Seg Warp Up Report

Github repo : https://github.com/bcaitech1/p3-ims-obd-garbagecollector

0. 목차

1. Competition 개요 및 결과

- 이미지에서 쓰레기나 물건 등을 픽셀 단위로 분류하는 task
- train[image:2617, annotation:21116], test[image: 837개]
- background, general trash, paper, plastic bag 등 총 12개 class
- class간 불균형이 매우 심하다. (개체 수 기준 7000개~50개)
- 학습용 데이터는 매우 깔끔하게 전처리되어 제공되었으며, 512×512 사이즈

최종 스코어(mIoU) : **0.6892**
최종순위(팀) : **3등**
나의 총 제출횟수 : 46회

---

2. 대회 목표

- 가설을 세우고 실험을 진행할 것
- LB 성능 향상을 위한 기능 구현

3. 대회 진행 11일간의 과정

Day01 4/27 (0.5754 : mIoU)

- baseline 코드 작성
- backbone 실험
- 각 아키텍쳐별로 러닝타임 계산

Day02 4/28 (0.5795)

- train데이터에 overfitting 실험 (loss값 기준으로 확인)
- input size를 256으로 줄여 실험 시간이 4배 이상 빨라짐
- loss함수 및 scheduler 탐색

Day03 4/29 (0.6020)

- validation metric ver.2 추가. (mIoU 계산 방식 변경)
- wandb의 AutoML로 배치사이즈 및 시드 탐색
- backbone(encoder) 탐색

Day04 4/30 (0.6275)

- classmix 구현 및 실험
- augmentation 탐색
- backbone(decoder) 탐색
- validation metric ver.3 추가 (mIoU 계산 방식 변경)

Day05 5/1 (0.6521)

- classmix : 불균형한 클래스에 weight를 크게 주어 실험
- Flip을 이용한 TTA
- scheduler 추가 탐색
- Swin-transformer model 테스트 시작 256 input size로 0.6189

Day06 5/2 (0.6773)

- pseudo labeling

- Scale TTA
- model Ensemble
- hrnet model 테스트 - 0.5812

Day07 5/3 (0.6798)

- KFold
- TTA 추가 (rotate90, counterRotate90)

Day08 5/4 (0.6913)

- 2차 pseudo labeling (로직 변경)
- EfficientDet 구현 (segmentation head 연결)

Day09~10 5/5~6 (0.6991)

- Ensemble list에 EfficientNet+FPN 모델 추가
- team Ensemble (4가지 모델 사용)

4. 최종 앙상블에 사용한 모델

- DeepLabV3+, ResNeXt101_32×16d (0.6748)
- PAN, ResNeXt101_32×8d (0.6786)
- FPN, EfficientNet-b6 (0.6379)
- Swin-Transformer Base (0.6543)

5. 성능 향상에 성공한 아이디어

iou loss + CrossEntropy

- giou라고 하는 detection용 loss함수에서 1-iou값을 사용하겠다는 포인트를 착안하였습니다.
- iou loss 자체만으로는 성능 하락이 있었지만, CrossEntropy와 적절히 섞어 사용했더니 조금의 성능 향상을 보였습니다.
- 여러 비율을 실험해본 결과, (1-iou)*0.4+CE*0.6 비율이 가장 좋았습니다.
- 논문 출처, 블로그 설명글

Augmentation : RandomRotate90, HorizontalFlip

- 수많은 image augmentation 실험 결과 위 두 가지를 조합했을 때, 가장 효과가 있었습니다.
- 많은 성능 향상을 보였습니다.

TTA (Test Time Augmentation)

- 학습 때 rotate와 flip을 사용했으니 inference 때도 분명 효과가 있을 것이라 판단하여 flip을 적용한 결과와 soft ensemble 하였습니다.
- 학습 때는 RadomRotate를 사용했는데, test time에서는 inference 후에 다시 회전하기 전으로 돌려줘야 하기 때문에 직접 회전시켜 주었습니다. 시계 방향 90도, 반시계 방향 90도를 추가하였습니다.
- 총 3가지의 TTA를 추가하였는데, 꽤 괜찮은 성능 향상이 있었습니다.

