 indexing 하는데 사용된다.

- train 파라미터 정의 및 coco\_train 다시 정의하기

```

trans_params_train={

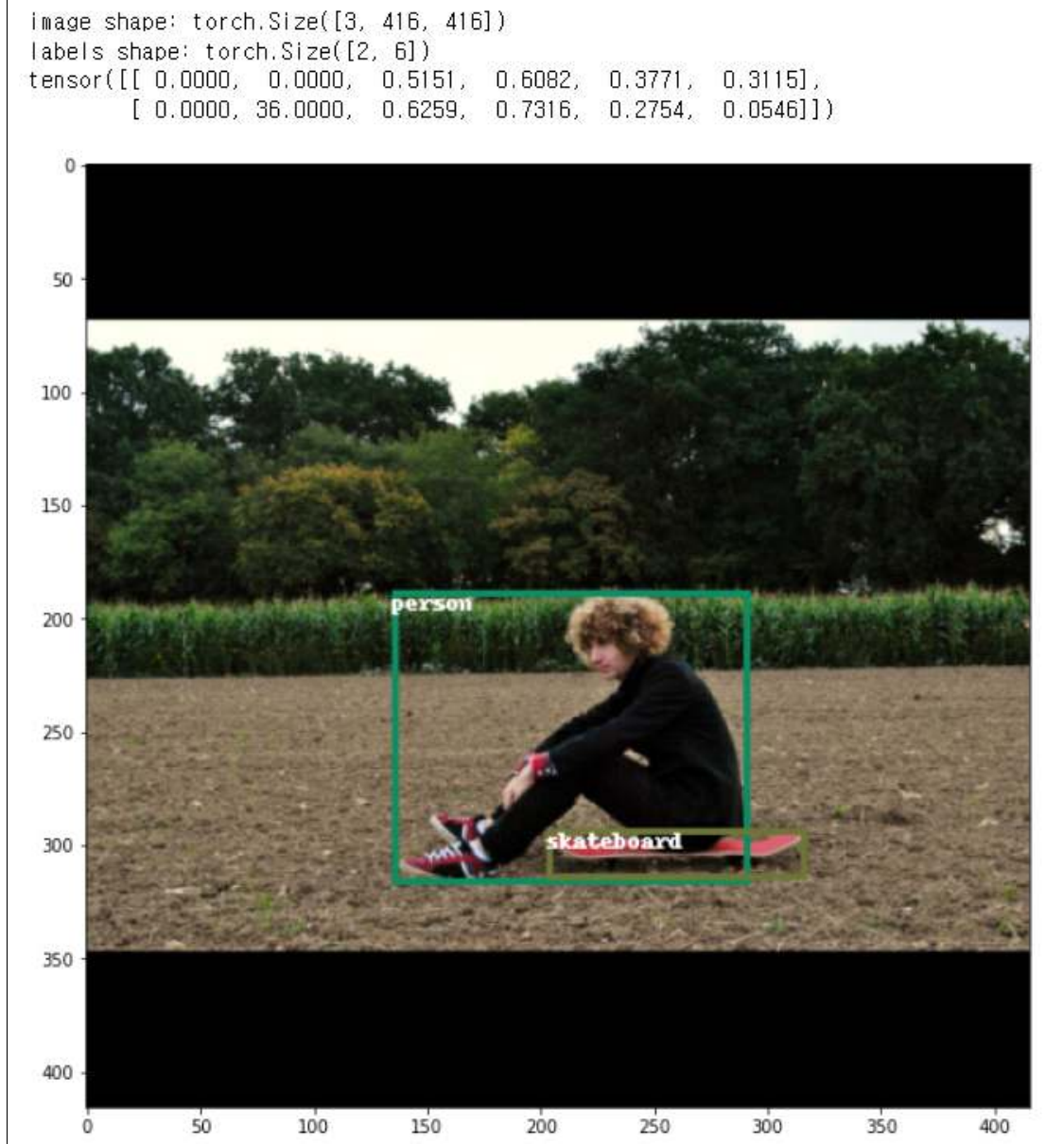
```

```
"target_size" : (416, 416),
"pad2square": True,
"p_hflip" : 1.0,
"normalized_labels": True,
}
coco_train= CocoDataset(path2trainList,
                        transform=transformer,
                        trans_params=trans_params_train)
```

- train 데이터 시각화

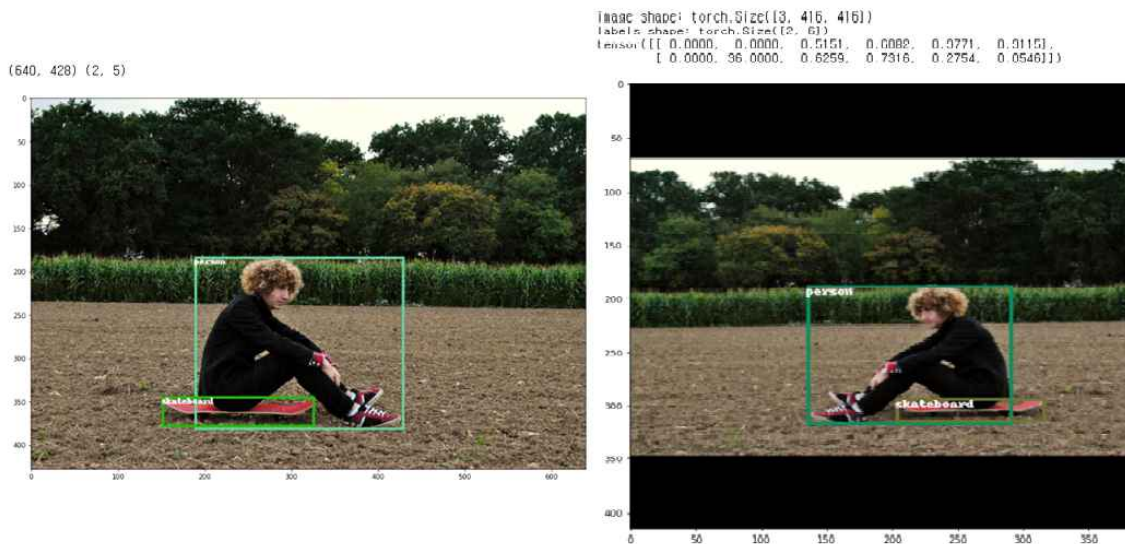
```
np.random.seed(2)
rnd_ind=np.random.randint(len(coco_train))
img, targets, path2img = coco_train[rnd_ind]
print("image shape:", img.shape)
print("labels shape:", targets.shape)
print(targets)

plt.rcParams['figure.figsize'] = (20, 10)
COLORS = np.random.randint(0, 255, size=(80, 3),dtype="uint8")
show_img_bbox(img,targets)
```



앞에 transform을 하기 전과 후를 비교하면 다음과 같다.





**transform 적용 전**

**transform 적용 후**

- val 파라미터 정의 및 coco\_val 정의

```

trans_params_val={
    "target_size" : (416, 416),
    "pad2square": True,
    "p_hflip" : 0.0,
    "normalized_labels": True,
}
coco_val= CocoDataset(path2valList,
                      transform=transformer,
                      trans_params=trans_params_val)
  
```

- val 데이터 시각화

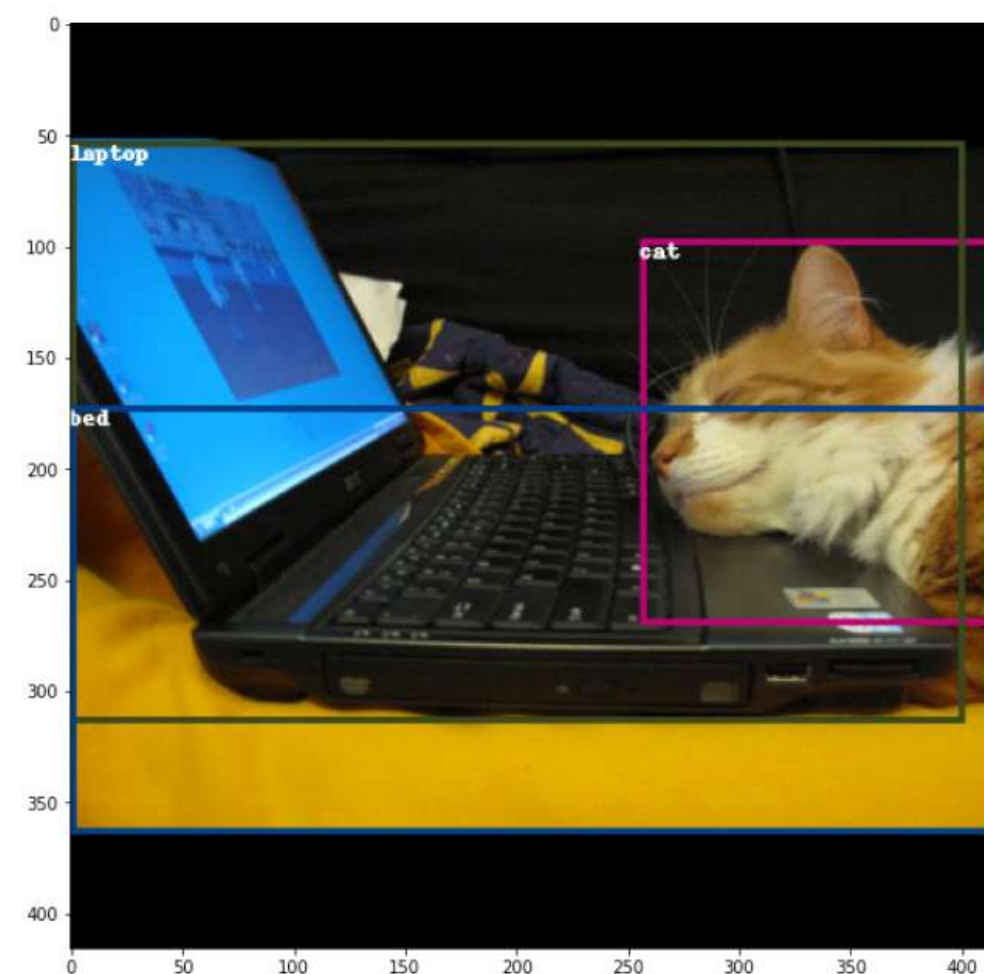
```

np.random.seed(0)
rnd_ind=np.random.randint(len(coco_val))
img, targets, path2img = coco_val[rnd_ind]
print("image shape:", img.shape)
print("labels shape:", targets.shape)

plt.rcParams['figure.figsize'] = (20, 10)
COLORS = np.random.randint(0, 255, size=(80, 3),dtype="uint8")
show_img_bbox(img,targets)
  
