

스마트팩토리융합캡스톤디자인 2

AI 알고리즘 기반 연구논문

구현 및 실습 결과보고서

2020312336 김나현

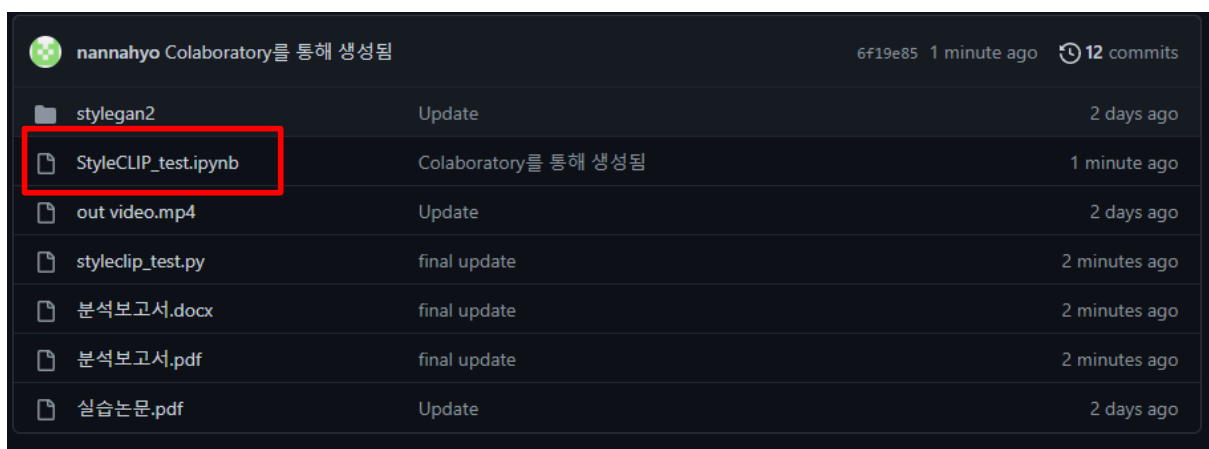
연구 논문 : StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery

출처 : <https://paperswithcode.com/paper/styleclip-text-driven-manipulation-of>

구현 환경 : 구글 코랩 (Google Colab)

<Github 에서 소스코드 열기 및 구현 환경 설정>

- 김나현(nannahyo)의 Github 에 접속한다. <https://github.com/nannahyo/test.git>
- 아래의 그림과 같은 화면에서 “StyleCLIP_test.ipynb”를 누른다.



* ‘StyleCLIP_test.ipynb’ : 소스코드 및 Colab 연결 파일

* ‘out video’ : 실행 결과로 저장된 이미지 변경 과정 비디오

* ‘실습논문’ : 연구 논문 pdf 파일

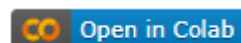
* ‘분석보고서.docx’ : 분석보고서 word 파일

* ‘분석보고서.pdf’ : 분석보고서 pdf 파일

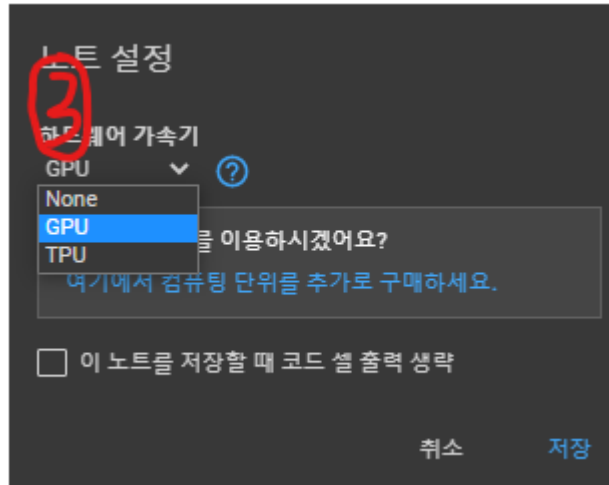
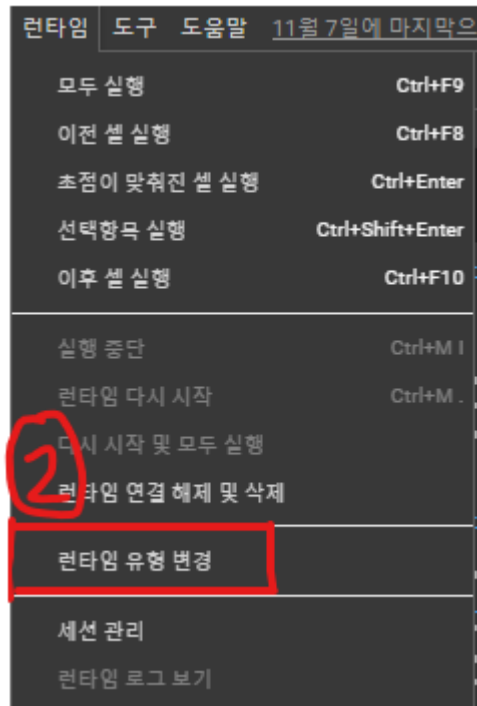
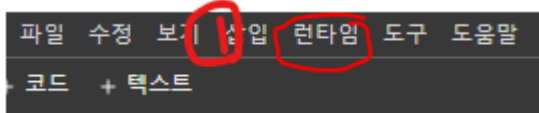
* ‘StyleCLIP_test.py’ : Python 형식의 소스코드

