AUcGAN 구조
- Detail 부분 정리

DeepThinkers 장현웅

조사내용

1. Detail 단계 란 무엇인가?
2. Detail 단계를 좀 더 구체적으로 정의해보자
3. 코드에서 “Detail 단계” 는 어느 부분일까
4. 딥러닝 객체, 손실 함수, Configuration 연결, 데이터 흐름

1. Detail 단계란 무엇인가?

1. Detail 단계란 무엇인가?

- AU(표정 근육 움직임) 정보를 조건으로 사용해, GAN 기반 디코더(**D_detail**)가 **고해상도 displacement map**을 생성하고, 이를 통해 정밀한 normal map 및 texture map 복원.
- Coarse 단계는 FLAME 기반 기본 shape, expression, pose 추정
- Detail 단계는 이를 바탕으로 고해상도 Displacement/Normal맵 생성
- 이 과정에서 AU feature (**afn**)를 조건으로 사용해, 표정의 섬세한 디테일을 생성.

2. 최상위 모듈(클래스) 요약

- Trainer

학습 전반에 걸쳐서 학습과정을 조율하는 역할 + loss function 계산 역할

__init__ 내에서

1. config 학습모델 설정 불러오기
2. DECA모델 만들기
3. 데이터 증강 파이프라인 정의 **(문제있음)**
4. optimizer에 deca의 파라미터 등록

*configure_optimizers()*에서 **E_detail, D_detail, D만 등록**

optimizer에 포함되지 않으면 학습 불가

(+) Augumentation?

cfg에서 batch size, image size, uvmap size 등 가져오고

```
self.train_detail = self.cfg.train.train_detail        #False: coarse학습, 얼굴 모양, 표정, 포즈 등
# NOTE:***** Augumentation 파이프라인 정의
self.transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # 좌우 반전
    transforms.RandomRotation(90),      # NOTE:***** 90도 회전
    transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1),  # 색상 변화
    transforms.GaussianBlur(3),]) # # 가우시안 블러
# deca model
```

좌우반전 **/ 90도회전** / 조명채도색조변화 / 약간의 노이즈 흐림.
편의상 코드 복붙 진행한 것으로 보임

2. 최상위 모듈(클래스) 요약

```
# AU network 모듈도 학습시키려는 흔적이 있다.
self.optimizer_G = torch.optim.Adam(
                        list(self.deca.E_detail.parameters()) + ₩
                        list(self.deca.D_detail.parameters()),# + ₩
                        # list(self.deca.AUEncoder.parameters()),
                        # list(self.deca.AUNet.parameters()),
                        # list()
                        # list(self.deca.DAT.parameters()),
                        lr=0.000005,
                        betas=(0.5, 0.999))
# optimizer_G = torch.optim.Adam(self.deca.E_detail.parameters(), lr=0.0001, betas=(0.5, 0.999))

# STEP2. Discriminator Optimizer 설정.
self.optimizer_D = torch.optim.Adam(self.deca.D.parameters(), lr=0.0001, betas=(0.5, 0.999))
```

*configure_optimizers()*에서 **E_detail, D_detail, D만** 등록한 모습

2. 최상위 모듈(클래스) 요약

```
# STEP2. model dict 불러오기.
# model_dict: 6개 모듈의 state_dict 포함.
# E_flame, E_detail, D_detail, D, AUNet, AUEncoder
model_dict = self.deca.model_dict()
```

fit()에서

4. ***prepare_data()로 데이터 준비, load checkpoint()***로 DECA 내부 모델들의 학습된 파라미터 (state_dict) 가져오기; **E_flame, E_detail, D_detail, emoca, D, AUNet**의 파라미터 가져옴. (학습 안하는 모델도.) train 진행중이었다면 옵티마이저 상태 등도 가져옴

