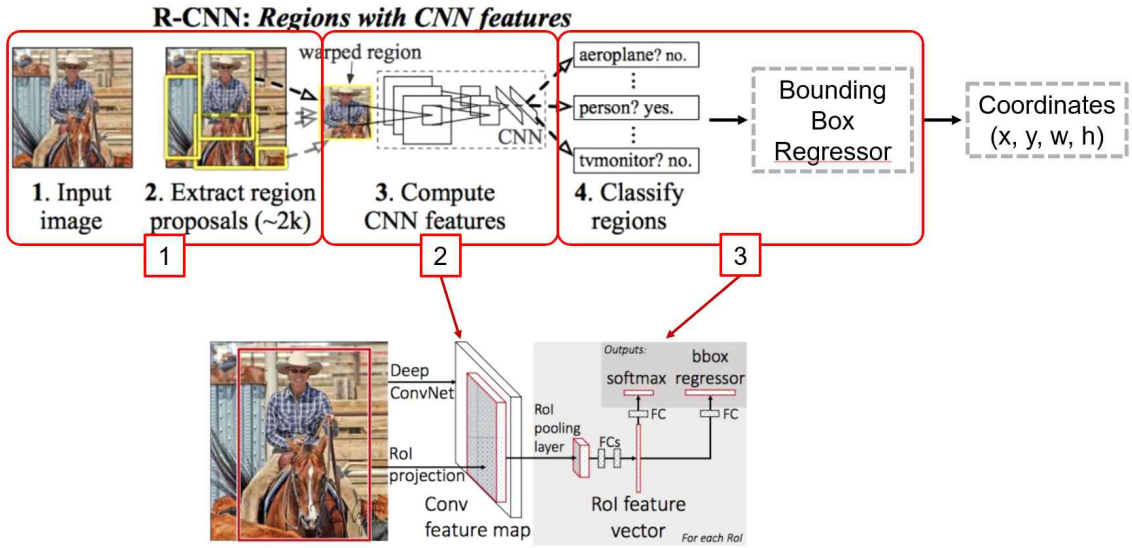
Chapter7. Basics of Object Detection

2021210088 허지혜

Training Fast R-CNN-based custom object detectors

R-CNN의 단점 중 하나는 시간이 오래 걸린다는 것이다. 특히, Proposals에 대해 각각 crop, CNN(Convolutional Neural Network) 과정을 거쳐야 하는 병목(bottleneck) 구조때문에 더 두각되며 또한 이미지 변형으로 인한 성능 손실이 존재한다. 이 단점을 보완하여 나온 모형 중 SPPNet이 있다. SPPNet은 기존 R-CNN이 Selective Search를 이용하여 찾아낸 Region proposal에 대하여 모두 CNN을 하는 방식에서 CNN을 전체 이미지에 한 번만 수행하고 feaure map을 공유하는 방식으로 해결하였다. 하지만 여러 단계를 거치고, FC Layer만 학습 시키지 못하는 한계에 부딪혔다. 따라서 Fast R-CNN의 저자는 CNN feature map으로부터 Classification, Box Regression까지 하나의 모형으로 학습시키자는 아이디어로 Fast R-CNN을 발표하였다. 실제로 Fast R-CNN은 시간도 단축시키며 detection의 정확도도 증가하는 모습을 보였다.



출처. https://seongkyun.github.io/papers/2019/01/06/Object_detection/

위 사진은 R-CNN과 Fast R-CNN을 비교한 사진이다. Network Architecture를 살펴보면 어떻게 시간을 단축 시켰는지 보인다, Fast R-CNN의 과정을 나타내면 다음과 같이 나타낼 수 있다.

