YOLOv1 결과보고서

You Only Look Once
Unified, Real-Time Object Detection

Joseph Redmon, Santosh Divvalay, Ross Girshick, Ali Farhadiy
University of Washington, Allen Institute for AIy, Facebook AI Research

1. 논문 요약

가. 개요

1) YOLOv1에서 저자는 DPM 혹은 R-CNN 등 기존 모델과 비교하여 구조적, 개념적 변화와 더불어 성능에서 역시 많은 향상을 이뤄냈다고 한다. 기존의 object detection task를 bounding box와 class에 대한 확률의 regression 문제로 재정의 하였다.

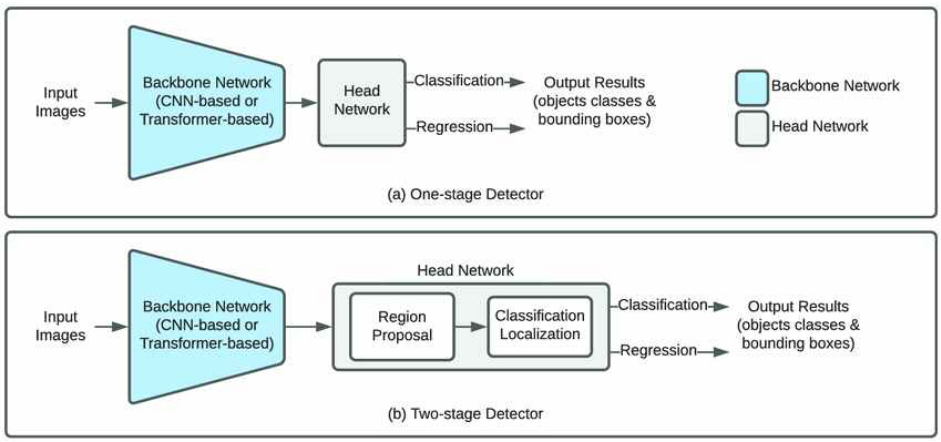

그림 1 1stage 및 2stage detection 모델 구조(이미지 출처)

2) 기존 2stage 모델에서는 region proposal과 classification이 별도의 모델에서 순차적으로 이루어졌다. 이를 1stage 모델로 변경하여 하나의 모델에서 classfication과 localization이 한 번에 이루어지는 end to end 학습이 가능해졌다.

3) YOLO는 real time 45 frame/sec의 처리 속도를 가지며 Fast YOLO는 155 frame/sec의 성능을 가지는 동시에 다른 디텍터의 두배에 해당하는 mAP를 가질 정도로 정확한 성능을 보여준다. 하지만 localization 문제에 있어서는 개선이 필요하다.

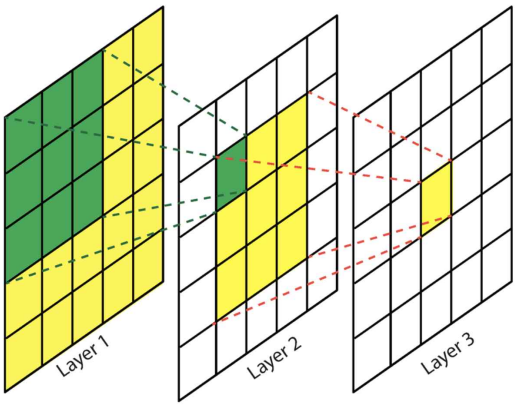

그림 2 CNN layer에서의 receptive filed

4) sliding window나 region proposal을 사용하는 기존 모델에서는 실제 객체가 존재하는 영역만 떼어서 보기 때문에 주변 배경을 인식하는 문맥적 정보가 떨어졌다. CNN 기반의 YOLOv1에서는 feature를 기반으로 bbox와 class를 추론하는 과정에서 객체 주변 픽셀들을 전부 고려할 수 있기 때문에 background error가 크게 향상되었으며 이는 Fast R-CNN의 절반 수준이다.

5) YOLOv1은 객체들의 일반화된 feature들을 학습하기 때문에 새로운 도메인이나 처음보는 이미지에 대해서도 성능이 강건한 편이다.

나. 모델 구성

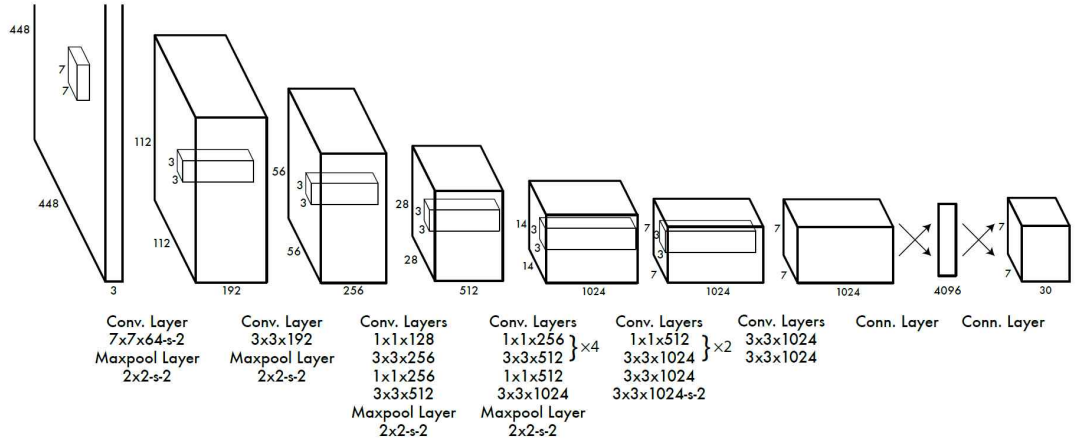

그림 3 YOLOv1 모델 구조

YOLOv1은 [3, 448, 448] 이미지를 받아 [7, 7, 30]의 텐서를 출력하는 모델이다. 전체적인 뼈대는 convolution layer의 결합이며 적절한 파라미터 수를 위해 3x3과 1x1 convolution을 사용하였다. max pool layer를 통해 5번의 down sampling을 하며 마지막에는 2개의 fully connected layer가 배치되며 두 레이어 사이에는 dropout이 배치된다. 활성화 함수로는 기울기 0.1의 leaky relu를 사용하였다.

다. 손실 함수

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

그림 4 YOLOv1 loss function

YOLOv1의 손실 함수는 bbox의 coordinate에 대한 loss, confidence scor에 대한 loss, class에 대한 loss 총 세 부분으로 구성되며 SSE(Sum Squared Error)로 계산한다. YOLOv1 학습에는 여러 가지 데이터 불균형이 존재한다. 단순히 데이터셋의 클래스, 객체의 크기, 객체의 형태 등을 넘어 grid cell 간의 불균형까지 고려해야 한다. 학습에 사용하는 Pascal VOC의 경우 이미지에 분포하는 객체들이 전반적으로 크고 이미지 하나 당 객체 수가 적기 때문에 49개 grid cell에서 실제 객체가 존재하는 cell은 보통 5개를 넘지 않는다. loss는 49개 grid cell의 loss들의 총합으로 구성된다. 데이터가 존재하지 않는 cell의 타겟은 0이며 이를 학습하는 cell 들의 loss가 실제 데이터가 존재하는 소수 셀의 bbox

coordinate, confidence score, class probability에 의한 loss를 압도하기 때문에 몇몇 계수를 도입한다.