* ‘stylegan2’ : 모델 및 파일 저장소

- “Open in Colab”을 누르고, Colab 을 실행한다.



- Colab 의 런타임 유형을 “GPU”로 변경한다. (아래 사진 참고)



<코드 실행 및 설명>

1. Install CLIP

가장 먼저 CLIP 라이브러리를 설치한다. CLIP은 image Encoder와 text Encoder를 포함하고 있는 네트워크로, 특정한 이미지와 특정한 문장이 서로 얼마나 닮아 있는지, 그 similarity를 구할 수 있도록 해준다.

[코드]

```
!pip install ftfy regex tqdm
!pip install git+https://github.com/openai/CLIP.git
```

[구현 사진]

* 아래와 같은 경고 메시지가 나올 경우, '무시하고 계속하기'를 누르고 기다린다.

경고: 이 노트는 Google에서 작성하지 않았습니다.

이 노트는 GitHub에서 로드됩니다. 노트가 Google에 저장된 데이터에 액세스하거나 다른 세션의 데이터 및 사용자 인증 정보를 읽을 권한을 요청할 수 있습니다. 노트를 실행하기 전에 소스 코드를 검토하세요.

취소

무시하고 계속하기

+ 설치가 완료되면, 한번 더 실행해서 아래와 같이 'Requirement already satisfied' 메시지를 확인한다.

```
!pip install ffmpeg regex tqdm
!pip install git+https://github.com/openai/CLIP.git

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: ffmpeg in /usr/local/lib/python3.7/dist-packages (6.1.1)
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (2022.6.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (4.64.1)
Requirement already satisfied: wwidth=0.2.5 in /usr/local/lib/python3.7/dist-packages (from ffmpeg) (0.2.5)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-l8bhif_e
  Running command git clone --https://github.com/openai/CLIP.git /tmp/pip-req-build-l8bhif_e
Requirement already satisfied: ffmpeg in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (6.1.1)
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (2022.6.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (4.64.1)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (1.12.1+cu113)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (0.13.1+cu113)
Requirement already satisfied: wwidth=0.2.5 in /usr/local/lib/python3.7/dist-packages (from ffmpeg->clip==1.0) (0.2.5)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch->clip==1.0) (4.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (1.21.6)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (2.23.0)
Requirement already satisfied: pillow<3.4, >=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (7.1.2)
Requirement already satisfied: charset-normalizer<3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->clip==1.0) (3.0.4)
Requirement already satisfied: urllib3<1.25.0, >=1.25.1, <1.26, >=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->clip==1.0) (1.24.3)
Requirement already satisfied: idna<3, >=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->clip==1.0) (2.10)
Requirement already satisfied: certifi<2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->clip==1.0) (2022.9.24)
Building wheels for collected packages: clip
  Building wheel for clip (setup.py) ... done
  Created wheel for clip: filename=clip-1.0-py3-none-any.whl size=1369409 sha256=3d8287a7aae1d35d6fb601af5813bff9c055eaa27ff7460aee40ecb00a26c348
  Stored in directory: /tmp/pip-ephem-wheel-cache-v7m941tz/wheels/fd/b9/c3/5b4470e35ed76e174bfff77c92f91da82098d5e35fd5bc8dcac
Successfully built clip
Installing collected packages: clip
Successfully installed clip-1.0
```

2. Load the Pre-trained StyleGAN Mode

사전 학습된 StyleGAN Model을 로드한다. StyleGAN은 고해상도의 이미지를 생성하기에 적합한 architecture이다. 본 논문에서는 기본 method로 StyleGAN version 2 model을 사용하기 때문에, 이 모델을 다운로드 받아 실행할 수 있도록 한다.

