Stable Diffusion의 이해와 실습

SCPARK

강사 소개

프리랜서 딥러닝 연구자, 딥러닝 강사

~현재
음악-가사 동기화 @ 가우디오랩

Stable Diffusion 3 풀잎스쿨
생성모델 연구모임 logp(x)
@ 모두의연구소

~2023
버츄얼 휴먼 @ 비브스튜디오스

~2022
음악 음원 분리 @ 가우디오랩

~2019
개인화 TTS @ SM 엔터테인먼트

~2016
안드로이드 앱, 오디오 모듈 개발 등

2023 가을 일본 비와코에서

목차

1. Stable Diffusion 개요

Stable Diffusion 개요

- CompVis 연구실에서 제안한 latent diffusion을 시작으로 HuggingFace의 diffusers 라이브러리에서 지원하여 확장성이 늘어났으며, WebUI, ComfyUI의 등장으로 유저층이 비개발자, 아티스트, 아마추어까지 확대되어 이미지 생성모델로서 가장 사랑받는 모델 중 하나가 되었음

- OpenAI의 DALL-E, Google의 Imagen, Microsoft의 Designer, ImageArt의 Midjourney와는 다르게 오픈소스라는 강력한 장점이 있어 널리 대중들에게 fine-tuning되어 사용됨

- Stability.ai에서 배포한 기본 모델 자체는 이미지 퀄리티가 다른 상용 서비스에 비해 만족스럽지 못하지만 fine-tuning을 통해 특정 주제에 사용될 수 있는 이미지를 만드는데 더욱 치중할 수 있고, Civitai 등의 커뮤니티 사이트를 통해서 fine-tuning된 모델들이 활발히 공유되면서 유저층을 넓혀옴

- 오픈소스라는 장점은 연구분야에도 매우 큰 장점으로 작용하여 생성 속도 최적화, 퀄리티 최적화, 인물 정체성 유지 등의 다양한 주제들이 stable diffusion을 기반으로 연구되고 있음

Rombach, Robin, et al. "High-resolution image synthesis with latent diffusion models." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

Stable Diffusion World

General Generative Models

- Diffusion 모델

  DDPM
  Score-based
  SDE, ODE
  ADM
  EDM

- Flow-matching 모델

  Flow-matching
  Rectified Flow

Stable Diffusion

- Core Model
  SD 1.X
  github.com/CompVis/stable-diffusion
  SD 2.X
  github.com/Stability-AI/stablediffusion
  SDXL
  https://github.com/Stability-AI/generative-models
  SD 3

- Fine-tuning, Adapter Model
  Lora
  ControlNet
  DreamBooth
  Textual Inversion
  IPAdapter

Application

- UI 개발

  WebUI (AUTOMATIC1111)
  github.com/AUTOMATIC1111/stable-diffusion-webui

  ComfyUI
  github.com/comfyanonymous/ComfyUI

- Training, Inference 시스템 개발

  Diffusers
  github.com/huggingface/diffusers

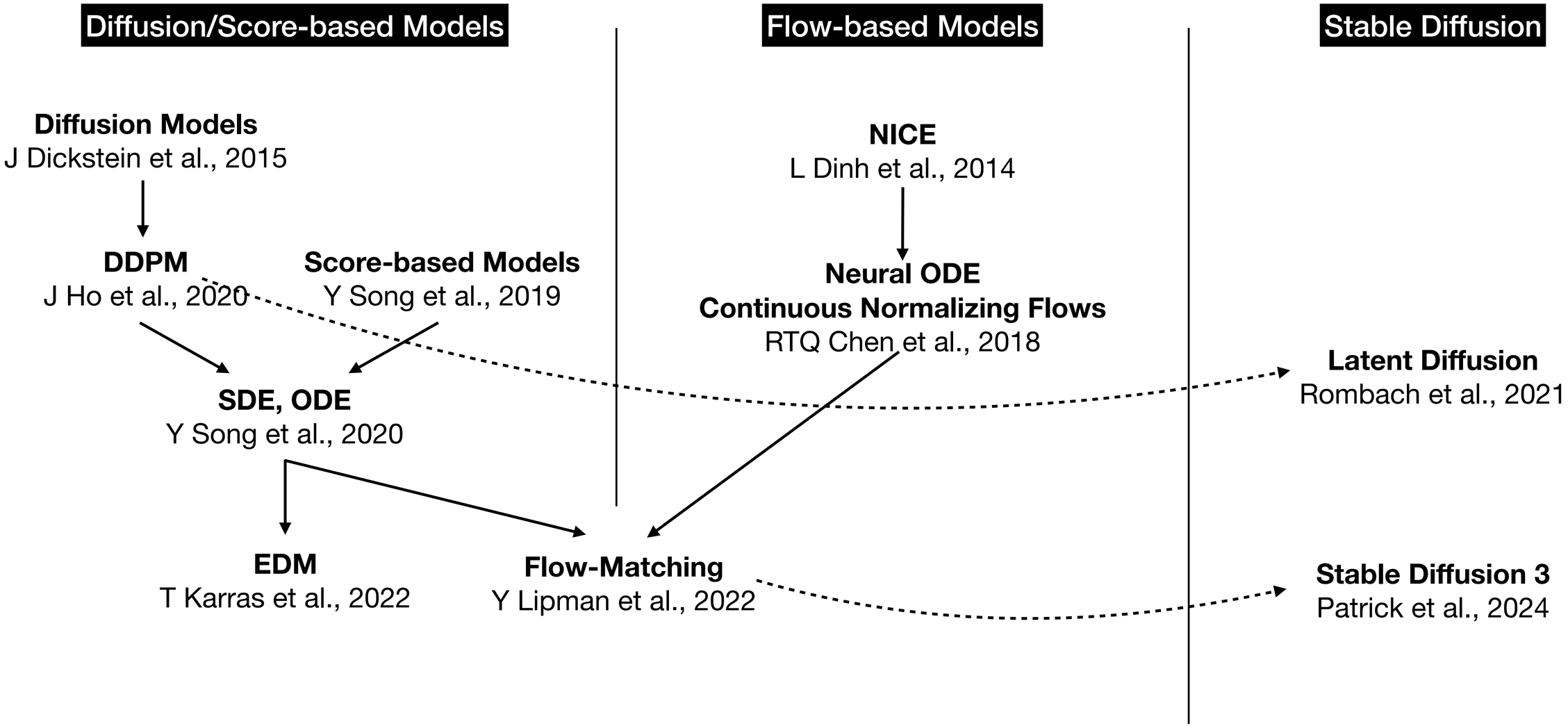
  sd-scripts
  github.com/kohya-ss/sd-scripts

- Community, 자동화 시스템 개발

  CIVITAI
  civitai.com

  TensorArt
  https://tensor.art

Generative Models & Stable Diffusion



SD 1.x, SD 2.x, SDXL and SD3

	SD 1.x	SD 2.x	SDXL	SD3
제작단체	CompVis	Stability.ai	Stability.ai	Stability.ai
Github	https://github.com/CompVis/stable-diffusion	https://github.com/Stability-AI/stablediffusion	https://github.com/Stability-AI/generative-models	없음
HuggingFace	https://huggingface.co/CompVis/stable-diffusion-v1-4 https://huggingface.co/runwayml/stable-diffusion-v1-5	https://huggingface.co/stabilityai/stable-diffusion-2 https://huggingface.co/stabilityai/stable-diffusion-2-1	https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0	https://huggingface.co/stabilityai/stable-diffusion-3-medium
해상도	512	512, 768	1024	1024
모델 구조	Diffusion/U-net	Diffusion/U-net	Diffusion/U-net	Flow-Matching/DiT
장점	가장 가벼움, 널리 사용됨	SD 1.x보다 퀄리티가 좋음	고화질로 사실적 표현 가능	가장 퀄리티가 좋고 고화질
단점	사실적 표현이 미흡	많이 안쓰임	무거움	Medium만 weight 공개, Large는 API로 공개

Diffusers

- Hugging Face에서 관리되고 있는 diffusion관련 라이브러리
- Stable diffusion이 사실상 diffusers에 의해서 배포 및 관리되고 있음
- WebUI, ComfyUI, sd-scripts 등 모든 training/inference 관리 도구들이 diffusers를 기준으로하여 만들어져 있음
- Text를 입력해 image를 만들기까지의 일련의 과정들을 수행하는 Pipeline이라는 class로 추상화되어 명확하고 구조적으로 사용할 수 있음

```python
import torch
from diffusers import StableDiffusionPipeline

# 모델을 로드하고 설정합니다.
model_id = "CompVis/stable-diffusion-v1-4"
device = "cuda" if torch.cuda.is_available() else "cpu"

# Stable Diffusion Pipeline을 초기화합니다.
pipe = StableDiffusionPipeline.from_pretrained(model_id)
pipe = pipe.to(device)

# 텍스트 설명을 정의합니다.
prompt = "A futuristic cityscape at sunset, with flying cars and skyscrapers"

# 이미지를 생성합니다.
with torch.autocast("cuda"):
    image = pipe(prompt).images[0]

# 이미지를 파일로 저장하거나 표시할 수 있습니다.
image.save("generated_image.png")
image.show()
```

https://github.com/huggingface/diffusers

Diffusers - 실습

- 실습 ‘diffusers-example/stable diffusion by diffusers.ipynb’

Generate Image

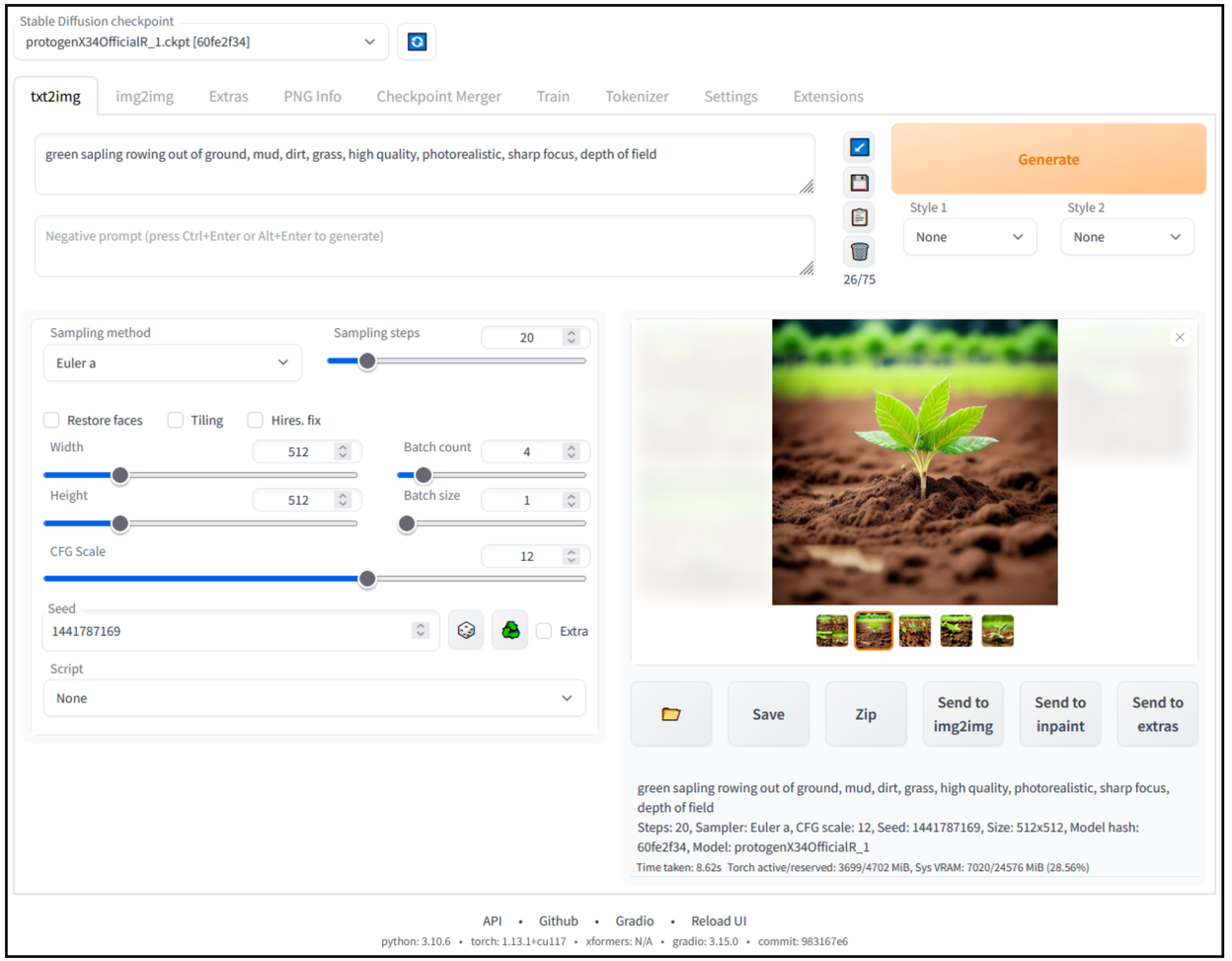
```
# prompt from https://civitai.com/images/16407281

prompt = "(RAW photo, best quality), (realistic, photo-realistic:1.4), masterpiece, extremely beautiful, extremely d
n_prompt = "3d, sepia, painting, cartoons, sketch, watercolor, flat color, (worst quality:2), (low quality:2), (norm

images = pipe(prompt=prompt,
              negative_prompt=n_prompt,
              height=512,
              width=512,
              num_inference_steps=20,
              guidance_scale=7,
              num_images_per_prompt=8
        ).images

import matplotlib.pyplot as plt
fig, axs = plt.subplots(2, 4, figsize=(15, 8))
for i, ax in enumerate(axs.flatten()):
    ax.imshow(images[i])
    ax.axis('off')

plt.tight_layout()
plt.show()
```

```
The following part of your input was truncated because CLIP can only handle sequences up to 77 tokens: [', hair orn
ament, long sleeves, white shirt, upper body, necktie, black eyes, lips, realistic, kooo 1 2 3, lora chaewo']
```

```
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a
different prompt and/or seed.
```

WebUI(AUTOMATIC1111)

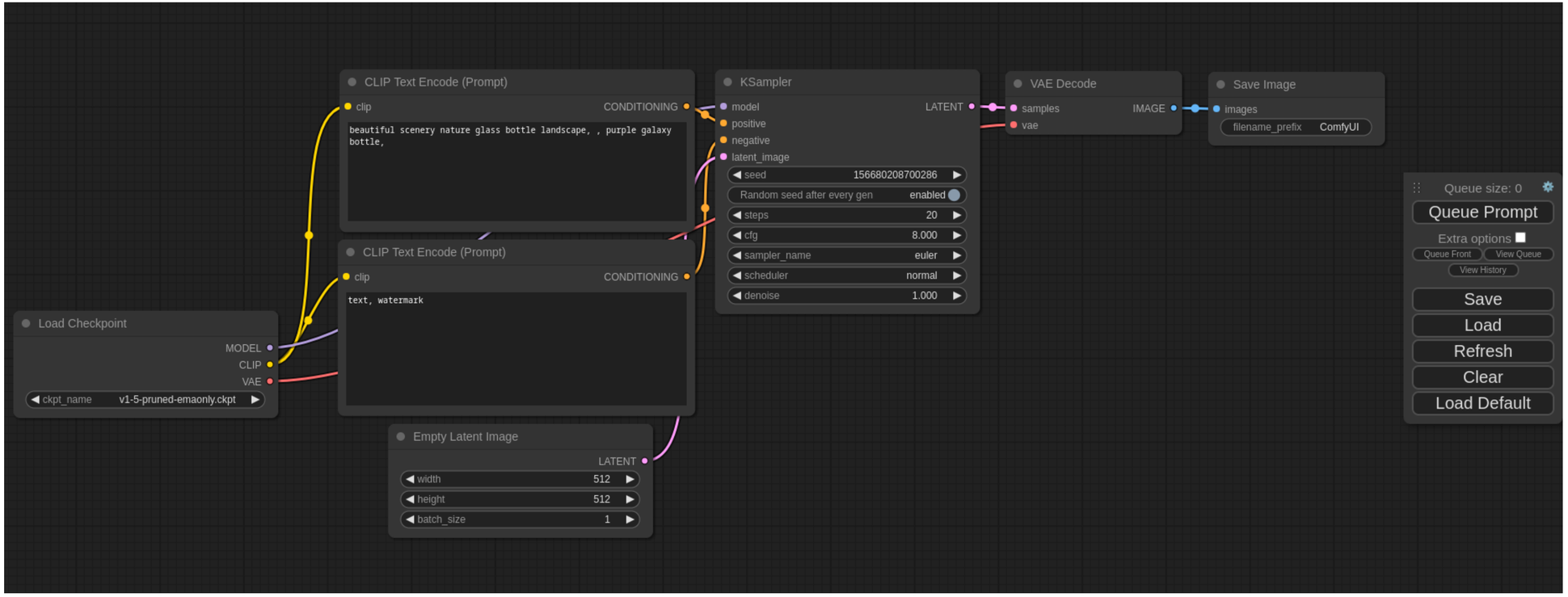
- Stable diffusion을 웹페이지로 구현된 프로그램으로 이용할 수 있도록 해주는 UI
- 만든 사람의 github ID가 AUTOMATIC1111
- 텍스트 상자, 슬라이드 등의 다양한 컨트롤러를 이용하여 직관적으로 기능들을 수행할 수 있음
- 대부분의 아마추어 stable diffusion 유저들이 이용
- 설치가 쉽고 배우기 간단하지만 stable diffusion의 원리를 파악하기 힘들며, 확장성이 ComfyUI에 비해 떨어짐



https://github.com/AUTOMATIC1111/stable-diffusion-webui

ComfyUI

- Stable diffusion을 구성하는 모듈들 (VAE Encoder/Decoder, Text Encoder, U-net), 입/출력 모듈 (Model Load, Latent Init., Image View, Image Save) 등을 컴포넌트화하여 pipeline을 커스터마이징할 수 있도록 만듬
- WebUI에 비해 초보의 경우 배우기 어렵고, 각 컴포넌트에 대한 기능의 이해가 필요하나, 확장성이 뛰어나고 매우 직관적임
- Pipeline이 복잡해지면 컴포넌트들을 연결하는게 번거로워지므로 code로 작성하는게 나음

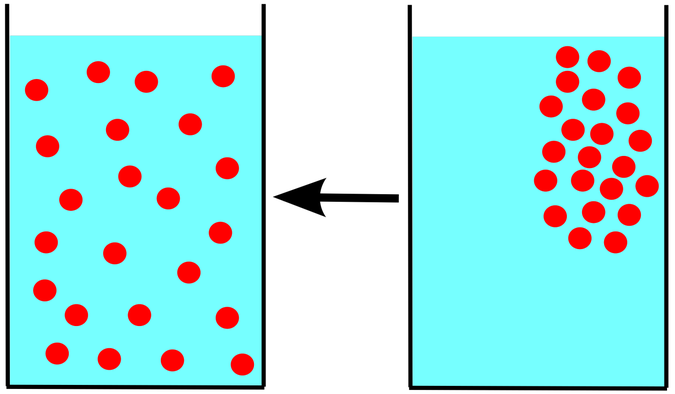


https://github.com/comfyanonymous/ComfyUI

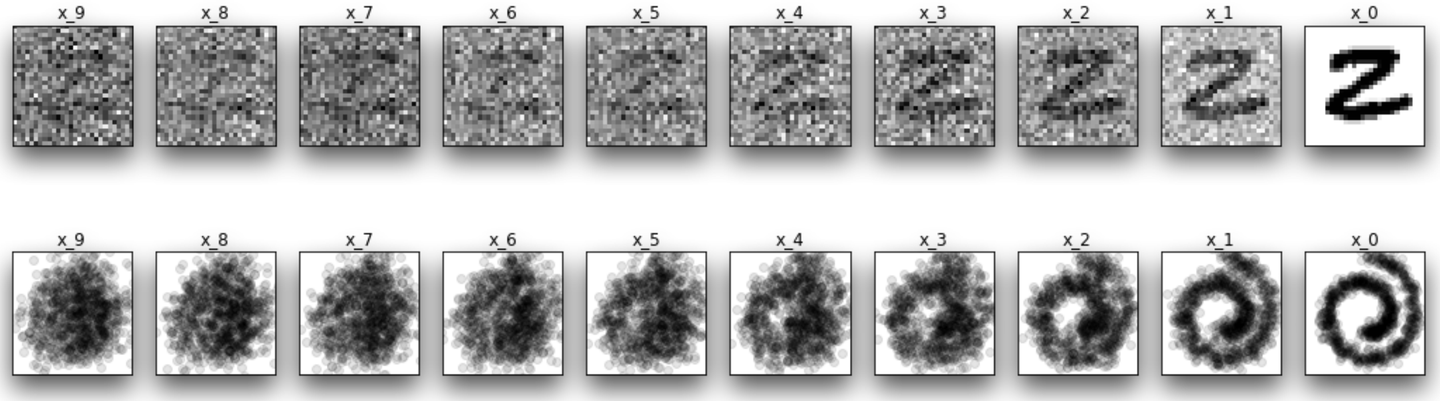
2. Diffusion의 이해와 실습

Diffusion?

- Sohl-Dickstein, Jascha의 논문 Deep unsupervised learning using nonequilibrium thermodynamics에서 제안



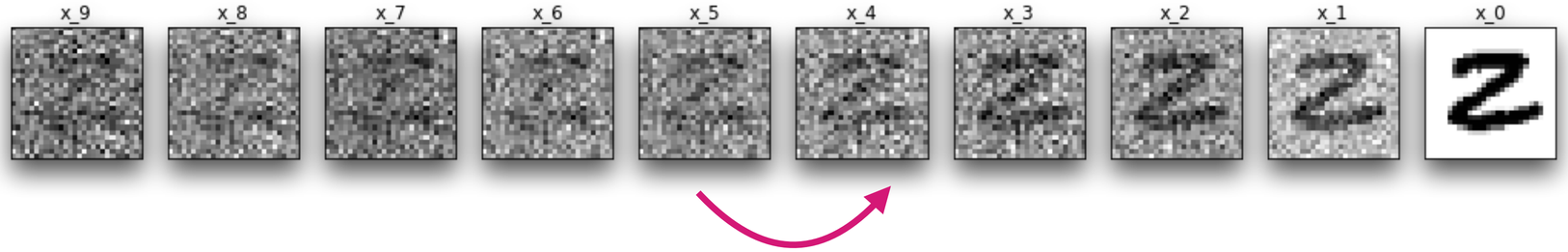
Diffusion
https://en.wikipedia.org/wiki/Diffusion
(Flipped)



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion!

