Object Detection with YOLO v2 – 2016310703 남득윤

1. YOLOv2

1.1 Architecture and Feature Map Explained

저는 YOLOv2를 이용하여 과제를 수행하였습니다. YOLOv2(YOLO9000: Better, Faster, Stronger)는 CVPR'17에 발표된 CNN architecture로 전작 YOLOv1을 다방면으로 개선한 구조입니다.

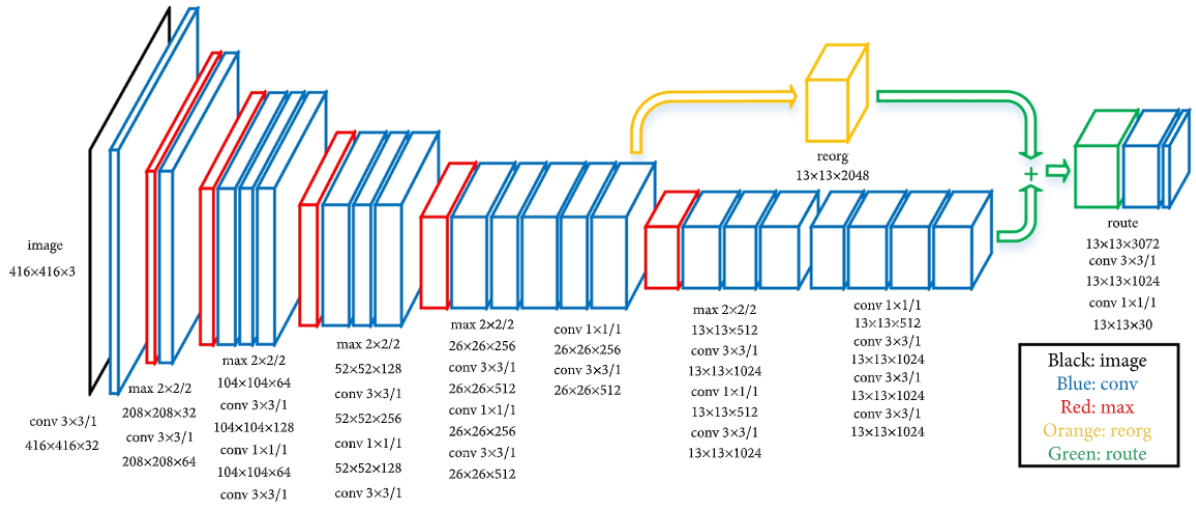

그림 1 YOLO v2 Architecture

(그림의 최종 convolution의 결과인 13x13x30은 Pascal VOC를 활용한 이번과제의 경우 13x13x125입니다.)

YOLOv2는 총 세계의 Stage로 이루어져 있습니다. Input Image로 높은 해상도의 feature를 extract하여 residual을 생성하는 Stage1, residual을 추가로 처리하여 낮은 해상도의 feature까지 extract하는 Stage2, Rescale된 residual (Pass Through)과 Stage2의 feature map을 concatenate하여 최종 grid cell별 Anchor box와 Class에 따른 예측 정보를 담은 feature map을 생성하는 Stage3로 구성됩니다.

이때 최종 output 13x13x125는 다음과 같은 의미를 가집니다.

. 13x13 : grid cell

. 125=5 * (5+20): 각 grid cell의 5개의 anchor box별 class 예측 정보

Grid cell이란 말 그대로 격자를 의미하며 416x416x3의 image를 13x13개의 32x32x3의 grid로 나누어 grid별 정보를 extract하겠다는 의미입니다.

Anchor box란 grid cell별로 예측할 object의 shape으로 미리 상정하고 시작하는 ratio입니다. 앵커 박스는 적당히 직관적인 크기의 박스로 결정하고, 비율을 1:2, 1:1, 2:1로 설정하는 것이 일반적이었습니다만, yolo v2는 여기에 learning algorithm을 적용합니다. 바로 coco 데이터 셋의 바운딩 박스에 K-means clustering을 적용하였습니다. 그 결과 앵커 박스를 5개로 설정하는 것이 precision과 recall 측면에서 좋은 결과를 낸다고 결론짓습니다. 일반적으로 가장 많이 존재하는 bounding box의 shape을 learning algorithm으로 구하였다는 점에서 mAP상승에 기여하였다고 합니다.