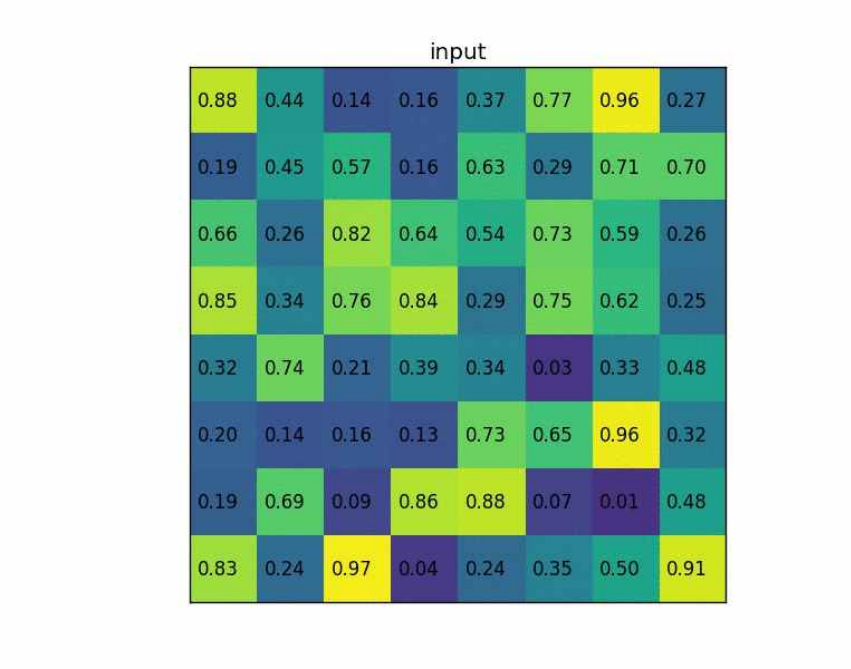
1. Selective Serach를 이용해 ROI(Region of Interest) or Region Prposal을 찾는다.
2. 전체 Image를 CNN에 통과시켜 feature map을 추출한다.
3. Selective Search로 찾았던 RoI를 feature map 크기에 맞춰 projection(투영) 시킨다.
4. projection된 ROI에 대하여 ROI pooling을 진행하여 고정된 크기의 feature vector를 얻는다. 이는 후에 FC layer에 넣기 위해 같은 resolution의 feature map이 필요하기 때문이다.

5. feature vector는 FC layer를 통과한 후,
    - softmax를 통과하여 ROI에 대해 Object Classification을 진행
    - bounding box regression을 통해 selective search로 찾은 box의 위치를 조정

- 핵심 아이디어

㉠ RoI Pooling

앞서 얘기했듯, 먼저 전체 Image를 CNN에 통과시켜 feature map을 추출한다. 추출된 feature map을 미리 정해둔 크기에 맞게 grid를 설정한다. 그리고 각각의 칸 별로 가장 큰 값을 추출하는 max pooling을 실시하면 결과값은 항상 같은 크기의 feature map이 되고 이를 펼쳐 feature vector를 추출하게 된다.



https://nuggy875.tistory.com/33

코드 구현은 다음의 torchvision.ops.RoIPool 함수를 이용하여 사용하면 된다.

ROI_POOL

```
torchvision.ops.roi_pool( 입력 : torch.Tensor, 상자 : Union [ torch.Tensor , List
[ torch.Tensor ] ], output_size : None, spatial_scale : float = 1.0 ) → torch.Tensor [원천]
```

Fast R-CNN에 설명된 관심 영역(RoI) 풀 연산자 수행

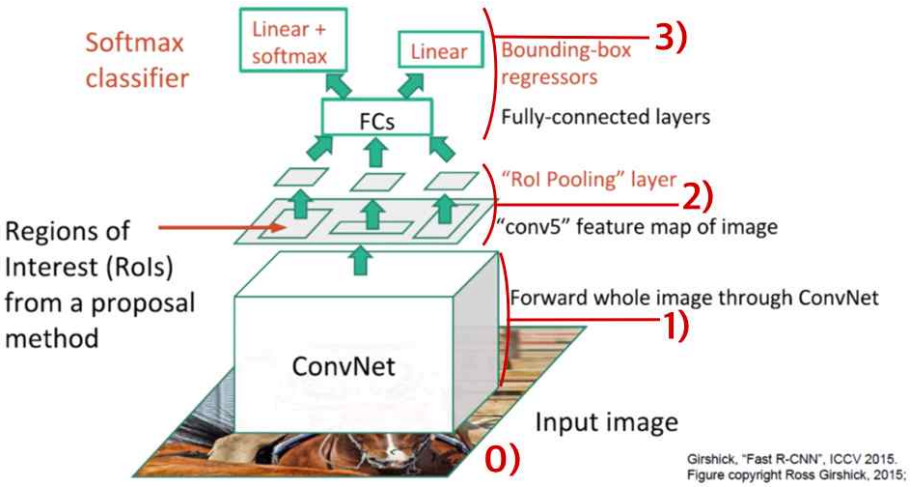
그림 3.https://pytorch.org/vision/stable/generated/torchvision.ops.roi_pool.html#torchvision.ops.roi_pool