5. 손실함수 객체 생성. ※ 이게 loss 전부가 아님

IDMRF Loss, BCELoss, AU FeatureLoss 객체 생성

```
if self.train_detail:      #Detail단계 학습중. E_detail, D_detail, self.deca.D 학습됨.
    self.mrf_loss = lossfunc.IDMRFLoss()  #Markov Random Field 기반 손실. 디테일 복원시 국소적인 패턴 유지 강제
    self.adversarial_loss = nn.BCELoss()  # Binary Cross Entropy 손실 함수. GAN의 판별자 학습에 사용.
    #NOTUSE
    self.au_feature_loss=lossfunc.AU_Feature_Loss() # AU feature 추출 및 감정 분석을 위한 손실 함수.
```

2. 최상위 모듈(클래스) 요약

6. **training_step()**으로 Detail map 생성 (forward)

마지막에 손실 항목별 계산을 묶음. backward(), step()안함.
전체 손실 모니터링: all_loss.

7. **discriminator_step()**에서 **E_detail, D_detail, D 학습.**

(Encoder, Generator/Decoder, Discriminator) **D_loss G_loss**

```
for epoch in range(start_epoch, self.cfg.train.max_epochs):

    # for step, batch in enumerate(tqdm(self.train_dataloader, desc=f"Epoch: {epoch}/{self.cfg.train.max_epochs}")):
    for step in tqdm(range(iters_every_epoch), desc=f"Epoch[{epoch+1}/{self.cfg.train.max_epochs}]"):
        if epoch*iters_every_epoch + step < self.global_step:
            continue
        try:
            batch = next(self.train_iter)
        except:
            self.train_iter = iter(self.train_dataloader)
            batch = next(self.train_iter)
        losses, opdict, final_image,codedict = self.training_step(batch, step)
        losses['D_loss'], losses['G_loss'] = self.discriminator_step(self.transform(opdict['images']), opdict['images'], final_image, losses, codedict)
```

2. 최상위 모듈(클래스) 요약

- `E_detail`과 `D_detail`:
- 학습 함수: `Trainer.discriminator_step()`
- 위치:

```python
self.deca.E_detail.zero_grad()
self.deca.D_detail.zero_grad()
...
G_loss.backward(retain_graph=True)
self.optimizer_G.step()
```

- `D` (Discriminator):
- 학습 함수: `Trainer.discriminator_step()`
- 위치:

```python
self.optimizer_D.zero_grad()
D_loss.backward(retain_graph=True)
self.optimizer_D.step()
```

2.최상위 모듈 설명

- **DECA: loss 계산을 제외하고, 모든 reconstruction 과정수행**

1. 모든 Encoder/Decoder/Discriminator/Renderer 초기화: ***_create_model()***
2. AU-Net(AFG)까지 포함한 Encoding: ***encode()*** 에서 batch 단위 **이미지 텐서 -> detailcode**
3. Decoding, Coarse + Detail 복원 ***decode()***에서 **pose,exp,detailcode,afn**을 **D_detail** 넣어 -> **uv_z(displacement map)**
4. 렌더링(시각화) 단계에서 Normal map, texture map 생성 ***self.render*** **로 결과 렌더링**
5. 학습된 모델 파라미터 저장 / 로드 가능 ***model_dict()*** **함수로 state_dict 반환 (Trainer가 호출)**

3.DECA 클래스의 딥러닝 모델

코드 구조의 핵심

3.DECA 클래스의 딥러닝 모델

```
58  class DECA(nn.Module):
59      """
60      설명:
61      모델 구성 및 전처리-후처리의 모든 흐름을 담당하는 구조적 중심 클래스.
62      forward 경로에서 입력 이미지를 인코딩하고 디코딩하며,
63      AU 조건 기반 디테일 복원을 수행.
```

deca_EM_AU....py
line 58

- 모델 구성요소 컨테이너.
  - 인코더, 디코더, AUNet, FLAME, 렌더러 등 모든 하위모듈 인스턴스화
- 데이터흐름 정의
  - encode(),decode() 등 메서드로
    입력 이미지 -> latent -> output forward path 직접 수행
- 핵심 담당
  - 얼굴 표현 인코딩
  - AU기반 디테일 생성
  - Displacement, Normal, Texture 생성
  - forward시 reconstruction 결과 반환