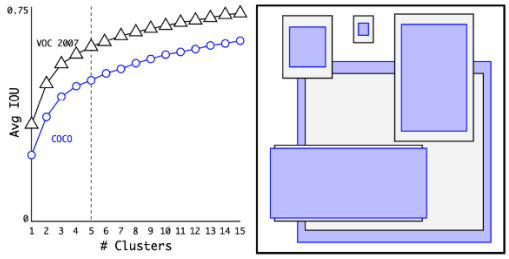

그림 2 Dimension Cluster

```
anchors=[(1.3221, 1.73145), (3.19275, 4.00944), (5.05587, 8.09892), (9.47112, 4.84053),
         (11.2364, 10.0071)]):
```

코드 1 utils_2016310703.py Ln165

이제 grid cell별로 존재하는 125길이의 vector의 의미를 분석해 보겠습니다.

YOLOv2는 5개의 Anchor Box를 가지므로 Anchor Box당 25길이의 vector를 담당합니다. 25길이의 vector는 5 + 20으로 나뉘며 앞의 5는 bounding box의 위치정보(t_x, t_y, t_w, tV)와 object가 존재할 확률(t_o)이며 뒤의 20은 class별 confidence를 의미합니다.

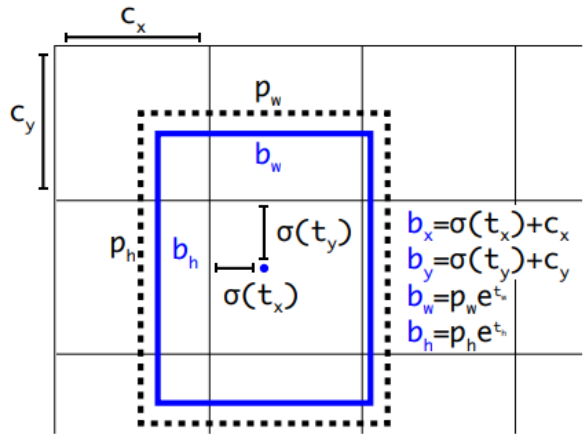

그림 3 Bounding boxes with location prediction

이 정보를 통해 실재 box의 위치와 Confidence를 구하는 과정은 아래와 같습니다.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

그림 4 Location prediction

이때 cx, cy는 좌상단을 0,0으로 하였을 때 가지게 되는 grid cell의 좌표를 의미합니다. Pw, ph는 미리 정해진 anchor box의 width, height를 의미합니다.

이를 통해 loss를 계산하면 정확한 coordinate를 학습해 나가게 됩니다.

1.2. Other Features

Batch Normalization 의 적용

기존 모델(YOLOv1)에서 Dropout Layer 를 제거하고 Batch Normalization 을 추가합니다. 이를 통해 mAP 가 2% 증가합니다

```
self.stage1_conv1 = nn.Sequential(nn.Conv2d(3, 32, 3, 1, 1, bias=False), nn.BatchNorm2d(32),
                                  nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
self.stage1_conv2 = nn.Sequential(nn.Conv2d(32, 64, 3, 1, 1, bias=False), nn.BatchNorm2d(64),
                                  nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
self.stage1_conv3 = nn.Sequential(nn.Conv2d(64, 128, 3, 1, 1, bias=False), nn.BatchNorm2d(128),
                                  nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv4 = nn.Sequential(nn.Conv2d(128, 64, 1, 1, 0, bias=False), nn.BatchNorm2d(64),
                                  nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv5 = nn.Sequential(nn.Conv2d(64, 128, 3, 1, 1, bias=False), nn.BatchNorm2d(128),
                                  nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
self.stage1_conv6 = nn.Sequential(nn.Conv2d(128, 256, 3, 1, 1, bias=False), nn.BatchNorm2d(256),
                                  nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv7 = nn.Sequential(nn.Conv2d(256, 128, 1, 1, 0, bias=False), nn.BatchNorm2d(128),
                                  nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv8 = nn.Sequential(nn.Conv2d(128, 256, 3, 1, 1, bias=False), nn.BatchNorm2d(256),
                                  nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
self.stage1_conv9 = nn.Sequential(nn.Conv2d(256, 512, 3, 1, 1, bias=False), nn.BatchNorm2d(512),
                                  nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv10 = nn.Sequential(nn.Conv2d(512, 256, 1, 1, 0, bias=False), nn.BatchNorm2d(256),
                                   nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv11 = nn.Sequential(nn.Conv2d(256, 512, 3, 1, 1, bias=False), nn.BatchNorm2d(512),
                                   nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv12 = nn.Sequential(nn.Conv2d(512, 256, 1, 1, 0, bias=False), nn.BatchNorm2d(256),
                                   nn.LeakyReLU(0.1, inplace=True))
self.stage1_conv13 = nn.Sequential(nn.Conv2d(256, 512, 3, 1, 1, bias=False), nn.BatchNorm2d(512),
                                   nn.LeakyReLU(0.1, inplace=True))
```

