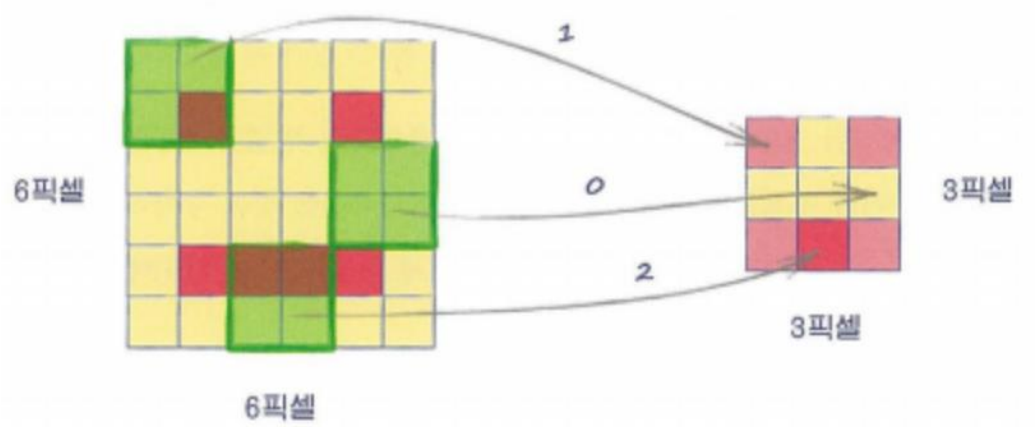
DCGAN

Deep-Convolution GAN

201600779
김영민

Convolution

Convolution Filter



Filter를 통해 Image의 Locality를 찾아냄

Convolution Filter를 통과함으로써 이미지의 해당 부분을 압축한 정보를 보냄

Convolution

Feature map

압축한 정보들을 Feature Map이라고 함

CNN

MNIST CNN

```
class Classifier(nn.Module):

    def __init__(self):
        # 부모 클래스 초기화
        super().__init__()

        # 신경망 레이어 정의
        self.model = nn.Sequential(
            # expand 1 to 10 filters
            nn.Conv2d(1, 10, kernel_size=5, stride=2),
            nn.LeakyReLU(0.02),
            nn.BatchNorm2d(10),

            # 10 filters to 10 filters
            nn.Conv2d(10, 10, kernel_size=3, stride=2),
            nn.LeakyReLU(0.02),
            nn.BatchNorm2d(10),

            View(250),
            nn.Linear(250, 10),
            nn.Sigmoid()
        )

        # 손실함수 설정
        self.loss_function = nn.BCELoss()

        # 옵티마이저 설정
        self.optimiser = torch.optim.Adam(self.parameters())

        # 변수 초기화
        self.counter = 0
        self.progress = []

        pass
```

