

MBTI 유형별 얼굴 생성 모델

비트캠프 혁신 인공지능(서울) 4조

팀장 및 발표 : 전병준

팀원: 강청순, 황채연, 김현수

목차



왜 MBTI 인가?

'카를 융'의 분석심리학을 근거로 개발한
성격 유형 선호 지표

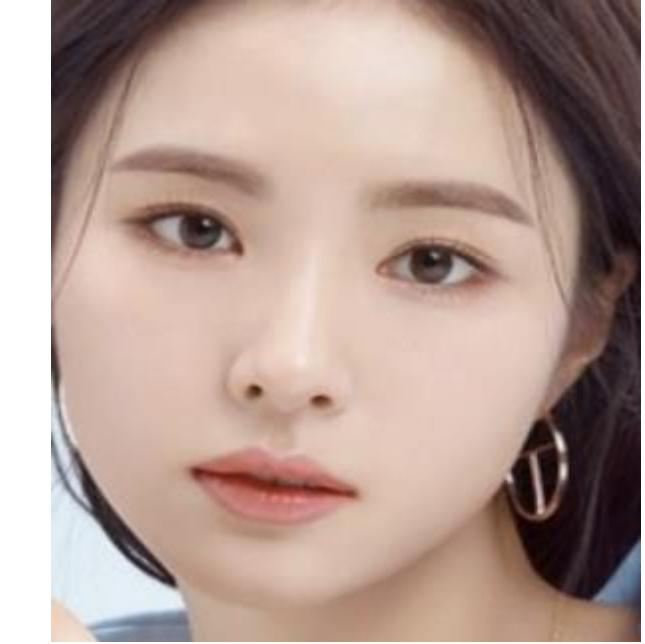
'사람의 성향을 16가지의 유형으로 구분하면
그 차이가 얼굴형에도 구분되어 나타나지 않
을까'라는 질문으로 시작



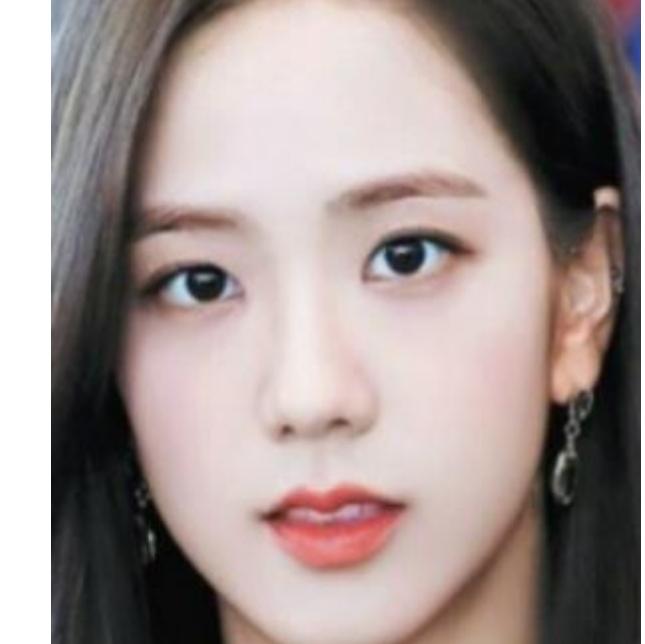
| MBTI 유형별 얼굴 생성 모델

INFP

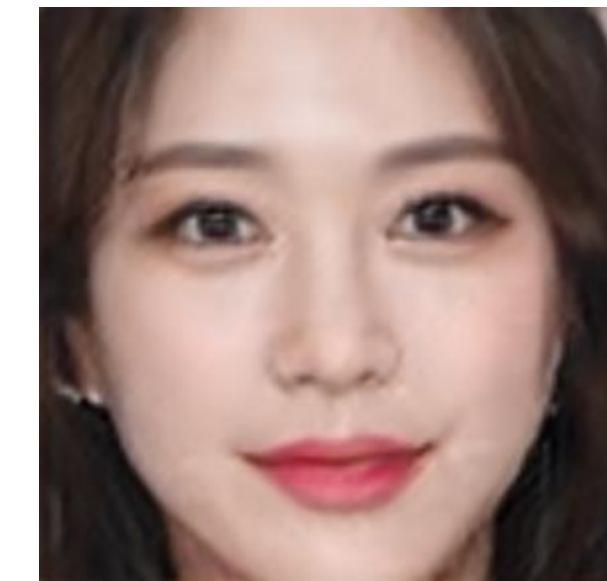
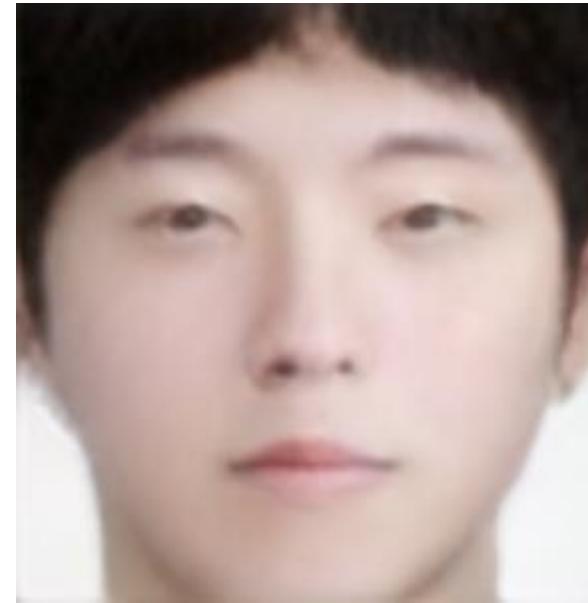
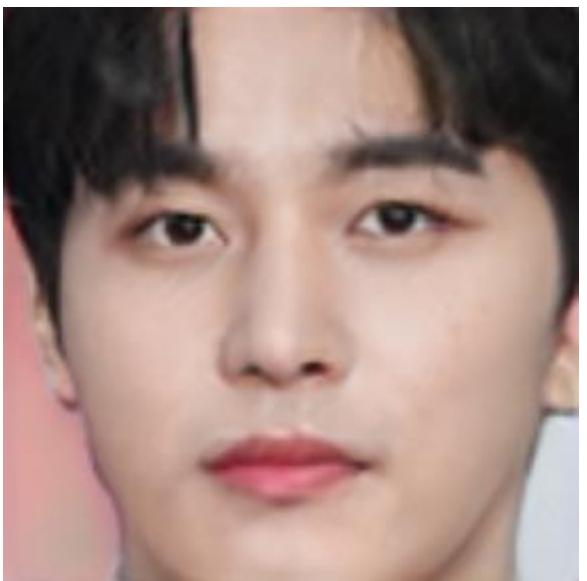
•
•
•



ESTJ



| MBTI 유형별 얼굴 생성 모델



16가지 유형 X 2 (남, 여)의
경우의 수에 따라서 얼굴 제공

내 얼굴을 더한
유형별 얼굴 완성!

내 유형과 최고의 궁합 얼굴 생성

| 데이터 수집 및 가공

데이터 셋 구성

1. 각 MBTI 유형별 연예인 얼굴 사진 수집



2. 약 1000여명의 연예인 얼굴 데이터

: 1명의 연예인 당 5장의 사진 데이터 수집

: 약 4000여장의 데이터 수집

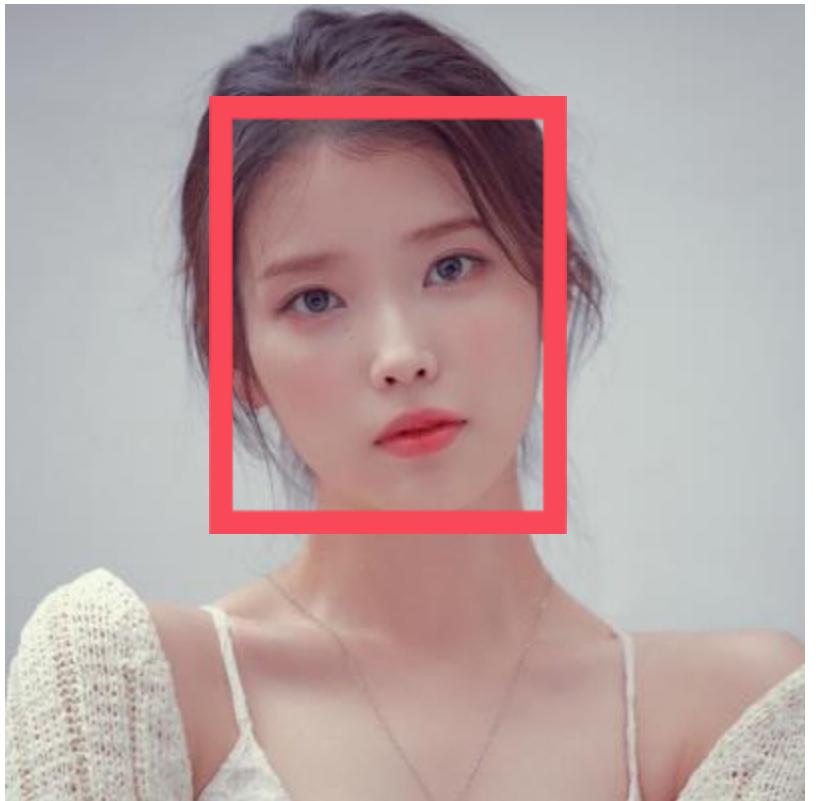


3. 한국 연예인 뿐만 아니라 중국, 일본 등

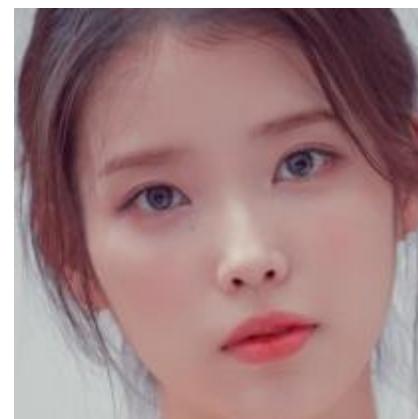
아시아권 연예인 포함해서 수집



| 연예인 데이터 수집 및 가공



원본사진



크로핑



리사이즈

| 개발 툴(tool)



Web(Backend, Frontend)

모델링(Modeling)

이미지 전처리 - 크로핑(Cropping)

```
baseDirectory = os.path.abspath(os.path.dirname(__file__)) # 현재 해당 파일 위치  
  
imageDirectory = f'{baseDirectory}/Image/ESTP/m' # 이미지 폴더 위치  
  
haarCascade = cv2.CascadeClassifier(f'{baseDirectory}/haarcascade_frontalface_alt2.xml') # 검출 대상 : 정면 얼굴 검출  
  
savePath = f'{baseDirectory}/Cropped'  
os.makedirs(savePath, exist_ok = True)  
  
filenameList = os.listdir(imageDirectory)  
  
for filename in filenameList:  
  
    image = numpy.fromfile(f'{imageDirectory}/{filename}', numpy.uint8) # 한글경로 읽기  
    image = cv2.imdecode(image, cv2.IMREAD_UNCHANGED)  
  
    cv2.imshow('Original', image)  
  
    # scaleFactor : 이미지 피라미드 스케일, minNeighbors : 인접 객체 최소 거리 픽셀, minSize : 탐지 객체 최소 크기  
    boundingBoxes = haarCascade.detectMultiScale(image, scaleFactor = 1.1, minNeighbors = 3, minSize = (20,20))  
  
    for boundingBox in boundingBoxes:  
  
        x, y, width, height = boundingBox  
  
        croppedImage = image[y : y + height, x : x + width, : ]  
        # 얼굴 인식 후 크로핑 실행  
  
        cv2.imshow(f'Cropped', croppedImage)  
  
        cv2.imwrite(f'{savePath}/{filename}', croppedImage)
```

=> opencv 라이브러리를 통해
각 이미지별 좌표를 맞춰 크로핑

| 이미지 전처리 - 크로핑(Cropping)

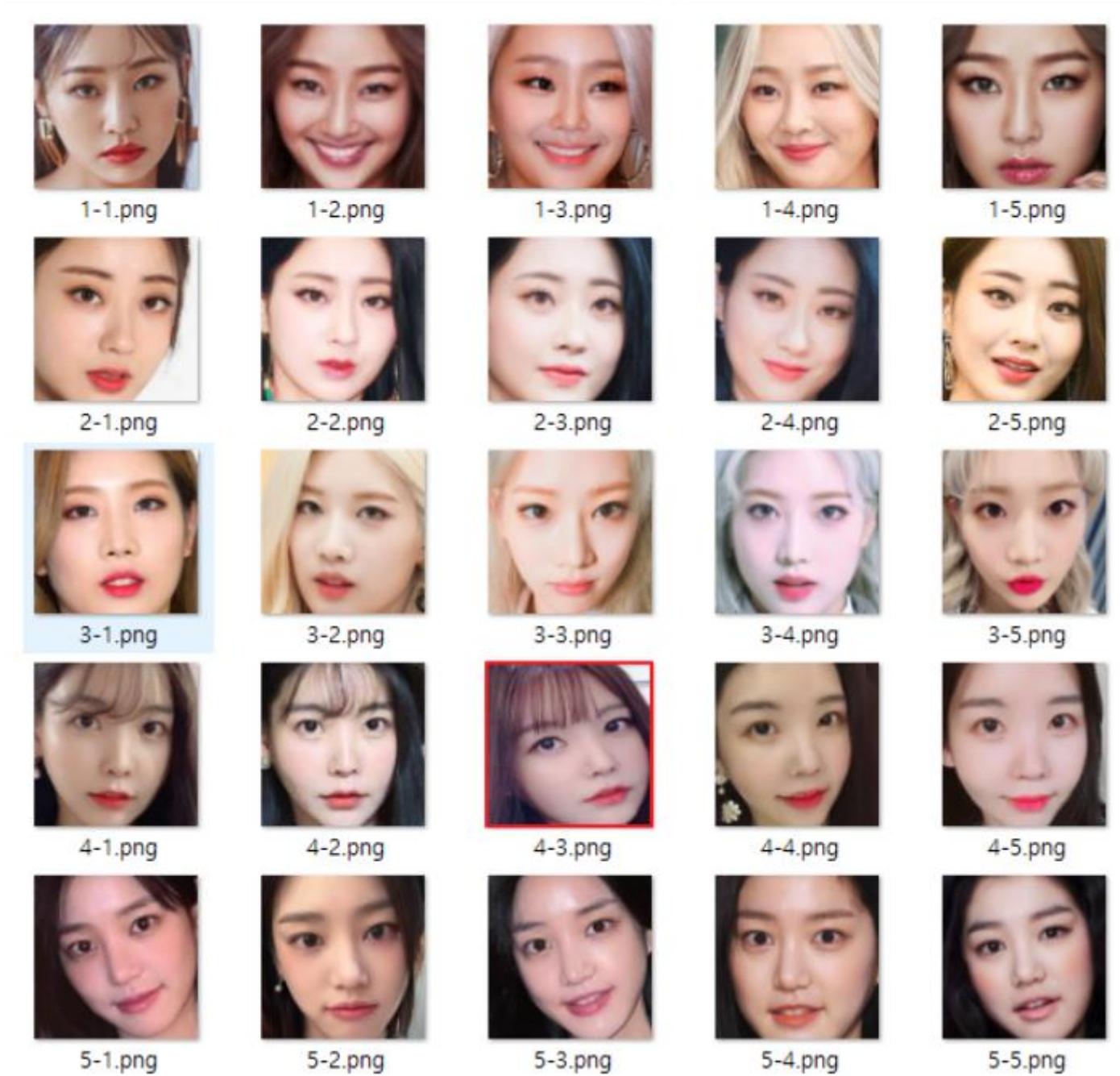
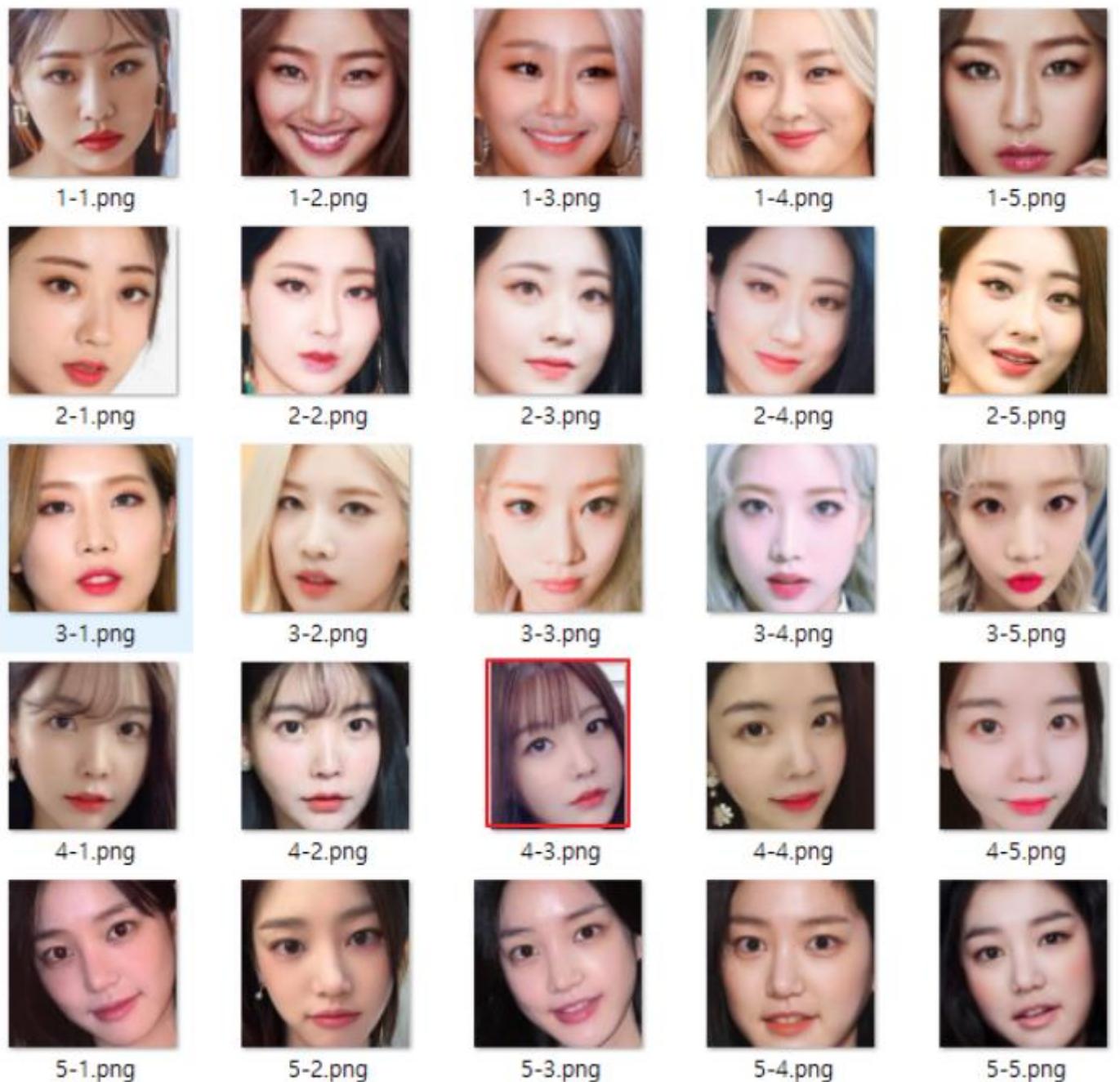


