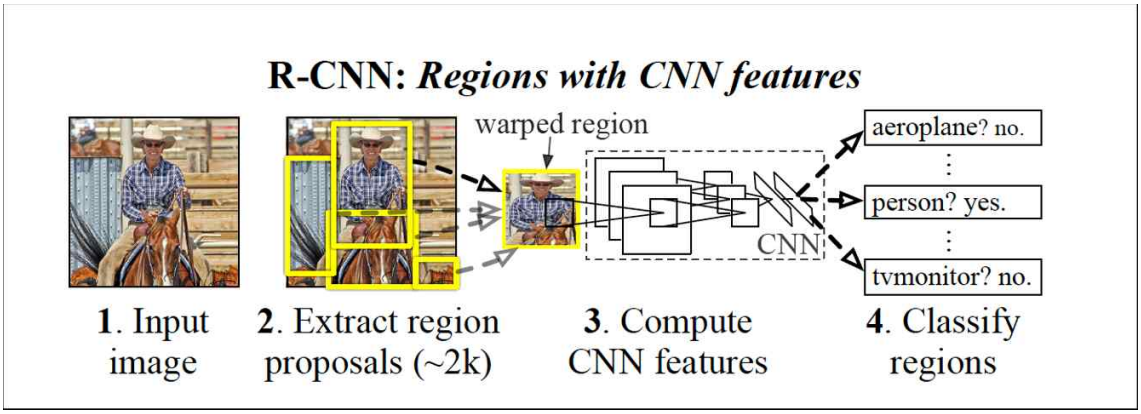
Chapter7. Basics of Object Detection

2021210088 허지혜

Training R-CNN-based custom object detectors

논문 이름 : Ross Girshich, Jeff Donahue, Trevor Darrell, Jitendra Malik et al, "Rich feature hierarchies for accurate onject detection and semantic segmentation", **R-CNN(Region-based Convolution Neural Network)**



코드

- 패키지 불러오기

```
!pip install -q --upgrade selectivesearch torch_snippets
from torch_snippets import *
import selectivesearch
from torchvision import transforms, models, datasets
from torch_snippets import Report
from torchvision.ops import nms
#device = 'cuda' if torch.cuda.is_available() else 'cpu'
device='cpu'
```

