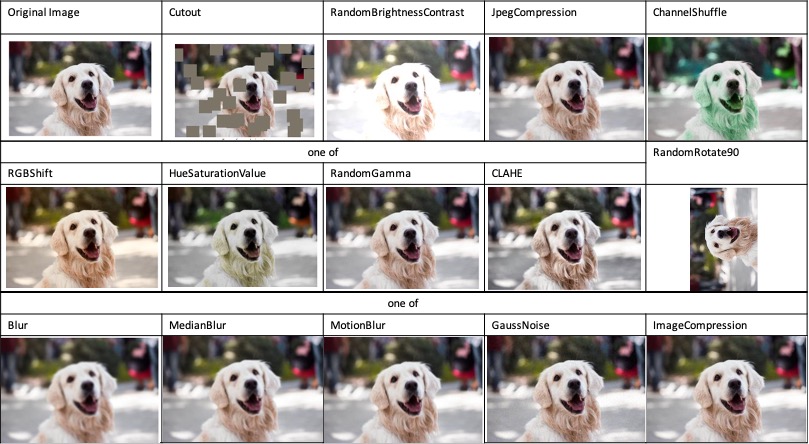
### 대회 개요

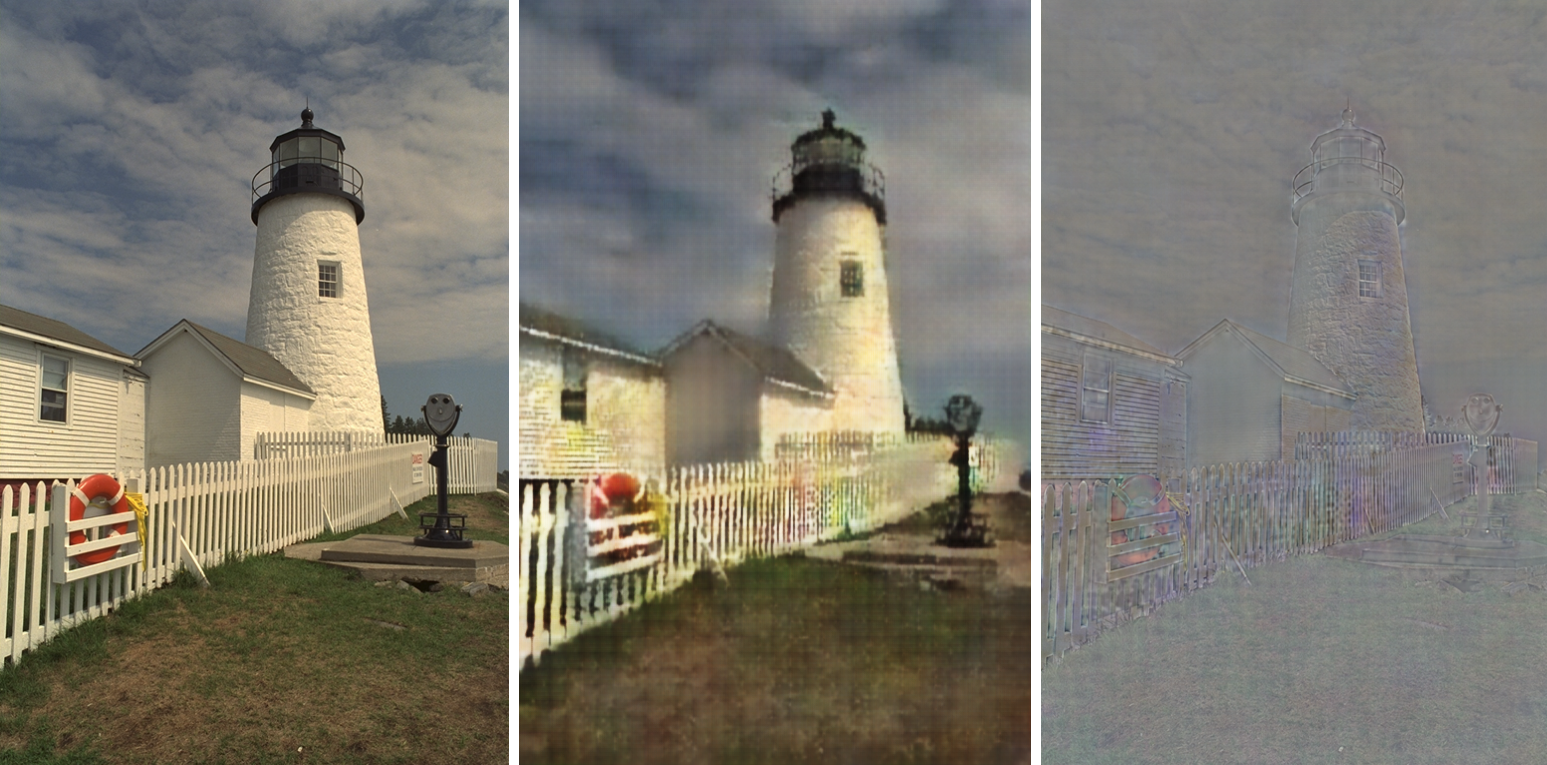
* 대회명: 재활용 품목 분류를 위한 Object Detection
* 구성원: 서석민, 서주원, 이근재, 이동빈, 이재영
* 기간: 5/10 ~ 5/20
* 프로젝트 진행 방법: Segmentation Task와 같은 방법으로 진행

