AU_Sequence 분석

분석할 프로그램

분석할 레거시 코드

- AU_sequence/
  main_AU_video _OnlyE.py
  - 최종수정일자 24년 3월 중순
  - 목표:
    MEAD_ Dataset_25,
    Aff-wild2 dataset으로부터
    ViT 트랜스포머를 학습시키자.
- AU_sequenceTest는 demo
  (추론 파일) 만 포함, 코드 비교중
- (이후의 코드부터는
  EMOCA 사용 흔적 있음.)

분석할 프로그램

- 다양한 경로로의 입력
  - **cfg :** deca_pretrain_video_OnlyE2.yml
  - Dataset 설정: **MEAD, Aff-Wild2**
  - Pretrained model : **GANE 기반**

Fig 2. where's cfg?
main_AU_video_OnlyE.py
line 84

```
# STEP B. set cfg file to use
cfg = parse_args(cfg_name='configs/release_version/deca_pretrain_video_onlyE2.yml')
# os.environ["CUDA_VISIBLE_DEVICES"] = "2,3"
# cfg = parse_args(cfg_name='configs/release_version/deca_detail.yml')
```

Fig 3. what Dataset?
build_datasets_video,py
line 32,33

```
self.source1 = '/media/cine/de6afd1d-c444-4d43-a787-079519ace719/MEAD_Dataset_25/masks/'
self.source2 = '/media/cine/First/Aff-wild2/masks/'
```

Fig 4. output directory
& pretrained model weight?
deca_pretrain_video_OnlyE2.yml
Line 7~9

```
7 output_dir: "/media/cine/First/HJCode2/AUSequence/Training1_videoC_OnlyE_AULoss/pretrain1"
8 pretrained_modelpath: ''
9 #model_path_HJ: '/home/cine/Documents/HJCode/GANE_code/Training/testGATE30/model.tar'
```

분석할 프로그램

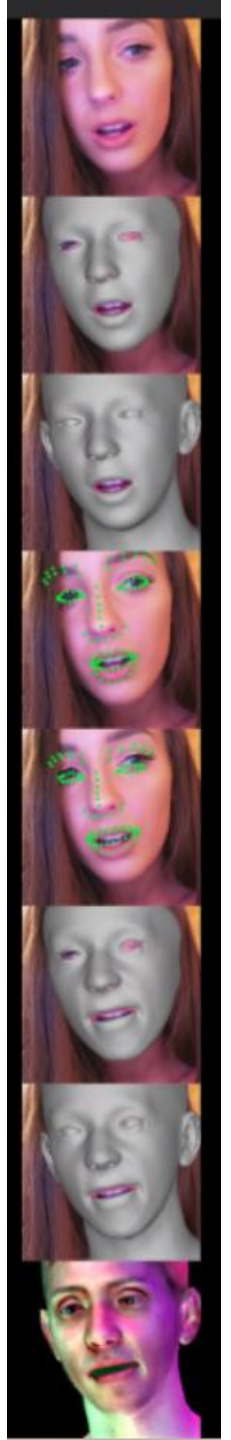
- 결과 출력
  - Log
  - Model 가중치
  - 학습결과 visualization

- 최종 실험일자 3월 중순

저장주기 20회

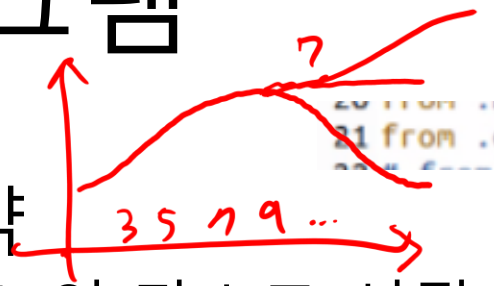
Q: 과연 동일한 코드, 환경에는 똑같이 돌아갈까?

1. randomness ☆ 다른 batch로 다른 순열.
2. 우연히 가장 잘된 실험결과,



분석할 프로그램

Fig 5. where do we build dataset?
trainer_Video_OnlyExpress.py




```
21 from .datasets import build_datasets_video as build_datasets
```

frame증가 실험

- 데이터셋 처리 요약
  - **MEAD와 Aff-Wild2**의 마스크 시퀀스 폴더를 섞어 샘플을 만들고,
  - 각 샘플에서 시간순 **연속 3프레임을** 뽑아
  - 정규화된 이미지(3×224×224), 랜드마크(선택된 촘촘 점 포함), 3채널 마스크(0 = 쓸데없는 부분/1/2 = 중요한부분) 로 묶어
  - 텐서 딕셔너리 형태로 모델에 넘긴다.

