SinGAN

발표자 : 문하진

출저

SinGAN: Learning a Generative Model from a Single Natural Image

ICCV 2019 · Tamar Rott Shaham, Tali Dekel, Tomer Michaeli

문하진

석 성 훈

황 성 은

최 준 호

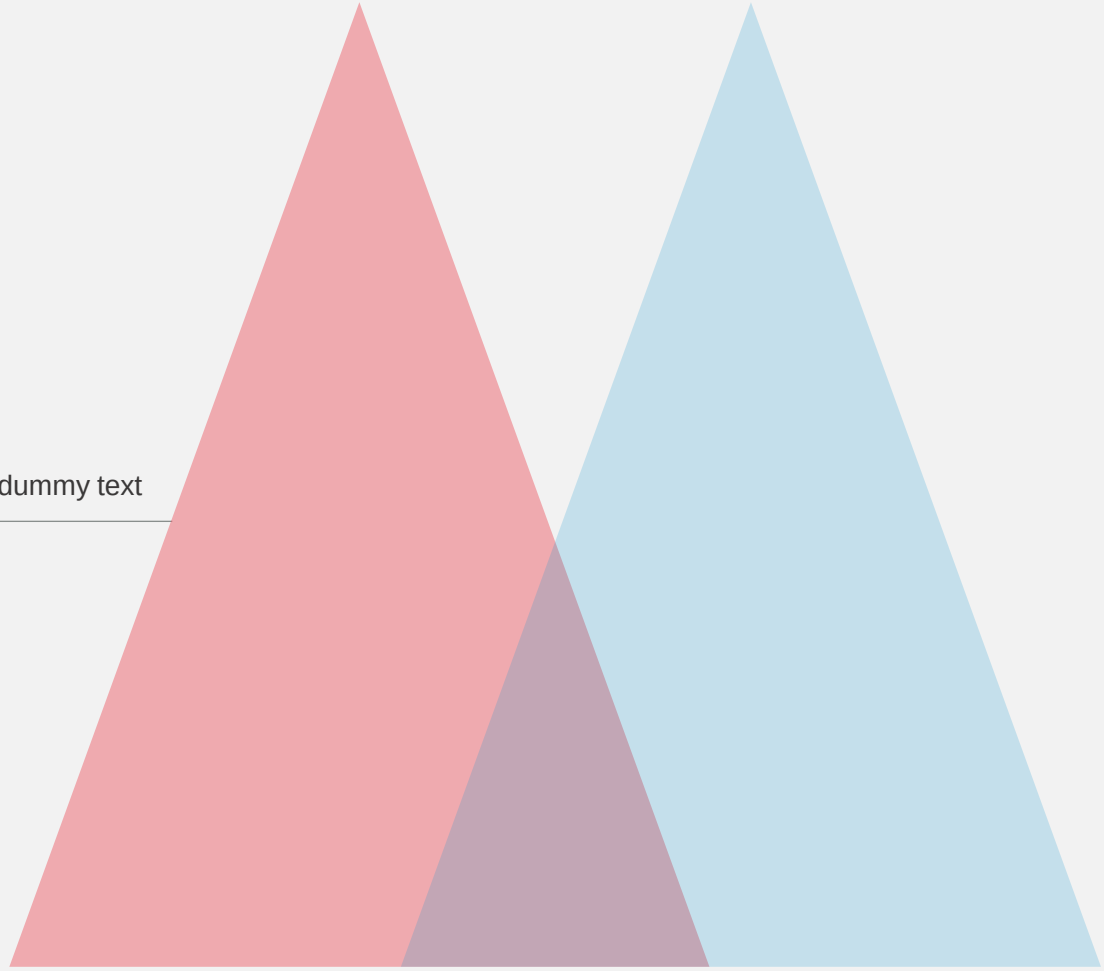
서동재

Contents

001

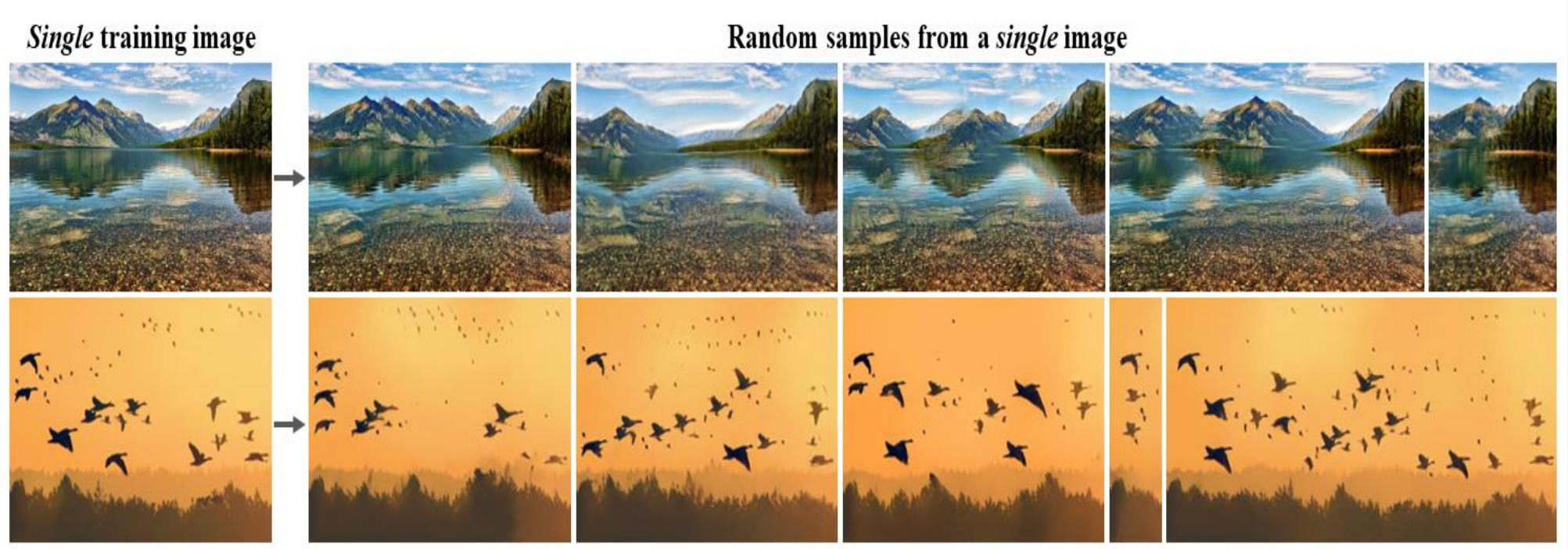
Lorem Ipsum is simply dummy text

SinGAN?



001 SinGAN

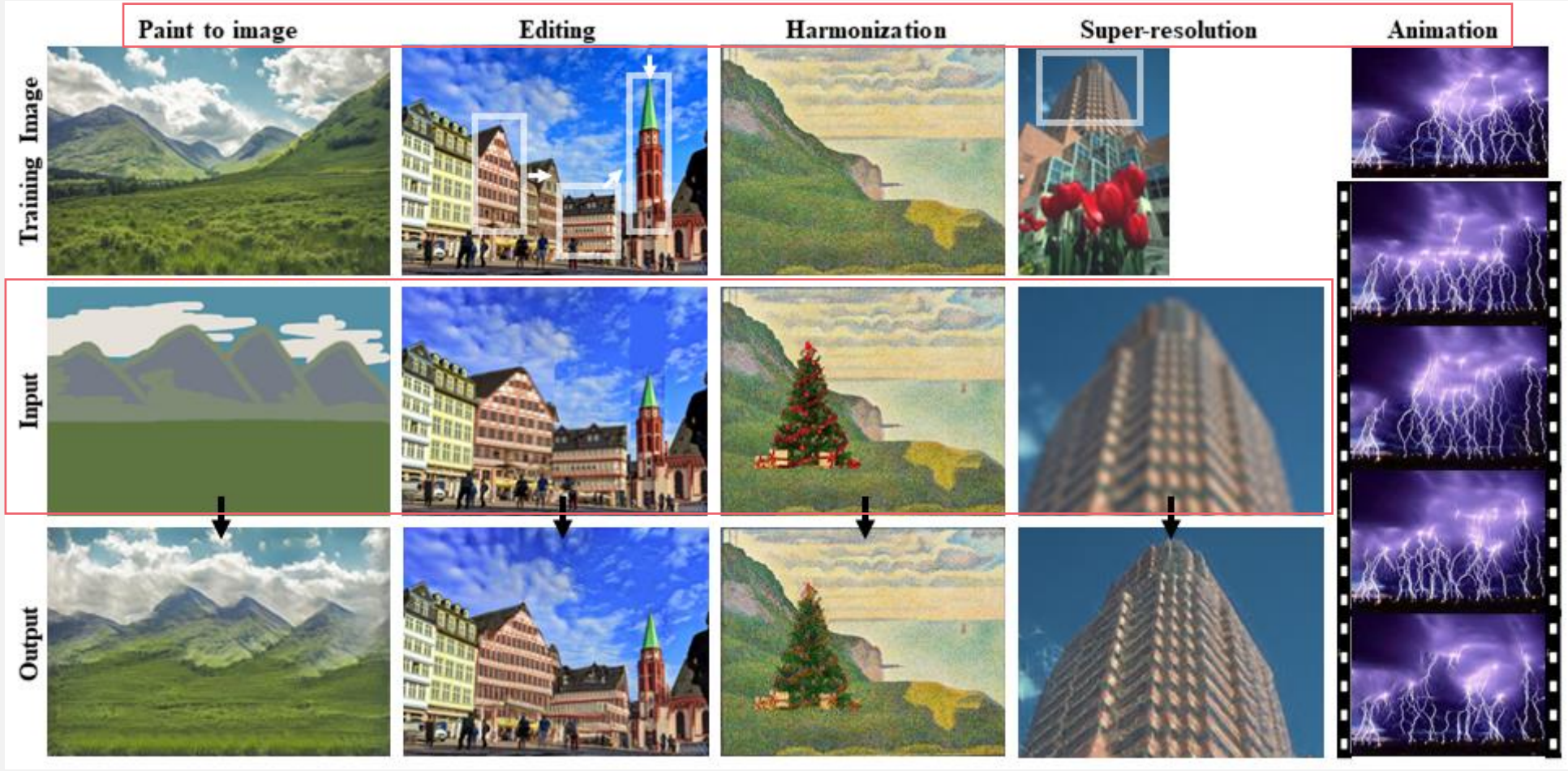
SinGAN이란?