Pseudo labeling

- P stage 1,2 때도 사용해봤었지만 많은 성능 향상을 보진 못했습니다. 이번 대회에서는 pseudo labeling 덕분에 많은 성능 향상을 보았습니다.
- 픽셀 단위로 학습에 사용할지 말지를 정해야하니 고민이 컸습니다. 처음에는 픽셀 단위로 max prob 값이 0.9 이상인 값만 사용하고 나머지는 backgroung(0)로 채웠습니다. 깨끗하지 못한 데이터가 너무 많이 생겼고, 손으로 일일이 걸러줘야 했습니다.
- 두번째 방법에서 픽셀 단위보다는 이미지 단위로 생각해 봤습니다. 한 이미지에서 백그라운드가 아닌 개체로 분류된 픽셀들의 개수를 구하고, 그 중에서 max prob 값이 0.8 이하인 픽셀들의 수를 구했습니다. 0.8 이하인 픽셀들의 수가 10%보다 많으면 해당 이미지는 제외했습니다.
  (팀원 분들이 도움을 주신 아이디어입니다.)

Ensemble

- 어떤 대회에서보다 ensemble 방법에 가장 집중했던 것 같습니다.
- 이미지 scale(resolution)별로, 여러가지 augmentation별로, 같은 모델이라도 epoch별로, 여러가지 아키텍쳐별로 앙상블을 진행하였습니다.
- scale앙상블 ≥ KFold 앙상블 ≥여러가지 아키텍쳐 앙상블 > augmentation 앙상블(TTA) > epoch 앙상블 순으로 효과적이었습니다.
- 이번 실험을 통해 receptive field의 중요성을 다시금 깨닫게 되었습니다. 넓고 다양할수록 성능이 올라갔습니다.

- 데이터에 대한 중요성도 다시 한번 느꼈습니다. 단순히 학습 데이터를 일부만 바꿔가며 만든 모델들을 앙상블(5-Fold Ensemble) 했더니 많은 성능 향상이 있었습니다.

validation metric 추가

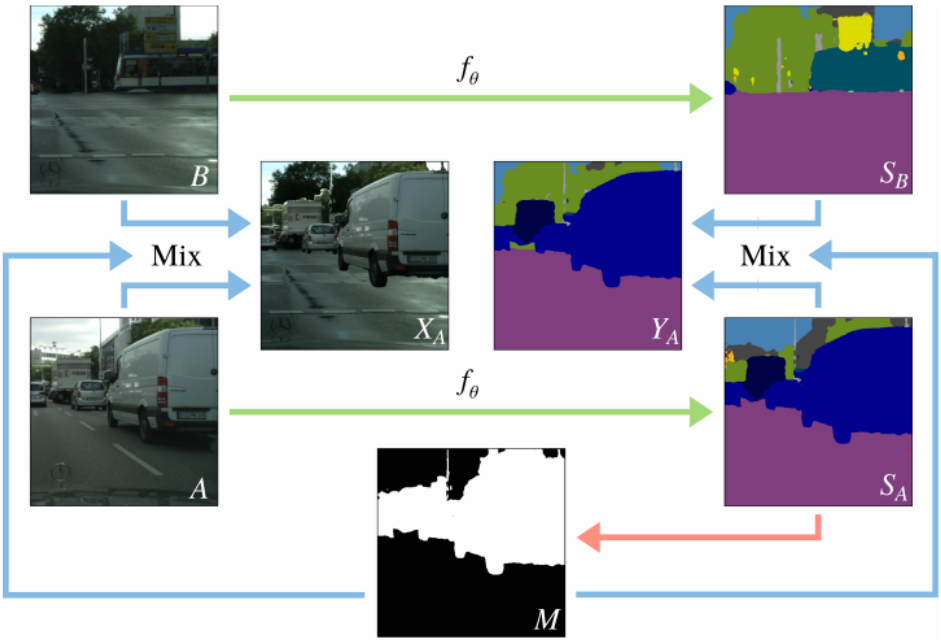
- 성능 향상에 직접적인 도움은 아니었지만, 개선 덕분에 validation 스코어의 신뢰도가 높아졌습니다. 제출해보지 않고도 진행한 실험의 결과를 기록할 수 있었습니다.
- 기본으로 제공된 miou metric 함수는 LB스코어와 갭이 0.2가까이로 너무 컸습니다.
- 기존 metric 함수는 배치 단위로 miou를 계산하는 문제점이 있었고, 모든 이미지에 대해 한번에 miou를 계산하는 방식으로 수정하였습니다. (ver.2)
- ver.2가 LB스코어와 어느정도 근접하게 되었지만, 그래도 갭은 존재했기에 아예 대회 스코어 계산 방식과 동일하게 metric을 수정해 주었습니다. (ver.3)
  각각 이미지에 대해 miou 계산 후 누적하여 전체를 평균하는 방식입니다.

