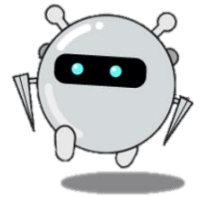
Guideline for Final Project



2023.08.10. (목)

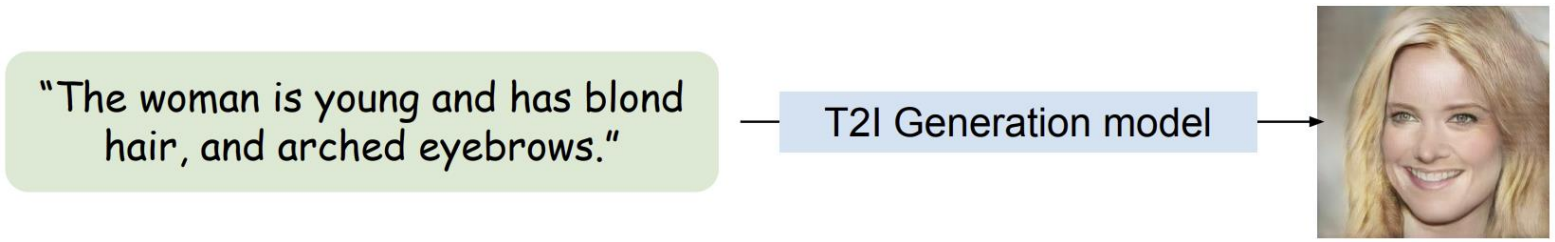
작성자 : 이현수, 현시은

OUTTA

Task: Text-to-Image Generation

T2I Generation

- Given text prompt 에 대응되는 image 를 generation 하는 task 입니다.
- 최신 논문들의 경우 **Conditional GAN** 및 Conditional Diffusion Models 를 사용합니다.
- Train data 로는 (image, text) pair 가 input 됩니다 (fully-supervised).
  - 최근에는, 학습 과정에서 text data 를 이용하지 않거나 일부만 이용하는 zero-shot / few-shot text-to-image generation 도 활발히 연구되고 있습니다.
- Validation/Test 시에는 text data 가 input 되고, generated image 가 output 됩니다.
- Metric: **FID, Inception score**, Precision 등이 주로 사용됩니다.



T2I Generation

- **Fully-supervised** Text-to-Image Generation Models using **Conditional GAN**
  - StackGAN
  - StackGAN++
  - AttnGAN
  - DF-GAN
  - TediGAN

- **Zero-shot / Few-shot** Text-to-Image Generation Models using **Conditional GAN**
  - LAFITE
  - LAFITE2
  - VDLGAN

T2I Generation

- 이번 프로젝트에서는 Conditional GAN 을 이용하여 간단한 T2I Generation model 을 구현해 보는 것을 목표로 합니다.

Dataset

- **Multimodal CelebA-HQ**
  - https://github.com/IIGROUP/MM-CelebA-HQ-Dataset
- 전체 이미지 수: 30000
  - Train data: 24000, Test data: 6000
  - 1개의 image 당 10개의 text caption 존재

- 이번 project 에서는 Colab CPU/GPU memory issue 로 인하여, **1개의 image 당 4개의 text caption 만을 이용**하여 학습을 진행합니다.
- 학습 과정에서 on-the-fly 로 image, text file 을 open 하고 CLIP embedding 을 추출하는 과정은 비효율적이므로, **학습 전 data preprocessing 을 통해 train/test data 에 대한 CLIP embedding 을 미리 추출 및 저장**합니다.

High-level Architecture

- 기본적으로 멘토들에 의해 제공되는 모델 구조는 아래와 같습니다.

1) VAE Encoder 구조를 이용한 Conditioning augmentation 이 적용되어 있습니다.

2) Three-stage GANs

- 1st stage: Generates **64 x 64** image
- 2nd stage: Generates **128 x 128** image
- 3rd stage: Generates **256 x 256** image
- # of stages can be expanded, but not recommended,
  since there's no official results for this experiments.
- # of stages are not fixed (but at least 1). Detailed explanation in implementation section.

3) 기존 GAN 의 loss term 과 더불어, Constrastive learning 을 위한 loss term 을 추가로 사용합니다.

- 성능 향상을 위해 다른 논문의 모델구조를 이용 및 변형할 수 있습니다. (Optional, extra points)

Restriction

- 공정성을 위해 구현 및 학습 과정에서 다음과 같은 제한사항이 존재합니다.
- (1) **반드시 무료 Google Drive 계정 및 Google Colab 계정을 사용해 주시기 바랍니다.**
  - 본 프로젝트는 무료 계정으로도 충분히 진행할 수 있음이 검증된 후 여러분들께 배포되었습니다.
  - 추가적인 storage 및 무료 Colab 에서 제공되지 않는 고사양의 CPU/GPU 사용은 금지됩니다.
  - 즉, Colab 무료 계정에서 제공되는 CPU (12.7GB) 및 T4 GPU (15.0GB) 만을 이용해 주시기 바랍니다.
  - 런타임 시간이 제한되어 있으나, checkpoint 를 저장하고 이어서 학습시킬 수 있게 구현되어 있습니다.

- (2) **추가적인 train data 사용은 불가능합니다.**
  - 특히, test data 를 이용해 학습을 진행하시는 것은 강력히 금지되며, 적발 시 캠프 미이수 처리 대상입니다.

Restriction

- 공정성을 위해 구현 및 학습 과정에서 다음과 같은 제한사항이 존재합니다.
- (3) **FID와 IS라는 Metric으로 성능을 측정할 예정입니다.**
  - Metric을 측정하는 코드는 제공되지 않습니다.(측정하는 코드를 직접 구축하면 가산점 부여)

- (4) **Random seed 는 0으로 고정되어 있으며, 이 값은 그대로 사용하셔야 합니다.**
  - Seed 고정은 이미 구현되어 있으므로, 추가 구현 및 수정은 불필요합니다.

Tips

- (1) 학습을 진행할 때, 데이터셋을 불러오는 데 시간이 유의미하게 소요됩니다 (2-3분).
  - 즉, 구현 시 전체 데이터셋을 이용해 코드가 실행되는지 확인하는 것은 비효율적입니다.
  - 따라서, 구현 시에는 제공된 sample_train.zip 파일을 사용하시고,구현이 완료하시고 본격적으로 학습하실 때만 전체 데이터셋 (train_data_4cap.zip) 을 사용하시기 바랍니다.

- (2) 학습 도중 일부 epoch 에서 비정상적인 이미지가 출력될 수도 있습니다.
  - 가능한 원인으로는 GAN 의 불안정한 학습/구현 오류/Hyperparameter issue 가 있습니다.
  - 첫 번째 원인일 경우, 1-2 epoch 정도 더 학습시키면 다시 안정화 됩니다. 따라서, 1-2 epoch 정도 더 학습 시켜 보시고 결정을 내리시는 것을 추천드립니다.

Tips

- (3) 매 Epoch 마다 checkpoint 가 저장되므로 Google Drive 저장공간이 부족해질 수 있습니다.
  - 필요 없는 checkpoint 는 수시로 삭제해 주시기 바랍니다.

- (4) 앞으로 설명드릴 모델 구조는 멘토들이 시범으로 사용한 구조로, **층을 더 쌓거나/활성화 함수를 변경하시거나/학습률, scheduler 등과 같은 hyperparameter 수정이 가능**합니다.
  - 필수 구현사항 구현 후 다른 논문을 구현하시는 것 역시 가능합니다 (Optional, extra credit).
  - 뒷부분에도 설명 되어 있습니다.

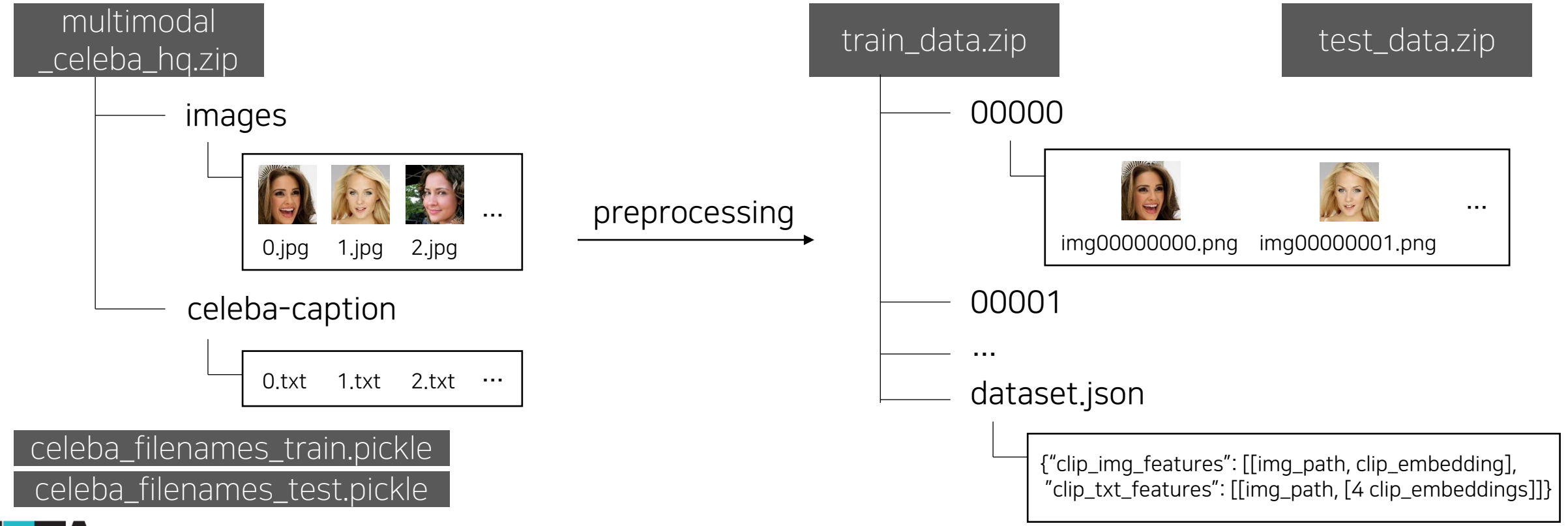
Tips

- (5) 1개의 구글 무료 계정 당 4~6 epoch가 가능합니다. (1 epoch당 40~50분 가량 소요됨)
  - 따라서, 미루지 마시고 빠른 시일 내로 모델을 구축하시고 학습을 진행하시길 바랍니다.
  - 성능을 높이기 위해선 하이퍼 파라미터 튜닝 및 모델 구조 변경 등의 많은 실험이 필요할 것입니다.