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$



Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

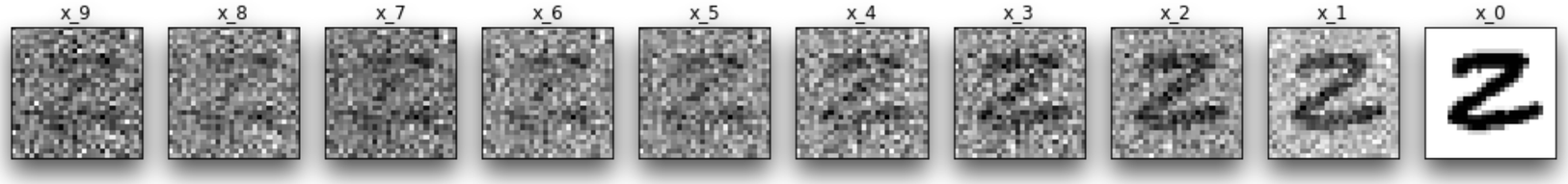
Backward Process (Neural Networks) $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$

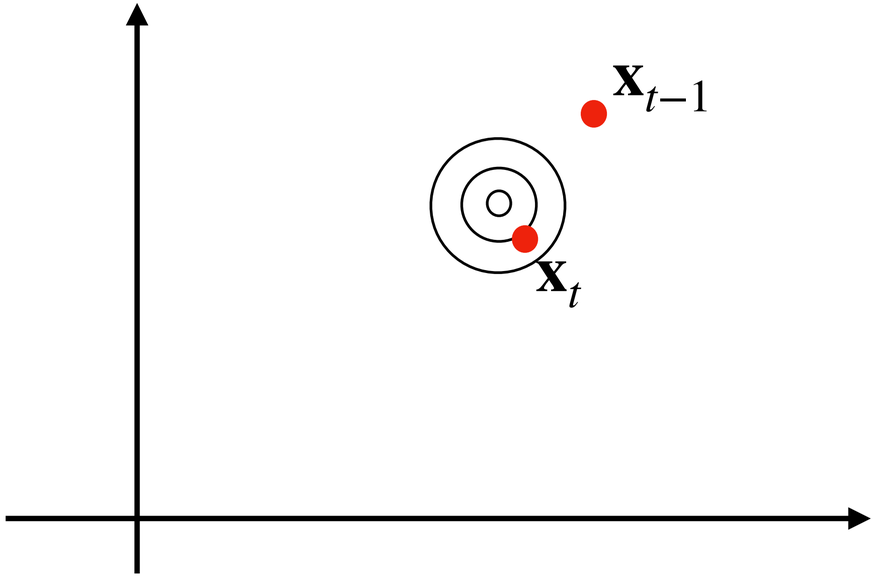
Loss Function $D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$

Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion - Forward Process

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$





Distribution of $\mathbf{x}_t$ at an arbitrary timestep t in closed form

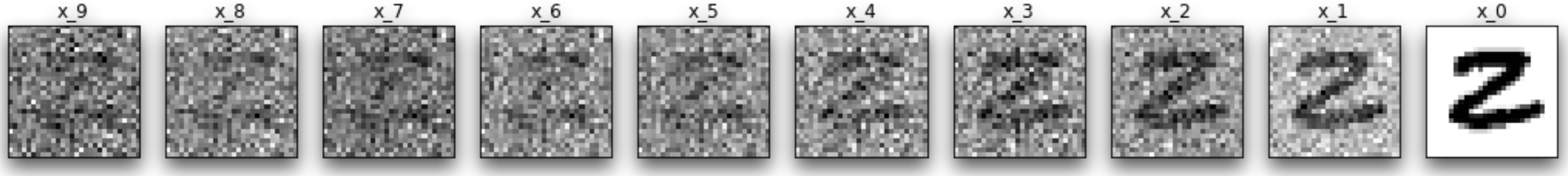
$q\left(\mathbf{x}_t \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \left(1-\bar{\alpha}_t\right)\mathbf{I}\right)$, where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$

식 유도는Lil'Log 참고
https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

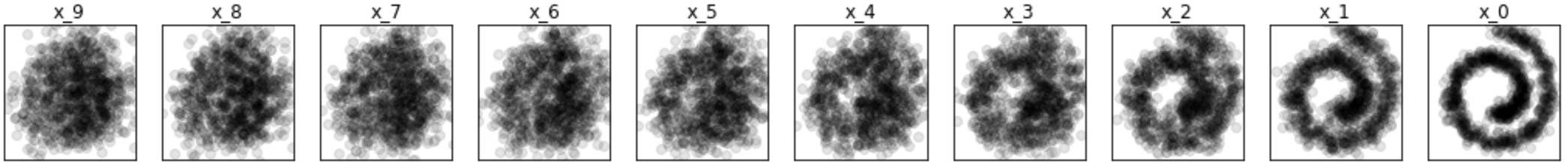
Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion - Forward Process

MNIST single data, $\beta = 0.2$, $T = 10$



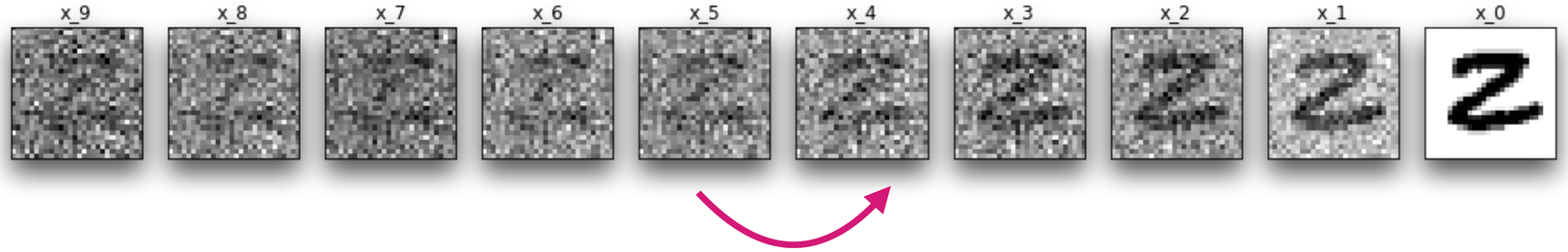
Swiss roll dataset, $\beta = 0.05$, $T = 10$



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion - Posterior

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$



Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

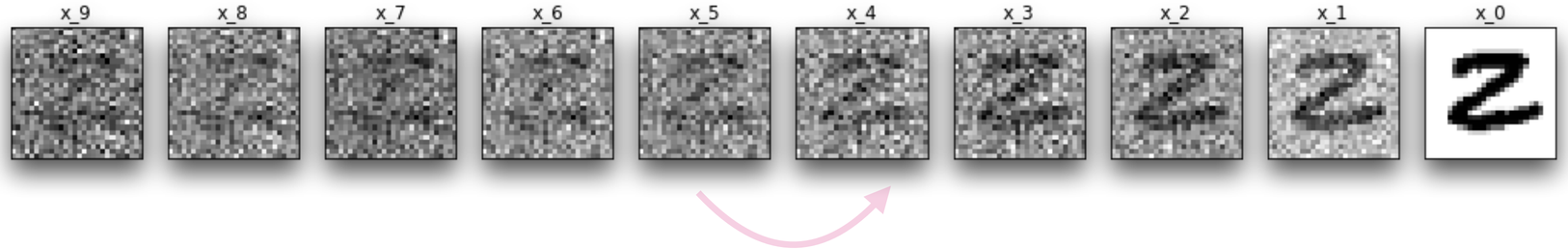
where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0) q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)}$$ by Bayes' Rule

Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion - Backward Process

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$

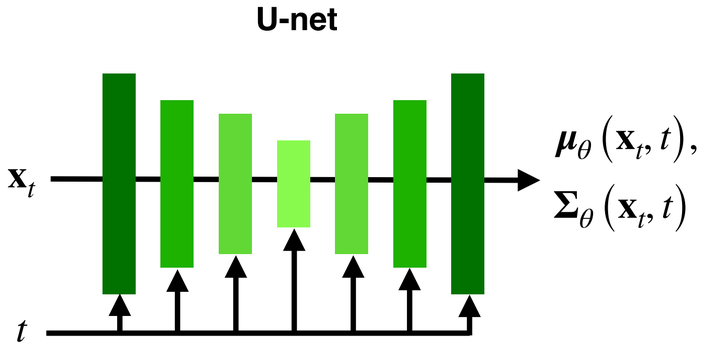


Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

Backward Process (Neural Networks) $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$

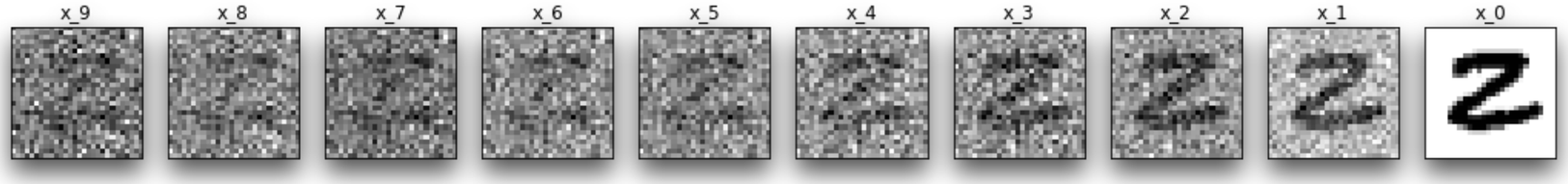
Loss Function $D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion - KL-Divergence

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$

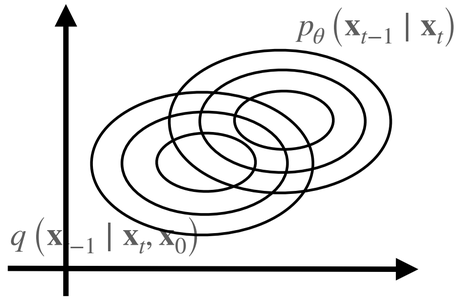


Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

Backward Process (Neural Networks) $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$

Loss Function $D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion - Output Samples

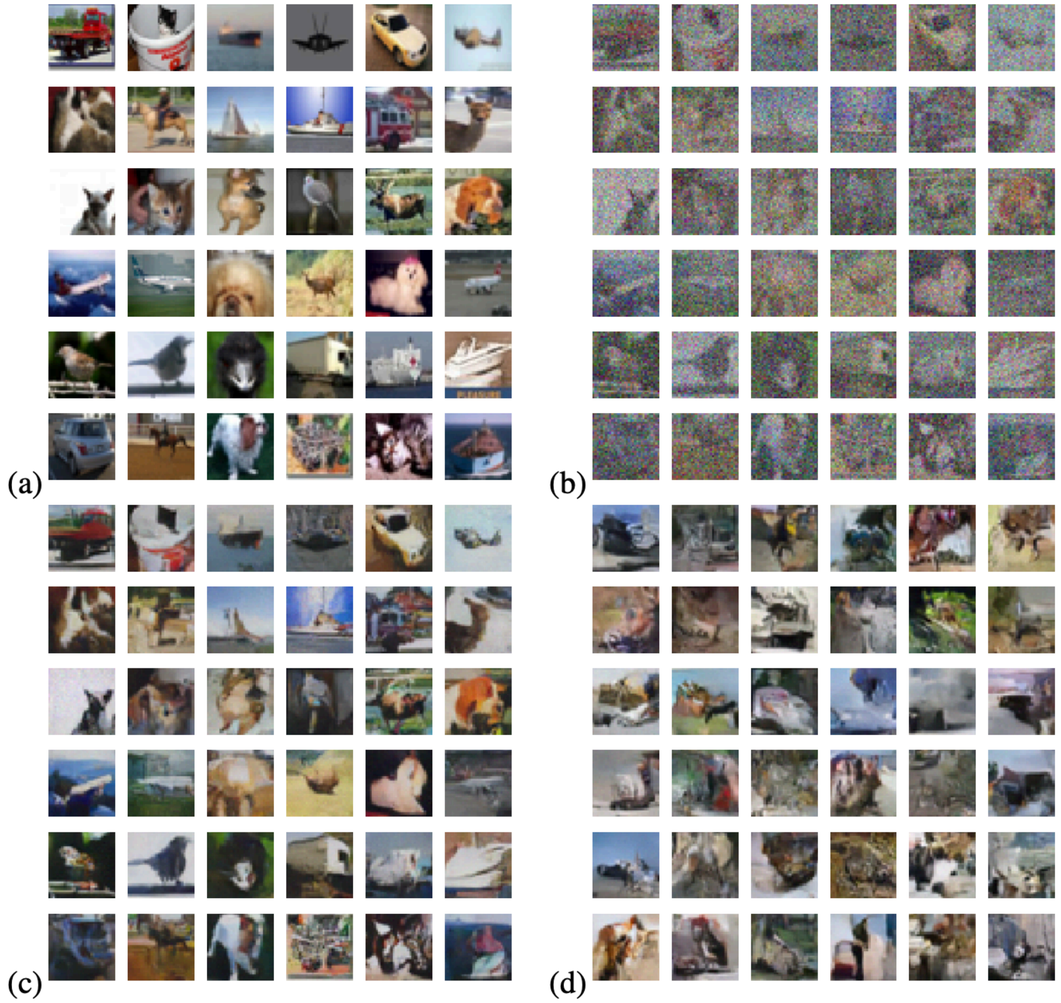


Figure 3. The proposed framework trained on the CIFAR-10 (Krizhevsky & Hinton, 2009) dataset. *(a)* Example holdout data (similar to training data). *(b)* Holdout data corrupted with Gaussian noise of variance 1 (SNR = 1). *(c)* Denoised images, generated by sampling from the posterior distribution over denoised images conditioned on the images in (b). *(d)* Samples generated by the diffusion model.

Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion - DDPM

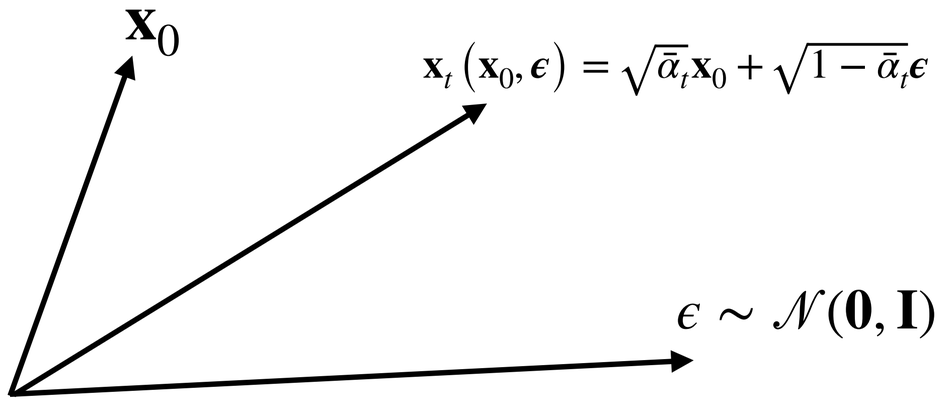
- Jonathan Ho의 논문 Denoising diffusion probabilistic models에서 제안

Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)=\mathcal{N}\left(\mathbf{x}_{t-1} ; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

Distribution of $\mathbf{x}_t$ at an arbitrary timestep t in closed form

$q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)=\mathcal{N}\left(\mathbf{x}_t ; \sqrt{\bar{\alpha}_t} \mathbf{x}_0,\left(1-\bar{\alpha}_t\right) \mathbf{I}\right)$, where $\alpha_t:=1-\beta_t$ and $\bar{\alpha}_t:=\prod_{s=1}^{t} \alpha_s$

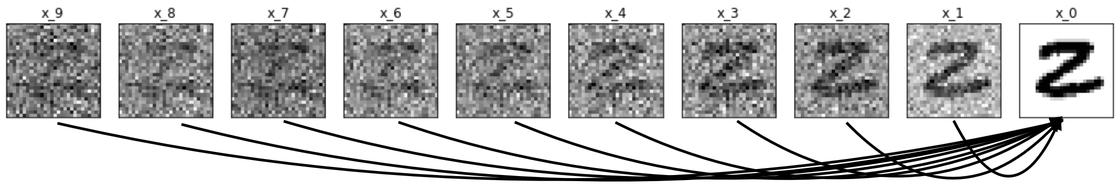
Loss Function $L_{\text{simple}}(\theta):=\mathbb{E}_{t, \mathbf{x}_0, \epsilon}\left[\left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0+\sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon}, t\right)\right\|^2\right]$



$\mathbf{x}_t$ is a linear combination of $\mathbf{x}_0$ and ϵ

```
106       def p_loss(self, model, x_0, t, noise=None):
107           if noise is None:
108               noise = torch.randn_like(x_0)          generate ϵ
109
110           x_noise = self.q_sample(x_0, t, noise)     sample x_t
111           x_recon = model(x_noise, t)                predict ϵ
112
113           return F.mse_loss(x_recon, noise)
```

https://github.com/rosinality/denoising-diffusion-pytorch/blob/master/diffusion.py

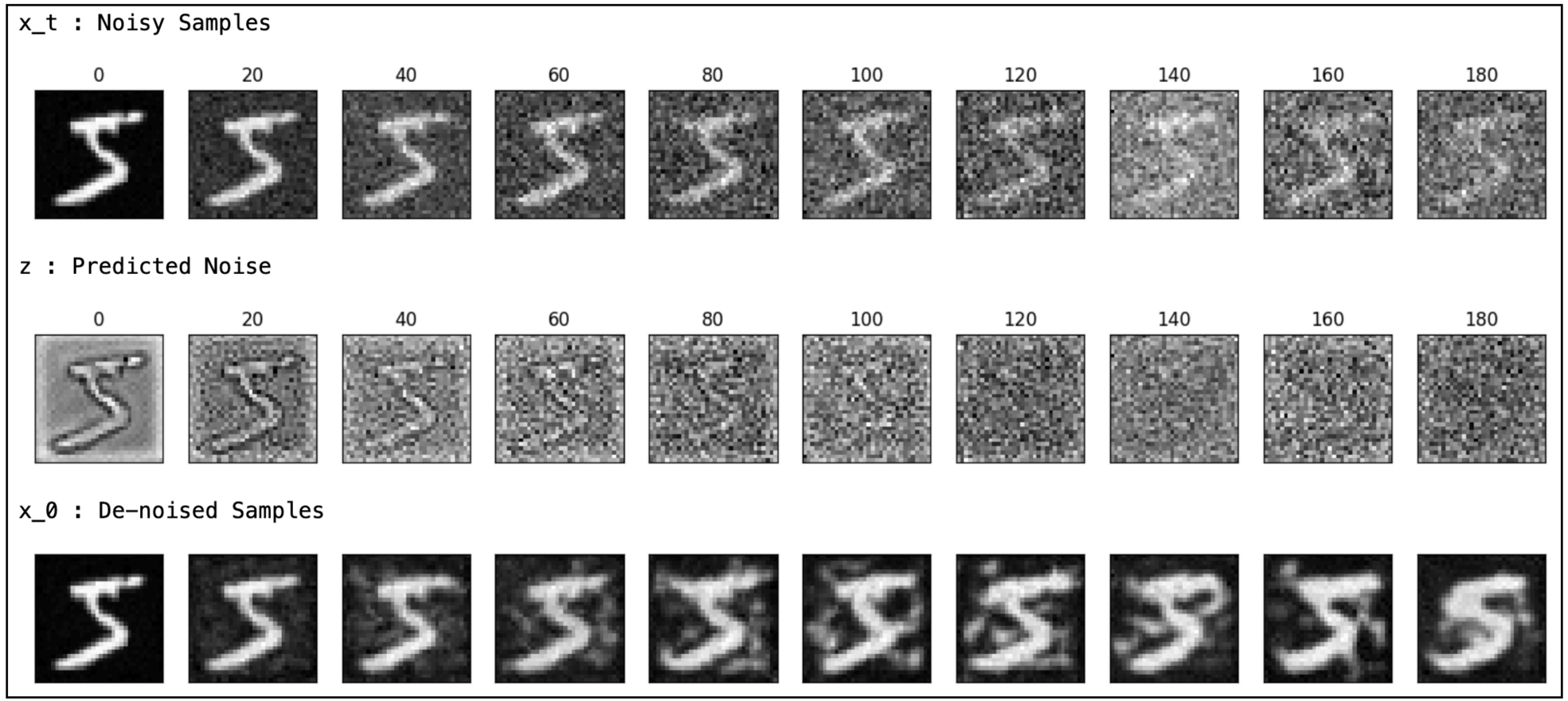


Predict ϵ (or $\mathbf{x}_0$) at each step

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in Neural Information Processing Systems* 33 (2020): 6840-6851.

Diffusion - 실습 - DDPM in 100 Lines

- Diffusers 라이브러리를 이용하여 Unet을 구성하고 DDPM 알고리즘을 이용하여 트레이닝
실습 “diffusers-example/ddpm in 100 lines.ipynb”



Image에 noise를 추가하고 모델로 noise를 prediction하여 de-noising하는 모습

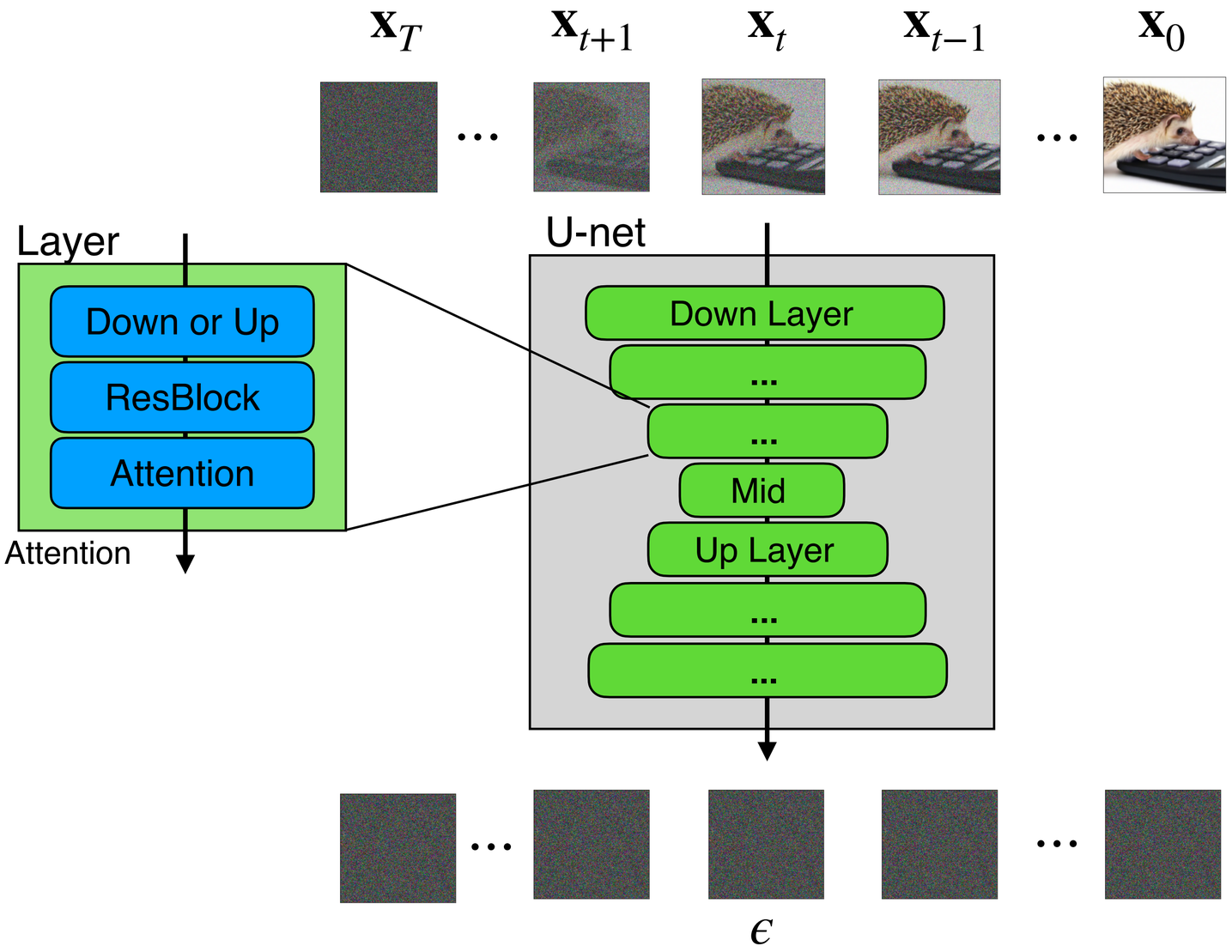
Diffusion - 실습 - DDPM in 100 Lines - U-net 모델 구조

- U-net은 Down Layers, Mid Layer, Up Layers의 대칭구조로 이루어짐

- 모든 Layer는 Down or Up, ResBlock, Attention 모듈로 이루어짐

- Down 모듈은 이미지를 H, W 각각 2배로 줄이고, ResBlock은 residual connection을 이용한 convolution networks로 이루어짐

- 특정 depth 이하의 layer에서는 attention 모듈을 이용하여 모델 flexibility 강화

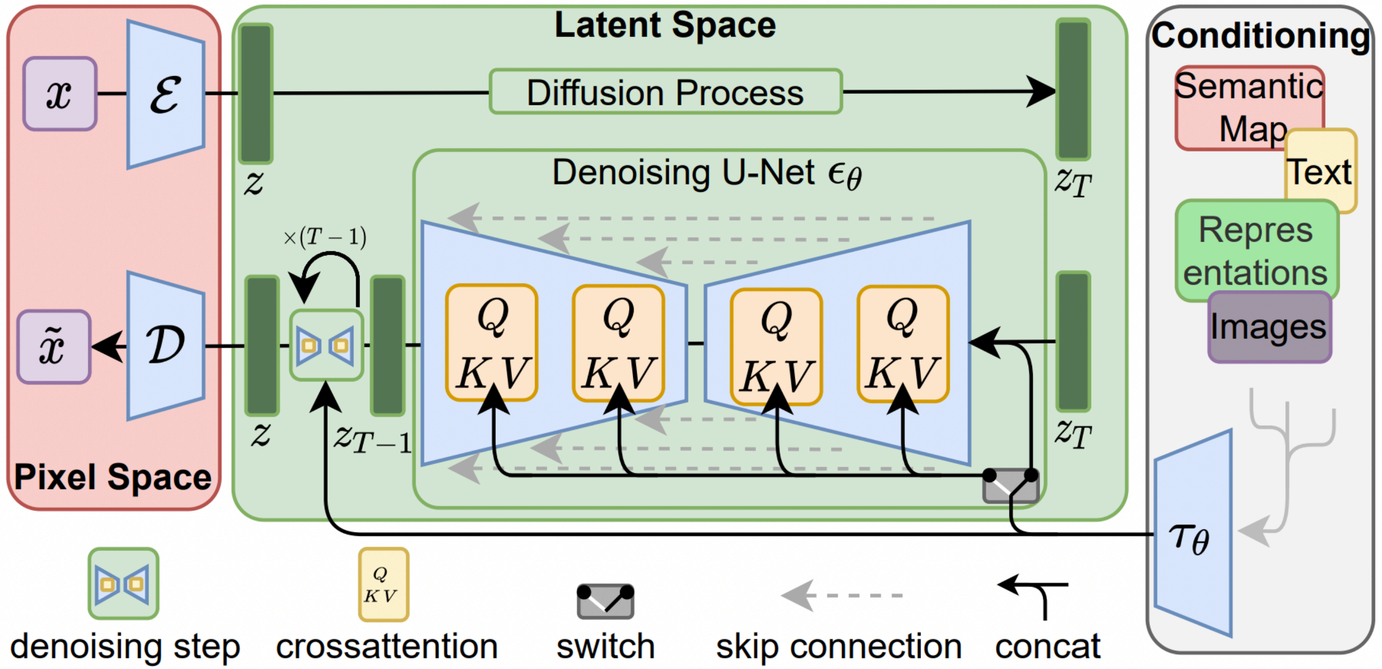


3. Stable Diffusion

Model Architecture

Stable Diffusion 전체 구조

- VAE Encoder/Decoder, Denoising U-net, Conditioning 부분으로 나누어져 있음
- VAE는 image를 encoding, 압축하여 latent로 보내고 다시 image로 복원하는 기능
- Denoising U-net은 noise로부터 latent 데이터를 생성하는 기능
- Conditioning은 latent 데이터를 생성하는데 있어 필요한 조건 (Text, Image)을 준비하는 기능을 담당