- 데이터 불러오기

• 데이터 설명

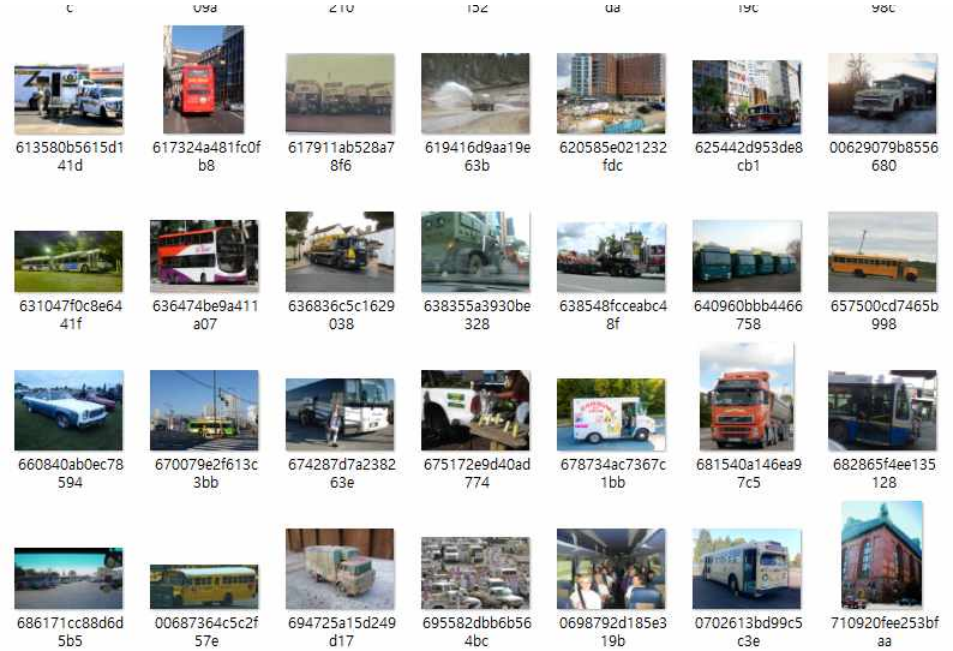


그림 2.https://www.kaggle.com/datasets/sixhky/open-images-bus-trucks

데이터는 kaggle에서 다운받았다. image가 들어있는 **images 폴더**와 bounding box 좌표, image_id, image_path 등이 들어있는 bus와 truck이 있는 **df.csv 파일**이 있다.
먼저, df.csv 파일을 살펴보자.

```
IMAGE_ROOT = './data/images/images'
DF_RAW = pd.read_csv('./data/df.csv')
print(DF_RAW.head())
```

```
             ImageID  Source LabelName  Confidence      XMin      XMax  \
0  0000599864fd15b3  xclick       Bus           1  0.343750  0.908750
1  00006bdb1eb5cd74  xclick     Truck           1  0.276667  0.697500
2  00006bdb1eb5cd74  xclick     Truck           1  0.702500  0.999167
3  0001Obf498b64bab  xclick       Bus           1  0.156250  0.371250
4  00013f14dd4e168f  xclick       Bus           1  0.287500  0.999375

       YMin      YMax  IsOccluded  IsTruncated  ...  IsDepiction  IsInside  \
0  0.156162  0.650047           1            0  ...            0         0
1  0.141604  0.437343           1            0  ...            0         0
2  0.204261  0.409774           1            1  ...            0         0
3  0.269188  0.705228           0            0  ...            0         0
4  0.194184  0.999062           0            1  ...            0         0

   XClick1X  XClick2X  XClick3X  XClick4X  XClick1Y  XClick2Y  XClick3Y  \
0  0.421875  0.343750  0.795000  0.908750  0.156162  0.512700  0.650047
1  0.299167  0.276667  0.697500  0.659167  0.141604  0.241855  0.352130
2  0.849167  0.702500  0.906667  0.999167  0.204261  0.398496  0.409774
3  0.274375  0.371250  0.311875  0.156250  0.269188  0.493882  0.705228
4  0.920000  0.999375  0.648750  0.287500  0.194184  0.303940  0.999062

   XClick4Y
0  0.457197
1  0.437343
2  0.295739
3  0.521691
4  0.523452

[5 rows x 21 columns]
```

df.csv 파일이 가지고 있는 컬럼은 21개가 존재하지만 주로 쓰는 열은 6개 정도이다.

'ImageID, LabelName, XMin, XMax, YMin, YMax'

◆ imageID : 이미지 파일 이름

◆ Source : xclick or activemil

◆ LabelName : Truck or Bus

◆ XMin : X 최소(Ground Truth의 Bounding Box 좌표 값)

◆ XMax : X 최대(Ground Truth의 Bounding Box 좌표 값)

◆ YMin : Y 최소(Ground Truth의 Bounding Box 좌표 값)

◆ YMax : Y 최대(Ground Truth의 Bounding Box 좌표 값)

◆ Confidence ,IsOccluded, IsTruncated, IsGroupOf, IsDepiction, IsInside, XClick1X, XClick2X, XClick3X, XClick4X, XClick1Y, XClick2Y, XClick3Y, XClick4Y 열이 존재

df.csv 파일이 가지고 있는 행은 24062개이다. 같은 ImageID를 가진 행들이 존재한다.