```



```
image shape: torch.Size([3, 416, 416])  
labels shape: torch.Size([3, 6])
```



## Defining Data Loaders

이 단계에서는 train dataset과 validation dataset에 대하여 각각 Data Loaders class를 정의할 것이다.

- **DataLoader?**

-> DataLoader class는 Batch 기반의 딥러닝 모델 학습을 위해 mini batch를 만들어는 역할을 dataloader를 통해 dataset의 전체 데이터가 batch size로 slice된다. 앞서 만들었던 dataset을 input으로 넣어주면 여러 옵션(데이터 묶기, 섞기, 알아서 병렬처리)을 통해 batch를 만들어준다.

torch.utils.data의 DataLoader 옵션은 다음과 같다.

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
            batch_sampler=None, num_workers=0, collate_fn=None,
            pin_memory=False, drop_last=False, timeout=0,
            worker_init_fn=None, *, prefetch_factor=2,
            persistent_workers=False)
```

collate\_fn 옵션에 collate\_fn함수를 정의하는데 이는 mini-batch와 tensor를 return하기 위해 정의한다고 한다.

- collate\_fn 함수 정의

```
from torch.utils.data import DataLoader
def collate_fn(batch):
    imgs, targets, paths = list(zip(*batch)) # mini-batch로 묶어주기

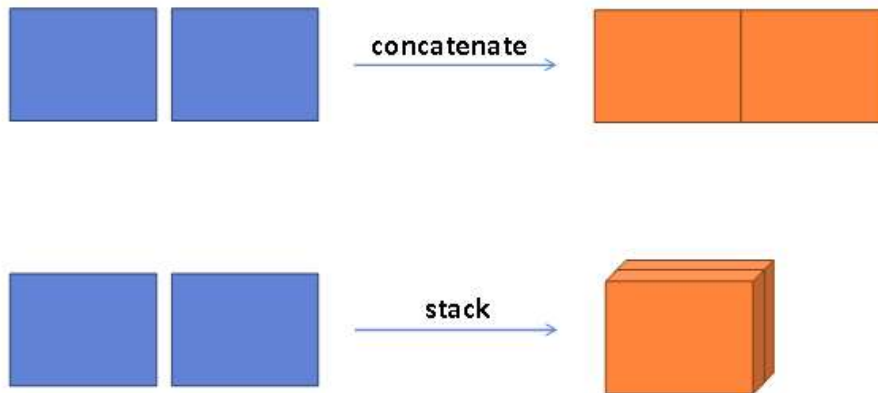
    # targets에서 empty boxes는 다 없애기
    targets = [boxes for boxes in targets if boxes is not None]

    # mini-batch 안에서 set the sample index
    for b_i, boxes in enumerate(targets):
        boxes[:, 0] = b_i
    targets = torch.cat(targets, 0) # 0차원끼리 합쳐주기
    imgs = torch.stack([img for img in imgs])
    return imgs, targets, paths
```

- **torch.cat vs torch.stack**

torch.cat : 합치려고 하는 차원을 증가시킨다.(차원의 수는 유지)

torch.stack : 지정하는 차원으로 확장하여 tensor를 쌓아주는 함수이다.



- train dataset -> train DataLoader에 넣기

```
batch_size=16
train_dl = DataLoader(
    coco_train,
    batch_size=batch_size,
    shuffle=True,
    num_workers=0,
    pin_memory=True,
    collate_fn=collate_fn,
)

torch.manual_seed(0)
for imgs_batch, tg_batch, path_batch in train_dl:
    break
print(imgs_batch.shape)
print(tg_batch.shape, tg_batch.dtype)
```

```
torch.Size([16, 3, 416, 416])
torch.Size([118, 6]) torch.float32
```

118개의 bounding boxes가 mini-batch에서 그려졌다.

- val dataset -> val DataLoader에 넣기

```
batch_size=32
val_dl = DataLoader(
    coco_val,
    batch_size=batch_size,
    shuffle=False,
```

```

num_workers=0,
pin_memory=True,
collate_fn=collate_fn,
)

torch.manual_seed(0)
for imgs_batch, tg_batch, path_batch in val_dl:
    break
print(imgs_batch.shape)
print(tg_batch.shape, tg_batch.dtype)