Rombach, Robin, et al. "High-resolution image synthesis with latent diffusion models." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

Stable Diffusion - Model Architecture - VAE 구조

- SRGAN에서 L1 loss와 함께 VGG19를 이용한 perceptual loss, discriminator를 이용한 adversarial loss를 사용
- Interpolation을 이용한 bicubic과 L1 loss만을 이용한 SRResNet 대비 선명해지는 것을 볼 수 있음

bicubic
(21.59dB/0.6423)

SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

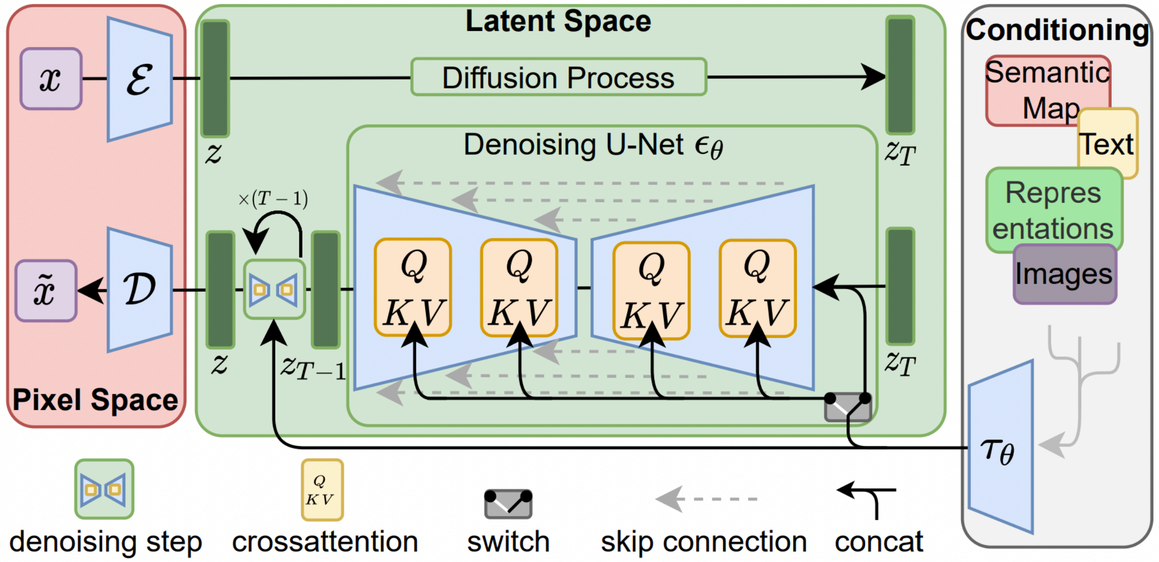
Stable Diffusion - Training

- **Stable diffusion은 training은 2 stages로 나뉨**

- **1 stage : VAE를 트레이닝하는 단계**
  - Training dataset의 image x를 이용,
  VAE network를 통해 encoder $\mathscr{E}$/decoder $\mathscr{D}$를 트레이닝
  x : Image
  $z = \mathscr{E}(x)$: Latent, Encoded Image
  $\tilde{x} = \mathscr{D}(z)$: Reconstructed Image

- **2 stage : Denoising U-net을 트레이닝하는 단계**
  - Training dataset의 image x와 text y를 이용,
  $z_0 = \mathscr{E}(x)$: Latent, Encoded Image
  $z_t = \sqrt{\bar{\alpha}_t} z_0 + (1 - \bar{\alpha}_t)\epsilon$: Noisy Latent
  $\tau_\theta(y)$: Text Encoding Sequence
  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$: Noise
  $t \sim \mathcal{U}(0,999)$: Time Index

$$L_{LDM} := \mathbb{E}_{\mathscr{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}\left[\left\| \epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y)) \right\|_2^2 \right]$$



Rombach, Robin, et al. "High-resolution image synthesis with latent diffusion models." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

Stable Diffusion - Denoising U-net - Loss Functions

```
1012      def p_losses(self, x_start, cond, t, noise=None):
1013          noise = default(noise, lambda: torch.randn_like(x_start))
1014          x_noisy = self.q_sample(x_start=x_start, t=t, noise=noise)
1015          model_output = self.apply_model(x_noisy, t, cond)
1016
1017          loss_dict = {}
1018          prefix = 'train' if self.training else 'val'
1019
1020          if self.parameterization == "x0":
1021              target = x_start
1022          elif self.parameterization == "eps":
1023              target = noise
1024          else:
1025              raise NotImplementedError()
1026
1027          loss_simple = self.get_loss(model_output, target, mean=False).mean([1, 2, 3])
1028          loss_dict.update({f'{prefix}/loss_simple': loss_simple.mean()})
```

- x_start : image latent
  cond : text encoding sequence
  t : time index
  를 입력 받아 loss를 계산

- x_noisy는 marginal distribution $q(x_t \mid x_0)$ 에서 샘플링

- x_noisy와 cond 그리고 t를 Denoising U-net 모델에 입력한 뒤 나온 결과 model_output이 샘플링된 noise를 regression하도록 MSE loss 설정

https://github.com/CompVis/stable-diffusion/blob/main/ldm/models/diffusion/ddpm.py

Stable Diffusion 전체 구조 - 기본 모듈 - Resnet Block

```python
class ResnetBlock(nn.Module):
    def __init__(self, *, in_channels, out_channels=None, conv_shortcut=False,
                 dropout, temb_channels=512):
        super().__init__()
        self.in_channels = in_channels
        out_channels = in_channels if out_channels is None else out_channels
        self.out_channels = out_channels
        self.use_conv_shortcut = conv_shortcut

        self.norm1 = Normalize(in_channels)
        self.conv1 = torch.nn.Conv2d(in_channels,
                                     out_channels,
                                     kernel_size=3,
                                     stride=1,
                                     padding=1)
        if temb_channels > 0:
            self.temb_proj = torch.nn.Linear(temb_channels,
                                             out_channels)
        self.norm2 = Normalize(out_channels)
        self.dropout = torch.nn.Dropout(dropout)
        self.conv2 = torch.nn.Conv2d(out_channels,
                                     out_channels,
                                     kernel_size=3,
                                     stride=1,
                                     padding=1)
        if self.in_channels != self.out_channels:
            if self.use_conv_shortcut:
                self.conv_shortcut = torch.nn.Conv2d(in_channels,
                                                     out_channels,
                                                     kernel_size=3,
                                                     stride=1,
                                                     padding=1)
            else:
                self.nin_shortcut = torch.nn.Conv2d(in_channels,
                                                    out_channels,
                                                    kernel_size=1,
                                                    stride=1,
                                                    padding=0)
```

Conv. Block 1

Conv. Block 2

```python
    def forward(self, x, temb):
        h = x
        h = self.norm1(h)
        h = nonlinearity(h)
        h = self.conv1(h)

        if temb is not None:
            h = h + self.temb_proj(nonlinearity(temb))[:,:,None,None]

        h = self.norm2(h)
        h = nonlinearity(h)
        h = self.dropout(h)
        h = self.conv2(h)

        if self.in_channels != self.out_channels:
            if self.use_conv_shortcut:
                x = self.conv_shortcut(x)
            else:
                x = self.nin_shortcut(x)

        return x+h
```

Conv. Block 1

Conv. Block 2

Residual Connection

- 실습 “diffusers-example/resnet block.ipynb”

- Resnet Block은 2개의 conv. block과 residual connection으로 구성
- Conv. block은 normalization (group), activation(swish), convolution으로 구성
- Residual connection은 channel을 맞추기 위한 convolution을 input x에 적용 후 hidden h와 더하는 연산 수행

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - 전체 구조 - 기본 모듈 - Attention Block

```python
class CrossAttention(nn.Module):
    def __init__(self, query_dim, context_dim=None, heads=8, dim_head=64, dropout=0.):
        super().__init__()
        inner_dim = dim_head * heads
        context_dim = default(context_dim, query_dim)

        self.scale = dim_head ** -0.5
        self.heads = heads

        self.to_q = nn.Linear(query_dim, inner_dim, bias=False)
        self.to_k = nn.Linear(context_dim, inner_dim, bias=False)
        self.to_v = nn.Linear(context_dim, inner_dim, bias=False)

        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, query_dim),
            nn.Dropout(dropout)
        )

    def forward(self, x, context=None, mask=None):
        h = self.heads

        q = self.to_q(x)
        context = default(context, x)
        k = self.to_k(context)
        v = self.to_v(context)

        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> (b h) n d', h=h), (q, k, v))

        sim = einsum('b i d, b j d -> b i j', q, k) * self.scale  # QK^T / sqrt(d_k)

        if exists(mask):
            mask = rearrange(mask, 'b ... -> b (...)')
            max_neg_value = -torch.finfo(sim.dtype).max
            mask = repeat(mask, 'b j -> (b h) () j', h=h)
            sim.masked_fill_(~mask, max_neg_value)

        # attention, what we cannot get enough of
        attn = sim.softmax(dim=-1)  # softmax(·)

        out = einsum('b i j, b j d -> b i d', attn, v)  # · V
        out = rearrange(out, '(b h) n d -> b n (h d)', h=h)
        return self.to_out(out)
```

- 실습 ‘diffusers-example/attention block.ipynb’
- Attention Block은 일반적인 Transformer의 attention 연산으로 구성됨
- Text Encoding 정보는 forward의 context 변수를 통해서 전달되고 key와 value로 사용됨
- VAE에서는 Text Encoding의 정보가 None이며, 이 경우 입력 x (image encoding) 정보가 query, key, value로 모두 사용됨
- $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/attention.py

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Stable Diffusion

Model Architecture - VAE

Stable Diffusion - Model Architecture - VAE 구조

- VAE는 Encoder 와 Decoder로 나누어져 있음
- Encoder 는 주어진 이미지 x를 encoding하여 latent z를 만듬
  예) 3x512x512 -> 4x64x64
- Decoder는 주어진 latent z를 다시 이미지 x로 복원
  Dimension reduction으로 인해 정보 손실이 발생하므로 100% 완벽하게 복원하는 것은 불가능
- Perceptually 원본과 가깝게 복원하기 위해 L1 Loss 뿐만 아니라, VGG Perceptual Loss, Adversarial Loss를 함께 사용함
- Regularization을 위해 KL-divergence loss를 추가로 사용함

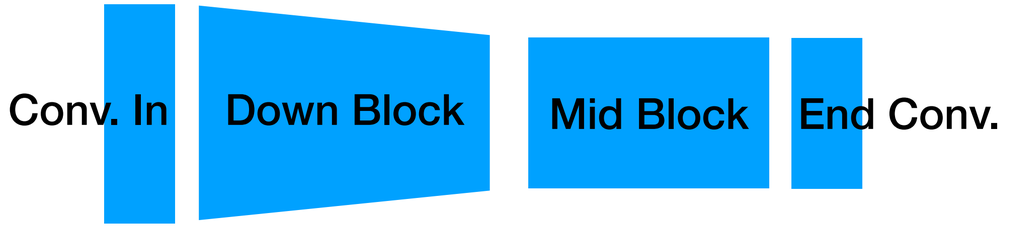
```python
class AutoencoderKL(pl.LightningModule):
    def __init__(self,
                 ddconfig,
                 lossconfig,
                 embed_dim,
                 ckpt_path=None,
                 ignore_keys=[],
                 image_key="image",
                 colorize_nlabels=None,
                 monitor=None,
                 ):
        super().__init__()
        self.image_key = image_key
        self.encoder = Encoder(**ddconfig)
        self.decoder = Decoder(**ddconfig)
        self.loss = instantiate_from_config(lossconfig)
        assert ddconfig["double_z"]
        self.quant_conv = torch.nn.Conv2d(2*ddconfig["z_channels"], 2*embed_dim, 1)
        self.post_quant_conv = torch.nn.Conv2d(embed_dim, ddconfig["z_channels"], 1)
```

```python
    def encode(self, x):
        h = self.encoder(x)
        moments = self.quant_conv(h)
        posterior = DiagonalGaussianDistribution(moments)
        return posterior

    def decode(self, z):
        z = self.post_quant_conv(z)
        dec = self.decoder(z)
        return dec

    def forward(self, input, sample_posterior=True):
        posterior = self.encode(input)
        if sample_posterior:
            z = posterior.sample()
        else:
            z = posterior.mode()
        dec = self.decode(z)
        return dec, posterior
```

https://github.com/CompVis/stable-diffusion/blob/main/ldm/models/autoencoder.py

Stable Diffusion - Model Architecture - VAE 구조 - Encoder

- Encoder는 Conv. In, Down Blocks, Middle Block, End Conv.로 나누어짐
- Conv. In은 input image의 채널을 확장하는 기능을 함
- Down Block는 ResnetBlock, Attention, Downsample로 나누어지며, 점차 1/2씩 H, W의 크기가 줄어듬
- Mid Block은 ResnetBlock, Attention, ResnetBlock으로 나누어짐
- End Conv.는 원하는 z의 dimension을 맞추어 출력하는 기능을 함




```python
class Encoder(nn.Module):
    def __init__(self, *, ch, out_ch, ch_mult=(1,2,4,8), num_res_blocks,
                 attn_resolutions, dropout=0.0, resamp_with_conv=True, in_channels,
                 resolution, z_channels, double_z=True, use_linear_attn=False, attn_type="vanilla",
                 **ignore_kwargs):
        super().__init__()
        if use_linear_attn: attn_type = "linear"
        self.ch = ch
        self.temb_ch = 0
        self.num_resolutions = len(ch_mult)
        self.num_res_blocks = num_res_blocks
        self.resolution = resolution
        self.in_channels = in_channels

        # downsampling
        self.conv_in = torch.nn.Conv2d(in_channels,
                                       self.ch,
                                       kernel_size=3,
                                       stride=1,
                                       padding=1)
```

Conv. In

```python
        curr_res = resolution
        in_ch_mult = (1,)+tuple(ch_mult)
        self.in_ch_mult = in_ch_mult
        self.down = nn.ModuleList()
        for i_level in range(self.num_resolutions):
            block = nn.ModuleList()
            attn = nn.ModuleList()
            block_in = ch*in_ch_mult[i_level]
            block_out = ch*ch_mult[i_level]
            for i_block in range(self.num_res_blocks):
                block.append(ResnetBlock(in_channels=block_in,
                                         out_channels=block_out,
                                         temb_channels=self.temb_ch,
                                         dropout=dropout))
                block_in = block_out
                if curr_res in attn_resolutions:
                    attn.append(make_attn(block_in, attn_type=attn_type))
            down = nn.Module()
            down.block = block
            down.attn = attn
            if i_level != self.num_resolutions-1:
                down.downsample = Downsample(block_in, resamp_with_conv)
                curr_res = curr_res // 2
            self.down.append(down)
```

Down Block

```python
        # middle
        self.mid = nn.Module()
        self.mid.block_1 = ResnetBlock(in_channels=block_in,
                                       out_channels=block_in,
                                       temb_channels=self.temb_ch,
                                       dropout=dropout)
        self.mid.attn_1 = make_attn(block_in, attn_type=attn_type)
        self.mid.block_2 = ResnetBlock(in_channels=block_in,
                                       out_channels=block_in,
                                       temb_channels=self.temb_ch,
                                       dropout=dropout)

        # end
        self.norm_out = Normalize(block_in)
        self.conv_out = torch.nn.Conv2d(block_in,
                                        2*z_channels if double_z else z_channels,
                                        kernel_size=3,
                                        stride=1,
                                        padding=1)
```

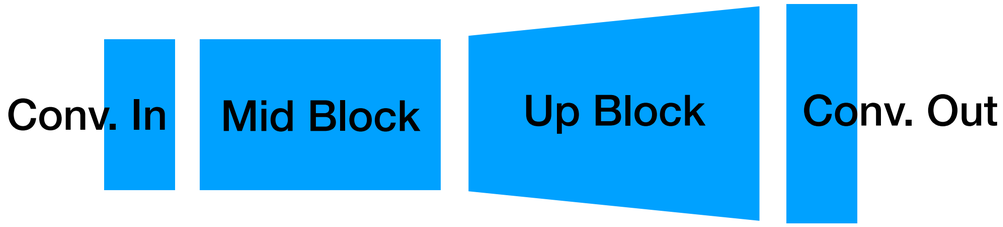
Mid Block

End Conv.

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Model Architecture - VAE 구조 - Decoder

- Decoder는 Encoder와 정확히 대칭되는 구조를 가짐
- Conv. In은 encoded image의 채널을 확장하는 기능을 함
- Mid Block은 ResnetBlock, Attention, ResnetBlock으로 나누어짐
- Up Block는 ResnetBlock, Attention, Upsample로 나누어지며, 점차 2배씩 H, W의 크기를 키움
- Conv out은 원하는 image의 dimension을 맞추어 출력하는 기능을 함




```
462  class Decoder(nn.Module):
463      def __init__(self, *, ch, out_ch, ch_mult=(1,2,4,8), num_res_blocks,
464                   attn_resolutions, dropout=0.0, resamp_with_conv=True, in_channels,
465                   resolution, z_channels, give_pre_end=False, tanh_out=False, use_linear_attn=False,
466                   attn_type="vanilla", **ignorekwargs):
467          super().__init__()
468          if use_linear_attn: attn_type = "linear"
469          self.ch = ch
470          self.temb_ch = 0
471          self.num_resolutions = len(ch_mult)
472          self.num_res_blocks = num_res_blocks
473          self.resolution = resolution
474          self.in_channels = in_channels
475          self.give_pre_end = give_pre_end
476          self.tanh_out = tanh_out
477
478          # compute in_ch_mult, block_in and curr_res at lowest res
479          in_ch_mult = (1,)+tuple(ch_mult)
480          block_in = ch*ch_mult[self.num_resolutions-1]
481          curr_res = resolution // 2**(self.num_resolutions-1)
482          self.z_shape = (1,z_channels,curr_res,curr_res)
483          print("Working with z of shape {} = {} dimensions.".format(
484              self.z_shape, np.prod(self.z_shape)))
485
486          # z to block_in
487          self.conv_in = torch.nn.Conv2d(z_channels,
488                                         block_in,
489                                         kernel_size=3,
490                                         stride=1,
491                                         padding=1)
```

Conv. In

```
493          # middle
494          self.mid = nn.Module()
495          self.mid.block_1 = ResnetBlock(in_channels=block_in,
496                                         out_channels=block_in,
497                                         temb_channels=self.temb_ch,
498                                         dropout=dropout)
499          self.mid.attn_1 = make_attn(block_in, attn_type=attn_type)
500          self.mid.block_2 = ResnetBlock(in_channels=block_in,
501                                         out_channels=block_in,
502                                         temb_channels=self.temb_ch,
503                                         dropout=dropout)
```

Mid Block

```
505          # upsampling
506          self.up = nn.ModuleList()
507          for i_level in reversed(range(self.num_resolutions)):
508              block = nn.ModuleList()
509              attn = nn.ModuleList()
510              block_out = ch*ch_mult[i_level]
511              for i_block in range(self.num_res_blocks+1):
512                  block.append(ResnetBlock(in_channels=block_in,
513                                           out_channels=block_out,
514                                           temb_channels=self.temb_ch,
515                                           dropout=dropout))
516                  block_in = block_out
517                  if curr_res in attn_resolutions:
518                      attn.append(make_attn(block_in, attn_type=attn_type))
519              up = nn.Module()
520              up.block = block
521              up.attn = attn
522              if i_level != 0:
523                  up.upsample = Upsample(block_in, resamp_with_conv)
524                  curr_res = curr_res * 2
525              self.up.insert(0, up) # prepend to get consistent order
526
527          # end
528          self.norm_out = Normalize(block_in)
529          self.conv_out = torch.nn.Conv2d(block_in,
530                                          out_ch,
531                                          kernel_size=3,
532                                          stride=1,
533                                          padding=1)
```

Up Block

Conv. Out

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Model Architecture - VAE 실습

- 실습 “diffusers-example/vae.ipynb”
- 3x900x900 크기의 원본 이미지를 height와 width에 대하여 8배씩 축소하고 channel은 4로 압축 (크기가 8의 배수가 아닌 경우 버림)

3x900x900 4x112x112 3x896x896

Stable Diffusion - Model Architecture - VAE 실습

```
from torchvision import transforms

preprocess = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5]),
])

image_tensor = preprocess(original_image).unsqueeze(0)  # Add batch dimension
print(image_tensor.shape)

# Encode the image to latent space
latent = pipeline.vae.encode(image_tensor).latent_dist.sample()
print(latent.shape)

# Decode the latent space back to image
reconstructed_image = pipeline.vae.decode(latent).sample
reconstructed_image = ((reconstructed_image + 1) * 127.5).clamp(0, 255).to(torch.uint8)
print(reconstructed_image.shape)
reconstructed_image = Image.fromarray(reconstructed_image.permute(0, 2, 3, 1)[0].data.cpu().numpy())
```

평균과 표준편차를 각각 0.5로 맞춤

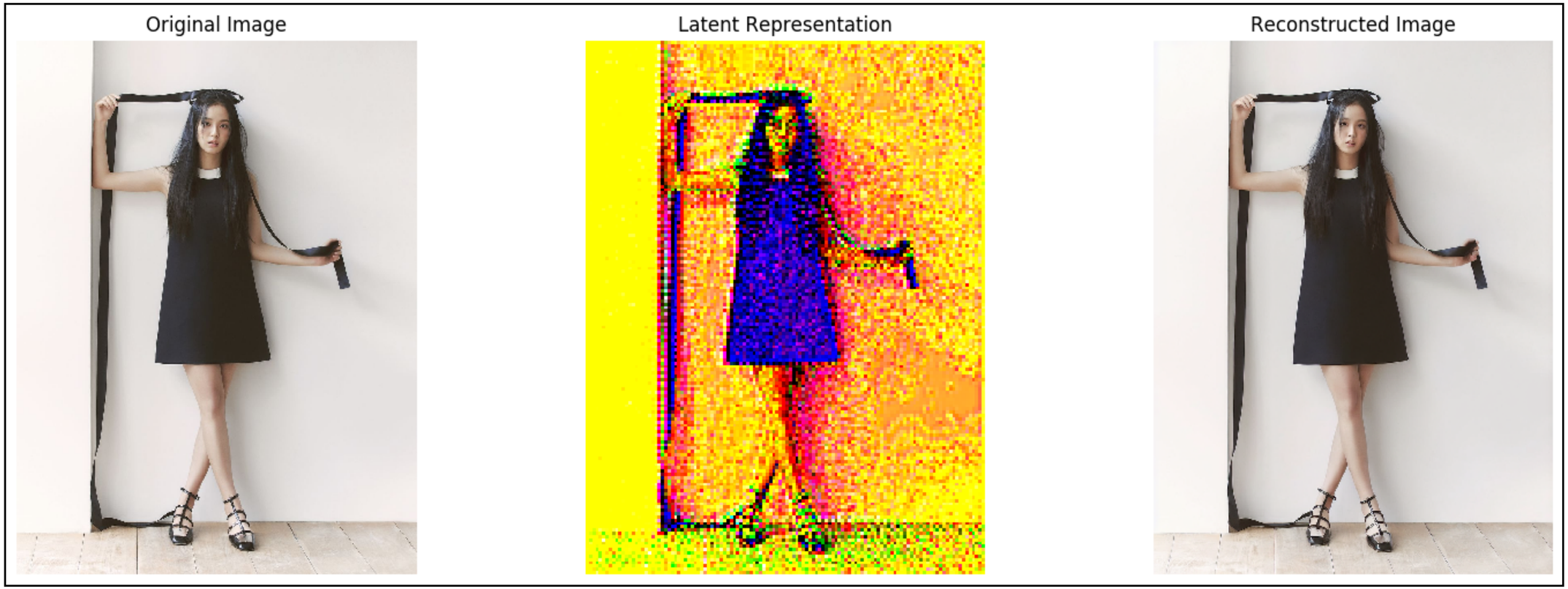
Encoding

Decoding

```
torch.Size([1, 3, 900, 900])
torch.Size([1, 4, 112, 112])
torch.Size([1, 3, 896, 896])
```

Stable Diffusion - Model Architecture - VAE 실습

- 실습 “diffusers-example/vae_small_face.ipynb”
- 얼굴 같이 디테일이 중요한 영역의 사이즈가 작으면 정보 손실로 인해 이미지 왜곡이 발생



3x1306x980 4x163x122 3x1304x976

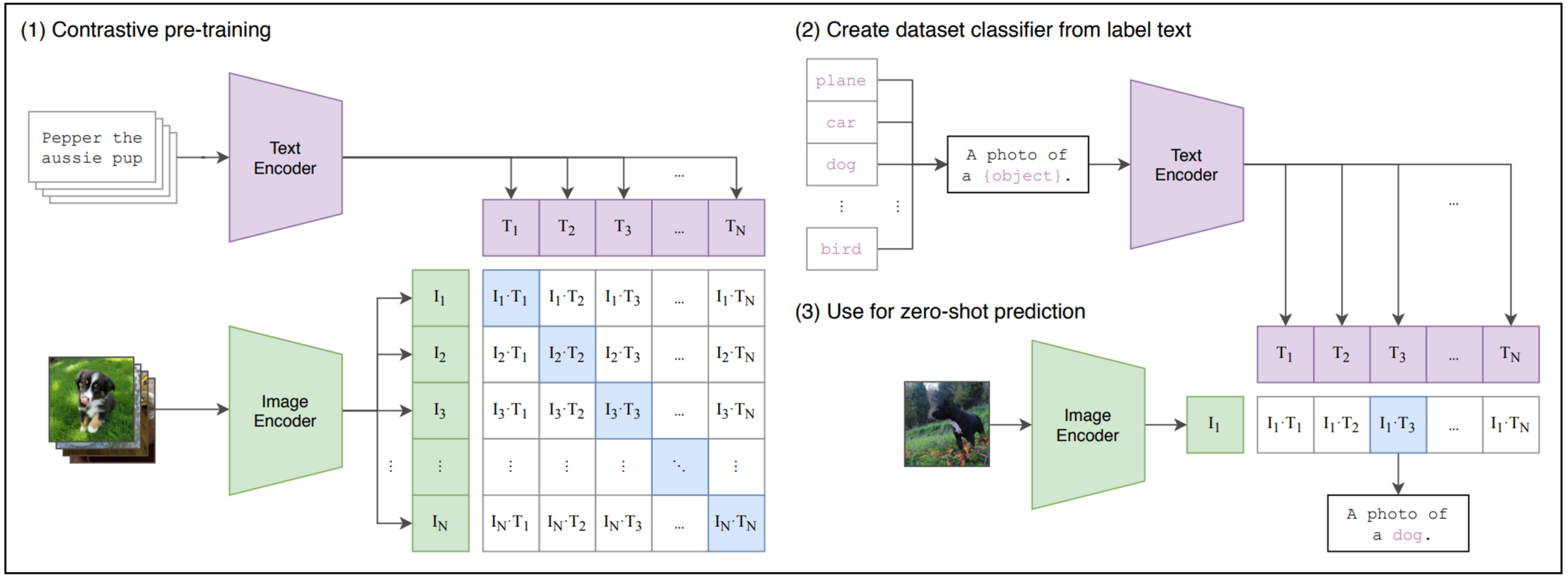
Stable Diffusion

Model Architecture

- Text Encoder

Stable Diffusion - Text Encoder 구조 - CLIP

- Text Encoder는 CLIP의 Text Encoder를 사용 (Pre-trained Model)
- Text/Image Pair Dataset을 가지고 Contrastive Learning을 통해 Text-Encoding과 Image-Encoding간의 Correlation을 높힘



Radford, Alec, et al. "Learning transferable visual models from natural language supervision." International conference on machine learning. PMLR, 2021.