- image 시각화

```
class OpenImages(Dataset):
    def __init__(self, df, image_folder=image_root):
        self.root = image_folder
        self.df = df
        self.unique_images = df['ImageID'].unique()
    def __len__(self):
```

```
        return len(self.unique_images)

    # ix는 image_id의 index
    def __getitem__(self, ix):
        image_id = self.unique_images[ix]
        image_path = f'{self.root}/{image_id}.jpg'
        # Image : BGR -> RGB로 변경
        image = cv2.imread(image_path, 1)[...,::-1]
        h, w, _ = image.shape
        # 원본 훼손 방지를 위해 copy
        df = self.df.copy()
        df = df[df['ImageID'] == image_id]
        boxes = df['XMin,YMin,XMax,YMax'.split(',')].values
        # (24062,4) * (4,) 를 list로 변환
        boxes = (boxes * np.array([w,h,w,h])).astype(np.uint16).tolist()
        classes = df['LabelName'].values.tolist()
        return image, boxes, classes, image_path
ds = OpenImages(df=data)
im, bbs, clss, _ = ds[9]
print(_)
show(im, bbs=bbs, texts=clss, sz=10)
```

```
./data/images/images/00072b81abc72d21.jpg
```

OpenImages class를 통하여 얻을 수 있는 값은 4가지이다.

◆ image : image의 numpy값

◆ boxes : bounding box 좌표값

◆ classes : Truck or Bus
◆ image_path : 이미지 경로

얻은 값을 시각화하기 위해 torch_snippets의 show 함수를 이용하였다.
print(-)를 통해 얻은 값은 위 이미지의 경로이다.

- Region Proposal 구하기

• 필요한 함수 정의 - (1) extract_candidates

extract_candidates() 함수는 region proposal의 좌표값을 얻기 위해 selective search 알고리즘을 적용시켜 결과값을 추출하는 함수이다.

```
def extract_candidates(img):
    img_lbl, regions = selectivesearch.selective_search(img, scale=200, min_size=100)
    img_area = np.prod(img.shape[:2])
    candidates = []
    for r in regions:
        if r['rect'] in candidates: continue
        if r['size'] < (0.05*img_area): continue
        if r['size'] > (1*img_area): continue
        x, y, w, h = r['rect']
        candidates.append(list(r['rect']))
    return candidates
```

• 필요한 함수 정의 - (2) extract_iou

extract_iou() 함수는 boxA와 boxB의 iou를 구해주는 함수이다.

```
def extract_iou(boxA, boxB, epsilon=1e-5):
    x1 = max(boxA[0], boxB[0])
    y1 = max(boxA[1], boxB[1])
    x2 = min(boxA[2], boxB[2])
    y2 = min(boxA[3], boxB[3])
    width = (x2 - x1)
    height = (y2 - y1)
    if (width<0) or (height <0):
        return 0.0
    area_overlap = width * height
    area_a = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1])
    area_b = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1])
    area_combined = area_a + area_b - area_overlap
    iou = area_overlap / (area_combined+epsilon)
```

```
    return iou
```

- 학습에 필요한 FPATHS, GTBBS, CLSS, DELTAS, ROIS 구하기

이제는 앞의 함수들을 이용하여 region proposal의 좌표값을 구한 후, 실제값과 대조하여 각각의 필요 변수들을 채워줄 것이다.