torch.Size([32, 3, 416, 416])
torch.Size([250, 6]) torch.float32

```

250개의 bounding boxes가 mini-batch에서 그려졌다.

## Creating YOLO-v3 Model

- YOLO-v3

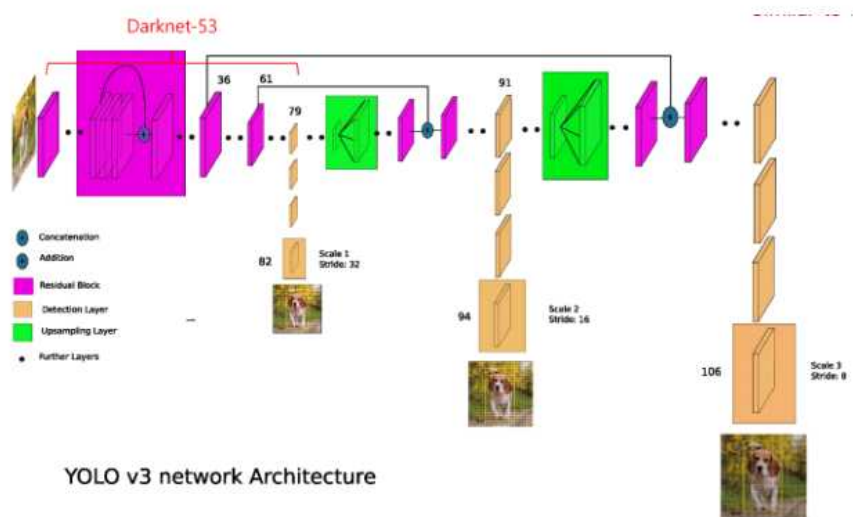


그림. <https://roots2019.tistory.com/325>

요 모형을 쓰기 전에 Object Detection을 하기 위해서 필요한 개념들이 있다.  
이를 먼저 알아보고 시작해보자.

### (1) IoU(Intersection of Union)

IoU는 실제 객체가 있는 Bounding Box(GT, Ground Truth)와 ahef이 예측한 Bounding Box(Predicted)가 포함하고 있는 영역을 계산할 수 있다.

IoU의 계산식은 다음과 같다.

IoU = 교집합(Area of Overlap) / 합집합(Area of Union)

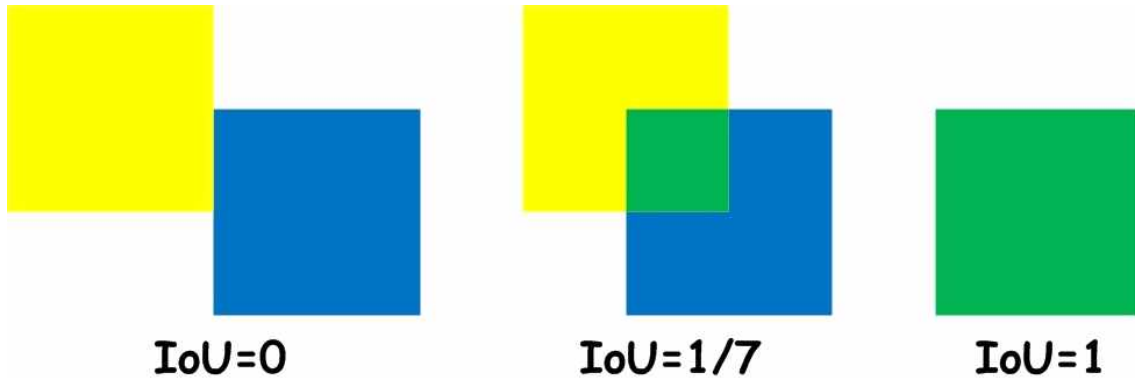


그림. <https://techblog-history-youngjunjo1.tistory.com/178>

## (2) NMS(Non-Max Suppression)

NMS는 Detection 모형이 객체를 정확하게 예측하도록 도와주는 기술이다.

이는 여러 개의 예측된 Bounding Box들 중에 객체의 Confidence score가 가장 높은 Box를 선정하고 score보다 낮은 Box는 삭제한다.

\* Confidence score(objectness) : 검출한 것에 대해 알고리즘이 얼마나 확신이 있는지 알려주는 값  
예로 밑의 그림에서 자동차에 대한 Confidence score가 0.8이 나왔다면 bounding box안 객체가 자동차일 확률이 80%라고 확신한다고 생각하면 쉽다.

$$Pr(Object) * IOU_{pred}^{truth}$$

그리고 score가 가장 높은 Box와 다른 Box들 간의 IoU를 계산하고, 미리 지정한 IoU threshold보다 큰 다른 Box를 모두 제거한다.



- Confidence Score Threshold : 0.6
- IoU Threshold : 0.5

1. 가장 Confidence Score가 높은 바운딩 박스를 빨간색 박스로 선정
2. Confidence Score Threshold인 0.6보다 작은 바운딩 박스들 삭제  
→ 0.59인 파란색 박스 삭제
3. 빨간색 박스와 IoU를 계산했을 때 IoU Threshold인 0.5보다 큰 박스들 삭제  
→ 0.7인 파란색 박스 삭제

• Confidence score threshold : 그림처럼 파란색 자동차 객체를 탐지하기 위해 3개의 Box가 도출되었다. 이를 NMS를 수행하므로 자동차 객체를 가장 잘 탐지한 Box를 제외한 나머지를 삭제하여 예측값이 여러개가 나오는 것을 막을 수 있다.

• IoU threshold : 파란색 차와 전혀 관련없는 초록색 Box를 삭제하지 않고 유지시켜 다른 곳에 있는 객체들을 탐지할 수 있게 한다.

### (3) mAP(mean-Average Precision)

		Predicted	
		Positive	Negative
Actual	Positive	TP(True Positive)	FN(False Negative)
	Negative	FP(False Positive)	TN(True Negative)

그림. 딥러닝 기반 COVID-19 CT 영상 분류 포스터 발췌  
mAP에 대하여 하기 전에 Precision과 Recall에 대하여 다시 한 번 상기시켜보자.

위 표에서 얻을 수 있는 판단척도는 엄청 많다. 대표적으로 몇 가지를 정리하면 다음과 같다.

- Accuracy

전체에서 모형이 (True -> 실제 True인 값)과 (False -> 실제 False)인 경우이다.

$$(Accuracy) = \frac{TP + TN}{TP + FN + FP + TN}$$

- Precision

모형의 입장에서 전체에서 모형이 True라고 했을 때 실제 True인 비율을 의미한다.

$$(Precision) = \frac{TP}{TP + FP}$$

- Recall

정답의 입장에서 전체에서 실제 True라고 했을 때 모형에서 True라고 예측한 비율을 의미한다.

$$(Recall) = \frac{TP}{TP + FN}$$

컴퓨터 비전에서 위를 표시하면 다음과 같다.

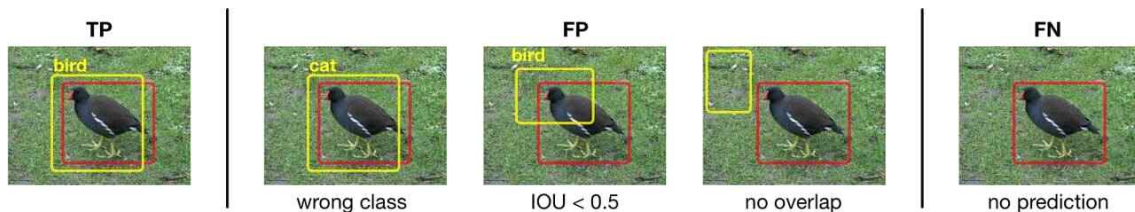


그림. <https://techblog-history-younghunjo1.tistory.com/178>

위 Precision과 Recall은 수치를 잘못 사용하면 오류가 발생할 수 있다. 따라서 Confidence score threshold를 조절하면서 얻은 Precision-Recall Curve 그래프를 그려 아래 면적을 나타내는 AP(Average Precision)값을 이용하는 것이 좋다.

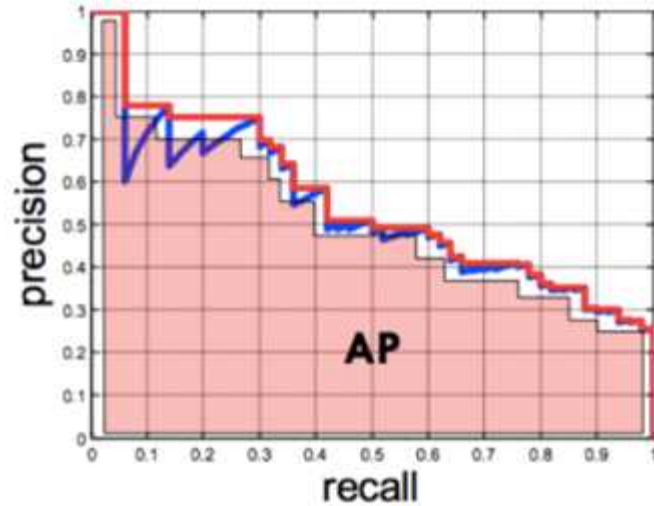


그림. 다크프로그래머님의 블로그  
하나의 AP는 하나의 객체에 대한 탐지 성능이다. 그럼 mAP는?  
여러개의 객체들이 담긴 이미지를 객체 탐지했을 때 하나의 객체에 대한 AP값을  
구하고 이미지에 존재하는 객체들의 수만큼 평균값을 취해주는 것이다.

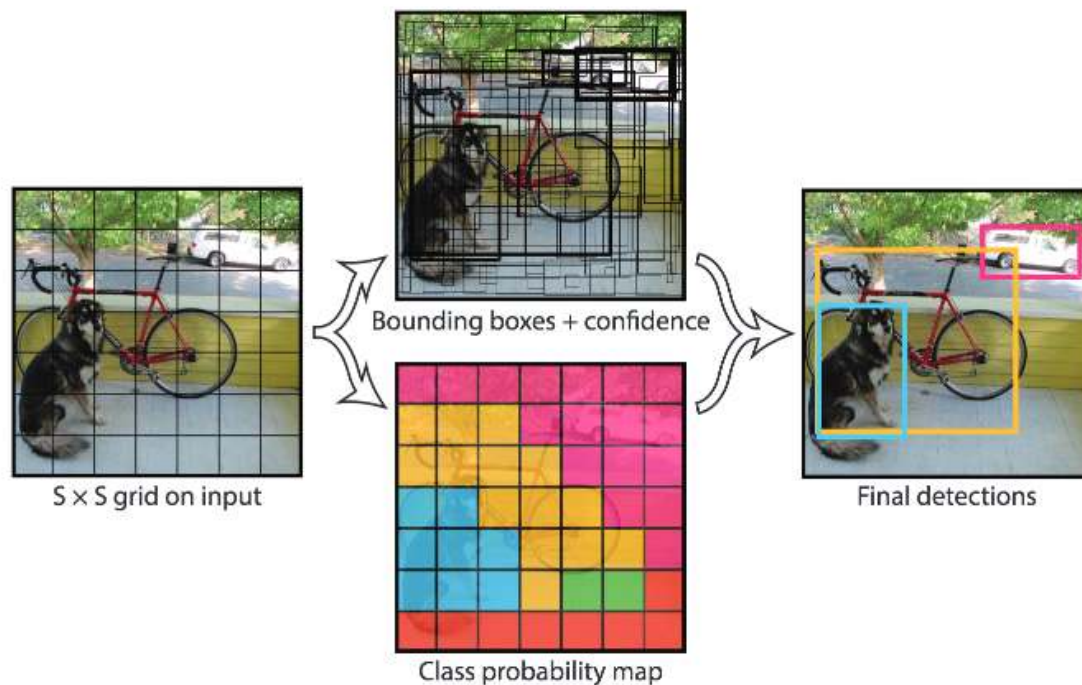


그림. Example of application. The input image is divided into an  $S \times S$  grid,  $B$  bounding boxes are predicted (regression) and a class is predicted among  $C$  classes (classification) over the most confident ones. Source: J. Redmon and al. (2016)

원리를 그림으로 나타내면 위 그림과 같다.

- 1) 이미지를 input으로 입력을 받는다. 이때 이미지를  $S \times S$ 의 Grid cell로 분할한다.
- 2) 각 Grid cell은 Bounding Box의 좌표값과 Bounding Box에 대한 Confidence score를 가진다. 따라서 각 Bounding Box의 값은  $x, y, w, h$ , Confidence score로 구성되어 있다.

\* Confidence score(objectness) : 검출한 것에 대해 알고리즘이 얼마나 확신이 있는지 알려주는 값 만약 해당 cell에서 객체가 감지되지 않는다면 Confidence score는 0이다.

- 3) 각 Grid cell은  $C$ 개(물체)의 Conditional Class Probability(조건부 확률)를 가진다.

- 4) 따라서 나올 수 있는 output은  $[x, y, w, h, \text{confidence score}, c\text{개의 확률}]$ 이다. 위 값이  $S \times S$ 만큼 나온다.

예시) 보행자, 자동차, 오토바이에 대하여 anchor box= 1





- 특징

(1) Bounding Box Prediction

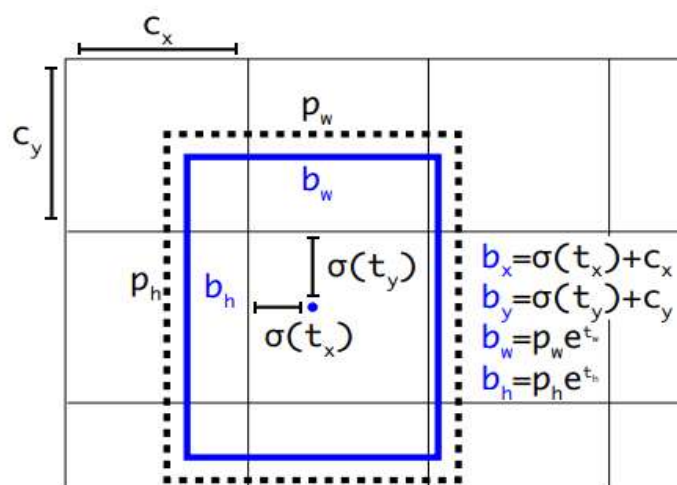


그림. <https://visionhong.tistory.com/16>

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

$$t_x = \sigma^{-1}(b_x - c_x)$$

$$t_y = \sigma^{-1}(b_y - c_y)$$

$$t_w = \ln b_w / p_w$$

$$t_h = \ln b_h / p_h$$

**YOLOv2 BB prediction**

**YOLOv3 BB prediction**

그림. YOLOv2와 YOLOv3 BB 예측값 차이

- YOLOv2(YOLO9000) - 각 Bounding Box의 4개 tx, ty, tw, th를 예측하여 bx, by, bw, bh parameter들을 얻고 이 값들을 통해 L2 loss(오차제곱합)를 구하면서 학습을 진행한다.
- 식이 바뀐 이유 - Bounding Box에 대한 score를 YOLOv3에서는 Logistic regression(Sigmoid Function)으로 측정하기 때문이다.
- YOLOv3 - 기존 식의 역함수를 구하여 L1 loss(오차절대값합)를 구하면서 학습을 진행한다.

tx, ty : 중심점을 예측한다. Sigmoid Function을 이용해 [0,1] 범위로 표현

tw, th : e를 붙여줘서 음수가 되는 것을 방지하고 훈련 데이터셋으로 군집화(k-means)를 통해 구한 box prior px, pw에 이를 곱해준다.

## (2) Class Prediction

위에서 YOLOv3는 Logistic regression으로 측정한다고 했는데, 왜 그렇게 할까? 보통 Multi-labels에 대하여 softmax함수를 적용한다. 그런데 softmax 함수를 사용하면 세밀하게 사람(여자, 남자)처럼 너무 많은 overlapping(겹치는) label이 존재해버려서 계산량이 비효율적으로 많아진다. 이러한 이유로 Logistic regression(Sigmoid Function)을 이용하여 계산을 해준다.

## (3) Predictions Across Scales

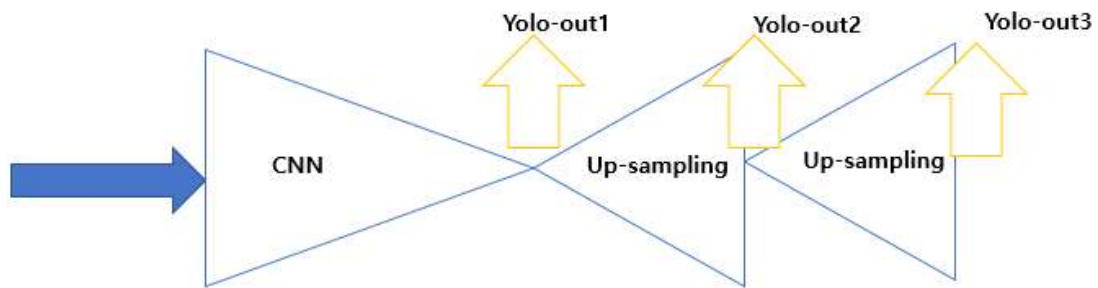
위 모형은 3개의 다른 scale에서 feature map을 추출한다. 기본적인 feature 추출기에 몇 개의 conv layer만 추가하면 된다. 위 논문에서는 coco data를 가지고 실험하였는데 scale이 다른 3개의 box에 대한 tensor는  $n \times n \times [3 \times (4 + 1 + 80)]$ 라고 한다.

- \* n은 feature map의 개수,
- \* 3는 anchor box,
- \* 4개의 box shift and scaling value,
- \* 1개의 objectness score,
- \* 80개의 class

를 예측한다.

그다음으로는 이전 2번째 layer에서 feature map을 가져와 2배로 upscaling을 하고, 제일 앞단의 feature map을 가져와 둘을 concatenation을 사용해서 upsampling 된 feature map과 합친다. 이렇게 되면 의미 있는 semantic information을 이전 layer의 feature map으로 얻을 수 있고, finer-grained information을 제일 앞단의 feature map으로부터 얻을 수 있다. 결국은 skip layer와 개념이 유사하다. 무튼 size는 2배가 되지만 유사한 tensor들을 더욱더 잘 예측할 수 있다.

이를 책에 나와 있는대로 표시하면 다음과 같다.



- 1) input : Normalized image size (416, 416), RGB color
- 2) Convolution layers with stride 2, skip connections, up-sampling layers
- 3) 3 곳(82,94,106 layer)에서 output이 나온다.  
 -> 서로 다른 scale(13,26,52)을 사용하여 최종 결과를 예측한다. 이때 multi-scale feature map을 얻는 방법은 FPN 모형과 유사하다고 한다.
  - Yolo-out1 : feature map size 13\*13
  - Yolo-out2 : feature map size 26\*26
  - Yolo-out3 : feature map size 52\*52
- 4) **Feature Map 각 Cell은 predefined anchors box 3개에 해당하는 3개의 Bounding Box를 예측한다. 따라서 나올 수 있는 Bounding Box의 개수는  $13*13*3 + 26*26*3 + 52*52*3 = 10647$ 개이다.**
- 5) 하나의 Bounding Box 안에는 85개의 숫자가 나타나있다.
  - [x, y, w, h] 좌표값
  - objectness score
  - 80 categories class predictions

이 과정들을 나타내보자.

- myutils.py 파일 import