위 수식의 첫 번째 두 번째 행은 bbox coordinate에 대한 regression loss이다. 수식을 뜯어보면 λ_{coord}는 앞의 grid cell간 불균형 해소를 위한 계수로 YOLOv1의 localization 취약점을 고려하여 5로 설정한다. xy 좌표에 대해서는 직접적인 차이를 loss에 반영하며 wh 좌표에 대해서는 루트를 적용하여 bbox가 큰 객체에 대한 loss가 감소하도록 하였다. i는 49개 grid cell의 인덱스를 나타내며 j는 2개의 bbox를 나타낸다. 학습 시 타겟은 grid cell 당 하나로 총 49개가 배치되는데 모델의 출력은 98개이다. 따라서 1_{ij}^{obj} 마스크를 통해 각 grid cell당 2개의 bbox 중 타겟과 iou가 높은 bbox의 j 인덱스를 선택해 타겟과 1:1 대응을 시켜준다.

수식의 세 번째 네 번째 행은 confidence score에 대한 loss이다. λ_{noobj}를 통해 loss의 가중치를 조절하며 계수는 0.5이다. 1_{ij}^{obj}는 위와 마찬가지로 높은 iou를 가지는 bbox의 인덱스를 나타낸다. 하지만 1_{ij}^{noobj}는 1_{ij}^{obj}의 반대인 낮은 iou를 갖는 bbox를 나타내는 인덱스가 아니니 착각하지 말자. 1_{ij}^{noobj}는 객체가 존재하지 않는 셀을 표시하는 마스크이다.

수식의 마지막 행은 classification에 대한 loss이다. 여기서도 마찬가지로 1_{ij}^{obj}는 위의 사용 용도와는 다르게 두 bbox 중 하나를 선택하기 위한 마스크가 아닌 실제 객체가 존재하는 cell에 대한 마스크이다. class가 20개이고 44개의 cell이 빈 셀이라고 가정하면 20*44 가 0으로 학습되고 나머지 5개 셀에 대해 5*1은 1로 5*19는 0으로 학습된다. 1_{ij}^{obj} 마스크로 0으로 학습되는 셀의 비율을 크게 줄일 수 있다.

사실 coordinate loss와 confidence loss의 1_{ij}^{obj}도 객체가 존재하는 cell에 대한 마스크 개념을 포함하는 듯 하다. 논문상에 다음과 같이 표기되어 있는데 여기서 responsible 하다는 것이 객체가 존재하는 cell 한정이라는 의미로 해석된다.

It also only penalizes bounding box coordinate error if that predictor is “responsible” for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell)

라. 학습 설정

논문 저자는 학습 시에 아래의 설정을 적용하였다.

- dataset : Pascal VOC
- epoch : 135
- batch size : 64
- optimizer : SGD
- momentum : 0.9
- decay : 0.0005
- learning rate : 1 epoch에서는 발산 방지를 위해 1e-3에서 1e-2로 점진적 증가, 75 epoch까지 1e-2, 105 epoch까지 1e-3, 135 epoch까지 1e-4
- dropout : 첫 번째와 두 번째 fully connected layer 사이에 0.5 설정
- augmentation : 이미지 사이즈 및 색상 변경

마. 한계점

1) bbox 예측에 있어 구조적 한계점이 있다. 모델 구조 상 grid cell 하나 당 2개의 box와 1개의 class 예측만 가능하기 때문에 하나의 grid cell에 여러 객체의 중심 좌표가 존재하는 경우에 대응할 수 없다. 아래에서 그림으로 예시를 들어보면

그림 5 객체 배치 예시

첫 번째 그림에서 중심 cell에서 class가 같은 작은 객체들이 존재한다. cell 당 최대 감지 개수인 2개를 넘어서기 때문에 한 개는 감지에서 누락된다.

두 번째 그림에서는 중심 cell에서 class가 다른 두 객체가 존재한다. 중심 cell의 두 개의 bbox가 서로 다른 객체를 감지하더라도 하나의 class만 감지 가능하기 때문에 한 개의 bbox는 오답이 된다.

세 번째 그림에서는 중심 cell에서 가구 class를 감지했기 때문에 두 bbox 모두 정답이 될 수 있다.

2) darknet은 feature를 down sampling하는 과정에서 1번의 stride2 convolution 그리고 5번의 max pooling을 사용한다. 따라서 마지막에 출력되는 feature는 원본 대비 1/64로 압축된다. 이는 객체의 가로 세로 폭이 64 pixel 이하이면 마지막 feature는 1 pixel이 되며 bbox coordinate, confidence, class 정보를 [1024(C), 1(H), 1(W)]의 값으로 추론하게 됨을 의미한다. 64(1 pixel), 128(2 pixel), 192(3 pixel) 등 낮은 가로 세로 폭을 갖는 객체에 대한 학습 및 추론 성능 저하가 예상된다.

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

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

그림 6 coordinate loss 수식

3) bbox가 큰 객체와 bbox가 작은 객체 사이의 손실 함수 설정 문제가 존재한다. 가로 세로 폭 300 pixel인 객체에서 15 pixel 만큼의 loss가 발생하면 그림 비율 상 5%의 오차에 해당한다. 하지만 가로 세로 폭 30 pixel인 객체에서 15 pixel 만큼의 loss가 발생하면 그림 비율 상 50%의 오차에 해당한다. 손실 함수에서는 coordinate error의 절대값이 적용되는 반면 IoU에서는 타겟과 추론 사이의 error 비율로 적용되기 때문에 문제가 발생한다. YOLOv1에서는 루트 함수를 적용하여 손실함수에서 큰 객체와 작은 객체 사이의 error 격차를 줄이려고 한 것으로 보이나 완벽한 해결법은 아닌 것으로 보인다.

4) 종합적으로 보았을 때 논문 저자는 localization error를 논문의 취약점으로 판단하였다.

2. 모델 구현

모델의 구현은 다음 주소의 ppt 자료를 이용하여 설명하도록 하겠다.
https://docs.google.com/presentation/d/1aeRvtKG21KHdD5lg6Hgyhx5rPq_ZOsGjG5rJ1HP7BbA/pub?start=false&loop=false&delayms=3000&slide=id.p