### 파이프라인

#### Data

* + Validation이랑 LB score랑 차이가 많이 난다는 것을 확인하고 인지 후 test를 진행하였고, 중요한 test들은 LB점수를 통해 확인하면서 실험을 진행하였다.
  + faster rcnn 기준
    - Kfold : 0.4150 LB
    - train\_all : 0.4284 LB
      1. 위 결과를 통해 kfold보다는 train\_all 점수가 더 높다는 것을 알게 되었고 모델의 최종 score를 산출할때에도 train\_all로 진행하였다.
* Augmentation



* + - Kaggle [Wheat Detection](https://www.kaggle.com/c/global-wheat-detection/discussion/172569) 참고함
    - faster rcnn r50 1x 기준, 기본 augmentation을 적용했을 때의 점수 0.4433 -> 아래 augmentation 적용했을 때의 점수 0.4624 로 상승하였다.
      1. Cutout: 이미지의 부분을 잘라내어 몇 가지 object를 없애는 효과가 있다.
      2. RandomBrightnessContrast: Contrast를 높게 주어 다른 이미지처럼 보이게 함.
      3. ImageCompression : 

이미지를 압축시켜 사진의 선명도를 조절시킨다. parameter로 quality\_lower 와 quality\_upper를 통해 이미지의 압축률을 조절할 수 있다.

* + - 1. ChannelShuffle: 각 channel을 바꿔 다른 이미지처럼 보이게 하였다.
      2. OneOf: RGBShift, HueSaturationValue, RandomGamma, CLAHE: 이미지의 색상 변화를 통해 변형을 주었다.
      3. RandomRotate90: 이미지를 90도 회전하여 다른 이미지처럼 보이게 하였다.
      4. OneOf: Blur, MedianBlur, MotionBlur, GaussNoise, ImageCompression: 뿌옇게 하거나 노이즈를 주어 다른 이미지처럼 보이게 하였다.
    - Flip
      1. horizontal, vertical중 어느 augmentation이 이번 task에 적합한지에 대해 test를 진행하였다.
         * vfnet multiscale train 을 통해 학습 후, test TTA 또한 학습의 flip방법과 맞추어 주었다.

no flip : 0.5317 LB

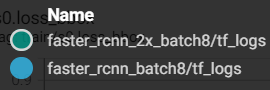
horizontal : 0.5336 LB

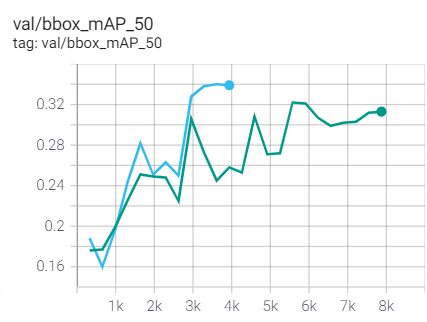
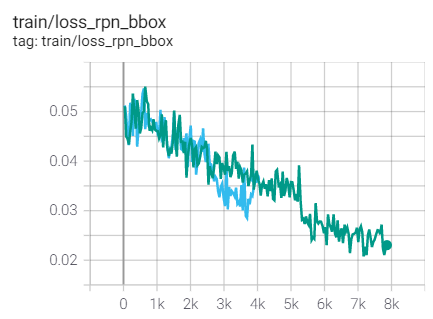
vertical : 0.5277 LB

horizontal + vertical : 0.5287 LB

* + - * + 실험을 통해 default로 되어있던 horizontal이 가장 좋은 성능을 보였다.

#### Scheduler

* + Faster\_RCNN\_1x VS Faster\_RCNN\_2x
    - 결과 : 2x Scheduler 제출 mAP 상승 0.4699 -> 0.4746
    - idea : 아래 Train loss graph에서 1x의 경우 여전히 loss가 줄어들고 있고, 좀 더 길게 학습할 경우 점수 향상이 있을 것이라 판단하여 lr scheduler 및 epoch가 2배인 2x config로 학습 결과를 확인하였다.