Inference time 개선

- 역시 성능 향상에 직접적인 도움은 아니었지만, 더 많은 실험을 진행할 수 있었습니다.
- 제공된 inference 로직은 시간이 너무 오래걸렸고, 이를 개선해 보았습니다.
- inference의 배치마다 reshape 하는 것을 발견하여, inference를 마친 후 나중에 이미지 단위로 flatten()해주었습니다.
- 기존 20분 가까이 걸리던 시간을 2분 미만으로 단축시켰습니다.

6. 성능 향상에 실패한 아이디어

ClassMix



- augmentation 방법 중 하나로, 다른 이미지에 있는 개체를 픽셀 단위로 가져와 붙여넣는 방식입니다. torch.where함수를 이용해 간단하게 구현에 성공했지만 성능은 하락했습니다.
- 부족한 클래스의 개체 위주로 붙여넣는 방식으로도 사용해봤지만, 성능 향상은 없어서 매우 아쉬웠습니다.
- 논문 출처

EfficientDet 사용

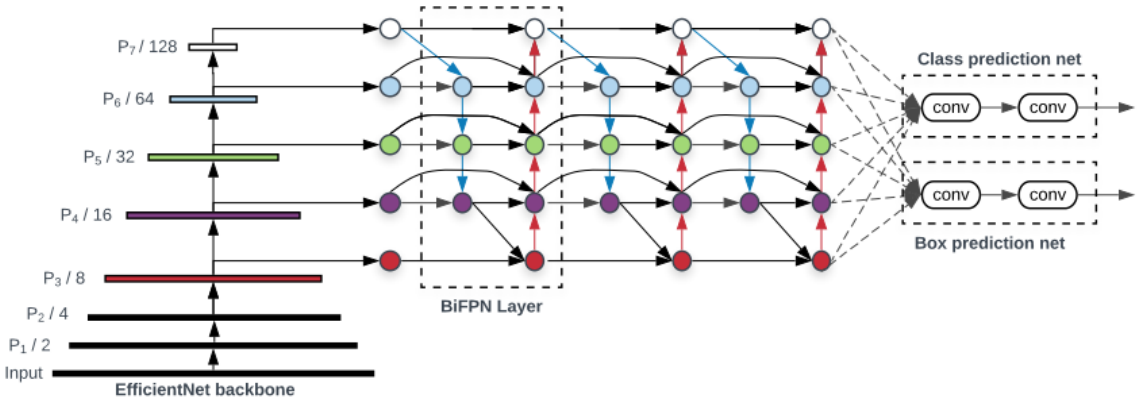


Figure 3: **EfficientDet architecture –** It employs EfficientNet [31] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

- EfficientNet + BiFPN 형태로, 구글링을 통해 detection head가 연결된 아키텍쳐 구현 코드를 발견했습니다.

- detection head를 제거하고 segmentation head를 연결하는데 성공하였지만, 성능이 좋지 않았습니다.

Unet3+ 구현

- Unet과 Unet++에 EfficientNet backbone을 사용한 모델을 사용하였지만 성능이 좋지 않았습니다. 현 시점에서 비교적 오래된 모델이기 때문에 성능이 좋지 못하다고 생각합니다.
- Unet3+ 를 사용하였으나 성능이 좋지 않았습니다. pretrained weight가 없어 처음부터 학습시켰기 때문에, 성능이 좋지 않았다고 생각합니다.
- Unet3+의 encoder로 resnext101를 연결하는데 성공하였으나, 성능이 좋지 않았습니다. Unet3+는 모델 구조 특성상 Full-scale Skip Connections를 이용하기 때문에, 억지로 resnext를 연결할경우, 논문에서 제시된 모델 scale과 많이 멀어져, pretrained 된 weight가 망가지기 때문에 좋은 성능을 기대하기 힘들다고 생각합니다.

