Config B를 구현하기 위해 새로 만든 주석에는 왜 그 주석을 만들었는지 기록해 두었습니다.

\*Config A 기준 학습하기 전 fid=342.6042459543894

\*Z normalization 제거(fid=342.63558870037474)

stylegan2-ada-pytorch -> training -> [network.py](http://network.py) -> mappingnetwork 클래스

수정 전

if self.z\_dim > 0:

misc.assert\_shape(z, [None, self.z\_dim])

x = normalize\_2nd\_moment(z.to(torch.float32))

수정 후

if self.z\_dim > 0:

misc.assert\_shape(z, [None, self.z\_dim])

#x = normalize\_2nd\_moment(z.to(torch.float32))

x = z.to(torch.float32) # normalize\_2nd\_moment 제거

\*minibatch stddev 제거(fid=342.6653200327369)

[network.py](http://network.py) -> DiscriminatorEpilogue 클래스

수정 전

self.mbstd = MinibatchStdLayer(group\_size=mbstd\_group\_size, num\_channels=mbstd\_num\_channels) if mbstd\_num\_channels > 0 else None

self.conv = Conv2dLayer(in\_channels + mbstd\_num\_channels, in\_channels, kernel\_size=3, activation=activation, conv\_clamp=conv\_clamp)

수정 후

# self.mbstd = MinibatchStdLayer(group\_size=mbstd\_group\_size, num\_channels=mbstd\_num\_channels) if mbstd\_num\_channels > 0 else None

# self.conv = Conv2dLayer(in\_channels + mbstd\_num\_channels, in\_channels, kernel\_size=3, activation=activation, conv\_clamp=conv\_clamp)

self.mbstd = None

self.conv = Conv2dLayer(in\_channels, in\_channels, kernel\_size=3, activation=activation, conv\_clamp=conv\_clamp)

\*Equalized learning rate(fid=296.3908898862304)

[network.py](http://network.py) -> Fullyconnectedlayer 부분

수정 전

class FullyConnectedLayer(torch.nn.Module):

def \_\_init\_\_(self,

in\_features, # Number of input features.

out\_features, # Number of output features.

bias = True, # Apply additive bias before the activation function?

activation = 'linear', # Activation function: 'relu', 'lrelu', etc.

lr\_multiplier = 1, # Learning rate multiplier.

bias\_init = 0, # Initial value for the additive bias.

):

super().\_\_init\_\_()

self.activation = activation

self.weight = torch.nn.Parameter(torch.randn([out\_features, in\_features]) / lr\_multiplier)

self.bias = torch.nn.Parameter(torch.full([out\_features], float(bias\_init))) if bias else None

self.weight\_gain = lr\_multiplier / np.sqrt(in\_features)

self.bias\_gain = lr\_multiplier

def forward(self, x):

w = self.weight.to(x.dtype) \* self.weight\_gain

b = self.bias

if b is not None:

b = b.to(x.dtype)

if self.bias\_gain != 1:

b = b \* self.bias\_gain

if self.activation == 'linear' and b is not None:

x = torch.addmm(b.unsqueeze(0), x, w.t())

else:

x = x.matmul(w.t())

x = bias\_act.bias\_act(x, b, act=self.activation)

return x

수정 후

class FullyConnectedLayer(torch.nn.Module):

def \_\_init\_\_(self,

in\_features,

out\_features,

bias=True,

activation='linear',

bias\_init=0,

):

super().\_\_init\_\_()

self.activation = activation

self.weight = torch.nn.Parameter(torch.randn([out\_features, in\_features]) \* np.sqrt(2 / in\_features))

self.bias = torch.nn.Parameter(torch.full([out\_features], float(bias\_init))) if bias else None

def forward(self, x):

b = self.bias.to(x.dtype) if self.bias is not None else None

if self.activation == 'linear' and b is not None:

x = torch.addmm(b.unsqueeze(0), x, self.weight.t())

else:

x = x.matmul(self.weight.t())

x = bias\_act.bias\_act(x, b, act=self.activation)

return x

[network.py](http://network.py) -> Conv2dlayer 부분

수정 전

class Conv2dLayer(torch.nn.Module):

def \_\_init\_\_(self,

in\_channels, # Number of input channels.

out\_channels, # Number of output channels.

kernel\_size, # Width and height of the convolution kernel.

bias = True, # Apply additive bias before the activation function?

activation = 'linear', # Activation function: 'relu', 'lrelu', etc.

up = 1, # Integer upsampling factor.

down = 1, # Integer downsampling factor.

resample\_filter = [1,3,3,1], # Low-pass filter to apply when resampling activations.

conv\_clamp = None, # Clamp the output to +-X, None = disable clamping.

channels\_last = False, # Expect the input to have memory\_format=channels\_last?

trainable = True, # Update the weights of this layer during training?