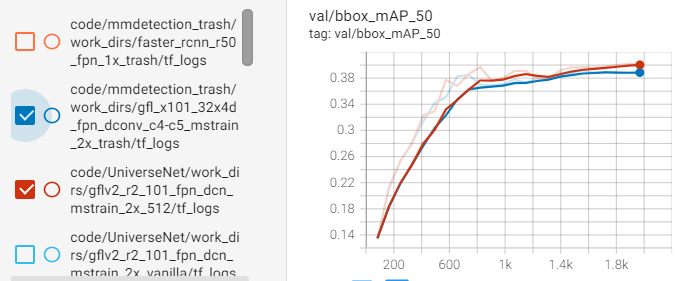
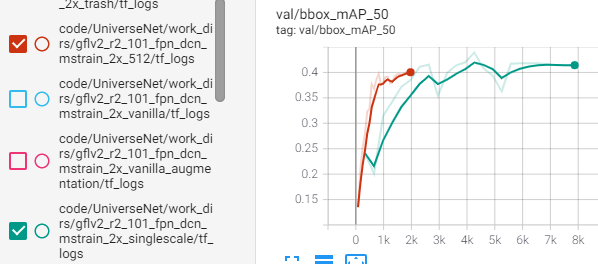
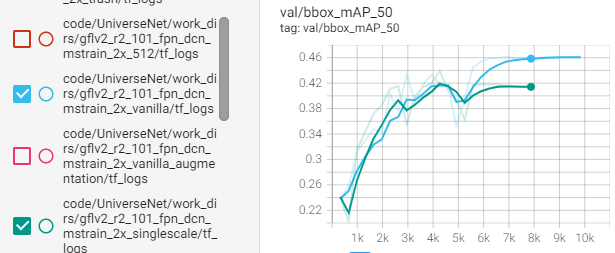
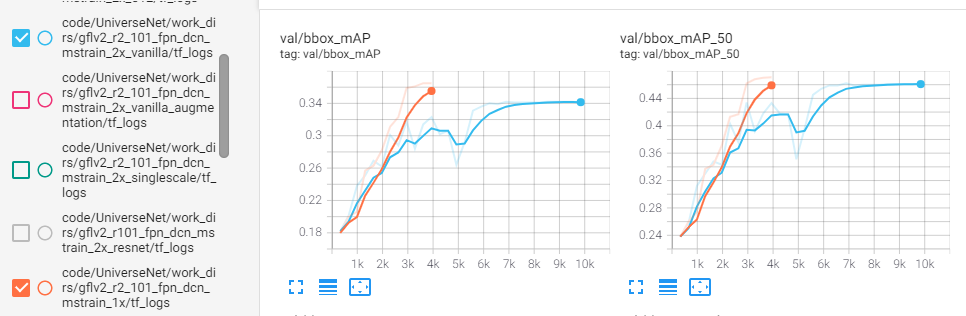
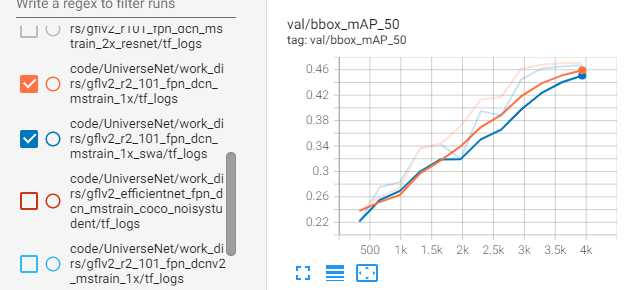
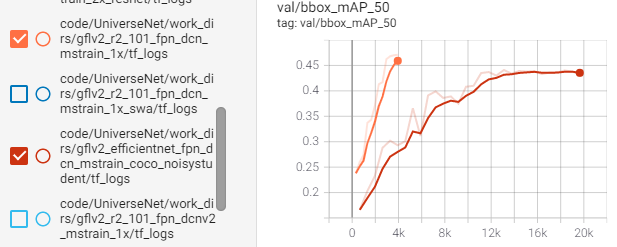
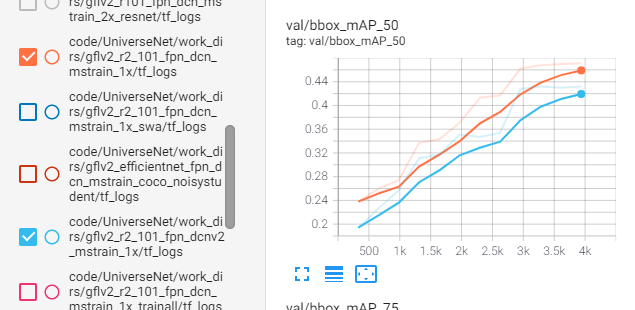
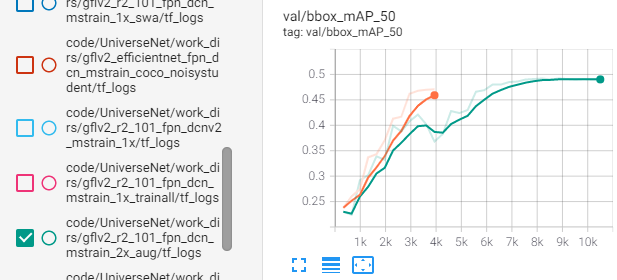


