AI VIETNAM
All-in-One Course

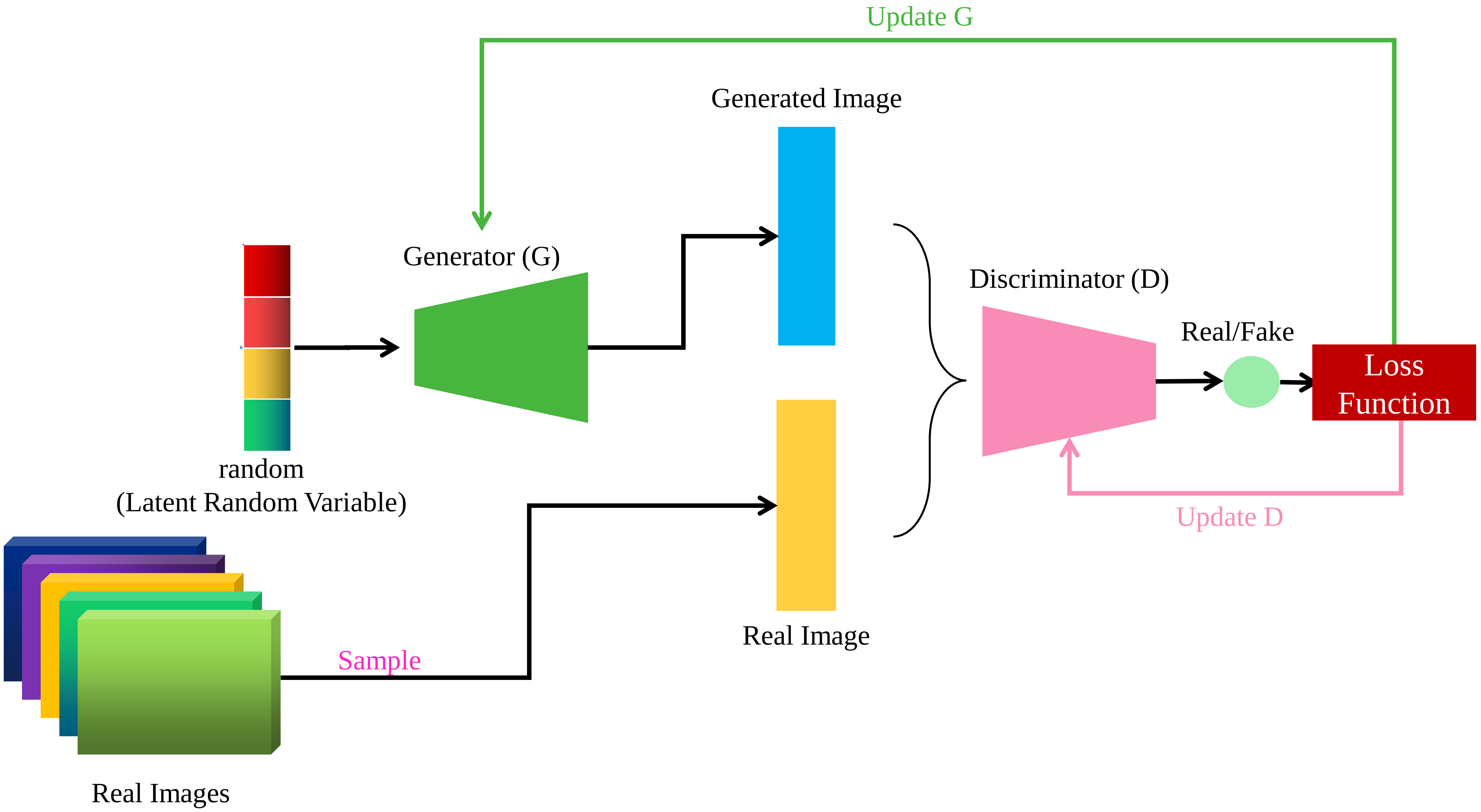
Generative Adversarial Networks

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2024

Objectives

- ✓ Study Generative Adversarial Network
- ✓ Study Deep Convolutional GAN



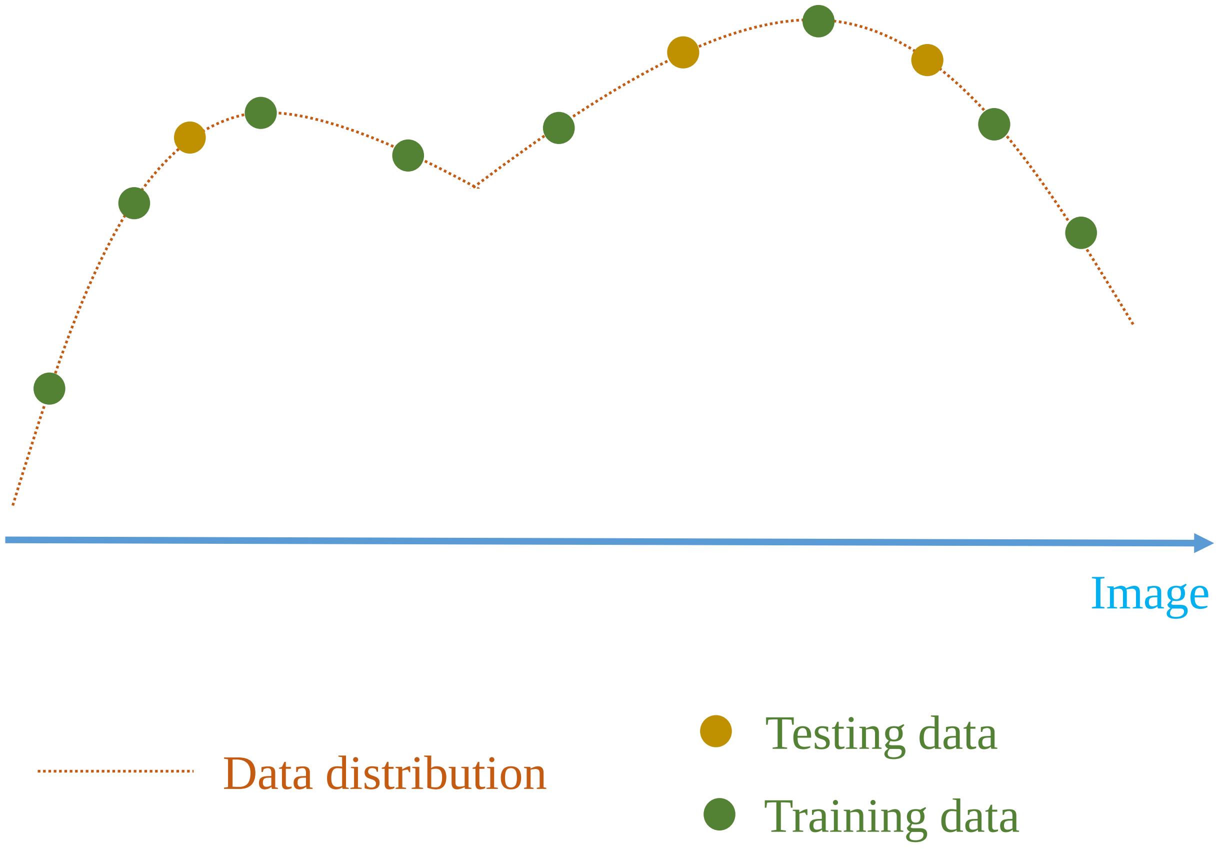
Outline

- Introduction
- GAN
- DCGAN
- Implementation

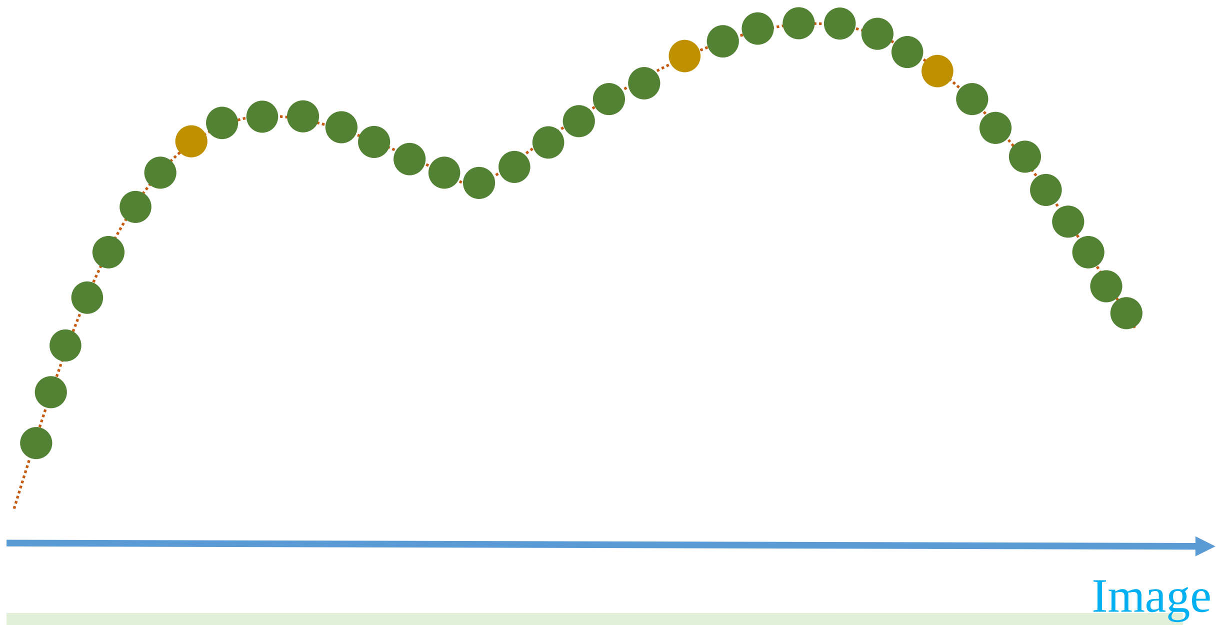
Introduction

❖ Introduction

A normal case



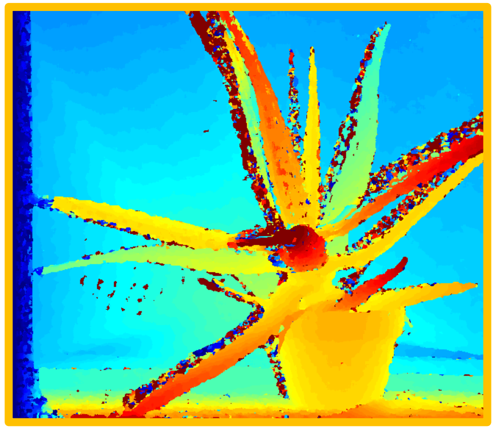
A perfect case: Have unlimited training

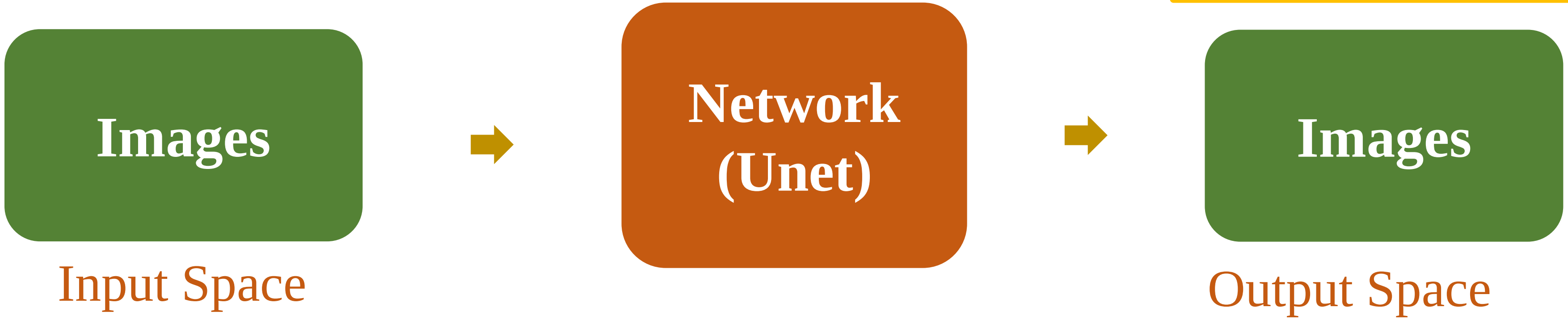


Training data cover the whole distribution

But, impractical!!!

Introduction





Images

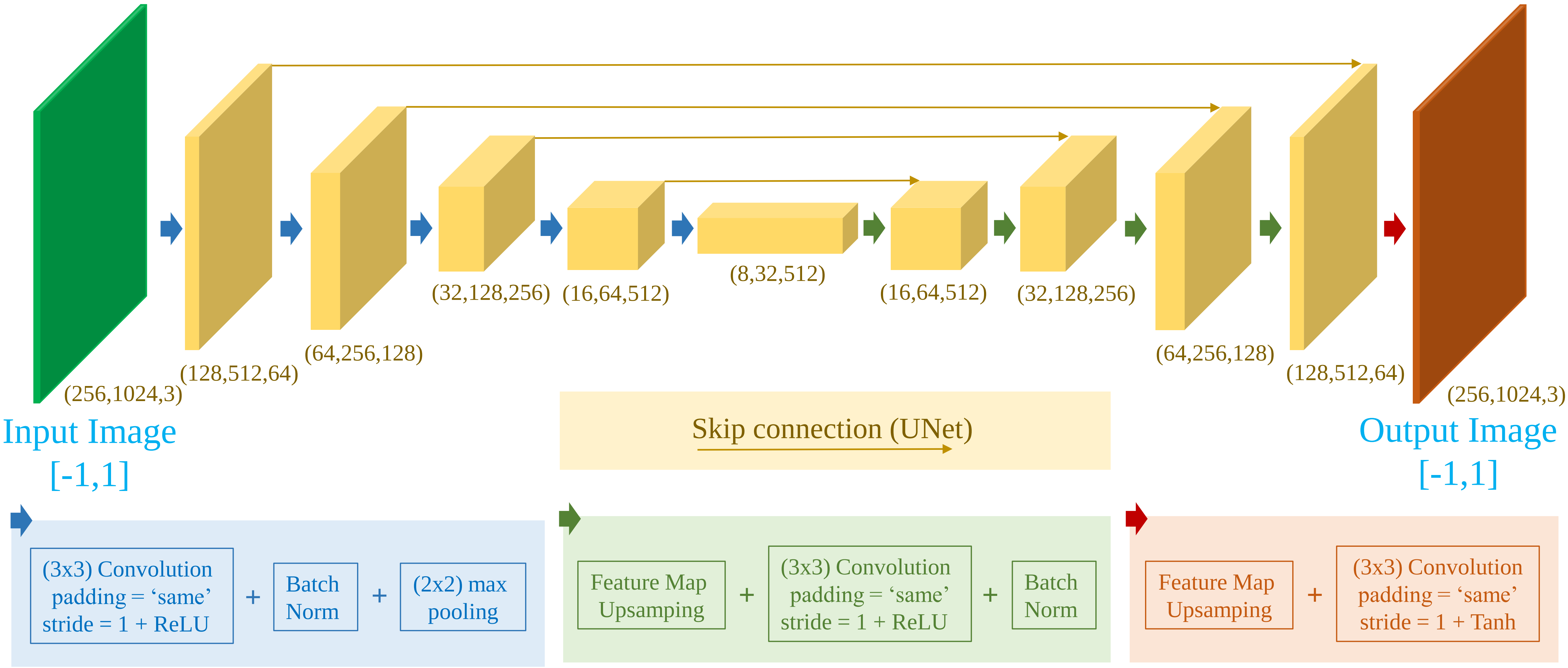
Network (Unet)

Images

Input Space

Output Space

Introduction



(256,1024,3)
(128,512,64)
(64,256,128)
(32,128,256)
(16,64,512)
(8,32,512)
(16,64,512)
(32,128,256)
(64,256,128)
(128,512,64)
(256,1024,3)

Input Image
[-1,1]

Skip connection (UNet)

Output Image
[-1,1]

(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

Feature Map Upsampling + (3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm

Feature Map Upsampling + (3x3) Convolution padding = 'same' stride = 1 + Tanh

Introduction

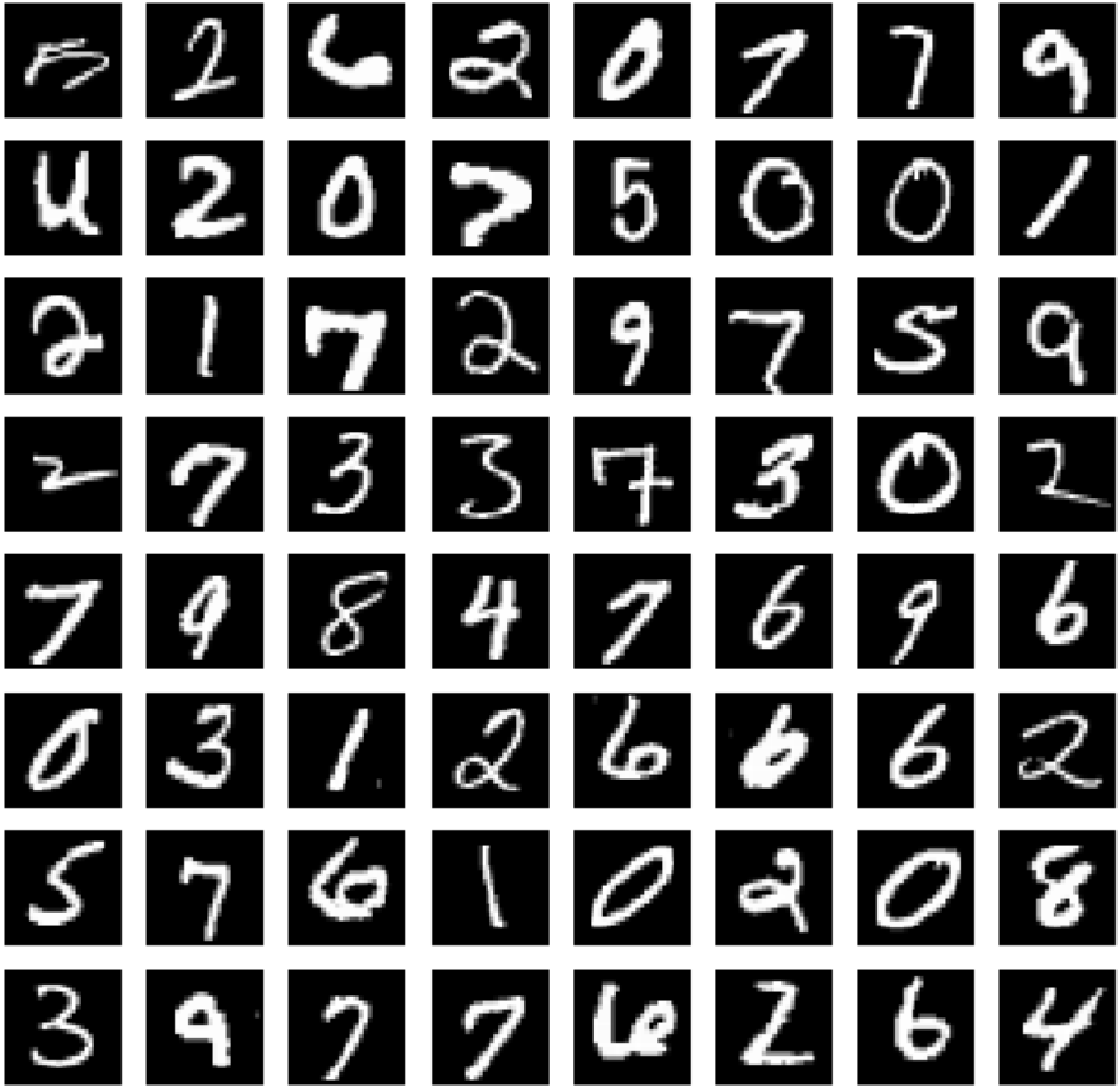
MNIST dataset

Grayscale images

Resolution=28x28

Training set: 60000 samples

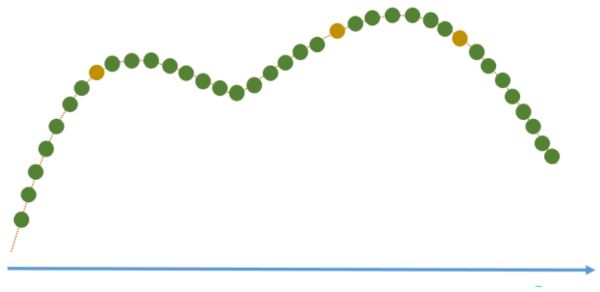
Testing set: 10000 samples

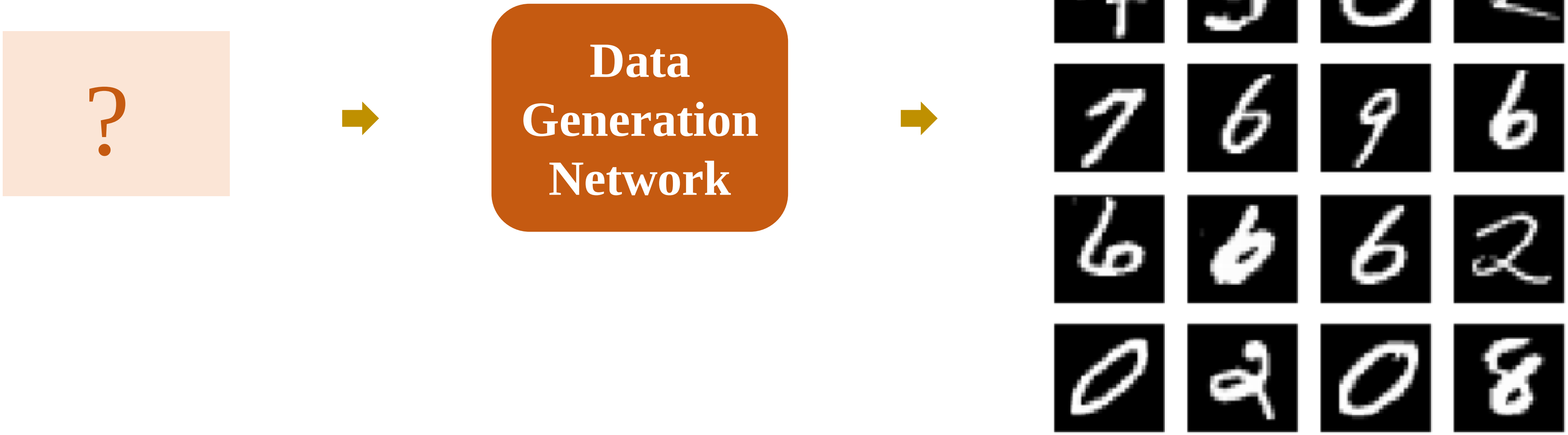


Introduction

❖ Input?

A perfect case: Have unlimited training





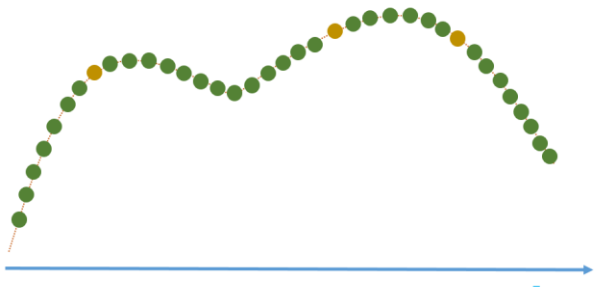
?

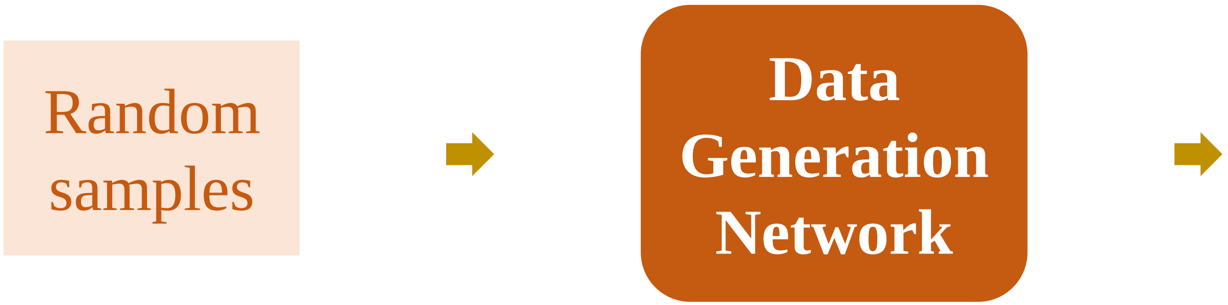
Data Generation Network

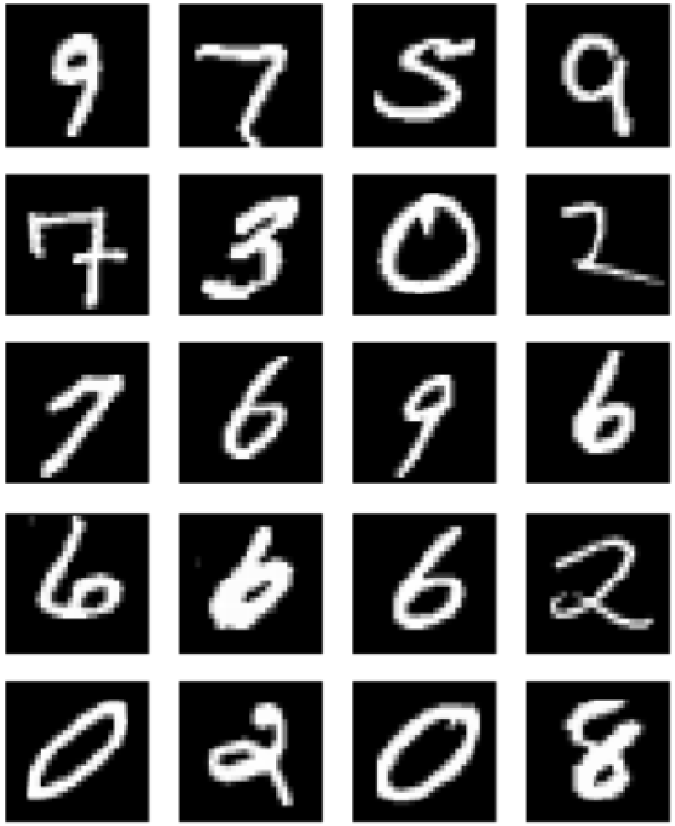
Generative Adversarial Networks

❖ Input?

A perfect case: Have unlimited training



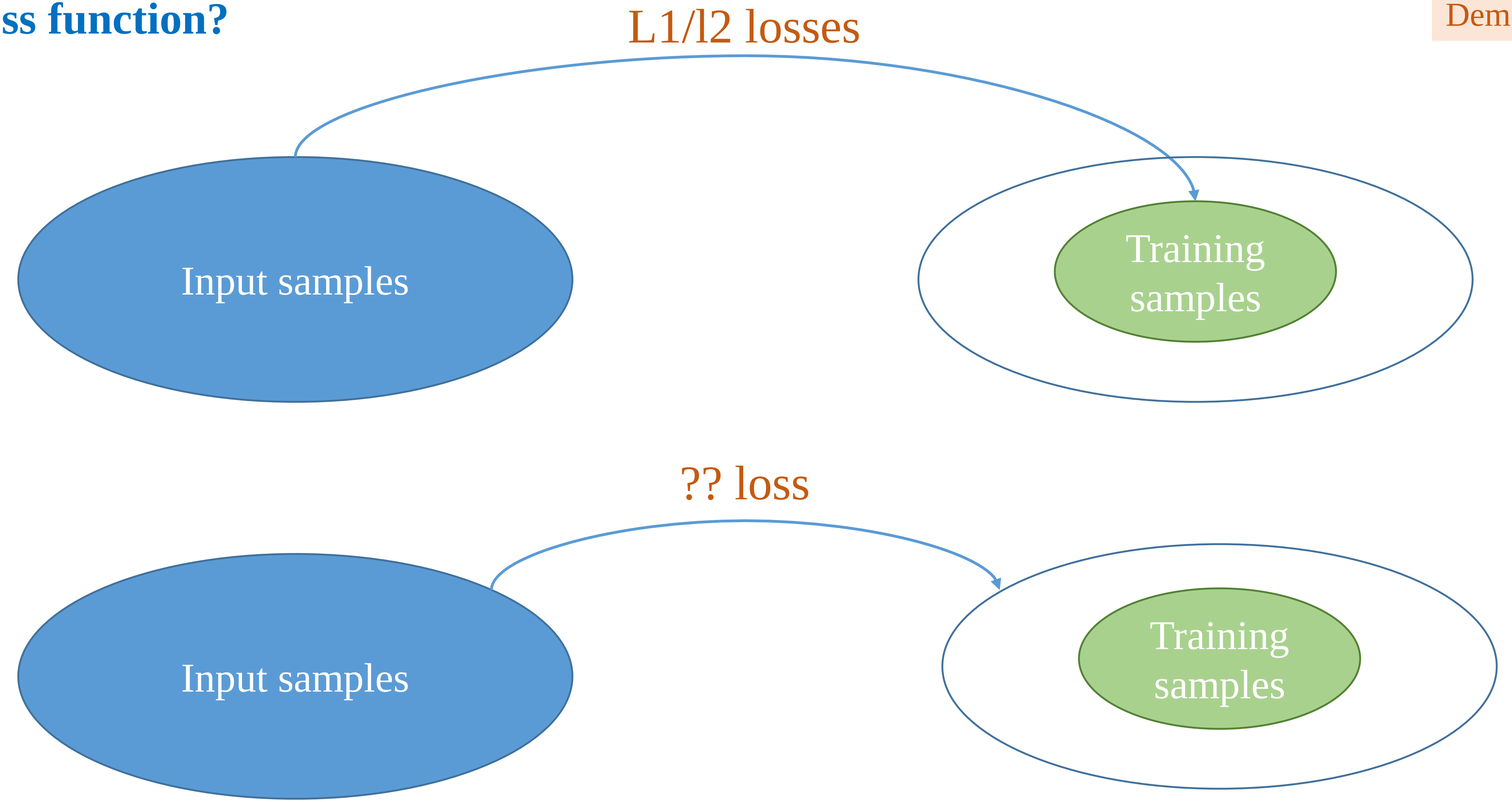




Generative Adversarial Networks

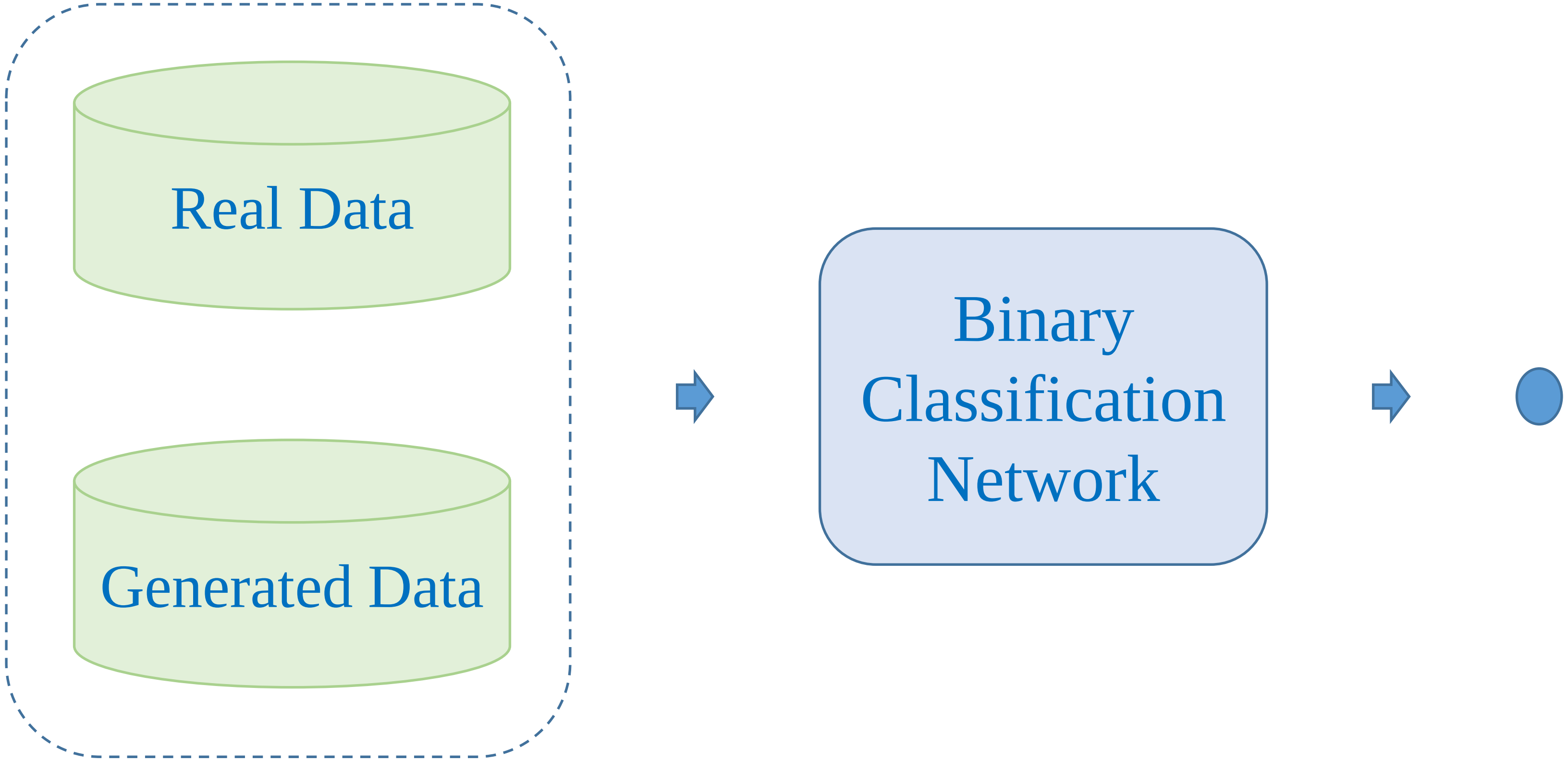
❖ Loss function?

Demo



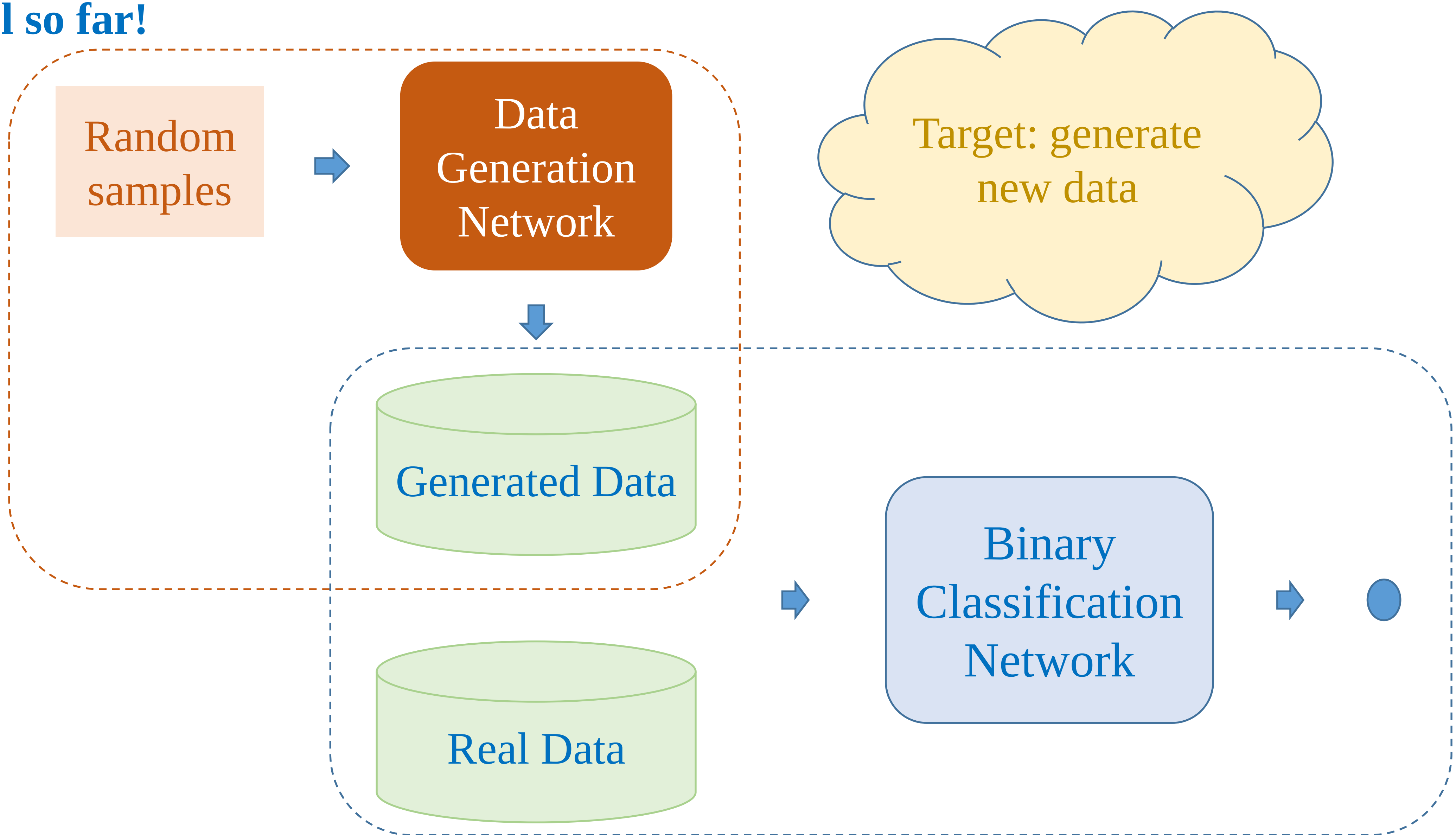
Generative Adversarial Networks

❖ Loss function: A network



Generative Adversarial Networks

Model so far!



Random samples

Data Generation Network

Target: generate new data

Generated Data

Real Data

Binary Classification Network

Outline

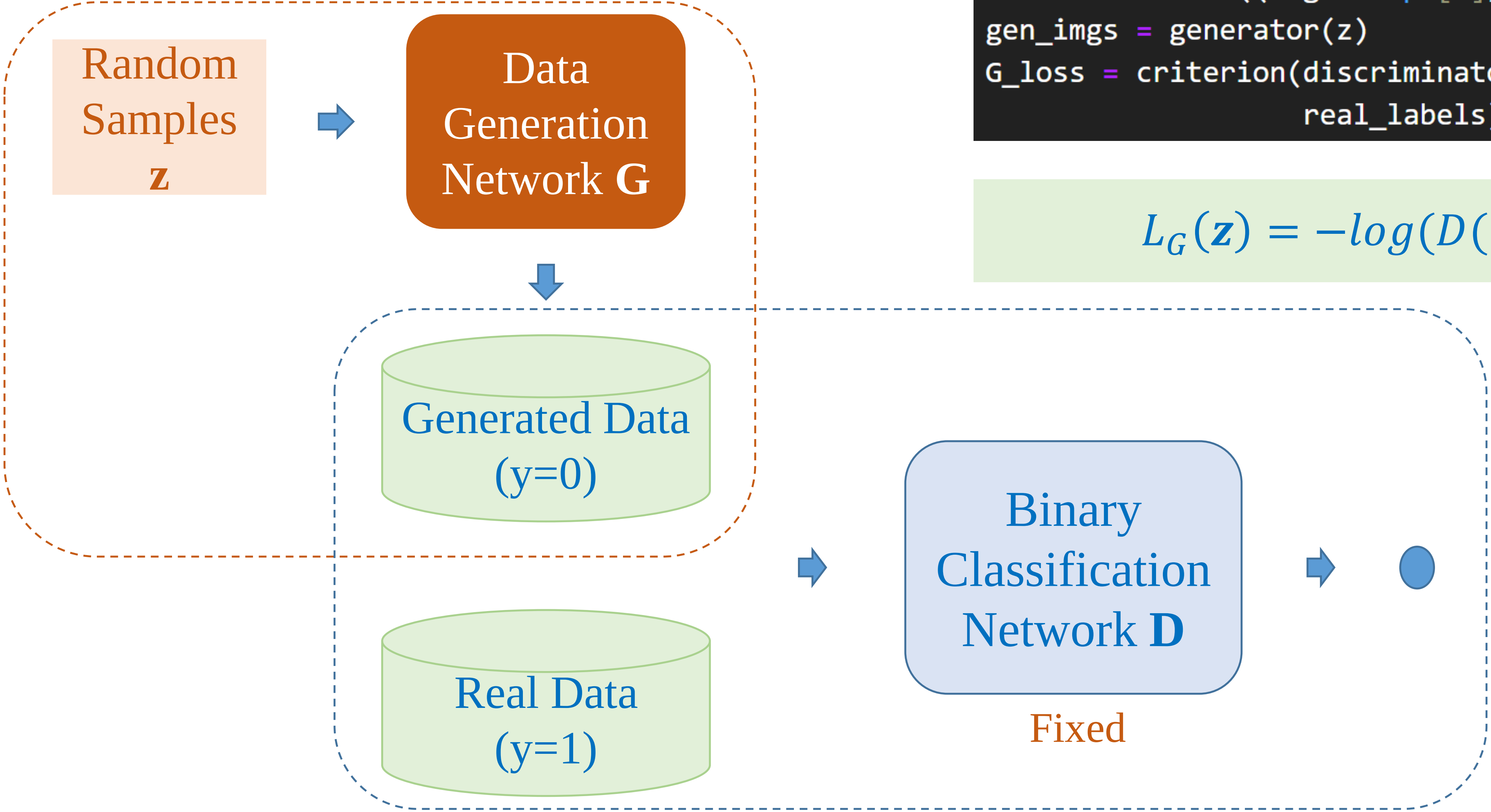
- ➢ Introduction
- ➢ GAN
- ➢ DCGAN
- ➢ Implementation

Generator Loss

```
criterion = nn.BCELoss()
real_labels = torch.ones(imgs.shape[0], 1)

z = torch.randn((imgs.shape[0], latent_dim))
gen_imgs = generator(z)
G_loss = criterion(discriminator(gen_imgs),
                   real_labels)
```

$$L_G(\boldsymbol{z}) = -log(D(G(\boldsymbol{z})))$$

Model so far!



Random Samples **z**

Data Generation Network **G**

Generated Data (y=0)

Real Data (y=1)

Binary Classification Network **D**

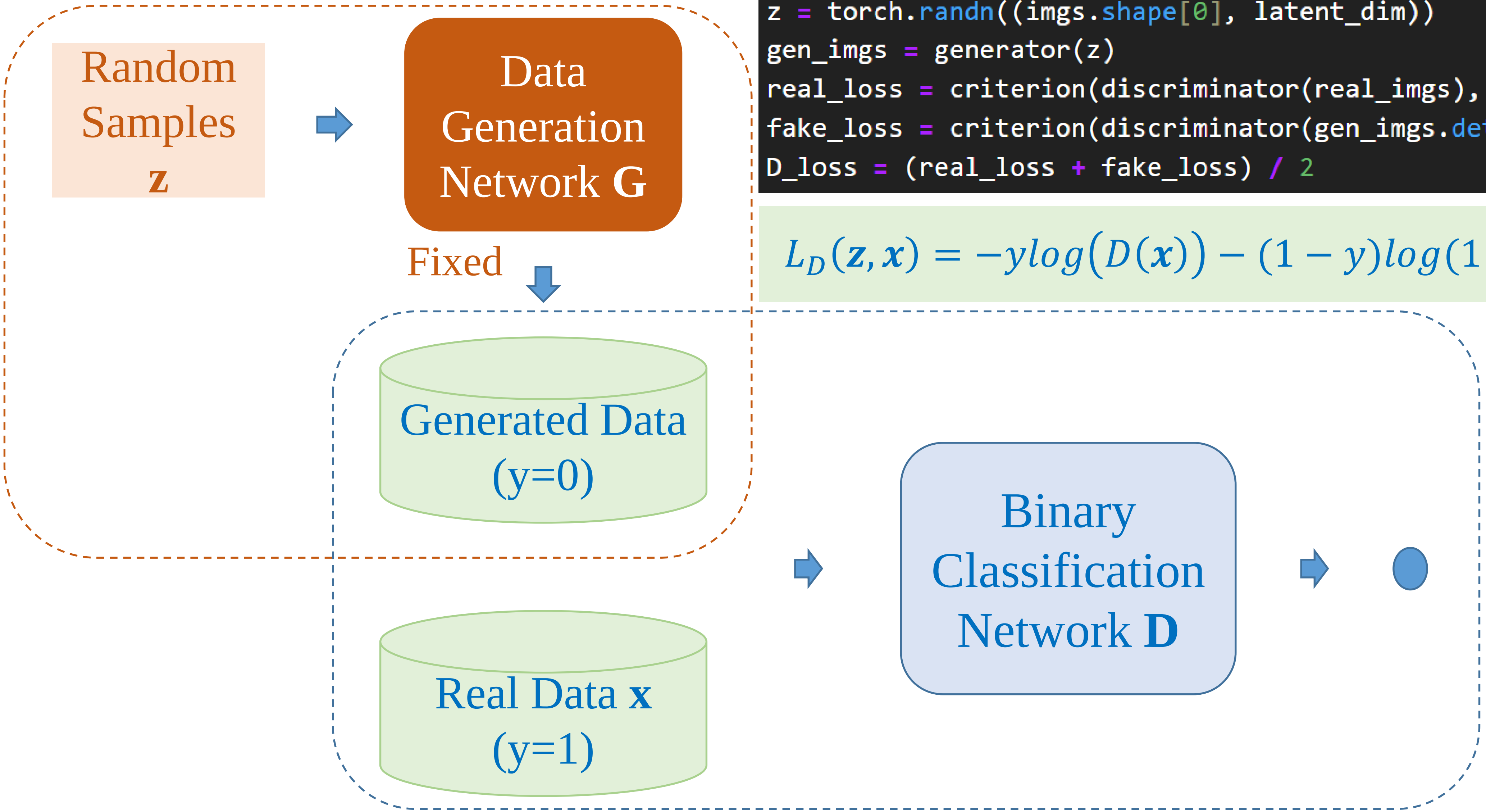
Fixed

Discriminator Loss

```
criterion = nn.BCELoss()
real_labels = torch.ones(imgs.shape[0], 1)
fake_labels = torch.zeros(imgs.shape[0], 1)

z = torch.randn((imgs.shape[0], latent_dim))
gen_imgs = generator(z)
real_loss = criterion(discriminator(real_imgs), real_labels)
fake_loss = criterion(discriminator(gen_imgs.detach()), fake)
D_loss = (real_loss + fake_loss) / 2
```

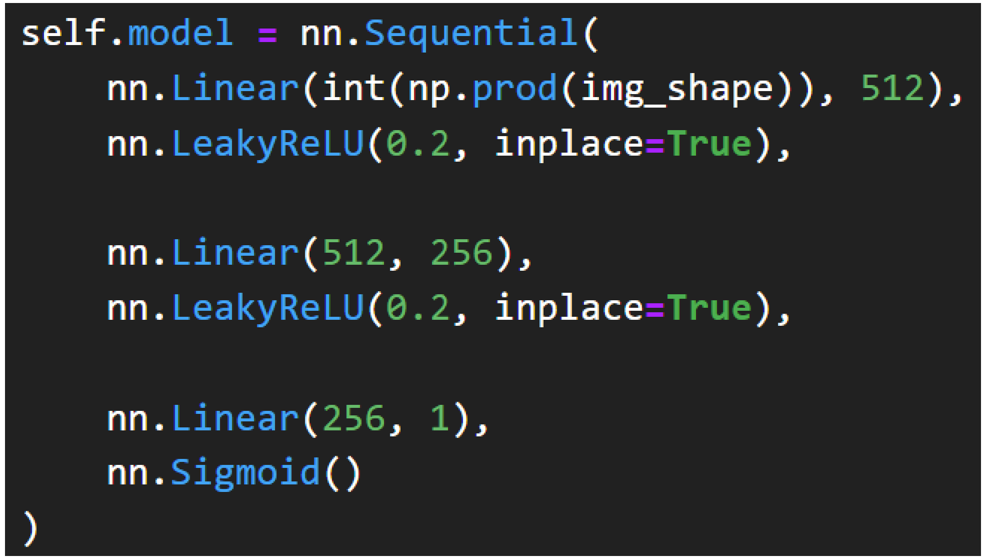
$$L_D(\boldsymbol{z},\boldsymbol{x}) = -y\log\big(D(\boldsymbol{x})\big) - (1-y)\log(1-D(G(\boldsymbol{z})))$$

Model so far!

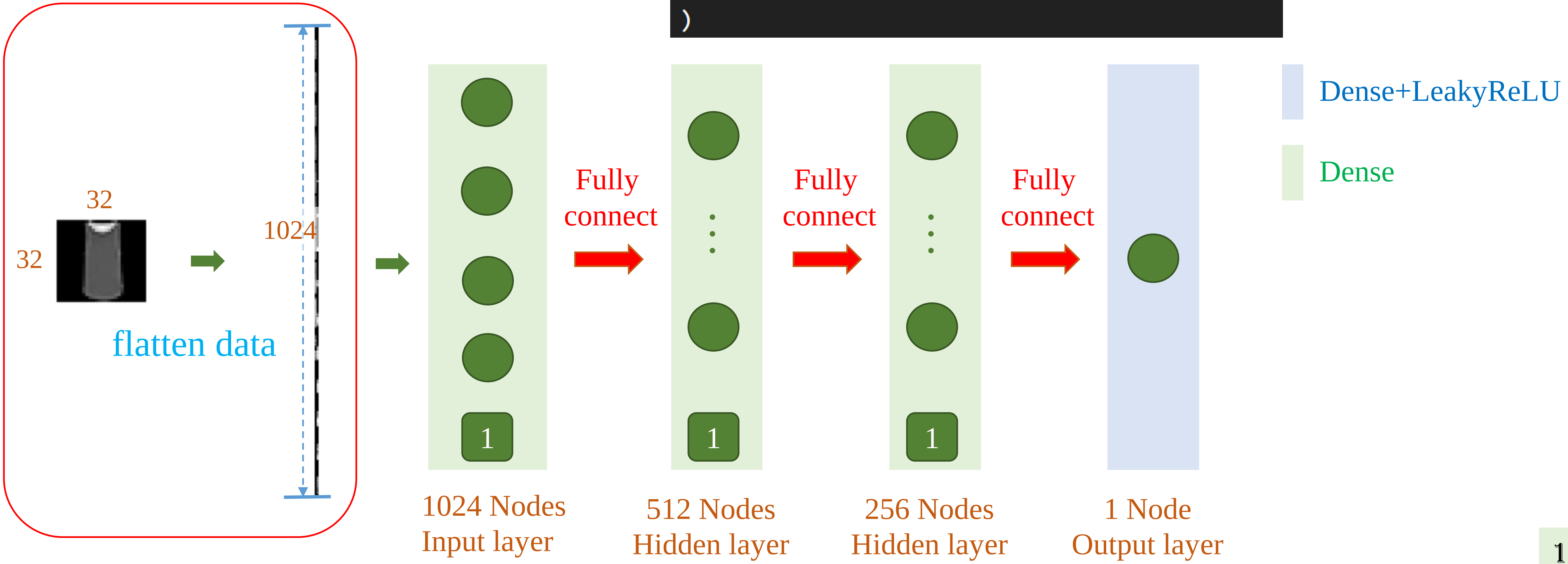


Standard GAN

❖ Architecture

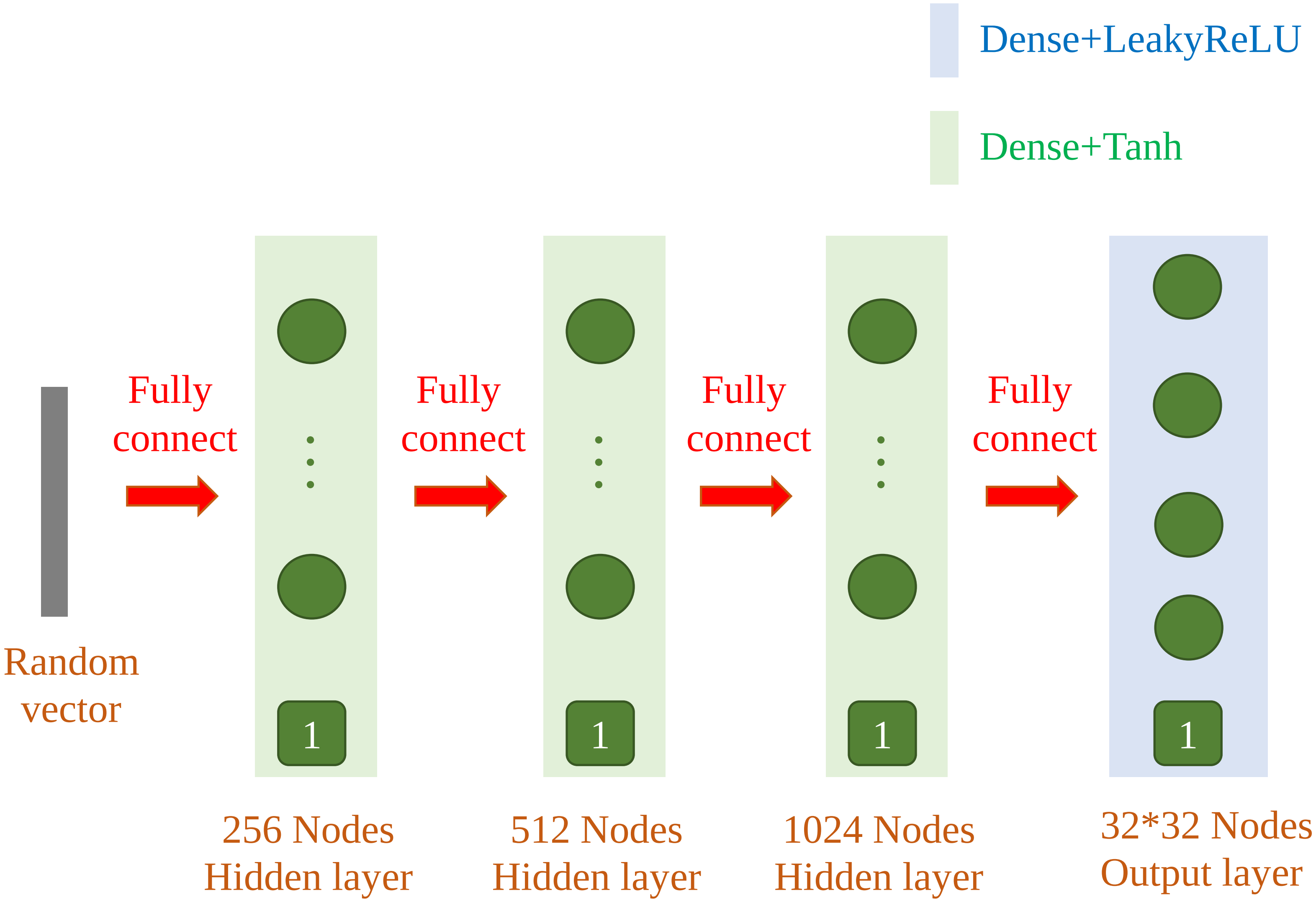


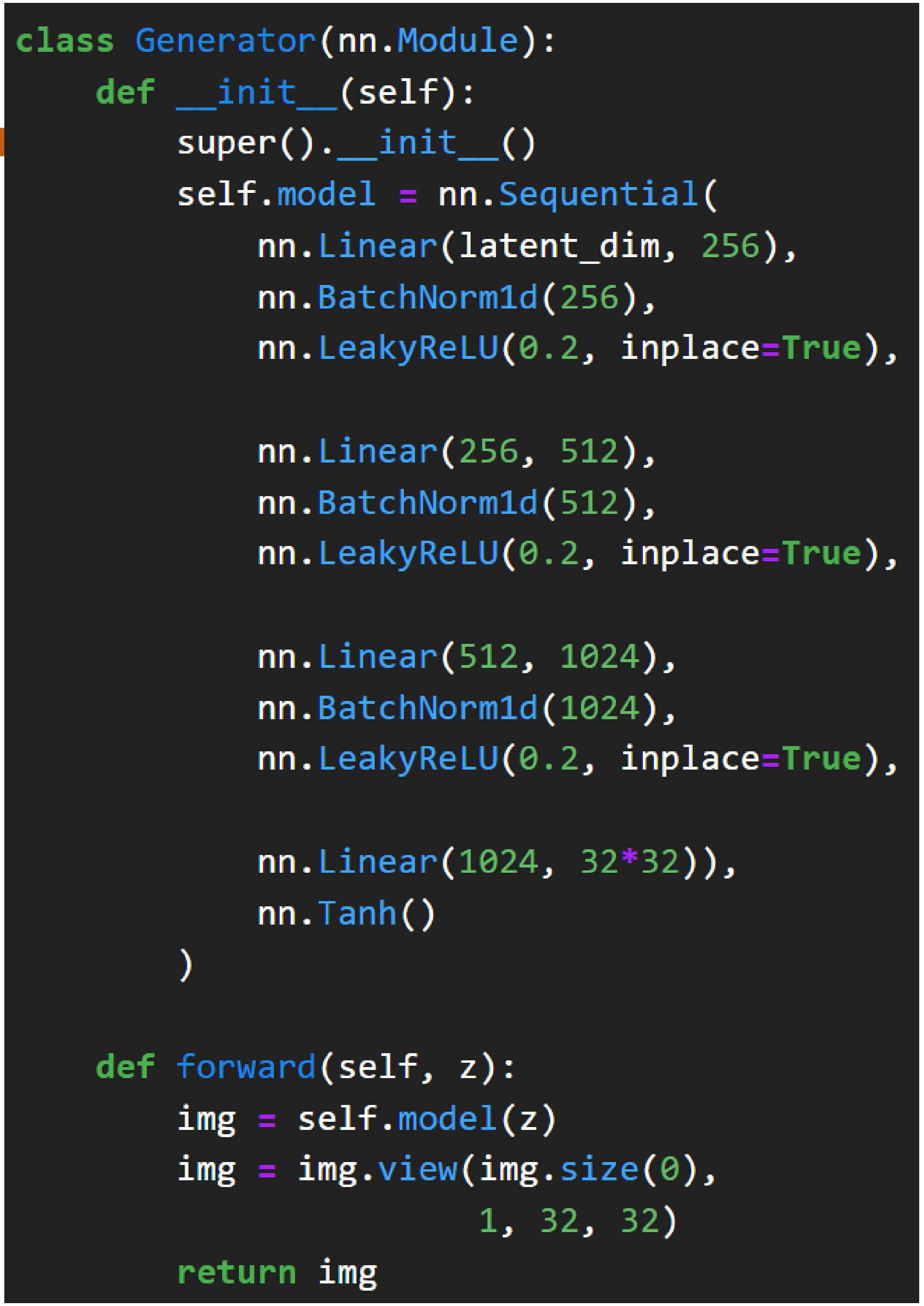
```
self.model = nn.Sequential(
    nn.Linear(int(np.prod(img_shape)), 512),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Linear(512, 256),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Linear(256, 1),
    nn.Sigmoid()
)
```



Standard GAN

❖ Architecture

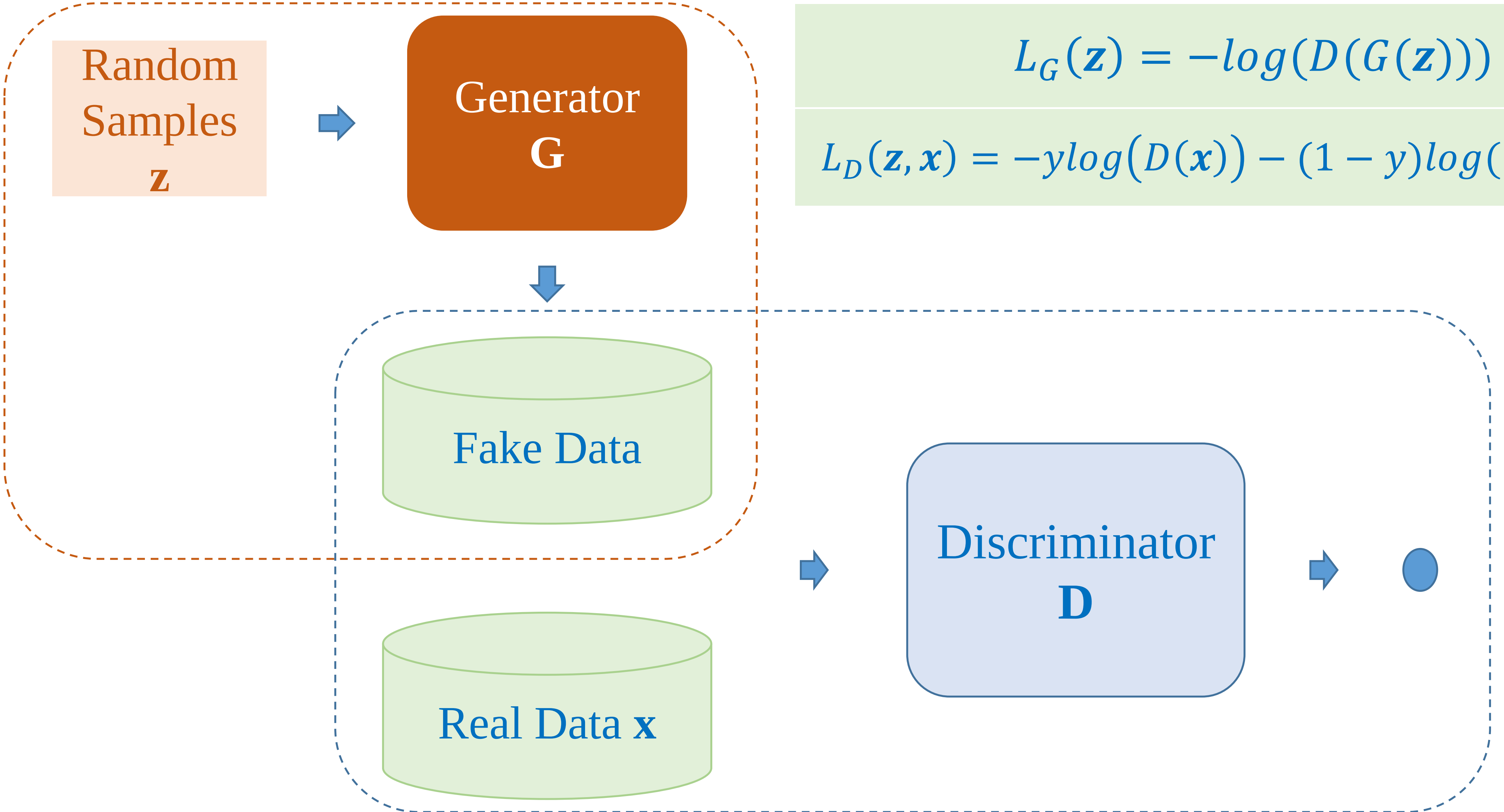




```python
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_dim, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Linear(512, 1024),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Linear(1024, 32*32)),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        img = img.view(img.size(0),
                       1, 32, 32)
        return img
```

Generative Adversarial Networks

Model so far!

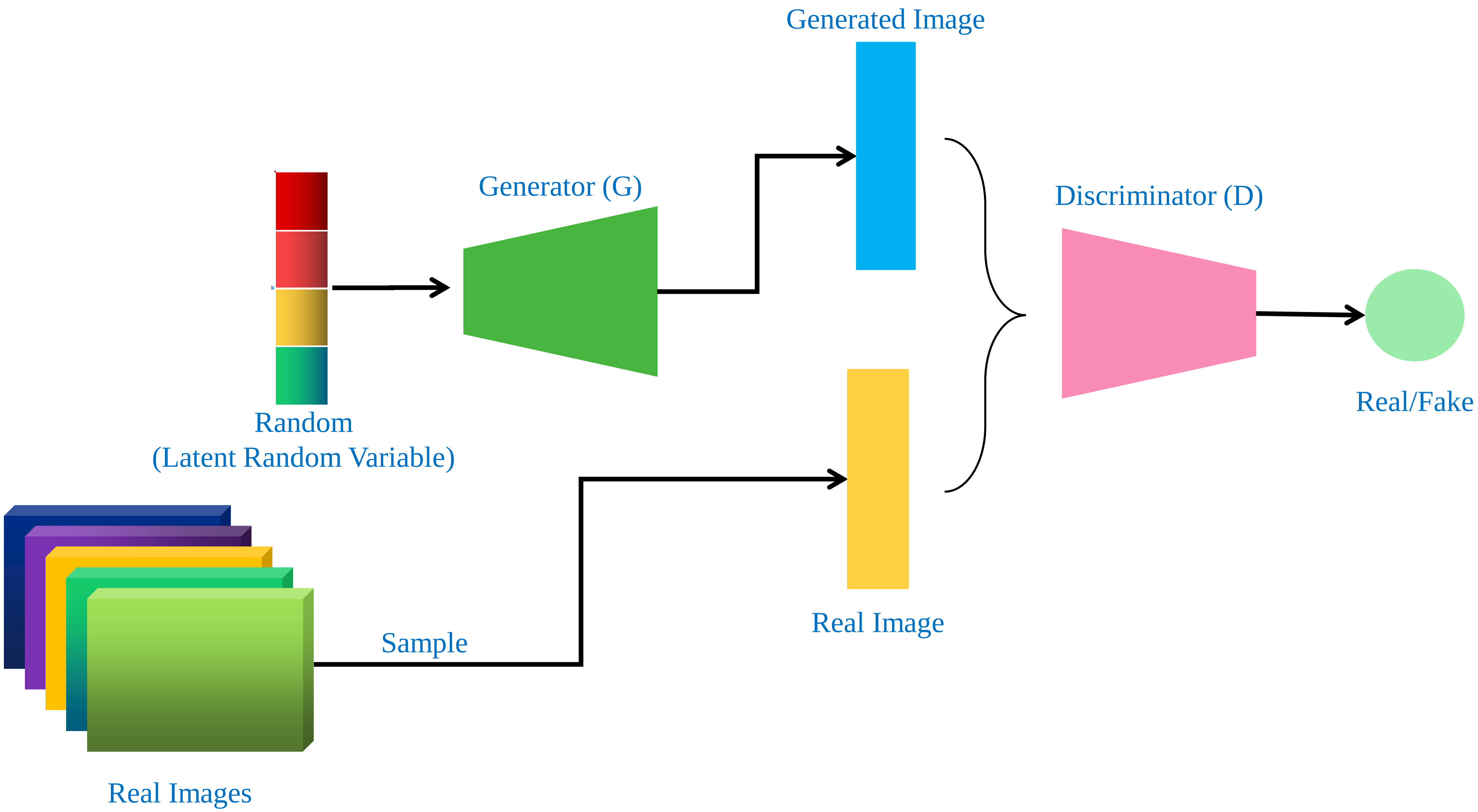


$$L_G(\boldsymbol{z}) = -log(D(G(\boldsymbol{z})))$$

$$L_D(\boldsymbol{z}, \boldsymbol{x}) = -ylog\big(D(\boldsymbol{x})\big) - (1-y)log(1 - D(G(\boldsymbol{z})))$$

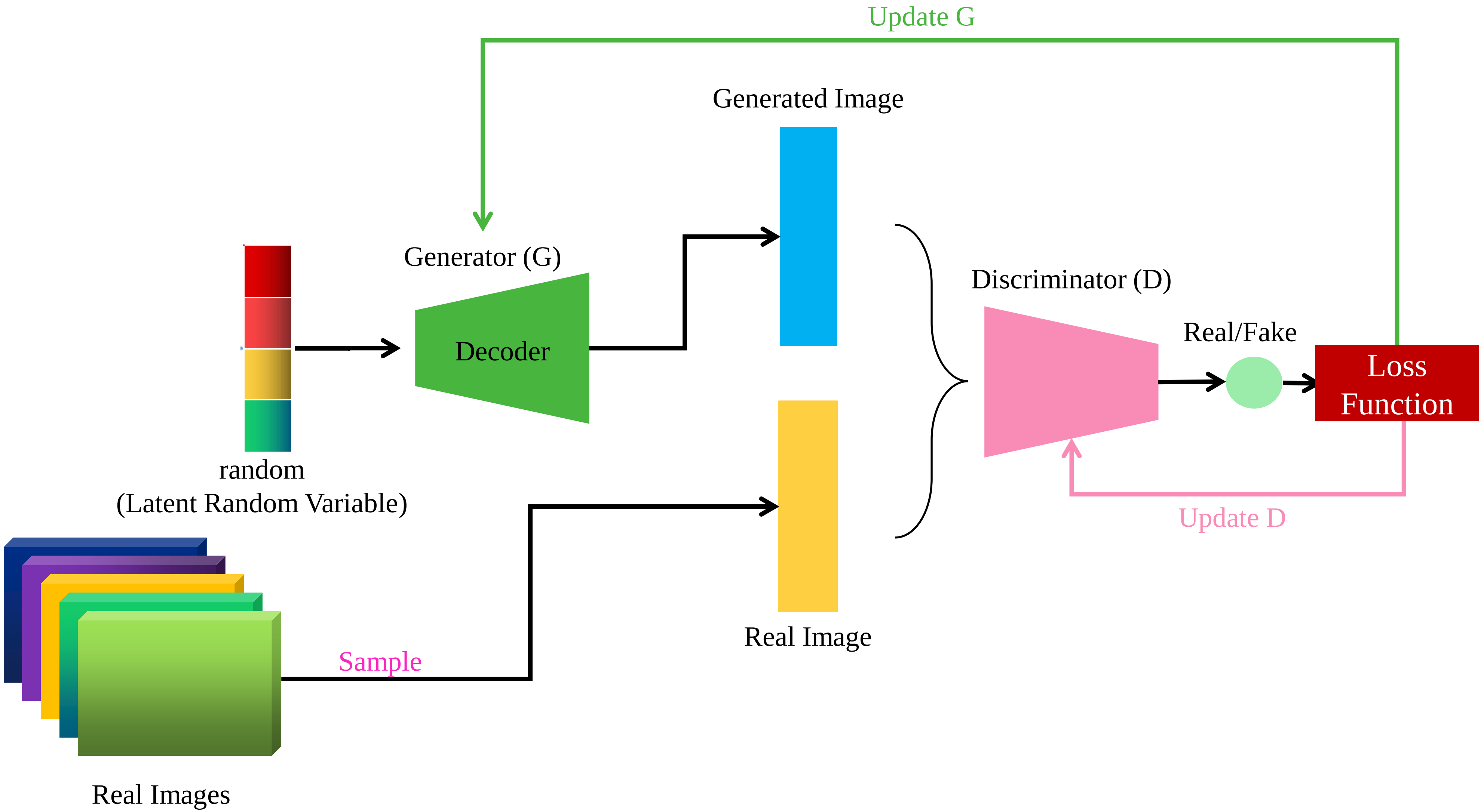
Generative Adversarial Networks

Workflow

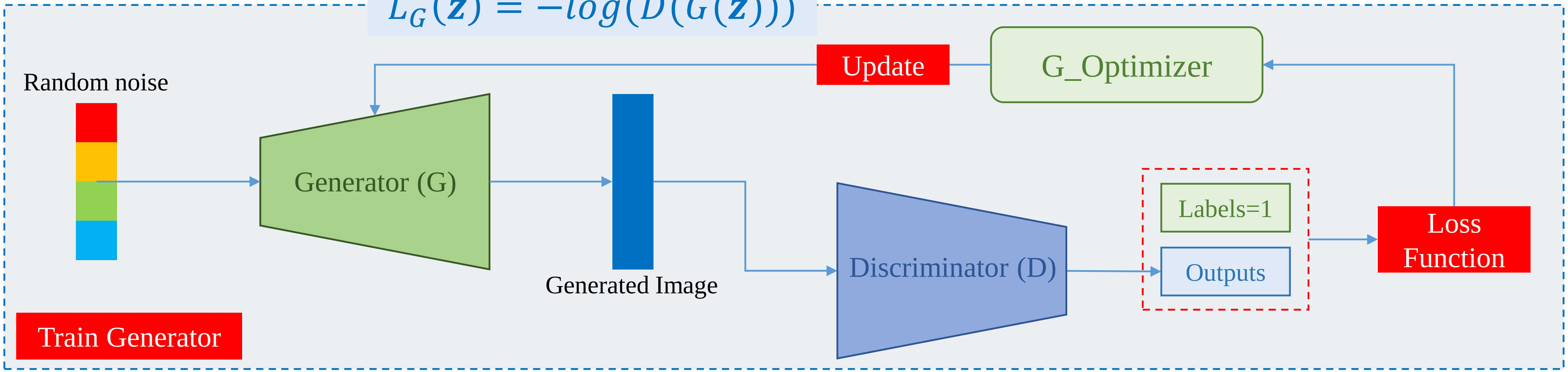


Generative Adversarial Networks

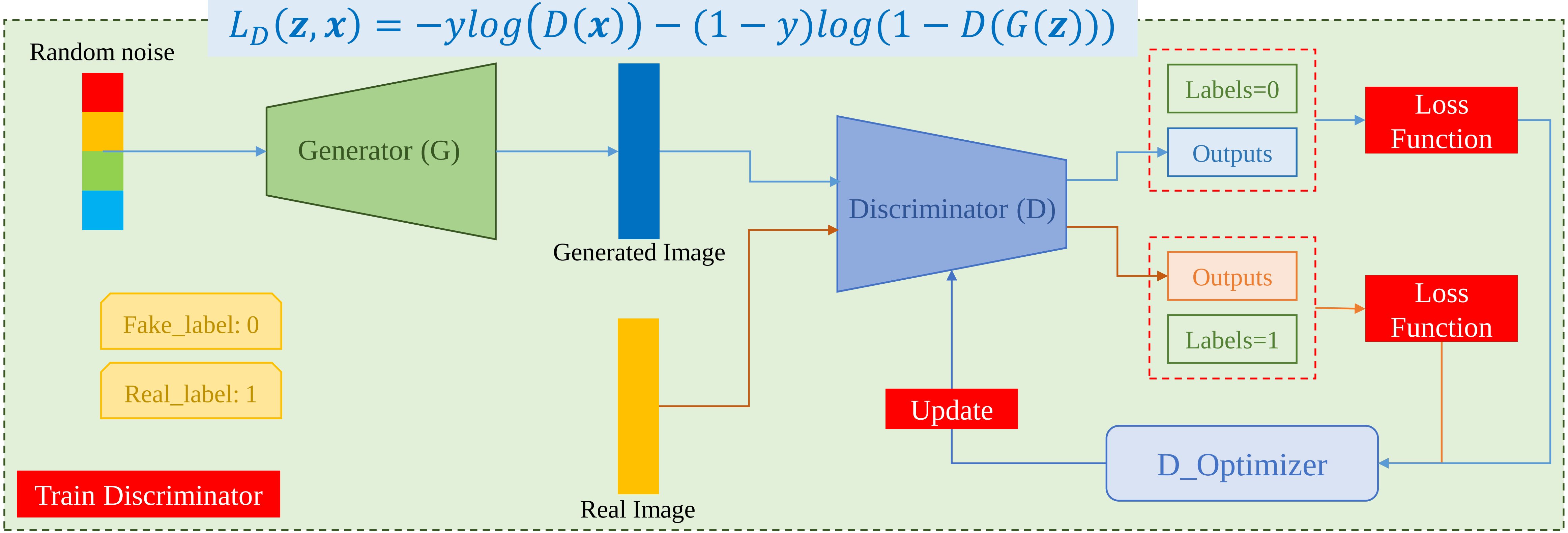
GAN Training



$$L_G(\boldsymbol{z}) = -log(D(G(\boldsymbol{z})))$$

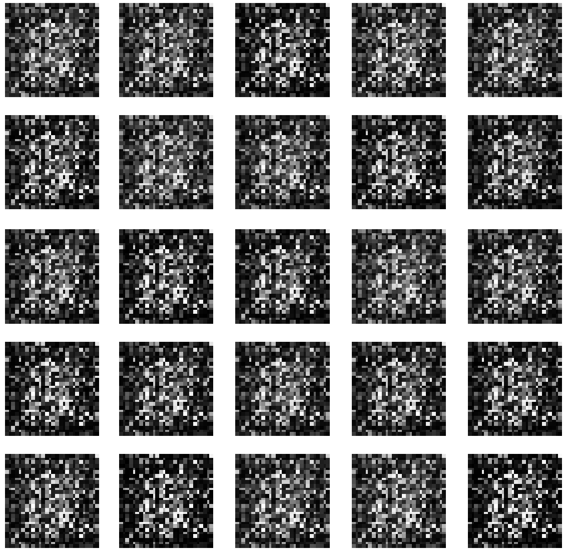


$$L_D(\boldsymbol{z},\boldsymbol{x}) = -ylog\big(D(\boldsymbol{x})\big) - (1-y)log(1-D(G(\boldsymbol{z})))$$

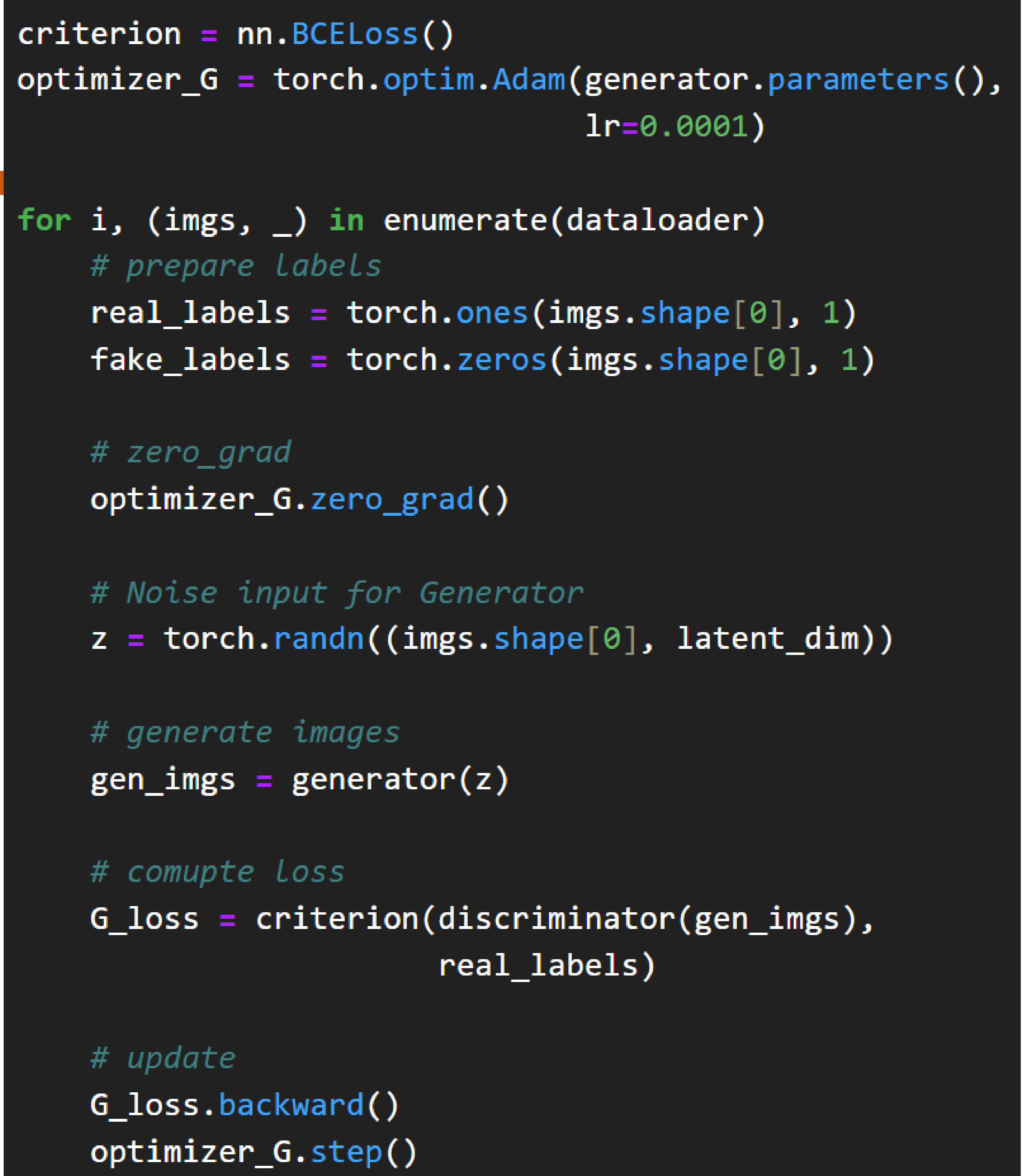


Standard GAN

❖ Train Generator



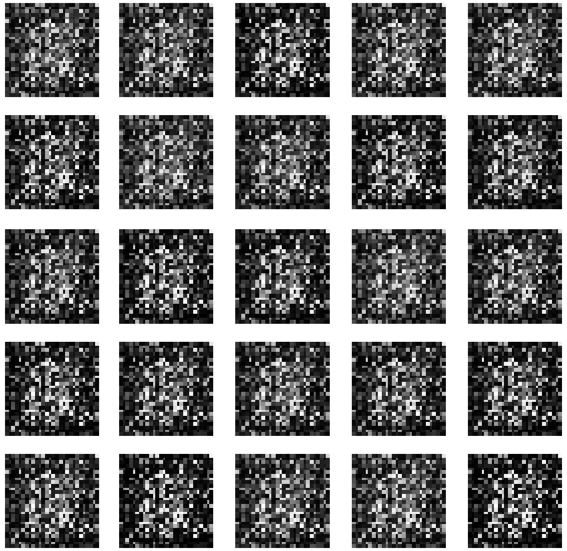
Epoch 1



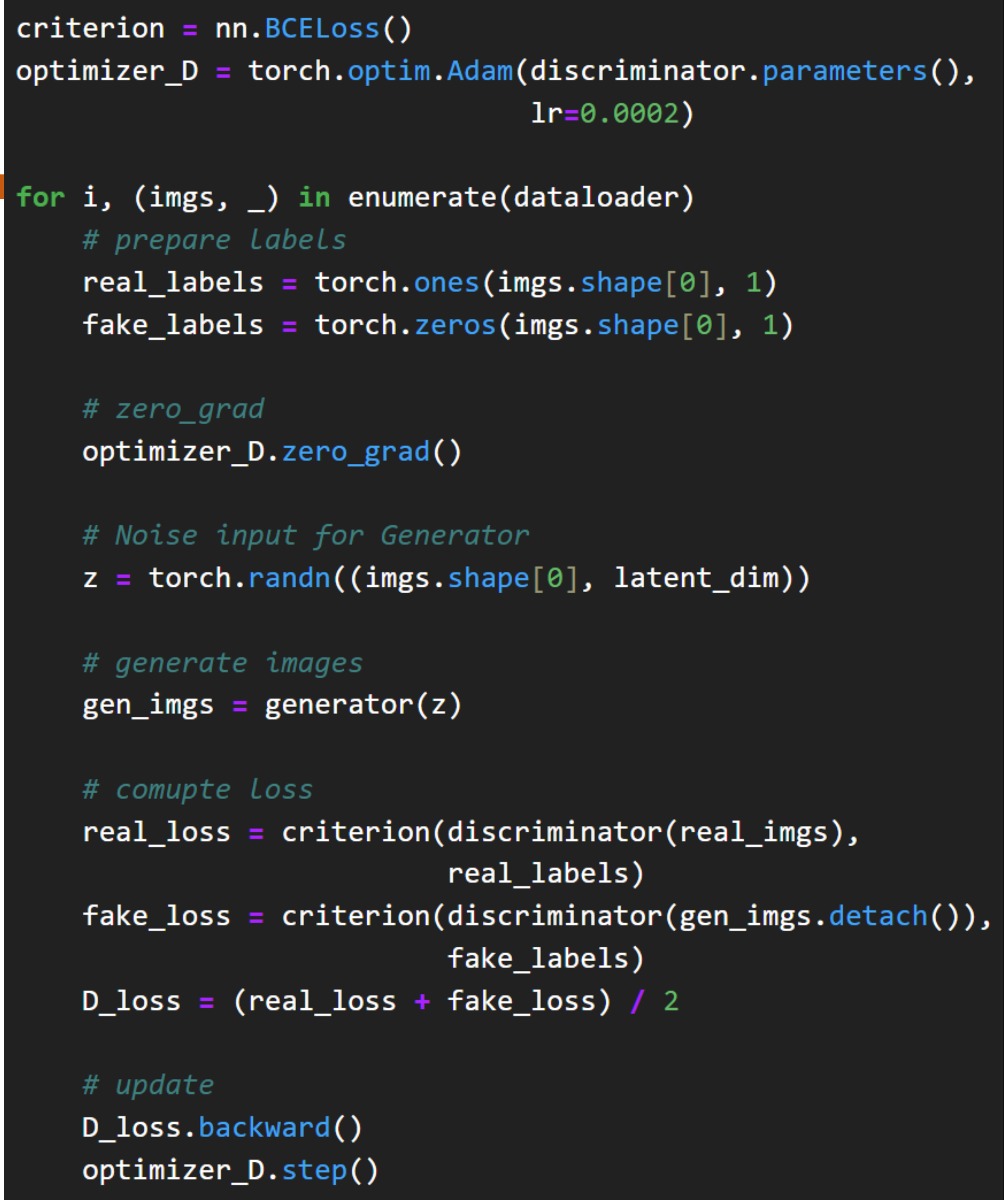
```
criterion = nn.BCELoss()
optimizer_G = torch.optim.Adam(generator.parameters(),
                               lr=0.0001)

for i, (imgs, _) in enumerate(dataloader)
    # prepare labels
    real_labels = torch.ones(imgs.shape[0], 1)
    fake_labels = torch.zeros(imgs.shape[0], 1)

    # zero_grad
    optimizer_G.zero_grad()

    # Noise input for Generator
    z = torch.randn((imgs.shape[0], latent_dim))

    # generate images
    gen_imgs = generator(z)

    # comupte loss
    G_loss = criterion(discriminator(gen_imgs),
                       real_labels)

    # update
    G_loss.backward()
    optimizer_G.step()
```

Standard GAN

❖ Train Discriminator



Epoch 1



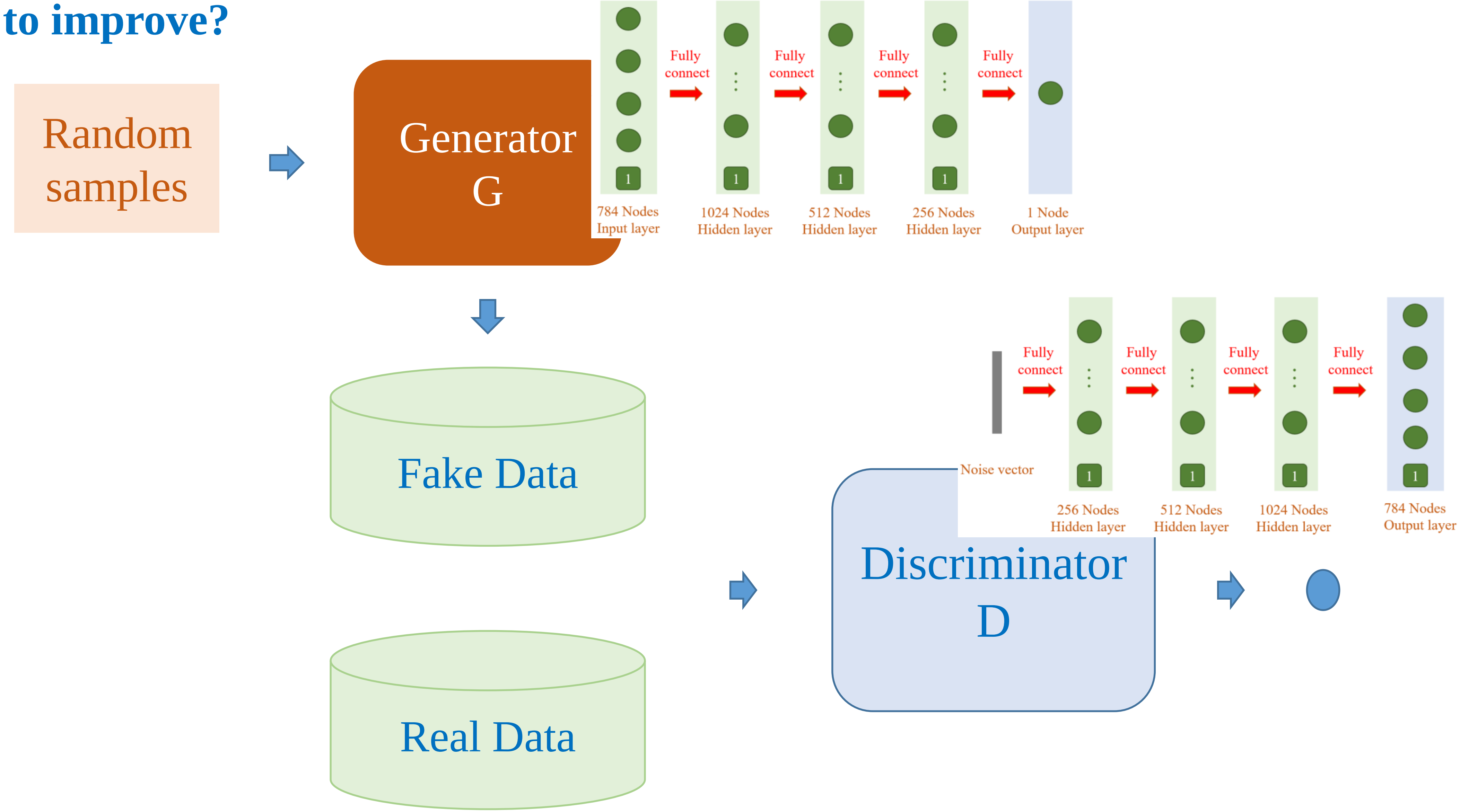
```
criterion = nn.BCELoss()
optimizer_D = torch.optim.Adam(discriminator.parameters(),
                               lr=0.0002)

for i, (imgs, _) in enumerate(dataloader):
    # prepare labels
    real_labels = torch.ones(imgs.shape[0], 1)
    fake_labels = torch.zeros(imgs.shape[0], 1)

    # zero_grad
    optimizer_D.zero_grad()

    # Noise input for Generator
    z = torch.randn((imgs.shape[0], latent_dim))

    # generate images
    gen_imgs = generator(z)

    # comupte loss
    real_loss = criterion(discriminator(real_imgs),
                          real_labels)
    fake_loss = criterion(discriminator(gen_imgs.detach()),
                          fake_labels)
    D_loss = (real_loss + fake_loss) / 2

    # update
    D_loss.backward()
    optimizer_D.step()
```

Outline

- Introduction
- GAN
- DCGAN
- Implementation

Generative Adversarial Networks

❖ How to improve?



Random samples

Generator G

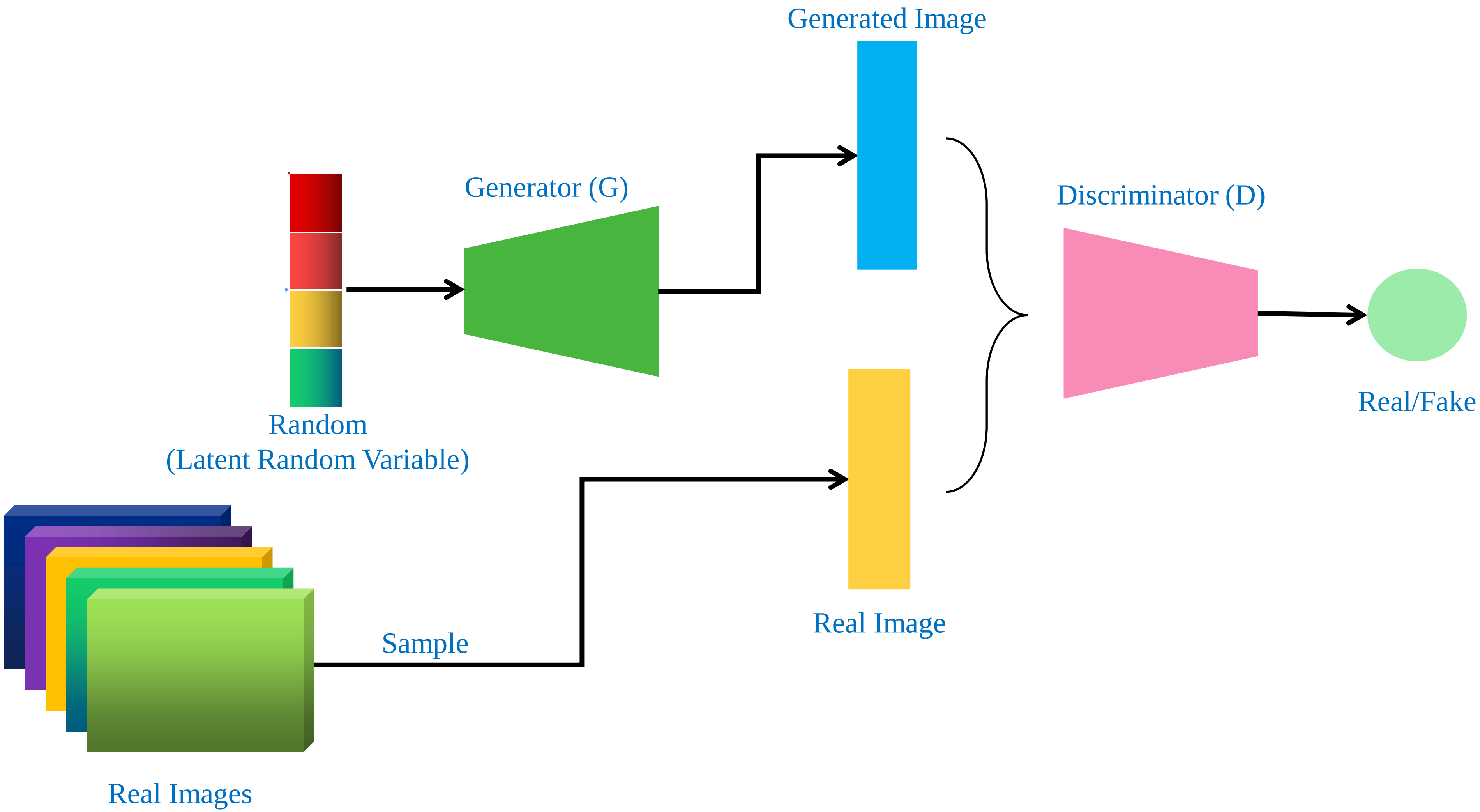
Fake Data

Real Data

Discriminator D

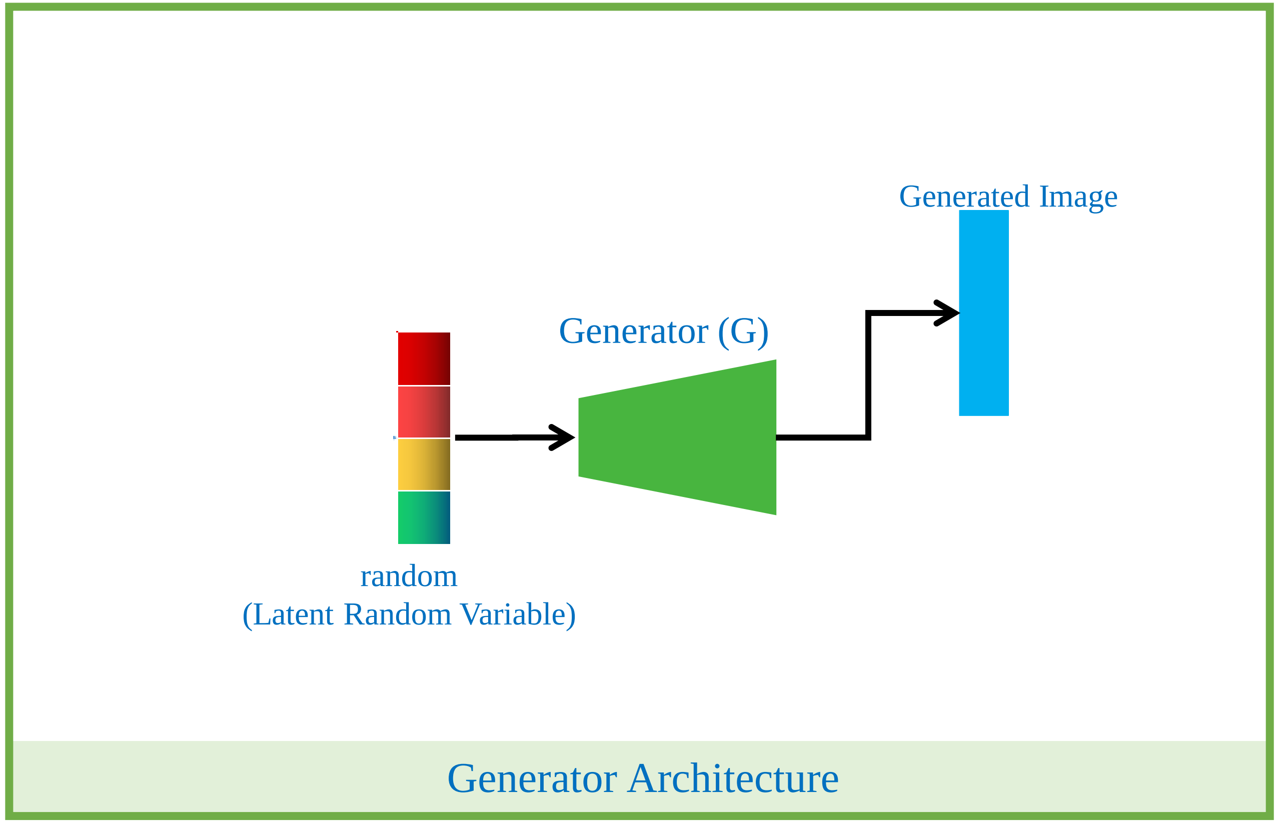
Generative Adversarial Networks

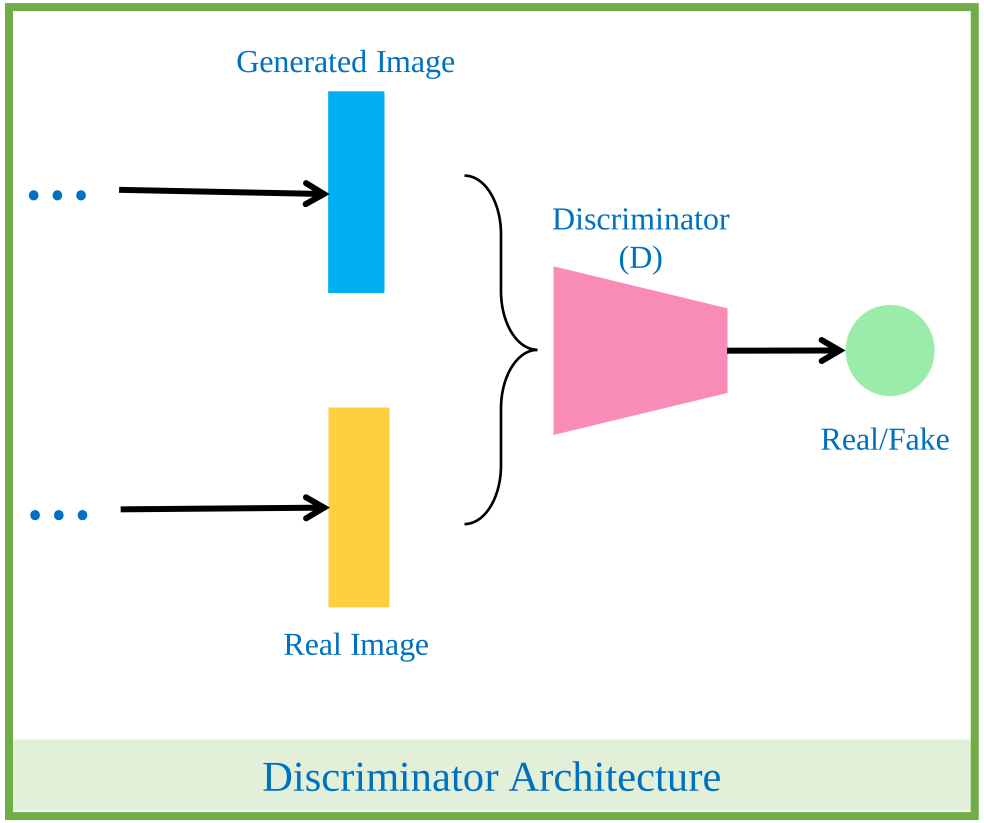
Network architecture



Generative Adversarial Networks

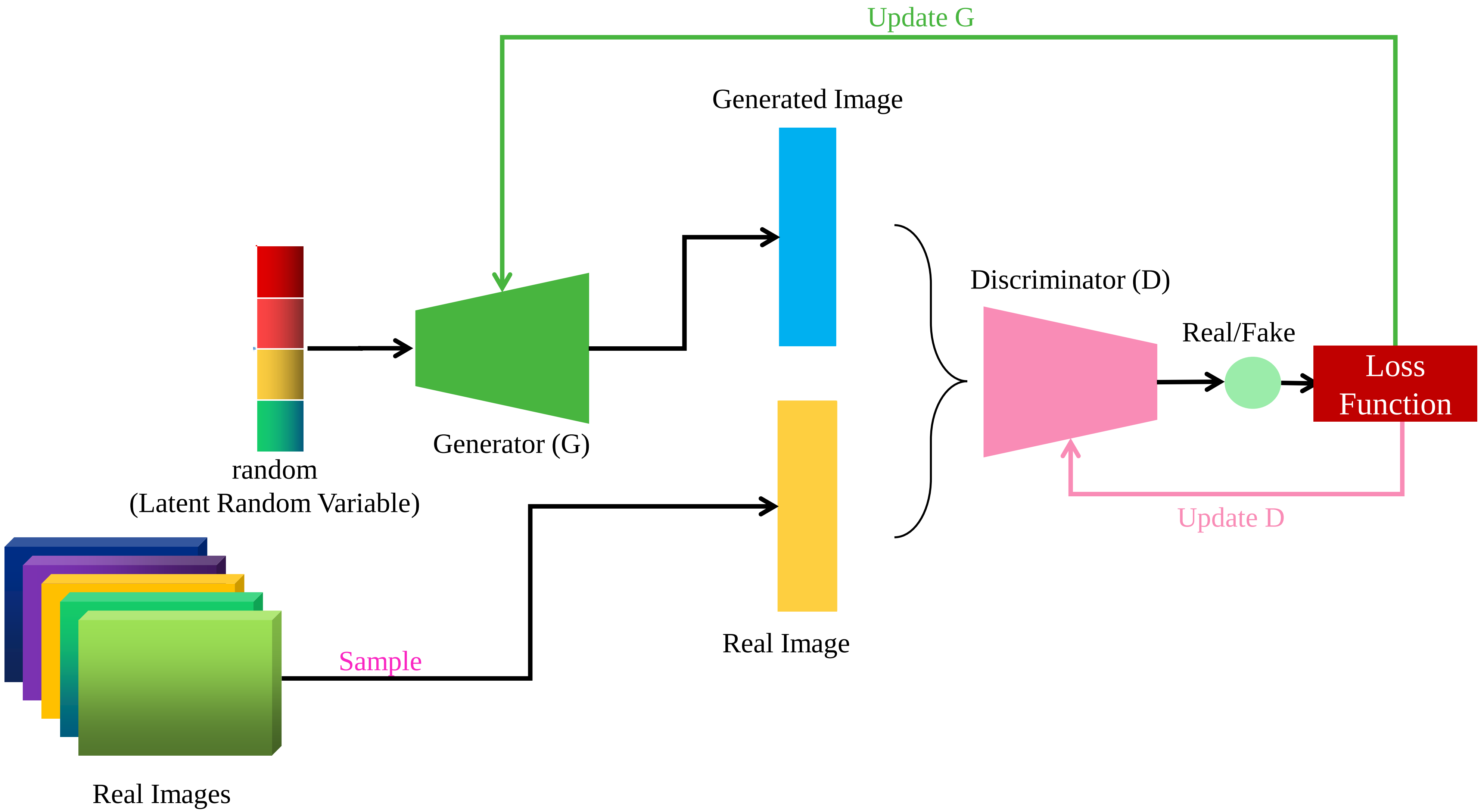
❖ Architecture



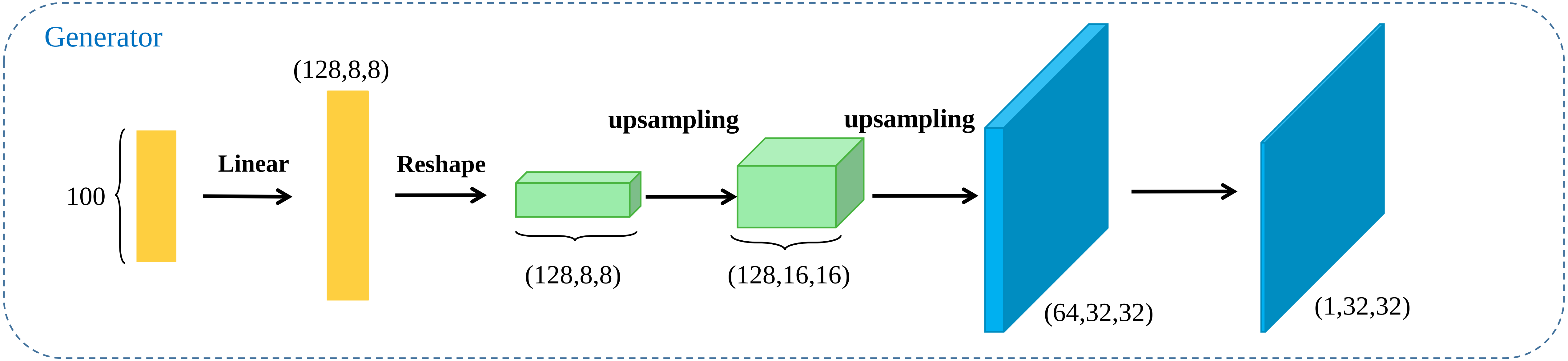


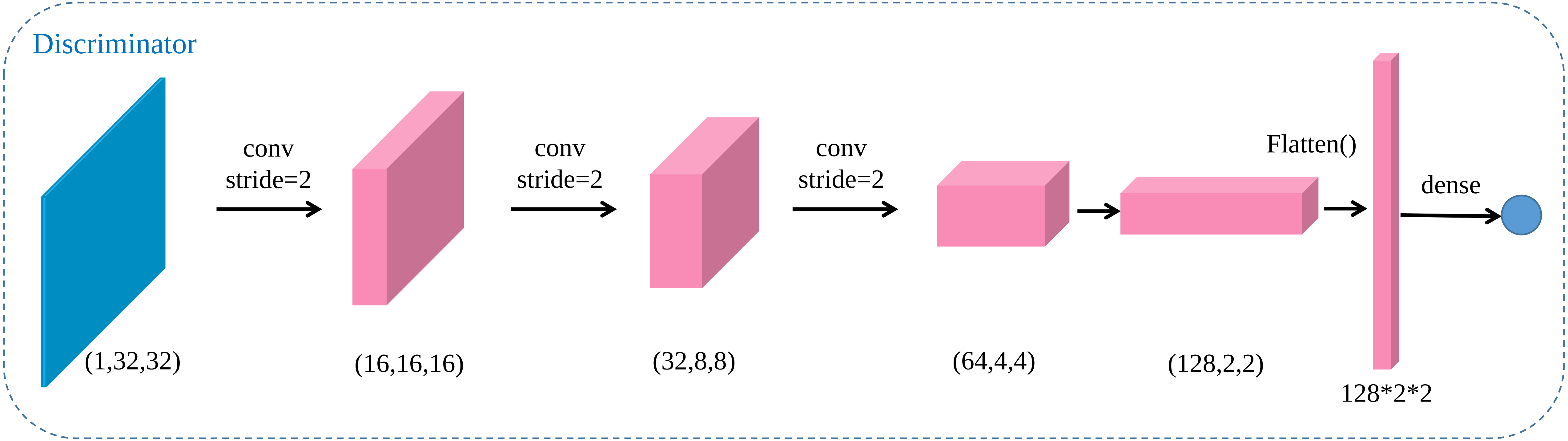
Generative Adversarial Networks

GAN Training

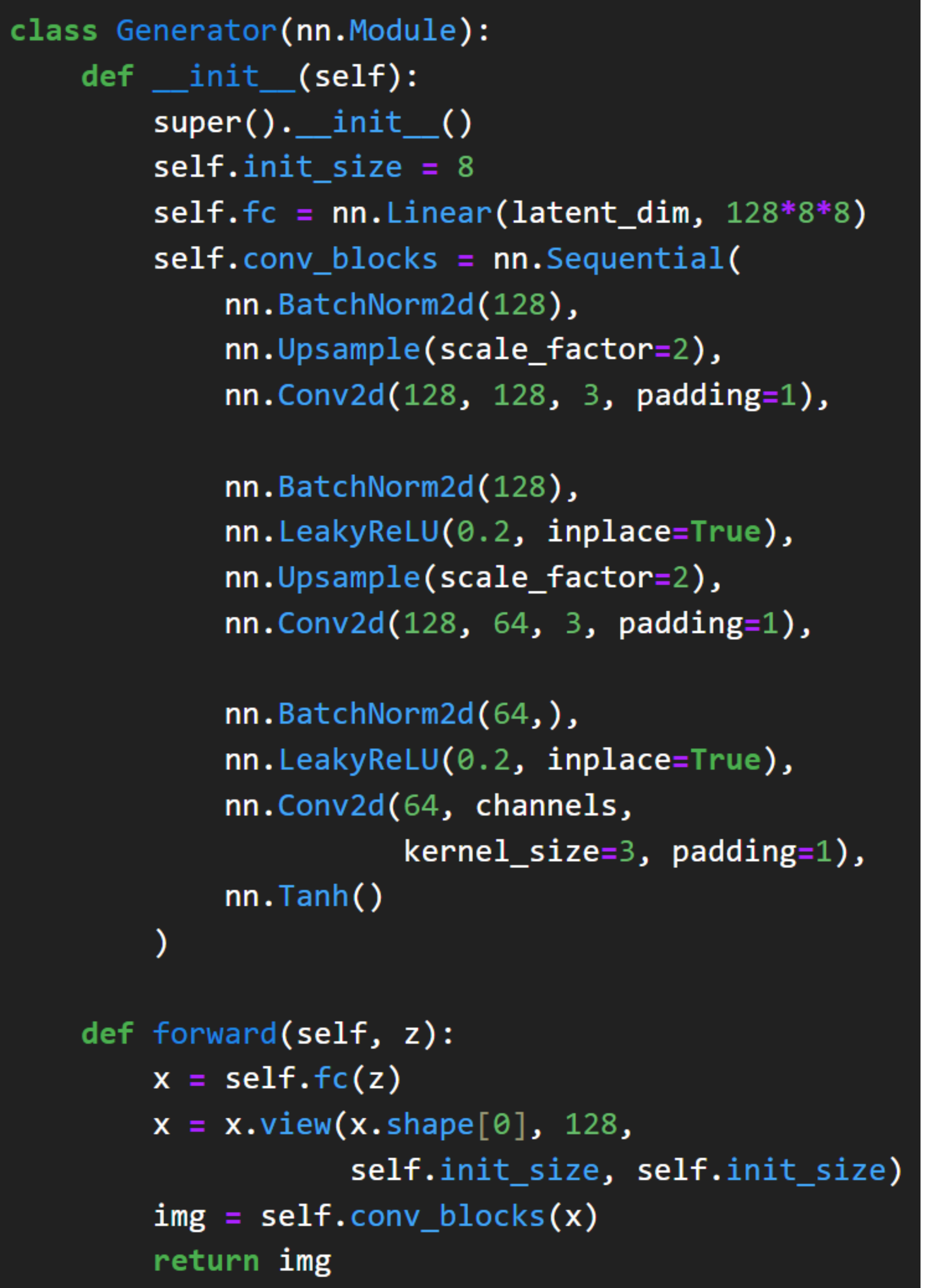


DCGAN

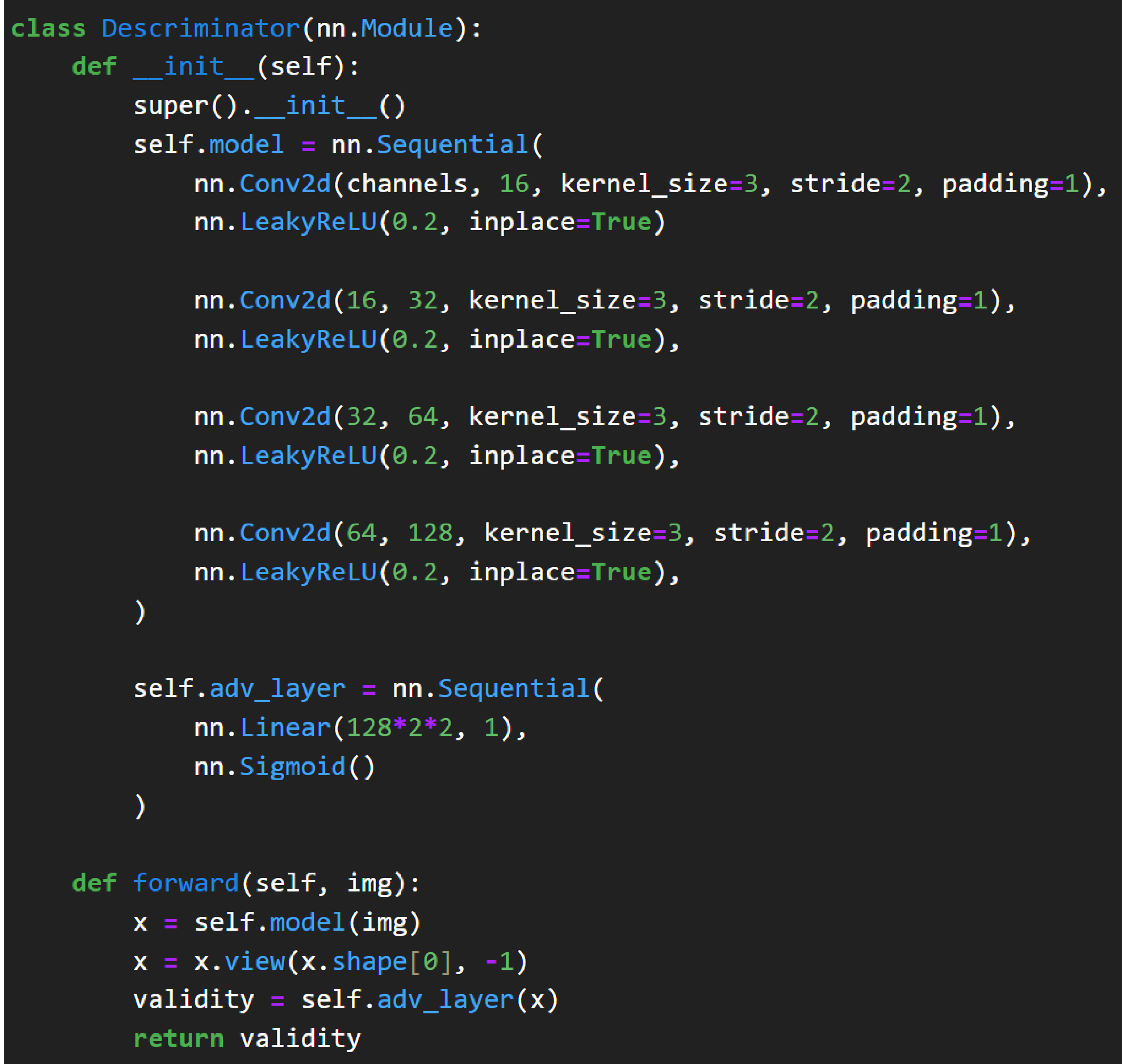

Generator
(128,8,8)
upsampling
upsampling
100
Linear
Reshape
(128,8,8)
(128,16,16)
(64,32,32)
(1,32,32)


Discriminator
conv
stride=2
conv
stride=2
conv
stride=2
Flatten()
dense
(1,32,32)
(16,16,16)
(32,8,8)
(64,4,4)
(128,2,2)
128*2*2

QUIZ TIME



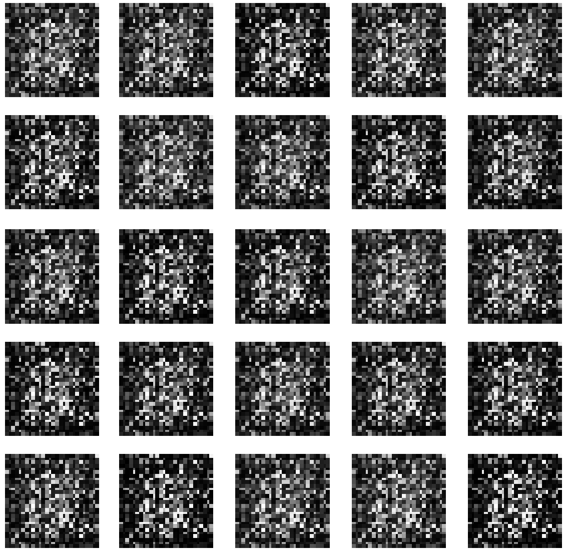
```python
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.init_size = 8
        self.fc = nn.Linear(latent_dim, 128*8*8)
        self.conv_blocks = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 128, 3, padding=1),

            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, 3, padding=1),

            nn.BatchNorm2d(64,),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, channels,
                      kernel_size=3, padding=1),
            nn.Tanh()
        )

    def forward(self, z):
        x = self.fc(z)
        x = x.view(x.shape[0], 128,
                   self.init_size, self.init_size)
        img = self.conv_blocks(x)
        return img
```



```python
class Descriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(channels, 16, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True)

            nn.Conv2d(16, 32, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
        )

        self.adv_layer = nn.Sequential(
            nn.Linear(128*2*2, 1),
            nn.Sigmoid()
        )

    def forward(self, img):
        x = self.model(img)
        x = x.view(x.shape[0], -1)
        validity = self.adv_layer(x)
        return validity
```

DCGAN

❖ Train Generator

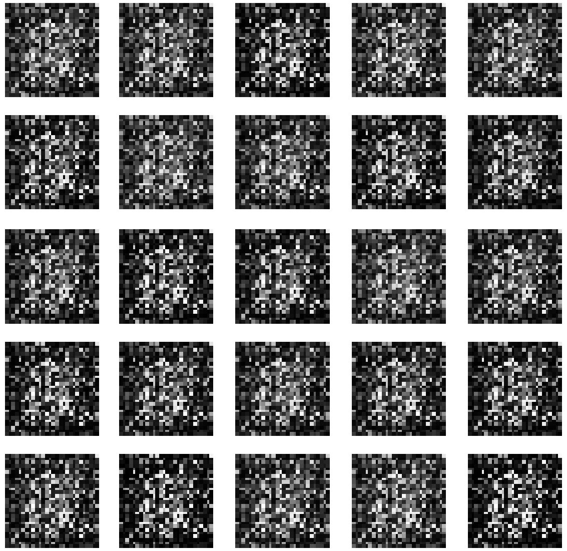


Epoch 1

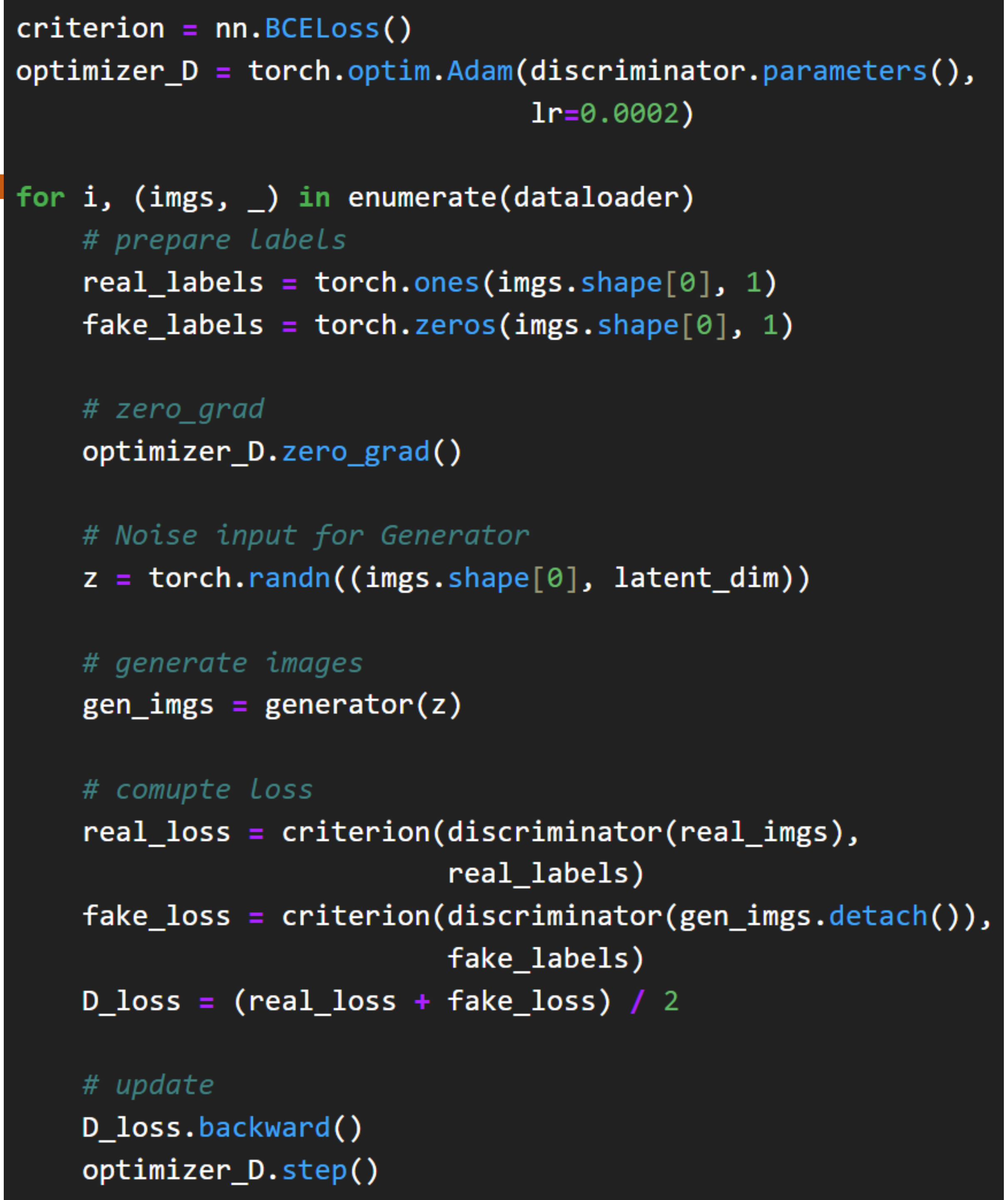
```
criterion = nn.BCELoss()
optimizer_G = torch.optim.Adam(generator.parameters(),
                               lr=0.0001)

for i, (imgs, _) in enumerate(dataloader)
    # prepare labels
    real_labels = torch.ones(imgs.shape[0], 1)
    fake_labels = torch.zeros(imgs.shape[0], 1)

    # zero_grad
    optimizer_G.zero_grad()

    # Noise input for Generator
    z = torch.randn((imgs.shape[0], latent_dim))

    # generate images
    gen_imgs = generator(z)

    # comupte loss
    G_loss = criterion(discriminator(gen_imgs),
                       real_labels)

    # update
    G_loss.backward()
    optimizer_G.step()
```

DCGAN

❖ Train Discriminator



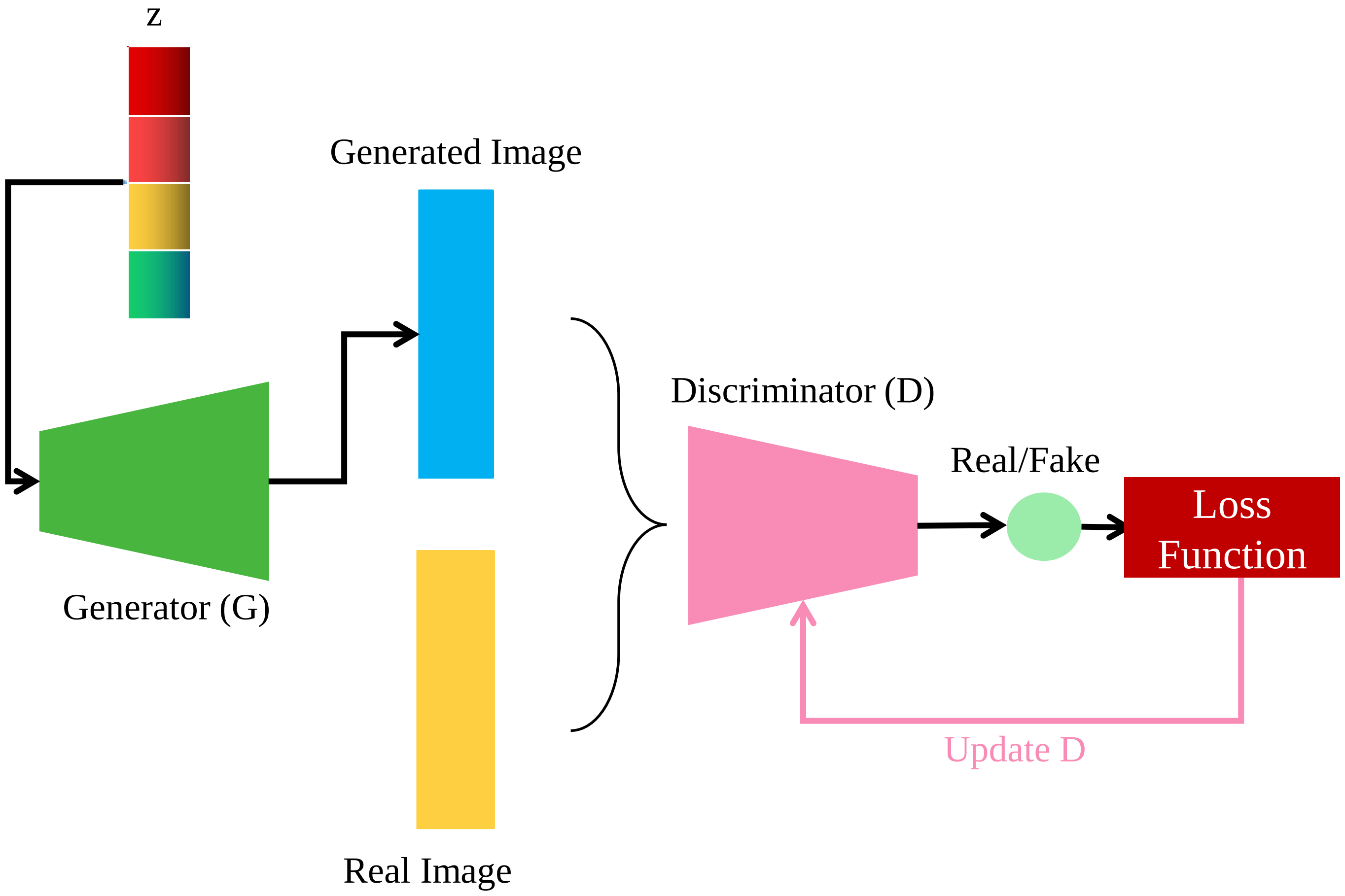
Epoch 1

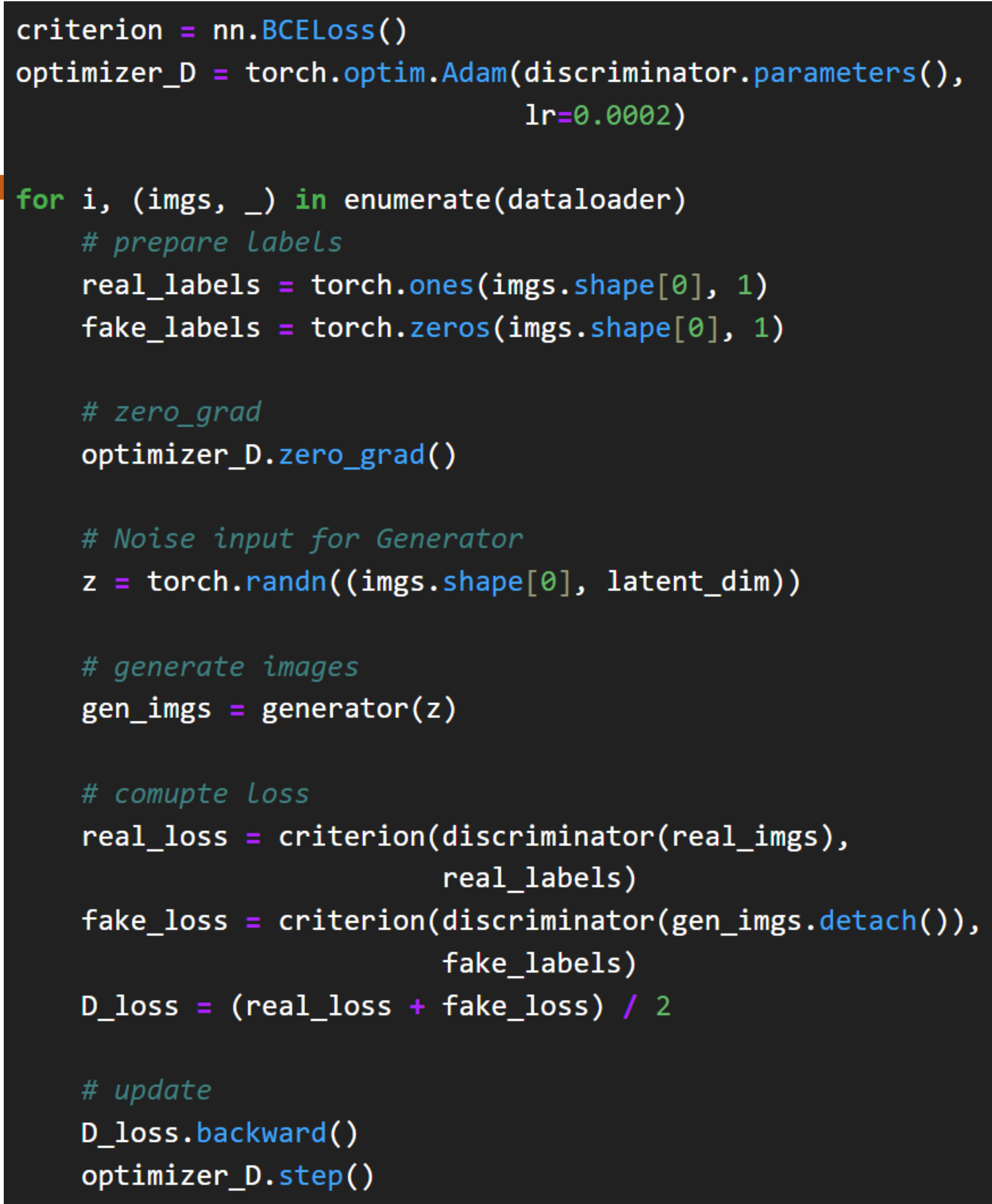


```
criterion = nn.BCELoss()
optimizer_D = torch.optim.Adam(discriminator.parameters(),
                               lr=0.0002)

for i, (imgs, _) in enumerate(dataloader):
    # prepare labels
    real_labels = torch.ones(imgs.shape[0], 1)
    fake_labels = torch.zeros(imgs.shape[0], 1)

    # zero_grad
    optimizer_D.zero_grad()

    # Noise input for Generator
    z = torch.randn((imgs.shape[0], latent_dim))

    # generate images
    gen_imgs = generator(z)

    # comupte loss
    real_loss = criterion(discriminator(real_imgs),
                          real_labels)
    fake_loss = criterion(discriminator(gen_imgs.detach()),
                          fake_labels)
    D_loss = (real_loss + fake_loss) / 2

    # update
    D_loss.backward()
    optimizer_D.step()
```

DCGAN

❖ Implementation

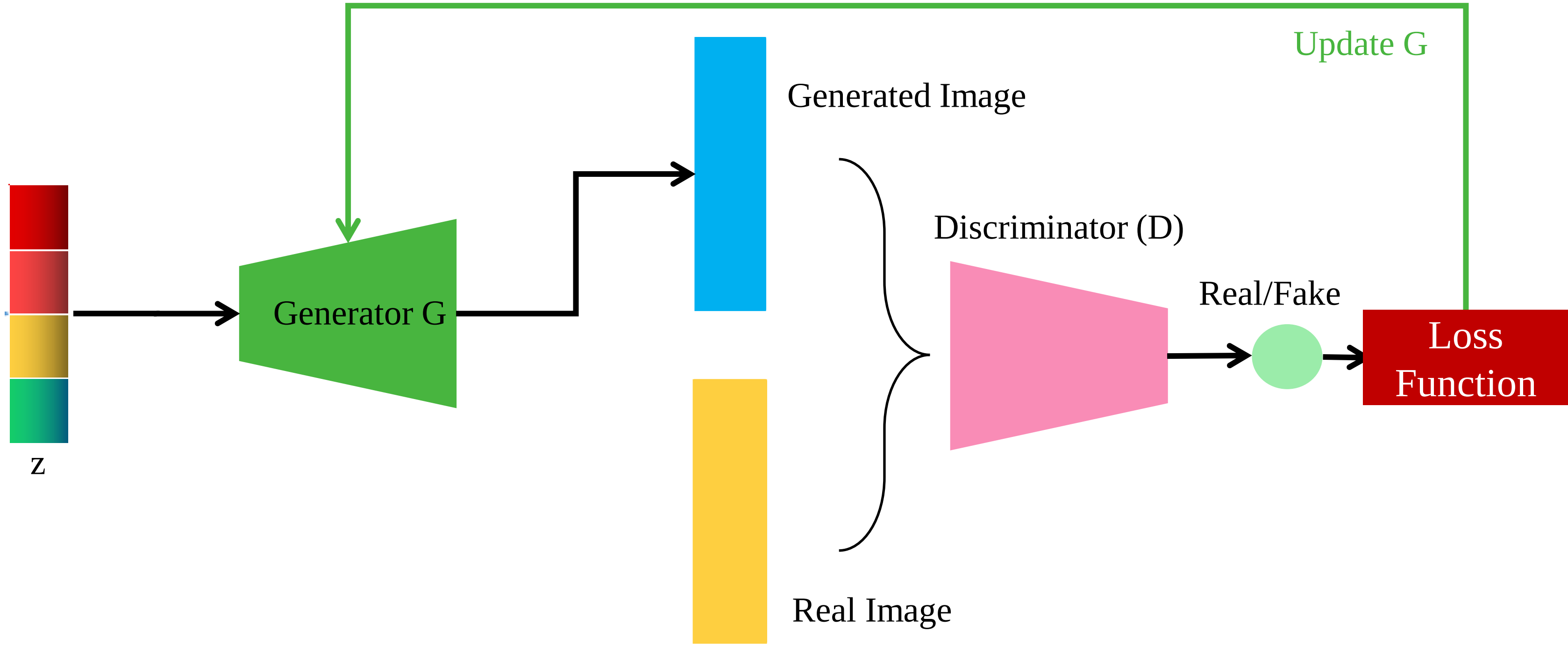


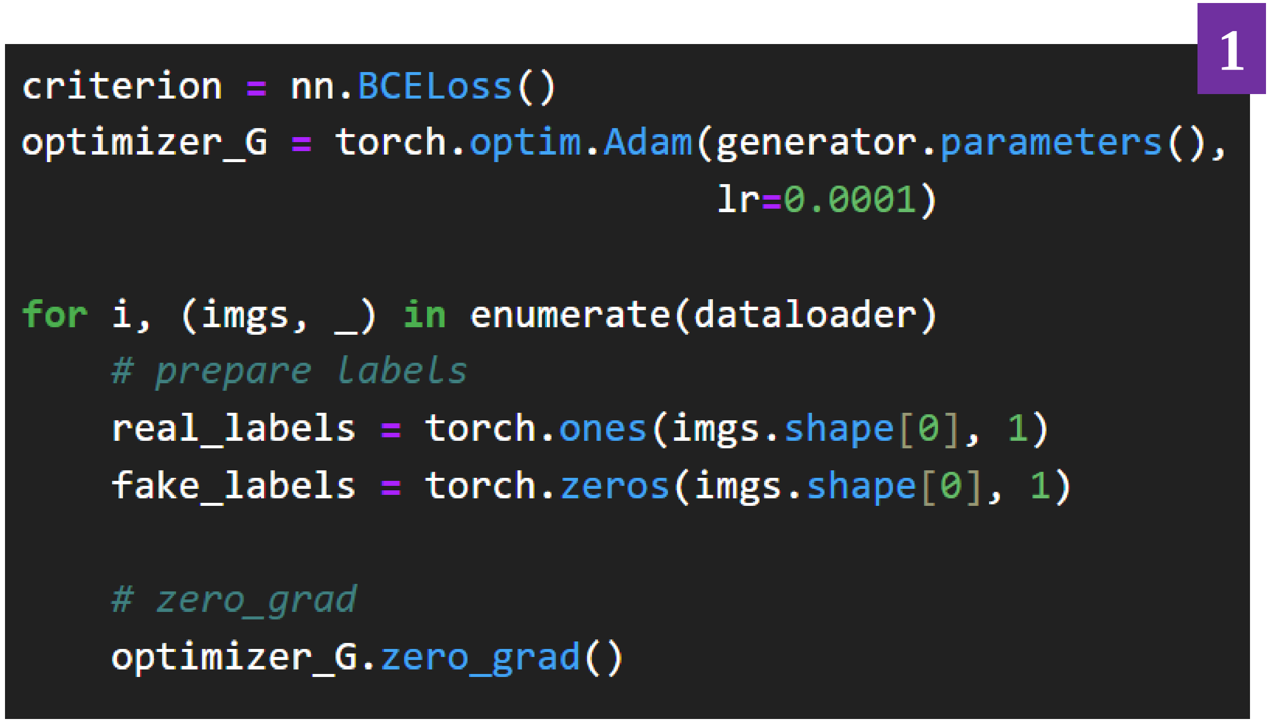


```
criterion = nn.BCELoss()
optimizer_D = torch.optim.Adam(discriminator.parameters(),
                               lr=0.0002)

for i, (imgs, _) in enumerate(dataloader)
    # prepare labels
    real_labels = torch.ones(imgs.shape[0], 1)
    fake_labels = torch.zeros(imgs.shape[0], 1)

    # zero_grad
    optimizer_D.zero_grad()

    # Noise input for Generator
    z = torch.randn((imgs.shape[0], latent_dim))

    # generate images
    gen_imgs = generator(z)

    # comupte loss
    real_loss = criterion(discriminator(real_imgs),
                          real_labels)
    fake_loss = criterion(discriminator(gen_imgs.detach()),
                          fake_labels)
    D_loss = (real_loss + fake_loss) / 2

    # update
    D_loss.backward()
    optimizer_D.step()
```

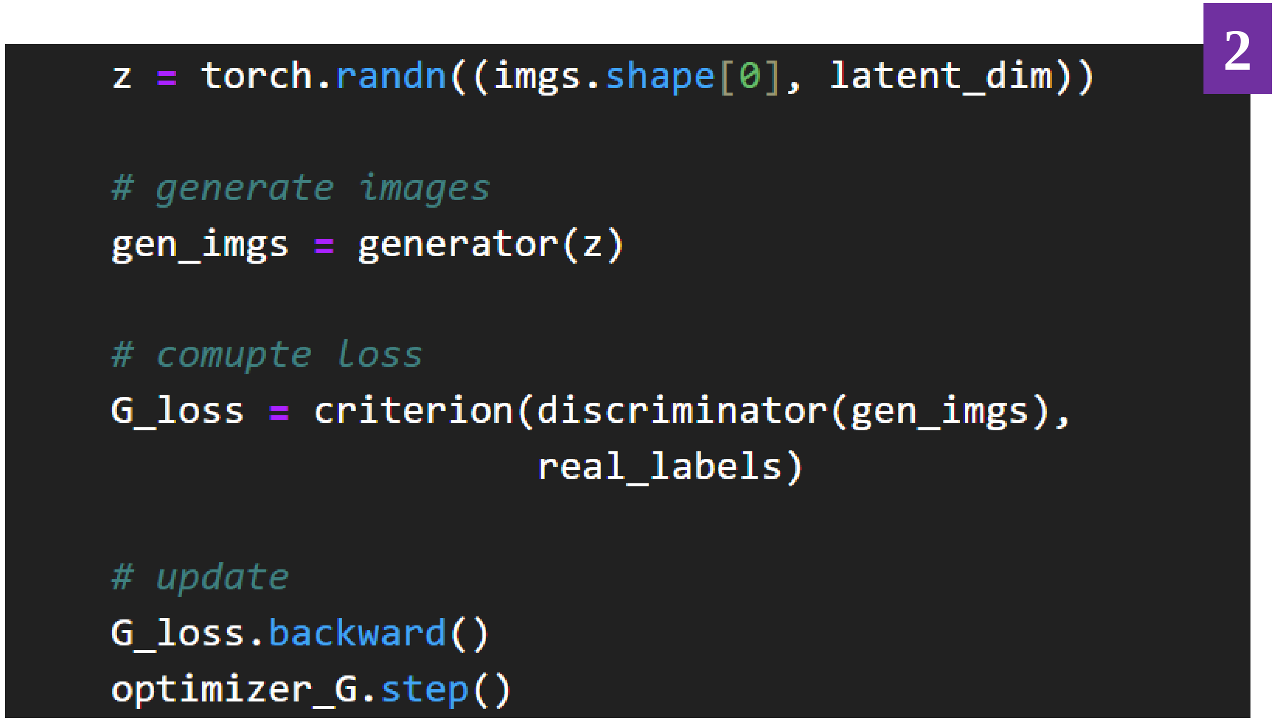
DCGAN

Implementation




1

```
criterion = nn.BCELoss()
optimizer_G = torch.optim.Adam(generator.parameters(),
                               lr=0.0001)

for i, (imgs, _) in enumerate(dataloader)
    # prepare labels
    real_labels = torch.ones(imgs.shape[0], 1)
    fake_labels = torch.zeros(imgs.shape[0], 1)

    # zero_grad
    optimizer_G.zero_grad()
```


2

```
    z = torch.randn((imgs.shape[0], latent_dim))

    # generate images
    gen_imgs = generator(z)

    # comupte loss
    G_loss = criterion(discriminator(gen_imgs),
                       real_labels)

    # update
    G_loss.backward()
    optimizer_G.step()
```

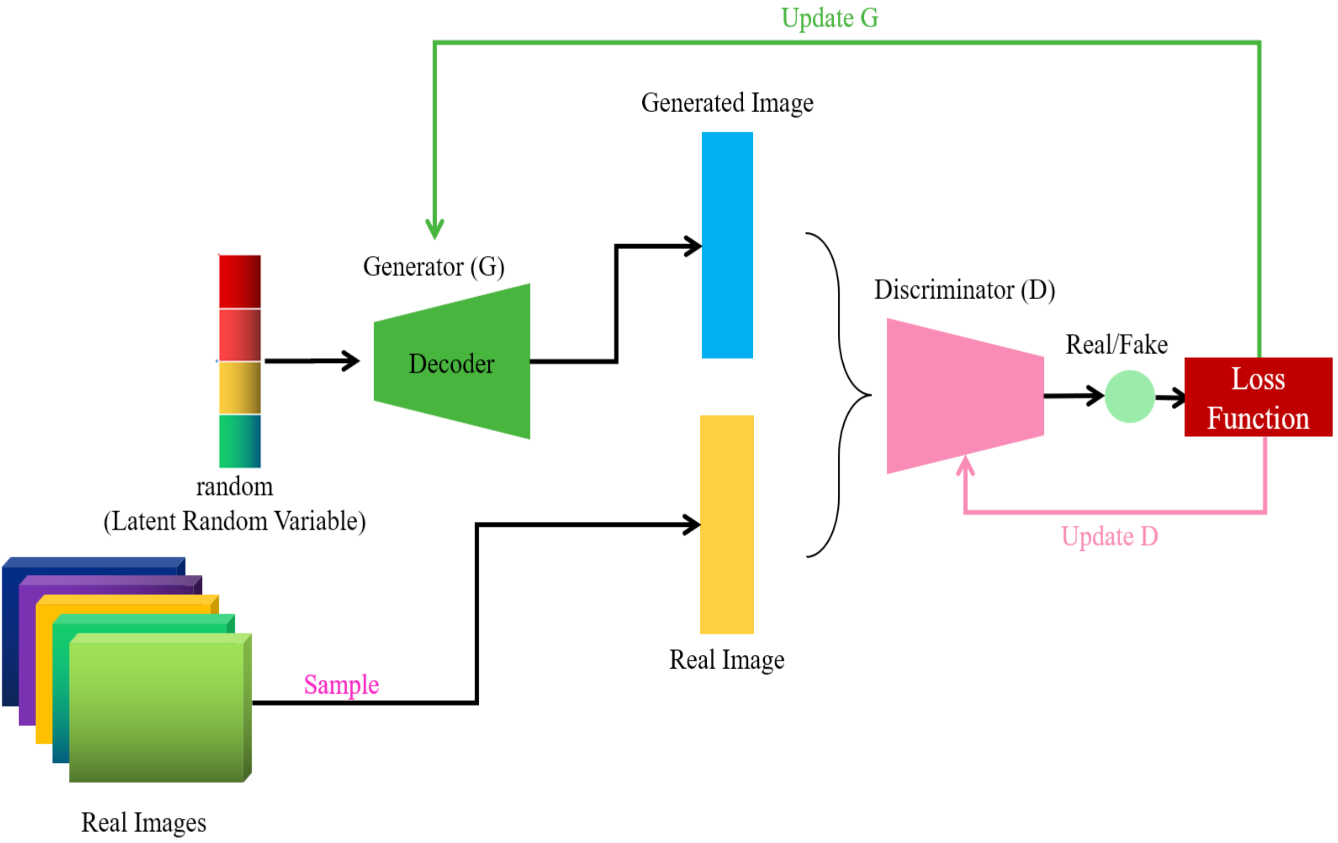
Outline

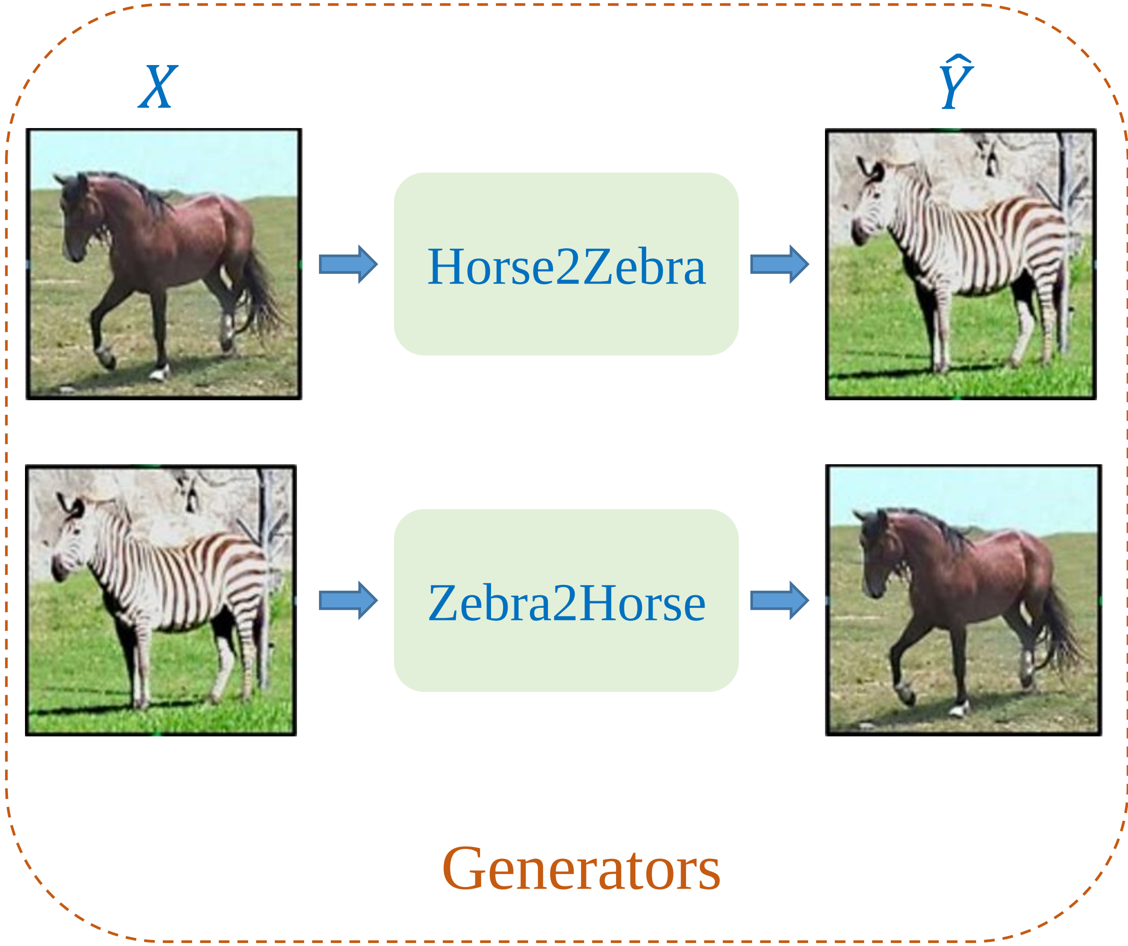
- Introduction
- GAN
- DCGAN
- Implementation

Summary

- ✓ Studied Generative Adversarial Network
- ✓ Studied Deep Convolutional GAN

✓ GAN Extensions


Update G
Generated Image
Generator (G)
Decoder
random
(Latent Random Variable)
Discriminator (D)
Real/Fake
Loss Function
Update D
Real Image
Sample
Real Images


X
$\hat{Y}$
Horse2Zebra
Zebra2Horse
Generators