```
from myutils import *
```

myutils.py 파일 안에는 두 가지의 함수와 두 가지의 class가 정의되어 있다.

- Function : parse\_model\_config(path2file), create\_layers(blocks\_list)
- Class : EmptyLayer, YOLOLayer

이를 먼저 하나씩 살펴보자.

#### ① parse\_model\_config 함수

path2file(경로)을 받아와서 block 또는 layer list를 만드는 함수이다.  
이 함수를 통해 block\_list(layer의 파라미터 list)를 return한다.

```
def parse_model_config(path2file):
    cfg_file = open(path2file, 'r')

    # split file into lines '\n'
    lines = cfg_file.read().split('\n')

    # remove any lines and comments
    lines = [x for x in lines if x and not x.startswith('#')]

    # remove white space
    lines = [x.rstrip().lstrip() for x in lines]

    blocks_list = []
    for line in lines:
        # start of a new block(layer)
        # '['로 시작하면 새 dictionary 추가, key = type
        if line.startswith '['):
            blocks_list.append({})
            blocks_list[-1]['type'] = line[1:-1].rstrip()
        else:
            # attribute of the layer으로 dictionary 만들기
            key, value = line.split("=")
            value = value.strip()
            blocks_list[-1][key.rstrip()] = value.strip()

    return blocks_list
```

① 예시)

```
In [6]: c = 'absockde'
        print(c[1:-1])
```

bsockd

```
In [8]: a = []
        b = {}
        a.append(b)
        print(a)
        a[-1]['type'] = c[1:-1].rstrip()
        print(a)
```

[{}]

[{'type': 'bsockd'}]

jupyter yolov3.cfg ✓ 2022.03.07

File Edit View Language

```
1 [net]
2 # Testing
3 # batch=1
4 # subdivisions=1
5 # Training
6 batch=64
7 subdivisions=16
8 width=608
9 height=608
10 channels=3
11 momentum=0.9
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18 learning_rate=0.001
19 burn_in=1000
20 max_batches = 500200
21 policy=steps
22 steps=400000,450000
23 scales=.1,.1
24
25 [convolutional]
26 batch_normalize=1
27 filters=32
28 size=3
29 stride=1
30 pad=1
31 activation=leaky
32
33 # Downsample
34
35 [convolutional]
36 batch_normalize=1
37 filters=64
38 size=3
39 stride=2
40 pad=1
41 activation=leaky
42
```

요기 파라미터들을 각각 넣어서 blocks\_list를 만들어줄 것이다.

## ② create\_layers 함수

block\_list를 받아 적혀 있는 파라미터에 맞춰 모형 안에 들어가는 layer를 쌓아주는 함수이다.