- 한 장의 이미지를 사용해서 **GAN** 네트워크를 학습합니다
- 한 장의 이미지에 대한 수정**(editing),** 애니메이션**(animation), paint to image** 등의 다양한 어플리케이션을 적용할 수 있습니다**.**

001 SinGAN

다양한 활용

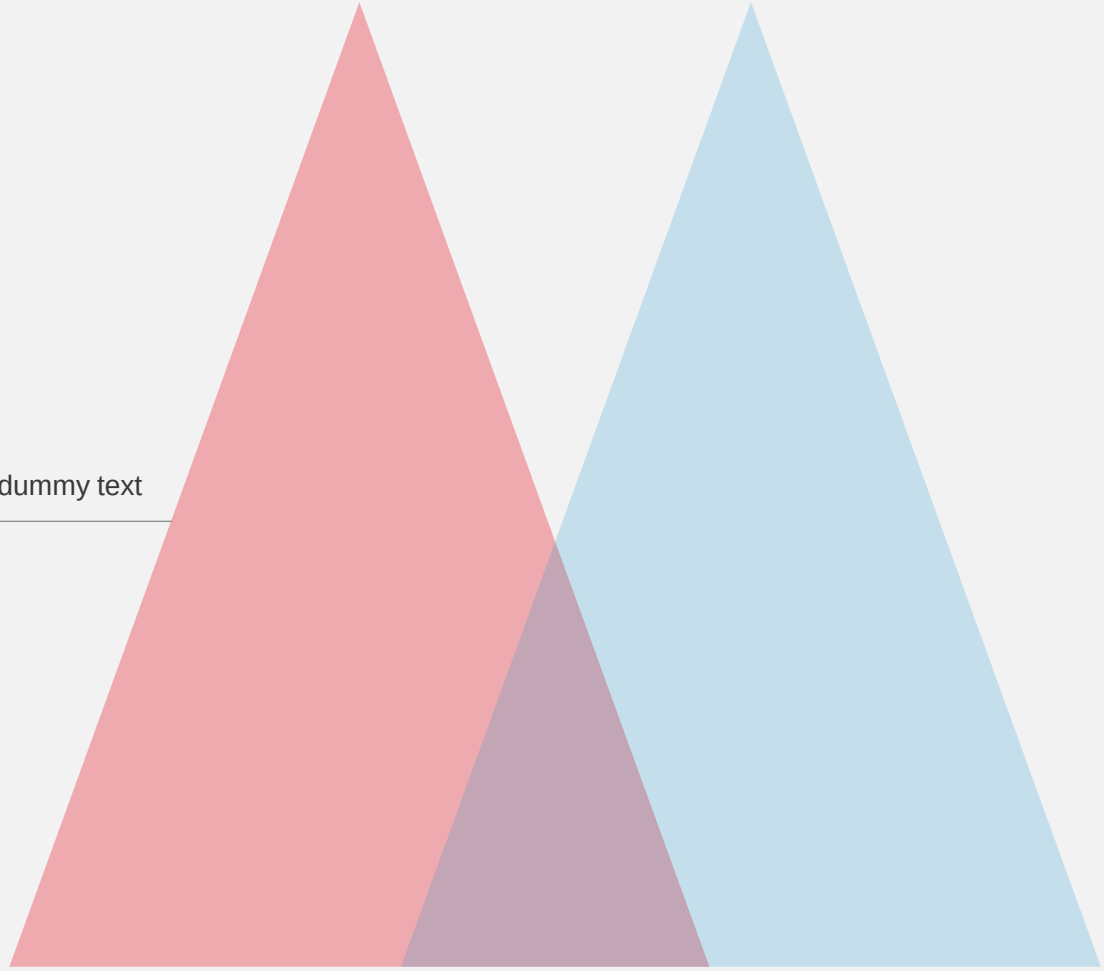


Input을 주었을때 어느정도 있을법한 이미지를 **Output** 함

002

Lorem Ipsum is simply dummy text

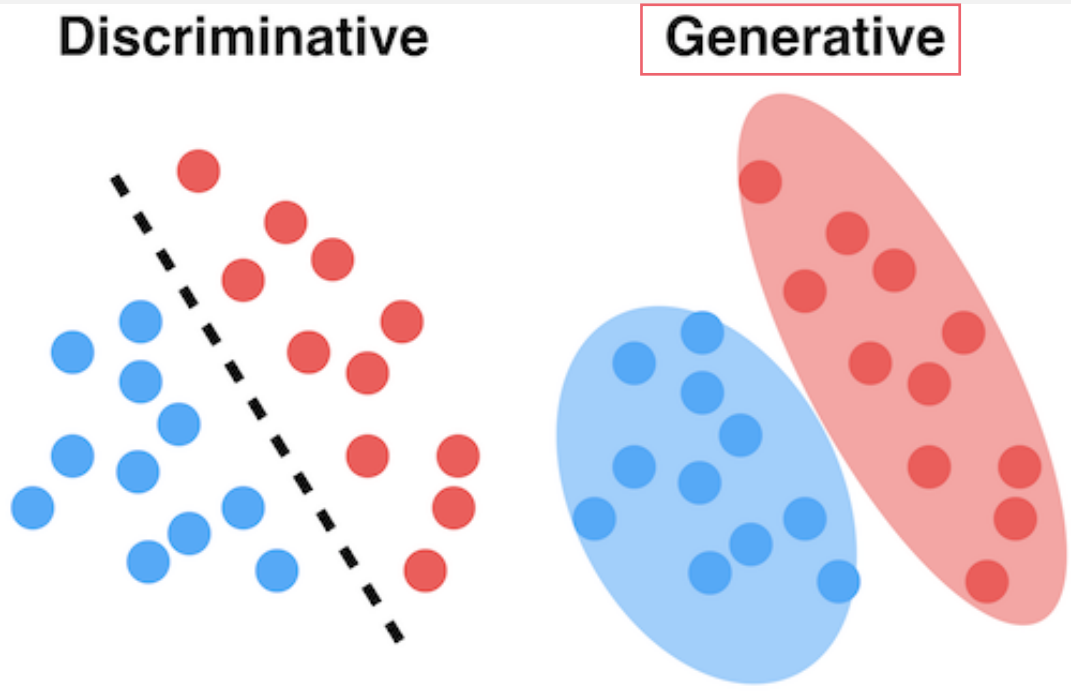
GAN?