[코드]

```
!git clone https://github.com/ndb796/StyleCLIP-Tutorial
%cd StyleCLIP-Tutorial
```

```
!wget https://postechackr-my.sharepoint.com/:u:/g/personal/dongbinna_postech_ac_kr/EVv6yusEt1tFhrL3TCu0Ta4BlpzW3eBMTS0yTPKodNHsNA?download=1 -O stylegan2-ffhq-config-f.pt
```

[구현 사진]

```
[3] git clone https://github.com/nannahyo/test
    kcd test

Cloning into 'test'...
remote: Enumerating objects: 277, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 277 (delta 3), reused 21 (delta 2), pack-reused 253
Receiving objects: 100% (277/277), 57.09 MiB | 17.19 MiB/s, done.
Resolving deltas: 100% (97/97), done.
/content/test

[4] wget https://postechackr-my.sharepoint.com/:u:/g/personal/dongbinna_postech_ac_kr/EVv6yusEt1tFhrL3TCu0Ta4B1pzW3eBMTS0vTPKodNHsNA?download=1 -O stylegan2-ffhq-config-f.pt

--2022-11-08 01:11:39-- https://postechackr-my.sharepoint.com/:u:/g/personal/dongbinna_postech_ac_kr/EVv6yusEt1tFhrL3TCu0Ta4B1pzW3eBMTS0vTPKodNHsNA?download=1
Resolving postechackr-my.sharepoint.com (postechackr-my.sharepoint.com)... 13.107.136.9, 13.107.138.9
Connecting to postechackr-my.sharepoint.com (postechackr-my.sharepoint.com)|13.107.136.9|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /personal/dongbinna_postech_ac_kr/Documents/Research/models/stylegan2-ffhq-config-f.pt?ga=1 [following]
--2022-11-08 01:11:40-- https://postechackr-my.sharepoint.com/personal/dongbinna_postech_ac_kr/Documents/Research/models/stvlegan2-ffhq-config-f.pt?ga=1
Reusing existing connection to postechackr-my.sharepoint.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 381462551 (364M) [application/octet-stream]
Saving to: 'stylegan2-ffhq-config-f.pt'

stylegan2-ffhq-conf 100%[=====] 363.79M  78.4MB/s   in 5.2s

2022-11-08 01:11:45 (69.7 MB/s) - 'stylegan2-ffhq-config-f.pt' saved [381462551/381462551]
```

3. 위의 작업을 통해 모델 가중치 파일이 모두 다운로드 완료되었으면, 생성자 모델을 초기화한다.

[코드]

```
import torch
from stylegan2.model import Generator

g_ema = Generator(1024, 512, 8)
g_ema.load_state_dict(torch.load('stylegan2-ffhq-config-f.pt')['g_ema'], strict=False)
g_ema.eval()
g_ema = g_ema.cuda()
```

[구현 사진]

```
[5] import torch
    from stylegan2.model import Generator

    g_ema = Generator(1024, 512, 8)
    g_ema.load_state_dict(torch.load('stylegan2-ffhq-config-f.pt')['g_ema'], strict=False)
    g_ema.eval()
    g_ema = g_ema.cuda()
```

4. CLIP Loss

CLIP Loss함수를 정의한다. CLIP Loss는 하나의 이미지와 텍스트를 입력으로 받아 그 이미지와 텍스트의 similarity를 return하는 방식으로 구현되었다.

[코드]

```
import clip

class CLIPLoss(torch.nn.Module):

    def __init__(self):
        super(CLIPLoss, self).__init__()
        self.model, self.preprocess = clip.load("ViT-B/32", device="cuda")
        self.upsample = torch.nn.Upsample(scale_factor=7)
        self.avg_pool = torch.nn.AvgPool2d(kernel_size=32)

    def forward(self, image, text):
        image = self.avg_pool(self.upsample(image))
        similarity = 1 - self.model(image, text)[0] / 100
        return similarity
```

[구현 사진]



```
import clip

class CLIPLoss(torch.nn.Module):

    def __init__(self):
        super(CLIPLoss, self).__init__()
        self.model, self.preprocess = clip.load("ViT-B/32", device="cuda")
        self.upsample = torch.nn.Upsample(scale_factor=7)
        self.avg_pool = torch.nn.AvgPool2d(kernel_size=32)

    def forward(self, image, text):
        image = self.avg_pool(self.upsample(image))
        similarity = 1 - self.model(image, text)[0] / 100
        return similarity
```

5. Latent Optimization

기본 설명 : Latent Optimization은 CLIP 네트워크에 포함되어있는 Text Encoder와 image Encoder, StyleGAN을이용해서 image manipulation을 진행한다. 즉 Latent vector w 를 업데이트하되, Latent vector w 로 만들어진 이미지의 그 inbuilding이 특정한 text prompt와 similarity가 높아질 수 있는 방향으로 Latent vector를 업데이트하는 것이다. 일반적으로 latent vector를 만들 때에는 특정한 이미지에서부터 인코딩을 하거나 랜덤 샘플링을 이용해 latent vector를 만들 수 있다. 여기서는 랜덤 latent vector를 사용한다.