이미지 전처리 – 리사이즈(Resize)

```
from PIL import Image
import os.path
targerdir = r"D:\OPENCV\cropinfjW"
newpath = r"D:\OPENCV\cropinfjW"
files = os.listdir(targerdir)
format = [".jpg", ".png", ".jpeg", "bmp", ".JPG", ".PNG", "JPEG", "BMP"]
for (path, dirs, files) in os.walk(targerdir):
    for file in files:
        if file.endswith(tuple(format)):
            image = Image.open(path+"\\"+file)
            print(image.filename)
            print(image.size)
            image=image.resize((128, 128))
            image.save(newpath+"\\"+file)
            print(image.size)
        else:
            print(path)
            print("InValid",file)
```

=> (128, 128)로 모든 사이즈를
동일하게 맞춤

| 이미지 전처리 – 리사이즈(Resize)



DCGAN

DCGAN

(Deep Convolutional Generative Adversarial Nets)

CNN구조로 판별자 D와 생성자 G를 구성한 GAN

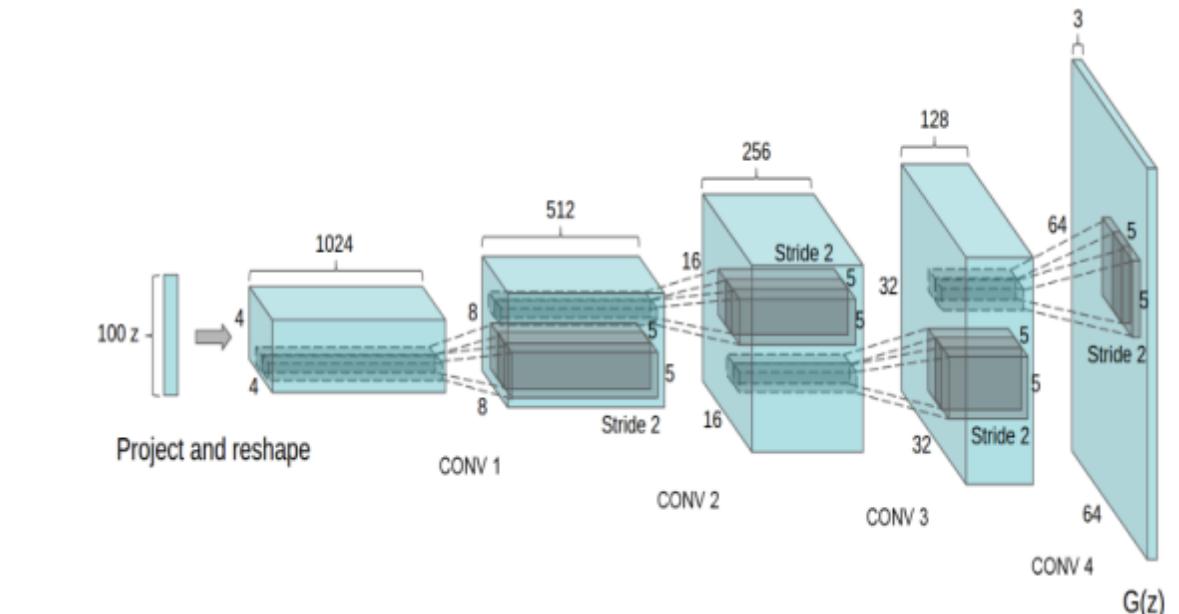
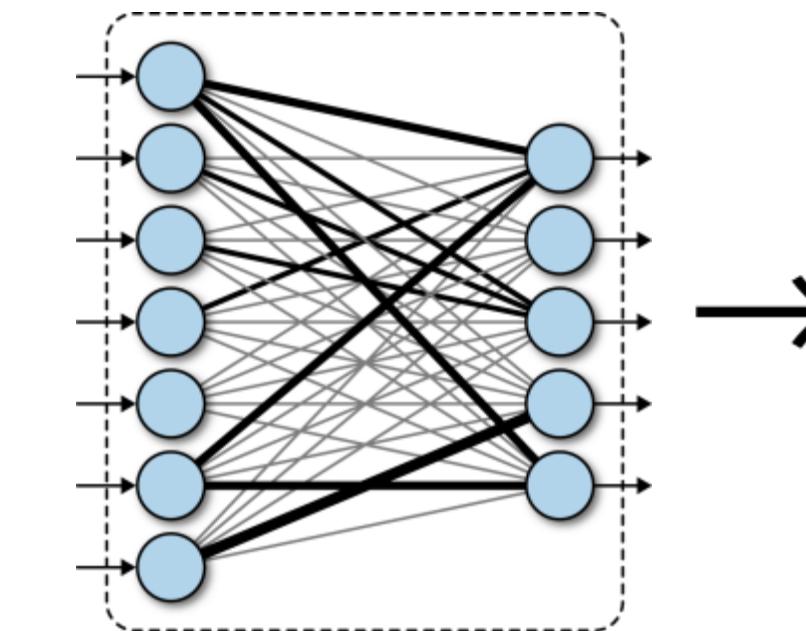
즉, 기존 GAN에 존재했던 fully-connected구조의 대부분을 CNN 구조로 대체

1. 고해상도 이미지를 생성

2. 기존 GAN에 비해 안정적인 학습

3. King-man+woman을 하면 “queen”이 나오는 것

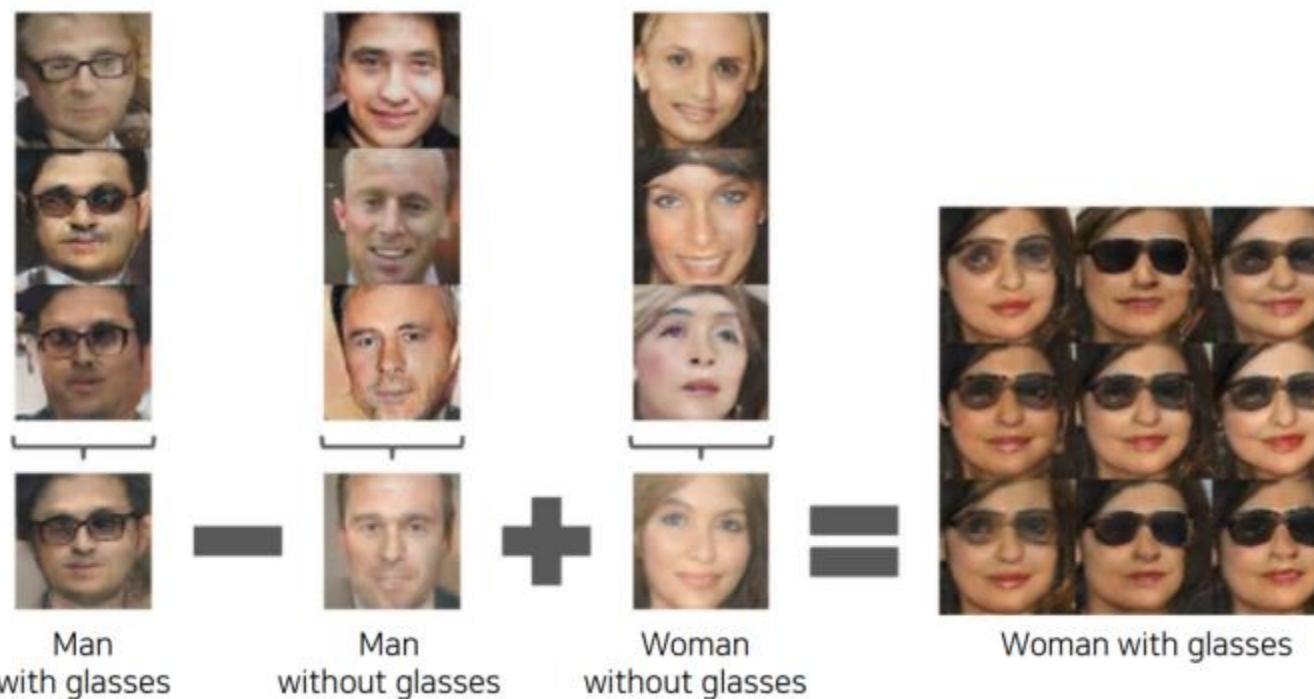
처럼 이미지도 단순암기가 아닌, 벡터 연산이 가능하다는 것을 보여줌



fully connected Layer

특징 : 모든 뉴런들이 서로 거미줄 같이 촘촘하게 연결

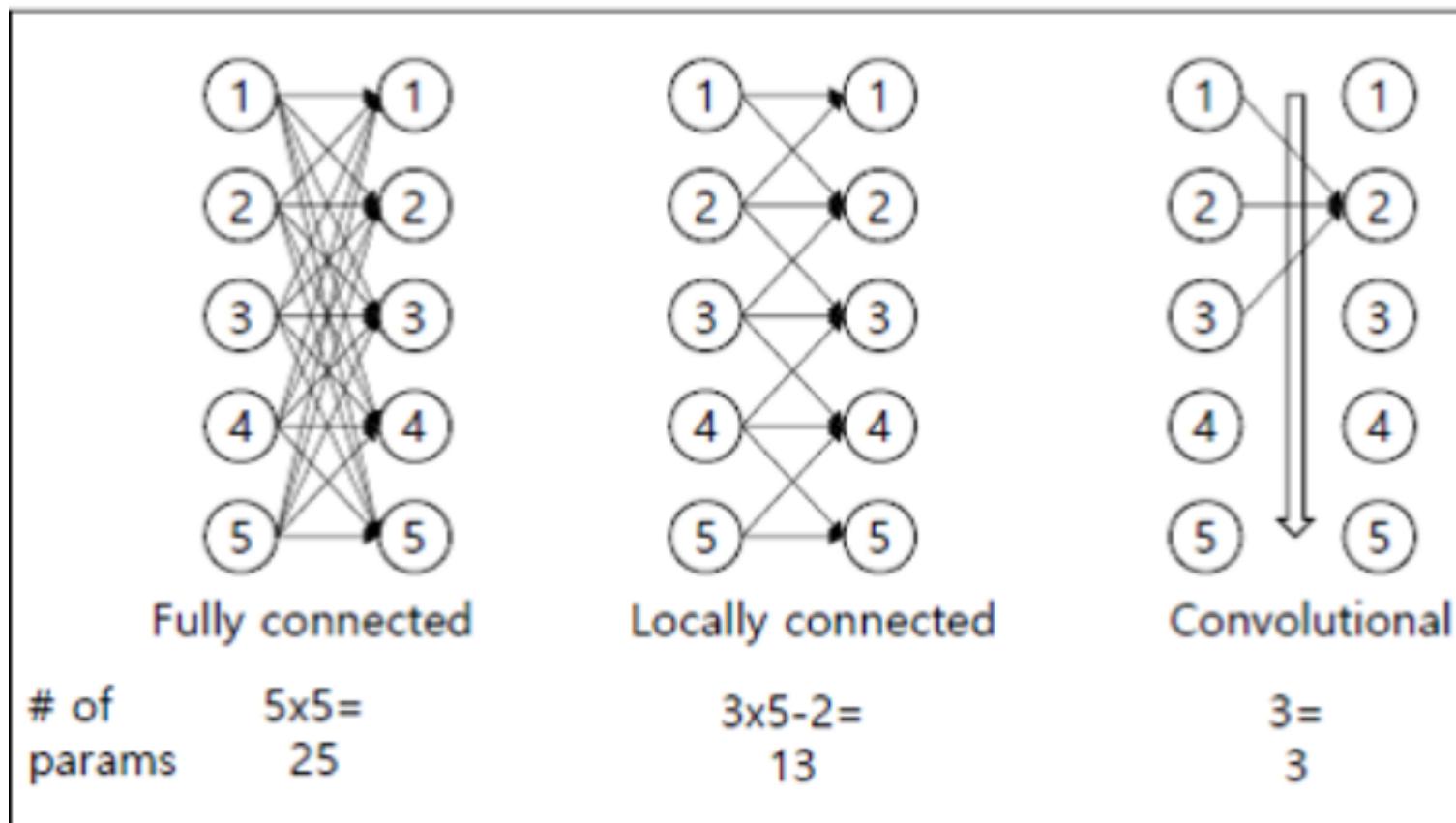
단점 : 매개변수가 너무 많아 연산 복잡해 연산의 양 증가



CNN

CNN은 서로 가깝게 관계있는 패턴만 찾기 때문에 연산의 양이 감소

Fully connected & CNN



What if with 10000 neurons?