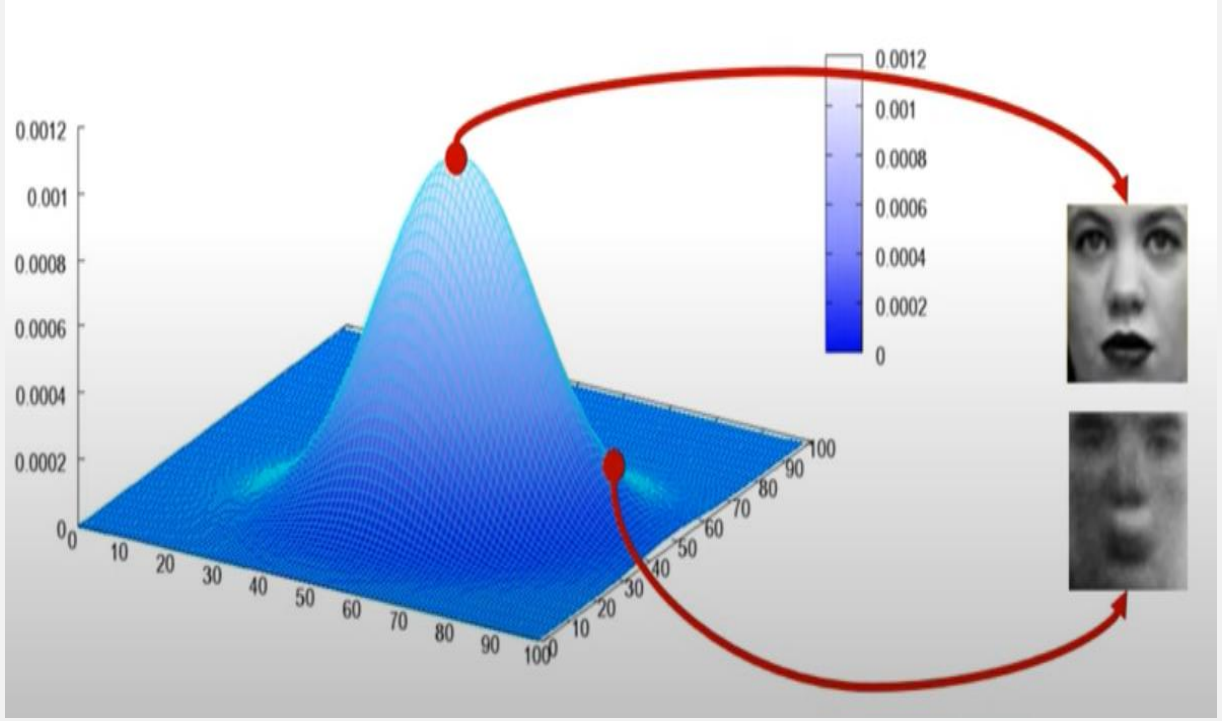


002 GAN?

생성모델

이미지의 분포를 학습시켜 있을 법한 이미지를 생성하는 모델





002 GAN?

Generative Adversarial Network

생성자(Generator)와 판별자(Discriminator) 두 개의 네트워크를 활용한 생성모델

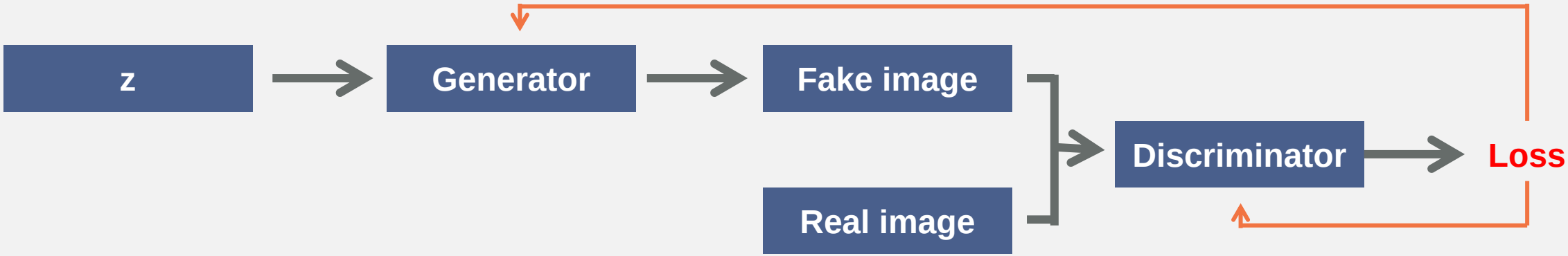
$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

