# Object Detection

# Pytorch Practice – Object detection

- https://tutorials.pytorch.kr/intermediate/torchvision_tutorial.html

# Object Detection Practice

- **Pet detection**
- **https://www.robots.ox.ac.uk/~vgg/data/pets/**
- **주어진 데이터를 이용해 실제 객체인식기 만들어보기**

# Object Detection Practice

```python
class ObjectDetector(Module):
    def __init__(self, baseModel, numClasses):
        super(ObjectDetector, self).__init__()
        # initialize the base model and the number of classes
        self.baseModel = baseModel
        self.numClasses = numClasses
        # build the regressor head for outputting the bounding box
        # coordinates
        self.regressor = Sequential(
            Linear(baseModel.fc.in_features, 128),
            ReLU(),
            Linear(128, 64),
            ReLU(),
            Linear(64, 32),
            ReLU(),
            Linear(32, 4),
            Sigmoid()
        )
```

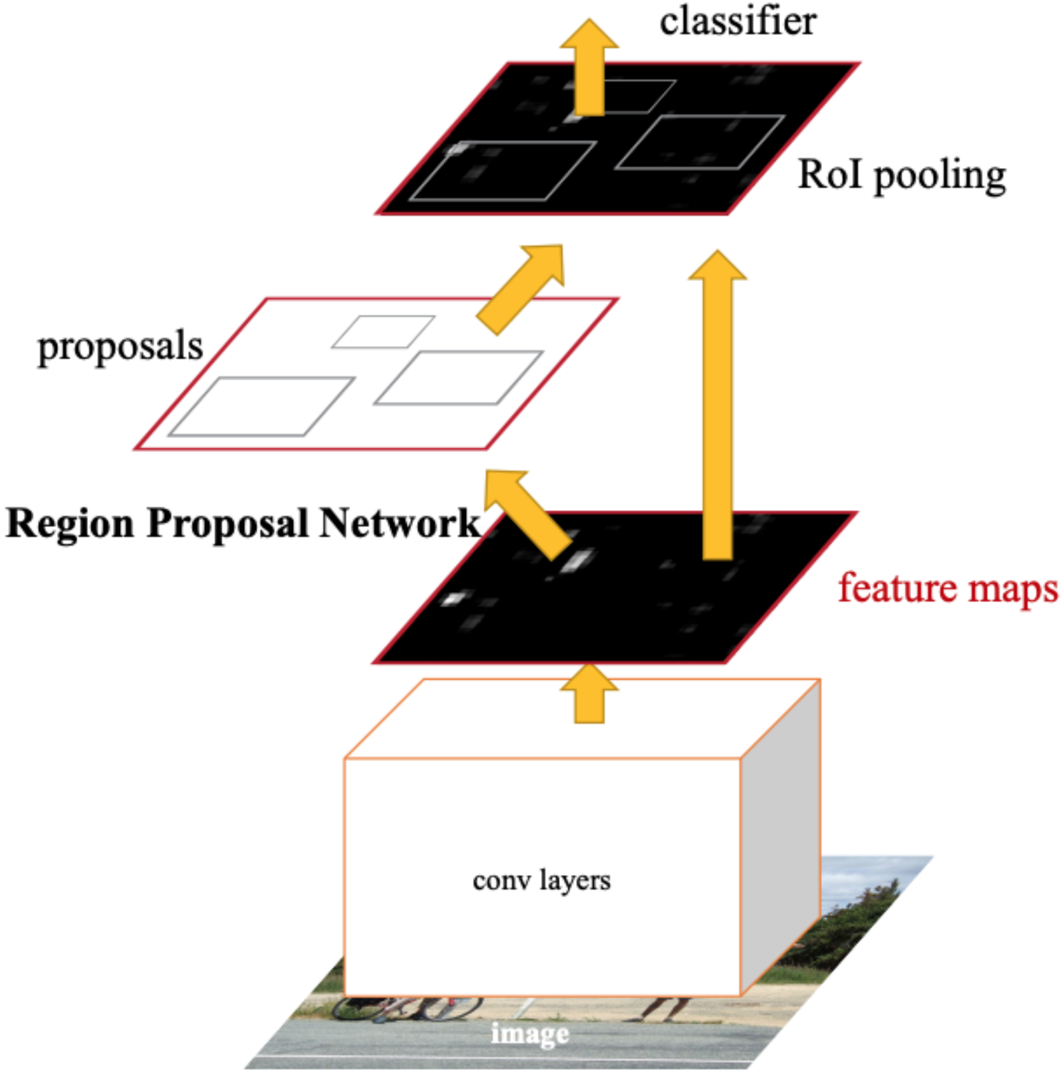
# Object Detection Practice

```python
# build the classifier head to predict the class labels
self.classifier = Sequential(
    Linear(baseModel.fc.in_features, 512),
    ReLU(),
    Dropout(),
    Linear(512, 512),
    ReLU(),
    Dropout(),
    Linear(512, self.numClasses)
)
# set the classifier of our base model to produce outputs
# from the last convolution block
self.baseModel.fc = Identity()
```

# Object Detection Practice

```python
    def forward(self, x):
        # pass the inputs through the base model and then obtain
        # predictions from two different branches of the network
        features = self.baseModel(x)
        bboxes = self.regressor(features)
        classLogits = self.classifier(features)
        # return the outputs as a tuple
        return (bboxes, classLogits)
```

# Faster-RCNN


classifier
RoI pooling
proposals
Region Proposal Network
feature maps
conv layers
image

# Dependencies

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

```
[510] import torch
      import torchvision
      import torch.nn as nn
      import torch.nn.functional as F
      from torchvision import transforms

      import matplotlib.pyplot as plt
      import numpy as np
      import cv2
      from PIL import Image
```

```
[511] if torch.cuda.is_available():
          DEVICE = torch.device("cuda")
          print(DEVICE, torch.cuda.get_device_name(0))
      else:
          DEVICE = torch.device("cpu")
          print(DEVICE)
      DEVICE = torch.device("cpu")

      cuda Tesla T4
```

Google Colab 내 디렉터리에 있는 mount

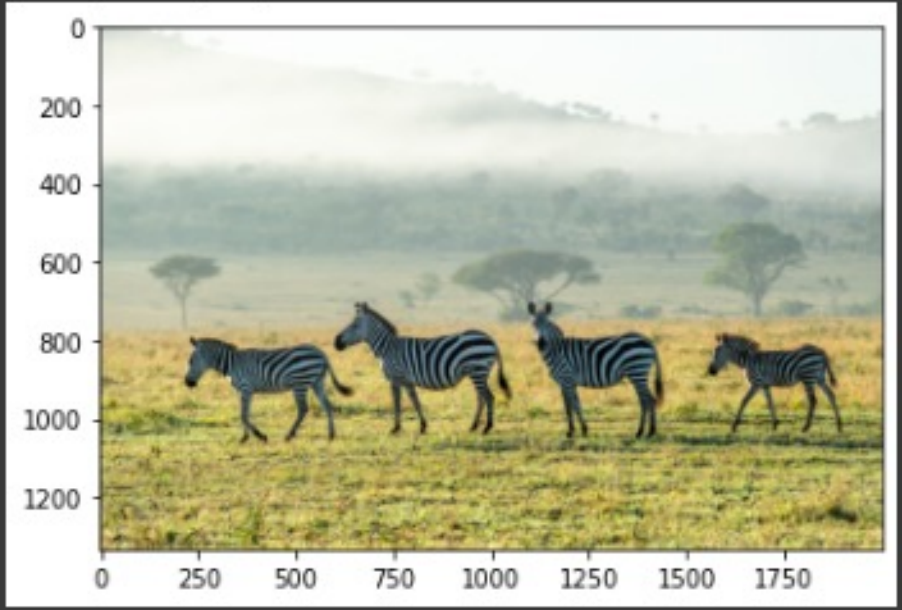
시각화 및 전처리를 위해 matplotlib와 cv2, PIL을 import

Colab 실행환경에서,
DEVICE는 cuda Tesla T4

# Data Processing Visualize image and bounding boxes

```python
# In this example, only use 1 image, i.e, batch_size=1
# input image could be of any size

img0 = cv2.imread("/content/drive/MyDrive/test/zebras.jpg")
img0 = cv2.cvtColor(img0, cv2.COLOR_BGR2RGB)
print(img0.shape)
plt.imshow(img0)
plt.show()
```

```
(1333, 2000, 3)
```




입력 이미지는 테스트를 위해
1개만 사용.

cv2.imread를 사용해
Drive 이미지 불러오기

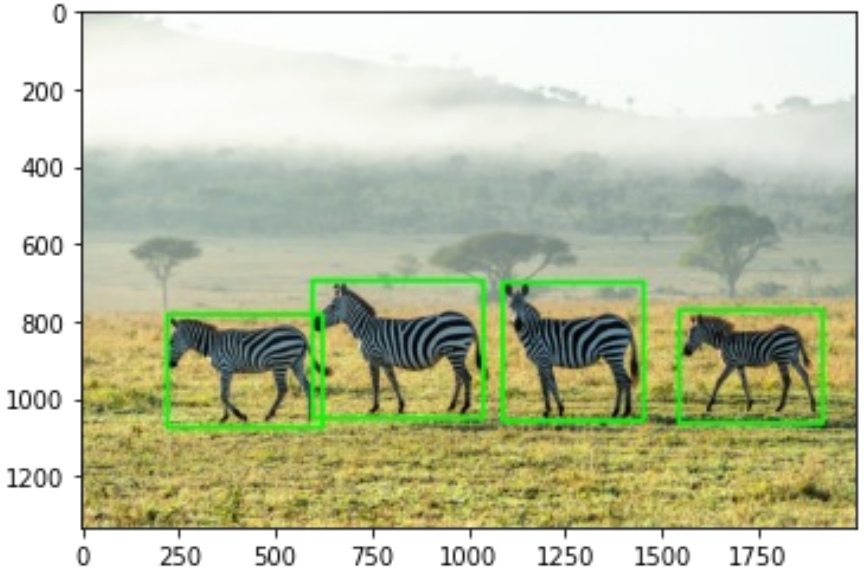
cv2.imread로 불러온 이미지는
BGR 형식이기 때문에

cv2.COLOR_BGR2RGB를 통해
RGB 형식으로 전환

원본 이미지의 크기는
1333(h), 2000(w), 3(c)

# Data Processing

```
[ ]  # object information : a set of bounding boxes [x1, y1, x2, y2]
     # and their labels
     bbox0 = np.array([[223, 782, 623, 1074], [597, 695, 1038, 1050],
                       [1088, 699, 1452, 1057], [1544, 771, 1914, 1063]])
     labels = np.array([1, 1, 1, 1]) # 0: background, 1: zebra

[ ]  # display bounding box and labels

     img0_clone = np.copy(img0)
     for i in range(len(bbox0)):
         cv2.rectangle(img0_clone, (bbox0[i][0], bbox0[i][1]),
                                   (bbox0[i][2], bbox0[i][3]),
                       color=(0, 255, 0), thickness=10)
     plt.imshow(img0_clone)
     plt.show()
```

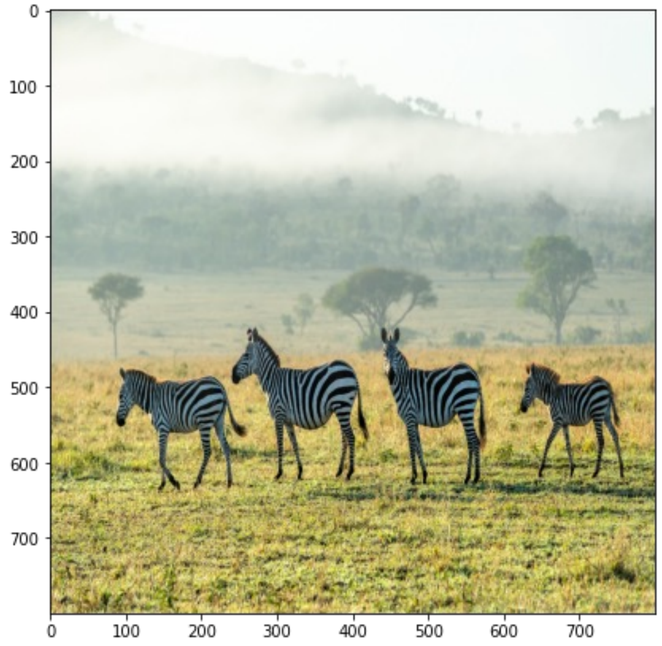



np.array를 이용해 좌표입력
cv2.rectangle에 해당 좌표 입력
사용자가 원하는 정답 영역을 그림

np.copy() = 값만 복사하고 새로운 개체 생성

# Data Processing Resize image and bounding boxes

```
[ ]  # resize the input images to h=800, w=800

     img = cv2.resize(img0, dsize=(800, 800), interpolation=cv2.INTER_CUBIC)
     plt.figure(figsize=(7, 7))
     plt.imshow(img)
     # plt.grid(True, color="black")
     plt.show()
```




cv2.resize()를 이용해 800*800 크기로 이미지 크기 조정
cv2.INTER_CUBIC = 3차 회선법으로 보간법 설정.
plt.figure(figsize=x,y) 표시할 그래프의 길이 설정(inch 단위)