코드 2 Batch Normalization

Fine-Grained Features - Pass Through

네트워크의 중간에 생성된 feature map을 보관했다가 pass하여 이후에 생성된 feature map과 결합하는 시도는 Resnet에서 처음 도입(Skip connection)되었고 여러 Network를 거친 feature map을 Concatenate하여 활용하는 방식은 GoogLenet(Inception module)에서 활용되었습니다.

YOLOv2의 Pass Through는 둘을 동시에 활용함으로써 해상도에 따른 검출 문제를 해결하고자 하였습니다.

1.3 YOLO loss – How it learns?

YOLOv2의 loss function은 다음의 세 가지 loss의 합으로 구성됩니다.

a. classification loss

b. localization loss

c. confidence loss

각 grid cell별(13x13) anchor box (5)마다 존재하는 5+20길이의 feature vector에 따라 loss를 학습해야 합니다. Classification loss는 20개의 class별 probability를 localization loss는 bounding box의 좌표정보(t_x, t_y, t_w, t_h)를 confidence loss는 object의 존재에 대한 confidence(t_o)를 바탕으로 학습합니다.

a. classification loss

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

where

$\mathbb{1}_i^{obj} = 1$ if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

그림 5 Classification Loss

P^i(c)는 model을 통해 예측한 class별 probability 즉 output feature map 13*13*125(=5*(5+20))의 20을 의미합니다. 이를 실제 target의 class 별 probability (pi(c))와 squared error를 적용합니다. 객체가 탐지되지 않았다면 0이

됩니다. 논문과 위 식에는 없지만 class_scale을 parameter로 정의하여 loss에 가중치를 줄 수 있도록 하였습니다. (default = 1)

b. localization loss (coordinate loss)

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

where

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} increase the weight for the loss in the boundary box coordinates.

그림 6 localization loss

localization loss는 예측된 boundary box의 위치와 크기에 대한 에러를 측정합니다. 마찬가지로, 객체가 탐지되지 않은 경우에 대해서는 loss 값은 0입니다. 수식을 살펴보면, 위치는 sum squared error를 그대로 적용하지만 크기에 대해서는 각 높이와 너비에 대해 루트 값을 씌워 계산했습니다. 이렇게 하는 이유는 절대 수치로 계산을 하게 되면 큰 box의 오차가 작은 box의 오차보다 훨씬 큰 가중치를 받게 된다. 예를 들어, 큰 box에서 4 픽셀 에러는 너비가 2 픽셀인 작은 box의 경우와 동일하게 됩니다. 따라서 YOLO는 bounding box 높이와 너비의 제곱근을 예측하게 됩니다. 추가적으로, 더 높은 정확도를 위해 λcoord (default: 5)를 loss에 곱해 가중치를 더 줍니다

c. Confidence loss

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2$$

where

$\hat{C}_i$ is the box confidence score of the box j in cell i.

$\mathbb{1}_{ij}^{obj}=1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

그림 7 Confidence Loss object

$$\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2$$

where

$\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$.

$\hat{C}_i$ is the box confidence score of the box j in cell i.

λ_{noobj} weights down the loss when detecting background.

그림 8 Confidence Loss no object

둘의 다른 점은 객체가 탐지되지 않은 경우는 λnoobj (default: 0.5)에 의해 loss의 가중을 적게 한다는 점입니다. 이유는 클래스 불균형 문제를 방지하기 위함이다. 사실 대부분의 box가 객체를 포함하고 있지 않은 경우가 더 많기 때문에 배경에 대한 가중을 줄이지 않는 경우 배경을 탐지하는 모델로 훈련될 수 있습니다. Classification과 마찬가지로 λobj (default: 1.0)으로 parameter를 추가하여 train시 argument로 받을 수 있도록 하였습니다.