```
def create_layers(blocks_list):
    # first block에는 hyperparams가 있다.
    hyperparams = blocks_list[0]
    # 각각 layer에 대하여 input channel의 수를 담는다.
    channels_list = [int(hyperparams["channels"])]
    module_list = nn.ModuleList()

    # layer 채우기
    for layer_ind, layer_dict in enumerate(blocks_list[1:]):
        modules = nn.Sequential()

        # nn.Conv2d, nn.BatchNorm2d, nn.LeakyReLU
        if layer_dict["type"] == "convolutional":
            filters = int(layer_dict["filters"])
            kernel_size = int(layer_dict["size"])
            pad = (kernel_size - 1) // 2
            bn=layer_dict.get("batch_normalize",0)

            conv2d= nn.Conv2d(
                in_channels=channels_list[-1],
                out_channels=filters,
                kernel_size=kernel_size,
                stride=int(layer_dict["stride"]),
                padding=pad,
                bias=not bn)
            modules.add_module("conv_{0}".format(layer_ind), conv2d)

            if bn:
                bn_layer = nn.BatchNorm2d(filters,momentum=0.9, eps=1e-5)
                modules.add_module("batch_norm_{0}".format(layer_ind), bn_layer)

            if layer_dict["activation"] == "leaky":
                activn = nn.LeakyReLU(0.1)
                modules.add_module("leaky_{0}".format(layer_ind), activn)
```

```

# nn.UpSample, stride = 2, default mode = 'nearest'
elif layer_dict["type"] == "upsample":
    stride = int(layer_dict["stride"])
    upsample = nn.Upsample(scale_factor = stride)
    modules.add_module("upsample_{}".format(layer_ind), upsample)

# skip-connections 만들기
elif layer_dict["type"] == "shortcut":
    backwards=int(layer_dict["from"])
    filters = channels_list[1:][backwards]
    modules.add_module("shortcut_{}".format(layer_ind), EmptyLayer())

elif layer_dict["type"] == "route":
    layers = [int(x) for x in layer_dict["layers"].split(",")]
    filters = sum([channels_list[1:][l] for l in layers])
    modules.add_module("route_{}".format(layer_ind), EmptyLayer())

elif layer_dict["type"] == "yolo":
    anchors = [int(a) for a in layer_dict["anchors"].split(",")]
    anchors = [(anchors[i], anchors[i + 1]) for i in range(0, len(anchors),
2)]

    mask = [int(m) for m in layer_dict["mask"].split(",")]

    anchors = [anchors[i] for i in mask]

    num_classes = int(layer_dict["classes"])
    img_size = int(hyperparams["height"])

    yolo_layer = YOLOLayer(anchors, num_classes, img_size)
    modules.add_module("yolo_{}".format(layer_ind), yolo_layer)

    module_list.append(modules)
    channels_list.append(filters)

return hyperparams, module_list

```

구체적으로 무엇인지 살펴보자.

#### - Upsampling layer

```
elif layer_dict["type"] == "upsample":
```

```

stride = int(layer_dict["stride"])
upsample = nn.Upsample(scale_factor = stride)
modules.add_module("upsample_{}".format(layer_ind), upsample)

```

Upsampling을 통해 데이터의 크기를 키우는 것이다.

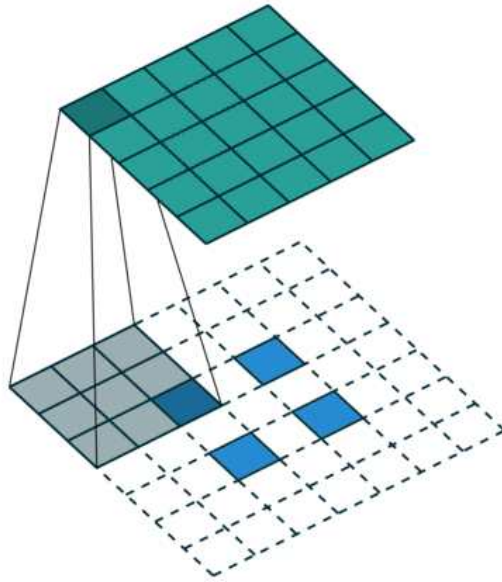


그림. stride = 2일 때

<https://wikidocs.net/149326>

mode가 nearest라는 것은 resizing 할 때 가까운 애로 하라는 그런 뜻이다.

예시)

```

>>> input = torch.arange(1, 5, dtype=torch.float32).view(1, 1, 2, 2)
>>> input
tensor([[[[ 1.,  2.],
           [ 3.,  4.]]]])

>>> m = nn.Upsample(scale_factor=2, mode='nearest')
>>> m(input)
tensor([[[[ 1.,  1.,  2.,  2.],
           [ 1.,  1.,  2.,  2.],
           [ 3.,  3.,  4.,  4.],
           [ 3.,  3.,  4.,  4.]]]]])

>>> m = nn.Upsample(scale_factor=2, mode='bilinear') # align_corners=False
>>> m(input)
tensor([[[[ 1.0000,  1.2500,  1.7500,  2.0000],
           [ 1.5000,  1.7500,  2.2500,  2.5000],
           [ 2.5000,  2.7500,  3.2500,  3.5000],
           [ 3.0000,  3.2500,  3.7500,  4.0000]]]])

```



### - skip-connection layer

```
elif layer_dict["type"] == "shortcut":
    backwards=int(layer_dict["from"])
    filters = channels_list[1:][backwards]
    modules.add_module("shortcut_{}".format(layer_ind), EmptyLayer())
```

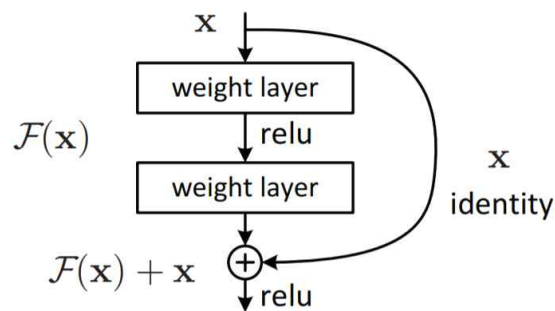


Figure 2. Residual learning: a building block.

그림. <https://arxiv.org/pdf/1512.03385.pdf>

자기 자신을 더해서 output으로 출력하는 layer이다.

from 값에 따라 어디에서 더할지 정해진다. 가령 from=-3이면 3번째 뒤에 있는 layer와 합쳐준다는 의미이다.

skip connection의 filter의 개수는 channels\_list에 저장되어 있다.

### - route layer

```
elif layer_dict["type"] == "route":
    layers = [int(x) for x in layer_dict["layers"].split(",")]
    filters = sum([channels_list[1:][l] for l in layers])
    modules.add_module("route_{}".format(layer_ind), EmptyLayer())
```

경로 layer이다. 실제로 연결은 forward function of the network class에서 한다. 여기에서는 filter 수를 합산한다.

### - yolo block

```
elif layer_dict["type"] == "yolo":
    anchors = [int(a) for a in layer_dict["anchors"].split(",")]
    anchors = [(anchors[i], anchors[i + 1]) for i in range(0, len(anchors),
2)]

    mask = [int(m) for m in layer_dict["mask"].split(",")]
```

```

anchors = [anchors[i] for i in mask]

num_classes = int(layer_dict["classes"])
img_size = int(hyperparams["height"])

yolo_layer = YOLOLayer(anchors, num_classes, img_size)
modules.add_module("yolo_{}".format(layer_ind), yolo_layer)

```

yolo block 안에서는 anchors와 num\_classes, image\_size를 정의할 수 있다.

이런식으로 되어 있는 애를 뜯어준다.

```

[yolo]
mask = 0,1,2
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1

```

전체적인 과정을 통해 우리는 module\_list과 hyperparams를 얻을 수 있다.

### ③ EmptyLayer Class

```

class EmptyLayer(nn.Module):
    def __init__(self):
        super(EmptyLayer, self).__init__()

```

EmptyLayer에서는 skip(shortcut) connection 안에서 아무것도 하지 않는다.

### ④ YOLOLayer Class

- anchors, num\_classes, img\_dim을 input으로 받아온다.
- 우리 데이터에서는 다음과 같은 수가 들어온다.
  - anchors : a list of three tuples
  - num\_classes : 80 categories
  - img\_dim : 416

```

class YOLOLayer(nn.Module):

    def __init__(self, anchors, num_classes, img_dim=416):
        super(YOLOLayer, self).__init__()
        self.anchors = anchors

```