Stable Diffusion - Text Encoder 구조 - CLIP Pseudo Code

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

1. Image Encoder (주로 Resent or Vision Transformer)와 Text Encoder(주로 Transformer)를 통해 Image와 Text를 받아 embedding vector를 출력함

2. 원하는 embedding차원으로 맞추기 위해 Weight를 곱해주고 normalize

3. Image Embedding과 Text Embedding을 dot-product하여 하나의 scalar값으로 만든다 (N개 instance를 가진 batch의 경우 NxN logits matrix 구성)

4. Image와 Text쌍이 맞는 경우를 정답으로하여 [0, 1, 2, …, N] logits에 text축과 image축으로 각각 cross entropy loss 적용

Radford, Alec, et al. "Learning transferable visual models from natural language supervision." International conference on machine learning. PMLR, 2021.

Stable Diffusion - Text Encoder 구조 - CLIP Embedder

```python
class FrozenCLIPEmbedder(AbstractEncoder):
    """Uses the CLIP transformer encoder for text (from Hugging Face)"""
    def __init__(self, version="openai/clip-vit-large-patch14", device="cuda", max_length=77):
        super().__init__()
        self.tokenizer = CLIPTokenizer.from_pretrained(version)
        self.transformer = CLIPTextModel.from_pretrained(version)
        self.device = device
        self.max_length = max_length
        self.freeze()
```

```python
    def forward(self, text):
        batch_encoding = self.tokenizer(text, truncation=True, max_length=self.max_length, return_length=True,
                                        return_overflowing_tokens=False, padding="max_length", return_tensors="pt")
        tokens = batch_encoding["input_ids"].to(self.device)
        outputs = self.transformer(input_ids=tokens)

        z = outputs.last_hidden_state
        return z
```

- 입력된 문장은 CLIPTokenizer에 의해서 여러개의 token들의 sequence로 변환됨 (integer sequence)
- Token sequence는 다시 CLIP Text Encoder(Transformer)에 입력되어 최종적으로 vector sequence를 반환

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/encoders/modules.py

Stable Diffusion - Text Encoder 구조 - Tokenzier & Text Encoder 정보

Tokenizer & TextEncoder

```
# pipeline에서 tokenizer와 text_encoder 가져오기
tokenizer = pipe.tokenizer
text_encoder = pipe.text_encoder
```

```
# tokenizer 정보 보기
tokenizer
```

```
CLIPTokenizer(name_or_path='openai/clip-vit-large-patch14', vocab_size=49408, model_max_length=77, is_fast=False, padding_side='right', truncation_side='right', special_tokens={'bos_token': '<|startoftext|>', 'eos_token': '<|endoftext|>', 'unk_token': '<|endoftext|>', 'pad_token': '<|endoftext|>'}, clean_up_tokenization_spaces=True),  added_tokens_decoder={
        49406: AddedToken("<|startoftext|>", rstrip=False, lstrip=False, single_word=False, normalized=True, special=True),
        49407: AddedToken("<|endoftext|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
}
```

```
# text encoder 정보 보기
text_encoder
```

```
CLIPTextModel(
  (text_model): CLIPTextTransformer(
    (embeddings): CLIPTextEmbeddings(
      (token_embedding): Embedding(49408, 768)
      (position_embedding): Embedding(77, 768)
    )
    (encoder): CLIPEncoder(
      (layers): ModuleList(
        (0-11): 12 x CLIPEncoderLayer(
          (self_attn): CLIPAttention(
            (k_proj): Linear(in_features=768, out_features=768, bias=True)
            (v_proj): Linear(in_features=768, out_features=768, bias=True)
            (q_proj): Linear(in_features=768, out_features=768, bias=True)
            (out_proj): Linear(in_features=768, out_features=768, bias=True)
          )
          (layer_norm1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (mlp): CLIPMLP(
            (activation_fn): QuickGELUActivation()
            (fc1): Linear(in_features=768, out_features=3072, bias=True)
            (fc2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (layer_norm2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        )
      )
    )
    (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
)
```

diffusers-example/stable diffusion step by step.ipynb

- 실습 “diffusers-example/stable diffusion step by ste.ipynb”
- Tokenizer와 Text Encoder에 대한 정보를 볼 수 있음
- Tokenizer는 49408개의 vocal size를 가지며, 77의 max length를 가짐
- Text Encoder는 Transformer로 이루어져 있으며 768dim의 모델 사이즈, 12개층의 layers를 가짐

Stable Diffusion - Text Encoder 구조 - Tokenzier 결과

```
# 프롬프트 설정
prompt = "(K-pop idol), 1girl, masterpiece, 8k, ultra high resolution, ultra high detail, ultra high quality, photo-

# 토크나이저로 프롬프트 인코딩
# https://github.com/huggingface/diffusers/blob/v0.29.1/src/diffusers/pipelines/stable_diffusion/pipeline_stable_dif
# line. 366-372
text_inputs = tokenizer(prompt,
                   padding="max_length",
                   max_length=tokenizer.model_max_length,
                   truncation=True,
                   return_tensors="pt").to(device)
text_inputs
```

```
{'input_ids': tensor([[49406,   263,   330,   268,  2852,  8884,  2361,   272,  1611,   267,
         12066,   267,   279,   330,   267,  8118,  1400,  9977,   267,  8118,
          1400,  7657,   267,  8118,  1400,  3027,   267,  1125,   268, 16157,
           267,  4353,   267, 18501,   267, 12609,  5799,   267,  1783,  7744,
         13899,  7929,   267, 16647,   267, 10350,   267,  8118, 16157,  3575,
           267,  1419,    75,  1735, 12609,  1215,  3095,   267,  1023,   579,
          8847,   267,  2045,  5915,   267, 12609,  5994,   267,  2782,   267,
         38147,   267, 13964,   267, 16652, 36205, 49407]], device='cuda:0'), 'attention_mask': tensor([[1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1]], device='cuda:0')}
```

```
# 각 토큰에 대응하는 단어 출력
tokens = text_inputs.input_ids[0]
tokens_str = [tokenizer.decode([token]) for token in tokens]

# 텍스트와 대응하는 토큰 출력
for i, (id, token) in enumerate(zip(text_inputs.input_ids[0], tokens_str)):
    print(f"Token {id}: {token}")
```

```
Token 49406: <|startoftext|>
Token 263: (
Token 330: k
Token 268: -
Token 2852: pop
Token 8884: idol
Token 2361: ),
Token 272: 1
Token 1611: girl
Token 267: ,
Token 12066: masterpiece
Token 267: ,
Token 279: 8
Token 330: k
```

diffusers-example/stable diffusion step by step.ipynb

- 입력된 prompt는 Tokenizer가 가지고 있는 vocabulary에 의해서 token으로 token으로 변환되고, 이 token에 대응하는 integer값이 최종적으로 출력됨
- <|startoftext|>라는 Start Token과 <|endoftext|>라는 End Token이 앞 뒤로 붙으며 이들을 포함하여 77개의 token sequence로 잘리므로 너무 긴 prompt는 유효하지 않음에 주의

Stable Diffusion - Text Encoder 구조 - Text Encoder 결과

- Tokenizer에 의해 출력된 integer sequence는 Transformer로 구성된 Text Encoder에 들어가 최종적으로 Text Embedding Sequence를 출력함
- Token Sequence Length = 77, Embedding Dimension = 768
- 이 정보가 이미지 생성 시, U-net의 cross attention의 Query, Key, Value 중 Key와 Value로 전달될 예정

```
# 텍스트 인코더로 인코딩된 입력을 통해 잠재 표현(embeddings) 추출
text_input_ids = text_inputs.input_ids.to(device)
attention_mask = text_inputs.attention_mask.to(device)

with torch.no_grad():
    prompt_embeds = text_encoder(text_input_ids,
                                 attention_mask=attention_mask,
                                 output_hidden_states=True
                )
    prompt_embeds = prompt_embeds[0]

print(prompt_embeds.shape)
prompt_embeds
```

```
torch.Size([1, 77, 768])

tensor([[[-0.3545,  0.0036, -0.0704,  ..., -0.4519, -0.2944,  0.0677],
         [ 0.4639, -1.4297,  0.2380,  ..., -1.2080,  1.9043, -0.1930],
         [ 0.4951, -1.0254, -0.3215,  ..., -0.8384, -0.1289,  0.0115],
         ...,
         [-0.0995, -0.9170,  1.3154,  ..., -0.1367, -0.7563, -0.7876],
         [ 0.4580, -0.6172,  1.3867,  ..., -0.1665, -1.1582, -0.7373],
         [ 0.2244, -0.8853,  1.0127,  ...,  0.1915, -1.5488, -0.1338]]],
       device='cuda:0', dtype=torch.float16)
```

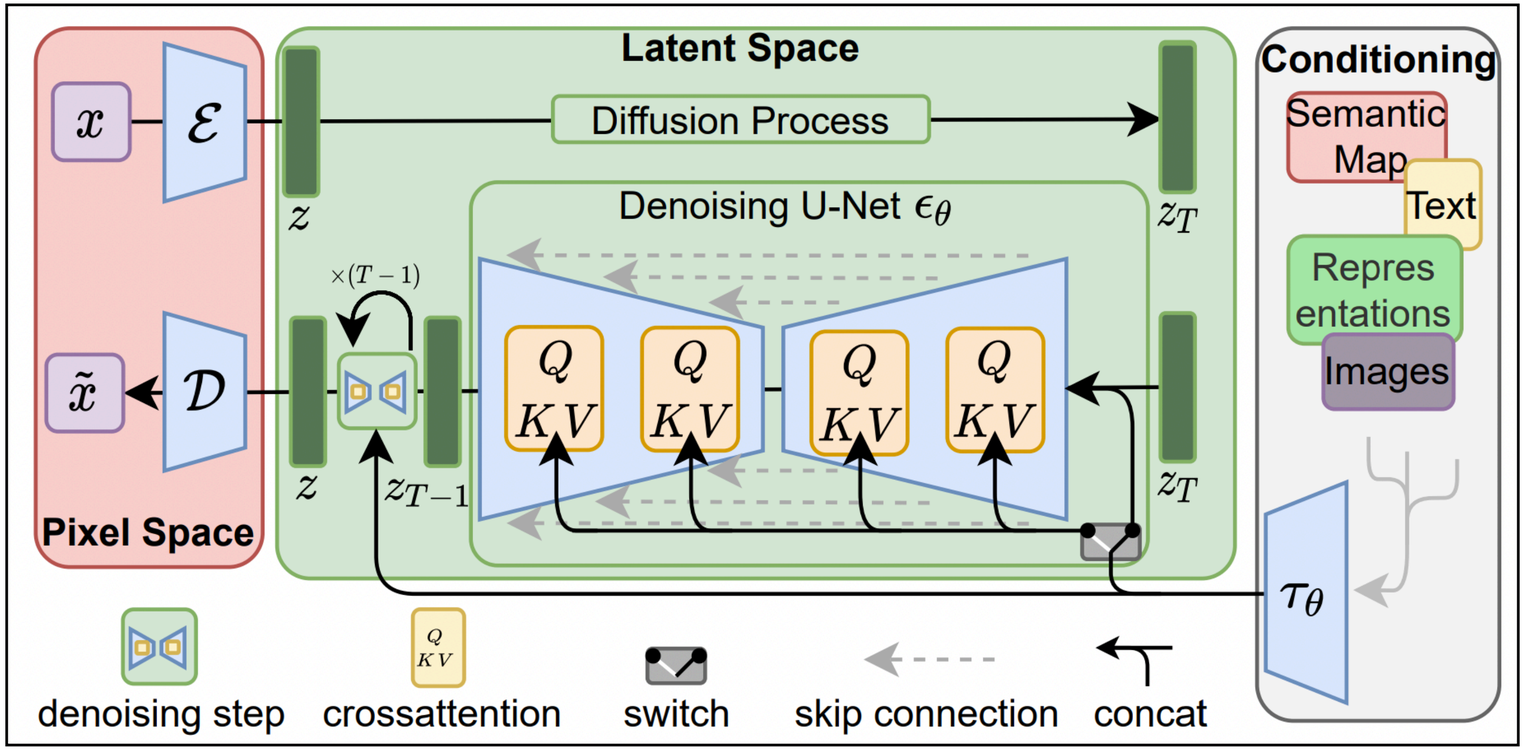
diffusers-example/stable diffusion step by step.ipynb

Stable Diffusion

Model Architecture

- Denoising U-net

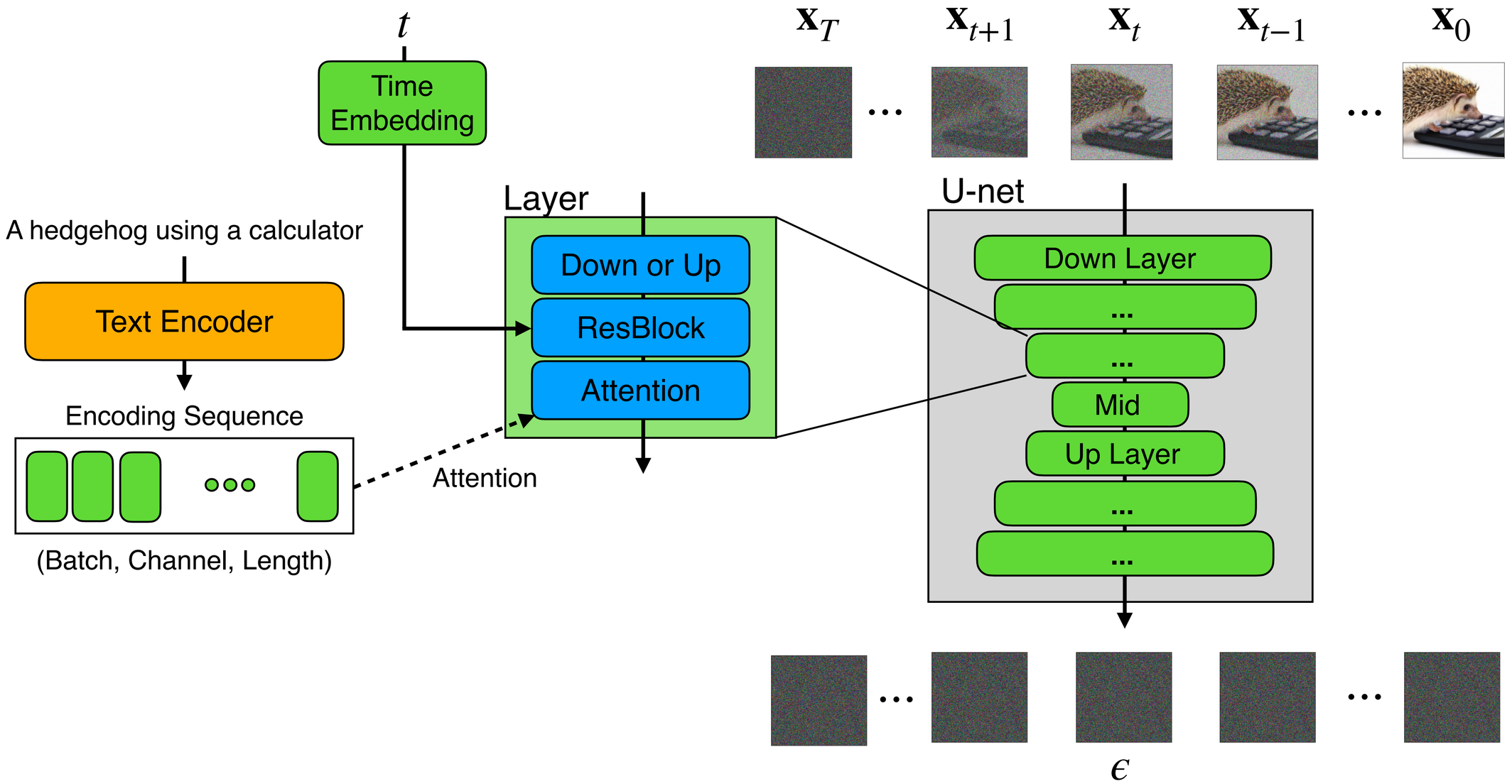
Stable Diffusion - Denoising U-net 구조



- Denoising U-net은 임의의 시점 t의 noisy latent $\mathbf{z}_t$를 입력받아 noise ϵ을 출력하는 역할을 함($\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ 공식을 통해 denoised image $\mathbf{z}_0$를 구할 수 있음)
- Down sampling을 수행하는 down layers와 up sampling을 수행하는 up layers를 기본으로 구성하고 있으며, 중간에 더욱 분석능력을 향상시키기 위해 mid layer를 추가하여 구성하고 있음
- 각 레이어들은 resnet block과 attention block을 이용하여 구성되어 있음
- Resnet block은 convolution을 이용하여 이미지의 공간적인 특성이 반영된 정보를 분석하고, attention block은 attention을 이용하여 원거리에 있는 연관된 정보들을 분석하는 역할을 함

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Denoising U-net 구조



https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Denoising U-net 구조 - Time Embedding

```
231                 # timestep embedding
232                 self.temb = nn.Module()
233                 self.temb.dense = nn.ModuleList([
234                     torch.nn.Linear(self.ch,
235                                     self.temb_ch),
236                     torch.nn.Linear(self.temb_ch,
237                                     self.temb_ch),
238                 ])
```

Time embedding network 선언

```
322                 # timestep embedding
323                 assert t is not None
324                 temb = get_timestep_embedding(t, self.ch)
325                 temb = self.temb.dense[0](temb)
326                 temb = nonlinearity(temb)
327                 temb = self.temb.dense[1](temb)
```

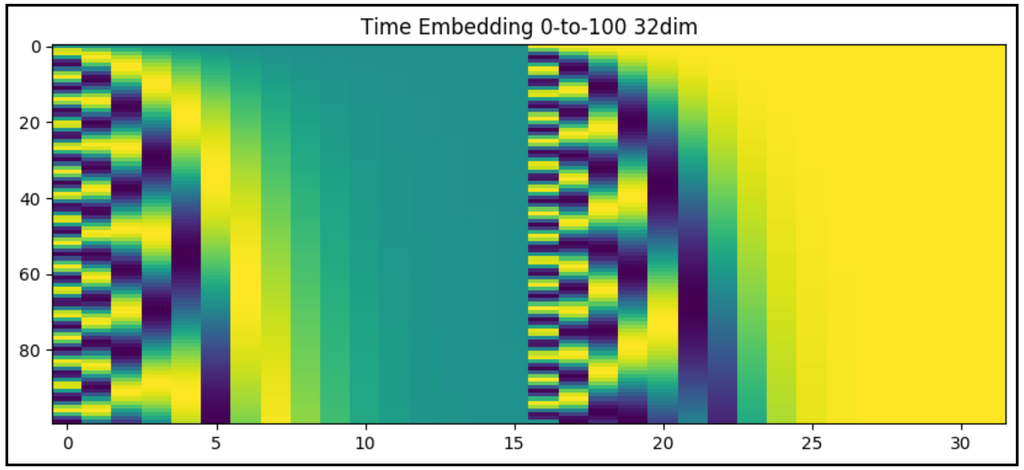
Time embedding function 호출 및 network 사용

- Time embedding은 time embedding function 부분, time embedding network 부분으로 나누어져 있음
- Time embedding function은 time index (0-999) integer를 입력 받아 float vector를 반환함
- Time embedding network는 float vector를 2개의 trainable linear network를 이용하여 non-linear transform을 하는 역할
- 이 두 mapping을 통해 time 정보가 U-net에 반영되도록 도움

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Denoising U-net 구조 - Time Embedding

```python
def get_timestep_embedding(timesteps, embedding_dim):
    """
    This matches the implementation in Denoising Diffusion Probabilistic Models:
    From Fairseq.
    Build sinusoidal embeddings.
    This matches the implementation in tensor2tensor, but differs slightly
    from the description in Section 3.5 of "Attention Is All You Need".
    """
    assert len(timesteps.shape) == 1

    half_dim = embedding_dim // 2
    emb = math.log(10000) / (half_dim - 1)
    emb = torch.exp(torch.arange(half_dim, dtype=torch.float32) * -emb)
    emb = emb.to(device=timesteps.device)
    emb = timesteps.float()[:, None] * emb[None, :]
    emb = torch.cat([torch.sin(emb), torch.cos(emb)], dim=1)
    if embedding_dim % 2 == 1:  # zero pad
        emb = torch.nn.functional.pad(emb, (0,1,0,0))
    return emb
```



diffusers-example/get_timestep_embedding.ipynb

- 실습 “diffusers-example/get_time_embedding.ipynb”
- Time embedding function은 embedding의 각 dimension별로 서로 다른 주기를 갖는 sin, cos 파형을 만들고, 주어진 timestep에 따라 값이 할당되도록 만듬

Stable Diffusion - Denoising U-net 구조 - Down Layer

```
240             # downsampling
241             self.conv_in = torch.nn.Conv2d(in_channels,
242                                            self.ch,
243                                            kernel_size=3,
244                                            stride=1,
245                                            padding=1)
246
247             curr_res = resolution
248             in_ch_mult = (1,)+tuple(ch_mult)
249             self.down = nn.ModuleList()
250             for i_level in range(self.num_resolutions):
251                 block = nn.ModuleList()
252                 attn = nn.ModuleList()
253                 block_in = ch*in_ch_mult[i_level]
254                 block_out = ch*ch_mult[i_level]
255                 for i_block in range(self.num_res_blocks):
256                     block.append(ResnetBlock(in_channels=block_in,
257                                              out_channels=block_out,
258                                              temb_channels=self.temb_ch,
259                                              dropout=dropout))
260                     block_in = block_out
261                     if curr_res in attn_resolutions:
262                         attn.append(make_attn(block_in, attn_type=attn_type))
263                 down = nn.Module()
264                 down.block = block
265                 down.attn = attn
266                 if i_level != self.num_resolutions-1:
267                     down.downsample = Downsample(block_in, resamp_with_conv)
268                     curr_res = curr_res // 2
269                 self.down.append(down)
```

- Down Layer는 down sampling을 위한 convolution, 여러 층의 ResnetBlock 그리고 Attention으로 구성됨
- Down sampling은 주어진 input image를 H, W에 대해 2배씩 줄이는 역할을 하며, ResnetBlock과 Attention으로 분석을 수행

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Denoising U-net 구조 - Middle Layer

```
271             # middle
272             self.mid = nn.Module()
273             self.mid.block_1 = ResnetBlock(in_channels=block_in,
274                                            out_channels=block_in,
275                                            temb_channels=self.temb_ch,
276                                            dropout=dropout)
277             self.mid.attn_1 = make_attn(block_in, attn_type=attn_type)
278             self.mid.block_2 = ResnetBlock(in_channels=block_in,
279                                            out_channels=block_in,
280                                            temb_channels=self.temb_ch,
281                                            dropout=dropout)
```