- 책에서 구현한 FRCNN 코드

```
from torchvision.ops import RoIPool
class FRCNN(nn.Module):
    def __init__(self):
        super().__init__()
        rawnet = torchvision.models.vgg16_bn(pretrained=True)
        for param in rawnet.features.parameters():
            param.requires_grad = True
        self.seq = nn.Sequential(*list(rawnet.features.children())[:-1])
        self.roipool = RoIPool(7, spatial_scale=14/224)
        feature_dim = 512*7*7
        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()
        self.sl1 = nn.L1Loss()
    def forward(self, input, rois, ridx):
        res = input
        res = self.seq(res)
        rois = torch.cat([ridx.unsqueeze(-1), rois*224], dim=-1)
        res = self.roipool(res, rois)
        feat = res.view(len(res), -1)
        cls_score = self.cls_score(feat)
        bbox = self.bbox(feat) # .view(-1, len(label2target), 4)
        return cls_score, bbox
    def calc_loss(self, probs, _deltas, labels, deltas):
        detection_loss = self.cel(probs, labels)
        ixs, = torch.where(labels != background_class)
        _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
```

Fast R-CNN Class를 정의하였다.
이를 자세히 뜯어서 살펴보자.
첫 번째 부분은 다음과 같다.



https://nuggy875.tistory.com/33

```
class FRCNN(nn.Module):
    def __init__(self):
        super().__init__()
        rawnet = torchvision.models.vgg16_bn(pretrained=True)
        for param in rawnet.features.parameters():
            # 신경망의 매개변수에서 계산
            # tensor에 대한 모든 연산이 True
            param.requires_grad = True
        # rawnet의 features의 하위 항목까지 뽑아온다. 마지막 layer만 빼고!
        self.seq = nn.Sequential(*list(rawnet.features.children())[:-1])
        # RoIPool 적용, spatial_scale=상자 좌표를 입력 좌표에 매핑하는 배율 인수
        # output=Tensor([K,C,ouput_size[0], output_size[1]])
        self.roipool = RoIPool(7, spatial_scale=14/224)
        feature_dim = 512*7*7
        # Truck, Bus, Background로 분류해야 하니까 len(label2target)만큼 출력
        self.cls_score = nn.Linear(feature_dim, len(label2target))
        # bounding box의 값을 출력으로 내야 하니까 4 출력
        self.bbox = nn.Sequential(
            nn.Linear(feature_dim, 512),
```

```
            nn.ReLU(),
            nn.Linear(512, 4),
            nn.Tanh(),
        )
        # 3가지 분류니까 CrossEntropyLoss(), 이진 분류면, BCELoss()
        self.cel = nn.CrossEntropyLoss()
        # GT와 예측 bounding box의 절대값 차이(Bounding box)
        self.sl1 = nn.L1Loss()
```

두 번째 부분은 다음과 같다.

```
def forward(self, input, rois, ridx):
        res = input
        res = self.seq(res)
        # unsqeeze() : 특정 위치에 1인 차원 추가
        #
        rois = torch.cat([ridx.unsqueeze(-1), rois*224], dim=-1)
        res = self.roipool(res, rois)
        # roipool을 거친 후 쫙 펴야해서 view() 함수 이용
        feat = res.view(len(res), -1)
        cls_score = self.cls_score(feat)
        bbox = self.bbox(feat) # .view(-1, len(label2target), 4)
        return cls_score, bbox
```

세 번째 부분은 다음과 같다.

```
def calc_loss(self, probs, _deltas, labels, deltas):
        detection_loss = self.cel(probs, labels)
        ixs, = torch.where(labels != background_class)
        _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
```

Fast R-CNN에서는 Loss Function으로 Multi-task Loss를 쓴다. Multi-task Loss는 Classification Loss과 Regressin Loss를 합쳐서 사용한다.

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v),$$

위 식이 나타내는 각각의 변수는 다음과 같다.

- u : ground truth class
- Classification loss : CrossEntropyLoss()
- Regression loss : L1Loss()
- lambda[u≥1] : 논문에서는 lambda를 1로 잡고, u가 1보다 크면 1, 아니면 0. 1에 해당하는 클래스에 대해서만 regression loss 적용한다. 책 코드에서는 10으로 잡았다.

- Fast R-CNN의 Contribution

㉠ mAP R-CNN, SPPnet보다 높다.

㉡ multi-task loss를 이용하여 Single-stage로 training

- 하지만 Single Detector는 아님. Region Proposal을 구하기 때문.