하나의 이미지를 샘플링 한 뒤 log값을 취한 평균값(기대값)이 높아지는 방향으로 학습

노이즈를 추가하여 다양한 랜점 이미지 생성하여 로그를씌운 평균값을 낮추는 방향

Generator 다양한 랜덤 New data 생성하기위한 학습목적

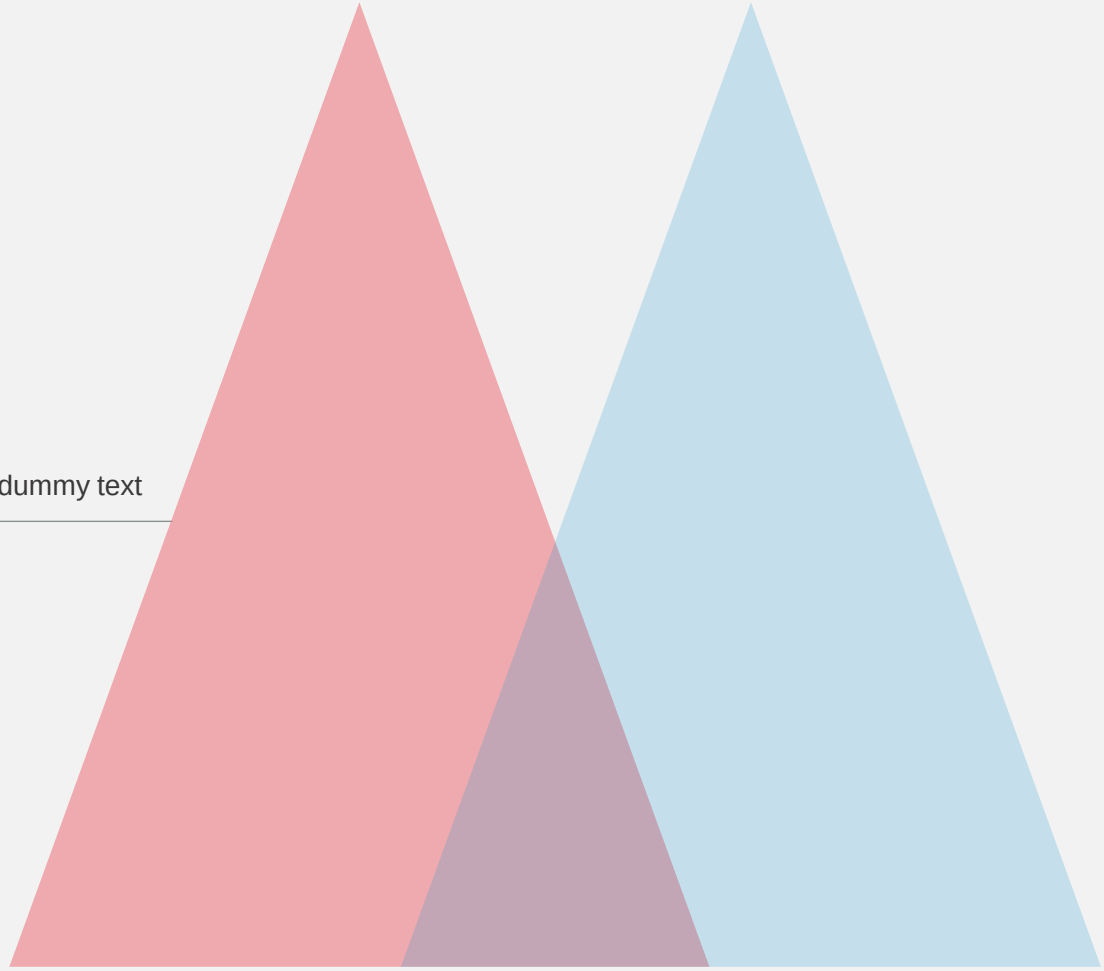
Discriminator Real : 1 ~ Fake : 0 구분하기위한 학습목적