Q: 3프레임만 하는게, 유리할까? – 정보 예측

https://www.notion.so/about-dataset-25ce8369193380cda21fcce1a0c86a26?source=copy_link

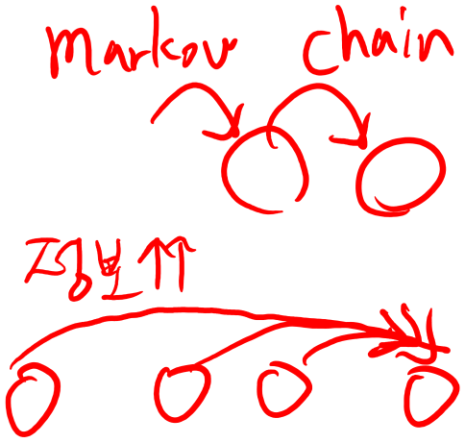
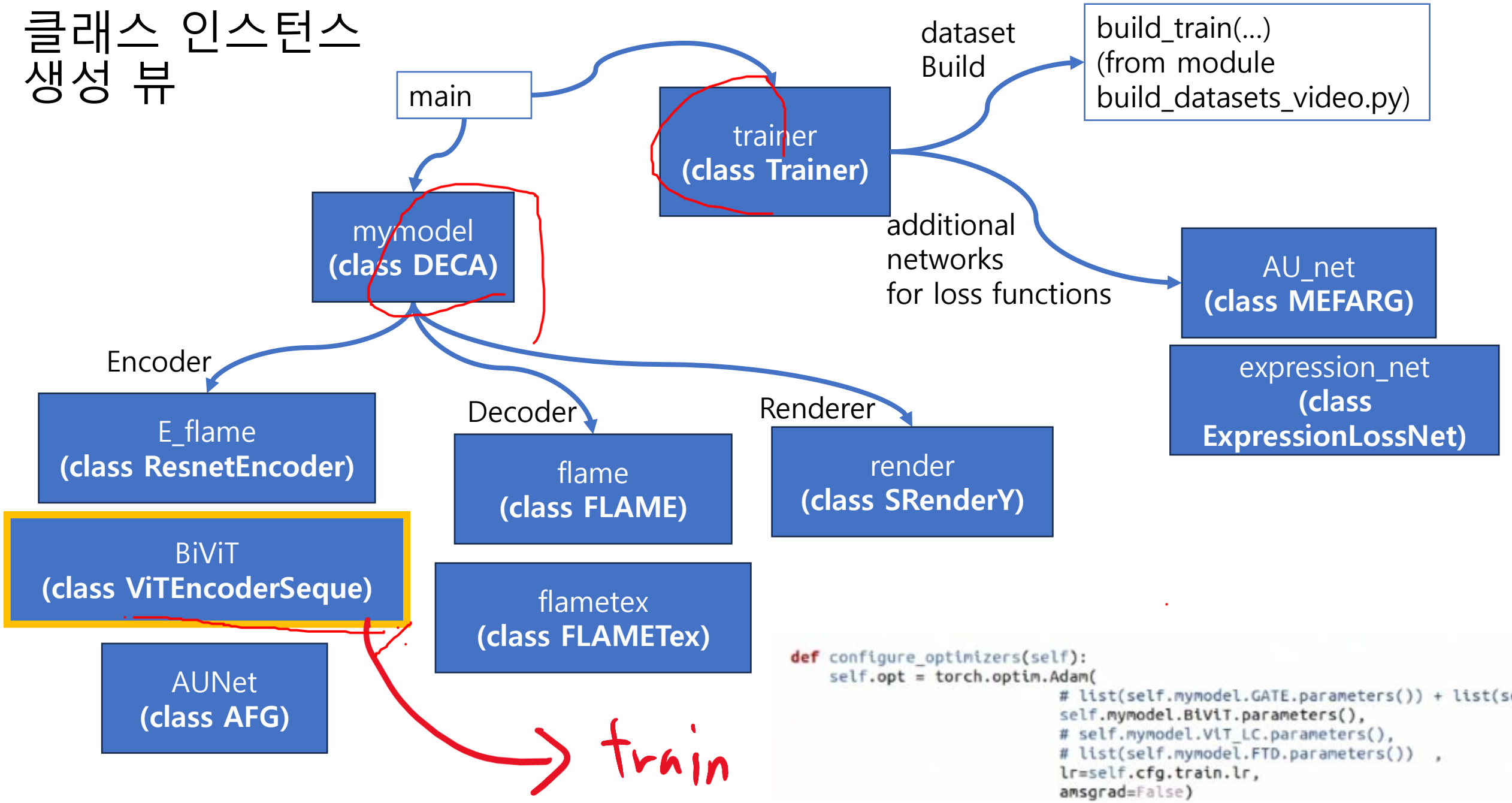