Details on dataset and models

0) Data Preprocessing

- 주어진 데이터 (multimodal_celeba_hq.zip) 으로부터 image/text caption 전처리를 진행하고 clip image embedding 을 추출해 zip 파일로 저장합니다.
  - 자세한 내용은 To-do 슬라이드 및 스켈레톤 코드를 참고해 주시기 바랍니다.



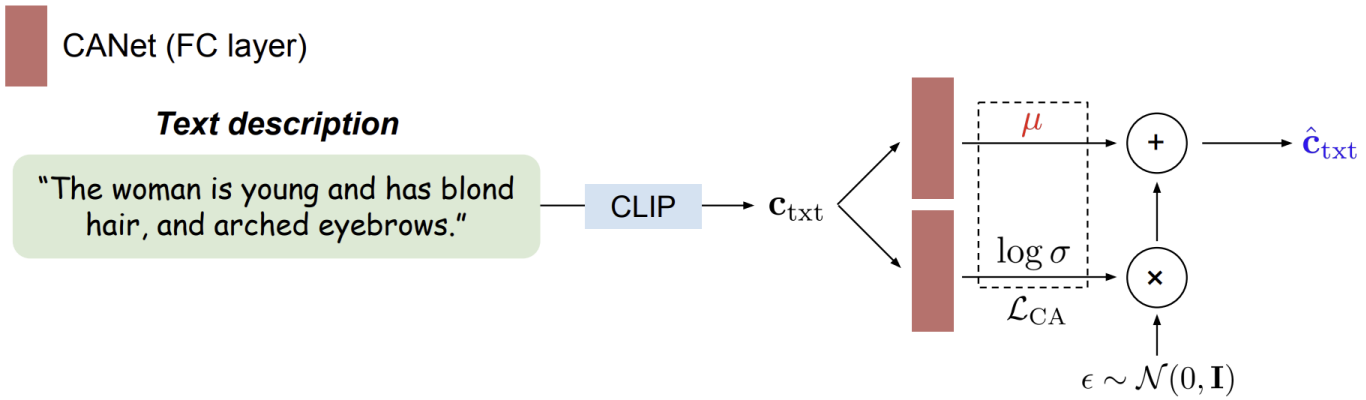
0) Data Preprocessing

- 데이터셋은 공식 Github 에서 다운로드 할 수 있지만, 편리성 및 공정성을 위해 멘토들이 준비한 zip 파일을 사용해 주시기 바랍니다.
- 데이터셋 zip file 을 Google Drive 에 업로드하고, 데이터 전처리 코드를 실행해 주시기 바랍니다.
  - Train data 전처리 과정은 **Colab 무료 GPU 기준 약 35-40분 내외**가 소요됩니다.

- 데이터 전처리 코드 구현 이후에는 sample data point 10개만을 이용해 정상적으로 실행이 되는지 확인하신 후, 코드가 완벽히 완성된 후 전체 dataset 에 대해 실행해 주시기 바랍니다.
  - 제공된 스켈레톤 코드 상에 더 자세히 설명되어 있습니다.
- 데이터 전처리가 종료된 후 zip 파일이 저장되기까지 시간이 소요됩니다 (~10분). Google Drive 상에서 파일 용량을 통해 확인해 주시기 바라며, **이 시간 동안은 런타임을 연결해 두셔야 합니다.**
  - (참고) Preprocess 된 train data zip file 의 용량: ~ 5.37GB

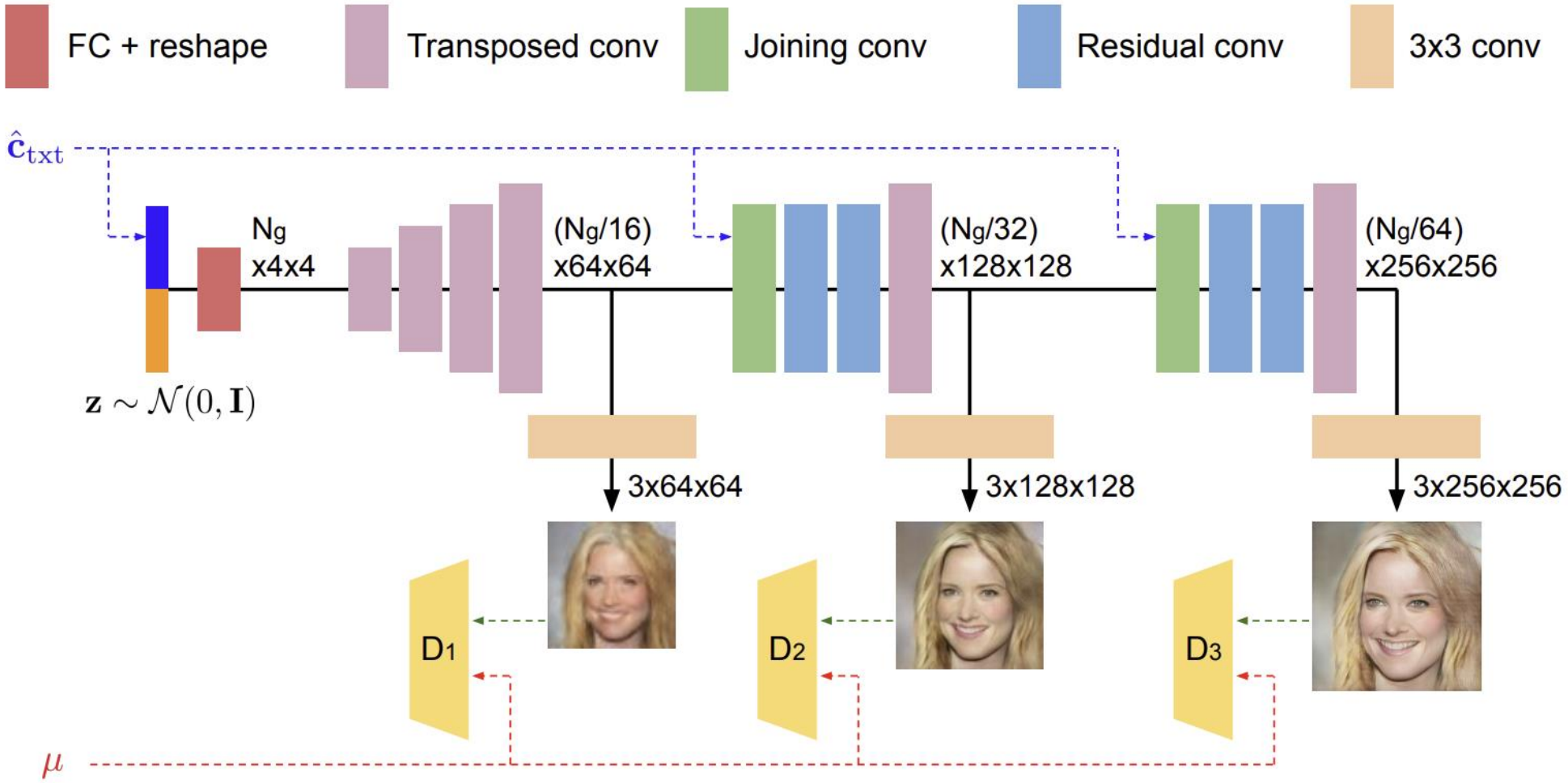
1) **C**onditioning **A**ugmentation **net**work (CANet)

- Augmentation of text embedding vector
  - Text embedding is extracted using pre-trained CLIP model.
  - $\dim(\mathbf{c}_{\text{txt}})$ = 512 (default, w/ CLIP ViT-B/32), 768 (w/ CLIP ViT-L/14), ⋯
  - $\dim(\hat{\mathbf{c}}_{\text{txt}})$ is a hyperparameter (default: 128)
- CANet is implemented with FC layers
- Loss function for CANet: $\mathcal{L}_{\text{CA}} = D_{\text{KL}}(\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t) \| \mathcal{N}(0, \mathbf{I}))$
  - Loss term is already implemented by mentors, so you can just use it!
  - In practice, CANet calculates 'log(std)' for easier computation of KL-divergence.