loss 계산 안하는 것
제외하면, model 100%
담당

3.DECA 클래스의 딥러닝 모델

```
58  class DECA(nn.Module):
59      """
60      설명:
61      모델 구성 및 전처리-후처리의 모든 흐름을 담당하는 구조적 중심 클래스.
62      forward 경로에서 입력 이미지를 인코딩하고 디코딩하며,
63      AU 조건 기반 디테일 복원을 수행.
```

deca_EM_AU….py
line 58

- Detail에 도움이 되는 부분만 구체적으로 설명
- *나머지 부분은 문서 "핵심 객체 리스트" 참고*

3.DECA 클래스의 딥러닝 모델

(self.__)

Coarse 모듈 객체(자료형)

- E_flame(ResnetEncoder)
- flame(FLAME)
- flametex(FLAMETex)
- emoca(nn.Module)
- render(SRenderY)

Detail 모듈

- **E_detail(Encoder)**
- **D_detail(Generator3)**
- **AUNet(AFG)**
- **D(ConditionalDiscriminator)**

*비활성화 모듈

AUEncoder(AUEncoder)

TDC(TDC)

3.DECA 클래스의 딥러닝 모델

- **E_detail(Encoder) : DCGAN,StyleGAN유사**
  - 8-layer CNN.
  - 이미지에서 미세 피부 디테일 인코딩 **(예상)**
  - (B,3,256,256) -> (B,128,1,1).
  - input: image / output: **detailcode**
- 특이사항:
  - stride 2.
  - norm: Batch 독립적으로 스타일 정규화.
    - 표정, texture encode 유리
  - LeakyReLU
  - 내부적으로 채널수를 크게 확장했다 줄이는 구조 **(유사 코드 조사중)**

```
self.E_detail = Encoder().to(self.device)
```

deca_EM_AU....py
DECA.create_model()
line 211

연산	출력 크기 (H×W 기준)	채널 수
`F.interpolate(x, size=256)`	256×256	3
`conv1` + `ReLU`	128×128	64
`conv2` → `norm2` → `LeakyReLU`	64×64	128
`conv3` → `norm3` → `LeakyReLU`	32×32	256
`conv4` → `norm4` → `LeakyReLU`	16×16	512
`conv5` → `norm5` → `LeakyReLU`	8×8	1024
`conv6` → `norm6` → `LeakyReLU`	4×4	512
`conv7` → `norm7` → `LeakyReLU`	2×2	256
`conv8` → `LeakyReLU`	1×1	128

3.DECA 클래스의 딥러닝 모델

- **E_detail(Encoder)**

단순한 컨벌루션 구조로,
출력 **detailcode** (z)는 Generator에서
이미지를 생성해내는
style latent vector값으로 작용.

GAN latent code는 당연하지만
UV 공간의 픽셀값
(주름깊이, 거침)...을 직접 표현하지 않음.
입력이미지의 세부 시각특징들을
CNN 통과하며 고차원 표현으로 압축한것.

연산	출력 크기 (H×W 기준)	채널 수
`F.interpolate(x, size=256)`	256×256	3
`conv1` + `ReLU`	128×128	64
`conv2` → `norm2` → `LeakyReLU`	64×64	128
`conv3` → `norm3` → `LeakyReLU`	32×32	256
`conv4` → `norm4` → `LeakyReLU`	16×16	512
`conv5` → `norm5` → `LeakyReLU`	8×8	1024
`conv6` → `norm6` → `LeakyReLU`	4×4	512
`conv7` → `norm7` → `LeakyReLU`	2×2	256
`conv8` → `LeakyReLU`	1×1	128