Fig 6. dataset src folder

```
32        self.source1 = '/media/cine/de6afd1d-c444-4d43-a787-079519ace719/MEAD_Dataset_25/masks/'
33        self.source2 = '/media/cine/First/Aff-wild2/masks/'
```

클래스 인스턴스별 설명

클래스 인스턴스 생성 뷰




```
def configure_optimizers(self):
    self.opt = torch.optim.Adam(
                                # list(self.mymodel.GATE.parameters()) + list(se
                                self.mymodel.BiViT.parameters(),
                                # self.mymodel.ViT_LC.parameters(),
                                # list(self.mymodel.FTD.parameters())  ,
                                lr=self.cfg.train.lr,
                                amsgrad=False)
```

클래스 인스턴스 별 간단설명

mymodel(**class DECA**)

입력 이미지 시퀀스를 3D 파라미터, 메쉬로 변환

- 인코더
  - E_flame: ResNet 인코더. 입력 이미지를 3D FLAME 파라미터 변환
  - BiViT: Vision Transformer 기반 인코더. **유일 학습 가능.**
    AUNet 추출 AU 특성 – E_flame 추출 글로벌 특성 결합해
    표정, 조명 파라미터 추정
  - AUNet: AU 인코더. AFG 부분만 사용
- 디코더
  - FLAME: 3D 얼굴 모델. shape,expression,pose param
    -> vertex, landmark
  - FLAMETex: (텍스쳐 생성 옵션 켜질때) 얼굴 텍스쳐 생성
- 렌더러
  - SRenderY: 예측된 mesh vertex, 텍스쳐, 조명 이용해 2D 이미지 재현

클래스 인스턴스 별 간단 설명

trainer(**class Trainer**)

학습 담당 클래스. DECA 모델을 받아 손실을 계산하고, 옵티마이저를 구성.

- 학습을 위해 추가 사용되는 네트워크
  - expression_net: 표정 손실 계산. 대상 이미지 – 렌더링 이미지 간.
  - AU_net: AU 예측 손실 계산. MEFARG 모델 사용해, 렌더링 이미지와 실제 이미지 간 AU 분포 비교.