‘mean_latent’는 평균치의 값을 갖고있는 latent를 의미한다. 이때 truncation 트릭을 이용해서 보다 realistic한 이미지를 만들어낼 수 있다. truncation은 특정한 latent vector가 mean_latent로부터 크게 벗어나지 않도록 latent를 잘라내주는 역할을 수행하는 트릭이다. 아래 코드에서처럼, 0.7정도의 값을 넣었을 때(truncation=0.7) 그럴싸한 랜덤 이미지가 만들어지는 경향이 있다.

StyleCLIP의 가장 기본적인 방법은 Optimization method이다.

$$\arg \min_{w \in \mathcal{W}} \underbrace{D_{\text{CLIP}}(G(w), t)}_{\text{For manipulation}} + \underbrace{\lambda_{\text{L2}} \|w - w_s\|_2 + \lambda_{\text{ID}} \mathcal{L}_{\text{ID}}(w)}_{\text{For similarity to the input image}}$$

$$\mathcal{L}_{\text{ID}}(w) = 1 - \langle R(G(w_s)), R(G(w)) \rangle$$

R : Pretrained *ArcFace* network

D_{clip} : Cosine distance between the CLIP embeddings

왼쪽에 보이는 CLIP Loss가 핵심이다. Latent vector w 를 원본 이미지와 유사하게 업테이트를 하는 동시에, 특정한 text prompt와 생성된 이미지가 유사한 similarity를 갖도록 같이 업데이트하는 것이다.

* 우리가 사용하고 있는 사전 학습된 StyleGAN 모델은 1024*1024의 고해상도 이미지를 만들어내기때문에 아래의 코드(image.resize((h // 2, w // 2)))처럼 너비와 높이를 2배씩 줄여서 화면에 간단히 출력될 수 있도록 하였다.

- a random latent vector를 a mean latent vector를 이용해 초기화하고 시각화하기

[코드]

```
from torchvision.utils import make_grid
from torchvision.transforms import ToPILImage

mean_latent = g_ema.mean_latent(4096)

latent_code_init_not_trunc = torch.randn(1, 512).cuda()
with torch.no_grad():
```

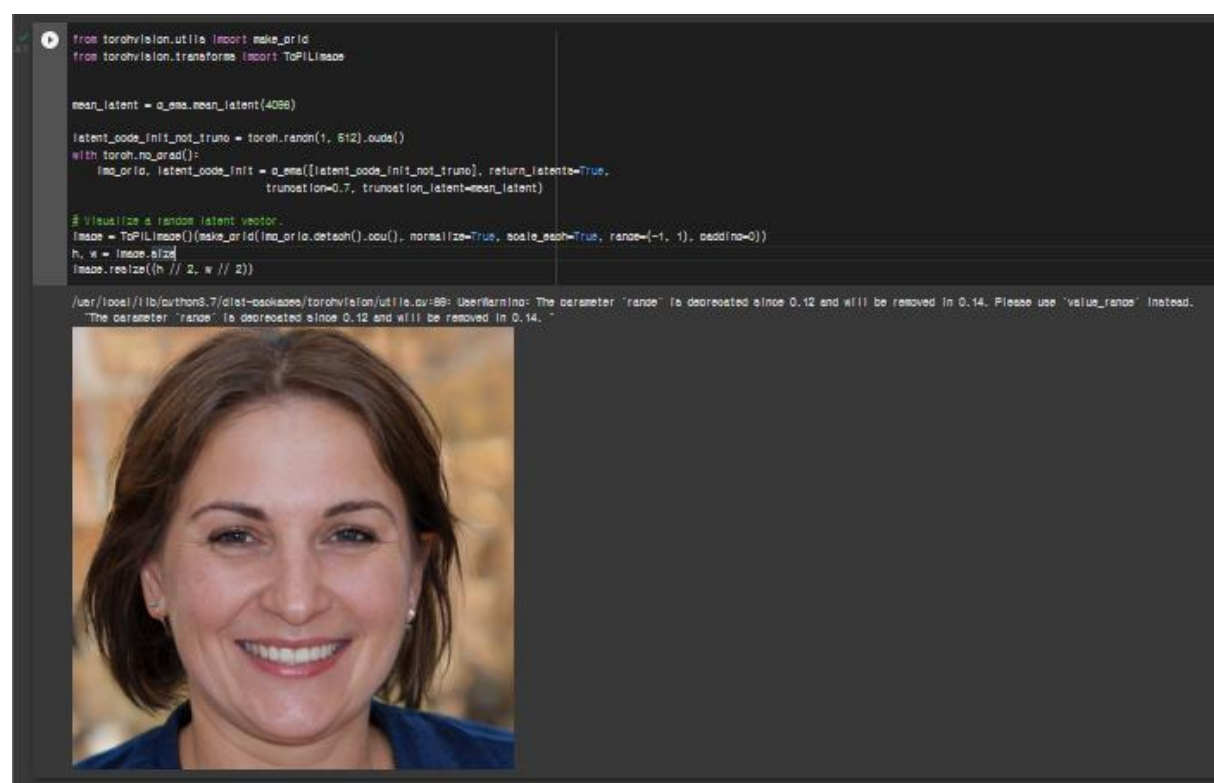
```

img_orig, latent_code_init = g_ema([latent_code_init_not_trunc], return_latents=True,
                                   truncation=0.7, truncation_latent=mean_latent)

# Visualize a random latent vector.
image = ToPILImage()(make_grid(img_orig.detach().cpu(), normalize=True, scale_each=True, range=(-1, 1), padding=0))
h, w = image.size
image.resize((h // 2, w // 2))

```

[구현 사진]



- 코드 구현을 위한 parameter set하기

아래의 코드에서 description은 text prompt이다.