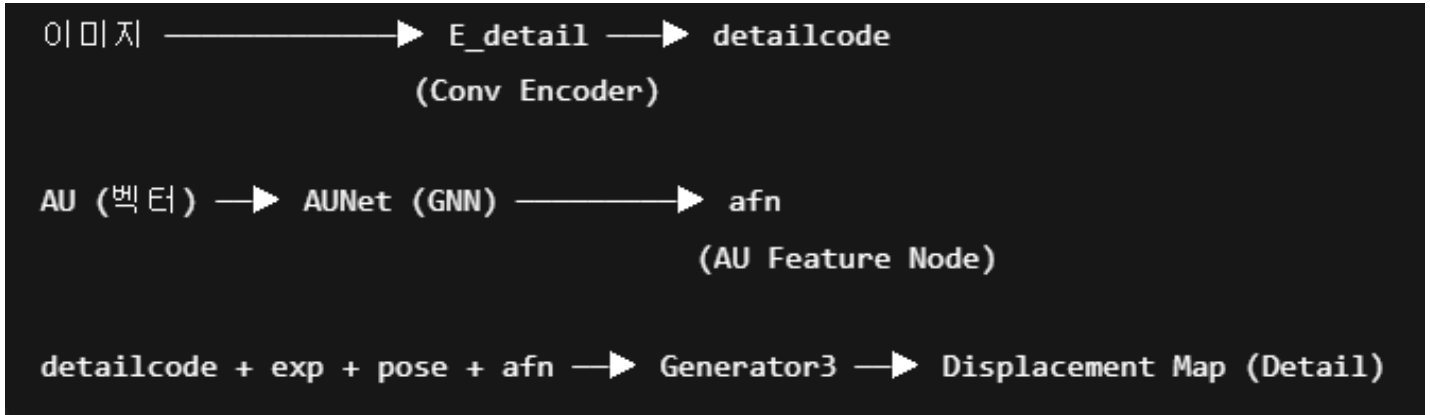
3.DECA 클래스의 딥러닝 모델

- **AUNet(AFG)** : MEFARG 모델 기반.
- AU classification을 위한 Graph 기반 Attention Network.
- 학습되지 않으며 고정된 사전학습 모델로 사용됨.
- 입력: image
- 출력: x(AU 클래스별 활성도), afn

(+)afn?

- AU Feature Network.
- MEFARG(AUNet) 에서 반환됨.
- shape: [B,27,512]
- B: batch shape
- 27: 주요 AU 클래스 수 (0~27)
- 512: 각 AU 클래스에 대해 추출된 512차원 피쳐벡터.

```
x, afn, _ = self.AUNet(images, use_gnn=True)
```



3.DECA 클래스의 딥러닝 모델

- **D_detail(Generator3)**
  - 입력:
    - pose, exp, afn (condition)
    - detailcode (latent code)
  - 출력: **uv_z**(displacement map)

- 특이사항
  - **noise에 pose_jaw, exp**
  - 여러 버전 개발 중이었음
  - 이름 문제 (8월에 수정 예정)

```python
uv_z = Generator3(
    concat([pose_jaw, exp, detailcode]),
    afn
)
```

Sudo code

```python
def forward(self, noise, cond):
    cond = cond.view(cond.size(0), -1, 1, 1)
    cond = self.lin(cond)
    out = torch.cat([noise, cond], dim=1)
    # out = out.view(out.shape[0], 222, 1, 1)
    img = self.model(out)
    return img
```

```python
self.D_detail = Generator3().to(self.device)
```

deca_EM_AU....py
DECA.create_model()
line 231

(+) noise 부분에 pose_jaw, exp가 concat 되어 전달되어도 되는가?

- condition으로 들어가야...?