OUTTA

2) Generator



OUTTA

2) Generator

- Overall structure is similar to StackGAN++
- Do NOT use bias in convolution/Transposed convolution layer
- Hyperparameters
  - Distribution type/Dim. of noise **z** (default: z ~ gaussian(0, 1), |z|=100)
  - Dim. of condition vector $\hat{\mathbf{c}}_{\text{txt}}$ extracted from CANet (default: 128)
  - Value of N_g (default: 1024, Strongly Recommended)
  - Stride, pooling, kernel size of Transposed convolution/convolution layer
  - # of Transposed conv layer (default: 4 layers for 1st stage, 1 layer for each 2nd/3rd stage)
  - # of residual conv layer (default: 2 layers for each 2nd/3rd stage)
  - Types of activation function except for last transposed convolution layer (e.g. ReLU)
  - # of stages (default: 3)
  - Etc ···

2) Generator

- Loss function
  - Conditional loss and Unconditional loss (m = # of stages)

$$\mathcal{L}_G = \sum_{i=1}^{m} \mathcal{L}_{G_i}, \qquad \mathcal{L}_{G_i} = \underbrace{-\mathbb{E}_{s_i \sim p_{G_i}} [\log D_i(s_i)]}_{\text{unconditional loss}} \underbrace{-\mathbb{E}_{s_i \sim p_{G_i}} [\log D_i(s_i, \mu)]}_{\text{conditional loss}}$$

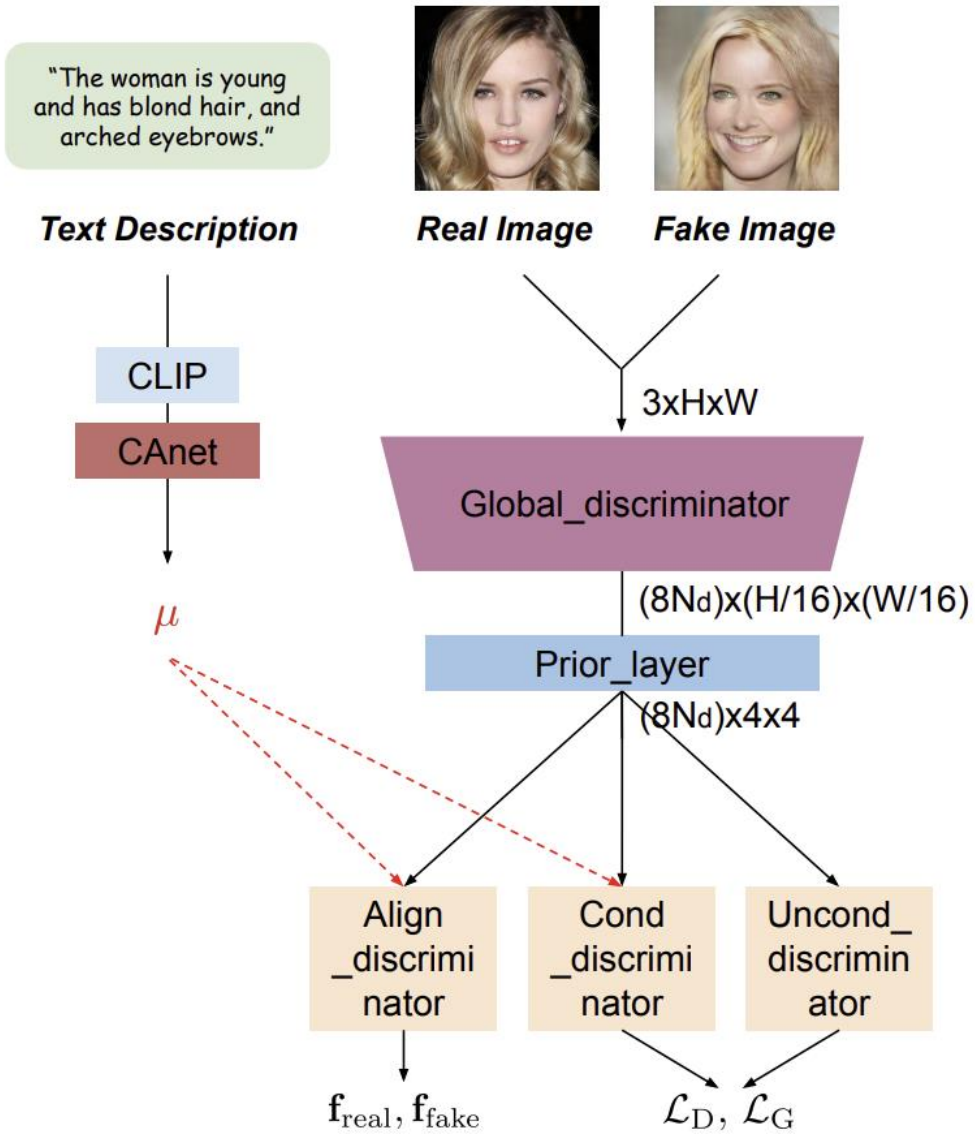
- Discriminator 부분의 설명대로, unconditional loss 의 사용은 optional 합니다.

2) Generator

- 전체적인 Text-to-Image generation 모델의 원리 및 작동 여부를 해치지 않는 선에서, hyperparameter 를 자유롭게 수정하실 수 있습니다.

- Generator 구현 시 꼭 지켜주셔야 하는 사항은 다음과 같습니다.
  - 모델에 의해 생성된 이미지를 return 하셔야 합니다.
  - 평가를 위해, 반드시 3x256x256 size 의 이미지를 생성해 주셔야 합니다.
    - 이와 다른 size의 이미지도 함께 생성하시는 것은 상관 없으나, 3x256x256 image 는 꼭 return 되어야 합니다.
  - Text 에 대한 정보 및 noise 가 반드시 condition 으로 input 되어야 합니다.
    - 즉, text-unconditional generative model 을 구현하시는 것은 불가능합니다.
    - Loss function 의 term 중 conditional loss term 은 반드시 사용해 주셔야 합니다.
  - 정상적으로 실행(train/eval) 이 되는 hyperparameter 조합을 찾으셔야 합니다.
    - Size mismatch 등의 에러가 날 경우, 모델 구조를 수정해 주시기 바랍니다.

3) Discriminator



OUTTA

3) Discriminator

- Overall structure is similar to StackGAN++
- Do NOT use bias in convolution/Transposed convolution layer
- Hyperparameters
  - Usage of unconditional_discriminator (default: Use)
  - Usage of align_discriminator (default: Use)
  - Value of N_d (default: 64, Strongly Recommended)
  - # of conv layer (default: 4 layers in global_discriminator, 1 in uncond, 2 in cond/align)
  - Stride, pooling, kernel size of convolution layer
  - Types of activation function for global_discriminator and prior_layer (e.g. ReLU)
    - For the last activation function of cond/uncond discriminator, you must use Sigmoid.
  - # of stages (default: 3)
    - Should be same as # of stages of generator.
  - Etc …

3) Discriminator

- Loss function
  - Conditional loss and Unconditional loss (m = # of stages)
  - Unconditional/conditional loss 계산을 위해, uncond/cond discriminator 가 각각 사용됩니다.
  - Align discriminator 는 constrastive loss 계산을 위해 사용됩니다 (뒷부분 참고).

$$\mathcal{L}_D = \sum_{i=1}^{m} \mathcal{L}_{D_i}, \qquad \mathcal{L}_{D_i} = \underbrace{-\mathbb{E}_{x_i \sim p_{\text{data}}}[\log D_i(x_i)] - \mathbb{E}_{s_i \sim p_{G_i}}[\log(1 - D_i(s_i))]}_{\text{unconditional loss}} + \underbrace{-\mathbb{E}_{x_i \sim p_{\text{data}}}[\log D_i(x_i, \mu)] - \mathbb{E}_{s_i \sim p_{G_i}}[\log(1 - D_i(s_i, \mu))]}_{\text{conditional loss}}$$

- Unconditional loss 의 사용은 optional 합니다. (구현은 필수입니다)

3) Discriminator

- 전체적인 Text-to-Image generation 모델의 원리 및 작동 여부를 해치지 않는 선에서, hyperparameter 를 자유롭게 수정하실 수 있습니다.

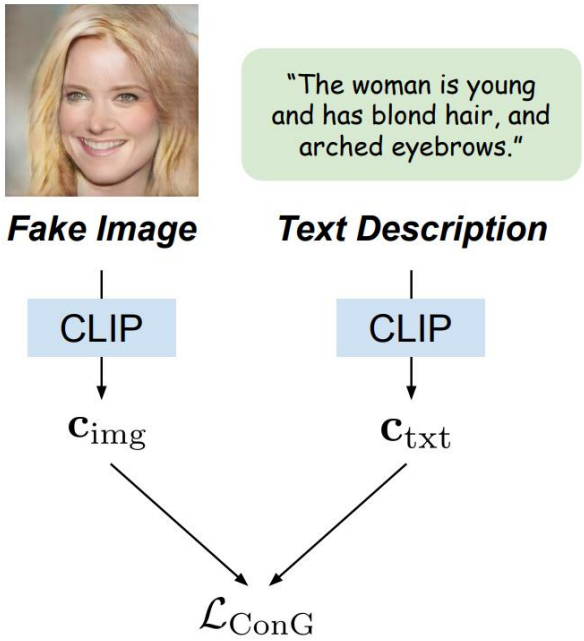
- Discriminator 구현 시 꼭 지켜주셔야 하는 사항은 다음과 같습니다.
  - Cond/Uncond/Align discriminator 를 모두 구현은 해 주셔야 합니다.
    - 학습 과정에서 Uncond/Align discriminator 의 사용 여부가 optional 한 것으로, 구현과는 별개입니다.
  - Conditional discriminator 에는 Text 에 대한 정보가 반드시 condition 으로 input 되어야 합니다.
  - 정상적으로 실행(train/eval) 이 되는 hyperparameter 조합을 찾으셔야 합니다.
    - Size mismatch 등의 에러가 날 경우, 모델 구조를 수정해 주시기 바랍니다.