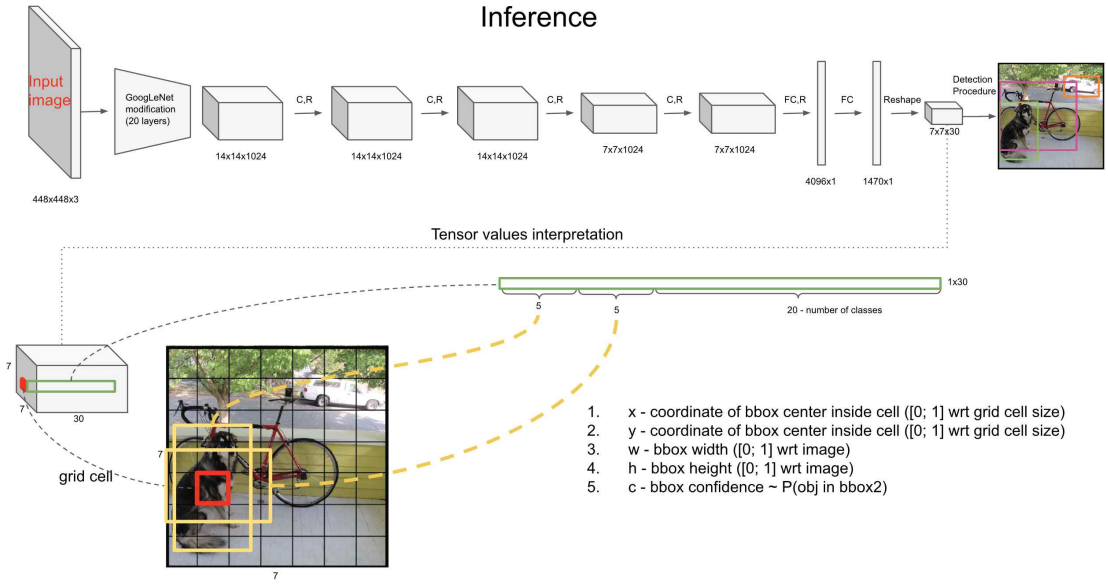

그림 7 YOLOv1 출력 데이터 구조

YOLOv1은 이미지를 [batch size, 7, 7, 30]의 형태로 변환한다. 7x7은 가로 세로 grid cell을 나타내며 하나의 cell에는 2개의 bbox(x, y, w, h, c) 그리고 20개의 클래스 확률이 포함된다. 이것이 YOLOv1의 데이터 형태이다.

- 데이터셋 이미지로부터 target을 생성한다. 기본적으로 객체의 중심이 존재하는 grid cell에 데이터가 할당되며 할당되지 않은 좌표의 모든 데이터는 0이다. 만약 객체가 인접하여 하나의 cell에 여러 객체가 존재할 경우 선착순으로 1개만 할당되고 나머지는 누락시킨다.
객체의 bbox 좌표를 i, j, x, y, w, h, c로 변경할건데 i, j는 grid cell의 좌표 인덱스로 0~6의 범위를 갖는다. x, y는 해당 cell에 할당된 bbox의 중심좌표에 해당하며 0~1의 범위를 갖는다. w, h는 bbox의 가로 세로 폭에 해당하며 0~7의 범위를 갖는다. c는 confidence score로 객체 할당된 셀에는 객체가 100% 존재하기 때문에 1로 설정한다.
- 모델의 출력은 [batch size, 7, 7, 30]이고 [x1, y1, w1, h1, c1, x2, y2, w2, h2, c2, class....]의 형태는 target의 형태와 동일하다. loss를 계산 시 이 출력 값과 target 값을 비교하며 모델을 학습시킨다.
- IoU, NMS, 시각화 등의 작업을 수행할 때는 target 형태의 좌표가 아닌 실제 좌표가 필요하기 때문에 1번의 과정을 역산하여 원래 좌표로 돌리는 과정을 수행 후 IoU, NMS, 시각화 등을 수행한다.

학습용 코드는 아래의 github와 영상을 참고하였다.
https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/object_detection/YOLO
https://www.youtube.com/watch?v=n9_XyCGr-MI&t=2526s

3. 결과 분석

결과 분석에 앞서 여러 조건으로 수행한 실험 결과에 의해 갖게 된 몇몇 관점을 먼저 기술해 보겠다.

가. obj mask

1) 논문에서 명시한 responsible의 범위가 confidence loss와 class loss에 대해서만 적용하는 것인지 coordinate loss에도 적용하는 것인지 명확하지 않았다. 그리고 CNN 구조에서의 receptive field 효과에 의해 최종 출력 노드를 학습시키지 않으면 앞단의 신경망도 광범위하게 학습되지 않는 문제를 고려하였다(물론 classifier를 global average pool을 사용하지 않고 fc layer를 사용하였기 때문에 YOLOv1에서는 해당 사항 없음). 같은 관점에서 bbox 두 개 중 하나를 선택하는 방식도 최종 출력 노드를 학습시키지 않는 효과가 있다고 생각하였다. 이 세 관점을 고려하여 높은 iou score의 bbox만 학습시키는 방식/두 박스를 모두 학습시키는 방식, 세 부분의 loss에 각각 obj mask를 적용하는 방식/모두 적용하지 않는 방식을 조합하여 20가지 정도의 loss 버전에 대해 짧은 epoch으로 실험을 해보았다. 결과는 모두 실패했으며 모든 경우에서 모델이 발산하거나 입력 이미지에 상관없이 0을 출력하는 모델이 학습되었다.

나. 0을 학습하는 모델

1) Pascal VOC의 이미지는 전반적으로 객체가 큰 편이고 이미지 당 객체 수가 적은 편이다. 따라서 YOLOv1 형식의 타겟을 생성할 경우 객체가 존재하는 grid cell과 존재하지 않는 grid cell의 비율이 20:1 정도가 되는데 0을 학습하는 노드의 수가 실제 데이터를 학습하는 노드의 수를 압도하게 되며 정답을 맞춰 loss를 개선하는 것보다 그냥 0을 출력해서 loss를 개선하는 것이 효과적이게 된다.

2) mask를 씌우는 것은 모델 학습을 방해한다고 생각하고 또 모델에게 정답과 오답을 같이 학습시키는 것이 아닌 정답만을 학습시키기 때문에 mask를 씌우지 않는 방식으로 문제를 해결해보려고 하였다. 0에 의한 loss의 영향을 줄이기 위해 객체가 존재하는 grid cell의 loss를 객체가 없는 grid cell의 loss보다 10배 가량 가중치를 주어 학습시켰다. 앞에서와 마찬가지로 0을 출력하며 전혀 효과가 없었다.

3) detection task의 난이도가 너무 높은 것이 문제인가? 라고 생각하여 Imagenet으로 학습시킨 darknet을 backbone으로 사용하여 loss가 개선되지 않는 지점까지 classifier를 학습시킨 이후 backbone을 unfreeze 하여 전체를 학습시켰다. 앞에서와 마찬가지로 0을 출력하며 전혀 효과가 없었다.

다. 실험1(고화질 이미지)

1) 최종적으로 iou가 높은 bbox 1개를 선택하고 3가지 loss 모두 grid cell이 있는 부분만 학습시키는 방식을 채택했으며 모델이 학습되기 시작했다.

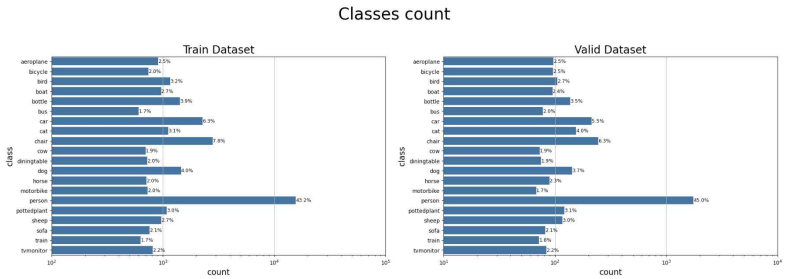

그림 8 train/valid dataset class 비율

2) 실험1에 사용한 데이터셋은 Pascal VOC에서 difficult 1인 객체를 제외한 이미지를 사용하였으며 train/valid 데이터셋 비율은 9:1로 분할하였다. 학습 초기에는 test 데이터셋을 추가하여 train/valid/test 7:1.5:1.5로 하였으나 학습 데이터가 부족하다고 생각하여 변경하게 되었다. train/valid 데이터셋의 클래스 비율은 비슷하게 분할이 되었고 person 클래스의 수가 압도적으로 많고 그 다음으로 car, chair 클래스의 수가 높게 나타났다.