# Data Processing

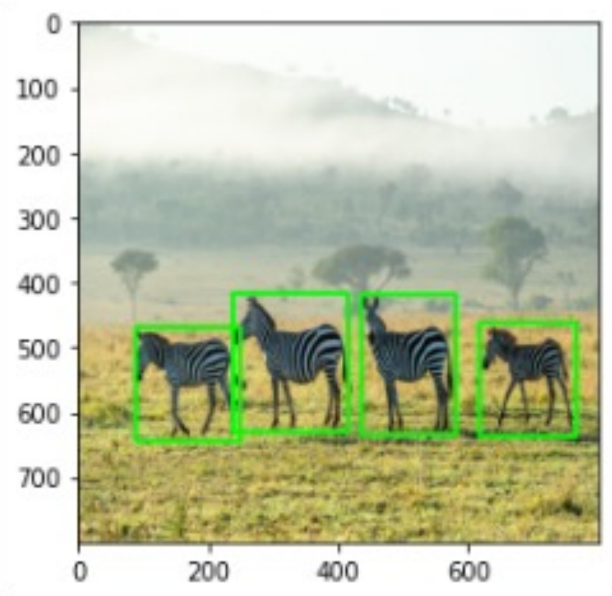
```
# change the bounding box coordinates
# original image size : (1333, 2000)

Wratio = 800/img0.shape[1]
Hratio = 800/img0.shape[0]

print(Wratio, Hratio)

ratioList = [Wratio, Hratio, Wratio, Hratio]
bbox = []

for box in bbox0:
    box = [int(a*b) for a, b in zip(box, ratioList)]
    bbox.append(box)

bbox = np.array(bbox)
print(bbox)

img_clone = np.copy(img)
for i in range(len(bbox)):
    cv2.rectangle(img_clone, (bbox[i][0], bbox[i][1]), (bbox[i][2], bbox[i][3]), color=(0, 255, 0), thickness=5)
plt.imshow(img_clone)
plt.show()
```

기존 이미지 크기로 나누어 변형한 이미지의 비율을 계산

변형된 bbox0의 각 rectangle 좌표 값을
zip() 함수로 묶어 bbox 라는 배열에 추가

bbox 배열 내의 좌표들을 이용해, 비율을
변형한 이미지에 rectangle을 그려준다.

```
0.4 0.6001500375093773
[[ 89 469 249 644]
 [238 417 415 630]
 [435 419 580 634]
 [617 462 765 637]]
```



# Define Feature Extractor Load pretrained VGG16

```python
# only print feature extraction part of VGG16

model = torchvision.models.vgg16(pretrained=True).to(DEVICE)
features = list(model.features)
print(len(features))
print(features)
```

**31**
**[Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False), Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False), Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False), Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False), Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)), ReLU(inplace=True), MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)]**

# Define Feature Extractor Only collect required layers

```python
# only collect layers with output feature map size (W, H) < 50

dummy_img = torch.zeros((1, 3, 800, 800)).float() # test image array
print(dummy_img.shape)

req_features = []
output = dummy_img.clone().to(DEVICE)

for feature in features:
    output = feature(output)
#     print(output.size()) => torch.Size([batch_size, channel, width, height])
    if output.size()[2] < 800//16: # 800/16=50
        break
    req_features.append(feature)
    out_channels = output.size()[1]

print(len(req_features))
# print(req_features)
print(out_channels)
```

```
torch.Size([1, 3, 800, 800])
30
512
```

sub-sampling 비율을 1/16으로 임의 설정

입력 이미지의 사이즈와 같은 Dummy 이미지를 만든다.

50 * 50크기의 feature맵을 출력하는 Layer 탐색

전체 모델에서 해당 Layer까지만을 사용 Layer 배열에 추가

전체 모델 중 30개의 Layer 사용

출력되는 feature map의 채널 수는 512개

# Define Feature Extractor

```
[ ]  # convert this list into a Seqeuntial module

     faster_rcnn_feature_extractor = nn.Sequential(*req_features)
```

사용 Layer 리스트를 nn.Sequential() 모듈의 형태로 변환

네트워크 모델 정의 및 Layer 형태로 순전파 작성

```
[ ]  # test the results of the input image pass through the feature extractor

     transform = transforms.Compose([transforms.ToTensor()])
     imgTensor = transform(img).to(DEVICE)
     imgTensor = imgTensor.unsqueeze(0)
     output_map = faster_rcnn_feature_extractor(imgTensor)

     print(output_map.size())
```

pixel 값들을 [0 ~ 255]에서 0.0 ~ 1.0]의
pytorch tensor 타입으로 변환한다.

unsqueeze(0)를 통해 첫번째 차원에 1이 추가된다.

```
torch.Size([1, 512, 50, 50])
```

Feature map은 다음과 같다.
1개의 이미지, 512개의 채널, 이미지의 크기는 50*50

# Define Feature Extractor

```
[ ]  # visualize the first 5 channels of the 50*50*512 feature maps

     imgArray = output_map.data.cpu().numpy().squeeze(0)
     fig = plt.figure(figsize=(12, 4))
     figNo = 1

     for i in range(5):
         fig.add_subplot(1, 5, figNo)
         plt.imshow(imgArray[i], cmap='gray')
         figNo += 1

     plt.show()
```

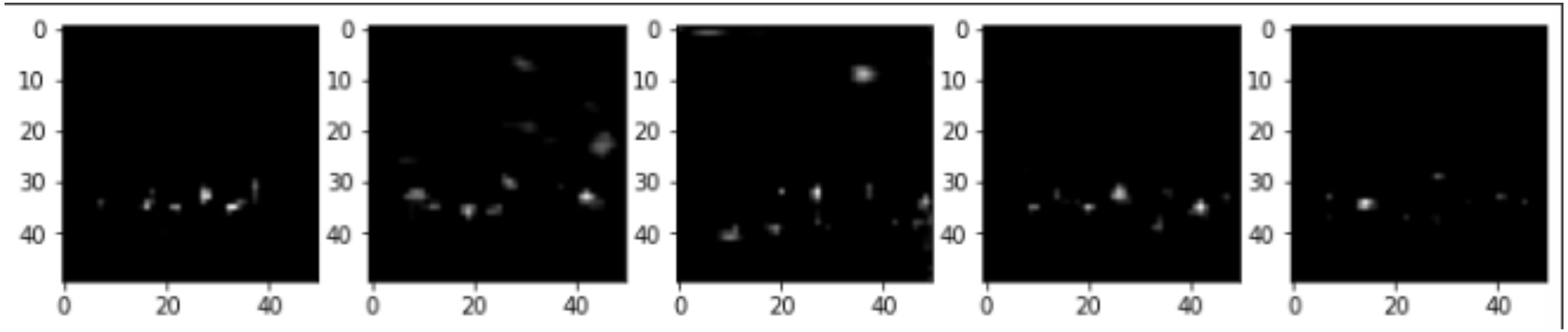
.cpu() = GPU 메모리에 올려진 tensor를 cpu 메모리로 복사

.numpy() = tensor를 numpy로 변환하여 반환

.squeeze() = 해당하는 차원에 있는 1을 제거한다.

plt.subplot 의 입력값은 행의 수, 열의 수, index 값

5개의 feature map을 gray 컬러맵으로 출력



# Generate Anchors Boxes Generate Anchors

```
[ ]  # sub-sampling rate = 1/16
     # image size : 800x800
     # sub-sampled feature map size : 800 x 1/16 = 50
     # 50 x 50 = 2500 anchors and each anchor generate 9 anchor boxes
     # total anchor boxes = 50 x 50 x 9 = 22500
     # x,y intervals to generate anchor box center

     feature_size = 800 // 16
     ctr_x = np.arange(16, (feature_size + 1) * 16, 16)
     ctr_y = np.arange(16, (feature_size + 1) * 16, 16)
     print(len(ctr_x))
     print(ctr_x)
```

Faster R-CNN에서는 anchor box를
Scale : (128, 256, 512), Aspect Ratio : (2:1, 1:1, 1:2)를 사용하여
9개로 설정하였다.

사전에 설정한 sub-sampling rate에 맞추어,
sub-sampled feature map의 크기는
50 * 50 이며, 각각의 anchor는 9개의 anchor box를
가지게 되므로 전체 anchor box의 개수는
50 * 50 * 9 = 22500 개이다.

원본 이미지를 일정하게 나누어 cell을 만들어줄
grid의 간격을 설정한다.
간격은 16, 16이다.

```
50
[ 16  32  48  64  80  96 112 128 144 160 176 192 208 224 240 256 272 288
 304 320 336 352 368 384 400 416 432 448 464 480 496 512 528 544 560 576
 592 608 624 640 656 672 688 704 720 736 752 768 784 800]
```

# Generate Anchors Boxes

```
[ ]  # coordinates of the 255 center points to generate anchor boxes

     index = 0
     ctr = np.zeros((2500, 2))

     for i in range(len(ctr_x)):
         for j in range(len(ctr_y)):
             ctr[index, 1] = ctr_x[i] - 8
             ctr[index, 0] = ctr_y[j] - 8
             index += 1

     # ctr => [[center x, center y], ...]
     print(ctr.shape)
     print(ctr[:10, :])
```

```
(2500, 2)
[[  8.   8.]
 [ 24.   8.]
 [ 40.   8.]
 [ 56.   8.]
 [ 72.   8.]
 [ 88.   8.]
 [104.   8.]
 [120.   8.]
 [136.   8.]
 [152.   8.]]
```

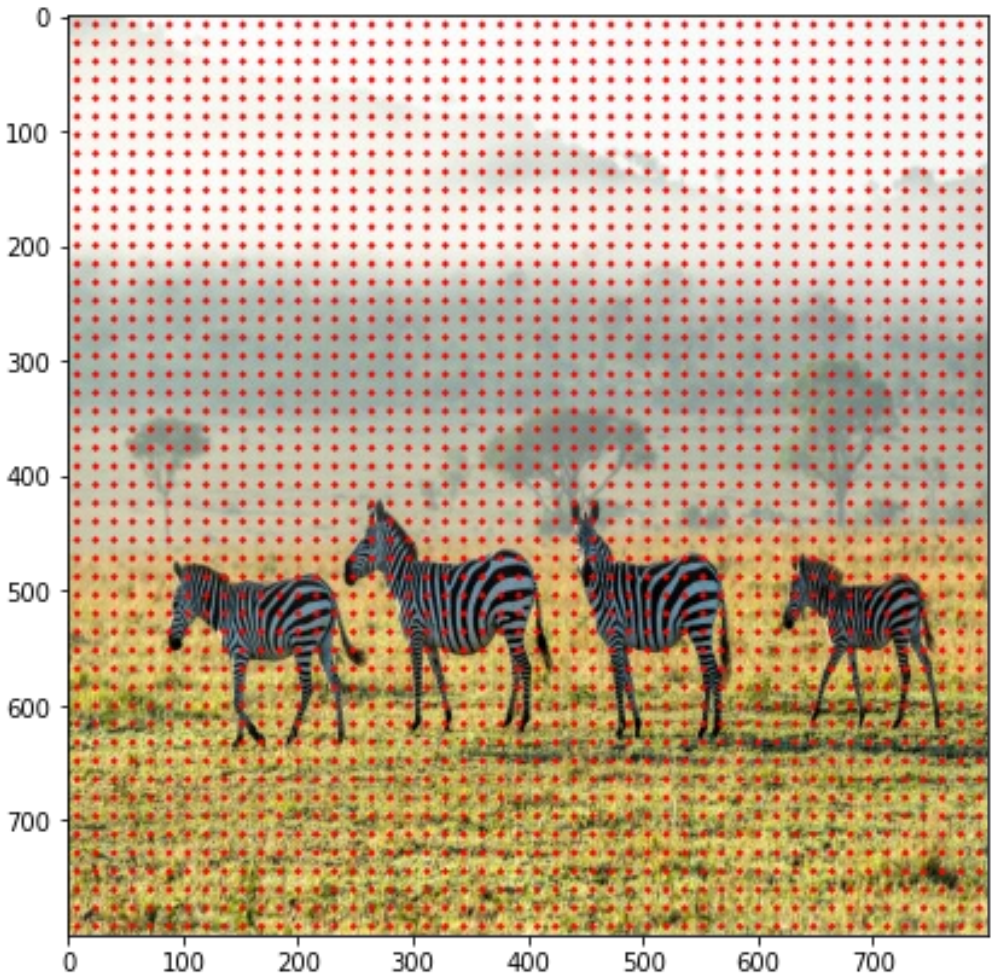
전체 anchor의 중심점 좌표를 담기 위해 np.zeros((2500,2)) 로 2500*2 크기의 행렬을 만든다.

전체 anchor들의 중심점 좌표를 계산한 뒤 shape을 출력해 확인한다.

좌상단부터, (8,8), (24, 8), (40, 8)... 의 순으로 좌표가 출력된다.

# Generate Anchors Boxes

```
[ ]  # display the 2500 anchors within image

     img_clone2 = np.copy(img)
     ctr_int = ctr.astype("int32")

     plt.figure(figsize=(7, 7))
     for i in range(ctr.shape[0]):
         cv2.circle(img_clone2, (ctr_int[i][0], ctr_int[i][1]),
                    radius=1, color=(255, 0, 0), thickness=3)
     plt.imshow(img_clone2)
     plt.show()
```




cv2.circle을 이용해 2500개의 anchor 위치를
기존 Resize한 이미지 위에 빨간 점으로 출력한다.