λ_{L2} 는 가중치 parameter(λ_{L2})이다. λ_{L2} 가 커질수록 원본 이미지와 유사해지려는 힘이 강해지고, λ_{L2} 가 작다면 특정한 text prompt와 높은 similarity를 갖게 될 것이다.

[코드]

```

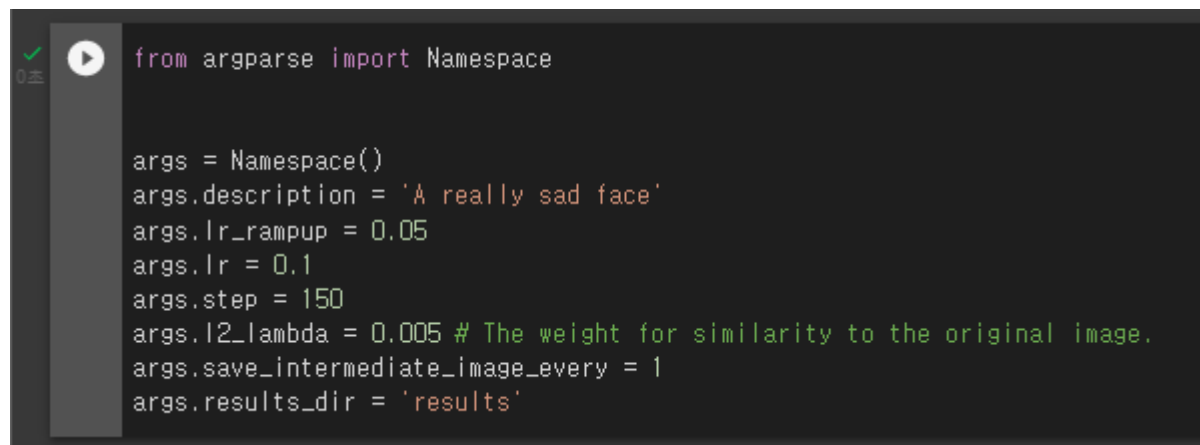
from argparse import Namespace

```



```
args = Namespace()
args.description = 'A really sad face'
args.lr_rampup = 0.05
args.lr = 0.1
args.step = 150
args.l2_lambda = 0.005 # The weight for similarity to the original image.
args.save_intermediate_image_every = 1
args.results_dir = 'results'
```

[구현 사진]



- latent vector를 optimize하고 결과 얻기

기본적으로 StyleCLIP은 learning rate을 각 스텝마다 적절히 조정해서 적용할 수 있도록 하는 learning rate adjustment function을 사용한다. 즉, 고정된 특정 learning rate만 사용하는 것이 아니라 스텝이 반복되는 과정에서 learning rate을 적절히 조정해서 optimization을 보다 매끄럽게 수행할 수 있도록 하는 것이다.

위에서 입력한 description 'A really sad face'는 먼저 tokenize 된 이후에, 실제 clip encoder에 들어갈 수 있다.

```
text_inputs = torch.cat([clip.tokenize(args.description)]).cuda()
```

Adam optimizer를 사용해서 latent vector를 업데이트할 수 있도록한다.

```
optimizer = optim.Adam([latent], lr=args.lr)
```

각각의 스텝마다 아래의 코드가 반복적으로 수행된다.

먼저 learning rate을 조정한다.

```
# Adjust the learning rate.  
t = i / args.step  
lr = get_lr(t, args.lr)  
optimizer.param_groups[0]["lr"] = lr
```

latent vector를 이용해서 image를 생성한다.

```
# Generate an image using the latent vector.  
img_gen, _ = g_ema([latent], input_is_latent=True, randomize_noise=False)
```

그렇게 생성된 image와 우리가 입력한 text prompt가 유사한 similarity를 갖도록 업데이트를 해준다.(c_loss)

```
c_loss = clip_loss(img_gen, text_inputs)
```

그와 동시에 초기 latent vector와 유사한 형태를 가지도록 latent vector를 제안하는 loss를 넣어준다.(l2_loss)

```
l2_loss = ((latent_code_init - latent) ** 2).sum()
```

이러한 두가지 loss를 이용해서 latent vector를 optimization한다.

```
loss = c_loss + args.l2_lambda * l2_loss
```

이러한 loss를 기반으로 backpropagation한 뒤에, gradient를 구한다.

```
# Get gradient and update the latent vector.  
optimizer.zero_grad()  
loss.backward()  
optimizer.step()
```

그렇게 구한 gradient를 이용하여 latent vector를 업데이트한다.

```
# Initialize the latent vector to be updated.  
latent = latent_code_init.detach().clone()  
latent.requires_grad = True  
  
clip_loss = CLIPLoss()  
optimizer = optim.Adam([latent], lr=args.lr)
```

이 코드 전체를 실행하는데에 약 2~3분 정도의 시간이 소요된다.