㉢ 모든 network layers를 update하며 training하였다.

-> 하지만 구조를 바꿨음에도 시간이 오래 걸린다. Faster R-CNN이 나오게 된 계기.

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

데이터는 kaggle에서 다운받았다. R-CNN과 마찬가지로 image가 들어있는 **images 폴더**와 bounding box 좌표, image_id, image_path 등이 들어있는 **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  00010bf498b64bab  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'

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)
    def __getitem__(self, ix):
        image_id = self.unique_images[ix]
        image_path = f'{self.root}/{image_id}.jpg'
        image = cv2.imread(image_path, 1)[...,::-1] # conver BGR to RGB
        h, w, _ = image.shape
        df = self.df.copy()
        df = df[df['ImageID'] == image_id]
        boxes = df['XMin,YMin,XMax,YMax'.split(',')].values
        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=DF_RAW)
im, bbs, clss, _ = ds[11]
print(bbs)
show(im, bbs=bbs, texts=clss, sz=10)
```

```
[[40, 36, 184, 105], [158, 58, 255, 93]]
```

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.
위 [[40, 36, 184, 105], [158, 58, 255, 93]]는 bounding box 좌표값이다.

- 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, GTBBS, CLSS, DELTAS, ROIS, IOUS = [], [], [], [], [], []
N = 500
for ix, (im, bbs, labels, fpath) in enumerate(ds):
    if(ix==N):
        break
    H, W, _ = im.shape
    candidates = extract_candidates(im)
    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
    for jx, candidate in enumerate(candidates):
        cx,cy,cX,cY = candidate
        candidate_ious = ious[jx]
        best_iou_at = np.argmax(candidate_ious)
        best_iou = candidate_ious[best_iou_at]
        best_bb = _x,_y,_X,_Y = bbs[best_iou_at]
        if best_iou > 0.3: clss.append(labels[best_iou_at])
        else : clss.append('background')
        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 FRCNNDataset(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]
        gtbbs = self.gtbbs[ix]
        rois = self.rois[ix]
        labels = self.labels[ix]
```

```
        deltas = self.deltas[ix]
        assert len(rois) == len(labels) == len(deltas), f'{len(rois)}, {len(labels)}, {len(deltas)}'
        return image, rois, labels, deltas, gtbbs, fpath

    def collate_fn(self, batch):
        input, rois, rixs, labels, deltas = [], [], [], [], []
        for ix in range(len(batch)):
            image, image_rois, image_labels, image_deltas, image_gt_bbs, image_fpath = batch[ix]
            image = cv2.resize(image, (224,224))
            input.append(preprocess_image(image/255.)[None])
            rois.extend(image_rois)
            rixs.extend([ix]*len(image_rois))
            labels.extend([label2target[c] for c in image_labels])
            deltas.extend(image_deltas)
        input = torch.cat(input).to(device)
        rois = torch.Tensor(rois).float().to(device)
        rixs = torch.Tensor(rixs).float().to(device)
        labels = torch.Tensor(labels).long().to(device)
        deltas = torch.Tensor(deltas).float().to(device)
        return input, rois, rixs, 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)
```

• **확인**

```
for input, rois, rixs, labels, deltas in train_loader:
    print(input.shape)
    print(labels) # labels 값이 각 batch_size마다 다름
    break
```

```
torch.Size([2, 3, 224, 224])
tensor([0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2,
        1, 1, 1, 1, 1, 1, 1, 1, 1])
```

- 모형 정의

앞서 공부한 대로 FRCNN 모형을 정의한다.

```
from torchvision.ops import RoIPool
class FRCNN(nn.Module):
    def __init__(self):
        super().__init__()
        rawnet = torchvision.models.vgg16_bn(pretrained=True)
        for param in rawnet.features.parameters():
            param.requires_grad = True
        self.seq = nn.Sequential(*list(rawnet.features.children())[:-1])
        self.roipool = RoIPool(7, spatial_scale=14/224)
        feature_dim = 512*7*7
        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()
        self.sl1 = nn.L1Loss()
    def forward(self, input, rois, ridx):
        res = input
        res = self.seq(res)
        rois = torch.cat([ridx.unsqueeze(-1), rois*224], dim=-1)
        res = self.roipool(res, rois)
        feat = res.view(len(res), -1)
        cls_score = self.cls_score(feat)
        bbox = self.bbox(feat) # .view(-1, len(label2target), 4)
```

```
        return cls_score, bbox
    def calc_loss(self, probs, _deltas, labels, deltas):
        detection_loss = self.cel(probs, labels)
        ixs, = torch.where(labels != background_class)
        _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
```