# Generate Anchors Boxes

```
[ ] # for each of the 2500 anchors, generate 9 anchor boxes
    # 2500 x 9 = 22500 anchor boxes

    ratios = [0.5, 1, 2]
    scales = [8, 16, 32]
    sub_sample = 16

    anchor_boxes = np.zeros(((feature_size * feature_size * 9), 4))
    index = 0

    for c in ctr:                        # per anchors
        ctr_y, ctr_x = c
        for i in range(len(ratios)):     # per ratios
            for j in range(len(scales)): # per scales

                # anchor box height, width
                h = sub_sample * scales[j] * np.sqrt(ratios[i])
                w = sub_sample * scales[j] * np.sqrt(1./ ratios[i])

                # anchor box [x1, y1, x2, y2]
                anchor_boxes[index, 1] = ctr_y - h / 2.
                anchor_boxes[index, 0] = ctr_x - w / 2.
                anchor_boxes[index, 3] = ctr_y + h / 2.
                anchor_boxes[index, 2] = ctr_x + w / 2.
                index += 1

    print(anchor_boxes.shape)
    print(anchor_boxes[:10, :])
```

```
(22500, 4)
[[ -82.50966799  -37.254834     98.50966799   53.254834  ]
 [-173.01933598  -82.50966799  189.01933598   98.50966799]
 [-354.03867197 -173.01933598  370.03867197  189.01933598]
 [ -56.          -56.           72.           72.        ]
 [-120.         -120.          136.          136.        ]
 [-248.         -248.          264.          264.        ]
 [ -37.254834    -82.50966799   53.254834     98.50966799]
 [ -82.50966799 -173.01933598   98.50966799  189.01933598]
 [-173.01933598 -354.03867197  189.01933598  370.03867197]
 [ -82.50966799  -21.254834     98.50966799   69.254834  ]]
```

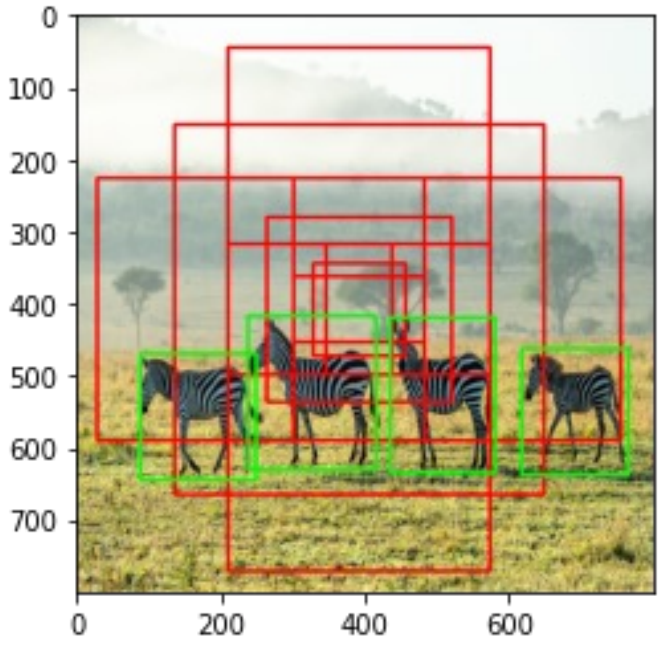
ratios, scales를 각각 3개씩 설정
따라서 anchor 당 box의 개수는 3*3개

sub_samping 비율은 16으로 설정
anchor_box의 전체개수를 정의

np.zeros로 22500 * 4 크기의 행렬 생성
22500개의 anchor box들의 좌표를 출력

# Generate Anchors Boxes

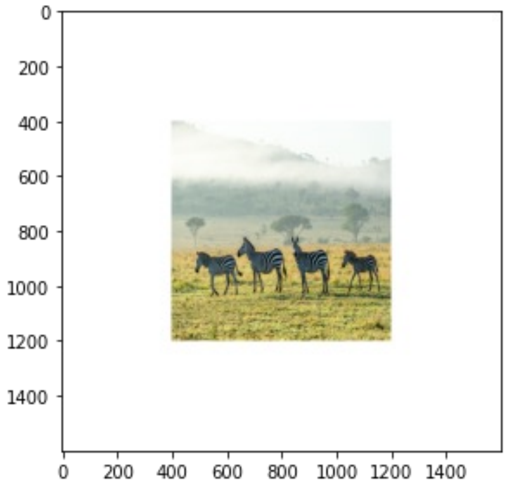
```
[ ]  # display the anchor boxes of one anchor and the ground truth boxes

     img_clone = np.copy(img)

     # draw random anchor boxes
     for i in range(11025, 11034):
         x1 = int(anchor_boxes[i][0])
         y1 = int(anchor_boxes[i][1])
         x2 = int(anchor_boxes[i][2])
         y2 = int(anchor_boxes[i][3])

         cv2.rectangle(img_clone, (x1, y1), (x2, y2), color=(255, 0, 0),
                       thickness=3)

     # draw ground truth boxes
     for i in range(len(bbox)):
         cv2.rectangle(img_clone, (bbox[i][0], bbox[i][1]),
                                  (bbox[i][2], bbox[i][3]),
                       color=(0, 255, 0), thickness=3)

     plt.imshow(img_clone)
     plt.show()
```

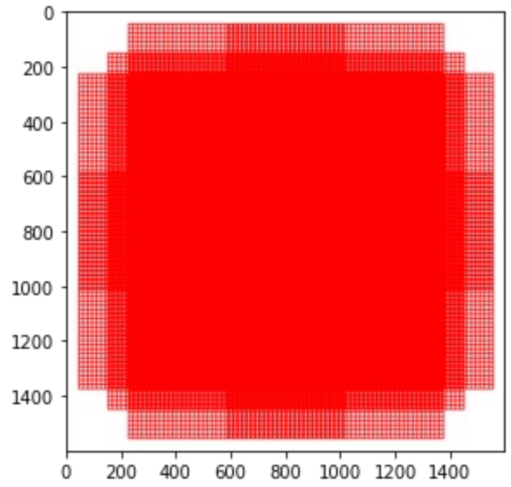



ground truth box와
anchor box를 함께 출력

# Generate Anchors Boxes

```python
# draw all anchor boxes

# add paddings(can't draw anchor boxes out of image boundary)
img_clone3 = np.copy(img)
img_clone4 = cv2.copyMakeBorder(img_clone3,400,400,400,400,cv2.BORDER_CONSTANT, value=(255, 255, 255))
img_clone5 = np.copy(img_clone4)

for i in range(len(anchor_boxes)):
    x1 = int(anchor_boxes[i][0])
    y1 = int(anchor_boxes[i][1])
    x2 = int(anchor_boxes[i][2])
    y2 = int(anchor_boxes[i][3])

    cv2.rectangle(img_clone5, (x1+400, y1+400), (x2+400, y2+400), color=(255, 0, 0),
                  thickness=3)

plt.figure(figsize=(10, 10))
plt.subplot(121), plt.imshow(img_clone4)
plt.subplot(122), plt.imshow(img_clone5)
plt.show()
```




원본 이미지에 padding 을 하고, 전체 anchor box를 표시한 이미지를 함께 출력

# Target Anchors

**Only choose anchor boxes inside the image**

```
[ ]  # ignore the cross-boundary anchor boxes
     # valid anchor boxes with (x1, y1) > 0 and (x2, y2) <= 800

     index_inside = np.where(
             (anchor_boxes[:, 0] >= 0) &
             (anchor_boxes[:, 1] >= 0) &
             (anchor_boxes[:, 2] <= 800) &
             (anchor_boxes[:, 3] <= 800))[0]

     print(index_inside.shape)

     # only 8940 anchor boxes are inside the boundary out of 22500
     valid_anchor_boxes = anchor_boxes[index_inside]
     print(valid_anchor_boxes.shape)
```

```
(8940,)
(8940, 4)
```

입력 이미지의 크기인 800 * 800 이내에
모든 box의 좌표를 가지고 있는
anchor box들의 개수와
행렬의 크기를 출력

22500개 중 경계 안의
Anchor box는 8940개 뿐인 것을 확인

따라서 해당 8940개의 anchor box를
RPN을 학습할 용도로써 선택

# Target Anchors

**Calculate IoUs**

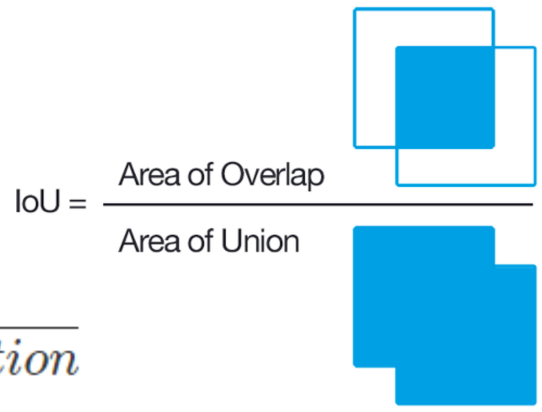
```python
ious = np.empty((len(valid_anchor_boxes),4), dtype=np.float32)
ious.fill(0)

# anchor boxes
for i, anchor_box in enumerate(valid_anchor_boxes):
    xa1, ya1, xa2, ya2 = anchor_box
    anchor_area = (xa2 - xa1) * (ya2 - ya1)

    # ground truth boxes
    for j, gt_box in enumerate(bbox):
        xb1, yb1, xb2, yb2 = gt_box
        box_area = (xb2 - xb1) * (yb2 - yb1)

        inter_x1 = max([xb1, xa1])
        inter_y1 = max([yb1, ya1])
        inter_x2 = min([xb2, xa2])
        inter_y2 = min([yb2, ya2])

        if (inter_x1 < inter_x2) and (inter_y1 < inter_y2):
            inter_area = (inter_x2 - inter_x1) * (inter_y2 - inter_y1)
            iou = inter_area / (anchor_area + box_area - inter_area)
        else:
            iou = 0

        ious[i, j] = iou

print(ious.shape)
print(ious[8930:8940, :])
```

```
(8940, 4)
[[0.         0.         0.         0.37780452]
 [0.         0.         0.         0.33321926]
 [0.         0.         0.         0.29009855]
 [0.         0.         0.         0.24967977]
 [0.         0.         0.         0.2117167 ]
 [0.         0.         0.         0.17599213]
 [0.         0.         0.         0.14231375]
 [0.         0.         0.         0.11051063]
 [0.         0.         0.         0.08043041]
 [0.         0.         0.         0.05193678]]
```

유효한 anchor box의 영역을 계산
ground truth 의 영역을 계산
두 영역 간의 IoU를 측정
IoU의 크기를 출력

Intersection over Union (IoU)
=>

$$IoU = \frac{A \cap B}{A \cup B}$$




$$IoU = \frac{Intersection}{A + B - Intersection}$$

# Target Anchors **Sample positive/negative anchor boxes**

```
[ ]  # what anchor box has max ou with the ground truth box

     gt_argmax_ious = ious.argmax(axis=0)
     print(gt_argmax_ious)

     gt_max_ious = ious[gt_argmax_ious, np.arange(ious.shape[1])]
     print(gt_max_ious)

     gt_argmax_ious = np.where(ious == gt_max_ious)[0]
     print(gt_argmax_ious)
```

```
[1008 2862 5935 8699]
[0.58514285 0.5752716  0.5255493  0.6325869 ]
[1008 1013 1018 1226 1232 1238 2862 2869 2876 3108 3115 3122 3336 3343
 3350 3354 3357 3361 3364 3368 3371 3377 3383 3389 3600 3607 3614 3846
 3853 3860 5935 5942 6164 6171 6178 6181 6185 6188 6192 6198 6427 6434
 8699 8703 8707]
```

axis=0을 설정하면 각 열을 따라
가장 높은 값의 인덱스를 반환

Ground Truth와의 IoU에서
가장 높은 Iou를 가진 anchor box의 인덱스를
반환하고, 해당 anchor box의 IoU를 출력

np.where 를 사용하여, gt_max_ious와
같은 값을 가지는 anchor box들의
인덱스 값을 출력

# Target Anchors

```
[ ]  # what ground truth bbox is associated with each anchor box

     argmax_ious = ious.argmax(axis=1)
     print(argmax_ious.shape)
     print(argmax_ious)

     max_ious = ious[np.arange(len(index_inside)), argmax_ious]
     print(max_ious)
```

```
(8940,)
[0 0 0 ... 3 3 3]
[0.         0.         0.         ... 0.11051063 0.08043041 0.05193678]
```

```
[ ]  # set the labels of 8940 valid anchor boxes to -1(ignore)

     label = np.empty((len(index_inside),), dtype=np.int32)
     label.fill(-1)
     print(label.shape)
```

```
(8940,)
```

axis=1을 설정하면 각 행을 따라
가장 높은 값의 인덱스를 반환

argmax_ious 의 크기와 값을 출력
⇒ (8940,)

유효 anchor_box 배열의 길이만큼,
모든 요소를 -1로 채운 label 행렬을 생성

# Target Anchors

```
[ ]  # use IoU to assign 1 (objects) to two kind of anchors
     # a) the anchors with the highest IoU overlap with a ground truth box
     # b) an anchor that has an IoU overlap higher than 0.7 with ground truth box

     # Assign 0 (background) to an anchor if its IoU ratio is lower than 0.3

     pos_iou_threshold = 0.7
     neg_iou_threshold = 0.3

     label[gt_argmax_ious] = 1
     label[max_ious >= pos_iou_threshold] = 1
     label[max_ious < neg_iou_threshold] = 0
```

positive IoU의 임계값을 0.7로
negative IoU의 임계값을 0.3으로 설정한다.
IoU의 최대값이 0.7보다 크거나 같으면 Label 값을 1, 즉 오브젝트로 배정하고
IoU의 최대값이 0.3보다 작으면 Label 값을 0, 즉 배경으로 배정한다.

# Target Anchors

```
[ ]  # Every time mini-batch training take only 256 valid anchor boxes to train RPN
     # of which 128 positive examples, 128 negative-examples
     # disable leftover positive/negative anchors
     n_sample = 256
     pos_ratio = 0.5
     n_pos = pos_ratio * n_sample

     pos_index = np.where(label == 1)[0]

     if len(pos_index) > n_pos:
         disable_index = np.random.choice(pos_index,
                                          size = (len(pos_index) - n_pos),
                                          replace=False)
         label[disable_index] = -1

     n_neg = n_sample * np.sum(label == 1)
     neg_index = np.where(label == 0)[0]

     if len(neg_index) > n_neg:
         disable_index = np.random.choice(neg_index,
                                          size = (len(neg_index) - n_neg),
                                          replace = False)
         label[disable_index] = -1
```

mini-batch의 수는 256개,
positive/negative sample의 비율이
1:1이 되도록 구성.

앞에서 정의한 positive, negative의
label을 이용

positive sample의 수가 128개
이상인 경우, 남는 positive sample에
해당  sample은 label 변수에  -1로 지정.

negative sample 또한 마찬가지.