003

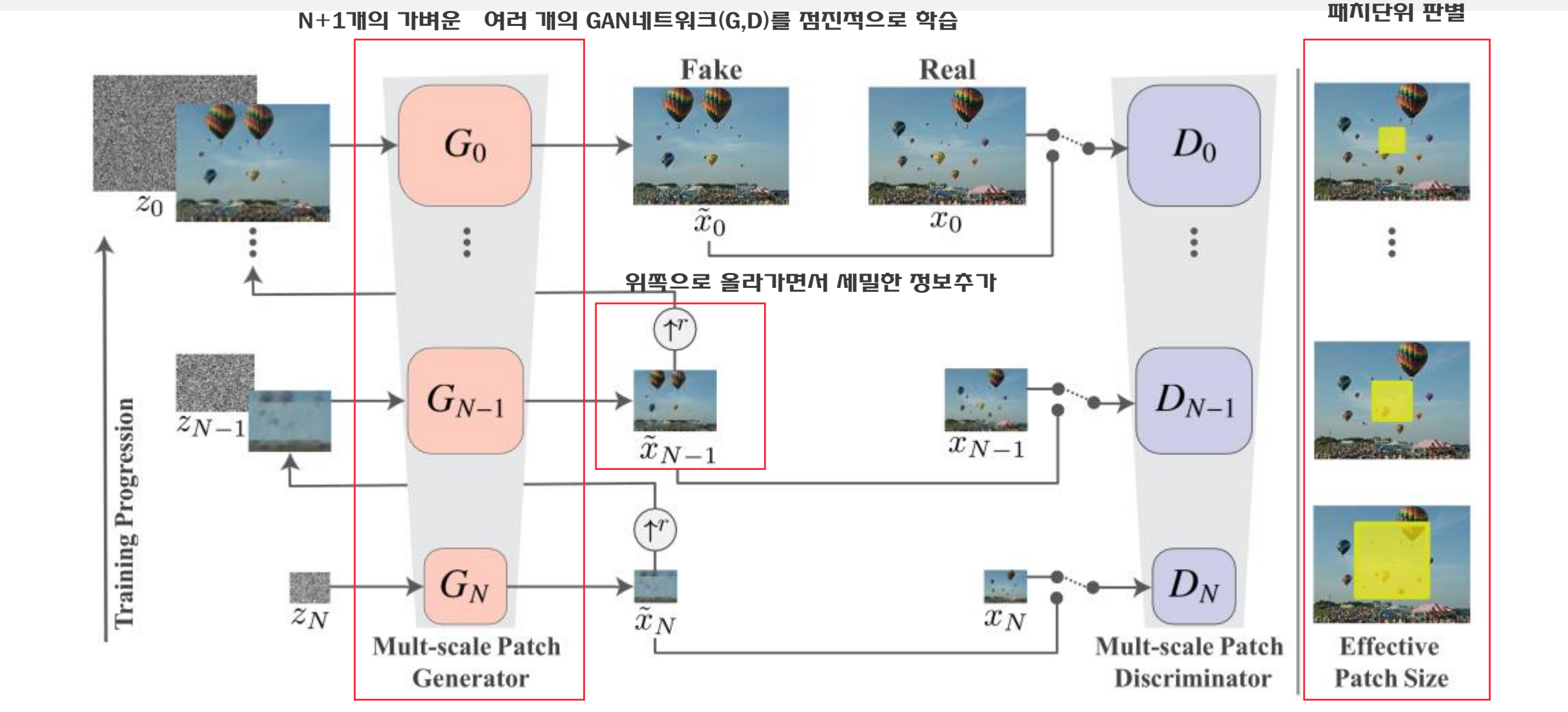
Lorem Ipsum is simply dummy text

구조



003 구조

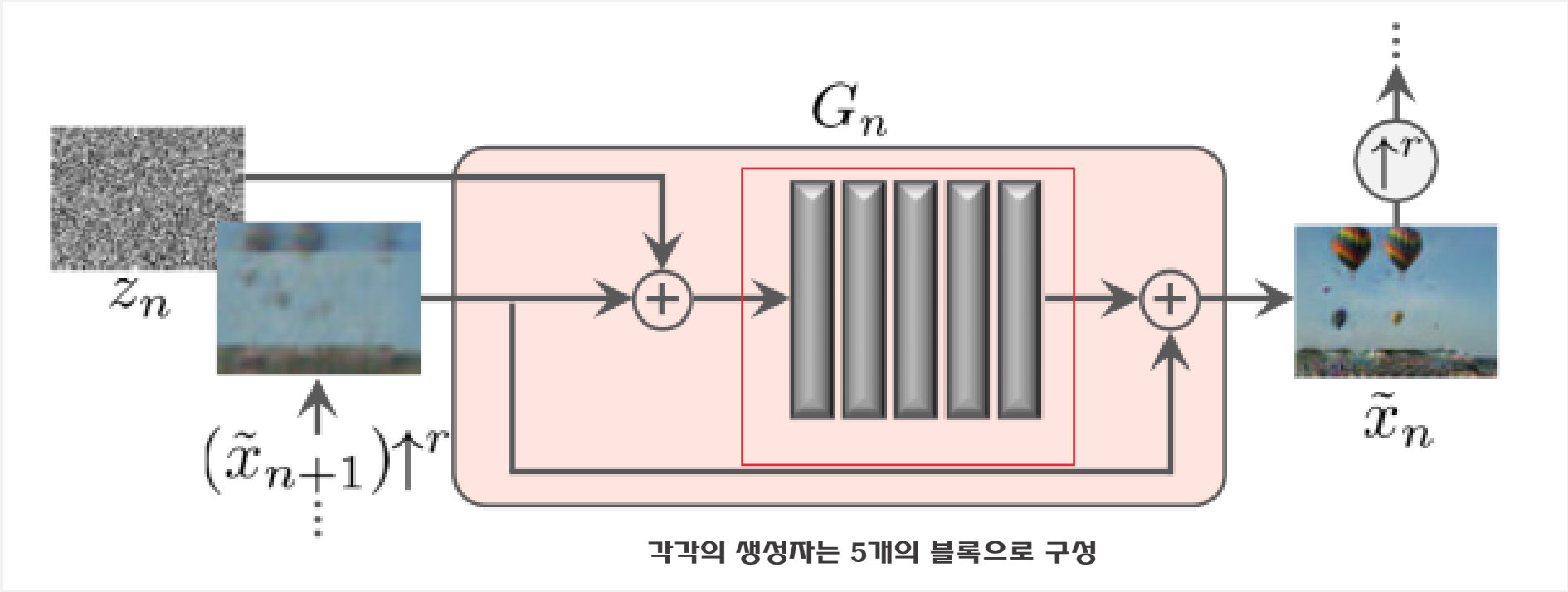
전체 파이프라인



003 구조

architecture

잔여학습을 이용하여 세밀한 정보를 추가



각각의 생성자는 5개의 블록으로 구성

(각 블록은 Conv(3x3) – BatchNorm–LeakyReLU)

003 구조

SinGAN 목적함수

$$\min_{G_n} \max_{D_n} \mathcal{L}_{\text{adv}}(G_n, D_n) + \alpha \mathcal{L}_{\text{rec}}(G_n).$$