#### Modeling

##### Faster RCNN

* + - 여러 실험을 하기에 baseline을 통해 진행하는 것이 시간이 훨씬 적게 들어 이를 통해 진행하였다.
      1. 기존의 pretrain이 (1333, 800) scale을 통해 진행되었으며, 이에 대한 train, test의 scale에 대해 실험을 진행하였다.
         * train (512,512), test (512,512) : 0.4411 LB
         * train (1333,800), test (512,512) : 0.4077 LB
         * train (1333,800), test (1333,800) : 0.4437 LB

##### GFL V2

* + - mmDetection에 포함되어 있는 GFL이 좋은 성능을 보여주는것으로 알려져 있어 사용해보았다가 GFLv2가 새로 발표되었다는것을 알고 성능을 비교해보게 되었다. 기존의 GFL은 20epoch, mAP 50값 기준, 0.3886의 성능을 보여주었으나, GFLv2는 0.4003으로 더 높은 성능을 보여주어 GFL을 사용하지않고 GFLv2를 사용하게 되었다.
    - GFLv2의 초기 config는 width를 1333으로 고정하고 진행되기 때문에, 우리 데이터의 width인 512와는 맞지 않다고 생각되었고, width를 512로 변경한 후 성능을 비교하였다. width를 1333으로 변경 후 학습하는 경우는 메모리 사용량이 더 많았기 때문에 배치사이즈를 기존의 32에서 8로 줄여서 학습하게 되었다.빨간색 그래프가 width가 512, 초록색 그래프가 width가 1333으로 고정된 되었을때이다. 결과적으로 width가 1333으로 고정된 경우가 더 좋은 성능을 보여주었는데, 이는 제공된 pretrained 모델이 width가 1333으로 고정되어서 학습되었기 때문인 것으로 보인다.
    - 학습시 성능이 width가 1333으로 고정된 경우가 더 좋았기 때문에, width는 1333으로 고정하고, [1333, 800]의 single scale로 학습되었을 때와, [1333, 960], [1333, 480] 사이에서 랜덤 multi scale로 학습한 경우의 성능을 비교하였다.24epoch 기준으로 multiscale 성능이 0.46, singlescale 성능이 0.414로 multiscale 학습할때의 성능이 월등히 높았다. multiscale 학습을 더 진행할 경우 성능 향상이 예상되어 30epoch까지 추가로 학습하였으나 더이상의 성능향상은 볼 수 없었다.
    - 학습을 24epoch 기준으로 진행했을때, 중간부분에서 성능이 하락하는 부분이 공통적으로 발생했다. 그래서 12epoch을 기준으로 진행하여 중간에 과적합되는 부분을 없애보기로 했다. 결과적으로 12epoch 기준 학습일때 0.471, 24epoch 기준 학습일때 0.461의 성능을 보여주었다. 많이 차이나는 성능은 아니지만, 전체적인 mAP성능은 0.341에서 0.365로 더 많이 상승하였고 12epoch 기준 학습일때 학습시간이 절반이므로 12epoch 기준 학습으로 변경하였다.
    - 딥러닝 모델의 일반화 성능을 높여주는 SWA(Stochastic Weight Averaging)이 validation 및 test 성능을 더 높여줄 것으로 예상되어 적용하고 실험하였다.이유는 알 수 없지만 GFLv2에서는 SWA를 적용하지 않았을때의 성능이 조금 더 높았기 때문에 SWA는 채택하지 않았다.
    - 기존의 GFLv2는 Res2Net이 Backbone이나, 이전 segmentation 대회에서 EfficientNet이 backbone으로 좋은 성능을 보여주었기 때문에 Res2Net 대신 EfficientNet을 backbone으로 하여 실험을 진행하였다. EfficientNet을 backbone으로 하여 학습된 pretrained 모델이 없었기 때문에, 전체 모델에서의 pretrained 모델은 사용하지 못했고, EfficientNet의 pretrained 모델로는 NoisyStudent dataset으로 학습한 pretrained 모델을 사용하였다. 전체 모델에서의 Pretrained 모델을 사용하지 않았기 때문에, 24 epoch을 기준으로 하여 더 오래 학습하도록 하였고, EfficientNet B4모델을 사용할 경우 메모리 초과가 발생하였기 때문에 배치를 기존의 절반으로 감소시켰다.빨간색 그래프가 EfficientNet을 backbone으로 사용한 그래프이다. 기존의 Res2Net의 성능에 가깝게 도달했으나 그 이상의 성능은 보여주지 않았고, 30epoch까지 추가학습을 진행해도 더 이상의 성능 향상은 보여주지 않았다. 학습시간도 기존 2시간에서 8시간으로 늘어났기 때문에 EfficientNet 사용은 채택하지 않았다.
    - 기존의 config는 dcn을 사용하고 있었는데, dcnv2를 사용할 경우 성능향상을 보이는지 테스트를 진행하였다.하늘색 그래프가 dcnv2를 사용한 경우인데, 기존보다 전체적으로 낮은 성능을 보여주었기 때문에 채택하지 않았다.
    - augmentation을 적용하여 성능 비교를 진행하였다. augmentation을 적용한 경우 12epoch으로는 underfit 될것으로 예상되어 24epoch으로 늘려서 학습하게 되었다.최종 학습시 성능은 0.471에서 0.49로 증가하였다.