):

super().\_\_init\_\_()

self.activation = activation

self.up = up

self.down = down

self.conv\_clamp = conv\_clamp

self.register\_buffer('resample\_filter', upfirdn2d.setup\_filter(resample\_filter))

self.padding = kernel\_size // 2

self.weight\_gain = 1 / np.sqrt(in\_channels \* (kernel\_size \*\* 2))

self.act\_gain = bias\_act.activation\_funcs[activation].def\_gain

memory\_format = torch.channels\_last if channels\_last else torch.contiguous\_format

weight = torch.randn([out\_channels, in\_channels, kernel\_size, kernel\_size]).to(memory\_format=memory\_format)

bias = torch.zeros([out\_channels]) if bias else None

if trainable:

self.weight = torch.nn.Parameter(weight)

self.bias = torch.nn.Parameter(bias) if bias is not None else None

else:

self.register\_buffer('weight', weight)

if bias is not None:

self.register\_buffer('bias', bias)

else:

self.bias = None

def forward(self, x, gain=1):

w = self.weight \* self.weight\_gain

b = self.bias.to(x.dtype) if self.bias is not None else None

flip\_weight = (self.up == 1) # slightly faster

x = conv2d\_resample.conv2d\_resample(x=x, w=w.to(x.dtype), f=self.resample\_filter, up=self.up, down=self.down, padding=self.padding, flip\_weight=flip\_weight)

act\_gain = self.act\_gain \* gain

act\_clamp = self.conv\_clamp \* gain if self.conv\_clamp is not None else None

x = bias\_act.bias\_act(x, b, act=self.activation, gain=act\_gain, clamp=act\_clamp)

return x

수정 후

class Conv2dLayer(torch.nn.Module):

def \_\_init\_\_(self,

in\_channels,

out\_channels,

kernel\_size,

bias=True,

activation='linear',

up=1,

down=1,

resample\_filter=[1,3,3,1],

conv\_clamp=None,

channels\_last=False,

trainable=True,

):

super().\_\_init\_\_()

self.activation = activation

self.up = up

self.down = down

self.conv\_clamp = conv\_clamp

self.register\_buffer('resample\_filter', upfirdn2d.setup\_filter(resample\_filter))

self.padding = kernel\_size // 2

memory\_format = torch.channels\_last if channels\_last else torch.contiguous\_format

weight = torch.randn([out\_channels, in\_channels, kernel\_size, kernel\_size]) \* np.sqrt(2 / (in\_channels \* kernel\_size \* kernel\_size))

weight = weight.to(memory\_format=memory\_format)

bias = torch.zeros([out\_channels]) if bias else None

if trainable:

self.weight = torch.nn.Parameter(weight)

self.bias = torch.nn.Parameter(bias) if bias is not None else None

else:

self.register\_buffer('weight', weight)

if bias is not None:

self.register\_buffer('bias', bias)

else:

self.bias = None

def forward(self, x, gain=1):

w = self.weight

b = self.bias.to(x.dtype) if self.bias is not None else None

flip\_weight = (self.up == 1)

x = conv2d\_resample.conv2d\_resample(x=x, w=w.to(x.dtype), f=self.resample\_filter, up=self.up, down=self.down, padding=self.padding, flip\_weight=flip\_weight)

act\_clamp = self.conv\_clamp \* gain if self.conv\_clamp is not None else None

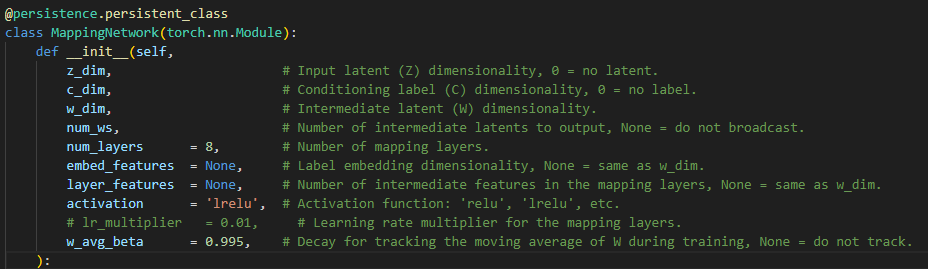
x = bias\_act.bias\_act(x, b, act=self.activation, gain=gain, clamp=act\_clamp) # gain은 그대로 유지하거나 1.0으로 고정 가능

return x

추가로 수정해야 할 부분

lr\_multiplier을 활용하는 모든 부분 제거

\*[network.py](http://network.py) -> mappingnetwork -> lr\_multiplier = 0.01 제거



\*[network.py](http://network.py) -> mappingnetwork -> 매개변수 제거

layer = FullyConnectedLayer(in\_features, out\_features, activation=activation) #lr\_multiplier=lr\_multiplier도 괄호 안에 넣는 게 원래 코드