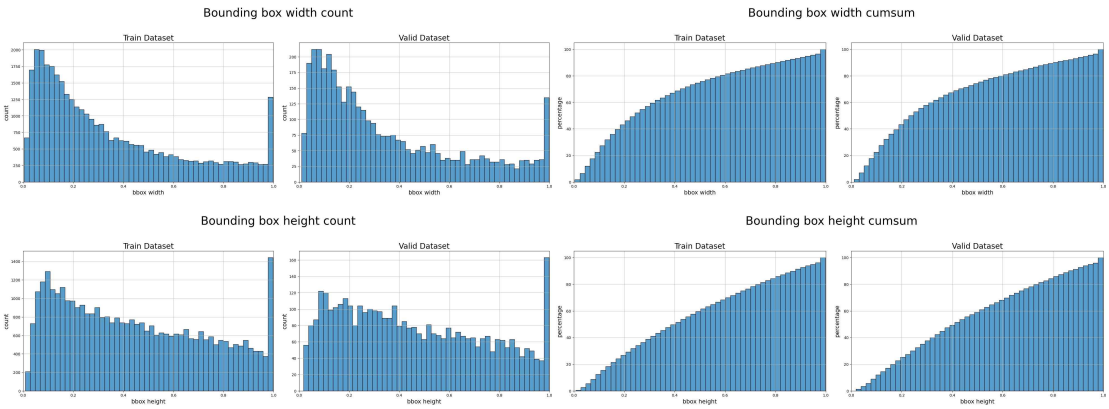

그림 9 bounding box 가로 세로 길이 분포

3) 데이터셋의 normalized width는 작은 값의 비율이 좀 높고, normalized height는 비율상 약간의 차이가 보인다. 이미지 크기와 동일한 가로 세로 길이의 객체가 많은 것으로 보인다.
4) 여기서부터는 이미지가 너무 커서 문서에 담지 못했으니 이미지 원본을 참조하길 바란다. dataset1_7 이미지를 보겠다. 이미지의 객체를 YOLOv1 형식으로 변경하였을 때 어느 grid cell에 분포하는지를 확인해보았다. 그림의 i, j는 height, width의 grid cell 좌표를 나타낸다. 객체의 개수를 보았을 때 이미지는 대부분 중심에 위치하고 있으며 이미지 모서리로 갈수록 아주 소수의 객체만 존재하는 것을 확인할 수 있다. 심지어 valid dataset은 모서리에 객체가 존재하지 않는 grid cell도 있다.
5) dataset1_8 이미지를 보겠다. 이미지의 객체를 YOLOv1 형식으로 변경하였을 때 하나의 grid cell에는 하나의 객체만 할당할 수 있기 때문에 중복 할당되는 객체는 누락되게 된다. 대부분의 객체가 중심에 위치하기 때문에 중심 부분에서 많은 수의 객체가 누락되며 중심부에 많은 인접 객체들이 있을 것을 유추할 수 있다.
6) dataset1_9 이미지를 보겠다. 그림에서 각 grid cell 별로 분포하는 클래스의 종류와 개수를 확인할 수 있다. 역시나 이미지 모서리로 갈수록 데이터 개수가 크게 줄어드는 것을 확인할 수 있다. 또한 전체 클래스의 분포와 grid cell 별 클래스 분포가 다른 것을 확인할 수 있고 valid dataset에는 있는 클래스가 train dataset에는 없는 경우도 볼 수 있다.
7) 전체적으로 보았을 때 train/valid 전체 dataset의 불균형 문제, train/valid dataset 내부의 클래스 불균형 문제 이외에도 grid cell로 분산되는 과정에서의 불균형함과 특정 cell의 학습 데이터 부족이 문제가 될 것으로 예상된다.

```
self.batch_size = 16
self.lr = 2e-5
self.lr_schedule = [
    [1, 1e-4],
    [2, 5e-5],
    [200, 2e-5],
    [1e+10, 1e-5],
]
scale = 1.1
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]

self.train_transform = A.Compose([
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2, p=0.5),
    A.Blur(p=0.1),
    A.CLAHE(p=0.1),
    A.ToGray(p=0.1),
    A.Normalize(mean=mean, std=std, max_pixel_value=255),
    A.LongestMaxSize(max_size=int(self.img_size * scale)),
    A.PadIfNeeded(
        min_height=int(self.img_size * scale),
        min_width=int(self.img_size * scale),
        border_mode=cv2.BORDER_CONSTANT,
    ),
    A.RandomCrop(width=self.img_size, height=self.img_size),
    A.HorizontalFlip(p=0.5),
    ToTensorV2(),
], bbox_params=A.BboxParams(format='yolo', min_visibility=0.4, label_fields=[]))

self.test_transform = A.Compose([
    A.Normalize(mean=mean, std=std, max_pixel_value=255),
    A.LongestMaxSize(max_size=self.img_size),
    A.PadIfNeeded(
        min_height=self.img_size,
        min_width=self.img_size,
        border_mode=cv2.BORDER_CONSTANT,
    ),
    ToTensorV2(),
], bbox_params=A.BboxParams(format='yolo', min_visibility=0.4, label_fields=[]))
```

그림 10 실험1 학습 설정

8) 실험1의 학습 설정은 그림과 같으며 100 epoch 학습 수행하였다. 문서에 기록하지 않은 여러 실험들도 있었지만 실험1의 데이터셋에서는 학습의 속도가 매우 느리고 loss나 mAP score의 개선도 낮은 수준에서 머무는 등 한계가 명확하였다.