기반 코드의 사용 분석

+ 코드 확장 예정

DECA

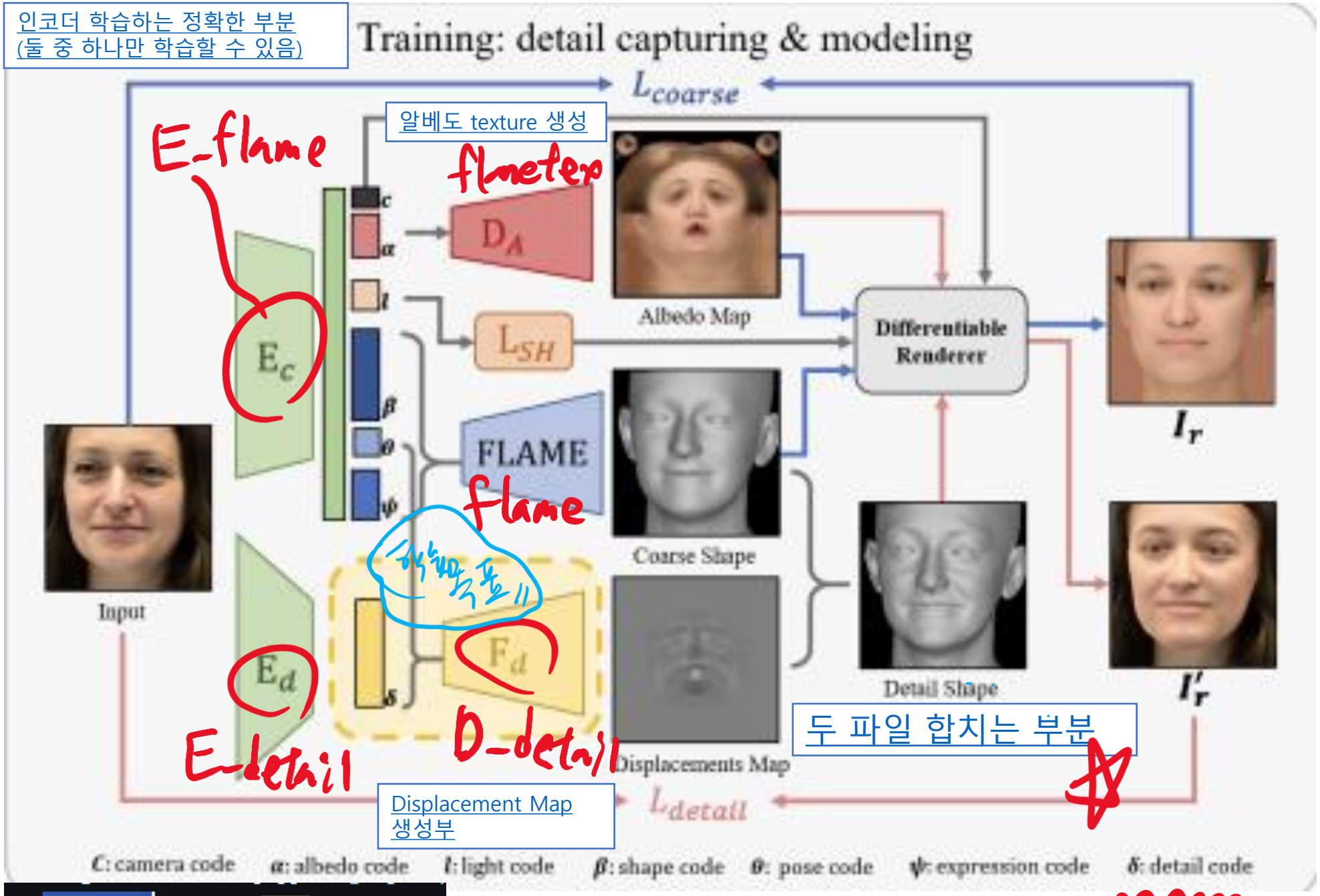
학습:

설정을 바꿔

2번씩 함.

[DECA readme]