전통적 GAN에서는 생성기에 랜덤 잠재벡터 z를 입력으로 사용.

z는 학습 초반엔 특별한 의미를 갖지 X (명시적으로 "포즈""표정)

(일부 연구에서는 그 z값 연산이 의미를 갖는단걸 밝혔으나, 처음부터 그렇게 나타내도록 설계한건 아님)

조건부 GAN에서는 아예 원하는 속성을 입력으로 명시적으로 제공해 생성기가 그 조건에 맞는 이미지를 생성하도록 훈련.

애초에 noise라는 말이 부적절. latent vector와 afn 정도가 적절. refactoring 필요

3.DECA 클래스의 딥러닝 모델

적절한가?

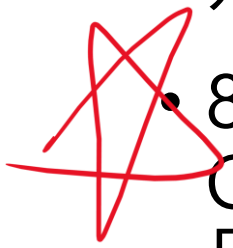
- **D_detail(Generator3)**

1. cond = afn 이 [B, 27, 512] -> [B,13824] 벡터로 변형 후 spatial resolution이 H=1 X W=1 되게 reshape
2. 1x1 convolution 통해 13824 -> 41채널 축소
3. noise + cond (채널방향)
4. 총 8번 upsampling & conv 블록 거쳐 최종적으로 256 X 256 1채널 이미지 생성.
5. Tanh()로 -1~1 discplacement map 완성

```
def forward(self, noise, cond):
    cond = cond.view(cond.size(0), -1, 1, 1)
    cond = self.lin(cond)
    out = torch.cat([noise, cond], dim=1)
    # out = out.view(out.shape[0], 222, 1, 1)
    img = self.model(out)
    return img
```

(+)기존의 어떤 코드와 유사한가?
- 비교대조 결과

- cGAN 구조와는 다소 유사하나 차이점이 있었음
- 동주님과의 contact 없이, 어떤 기존 GAN 코드들과 유사한지 일일히 확인하는 것은 어렵다는 판단
- GPT 심층 리서치 기능을 사용함
- 그 결과, E_decoder, D_decoder구조는 **StyleGAN, DCGAN**과 유사하다는 결과를 얻음 (코드 얻는 중)
- 8월 실험주간 진행 전 기존 논문들 설명과 함께, GPT 심층 리서치 기능을 사용해 명확하게 보고할 예정



3.DECA 클래스의 딥러닝 모델

27 → 3x9 (X)

- D(ConditionalDiscriminator)
- 이상한 명칭
- 올바르지 않은 AU concat 방법
- 타 논문들의 방법론을 통해 개선할 대상

```
def forward(self, img, condition):
    condition = condition.squeeze(0)
    # print(condition.shape)
    condition = condition.view(condition.size(0),3,9,512)
    condition = F.interpolate(condition, size=(224,224), mode='bilinear', align_corners=False)

    # print(condition.shape)
    # condition = condition.view(condition.size(0), 1, 1, 1).repeat(1, 1, img.size(2), img.size(3))
    d_in = torch.cat((img, condition), 1)
    return self.model(d_in)
```

그 외 사항

더 구체적인 각각의 method 동작 과정은 뒤로 미뤘음.

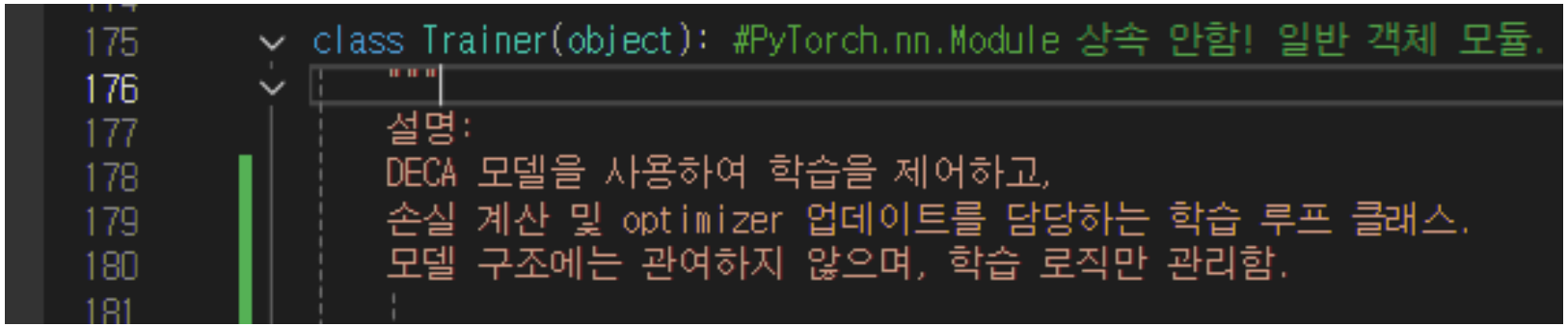
아직 각 클래스 method들의 알고리즘 레벨의 타당성은 다 검증하지 못한 상태.