4) Contrastive Learning

- Preliminary: Contrastive learning
  - 사전에 진행된 이론 수업을 참고해 주시기 바랍니다.
- LAFITE 논문에서 사용된 contrastive loss 를 사용합니다.
  - LAFITE (CVPR 2022): https://arxiv.org/abs/2111.13792
  - 사전에 부여된 과제(논문리뷰) 를 참고해 주시기 바랍니다.
- Discriminator 에서 구현한 align discriminator 가 contrastive learning 에 사용됩니다.

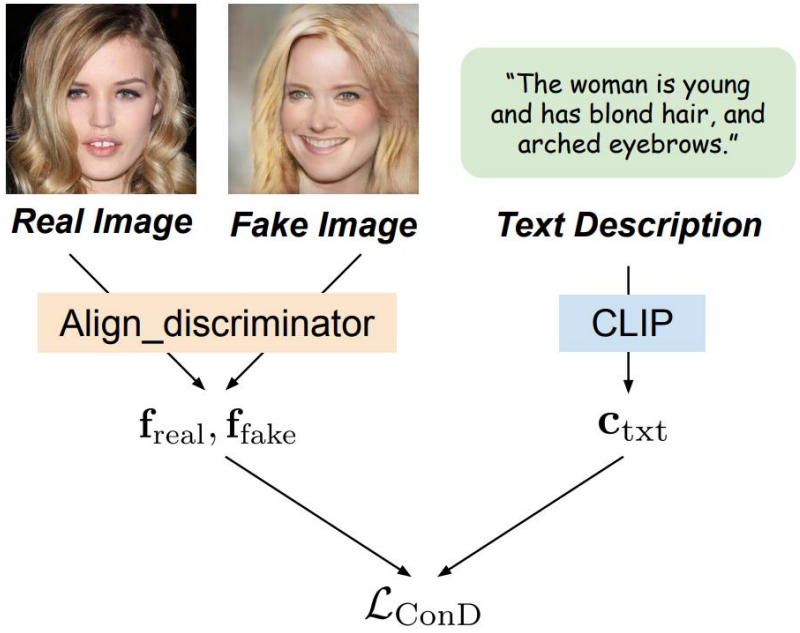
4) Contrastive Learning

- Loss function
  - Contrastive loss for Generator:



$$\mathcal{L}_{\text{ConG}} = -\tau \sum_{i=1}^{n} \log \frac{\exp(\text{Sim}(f_{\text{img}}(\mathbf{x}'_i), \mathbf{h}'_i)/\tau)}{\sum_{j=1}^{n} \exp(\text{Sim}(f_{\text{img}}(\mathbf{x}'_j), \mathbf{h}'_i)/\tau)}.$$

  - Contrastive loss for Discriminator:



$$\mathcal{L}_{\text{ConD}} = -\tau \sum_{i=1}^{n} \log \frac{\exp(\text{Sim}(f_{\text{s}}(\mathbf{x}_i), \mathbf{h}'_i)/\tau)}{\sum_{j=1}^{n} \exp(\text{Sim}(f_{\text{s}}(\mathbf{x}_j), \mathbf{h}'_i)/\tau)},$$

Training Objective

- Overall loss function

$$\mathcal{L}'_D = \mathcal{L}_D + \gamma \mathcal{L}_{\text{ConD}} \tag{1}$$

$$\mathcal{L}'_G = \mathcal{L}_G + \gamma \mathcal{L}_{\text{ConD}} + \lambda \mathcal{L}_{\text{ConG}} + \beta \mathcal{L}_{\text{CA}} \tag{2}$$

  - Hyperparamter: $\tau, \gamma, \lambda, \beta$
    - Default: $\tau = 0.5, \gamma = 5, \lambda = 10, \beta = 1$

5) Training loop

- Setup: train function
- Each epoch: train_step function
- Loss computation: contrastive_loss_G, contrastive_loss_D, D_loss, G_loss function

- Optimizer 및 scheduler 와 같은 hyperparamter 를 자유롭게 변경하실 수 있습니다.
- 자세한 내용은 직접 train.py 코드를 읽어보시기 바랍니다.

To-do: Implementation

Skeleton Codes

- 각 코드 파일에 대한 기본적인 설명
- 노란색 파일이 필수로 구현해 주셔야 하는 파일이며, 연두색 파일은 구현에 성공 시 가산점을 드립니다.
  - main.py: 실제로 실행하는 파일, hyperparameter 및 arguments 정의
  - train.py: dataset open 및 dataloader 정의, training loop 존재, loss 및 gradient 계산, main.py 에서 호출
  - network.py: 모델을 구현한 파일, train.py 에서 호출
  - read_dataset.py: dataset open, train.py 에서 호출
  - dataloader.py: dataloader 생성 후 반환, train.py 에서 호출
  - train_utils.py: loss 계산 및 checkpoint 관리에 필요한 여러 함수 내장, train.py 에서 호출
  - fix_seed.py: seed 고정, main.py 에서 호출
  - preproc_datasets_celeba_zip_train.py : train dataset을 만들기 위한 함수 (Optional)
  - preproc_datasets_celeba_zip_test.py : test dataset을 만들기 위한 함수 (Optional)

Arguments of main.py

Arg Name	Type	Explanation	default
train_data	Path	Train data (zip file) 의 경로	
batch_size	int	batch size	24
num_epochs	int	전체 Epoch 수	
learning_rate	float	학습률	0.0002
report_interval	int	몇 iteration 마다 학습 과정 (loss 값)을 출력할 것인지	50
noise_dim	int	$\dim(\mathbf{z})$	100
projection_dim	int	$\dim(\mathbf{c}_{\text{txt}})$	128
clip_embedding_dim	int	$\dim(\hat{\mathbf{c}}_{\text{txt}})$	512
checkpoint_path	Path	checkpoint 가 저장되는 폴더 경로	
result_path	Path	result image 가 저장되는 폴더 경로	
use_uncond_loss	bool	Unconditional loss 를 사용할지 여부 (True: 사용)	True
use_contrastive_loss	bool	Contrastive learning 을 적용할지 여부 (True: 사용)	True

Arguments of main.py (Cont'd)

Arg Name	Type	Explanation	default
resume_checkpoint_path	str	Resume 할 weight 가 저장되어 있는 checkpoint 폴더 경로, **None** 을 입력할 경우 resume 하지 않고 처음부터 학습 진행.	None
resume_epoch	int	Resume 을 시작할 epoch – 1 의 값, 즉 이전에 5epoch 까지 학습되었다면 5를 입력하면 6epoch 부터 학습 시작. **-1** 을 입력하면 resume 하지 않고 처음부터 학습 진행.	-1

* 두 변수 각각 None 과 -1 이 아닌 값이 모두 입력되어야 checkpoint 를 load 해 이어서 학습합니다.

To-do

- 최종 프로젝트의 점수는 다음의 기준으로 부여됩니다.

: 코드 구현 점수 200점 + 모델 성능 점수 200점 = 총점 400점 만점

OUTTA

To-do

- 코드 구현에 해당하는 점수는 다음과 같습니다. (모델 성능 점수와 별개)

Problem	Subproblem	Type	Points
1. Data preprocessing	(a) Make train dataset	Optional	2.5
	(b) Make test dataset		2.5
2. Generator 구현	(a) CA	Essential	5
	(b) ImageExtractor		5
	(c) Type 1		15
	(d) Type 2		15
	(e) Generator		10
3. Discriminator 구현	(a) Discriminator		25
	(b) Uncond		5
	(c) Cond		10
	(d) AlignCond		5

To-do

- 코드 구현에 해당하는 점수는 다음과 같습니다. (Cont'd)

Problem	Subproblem	Type	Points
4. Training loop 구현	(a) G_loss	Essential	10
	(b) D_loss		10
	(c) Con_loss_G		15
	(d) Con_loss_D		5
	(e) etc		5
	(f) Language-free	Optional	5
5. 다른 논문 구현		Optional	40
6. FID/IS 출력		Optional	10

Essential	140pt	70%
Optional	60pt	30%
Total	**200pt**	**100%**

OUTTA

To-do

- 모델 성능 점수는 다음의 기준으로 부여됩니다.

- FID와 IS로 모델의 성능을 측정할 예정입니다.
- FID는 낮을수록, IS는 높을수록 좋은 성능의 모델임을 시사합니다.
- FID가 가장 낮은 조에게 90점, IS가 가장 높은 조에게 90점을 부여할 예정입니다.
- 1등 조를 기준으로, 순위가 하나 하락할 때마다 5점씩 감점됩니다.