Final Loss

$$\begin{aligned}
&\lambda_{\text{coord}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]\\
&+\lambda_{\text{coord}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]\\
&+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2\\
&+\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2\\
&+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{\text{obj}}\sum_{c\in\text{classes}}\left(p_i(c)-\hat{p}_i(c)\right)^2
\end{aligned}$$

그림 9 Final Loss

최종 loss 는 위의 모든 loss 를 더한 것입니다.

```
Epoch: 1/160, Iteration: 1/1370, Lr: 1e-05, Loss:123.03 (Coord:6.28 Conf:102.79 Cls:13.96)
Epoch: 1/160, Iteration: 2/1370, Lr: 1e-05, Loss:120.88 (Coord:6.75 Conf:100.95 Cls:13.18)
Epoch: 1/160, Iteration: 3/1370, Lr: 1e-05, Loss:119.35 (Coord:7.37 Conf:96.60 Cls:15.39)
Epoch: 1/160, Iteration: 4/1370, Lr: 1e-05, Loss:108.99 (Coord:5.67 Conf:91.15 Cls:12.17)
Epoch: 1/160, Iteration: 5/1370, Lr: 1e-05, Loss:96.82 (Coord:3.57 Conf:84.59 Cls:8.66)
```

코드 3 train loss 출력

이때 그림 9 에서 hat 이 붙지 않은 ground truth 값들은 그림 4 의 step 을 inverse 로 진행함으로써 얻을 수 있다. 이때 bounding box 와 shape 이 가장 유사한 anchor box 에 대해서만 진행합니다. (유사한 정도는 IOU 로 결정) Class 의 probability 는 존재하는 class 에 대해서만 1 로 set 하고 나머지는 0 으로 둡니다. 이렇게 yolo txt format(: class index, x, y , w, h)로 작성된 ground truth 를 학습에 필요한 target 으로 변환하는 과정이 추가적으로 필요합니다. 이는 utils_2016310703.py 의 class build_targets( )에 구현 되어 있습니다.

```
# Get target values
coord_mask, conf_mask, cls_mask, tcoord, tconf, tcls = self.build_targets(pred_boxes, target, height, width)
```

코드 4 apply build_target/ utils_201631073.py Ln65

```
# Compute losses
mse = nn.MSELoss(size_average=False)
ce = nn.CrossEntropyLoss(size_average=False)
self.loss_coord = self.coord_scale * mse(coord * coord_mask, tcoord * coord_mask) / batch_size
self.loss_conf = mse(conf * conf_mask, tconf * conf_mask) / batch_size
self.loss_cls = self.class_scale * 2 * ce(cls, tcls) / batch_size
self.loss_tot = self.loss_coord + self.loss_conf + self.loss_cls

return self.loss_tot, self.loss_coord, self.loss_conf, self.loss_cls
```

코드 5 YOLOLOSS/ apply build_target/ utils_201631073.py Ln77

구현 시 classification loss는 코드를 보면 논문의 SSE(sum squared error)와 다르게 cross entropy loss로 구현되어 있는데 이는 YOLOv3이후 도입된 방식이지만 YOLOv2에서도 많은 경우 구현단계에서 활용하는데 무리가 없기에 사용되었습니다.

1.4 Post Process Feature Map

최종 output인 13x13x125 size의 feature map으로 어떻게 bounding box를

predict하는가에 관한 내용입니다. 이 부분은 YOLOv2 혹은 YOLO를 넘어 대부분의 CNN기반 object detection algorithm에 적용되는 process입니다. Grid cell 별로 가장 t_o (Object가 있을 confident)가 높은 anchor를 찾아 class probability가 가장 높은 class를 찾습니다. 둘을 곱한 값을 최종 class에 대한 confidence value로 정하여 이 값이 user가 정한 conf_threshold 값 (default: 0.5)보다 높다면 해당 box를 select합니다.

이때 인접한 grid cell이 같은 object에대한 box를 여러 번 predict하여 False Positive가 많이 발생할 수 있습니다. 이를 방지하기위해 인접한 box에대해 같은 class label을 가지고 있다면 non max suppression을 적용하여 중복을 제거합니다. No max suppression은 IOU를 계산하여 user가 정한 nms_threshold 값 (default:0.35)보다 높다면 해당 box중 가장 confident score가 높은 box만 남기고 나머지를 제거하는 algorithm입니다. Nms를 포함한 post processing은 utils_2016310703.py의 post_processing( )에 구현 되어 있습니다.