◆ FPATHS : 저장된 파일의 path
◆ GTBBS : Ground Truth의 Bounding Box
◆ CLSS : Object의 class
◆ DELTAS : Ground Truth, Region Proposal의 offset(계산 값)
◆ ROIS : Region Proposal Locations
◆ IOUS : Ground Truth, Region Proposal의 IoU

```
FPATHS, GTBBS, CLSS, DELTAS, ROIS, IOUS = [], [], [], [], [], []
N = 500
for ix, (im, bbs, labels, fpath) in enumerate(ds):
    print(ix)
    if(ix==N):
        break
    H, W, _ = im.shape
    # region proposal의 x,y,w,h
    candidates = extract_candidates(im)
    # (x1,y1,x2,y2)
    candidates = np.array([(x,y,x+w,y+h) for x,y,w,h in candidates])
    ious, rois, clss, deltas = [], [], [], []
    ious = np.array([[extract_iou(candidate, _bb_) for candidate in candidates]
                     for _bb_ in bbs]).T
    # -> 각각의 caandidates(region proposal)와 bb_(정답)의 ious 저장
    for jx, candidate in enumerate(candidates):
        cx,cy,cX,cY = candidate
        candidate_ious = ious[jx]
        # 최대값 위치 반환
        best_iou_at = np.argmax(candidate_ious)
         # 가장 높은 iou값이 있는 곳의 iou값 가져오기
        best_iou = candidate_ious[best_iou_at]
        # 가장 높은 iou값이 있는 곳의 실제 정답값
        best_bb = _x,_y,_X,_Y = bbs[best_iou_at]
        if best_iou > 0.3:
            clss.append(labels[best_iou_at]) # Object 종류
        else :
            clss.append('background') # Background
        # 현재 proposal의 margins를 조정해서 best_bb와 일치하도록 함
```

```
            # proposal이 gt와 달라서 맞추는 delta 값으로 추정
            delta = np.array([_x-cx, _y-cy, _X-cX, _Y-cY]) / np.array([W,H,W,H])
            deltas.append(delta)
            rois.append(candidate / np.array([W,H,W,H]))
        FPATHS.append(fpath)
        IOUS.append(ious)
        ROIS.append(rois)
        CLSS.append(clss)
        DELTAS.append(deltas)
        GTBBS.append(bbs)
FPATHS = [f'{image_root}/{stem(f)}.jpg' for f in FPATHS]
FPATHS, GTBBS, CLSS, DELTAS, ROIS = [item for item in [FPATHS, GTBBS,
CLSS, DELTAS, ROIS]]
```

- **targets labeling**

Q. flatten 함수는 어디서 나왔을까요? 추측은 torch_snippet 패키지

```
targets = pd.DataFrame(flatten(CLSS), columns=['label'])
label2target = {l:t for t,l in enumerate(targets['label'].unique())}
target2label = {t:l for l,t in label2target.items()}
background_class = label2target['background']
print(label2target)
```

```
{'Bus': 0, 'background': 1, 'Truck': 2}
```

- **필요한 함수 정의**

밑에서 필요한 함수를 정의한다.

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
def preprocess_image(img):
    img = torch.tensor(img).permute(2,0,1)
    img = normalize(img)
    return img.to(device).float()
def decode(_y):
    _, preds = _y.max(-1)
    return preds
```

- **DATA 정의 및 Data Loader**

- **Custom Dataset**

이제는 FRCNN 모형에 잘 학습시킬 수 있도록 앞에서 추출한 데이터를 변형할 수 있는 class를 정의한다.

```
class RCNNDataset(Dataset):
    def __init__(self, fpaths, rois, labels, deltas, gtbbs):
        self.fpaths = fpaths
        self.gtbbs = gtbbs
        self.rois = rois
        self.labels = labels
        self.deltas = deltas
    def __len__(self): return len(self.fpaths)
    def __getitem__(self, ix):
        fpath = str(self.fpaths[ix])
        image = cv2.imread(fpath, 1)[...,::-1]
        H, W, _ = image.shape
        sh = np.array([W,H,W,H])
        gtbbs = self.gtbbs[ix]
        rois = self.rois[ix]
        bbs = (np.array(rois)*sh).astype(np.uint16) # 아까 나눴으니까
        labels = self.labels[ix]
        deltas = self.deltas[ix]
        crops = [image[y:Y,x:X] for (x,y,X,Y) in bbs]
        return image, crops, bbs, labels, deltas, gtbbs, fpath
    def collate_fn(self, batch):
        input, rois, rixs, labels, deltas = [], [], [], [], []
        for ix in range(len(batch)):
            image, crops, image_bbs, image_labels, image_deltas, image_gt_bbs,
image_fpath = batch[ix]
            crops = [cv2.resize(crop, (224,224)) for crop in crops]
            crops = [preprocess_image(crop/255.)[None] for crop in crops]
            input.extend(crops)
            labels.extend([label2target[c] for c in image_labels])
            deltas.extend(image_deltas)
        input = torch.cat(input).to(device)
        labels = torch.Tensor(labels).long().to(device)
        deltas = torch.Tensor(deltas).float().to(device)
        return input, labels, deltas
```

• **Data Loader**

전체 데이터의 90%를 train으로 10%를 test 데이터로 이용하기 위해 자른 후, DataLoader 안에 데이터를 넣어 batch_size만큼 정돈해준다.