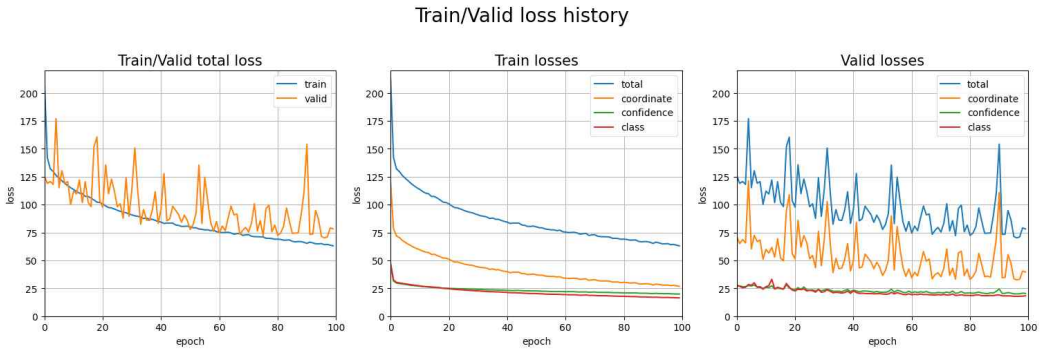

그림 11 실험1 loss history

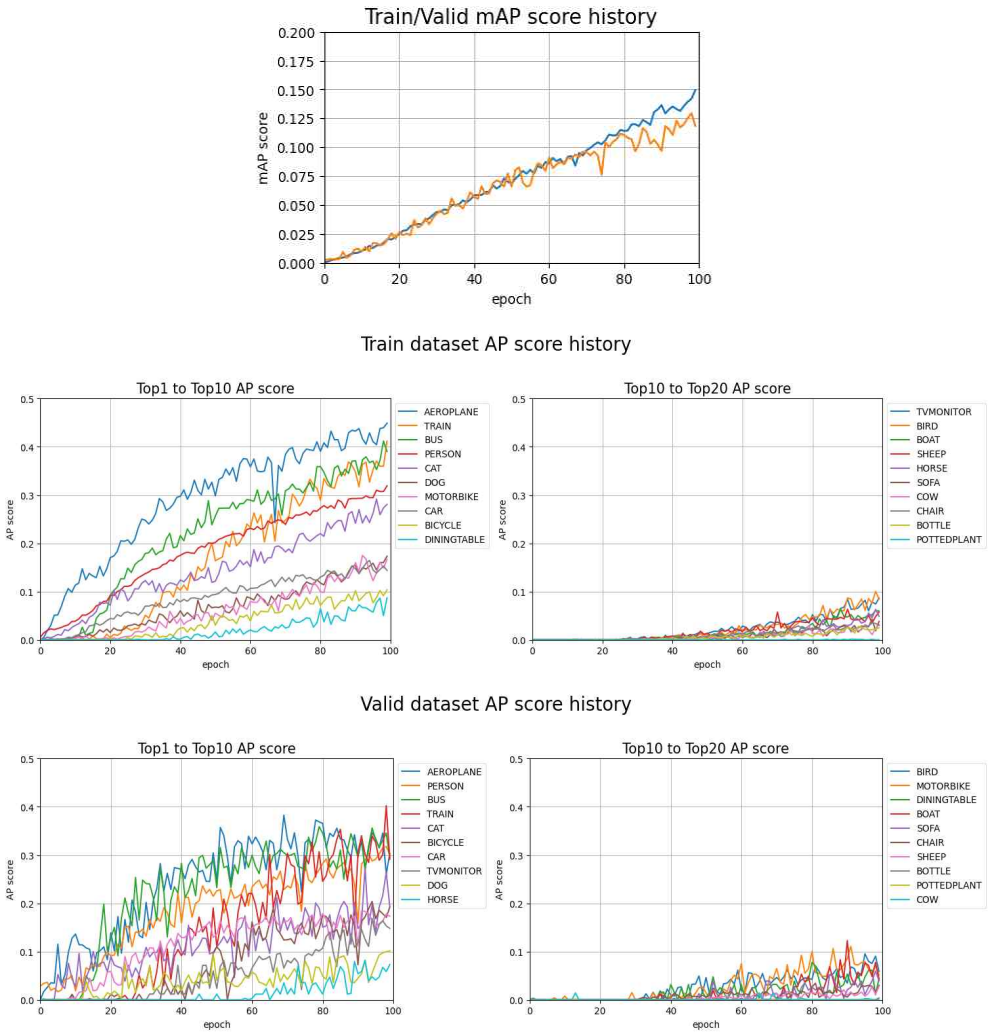

그림 12 실험1 AP score history

다. 실험2(고화질 이미지)

1) 실험1은 difficult가 1인 객체만 제외하고 모든 객체를 학습에 사용하였다. YOLOv1에서 classifier에 입력되는 feature size는 원본 대비 1/64 압축된 것을 사용하기 때문에 가로 세로 폭 64~128 pixel의 객체는 학습에 악영향을 줄 것으로 판단하였다. 이에 의한 영향을 보기 위해 실험2부터는 albumentations.LongestMaxSize 함수를 이용하여 모델 입력(448x448)로 resize했을 때를 기준으로 1.5 grid cell 이하의 객체는 제외하고 데이터셋을 구성하였다. train/valid 데이터셋 분할 비율은 9:1이다.

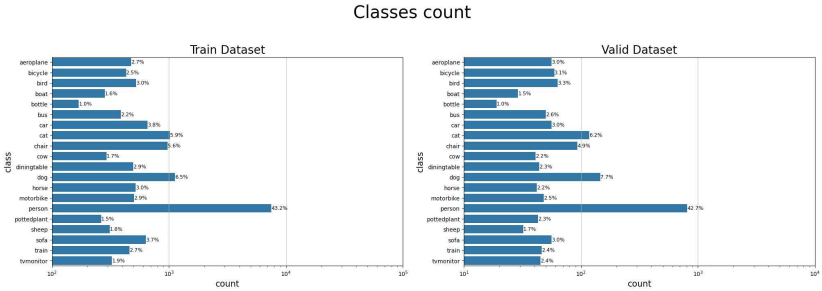

그림 13 train/valid dataset class 비율

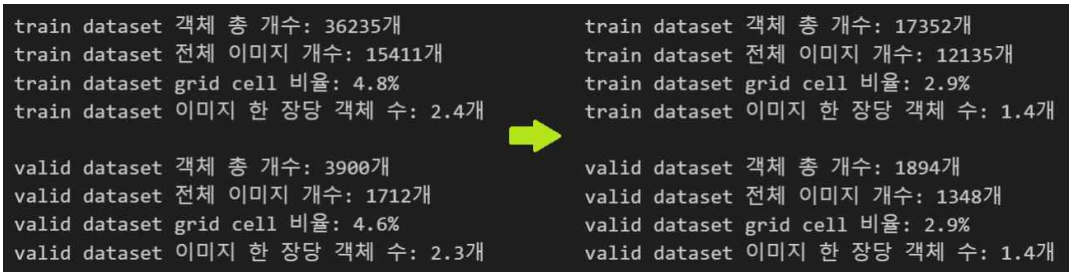

그림 14 dataset 객체 수 변화

2) 1.5 pixel을 제외하는 과정에서 데이터 수에 이미지와 같은 변화가 있었으며 데이터 개수가 크게 감수하였다.

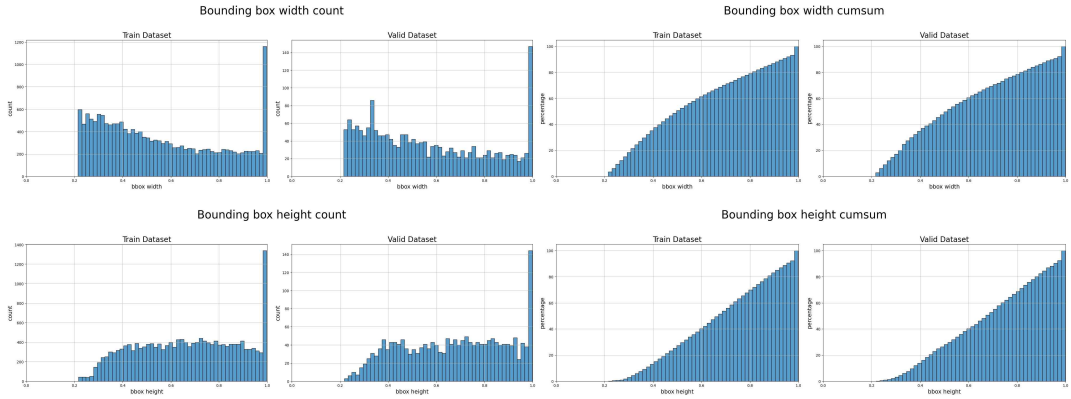

그림 15 bounding box 가로 세로 길이 분포

3) 1.5 grid cell 이하의 객체들을 제외하여 0.21 이상의 normalized width, height만 남게 되었다.

