# 예측 결과에 WBF(Weighted Boxes Fusion) 적용하여 csv파일로 저장

In [1]:

```
import os
import pandas as pd
import numpy as np
# for modeling
import torch
torch.manual_seed(0)
import matplotlib.patches as patches
import matplotlib.pyplot as plt
from PIL import Image
import torchvision
from torchvision import transforms, datasets, models
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
import time
import pickle
from tqdm import tqdm
```

In [2]:

```
os.chdir('/home/work/sample-notebooks/')
```

In [3]:

```
test_3 = os.listdir('./test_final/test/images/')
test_3.sort()
#del test_3[0]
print(len(test_3))
test_3[:5]
```

```
10000
```

Out[3]:

```
['sk_te_000000.jpg',
 'sk_te_000001.jpg',
 'sk_te_000002.jpg',
 'sk_te_000003.jpg',
 'sk_te_000004.jpg']
```

In [25]:

```
df_size = pd.DataFrame(columns=['H','W'])
h = []
w = []

for file_name in tqdm(test_3):
    test_img_path = './test_final/test/images/' + file_name
    img = Image.open(test_img_path) #.convert('RGB')

    h.append(img.size[1])   # H
    w.append(img.size[0])   # W

df_size['H'] = h
df_size['W'] = w
df_size
```

```
100%|██████████| 10000/10000 [00:03<00:00, 2633.75it/s]
```

Out[25]:

| | H | W |
|---|---|---|
| **0** | 240 | 320 |
| **1** | 611 | 1033 |
| **2** | 600 | 1064 |
| **3** | 611 | 1033 |

| | H | W |
|---|---|---|
| **4** | 800 | 982 |
| **...** | ... | ... |
| **9995** | 1080 | 1920 |
| **9996** | 611 | 1033 |
| **9997** | 611 | 1033 |
| **9998** | 800 | 982 |
| **9999** | 600 | 1064 |

10000 rows × 2 columns

# 데이터프레임 생성

**1.** 컬럼 **: img_name, score, x1, y1, x2, y2 (**제출 **CSV** 파일 내 **Header**로 포함**)**

**2.** 제출 파일명 **: submission_**우린_깐부잖아**_4.csv**

**3.** 형식 **: CSV(Comma**로 구분**)**

**4.** 제약사항

① 전체 제출 객체 수는 최대 2,000,000 개
② 제출 객체의 최소 크기(면적, normalized area of bounding box)는 0.0001 보다 커야함 (Area > 0.0001)★

**5.** 정렬 기준 **: img_name** 컬럼을 기준으로 오름차순으로 정렬

**6. bbox** 좌표값 **: 0**과 **1** 사이의 정규화된**(normalized)** 값 ★

In [4]:

```python
def generate_box(df_obj, size):  # 객체 하나씩 (한 이미지에 객체 여러개여도 하나씩)
    W = size[0]
    H = size[1]
    xmin = df_obj['xmin']*W
    ymin = df_obj['ymin']*H
    xmax = df_obj['xmax']*W
    ymax = df_obj['ymax']*H

    return [xmin, ymin, xmax, ymax]


def generate_label(df_obj): # 원래 label에 1씩 더해서 반환
    adjust_label = 1

    return int(df_obj['class'] + adjust_label)


def generate_target(file, size):
    # 텍스트 파일 불러오기
    df = pd.read_table(file, sep = ' ', header = None, names = ['class','xmin','ymin

    boxes = []
    labels = []
    for obj in range(df.shape[0]):

        boxes.append(generate_box(df.iloc[obj], size))
        labels.append(generate_label(df.iloc[obj]))
    boxes = torch.as_tensor(boxes, dtype = torch.float32)
    labels = torch.as_tensor(labels, dtype = torch.int64)

    target = {}
    target["boxes"] = boxes
    target["labels"] = labels

    return target
```

In [5]:

```
### Test Data Loader

class MaskDataset(object):
    def __init__(self, transforms, path, imgs):
        self.transforms = transforms
        self.path = path  # img path
        self.imgs = imgs   # img 파일명 list

    def __getitem__(self, idx):
        # load image and masks
        file_image = self.imgs[idx]
        img_path = os.path.join(self.path, file_image)

        img = Image.open(img_path).convert('RGB')

        if self.transforms is not None:
            img = self.transforms(img)

        target = 0
        return img, target

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

data_transform = transforms.Compose([
    transforms.ToTensor()
    ])


def collate_fn(batch):
    return tuple(zip(*batch))

test_dataset = MaskDataset(data_transform, './test_final/test/images/', test_3)

test_data_loader = torch.utils.data.DataLoader(test_dataset, batch_size = 2, collate_
```