##### VFNET

* + - 각 모델의 config md를 통해 box AP중 높은 모델을 찾아보다가 50.4(val)의 조금 높은 기록을 보게되어 채택하게 되었다.
    - 간단하게 epoch=9 로 설정하여 모델의 성능을 test
      1. single scale test (512) : 0.4830 LB
      2. single scale test + soft nms : 0.4812 LB
      3. multi scale test (256, 512, 768) : 0.4844 LB
      4. multi scale test (384, 512, 640) : 0.4921 LB

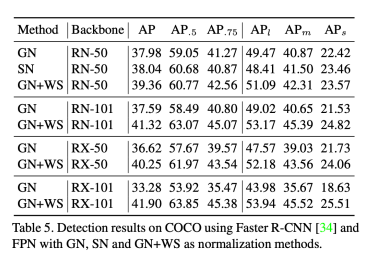
강사님의 자료를 본 팀원의 의견을 토대로 512기준 0.75, 1.0, 1.25배의 scale을 적용하였고, LB점수의 상승이 있었다.

* + - 1. 조금의 에폭으로도 baseline의 faster rcnn (LB : 0.4초반)보다 큰 LB Score차이를 보아 이 모델을 통해 다양한 실험을 진행하였다.
    - multi scale test에서의 성능이 가장 높았고 논문상으로도 multiscale로 학습이 진행되었기에 학습 자체를 multi scale로 학습을 진행한다면 더 좋은 성능이 나올 것이라는 가설을 통해 실험을 진행했다.
      1. train size (256, 512, 1024), test size (256, 512, 1024) epoch 9
         * 0.4126 LB
         * 성능이 많이 떨어졌으며, scale 기준을 모호하게 준 느낌이 들어 scale의 수와 size를 바꾸어 진행하였다.
      2. train size (128. 256. 512. 768. 1024), test size(128, 256, 512, 768, 1024) epoch 9
         * 0.4737 LB
         * 위 실험보다는 높은 성능을 보였지만, single scale train보다는 낮은 성능을 보였다.
         * 위 실험을 진행하기전에 논문에서는 어떻게 진행했을지에 대해 먼저 보았어야 했다는 생각이 들었고, 이를 반영하여 실험을 진행했다.
      3. train size [(1333, 480), (1333, 960)] epoch 24
         * test size [(1333,600), (1333, 800), (1333, 1000)] : 0.4950 LB
         * test size [(1333, 480), (1333, 800), (1333, 960)] : 0.4879 LB

GFL v2를 진행하는 다른 조원의 결과에서는 480, 960으로 학습을 진행하였으므로 [(1333, 480), (1333, 800), (1333, 960)]으로 test를 하니 성능이 훨씬 좋아졌다는 조언을 듣게 되었고, 적용하였다.

모델의 backbone, neck, loss 등의 구조에 따라 inference scale이 다르다는 것을 확인할 수 있었다.

* + - * + 그럼 (1333, 800)으로 single scale 학습을 하고 이 사이즈에 맞추어 가로세로 모두 0.75, 1, 1.25배를 하여 multiscale하면 어떨까 하는 생각에 다음 실험을 진행하였다.
      1. train size (1333, 800), test size [(999, 600), (1333, 800), (1666,1000)]
         * 0.5030 LB
         * 위 실험들로는 multiscale train, test 차이의 근본적인 원인을 찾을 수 없었기에, faster rcnn을 통해 backbone을 바꾸어가며 실험을 진행하였다.
    - 위 faster rcnn의 실험을 통해 resnext에 비해 resnet이 이번 dataset에는 더 적합한 것 같은 판단이 들어 backbone을 resnet 101을 통해 실험을 진행하였다.
      1. vfnet r101 24epoch : 0.5085 LB
      2. vfnet r101 24epoch + DCN : 0.5178 LB
      3. vfnet r101 24epoch + multi scale train : 0.5071 LB
      4. vfnet r101 24epoch + multi scale train + DCN : 0.5235 LB
         * DCN (Deformable Convolutional Networks)의 구조를 검색 (<https://eehoeskrap.tistory.com/406>)을 통해 확인하고 multi scale과 같이 사용하면 multiscale의 단점을 조금 융화시켜줄 수 있을 것 같다는 생각으로 실험을 진행하였다.
         * 결과는 multi scale과 DCN을 단일로 사용할 때 보다, 같이 사용할 경우 성능이 크게 상승하였다.
    - res2net(<https://arxiv.org/abs/1904.01169>)이 논문 제목부터 A New Multi-scale Backbone Architecture이며, 실제 md상의 box AP 또한 높게 기록하고 있어 좋을 것 이라는 기대를 가지고 실험을 진행하였다.
      1. vfnet r2 101 + multi scale train 24epoch : 0.5336 LB
         * lr과 train loss, val AP를 분석해 lr schedule을 조절하여 실험을 진행해 보았다.
      2. vfnet r2 101 + multi scale train 12epoch + fix lr schedule : 0.5253 LB
         * 더 낮은 성능을 기록하였고, 24에폭으로 이후 실험도 진행되었다.
    - vfnet논문상으로 swa를 적용할 경우 조금 더 좋은 성능을 보인다고 나와있어 적용을 해보았다.
      1. vfnet r2 101 + multi scale train + SWA : 0.5453 LB
         * 실제로 성능향상을 볼 수 있었다.
    - 조금더 모델 구조를 바꾸어 성능을 올릴 방법이 어떤게 있을까 찾아보던 중