Fully connected:

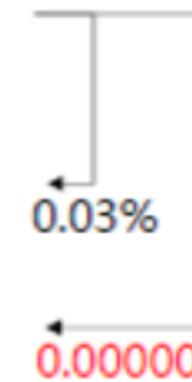
$$10000 \times 10000 = 10e8$$

Locally connected:

$$3 \times 10000 = 3 \times 10e4$$

Convolutional:

$$3$$



DNN : 입력 노드 수 * hidden layer 노드 수

CNN : Feature map= $(\text{input size} + 2 * \text{padding} - \text{filter size}) / \text{stride} + 1$

DCGAN 모델링 – Pytorch

```
des_dir = "../mbti/"

imageSize = 64
batchSize = 64
# batchSize = 64

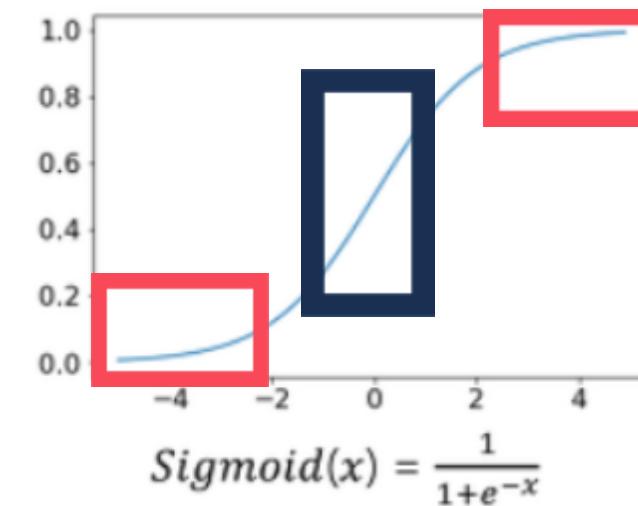
dataset = dset.ImageFolder(root=des_dir,
                           transform=transforms.Compose([ # 전처리 작업
                               transforms.Scale(imageSize), # 이미지 크기 64로 조정
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                               # 이미지의 경우 픽셀 값 하나는 0~255의 값
                               # ToTensor()로 타입 변경시 0 ~ 1 사이의 값으로 바뀜
                               # Normalize -> -1 ~ 1사이의 값으로 normalized 시킴
                           ]))

# DataLoader 를 통해 데이터를 배치사이즈로 나누어준다
dataloader = torch.utils.data.DataLoader(dataset,
                                         batch_size=batchSize,
                                         shuffle=True)

nz      = 100      # dimension of noise vector
nc      = 3        # number of channel - RGB (채널 수 )
ngf     = 64       # generator 필터 조정
ndf     = 64       # discriminator 필터 조정
niter   = 1300     # epoch
lr      = 0.0001   # learning rate
beta1   = 0.5      # hyper parameter of Adam optimizer
ngpu    = 1        # number of using GPU
```

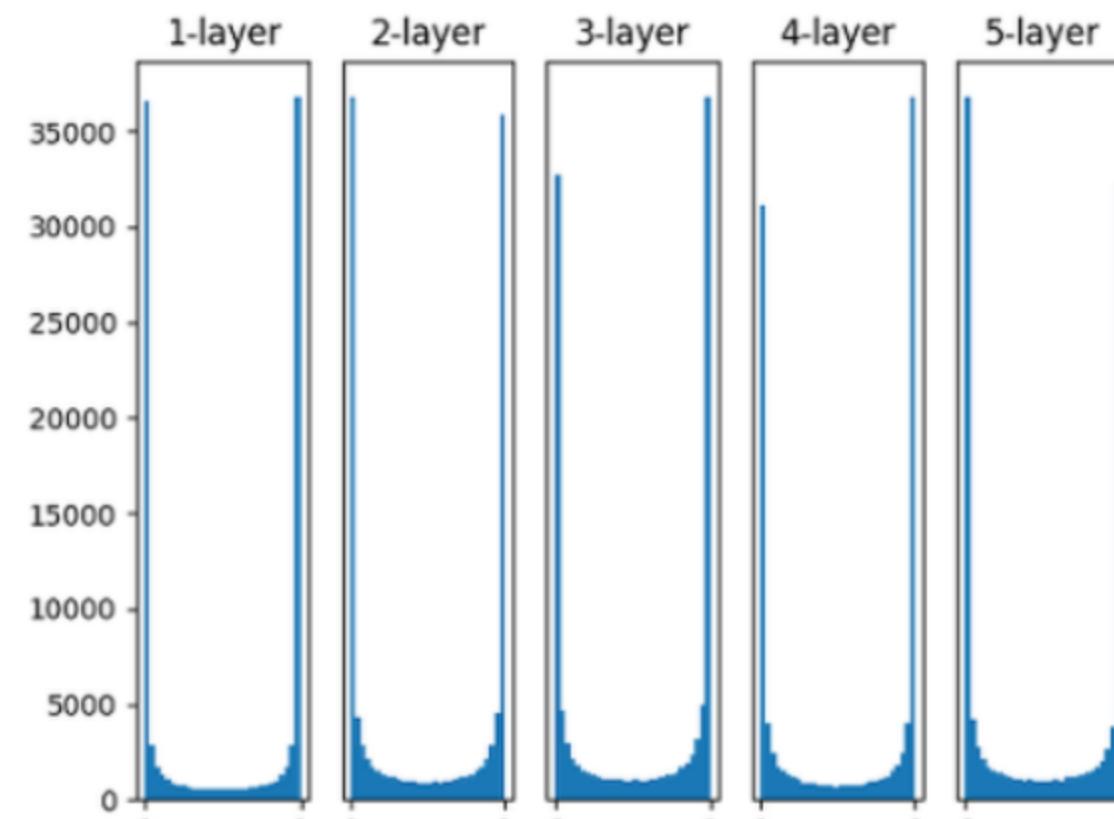
DCGAN 모델링 - 가중치 초기화

```
def weights_init(m): # 모든 모델의 weight는 랜덤하게 초기화 되어야 한다
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:          # Conv weight init
        m.weight.data.normal_(0.0, 0.02) # 평균 0, 표준편차 0.02
    elif classname.find('BatchNorm') != -1: # BatchNorm weight init
        m.weight.data.normal_(1.0, 0.02) # 평균 1.0, 표준편차 0.02
        m.bias.data.fill_(0)
```

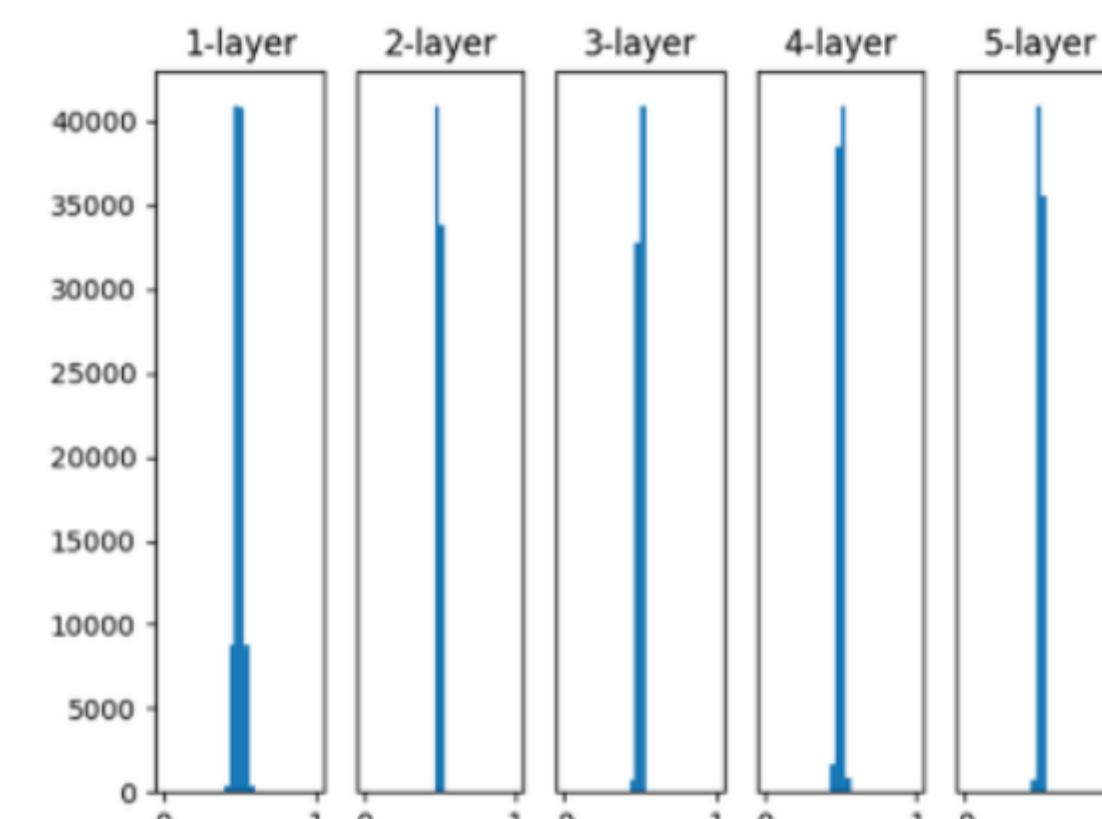


기울기 소실 문제

역전파 과정에서 전파 시킬 기울기가 소실되면 입력 층 방향으로 갈 수록 제대로 역전파가 되지 않는 기울기 소실 문제 발생



표준 편차 : 1, 가중치 0과 1



표준 편차 : 0.02

DCGAN 모델링 – Generator 코드

```
def __init__(self, ngpu):
    super(_netG, self).__init__() # nn.Module.__init__() 를 실행
    self.ngpu = ngpu
    self.main = nn.Sequential(
        # input is Z, going into a convolution
        nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False),
        # ConvTranspose2d(a, b, c, d, e): a는 input 채널의 수, b는 만들어지는 결과값의 채널의
        # c는 커널의 크기, 즉 Convolution 연산을 수행하는 필터의 크기
        # d는 stride, e는 padding

        nn.BatchNorm2d(ngf * 8),
        nn.ReLU(True),

        # state size. (ngf*8) x 4 x 4
        nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
        nn.BatchNorm2d(ngf * 4),
        nn.ReLU(True),

        # state size. (ngf*4) x 8 x 8
        nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
        nn.BatchNorm2d(ngf * 2),
        nn.ReLU(True),

        # state size. (ngf*2) x 16 x 16
        nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False),
        nn.BatchNorm2d(ngf),
        nn.ReLU(True),

        # state size. (ngf) x 32 x 32
        nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
        nn.Tanh()
```

<Pytorch.ver>

```
# Generator
model = Sequential()
model.add(Convolution2DTranspose(2048, kernel_size=(3,3), input_dim=100))
model.add(BatchNorm2d())
model.add(Activation('ReLU'))

model.add(Convolution2DTranspose(128*7*7))
model.add(BatchNorm2d())
model.add(Activation('ReLU'))

model.add(Convolution2DTranspose[(7, 7, 128), input_shape=(128*7*7,)])
model.add(BatchNorm2d(size=(2, 2)))
model.add(Activation('ReLU'))

model.add(Convolution2DTranspose(64, (5, 5), padding='same'))
model.add(Activation('tanh'))
model.summary()
```

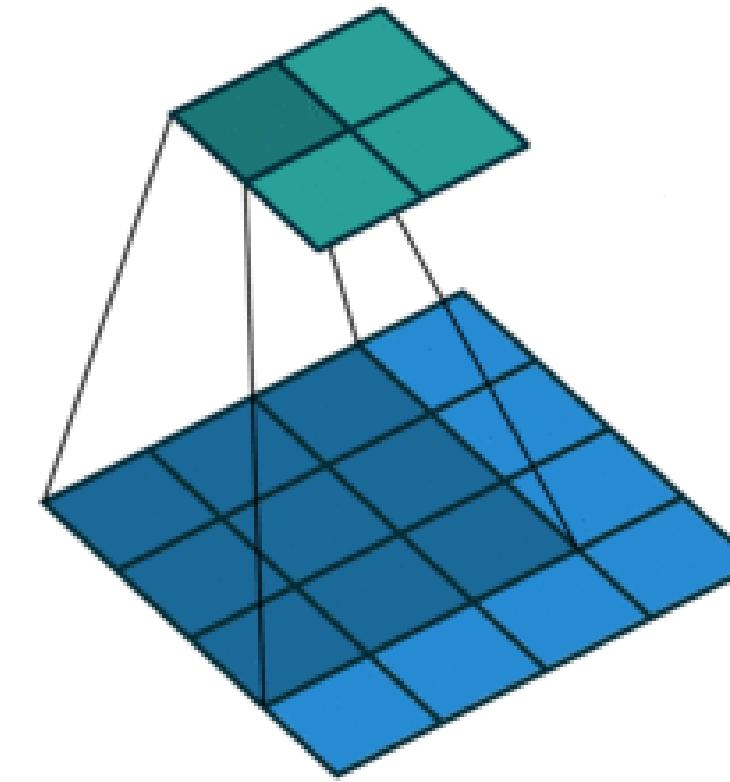
<Tensorflow.ver>

```
self.main = nn.Sequential(
    # input is Z, going into a convolution
    # nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False),
    nn.Upsample(scale_factor = 2), # nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False)
    nn.Conv2d(nz, ngf * 8, 2, 1, 1, bias = False), # nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False)
    nn.BatchNorm2d(ngf * 8),
    nn.ReLU(True),

    # state size. (ngf*8) x 4 x 4
    # nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
    nn.Upsample(scale_factor = 2), # nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
    nn.Conv2d(ngf * 8, ngf * 4, 2, 1, 1, bias = False), # nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf * 4),
    nn.ReLU(True),
```

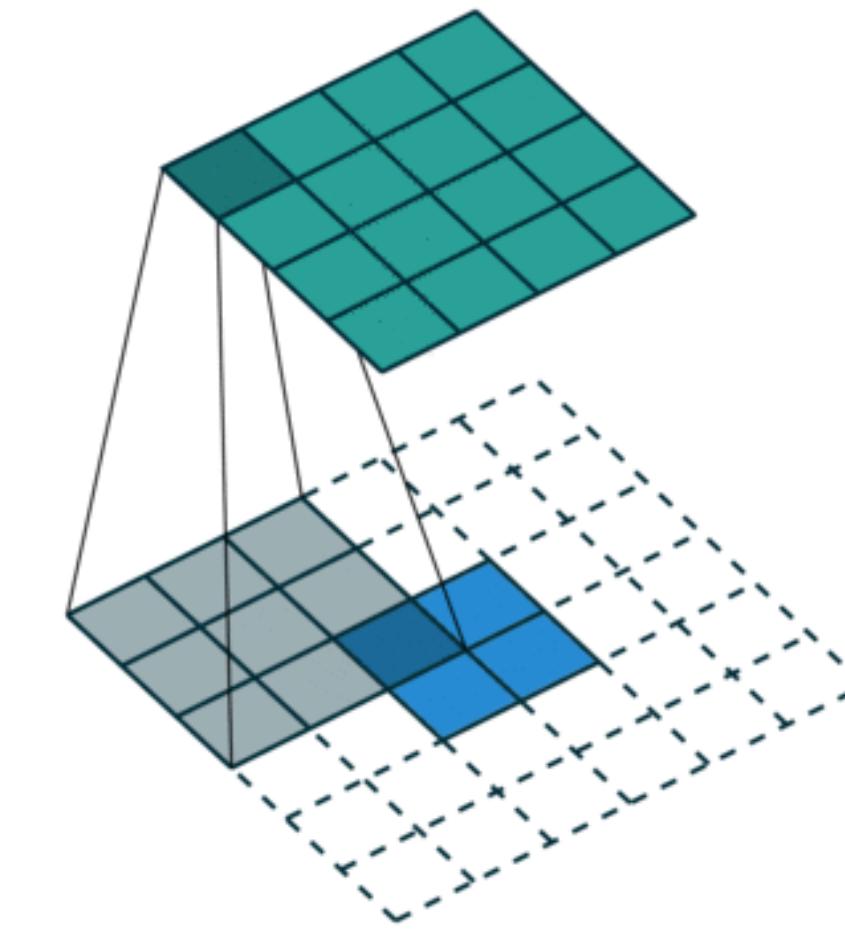
<conv2d + upsample_Pytorch.ver>

DCGAN 모델링 – conv2d와 convTransposed2d



Convolution layer[Downsampling]

: Convolution 연산은 Filter의 크기와 Stride에 따라서 이미지의 크기를 줄인다



Transposed layer[Upsampling]

: Transposed Convolution은 Convolution layer와 반대로 이미지의 크기를 키우는 역할을 수행한다.