Adversarial loss : 실제이미지 와 가짜 이미지 패치의 분포가 같도록 학습시킴

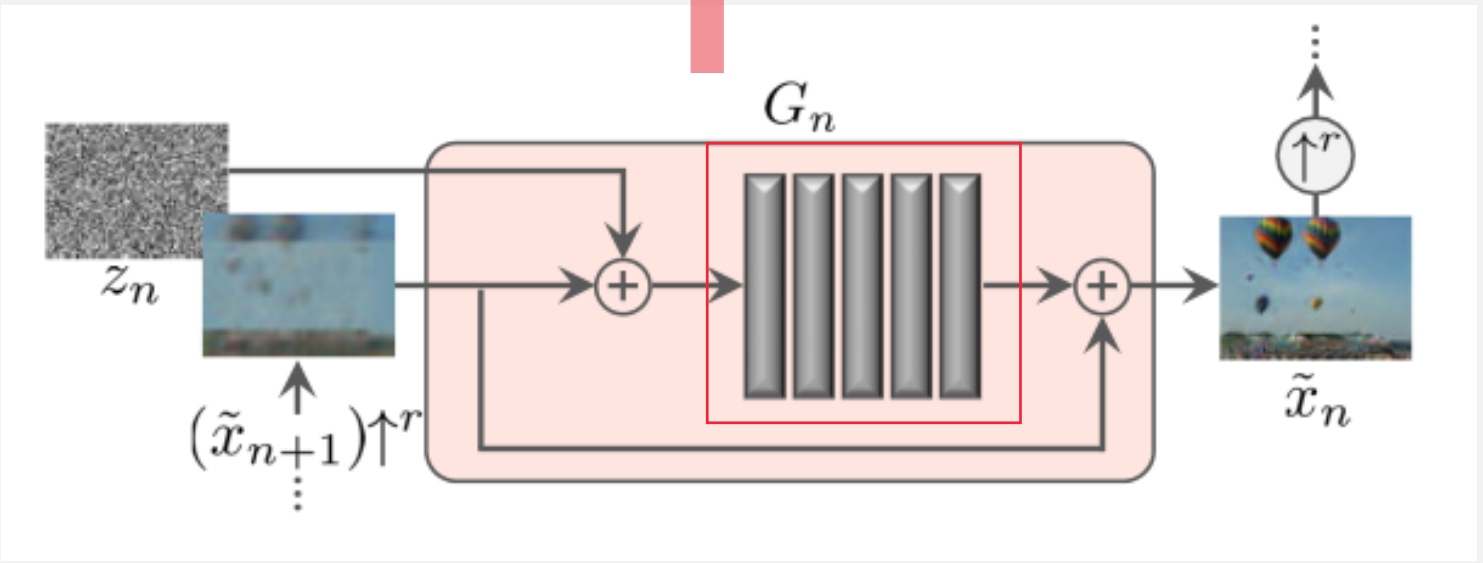
Reconstruction loss : 실제이미지를 정확히 생성할수있도록 학습

노이즈의 값으로 **0**을 넣었을 때, 실제 이미지와 동일한 이미지를 생성

003 구조

Code model

```python
class ConvBlock(nn.Sequential):
    def __init__(self, in_channel, out_channel, ker_size, padd, stride):
        super(ConvBlock,self).__init__()
        self.add_module('conv',nn.Conv2d(in_channel ,out_channel,kernel_size=ker_size,stride=stride,padding=padd)),
        self.add_module('norm',nn.BatchNorm2d(out_channel)),
        self.add_module('LeakyRelu',nn.LeakyReLU(0.2, inplace=True))
```



003 구조

Code model

```
class WDiscriminator(nn.Module):
    def __init__(self, opt):
        super(WDiscriminator, self).__init__()
        self.is_cuda = torch.cuda.is_available()
        N = int(opt.nfc)
        self.head = ConvBlock(opt.nc_im,N,opt.ker_size,opt.padd_size,1)
        self.body = nn.Sequential()
        for i in range(opt.num_layer-2):
            N = int(opt.nfc/pow(2,(i+1)))
            block = ConvBlock(max(2*N,opt.min_nfc),max(N,opt.min_nfc),opt.ker_size,opt.padd_size,1)
            self.body.add_module('block%d'%(i+1),block)
        self.tail = nn.Conv2d(max(N,opt.min_nfc),1,kernel_size=opt.ker_size,stride=1,padding=opt.padd_size)

    def forward(self,x):
        x = self.head(x)
        x = self.body(x)
        x = self.tail(x)
        return x
```

003 구조

Code model

```
class GeneratorConcatSkip2CleanAdd(nn.Module):
    def __init__(self, opt):
        super(GeneratorConcatSkip2CleanAdd, self).__init__()
        self.is_cuda = torch.cuda.is_available()
        N = opt.nfc
        self.head = ConvBlock(opt.nc_im,N,opt.ker_size,opt.padd_size,1) #GenConvTransBlock(opt.nc_z,N,opt.ker_size,opt.padd
        self.body = nn.Sequential()
        for i in range(opt.num_layer-2):
            N = int(opt.nfc/pow(2,(i+1)))
            block = ConvBlock(max(2*N,opt.min_nfc),max(N,opt.min_nfc),opt.ker_size,opt.padd_size,1)
            self.body.add_module('block%d'%(i+1),block)
        self.tail = nn.Sequential(
            nn.Conv2d(max(N,opt.min_nfc),opt.nc_im,kernel_size=opt.ker_size,stride =1,padding=opt.padd_size),
            nn.Tanh()
        )
    def forward(self,x,y):
        x = self.head(x)
        x = self.body(x)
        x = self.tail(x)
        ind = int((y.shape[2]-x.shape[2])/2)
        y = y[:,:,ind:(y.shape[2]-ind),ind:(y.shape[3]-ind)]
```

003 구조

Training.py

```
def train_single_scale(netD,netG,reals,Gs,Zs,in_s,NoiseAmp,opt,centers=None):

    real = reals[len(Gs)]
    opt.nzx = real.shape[2]#+(opt.ker_size-1)*(opt.num_layer)
    opt.nzy = real.shape[3]#+(opt.ker_size-1)*(opt.num_layer)
    opt.receptive_field = opt.ker_size + ((opt.ker_size-1)*(opt.num_layer-1))*opt.stride
    pad_noise = int(((opt.ker_size - 1) * opt.num_layer) / 2)
    pad_image = int(((opt.ker_size - 1) * opt.num_layer) / 2)
    if opt.mode == 'animation_train':
        opt.nzx = real.shape[2]+(opt.ker_size-1)*(opt.num_layer)
        opt.nzy = real.shape[3]+(opt.ker_size-1)*(opt.num_layer)
        pad_noise = 0
    m_noise = nn.ZeroPad2d(int(pad_noise))
    m_image = nn.ZeroPad2d(int(pad_image))

    alpha = opt.alpha
```

```
# setup optimizer
optimizerD = optim.Adam(netD.parameters(), lr=opt.lr_d, betas=(opt.beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=opt.lr_g, betas=(opt.beta1, 0.999))
schedulerD = torch.optim.lr_scheduler.MultiStepLR(optimizer=optimizerD,milestones=[1600],gamma=opt.gamma)
schedulerG = torch.optim.lr_scheduler.MultiStepLR(optimizer=optimizerG,milestones=[1600],gamma=opt.gamma)
```

