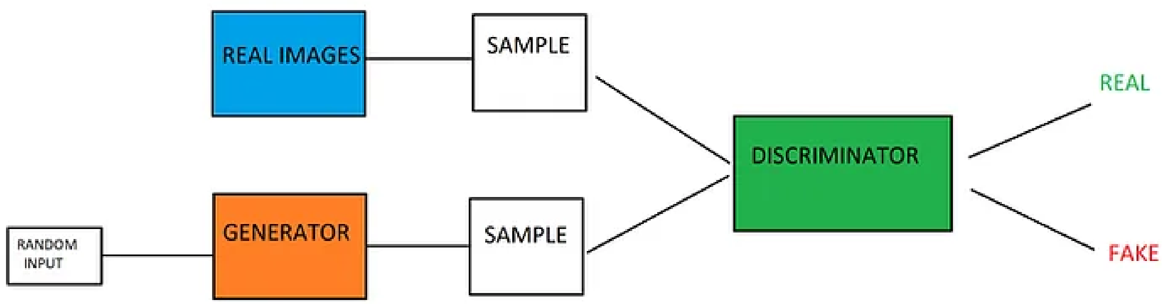
GAN(Generative Adversarial Network)

Written by: 류채은 Chaeeun Ryu (superbunny38 *at* gmail *dot* com)

Generative: Generates data (cf. Autoencoder)

Adversarial: Generator vs. Discriminator

Network: Neural Network



G: **Generator**: Generates fake data from the input z

D: **Discriminator**: Discriminates the fake data and real data, respectively.

z: random noise

$x \sim P_{data}(x)$: sample x from data

$P_{data}(x)$: real data distribution

Objective Function

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

- in the aspect of **Discriminator** only:

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

  $0 \leq D(x) \leq 1$

  → aims to maximize $logD(x)$

  → aims to maximize $log(1-D(G(z))$

  → $D(x) \mapsto 1$

  → $D(G(z)) \mapsto 0$

- int the aspect of **Generator** only:

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

  → aims to minimize $log(1-D(G(z))$

  → aims to maximize $D(G(z))$

  → $D(G(z)) \mapsto$ 1

When the objective function is optimized the fullest for discriminator,

$P_{data}(x) == P_g(x)$

Converting Objective Function to Loss Function

(to minimize)

- in the aspect of **Discriminator** only:

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

  - in case of real data input

    $\max_D V(G,D) = E_{x\sim P_{data}(x)}[logD(x)]$

    → $\min_D V(D,G)$ = $-E_{x\sim P_{data}(x)}[logD(x)]$

- in the aspect of Generator only:

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

  - in case of fake data input

$$\min_{G} V(G, D) = E_{z \sim P_z(x)}[log(1 - D(G(z))]$$
$$\rightarrow \max_{G} E_{z \sim P_z(x)}[log(D(G(z))]$$

Generator Code

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.layer1 = nn.Sequential(OrderedDict([
            ('fc1', nn.Linear(z_size, middle_size)),
            ('bn1', nn.BatchNorm1d(middle_size)),
            ('act1', nn.ReLU()),
            ]))
        self.layer2 = nn.Sequential(OrderedDict([
            ('fc2', nn.Linear(z_size, middle_size)),
            #('bn1', nn.BatchNorm1d(middle_size)),
            ('tanh', nn.Tanh()),
            ]))
    def forward(self, z):
        out = self.layer1(z)
        out = self.layer2(out)
        out = out.view(batch_size, 1, 28,28)
        return out
```

Discriminator Code

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.layer1 = nn.Sequential(OrderedDict([
            ('fc1', nn.Linear(784, middle_size)),
            #('bn1', nn.BatchNorm1d(middle_size)),
            ('act1',nn.LeakyReLU()),
            ]))
        self.layer2 = nn.Sequential(OrderedDict([
            ('fc2', nn.Linear(784, middle_size)),
            ('bn2', nn.BatchNorm1d(middle_size)),
            ('act2',nn.Sigmoid()),#binary classification
            ]))

    def forwarrd(self, x):
        out = x.view(batch_size, -1)
        out = self.layer1(out)
        out = self.layer2(out)
        return out
```

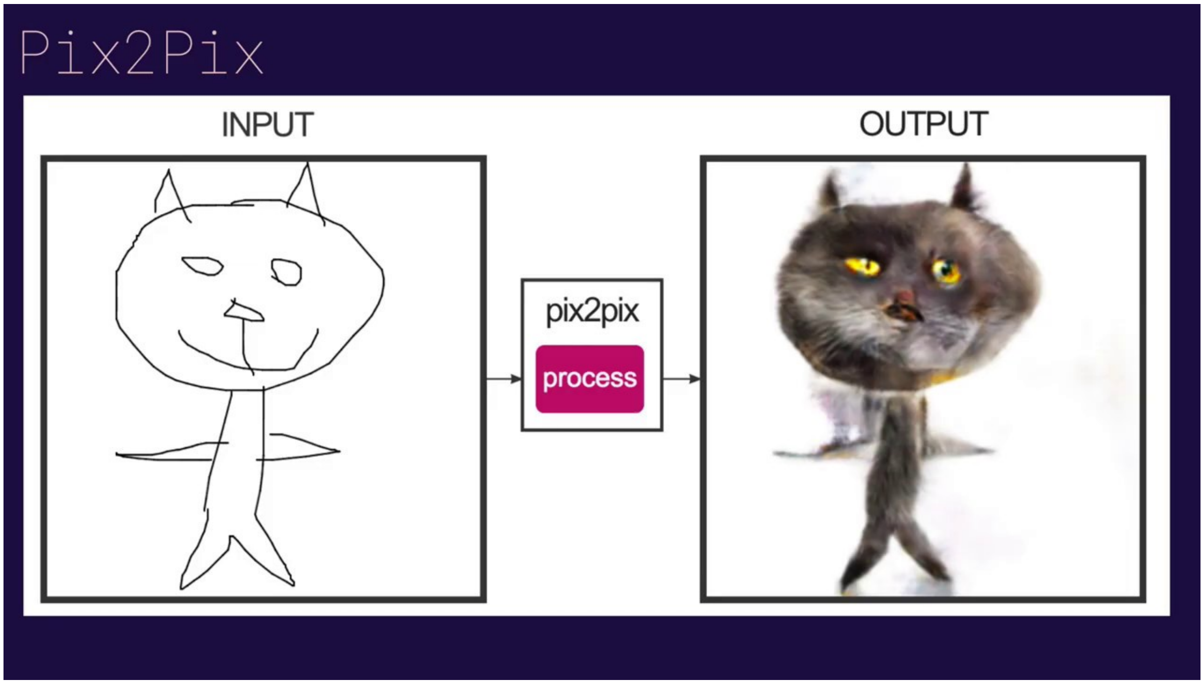
Loss function: L2 loss (LSGAN, Least Squares GAN)

TRAINING

```
for i in range(epoch):
    for j, (image, label) in enumerate(train_loader):
        #구분자 학습
        dis_optim.zero_grad()
        z = init.normal_(torch.Tensor(batch_size, z_size), mean = 0, std = 0.1)
        gen_fake = generator.forward(z)
        dis_fake = discriminator.forward(gen_fake)

        dis_real = discriminator.forward(image)
        dis_loss = torch.sum(loss_func(dis_fake, zeros_label))+ torch.sum(loss_func(dis_real, ones_label))
        dis_loss.backward(retain_graph = True)
        dis_optim.stem()

        #생성자 학습
        gen_optim.zero_grad()

        z = init.normal_(torch.Tensor(batch_size, z_size), mean=0, std = 0.1)
        gen_fake = generator.forward(z)
        dis_fake = discriminator.forward(gen_fake)
        gen_loss = torch.sum(loss_func(dis_fake, ones_label))#fake classified as real
        gen_loss.backward()
        gen_optim.step()
```

Other Concepts

- DCGAN: Deep Convolutional GAN
- Latent Space Interpolation: A method to explore the latent space
- cGAN: Conditional GAN
- SRGAN: Super-resolution GAN: 낮은 화질→고화질
- text to image synthesis
- Pix2Pix: image to image translation



- Cycle GAN
- Disco GAN