DCGAN 모델링 – Discriminator 코드

```
class _netD(nn.Module): # Discriminator
    def __init__(self, ngpu):
        super(_netD, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False), # 이미지 채널
            nn.LeakyReLU(0.2, inplace=True),

            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),

            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),

            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),

            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid() # 확률 구하기

            # state size. 1
        )

        def forward(self, input):
            if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
                output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
            else:
                output = self.main(input)

            return output.view(-1, 1).squeeze(1)
```

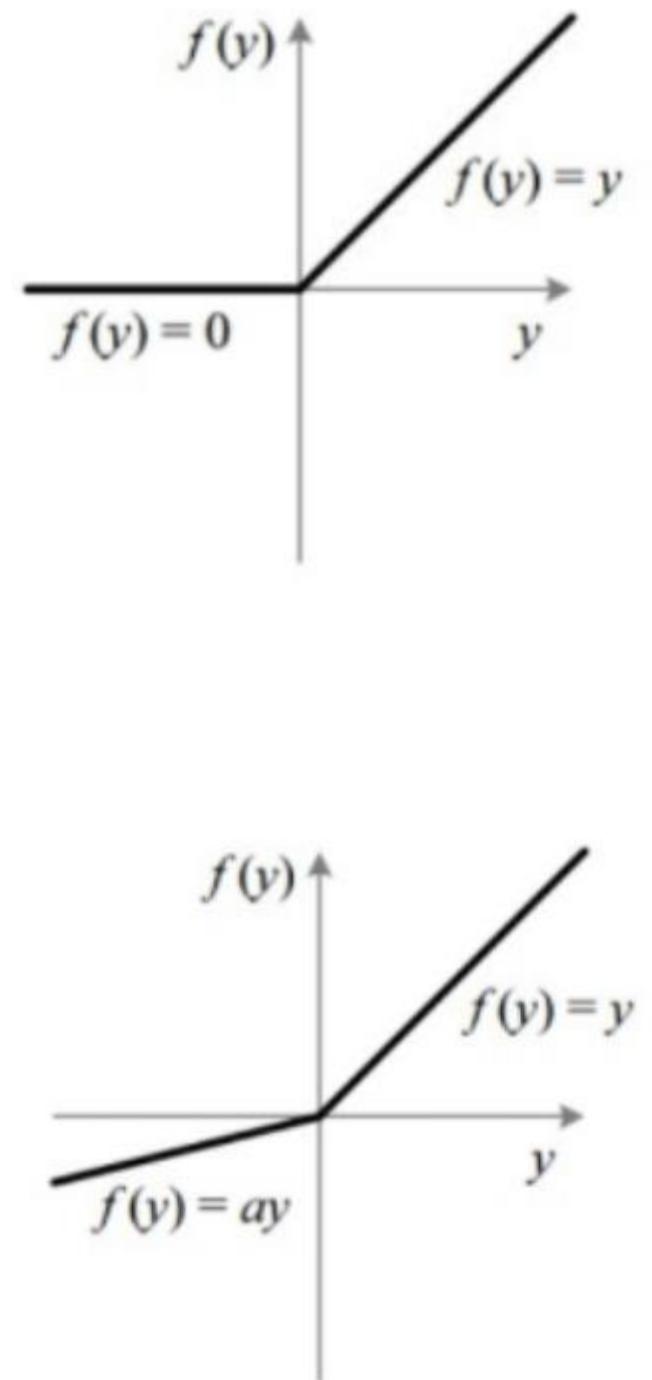
```
# Discriminator
def discriminator_model():
    model = Sequential()
    model.add(
        Conv2D(64, (5, 5),
               padding='same',
               input_shape=(28, 28, 1))
    )
    model.add(Activation('LeakyReLu'))

    model.add(Conv2D(128, (5, 5)))
    model.add(BatchNorm2d())
    model.add(Activation('LeakyReLu'))

    model.add(Conv2D(512, (5, 5)))
    model.add(BatchNorm2d())
    model.add(Activation('LeakyReLu'))

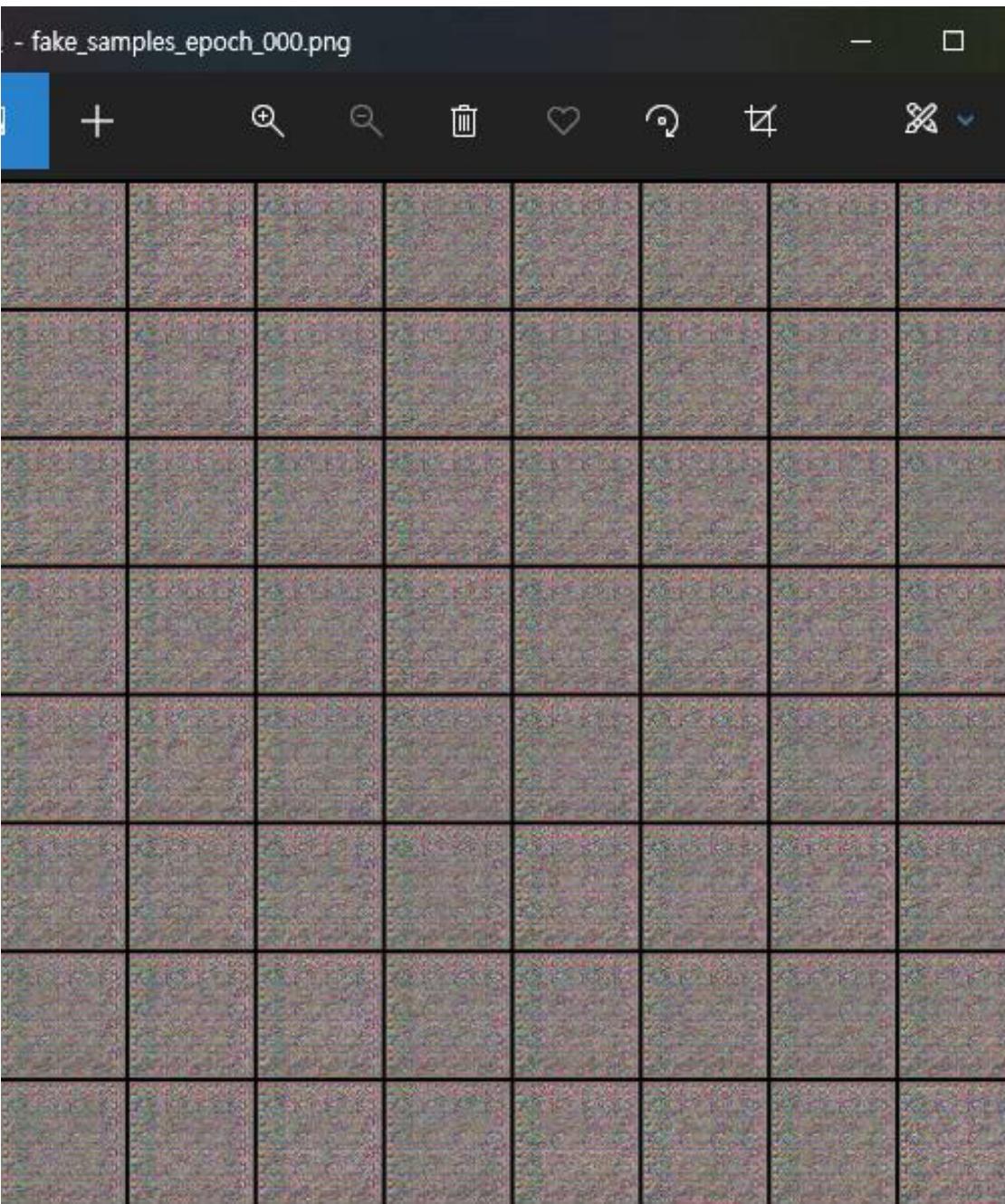
    model.add(Conv2D(1024, (5, 5)))
    model.add(BatchNorm2d())
    model.add(Activation('LeakyReLu'))

    model.add(Conv2D(2048, (5, 5)))
    model.add(Activation('sigmoid'))
    return model
```



<텐서플로우 code>

DCGAN 모델링 – 결과



1) epoch를 500으로 설정한 뒤 얼굴의 변화

: epoch 100 내외에서 전제적인 얼굴윤곽이 잡하고 200까지 발전하는 모습

2) 연산속도의 이점

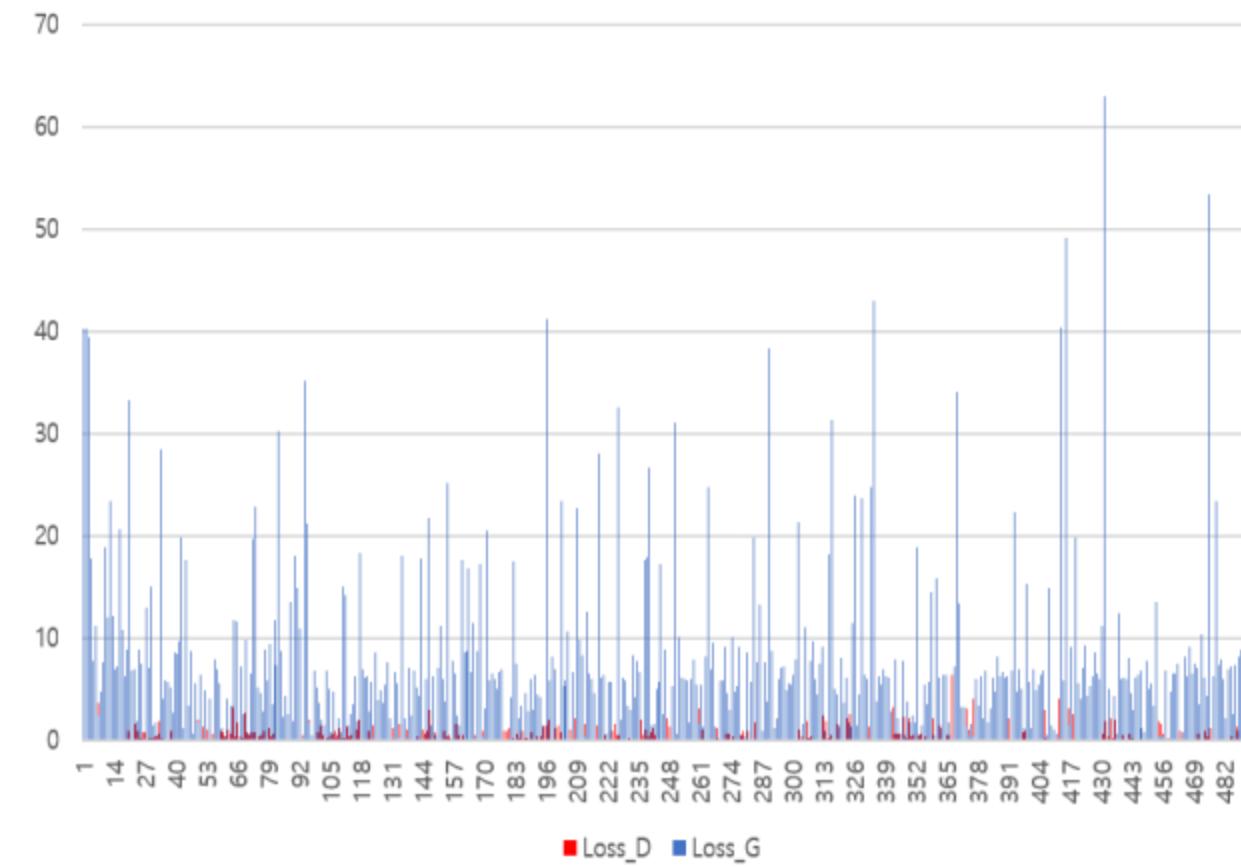
: 2014년 발표된 vanilla GAN만큼 빠른 속도로 연산 가능

3) 해상도의 한계

: 해상도가 나아졌다는 것은 vanilla GAN에 비해서 나아졌다는 것, 이후 발표된 StyleGAN 이후에 비해서는 여전히 낮은 해상도(input 해상도 이상의 화질 개선 불가)

DCGAN 모델링 - Loss (Binary Cross Entropy, BCE)

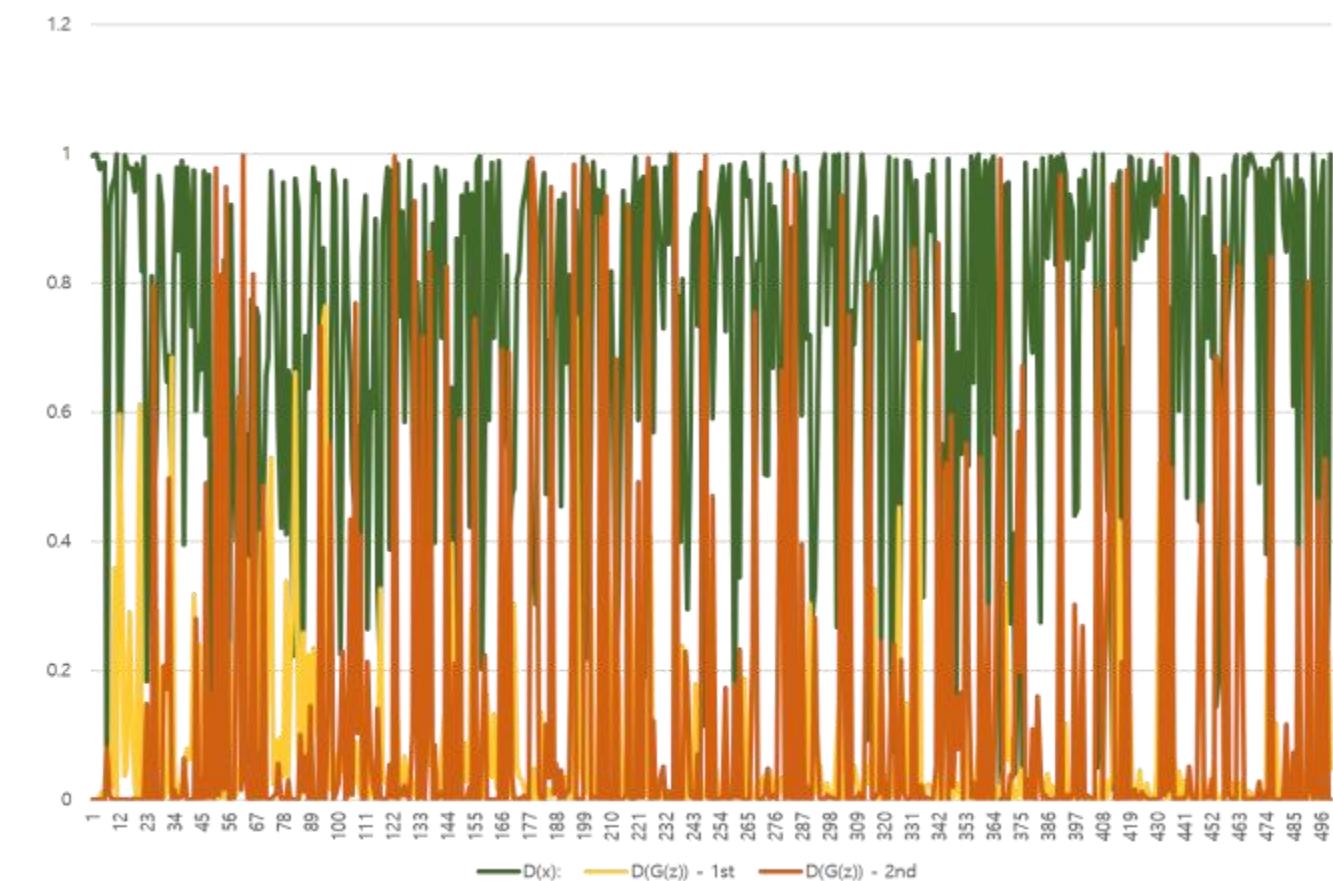
Loss of Discriminator and Generator



epochs가 진행함에 따라 Loss_G 값이 수렴하는 현상은
보이지 않음

Loss_D는 epochs와 상관없이 0에 가까운 값을 보였다.