- **모형 확인**

```
frcnn = FRCNN().to(device)
print(frcnn)
```

```
FRCNN(
  (seq): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (32): ReLU(inplace=True)
    (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (36): ReLU(inplace=True)
    (37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (39): ReLU(inplace=True)
    (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (42): ReLU(inplace=True)
```

```
  )
  (roipool): RoIPool(output_size=7, spatial_scale=0.0625)
  (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, rois, rixs, clss, deltas = inputs
    model.train()
    optimizer.zero_grad()
    _clss, _deltas = model(input, rois, rixs)
    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.

```
def validate_batch(inputs, model, criterion):
    input, rois, rixs, clss, deltas = inputs
    with torch.no_grad():
        model.eval()
        _clss,_deltas = model(input, rois, rixs)
        loss, loc_loss, regr_loss = criterion(_clss, _deltas, clss, deltas)
        _clss = decode(_clss)
        accs = clss == _clss
    return _clss, _deltas, loss.detach(), loc_loss, regr_loss, accs.cpu().numpy()
```

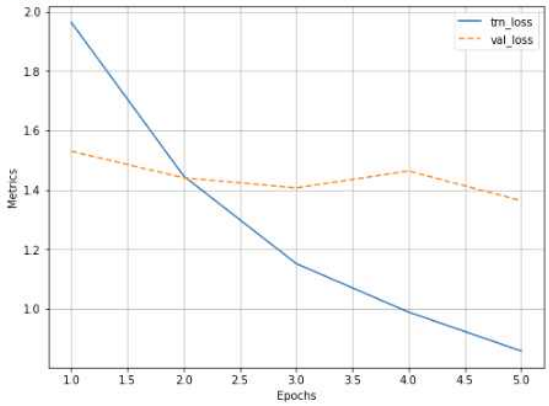
- **필요 파라미터 정의**

```
frcnn = FRCNN().to(device)
criterion = frcnn.calc_loss
optimizer = optim.SGD(frcnn.parameters(), lr=1e-3)

n_epochs = 5
```

- **epoch만큼 학습**

```
log = Report(n_epochs)
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, frcnn,
                                                      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,
                                                   frcnn, 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')

    log.report_avgs(epoch+1)

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

```
/usr/local/lib/python3.8/dist-packages/torch/nn/functional.py:718: UserWarning: Named tensors and all their associated APIs
are an experimental feature and subject to change. Please do not use them for anything important until they are released as
stable. (Triggered internally at  /pytorch/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)

EPOCH: 1.000    trn_loss: 1.963 trn_loc_loss: 0.860     trn_regr_loss: 0.110    trn_acc: 0.651  val_loss: 1.530 val_loc_los
s: 0.656        val_regr_loss: 0.087    val_acc: 0.741  (56.28s - 225.12s remaining)
EPOCH: 2.000    trn_loss: 1.446 trn_loc_loss: 0.634     trn_regr_loss: 0.081    trn_acc: 0.757  val_loss: 1.440 val_loc_los
s: 0.607        val_regr_loss: 0.083    val_acc: 0.767  (112.74s - 169.11s remaining)
EPOCH: 3.000    trn_loss: 1.152 trn_loc_loss: 0.517     trn_regr_loss: 0.064    trn_acc: 0.810  val_loss: 1.406 val_loc_los
s: 0.588        val_regr_loss: 0.082    val_acc: 0.763  (169.14s - 112.76s remaining)
EPOCH: 4.000    trn_loss: 0.988 trn_loc_loss: 0.432     trn_regr_loss: 0.056    trn_acc: 0.850  val_loss: 1.463 val_loc_los
s: 0.580        val_regr_loss: 0.088    val_acc: 0.760  (226.84s - 56.71s remaining)
EPOCH: 5.000    trn_loss: 0.858 trn_loc_loss: 0.367     trn_regr_loss: 0.049    trn_acc: 0.878  val_loss: 1.363 val_loc_los
s: 0.580        val_regr_loss: 0.078    val_acc: 0.763  (284.34s - 0.00s remaining)

100%|██████████| 106/106 [00:00<00:00, 5603.54it/s]
```



일단, cpu를 이용하여 결과를 냈기 때문에 시간이 좀 걸려 5 epoch를 기준으로 돌려보았다. 그래프에서 보는바와 같이 epoch가 증가할수록 loss 값이 줄어드는 것을 확인할 수 있다.