# Target Anchors

```
max_iou_bbox = bbox[argmax_ious]
print(max_iou_bbox.shape)

height = valid_anchor_boxes[:, 3] - valid_anchor_boxes[:, 1]
width = valid_anchor_boxes[:, 2] = valid_anchor_boxes[:, 0]
ctr_y = valid_anchor_boxes[:, 1] + 0.5 * height
ctr_x = valid_anchor_boxes[:, 0] + 0.5 * width

base_height = max_iou_bbox[:, 3] - max_iou_bbox[:, 1]
base_width = max_iou_bbox[:, 2] - max_iou_bbox[:, 0]
base_ctr_y = max_iou_bbox[:, 1] + 0.5 * base_height
base_ctr_x = max_iou_bbox[:, 0] + 0.5 * base_width

eps = np.finfo(height.dtype).eps
height = np.maximum(height, eps)
width = np.maximum(width, eps)

dy = (base_ctr_y - ctr_y) / height
dx = (base_ctr_x - ctr_x) / width
dh = np.log(base_height / height)
dw = np.log(base_width / width)

anchor_locs = np.vstack((dx, dy, dw, dh)).transpose()
print(anchor_locs.shape)
```

```
(8940, 4)
(8940, 4)
```

유효한 anchor box의 형식을

x1, y1, x2, y2의 형식으로 변환

즉, anchor box의 location인
height, width, ctr_y, ctr_x를 도출

**eps:** 표현 가능한 가장 작은 값을 반환
**np.finfo:** float 데이터형의 범위를 검출
**np.maximum:** 여러 array 사이에서 최댓값 도출
**vstack:** 세로로 배열을 결합할 때 사용
**np.log:** log을 적용한 값을 리턴

# Target Anchors

```
anchor_labels = np.empty((len(anchor_boxes),), dtype=label.dtype)
anchor_labels.fill(-1)
anchor_labels[index_inside] = label
print(anchor_labels.shape)
print(anchor_labels[:10])

anchor_locations = np.empty((len(anchor_boxes),) + anchor_boxes.shape[1:], dtype=anchor_locs.dtype)
anchor_locations.fill(0)
anchor_locations[index_inside, :] = anchor_locs
print(anchor_locations.shape)
print(anchor_locations[:10, :])
```

```
(22500,)
[-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
(22500, 4)
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

전체 anchor box 개수 크기의 행렬 anchor_labels,
이후 RPN에서 ROI 손실 계산을 위해
입력 받는 모든 anchor box의 좌표값이 필요하므로
22500 * 4 크기의 anchor_locations 행렬 생성

# RPN **Define RPN**

```
in_channels = 512
mid_channels = 512
n_anchor = 9

conv1 = nn.Conv2d(in_channels, mid_channels, 3, 1, 1).to(DEVICE)
conv1.weight.data.normal_(0, 0.01)
conv1.bias.data.zero_()

# bounding box regressor
reg_layer = nn.Conv2d(mid_channels, n_anchor * 4, 1, 1, 0).to(DEVICE)
reg_layer.weight.data.normal_(0, 0.01)
reg_layer.bias.data.zero_()

# classifier(object or not)
cls_layer = nn.Conv2d(mid_channels, n_anchor * 2, 1, 1, 0).to(DEVICE)
cls_layer.weight.data.normal_(0, 0.01)
cls_layer.bias.data.zero_()
```

VGG16 모델을 통과하며 생성된
feature map에 3x3 conv 연산을
적용하는 layer를 정의

1x1 conv 연산을 적용하여
9x4(anchor box의 수 x bounding box coordinates)개의
channel을 가지는 feature map을 반환하는
Bounding box regressor를 정의

마찬가지로 1x1 conv 연산을 적용하여
9x2(anchor box의 수 x object 여부)개의
channel을 가지는 feature map을
반환하는 Classifier를 정의

```
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

# RPN **Classification and Bounding box regression**

```
[538] x = conv1(output_map.to(DEVICE)) # output_map = faster_rcnn_feature_extractor(imgTensor)
      pred_anchor_locs = reg_layer(x) # bounding box regresor output
      pred_cls_scores = cls_layer(x)  # classifier output

      print(pred_anchor_locs.shape, pred_cls_scores.shape)
```

```
torch.Size([1, 36, 50, 50]) torch.Size([1, 18, 50, 50])
```

```
pred_anchor_locs = pred_anchor_locs.permute(0, 2, 3, 1).contiguous().view(1, -1, 4)
print(pred_anchor_locs.shape)

pred_cls_scores = pred_cls_scores.permute(0, 2, 3, 1).contiguous()
print(pred_cls_scores.shape)

objectness_score = pred_cls_scores.view(1, 50, 50, 9, 2)[:, :, :, :, 1].contiguous().view(1, -1)
print(objectness_score.shape)

pred_cls_scores = pred_cls_scores.view(1, -1, 2)
print(pred_cls_scores.shape)
```

```
torch.Size([1, 22500, 4])
torch.Size([1, 50, 50, 18])
torch.Size([1, 22500])
torch.Size([1, 22500, 2])
```

50x50x512 크기의 feature map을
3x3 conv layer에 입력

이를 통해 얻은 50x50x512 크기의
feature map을 Bounding box regressor,
Classifier에 입력하여 각각
bounding box coefficients
(=pred_anchor_locs)와
objectness score(=pred_cls_scores)를 획득

이를 target 값(bbox를 통해 만든 ground
truth 값들)과 비교하기 위해
permute, contiguous, view를 사용하여
tensor를 적절히 resize

# RPN

```
[540] # According to the 22500 ROIs predicted by RPN and 22500 anchor boxes,
      # calculate the RPN loss¶
      print(pred_anchor_locs.shape)
      print(pred_cls_scores.shape)
      print(anchor_locations.shape)
      print(anchor_labels.shape)
```

```
torch.Size([1, 22500, 4])
torch.Size([1, 22500, 2])
(22500, 4)
(22500,)
```

ROI : Region of Interest

RPN과 22500개의 Anchor box에 의해 예측된
22500개의 ROI에 대한 RPN score를 측정

```
[541] rpn_loc = pred_anchor_locs[0]
      rpn_score = pred_cls_scores[0]

      gt_rpn_loc = torch.from_numpy(anchor_locations).to(DEVICE)
      gt_rpn_score = torch.from_numpy(anchor_labels).to(DEVICE)

      print(rpn_loc.shape, rpn_score.shape,
            gt_rpn_loc.shape, gt_rpn_score.shape)
```

rpn_loc, rpn_score를 gt_rpn_loc과 gt_rpn_score의
shape과 비교하여 손실 정도를 비교

```
torch.Size([22500, 4]) torch.Size([22500, 2]) torch.Size([22500, 4]) torch.Size([22500])
```

# RPN **Multi-task loss**

```
[542] # For classification we use cross-entropy loss
      rpn_cls_loss = F.cross_entropy(rpn_score, gt_rpn_score.long().to(DEVICE), ignore_index = -1)
      print(rpn_cls_loss)
```

```
tensor(0.6827, grad_fn=<NllLossBackward0>)
```

```
[543] # only positive samples
      pos = gt_rpn_score > 0
      mask = pos.unsqueeze(1).expand_as(rpn_loc)
      print(mask.shape)

      # take those bounding boxes whick have positive labels
      mask_loc_preds = rpn_loc[mask].view(-1, 4)
      mask_loc_targets = gt_rpn_loc[mask].view(-1, 4)
      print(mask_loc_preds.shape, mask_loc_targets.shape)

      x = torch.abs(mask_loc_targets.cpu() - mask_loc_preds.cpu())
      rpn_loc_loss = ((x < 1).float() * 0.5 * x ** 2) + ((x >= 1).float() * (x - 0.5))
      print(rpn_loc_loss.sum())
```

```
torch.Size([22500, 4])
torch.Size([45, 4]) torch.Size([45, 4])
tensor(15.9745, dtype=torch.float64, grad_fn=<SumBackward0>)
```

Classification loss는 cross entropy loss를 손실함수로 활용

Bounding box regression loss는
오직 positive에 해당하는 sample에 대해서만 loss를 계산하므로, positive/negative 여부를 저장하는 배열인 mask를 생성

# RPN

```
[544] # Combining both the rpn_cls_loss and rpn_reg_loss

      rpn_lambda = 10
      N_reg = (gt_rpn_score > 0).float().sum()
      rpn_loc_loss = rpn_loc_loss.sum() / N_reg
      rpn_loss = rpn_cls_loss + (rpn_lambda * rpn_loc_loss)
      print(rpn_loss)
```

```
tensor(4.2326, dtype=torch.float64, grad_fn=<AddBackward0>)
```

$i$ : mini-batch 내의 anchor의 index

$p_i$ : anchor $i$에 객체가 포함되어 있을 예측 확률

$p_i^*$ : anchor가 양성일 경우 1, 음성일 경우 0을 나타내는 index parameter

$t_i$ : 예측 bounding box의 파라미터화된 좌표(coefficient)

$t_i^*$ : ground truth box의 파라미터화된 좌표

$L_{cls}$ : Loss loss

$L_{reg}$ : Smooth L1 loss

$N_{cls}$ : mini-batch의 크기(논문에서는 256으로 지정)

$N_{reg}$ : anchor 위치의 수

$\lambda$ : balancing parameter(default=10)

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

# Proposal layer **Transform anchor boxes**

```
[545] # Send the 22500 ROIs predicted by RPN to Fast RCNN to predict bbox + classifications
      # First use NMS (Non-maximum supression) to reduce 22500 ROI to 2000

      nms_thresh = 0.7  # non-maximum supression (NMS)
      n_train_pre_nms = 12000 # no. of train pre-NMS
      n_train_post_nms = 2000 # after nms, training Fast R-CNN using 2000 RPN proposals
      n_test_pre_nms = 6000
      n_test_post_nms = 300 # During testing we evaluate 300 proposals,
      min_size = 16
```

```
[546] # the labelled 22500 anchor boxes
      # format converted from [x1, y1, x2, y2] to [ctrx, ctry, w, h]

      anc_height = anchor_boxes[:, 3] - anchor_boxes[:, 1]
      anc_width = anchor_boxes[:, 2] - anchor_boxes[:, 0]
      anc_ctr_y = anchor_boxes[:, 1] + 0.5 * anc_height
      anc_ctr_x = anchor_boxes[:, 0] + 0.5 * anc_width
      print(anc_ctr_x.shape)
```

```
(22500,)
```

Anchor generation layer에서 생성된
anchor boxes와 RPN에서 반환한 class scores,
bounding box regressor를 사용하여
region proposals를 추출

object score에서 상위 N개의
anchor box에 대하여
Non maximum suppression 알고리즘 수행
anchor box 중 상위 N개의
region proposals를 학습에 사용
=> anchor box 가 객체의 위치를 더 잘 detect

# Proposal layer

```
[547] # The 22500 anchor boxes location and labels predicted by RPN (convert to numpy)
      # format = (dx, dy, dw, dh)

      pred_anchor_locs_numpy = pred_anchor_locs[0].cpu().data.numpy()
      objectness_score_numpy = objectness_score[0].cpu().data.numpy()

      dy = pred_anchor_locs_numpy[:, 1::4]
      dx = pred_anchor_locs_numpy[:, 0::4]
      dh = pred_anchor_locs_numpy[:, 3::4]
      dw = pred_anchor_locs_numpy[:, 2::4]
      print(dy.shape)
```

```
(22500, 1)
```

```
ctr_y = dy * anc_height[:, np.newaxis] + anc_ctr_y[:, np.newaxis]
ctr_x = dx * anc_width[:, np.newaxis] + anc_ctr_x[:, np.newaxis]
h = np.exp(dh) * anc_height[:, np.newaxis]
w = np.exp(dw) * anc_width[:, np.newaxis]
print(w.shape)
```

```
(22500, 1)
```

pred_anchor_locs와 objectness score 를 (dx, dy, dw, dh) 형태의 넘파이 객체로 변환

array의 차원을 늘려주는 np.newaxis를 활용

중심좌표 ctr_x, ctr_y와 h, w를 정의

# Proposal layer

```
[549] roi = np.zeros(pred_anchor_locs_numpy.shape, dtype=anchor_locs.dtype)
      roi[:, 0::4] = ctr_x - 0.5 * w
      roi[:, 1::4] = ctr_y - 0.5 * h
      roi[:, 2::4] = ctr_x + 0.5 * w
      roi[:, 3::4] = ctr_y + 0.5 * h

      print(roi.shape)
```

```
(22500, 4)
```

np.zeros를 사용하여
예측한 anchor box 만큼의
크기를 가진 행렬을 생성하고

roi 행렬의 크기를 반환한다.