[코드]

```
import os
import math
import torchvision
from torch import optim

# The learning rate adjustment function.
def get_lr(t, initial_lr, rampdown=0.50, rampup=0.05):
    lr_ramp = min(1, (1 - t) / rampdown)
    lr_ramp = 0.5 - 0.5 * math.cos(lr_ramp * math.pi)
    lr_ramp = lr_ramp * min(1, t / rampup)

    return initial_lr * lr_ramp

text_inputs = torch.cat([clip.tokenize(args.description)]).cuda()
os.makedirs(args.results_dir, exist_ok=True)

# Initialize the latent vector to be updated.
latent = latent_code_init.detach().clone()
latent.requires_grad = True

clip_loss = CLIPLoss()
optimizer = optim.Adam([latent], lr=args.lr)

for i in range(args.step):
    # Adjust the learning rate.
    t = i / args.step
    lr = get_lr(t, args.lr)
    optimizer.param_groups[0]["lr"] = lr

    # Generate an image using the latent vector.
    img_gen, _ = g_ema([latent], input_is_latent=True, randomize_noise=False)

    # Calculate the loss value.
```

```

c_loss = clip_loss(img_gen, text_inputs)
l2_loss = ((latent_code_init - latent) ** 2).sum()
loss = c_loss + args.l2_lambda * l2_loss

# Get gradient and update the latent vector.
optimizer.zero_grad()
loss.backward()
optimizer.step()

# Log the current state.
print(f"lr: {lr}, loss: {loss.item():.4f}")
if args.save_intermediate_image_every > 0 and i % args.save_intermediate_image_
every == 0:
    with torch.no_grad():
        img_gen, _ = g_ema([latent], input_is_latent=True, randomize_noise=False)
        torchvision.utils.save_image(img_gen, f"results/{str(i).zfill(5)}.png", normalize=True, range=(-1, 1))

with torch.no_grad():
    img_orig, _ = g_ema([latent_code_init], input_is_latent=True, randomize_noise=False)

# Display the initial image and result image.
final_result = torch.cat([img_orig, img_gen])
torchvision.utils.save_image(final_result.detach().cpu(), os.path.join(args.results_dir
, "final_result.jpg"), normalize=True, scale_each=True, range=(-1, 1))

```

[구현 사진]

```
t = 1 / args.steps
lr = opt_lr(t, args.lr)
optimizer.param_groups[0]["lr"] = lr

# Generate an image using the latent vector.
img_gen, _ = d_ema([latent], input_is_latent=True, randomize_noise=False)

# Calculate the loss value.
o_loss = o_ld_loss(img_gen, text_inputs)
l2_loss = ((latent_code_init - latent) ** 2).sum()
loss = o_loss + args.l2_lambda * l2_loss

# Get gradient and update the latent vector.
optimizer.zero_grad()
loss.backward()
optimizer.step()

# Log the current state.
print(f"lr: {lr}, loss: {loss.item():.4f}")
if args.save_intermediate_image_every > 0 and 1 % args.save_intermediate_image_every == 0:
    with torch.no_grad():
        img_gen, _ = d_ema([latent], input_is_latent=True, randomize_noise=False)
        torchvision.utils.save_image(img_gen, f"results/{str(1).zfill(5)}.png", normalize=True, range=(-1, 1))

with torch.no_grad():
    img_orio, _ = d_ema([latent_code_init], input_is_latent=True, randomize_noise=False)

# Display the initial image and result image.
final_result = torch.cat([img_orio, img_gen])
torchvision.utils.save_image(final_result.detach().cpu(), os.path.join(args.results_dir, "final_result.jpg"), normalize=True, scale_each=True, range=(-1, 1))
```