4) 여기서부터는 이미지가 너무 커서 문서에 담지 못했으니 이미지 원본을 참조하길 바란다. dataset2_7 이미지를 보겠다. 대부분의 cell에서 dataset1 대비 객체 수가 감소했으며 valid data가 존재하지 않는 cell의 수도 증가하였다.

5) dataset2_8 이미지를 보겠다. 모든 cell에서 누락되는 이미지가 거의 사라진 것을 확인할 수 있다. 인접하는 이미지들은 대부분 normalized width, height 0.21 이하의 작은 개체들인 것을 유추할 수 있다.

6) dataset2_9 이미지를 보겠다. 이미지 모서리의 cell에서 특정 class가 나타나지 않는 문제점이 증가하였다.

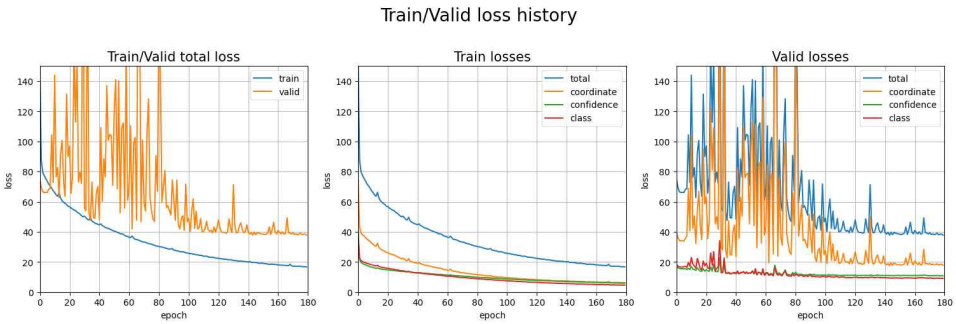

그림 16 loss history

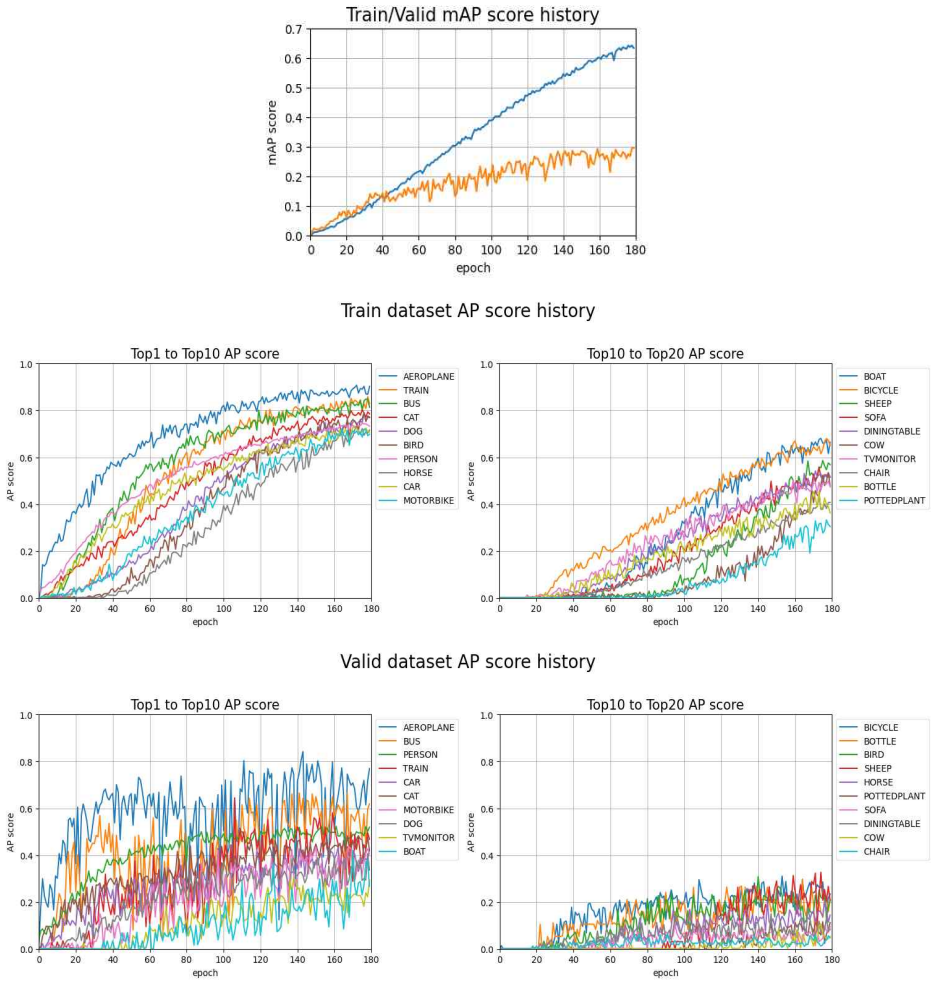

그림 17 AP score history

7) 실험2의 학습 설정 실험1과 동일하며 180 epoch 학습 수행하였다. 180 epoch에서 mAP 성능은 train 0.63, valid 0.30으로 마무리하였다. AP score에서 aeroplane 클래스는 score가 0.9에 근접하는 반면 potted 클래스는 0.3 정도로 클래스 별로 탐지 성능 차이가 많이 나는 것도 확인할 수 있다. loss와 mAP score의 그래프를 보아 40~60 epoch 근처부터 과적합이 발생하며 모델의 일반화 성능 획득에는 실패하였다.

8) 원래대로라면 valid dataset에 대한 성능을 분석해야겠으나 모델 일반화에 실패했기 때문에 분석 신뢰도는 떨어지겠지만 train datset에 대한 분석으로 대신하겠다.

total loss: 12.3524	mAP: 0.7194955304265023	CAR: 0.75	MOTORBIKE: 0.80
coordinate loss: 4.5233	AEROPLANE: 0.92	CAT: 0.85	PERSON: 0.80
confidence loss: 4.7422	BICYCLE: 0.75	CHAIR: 0.43	POTTEDPLANT: 0.36
class loss: 3.0869	BIRD: 0.85	COW: 0.66	SHEEP: 0.70
	BOAT: 0.76	DININGTABLE: 0.64	SOFA: 0.67
	BOTTLE: 0.50	DOG: 0.83	TRAIN: 0.88
	BUS: 0.85	HORSE: 0.79	TVMONITOR: 0.60

그림 18 loss 및 AP score

transform을 test transform으로 설정 후 train dataset에 대해 추론한 결과이다. bottle, chair, pottedplant 클래스에서 0.6 이하의 AP score가 나타났다.