2. Data Preprocessing

Data preprocessing은 train/ test dataset에서 __getitem__으로 data를 load할때 즉각적으로 적용되도록 하였습니다. training에는 HSVAdjst( ), VerticalFlip(), Crop(), Resize( )를 test시에는 Resize( )를 적용하였습니다.

```
if self.is_training:
    transformations = Compose([HSVAdjust(), VerticalFlip(), Crop(), Resize(self.image_size)])
else:
    transformations = Compose([Resize(self.image_size)])
image, objects = transformations((image, objects))
```

코드 6 Image Preprocessing/ utils_2016310703.py Ln419

또한 annotation도 voc xml format에서 활용하기 좋도록 list형태로 뽑아 활용하도록 전처리 합니다.

```
objects = []
for obj in annot.findall('object'):
    xmin, xmax, ymin, ymax = [int(float(obj.find('bndbox').find(tag).text))  - 1 for tag in
                              ["xmin", "xmax", "ymin", "ymax"]]
    label = self.classes.index(obj.find('name').text.lower().strip())
    objects.append([xmin, ymin, xmax, ymax, label])
```

코드 7 Annotation Preprocessing/ utils_2016310703.py Ln415

3. mAP

평가 metric 으로는 가장 널리 활용되는 mAP(mean average precision)을 활용하였습니다. Class 별 Precision/Recall Curve(blue line)의 monotonically decreasing part(red line)아래의 면적 (light blue)을 Average Precision 으로 정의하고 이를 class label 별로 mean 을 적용한 것입니다. GitHub repo(github.com/Cartucho/mAP)를 활용하였습니다.

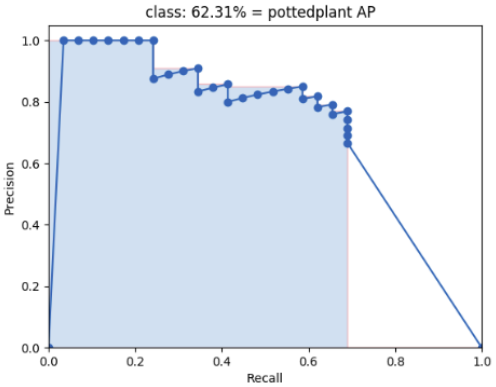

그림 10 Precision/Recall Curve Example

Test 시 option 을 통해 mAP 처리 유무를 argument 로 받아 처리합니다.

```
if(opt.mAP):
    voc_xml_to_yolo_txt(opt.input +'/test_annotations/voc_xml', opt.input +'/test_annotations/yolo_text/')
    mAP(opt.input +'/test_annotations/yolo_text',  opt.output + '/yolo_text')
```