```
n_train = 9*len(FPATHS)//10
train_ds = FRCNNDataset(FPATHS[:n_train], ROIS[:n_train], CLSS[:n_train],
```

```
DELTAS[:n_train], GTBBS[:n_train])
test_ds = FRCNNDataset(FPATHS[n_train:], ROIS[n_train:], CLSS[n_train:], DELTAS[n_train:], GTBBS[n_train:])

from torch.utils.data import TensorDataset, DataLoader
train_loader = DataLoader(train_ds, batch_size=2, collate_fn=train_ds.collate_fn, drop_last=True)
test_loader = DataLoader(test_ds, batch_size=2, collate_fn=test_ds.collate_fn, drop_last=True)
```

- 모형 정의

RCNN 모형을 정의한다. backbone 모형으로는 vgg16을 쓴다.

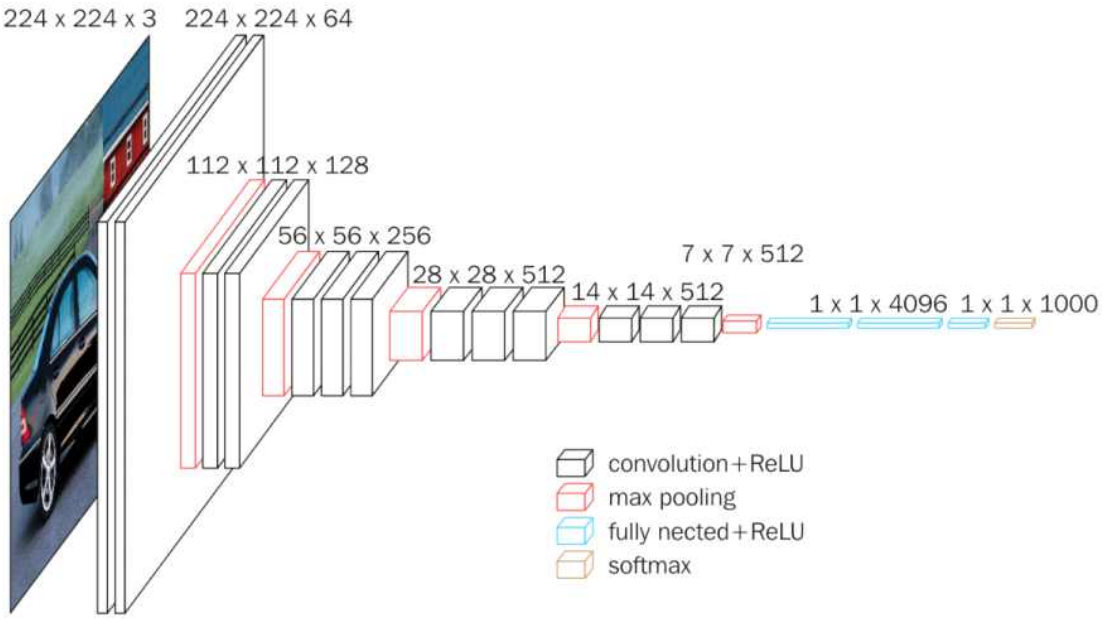


그림 5. https://bskyvision.com/504

```
vgg_backbone = models.vgg16(pretrained=True)
vgg_backbone.classifier = nn.Sequential()

# pretrained에서 썼던 weights를 그대로 가져오겠다.
for param in vgg_backbone.parameters():
    param.requires_grad = False

vgg_backbone.eval().to(device)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential()
)
```

vgg16 모형 뒤를 바꾸어서 원하는 결과를 얻게 해준다.
먼저, class score는 뒤에 층을 더 추가하여 target의 개수만큼 값이 나오게 한다. 이때의 loss function은 CrossEntropyLoss함수이다.
bounding box는 몇 개의 층을 더 추가하여 마지막에 4개의 값이 나오게 한다. 이때의 loss function은 L1Loss함수이다.
전체적인 loss 함수는 두 함수의 값을 더해주는데 이때, L1Loss 앞에 lamda가 곱해진다.

detection_loss + self.lmb * regression_loss