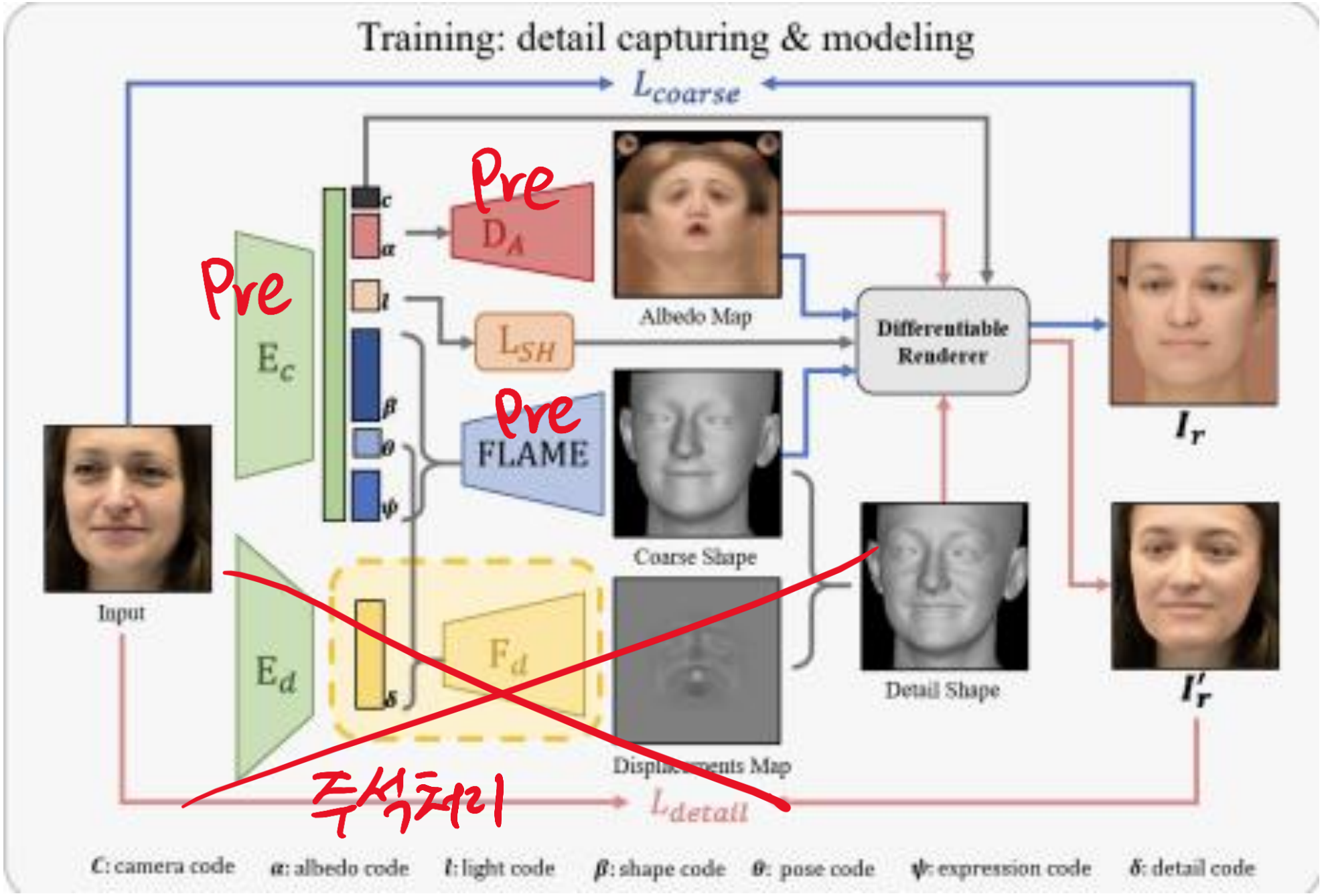
인코더 학습하는 정확한 부분
(둘 중 하나만 학습할 수 있음)




```
codedict = self.encode(images)
```