- Middle Layer는 down sampling이나 up sampling 없이 2개의 ResnetBlock과, Attention으로 구성됨

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Denoising U-net 구조 - Up Layer

```
283             # upsampling
284             self.up = nn.ModuleList()
285             for i_level in reversed(range(self.num_resolutions)):
286                 block = nn.ModuleList()
287                 attn = nn.ModuleList()
288                 block_out = ch*ch_mult[i_level]
289                 skip_in = ch*ch_mult[i_level]
290                 for i_block in range(self.num_res_blocks+1):
291                     if i_block == self.num_res_blocks:
292                         skip_in = ch*in_ch_mult[i_level]
293                     block.append(ResnetBlock(in_channels=block_in+skip_in,
294                                              out_channels=block_out,
295                                              temb_channels=self.temb_ch,
296                                              dropout=dropout))
297                     block_in = block_out
298                     if curr_res in attn_resolutions:
299                         attn.append(make_attn(block_in, attn_type=attn_type))
300                 up = nn.Module()
301                 up.block = block
302                 up.attn = attn
303                 if i_level != 0:
304                     up.upsample = Upsample(block_in, resamp_with_conv)
305                     curr_res = curr_res * 2
306                 self.up.insert(0, up) # prepend to get consistent order
307
308             # end
309             self.norm_out = Normalize(block_in)
310             self.conv_out = torch.nn.Conv2d(block_in,
311                                             out_ch,
312                                             kernel_size=3,
313                                             stride=1,
314                                             padding=1)
```

- Up Layer는 여러층의ResnetBlock, Attention 그리고 up sampling을 위한 convolution으로 이루어짐
- Down Layer와 대칭적으로 구성

https://github.com/CompVis/stable-diffusion/blob/main/ldm/modules/diffusionmodules/model.py

Stable Diffusion - Inference Step by Step 실습

- 실습 “diffusers-example/stable diffusion step-by-step.ipynb”
- Tokenizer, Text-Encoder, Unet, Scheduler, Sampling, VAE Decoder 작동 과정 살펴보기

```python
with torch.no_grad():
    images = vae.decode(latents / vae.config.scaling_factor, return_dict=False)[0]

# 이미지를 [0, 1] 범위로 스케일링
images = (images / 2 + 0.5).clamp(0, 1)

# 이미지를 CPU로 이동하고 numpy 배열로 변환
images = images.cpu().permute(0, 2, 3, 1).numpy()

# 이미지 출력
import matplotlib.pyplot as plt

plt.imshow(images[0].astype(np.float32))
plt.axis("off")
plt.show()
```

Guided Diffusion Sampling

- Diffusion Models Beat GANs on Image Synthesis 논문에서 제안

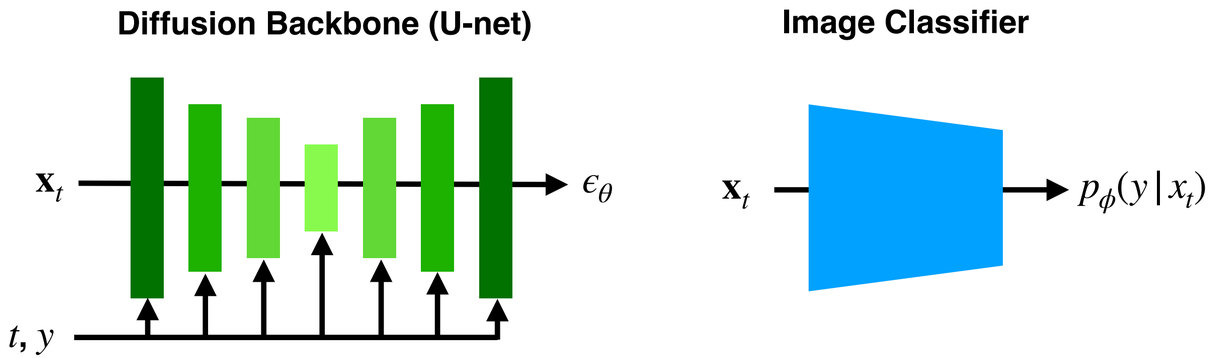
- Diffusion 모델 외 추가적인 image classifier를 학습시키고 sampling 과정에서 classifier로부터 gradient를 받아 sampling에 도움을 줌

Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

Nichol, Alex, et al.에서 재인용

$$\hat{\mu}_\theta\left(x_t \mid y\right) = \mu_\theta\left(x_t \mid y\right) + s \cdot \Sigma_\theta\left(x_t \mid y\right) \nabla_{x_t} \log p_\phi\left(y \mid x_t\right)$$

- $\hat{\mu}_\theta\left(x_t \mid y\right)$: 수정된 mean
- $\mu_\theta\left(x_t \mid y\right)$: 기존 mean
- s: guidance scale
- $\Sigma_\theta\left(x_t \mid y\right)$: 기존 covariance
- $\nabla_{x_t} \log p_\phi\left(y \mid x_t\right)$: classifier로부터 전해진 gradient



Dhariwal, Prafulla, and Alexander Nichol. "Diffusion models beat gans on image synthesis." *Advances in Neural Information Processing Systems* 34 (2021): 8780-8794.

Nichol, Alex, et al. "Glide: Towards photorealistic image generation and editing with text-guided diffusion models." arXiv preprint arXiv:2112.10741 (2021).

Guided Diffusion Sampling

Figure 13: Samples from our best 512×512 model (FID: 3.85). Classes are 1: goldfish, 279: arctic fox, 323: monarch butterfly, 386: african elephant, 130: flamingo, 852: tennis ball.

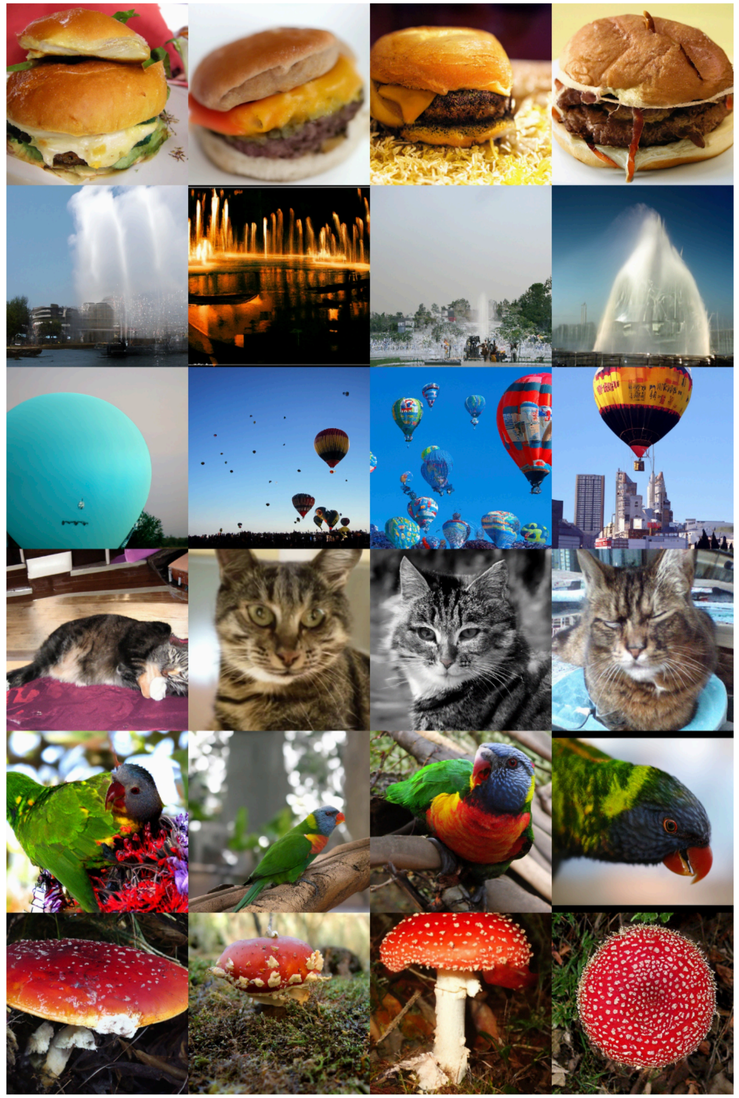


Figure 14: Samples from our best 512×512 model (FID: 3.85). Classes are 933: cheeseburger, 562: fountain, 417: balloon, 281: tabby cat, 90: lorikeet, 992: agaric.

Figure 15: Difficult class samples from our best 512×512 model (FID: 3.85). Classes are 432: bassoon, 468: cab, 424: barbershop, 444: bicycle-built-for-two, 981: ballplayer, 550: espresso maker.

Dhariwal, Prafulla, and Alexander Nichol. "Diffusion models beat gans on image synthesis." *Advances in Neural Information Processing Systems* 34 (2021): 8780-8794.

Classifier-free guidance

- Ho, Jonathan과 Tim Salimans의 논문 Classifier-free diffusion guidance에서 제안
- 추가적인 classifier를 트레이닝할 필요없이 diffusion 모델만 가지고 guided sampling을 가능하게 만듬
- Stable diffusion에서는 unconditional 대신 negative condition을 이용하여 guided sampling을 진행

Nichol, Alex, et al.에서 재인용

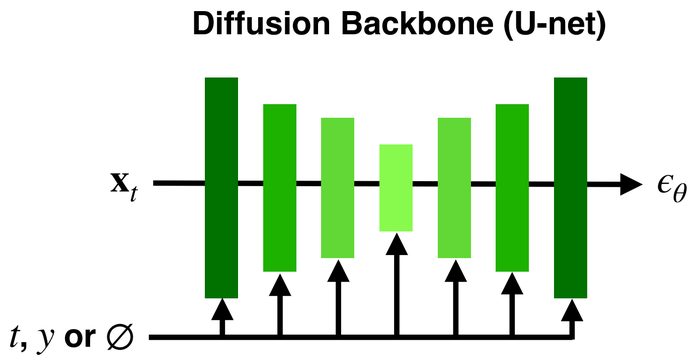
$$\hat{\epsilon}_\theta\left(x_t \mid y\right) = \epsilon_\theta\left(x_t \mid y\right) + s \cdot \left(\epsilon_\theta\left(x_t \mid y\right) - \epsilon_\theta\left(x_t \mid \varnothing\right)\right)$$

수정된 score

conditional predicted score

guidance scale

unconditional predicted score



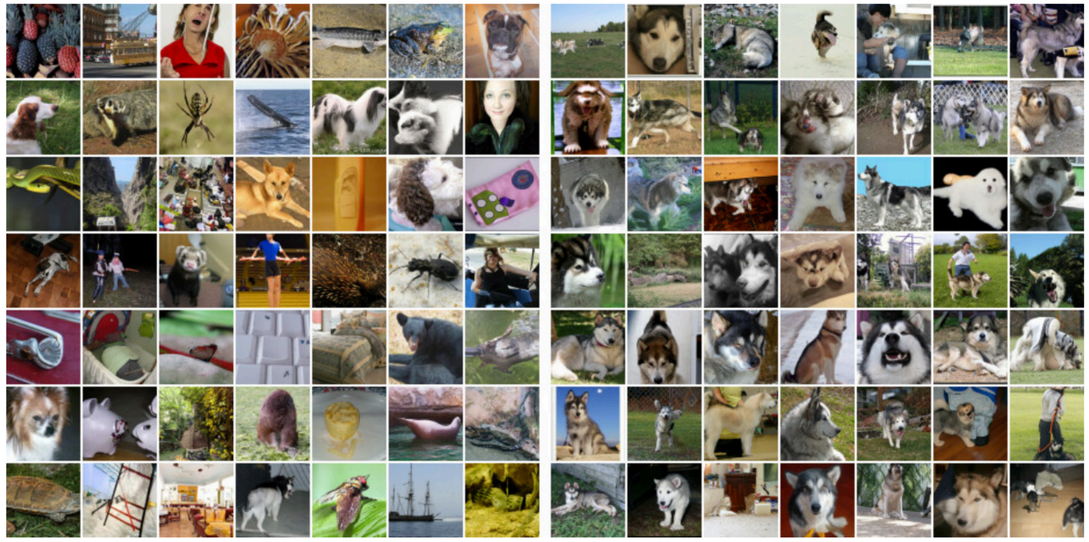

```python
from tqdm import tqdm

with torch.no_grad():
    # Sampling
    guidance_scale = 7.0
    for t in tqdm(scheduler.timesteps):
        # Latent와 텍스트 임베딩을 사용하여 노이즈 예측
        latent_model_input = torch.cat([latents] * 2)
        latent_model_input = scheduler.scale_model_input(latent_model_input, t)

        noise_pred = unet(latent_model_input,
                          t,
                          encoder_hidden_states=prompt_concat_embeds,
                          return_dict=False
                         )[0]

        # Classifier-Free Guidance Sampling
        noise_pred_uncond, noise_pred_text = noise_pred.chunk(2)
        noise_pred = noise_pred_uncond + guidance_scale * (noise_pred_text - noise_pred_uncond)

        # Scheduler를 사용하여 latents 업데이트
        latents = scheduler.step(noise_pred, t, latents, return_dict=False)[0]
```

diffusers-example/stable diffusion step-by-step.ipynb

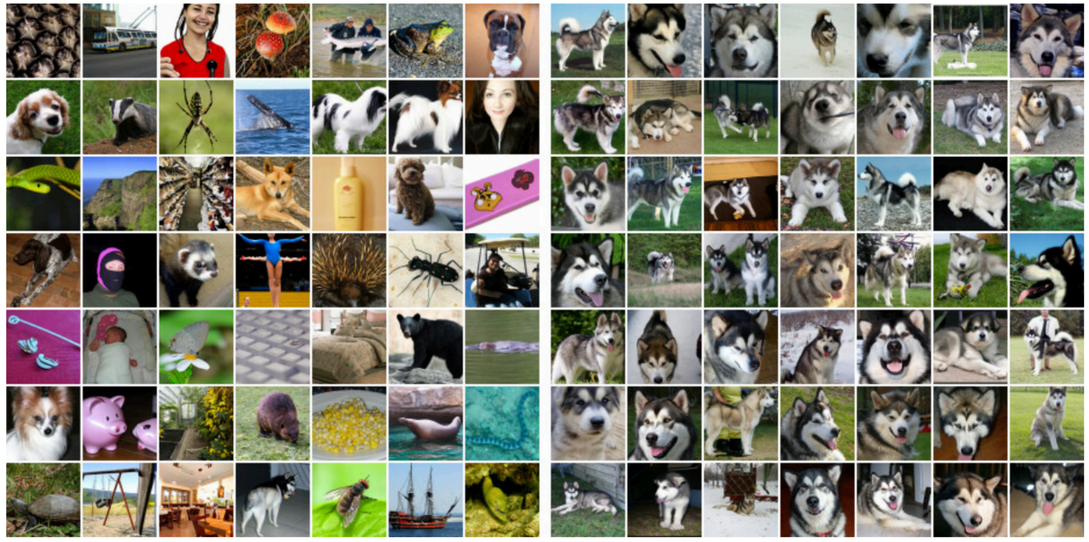
Ho, Jonathan, and Tim Salimans. "Classifier-free diffusion guidance." NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications. 2021.

Nichol, Alex, et al. "Glide: Towards photorealistic image generation and editing with text-guided diffusion models." arXiv preprint arXiv:2112.10741 (2021).

Classifier-free guidance



(a) Non-guided conditional sampling: FID=1.80, IS=53.71



(b) Classifier-free guidance with $w = 1.0$: FID=12.6, IS=170.1

Ho, Jonathan, and Tim Salimans. "Classifier-free diffusion guidance." NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications. 2021.

Fine-Tuning

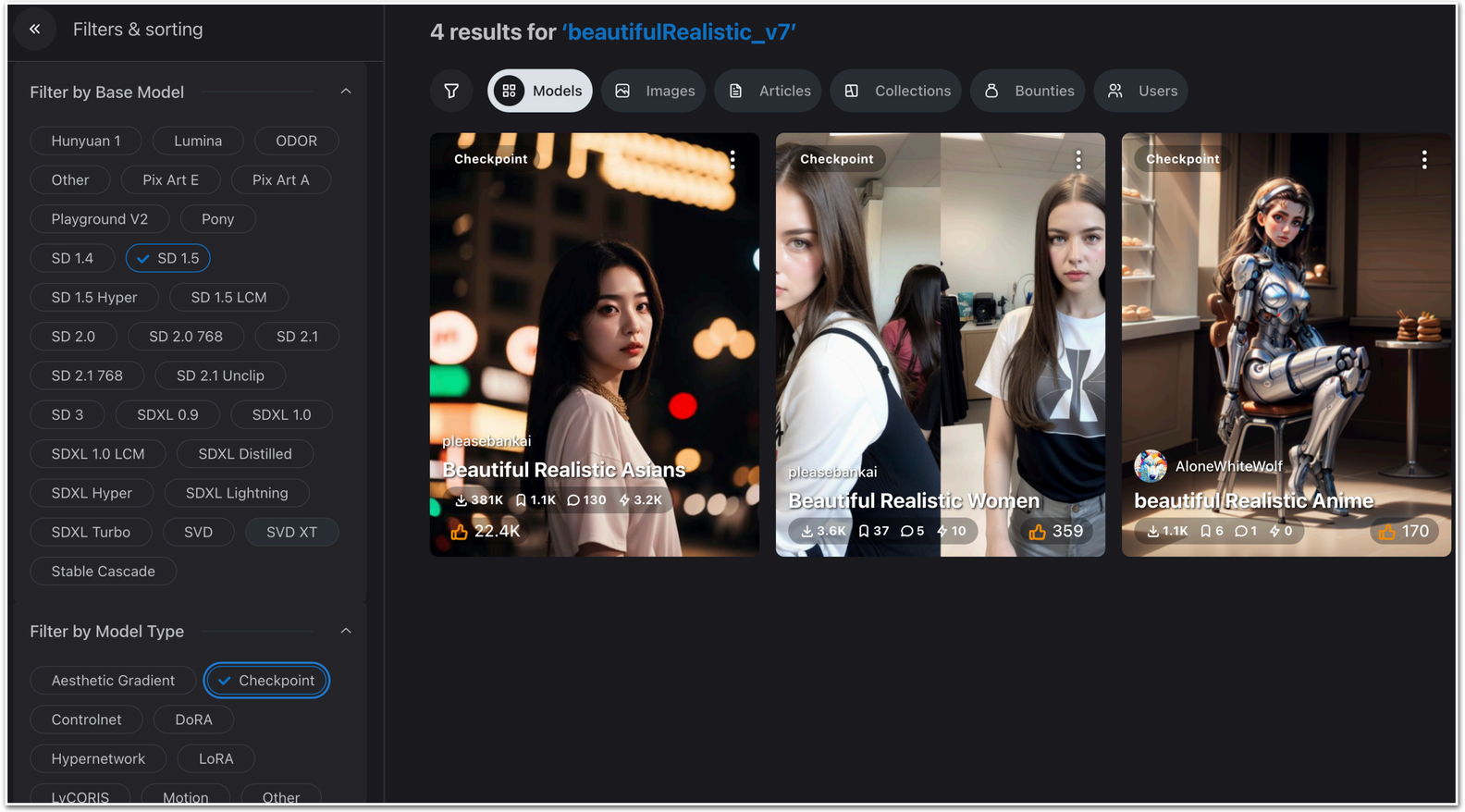
Fine-Tuning

- Stability.ai나 civitai에서 제공하는 checkpoint에서부터 자신이 가지고 있는 데이터들로 새로 트레이닝
- Weight 전체를 다시 학습하는 것인 만큼 가장 데이터를 표현하는데 좋은 학습 방법이나 상당히 많은 양의 데이터가 필요함
- Fine-tuning을 할 수 있는 트레이닝 소스코드로 diffusers [1], kohya [2] 등이 있음
  webUI [4], civitai [5] 등에 호환되는 방식은 kohya임 (diffusers는 metadata 등에서 차이가 나서 로드가 안될 때가 있음)
- Kohya의 변종으로 윈도우에서 GUI로 트레이닝할 수 있게하는 Kohya's GUI [3]이나, Google colab을 통해서 트레이닝할 수 있게 해주는 Kohya Colabs [6] 등이 개발이 가능한 팬층에서 github을 통해 오픈소스로 공개되고 있음

[1] https://github.com/huggingface/diffusers/tree/main/examples
[2] https://github.com/kohya-ss/sd-scripts
[3] https://github.com/bmaltais/kohya_ss
[4] https://github.com/AUTOMATIC1111/stable-diffusion-webui
[5] https://civitai.com
[6] https://github.com/hollowstrawberry/kohya-colab

Fine-Tuning - 실습 - 1. Base Checkpoint Download

- Civitai에서 원하는 checkpoint를 다운로드 받습니다. 최대한 우리의 dataset과 성격이 비슷한 것을 고릅니다. (SD 1.5, Checkpoint로 검색)



[1] https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

Fine-Tuning - 실습 - 2. 현재 checkpoint의 결과 확인

- “sd-scripts-example/fine tuning/diffusers run.ipynb”를 실행하여 현재 checkpoint로 얻어지는 결과를 확인해봅니다.

```
prompt = "celeba, blond_hair, pale_skin, wearing_necklace, wearing_hat, young, male"

image = pipe(prompt=prompt,
         height=512,
         width=512,
         num_inference_steps=20,
         guidance_scale=7,
        ).images[0]
image
```

Error rendering Jupyter widget: missing widget manager

Fine-Tuning - 실습 - 3. CelebA dataset download

- https://drive.google.com/file/d/1m8-EBPgi5MRubrm6iQjafK2QMHDBMSfJ/view?usp=sharing
  에서 celeba.zip를 다운로드 받고 /data에 압축을 풉니다
  unzip celeba.zip -d /data

```
(base) ➜  celeba pwd
/data/celeba
(base) ➜  celeba ls -l
total 1471752
-rw-rw-r-- 1 gaudio gpuadmin    3424458 Aug 21  2017 identity_CelebA.txt
-rw-r--r-- 1 gaudio gpuadmin 1443490838 Jun 28  2021 img_align_celeba.zip
-rw-rw-r-- 1 gaudio gpuadmin   26721026 Oct 18  2016 list_attr_celeba.txt
-rw-rw-r-- 1 gaudio gpuadmin    6082035 May 29  2017 list_bbox_celeba.txt
-rw-r--r-- 1 gaudio gpuadmin    2836386 Jun 28  2021 list_eval_partition.txt
-rw-rw-r-- 1 gaudio gpuadmin   12156055 May  3  2016 list_landmarks_align_celeba.txt
-rw-rw-r-- 1 gaudio gpuadmin   12352635 Sep 28  2015 list_landmarks_celeba.txt
drwxr-xr-x 3 gaudio gpuadmin         28 Jul  5 14:36 __MACOSX
```

- img_align_celeba.zip 압축도 풀어줍니다.

```
(base) ➜  celeba ls
identity_CelebA.txt   list_attr_celeba.txt     list_landmarks_align_celeba.txt
img_align_celeba      list_bbox_celeba.txt     list_landmarks_celeba.txt
img_align_celeba.zip  list_eval_partition.txt  __MACOSX
```

Fine-Tuning - 실습 - 4. Image Pre-processing

- “Sd-scripts-examples/fine tuning/make celeba dataset.ipynb”을 실행합니다.

Celeba Pytorch Dataset

```
root = '/data'

train_transforms = transforms.Compose([transforms.CenterCrop(148),
                                       transforms.Resize(512),
                                       transforms.ToTensor(),
                                       transforms.Normalize(0.5, 0.5)])

dataset = CelebA(root, split='train', transform=train_transforms, download=False)
dataset
```

```
attributes = [
    "5_o_Clock_Shadow", "Arched_Eyebrows", "Attractive", "Bags_Under_Eyes",
    "Bald", "Bangs", "Big_Lips", "Big_Nose", "Black_Hair", "Blond_Hair",
    "Blurry", "Brown_Hair", "Bushy_Eyebrows", "Chubby", "Double_Chin",
    "Eyeglasses", "Goatee", "Gray_Hair", "Heavy_Makeup", "High_Cheekbones",
    "Male", "Mouth_Slightly_Open", "Mustache", "Narrow_Eyes", "No_Beard",
    "Oval_Face", "Pale_Skin", "Pointy_Nose", "Receding_Hairline", "Rosy_Cheeks",
    "Sideburns", "Smiling", "Straight_Hair", "Wavy_Hair", "Wearing_Earrings",
    "Wearing_Hat", "Wearing_Lipstick", "Wearing_Necklace", "Wearing_Necktie",
    "Young"
]
```

```
def get_text(y, attributes):
    text = ""
    for i in range(len(attributes)):
        if y[i] == 1:
            text += attributes[i].lower()
            if i < len(attributes)-1:
                text += ', '
    return text
```

```
get_text(y, attributes)
```

```
'5_o_clock_shadow, attractive, bags_under_eyes, big_lips, big_nose, black_hair, bushy_eyebrows, male, no_beard, poi
nty_nose, straight_hair, young'
```

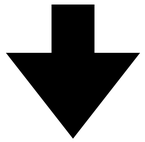
- 이미지를 불러와 crop과 resize를 수행하고,

- dataset에 있는 attributes를 이용해서 트레이닝에 사용될 text를 만들어줍니다.