Mean Prediction of Discriminator



D(x)는 실제 이미지에 대한 확률값, 대체로 1 근처에 위치

:D(G(z))는 가짜 이미지를 넣었을 때 나온 결과

discriminator가 가짜 이미지를 진짜 이미지를 잘 구별하고 있다는 것을 보여줌

GAN에서 Loss가 보조적 지표인 이유

I realized that often the losses go up or down almost randomly, and there is nothing wrong with that. I achieved some great and realistic results while the generator loss was way higher than the discriminator loss, which is perfectly normal. Thus, when you encounter a sudden instability in your training process, I recommend leaving the training going for a bit more, keeping an eye on the quality of the generated images during training, as a visual understanding is often more meaningful than some loss numbers.

저자 : Marco Pasini(Data Scientist)

출처: Jul 29, 2019 towards
data science

(<https://towardsdatascience.com/10-lessons-i-learned-training-generative-adversarial-networks-gans-for-a-year-c9071159628>)

→ (요약 및 해석)

GAN의 loss 값은 실제 결과물에 상관없이 높아지거나 낮아지는 경향이 있으므로 loss 보다는 실제 결과로 나오는 이미지의 품질을 주시하면서 훈련을 진행하는 것을 권한다.

→ DCGAN 이후의 모델의 훈련은 시각적인 변화가 충분히 이루어지는 부분까지 훈련을 하는 것으로 결정

| StyleGAN & StyleGAN2 with ada 비교

StyleGAN

(Style-based Generative Adversarial Networks)

PG GAN의 고화질 이미지처리를 개선

1. 저화질에서 고화질로 레이어를 쌓아 고화질 이미지 생성

2. 머리, 연령, 성별등을 제어할 수 있음

3. 머리카락의 위치, 주름, 얼굴 이미지를 사실적이고, 다양하게 만듦

StyleGAN2 with ada

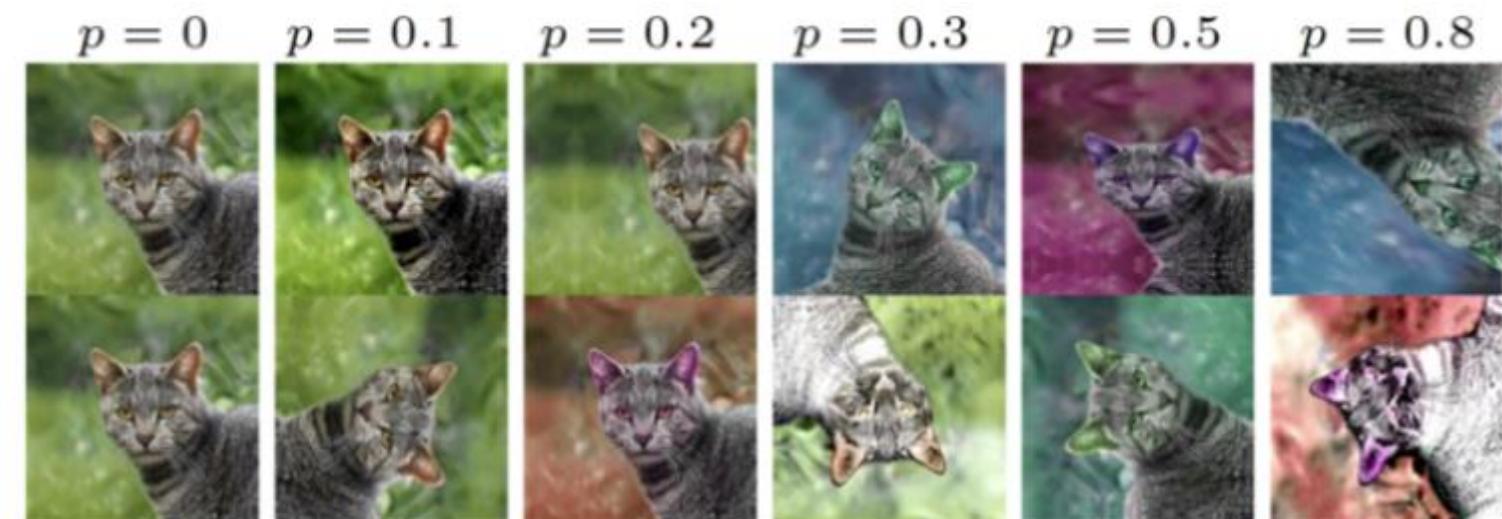
(Style-based Generative Adversarial Networks2 with ada)

adaptive discriminator augmentation(ADA)라는 데이터 증강 기법을 적용해 많은 데이터를 필요로 하는 StyleGAN2의 문제점 개선

1. StyleGAN의 물방울 노이즈 개선

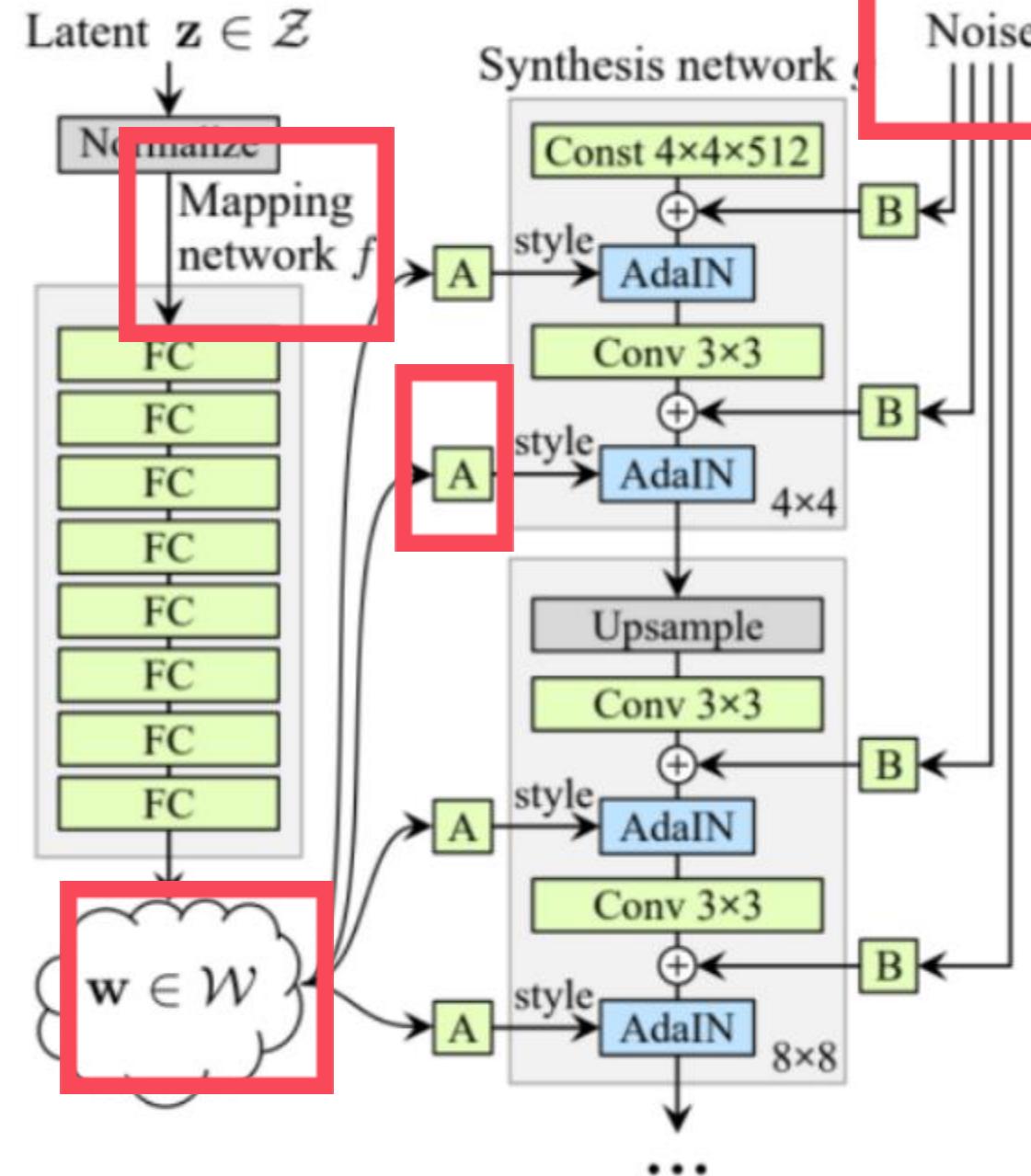
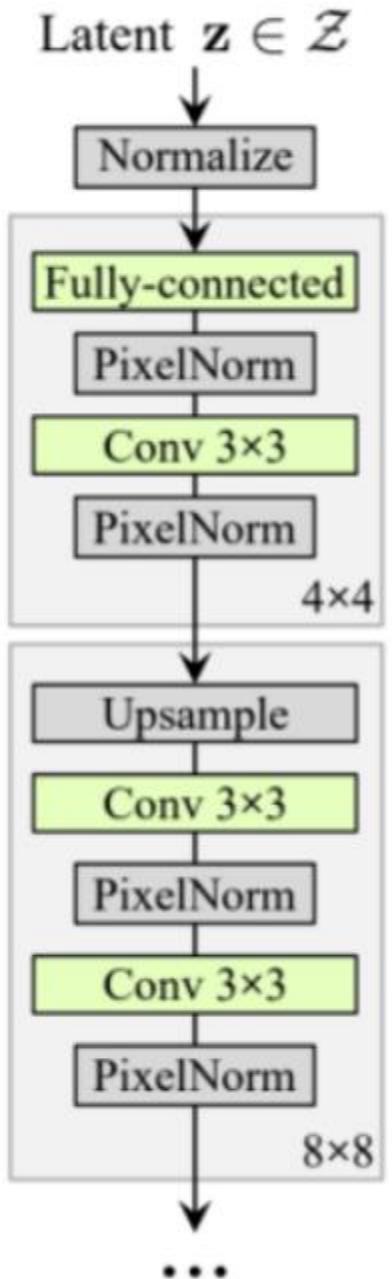
2. Label smoothing을 개선하여 이미지를 매끄럽게 생성

3. 이미지를 다양한 형태로 증강해주어 데이터 부족 현상을 어느 정도 극복 및 과적합(Overfitting) 개선



StyleGAN & StyleGAN2 with ada

StyleGAN



w vector:

스타일(얼굴형, 포즈, 안경의 유무), 노이즈(주근깨, 피부 모공)
조절 용이

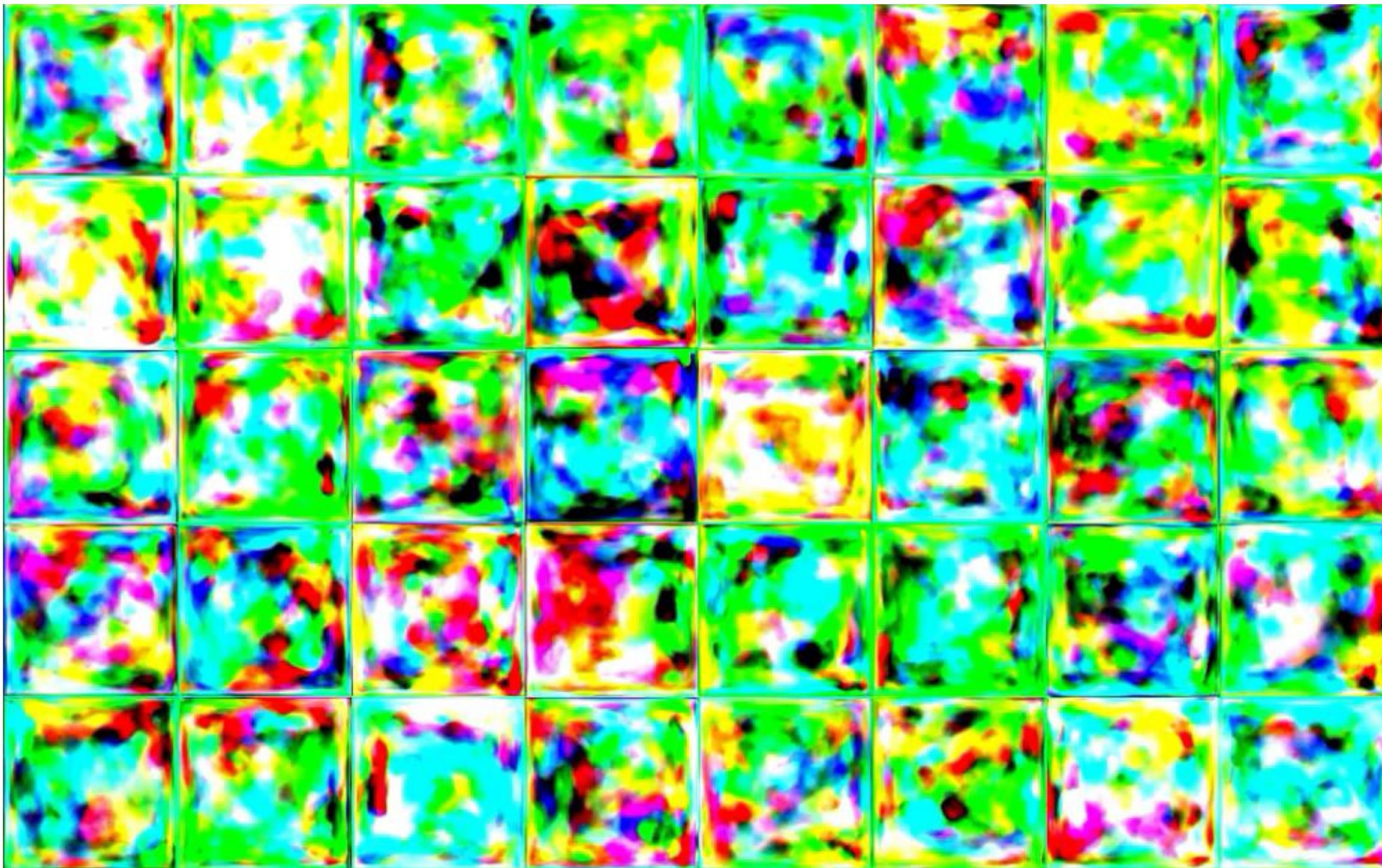
Affine transformation:

AdaIN에 들어가는 style 정보가 될 수 있도록 변환

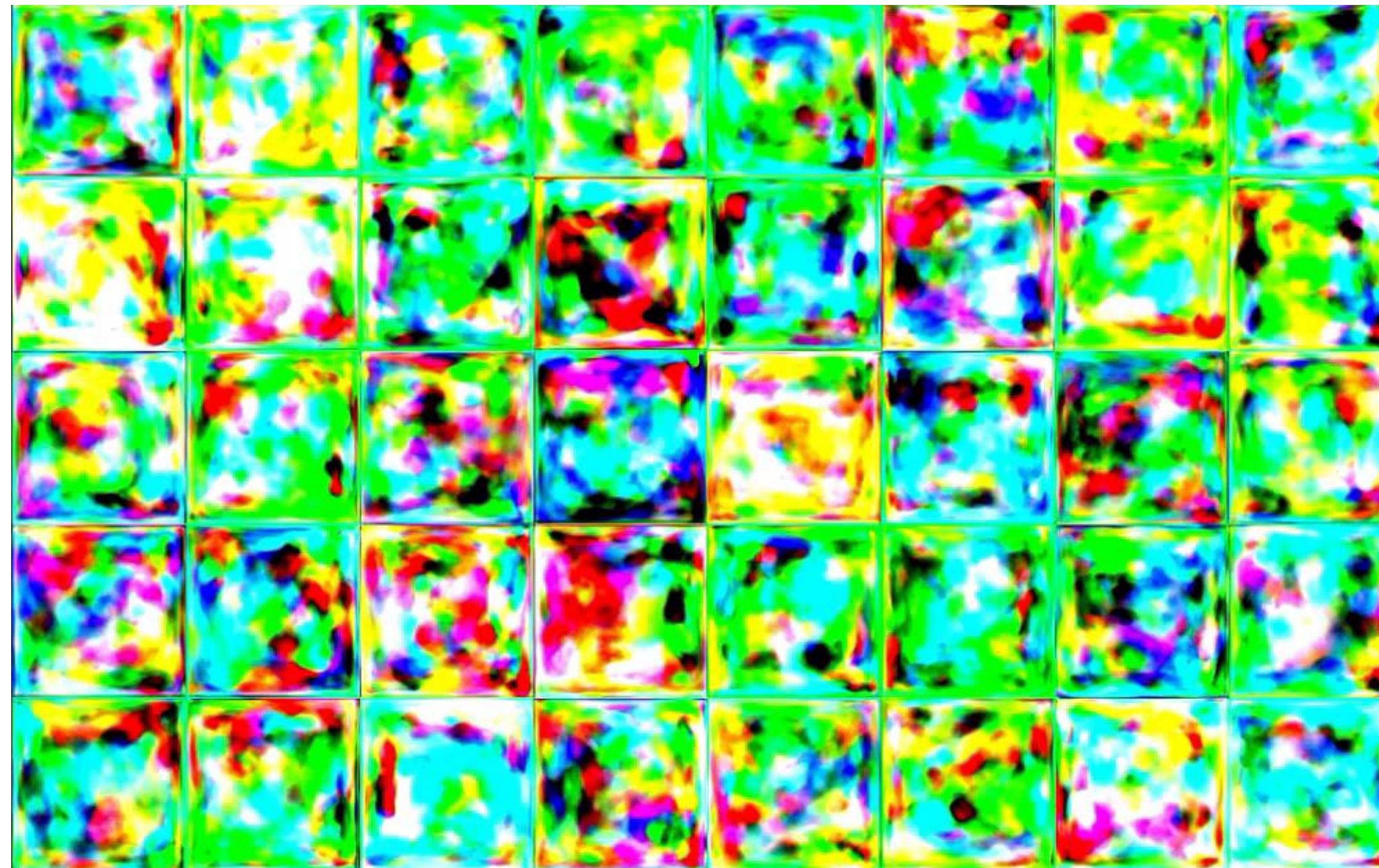
AdaIN:

Conv 연산 수행 결과를 처리, Style 정보를 포함

| StyleGAN 모델링 - 결과 (약 5000번 훈련, 시간 : 23h, RTX 2080)



| StyleGAN 모델링 - 결과 (약 5000번 훈련, 시간 : 36h, RTX 2080)



| StyleGAN 모델링 – 결과

1) epoch를 5000으로 설정한 뒤 얼굴의 변화

: 흐린 얼굴 윤곽이 3000을 기점으로 확연하게 잡히고 그 이후에는 스타일의 변화가 일어난다. 하지만 그 차이가 크지는 않다.