```

self.num_anchors = len(anchors)
self.num_classes = num_classes
self.img_dim = img_dim
self.grid_size = 0

def forward(self, x_in):
    batch_size = x_in.size(0)
    grid_size = x_in.size(2)
    device=x_in.device

    prediction=x_in.view(batch_size, self.num_anchors,
                        self.num_classes + 5, grid_size, grid_size)
    prediction=prediction.permute(0, 1, 3, 4, 2)
    prediction=prediction.contiguous()

    obj_score = torch.sigmoid(prediction[..., 4])
    pred_cls = torch.sigmoid(prediction[..., 5:])

    if grid_size != self.grid_size:
        self.compute_grid_offsets(grid_size, cuda=x_in.is_cuda)

    pred_boxes=self.transform_outputs(prediction)

    output = torch.cat(
        (
            pred_boxes.view(batch_size, -1, 4),
            obj_score.view(batch_size, -1, 1),
            pred_cls.view(batch_size, -1, self.num_classes),
        ), -1,)
    return output

def compute_grid_offsets(self, grid_size, cuda=True):
    self.grid_size = grid_size
    self.stride = self.img_dim / self.grid_size

    self.grid_x = torch.arange(grid_size, device=device).repeat(1, 1, grid_size,
1 ).type(torch.float32)
    self.grid_y = torch.arange(grid_size, device=device).repeat(1, 1, grid_size,
1 ).transpose(3, 2).type(torch.float32)

```

```

        scaled_anchors=[(a_w / self.stride, a_h / self.stride) for a_w, a_h in
self.anchors]
        self.scaled_anchors=torch.tensor(scaled_anchors,device=device)

        self.anchor_w = self.scaled_anchors[:, 0:1].view((1, self.num_anchors, 1,
1))
        self.anchor_h = self.scaled_anchors[:, 1:2].view((1, self.num_anchors, 1,
1))

    def transform_outputs(self,prediction):
        device=prediction.device
        x = torch.sigmoid(prediction[..., 0]) # Center x
        y = torch.sigmoid(prediction[..., 1]) # Center y
        w = prediction[..., 2] # Width
        h = prediction[..., 3] # Height

        pred_boxes = torch.zeros_like(prediction[..., :4]).to(device)
        pred_boxes[..., 0] = x.data + self.grid_x
        pred_boxes[..., 1] = y.data + self.grid_y
        pred_boxes[..., 2] = torch.exp(w.data) * self.anchor_w
        pred_boxes[..., 3] = torch.exp(h.data) * self.anchor_h

        return pred_boxes * self.stride

```

애도 구체적으로 무엇인지 살펴보자.

#### - forward

x\_in을 input으로 받아온다.

```

def forward(self, x_in):
    batch_size = x_in.size(0)
    grid_size = x_in.size(2)
    device=x_in.device

    prediction=x_in.view(batch_size, self.num_anchors,
                        self.num_classes + 5, grid_size, grid_size)

    # 위치 바꾸기
    prediction=prediction.permute(0, 1, 3, 4, 2)
    # 연속적인 메모리 tensor를 반환하는 메서드
    # 어떤 연산을 사용할 때 이를 사용하지 않으면 에러가 발생할 수도 있다.
    prediction=prediction.contiguous()

```

```

obj_score = torch.sigmoid(prediction[..., 4])
pred_cls = torch.sigmoid(prediction[..., 5:])

if grid_size != self.grid_size:
    self.compute_grid_offsets(grid_size, cuda=x_in.is_cuda)

pred_boxes=self.transform_outputs(prediction)

output = torch.cat(
    (
        pred_boxes.view(batch_size, -1, 4),
        obj_score.view(batch_size, -1, 1),
        pred_cls.view(batch_size, -1, self.num_classes),
    ), -1,)
return output

```

- contiguous() 메소드

tensor의 shape를 조작하는 과정에서 메모리 저장 상태가 변경될 수 있다. 주로 narrow(), view(), expand(), transpose() 등을 사용할 때이다. 오류가 발견이 안 될 수도 있지만 오류가 발생하는 경우 contiguous()를 tensor에 적용하여 contiguous 여부가 True인 상태로 메모리 상 저장구조를 바꿔줄 수 있다.

위 함수의 예로 예로 첫번째 yolo layer에 대한 input은 [batch\_size, 3\*85, 13, 13]이다. 이를 우리는 [batch\_size, 3, 13, 13, 85]로 reshape 해준다. 그다음 objectness score와 class predictions를 위해 Sigmoid Function에 넣어준다. 후에 compute\_grid\_offsets 함수를 이용하여 grid offset을 계산해준다. Bounding box의 예측값은 transform\_outputs 함수를 이용하여 계산해준다.

<https://wdprogrammer.tistory.com/50>

- compute\_grid\_offsets 함수

위 함수는 3가지 input을 가지고 있다.

- self : YOLOLayer class를 나타낸다.
- grid\_size : 그리드 사이즈를 나타낸다. 13, 26, 52
- cuda : True or False

grid\_size=13일 때, grid는 x와 y의 차원에서 13\*13개가 생성된다.

또한 Anchor의 크기도 조정된다. stride=32로.

```
def compute_grid_offsets(self, grid_size, cuda=True):
```

```

        self.grid_size = grid_size
        self.stride = self.img_dim / self.grid_size

        self.grid_x = torch.arange(grid_size, device=device).repeat(1, 1, grid_size,
1 ).type(torch.float32)
        self.grid_y = torch.arange(grid_size, device=device).repeat(1, 1, grid_size,
1).transpose(3, 2).type(torch.float32)

        scaled_anchors=[(a_w / self.stride, a_h / self.stride) for a_w, a_h in
self.anchors]
        self.scaled_anchors=torch.tensor(scaled_anchors,device=device)

        self.anchor_w = self.scaled_anchors[:, 0:1].view((1, self.num_anchors, 1,
1))
        self.anchor_h = self.scaled_anchors[:, 1:2].view((1, self.num_anchors, 1,
1))

```

• repeat(1,1,grid\_size,1)

해당 shape만큼 타일 형태로 tensor를 쌓는 메소드이다.

**\* 질문 \***

```

grid_size = 13
img_dim=416
stride = img_dim/grid_size

x = torch.arange(grid_size)
grid_x = x.repeat(1,1,grid_size).type(torch.float32)
grid_y = x.repeat(1,1,grid_size).transpose(3,2).type(torch.float32)

print(x)
print(grid_x.shape)
print(grid_y)

```

-----

```

IndexError                                Traceback (most recent call last)
Input In [119], in <module>
      5 x = torch.arange(grid_size)
      6 grid_x = x.repeat(1,1,grid_size).type(torch.float32)
----> 7 grid_y = x.repeat(1,1,grid_size).transpose(3,2).type(torch.float32)
      9 print(x)
     10 print(grid_x.shape)

IndexError: Dimension out of range (expected to be in range of [-3, 2], but got 3)

```

- transform\_output 함수

위 함수에서는 두 개의 input이 들어간다.

- self
- prediction : tensor shape는 (batch\_size, 3, grid\_size, grid\_size, 85)이다.

이때 3은 anchor box의 개수이다.

```
def transform_outputs(self, prediction):
    device = prediction.device
    x = torch.sigmoid(prediction[..., 0]) # Center x
    y = torch.sigmoid(prediction[..., 1]) # Center y
    w = prediction[..., 2] # Width
    h = prediction[..., 3] # Height

    # scaled bounding box predictions
    pred_boxes = torch.zeros_like(prediction[..., :4]).to(device)
    pred_boxes[..., 0] = x.data + self.grid_x
    pred_boxes[..., 1] = y.data + self.grid_y
    pred_boxes[..., 2] = torch.exp(w.data) * self.anchor_w
    pred_boxes[..., 3] = torch.exp(h.data) * self.anchor_h

    return pred_boxes * self.stride
```

얘네가 의미하는게 Bounding Box Prediction이다.

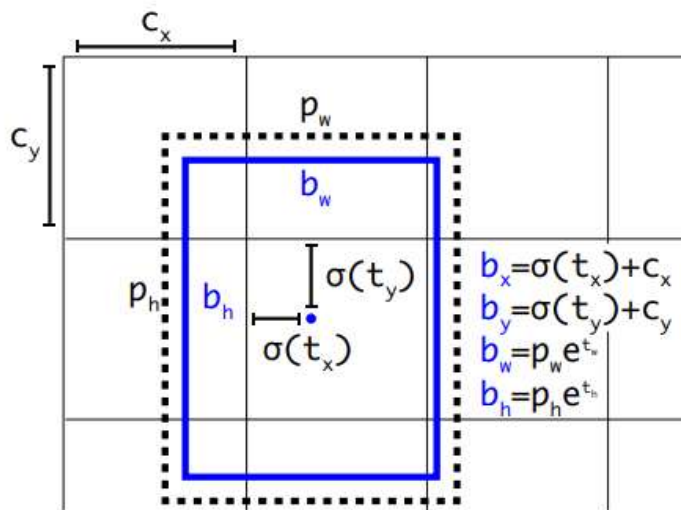


그림. <https://visionhong.tistory.com/16>

이렇게 myutils.py 파일 정의된 class와 function을 살펴보았다.

이제 이를 이용하여 YOLO-v3를 정의해보자.

- myutils에 있는 parse\_model\_config 함수 이용

```
path2config="/home/user303/jihyeheo/darknet/cfg/yolov3.cfg"
```

```
blocks_list = parse_model_config(path2config)
blocks_list[:2]
```

```
[{'type': 'net',
  'batch': '64',
  'subdivisions': '16',
  'width': '608',
  'height': '608',
  'channels': '3',
  'momentum': '0.9',
  'decay': '0.0005',
  'angle': '0',
  'saturation': '1.5',
  'exposure': '1.5',
  'hue': '.1',
  'learning_rate': '0.001',
  'burn_in': '1000',
  'max_batches': '500200',
  'policy': 'steps',
  'steps': '400000,450000',
  'scales': '.1,.1'},
 {'type': 'convolutional',
  'batch_normalize': '1',
  'filters': '32',
  'size': '3',
  'stride': '1',
  'pad': '1',
  'activation': 'leaky'}]
```

## Creating Pytorch modules

pased configuration file을 기반으로 Pytorch modules를 만들어 볼 것이다.

우리는 myutils.py라는 파일 안에, create\_layers를 이용하여 YOLOv3 network에 대하여 만들어낸다.