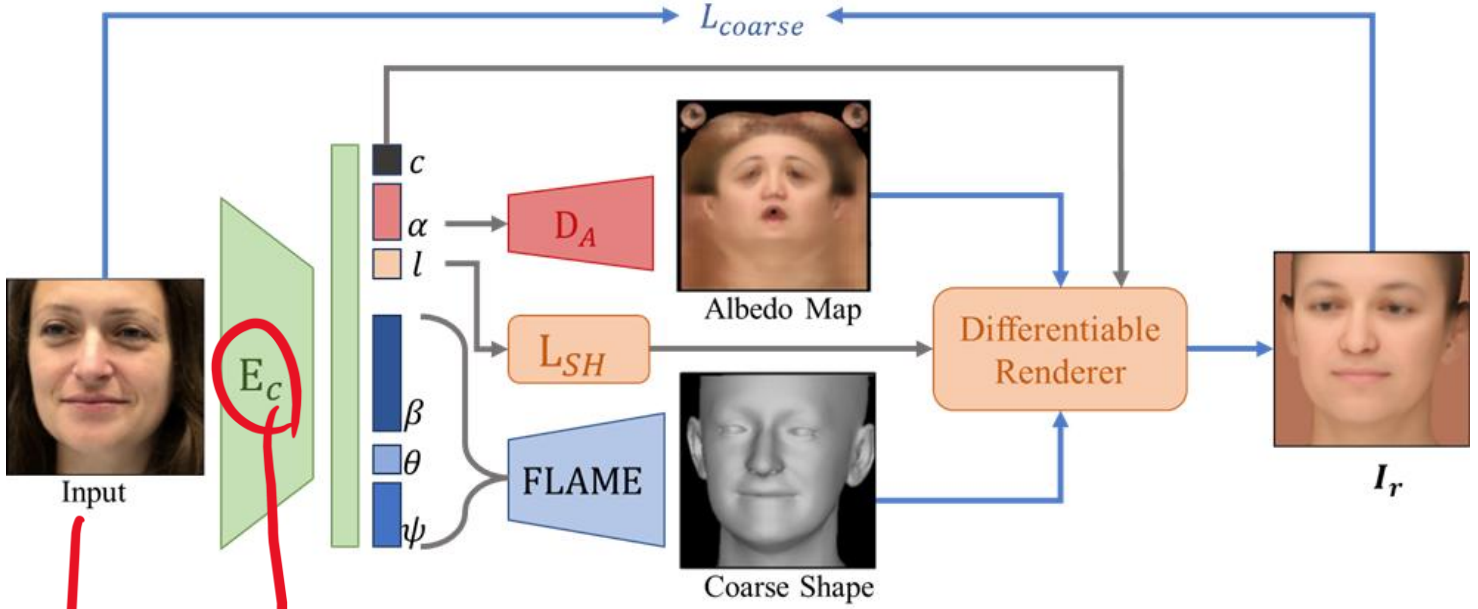
this code

① "Pre": pretrained



this code

② DECA coarse + BiViT (ViTEncoderSeq) + AUNet (AFG)



global_feature

AFG → afn

BiViT → parameter_ours

```
with torch.no_grad():
    parameters_224, global_feature = self.E_flame(images_224)
```

```
parameters_ours = self.BiViT(afn, global_feature)
```

```
with torch.no_grad():
    parameters_224, global_feature = self.E_flame(images_224)
    _, afn = self.AUNet(images_224, use_gnn=False)
# feats = torch.concat([afn, global_feature], dim=1)
```

this code

③

code slice

```
codedict_our = self.decompose_code_part(parameters_ours, self.param_dict_OnlyE)
codedict = self.decompose_code(parameters_224, self.param_dict)

codedict['images'] = images_224[1:2]
codedict_our['images'] = images_224[1:2]
codedict_our['shape'] = codedict['shape']
# codedict_our['tex'] = codedict['tex']
codedict_our['cam'] = codedict['cam']
codedict_our['pose'] = deepcopy(codedict['pose'])
# codedict_our['pose'][:,3:]=codedict_our['jaw_pose']
```

E_flame의 FLAME param vector (float)

codedict: 기존 6개 param

성능비교

codedict_our:

```
''' Convert a flattened parameter vector to a dictionary of parameters
code_dict.keys() = ['shape', 'tex', 'exp', 'pose', 'cam', 'light']
'''
```

codedict 3개값 copy + 시퀀스로 보정 학습 파라메터

```
# return code_list
''' Convert a flattened parameter vector to a dictionary of parameters
code_dict.keys() = [ 'tex', 'exp', 'jaw_pose', 'light']
'''
```

loss function

Loss function

https://github.com/yfeng95/DECA/blob/a11554ae2a2b0f3998cf1fa94dd4db03babb34a2/decalib/trainer.py#L168C13-L203C109

Original DECA's Coarse Step Loss function

2D vs GT Self Supervision

- Landmark Loss: 랜드마크 거리간 비교
- eye/lip distance loss: 눈동자 간격, 입술 거리 간
- photometric loss: 모든 픽셀 밝기값에 대한 L1
- Identity Loss: VGGFace2 기반 FR네트워크로 정체성 보존
- Regularization: 합 제약

```
14 from .utils import lossfuncN as lossfunc
```