# Proposal layer **Clip the anchor boxes to the image**

```
[550] # clip the predcited boxes to the image

      img_size = (800, 800)
      roi[:, slice(0, 4, 2)] = np.clip(roi[:, slice(0, 4, 2)], 0, img_size[0]) # [:, 0, 2]
      roi[:, slice(1, 4, 2)] = np.clip(roi[:, slice(1, 4, 2)], 0, img_size[1]) # [:, 1, 3]

      print(roi.shape, np.max(roi), np.min(roi))
```

```
(22500, 4) 800.0 0.0
```

```
hs = roi[:, 3] - roi[:, 1]
ws = roi[:, 2] - roi[:, 0]

keep = np.where((hs >= min_size) & (ws >= min_size))[0]
roi = roi[keep, :]
score = objectness_score_numpy[keep]
print(keep.shape, roi.shape, score.shape)
```

```
(22500,) (22500, 4) (22500,)
```

np.clip() 을 이용해 array의 최소값, 최댓값을 지정
hs와 ws를 각 roi의 열을 뺀 값으로 정의

# Proposal layer **Select top-12000 anchor boxes by objectness score**

```
[552] # sort all (proposal, score) pairs by score from highest to lowest

      order = score.ravel().argsort()[::-1]
      print(order.shape)
```

```
(22500,)
```

```
[553] # take top pre_nms_topN (e.g. 12000 while training and 300 while testing)
      order = order[:n_train_pre_nms]
      roi = roi[order, :]
      print(order.shape, roi.shape)
```

```
(12000,) (12000, 4)
```

ravel() = 다차원 배열을
1차원으로 풀어줄 때 사용

argsort()[::-1] = 크기가 큰 순서부터
순서대로 데이터의 index를 반환

상위에서 12000개의 proposal을 선정

# Proposal layer Non maximum suppression(select 2000 bounding boxes)

```
[554] # take all the roi boxes
      x1 = roi[:, 0]
      y1 = roi[:, 1]
      x2 = roi[:, 2]
      y2 = roi[:, 3]

      # find the areas of all the boxes

      areas = (x2 - x1 + 1) * (y2 - y1 + 1)
```

모든 roi box를 호출하고

모든 box의 영역을 계산하여

areas에 저장

# Proposal layer

```python
order = order.argsort()[::-1]
keep = []

while (order.size > 0):
  i = order[0] # take the 1st elt in roder and append to keep
  keep.append(i)

  xx1 = np.maximum(x1[i], x1[order[1:]])
  yy1 = np.maximum(y1[i], y1[order[1:]])
  xx2 = np.minimum(x2[i], x2[order[1:]])
  yy2 = np.minimum(y2[i], y2[order[1:]])

  w = np.maximum(0.0, xx2 - xx1 + 1)
  h = np.maximum(0.0, yy2 - yy1 + 1)

  inter = w * h
  ovr = inter / (areas[i] + areas[order[1:]] - inter)
  inds = np.where(ovr <= nms_thresh)[0]
  order = order[inds + 1]

keep = keep[:n_train_post_nms] # while training/testing, use accordingly
roi = roi[keep]
print(len(keep), roi.shape)
```


nms를 적용하여 ROI proposal을

12000개에서 2000개로 줄인다.

즉, 이후에 Fast R-CNN을 학습할 때
쓰일 proposal은 2000개로 줄어든다.

그러므로 roi의 shape은
(22500, 4) -> (12000, 4) -> (2000, 4)로 변화

```
2000 (2000, 4)
```

# Proposal Target layer **Calculate IoUs**

```
n_sample = 128 # number of samples from roi
pos_ratio = 0.25 # number of positive examples out of the n_samples
pos_iou_thresh = 0.5 # min iou of region proposal with any ground truth object to consider it as positive label
neg_iou_thresh_hi = 0.5 # iou 0~0.5 is considered as negative (0, background)
neg_iou_thresh_lo = 0.0
```

RoI sample의 수, 전체 샘플 중 positive sample의 비율,
최소 IoU를 가지는 positive sample,
negative sample의 최대 최소 범위를 지정

# Proposal Target layer

```
ious = np.empty((len(roi), bbox.shape[0]), dtype = np.float32)
ious.fill(0)

for num1, i in enumerate(roi):
  ya1, xa1, ya2, xa2 = i
  anchor_area = (ya2 - ya1) * (xa2 - xa1)

  for num2, j in enumerate(bbox):
    yb1, xb1, yb2, xb2 = j
    box_area = (yb2 - yb1) * (xb2 - xb1)
    inter_x1 = max([xb1, xa1])
    inter_y1 = max([yb1, ya1])
    inter_x2 = min([xb2, xa2])
    inter_y2 = min([yb2, ya2])

    if (inter_x1 < inter_x2) and (inter_y1 < inter_y2):
      inter_area = (inter_y2 - inter_y1) * (inter_x2 - inter_x1)
      iou = inter_area / (anchor_area + box_area - inter_area)
    else:
      iou = 0
    ious[num1, num2] = iou

print(ious.shape)
```

각 region proposal과
ground truth간의
IoU를 조정한다.

앞과 같은 IoU 도출 공식

```
(2000, 4)
```

# Proposal Target layer

```
[558] # find out whick ground truth has high IoU for each region proposal
      # also find the maximum IoU

      gt_assignment = ious.argmax(axis=1)
      max_iou = ious.max(axis=1)

      print(gt_assignment)
      print(max_iou)

      # assign the labels to each proposal
      gt_roi_label = labels[gt_assignment]
      print(gt_roi_label)
```

```
[3 3 0 ... 0 0 0]
[0.39086348 0.17093568 0.         ... 0.14352609 0.         0.        ]
[1 1 1 ... 1 1 1]
```

가장 높은 IoU를 보이는 region proposal과
IoU의 최댓값을 탐색

argmax(axis=1)을 이용하여
ious의 각 행에 대한
모든 열에 대해서
최댓값의 인덱스 데이터를 반환

max()를 사용하여
가장 큰 IoU 값 순서대로 도출

# Proposal Target layer **Select foreground(positive) samples**

```
[559] # select the foreground rois as pre the pos_iou_thresh
      # and n_sample x pos_ratio (128 x 0.25 = 32) foreground samples

      pos_roi_per_image = 32
      pos_index = np.where(max_iou >= pos_iou_thresh)[0]
      pos_roi_per_this_image = int(min(pos_roi_per_image, pos_index.size))

      if pos_index.size > 0:
        pos_index = np.random.choice(
            pos_index, size=pos_roi_per_this_image, replace=False)

      print(pos_roi_per_this_image)
      print(pos_index)
```

```
12
[1435 1486   22  511 1024  552  474   17  553   81  392 1488]
```

np.random.choice() 형태와
size=pos_roi_per_this_image를
이용하여
positive sample의 개수를 정수로 반환

각각의 인덱스 값을 반환

# Proposal Target layer

**Select background(negative) samples**

```
[560] # similarly we do for negative(background) region proposals

      neg_index = np.where((max_iou < neg_iou_thresh_hi) &
                           (max_iou >= neg_iou_thresh_lo))[0]
      neg_roi_per_this_image = n_sample - pos_roi_per_this_image
      neg_roi_per_this_image = int(min(neg_roi_per_this_image, neg_index.size))

      if neg_index.size > 0:
        neg_index = np.random.choice(
          neg_index, size = neg_roi_per_this_image, replace=False)

      print(neg_roi_per_this_image)
      print(neg_index)
```

```
116
[ 895  618 1610 1589  936  349  246  133 1827 1342  772  182 1266 1291
  892 1433  797  470  113  924   35 1124  831  770  479  292  395  546
  290 1778  263 1106 1759  865  381  400   57  509  672 1592  868  272
 1511 1473 1468 1552  225  968  379  447  978 1851 1231 1199 1249 1120
 1416 1770 1172 1402 1581 1396  421  128 1650 1490 1669  597 1185 1053
 1062  471  467  759 1832  339   49 1359 1782 1472  329 1594 1240  884
 1083 1791  101 1901  943 1226 1800 1687 1287 1520 1691 1857  773  161
 1899 1940 1911  192 1742 1198  853 1054  752  402 1834  350 1489 1424
  891  418  683 1968]
```

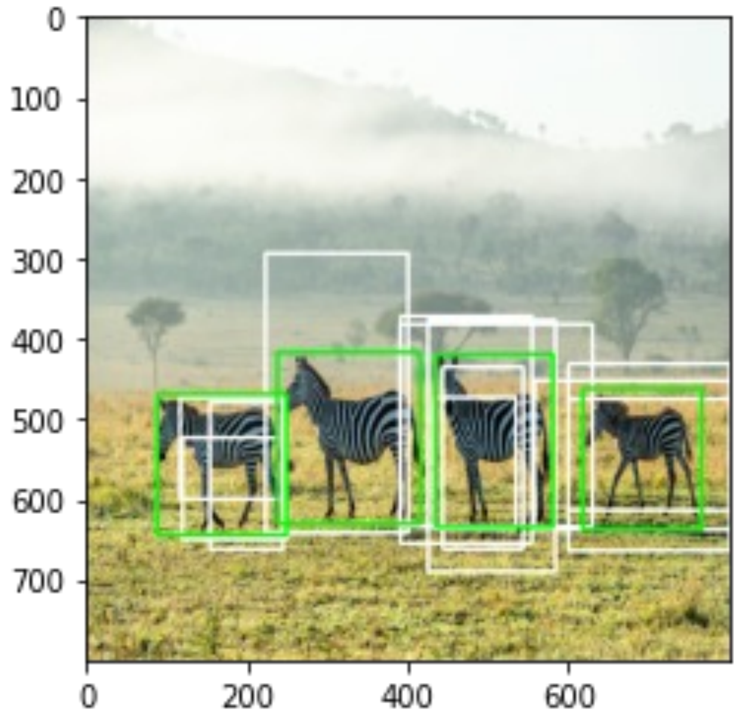
동일한 방식으로

negative samples의 개수와

각각의 인덱스 값을 반환

# Proposal Target layer **Visualization**

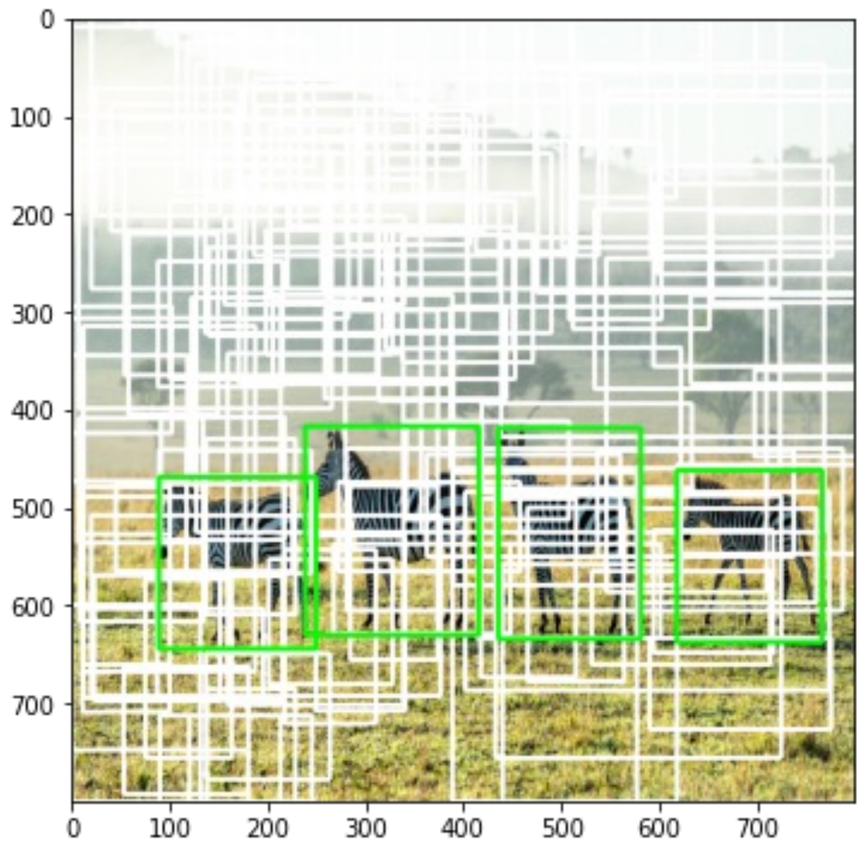
```
[561] # display RoI samples with positive

    img_clone = np.copy(img)

    for i in range(pos_roi_per_this_image):
      x1, y1, x2, y2 = roi[pos_index[i]].astype(int)
      cv2.rectangle(img_clone, (x1, y1), (x2, y2), color=(255,255,255),
                    thickness=3)

    for i in range(len(bbox)):
      cv2.rectangle(img_clone, (bbox[i][0], bbox[i][1]), (bbox[i][2], bbox[i][3]),
                    color = (0, 255, 0), thickness=3)

    plt.imshow(img_clone)
    plt.show()
```




positive samples를 gt와 함께 시각화

# Proposal Target layer

```python
# display RoI samples with negative

img_clone = np.copy(img)

plt.figure(figsize=(9, 6))

for i in range(neg_roi_per_this_image):
  x1, y1, x2, y2 = roi[neg_index[i]].astype(int)
  cv2.rectangle(img_clone, (x1, y1), (x2, y2), color=(255, 255, 255),
                thickness=3)

for i in range(len(bbox)):
  cv2.rectangle(img_clone, (bbox[i][0], bbox[i][1]), (bbox[i][2], bbox[i][3]),
                color = (0, 255, 0), thickness=3)

plt.imshow(img_clone)
plt.show()
```




negative samples를 gt와 함께 시각화

# Proposal Target layer **Gather positive/negative samples**

```
[563] # now we gather positive samples index and negative samples index
      # their respective labels and region proposals

      keep_index = np.append(pos_index, neg_index)
      gt_roi_labels = gt_roi_label[keep_index]
      gt_roi_labels[pos_roi_per_this_image:] = 0 # negative labels => 0
      sample_roi = roi[keep_index]
      print(sample_roi.shape)
```

```
(128, 4)
```

positive index와 negative index의 label과 region proposal를 수집하기 위해 keep_index 배열에 집어넣는다.

```
# pick the ground truth objects for these sample_roi and
# later parameterized as we have done while assigning locations to
# anchor boxes

bbox_for_sampled_roi = bbox[gt_assignment[keep_index]]
print(bbox_for_sampled_roi.shape)
```

```
(128, 4)
```

# Proposal Target layer

```
[565] width = sample_roi[:, 2] - sample_roi[:, 0]
      height = sample_roi[:, 3] - sample_roi[:, 1]
      ctr_x = sample_roi[:, 0] + 0.5 * width
      ctr_y = sample_roi[:, 1] + 0.5 * height

      base_width = bbox_for_sampled_roi[:, 2] - bbox_for_sampled_roi[:, 0]
      base_height = bbox_for_sampled_roi[:, 3] - bbox_for_sampled_roi[:, 1]
      base_ctr_x = bbox_for_sampled_roi[:, 0] + 0.5 * base_width
      base_ctr_y = bbox_for_sampled_roi[:, 1] + 0.5 * base_height
```

