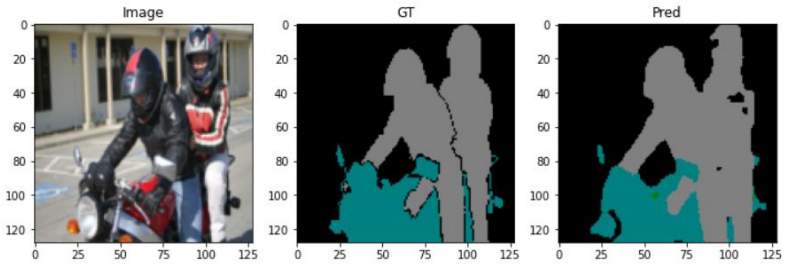
Final Project

20190486 이의인

1. Semantic segmentation



이번 프로젝트에서는 raw image를 10개의 class로 분류하는 Semantic segmentation을 하였다. 위 예시와 같이 사진에서 배경/사람/오토바이로 분류된 것을 알 수 있다. 주의할 점으로는 같은 class의 경우는 객체를 구분하지 않는다.

2. Network Design

다음은, 프로젝트에 잘 맞는 model을 만들기 위해 바꿔 줄 수 있는 변수이다.

Model_baseline : Network 종류, layer 개수, drop out 등을 결정할 수 있다.

Util : data augment를 지정해 줄 수 있다.

Train.ipynb : n_epoch (number of training epochs), batch_size, learning_rate, dataaugment by tramsform, optimizer, scheduler 등 조절 할 수 있다.

3. Attempt

: 다양하게 변수를 바꿔가며 시도했지만 모두 report에 적지 않았다.

: model로는 Unet을 사용하였다. 먼저 느린 연산속도를 개선할 수 있고, Image segmentation에서는 class분류를 위한 객체의 위치 판단과, 인접한 문맥 파악이 동시에 수행 되야 하는데 각 성능은 Trade-off의 관계를 가진다. Unet의 여러 layer는 이런 trade-off를 개선할 수 있다.[2]

3-1. Just trying skeleton code

: 스켈레톤 코드만 돌려보면 문제점을 찾기로 한다.

: train_loss = 0.4337, valid_loss = 25.5593, train_mIoU = 0.8828, val_mIoU = 0.1779, , best mIoU = 0.1880

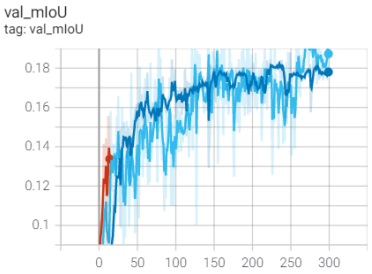
3-2. Batch normalization

: 일단 Training의 속도를 줄이기 위해, 또한 higher learning rate에서도 converge가 가능하도록 convolution 다음마다 Batch normalization을 넣어주었다.

: model_baseline에서 수정하였다..

➔ 다만 network design을 바꾸지 않은 채로 BN만 한 경우에는 , best mIoU에 큰 변화는 없고 수렴 속도만 빨라졌다.

➔ train_loss = 0.8274, valid_loss = 23.0317, train_mIoU = 0.8739, val_mIoU = 0.1796, , best mIoU = 0.2103

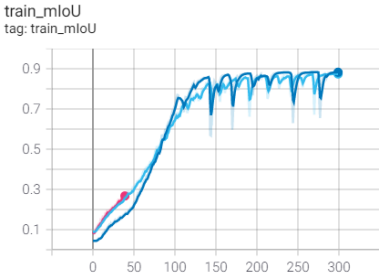


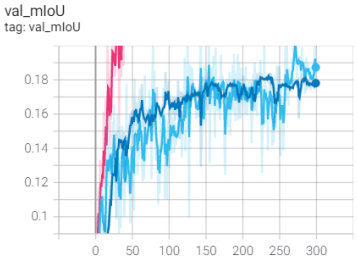
3-3. layer 추가 + optimizer (adam) hyper parameter

: network design을 바꾸지 않고서 다른 parameter를 바꾸는 것은 의미가 없다고 판단되었다. 따라서 decoding, encoding layer들을 하나씩 추가해주었다.