In [6]:

```
def get_model_instance_segmentation(num_classes):  # num_classes 는 background 클래스

    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained = True)
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    return model
```

In [7]:

```
model = get_model_instance_segmentation(8)  # 실제 클래스 개수 : 7 (0~6)

device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)
```

Out[7]:

```
FasterRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=F
alse)
      (bn1): FrozenBatchNorm2d(64)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=F
alse)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
          (bn2): FrozenBatchNorm2d(64)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): FrozenBatchNorm2d(256)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
          (bn2): FrozenBatchNorm2d(64)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
          (bn2): FrozenBatchNorm2d(64)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256)
          (relu): ReLU(inplace=True)
        )
      )
      (layer2): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(128)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): FrozenBatchNorm2d(512)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(128)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(128)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(128)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(128)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(512)
          (relu): ReLU(inplace=True)
        )
      )
      (layer3): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): FrozenBatchNorm2d(1024)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (4): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
        (5): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(256)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(256)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(1024)
          (relu): ReLU(inplace=True)
        )
      )
      (layer4): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(512)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(512)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(2048)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): FrozenBatchNorm2d(2048)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(512)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(512)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(2048)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(512)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
          (bn2): FrozenBatchNorm2d(512)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(2048)
          (relu): ReLU(inplace=True)
        )
      )
    )
    (fpn): FeaturePyramidNetwork(
      (inner_blocks): ModuleList(
        (0): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
        (2): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
      )
      (layer_blocks): ModuleList(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (extra_blocks): LastLevelMaxPool()
    )
  )
  (rpn): RegionProposalNetwork(
    (anchor_generator): AnchorGenerator()
    (head): RPNHead(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cls_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (roi_heads): RoIHeads(
    (box_roi_pool): MultiScaleRoIAlign()
    (box_head): TwoMLPHead(
      (fc6): Linear(in_features=12544, out_features=1024, bias=True)
      (fc7): Linear(in_features=1024, out_features=1024, bias=True)
    )
    (box_predictor): FastRCNNPredictor(
      (cls_score): Linear(in_features=1024, out_features=8, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=32, bias=True)
    )
  )
)
```

In [9]:

```
#num_epochs = 10
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.Adam(params, lr = 0.005, weight_decay = 0.0005)
```

In [103…

```
model.load_state_dict(torch.load(f'./Miso/weight/model_140.pt'))
```

Out[103…

```
<All keys matched successfully>
```

## 예측

In [11]:

```
def make_prediction(model, img, threshold):
    model.eval()
    preds = model(img)
    for id in range(len(preds)): # batch size = 2
        idx_list = []

        for idx, score in enumerate(preds[id]['scores']):  # 한 이미지 내에 검출된 객
            if score > threshold:  # 신뢰도가 threshold보다 높은것만 idx_list에 저장
                idx_list.append(idx)

        # thr보다 높은 객체들만 boxes, labels, scores 정보 추출해서 덮어쓰기로 저장
        preds[id]['boxes'] = preds[id]['boxes'][idx_list]
        preds[id]['labels'] = preds[id]['labels'][idx_list]-1  # label : 0-6
        preds[id]['scores'] = preds[id]['scores'][idx_list]
    return preds
```

In [12]:

```
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

In [104…

```
pred_list = []
with torch.no_grad():
    for imgs, annotations in tqdm(test_data_loader):
        imgs = list(img.to(device) for img in imgs)

        pred = make_prediction(model, imgs, 0.2) # threshold   # imgs: [C, H, W]
        pred_list.append(pred)
```

```
100%|██████████| 5000/5000 [49:27<00:00,  1.68it/s]
```

## WBF (Weighted Boxes Fusion)

In [45]:

```
def make_annots_list(pred_list, df_size):
    img_num = 0
    boxes_list = []
    scores_list = []
    labels_list = []
    for pred in tqdm(pred_list):
        for pr in pred:
            W = df_size.iloc[img_num]['W']
            H = df_size.iloc[img_num]['H']
            temp_boxes = []
            temp_scores = []
            temp_labels = []

            for i, p in enumerate(pr['boxes']):
                xmin = float(p[0])/W
                ymin = float(p[1])/H
                xmax = float(p[2])/W
                ymax = float(p[3])/H

                temp_boxes.append([xmin,ymin,xmax,ymax])

                temp_scores.append(float(pr['scores'][i]))

                temp_labels.append(int(pr['labels'][i]))

            boxes_list.append(temp_boxes)
            scores_list.append(temp_scores)
            labels_list.append(temp_labels)

            img_num += 1

    return boxes_list, scores_list, labels_list
```

In [113…

```
# pickle 파일 로드
with open('./wbf_weight_ssl_model_1_04.pkl', 'rb') as f:
    pred_list_1 = pickle.load(f)
```

In [114…

