annotation 구한 unlabeled data + labeled data로 모델 학습

In [8]:

```
import os
import numpy as np
import pandas as pd
import pickle
```

In [2]:

```
# 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
from tqdm import tqdm
import csv
```

In [3]:

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

labeled data(train, test) 파일명 list 로드

In [4]:

```
# train 데이터
with open('./labeled_data/train_img.pkl', 'rb') as file:
    train_img_list = pickle.load(file)

# test 데이터
with open('./labeled_data/test_img.pkl', 'rb') as file:
    test_img_list = pickle.load(file)
```

unlabeled data 파일명 list 생성

In [5]:

```
ul_img_path = './unlabeled_data/'
ul_img_list = os.listdir(ul_img_path)
ul_img_list.sort()
del ul_img_list[-1]
print(len(ul_img_list))
print(ul_img_list[:5])
```

```
19000
['sk_ul_000000.jpg', 'sk_ul_000001.jpg', 'sk_ul_000002.jpg', 'sk_ul_000003.jpg', 'sk_u
l_000004.jpg']
```

In [6]:

```
# train_img_list와 ul_img_list 합치기
ssl_train_img = train_img_list + ul_img_list
print(len(ssl_train_img))
print(ssl_train_img[:2], ssl_train_img[-2:])
```

```
19900
['sk_tr_000863.jpg', 'sk_tr_000822.jpg'] ['sk_ul_018998.jpg', 'sk_ul_018999.jpg']
```

데이터셋 클래스 정의

In [9]:

```
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):
    if 'ul' in file:
        df = pd.read_table(file, sep = ',', header = None, names = ['class','xmin','
    else:
        df = pd.read_table(file, sep = ' ', header = None, names = ['class','xmin','

    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 [10]:

```
## <labeled_train + unlabeled (19900개)>, <labeled_test(100개)>
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]
        file_label = self.imgs[idx][:-3] + 'txt'

        if 'tr' in file_image:  ## labeled
            img_path = os.path.join(self.path, file_image)

            if 'test' in self.path:  ## labeled - test
                label_path = os.path.join('./labeled_data/test_annotations/', file_la
            else:                    ## labeled - train
                label_path = os.path.join('./labeled_data/train_annotations/', file_l

        elif 'ul' in file_image:  ## unlabeled
            self.path = './unlabeled_data/'
            img_path = os.path.join(self.path, file_image)
            label_path = os.path.join('./ul_annots_rate_ssl/', file_label)
        img = Image.open(img_path).convert('RGB')
        size = img.size

        # generate label
        target = generate_target(label_path, size)

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

        return img, target

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

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


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

dataset = MaskDataset(data_transform, './labeled_data/train_images/', ssl_train_img) #
test_dataset = MaskDataset(data_transform, './labeled_data/test_images/', test_img_lis

data_loader = torch.utils.data.DataLoader(dataset, batch_size = 2, collate_fn = colla
test_data_loader = torch.utils.data.DataLoader(test_dataset, batch_size = 1, collate_
```

모델 불러오기

In [11]:

```
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
```

전이학습

In [12]:

```
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)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipykernel_10120/2218573399.py in <module>
      2 
      3 device = torch.device('cuda') if torch.cuda.is_available() else torch.device(
'cpu')
----> 4 model.to(device)

/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in to(self, *args, *
*kwargs)
    610             return t.to(device, dtype if t.is_floating_point() else None, non
_blocking)
    611 
--> 612         return self._apply(convert)
    613 
    614     def register_backward_hook(
/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in _apply(self, fn)
    357     def _apply(self, fn):
    358         for module in self.children():
--> 359             module._apply(fn)
    360 
    361         def compute_should_use_set_data(tensor, tensor_applied):

/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in _apply(self, fn)
    357     def _apply(self, fn):
    358         for module in self.children():
--> 359             module._apply(fn)
    360 
    361         def compute_should_use_set_data(tensor, tensor_applied):

/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in _apply(self, fn)
    357     def _apply(self, fn):
    358         for module in self.children():
--> 359             module._apply(fn)
    360 
    361         def compute_should_use_set_data(tensor, tensor_applied):

/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in _apply(self, fn)
    379                 # `with torch.no_grad():`
    380                 with torch.no_grad():
--> 381                     param_applied = fn(param)
    382                 should_use_set_data = compute_should_use_set_data(param, para
m_applied)
    383                 if should_use_set_data:

/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in convert(t)
    608             if convert_to_format is not None and t.dim() == 4:
    609                 return t.to(device, dtype if t.is_floating_point() else None,
non_blocking, memory_format=convert_to_format)
--> 610             return t.to(device, dtype if t.is_floating_point() else None, non_
blocking)
    611 
    612         return self._apply(convert)

RuntimeError: CUDA error: out of memory
```

학습

In [13]:

```
num_epochs = 1
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.Adam(params, lr = 0.005, weight_decay = 0.0005)
#optimizer = torch.optim.SGD(params, lr = 0.005, momentum = 0.9, weight_decay = 0.0005
```

In [14]:

```
# 새로 학습 (start:0)
print('------------------------train start------------------------')
for epoch in range(num_epochs):
    start = time.time()
    model.train()
    i = 0
    epoch_loss = 0
    #numb = 0
    for imgs, annotations in tqdm(data_loader):
        i += 1
        imgs = list(img.to(device) for img in imgs)
        annotations = [{k: v.to(device) for k,v in t.items()} for t in annotations]
        try:
            loss_dict = model(imgs, annotations)
            losses = sum(loss for loss in loss_dict.values())
        except:
            #print(numb)
            continue


        optimizer.zero_grad()
        losses.backward()
        optimizer.step()
        epoch_loss += losses
        #numb += 1
    print(f'epoch : {epoch + 1}, Loss : {epoch_loss}, time : {time.time() - start}')

    torch.save(model.state_dict(), f'../Miso/weight_ssl/model_{epoch+1}.pt')
```

```
------------------------train start------------------------
100%|██████████| 9950/9950 [42:25<00:00,  3.91it/s]
epoch : 1, Loss : 0, time : 2545.747303247452
```

In [41]:

```
# 학습 과정을 반복해야 하므로 Train 함수 생성
def Train(start_epoch, end_epoch):
    print('------------------------train start------------------------')
    # 이전 가중치 불러오기
    model.load_state_dict(torch.load(f'../Miso/weight_ssl/model_{start_epoch}.pt'))
    for epoch in range(start_epoch, end_epoch):
        start = time.time()
        model.train()
        i = 0
        epoch_loss = 0
        for imgs, annotations in tqdm(data_loader):
            i += 1
            imgs = list(img.to(device) for img in imgs)
            annotations = [{k: v.to(device) for k,v in t.items()} for t in annotatic
            try:
                loss_dict = model(imgs, annotations)
                losses = sum(loss for loss in loss_dict.values())
            except:
                continue

            optimizer.zero_grad()
            losses.backward()
            optimizer.step()
            epoch_loss += losses
        print(f'epoch : {epoch + 1}, Loss : {epoch_loss}, time : {time.time() - start
        torch.save(model.state_dict(), f'../Miso/weight_ssl/model_{epoch + 1}.pt')
```

- 학습

In [ ]:

```
Train(1,2)
```

In [42]:

```
# 중간에 optimizer 바꿔도 되는지 확인
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.Adam(params, lr = 0.001, weight_decay = 0.0005)
# momentum 0.95, 0.99로 증가시키며 사용
# lr = 0.005 -> lr = 0.001
Train(2,3)
```

```
------------------------train start------------------------
100%|██████████| 9950/9950 [1:40:49<00:00,  1.64it/s]
epoch : 3, Loss : 273555191234560.0, time : 6049.940295219421
```

모델에 weight 로드

In [ ]:

```
model.load_state_dict(torch.load(f'../Miso/weight_ssl/model_1.pt'))
```

예측

In [29]:

```
def make_prediction(model, img, threshold):
    model.eval()
    preds = model(img)
    for id in range(len(preds)):
        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 [33]:

```
with torch.no_grad():
    for imgs, annotations in test_data_loader:
        im = imgs
        imgs = list(img.to(device) for img in imgs)

        pred = make_prediction(model, imgs, 0.4) # threshold
        print(pred)
        break   # test_data_loader의 첫 번째 배치만 결과 출력
```

```
[{'boxes': tensor([[ 703.4944,   59.9619,  725.3139,   99.6871],
        [ 774.2667,   55.1289,  796.3938,   99.5487],
        [ 880.0817,  209.7026, 1001.0000,  321.6533],
        [ 554.2569,   20.5485,  590.6553,   49.1402],
        [ 312.9951,  132.5156,  339.6205,  185.6905],
        [ 775.7050,   69.8275,  794.6923,  108.3002],
        [ 449.7556,   18.2737,  483.4883,   32.0022],
        [ 752.9364,   19.0609,  782.9688,   44.3429],
        [ 956.2925,   91.0808,  991.5442,  129.3998],
        [ 485.5266,   18.3976,  520.0612,   33.1010],
        [ 769.5856,   21.2229,  811.9795,   49.7759],
        [ 312.1575,  145.5818,  337.2722,  194.1991],
        [ 709.4665,  296.4266,  855.6778,  434.6549],
        [ 822.0977,   48.4249,  878.2283,   94.5753],
        [ 866.3963,   40.7793,  886.5568,   65.0910],
        [ 796.1738,   42.0485,  839.4514,   79.5353],
        [ 705.5055,   74.3117,  722.3051,  102.3828],
        [ 868.8712,   32.0100,  946.5863,   65.4043],
        [ 827.4374,   80.7260,  838.3604,  101.3921],
        [ 477.8173,   19.3210,  488.2294,   35.6006],
        [ 514.8702,   18.9247,  527.0136,   33.8235],
        [ 557.6417,   19.4193,  587.9046,   31.1005],
        [ 515.4282,   19.2760,  526.1896,   35.8974],
        [ 773.1207,   47.7967,  792.6917,   86.2238],
        [ 777.9705,   23.8157,  813.5848,   56.7420],
        [ 816.8560,   52.9844,  830.9999,   89.8972]], device='cuda:0'), 'labels': ten
sor([0, 0, 1, 1, 0, 6, 1, 1, 5, 1, 1, 6, 1, 1, 1, 1, 6, 1, 6, 1, 1, 1, 6, 0,
        3, 0], device='cuda:0'), 'scores': tensor([0.9992, 0.9981, 0.9974, 0.9972, 0.9
970, 0.9969, 0.9951, 0.9930, 0.9924,
        0.9920, 0.9903, 0.9899, 0.9897, 0.9854, 0.9786, 0.9756, 0.9632, 0.9591,
        0.9361, 0.7834, 0.7581, 0.6922, 0.6107, 0.5900, 0.4690, 0.4012],
       device='cuda:0')}]
```