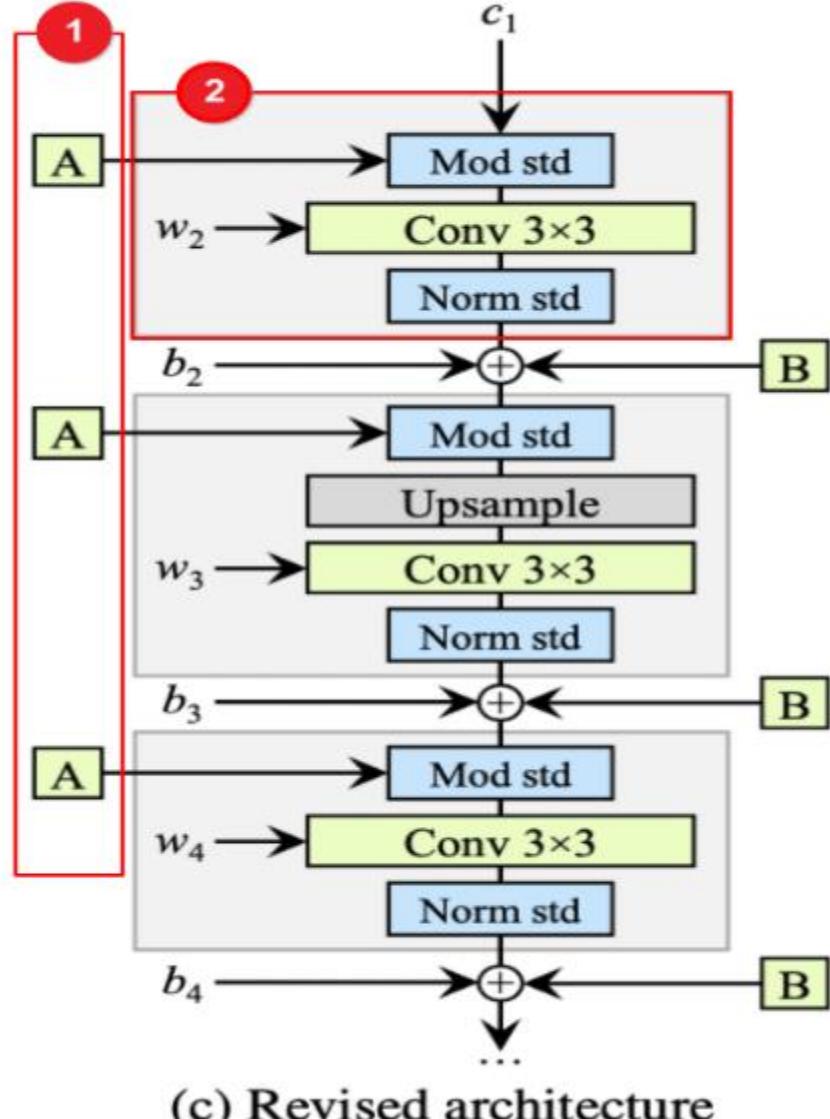
2) 확연하게 길어진 훈련시간

: 훈련량의 비해 훈련시간이 길어진 것은 선형적으로 늘어나는 게 아니라 지수함수의 형태로 증가함을 알 수 있다.

3) DCGAN에 비해 고화질의 이미지 생성 가능

: PGGAN에서 도입한 ‘Progressive Growing’ 기법으로 본격적인 고화질의 이미지 생성이 용이해짐

StyleGAN2 with ada모델링 - 코드

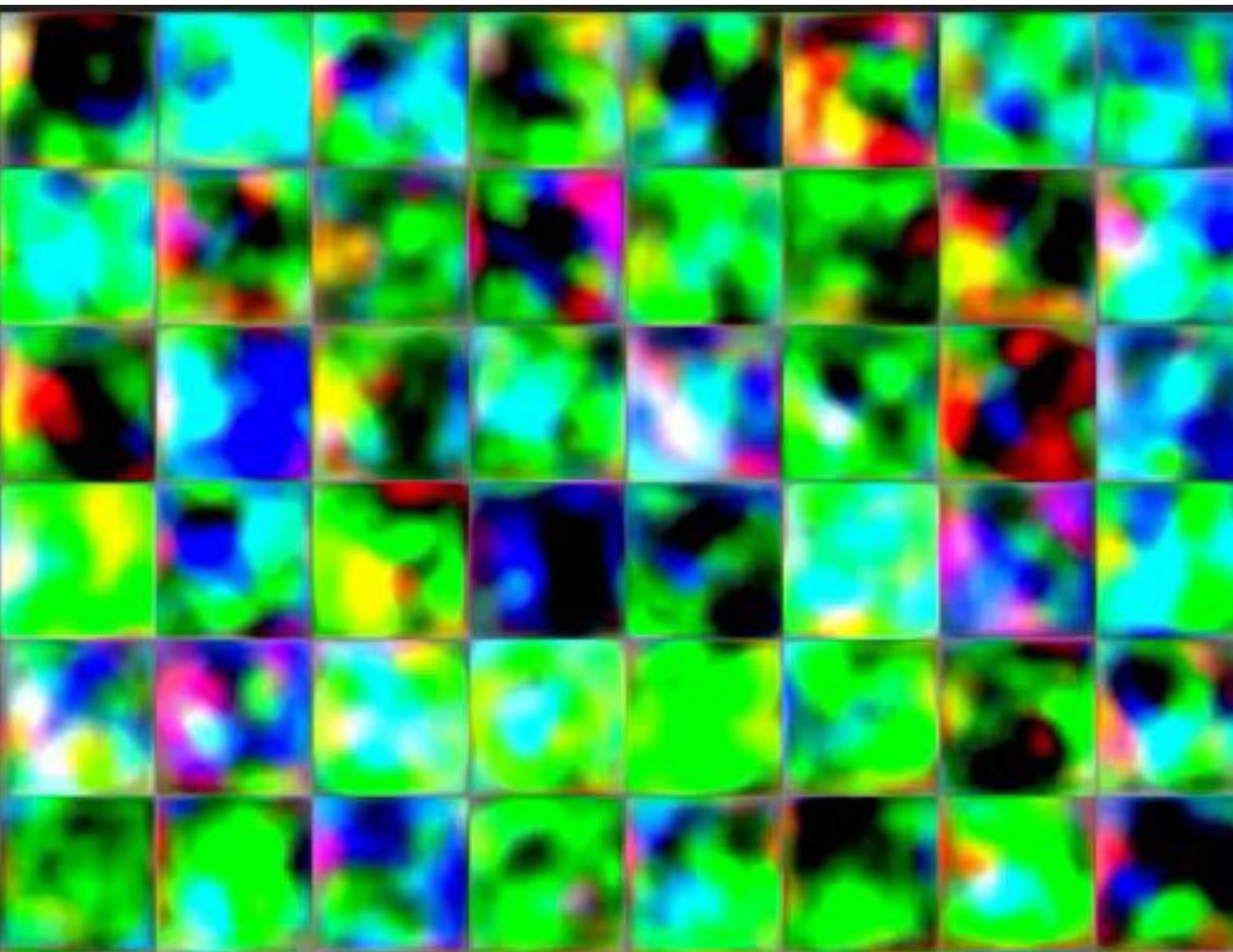


```
@persistence.persistent_class
class Generator(torch.nn.Module):
    def __init__(self,
                 z_dim,                         # Input latent (Z) dimensionality.
                 c_dim,                         # Conditioning label (C) dimensionality.
                 w_dim,                          # Intermediate latent (W) dimensionality.
                 img_resolution,                # Output resolution.
                 img_channels,                  # Number of output color channels.
                 mapping_kwargs = {},           # Arguments for MappingNetwork.
                 synthesis_kwargs = {}):       # Arguments for SynthesisNetwork.

        super().__init__()
        self.z_dim = z_dim
        self.c_dim = c_dim
        self.w_dim = w_dim
        self.img_resolution = img_resolution
        self.img_channels = img_channels
        self.synthesis = SynthesisNetwork(w_dim=w_dim, img_resolution=img_resolution, img_channels=img_channels, **synthesis_kwargs)
        self.num_ws = self.synthesis.num_ws
        self.mapping = MappingNetwork(z_dim=z_dim, c_dim=c_dim, w_dim=w_dim, num_ws=self.num_ws, **mapping_kwargs)

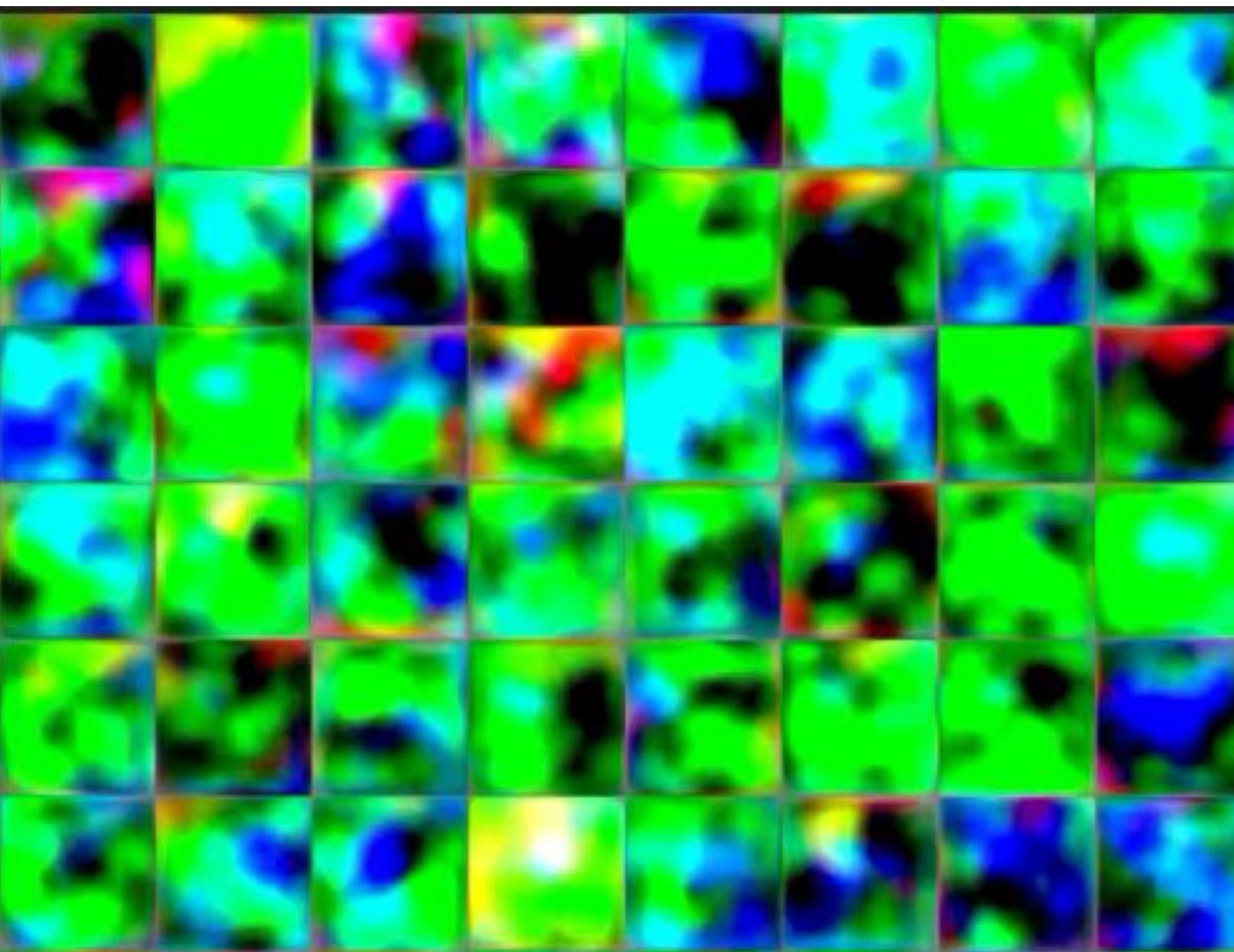
    def forward(self, z, c, truncation_psi=1, truncation_cutoff=None, **synthesis_kwargs):
        ws = self.mapping(z, c, truncation_psi=truncation_psi, truncation_cutoff=truncation_cutoff)
        img = self.synthesis(ws, **synthesis_kwargs)
        return img
```

| StyleGAN2 with ada모델링 - 결과



남

| StyleGAN2 with ada모델링 - 결과



여

| StyleGAN2 with ada모델링 – 결과

1) 구동에 많은 리소스를 요구

:최소 구동 VRAM 12G를 요구하여 V100 24G 모델 2대와 RTX 3090 1대에서만 구동가능

2) 긴 훈련시간

:RTX 3090 기준 25000 epoch를 훈련하는데 ‘4일 반’의 긴 훈련시간이 필요함

3) 증강하는 과정을 알 수 있다

: epoch가 늘어나면서 원본 이미지에서 90도, 180도 회전하고, 전체적인 색감이 R,G,B 순으로 변화함을 확인할 수 있다. 이 과정을 통해 적은 데이터로도 증강하여 훈련할 수 있다.

4) parameter 튜닝의 어려움

: 결과물을 보면 인물의 얼굴과 얼굴을 제외한 부분의 조합이 계속 바뀌는 것을 알 수 있으나 이를 임의로 선택해서 할 수 있는지의 여부에 대해서는 아직 연구가 필요

FreezeD: Freeze the Discriminator

Freeze-D Mo et al. [33] propose to freeze the first k layers of the discriminator to improve results with transfer learning. We tested several different choices for k ; the best results were given by $k = 10$ in Figure 9 and by $k = 13$ in Figure 11b. In practice, this corresponds to freezing all layers operating at the 3 or 4 highest resolutions, respectively.