코드 8 test_2016310703.py Ln91

4. Results

4.1 environments

GPU – RTX 2080

CUDA – 11.2

Python – 3.6.9

Pytorch – 1.8.1

Test set – VOC 2007 test dataset

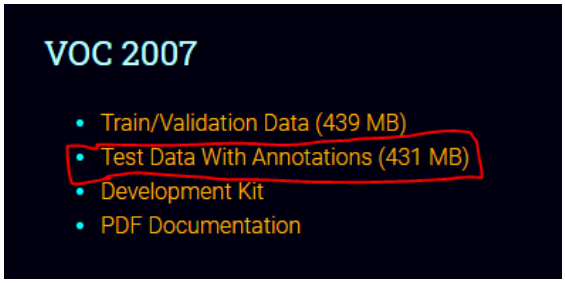

그림 11 test set source https://pjreddie.com/projects/pascal-voc-dataset-mirror/

Ground Truth

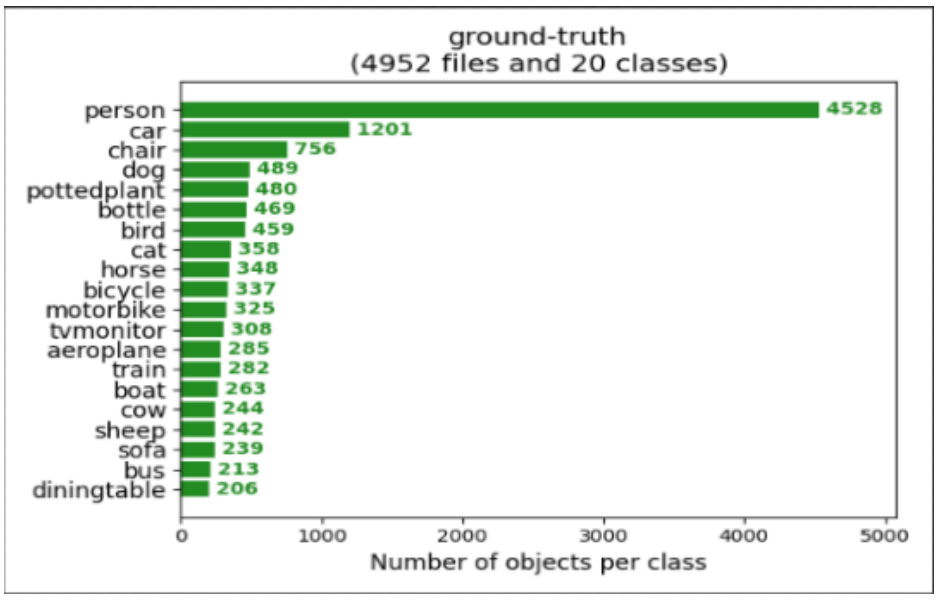

그림 12 output/ground-truth-info.png

4.2 실험 1 (default)

학습시간 - 11시간 30분

mAP - 0.69... (69%아닙니다...)

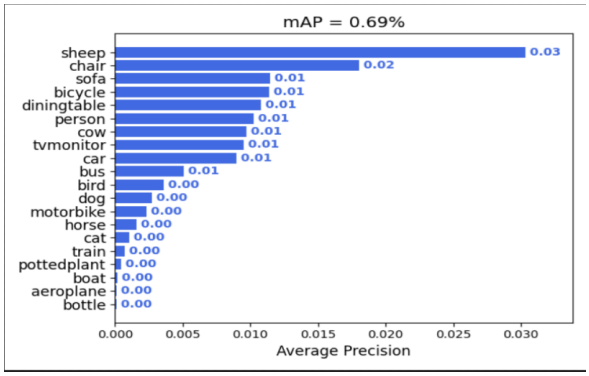

그림 14 실험1 output/mAP.png

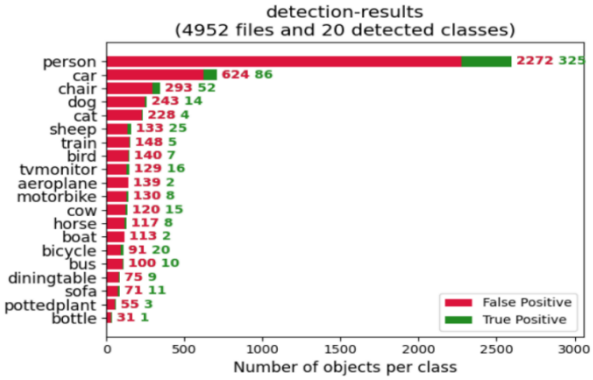

그림 13 실험1 output/detection-results-info.png

모든 hyperparameters를 default로 두고 실험하였습니다. 성능이 안 좋습니다.

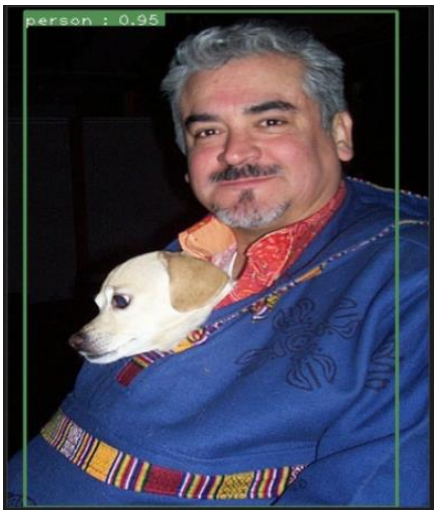

그림 15 실험1 test_out/images/000001~3_prediction.jpg

미검출 된 object가 많은 것을 확인할 수 있습니다.