실험을 진행하면서 지속적인 팀원과의 교류가 필요함.

조사 결과 렌더러의 경우 사용 렌더러마다 쉐이딩 등의 방법이 다르므로 퀄리티가 다르지만, 대부분 **.obj, .jpg, .mtl 내보내기 방법을 지원**하므로, 렌더러 선택은 부가적인 문제.

4. Trainer

4. Trainer(큰 내용 없음)



Trainer 파일 line 175

- 훈련 엔진(Training Controller)
  - fit(), train_detail() 등 학습 루프, 학습 시나리오관리
- **손실 계산 및 최적화**
  - **detail/coarse 단계에 따라 손실함수계산**
  - loss 별 가중치 설정(cfg 기반) 포함
- 핵심 책임
  - 학습 반복, loss 로깅
  - optimizer step (파라미터 업데이트), scheduler(학습률)제어
  - config에 따라 coarse/detail
  - 실험 로깅, 체크포인트 저장 등

4. Trainer

- Trainer의 핵심 분기 구조 in Trainer.training_step()

self.train_detail = true시 detail 단계

!= true 시 coarse 단계

현재 cfg 설정:

train_detail = true

(coarse코드 bypass)

coarse 코드는 있으나 수정 필요

(어차피 freeze시키니 안 씀)

파일: `trainer_EM_AU_enc_dec_origT_aucGAN.py`

함수: `Trainer.training_step()`

python 복사 편집

```python
if self.train_detail:
    # ✅ Detail 단계일 때 실행되는 손실 계산
    losses['photo_detail'] = ...
    losses['z_reg'] = ...
    losses['z_diff'] = ...
    ...
else:
    # ✅ Coarse 단계일 때 실행되는 손실 계산
    losses['photo'] = ...
    losses['landmark'] = ...
    losses['shape_reg'] = ...
    ...
```

5. Loss function classes

어떤 loss function을 사용하는가?

5. Loss function classes

- **loss functions** in lossfunc_AU_enc_dec…file
- 다양한 목적, 다양한 버전의 loss function 파일에 정의
  - GAN 관련
  - 정규화/분포 관련
  - Shading 관련
  - Albedo 관련
  - Landmark 관련
  - AU Feature 관련
  - Perceptual Loss
  - Gradient shapness관련
- *전체 lossfunction 쓰임은 아직 검토중, 필요시 별도 문서 참고*

5. Loss function classes

- **loss functions** in lossfunc_AU_enc_dec...file
- **training_step(): 5개의 loss function 값을 계산해서**
- **all loss에서**