- **Test**

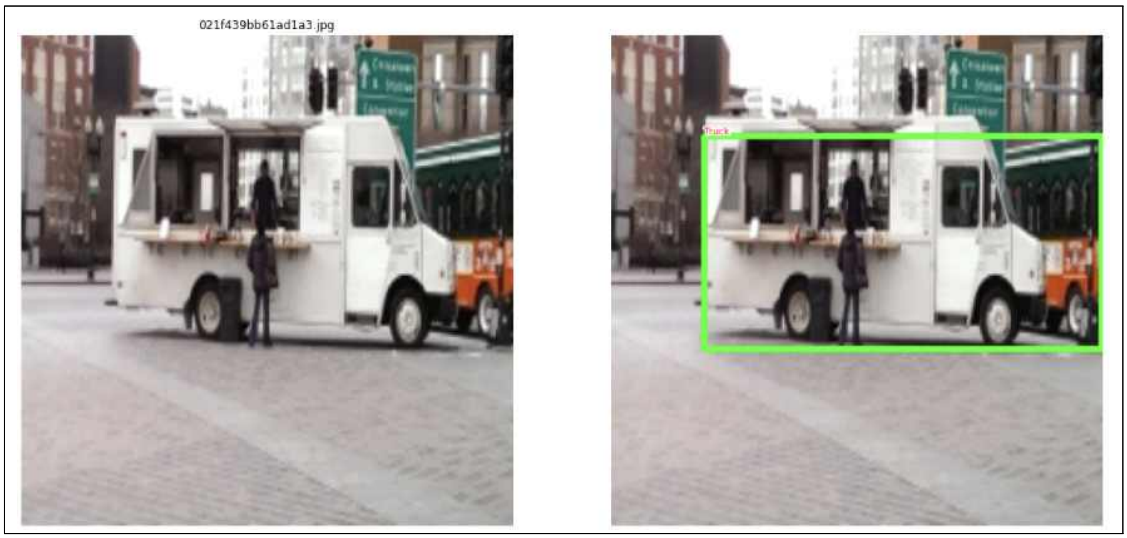
앞선 train 코드와 비슷하지만 test이기 때문에 frcnn 모형에 .eval()을 붙여준다. **.eval()**은 test 과정에서 사용하지 않아야 하는 layer를 알아서 off 시키는 평가 역할을 한다.
보통 **with no_grad()**와 함께 쓰이는데 이는 해당 시점의 gradient 값을 고정시킨다는 의미이다(노 업데이트).

```
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.patches as mpatches
from torchvision.ops import nms
from PIL import Image
def test_predictions(filename):
    img = cv2.resize(np.array(Image.open(filename)), (224,224))
    candidates = extract_candidates(img)
    candidates = [(x,y,x+w,y+h) for x,y,w,h in candidates]
    input = preprocess_image(img/255.)[None]
    rois = [[x/224,y/224,X/224,Y/224] for x,y,X,Y in candidates]
    rixs = np.array([0]*len(rois))
    rois, rixs = [torch.Tensor(item).to(device) for item in [rois, rixs]]
```

```
    with torch.no_grad():
        frcnn.eval()
        probs, deltas = frcnn(input, rois, rixs)
        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
    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]]

    bbs = bbs.astype(np.uint16)
    _, ax = plt.subplots(1, 2, figsize=(20,10))
    show(img, ax=ax[0])
    ax[0].grid(False)
    ax[0].set_title(filename.split('/')[-1])
    if len(confs) == 0:
        ax[1].imshow(img)
        ax[1].set_title('No objects')
        plt.show()
        return
    else:
        show(img, bbs=bbs.tolist(), texts=[target2label[c] for c in clss.tolist()], ax=ax[1])
        plt.show()
```

- **결과 확인**

```
test_predictions(test_ds[29][-1])
```



결과를 확인해봤더니 5번밖에 학습을 못해서 엄청 정확하게 bounding box를 예측한 건 아니지만 나름 차 근처에서 그려진 것을 확인하였다. 조금 더 학습을 시킨다면 정확하게 예측을 할 것이다.

- 출처

https://yeomko.tistory.com/15
https://wiserloner.tistory.com/1175?category=825446
https://ganghee-lee.tistory.com/36
https://seongkyun.github.io/papers/2019/01/06/Object_detection/