  (ex) 10조가 FID가 3등, IS는 5등을 했을 때 : 80점 + 70점 = 150점이 10조의 성능 점수
- 추가로, 저희가 미리 정해둔 2개의 text를 입력하여 올바른 이미지가 출력되는지를 확인할 예정

  입니다 : 1개의 text 당 10점 부여

제출해야 하는 것 [Essential]

- (1) network.py
- (2) train.py
- (3) 모델의 weight
    - epoch_{num}_Dis_0.pt
    - epoch_{num}_Dis_1.pt
    - epoch_{num}_Dis_2.pt
    - epoch_{num}_Gen.pt
    - hyperparameter.pt
- (4) 간단한 Report : 3 page 이내로 network.py 및 train.py의 코드를 설명

제출해야 하는 것 [Optional]

- (5) preproc_datasets_celeba_zip_train.py
- (6) preproc_datasets_celeba_zip_test.py
- (7) 직접 구현한 FID/IS 측정 코드
- (8) 멘토가 제공한 스켈레톤 코드가 아닌 본인들의 조에서 직접 제작한 모델
    - 모델을 만들 때 사용한 모든 코드
    - 가장 좋은 성능의 weight
    - 간단한 설명서 (5 page 내외)
- (9) 그 외 자체적으로 추가 및 수정한 코드 전부

- 단, 여러분들이 만든 모델은 반드시 저희가 제공한 모듈들과 잘 연결이 되어야 합니다. 저희가 만든 dataloader와 main 및 generate_image 모듈과 연결되지 않으면 점수를 드리지 않습니다.

- 과제 제출 마감일 : 8월 22일(화) 오전 11시 59분
- 다음의 메일로 과제를 제출 받을 것입니다. : admin@outta.ai

<유의 사항>
- 각 팀당 멘토에게 3번의 질문 기회 제공 (7월처럼 무제한으로 질문 불가)
- 단, 3명으로 이루어진 조는 4번, 2명으로 이루어진 조는 5번의 질문이 가능하다.
- 어떤 종류의 질문이든 질문 기회는 차감됩니다.
- 질문은 조 당 대표 1명만 해주셔야 합니다.
- 2번의 실시간 QnA 세션 진행 예정 (매주 토요일 오후 2시 ~ 4시)
  - 8월 12일(토) / 8월 19일(토) 오후 2시 ~ 4시 예정
- QnA 세션에서의 질문도 질문 기회에서 차감됩니다.

To-do

- **구체적인 구현 사항 및 Hint 는 스켈레톤 코드에 기재되어 있습니다.**
- [Optional] Problem 1. *preproc_datasets_celeba_zip.py*

1-(a). multimodal_celeba_hq.zip 데이터셋에서 train data를 만들어라. [2.5pt]

데이터셋과 함께 제공된 celeba_filenames_train.pickle 파일을 이용해 문제를 해결하자.

preproc_datasets_celeba_zip_train.py의 빈 칸을 채워넣는다.

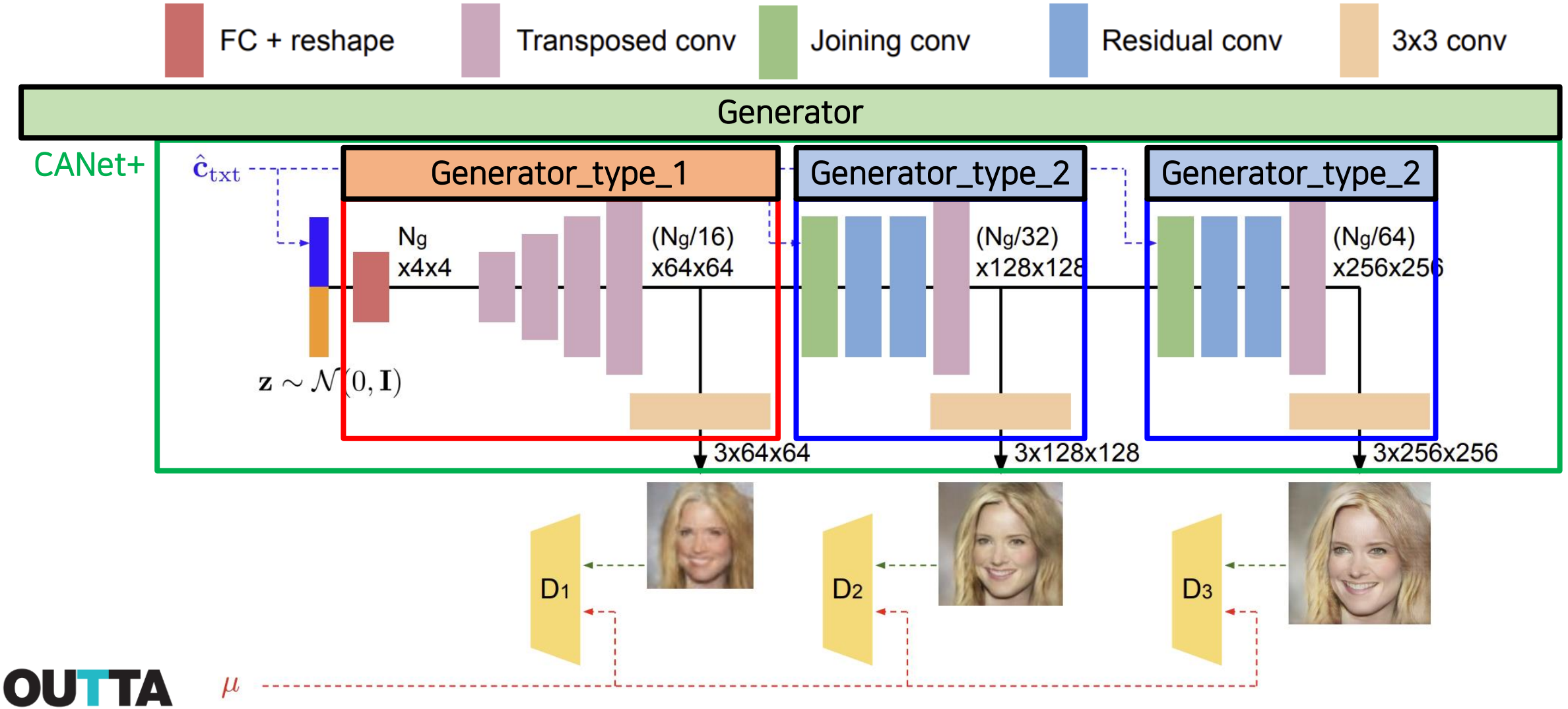
1-(b). Multimodal_celeba_hq.zip 데이터셋에서 test data를 만들어라. [2.5pt]

데이터셋과 함께 제공된 celeba_filenames_test.pickle 파일을 이용해 문제를 해결하자.

preproc_datasets_celeba_zip_test.py의 빈 칸을 채워넣는다.

To-do

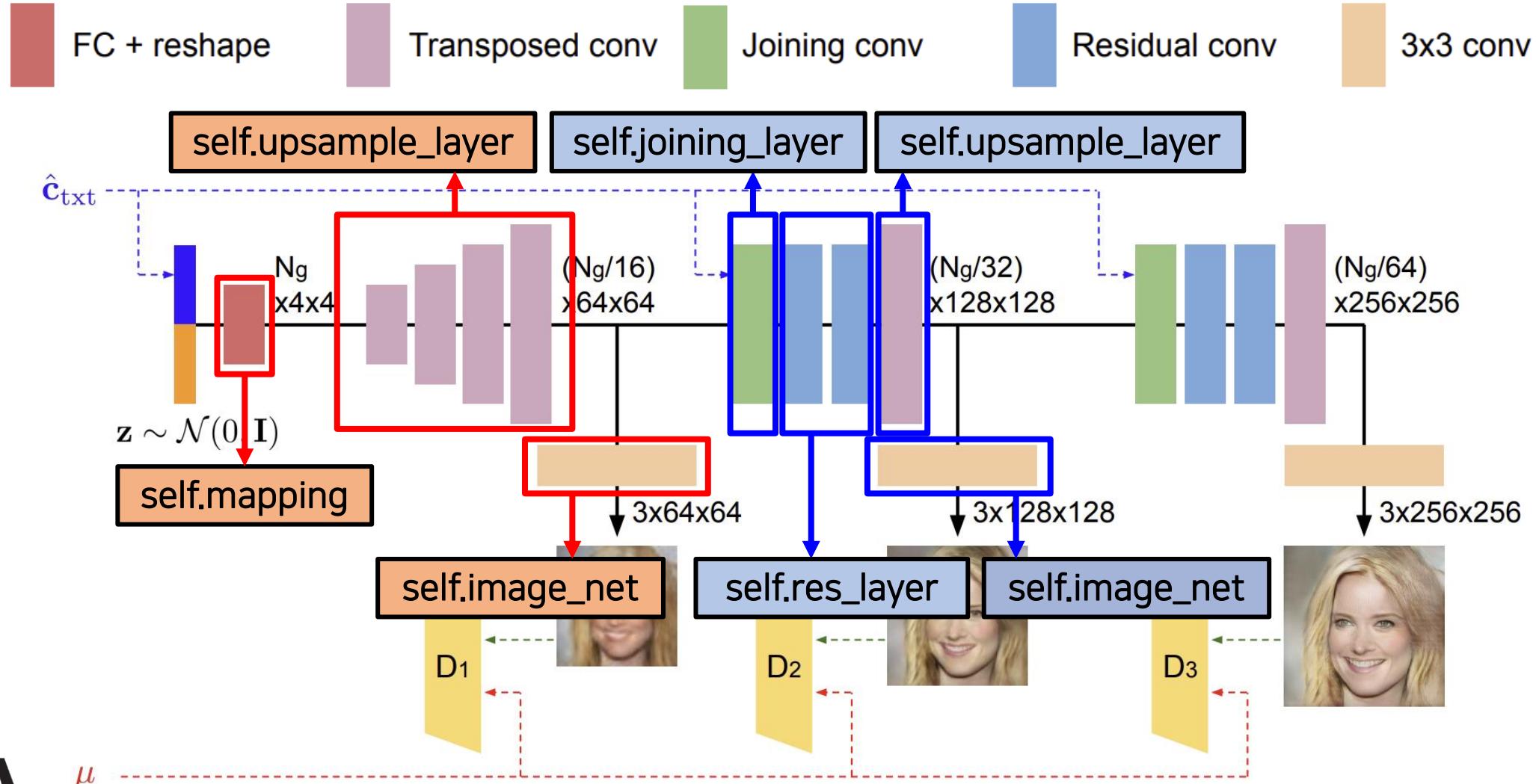
class Generator/Generator_type_1/Generator_type_2:



OUTTA

To-do

class Generator/Generator_type_1/Generator_type_2:



OUTTA

To-do

- Problem 2. *network.py*

여기에는 여러분들의 모델 구축을 도와주기 위한 도움말을 상세하게 적어두었습니다.

그러나 그 힌트에는 이미지의 사이즈에 대한 정보만 포함되어 있고, 배치 사이즈의 정보는 빠져 있습니다.

따라서 여러분들이 코드를 작성할 때는 반드시 'batch'를 고려하셔야 합니다.

Batch는 tensor의 dim=0 차원에 나타납니다.

To-do

- **Problem 2.** *network.py*

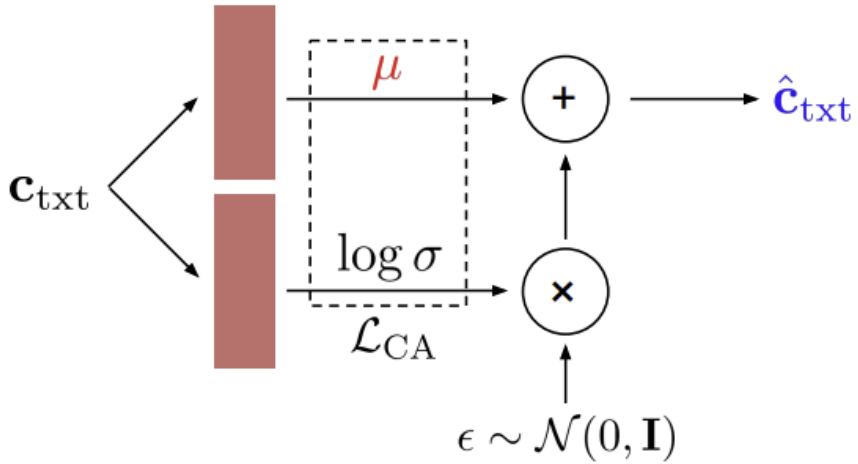
  2-(a). ConditioningAugmentation class 를 구현하여라. [5pt]

  (1) [FC+Activation] 을 이용해 input tensor $\mathbf{x}$의 $\mu,\ \log(\sigma)$ 를 계산하고

  (2) Reparameterization trick $\hat{\mathbf{x}} = \mu + \sigma \cdot \mathbf{z},\ \mathbf{z} \sim N(\mathbf{0}, \mathbf{I})$ 을 이용해 output tensor $\hat{\mathbf{x}}$ 을 계산한다.

  def __init__(self, input_dim, emb_dim)

  - self.input_dim : *clip_embedding_dim* (same as $\dim(\mathbf{c}_{\text{txt}})$)
  - self.emb_dim : *projection_dim* (same as $\dim(\hat{\mathbf{c}}_{\text{txt}})$)



To-do

- **Problem 2.** *network.py*

  2-(a). ConditioningAugmentation class 를 구현하여라. [5pt]

  (1) [FC+Activation] 을 이용해 input tensor $\mathbf{x}$의 $\mu,\ \log(\sigma)$ 를 계산하고

  (2) Reparameterization trick $\hat{\mathbf{x}} = \mu + \sigma \cdot \mathbf{z},\ \mathbf{z} \sim N(0, \mathbf{I})$ 을 이용해 output tensor $\hat{\mathbf{x}}$ 을 계산한다.

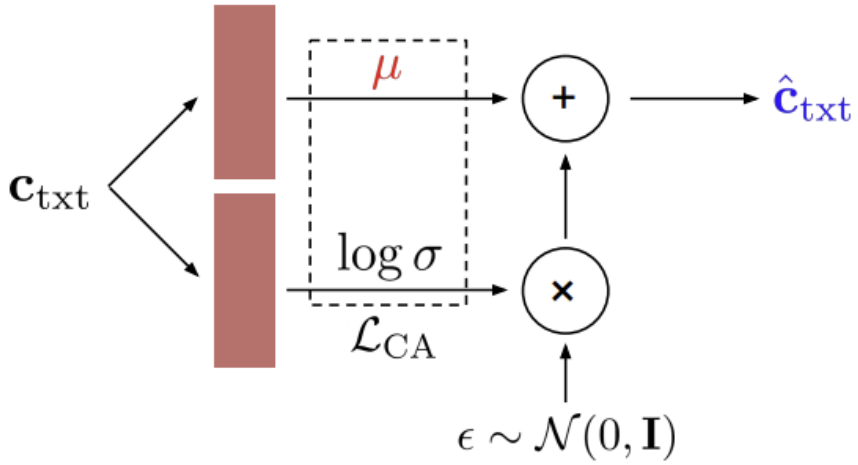
  def forward(self, x)

  Inputs

  - x: $\mathbf{c}_{\text{txt}}$

  Returns

  - condition: augmented text embedding $\hat{\mathbf{c}}_{\text{txt}}$
  - mu: calculated mean of input tensor x using CANet
  - log_sigma: calculated log(sigma) of input tensor x using CANet



To-do

- **Problem 2.** *network.py*

  2-(b). ImageExtractor class 를 구현하여라. [5pt]

  self.image_net

  - [TransposeConv2d + Activation] 을 구현해 이미지를 생성하는 class 를 구현하자.
  - 앞선 강의 및 실습들을 기반으로, 이미지를 생성하기 위해 가장 마지막 upsample 단계에서 어떠한 활성화 함수를 사용해야 하는지 생각해 보자.

  def forward(self, x)

  Inputs

  - x: input tensor, shape [C, H, W]

  Returns

  - out: output image, shape [3, H, W]

To-do

- **Problem 2.** *network.py*

  2-(c). Generator_type_1 class를 구현하여라. [15pt]

  def __init__(self, in_chans, input_dim)

  - self.in_chans = N_c
  - self.input_dim = *noise_dim* + *projection_dim*

  def forward(self, condition, noise):

  Inputs

  - condition: $\hat{\mathbf{c}}_{\text{txt}}$ extracted from CANet, shape [*projection_dim*]
  - noise: gaussian noise $\mathbf{z} \sim N(0, \mathbf{I})$, shape [*noise_dim*]

  Outputs

  - out: self.upsample_layer 를 통해 얻어진 shape [$(N_g/16)$, 64, 64] tensor
  - out_image: out 을 self.image_net 에 input 시켜 얻은 shape [3, 64, 64] image

To-do

- **Problem 2.** *network.py*

  2-(c). Generator_type_1 class를 구현하여라. [15pt]

  self.mapping

  - Implement [FC + BN + LeakyReLU] x1
  - [c, z] tensor 가 upsample_layer 에 input 될 수 있는 dimension (N_g * 4 * 4) 을 가지게끔 mapping 시켜주는 역할을 한다.

  self.upsample_layer

  - Implement [ConvTransposed2d + BN + ReLU] x4
  - 각 Block 은 통과 전/후 height/width 를 각각 2배로 만들고 channel 수를 0.5배 한다.
  - Shape of output tensor: [(N_g/16), 64, 64]

To-do

- **Problem 2.** *network.py*

  2-(d). Generator_type_2 및 ResModule class를 구현하여라. [15pt]

  def __init__(self, in_chans, condition_dim, num_res_layer)

  - self.in_chans = # of channel of previous stage generator's 'out' tensor
  - self.condition_dim = *projection_dim*
  - self.num_res_layer = # of residual conv layer inside self.res_layer (default: 2)

To-do

- **Problem 2.** *network.py*

  2-(d). Generator_type_2 및 ResModule class를 구현하여라. [15pt]

  def forward(self, condition, prev_output)

  Inputs

  - condition: μ extracted from CANet, shape [*projection_dim*]
  - prev_output: output ('out' tensor) from previous stage Generator, shape [C, H, W]

  Outputs

  - out: [self.joining_layer + self.upsample_layer] 를 통해 얻어진 shape [C/2, 2H, 2W] tensor
  - out_image: out 을 self.image_net 에 input 시켜 얻은 [3, 2H, 2W] image