다양한 Encoder 사용

- Resnet계열 SOTA를 달성한 다양한 후속 모델들을 Encoder로 사용하였습니다.
- SE-net과 Resnest는 구조상 dilated convolution을 지원하지 않아 deeplabv3plus와 호환되지 않아 사용할 수 없었습니다.
- SKresnext, Regnetx, Regnety는 비교적 좋은 성능을 나타냈지만, Resnext보다 좋은 성능을 보이진 않았습니다. Resnext의 경우 semi-weakly supervised learning을 통한 pretrained weight를 사용했기 때문에, 기본 pretrained weight를 사용하는 다른 backbone 모델에 비해 더 좋은 성능이 나왔다고 생각합니다.

dice, lovasz Loss Function 탐색

- segmentation task에 사용되는 loss함수가 정말 많았는데, 두 가지만 실험해봤습니다.
- 단일로도 사용해보고 다른 loss와 섞어서도 사용해봤지만 두 loss 함수 모두 성능이 하락하였습니다.

효과를 보지 못한 Augmentation

- 낮, 밤에 찍은 사진이 있고 개체가 눈으로 덮인 사진이 있어서 밝기 및 대조 효과, 그리고 날씨 및 snow효과 등이 성능 향상에 도움이 될 것이라고 생각했는데 모두 성능 하락을 경험했습니다.
  이 부분은 매우 아쉬웠습니다.

- RandomResizedCrop 또한 성능이 하락해서, 개체가 포함되지 않게 잘린 이미지가 포함돼서 그런건가 싶어 CropNonEmptyMaskIfExists방법 또한 사용해봤지만, 역시 하락했습니다.
- 이미지의 일부를 랜덤하게 검정색으로 블록처리하는 cutout
- 이미지 밝기에 따라, object가 어두워 잘 보이지 않는 문제가 있어, histogram을 조절하여 이미지를 선명하게 해주는 CLAHE 기법을 사용하였지만, 성능이 하락하였습니다.

효과를 보지 못한 Method

- semi-supervised인 fixmatch를 swin model에 적용을 하였으나 classification과는 달리 confidence가 0.95 이상으로 높아지는 경우가 거의 없고 max가 0.139 정도여서 정확도가 상승하지 않았던것 같습니다.

7. 회고

새로운 분야와 task에 대한 적응

기본 코드를 작성하는게 많이 어려울 줄 알았던 segmentation이었지만, 생각보다 하루만에 정리가 되어서 기분 좋았습니다. 다만 inference, ensemble, pseudo labeling 등의 코드 구현 방법이 지금까지 접했던 image classification, 자연어 개체 분석 task와는 전혀 달라서 시간을 많이 들였습니다.
그래도 stage를 거듭하면서 각 task 마다 처음부터 끝까지 제대로 구현해보니, 코딩에 대한 자신감이 점점 늘어나고 있습니다.

칭찬할만한 부분

- 코드를 구현하는 데에만 그치지 않고, 공유하여 다른 팀원들도 쉽게 사용할 수 있도록 편의성있는 코드를 작성하는데 힘을 쏟았습니다.
- 늦었지만 토론 게시판에 정보 및 코드를 공유한 점 : 대회 막바지였지만, 그래도 한번 용기를 내어 글을 작성하였습니다. 다음 대회에서도 작성할 수 있겠다고 생각했습니다.
- github을 통한 협업능력. 팀과 repo를 만들어 branch를 분리하고, 업로드 및 공유하며 사용했습니다. discussion 게시판도 적극활용하여 소통했습니다.

부족했던 부분

- 데이터를 깊게 보지 못한점 : 이번 대회는 LB 스코어에 너무 목멘 탓인지 데이터를 보기보다 점수 올리는 데에 급했던 것 같습니다. 본래 목표인 코드로의 구현은 많이 해봤지만, 데이터를 분석하는 능력을 많이 길러야한다고 생각합니다.

- 내 모델이 어떤 데이터에 취약한지 확인하고, 문제점을 바탕으로 가설을 세워 진행하려는 노력이 부족했습니다. 다음 대회에서는 가설을 세우려는 노력을 해야겠다고 생각합니다.
- p40으로 swin_base와 같이 너무 무거운 모델을 실험하다 보니 많은 아이디어를 적용하지 못했습니다. 다음 대회부터는 작은 모델로 여러 실험을 진행하고 이후 큰 모델을 사용하는 것이 조금 더 빠른 접근이 가능하다고 생각합니다.

시도해보지 못한 부분

- 객체의 스케일을 바꾸어 class mix하는 방법
- 후처리 CRF
- Classification head
- knowledge distillation
- label noise
- SAM (optimizer)
- Post Processing
- Sobel filtering