Fine-Tuning - 실습 - 5. Caption .json 파일 만들기

- sd-scripts 디렉토리 상에서 다음과 같이 명령
  > mkdir toml
  > python -m finetune.merge_captions_to_metadata \
  /data/sd_dataset/celeba \
  toml/celeba.json \
  --caption_extension .txt \
  --full_path \
  --recursive

```
(ste) ➜  sd-scripts git:(main) ✗ cat /data/sd_dataset/celeba/0.txt
celeba, arched_eyebrows, attractive, brown_hair, heavy_makeup, high_cheekbones, mouth_slightly_open, no_beard, pointy_nose, smili
ng, straight_hair, wearing_earrings, wearing_lipstick, young
```



.txt에 있던 text들이 .json 파일에 담김

```
{
  "/data/sd_dataset/celeba/0.jpg": {
    "caption": "celeba, arched_eyebrows, attractive, brown_hair, heavy_makeup, high_cheekbones, mouth_slightly_open, no_beard, pointy_nose, smiling, straight_hair, wearing_earrings, wearing_lipstick, young"
  },
  "/data/sd_dataset/celeba/1.jpg": {
    "caption": "celeba, bags_under_eyes, big_nose, brown_hair, high_cheekbones, mouth_slightly_open, no_beard, smiling, young"
  },
  "/data/sd_dataset/celeba/10.jpg": {
    "caption": "celeba, attractive, black_hair, mouth_slightly_open, no_beard, smiling, young"
  },
  "/data/sd_dataset/celeba/100.jpg": {
    "caption": "celeba, arched_eyebrows, attractive, black_hair, high_cheekbones, no_beard, wearing_lipstick, young"
  },
```

toml/celeba.json

Fine-Tuning - 실습 - 6. Dataset .toml 파일 작성

- sd-scripts/toml 디렉토리에 celeba.toml 파일 만들고 dataset 정보 입력
  “sd-scripts-example/fine tuning/celeba.toml” 참고

```toml
[general]
shuffle_caption = true                          # 캡션을 셔플합니다.

[[datasets]]
resolution = 512                                # 학습 해상도
batch_size = 4                                  # 배치 크기

  [[datasets.subsets]]
  image_dir = '/data/sd-dataset/celeba'         # 학습용 이미지 폴더의 경로
  metadata_file = 'toml/celeba.json'  # 메타데이터 파일의 경로
```

“sd-scripts-example/fine tuning/celeba.toml”

Fine-Tuning - 실습 - 7. Training 시작

- sd-scripts 디렉토리에서 다음과 같이 입력 (1000 steps 15분 정도 소요)

```
accelerate launch --num_cpu_threads_per_process 1 fine_tune.py \
    --pretrained_model_name_or_path="/data/sd_files/checkpoint/beautifulRealistic_v7.safetensors" \
    --dataset_config=toml/celeba.toml \
    —output_dir=/data/sd_results/celeba \
    --output_name=celeba \
    --save_model_as=safetensors  \
    --max_train_steps=100000 \
    --save_every_n_steps=1000 \
    --learning_rate=1e-5  \
    --optimizer_type="AdamW"  \
    --mixed_precision="fp16"  \
    --gradient_checkpointing
```

```
running training / 学習開始
  num examples / サンプル数: 50000
  num batches per epoch / 1epochのバッチ数: 12500
  num epochs / epoch数: 8
  batch size per device / バッチサイズ: 1
  total train batch size (with parallel & distributed & accumulation) / 総バッチサイズ（並列学習、勾配合
計含む）: 1
  gradient accumulation steps / 勾配を合計するステップ数 = 1
  total optimization steps / 学習ステップ数: 100000
steps:   0%|                                                                          | 0/100000 [00:00<?, ?it/s]
epoch 1/8
```

트레이닝 화면

Fine-Tuning - 실습 - 8. 트레이닝 결과 확인

- “sd-scripts-example/fine tuning/diffusers run.ipynb”를 실행하여 트레이닝된 결과를 확인합니다.

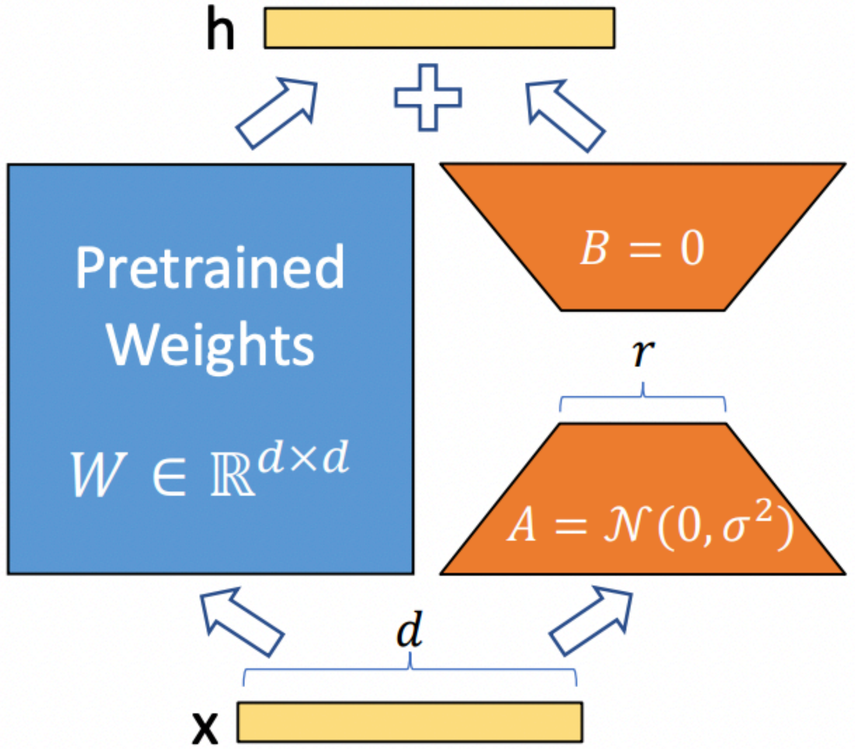
```
prompt = "celeba, blond_hair, pale_skin, wearing_necklace, wearing_hat, young, male"

image = pipe(prompt=prompt,
         height=512,
         width=512,
         num_inference_steps=20,
         guidance_scale=7,
        ).images[0]
image
```

```
100%  20/20 [00:00<00:00, 21.32it/s]
```

LoRA

LoRA - 이론

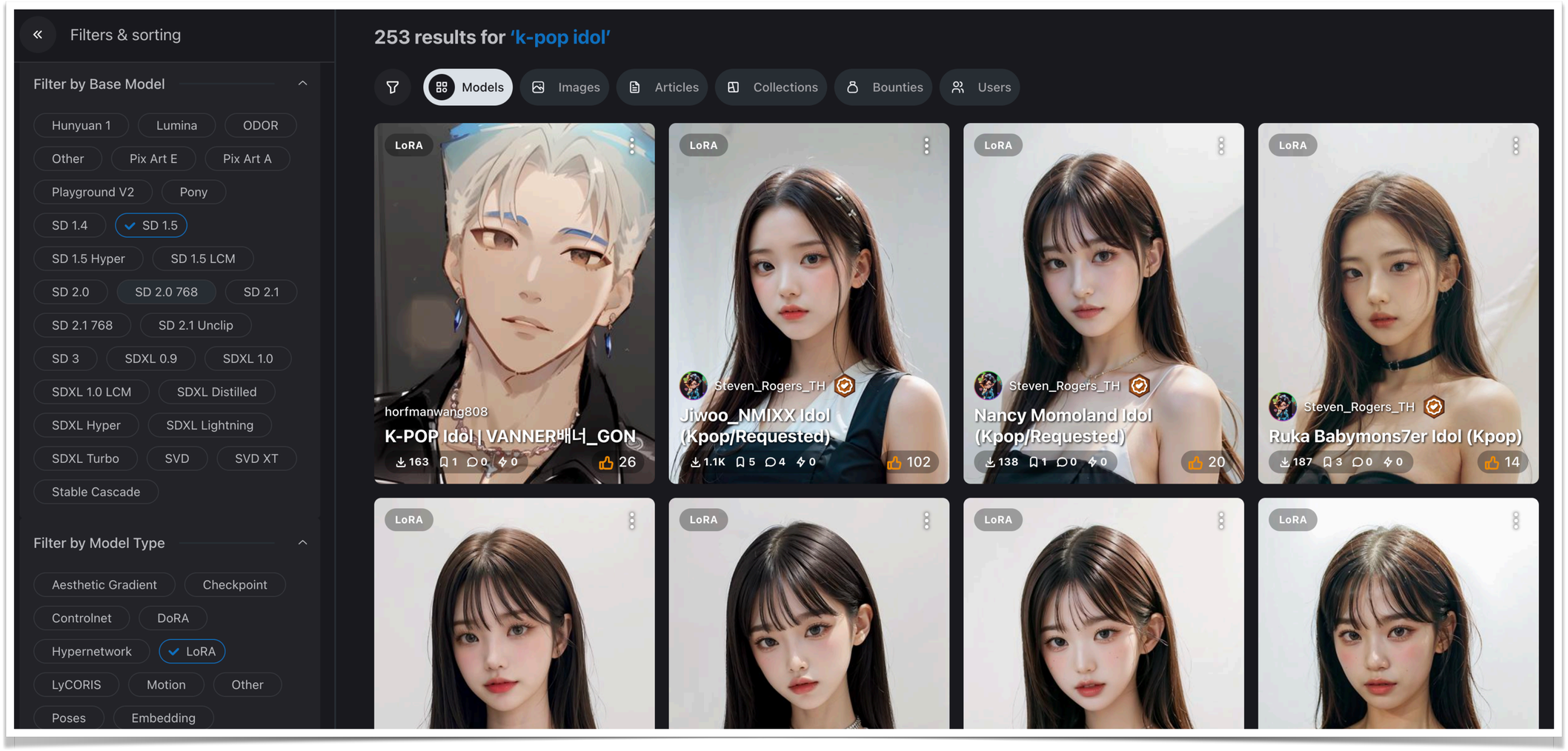
- Lora는 원래 GPT-3와 같은 LLM 모델의 fine-tuning을 위해 제안된 방법임
- 기존에 트레이닝된 weight $W \in \mathbb{R}^{d \times d}$를 그대로 유지하면서 추가적인 layer $A \in \mathbb{R}^{d \times r}$와 $B \in \mathbb{R}^{r \times d}$를 두어 weight에 대한 증분 ΔW으로 설정
  $h = W_0 x + \Delta W x = W_0 x + BAx$
- r은 d보다 훨씬 작게 설정하여 트레이닝 되어야할 파라메터 수를 줄임
- 이를 위해 B는 zero matrix로 초기화하여 초기에 계산의 결과가 W만 사용한 것과 같도록 함
- Matrix의 linearity를 이용하여 트레이닝되고 난 뒤에는 하나의 matrix로 합하여 연산 시간 손실 없이 inference 가능
  $W_0 x + BAx = (W_0 + BA)x$



Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." arXiv preprint arXiv:2106.09685 (2021).

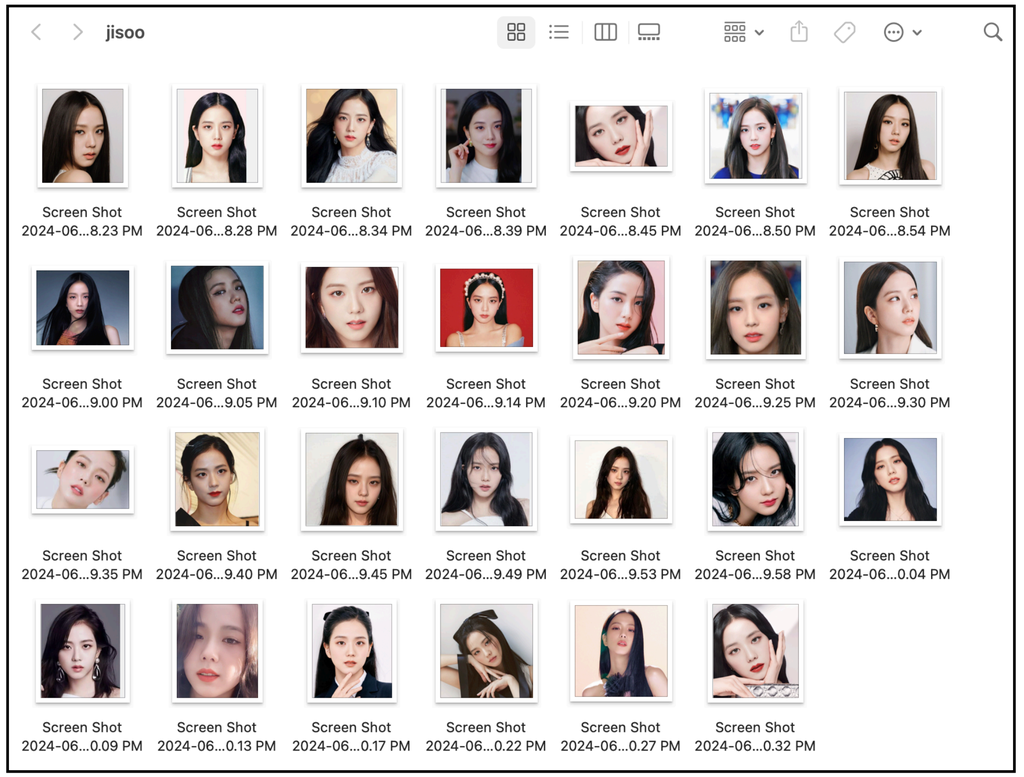
LoRA - Civitai

- Civitai와 같은 커뮤니티에서 Model Type LoRA를 선택하여 사용자들이 트레이닝하여 올린 다양한 LoRA 모델을 받아볼 수 있음



LoRA - 실습 - 1. Data 수집

- 인터넷에서 크롤링을 하거나 직접 가져오는 방식으로 원하는 종류나 사람의 이미지를 수집
- 얼굴이나 포즈 등이 같을수록 적은 데이터로도 트레이닝 가능하고, 다양한 얼굴, 포즈의 경우 데이터가 많아야함
- 수십장 정도로도 트레이닝 가능하나 쓸만한 품질을 낳기 위해서는 수백-수천장 필요
- 얼굴 사진을 원할 경우 이미지에 찰 정도로 크게 자르는 작업이 필요 (stable diffusion은 작은 얼굴이 깨짐, VAE의 영향)



LoRA - 2. Image to PNG

- Sd-scripts-example/lora/raw images to png.ipynb 실행
  Sd-scripts에서 기본적인 이미지 파일들 .png, jpg 등을 지원하지만 일관성을 위해 .png 파일로 모두 변환

```python
import os
from PIL import Image

def convert_images_to_png(source_dir, target_dir):
    # Ensure target directory exists
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)

    # Loop through all files in the source directory
    for i, filename in enumerate(os.listdir(source_dir)):
        # Check if the file is an image
        if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.tiff', '.wel
            # Open the image
            img = Image.open(os.path.join(source_dir, filename))
            # Get the file name without the extension
            file_name_without_ext = os.path.splitext(filename)[0]
            # Save the image as PNG in the target directory
            img.save(os.path.join(target_dir, f'{i}.png'))
            print(i, f'Converted {filename} to PNG format.')
        else:
            print(filename, 'skip')

source_directory = '/data/sd_dataset/jisoo_raw'  # 원본 디렉토리 경로
target_directory = '/data/sd_dataset/jisoo_png'  # 변환된 파일을 저장할 디렉토리 경로

convert_images_to_png(source_directory, target_directory)
```

```
0 Converted Screen Shot 2024-06-11 at 7.39.58 PM.png to PNG format.
1 Converted Screen Shot 2024-06-11 at 7.40.04 PM.png to PNG format.
2 Converted Screen Shot 2024-06-11 at 7.39.53 PM.png to PNG format.
3 Converted Screen Shot 2024-06-11 at 7.40.13 PM.png to PNG format.
4 Converted Screen Shot 2024-06-11 at 7.40.09 PM.png to PNG format.
```

Sd-scripts-example/lora/raw images to png.ipynb

LoRA - 3. Image to Text

- “sd-scripts-example/lora/image to text.ipynb” 실행하여 text caption을 만들고 저장 (모델 다운로드 하는데 시간 오래 걸림)
- BLIP2 모델을 이용하며 필요에 따라 다른 다양한 모델을 이용 가능

Image to Text

```
def image_to_text(image_file, text=None):
    raw_image = Image.open(image_file).convert('RGB')

    inputs = processor2(images=raw_image, text=text, return_tensors="pt").to('cuda:0', torch

    generated_ids = model2.generate(**inputs, max_length=50)
    generated_text = processor2.batch_decode(generated_ids, skip_special_tokens=True)[0].str
    return generated_text
```

```
from tqdm import tqdm
import os

prefix = 'jisoo'
for image_file in tqdm(png_files):
    save_file = image_file[:-3] + 'txt'
    prompt = prefix + ', ' + image_to_text(image_file)
    with open(save_file, 'w') as f:
        f.write(prompt)
```

sd-scripts-example/lora/image to text.ipynb

LoRA - 4. Dataset .toml 파일 작성

- sd-scripts/toml 디렉토리에 .toml 파일 작성
- 준비한 데이터 디렉토리를 datasets.subsets에 기록

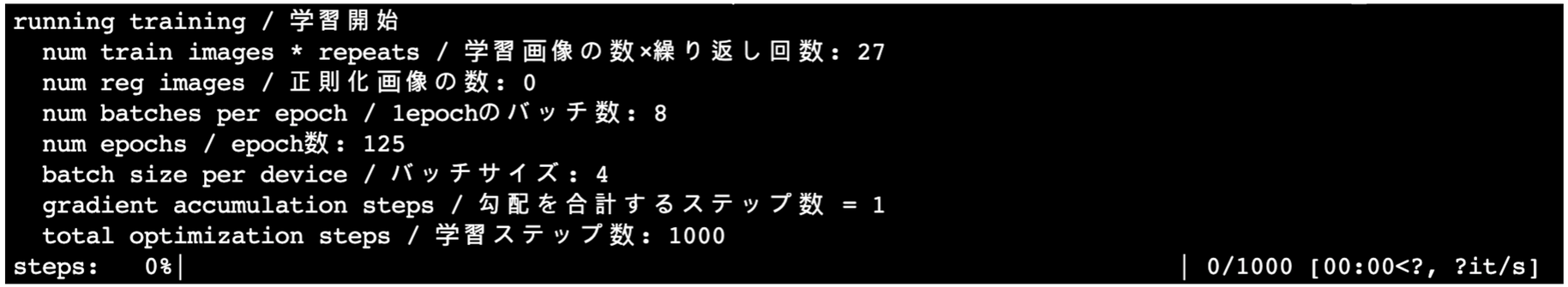
```toml
[general]
enable_bucket = true
caption_extension = '.txt'

[[datasets]]
resolution = 512
batch_size = 4

  [[datasets.subsets]]
  image_dir = '/data/sd-dataset/jisoo_png'
  num_repeats = 1
```

sd-scripts-example/lora/jisoo_lora.toml

LoRA - 5. Training 시작

- sd-scripts 디렉토리에서 다음과 같이 입력 (10분 정도 소요)

```
accelerate launch --num_cpu_threads_per_process 1 train_network.py \
    --pretrained_model_name_or_path="/data/sd_files/checkpoint/beautifulRealistic_v7.safetensors" \
    --dataset_config=toml/jisoo_lora.toml \
    --output_dir=/data/sd_results/jisoo_lora \
    --output_name=jisoo_lora \
    --save_model_as=safetensors  \
    --max_train_steps=1000 \
    --learning_rate=1e-4  \
    --optimizer_type="AdamW"  \
    --mixed_precision="fp16"  \
    --gradient_checkpointing \
    --network_module=networks.lora
```

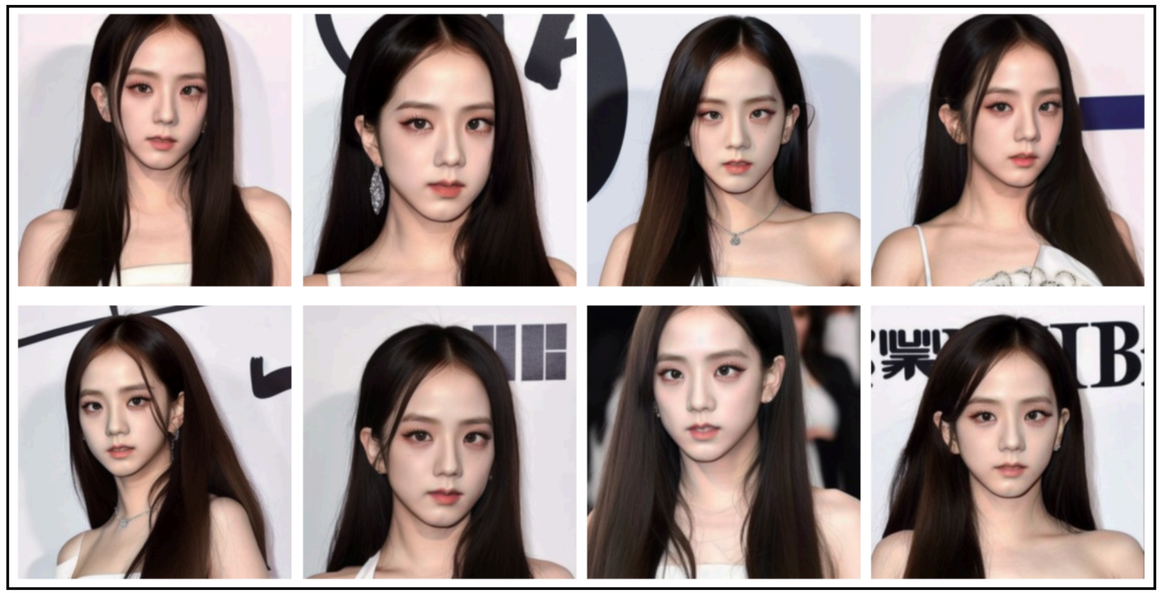


트레이닝 화면

LoRA - 6. Training 결과 확인

- “sd-scripts-example/lora/diffusers run.ipynb”
  “sd-scripts-example/lora/diffusers lora run.ipynb” 실행하여 결과 비교

diffusers run.ipynb

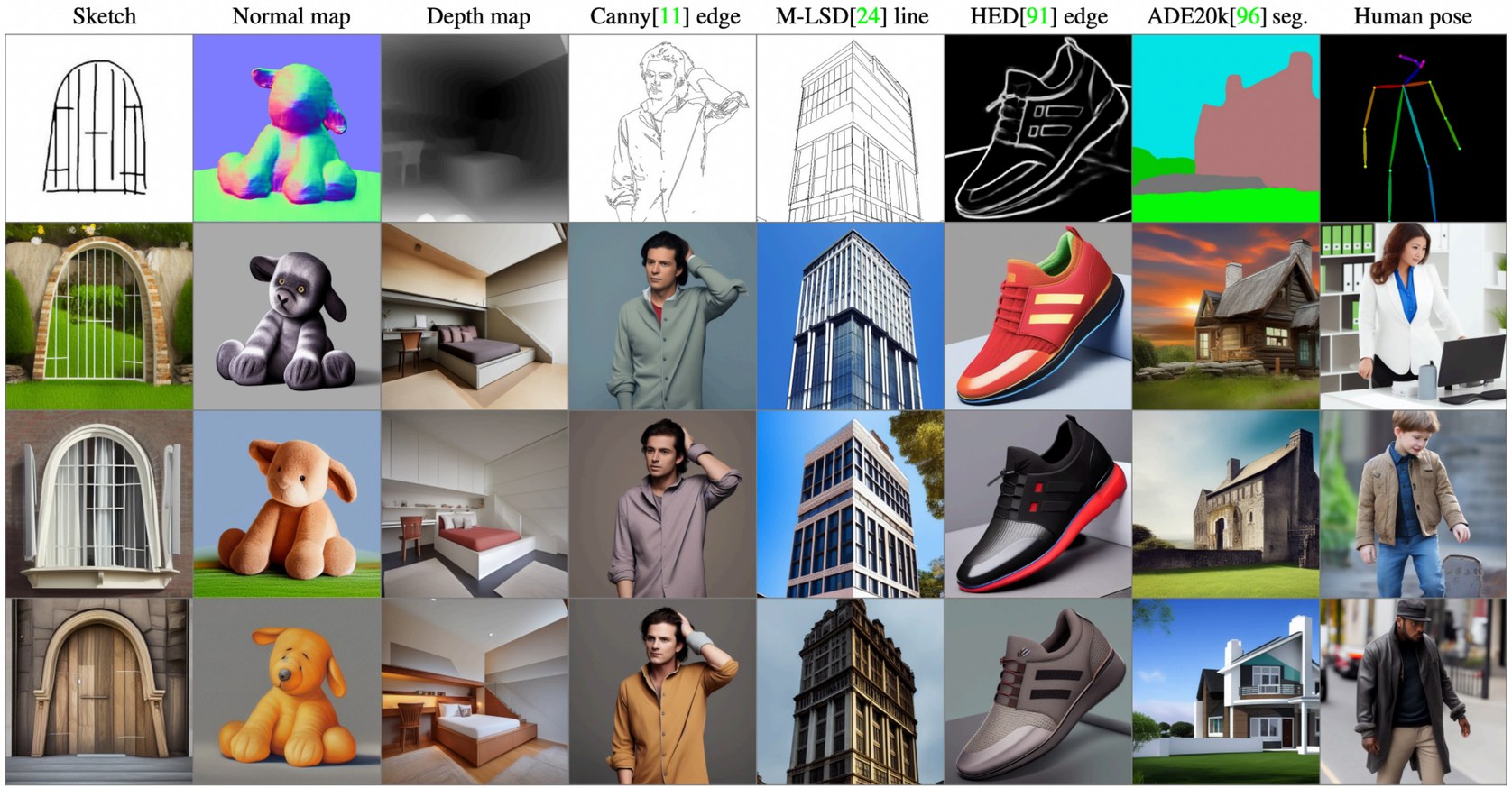


diffusers lora run.ipynb

ControlNet

ControlNet - 개요

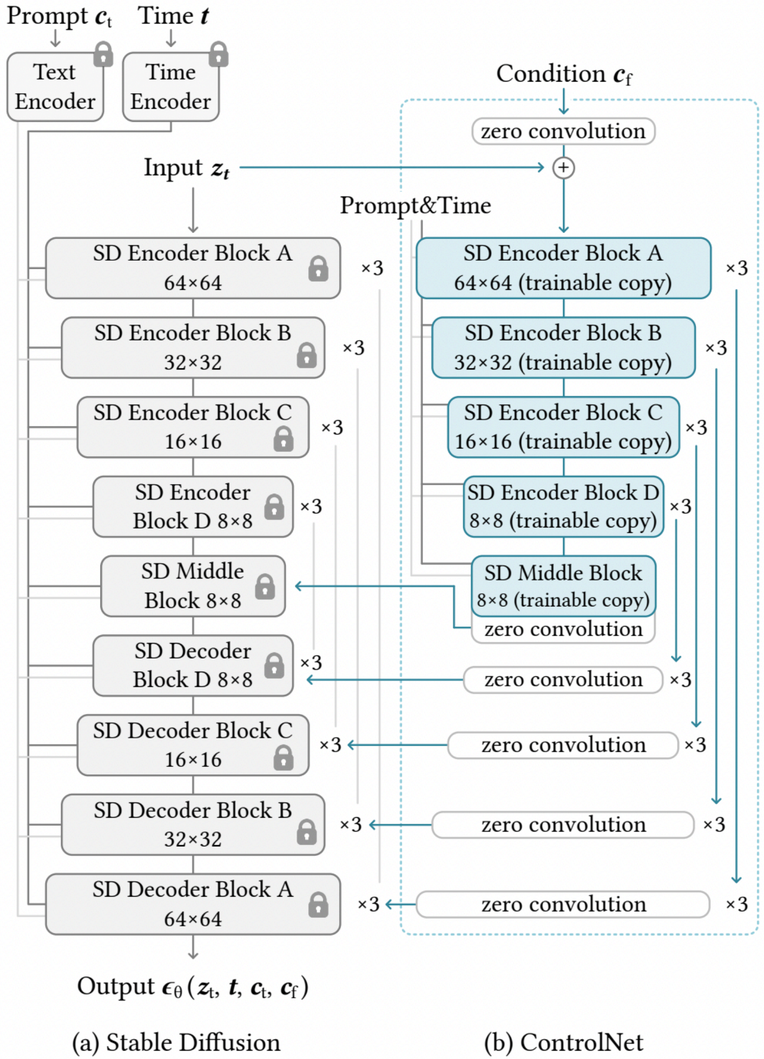
- Controlnet은 sketch, canny, pose 등의 abstract된 condition 정보를 통해 image를 생성할 수 있도록 하기 위해 고안되었다.