, 

WS(<https://blog.lunit.io/2019/05/28/weight-standardization/>) 와 GN(<https://blog.lunit.io/2018/04/12/group-normalization/>)에 관련된 논문을 보게되었고, 바로 적용하여 실험을 진행해 보았다.

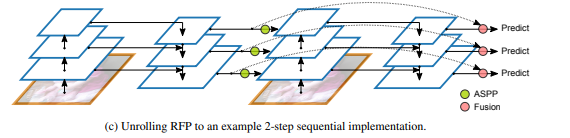
* + - 1. vfnet r2 101 + multi scale train + SWA + WS : 0.5427 LB
      2. vfnet r2 101 + multi scale train + SWA + WS + GN : 0.5445 LB
         * 실제로 ws gn을 같이 사용했을 때에 효과가 더좋았고, pretrained weight가 bn으로 구성되어있어 gn으로 바꿀시에 ws와 gn 레이어는 pretrained가 없이 진행되므로 성능이 낮게 나온 것 같다.
         * lr을 낮추고 epoch를 길게하여 성능을 조금더 끌어올릴 수 있을 것 같았지만, 대회의 남은 기간이 짧은 관계로 추가적인 실험은 진행하지 못했다.
    - vfnet에 관련된 실험들과 augmentation, schedule실험들을 전부 종합하여 적용하여 최종 vfnet의 성능을 만들었다.
      1. 0.5608 LB

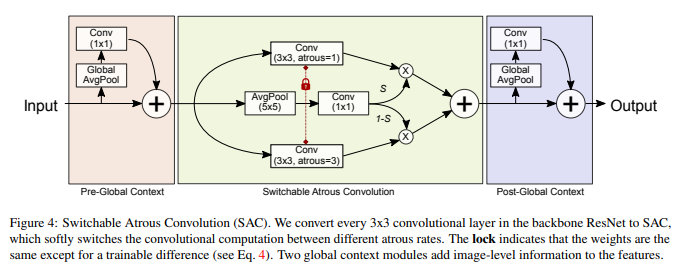
##### UniverseNet

* + - GFLv2보다 높은 벤치마크 성능을 보여주고있어 사용하게 되었다. 대회 마지막에 학습을 진행하였기 때문에 기존의 validation set으로는 성능을 측정할 시간이 부족하였고, train\_all로 바로 학습을 진행하고 리더보드를 통해 기존 모델의 train\_all 성능을 비교하였다.
    - 기존 GFLv2에 모든 방법론을 적용했을때의 최대 리더보드 성능은 0.5706의 성능을 보여주었는데, UniverseNet에 같은 방법론을 적용했을때의 리더보드 성능이 0.5820의 성능을 보여주었다. 따라서 앙상블에 바로 이 모델을 사용하게 되었다.

##### DetectoRS

##### 모델개요: DetectoRS는 그림은 두 번 보고 생각한다는 아이디어를 기반으로 제안된 모델로 RFP(Recursive Feature Pyramid) 와 SAC(Switchable Atrous Convolution)로 이루어져있다.





* + - RFP는 FPN의 결과물을 피드백 연결을 통해 각 stage의 bottom-up backbone으로 전달된다. SAC는 micro level에서 두 번 보는 것을 instantiate하며 두 가지 다른 atrous rate으로 인풋 이미지를 convolve처리한다. RFP와 FPN 하나씩만 적용시킨 모델이 있으며, DetectoRS는 이 두가지 모델을 모두 적용시킨 것을 의미한다. 또한 DetectoRS에는 Cascade를 적용시킨 모델과 HTC를 적용시킨 모델이 있다. 이번 실험에서는 mask 정보가 필요하지 않은 Cascade 기반 Resnet50-DetectoRS를 사용했다.   
      pretrained-model: <https://github.com/joe-siyuan-qiao/DetectoRS>
    - 선택이유: 현재 PaperwithCode 에서 boxAP 55.7로 7번째 순위를 기록하고 있다. mmdetection에 구현되어 있기 때문에 선택했다.
    - 결과: mAP score: 0.4848, PaperwithCode에서 는 ResNeXt101인 반면 적용시킨 코드는 Resnet 이어서 차이가 있었던 것 같다.