```
# pred_list <1> - weight_ssl/model_1.pt (threshold = 0.4)
boxes_list_1, scores_list_1, labels_list_1 = make_annots_list(pred_list_1, df_size)
```

```
100%|██████████| 5000/5000 [00:34<00:00, 145.90it/s]
```

In [46]:

```
# pred_list <2> - weight_ssl/model_1.pt (threshold = 0.2)
boxes_list_2, scores_list_2, labels_list_2 = make_annots_list(pred_list_2, df_size)
```

```
100%|██████████| 5000/5000 [00:49<00:00, 100.27it/s]
```

In [95]:

```
# pickle 파일 로드
with open('./wbf_weight_model_100_0.pkl', 'rb') as f:
    pred_list_3 = pickle.load(f)
```

In [96]:

```
# pred_list <3> - weight/model_100.pt (threshold = 0)
boxes_list_3, scores_list_3, labels_list_3 = make_annots_list(pred_list_3, df_size)
```

```
100%|██████████| 5000/5000 [00:45<00:00, 110.77it/s]
```

In [105…

```
# pred_list copy 저장
pred_list_4 = pred_list.copy()
```

In [106…

```
# pred_list <4> - weight/model_140.pt (threshold = 0.2)
boxes_list_4, scores_list_4, labels_list_4 = make_annots_list(pred_list_4, df_size)
```

```
100%|██████████| 5000/5000 [00:31<00:00, 156.81it/s]
```

## WBF 적용 후 결과 dataframe 생성

In [107…

```
## boxes_list_1, boxes_list_2, ... 이미지 하나씩 불러서 wbf 적용후 결과 바로 저장
#========== <2,3,4> ==========
from ensemble_boxes import weighted_boxes_fusion
img_len = len(boxes_list_2)
weights = [3,2,1]
iou_thr = 0.5
skip_box_thr = 0.0001

df_result = pd.DataFrame(columns=['img_name', 'class_id', 'score', 'x1', 'y1', 'x2',

x1 = []; y1 = []; x2 = []; y2 = []
class_id = []; score = []
img_name = []
small_num = 0; img_num = 0

for i in tqdm(range(img_len)):
    boxes_list_sum =  [boxes_list_2[i]] + [boxes_list_3[i]] + [boxes_list_4[i]]
    scores_list_sum = [scores_list_2[i]] + [scores_list_3[i]] + [scores_list_4[i]]
    labels_list_sum = [labels_list_2[i]] + [labels_list_3[i]] + [labels_list_4[i]]

    boxes, scores, labels = weighted_boxes_fusion(boxes_list_sum, scores_list_sum, lab
                                                  weights=weights, iou_thr=iou_thr, skip_b

    # 바로 결과 제출용 csv로 저장
    W = df_size.iloc[img_num]['W']
    H = df_size.iloc[img_num]['H']
    obj_num = len(boxes)

    for i in range(obj_num):
        # bbox
        xmin = float(boxes[i][0])
        ymin = float(boxes[i][1])
        xmax = float(boxes[i][2])
        ymax = float(boxes[i][3])

        if (xmax-xmin)*(ymax-ymin) > 0.0001:   # normalized area of bbox > 0.0001
            x1.append(xmin)
            y1.append(ymin)
            x2.append(xmax)
            y2.append(ymax)
        else:
            small_num += 1
            continue

        # img_name
        img_name.append(test_3[img_num])

        # class_id
        class_id.append(int(labels[i]))

        # score
        score.append(float(scores[i]))

    img_num += 1

print("bbox의 normalized area가 0.0001보다 작은 경우(포함X):", small_num)
print("img_name 길이:", len(img_name))
print("class_id 길이:", len(class_id))
print("score 길이:", len(score))
print("x1 길이:", len(x1))
print("y1 길이:", len(y1))
print("x2 길이:", len(x2))
print("y2 길이:", len(y2))

df_result['img_name'] = img_name
df_result['class_id'] = class_id
df_result['score'] = score
df_result['x1'] = x1
df_result['y1'] = y1
df_result['x2'] = x2
df_result['y2'] = y2
df_result
```