100% [██] 335M/335M [00:09:00:00, 86.6M/s]

lr: 0.0, loss: 0.5146
lr: 0.013333333333333334, loss: 0.5146
lr: 0.026666666666666667, loss: 0.5001
lr: 0.04, loss: 0.5140
lr: 0.053333333333333334, loss: 0.5282
lr: 0.06666666666666667, loss: 0.5485
lr: 0.08, loss: 0.5425
lr: 0.09333333333333334, loss: 0.5374
lr: 0.1, loss: 0.5701
lr: 0.1, loss: 0.5525
lr: 0.1, loss: 0.5774
lr: 0.1, loss: 0.5823
lr: 0.1, loss: 0.5823
lr: 0.1, loss: 0.5498
lr: 0.1, loss: 0.5524
lr: 0.1, loss: 0.5594
lr: 0.1, loss: 0.5482
lr: 0.1, loss: 0.5311
lr: 0.1, loss: 0.5216
lr: 0.1, loss: 0.5168
lr: 0.1, loss: 0.5091
lr: 0.1, loss: 0.5168
lr: 0.1, loss: 0.5183
lr: 0.1, loss: 0.5146
lr: 0.1, loss: 0.5140
lr: 0.1, loss: 0.5003
lr: 0.1, loss: 0.7091
lr: 0.1, loss: 0.7803
lr: 0.1, loss: 0.7859

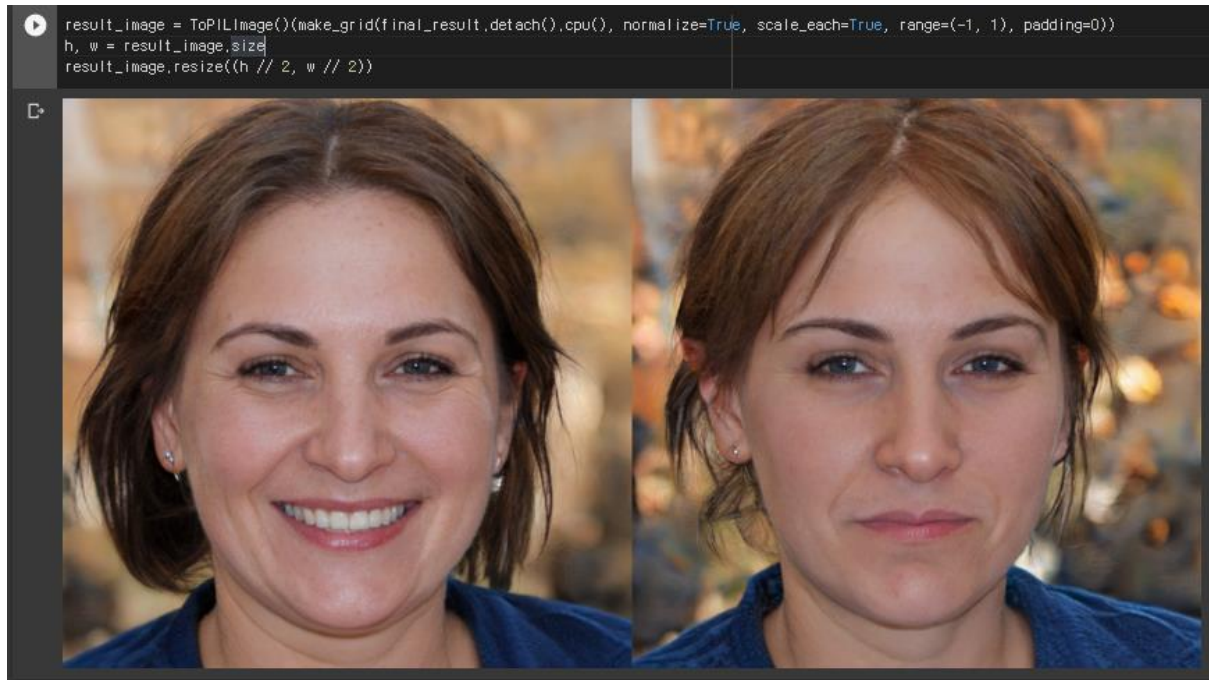
6. 결과 시각화하기

최적화 결과, 조금 슬픈 표정을 짓는 얼굴을 확인할 수 있다.

[코드]

```
result_image = ToPILImage()(make_grid(final_result.detach().cpu(), normalize=True, scale_each=True, range=(-1, 1), padding=0))
h, w = result_image.size
result_image.resize((h // 2, w // 2))
```

[구현 사진]



7. amination video 생성하고 다운로드받기

이미지가 서서히 변화하는 과정을 담은 비디오를 다운로드 받을 수 있다. 총 150번의 스텝을 반복했기 때문에 150 프레임으로 구성된 동영상을 생성하여 다운로드할 수 있다. 다운로드하는 코드를 실행하면 자동으로 된다. 보고서에 작성된 이미지에 대한 비디오는 Github에 'out video'으로 올라가있다.