##### SWIN

* + - 사용한 이유 : 마스터 클레스로 부터 시작된 불씨가 토론게시글을 통해서 번져갔다. 1등 팀의 swin 사용 사실이 공개되면서 너도 나도 할 것 없이 모든 팀이 swin으로 뛰어들었다. 우리 팀도 마찬가지였다. swin을 쓰면 0.60 점을 넘길 수 있다는 기대로 1등팀의 공략에 우리팀의 augmentation을 추가하여 swin 모델을 구현하였다.
    - 결과 : 제출 mAP 0.54 점 (예상 점수 0.60대비 0.06점 낮음)
    - 낮은 원인 : 논문에는 60epoch까지 train을 해야 최대 성능이 나온다고 되어있으나, 시간 부족으로 30에폭에서 멈추게 되었다. 더 학습했다면 높은 점수를 기록 할 수 있었을 것이다.

##### DETR

* + - 모델 개요 : Transformer를 활용한 Encoder Decoder 구조로 된 모델이다. 아래 이미지에서 볼 수 있듯이 Backbone에서 Feature Maps을 뽑아내고, Query를 Feature map에서 뽑아낸다. 그리고 그 쿼리에 대한 대답을 디코더에서 계산을 해 Feature Map과 함께 최종 예측을 진행한다. 쿼리에 대한 대답을 찾아내는 과정에서 집중하는 부분이 결국 Classification을 이끌어 내는 결정적인 인자이기 때문에, 역으로 그 부분을 시각화 함으로써 모델이 같은 부분을 보고 결과를 내고 있는지를 확인할 수 있다.

## 

* + - 선택 이유 : 인코더 디코더를 사용한 Transformer 구조로 되어있어서, 한 물체를 감지하고 Classification을 하는데 집중할 부분을 학습시킬 수 있다. 그래서 동일한 기준에서 동일 Class를 찾는 일관성을 갖출 수 있고, 좋은 성능을 낼 것이라 생각했다.
    - 결과 : 제출 점수 mAP 0.43으로 많이 낮았다.
    - 왜 좋지 않은 결과를 냈을까?  
      논문에서는 DETR은 V100 16개를 활용하여 3일간 학습한 모델이라고 한다. 우리가 사용하는 V100 한개에 짧은 기간에 성능을 내지 못 하는 것 같다. 조금 더 시간의 여유를 두고 길게 학습했다면, 조금 좋은 성능을 냈을 것이라 예상한다.

##### Emperical Attention

* + - 모델 개요: NLP에서 사용하던 Attention 모델을 CV에서 경험적(empirical)으로 검증한 모델이다. 기존의 트렌스포머 모델에 이미지 픽셀 또는 RoI와 같은 같은 시각요소가 쿼리와 키로 사용된다.
    - 선택 이유: detectoRS에 attention 모델을 붙이기 위해 사용했으며, 성능을 보기 위해 단독으로도 성능을 평가해봤다.
    - 결과: Resnet-50을 backbone으로 사용한 Emprical Attention의 단독 사용의 결과 mAP score: 0.4805가 나왔다.

#### Inference

##### NMS (non-maximum suppression)

##### 기능 모델이 예측한 bounding box 중 가장 정답에 가까운 bounding box를 선택하는 기법이다. 우선 예측된 클래스 중 가장 신뢰도가 높은 bbox를 predict box로 설정한다. 그 후 모델의 정확도를 측정하는 IoU(Intersection over Union) 지표를 이용해 ground truth와 같은 클래스인 bbox들 중 겹치는 영역이 설정한 iou\_threshold를 넘는 bounding box들은 거의 비슷한 대상을 잡았다고 생각해 제거한다. 실험을 위해 신뢰도를 나타내는 score\_thr와 IoU 기준값을 나타내는 iou\_threshold를 parameter로 사용했다.

##### 실험결과

|  |  |  |
| --- | --- | --- |
| nms\_score\_thr | iou\_threshold | mAP |
| - | - | 0.436 |
| 0.00 | 0.40 | 0.4481(채택) |
| 0.06 | 0.50 | 0.4351 |
| 0.04 | 0.50 | 0.4373 |
| 0.00 | 0.50 | 0.4439 |
| 0.00 | 0.40 | 0.4481 |
| 0.00 | 0.35 | 0.4462 |
| 0.00 | 0.00 | 0.3296 |

실험 1~3을 통해 nms가 줄어들수록 결과가 높아진다는 것을 확인했고 실험 4~6을 통해 iou\_threshold는 0.4 내외에서 가장 높은 점수를 낸다는 것을 확인했다. 결과론적인 이야기이지만, 이 실험결과는 private score에 오히려 나쁜 영향을 끼친 것 같다. 팀에서 채택한 nms score\_thr=0.00 iou\_threshold=0.4 값의 의미를 살펴보면 신뢰도가 0인 이미지까지 모두 살펴보며, 겹치는 부분이 0.4인 기준으로 이미지를 지운다는 뜻이다. 다시 말해 정말 멀직히 떨어져 있지 않은 이상 모든 이미지를 채택하겠다는 의미로 nms의 기준을 대폭 완화해 적용시켰다. 그 결과 의미없는 이미지들을 제대로 걸러내지 못했고 이것이 private score의 하락에 영향을 끼쳤다고 생각한다.