```
  0%|          | 0/10000 [00:00<?, ?it/s]/opt/conda/lib/python3.7/site-packages/ensemb
le_boxes/ensemble_boxes_wbf.py:85: UserWarning: Y2 > 1 in box. Set it to 1. Check that
you normalize boxes in [0, 1] range.
  warnings.warn('Y2 > 1 in box. Set it to 1. Check that you normalize boxes in [0, 1]
range.')
100%|██████████| 10000/10000 [01:02<00:00, 159.60it/s]
bbox의 normalized area가 0.0001보다 작은 경우(포함X): 213
img_name 길이: 546323
class_id 길이: 546323
score 길이: 546323
x1 길이: 546323
y1 길이: 546323
x2 길이: 546323
y2 길이: 546323
```

Out[107…

| | img_name | class_id | score | x1 | y1 | x2 | y2 |
|---|---|---|---|---|---|---|---|
| **0** | sk_te_000000.jpg | 0 | 0.222756 | 0.018926 | 0.454982 | 0.150957 | 0.872141 |
| **1** | sk_te_000000.jpg | 1 | 0.204719 | 0.046209 | 0.837213 | 0.249866 | 0.997759 |
| **2** | sk_te_000000.jpg | 3 | 0.202387 | 0.048330 | 0.842149 | 0.237080 | 0.994879 |
| **3** | sk_te_000000.jpg | 0 | 0.169251 | 0.380457 | 0.025875 | 0.399217 | 0.080578 |

| | img_name | class_id | score | x1 | y1 | x2 | y2 |
|---|---|---|---|---|---|---|---|
| **4** | sk_te_000000.jpg | 0 | 0.159782 | 0.053086 | 0.824863 | 0.240058 | 0.999641 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **546318** | sk_te_009999.jpg | 0 | 0.828393 | 0.301744 | 0.782215 | 0.372072 | 0.994465 |
| **546319** | sk_te_009999.jpg | 0 | 0.293448 | 0.576365 | 0.167545 | 0.618299 | 0.289454 |
| **546320** | sk_te_009999.jpg | 0 | 0.201851 | 0.556465 | 0.110726 | 0.568553 | 0.157259 |
| **546321** | sk_te_009999.jpg | 0 | 0.108446 | 0.280618 | 0.421066 | 0.328966 | 0.759917 |
| **546322** | sk_te_009999.jpg | 0 | 0.079244 | 0.150159 | 0.674817 | 0.236831 | 0.969700 |

546323 rows × 7 columns

## pred_list로 바로 결과 dataframe 생성 (WBF 적용X)

In [83]:

```
df_result = pd.DataFrame(columns=['img_name', 'class_id', 'score', 'x1', 'y1', 'x2',
x1 = []; y1 = []; x2 = []; y2 = []
class_id = []; score = []
img_name = []
small_num = 0; img_num = 0

for prd in tqdm(pred_list): # pred_list길이: 1000, prd길이: 2
    for pr in prd:  # prd길이: 2

        W = df_size.iloc[img_num]['W']
        H = df_size.iloc[img_num]['H']
        obj_num = len(pr['boxes'])

        for i in range(obj_num):
            # bbox
            xmin = float(pr['boxes'][i][0])/W
            ymin = float(pr['boxes'][i][1])/H
            xmax = float(pr['boxes'][i][2])/W
            ymax = float(pr['boxes'][i][3])/H

            if (xmax-xmin)*(ymax-ymin) > 0.0001:   # normalized area of bbox > 0.0001
                x1.append(xmin)
                y1.append(ymin)
                x2.append(xmax)
                y2.append(ymax)
            else:
                small_num += 1
                continue

            # img_name
            img_name.append(test_3[img_num])

            # class_id
            class_id.append(int(pr['labels'][i]))

            # score
            score.append(float(pr['scores'][i]))

        img_num += 1
print("bbox의 normalized area가 0.0001보다 작은 경우(포함X):", small_num)
print("img_name 길이:", len(img_name))
print("class_id 길이:", len(class_id))
print("score 길이:", len(score))
print("x1 길이:", len(x1))
print("y1 길이:", len(y1))
print("x2 길이:", len(x2))
print("y2 길이:", len(y2))

df_result['img_name'] = img_name
df_result['class_id'] = class_id
df_result['score'] = score
df_result['x1'] = x1
df_result['y1'] = y1
df_result['x2'] = x2
df_result['y2'] = y2
df_result
```

