GANs in action

CGAN, CycleGAN

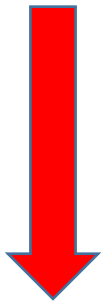
한병찬

Contents

1. CGAN
2. CycleGAN

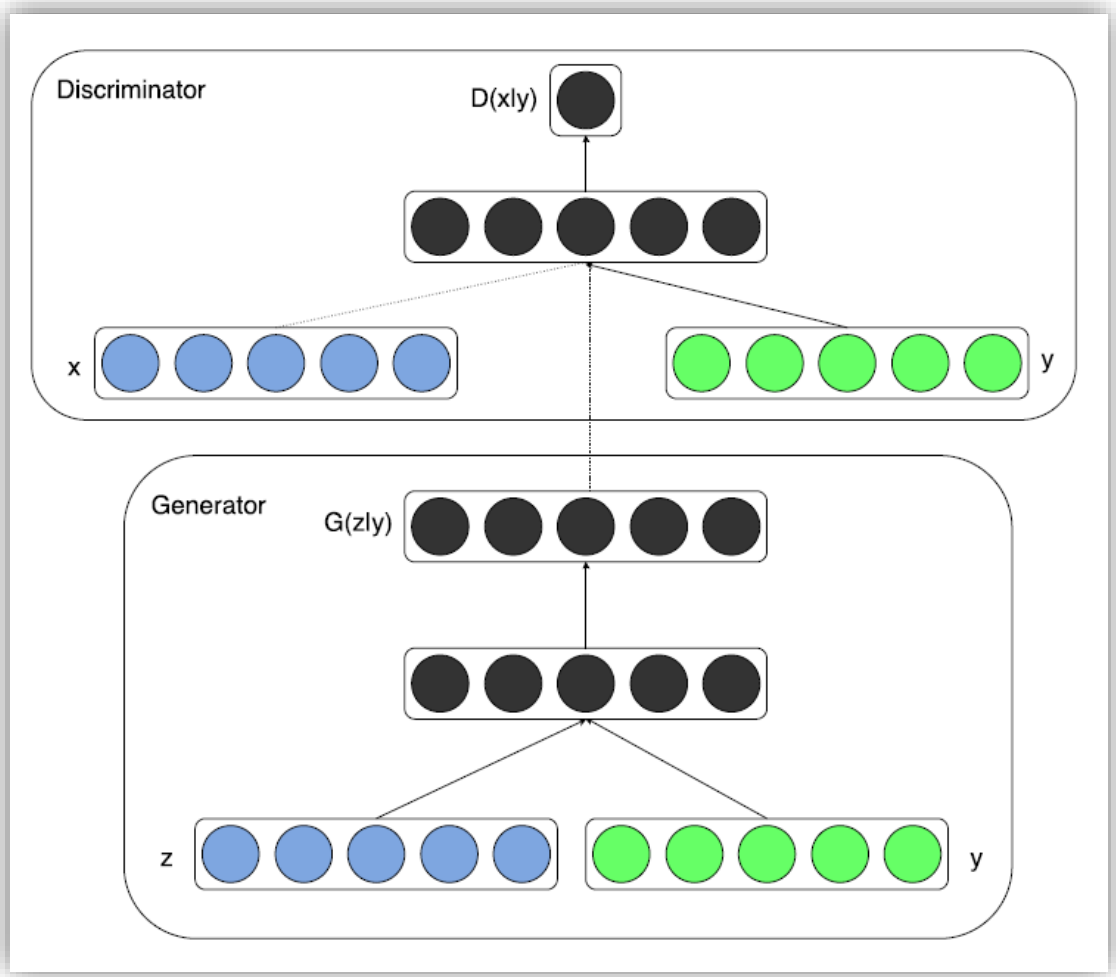
CGAN

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



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

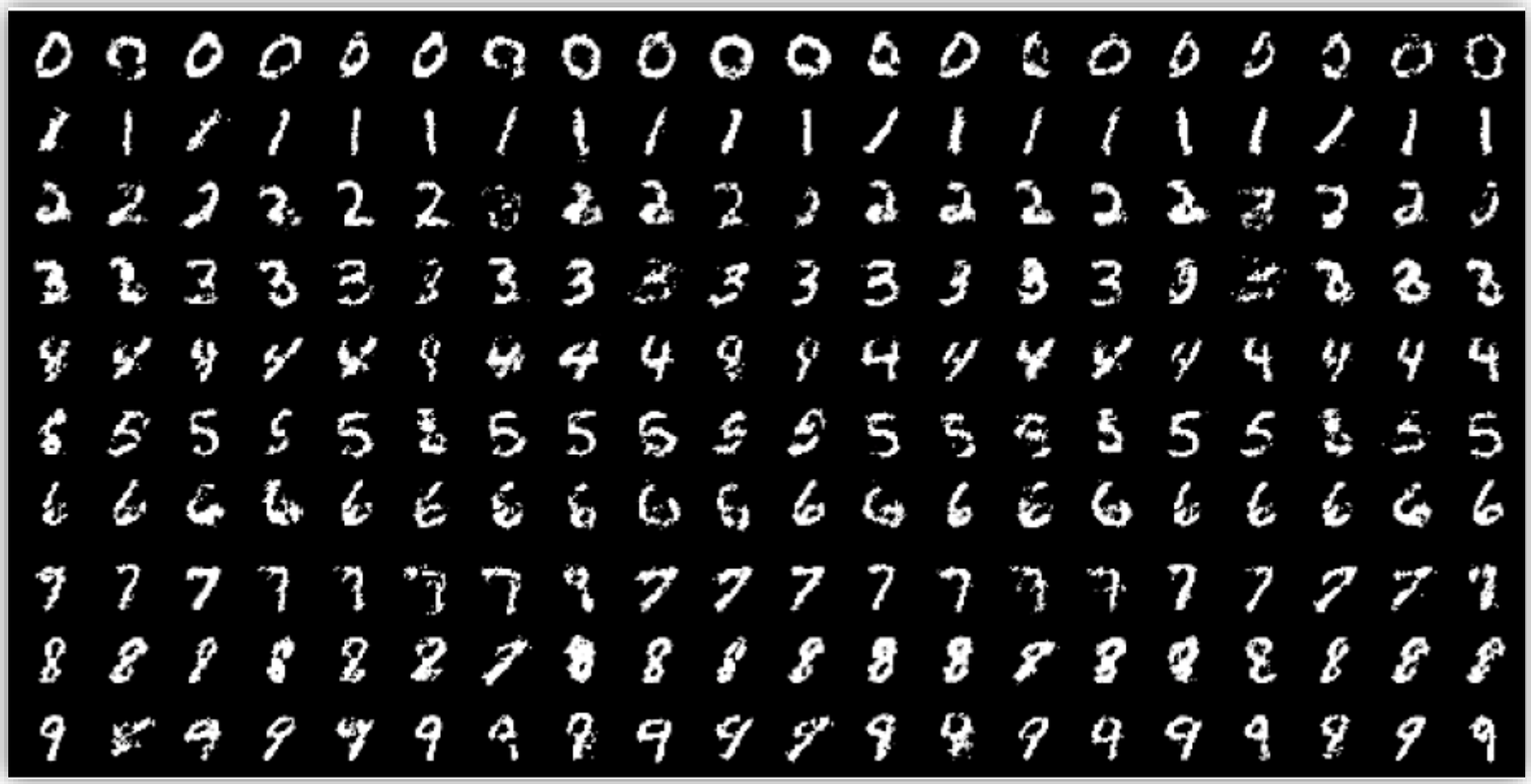
CGAN



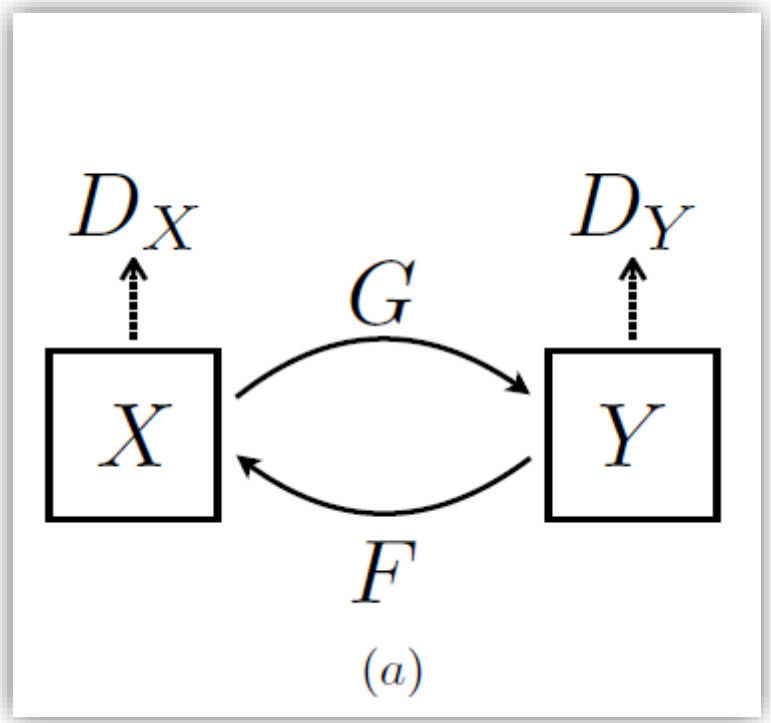

```
label_embedding = Embedding(num_classes, np.prod(img_shape), input_length=1)(label)
label_embedding = Flatten()(label_embedding)
label_embedding = Reshape(img_shape)(label_embedding)
concatenated = Concatenate(axis=-1)([img, label_embedding])
```

```
label_embedding = Embedding(num_classes, z_dim, input_length=1)(label)
label_embedding = Flatten()(label_embedding)
joined_representation = Multiply()([z, label_embedding])
```

CGAN

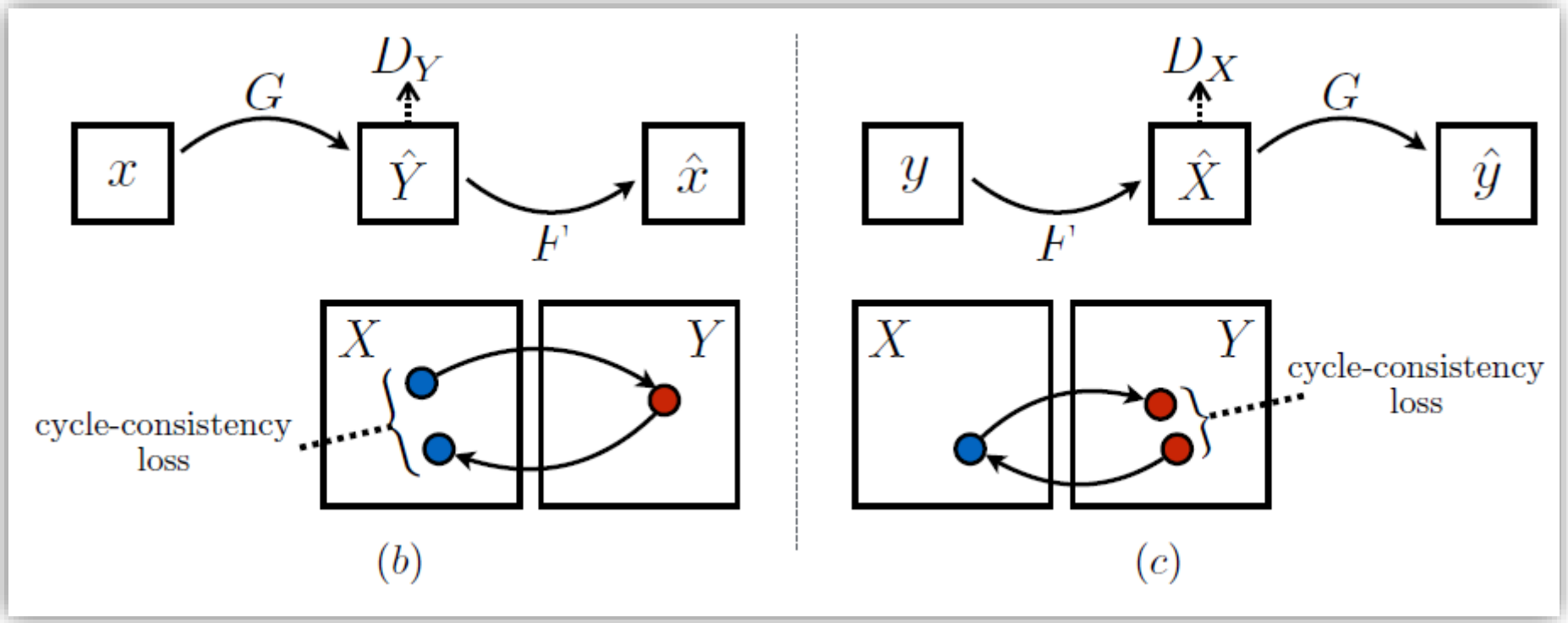


CycleGAN



$G : X \rightarrow Y$ and $F : Y \rightarrow X$

CycleGAN



$$x \to G(x) \to F(G(x)) \approx x$$

$$y \to F(y) \to G(F(y)) \approx y$$

CycleGAN

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

```
criterion_GAN = torch.nn.MSELoss()
criterion_cycle = torch.nn.L1Loss()
```

```
criterion_identity = torch.nn.L1Loss()
```

$$\mathcal{L}_{\text{identity}}(G, F) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(y) - y\|_1] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(x) - x\|_1]$$

(그림을 사진으로 변경할때 사용)

CycleGAN

```python
class GeneratorResNet(nn.Module):
    def __init__(self, input_shape, num_residual_blocks):
        super(GeneratorResNet, self).__init__()

        channels = input_shape[0]

        # Initial convolution block
        out_features = 64
        model = [
            nn.ReflectionPad2d(channels),
            nn.Conv2d(channels, out_features, 7),
            nn.InstanceNorm2d(out_features),
            nn.ReLU(inplace=True),
        ]
```

```python
class Discriminator(nn.Module):
    def __init__(self, input_shape):
        super(Discriminator, self).__init__()

        channels, height, width = input_shape

        self.output_shape = (1, height // 2 ** 4, width // 2 ** 4)

        def discriminator_block(in_filters, out_filters, normalize=True):
            layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
            if normalize:
                layers.append(nn.InstanceNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *discriminator_block(channels, 64, normalize=False),
            *discriminator_block(64, 128),
            *discriminator_block(128, 256),
            *discriminator_block(256, 512),
            nn.ZeroPad2d((1, 0, 1, 0)),
            nn.Conv2d(512, 1, 4, padding=1)
        )
```

CycleGAN

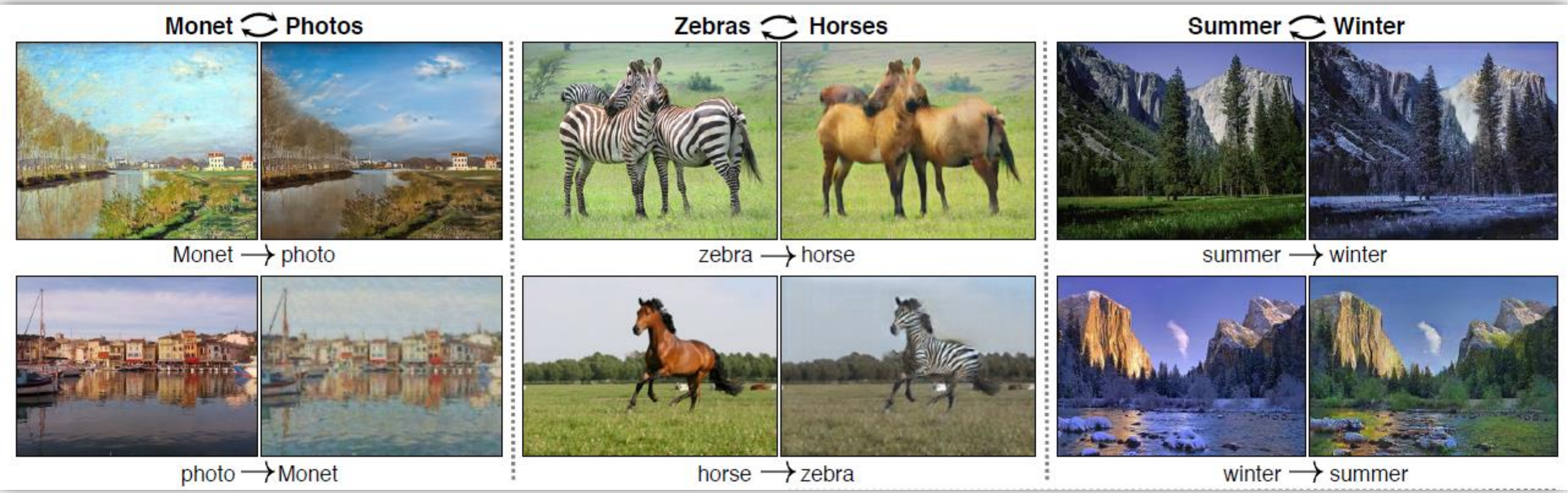
Replay Buffer

```python
class ReplayBuffer:
    def __init__(self, max_size=50):
        assert max_size > 0, "Empty buffer or trying to create a black hole. Be careful."
        self.max_size = max_size
        self.data = []

    def push_and_pop(self, data):
        to_return = []
        for element in data.data:
            element = torch.unsqueeze(element, 0)
            if len(self.data) < self.max_size:
                self.data.append(element)
                to_return.append(element)
            else:
                if random.uniform(0, 1) > 0.5:
                    i = random.randint(0, self.max_size - 1)
                    to_return.append(self.data[i].clone())
                    self.data[i] = element
                else:
                    to_return.append(element)
        return Variable(torch.cat(to_return))
```

```python
fake_A_ = fake_A_buffer.push_and_pop(fake_A)
loss_fake = criterion_GAN(D_A(fake_A_.detach()), fake)
```

CycleGAN


Monet ⇄ Photos
Zebras ⇄ Horses
Summer ⇄ Winter
Monet → photo
zebra → horse
summer → winter
photo → Monet
horse → zebra
winter → summer

CycleGAN