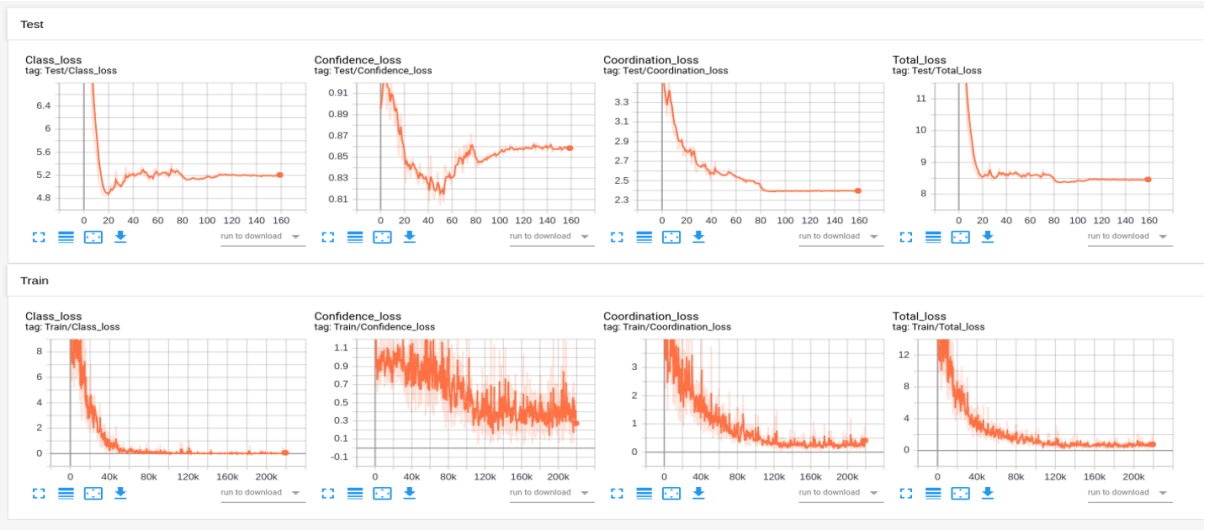

그림 16 실험1 validation, train loss / tensorboard를 설치하고 terminal에

tensorboard –logdir=/path/to/dir of events.out.tfevents.*.*

를 입력하여 생성되는 host에 browser를 통해 접속하여 확인할 수 있습니다.

이를 참고한 github repo(github.com/uvipen/Yolo-v2-pytorch)와 비교하여 보니 학습의 경향성은 비슷하지만 training step이 두배 이상으로 많아 overfitting이 의심되는 상황입니다.

4.3 실험 2 (train/val – 6:4)

train과정에 너무 많은 data가 활용되어 과적합이 발생하는 것 같아 Val_ratio를 0.4로 set하여 진행하였습니다.