Zhang, Lvmin, Anyi Rao, and Maneesh Agrawala. "Adding conditional control to text-to-image diffusion models." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023.

ControlNet - 구조

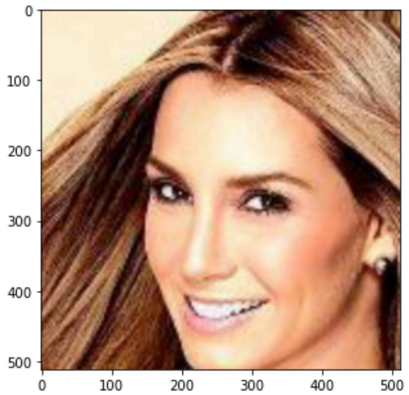
- Pre-trained된 denoising U-net으로부터 encoder를 복사하고, decoder 부분은 zero initialized convolution으로 대체하고 decoder의 결과를 기존 U-net의 decoder에 더한다. (add operation)
- 이렇게 구성하면 새로 구성한 decoder의 결과가 처음에는 zero여서 기존의 결과에 영향을 주지 않는다.
- 트레이닝이 진행됨에 따라 자연스럽게 새 encoder와 decoder가 condition의 정보를 이용하게끔 된다.
- 기존의 U-net은 트레이닝이 추가로 되지 않는 freez 상태로 두므로 기존에 가지고 있던 이미지 생성능력에 손상을 가하지 않는다.

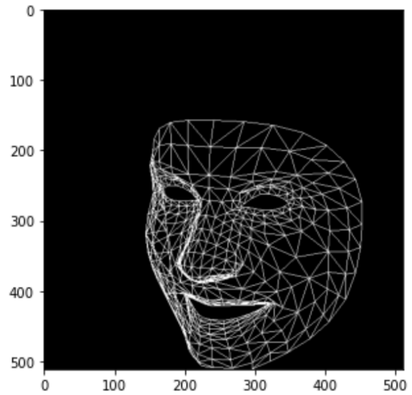


Zhang, Lvmin, Anyi Rao, and Maneesh Agrawala. "Adding conditional control to text-to-image diffusion models." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023.

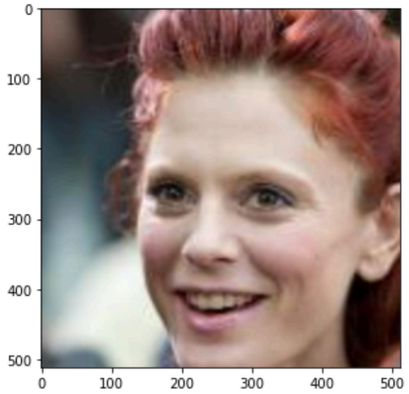
ControlNet - 실습 - 1. ControlNet Dataset 만들기

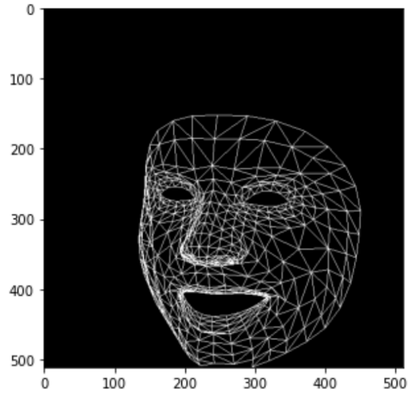
- “sd-scripts-example/contolnet/celeba landmarks dataset create.ipynb” 실행
  CelebA의 얼굴 이미지를 mediapipe 라이브러리를 이용하여 face mesh 이미지 생성 (시간 오래 걸림)





/data/sd_dataset/celeba/1.jpg





ControlNet - 실습 - 2. Dataset .toml 파일 만들기

- sd-scripts/toml/celeba_mesh_controlnet.toml 파일을 만들어 image와 condition 정보를 저장

```toml
[general]
shuffle_caption = true

[[datasets]]
resolution = 512
batch_size = 2

  [[datasets.subsets]]
  image_dir = '/data/sd-dataset/celeba_controlnet/images'
  caption_extension = ".txt"
  conditioning_data_dir = "/data/sd-dataset/celeba_controlnet/conditioning_images"
```

sd-scripts-example/controlnet/celeba_mesh_controlnet.toml

ControlNet - 실습 - 3. Training

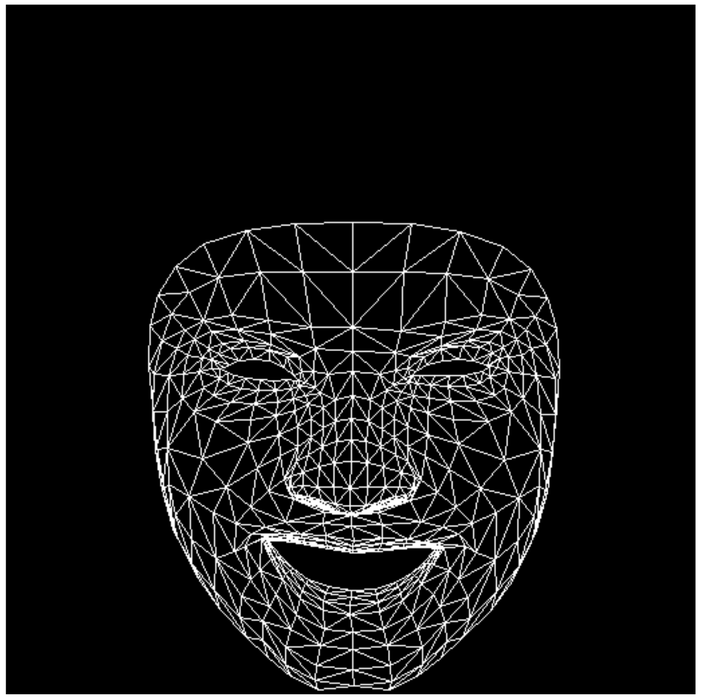
- sd-scripts 디렉토리에서 다음 명령을 입력

```
accelerate launch train_controlnet_fixed.py \
    --pretrained_model_name_or_path="/data/sd_files/checkpoint/beautifulRealistic_v7.safetensors" \
    --dataset_config=toml/celeba_mesh_controlnet.toml \
   --output_dir=/data/sd_results/celeba_mesh_controlnet \
    --output_name=celeba_mesh_controlnet \
    --save_model_as=safetensors  \
    --max_train_steps=100000 \
    --save_every_n_steps=1000 \
    --learning_rate=2e-4 \
    --optimizer_type="AdamW" \
    --mixed_precision="fp16" \
    --gradient_checkpointing
```

ControlNet - 실습 - 4. 트레이닝 결과 확인

- “sd-scripts-example/controlnet/diffusers controlnet run.ipynb” 실행

원본



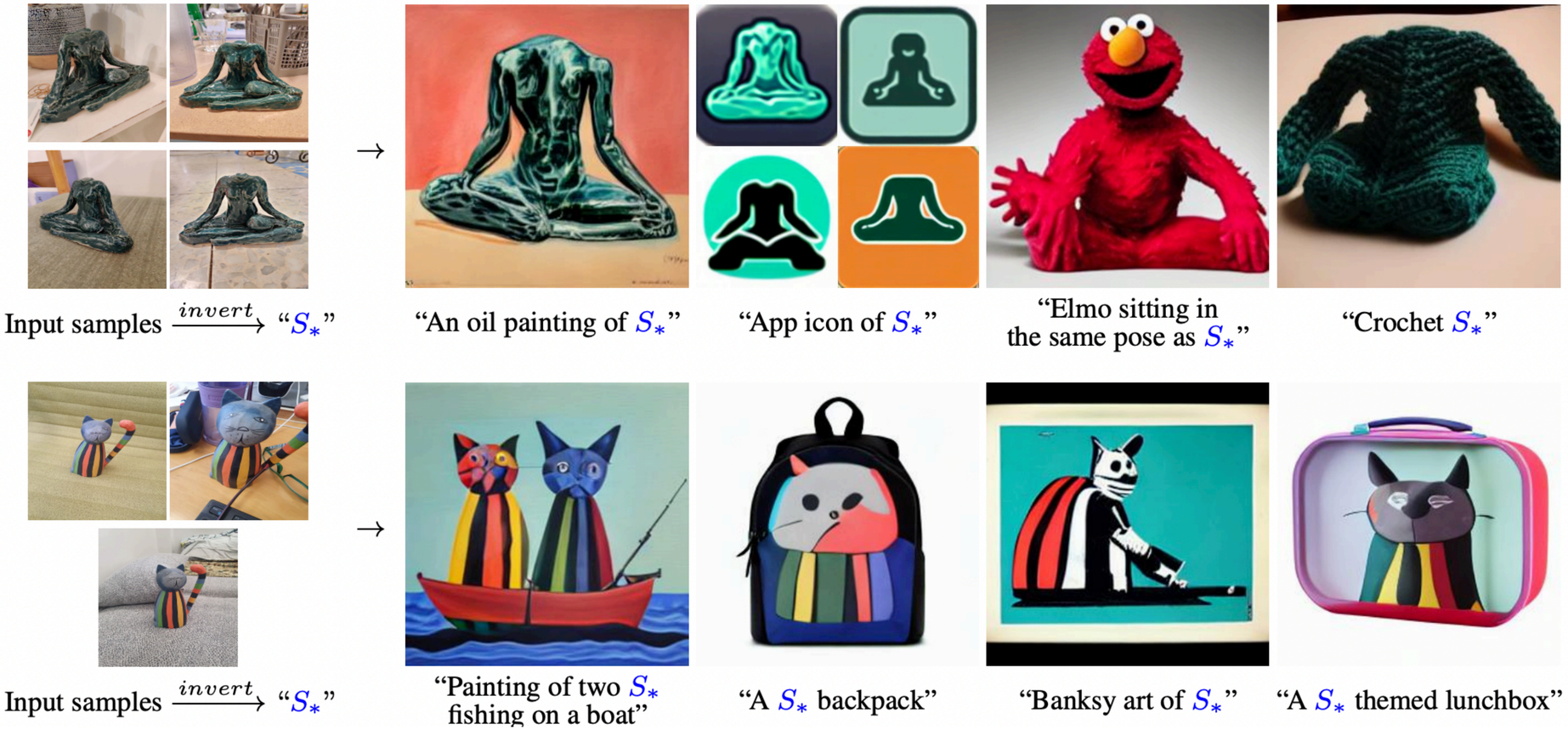
추출된 face mesh

생성된 이미지

Textual Inversion

Textual Inversion - 개요

- Textual Inversion도 DreamBooth와 마찬가지로 3-5장의 객체 이미지 만으로 새로운 이미지를 만들어내는 작업을 함



Gal, Rinon, et al. "An image is worth one word: Personalizing text-to-image generation using textual inversion." arXiv preprint arXiv:2208.01618 (2022).

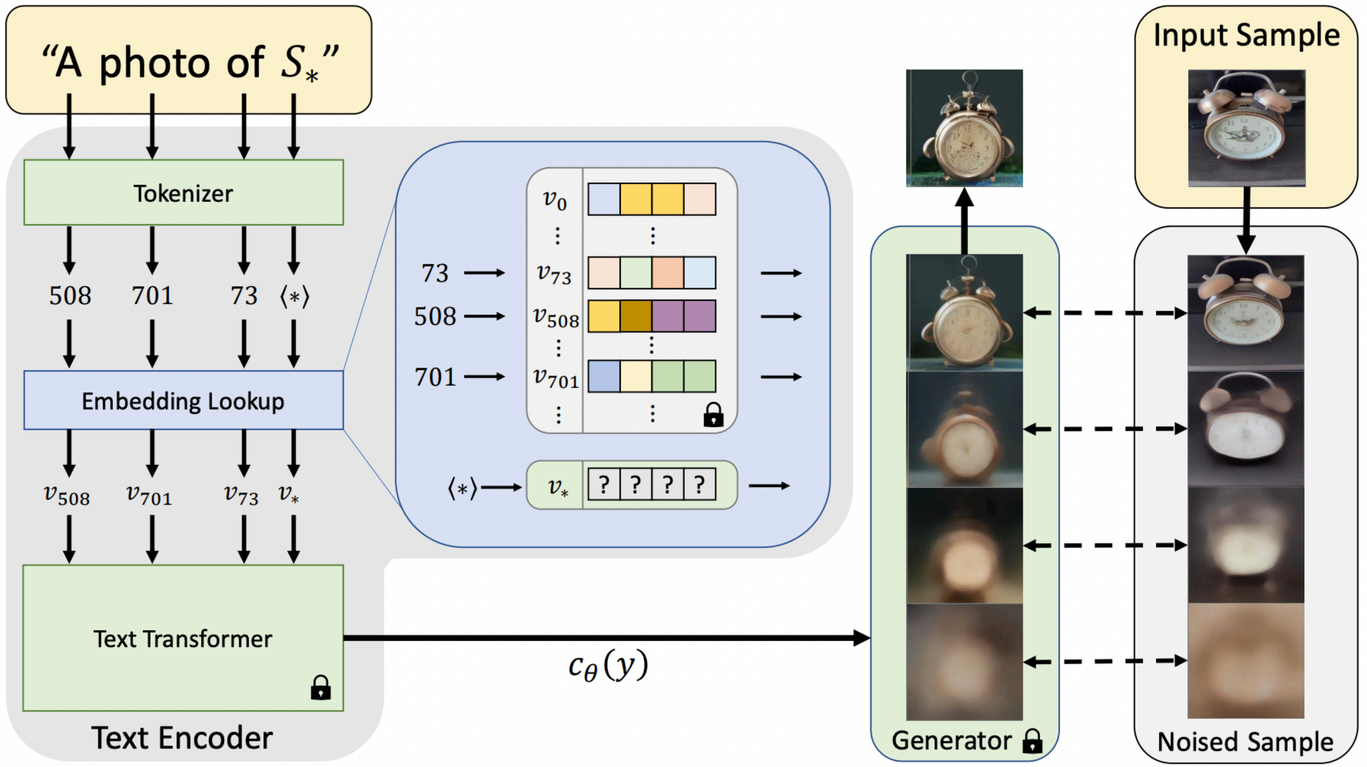
Textual Inversion - 구조

- Textual Inversion도 DreamBooth와 같이 personalizing된 이미지 생성을 제공하지만 방식이 다름

- DreamBooth는 모델 전체를 트레이닝하거나 LoRA를 이용해서 트레이닝하는데 반해 Textual Inversion은 Embedding Lookup Table을 트레이닝함

  사용자가 새로 지정한 S_* text는 새로운 token ID를 부여받게 되고, Embedding Lookup Table을 통해 주어진 이미지를 표현하는 Vector가 되도록 트레이닝됨

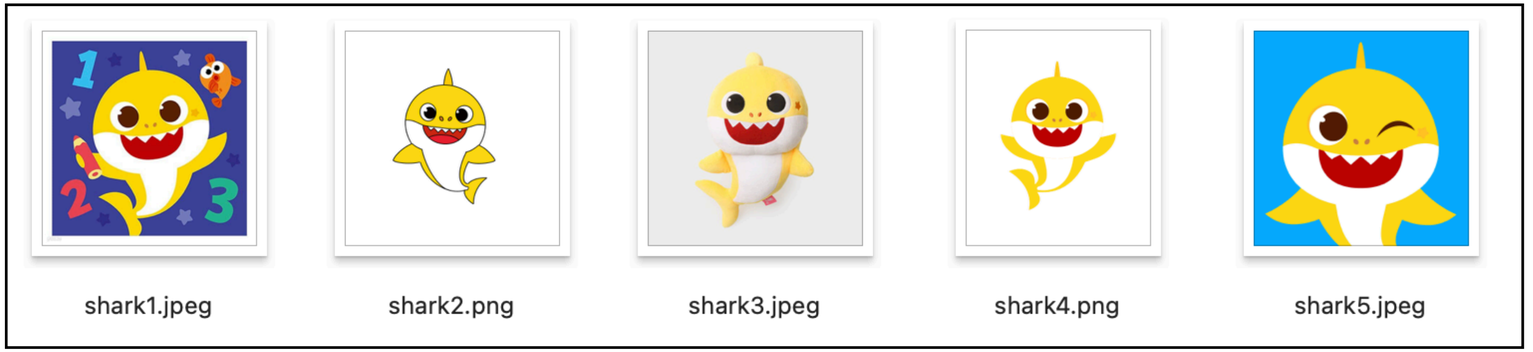
- 이는 논문의 제목인 “An Image is worth one word”처럼 원하는 객체가 하나의 단어에 대응할 수 있음을 의미함



Gal, Rinon, et al. "An image is worth one word: Personalizing text-to-image generation using textual inversion." arXiv preprint arXiv:2208.01618 (2022).

Textual Inversion - 실습 - 1. 데이터 수집

- 인터넷에서 원하는 오브젝트 5장을 다운로드 받습니다.



Textual Inversion - 실습 - 2. PNG 변환

- 데이터의 일관성을 위해 .png로 변환합니다.
  sd-scripts-example/textual_inversion/raw images to png.ipynb 실행

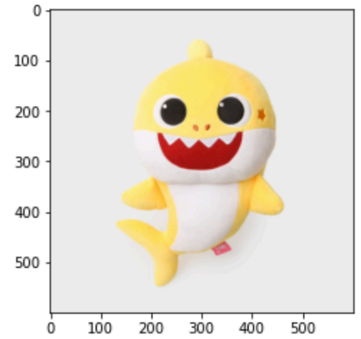
```python
import os
from PIL import Image

def convert_images_to_png(source_dir, target_dir):
    # Ensure target directory exists
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)

    # Loop through all files in the source directory
    for i, filename in enumerate(os.listdir(source_dir)):
        # Check if the file is an image
        if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.tiff', '.webp')):
            # Open the image
            img = Image.open(os.path.join(source_dir, filename))
            # Get the file name without the extension
            file_name_without_ext = os.path.splitext(filename)[0]
            # Save the image as PNG in the target directory
            img.save(os.path.join(target_dir, f'{i}.png'))
            print(i, f'Converted {filename} to PNG format.')
        else:
            print(filename, 'skip')

source_directory = '/data/sd_dataset/shark_raw'  # 원본 디렉토리 경로
target_directory = '/data/sd_dataset/shark_png'  # 변환된 파일을 저장할 디렉토리 경로

convert_images_to_png(source_directory, target_directory)
```

```
0 Converted shark3.jpeg to PNG format.
1 Converted shark2.png to PNG format.
2 Converted shark5.jpeg to PNG format.
3 Converted shark4.png to PNG format.
4 Converted shark1.jpeg to PNG format.
```

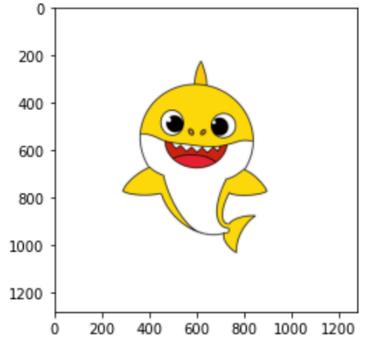
Textual Inversion - 실습 - 3. Image to Text

- “sd-scripts-example/textual_inversion/image to text.ipynb” 실행하여 text caption을 만들고 저장 (모델 다운로드 하는데 시간 오래 걸림)

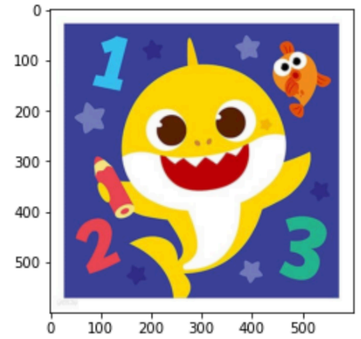
- BLIP2로 만든 prompt에 적절하게 user-defined char (‘mychar4’)를 추가 혹은 대체
  예) ‘a yellow and white stuffed shark with a big smile’
  ‘a yellow and white stuffed mychar4 with a big smile’



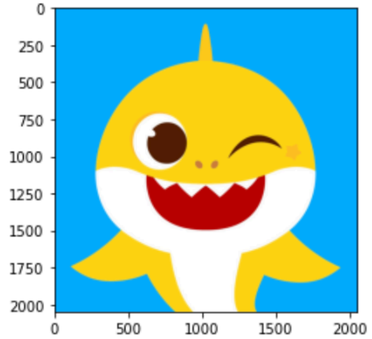
a yellow and white stuffed shark with a big smile



a cartoon shark with a big smile



a cartoon shark with numbers and fish



a cartoon shark with big eyes and a smile

Textual Inversion - 실습 - 4. Dataset .toml 파일 작성

- sd-scripts/toml에 .toml 파일 작성
  sd-scripts-example/textual_inversion/shark_inversion.toml 참고

```toml
[general]
enable_bucket = true
caption_extension = '.txt'

[[datasets]]
resolution = 512
batch_size = 2

  [[datasets.subsets]]
  image_dir = '/data/sd_dataset/shark_png'
  num_repeats = 1
```

sd-scripts-example/textual_inversion/shark_inversion.toml

Textual Inversion - 실습 - 5. Training

- sd-scripts 디렉토리에서 다음과 같이 입력하여 트레이닝 실행

```
accelerate launch --num_cpu_threads_per_process 1 train_textual_inversion.py \
    --pretrained_model_name_or_path="/data/sd_files/checkpoint/beautifulRealistic_v7.safetensors" \
    --dataset_config=toml/shark_inversion.toml \
    --output_dir=/data/sd_results/shark_inversion \
    --output_name=shark_inversion \
    --save_model_as=safetensors   \
    --max_train_steps=5000  \
    --save_every_n_steps=1000 \
    —learning_rate=2e-2  \
    --optimizer_type="AdamW" \
    --mixed_precision="fp16"  \
    --gradient_checkpointing \
    --token_string=mychar4 \          ← User-defined token에 해당하는 string
    --init_word=shark \               ← User-defined token의 초기화 값 설정
    --num_vectors_per_token=4         ← 학습될 token에 대응하는 vector 갯수
```

Textual Inversion - 실습 - 6. 결과보기

- sd-scripts-example/textual_inversion/diffusers textual_inversion.ipynb 실행하여 결과 확인

DreamBooth

DreamBooth - 개요

- DreamBooth는 3-5장의 이미지만으로 해당 객체를 이용한 새로운 이미지를 생성한다.