```
class RCNN(nn.Module):
    def __init__(self):
        super().__init__()
        feature_dim = 25088 # 7*7*512
        self.backbone = vgg_backbone
        self.cls_score = nn.Linear(feature_dim, len(label2target))
        self.bbox = nn.Sequential(
              nn.Linear(feature_dim, 512),
              nn.ReLU(),
              nn.Linear(512, 4),
              nn.Tanh(),
            )
        self.cel = nn.CrossEntropyLoss() # Classification
        self.sl1 = nn.L1Loss() # Regression
    def forward(self, input):
        feat = self.backbone(input)
        cls_score = self.cls_score(feat)
        bbox = self.bbox(feat)
        return cls_score, bbox
    def calc_loss(self, probs, _deltas, labels, deltas):
        detection_loss = self.cel(probs, labels)
        ixs, = torch.where(labels != 0) # index 따오기 위해
        _deltas = _deltas[ixs]
        deltas = deltas[ixs]
        self.lmb = 10.0
        if len(ixs) > 0:
            regression_loss = self.sl1(_deltas, deltas)
            return detection_loss + self.lmb * regression_loss, detection_loss.detach(), regression_loss.detach()
        else:
            regression_loss = 0
            return detection_loss + self.lmb * regression_loss, detection_loss.detach(), regression_loss
```

■ torch.where 부연 설명

```
import torch
x = torch.randn(3, 2)
y = torch.ones(2, 3)
#print(x)
#print(y)
print(y!=0)
print(torch.where(y!=0))
```

```
tensor([[True, True, True],
        [True, True, True]])
(tensor([0, 0, 0, 1, 1, 1]), tensor([0, 1, 2, 0, 1, 2]))
```

condition만 있으면 행과 열을 반환해준다. 따라서 우리가 필요한 index값을 얻기 위해 반환된 행을 저장하는 것을 위에서 볼 수 있따.

- 모형 확인

```
rcnn = RCNN().to(device)
print(rcnn)
```

```
RCNN(
  (backbone): VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU(inplace=True)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (6): ReLU(inplace=True)
      (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): ReLU(inplace=True)
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): ReLU(inplace=True)
      (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (13): ReLU(inplace=True)
      (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): ReLU(inplace=True)
      (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
        (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (18): ReLU(inplace=True)
        (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (20): ReLU(inplace=True)
        (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (22): ReLU(inplace=True)
        (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (25): ReLU(inplace=True)
        (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (27): ReLU(inplace=True)
        (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (29): ReLU(inplace=True)
        (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      )
      (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
      (classifier): Sequential()
    )
    (cls_score): Linear(in_features=25088, out_features=3, bias=True)
    (bbox): Sequential(
      (0): Linear(in_features=25088, out_features=512, bias=True)
      (1): ReLU()
      (2): Linear(in_features=512, out_features=4, bias=True)
      (3): Tanh()
    )
    (cel): CrossEntropyLoss()
    (sl1): L1Loss()
)
```

- 학습

• 배치사이즈마다 학습 - (1) train_batch()

전체 loss 값, loc_loss, regr_loss, 정확도를 return 해주는 train_batch 함수를 정의한다.

```
def train_batch(inputs, model, optimizer, criterion):
    input, clss, deltas = inputs
    model.train()
    optimizer.zero_grad()
    _clss, _deltas = model(input)
```

```
    loss, loc_loss, regr_loss = criterion(_clss, _deltas, clss, deltas)
    accs = clss == decode(_clss)
    loss.backward()
    optimizer.step()
    return loss.detach(), loc_loss, regr_loss, accs.cpu().numpy()
```

• **배치사이즈마다 학습 - (1) validate_batch()**

전체 class_score, delta값, loss 값, loc_loss, regr_loss, 정확도를 return 해주는 train_batch 함수를 정의한다.