Target anchor를 변환할 때와
유사한 방법으로
anchor sample의 형태 변환을 진행한다.

```
[566] # transform anchor boxes

      eps = np.finfo(height.dtype).eps
      height = np.maximum(height, eps)
      width = np.maximum(width, eps)

      dx = (base_ctr_x - ctr_x) / width
      dy = (base_ctr_y - ctr_y) / height
      dw = np.log(base_width / width)
      dh = np.log(base_height / height)

      gt_roi_locs = np.vstack((dx, dy, dw, dh)).transpose()
      print(gt_roi_locs.shape)
```

**eps:** 표현 가능한 가장 작은 값을 반환
**np.finfo:** float 데이터형의 범위를 검출
**np.maximum:** 여러 array 사이에서 최댓값 도출
**vstack:** 세로로 배열을 결합할 때 사용
**np.log:** log을 적용한 값을 리턴

Fast R-CNN 모델을 학습시키기 위해
수집되어진 sample의 shape은 128 * 4

```
(128, 4)
```

# RoI Pooling

**Concatenate labels with bbox coordinates**

```
# Take out the features of 128 ROI samples and
# use max pooling to adjust to the same size, H=7, W=7 (ROI Pooling)

rois = torch.from_numpy(sample_roi).float()
roi_indices = 0 * np.ones((len(rois),), dtype=np.int32)
roi_indices = torch.from_numpy(roi_indices).float()
print(rois.shape, roi_indices.shape)
```

```
torch.Size([128, 4]) torch.Size([128])
```

```
indices_and_rois = torch.cat([roi_indices[:, None], rois], dim=1)
xy_indices_and_rois = indices_and_rois[:, [0, 2, 1, 4, 3]]
indices_and_rois = xy_indices_and_rois.contiguous()
print(xy_indices_and_rois.shape)
```

```
torch.Size([128, 5])
```

입력 값으로 VGG16 모델에 입력하여 얻은 feature map과 region proposal 단계에서 얻은 sample을 사용하여 ROI Pooling 수행

# RoI Pooling

```
[569] size = (7, 7)
      adaptive_max_pool = nn.AdaptiveMaxPool2d(size[0], size[1])

[570] output = []
      rois = indices_and_rois.data.float()
      rois[:, 1:].mul_(1/16.0) # sub-sampling ratio
      rois = rois.long()
      num_rois = rois.size(0)

      for i in range(num_rois):
        roi = rois[i]
        im_idx = roi[0]
        im = output_map.narrow(0, im_idx, 1)[..., roi[1]:(roi[3]+1), roi[2]:(roi[4]+1)]
        tmp = adaptive_max_pool(im)
        output.append(tmp[0])

      output = torch.cat(output, 0)

      print(output.size())
```

```
torch.Size([128, 512, 7, 7])
```

7*7 크기의 max pooling 진행

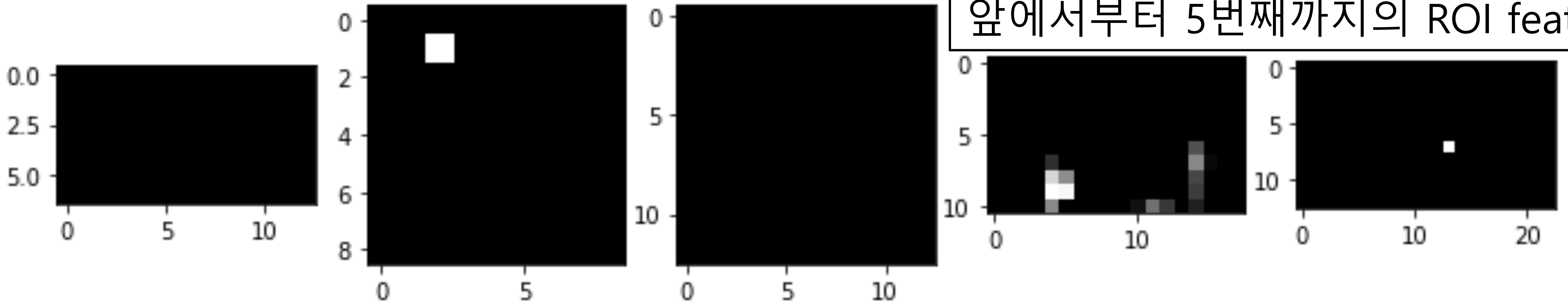
pooling 출력값의 size는

128, 512, 7, 7

# RoI Pooling

```
[571] # Visualize the first 5 ROI's feature map (for each feature map, only show the 1st channel of d=512)
      fig=plt.figure(figsize=(12, 4))
      figNo = 1
      for i in range(5):
          roi = rois[i]
          im_idx = roi[0]
          im = output_map.narrow(0, im_idx, 1)[..., roi[2]:(roi[4]+1), roi[1]:(roi[3]+1)]
          tmp = im[0][0].detach().cpu().numpy()
          fig.add_subplot(1, 5, figNo)
          plt.imshow(tmp, cmap='gray')

          figNo +=1
      plt.show()
```

앞에서부터 5번째까지의 ROI feature map을 출력한다.



# RoI Pooling

```
[572] # Reshape the tensor so that we can pass it through the feed forward layer.
      k = output.view(output.size(0), -1)
      print(k.shape) # 25088 = 7*7*512
```

```
torch.Size([128, 25088])
```

Fast R-CNN의 fc layer 입력 데이터 형식에 맞추어 주기 위하여,
출력된 pooling 결과값의 shape을 (128, 25088)로 수정한다.

# Faster R-CNN

## Classifier and Bounding box regressor

```
# 128 boxes + features (7x7x512) of ROI samples are sent to
# Detection network to predict the objects bounding box and clas of the input image

roi_head_classifier = nn.Sequential(*[nn.Linear(25088, 4096), nn.Linear(4096, 4096)]).to(DEVICE)
cls_loc = nn.Linear(4096, 2 * 4).to(DEVICE) # 1 class, 1 background, 4 coordiinates
cls_loc.weight.data.normal_(0, 0.01)
cls_loc.bias.data.zero_()

score = nn.Linear(4096, 2).to(DEVICE) # 1 class, 1 background
```

```
# passing the output of roi pooling to RoI head

k = roi_head_classifier(k.to(DEVICE))
roi_cls_loc = cls_loc(k)
roi_cls_score = score(k)

print(roi_cls_loc.shape, roi_cls_score.shape)
```

```
torch.Size([128, 8]) torch.Size([128, 2])
```

기존 Fast R-CNN 모델의 동작순서와 동일

RoI pooling을 통해 얻은 7x7 크기의
feature map을 입력 받을 fc layer를 정의

입력 받은 Feature map을 fc layer에 입력하여
4096크기의 feature vector 획득

class별로 bounding box coefficients를 예측하는 Bounding box
regressor(위치)와 class score를 예측하는 Classifier(유무)를 정의

# Faster R-CNN

```
[575] # Calculate the loss of Fast RCNN based on the gt bboxes and features (h, w, d=512)
      # corresponding to these 128 ROIs

      # predicted
      print(roi_cls_loc.shape)
      print(roi_cls_score.shape)

      #actual
      print(gt_roi_locs.shape)
      print(gt_roi_labels.shape)
```

```
torch.Size([128, 8])
torch.Size([128, 2])
(128, 4)
(128,)
```

모델이 예측한 box와 ground truth의 shape을 비교 출력해본다.

# Faster R-CNN

Classification loss

```
[577] # Converting ground truth to torch variable
      gt_roi_loc = torch.from_numpy(gt_roi_locs).to(DEVICE)
      gt_roi_label = torch.from_numpy(np.float32(gt_roi_labels)).long().to(DEVICE)
      print(gt_roi_loc.shape, gt_roi_label.shape)

      #Classification loss
      roi_cls_loss = F.cross_entropy(roi_cls_score, gt_roi_label, ignore_index=-1)
```

```
torch.Size([128, 4]) torch.Size([128])
```

ground truth를 이후 연산을 위해 torch.from_numpy().to(DEVICE)를 사용하여 pytorch 변수 형태로 변환시킨다.

classification loss는 cross_entropy 함수를 사용

# Faster R-CNN

Regression loss

```
[578] # regression loss

      n_sample = roi_cls_loc.shape[0]
      roi_loc = roi_cls_loc.view(n_sample, -1, 4)
      print(roi_loc.shape)
```

```
torch.Size([128, 2, 4])
```

```
[579] roi_loc = roi_loc[torch.arange(0, n_sample).long(), gt_roi_label]
      print(roi_loc.shape)
```

```
torch.Size([128, 4])
```

n_sample = 128
roi_loc 정의

# Faster R-CNN

```
[580] # for regression we use smooth l1 loss as defined in the Fast R-CNN paper
      pos = gt_roi_label > 0
      mask = pos.unsqueeze(1).expand_as(roi_loc)
      print(mask.shape)
```

```
torch.Size([128, 4])
```

Fast R-CNN 논문과 같이, regression의 loss는 smooth L1을 이용한다.

```
[581] # take those bounding boxes which have positive labels
      mask_loc_preds = roi_loc[mask].view(-1, 4)
      mask_loc_targets = gt_roi_loc[mask].view(-1, 4)
      print(mask_loc_preds.shape, mask_loc_targets.shape)

      x = torch.abs(mask_loc_targets - mask_loc_preds)
      roi_loc_loss = ((x < 1).float() * 0.5 * x ** 2) + ((x >= 1).float() * (x - 0.5))
      print(roi_loc_loss.sum())
```

```
torch.Size([12, 4]) torch.Size([12, 4])
tensor(3.5132, dtype=torch.float64, grad_fn=<SumBackward0>)
```

mask_loc_preds, mask_loc_targets를 이용하여 roi_loc_loss를 도출

# Faster R-CNN Multi-task loss

```
[582] roi_lambda = 10.
      roi_loss = roi_cls_loss + (roi_lambda * roi_loc_loss)
      print(roi_loss)
      total_loss = rpn_loss + roi_loss
      print(total_loss)
```

```
tensor([[3.6426, 1.8577, 3.7935, 0.9058],
        [1.2350, 1.5520, 0.9770, 0.9027],
        [1.0115, 1.2581, 0.9492, 0.7975],
        [0.9960, 1.8966, 0.7128, 1.6389],
        [0.7408, 0.9856, 1.0345, 4.2783],
        [0.7276, 0.7162, 1.4828, 0.7219],
        [1.2352, 2.2655, 0.7210, 3.3100],
        [0.7161, 1.6014, 0.7213, 1.0138],
        [0.7146, 0.7284, 1.2729, 1.3647],
        [0.7221, 2.0118, 2.2017, 1.0999],
        [1.4661, 1.5460, 0.7147, 2.7933],
        [2.2144, 0.9867, 2.1571, 0.8353]], dtype=torch.float64,
       grad_fn=<AddBackward0>)
tensor([[7.8752, 6.0904, 8.0262, 5.1384],
        [5.4676, 5.7846, 5.2096, 5.1353],
        [5.2441, 5.4907, 5.1818, 5.0301],
        [5.2286, 6.1293, 4.9454, 5.8716],
        [4.9734, 5.2182, 5.2671, 8.5109],
        [4.9602, 4.9488, 5.7154, 4.9545],
        [5.4678, 6.4981, 4.9536, 7.5426],
        [4.9487, 5.8340, 4.9539, 5.2464],
        [4.9472, 4.9610, 5.5055, 5.5973],
        [4.9547, 6.2444, 6.4343, 5.3325],
        [5.6987, 5.7786, 4.9473, 7.0259],
        [6.4471, 5.2194, 6.3897, 5.0679]], dtype=torch.float64,
       grad_fn=<AddBackward0>)
```

도출한 roi_loss, roi_loc_loss 등을 조합하여
Multi-task loss를 진행

lambda 값은 10으로 지정
roi_loss와 total_loss 출력

출력된 loss를 이용해 Fast R-CNN을 학습

# Faster R-CNN

- https://github.com/herbwood/pytorch_faster_r_cnn/blob/main/faster_r_cnn.ipynb

# MMDetection

- OpenMMLab에서는 많은 최신 모델을 Open Source Projects로 구현하여 공개
- Pytorch 기반의 Object Detection 오픈소스 라이브러리

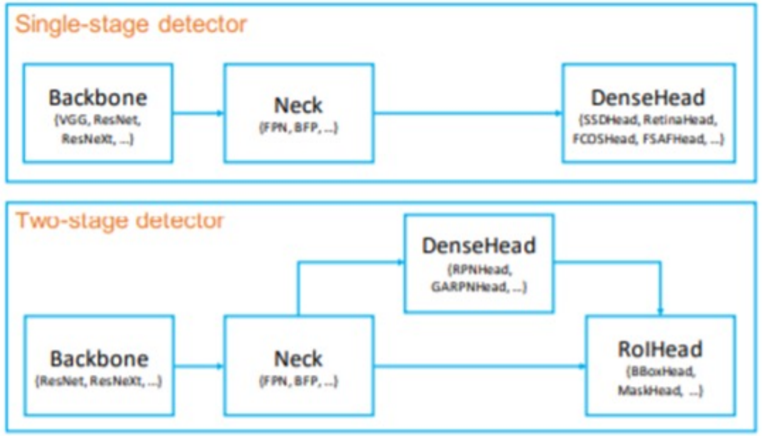



Figure 1: Framework of single-stage and two-stage detectors, illustrated with abstractions in MMDetection.