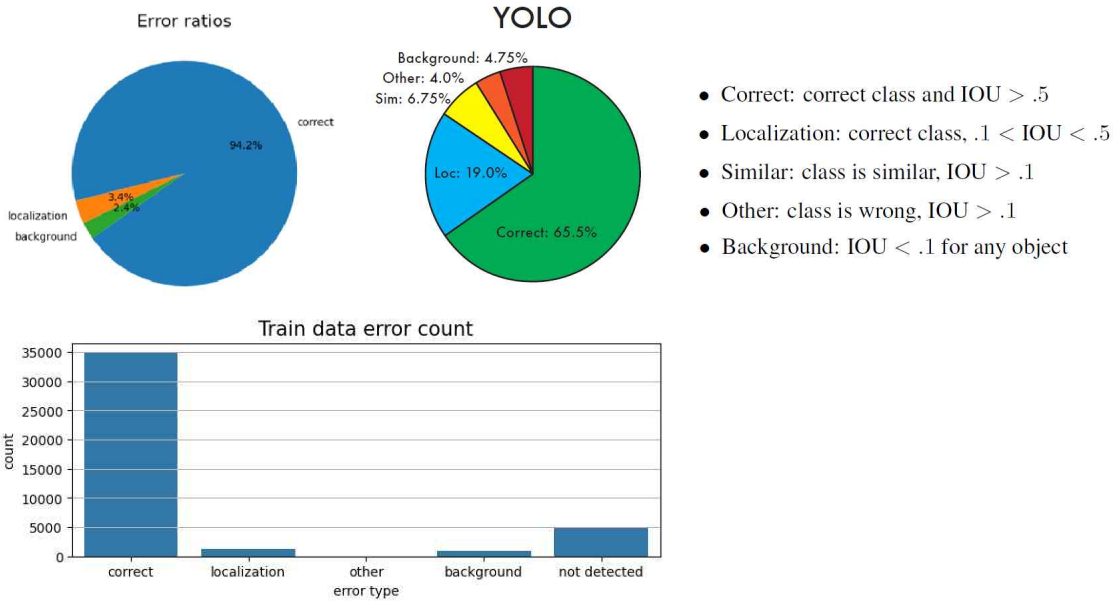

그림 19 에러 분류별 개수

논문을 참고하여 IOU 수치와 class 일치 여부에 따라 correct, localization, similar, other, background로 에러를 구분하였다. 거기에 추가하여 위의 분류에 걸리지 않는 ground truth가 존재하여 not detected 분류를 추가하였다. 첫 번째 파이차트가 실험2의 추론 결과, 두 번째 파이차트가 논문의 실험 결과이다. 에러 상 클래스 예측이 틀린 결과가 없고 NMS 과정에서 confidence score 이상의 bbox만 남기 때문에 이에 대한 영향을 따로 보기 어려워 coordinate 에러에 의한 결과를 주로 분석하면 될 것 같다.

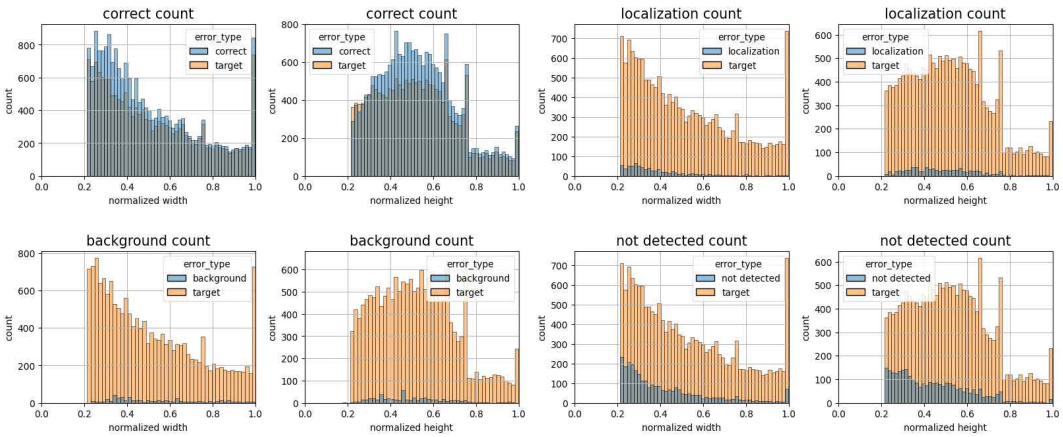

그림 20 에러 분류 별 bbox의 width, height 분포

9) 정답률이 너무 높아서 유의미한 분석이 어려워 보인다. not detected에 대해서만 살펴보면 작은 사이즈의 객체일수록 감지가 덜 되는 경향을 볼 수 있다. width, height가 1인 객체에서 에러 비율이 높은데 학습 시 이미지를 crop하는 과정에서 이미지가 일부 잘려나가기 때문에 test transform 적용 시 다른 이미지 분포를 보여서 그런 것 같다. 참고로 위 그래프에서 correct가 target보다 개수가 많은데 모델의 추론 결과는 타겟의 개수보다 많거나 적을 수 있으니 오해하지 말자.

10) experiment2_12~13 이미지를 보겠다. cell[3, 2~3], cell[3, 1~4], cell[3, 1:5] 부분은 높은 정답률을 보이는 반면 나머지 부분들은 정답률이 떨어진다.
11) experiment2_14~23 이미지를 보겠다. 클래스별로도 살펴보기 위해 AP score가 높은 person, bus 그리고 AP score가 낮은 potted plant, bottle, chair 등을 grid cell 별로 살펴보겠다. 이미지를 보면 person 클래스의 경우 객체 수도 많고 중심에서 정답률도 높다. bus 클래스의 경우 객체 수가 낮음에도 정답률이 높다. 나머지 세 개의 클래스는 객체 수도 낮고 중심부의 정답률마저 낮음을 보여준다.
12) 정리해보면 모델의 출력 결과물인 coordinate, confidence, class 세 관점에서 에러 요소를 분석해보려 하였다. 먼저 confidence에 의한 분석은 NMS를 통과하기 이전 결과물로 수행해야 되는데 이 부분은 수집하지 못해서 제외하였다. class에 의한 부분은 출력 결과가 전부 class를 맞춰서 other로 분류된 경우가 없었기 때문에 분석하지 않았다. 따라서 coordinate에 의한 분석만을 진행했고 분류 별 좌표의 분포, grid cell 별 correct 분포, grid cell 별 특정 클래스의 correct 분포 등을 살펴보았다. 이로부터 얻은 관점들에는 이미지 외곽에는 데이터가 적고 학습이 어려울 것이라는 점, 객체의 크기가 작을수록 학습이 어려울 것이라는 점 등을 생각해볼 수 있다.
이해가 가지 않는 부분은 person 클래스의 경우 데이터가 충분히 많은 cell인데도 정답률이 떨어지는 부분이 존재한다. 또 반대로 bus 클래스의 경우 데이터 개수가 굉장히 적은데도 정답률이 높다. 단순히 셀의 위치나 객체 숫자에 의해 성능 차이가 날 것이라는 가정은 틀린 것 같다.
13) 실험을 돌리고 나서 알게 된 사실인데 PadIfNeeded 함수는 이미지를 가운데 배치하고 상하좌우에 패딩을 하는 함수이기 때문에 이미지가 중앙으로 쏠리는 것에 일조하였다. 이번 실험에서 색상 변화, 비율변화, 중심좌표 이동 등 여러 데이터 증강 중 가장 중요한 것은 객체의 중심 좌표를 변화시켜 모든 cell에 다양한 객체를 학습시키는 것이라고 판단했는데 이런 기능을 전혀 제공하지 못했다.