: 블로그 글을 참고하여 optimizer adam's hyper parameter(10^)에 betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False 조건을 추가해 주었다.

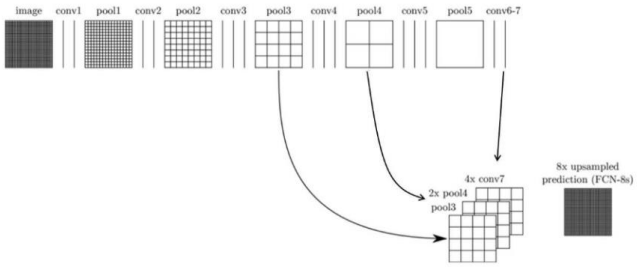
➔ 하지만 train mIoU앞의 결과와 똑같은 양상을 보였다. 이는 이미 pooling을 통해 많은 데이터를 잃어버렸기 때문이다. layer를 추가해봤자 얻을 수 있는 정보는 얻고 잃을 정보밖에 없다는 것이다. 따라서 dropout의 필요성을 느꼈다.





3-4. skip combing

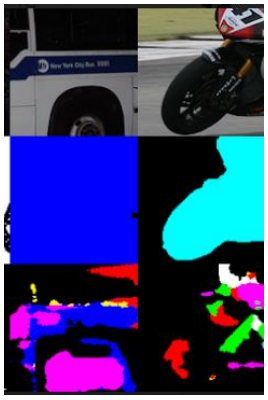
: 학습을 마친 사진을 보니, 해상도가 낮았다. 해상도를 높이기 위해서 skip combining을 해준다. Conv을 많이 하지 않는 단계들이 resolution면에서는 훨씬 높기 때문에 참고하기 위해서이다.



➔ 그럼에도 큰 변화 없었음

➔ train_loss = 1.0222, valid_loss = 21.2453, train_mIoU = 0.8731, val_mIoU = 0.1564, , best mIoU = 0.2011

3-5. Data augment



<- 예상 클래스 분할

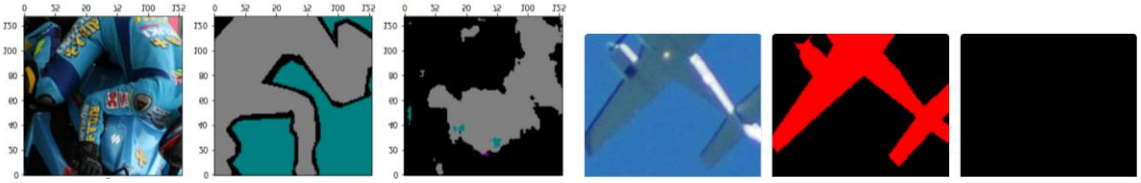
<- 학습 후 클래스 분할

:기존 클래스보다 더 많은 클래스로 분할이 되었다. Overfitting이 일어남.

: train_loss = 1.0222, valid_loss = 21.2453으로 계속 해서 over fitting이 일어나 학습할 data 수가 적다는 판단을 했다. 따라서 augment로 데이터 수를 늘려주기로 하였다. 단, 사진을 위아래로 뒤집을 경우, class 판단에 어려움을 겪는 경우가 있기 때문에 양을 늘려주기로 하였다.

➔ train_loss = 20.2506, valid_loss = 17.8800, train_mIoU = 0.6452, val_mIoU = 0.0993, , best mIoU = 0.1775

➔ 오히려 떨어졌다.



➔ 3-5 data augmentation을 잘못 해줬다는 것을 알았다.

➔ data양을 늘리지 않고, train data 900개 transform을 해줬다는 것을 알게 됐다.

➔ 다음은 util에서 data len를 늘리고 augmentation을 해주기로 했다.

➔ 또, validation data set에는 data augmentation을 하지 말아야하는데 했다.

➔ 이후에는 util 파일에서 train/valid의 data augmentation을 따로 구현해주었다.