- MMCV: Computer Vision
- MMDetection
- MMAction2
- MMClassification
- MMSegmentation
- MMDetection3D
- MMEditing: Image and Video Editing
- MMPose: Pose estimation
- MMTracking
- MMOCR
- MMGeneration

Supported backbones:

- ResNet (CVPR'2016)
- ResNeXt (CVPR'2017)
- VGG (ICLR'2015)
- HRNet (CVPR'2019)
- RegNet (CVPR'2020)
- Res2Net (TPAMI'2020)
- ResNeSt (ArXiv'2020)

Supported methods:

- RPN (NeurIPS'2015)
- Fast R-CNN (ICCV'2015)
- Faster R-CNN (NeurIPS'2015)
- Mask R-CNN (ICCV'2017)
- Cascade R-CNN (CVPR'2018)
- Cascade Mask R-CNN (CVPR'2018)
- SSD (ECCV'2016)
- RetinaNet (ICCV'2017)
- GHM (AAAI'2019)
- Mask Scoring R-CNN (CVPR'2019)
- Double-Head R-CNN (CVPR'2020)
- Hybrid Task Cascade (CVPR'2019)

# MMDetection

## Prerequisites

- Linux or macOS (Windows is in experimental support)
- Python 3.6+
- PyTorch 1.3+
- CUDA 9.2+ (If you build PyTorch from source, CUDA 9.0 is also compatible)
- GCC 5+
- MMCV

# MMDetection

```
# Check nvcc version
!nvcc -V
# Check GCC version
!gcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_14_21:12:58_PST_2021
Cuda compilation tools, release 11.2, V11.2.152
Build cuda_11.2.r11.2/compiler.29618528_0
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

# CUDA 버전 확인

# GNU 컴파일러

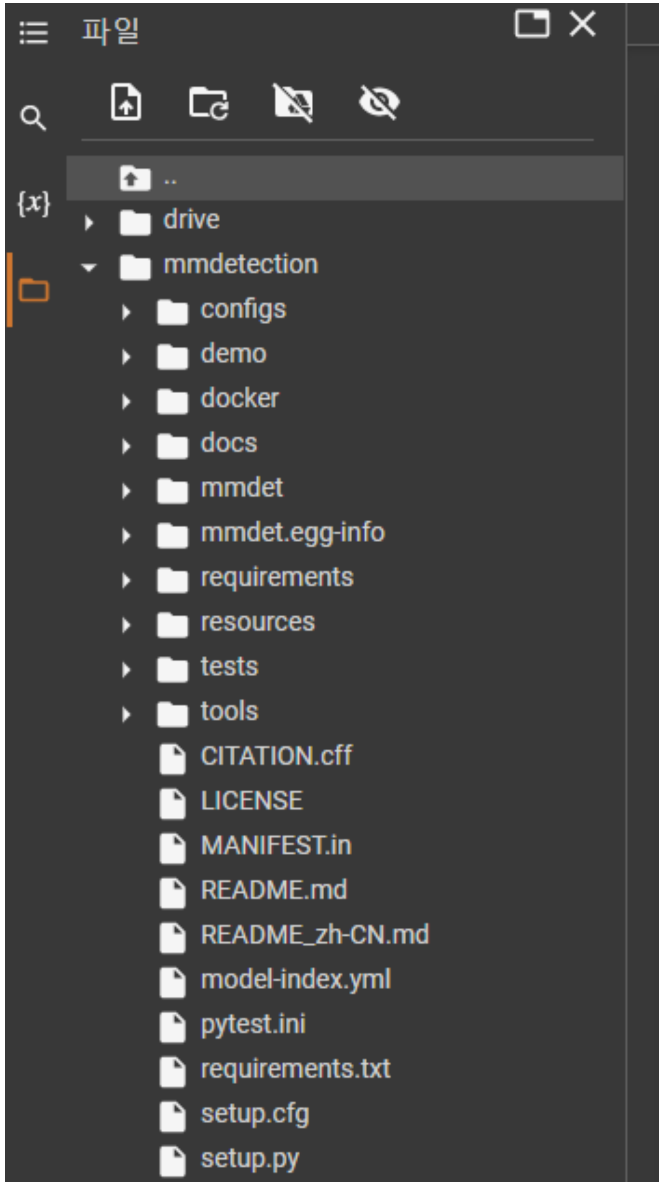
```
# install dependencies: (use cu111 because colab has CUDA 11.1)
!pip install torch==1.9.0+cu111 torchvision==0.10.0+cu111 -f https://download.pytorch.org/whl/torch_stable.htm

# install mmcv-full thus we could use CUDA operators
!pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu111/torch1.9.0/index.html

# Install mmdetection
!rm -rf mmdetection
!git clone https://github.com/open-mmlab/mmdetection.git
%cd mmdetection

!pip install -e .
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.9.0+cu111
  Downloading https://download.pytorch.org/whl/cu111/torch-1.9.0%2Bcu111-cp38-cp38-linux_x86_64.whl (2041.3 MB
                                                                        2.0/2.0 GB 52.6 MB/s eta
```

# rm –rf : 현재 디렉토리 내에 있는 mmdetection 폴더 삭제

# -e : This will install the current package in your Python environment in an editable mode.

# MMDetection



# MMDetection

```
# We download the pre-trained checkpoints for inference and finetuning.
!mkdir checkpoints
!wget -c https://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth \
      -O checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth
```

```
--2023-02-03 05:24:52--  https://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-
Resolving download.openmmlab.com (download.openmmlab.com)... 8.48.85.214, 8.48.85.211, 8.48.85.209, ...
Connecting to download.openmmlab.com (download.openmmlab.com)|8.48.85.214|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 167291982 (160M) [application/octet-stream]
Saving to: 'checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth'

checkpoints/faster_ 100%[===================>] 159.54M  10.6MB/s    in 15s

2023-02-03 05:25:08 (10.3 MB/s) - 'checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth' saved [167291982/167291982]
```

- mmdetection
  - checkpoints
    - faster_rcnn_r50_caffe_fpn_m...

# MMDetection

```python
import mmcv
from mmcv.runner import load_checkpoint

from mmdet.apis import inference_detector, show_result_pyplot
from mmdet.models import build_detector

# Choose to use a config and initialize the detector
config = 'configs/faster_rcnn/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco.py'
# Setup a checkpoint file to load
checkpoint = 'checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth'

# Set the device to be used for evaluation
device='cuda:0'

# Load the config
config = mmcv.Config.fromfile(config)
# Set pretrained to be None since we do not need pretrained model here
config.model.pretrained = None

# Initialize the detector
model = build_detector(config.model)

# Load checkpoint
checkpoint = load_checkpoint(model, checkpoint, map_location=device)

# Set the classes of models for inference
model.CLASSES = checkpoint['meta']['CLASSES']

# We need to set the model's cfg for inference
model.cfg = config

# Convert the model to GPU
model.to(device)
# Convert the model into evaluation mode
model.eval()
```

사용하고자 하는 모델의 config 경로

Checkpoint 파일

Config 로드

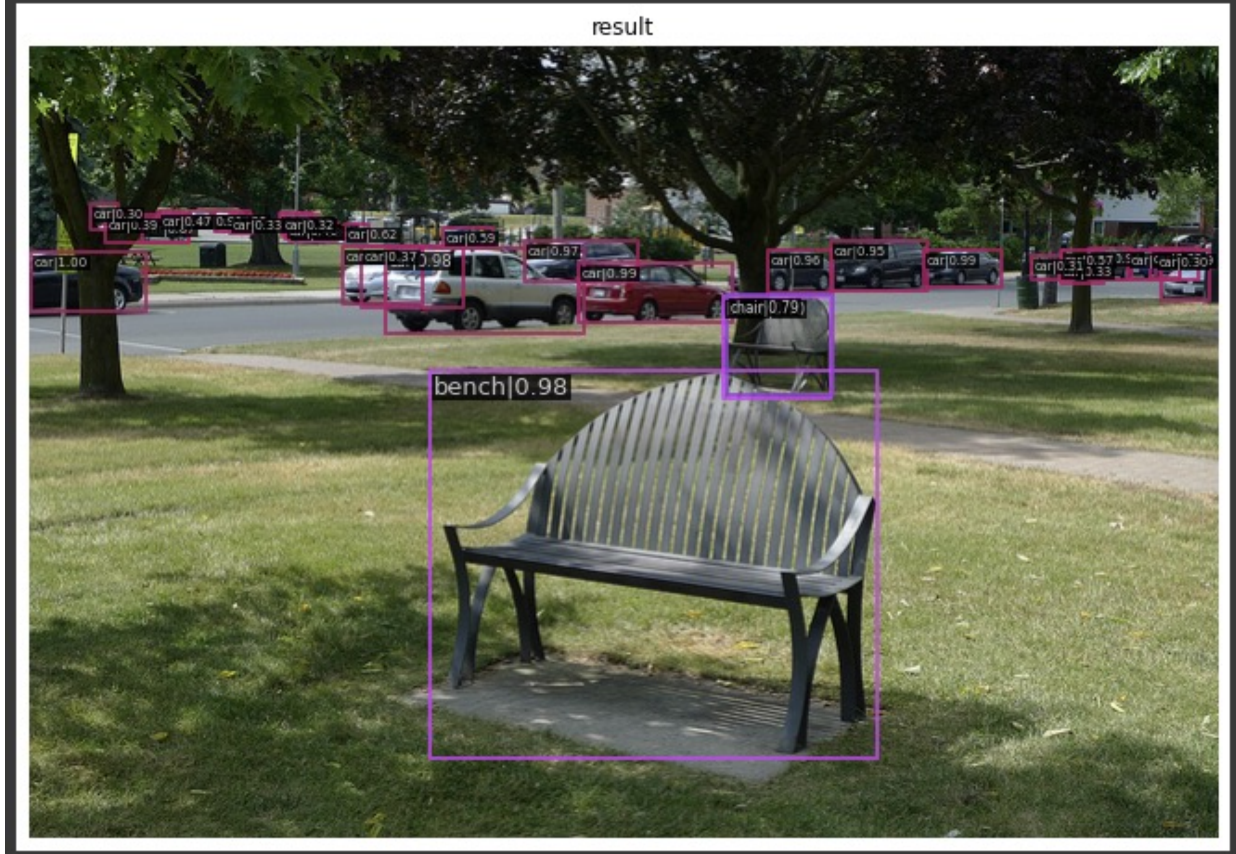
build_detector : detector 초기화

Load_checkpoint: checkpoint 로드

# MMDetection

```python
# Use the detector to do inference
img = 'demo/demo.jpg'
result = inference_detector(model, img)
```

```python
# Let's plot the result
show_result_pyplot(model, img, result, score_thr=0.3)
```



# Train A Detector on A Customized Dataset

1. Support a new dataset
2. Modify the config
3. Train a new detector

# Support a new dataset

```
# download, decompress the data
!wget https://download.openmmlab.com/mmdetection/data/kitti_tiny.zip
!unzip kitti_tiny.zip > /dev/null
```

```
# Check the directory structure of the tiny data

# Install tree first
!apt-get -q install tree
!tree kitti_tiny
```

```
(Reading database ... 129496 files and directories currently installed.)
Preparing to unpack .../tree_1.8.0-1_amd64.deb ...
Unpacking tree (1.8.0-1) ...
Setting up tree (1.8.0-1) ...
Processing triggers for man-db (2.9.1-1) ...
kitti_tiny
├── training
│   ├── image_2
│   │   ├── 000000.jpeg
│   │   ├── 000001.jpeg
│   │   ├── 000002.jpeg
│   │   ├── 000003.jpeg
│   │   ├── 000004.jpeg
│   │   ├── 000005.jpeg
```

디렉토리 구조 확인

# Train A Detector on A Customized Dataset

```
img = cv2.cvtColor(cv2.imread('./kitti_tiny/training/image_2/000068.jpeg'), cv2.COLOR_BGR2RGB)
plt.figure(figsize=(15, 10))
plt.imshow(img)

!cat ./kitti_tiny/training/label_2/000068.txt
```

```
Car 0.25 0 1.94 69.26 200.28 435.08 374.00 1.46 1.62 4.04 -3.00 1.79 6.98 1.55
Car 0.80 1 2.26 0.00 209.20 198.59 374.00 1.46 1.58 3.72 -5.44 1.85 6.22 1.56
Cyclist 0.97 0 2.34 1210.28 199.77 1241.00 374.00 1.55 0.57 1.73 4.04 1.69 3.57 -3.14
Car 0.00 2 1.68 478.18 187.68 549.54 249.43 1.57 1.60 3.99 -2.73 2.03 20.96 1.55
Car 0.00 1 1.66 530.03 187.79 573.10 226.09 1.52 1.54 3.68 -2.53 2.20 31.50 1.58
Van 0.00 1 1.63 547.61 171.12 584.05 212.41 2.47 1.98 5.81 -2.79 2.41 46.44 1.57
Car 0.00 1 -0.16 667.74 182.35 790.82 230.38 1.62 1.65 4.14 4.19 1.99 25.95 0.00
Car 0.00 2 -0.11 657.37 184.48 763.34 221.64 1.55 1.66 4.47 4.35 2.10 32.00 0.02
Car 0.00 1 -0.01 637.45 180.34 714.44 212.34 1.69 1.76 4.12 3.59 2.12 39.79 0.08
Van 0.00 1 1.61 572.52 175.02 596.26 199.95 2.13 1.91 6.40 -2.28 2.36 65.43 1.57
Van 0.00 1 1.77 380.78 167.69 523.29 288.56 1.95 1.75 4.63 -2.89 1.90 14.05 1.57
Cyclist 0.00 1 1.09 958.95 167.55 1036.88 254.43 1.68 0.53 1.96 7.95 1.59 14.95 1.57
```

# Train A Detector on A Customized Dataset