$$O = \frac{I - K + 2P}{S} + 1$$

In_channel : 1 5x5 filter size

Out_channel : ->10 Stride = 2

(28-5)//2 +1 -> 12x12

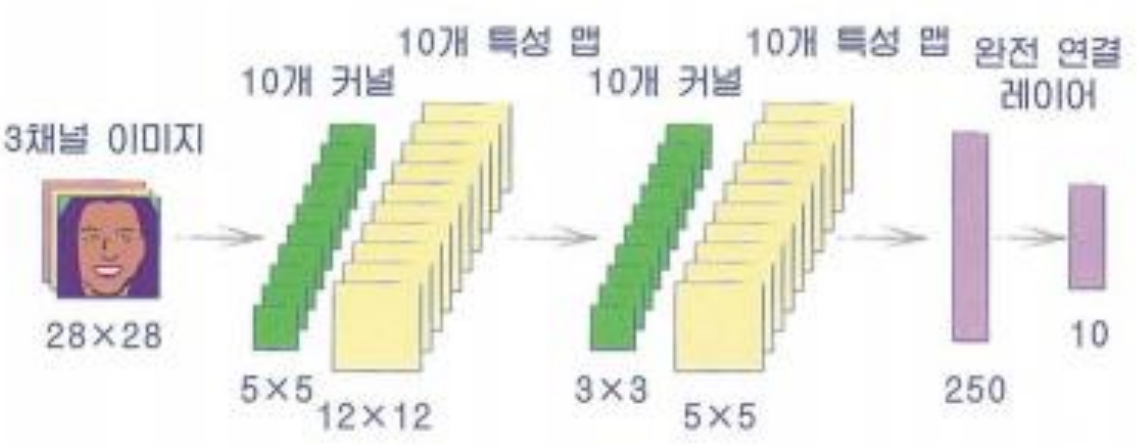
(12-3)//2 +1-> 5x5

5x5x10(number of filter map) = 250

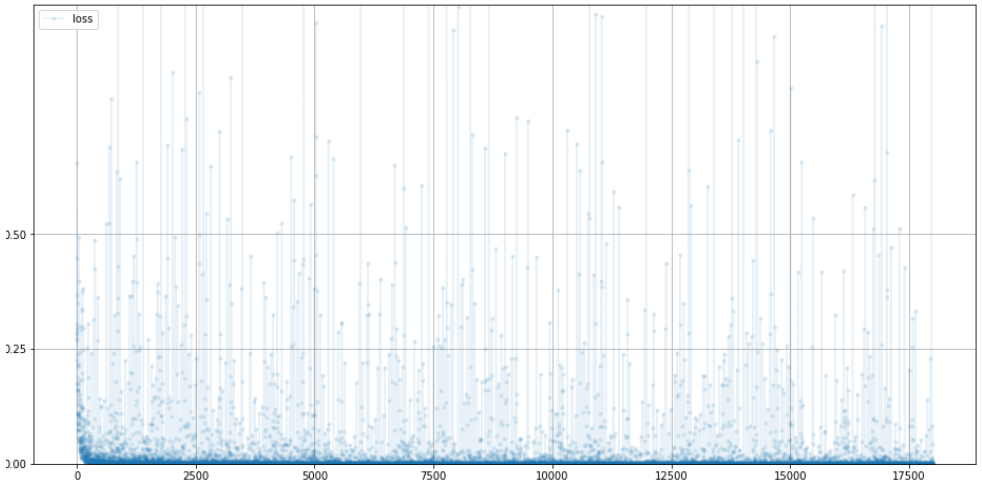
https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

CNN

MNIST CNN Architecture



Loss Function



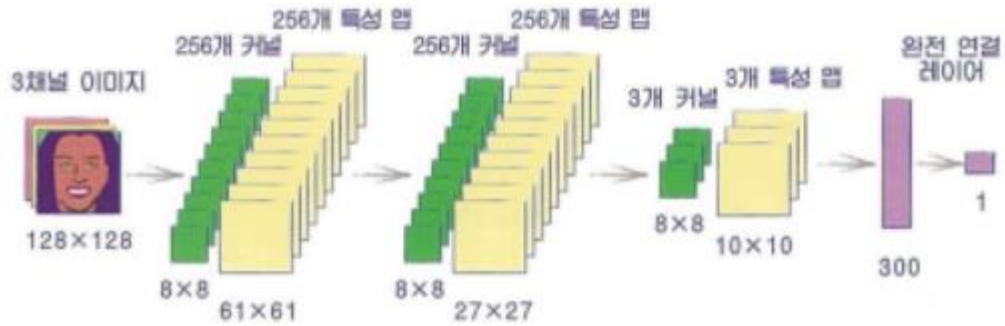
Accuracy : 97.93%

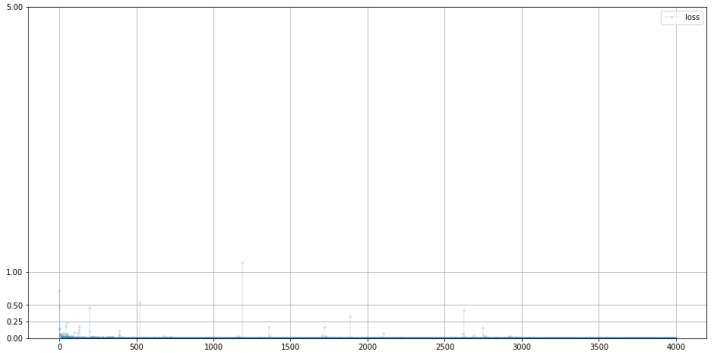
DCGAN

CelebA Discriminator

```
def crop_centre(img, new_width, new_height):
    height, width, _ = img.shape
    startx = width//2 - new_width//2
    starty = height//2 - new_height//2
    return img[  starty:starty + new_height, startx:startx + new_width, :]
class Discriminator(nn.Module):

    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()

        # 신경망 레이어 정의
        self.model = nn.Sequential(
            # (1,3,128,128) 형태를 의도
            nn.Conv2d(3, 256, kernel_size=8, stride=2),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),

            nn.Conv2d(256, 256, kernel_size=8, stride=2),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),

            nn.Conv2d(256, 3, kernel_size=8, stride=2),
            nn.LeakyReLU(0.2),

            View(3*10*10),
            nn.Linear(3*10*10, 1),
            nn.Sigmoid()
        )

        # 손실 함수 생성
        self.loss_function = nn.BCELoss()

        # 옵티마이저 생성
        self.optimiser = torch.optim.Adam(self.parameters(), lr=0.0001)

        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []

        pass
```