1. Stylegan을 기반으로 생성자를 얼린 모델
2. 판별자 레이어 일부층의 가중치를 반영하지 않음
3. 판별자의 네트워크 중 특징 추출기를 고정
4. 15k 기준으로 학습이 진행될수록 이미지가 깨짐
5. 사전 지식 활용 -> 빠른 학습

```
convs.2.conv1.0.weight False
convs.2.conv1.1.bias False
convs.2.conv2.1.weight False
convs.2.conv2.2.bias False
convs.2.skip.1.weight False
convs.3.conv1.0.weight False
convs.3.conv1.1.bias False
convs.3.conv2.1.weight False
convs.3.conv2.2.bias False
convs.3.skip.1.weight False
convs.4.conv1.0.weight False
convs.4.conv1.1.bias False
convs.4.conv2.1.weight False
convs.4.conv2.2.bias False
convs.4.skip.1.weight False
convs.5.conv1.0.weight True
convs.5.conv1.1.bias True
convs.5.conv2.1.weight True
convs.5.conv2.2.bias True
convs.5.skip.1.weight True
convs.6.conv1.0.weight True
convs.6.conv1.1.bias True
convs.6.conv2.1.weight True
convs.6.conv2.2.bias True
convs.6.skip.1.weight True
final_conv.0.weight True
final_conv.1.bias True
final_linear.0.weight True
final_linear.0.bias True
final_linear.1.weight True
final_linear.1.bias True
```

FreezeD_ Code

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--size', type=int, default=128, help='size of the image')
    parser.add_argument('--n_row', type=int, default=1, help=
    'number of rows of sample matrix')
    parser.add_argument('--n_col', type=int, default=1, help=
    'number of columns of sample matrix')
    parser.add_argument('path', type=str, help='path to checkpoint file')

    args = parser.parse_args()

    device = 'cuda'
    generator = StyledGenerator(512).to(device)
    generator.load_state_dict(torch.load(args.path)['g_running'])
    generator.eval()
    mean_style = get_mean_style(generator, device)
    step = int(math.log(args.size, 2)) - 2

    img = sample(generator, step, mean_style, args.n_row * args.n_col, device)
    # utils.save_image(img, 'sample.png', nrow=args.n_col, normalize=True, range=(-1, 1))
    # utils.save_image(img, 'sample.png', nrow=args.n_col, normalize=True, range=(-1, 1))

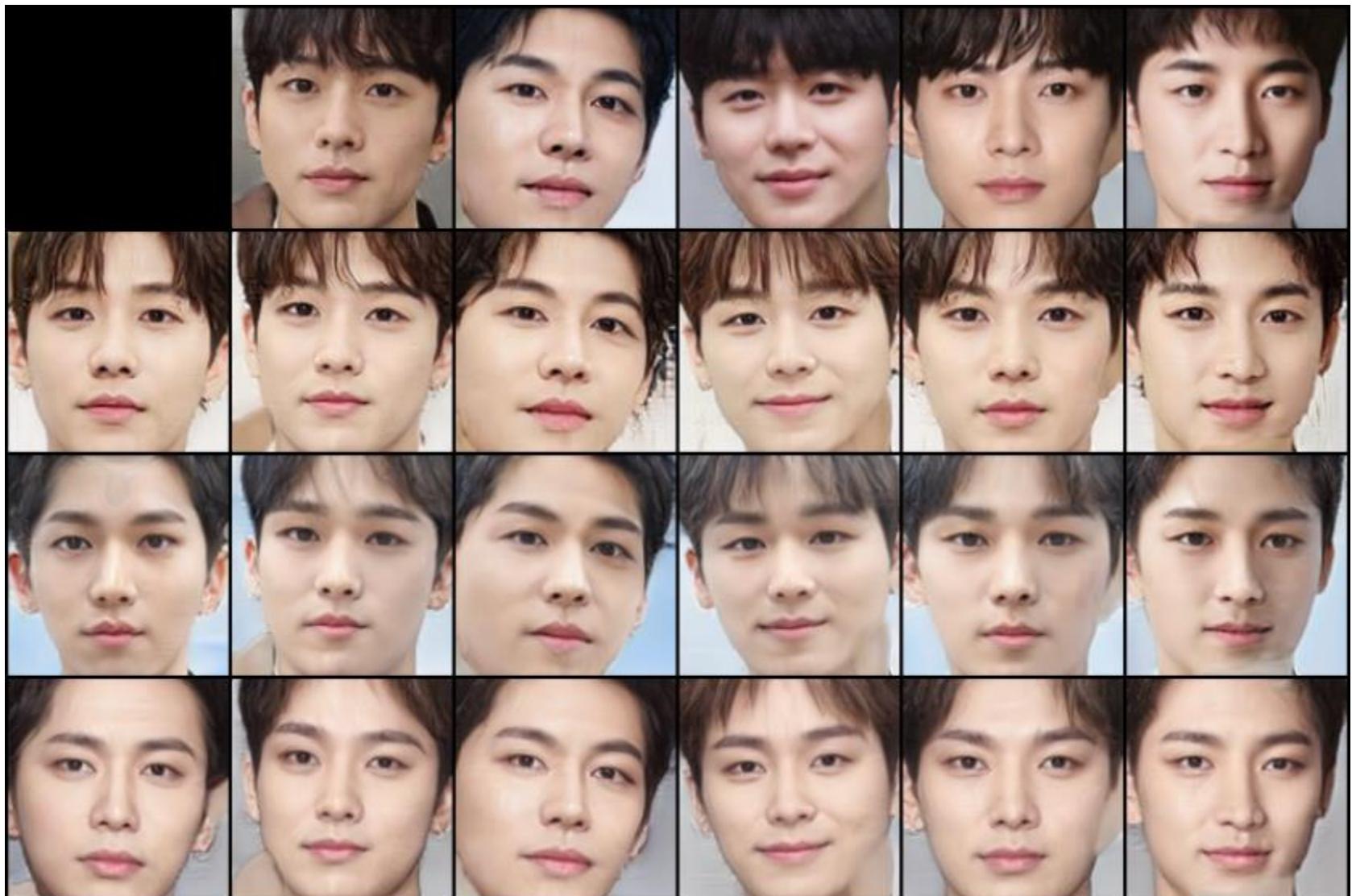
    for j in range(20):
        img = style_mixing(generator, step, mean_style, args.n_col, args.n_row, device)
        utils.save_image(
            img, f'sample_mixing_{j}.png', nrow=args.n_col + 1, normalize=True, range
            =(-1, 1)
        )
```

- 각각 이미지 픽셀 크기, 가로/세로 grid, pre-trained model을 선택할 수 있다.

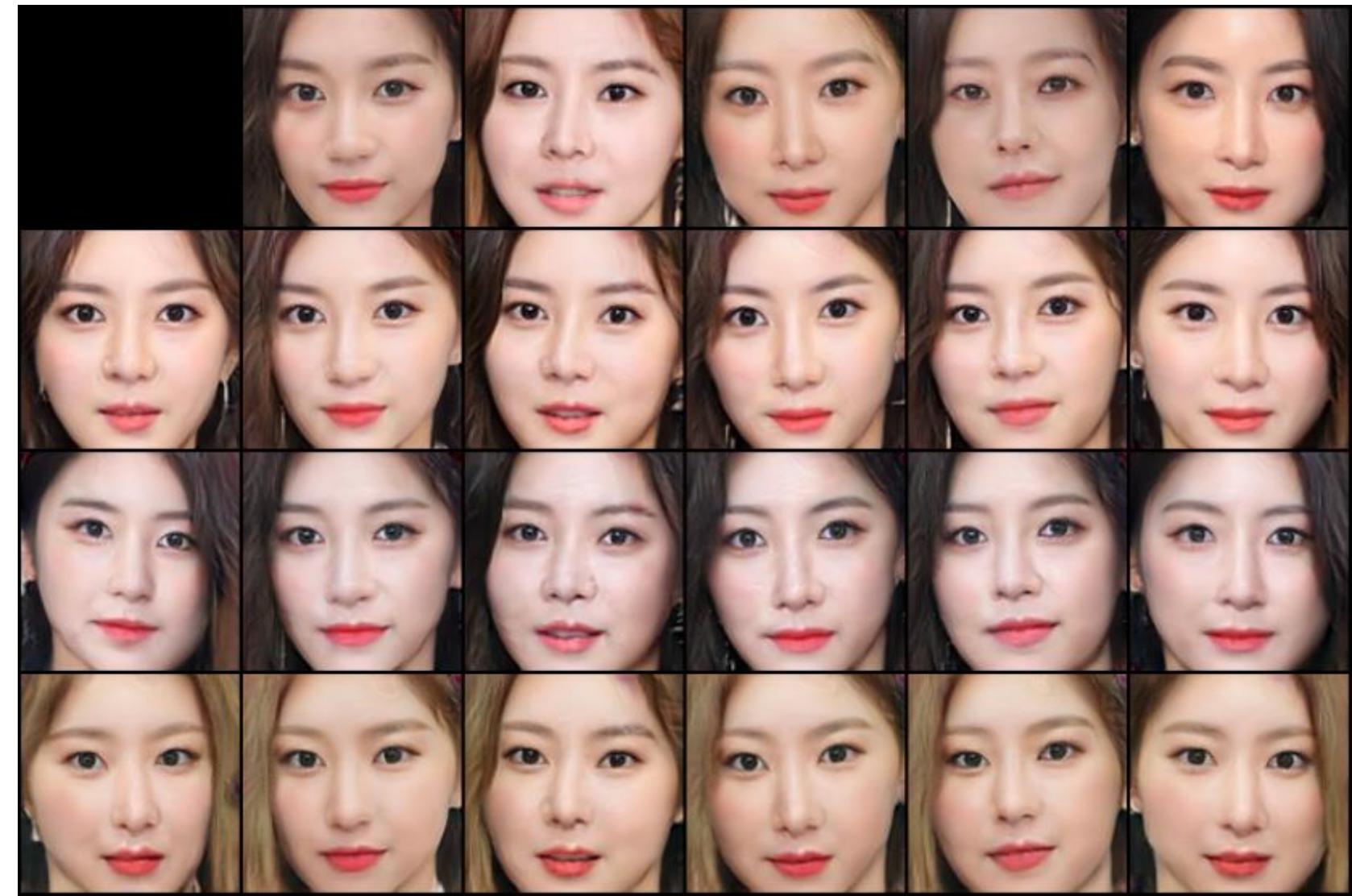
[이 때 finetune.py를 실행하여 생성한 best.model을 사용하여 stylemixing된 결과물을 생성하고자 한다]

- 결과적으로 sample과 stylemixing된 결과물을 save하여 확인할 수 있다.

| FreezeD_ Results



남



여

| 유형별 평균 얼굴 _ 방법1. FreezeD

Stylegan2ada train.py에서 남녀로 각각 훈련한 값의 pkl파일 생성

남녀 mbti 16가지 유형을 lmdb로 변환한 후 FreezeD finetune.py를 통해 best.model 생성

best.model과 average numpy 값으로 freezeD generate.py를 통해 평균 얼굴 값 도출

남녀 mbti 16가지 유형을 Stylegan2 ada projector.py와 위의 pkl파일로 32가지의 total numpy 값 생성 →

total numpy 값을 average numpy 값으로 변환

| 유형별 평균 얼굴 - 방법2. truncation_psi=0

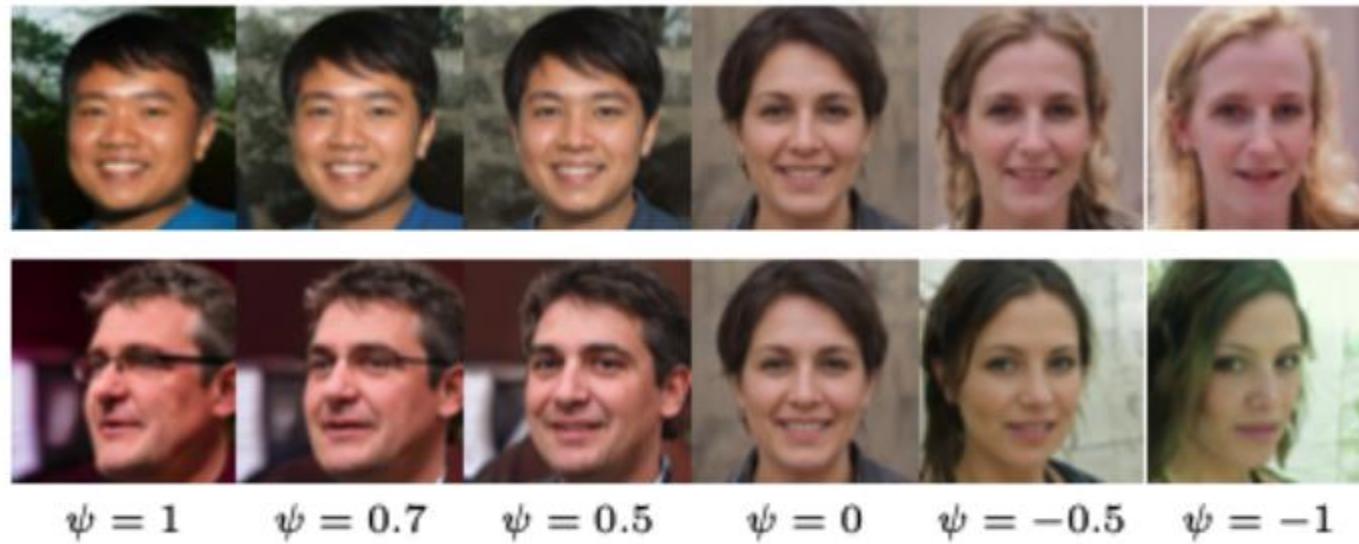


Figure 8. The effect of truncation trick as a function of style scale ψ . When we fade $\psi \rightarrow 0$, all faces converge to the “mean” face of FFHQ. This face is similar for all trained networks, and the interpolation towards it never seems to cause artifacts. By applying

출처: A Style-Based Generator Architecture

for Generative Adversarial Networks)

$$\bar{w} = \mathbb{E}_{z \sim P(z)} [f(z)]$$

$$w' = \bar{w} + \psi (w - \bar{w})$$

```
@persistence.persistent_class
class Generator(torch.nn.Module):
    def __init__(self,
                 z_dim,                                     # Input latent (Z) dimensionality.
                 c_dim,                                     # Conditioning label (C) dimensionality.
                 w_dim,                                     # Intermediate latent (W) dimensionality.
                 img_resolution,                            # Output resolution.
                 img_channels,                             # Number of output color channels.
                 mapping_kwargs     = {},                  # Arguments for MappingNetwork.
                 synthesis_kwargs   = {},                  # Arguments for SynthesisNetwork.
                 ):
        super().__init__()
        self.z_dim = z_dim
        self.c_dim = c_dim
        self.w_dim = w_dim
        self.img_resolution = img_resolution
        self.img_channels = img_channels
        self.synthesis = SynthesisNetwork(w_dim=w_dim, img_resolution=img_resolution,
                                         img_channels=img_channels, **synthesis_kwargs)
        self.num_ws = self.synthesis.num_ws
        self.mapping = MappingNetwork(z_dim=z_dim, c_dim=c_dim, w_dim=w_dim, num_ws=self.
                                      num_ws, **mapping_kwargs)