Larning rate 를 감소시킬 epoch을 지정해줌

Milestone : learning rate를 줄일 epoch index의 lsit
Gamma : gamma 비율로 lr 감소

003 구조

Training.py – update D network

```
# train with fake
if (j==0) & (epoch == 0):
    if (Gs == []) & (opt.mode != 'SR_train'):
        prev = torch.full([1,opt.nc_z,opt.nzx,opt.nzy], 0, device=opt.device)
        in_s = prev
        prev = m_image(prev)
        z_prev = torch.full([1,opt.nc_z,opt.nzx,opt.nzy], 0, device=opt.device)
        z_prev = m_noise(z_prev)
        opt.noise_amp = 1
    elif opt.mode == 'SR_train':
        z_prev = in_s
        criterion = nn.MSELoss()
        RMSE = torch.sqrt(criterion(real, z_prev))
        opt.noise_amp = opt.noise_amp_init * RMSE
        z_prev = m_image(z_prev)
        prev = z_prev
```

torch.full = Value로 채워진 크기의 텐서를 만듬

Loss를 RMSE로 사용하여 업데이트

003 구조

Training.py – update G network

```
for j in range(opt.Gsteps):
    netG.zero_grad()
    output = netD(fake)
    #D_fake_map = output.detach()
    errG = -output.mean()
    errG.backward(retain_graph=True)
    if alpha!=0:
        loss = nn.MSELoss()
        if opt.mode == 'paint_train':
            z_prev = functions.quant2centers(z_prev, centers)
            plt.imsave('%s/z_prev.png' % (opt.outf), functions.convert_image_np(z_prev), vmin=0, vmax=1)
        Z_opt = opt.noise_amp*z_opt+z_prev
        rec_loss = alpha*loss(netG(Z_opt.detach(),z_prev),real)
        rec_loss.backward(retain_graph=True)
        rec_loss = rec_loss.detach()
    else:
        Z_opt = z_opt
        rec_loss = 0
```

Rec_loss : 실제이미지처럼 생성하게하는 loss

003 구조

활용 코드

editing

animation

```
if __name__ == '__main__':
    parser = get_arguments()
    parser.add_argument('--input_dir', help='input image dir', default='Input/Images')
    parser.add_argument('--input_name', help='training image name', required=True)
    parser.add_argument('--ref_dir', help='input reference dir', default='Input/Editing')
    parser.add_argument('--ref_name', help='reference image name', required=True)
    parser.add_argument('--editing_start_scale', help='editing injection scale', type=int, required=True)
    parser.add_argument('--mode', help='task to be done', default='editing')
    opt = parser.parse_args()
    opt = functions.post_config(opt)
```

Paint to image

```
if __name__ == '__main__':
    parser = get_arguments()
    parser.add_argument('--input_dir', help='input image dir', default='Input/Images')
    parser.add_argument('--input_name', help='training image name', required=True)
    parser.add_argument('--ref_dir', help='input reference dir', default='Input/Paint')
    parser.add_argument('--ref_name', help='reference image name', required=True)
    parser.add_argument('--paint_start_scale', help='paint injection scale', type=int, required=True)
    parser.add_argument('--quantization_flag', help='specify if to perform color quantization training', type=bool,
    parser.add_argument('--mode', help='task to be done', default='paint2image')
    opt = parser.parse_args()
    opt = functions.post_config(opt)
```

활용 코드

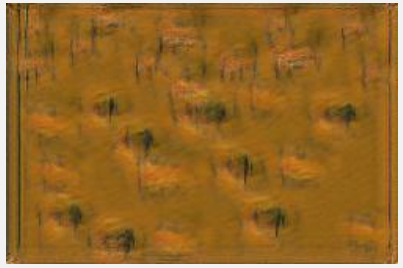
animation

```
if __name__ == '__main__':
    parser = get_arguments()
    parser.add_argument('--animation_start_scale', type=int, help='generation start scale', default=2)
    parser.add_argument('--alpha_animation', type=float, help='animation random walk first moment', default=0.1)
    parser.add_argument('--beta_animation', type=float, help='animation random walk second moment', default=0.9)
    parser.add_argument('--data_dir', default='C:/Users/bitcamp/SinGAN/Input/Images', help='path to dataset')
    parser.add_argument('--dataset', default='mountains', help='type of dataset', choices=['mountains'])
    parser.add_argument('--mode', help='task to be done', default='animation')
    opt = parser.parse_args()
    opt = functions.post_config(opt)
    Gs = []
    Zs = []
    reals = []
    NoiseAmp = []
    dir2save = functions.generate_dir2save(opt)
```

harmonization

```
if __name__ == '__main__':
    parser = get_arguments()
    parser.add_argument('C:/Users/bitcamp/', help='input image dir', default='Input/Images')
    parser.add_argument('image.jpg', help='training image name', required=True)
    parser.add_argument('C:/Users/bitcamp', help='input reference dir', default='Input/Harmonization')
    parser.add_argument('ref_image', help='reference image name', required=True)
    parser.add_argument('harmonization_start_scale=2', help='harmonization injection scale', type=int, required=True)
    parser.add_argument('--mode', help='task to be done', default='harmonization')
    opt = parser.parse_args()
    opt = functions.post_config(opt)
```

Output

	editing	harmonization	Paint to image	animation
Training image				
Input (ref image)				
Output				

004

Lorem Ipsum is simply dummy text

QnA