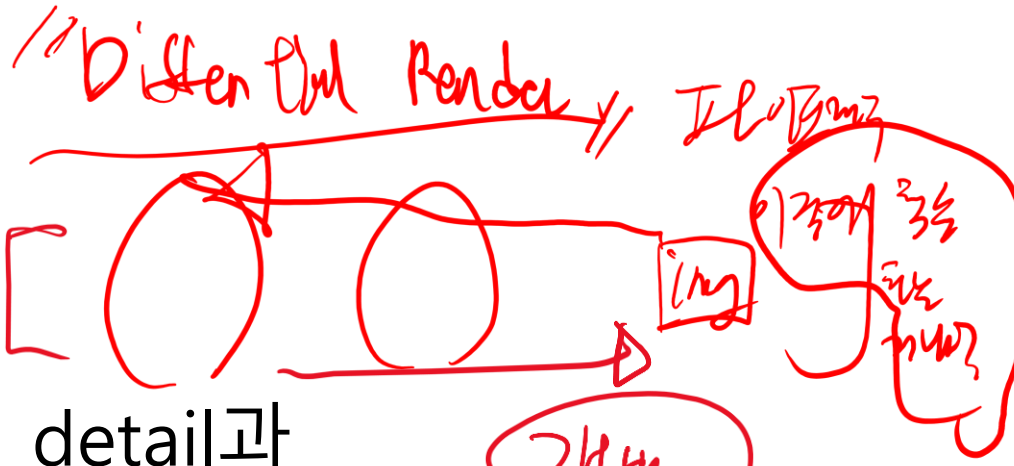
```
#########################################################
    all_loss = 0.
    losses_key = losses.keys()
    for key in losses_key:
        all_loss = all_loss + losses[key]
    losses['all_loss'] = all_loss
    return losses, opdict, final_image, codedict
```

5. Loss function classes

Trainer.training_step()에서 all_loss

이름	관련 함수	설명
`photo_detail`	L1 Loss	`uv_texture_patch`와 `uv_texture_gt_patch` 간 L1 손실
`z_reg`	`torch.mean(abs(z))`	변위맵(Displacement map)의 크기를 작게 유지
`z_diff`	`shading_smooth_loss`	디테일 shading의 gradient smoothness 유지
`z_sym`	L1 symmetry loss	좌우 대칭성 유지 손실
`au_feature_loss`	`BCEWithLogitsLoss`	AU 추정 결과의 일관성을 강제하는 손실
`photo_detail_mrf`	`IDMRFLoss` (선택적으로 사용됨)	MRF 기반 perceptual style 손실 (주석 처리된 버전에서 사용됨)

5. Loss function classes



- **optimizer_G : E_detail + D_detail용.**

이미지로부터 Detail latent를 추출하는 E_detail과 latent로부터 displacement map을 생성하는 D_detail을 엮음.

```
self.deca.E_detail.zero_grad()
self.deca.D_detail.zero_grad()
output = self.deca.D(final_image, cond_img)
real_target = torch.full_like(output, real_label, dtype=torch.float)
G_loss = self.adversarial_loss(output, real_target)
content_loss = losses['photo_detail'] + losses['z_reg'] + losses['z_diff'] + losses['z_sym'] + losses['au_feature_loss']
G_loss = G_loss + content_loss
G_loss.backward(retain_graph=True)
self.optimizer_G.step()

return D_loss, G_loss
```

$$\mathcal{L}_{\text{total}} = \lambda_{\text{photo}} \cdot \mathcal{L}_{\text{photo}} + \lambda_{\text{au}} \cdot \mathcal{L}_{\text{AU}} + \lambda_{\text{z-reg}} \cdot \mathcal{L}_{\text{z-reg}} + \lambda_{\text{z-diff}} \cdot \mathcal{L}_{\text{z-diff}} + \lambda_{\text{adv}} \cdot \mathcal{L}_{\text{adv}}$$

5. Loss function classes

- **optimizer_D : D 용.**

Discriminator 전용.