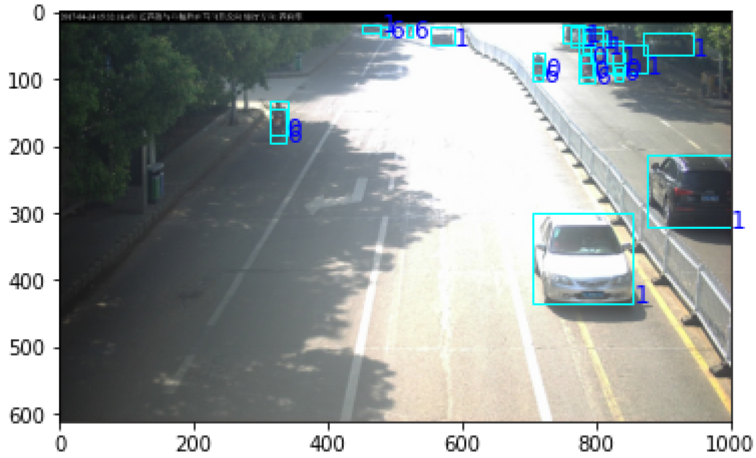
In [34]:

```
def plot_image_from_output(img, annotation):
    img = img.cpu().permute(1,2,0)  # 0위치:2에 있던게 옴, 1위치:0, 2위치:1 으로 차원

    fig, ax = plt.subplots(1)
    ax.imshow(img)  # (M,N,3) : M=row, N=col

    for idx in range(len(annotation["boxes"])):
        xmin, ymin, xmax, ymax = annotation["boxes"][idx]
        xmin = xmin.cpu().data.numpy()
        ymin = ymin.cpu().data.numpy()
        xmax = xmax.cpu().data.numpy()
        ymax = ymax.cpu().data.numpy()

        #print("xmin,ymin,xmax,ymax :", xmin,ymin,xmax,ymax)
        rect = patches.Rectangle((xmin,ymin),(xmax-xmin),(ymax-ymin), linewidth=1, ed
        label = int(annotation['labels'][idx])
        plt.text(xmax, ymax, label, color = 'blue', size = 12)
        ax.add_patch(rect)
    plt.show()
```

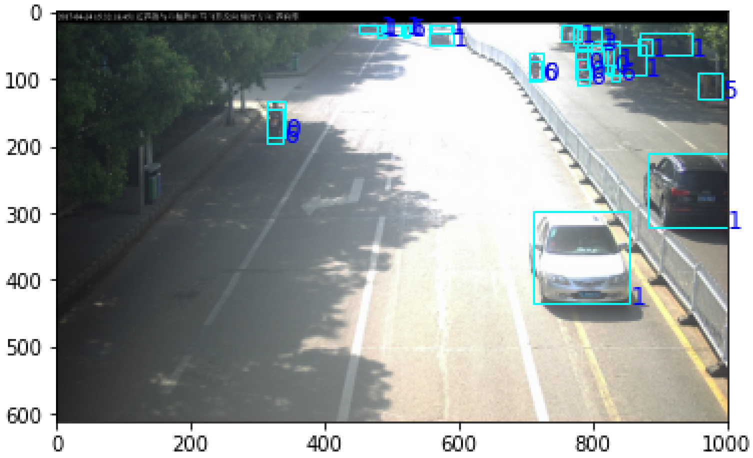
In [40]:

```
_idx = 0
print("Target :", annotations[_idx]['labels'])  # Ground Truth 값 (-1 해야됨)
plot_image_from_output(imgs[_idx], annotations[_idx])  # imgs : test_data_loader에서 t

print("Prediction :", pred[_idx]['labels'])
plot_image_from_output(imgs[_idx], pred[_idx])
```

```
Target : tensor([0, 0, 0, 0, 0, 1, 1, 6, 6, 6, 1, 1, 1, 1, 1, 1, 1, 1, 6, 6, 1, 6])
```



```
Prediction : tensor([0, 0, 1, 1, 0, 6, 1, 1, 5, 1, 1, 6, 1, 1, 1, 1, 6, 1, 6, 1, 1, 1,
6, 0,
        3, 0], device='cuda:0')
```



```
In [ ]:
```