Loss func. of this code

- https://www.notion.so/loss-functions-25de836919338004b007fdea79f1dec2

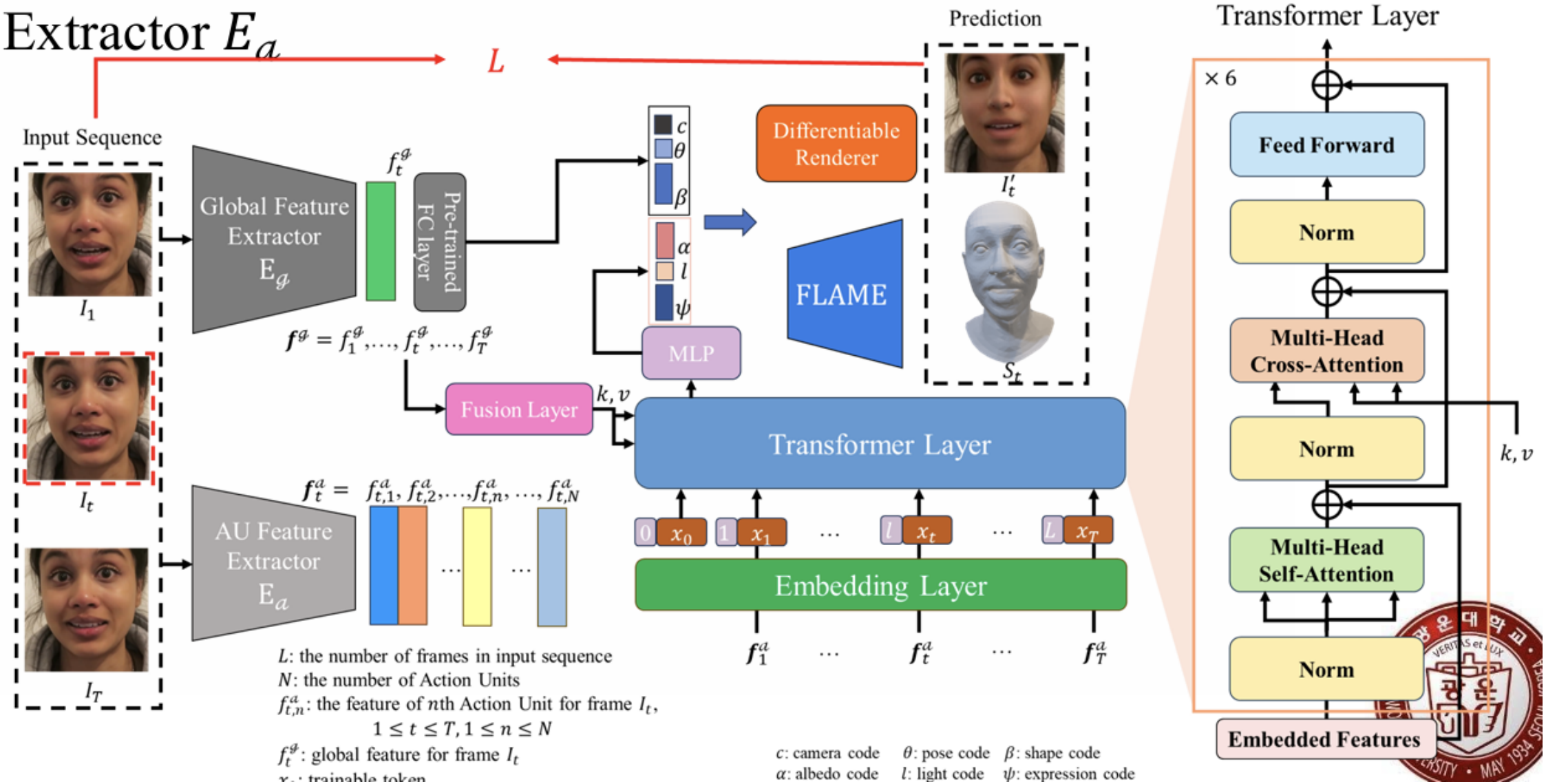
1. 프레임워크 논의: sequential DECA?