##### Ensemble

* + - WBF 앙상블 방법 구현
    - Kaggle [Wheat Detection](https://www.kaggle.com/c/global-wheat-detection/discussion/172569) 참고 및 ensemble\_boxes 라이브러리 사용
    - 토론 게시글에 공개된 code가 있었지만 submission 기준으로 앙상블 하는 코드였고, 우리 나름대로의 모델별 weight를 주어 앙상블을 하고자 하였기에, 새로운 코드를 작성하였다. 특히 kaggle Wheat Detection의 경우 detection box 분류 class가 1개였기 때문에 Box의 앙상블 코드 부분이 달랐고, 우리 대회에 맞게 수정해 주었다.
    - 동일 기준에서 weight를 자유롭게 조절 할 수 있어서 submission 앙상블 코드 대비 0.02점 정도 높은 점수를 낼 수 있었다.

##### TTA

* + - Filp

위 augmentation의 flip의 결과를 토대로 진행하였다.

* + - MultiScale

|  |  |
| --- | --- |
| 모델명 | scale |
| GFLv2 | [1333,960],[1333,800],[1333,480] |
| UniverseNet | [1333,960],[1333,800],[1333,480] |
| VFNet | [1333,800],[1333,900],[1333,1000] |

##### Pseudo Labeling

* + - Kaggle [Wheat Detection](https://www.kaggle.com/c/global-wheat-detection/discussion/172569) 참고함
    - 결과 : Pseudo Labeling은 0.01~0.05까지 제출 mAP 점수 하락이 발생
    - Idea : VFnet으로 학습한 후 Pseudo Labeling 및 추가 학습을 2번 진행, 그 후 UniverseNet을 Pseudo jason을 포함하여 학습을 시키고 앙상블을 하려는 계획을 세웠다.
    - code 구현 : pseudo\_score\_threshold, pseudo\_confidence\_threshold를 이용하여, TEST 이미지에서 Annotation box의 score로 거르고, 그 비율이 높은 이미지만 저장하는 형태로 구현하였다.
    - 잘 안된 이유 : 대회의 mAP score 특성상 모델이 매우 낮은 score로 예측하는 box들이 매우 많다는 것을 알 수 있었다. 그런 box들을 Threshold로 쳐내고 실제 pseudo labeling이 된 이미지를 확인하였다면, 좀 더 좋은 결과를 냈을 것이라 예상된다.

### 팀의 점수 및 순위

(Public) 0.6051, 4등

(Private) 0.4741, 7등

### 최종 제출

1. 아키텍처: VFnet, UniverseNet, GFLv2
   1. training time augmentation
      1. Horizontal Flip, Multi Scale
   2. Lr schedule
      1. 24epoch, WarmUpStepReduceLR
         1. 500 iteration warm up
         2. 16, 22 Epoch Reduce LR
   3. Augmentation
      1. Cutout, RandomBrightnessContrast, JpegCompression, ChannelShuffle, RGBShift, HueSaturationValue, RandomGamma, CLAHE, RandomRotate90, Blur, MedianBlur, MotionBlur, GaussNoise, ImageCompression
   4. SWA ([Stochastic Weight Averaging](https://pytorch.org/blog/stochastic-weight-averaging-in-pytorch/))
   5. Multi Scale Training
   6. Ensemble
      1. WBF
         1. Model Weight : 0.5, 0.5, 0.5
         2. IOU threshold : 0.55
         3. Confidence threshold : 0

### 대회 후기

* **시도했으나 잘 되지 않았던 것들**
  + pseudo labels 방법을 시도해 봤으나 유의미한 성능 향상을 보지 못했다. 쓰레기 detection task 특성상 Model 정확도가 낮기 때문에, 이를 활용해 pseudo labeling을 진행하였을 때 오히려 학습에 방해가 되었다.
  + Group Normalization, Weight Standardization, Emperical Attention들을 논문을 통해 성능향상을 기대하고 VFnet에 적용해서 실험을 했지만 오히려 기본 model보다 성능이 낮게 나와서 아쉬웠다.
  + SWIN 모델의 성능을 끌어내지 못 한 아쉬움이 남는다. 1등 팀이 좋다고 해서 믿고 밤을 새워가며 모델을 만들었다. 겨우 모든 오류를 해결하고 모델을 정상적으로 돌렸으나, 논문에서 제안한 60 epoch까지 돌리기엔 물리적인 시간이 부족하였다. 할수 없이 30 epoch에서 stop 하고 제출을 진행하였다. 결국 기대에 못 미치는 0.54점을 기록하였고, 앙상블을 해도 점수하락으로 이어져서 사용할 수 없었다.
* **학습과 관련하여 개인과 동료로서 얻은 교훈 (구체적인 상황을 기반으로)**
* **피어세션을 진행하며 좋았던 부분과 동료로부터 배운 부분**
* **한계/교훈을 바탕으로 다음 스테이지에서 새롭게 시도해볼 것**