To-do

- **Problem 2.** *network.py*

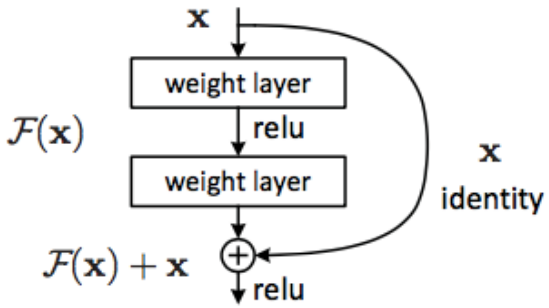
  2-(d). Generator_type_2 및 ResModule class를 구현하여라. [15pt]

  self.joint_conv

  - Implement [Conv2d + BN + ReLU] x1
  - [c, prev_output] tensor 의 height/width 는 바꾸지 않고, channel 차원만 self.in_chans 로 변경해준다.

  self.res_layer (ResModule)

  - Implement [Conv2d + BN] + ReLU + [Conv2d + BN]
  - Implement residual connection
  - 통과 전 후 tensor 의 shape 는 변하지 않는다.



  self.upsample_network

  - Implement [ConvTransposed2d + BN + ReLU] x1
  - 각 Block 은 통과 전/후 height/width 를 각각 2배로 만들고 channel 수를 0.5배 한다.
  - Shape of output tensor: [C/2, 2H, 2W]

To-do

- **Problem 2.** *network.py*

  2-(e). Generator class 를 구현하여라. [10pt]

  def __init__(self, text_embedding_dim, projection_dim, noise_input_dim, in_chans, out_chans, num_stage)

  - self.text_embedding_dim = *clip_embedding_dim*
  - self.condition_dim = *projection_dim*
  - self.noise_dim = *noise_dim*
  - self.input_dim = *projection_dim + noise_dim*
  - self.in_chans = N_c
  - self.out_chans = 3
  - self.num_stage = *num_stage*
  - self.num_res_layer = # of residual conv layer inside Type2 Gen.'s self.res_layer (default: 2)

To-do

- **Problem 2.** *network.py*

  2-(e). Generator class 를 구현하여라. [10pt]

  def _stage_generator(self, i)

  Inputs

  - i : 생성할 generator 의 stage 를 의미한다.

  Outputs

  - 2-(c), 2-(d) 에서 구현한 Type 1 Generator / Type 2 Generator class instance 를 반환한다.
    $i = 0$ 인 경우 Type 1 Generator instance 를, $i \geq 1$ 인 경우 Type 2 Generator instance 를 반환한다.
    Type 1 / Type 2 Generator class 의 __init__ 함수에 어떤 변수를 parameter 로 주어야 할지 생각해 보자.
    특히, Type 2 Generator 의 in_chans 값을 잘 생각해 보자 (High-level architecture 그림 참고).
    Hint: stage i generator 의 in_chans = stage $i-1$ generator 의 'out' tensor 의 channel 수

To-do

- **Problem 2.** *network.py*

  2-(e). Generator class 를 구현하여라. [10pt]

  def forward(self, text_embedding, noise)

  Inputs

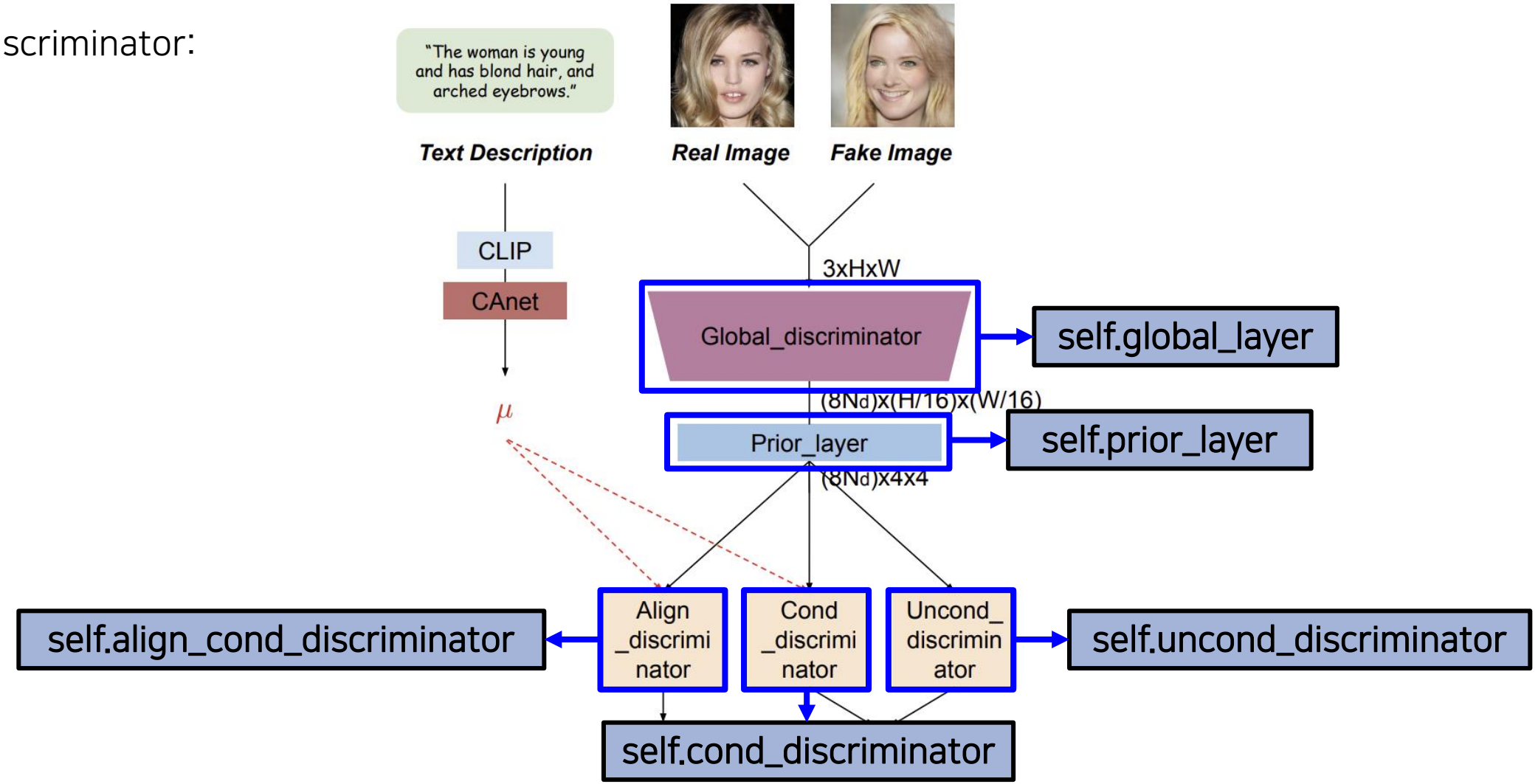
  - text_embedding: $\mathbf{c}_{\text{txt}}$ / noise: z

  Outputs

  - fake_images: 각 stage 에서 생성한 fake image 를 저장한 List
  - mu, log_sigma: CANet 에서 계산된 $\mathbf{c}_{\text{txt}}$ 의 평균과 log(표준편차)

To-do

class Discriminator:



OUTTA

To-do

- **Problem 3.** *network.py*

  3-(a). Discriminator class 를 구현하여라. [25pt]

  def __init__(self, projection_dim, img_chans, in_chans, out_chans, text_embedding_dim, curr_stage)

  - self.condition_dim = *projection_dim*
  - self.img_chans = 3
  - self.in_chans = N_d
  - self.out_chans = 1
  - self.text_embedding_dim = *clip_embedding_dim*
  - self.curr_stage = 3
  - self.num_stage = 생성할 discriminator 의 stage

To-do

- **Problem 3.** *network.py*

  3-(a). Discriminator class 를 구현하여라. [25pt]

  def forward(self, img, condition)

  Inputs

  - img: fake/real image, shape [3, H, W]
  - condition: μ extracted from CANet, shape [*projection_dim*]

  Outputs

  - out: 주어진 이미지에 대해 real/fake 를 판단한 결과 (Common output of discriminator)

    * Sigmoid function 이 적용되는 부분은 구현되어 있으니 중복 구현에 주의!

  - align_out: Align discriminator 로부터 output 된 tensor ($\mathbf{f}_{\text{real}}, \mathbf{f}_{\text{fake}}$)

To-do

- **Problem 3.** *network.py*

  3-(a). Discriminator class 를 구현하여라. [25pt]

  self.global_layer

  - Implement [Conv2d + LeakyReLU] + [Conv2d + BN + LeakyReLU] x3
  - 각 Block 은 통과 전/후 height/width 가 각각 0.5 배로 만들고 channel 수를 2배로 증가시킨다.
  - Shape of output tensor: [$8N_d$, H/16, W/16]

  self.prior_layer

  - self.global_layer 의 output tensor shape 를 [$8N_d$, 4, 4] 로 변환해준다. (이미 성립하는 경우 사용되지 않음)

  가능한 구현 방법은 아래와 같다.

  변수 k를 $k = \log_2\left(\frac{H/16}{4}\right)$ 로 정의하자.

  1) [Conv2d + BN + LeakyReLU] 를 이용해 channel 수를 2배씩 k 번 증가, height, width 를 0.5배씩 k 번 감소한다.

  2) [Conv2d + BN + LeakyReLU] 를 이용해 channel 수를 2배씩 k 번 감소시키고 height, width 는 유지한다.