Sequential model

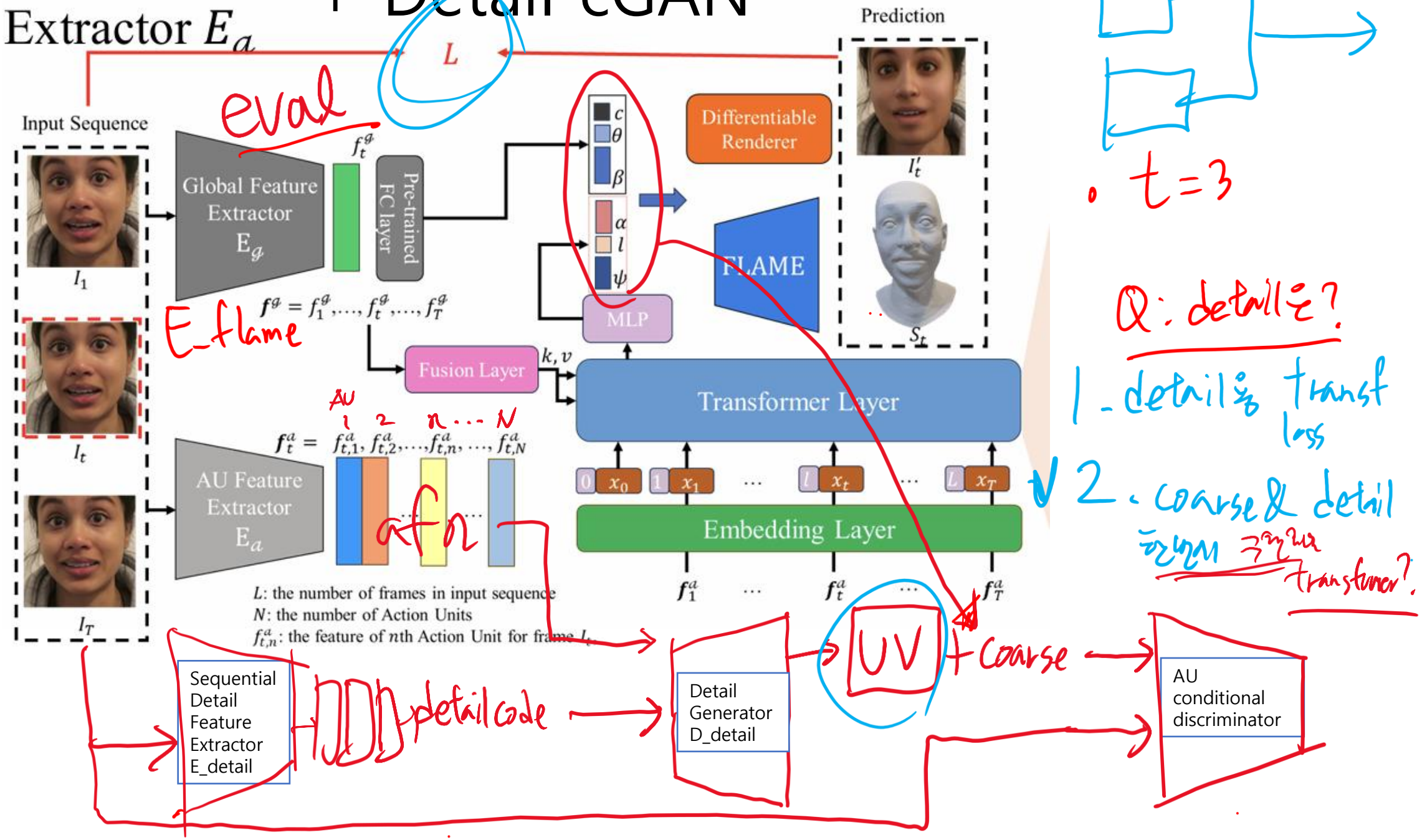
eature Extractor E_a

Layer

AME)



+ Detail cGAN



• t=3

Q: detail은?

1. detail을 transf loss

2. coarse & detail 합쳐서 같이 transformer?

2. 1D AU 조건을
2D cGAN 에 넣는법?

GANimation의 액션유닛 회귀 사용

- GANimation의 AU Regression Head를
  Detail cGAN의 앞부분에 붙여보자.
- GANimation은 입력 얼굴 이미지의 표정을 액션 유닛(AU) 벡터로 조절하는 조건부 GAN 기법을 제안한 연구.
- AU는 얼굴 표정을 구성하는 근육 움직임을 정량화한 것. FACS에서 정의.
- GANimation에서는 AU vector로 연속적 표정 변환 제어.
  Attention 매커니즘과, **AU 회귀 손실을** 도입해,
  어떤 AU와 어떤 pixel과 관련되는지 학습.