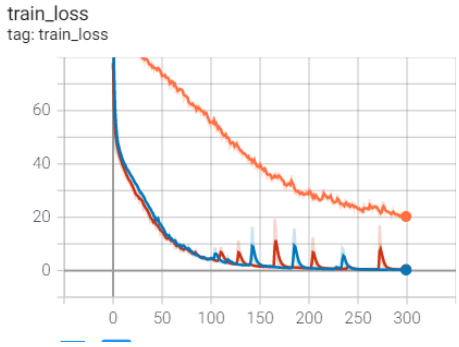
3-6. cropsize->resize

:기존 사이즈인 128,128은 유지

➔ train_loss = 0.3681, valid_loss = 11.4460, train_mIoU = 0.8707, val_mIoU = 0.2416, , best mIoU = 0.2704

➔ mIoU가 급격하게 상승했다.

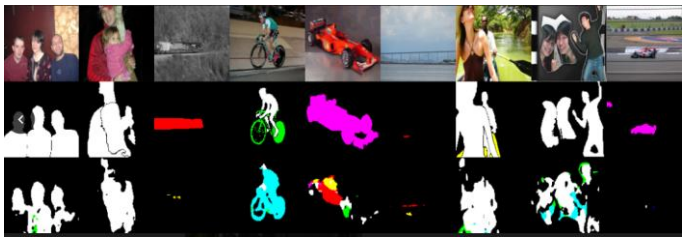
➔ 이유는 잘 모르겠다.

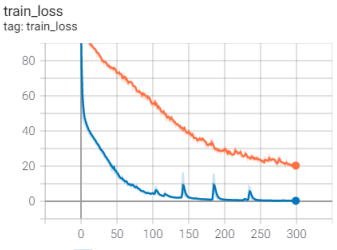


3-7. resize size

: 기존에는 (128,128)으로 resize를 했는데 이미지 크기에 비해 너무 작은 사이즈인거 같아 2배로 늘려주었다. (256,256)

➔ train_loss = 0.2777, valid_loss = 13.6775, train_mIoU = 0.8733, val_mIoU = 0.2477, , best mIoU = 0.2719





3-8. train validation data ratio

: train _mIoU 와 validation _mIoU 차이가 많이 나서 overfitting 예상된다. 따라서 train data set을 줄이고, validation set을 늘리기로 했다. (원래 9:1 비율)

• 8:2 train: validation

➔ train_loss = 0.3271, valid_loss = 26.2558, train_mIoU = 0.8739, val_mIoU = 0.2321, , best mIoU = 0.2689

• 7:3

➔ train_loss = 0.3514, valid_loss = 29.4601, train_mIoU = 0.8707, val_mIoU = 0.2648, , best

mIoU = 0.2824

➔ 큰 차이 없어보임

3-9. 스케쥴러 사용

```
scheduler = torch.optim.lr_scheduler.CyclicLR(optim, base_lr=0.00005, s
tep_size_up=5, max_lr=0.0001, gamma=0.5, mode='exp_range', cycle_moment
um = False)
```

➔ train_loss = 0.5157, valid_loss = 25.8033, train_mIoU = 0.8652, val_mIoU = 0.2067, , best mIoU = 0.2399

➔ 큰 변화 없음

3-10. learning rate 변화

• lr=10 ^ -4

➔ train_loss = 0.4754, valid_loss = 24.1246, train_mIoU = 0.8671, val_mIoU = 0.2620, , best mIoU = 0.2875

• lr=10 ^ -5

➔ train_loss = 1.1626, valid_loss = 6.1420, train_mIoU = 0.8523, val_mIoU = 0.3644, , best mIoU = 0.3871

➔ 최종 모델로 결정

• lr=10 ^ -6 (더 이상 큰 변화 없음)

➔ train_loss = 1.4645, valid_loss = 8.5344, train_mIoU = 0.8445, val_mIoU = 0.3126, , best mIoU = 0.3740

4. Test

Test 돌려보니까 14% 나왔다. train/valid set에만 맞게 fitting 된거 같다. 구글 계정의 한계로 더 이상 돌릴 수가 없다.

5. 투자한 시간에 비해 mIou가 많이 나오지 않아서 아쉽다. 너무 속상하다.

Reference)

[1] Deeplab

https://medium.com/hyunjulie/1%ED%8E%B8-semantic-segmentation-%EC%B2%AB%EA%B1%B8%EC%9D%8C-4180367ec9cb

[2] Unet,

https://tech.socarcorp.kr/data/2020/02/13/car-damage-segmentation-model.html