To-do

- **Problem 3.** *network.py*

  3-(b). UncondDiscriminator class 를 구현하여라. [5pt]

  def __init__(self, in_chans, out_chans)

  - self.in_chans = N_d
  - self.out_chans = 1

  self.uncond_layer

  - Implement Conv2d Block
  - Shape of output tensor: [1, 1, 1]

  def forward(self, x)

  Inputs

  - x: [Global layer + prior layer] 를 통해 output 된 tensor, shape [$8N_d$, 4, 4]

  Outputs

  - uncond_out: Unconditional discriminator 를 통과한 output tensor

To-do

- **Problem 3.** *network.py*

  3-(c). CondDiscriminator class 를 구현하여라. [10pt]

  def __init__(self, in_chans, condition_dim, out_chans)

  - self.in_chans = N_d
  - self.condition_dim = *projection_dim*
  - self.out_chans = 1

  self.cond_layer

  - Implement [Conv2d + BN + LeakyReLU] + Conv2d
  - Shape of output tensor: [1, 1, 1]
  - 가능한 구현 방식: Reduction of channel size into $8N_d$ while maintaining the height, width, then change the shape into [1, 1, 1]

  def forward(self, x, c)

  Inputs

  - x: [Global layer + prior layer] 를 통해 output 된 tensor, shape [$8N_d$, 4, 4]
  - c: μ extracted from CANet, shape [*projection_dim*]

  Outputs

  - cond_out: Conditional discriminator 를 통과한 output tensor

To-do

- **Problem 3.** *network.py*

  3-(d). AlignCondDiscriminator class 를 구현하여라. [5pt]

def __init__(self, in_chans, condition_dim, text_embedding_dim)

- self.in_chans = N_d
- self.condition_dim = *projection_dim*
- self.text_embedding_dim = text_embedding_dim

self.align_layer

- Implement [Conv2d + BN + SiLU] + Conv2d
- Shape of output tensor : [*clip_embedding_dim,* 1, 1]
- 가능한 구현 방식: Reduction of channel size into $8N_d$ while maintaining the height, width, then change the shape into [*clip_embedding_dim*, 1, 1]

def forward(self, x, c)

Inputs

- x: [Global layer + prior layer] 를 통해 output 된 tensor, shape [$8N_d$, 4, 4]
- c: μ extracted from CANet, shape [*projection_dim*]

Outputs

- align_out: Align discriminator 를 통과한 output tensor

To-do

- **Problem 4.** *train.py*

  4-(a). G_loss 함수를 구현하여라. [10pt]

  이전 수업을 통해 배운 내용을 바탕으로,

$$\mathcal{L}_G = \sum_{i=1}^{m} \mathcal{L}_{G_i}, \qquad \mathcal{L}_{G_i} = \underbrace{-\mathbb{E}_{s_i \sim p_{G_i}}[\log D_i(s_i)]}_{\text{unconditional loss}} \underbrace{-\mathbb{E}_{s_i \sim p_{G_i}}[\log D_i(s_i, \mu)]}_{\text{conditional loss}}$$

  을 구현하여라.

  4-(a) 에서는 Unconditional loss 및 conditional loss 만 구현하면 된다. $\mathcal{L}_{\text{ConD}}, \mathcal{L}_{\text{ConG}}$ 는 4-(c), (d) 에서 구현한다.

To-do

- **Problem 4.** *train.py*

  4-(b). D_loss 함수를 구현하여라. [10pt]

  이전 수업을 통해 배운 내용을 바탕으로,

$$\mathcal{L}_D = \sum_{i=1}^{m} \mathcal{L}_{D_i}, \qquad \mathcal{L}_{D_i} = \underbrace{-\mathbb{E}_{x_i \sim p_{\text{data}}}[\log D_i(x_i)] - \mathbb{E}_{s_i \sim p_{G_i}}[\log(1 - D_i(s_i))]}_{\text{unconditional loss}} + \underbrace{-\mathbb{E}_{x_i \sim p_{\text{data}}}[\log D_i(x_i, \mu)] - \mathbb{E}_{s_i \sim p_{G_i}}[\log(1 - D_i(s_i, \mu))]}_{\text{conditional loss}}$$

  을 구현하여라.

  4-(b) 에서는 Unconditional loss 및 conditional loss 만 구현하면 된다. $\mathcal{L}_{\text{ConD}}, \mathcal{L}_{\text{ConG}}$ 는 4-(c), (d) 에서 구현한다.

To-do

- **Problem 4.** *train.py*

  4-(c). contrastive_loss_G 함수를 구현하여라. [15pt]

  Contrastive learning 을 위한 loss term

$$\mathcal{L}_{\text{ConG}} = -\tau \sum_{i=1}^{n} \log \frac{\exp(\text{Sim}(f_{\text{img}}(\mathbf{x}'_i), \mathbf{h}'_i)/\tau)}{\sum_{j=1}^{n} \exp(\text{Sim}(f_{\text{img}}(\mathbf{x}'_j), \mathbf{h}'_i)/\tau)}.$$

  을 구현하여라.

  이 때 $f_{\text{img}}(\mathbf{x}'_j)$ 는 반드시 각 batch 에 대응되는 tensor 의 norm 이 1이 되게끔 정규화 시켜야 한다.

To-do

- **Problem 4.** *train.py*

  4-(d). contrastive_loss_D 함수를 구현하여라. [5pt]

  Contrastive learning 을 위한 loss term

$$\mathcal{L}_{\text{ConD}} = -\tau \sum_{i=1}^{n} \log \frac{\exp(\text{Sim}(f_s(\mathbf{x}_i), \mathbf{h}'_i)/\tau)}{\sum_{j=1}^{n} \exp(\text{Sim}(f_s(\mathbf{x}_j), \mathbf{h}'_i)/\tau)},$$

  을 구현하여라.

  이 때 $f_s(\mathbf{x}'_j)$ 는 반드시 각 batch 에 대응되는 tensor 의 norm 이 1이 되게끔 정규화 시켜야 한다.

To-do

- **Problem 4.** *train.py*

  4-(e). Loss function 계산에 필요한 label 을 생성하여라. [5pt]

  이전 수업에서 Discriminator / Generator 에 대한 loss 를 계산할 때, 각각 어떠한 binary label 을 정답으로 주어야 하는지 학습하였을 것이다. 이를 바탕으로, train_step 내의 빈칸을 채워보자.

  torch.zeros 또는 torch.ones 를 이용하자. 이후 dtype 을 torch.float32 로 cast 시키고 device 로 보내자.

  d_fake_label: Discriminator 를 학습시키기 위한 label 로, fake image 를 올바르게 분류해야 함을 이용하자.
  d_real_label: Discriminator 를 학습시키기 위한 label 로, real image 를 올바르게 분류해야 함을 이용하자.
  g_label: Generator 를 학습시키기 위한 label 로, Discriminator 를 속여야 한다는 점을 이용하자.

To-do

- **Problem 4.** *train.py*

  [Optional] 4-(f). Language-free training 을 위해 pseudo text feature 을 생성하여라. [5pt]

  LAFITE 논문에서는 language-free text-to-image generation 을 위해, pseudo text feature 를

  Fixed perturbations To generate pseudo text feature $\mathbf{h}'$, we propose to perturb the image feature $f_{\text{img}}(\mathbf{x})$ with adaptive Gaussian noise:

  $$\mathbf{h}' = \tilde{\mathbf{h}}/\|\tilde{\mathbf{h}}\|_2, \quad \tilde{\mathbf{h}} = f_{\text{img}}(\mathbf{x}) + \xi\epsilon\|f_{\text{img}}(\mathbf{x})\|_2/\|\epsilon\|_2, \quad (1)$$

  where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the Gaussian noise, $\xi > 0$ is a fixed hyper-parameter representing the level of perturbations, $\|\cdot\|_2$ denotes L2 norm. The added Gaussian noise is *adaptive* in the sense that it is normalized to a hyper-sphere, then re-scaled by the norm of image feature. We can prove that, with the adaptive noise, our LAFITE$_{\text{G}}$ can generate $\mathcal{H}(\mathbf{x})$ with a high probability which depends on ξ, c and d. The formal theorem and its proof are provided in the Appendix.

  와 같이 생성한다. 식 (1)을 구현해, train_step 내의 빈칸을 채우자.

To-do

- [Optional] Problem 5.

  앞 내용 구현과 별개로, 다른 논문의 모델을 구현한 경우 점수 부여 예정. [40pt]

  단순히 멘토들이 준 구조에서 hyperparameter 를 변경하거나 모델 층 수를 늘린 것은 인정되지 않음.

  새로운 논문 또는 방법을 이용하여 모델을 구현하여 Text to Image generation을 성공하면 점수 인정.

  멘토가 제작한 지금의 모델보다 성능이 좋거나 비슷할 경우 점수 부여.

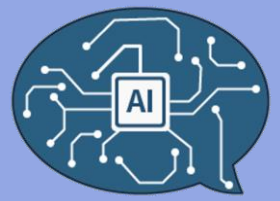
To-do

- [Optional] Problem 6.

  FID와 IS를 직접 측정하는 코드를 작성하여 제출하면 점수 부여 예정. [10pt]

  사전에 제공하는 Test 데이터로 측정해야 한다.(다른 데이터로 측정하는 것은 인정되지 않음).

  멘토들이 측정한 값과 큰 차이가 없을 경우 점수 부여 예정.



감사합니다