```
from myutils import create_layers

hy_pa, m_l= create_layers(blocks_list)
print(m_l)
print(hy_pa)
```

```
# m_l
```



```

ModuleList(
  (0): Sequential(
    (conv_0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
    bias=False)
    (batch_norm_0): BatchNorm2d(32, eps=1e-05, momentum=0.9, affine=True,
    track_running_stats=True)
    (leaky_0): LeakyReLU(negative_slope=0.1)
  )
  ...
  (batch_norm_104): BatchNorm2d(256, eps=1e-05, momentum=0.9, affine=True,
  track_running_stats=True)
  (leaky_104): LeakyReLU(negative_slope=0.1)
)
  (105): Sequential(
    (conv_105): Conv2d(256, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (106): Sequential(
    (yolo_106): YOLOLayer()
  )
)
# hy_pa
{'type': 'net', 'batch': '64', 'subdivisions': '16', 'width': '608', 'height': '608',
'channels': '3', 'momentum': '0.9', 'decay': '0.0005', 'angle': '0', 'saturation':
'1.5', 'exposure': '1.5', 'hue': '.1', 'learning_rate': '0.001', 'burn_in': '1000',
'max_batches': '500200', 'policy': 'steps', 'steps': '400000,450000', 'scales': '.1,.1'}

```

## Defining Darknet Model

Darknet class를 정의하여 앞서 만들었던 모형을 만든다.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

그림. YOLOv3 Darknet-53

```

import torch.nn as nn
class Darknet(nn.Module):
    def __init__(self, config_path, img_size=416):
        super(Darknet, self).__init__()
        # YOLO-v3 network block
        self.blocks_list = parse_model_config(config_path)
        self.hyperparams, self.module_list = create_layers(self.blocks_list)
        self.img_size = img_size
        # x = (batch_size, 3, 416, 416)
        # 두 개의 출력값이 있다. 각 layer에서의 output과 yolo output
    def forward(self, x):
        img_dim = x.shape[2]
        layer_outputs, yolo_outputs = [], []

        # first block에 hyperparameter가 있기 때문에 blocks_list[1:]이다.
        for block, module in zip(self.blocks_list[1:], self.module_list):
            if block["type"] in ["convolutional", "upsample", "maxpool"]:

```

```

        x = module(x)

    elif block["type"] == "shortcut":
        layer_ind = int(block["from"])
        x = layer_outputs[-1] + layer_outputs[layer_ind]
    elif block["type"] == "yolo":
        x= module[0](x)
        yolo_outputs.append(x)
    elif block["type"] == "route":
        x = torch.cat([layer_outputs[int(l_i)]
                        for l_i in block["layers"].split(",")], 1)
        layer_outputs.append(x)
    yolo_out_cat = torch.cat(yolo_outputs, 1)
    return yolo_out_cat, yolo_outputs

```

- Darknet 예시]

```
model = Darknet(path2config).to(device)
```

```

dummy_img=torch.rand(1,3,416,416).to(device)
with torch.no_grad():
    dummy_out_cat, dummy_out=model.forward(dummy_img)
    print(dummy_out_cat.shape)
    print(dummy_out[0].shape,dummy_out[1].shape,dummy_out[2].shape)

```

```

torch.Size([1, 10647, 85])
torch.Size([1, 507, 85]) torch.Size([1, 2028, 85]) torch.Size([1, 8112, 85])

```

## Defining the Loss Function

Loss Function을 계산하는데 필요한 함수를 정의하는 단계이다.

YOLO의 Loss Function은 다음과 같다.

- Bounding box : [x, y, w, h]
- Objectness Score
- Class predictions for 80 categories

$$loss = loss_x + loss_y + loss_w + loss_h + loss_{obj} + loss_{cls}$$

앞의 x, y, w, h에 대한 loss는 **MSE(Mean Square Error)**이다.

뒤의 Object와 Classification에 대한 loss는 **Cross-Entropy Loss**이다.

이를 참고하여 loss를 정의한다.

### - get\_loss\_batch 함수

mini-batch에 따른 loss값을 계산해주는 함수이다.

- output : YOLO-v3에서 나온 3가지 output들
- targets : GT, tensor shape : n\*6
- param\_loss : loss parameter
- opt : optimizer, default = None

```
def get_loss_batch(output, targets, params_loss, opt=None):
    # 필요 파라미터 정의
    ignore_thres=params_loss["ignore_thres"]
    scaled_anchors= params_loss["scaled_anchors"]
    mse_loss= params_loss["mse_loss"]
    bce_loss= params_loss["bce_loss"]

    num_yolos=params_loss["num_yolos"]
    num_anchors= params_loss["num_anchors"]
    obj_scale= params_loss["obj_scale"]
    noobj_scale= params_loss["noobj_scale"]

    loss=0.0
    # num_yolos만큼의 loop를 돌린다.
    # yolo_out(YOLO output), num_bbox(number of bounding box), grid_size
    # yolo_out : 507, 2028, 8112 bounding box
    # grid_size : 13, 26, 52
    for yolo_ind in range(num_yolos):
        yolo_out=output[yolo_ind]
        batch_size, num_bbxes, _=yolo_out.shape
```

```

# get grid size
gz_2=num_bbxs/num_anchors
grid_size=int(np.sqrt(gz_2))

# reshape (batch_size, 3, grid_size, grid_size, 85)
yolo_out=yolo_out.view(batch_size,num_anchors,grid_size,grid_size,-1)

# pred_boxes(predicted bounding boxes)
# pred_conf(objectness score)
# pred_cls_prob(class probabilities)
pred_boxes=yolo_out[:, :, :, :4]
x,y,w,h= transform_bbox(pred_boxes, scaled_anchors[yolo_ind])
pred_conf=yolo_out[:, :, :, 4]
pred_cls_prob=yolo_out[:, :, :, 5:]

yolo_targets = get_yolo_targets({
    "pred_cls_prob": pred_cls_prob,
    "pred_boxes":pred_boxes,
    "targets": targets,
    "anchors": scaled_anchors[yolo_ind],

    "ignore_thres": ignore_thres,
})

obj_mask=yolo_targets["obj_mask"]
noobj_mask=yolo_targets["noobj_mask"]
tx=yolo_targets["tx"]
ty=yolo_targets["ty"]
tw=yolo_targets["tw"]
th=yolo_targets["th"]
tcls=yolo_targets["tcls"]
t_conf=yolo_targets["t_conf"]

loss_x = mse_loss(x[obj_mask], tx[obj_mask])
loss_y = mse_loss(y[obj_mask], ty[obj_mask])
loss_w = mse_loss(w[obj_mask], tw[obj_mask])
loss_h = mse_loss(h[obj_mask], th[obj_mask])

loss_conf_obj = bce_loss(pred_conf[obj_mask], t_conf[obj_mask])
loss_conf_noobj = bce_loss(pred_conf[noobj_mask], t_conf[noobj_mask])

```

```

loss_conf = obj_scale * loss_conf_obj + noobj_scale * loss_conf_noobj
loss_cls = bce_loss(pred_cls_prob[obj_mask], tcls[obj_mask])
loss += loss_x + loss_y + loss_w + loss_h + loss_conf + loss_cls

if opt is not None:
    opt.zero_grad()
    loss.backward()
    opt.step()

return loss.item()

```

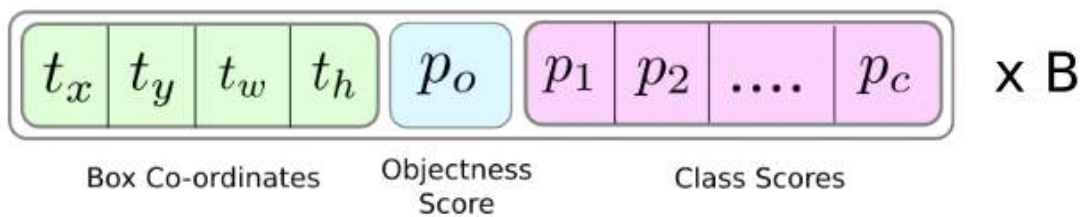


그림. <https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>

#### - transform\_bbox 함수 정의

predicted bounding box를 이용해 target값과 호환되도록 transform하는 함수

- bbox : (batch\_size, 3, grid\_size, grid\_size, 4)
- anchors : (3,2) 3개의 output에 대한 scaled widths, heights 포함..

ex) [[116,90],[156,198],[373,326]] =>

[[3.6250,2.8125],[4.8750,6.1875],[11.6562,10.1875]]