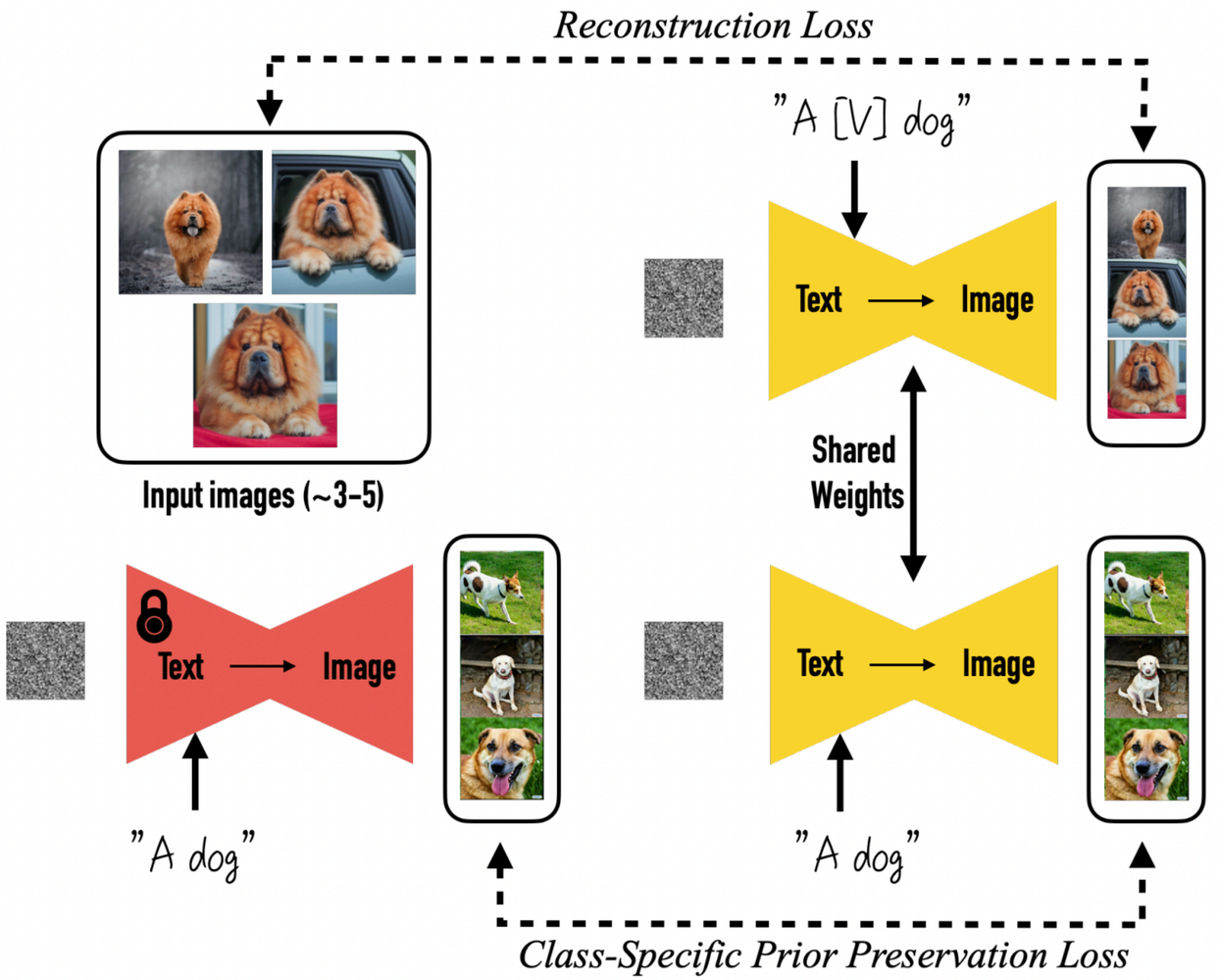
Ruiz, Nataniel, et al. "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023.

DreamBooth - 구조

- DreamBooth는 fine-tuning이나 LoRA로 트레이닝하는 방법에서 적은 데이터를 사용하기 위해 2가지 추가적인 수법을 제안합니다.

- Unique Identifier : text에서 [identifier] [class noun] 구조로 object를 명시합니다. Identifier는 model에서 잘 사용되지 않는 rare한 token이 되도록합니다. (대개 ohwx 사용)

- Prior Preservation Loss : object와 같은 class의 prior dataset들을 함께 트레이닝 합니다. 이 때, text에는 identifier가 없이 class noun만을 명시합니다.



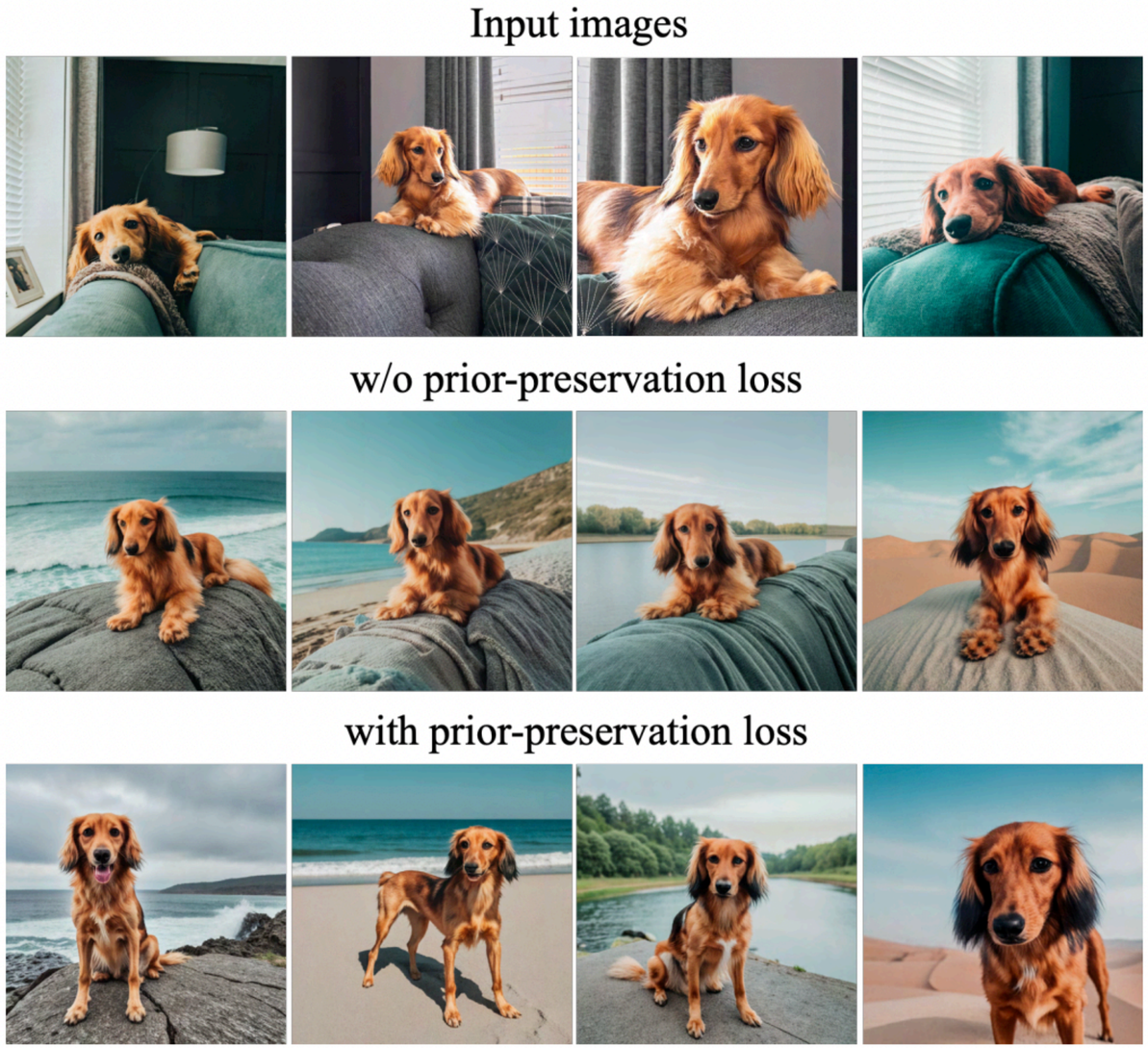
Ruiz, Nataniel, et al. "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023.

DreamBooth - Prior-Preservation Loss

- Prior-Preservation Loss를 사용하지 않으면 object와 다른 요소들이 잘 분리되지 않는 결과를 얻을 수 있습니다.

- 예를 들어, dog가 쇼파에 항상 앉아 있는 사진을 트레이닝할 경우, 바닷가의 모습을 표현하려고해도 쇼파의 모습이 같이 나타나거나, 항상 앉아있는 모습이 나올 수 있습니다.

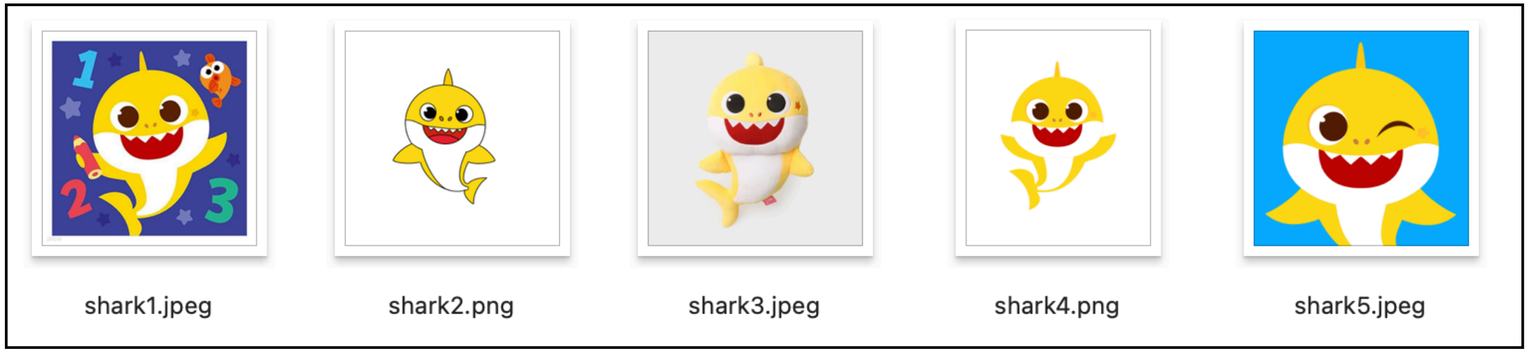
- 이를 방지 하기 위해 다양한 object의 모습을 담은 prior dataset을 함께 트레이닝 합니다.



Ruiz, Nataniel, et al. "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023.

DreamBooth - 실습 - 1. 데이터 수집 (Textual Inversion과 동일)

- 인터넷에서 원하는 오브젝트 5장을 다운로드 받습니다.



DreamBooth - 실습 - 2. PNG 변환 (Textual Inversion과 동일)

- 데이터의 일관성을 위해 .png로 변환합니다.
  sd-scripts-example/dreambooth/raw images to png.ipynb 실행

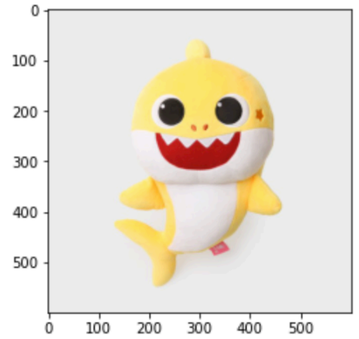
```python
import os
from PIL import Image

def convert_images_to_png(source_dir, target_dir):
    # Ensure target directory exists
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)

    # Loop through all files in the source directory
    for i, filename in enumerate(os.listdir(source_dir)):
        # Check if the file is an image
        if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.tiff', '.webp')):
            # Open the image
            img = Image.open(os.path.join(source_dir, filename))
            # Get the file name without the extension
            file_name_without_ext = os.path.splitext(filename)[0]
            # Save the image as PNG in the target directory
            img.save(os.path.join(target_dir, f'{i}.png'))
            print(i, f'Converted {filename} to PNG format.')
        else:
            print(filename, 'skip')

source_directory = '/data/sd_dataset/shark_raw'  # 원본 디렉토리 경로
target_directory = '/data/sd_dataset/shark_png'  # 변환된 파일을 저장할 디렉토리 경로

convert_images_to_png(source_directory, target_directory)
```

```
0 Converted shark3.jpeg to PNG format.
1 Converted shark2.png to PNG format.
2 Converted shark5.jpeg to PNG format.
3 Converted shark4.png to PNG format.
4 Converted shark1.jpeg to PNG format.
```

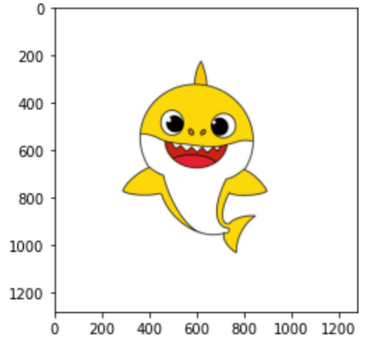
DreamBooth - 실습 - 3. Image to Text

- “sd-scripts-example/textual_inversion/image to text.ipynb” 실행하여 text caption을 만들고 저장 (모델 다운로드 하는데 시간 오래 걸림)

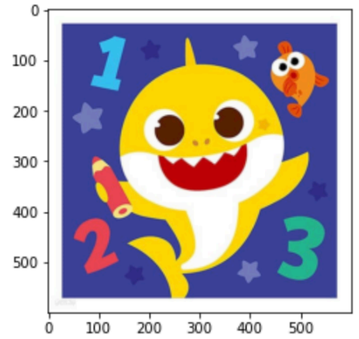
- BLIP2로 만든 prompt에 적절하게 rare한 identifier 추가
  예) ‘a yellow and white stuffed shark with a big smile’
  ‘a yellow and white stuffed ohwx shark with a big smile’



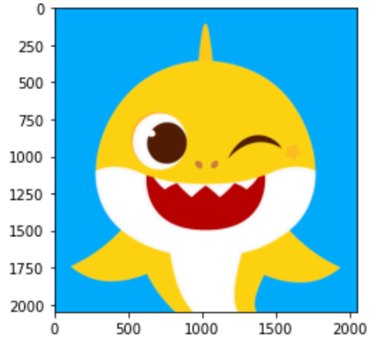
a yellow and white stuffed shark with a big smile



a cartoon shark with a big smile



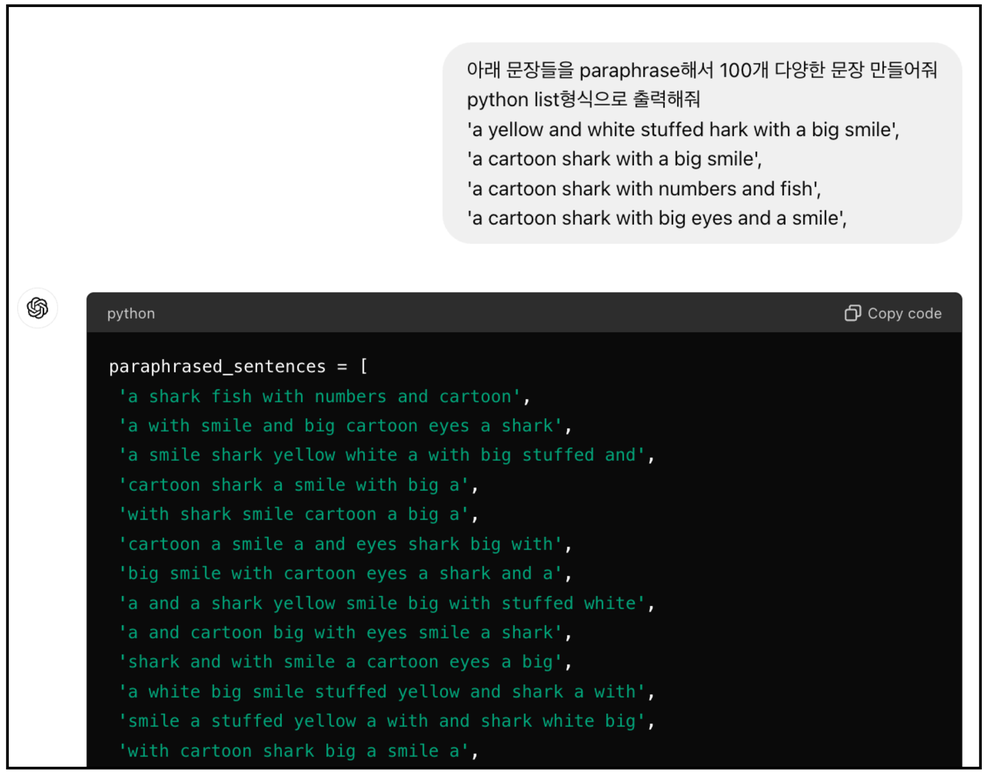
a cartoon shark with numbers and fish



a cartoon shark with big eyes and a smile

DreamBooth - 실습 - 4. Prior Dataset 만들기

- “sd-scripts-example/dreambooth/make prior dataset.ipynb” 실행하여 identifier가 없는 prior 이미지들을 생성합니다. (원래는 생성하지 않은 진짜 데이터셋 사용하는 것이 좋음)
- Prior 이미지를 만들기 위한 prompt는 ChatGPT를 이용해서 만들 수 있습니다.



ChatGPT를 이용하여 prompt 만들기

생성된 prior dataset

DreamBooth - 실습 - 5. Dataset .toml 파일 작성

- sd-scripts/toml에 .toml 파일 작성
  sd-scripts-example/dreambooth/shark_db.toml 참고

```toml
[general]
enable_bucket = true
caption_extension = '.txt'

[[datasets]]
resolution = 512
batch_size = 2

  [[datasets.subsets]]
  image_dir = '/data/sd_dataset/shark_png'
  num_repeats = 1

  [[datasets.subsets]]
  image_dir = '/data/sd-dataset/shark_prior'
  num_repeats = 1
  is_reg = true
```

Regularization Dataset이라는 의미

Sd-scripts-example/dreambooth/shark_db.toml

DreamBooth - 실습 - 6. Training

- sd-scripts 디렉토리에서 다음과 같이 입력하여 트레이닝 실행

```
accelerate launch --num_cpu_threads_per_process 1 train_db.py \
    --pretrained_model_name_or_path="/data/sd_files/checkpoint/beautifulRealistic_v7.safetensors" \
    --dataset_config=toml/shark_db.toml  \
    --output_dir=/data/sd_results/shark_db \
    --output_name=shark_db \
    --save_model_as=safetensors \
    --prior_loss_weight=1.0  \
    --max_train_steps=10000  \
    --save_every_n_steps=1000 \
    --learning_rate=1e-6  \
    --optimizer_type="AdamW"  \
    --mixed_precision="fp16"  \
    --gradient_checkpointing
```

DreamBooth - 실습 - 7. 결과보기

- sd-scripts-example/dreambooth/diffusers dreambooth run.ipynb 실행하여 결과 확인

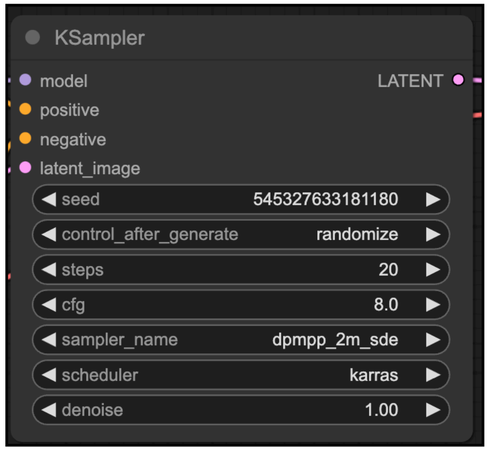
ComfyUI Inference

ComfyUI

- Stable diffusion을 inference하는 방법은 Diffusers, WebUI, ComfyUI를 사용하는 3가지 방법이 있음
- Diffusers를 이용한 방법은 code를 사용한다는 장점이 있으나 WebUI, ComfyUI 등으로 작성한 프로세스와 정확히 일치시키기 어렵다는 단점이 있음
- WebUI, ComfyUI에서는 stable diffusion의 기능 뿐 아니라 faceswap을 위한 Reactor 등 여러 외부 라이브러리들을 활용할 수 있는데 Diffusers를 사용하여 inference하려는 경우 이들과의 호환성 문제가 있음
- 따라서 code로 inference를 하려는 경우 ComfyUI의 소스코드를 이용하여 inference하는 방법이 실제 생산성 측면에서 유리함

https://github.com/crowsonkb/k-diffusion
https://github.com/Gourieff/comfyui-reactor-node

Stable Diffusion - ComfyUI Code Run




```python
NODE_CLASS_MAPPINGS = {
    "KSampler": KSampler,
    "CheckpointLoaderSimple": CheckpointLoaderSimple,
    "CLIPTextEncode": CLIPTextEncode,
    "CLIPSetLastLayer": CLIPSetLastLayer,
    "VAEDecode": VAEDecode,
    "VAEEncode": VAEEncode,
    "VAEEncodeForInpaint": VAEEncodeForInpaint,
    "VAELoader": VAELoader,
```

https://github.com/comfyanonymous/ComfyUI/blob/master/nodes.py

```python
class KSampler:
    @classmethod
    def INPUT_TYPES(s):
        return {"required":
                    {"model": ("MODEL",),
                    "seed": ("INT", {"default": 0, "min": 0, "max": 0xffffffffffffffff}),
                    "steps": ("INT", {"default": 20, "min": 1, "max": 10000}),
                    "cfg": ("FLOAT", {"default": 8.0, "min": 0.0, "max": 100.0, "step":0.1, "round": 0.01}),
                    "sampler_name": (comfy.samplers.KSampler.SAMPLERS, ),
                    "scheduler": (comfy.samplers.KSampler.SCHEDULERS, ),
                    "positive": ("CONDITIONING", ),
                    "negative": ("CONDITIONING", ),
                    "latent_image": ("LATENT", ),
                    "denoise": ("FLOAT", {"default": 1.0, "min": 0.0, "max": 1.0, "step": 0.01}),
                     }
                }

    RETURN_TYPES = ("LATENT",)
    FUNCTION = "sample"

    CATEGORY = "sampling"

    def sample(self, model, seed, steps, cfg, sampler_name, scheduler, positive, negative, latent_image, denoise=1.0):
        return common_ksampler(model, seed, steps, cfg, sampler_name, scheduler, positive, negative, latent_image, denoise=denoise)
```

https://github.com/comfyanonymous/ComfyUI/blob/master/nodes.py

- 예를들어 sampling을 담당하는 KSampler 컴포넌트는 node.py의 NODE_CLASS_MAPPING에 KSampler로 등록되어 있으며, 이는 class KSampler에 대응하고 sample() method를 호출하여 실행이 가능함

Stable Diffusion - ComfyUI Code Run

- 실습
  ComfyUI-example/ComfyUI code run.ipynb

Load Nodes

In [3]:
```
import comfy
from nodes import init_external_custom_nodes, NODE_CLASS_MAPPINGS

init_external_custom_nodes()

NODE_CLASS_MAPPINGS
```

```
Total VRAM 24268 MB, total RAM 257801 MB
pytorch version: 2.3.1
Set vram state to: NORMAL_VRAM
Device: cuda:0 NVIDIA GeForce RTX 3090 : native
Using pytorch cross attention

Import times for custom nodes:
   0.0 seconds: /home/gaudio/ste/projects/lsd/ComfyUI/custom_nodes/websocket_image_save.py
```

Out[3]:
```
{'KSampler': nodes.KSampler,
 'CheckpointLoaderSimple': nodes.CheckpointLoaderSimple,
 'CLIPTextEncode': nodes.CLIPTextEncode,
 'CLIPSetLastLayer': nodes.CLIPSetLastLayer,
 'VAEDecode': nodes.VAEDecode,
 'VAEEncode': nodes.VAEEncode,
 'VAEEncodeForInpaint': nodes.VAEEncodeForInpaint,
 'VAELoader': nodes.VAELoader,
 'EmptyLatentImage': nodes.EmptyLatentImage,
 'LatentUpscale': nodes.LatentUpscale,
```

In [4]:
```
checkpoint_loader_simple = NODE_CLASS_MAPPINGS['CheckpointLoaderSimple']()
clip_text_encode = NODE_CLASS_MAPPINGS['CLIPTextEncode']()
ksampler = NODE_CLASS_MAPPINGS['KSampler']()
empty_latent_image = NODE_CLASS_MAPPINGS['EmptyLatentImage']()
vae_encode = NODE_CLASS_MAPPINGS['VAEEncode']()
vae_decode = NODE_CLASS_MAPPINGS['VAEDecode']()
```

감사합니다