[코드]

```
!ffmpeg -r 15 -i results/%05d.png -c:v libx264 -vf fps=25 -  
pix_fmt yuv420p out.mp4  
  
from google.colab import files  
files.download('out.mp4')
```

[구현 사진]

```

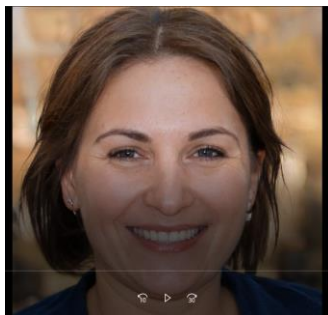
!ffmpeg -r 16 -i results/X05d.png -c:v libx264 -vf fps=25 -pix_fmt yuv420p out.mp4

from google.colab import files
files.download('out.mp4')

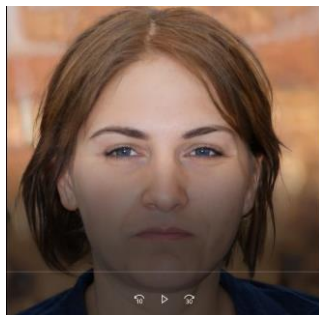
ffmpeg version 3.4.8-Ubuntu0.2 Copyright (c) 2000-2020 the FFmpeg developers
built with gcc 7 (Ubuntu 7.5.0-3ubuntu1~18.04)
configuration: --prefix=/usr --extra-version=Ubuntu0.2 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linux-gnu --enable-gpl --disable-stripping
libavutil 56. 78.100 / 56. 78.100
libavcodec 57.107.100 / 57.107.100
libavformat 57. 83.100 / 57. 83.100
libavdevice 57. 10.100 / 57. 10.100
libavfilter 6.107.100 / 6.107.100
libavresample 8. 7. 0 / 8. 7. 0
libswscale 4. 8.100 / 4. 8.100
libswresample 2. 9.100 / 2. 9.100
libpostproc 54. 7.100 / 54. 7.100
Input #0, image2, from 'results/X05d.png':
Duration: 00:00:09.00, start: 0.000000, bitrate: N/A
Stream #0:0: Video: png, rgb24(pc), 1024x1024, 25 fps, 25 tbr, 25 tbn, 25 tbc
Stream mapping:
Stream #0:0 -> #0:0 (png (native) -> h264 (libx264))
Press [q] to stop, [?] for help
[libx264 @ 0x556a1890de00] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 0x556a1890de00] profile High, level 3.2
[libx264 @ 0x556a1890de00] 264 - core 152 r2854 e9a5903 - H.264/MPEG-4 AVC codec - Copyleft 2003-2017 - http://www.videolan.org/x264.html - options: cabac=1 ref=3 deblock=1:0:0 analyse=0x
Output #0, mp4, to 'out.mp4':
Metadata:
encoder      : Lavf57.83.100
Stream #0:0: Video: h264 (libx264) (avc1 / 0x31637661), yuv420p, 1024x1024, q=1-1, 25 fps, 12800 tbn, 25 tbc
Metadata:
encoder      : Lavc57.107.100 libx264
Side data:
cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
frame= 250 fps= 18 q=1.0 Lsize= 3391kB time=00:00:09.88 bitrate=2811.8kbits/s speed=0.628x
video:3397kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.111070%
[libx264 @ 0x556a1890de00] frame I:3 Avg QP:21.46 size: 66370
[libx264 @ 0x556a1890de00] frame P:34 Avg QP:22.62 size: 25971
[libx264 @ 0x556a1890de00] frame B:169 Avg QP:25.10 size: 6170
[libx264 @ 0x556a1890de00] consecutive B-frames: 4.4% 24.0% 6.0% 66.6%
[libx264 @ 0x556a1890de00] mb I 116.4: 5.6% 79.4% 15.0%
[libx264 @ 0x556a1890de00] mb P 116.4: 8.0% 21.4% 2.2% P16.4: 37.4% 10.4% 5.3% 0.0% 0.0% skip:20.2%
[libx264 @ 0x556a1890de00] mb B 116.4: 0.8% 1.4% 0.1% B16.8: 17.6% 2.3% 0.6% direct: 9.9% skip:67.8% L0:42.7% L1:48.0% BI: 9.4%
[libx264 @ 0x556a1890de00] 8x8 transform intra:80.1% inter:72.7%

```

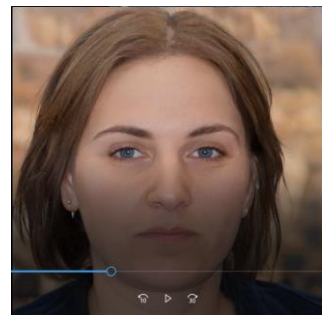
[동영상 캡처본]



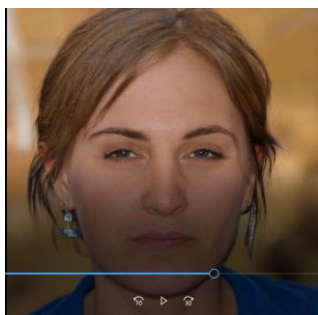
->



->



->



->