다. 실험3

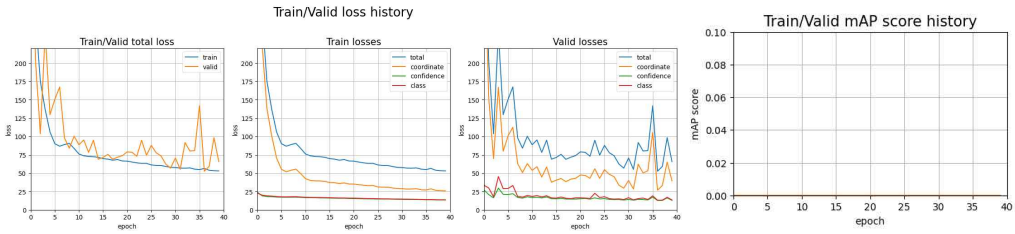

그림 21 loss history

그림 22 mAP history

1) 실험3은 실험2의 데이터셋으로 학습하며 coordinate loss에 obj mask를 사용하지 않고 학습하였다. 학습은 40 epoch 진행하였다. 실험 결과 loss는 꾸준히 떨어지는 것을 보여주나 AP score는 전부 0을 출력하며 모델의 출력 좌표를 살펴보면 역시 0을 출력하는 것을 확인할 수 있다. 이를 바탕으로 obj mask를 쓰지 않는 경우 모델이 0을 학습하는 것을 확인할 수 있다.

4. 생각해볼 것 들

가. 컴퓨터의 시각에 대한 이해

이번 학습을 통해 detection model의 이미지 외곽 객체 검출 성능이 현저히 떨어짐을 알게 되었다. 사실 생각해보면 외곽에 있는 객체는 일부가 잘릴 수밖에 없는 것이고 피사체를 모서리에 두고 찍는 경우는 잘 없기 때문에 학습 상 외곽 객체 검출 성능이 떨어지는 경우는 당연한 것이다.

사람도 마찬가지지만 초점을 맞춰야지만 사물을 볼 수 있고 초점 외에 인식은 어려운 편이다. 하지만 초점을 맞춰 중요한 객체와 아닌 객체를 구분해낼 수 있고 중요한 객체가 외곽에 있다면 눈을 돌려서 바라보면 그만인 것이다. 이것이 초점을 움직일 수 없어서 한 장의 이미지 안에서 모든 것을 해결해야 하는 기계와 사람의 큰 차이인 것 같다.

나. 양질의 데이터셋과 모델

학습 결과를 통해 YOLOv1의 작은 객체와 인접 객체 검출 성능이 떨어짐을 알게 되었다. Pascal VOC 데이터셋의 경우 1 grid cell 이하 크기의 객체, 인접 객체 등이 다수 있으며 모서리 근처에 대부분이 잘려나간 객체, 다른 객체 뒤에 가려진 객체 등 데이터가 난해한 경우도 많다. 이러한 데이터들은 label에서 제거하더라도 이미지 상으로는 남아있으며 데이터 증강에 의해 이미지가 변하게 되면 다시 검출되어 모델 학습에 혼란을 줄 수 있다. 모델이 이해할 수 있을 정도의 난해함과 객체 크기를 가진 데이터셋이 학습에 적합할 듯 하다.

다. detection model의 구조적 한계점

실험 초기에는 model에 obj mask를 적용하는 것이 모델에 정답과 오답 중 정답만을 학습하게 하고 특정 소수의 신경망만을 학습시키기 때문에 학습에 부정적인 영향을 줄 것이라고 생각했다. 실험 후기에는 detection task는 학습시키기 어려운 task이며 특정 소수의 신경망에 정답만을 학습시키는 것이 효율적인 방식이라고 생각을 바꾸었다. detection 모델은 모델이 이미지를 학습하는 개념보다는 출력 cell 하나하나가 이미지를 학습하는 개념으로 봐야한다. 여기서 문제점은 대부분의 cell과 거기에서 이어지는 신경망이 학습되지 않은 상태이고 모든 것을 confidence score 하나에 의존하기 때문에 객체의 중심이 살짝만 어긋나도 정답률에 크게 영향을 미칠 것이라고 생각된다.

confidence를 사용하는 구조가 feature가 원본보다 작아진 상태에서 객체를 추론할 수 있게 하고 NMS로 bbox를 줄이는데 도움을 주지만 단점도 있다고 생각한다. 가능한 방식인지는 모르겠으나 instance segmentation으로 객체의 픽셀을 찾고 픽셀의 가장자리의 좌표로 bbox를 추론하는 방식을 사용하면 좀 더 강건한 모델이 되지 않을까 생각해본다.

라. 멍청한 모델과 올바른 학습 방향

이번 실험 결과 모델이 데이터를 학습하는 것이 아닌 0을 학습하는 경우가 많았다. 손실함수를 어떻게 구성하느냐에 따라, 데이터셋의 비율 혹은 품질에 따라서 모델의 학습 방향이 크게 달라지는 것을 알게 되었다.

gradient descent 기법의 도입에 따라 인공지능이 손실 함수를 최소화하는 방향으로 신경망을 재배열할 수 있게 되었다. 하지만 gradient descent는 어느 방향으로 가야 loss가 증가하는지 감소하는지만 알려줄 뿐 학습 전체를 내려다보지는 못한다. 그렇기 때문에 여러

metric과 loss 기록을 통한 학습 모니터링, learning rate에 의한 학습 속도 모니터링 등 사람이 관여하게 되는 부분이 발생한다.
사람은 의식적으로 실패에 대한 경험을 걸러낼 수 있기 때문에 어려운 일을 하더라도 사람의 역량은 실패하는 쪽으로 수렴하지 않는다. 인공지능 학습도 마찬가지로 객체 수, 클래스 분포, bbox의 크기와 위치 분포, 학습 속도, metric 성능 변화 등을 컴퓨터가 고려하여 선택적인 학습이 가능하면 좋을 것 같다.

5. 향후 과제

가. 실험 데이터 관리 방법 도입

이번에 학습을 진행하면서 학습 회수가 증가함에 따라 학습 조건의 기록, checkpoint, 추론 결과의 관리에 많은 어려움을 겪었다. 다음 학습부터는 WandB를 적용해보겠다.

나. 데이터 증강 문제 해결

다음번으로 진행할 YOLOv3에서 같은 문제를 겪지 않기 위해 여러 grid cell을 학습시킬 수 있는 데이터 증강 기법을 조사해보겠다.

다. 누락 이미지에 대한 분석

데이터셋 변환 과정에서 누락되는 데이터들이 다수 발생했고 이러한 데이터들은 label에서는 제외되더라도 결국 이미지에 남아 학습 및 추론 시 오답을 유도하게 된다. 다음번에는 error analysis에서 이런 데이터도 고려할 수 있도록 함수를 수정해보겠다.

라. 이미지의 시각적 분석

error analysis 데이터들로부터 오류를 발생시킨 원인에 대한 인사이트를 얻는 것까지는 성공했지만 실제로 그런 관점들이 맞는지 확인하기 위해서는 에러가 난 이미지를 한 장 한 장 볼 수밖에 없을 것 같다. 시간이 더 있으면 이런 과정도 진행해보면 좋을 것 같다.