```

def transform_bbox(bbox, anchors):
    x=bbox[:, :, :, 0]
    y=bbox[:, :, :, 1]
    w=bbox[:, :, :, 2]
    h=bbox[:, :, :, 3]
    anchor_w = anchors[:, 0].view((1, 3, 1, 1))
    anchor_h = anchors[:, 1].view((1, 3, 1, 1))

    x=x-x.floor() # floor() : 내림
    y=y-y.floor()
    w= torch.log(w / anchor_w + 1e-16)
    h= torch.log(h / anchor_h + 1e-16) 왜?
    return x, y, w, h

```

- get\_yolo\_targets

```

def get_yolo_targets(params):
    pred_boxes=params["pred_boxes"]
    pred_cls_prob=params["pred_cls_prob"]
    target=params["targets"]
    anchors=params["anchors"]
    ignore_thres=params["ignore_thres"]

    batch_size = pred_boxes.size(0)
    num_anchors = pred_boxes.size(1)
    grid_size = pred_boxes.size(2)
    num_cls = pred_cls_prob.size(-1)

    sizeT=batch_size, num_anchors, grid_size, grid_size
    obj_mask = torch.zeros(sizeT,device=device,dtype=torch.bool)
    noobj_mask = torch.ones(sizeT,device=device,dtype=torch.bool)
    tx = torch.zeros(sizeT, device=device, dtype=torch.float32)
    ty= torch.zeros(sizeT, device=device, dtype=torch.float32)
    tw= torch.zeros(sizeT, device=device, dtype=torch.float32)
    th= torch.zeros(sizeT, device=device, dtype=torch.float32)

    sizeT=batch_size, num_anchors, grid_size, grid_size, num_cls

```

```

tcls= torch.zeros(sizeT, device=device, dtype=torch.float32)

target_bboxes = target[:, 2:] * grid_size
t_xy = target_bboxes[:, :2]
t_wh = target_bboxes[:, 2:]
t_x, t_y = t_xy.t() #transpose
t_w, t_h = t_wh.t()

grid_i, grid_j = t_xy.long().t() #long 타입 바꾸기

iou_with_anchors=[get_iou_WH(anchor, t_wh) for anchor in anchors]
iou_with_anchors = torch.stack(iou_with_anchors)
best_iou_wa, best_anchor_ind = iou_with_anchors.max(0)

batch_inds, target_labels = target[:, :2].long().t()
obj_mask[batch_inds, best_anchor_ind, grid_j, grid_i] = 1
noobj_mask[batch_inds, best_anchor_ind, grid_j, grid_i] = 0

for ind, iou_wa in enumerate(iou_with_anchors.t()):
    noobj_mask[batch_inds[ind], iou_wa > ignore_thres, grid_j[ind],
grid_i[ind]] = 0

tx[batch_inds, best_anchor_ind, grid_j, grid_i] = t_x - t_x.floor()
ty[batch_inds, best_anchor_ind, grid_j, grid_i] = t_y - t_y.floor()

anchor_w=anchors[best_anchor_ind][:, 0]
tw[batch_inds, best_anchor_ind, grid_j, grid_i] = torch.log(t_w / anchor_w +
1e-16)

anchor_h=anchors[best_anchor_ind][:, 1]
th[batch_inds, best_anchor_ind, grid_j, grid_i] = torch.log(t_h / anchor_h +
1e-16)

tcls[batch_inds, best_anchor_ind, grid_j, grid_i, target_labels] = 1

output={
    "obj_mask" : obj_mask,
    "noobj_mask" : noobj_mask,

```



```

        "tx": tx,
        "ty": ty,
        "tw": tw,
        "th": th,
        "tcls": tcls,
        "t_conf": obj_mask.float(),
    }
    return output

```

- get\_iou\_WH 함수 정의

**IoU(Intersection over Union) = 교집합 / 합집합**

- wh1 : anchor의 width, height (1,2)
- wh2 : targets bounding box의 width, height (n,2)

```

def get_iou_WH(wh1, wh2):
    wh2 = wh2.t() # (2,n)
    w1, h1 = wh1[0], wh1[1]
    w2, h2 = wh2[0], wh2[1]
    inter_area = torch.min(w1, w2) * torch.min(h1, h2)
    union_area = (w1 * h1 + 1e-16) + w2 * h2 - inter_area
    return inter_area / union_area

```

ex)

```

: import torch

wh1 = torch.tensor([3,10])
wh2 = torch.tensor([[3,4],[5,7]])

get_iou_WH(wh1,wh2)

: tensor([0.4000, 0.4773])

```

## Training the Model

- loss\_epoch 함수 정의

```

def loss_epoch(model,params_loss,dataset_dl,sanity_check=False,opt=None):
    running_loss=0.0
    len_data=len(dataset_dl.dataset)

```

```

running_metrics= {}

for xb, yb,_ in dataset_dl:
    yb=yb.to(device)
    _,output=model(xb.to(device))
    loss_b=get_loss_batch(output,yb, params_loss,opt)
    running_loss+=loss_b
    if sanity_check is True:
        break
loss=running_loss/float(len_data)
return loss

```

#### - train\_val 함수 정의

```

import copy
def train_val(model, params):
    num_epochs=params["num_epochs"]
    params_loss=params["params_loss"]
    opt=params["optimizer"]
    train_dl=params["train_dl"]
    val_dl=params["val_dl"]
    sanity_check=params["sanity_check"]
    lr_scheduler=params["lr_scheduler"]
    path2weights=params["path2weights"]

    loss_history={
        "train": [],
        "val": [],
    }
    best_model_wts = copy.deepcopy(model.state_dict())
    best_loss=float('inf')

    for epoch in range(num_epochs):
        current_lr=get_lr(opt)
        print('Epoch {}/ {}, current lr={}'.format(epoch, num_epochs - 1,
current_lr))
        model.train()
        train_loss=loss_epoch(model,params_loss,train_dl,sanity_check,opt)
        loss_history["train"].append(train_loss)
        print("train loss: %.6f" %(train_loss))

```

```

model.eval()
# val 단계에서는 굳이 gradient 계산을 할 필요가 없으니
# no_grad() 옵션을 한다.
with torch.no_grad():
    val_loss=loss_epoch(model,params_loss,val_dl,sanity_check)
loss_history["val"].append(val_loss)
print("val loss: %.6f" %(val_loss))

if val_loss < best_loss:
    best_loss = val_loss
    best_model_wts = copy.deepcopy(model.state_dict())
    torch.save(model.state_dict(), path2weights)
    print("Copied best model weights!")

# lr_scheduler 변화
lr_scheduler.step(val_loss)
if current_lr != get_lr(opt):
    print("Loading best model weights!")
    model.load_state_dict(best_model_wts)
print("-"*10)
model.load_state_dict(best_model_wts)
return model, loss_history

```

- get\_lr 함수 정의

```

def get_lr(opt):
    for param_group in opt.param_groups:
        return param_group['lr']

```

- 필요한 parameter 정의

```

from torch import optim
from torch.optim.lr_scheduler import ReduceLROnPlateau

opt = optim.Adam(model.parameters(), lr=1e-3)
lr_scheduler =
ReduceLROnPlateau(opt, mode='min',factor=0.5, patience=20,verbose=1)

path2models= "./models/"
if not os.path.exists(path2models):

```

```

os.mkdir(path2models)

scaled_anchors=[model.module_list[82][0].scaled_anchors,
                 model.module_list[94][0].scaled_anchors,
                 model.module_list[106][0].scaled_anchors]

```

- ReduceLROnPlateau

: 더 이상 학습이 진행되지 않을 때 learning rate를 감소시키는 scheduler이다.

- ReduceLROnPlateau(opt, mode='min', factor=0.5, patience=20, verbose=1)

: epoch 20동안 감소하면 learning rate를 절반으로 감소시킨다. 이때, get\_lr()가 따로 위 lr\_scheduler에 저장되어 있지 않은데 opt를 통해 learning rate를 받아올 수 있다고 한다.

- 각각의 loss 함수와 loss parameters 정의

```

mse_loss = nn.MSELoss(reduction="sum")
bce_loss = nn.BCELoss(reduction="sum")
params_loss={
    "scaled_anchors" : scaled_anchors,
    "ignore_thres": 0.5,
    "mse_loss": mse_loss,
    "bce_loss": bce_loss,
    "num_yolos": 3,
    "num_anchors": 3,
    "obj_scale": 1,
    "noobj_scale": 100,
}

```

- train, val 의 parameters 정의

```

params_train={
    "num_epochs": 5,
    "optimizer": opt,
    "params_loss": params_loss,
    "train_dl": train_dl,
    "val_dl": val_dl,
    "sanity_check": True,
    "lr_scheduler": lr_scheduler,
    "path2weights": path2models+"weights.pt",
}
model,loss_hist=train_val(model,params_train)

```

Epoch 0/4, current lr=0.001

train loss: 4.995495

val loss: 112.445463

Copied best model weights!

-----

Epoch 1/4, current lr=0.001

train loss: 4.736809

val loss: 104.389150

Copied best model weights!

-----

Epoch 2/4, current lr=0.001

train loss: 4.433953

val loss: 96.853119

Copied best model weights!

-----

Epoch 3/4, current lr=0.001

train loss: 4.249519

val loss: 90.667994

Copied best model weights!

-----

Epoch 4/4, current lr=0.001

train loss: 3.976205

val loss: 86.411213

Copied best model weights!

-----