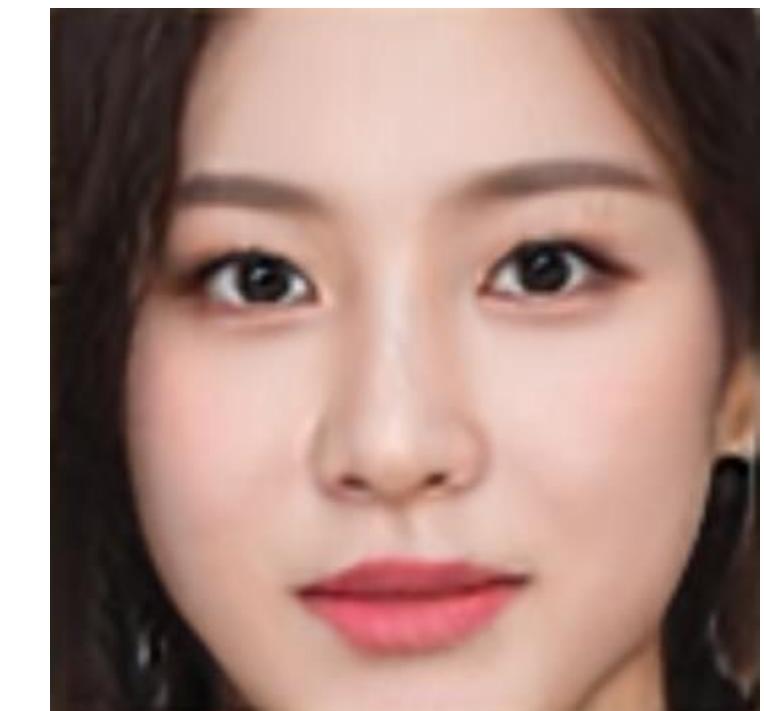
    def forward(self, z, c, truncation_psi=0, truncation_cutoff=None, **synthesis_kwargs):
        ws = self.mapping(z, c, truncation_psi=truncation_psi, truncation_cutoff=
                          truncation_cutoff)
        img = self.synthesis(ws, **synthesis_kwargs)
        return img
```

| 유형별 평균 얼굴 - 방법2. truncation_psi=0



→ 반복적으로 실행했을 때에도 input data와 다른 새로운 얼굴이
비슷하게 생성되는 것을 확인

| 유형별 평균얼굴 비교



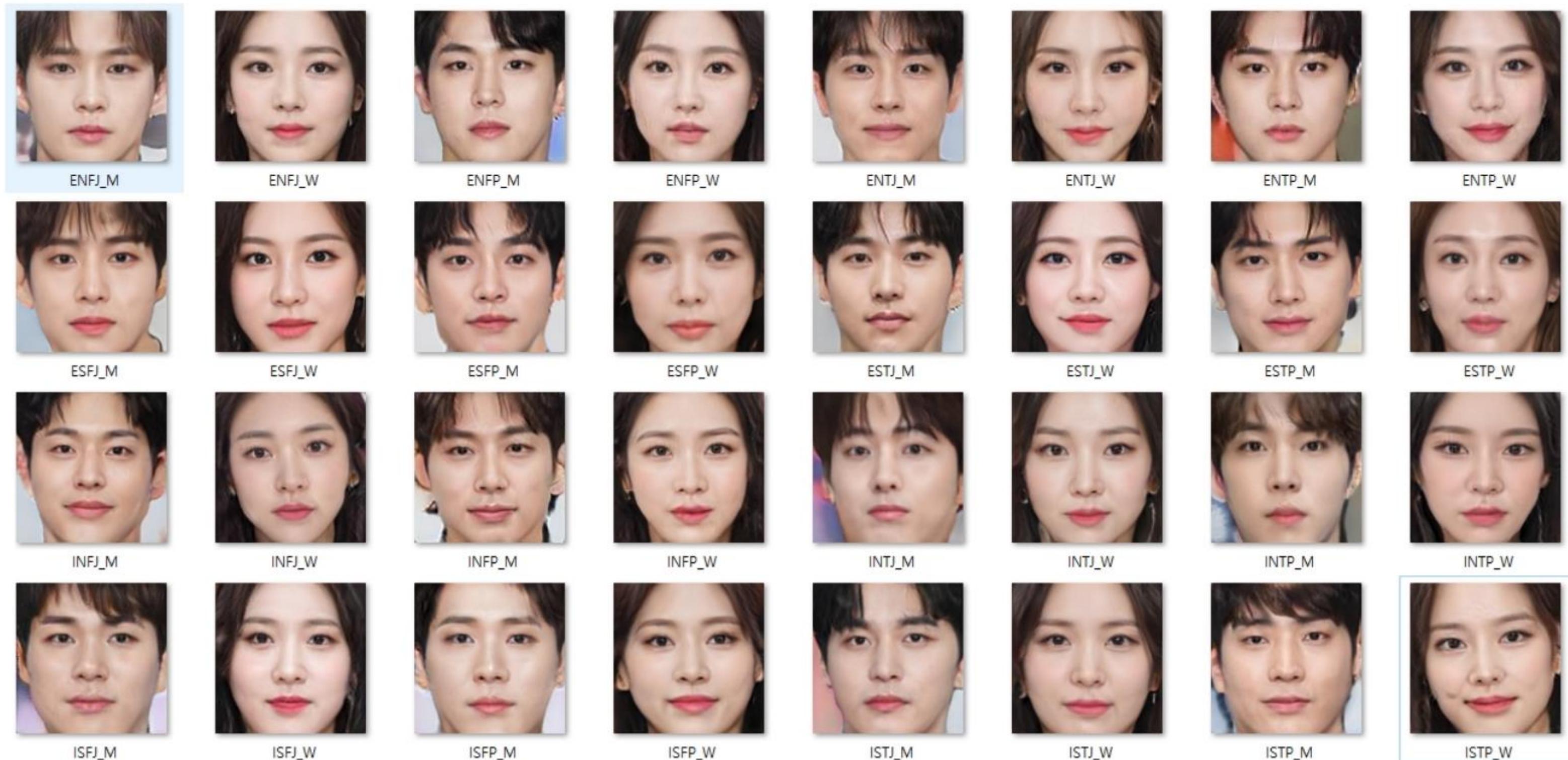
방법1. FreezeD

방법2. truncation_psi=0

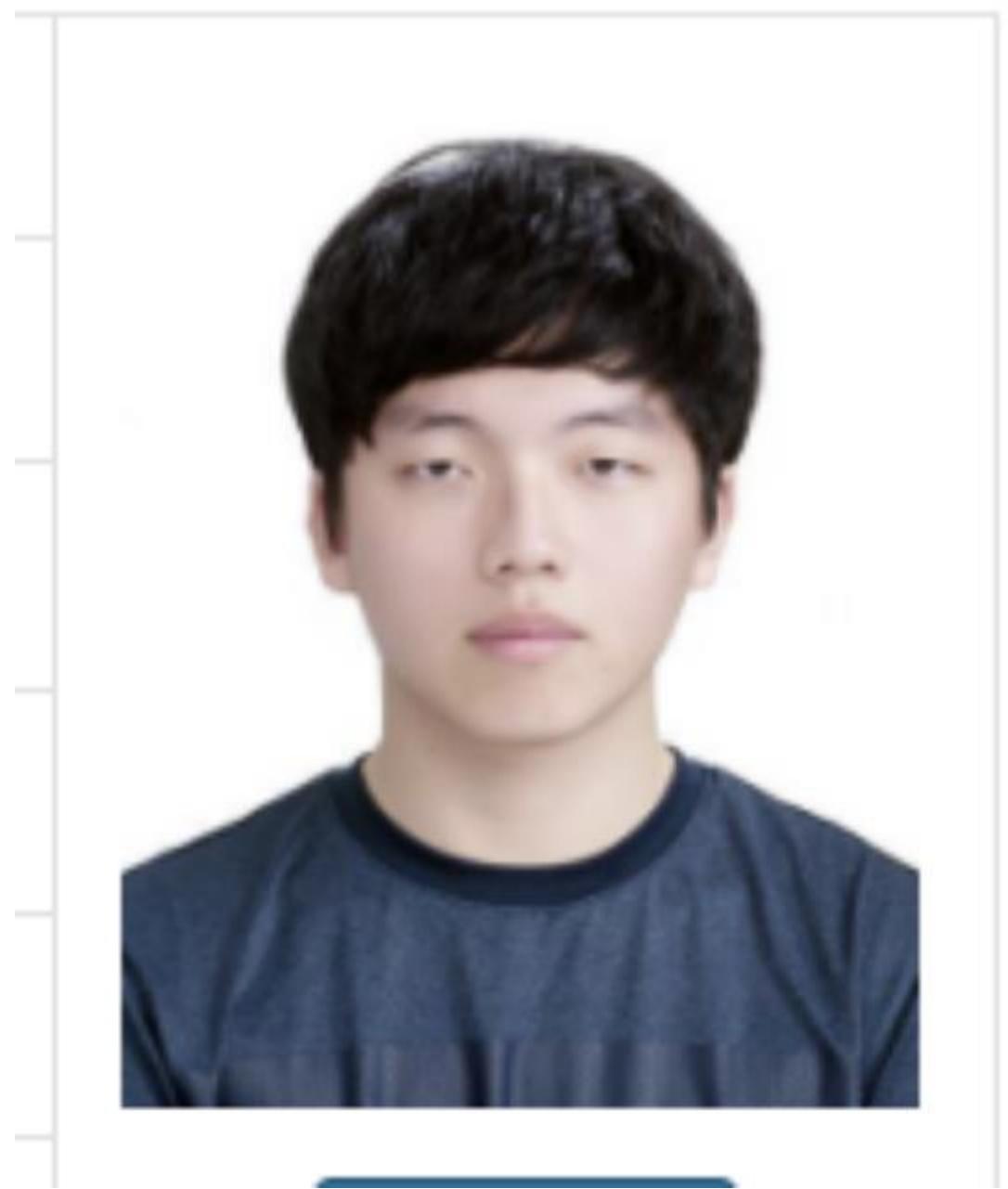


'방법2'의 결과물이 자연스럽고 이상적인 얼굴을 나타내었으므로 최종 결과물로 선택

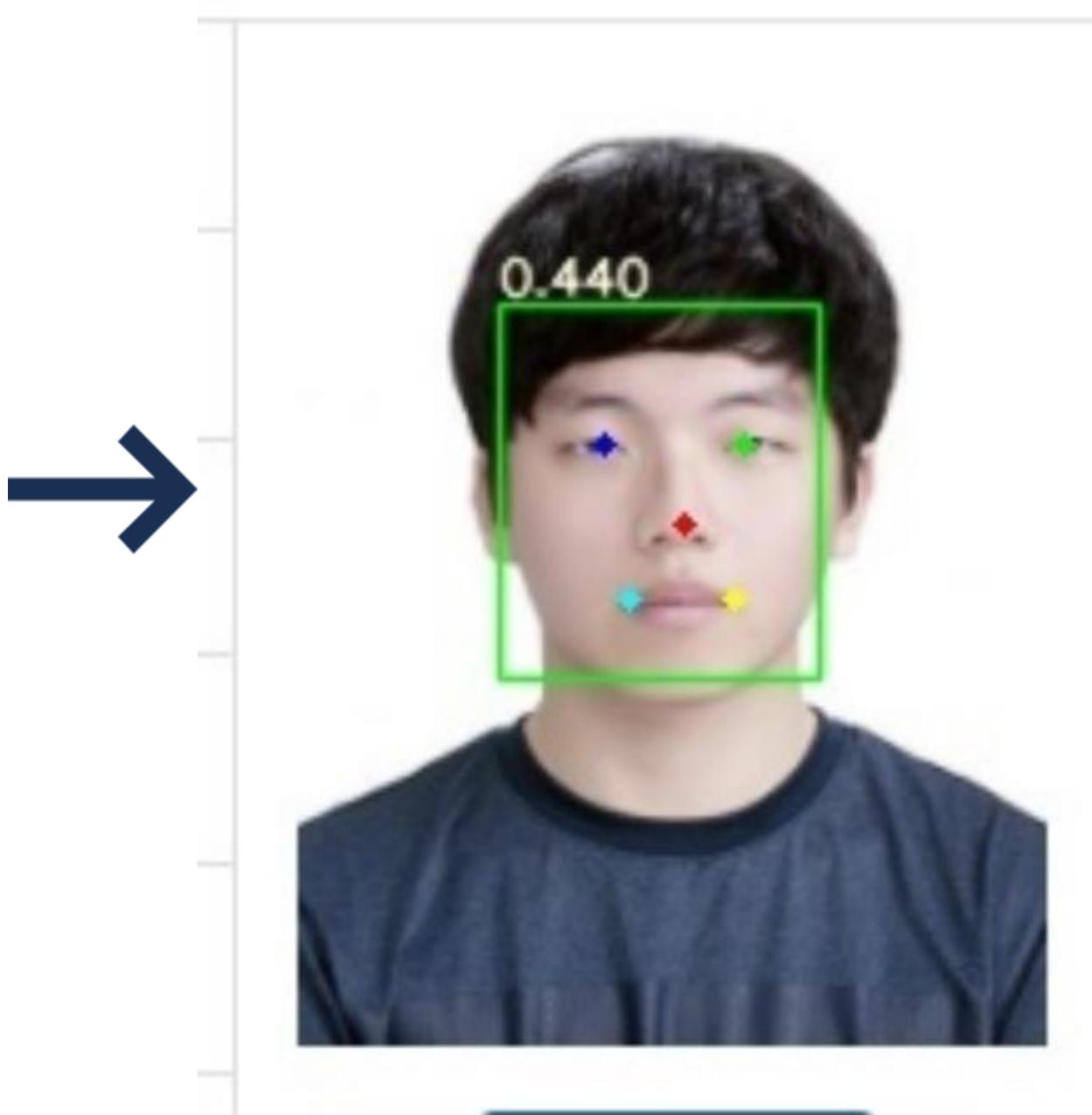
| 유형별 평균 얼굴 - 결과. truncation_psi=0



| YOLO v5



원본



YOLO v5



Crop & Resize

YOLO v5 코드

confidence threshold의 값을 list로 append시킨
뒤, max값을 index로 지정하여 confidence th
read의 max을 인식하도록 지정함



```
bboxes = [] #
confidences = [] #

# Process detections
for i, det in enumerate(pred): # detections per image
    gn = torch.tensor(orgimg.shape)[[1, 0, 1, 0]].to(device)
    # normalization gain whwh
    gn_lks = torch.tensor(orgimg.shape)[[1, 0, 1, 0, 1, 0, 1, 0, 1, 0]].to(
        device) # normalization gain landmarks
    if len(det):
        # Rescale boxes from img_size to im0 size
        det[:, :4] = scale_coords(img.shape[2:], det[:, :4], orgimg
            .shape).round()

        # Print results
        for c in det[:, -1].unique():
            n = (det[:, -1] == c).sum() # detections per class

            det[:, 5:15] = scale_coords_landmarks(img.shape[2:], det[:, 5:15],
                orgimg.shape).round()

            for j in range(det.size()[0]):
                xywh = (xyxy2xywh(det[j, :4].view(1, 4)) / gn).view(-1).tolist()
                conf = det[j, 4].cpu().numpy()
                landmarks = (det[j, 5:15].view(1, 10) / gn_lks).view(-1).tolist()
                class_num = det[j, 15].cpu().numpy()

                bboxes.append(xywh) #
                confidences.append(conf) #

                index = confidences.index(max(confidences)) #

                orgimg = show_results(orgimg, croppedImage, bboxes[index]) #
```

| Web 구현

Step 1.
MBTI를 입력하자!

당신의 성별은 무엇인가요?

남자 여자

당신의 MBTI를 작성해주세요!

Input Text Here

만약 내 MBTI를 모르겠다면? [Go to the link](#)
지금 바로 테스트 하러가기

CANCEL SUBMIT

Step 2.
당신의 유형별 평균관상은?

당신은 ENFP입니다!



유형별 평균얼굴

ENF는 재기발랄한 활동가, 스파크형!
따뜻하고 정열적이고 활기가 넘치며 재능이 많고 상상력이
풍부하다. 온정적이고 창의적이며

당신의 얼굴을 추가해서 더욱 정교화된 유형별
관상을 보여주세요!



당신의얼굴

Step 3.
당신과 최고의 궁합 관상!

당신과 최고의 궁합 유형은 ISTJ입니다!



ISTJ는 청렴결백한 논리주의자, 세상의 소금형!
실제 사실에 대하여 정확하고 체계적으로 기억하며 일 처리
에 있어서도

CANCEL Submit

Time Table

	9월 1주차	9월 2주차	9월 3주차	9월 4주차	10월 1주차	10월 2주차	10월 3주차	10월 4주차
데이터셋 수집	유형별 연예인 사진 수집							
이미지 전처리	크로핑, 리사이즈							
GAN 모델 적용		DCGAN, StyleGAN 모델 적용, StyleGAN2 with ada 모델 적용, FreezedD 적용						
모델링 튜닝			lr 변경, style gan 파라미터 변경 등 튜닝 진행					
Object Detection				yolo5 적용				
Backend 구현					Flask 구현 중			
Frontend 구현					Javascript 구현 중			
최종 완성 단계						완료		
발표							준비 중	

Q&A