학습시간 - 8시간 40분

mAP – 0.61

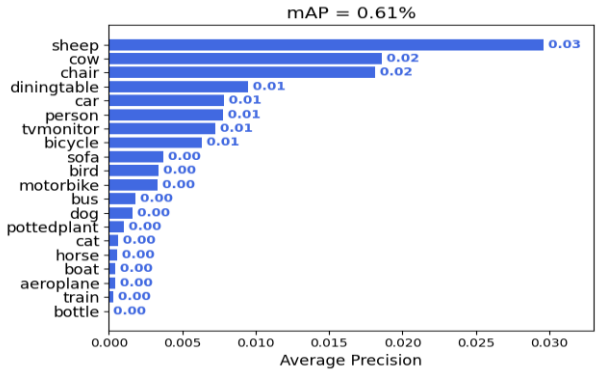

그림 18 실험2 output/mAP.png

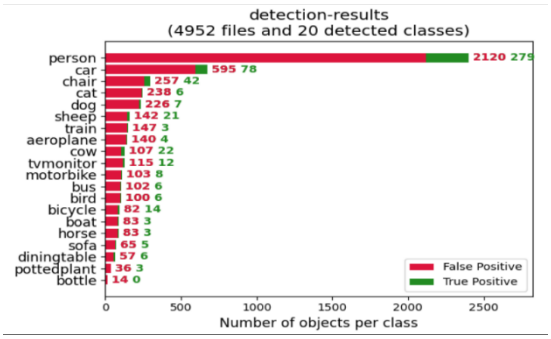

그림 17 실험2 output/detection-results-info.png

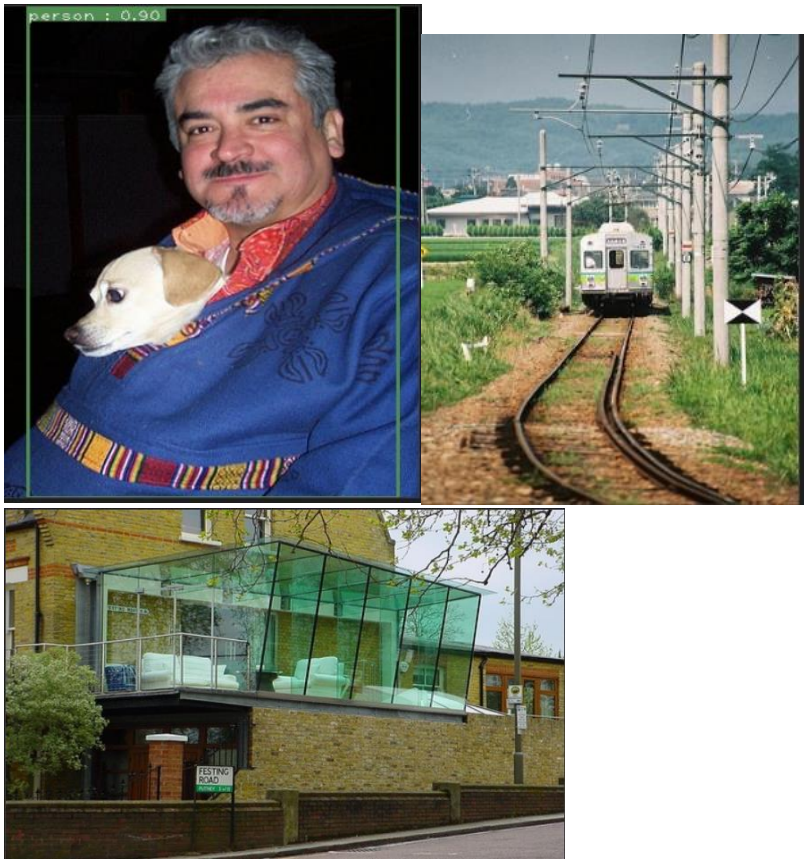

그림 19 실험2 test_out/images/000001~3_prediction.jpg

성능이 오히려 살짝 떨어졌습니다.

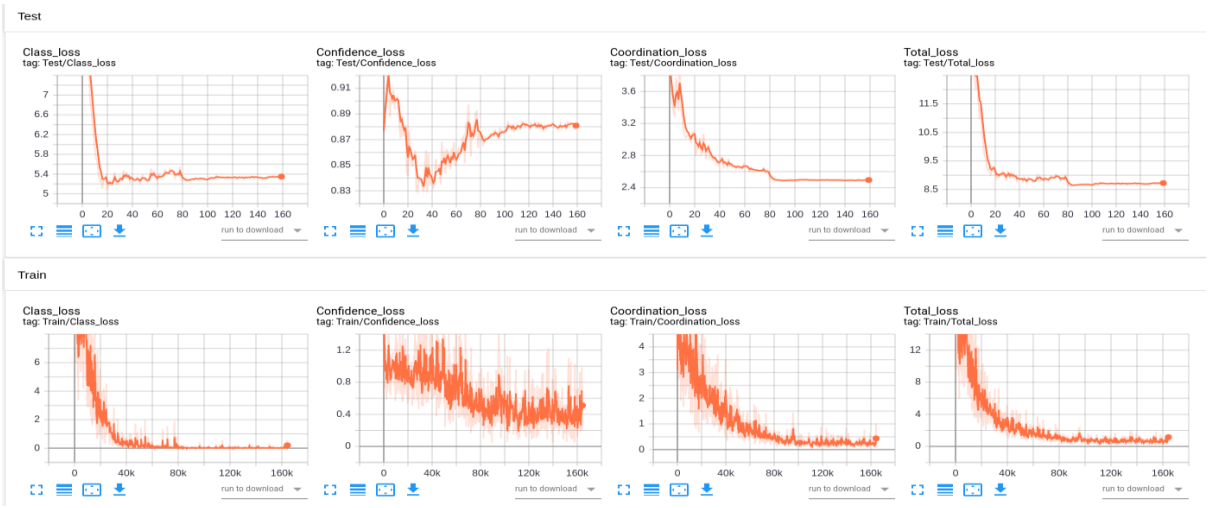

그림 20 실험2 validation, train loss

실험 1의 모델로 최종 train test 진행 후 최종 directory 구조를 제출하겠습니다.

Train, test directory의 경우 혹시 경로가 헷갈리실 까봐 image, annotation 7pair만 남겨두었습니다.