생성 모델과 시각 지능

Generative Model and Visual Intelligence

05 주차 | GAN 발전 1

상명대학교 컴퓨터과학과

민 경 하

학습목차

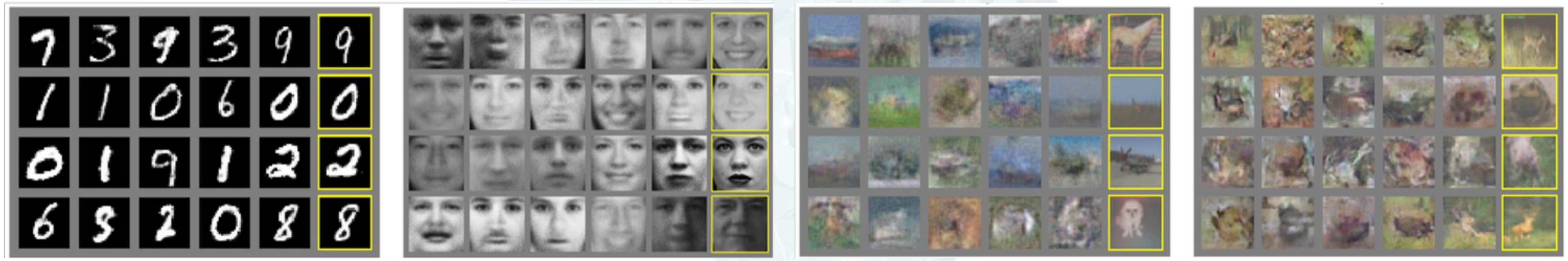
1. DCGAN의 기본 원리
2. DCGAN의 구성 요소 (1): Generator
3. DCGAN의 구성 요소 (2): Discriminator
4. DCGAN의 구성 요소 (3): loss 함수
5. DCGAN의 구성 요소 (4): 훈련

1. DCGAN의 기본 원리

Vanilla GAN (Goodfellow et al. 2014)의 문제점 및 해결 방법

(1) Generates images, but not visually pleasing results

- Datasets
  - MNIST
  - Toronto Face Database
  - CIFAR-10



- Use DCGAN

출처: https://velog.io/@changdaeoh/Generative-Adversarial-Nets-GAN

Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

$$\begin{bmatrix} 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}$$

Convolution (합성곱)

- Convolution의 주요 속성
  - size (kernel의 크기): 3x3, 4x4, 5x5, etc
  - stride (kernel의 적용 단위): 1, 2, 3, etc
  - padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	4x4
stride	1
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{4}\begin{bmatrix} 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	5x5
stride	1
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \frac{1}{5} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	2
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} \boxed{0} & \\ & \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	2
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \frac{1}{3}\underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 \\ \boxed{0} & \end{bmatrix}$$

상명대학교 SANGMYUNG UNIVERSITY

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	2
padding	None

$$\begin{array}{|c|c|c|c|c|c|}\hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} \otimes \frac{1}{3} \underbrace{\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array}}_{\text{kernel}} = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 0 & \boxed{1} \\ \hline \end{array}$$

Convolution (합성곱)

- Convolution의 주요 속성
  - size (kernel의 크기): 3x3, 4x4, 5x5, etc
  - stride (kernel의 적용 단위): 1, 2, 3, etc
  - padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Convolution (합성곱)

- Convolution의 주요 속성
  - size (kernel의 크기): 3x3, 4x4, 5x5, etc
  - stride (kernel의 적용 단위): 1, 2, 3, etc
  - padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	0-padding

$$
\begin{bmatrix}
0&0&0&0&0&0&0&0\\
0&1&1&1&0&0&0&0\\
0&1&1&1&0&0&0&0\\
0&1&1&1&0&0&0&0\\
0&1&1&1&0&0&0&0\\
0&1&1&1&0&0&0&0\\
0&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0
\end{bmatrix}
\otimes
\underbrace{\frac{1}{3}\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}}_{\text{kernel}}
=
\begin{bmatrix}
-.6&0&.6&.6&0&0\\
-1&0&1&1&0&0\\
-1&0&1&1&0&0\\
-1&0&1&1&0&0\\
-1&0&1&1&0&0\\
-.6&0&.6&.6&0&0
\end{bmatrix}
$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	1-padding

$$\begin{bmatrix} 1&1&1&1&1&1&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&1&1&1 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1&0&-1\\ 1&0&-1\\ 1&0&-1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0&0&.6&.6&0&-.6\\ 0&0&1&1&0&-1\\ 0&0&1&1&0&-1\\ 0&0&1&1&0&-1\\ 0&0&1&1&0&-1\\ 0&0&.6&.6&0&-.6 \end{bmatrix}$$

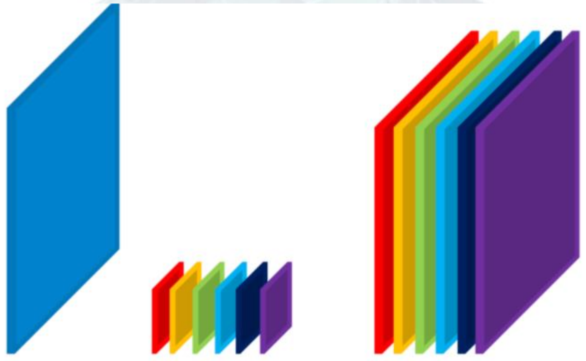
Convolution (합성곱)

Convolution in pytorch

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1,
padding=0,

dilation=1, groups=1, bias=True, padding_mode='zeros')
```

Conv2d(1, 5, 5);



출처: https://velog.io/@gyuho/Convolution-Neural-Network

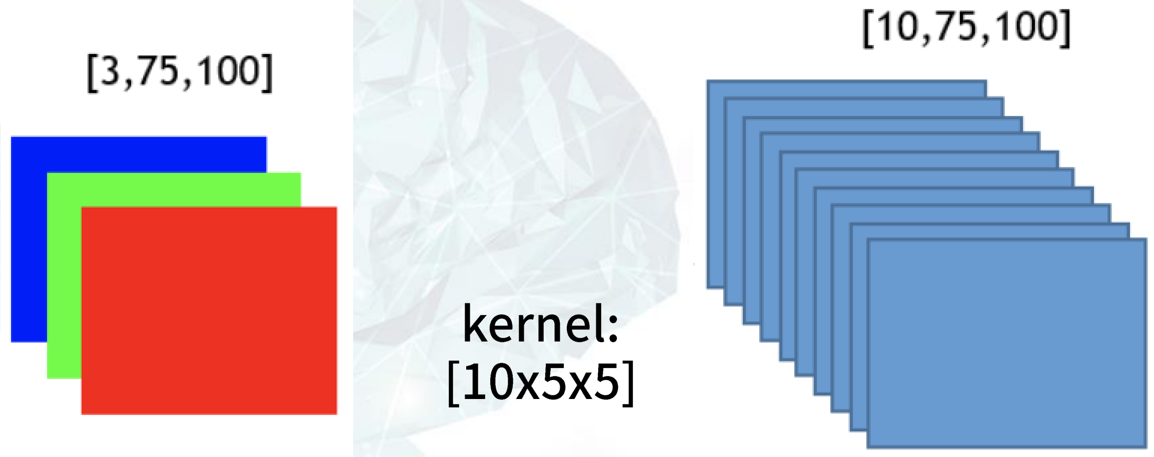
Convolution (합성곱)

Convolution in pytorch

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1,
padding=0,

dilation=1, groups=1, bias=True, padding_mode='zeros')
```

Conv2d(3, 10, 5, 1, 2);



Convolution과 행렬 곱

- Convolution with 3x3 filter

$$\begin{bmatrix} x_0 & x_1 & x_2 & X_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix} \otimes \underbrace{\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}}_{\text{kernel}} \rightarrow \begin{bmatrix} y_0 & y_1 \\ y_2 & y_3 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ X_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Transposed Convolution

- Inverse Convolution with 3x3 filter

$$
\begin{bmatrix}
w_9 & 0 & 0 & 0 \\
w_8 & w_9 & 0 & 0 \\
w_7 & w_8 & 0 & 0 \\
0 & w_7 & 0 & 0 \\
w_6 & 0 & w_9 & 0 \\
w_5 & w_6 & w_8 & w_9 \\
w_4 & w_5 & w_7 & w_8 \\
0 & w_4 & 0 & w_7 \\
w_3 & 0 & w_6 & 0 \\
w_2 & w_3 & w_5 & w_6 \\
w_1 & w_2 & w_4 & w_5 \\
0 & w_1 & 0 & w_4 \\
0 & 0 & w_3 & 0 \\
0 & 0 & w_2 & w_3 \\
0 & 0 & w_1 & w_2 \\
0 & 0 & 0 & w_1
\end{bmatrix}
\times
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}
=
\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ X_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix}
$$

Transposed Convolution

- Inverse Convolution with 3x3 filter
- Convolution filter의 transposed matrix를 곱함
- 영상의 크기가 커지는 연산

w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9	0	0	0	0	0
0	w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9	0	0	0	0
0	0	0	0	w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9	0
0	0	0	0	0	w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9

$\leftrightarrow$

w_9	0	0	0
w_8	w_9	0	0
w_7	w_8	0	0
0	w_7	0	0
w_6	0	w_9	0
w_5	w_6	w_8	w_9
w_4	w_5	w_7	w_8
0	w_4	0	w_7
w_3	0	w_6	0
w_2	w_3	w_5	w_6
w_1	w_2	w_4	w_5
0	w_1	0	w_4
0	0	w_3	0
0	0	w_2	w_3
0	0	w_1	w_2
0	0	0	w_1

Deep Convolutional GAN

- A strong candidate for unsupervised learning
- Walk in the latent space

출처: Radford, A., Metz, L., and Chintala, S., Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, Proc. of ICLR 2016, pp. ??-??, 2016.

Deep Convolutional GAN

- Vanilla GAN에 대한 가장 큰 궁금한 점
  - 왜 Convolutional layer를 사용하지 않았을까?
- 지금의 심층 학습을 만든 가장 중요한 2가지 기술
  - CNN (1998)
  - Deep Brief Net (2006)
- Convolutional layer를 다층 (deep) 구조로 연결함으로써 지금의 심층 학습 기술이 발전함 → AlexNet (2011)

! DC GAN = GAN + Convolutional layer

Deep Convolutional GAN

Design guideline for DCGAN

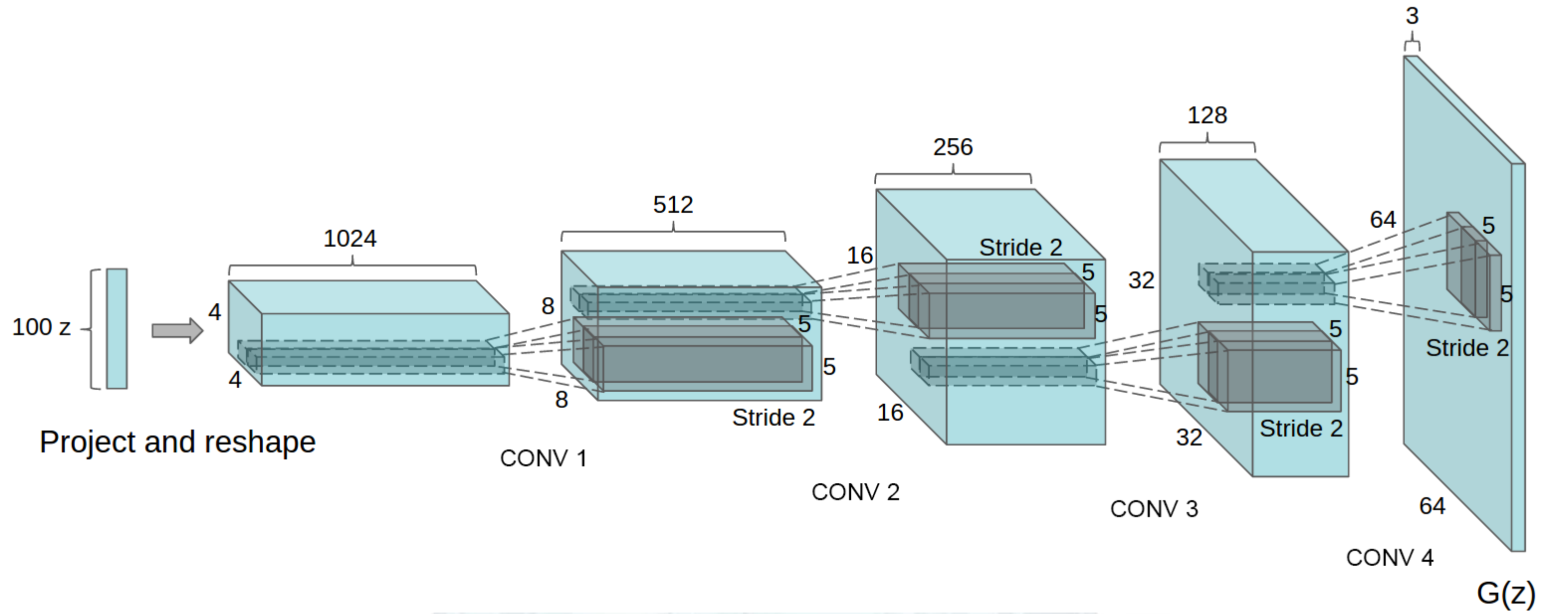
Architecture guidelines for stable Deep Convolutional GANs

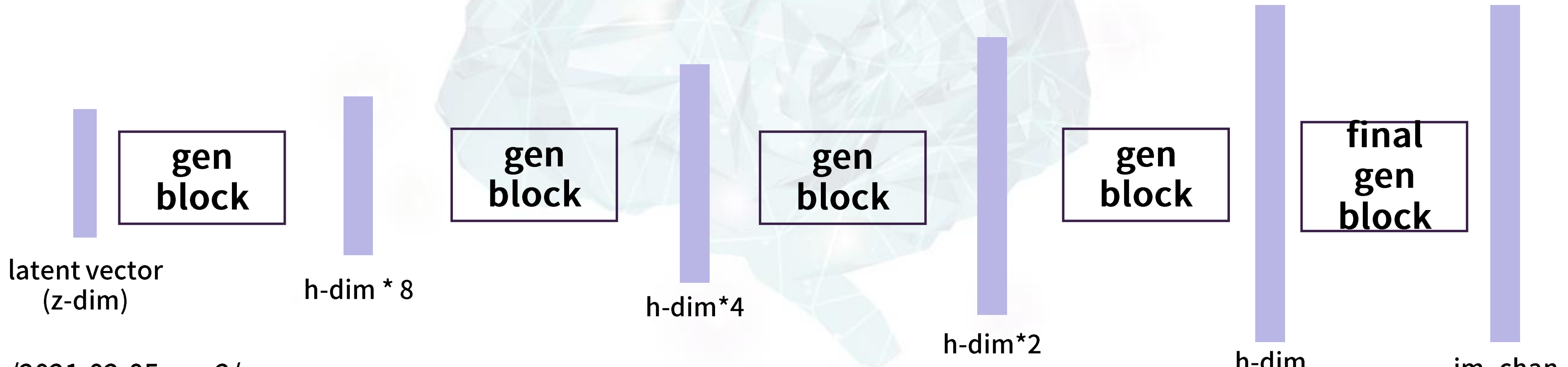
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

2. DCGAN의 구성 요소 (1): generator

2. DCGAN의 구성 요소 (1): Generator

Generator의 구조





출처: https://ydy8989.github.io/2021-02-05-gan2/

Gen block

- Parameter
: input_channels, output_channels, kernel, stride, final_layer

components for internal

transposed convolution
+ batch norm + ReLU

components for final

Transposed convolution
+ tanh

2. DCGAN의 구성 요소 (1): Generator

Gen block

```
def gen_block(self, in_channel, out_channel, kernel_size = 4, stride = 2, final_layer = False):
    if not final_layer:
        return nn.Sequential(
            nn.ConvTranspose2d(in_channel, out_channel, kernel_size = kernel_size, stride = stride),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(inplace=True),
        )
    else:
        return nn.Sequential (
            nn.ConvTranspose2d(in_channel, out_channel, kernel_size, stride),
            nn.Tanh( ),

        )
```

Gen block

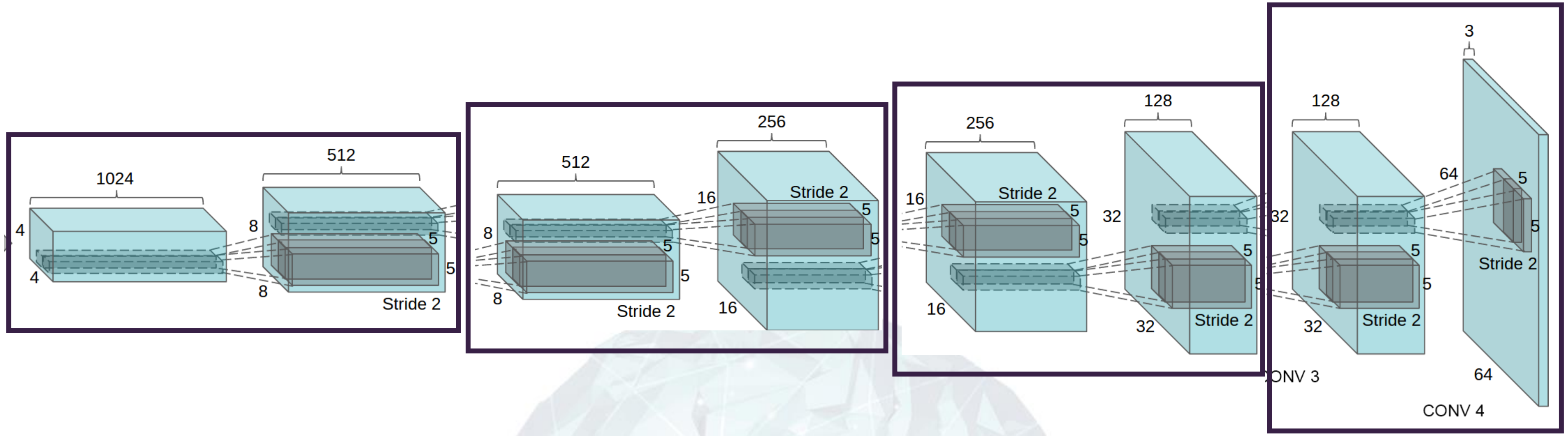
```
torch.nn.ConvTranspose2d ( in_channels,            // hidden_dim*4
                           out_channels,    // hidden_dim*2
                           kernel_size,     // 4
                           stride=1,                 // 2
                           padding=0,       // 1
                           output_padding=0,
                           groups=1,
                           bias=True,
                           dilation=1,
                           padding_mode='zeros')
```

```
H_out = (H_in - 1) * stride - 2 * padding + dilation * (kernel_size - 1) + output_padding + 1
```

```
1 -> (1 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 = 0 - 2 + 3 + 1 = 2
2 -> (2 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 = 2 - 2 + 3 + 1 = 4
4 -> (4 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 = 6 - 2 + 3 + 1 = 8
8 -> (8 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 = 14 - 2 + 3 + 1 = 16
```

2. DCGAN의 구성 요소 (1): Generator

Gen block




```
H_out = (H_in - 1) * stride - 2 * padding + dilation * (kernel_size - 1) + output_padding + 1
```

```
4   -> ( 4 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 =  6 - 2 + 3 + 1 = 8
8   -> ( 8 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 = 14 - 2 + 3 + 1 = 16
16  -> (16 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 = 30 - 2 + 3 + 1 = 32
32  -> (32 - 1) * 2 - 2 * 1 + 1 * (4 - 1) + 0 + 1 = 62 - 2 + 3 + 1 = 64
```

2. DCGAN의 구성 요소 (1): Generator

Generator

```
class Generator(nn.Module):
    def __init__(self, z_dim=10, im_chan=1, hidden_dim=64):
        super(Generator, self).__init__()
        self.z_dim = z_dim
        # Build the neural network
        self.gen = nn.Sequential(
            self.gen_block(z_dim, hidden_dim * 4),
            self.gen_block(hidden_dim * 4, hidden_dim * 2, kernel_size=4, stride=1),
            self.gen_block(hidden_dim * 2, hidden_dim),
            self.gen_block(hidden_dim, im_chan, kernel_size=4, final_layer=True),
        )
    # def gen_block

    def unsqueeze_noise(self, noise):
        return noise.view(len(noise), self.z_dim, 1, 1)

    def forward(self, noise):
        x = self.unsqueeze_noise(noise)
        return self.gen(x)
```

3. DCGAN의 구성 요소 (2): discriminator

Discriminator

Disc block

- parameter: input_channels, output_channels, kernel, stride, final_layer)

components for internal

convolution
+ batch norm
+ LeakyReLU (0.2)

components for final

convolution

Discriminator

- Disc block

```
def disc_block(self, in, out, kernel_size = 4, stride = 2, final_layer = False):
    if not final_layer:
        return nn.Sequential(
            nn.Conv2d(in, out, kernel_size, stride),
            nn.BatchNorm2d(output_channels),
            nn.LeakyReLU(0.2, inplace=True)
        )
    else: # Final Layer
        return nn.Sequential(
            nn.Conv2d(in, out, kernel_size, stride)
        )
```

Discriminator

```
class Discriminator(nn.Module):
    def __init__(self, im_chan=1, hidden_dim=16):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            self.disc_block(im_chan, hidden_dim),
            self.disc_block(hidden_dim, hidden_dim * 2),
            self.disc_block(hidden_dim * 2, 1, final_layer=True),
        )

    # def disc_block

    def forward(self, image):
        disc_pred = self.disc(image)
        return disc_pred.view(len(disc_pred), -1)
```

4. DCGAN의 구성 요소 (3): loss 함수

Discriminator loss

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\boxed{[\log D(x)]} + \mathbb{E}_{z\sim p_z(z)}\boxed{[\log(1-D(G(z)))]} \qquad (1)$$

```
disc_opt.zero_grad ()
latent = get_noise ( cur_batch_size, z_dim, device=device )   # z
fake = gen (latent)                                                                      # G(z)
disc_fake_pred = disc (fake.detach())                                              # D(G(z))
disc_fake_loss = criterion (disc_fake_pred, torch.zeros_like(disc_fake_pred))
```

Generator loss

```
disc_real_pred = disc (real)                                                             # D(x)
disc_real_loss = criterion (disc_real_pred, torch.ones_like(disc_real_pred))
disc_loss = (disc_fake_loss + disc_real_loss) / 2
```

Discriminator loss

4. DCGAN의 구성 요소 (3): loss 함수

Generator loss

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}\boxed{[\log(1-D(G(z)))]} \qquad (1)$$

```
gen_opt.zero_grad ()
        latent_2 = get_noise ( cur_batch_size, z_dim, device=device )# z
        fake_2 = gen (latent_2)                                                # G(z)
        disc_fake_pred = disc (fake_2.detach())                                # D(G(z))
        gen_loss = criterion (disc_fake_pred, torch.ones_like(disc_fake_pred))
        gen_loss.backward()
        gen_opt.step()
```

Generator loss

5. DCGAN의 구성 요소 (4): 훈련

Training process

```
n_epochs = 50
cur_step = 0
mean_generator_loss = 0
mean_discriminator_loss = 0
for epoch in range(n_epochs):
    for real, _ in tqdm(dataloader):
        # Update discriminator

        # Update generator

        # Visualize the results

        cur_step += 1
```

Update discriminator

```
disc_opt.zeo_grad ()

# Get disc loss
disc_real_pred = disc (real)                                            # D(x)
disc_real_loss = criterion (disc_real_pred, torch.ones_like(disc_real_pred))

# Get gen loss
latent = get_noise ( cur_batch_size, z_dim, device=device )       # z
fake = gen (latent)                                                     # G(z)
disc_fake_pred = disc (fake.detach())                                   # D(G(z))
disc_fake_loss = criterion (disc_fake_pred, torch.zeros_like(disc_fake_pred))

disc_loss = (disc_fake_loss + disc_real_loss) / 2

# Update gradients
disc_loss.backward(retain_graph=True)

# Update optimizer
disc_opt.step()
```

Update generator

```
gen_opt.zeo_grad ()

# Get gen loss
fake_noise_2 = get_noise(cur_batch_size, z_dim, device=device)
fake_2 = gen(fake_noise_2)
disc_fake_pred = disc(fake_2)
gen_loss = criterion(disc_fake_pred, torch.ones_like(disc_fake_pred))

# Update gradients
gen_loss.backward()

# Update optimizer
gen_opt.step()
```