```
@torch.no_grad()
def validate_batch(inputs, model, criterion):
    input, clss, deltas = inputs
    with torch.no_grad():
        model.eval()
        _clss,_deltas = model(input)
        loss, loc_loss, regr_loss = criterion(_clss, _deltas, clss, deltas)
        _, _clss = _clss.max(-1)
        accs = clss == _clss
    return _clss, _deltas, loss.detach(), loc_loss, regr_loss, accs.cpu().numpy()
```

• **필요 파라미터 정의**

```
rcnn = RCNN().to(device)
criterion = rcnn.calc_loss
optimizer = optim.SGD(rcnn.parameters(), lr=1e-3)
n_epochs = 5
log = Report(n_epochs)
```

• **epoch만큼 학습**

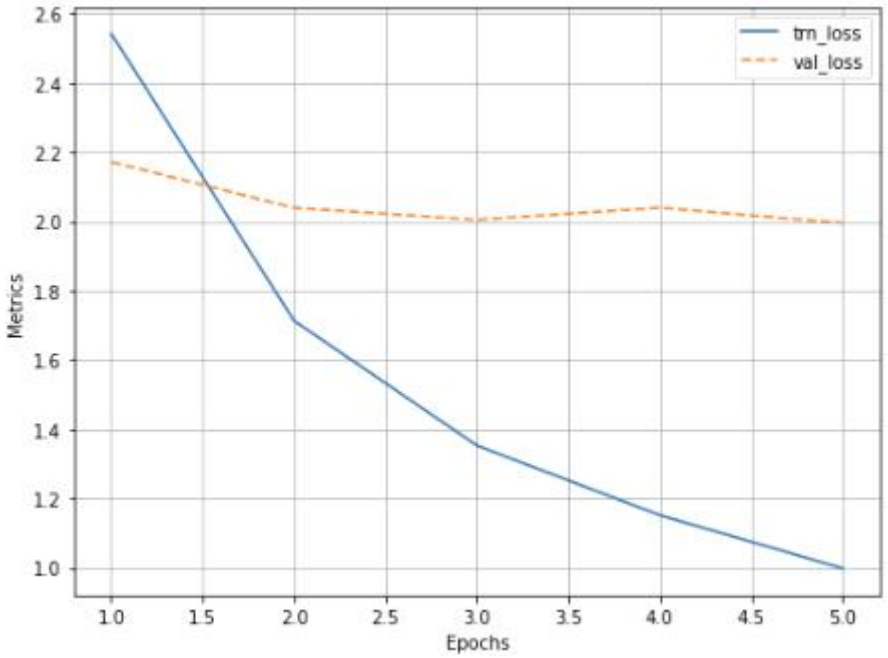
```
for epoch in range(n_epochs):

    _n = len(train_loader)
    for ix, inputs in enumerate(train_loader):
        loss, loc_loss, regr_loss, accs = train_batch(inputs, rcnn,
                                                      optimizer, criterion)
        pos = (epoch + (ix+1)/_n)
        log.record(pos, trn_loss=loss.item(), trn_loc_loss=loc_loss,
                   trn_regr_loss=regr_loss,
                   trn_acc=accs.mean(), end='\r')

    _n = len(test_loader)
```

```
    for ix,inputs in enumerate(test_loader):
        _clss, _deltas, loss, \
        loc_loss, regr_loss, accs = validate_batch(inputs,
                                                   rcnn, criterion)
        pos = (epoch + (ix+1)/_n)
        log.record(pos, val_loss=loss.item(), val_loc_loss=loc_loss,
                   val_regr_loss=regr_loss,
                   val_acc=accs.mean(), end='\r')

# Plotting training and validation metrics
log.plot_epochs('trn_loss,val_loss'.split(','))
```




```
EPOCH: 4.880    val_loss: 1.444 val_loc_loss: 0.488    val_regr_loss: 0.096    val_acc: 0.852  (21
8.57s - 5.37s remaining)(144, 256, 3)
(185, 256, 3)