오직 adversarial loss만 포함하고 있음

```
real_label = 1.0
fake_label = 0.0
self.optimizer_D.zero_grad()  # zero_grad()로 optimizer의 기울기를 초기화!!!
cond_img = codedict['afn']
# Conditional Discriminator를 사용하여 진짜 이미지에 대한 예측값을 계산.
disc_real = self.deca.D(images_gt, cond_img)
real_target = torch.full_like(disc_real, 0.9, dtype=torch.float)
d_loss_real = self.adversarial_loss(disc_real, real_target)
# d_loss_real.backward(retain_graph=True)

disc_fake = self.deca.D(final_image, cond_img)
fake_target = torch.full_like(disc_fake, 0.1, dtype=torch.float)
d_loss_fake = self.adversarial_loss(disc_fake, fake_target)
D_loss = d_loss_real + d_loss_fake  # Discriminator 손실 계산
D_loss.backward(retain_graph=True)  # 그다음에 backward() 호출
self.optimizer_D.step()  # 마지막으로 optimizer_D.step() 호출하여 파라미터 업데이트
```

cfg

```
15                cfg = CN()
```

decalib/utils/config.py

- 자료형: YAML 기반 Namespace (yacs.config.CfgNode)
- 전체 프로젝트 설정(학습에 필요한 모든 하이퍼파라미터, 경로, 모델 설정값) 저장하는 구조화된 객체.

- `get_cfg_defaults()` : 기본 cfg 복제본 리턴
- `update_cfg(cfg, cfg_file)` : `.yml` 파일과 병합
- `parse_args(cfg_name)` : CLI 인자 포함 파싱

cfg

```
cfg
├── deca_dir
├── device / device_id
├── pretrained_modelpath
├── output_dir
├── rasterizer_type
├── model          👈 얼굴 모델 관련 설정
├── dataset        👈 데이터셋 설정
├── train          👈 학습 파라미터
└── loss           👈 손실함수 파라미터
```

`cfg.model` : FLAME, 디테일 등 모델 설정

- `n_shape`, `n_tex`, `n_exp`, `n_detail`, `max_z`
- 텍스처 사용 여부 (`use_tex`)
- `jaw_type` : `aa` 또는 `euler`

`cfg.dataset` : 데이터 설정

- `training_data`, `image_size`, `batch_size`, `scale_min/max`, `isSingle`

`cfg.train` : 학습 설정

- `train_detail` : coarse / detail 선택
- `max_epochs`, `log_dir`, `log_steps`, `checkpoint_steps`, `resume`

`cfg.loss` : 손실 항목 가중치

- landmark, photo, ID, detail 관련
- `AU_feature`, `mrf`, `photo_D`, `reg_sym`, `reg_z` 등 다양한 가중치

cfg

- detail 관련된 cfg의 핵심 설정값들

`train.train_detail`	**Detail 학습 활성화 여부.** `True` 이면 detail 단계 학습 수행
`model.n_detail`	Detail latent vector 차원 수 (ex. 128)
`model.max_z`	Displacement map 생성 시 최대 변화 크기 (scale factor)
`loss.photo_D`	Photometric loss 가중치 (detail UV texture와 GT 비교)
`loss.reg_z`	Displacement 값 정규화 (z값 크기 제한용 loss)
`loss.reg_sym`	좌우 대칭 보존 loss 가중치 (detail map이 비대칭 되지 않게)

자세한 config는 "Detail관련 핵심 설정값들 요약" 문서로 대체

Detail 단계 파이프라인을 그려보자

Input Image
↓
E_detail (Encoder)
↓
detailcode
+ expcode
+ jaw_pose (pose[:,3:])
+ afn (AU feature vector)
↓

D_detail (Generator3)
↓
Displacement map (uv_z)
↓
Displacement2Normal
→ Normal map
↓
Shading map 생성
↓
Albedo (from FLAMETex) × Shading
↓
Final texture → rendered detail image

더 알아볼 것 정리

detail 부분이랑 coarse 부분이랑 모듈화를 시키자.
오버로딩이 될 것이다.

안 쓴 코드에 대해서 좀 더 조사하고,
동주님이 수정한 부분을 정확히 하여 뭐가 더 나은지 체크해보자

알고리즘 수준에서의 검토 : 세부적인 loss function 등도 코드를 맞게 짰나?

Albedo map, normal map 구성과정 구체화