```
[
    {
        'filename': 'a.jpg',
        'width': 1280,
        'height': 720,
        'ann': {
            'bboxes': <np.ndarray> (n, 4),
            'labels': <np.ndarray> (n, ),
            'bboxes_ignore': <np.ndarray> (k, 4), (optional field)
            'labels_ignore': <np.ndarray> (k, 4) (optional field)
        }
    },
    ...
]
```

filename: 이미지 파일명(디렉토리는 포함하지 않음)

- width: 이미지 너비
- height: 이미지 높이
- ann: bbounding box와 label에 대한 정보를 가지는 Dictionary

- bboxes: 하나의 이미지에 있는 여러 Object 들의 numpy array. 4개의 좌표값(좌상단, 우하단)을 가지고, 해당 이미지에 n개의 Object들이 있을 경우 array의 shape는 (n, 4)
- labels: 하나의 이미지에 있는 여러 Object들의 numpy array. shape는 (n, )
- bboxes_ignore: 학습에 사용되지 않고 무시하는 bboxes. 무시하는 bboxes의 개수가 k개이면 shape는 (k, 4)
- labels_ignore: 학습에 사용되지 않고 무시하는 labels. 무시하는 bboxes의 개수가 k개이면 shape는 (k,)

# Train A Detector on A Customized Dataset

```python
import copy
import os.path as osp

import mmcv
import numpy as np

from mmdet.datasets.builder import DATASETS
from mmdet.datasets.custom import CustomDataset

@DATASETS.register_module()
class KittiTinyDataset(CustomDataset):

    CLASSES = ('Car', 'Pedestrian', 'Cyclist')

    def load_annotations(self, ann_file):
        cat2label = {k: i for i, k in enumerate(self.CLASSES)}
        # load image list from file
        image_list = mmcv.list_from_file(self.ann_file)

        data_infos = []
        # convert annotations to middle format
        for image_id in image_list:
            filename = f'{self.img_prefix}/{image_id}.jpeg'
            image = mmcv.imread(filename)
            height, width = image.shape[:2]

            data_info = dict(filename=f'{image_id}.jpeg', width=width, height=height)
```

Dataset 객체를 Config에 등록

CustomDataset 상속

# Load_annotations : 원하는 dataset format으로 변환하는 함수.

# self.ann_file : ./kitti_tiny/train.txt

# data_infos : middle format 데이터를 담을 list

self.img_prefix: ./kitti_tiny/training/image_2

Data_info : 각 image별 annotation 정보 가지는 dict

# Train A Detector on A Customized Dataset

```python
# load annotations
label_prefix = self.img_prefix.replace('image_2', 'label_2')
lines = mmcv.list_from_file(osp.join(label_prefix, f'{image_id}.txt'))

content = [line.strip().split(' ') for line in lines]
bbox_names = [x[0] for x in content]
bboxes = [[float(info) for info in x[4:8]] for x in content]

gt_bboxes = []
gt_labels = []
gt_bboxes_ignore = []
gt_labels_ignore = []

# filter 'DontCare'
for bbox_name, bbox in zip(bbox_names, bboxes):
    if bbox_name in cat2label:
        gt_labels.append(cat2label[bbox_name])
        gt_bboxes.append(bbox)
    else:
        gt_labels_ignore.append(-1)
        gt_bboxes_ignore.append(bbox)
```

# bbox_names : 오브젝트의 클래스명

# bboxs : bbox 좌표

self.img_prefix : ./kitti_tiny/training/label_2

```python
lines=mmcv.list_from_file('./kitti_tiny/training/label_2/000068.txt')
lines
```

```
['Car 0.25 0 1.94 69.26 200.28 435.08 374.00 1.46 1.62 4.04 -3.00 1.79 6.98 1.55',
 'Car 0.80 1 2.26 0.00 209.20 198.59 374.00 1.46 1.58 3.72 -5.44 1.85 6.22 1.56',
 'Cyclist 0.97 0 2.34 1210.28 199.77 1241.00 374.00 1.55 0.57 1.73 4.04 1.69 3.57 -3.14',
 'Car 0.00 2 1.68 478.18 187.68 549.54 249.43 1.57 1.60 3.99 -2.73 2.03 20.96 1.55',
 'Car 0.00 1 1.66 530.03 187.79 573.10 226.09 1.52 1.54 3.68 -2.53 2.20 31.50 1.58',
 'Van 0.00 1 1.63 547.61 171.12 584.05 212.41 2.47 1.98 5.81 -2.79 2.41 46.44 1.57',
 'Car 0.00 1 -0.16 667.74 182.35 790.82 230.38 1.62 1.65 4.14 4.19 1.99 25.95 0.00',
 'Car 0.00 2 -0.11 657.37 184.48 763.34 221.64 1.55 1.66 4.47 4.35 2.10 32.00 0.02',
 'Car 0.00 1 -0.01 637.45 180.34 714.44 212.34 1.69 1.76 4.12 3.59 2.12 39.79 0.08',
 'Van 0.00 1 1.61 572.52 175.02 596.26 199.95 2.13 1.91 6.40 -2.28 2.36 65.43 1.57',
 'Van 0.00 1 1.77 380.78 167.69 523.29 288.56 1.95 1.75 4.63 -2.89 1.90 14.05 1.57',
 'Cyclist 0.00 1 1.09 958.95 167.55 1036.88 254.43 1.68 0.53 1.96 7.95 1.59 14.95 1.57']
```

```python
content = [line.strip().split(' ') for line in lines]
content
```

```
[['Car',
  '0.25',
  '0',
  '1.94',
  '69.26',
  '200.28',
  '435.08',
  '374.00',
  '1.46',
  '1.62',
  '4.04',
  '-3.00',
  '1.79',
  '6.98',
  '1.55'],
 ['Car',
  '0.80',
  '1',
  '2.26',
  '0.00',
  '209.20',
  '198.59',
```

```python
bboxes
```

```
[[69.26, 200.28, 435.08, 374.0],
 [0.0, 209.2, 198.59, 374.0],
 [1210.28, 199.77, 1241.0, 374.0],
 [478.18, 187.68, 549.54, 249.43],
 [530.03, 187.79, 573.1, 226.09],
 [547.61, 171.12, 584.05, 212.41],
 [667.74, 182.35, 790.82, 230.38],
 [657.37, 184.48, 763.34, 221.64],
 [637.45, 180.34, 714.44, 212.34],
 [572.52, 175.02, 596.26, 199.95],
 [380.78, 167.69, 523.29, 288.56],
 [958.95, 167.55, 1036.88, 254.43]]
```

# Train A Detector on A Customized Dataset

```
    data_anno = dict(
        bboxes=np.array(gt_bboxes, dtype=np.float32).reshape(-1, 4),
        labels=np.array(gt_labels, dtype=np.long),
        bboxes_ignore=np.array(gt_bboxes_ignore,
                               dtype=np.float32).reshape(-1, 4),
        labels_ignore=np.array(gt_labels_ignore, dtype=np.long))

    data_info.update(ann=data_anno)
    data_infos.append(data_info)

return data_infos
```

# 각 image별 annotation 정보를 가지는 Dict

# update() : key-value 추가

# data_infos : 전체 annotation 파일들에 대한 정보를 가지는 list

# Modify the config

```python
from mmcv import Config
cfg = Config.fromfile('./configs/faster_rcnn/faster_rcnn_r50_caffe_fpn_mstrain_1x_coco.py')
```

```python
from mmdet.apis import set_random_seed

# Modify dataset type and path
cfg.dataset_type = 'KittiTinyDataset'
cfg.data_root = 'kitti_tiny/'

cfg.data.test.type = 'KittiTinyDataset'
cfg.data.test.data_root = 'kitti_tiny/'
cfg.data.test.ann_file = 'train.txt'
cfg.data.test.img_prefix = 'training/image_2'

cfg.data.train.type = 'KittiTinyDataset'
cfg.data.train.data_root = 'kitti_tiny/'
cfg.data.train.ann_file = 'train.txt'
cfg.data.train.img_prefix = 'training/image_2'

cfg.data.val.type = 'KittiTinyDataset'
cfg.data.val.data_root = 'kitti_tiny/'
cfg.data.val.ann_file = 'val.txt'
cfg.data.val.img_prefix = 'training/image_2'

# modify num classes of the model in box head
cfg.model.roi_head.bbox_head.num_classes = 3
# If we need to finetune a model based on a pre-trained detector, we need to
# use load_from to set the path of checkpoints.
cfg.load_from = 'checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth'

# Set up working dir to save files and logs.
cfg.work_dir = './tutorial_exps'
```

# dataset type과 path, file명 수정

# 클래스 개수 수정 80 -> 3

# 사전학습된 모델의 checkpoints

# Modify the config

```
# The original learning rate (LR) is set for 8-GPU training.
# We divide it by 8 since we only use one GPU.
cfg.optimizer.lr = 0.02 / 8
cfg.lr_config.warmup = None
cfg.log_config.interval = 10

# Change the evaluation metric since we use customized dataset.
cfg.evaluation.metric = 'mAP'
# We can set the evaluation interval to reduce the evaluation times
cfg.evaluation.interval = 12
# We can set the checkpoint saving interval to reduce the storage cost
cfg.checkpoint_config.interval = 12

# Set seed thus the results are more reproducible
cfg.seed = 0
set_random_seed(0, deterministic=False)
cfg.gpu_ids = range(1)

# We can also use tensorboard to log the training process
cfg.log_config.hooks = [
    dict(type='TextLoggerHook'),
    dict(type='TensorboardLoggerHook')]


# We can initialize the logger for training and have a look
# at the final config used for training
print(f'Config:\n{cfg.pretty_text}')

cfg.device='cuda'
```

# learning rate 변경

# 객체 탐지 정확도 평가 지표

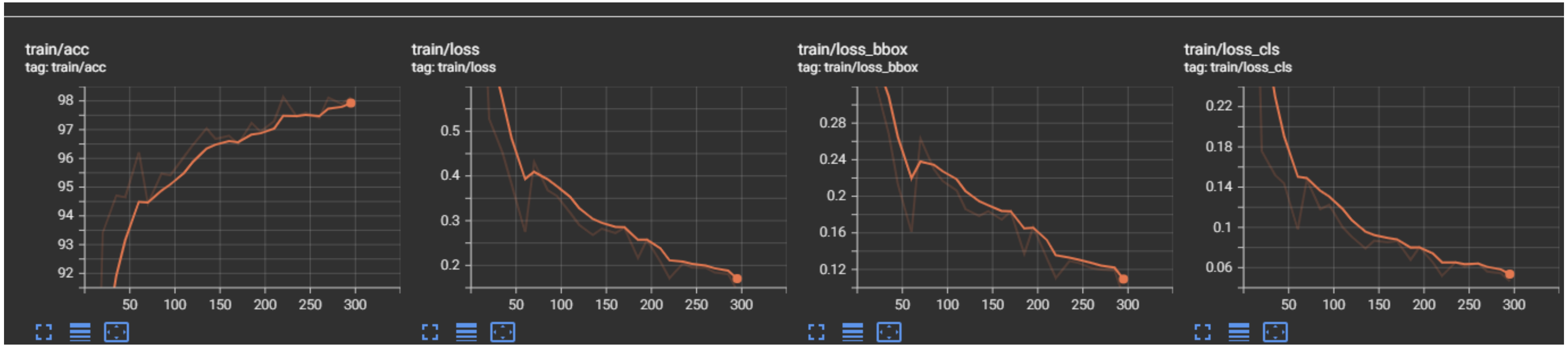
# seed 고정

# Train a new detector

```python
from mmdet.datasets import build_dataset
from mmdet.models import build_detector
from mmdet.apis import train_detector


# Build dataset
datasets = [build_dataset(cfg.data.train)]

# Build the detector
model = build_detector(cfg.model)
# Add an attribute for visualization convenience
model.CLASSES = datasets[0].CLASSES

# Create work_dir
mmcv.mkdir_or_exist(osp.abspath(cfg.work_dir))
train_detector(model, datasets, cfg, distributed=False, validate=True)
```

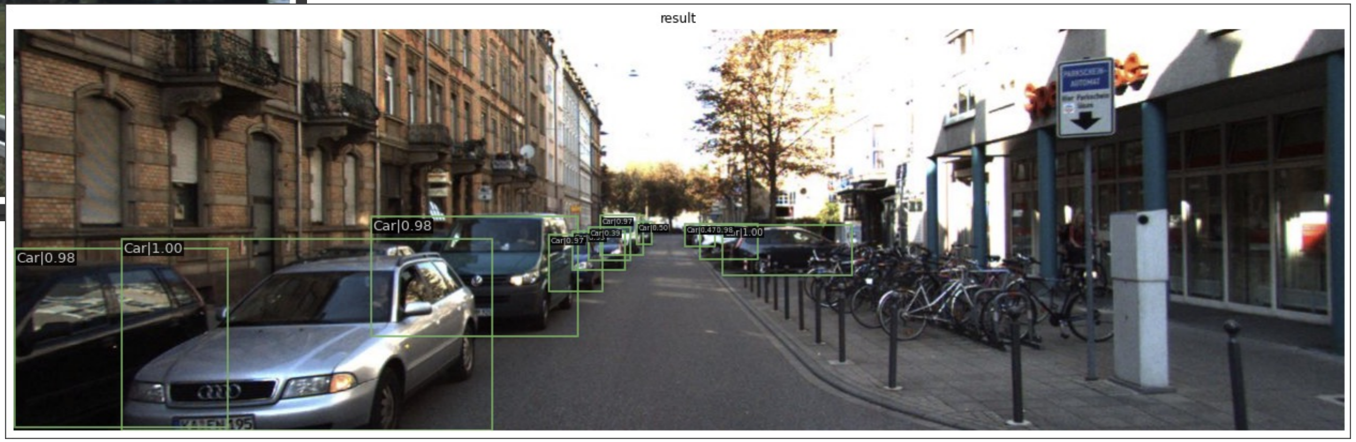
# train_detector : MMDtection에 의해 구현된 높은 수준의 API

# Train a new detector



# Train a new detector

```
img = mmcv.imread('kitti_tiny/training/image_2/000068.jpeg')

model.cfg = cfg
result = inference_detector(model, img)
show_result_pyplot(model, img, result)
```

# MMDetection

- https://colab.research.google.com/drive/17oj82eF3HI9DTNmip61v_mlJKPFn5AvZ