Image Cropping

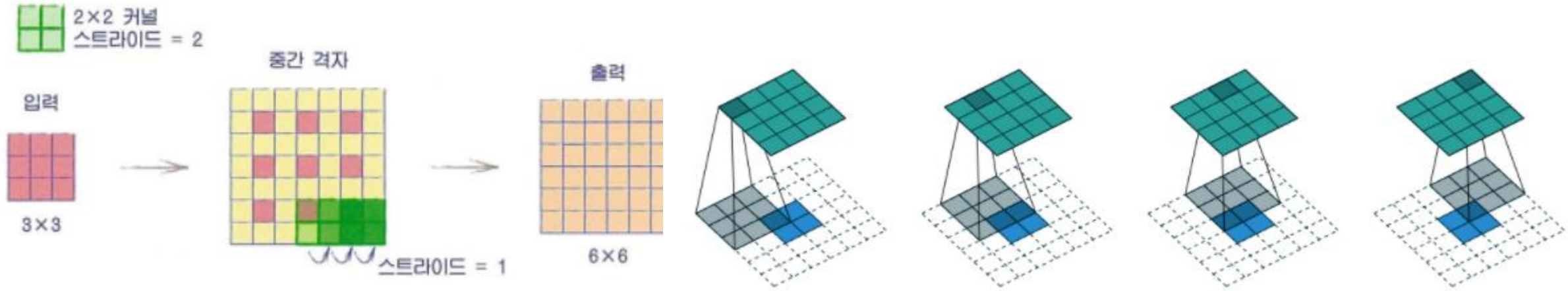




Loss 0으로 수렴

DCGAN

Transposed Convolution



Transposed Convolution을 이용하여 이미지를 복원

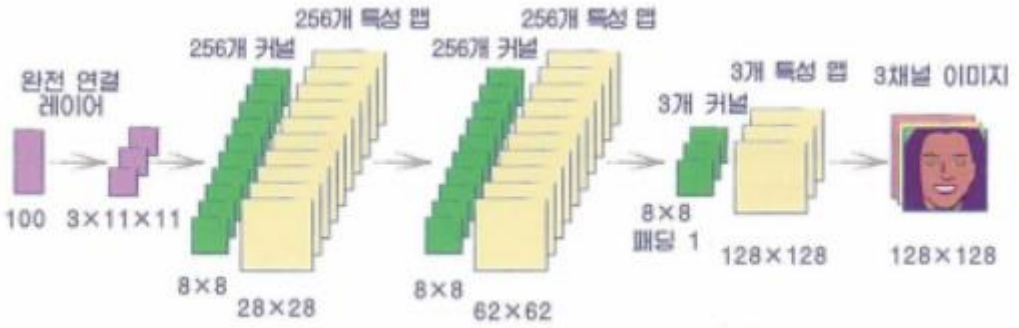
https://simonjisu.github.io/deeplearning/2019/10/27/convtranspose2d.html

https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

DCGAN

Generator

```
class Generator(nn.Module):

    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()

        # 신경망 레이어 정의
        self.model = nn.Sequential(
            # 입력은 1차원 행렬
            nn.Linear(100, 3*11*11),
            nn.LeakyReLU(0.2),

            # 4차원으로 변환
            View((1, 3, 11, 11)),

            nn.ConvTranspose2d(3, 256, kernel_size=8, stride=2),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),

            nn.ConvTranspose2d(256, 256, kernel_size=8, stride=2),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),

            nn.ConvTranspose2d(256, 3, kernel_size=8, stride=2, padding=1),
            nn.BatchNorm2d(3),

            # 출력은 (1,3,128,128) 형태여야 함
            nn.Sigmoid()
        )

        # 옵티마이저 생성
        self.optimiser = torch.optim.Adam(self.parameters(), lr=0.0001)

        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []
```



$$O = (I-1)S - 2P + (K-1) + 1$$

(11-1)*2 +(8-1)+1 = 28 -> 28x28x256

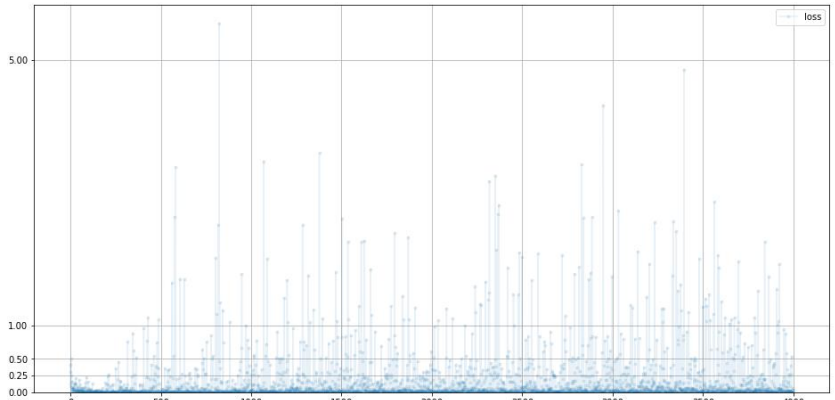
(62-1)*2 +-2+(8-1)+1 -> 128x128x3

https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

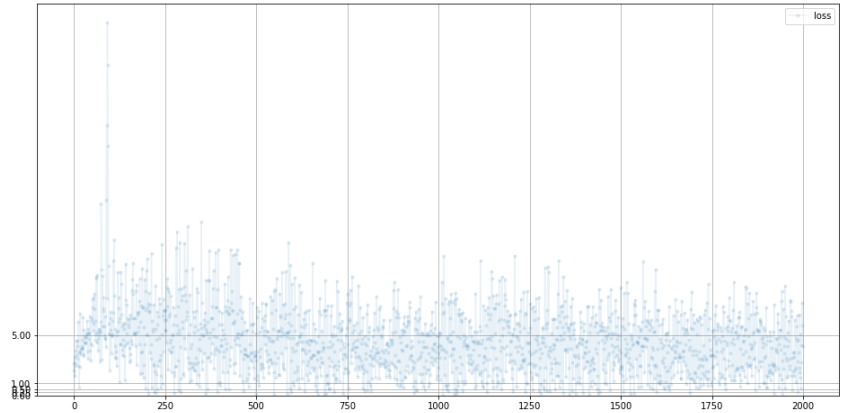
DCGAN

Result

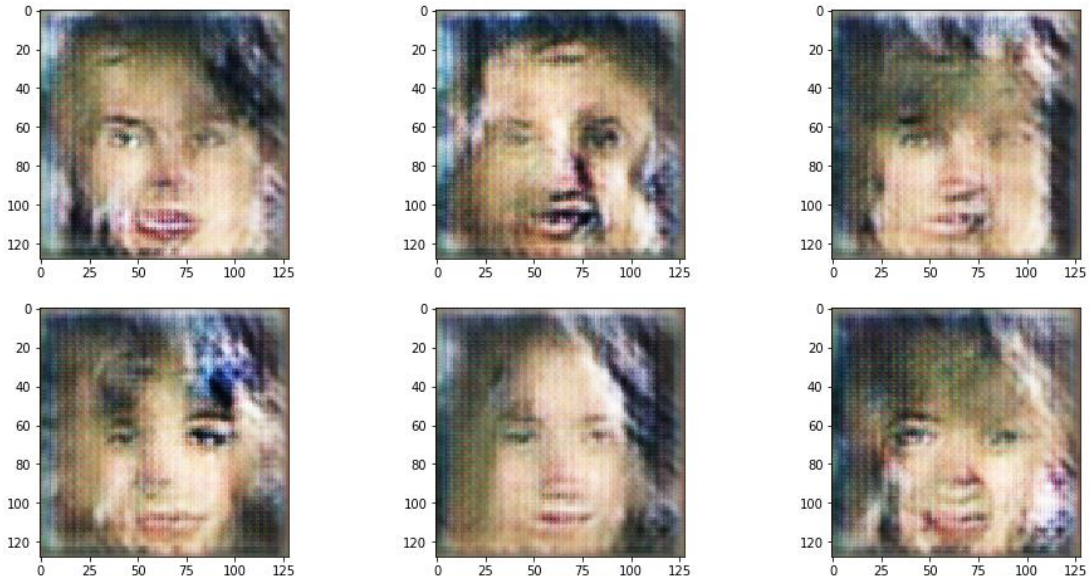
Discriminator Loss



Generator Loss



Result



감사합니다 ^^