```
100%|██████████| 5000/5000 [00:53<00:00, 94.21it/s]
bbox의 normalized area가 0.0001보다 작은 경우(포함X): 8
img_name 길이: 412668
class_id 길이: 412668
score 길이: 412668
x1 길이: 412668
y1 길이: 412668
x2 길이: 412668
y2 길이: 412668
```

Out[83]:

| | img_name | class_id | score | x1 | y1 | x2 | y2 |
|---|---|---|---|---|---|---|---|
| **0** | sk_te_000000.jpg | 0 | 0.338502 | 0.380457 | 0.025875 | 0.399217 | 0.080578 |
| **1** | sk_te_000000.jpg | 0 | 0.298765 | 0.650179 | 0.495710 | 0.683861 | 0.620811 |
| **2** | sk_te_000000.jpg | 0 | 0.248706 | 0.766455 | 0.089726 | 0.801147 | 0.240912 |
| **3** | sk_te_000000.jpg | 3 | 0.227225 | 0.798314 | 0.442519 | 0.985449 | 0.965378 |
| **4** | sk_te_000001.jpg | 0 | 0.999386 | 0.328962 | 0.207844 | 0.346477 | 0.253400 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **412663** | sk_te_009998.jpg | 5 | 0.202231 | 0.940465 | 0.473519 | 1.000000 | 0.513730 |
| **412664** | sk_te_009998.jpg | 1 | 0.200535 | 0.436289 | 0.181147 | 0.456263 | 0.203959 |
| **412665** | sk_te_009999.jpg | 0 | 0.715858 | 0.298529 | 0.788750 | 0.371004 | 1.000000 |
| **412666** | sk_te_009999.jpg | 0 | 0.586895 | 0.576365 | 0.167545 | 0.618299 | 0.289454 |
| **412667** | sk_te_009999.jpg | 0 | 0.290992 | 0.556421 | 0.109966 | 0.568312 | 0.156839 |

412668 rows × 7 columns

In [122…

```
df_result['x1'][df_result['x1'] > 1]
```

Out[122…

```
Series([], Name: x1, dtype: float64)
```

In [123…

```
df_result['y1'][df_result['y1'] > 1]
```

Out[123…

```
Series([], Name: y1, dtype: float64)
```

In [124…

```
df_result['x2'][df_result['x2'] > 1]  # 1.0은 무시 (~=1.000000000001)
```

Out[124…

```
61       1.0
11479    1.0
29242    1.0
40075    1.0
```

```
78617     1.0
86239     1.0
88646     1.0
92480     1.0
127043    1.0
138290    1.0
138719    1.0
146644    1.0
156820    1.0
169797    1.0
172985    1.0
189095    1.0
217182    1.0
225445    1.0
230820    1.0
240289    1.0
245899    1.0
248982    1.0
251068    1.0
260738    1.0
264348    1.0
264655    1.0
278388    1.0
281524    1.0
300376    1.0
300442    1.0
302871    1.0
309126    1.0
311617    1.0
329216    1.0
329293    1.0
347277    1.0
382422    1.0
384498    1.0
387496    1.0
404711    1.0
419057    1.0
451891    1.0
453852    1.0
461080    1.0
465795    1.0
478001    1.0
481435    1.0
494942    1.0
498716    1.0
506246    1.0
Name: x2, dtype: float64
```

In [125…

```
df_result['y2'][df_result['y2'] > 1]
```

Out[125…

```
2445      1.0
32669     1.0
35618     1.0
40329     1.0
61918     1.0
         ...
522496    1.0
523891    1.0
525647    1.0
529008    1.0
546299    1.0
Name: y2, Length: 71, dtype: float64
```

## csv로 저장

In [31]:

```
# weight_ssl/model_1.pt (threshold = 0.4)
df_result.to_csv('../result/submission_우린_깐부잖아_17.csv', index = False, sep = '
```

In [88]:

```
# weight_ssl/model_1.pt (threshold = 0.2) (결과: 0.5039)
df_result.to_csv('../result/submission_우린_깐부잖아_18.csv', index = False, sep = '
```

In [102…

```
# wbf (2,3) (결과: 0.5329)
df_result.to_csv('../result/submission_우린_깐부잖아_19.csv', index = False, sep = '
```

**wbf적용 (2,3,4) (결과: 0.5377) - 최고점수**

In [112…

```
# wbf (2,3,4) (결과: 0.5377)
df_result.to_csv('../result/submission_우린_깐부잖아_20.csv', index = False, sep = '
```

In [ ]:

In [120…

```
# wbf (1,2,3,4)   -weight 비 [2,3,2,1]
df_result.to_csv('../result/submission_우린_깐부잖아_21.csv', index = False, sep = '
```

In [126…

```
# wbf (1,2,3,4)   -weight 비 [3,4,2,1]
df_result.to_csv('../result/submission_우린_깐부잖아_22.csv', index = False, sep = '
```

In [ ]: