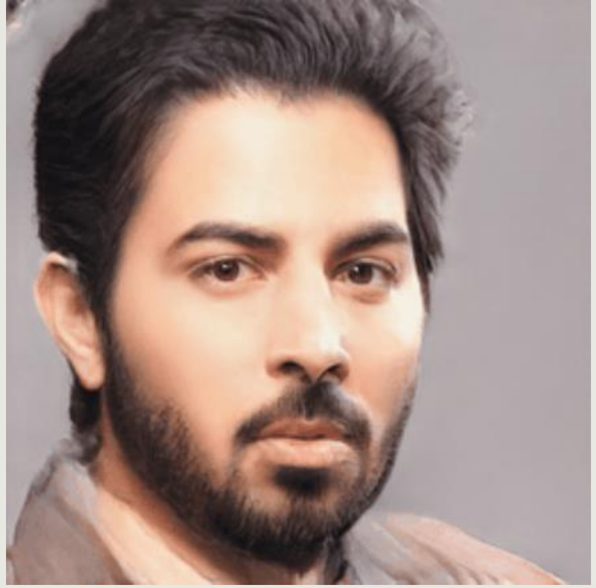


What is 'StyleGAN2 with ada '?

비트캠프 혁신 인공지능(서울) 4조

발표 및 팀장 : 전병준

팀원 : 강청순, 김현수, 황채연

2021.10.13

Contribution

NVIDIA labs의 2020년 10월에 발표된 논문

1.ADA 라는 augmentation 방법을 제시하여, 10배 적은 데이터로도 styleGan2와 대등한 성능을 보임.

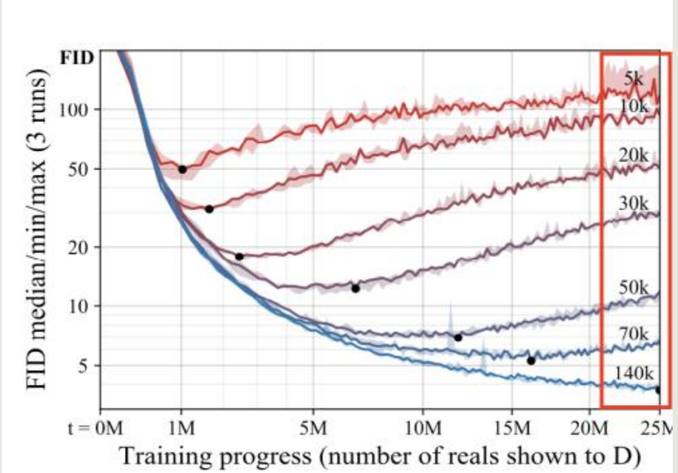
2. 적은 양의 데이터로 GAN 을 학습시키는 다양한 방법들을 비교, 분석하고 최적의 방법을 제시함.

Introduction

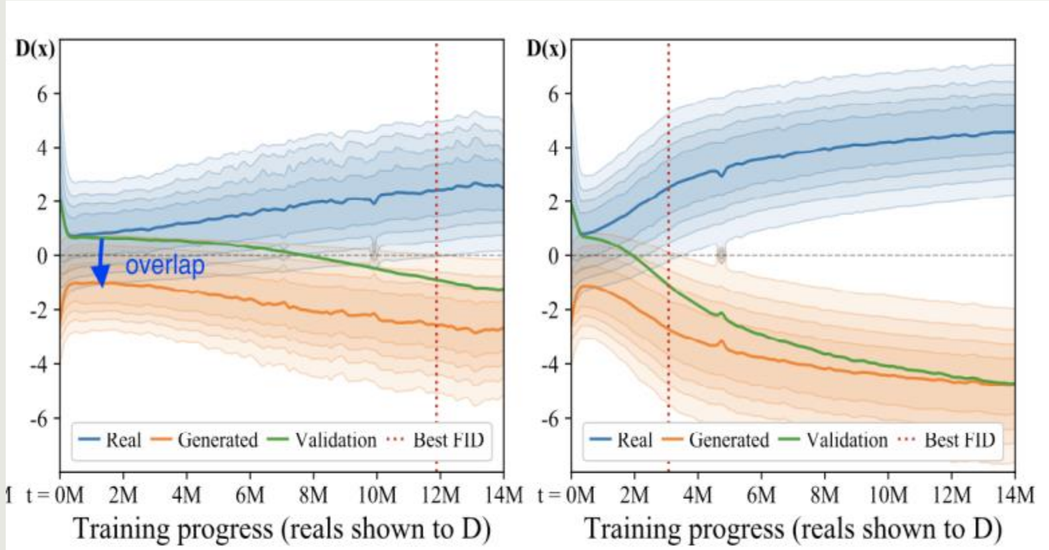
● Modern Gan 에서 좋은 결과를 얻으려면 100k ~ 1000k의 데이터가 필요

● 데이터가 적으면 Discriminator overfiting이 발생하여 Generator가 유의미한 feedback을 받을 수 없음

● 기존의 Data augmentation은 leaking 때문에 GAN에 적용할 수 없음.



(a) Convergence of FFHQ (256 × 256)



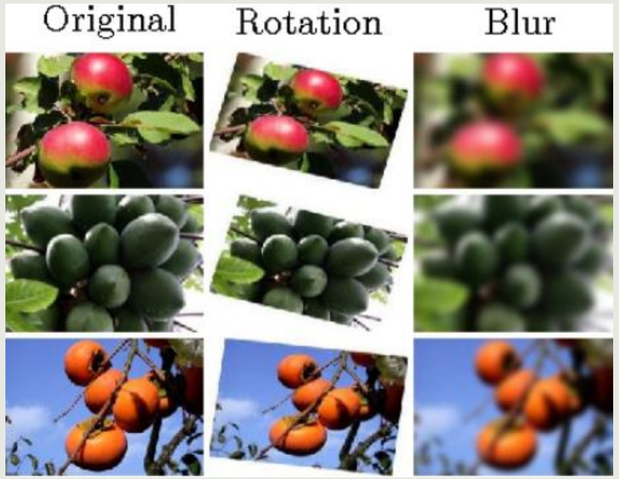
(b) Discriminator outputs, 50k (c) Discriminator outputs, 20k

① Designing augmentations that do not leak

Data Augmentation

-> 원본 이미지에 인위적인 변화를 준 이미지를 학습 데이터로 사용하는 것

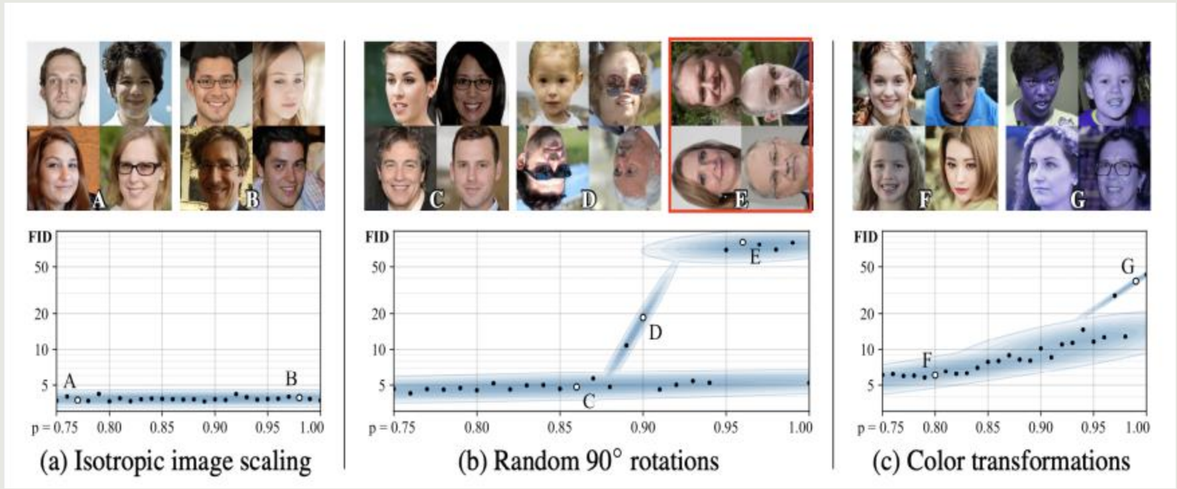
-> Overfiting의 문제를 해결하기 위한 방법 중 하나.



Leaking of Augmentation

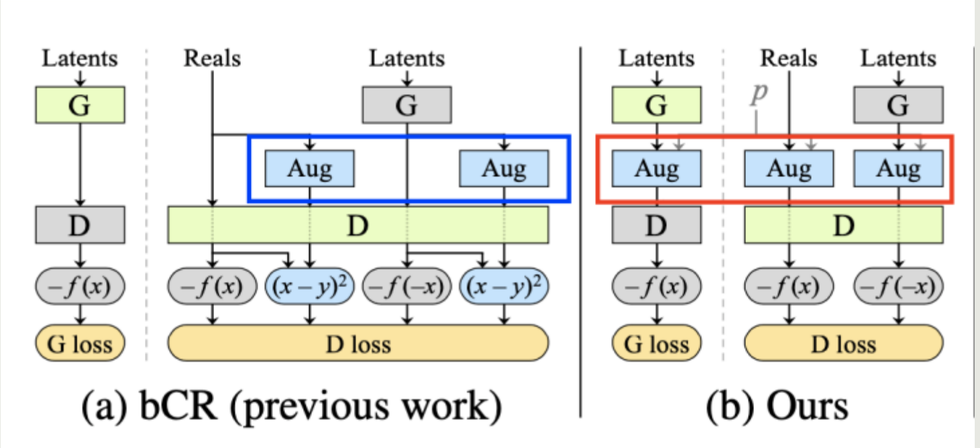
-> GAN에서 학습시킬 시, Augmentation을 포함한 이미지 생성

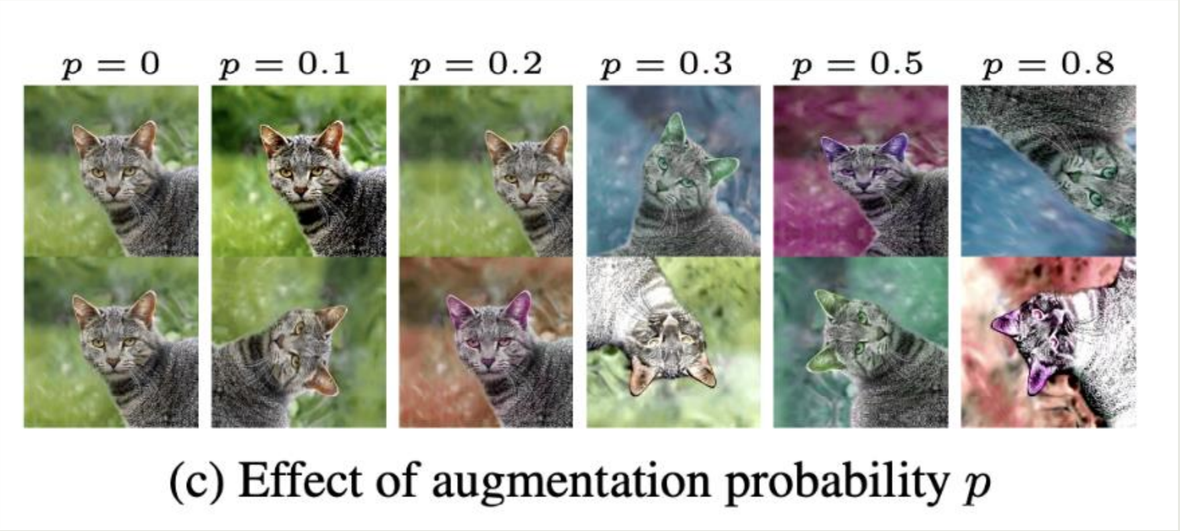
-> Augmentation은 단순히 오버피팅을 막아주려고 넣었지만, 잘못된 이미지가 생성되는 원인이 되게 함.



(a) Isotropic image scaling (b) Random 90° rotations (c) Color transformations

① Designing augmentations that do not leak





-> bCR의 저자는 (x-y)^2를 적용하여 augentation leakage 해결했다고 주장하지만 해결하지 못함

-> 해당 논문의 저자는 (x-y)^2 loss를 제거하고 Generating 하는 부분에 Augmentation을 추가

-> Discriminator와 Generator 모두 같은 augmentation을 진행하므로 Generator는 Augmentation을 제외한 feature만을 학습

ex) 90도 돌아가는 augmentation을 generator가 학습하더라도 discriminator가 90도 돌리는 안경으로 이미지를 보는 상황

② Adaptive discriminator augmentation

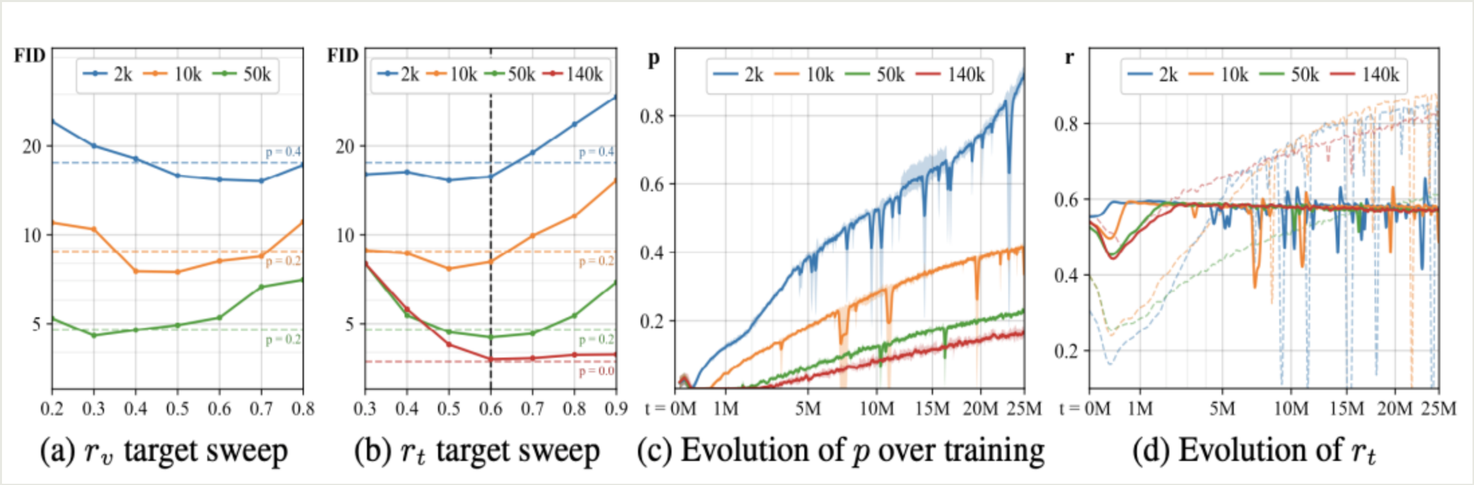
=〉 Augmentation을 무조건적으로 적용할 경우 오히려 성능 감소 가능
=〉 Overfiting 정도를 확인할 수 있는 Overfiting Measure 2가지 rv, rt

$$r_v = \frac{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{validation}}]}{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{generated}}]} \qquad r_t = \mathbb{E}[\text{sign}(D_{\text{train}})]$$

두 개의 지표 모두
0~1의 범위에서
1에 가까울 수록 Discriminator overfiting ⬆

ADA란?

target value를 0~1 사이의 임의의 값으로 정하고, 그 값을 기준으로 p값을 adaptive 하게 조절하는 것을 의미



(a) r_v target sweep (b) r_t target sweep (c) Evolution of p over training (d) Evolution of r_t

주요 코드

projector.py

```
@click.command()
@click.option('--network', 'network_pkl', help='Network pickle filename', required=True)
@click.option('--target', 'target_fname', help='Target image file to project to', required=True, metavar=
'FILE')
@click.option('--num-steps',              help='Number of optimization steps', type=int, default=1000,
show_default=True)
@click.option('--seed',                   help='Random seed', type=int, default=303, show_default=True)
@click.option('--save-video',             help='Save an mp4 video of optimization progress', type=bool,
default=True, show_default=True)
@click.option('--outdir',                 help='Where to save the output images', required=True, metavar='DIR'
)
def run_projection(
    network_pkl: str,
    target_fname: str,
    outdir: str,
    save_video: bool,
    seed: int,
    num_steps: int
):
```

```python
# Render debug output: optional video and projected image and W vector.
os.makedirs(outdir, exist_ok=True)
if save_video:
    video = imageio.get_writer(f'{outdir}/proj.mp4', mode='I', fps=10, codec='libx264', bitrate='16M')
    print (f'Saving optimization progress video "{outdir}/proj.mp4"')
    for projected_w in projected_w_steps:
        synth_image = G.synthesis(projected_w.unsqueeze(0), noise_mode='const')
        synth_image = (synth_image + 1) * (255/2)
        synth_image = synth_image.permute(0, 2, 3, 1).clamp(0, 255).to(torch.uint8)[0].cpu().numpy()
        video.append_data(np.concatenate([target_uint8, synth_image], axis=1))
    video.close()

# Save final projected frame and W vector.
target_pil.save(f'{outdir}/target.png')
projected_w = projected_w_steps[-1]
synth_image = G.synthesis(projected_w.unsqueeze(0), noise_mode='const')
synth_image = (synth_image + 1) * (255/2)
synth_image = synth_image.permute(0, 2, 3, 1).clamp(0, 255).to(torch.uint8)[0].cpu().numpy()
PIL.Image.fromarray(synth_image, 'RGB').save(f'{outdir}/proj.png')
np.savez(f'{outdir}/projected_w.npz', w=projected_w.unsqueeze(0).cpu().numpy())
```

```python
# Load networks.
print('Loading networks from "%s"...' % network_pkl)
device = torch.device('cuda')
with dnnlib.util.open_url(network_pkl) as fp:
    G = legacy.load_network_pkl(fp)['G_ema'].requires_grad_(False).to(device) # type: ignore

# Load target image.
target_pil = PIL.Image.open(target_fname).convert('RGB')
w, h = target_pil.size
s = min(w, h)
target_pil = target_pil.crop(((w - s) // 2, (h - s) // 2, (w + s) // 2, (h + s) // 2))
target_pil = target_pil.resize((G.img_resolution, G.img_resolution), PIL.Image.LANCZOS)
target_uint8 = np.array(target_pil, dtype=np.uint8)
```

적용 과정

① stylegan2 with ada train.py 실행

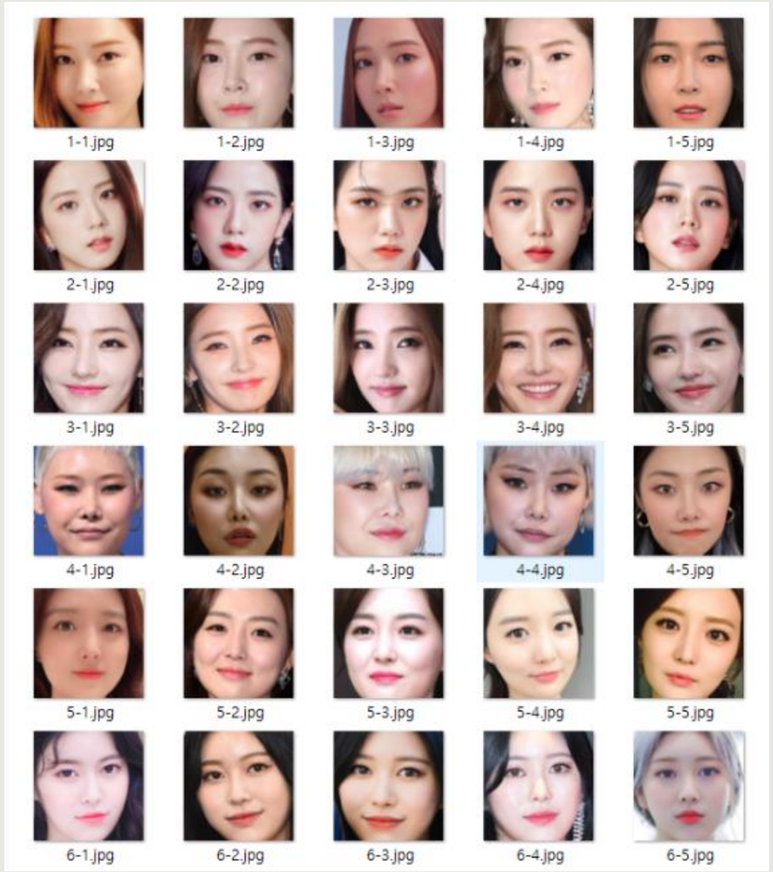
train.py를 실행하여 가중치가 저장된 pickle 파일 생성
-> 생성된 pickle 파일을 사용하여 해당 유형의 평균얼굴과
target으로 지정한 내 얼굴을 합성하여 새로운 평균얼굴 생성

② projector.py 실행

--network 에 train.py를 통해 생성한 pickle파일을 지정하고,
target image, outdir을 지정한다.

=> 내 얼굴과 해당유형의 평균 얼굴을 학습하여 새로운 평균얼굴 생성

최종 결과물



유형별 평균 얼굴

내 얼굴

새로운 평균 얼굴

Conclusion

1) 구동에 많은 리소스를 요구

:최소 구동 VRAM 12G를 요구하여 V100 24G 모델 2대와 RTX 3090 1대에서만 구동가능

2) 긴 훈련시간

:RTX 3090 기준 25000 epoch를 훈련하는데 ‘4일 반’의 긴 훈련시간이 필요함

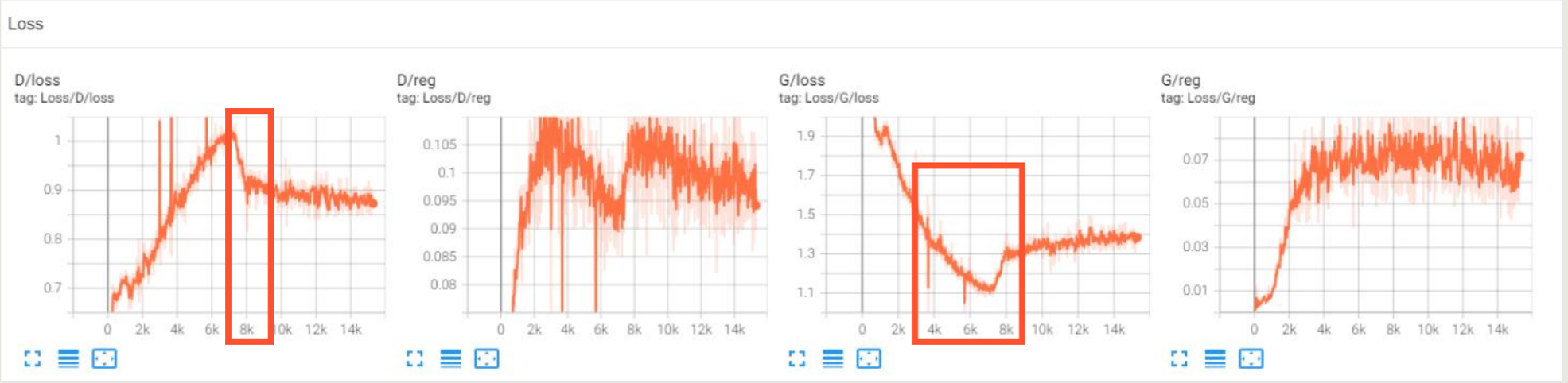
3) 일관적으로 생성되지 않는 pickle 파일

: epoch가 늘어나면서 input과 동일한 방향으로 평균유형 얼굴이 계속 생성되지 않고, 90도 또는 180도 회전하여 결과물이 산출되어 projector.py를 실행했을 때 성능이 떨어지는 문제가 발생한다. 랜덤하게 방향이 바뀌지 않고 동일하게 결과물이 생성될 수 있도록 보완할 예정이다.

Conclusion

4) 적합한 훈련량 탐색 어려움

epoch가 늘어나면서 성능이 무조건적으로 향상되는 것이 아니라, 오히려 주어진 25000 epoch를 모두 실행했을 때 overfiting이 발생하며 성능이 떨어짐을 발견하였다. Tensorboard를 활용하여 loss 값을 관찰하고 8000으로 전체 epoch를 실행하여 성능을 개선함.




Q & A