EPOCH: 4.920    val_loss: 1.340 val_loc_loss: 0.457    val_regr_loss: 0.088    val_acc: 0.766  (21
8.70s - 3.56s remaining)(167, 256, 3)
(192, 256, 3)
EPOCH: 4.960    val_loss: 2.185 val_loc_loss: 0.237    val_regr_loss: 0.195    val_acc: 0.905  (21
8.86s - 1.77s remaining)(192, 256, 3)
(171, 256, 3)
EPOCH: 5.000    val_loss: 1.659 val_loc_loss: 0.552    val_regr_loss: 0.111    val_acc: 0.753  (21
9.03s - 0.00s remaining)
```

5 epoch를 기준으로 돌려보았다. 그래프에서 보는바와 같이 epoch가 증가할수록 loss 값이 줄어드는 것을 확인할 수 있다.

- **Test**

앞선 train 코드와 비슷하지만 test이기 때문에 frcnn 모형에 .eval()을 붙여준다. **.eval()**은 test 과정에서 사용하지 않아야 하는 layer를 알아서 off 시키는 평가 역할을 한다.

보통 **with no_grad()**와 함께 쓰이는데 이는 해당 시점의 gradient 값을 고정시킨다는 의미이다(노 업데이트).

```
def test_predictions(filename, show_output=True):
    img = np.array(cv2.imread(filename, 1)[...,::-1])
    candidates = extract_candidates(img)
    candidates = [(x,y,x+w,y+h) for x,y,w,h in candidates]
    input = []
    for candidate in candidates:
        x,y,X,Y = candidate
        crop = cv2.resize(img[y:Y,x:X], (224,224))
        input.append(preprocess_image(crop/255.)[None])
    input = torch.cat(input).to(device)
    with torch.no_grad():
        rcnn.eval()
        probs, deltas = rcnn(input)
        probs = torch.nn.functional.softmax(probs, -1)
        confs, clss = torch.max(probs, -1)
    candidates = np.array(candidates)
    confs, clss, probs, deltas = [tensor.detach().cpu().numpy() for tensor in [confs, clss, probs, deltas]]

    ixs = clss!=background_class
    confs, clss, probs, deltas, candidates = [tensor[ixs] for tensor in [confs, clss, probs, deltas, candidates]]
    bbs = (candidates + deltas).astype(np.uint16)
    ixs = nms(torch.tensor(bbs.astype(np.float32)), torch.tensor(confs), 0.05)
    confs, clss, probs, deltas, candidates, bbs = [tensor[ixs] for tensor in [confs, clss, probs, deltas, candidates, bbs]]
    if len(ixs) == 1:
        confs, clss, probs, deltas, candidates, bbs = [tensor[None] for tensor in [confs, clss, probs, deltas, candidates, bbs]]
    if len(confs) == 0 and not show_output:
        return (0,0,224,224), 'background', 0
    if len(confs) > 0:
        best_pred = np.argmax(confs)
        best_conf = np.max(confs)
        best_bb = bbs[best_pred]
        x,y,X,Y = best_bb
    _, ax = plt.subplots(1, 2, figsize=(20,10))
    show(img, ax=ax[0])
```

```
    ax[0].grid(False)
    ax[0].set_title('Original image')
    if len(confs) == 0:
        ax[1].imshow(img)
        ax[1].set_title('No objects')
        plt.show()
        return
    ax[1].set_title(target2label[clss[best_pred]])
    show(img, bbs=bbs.tolist(), texts=[target2label[c] for c in clss.tolist()],
ax=ax[1], title='predicted bounding box and class')
    plt.show()
    return (x,y,X,Y),target2label[clss[best_pred]],best_conf
```

- **결과 확인**

```
image, crops, bbs, labels, deltas, gtbbs, fpath = test_ds[7]
test_predictions(fpath)
```

```
(176, 256, 3)
```


```
((16, 60, 218, 133), 'Bus', 0.9818872)
```

결과를 보면 저 bounding box 안에 class score 0.98로 bus라고 잘 예측한 것을 확인할 수 있다. 다만 아쉬운 점은 버스가 전부 다 bounding box 안에 들어와있진 않다. 조금 더 학습시킨다면 충분히 GT에 맞는 bounding box가 그려져 있을 것이다.