AI VIETNAM
All-in-One Course

Generative Adversarial Networks

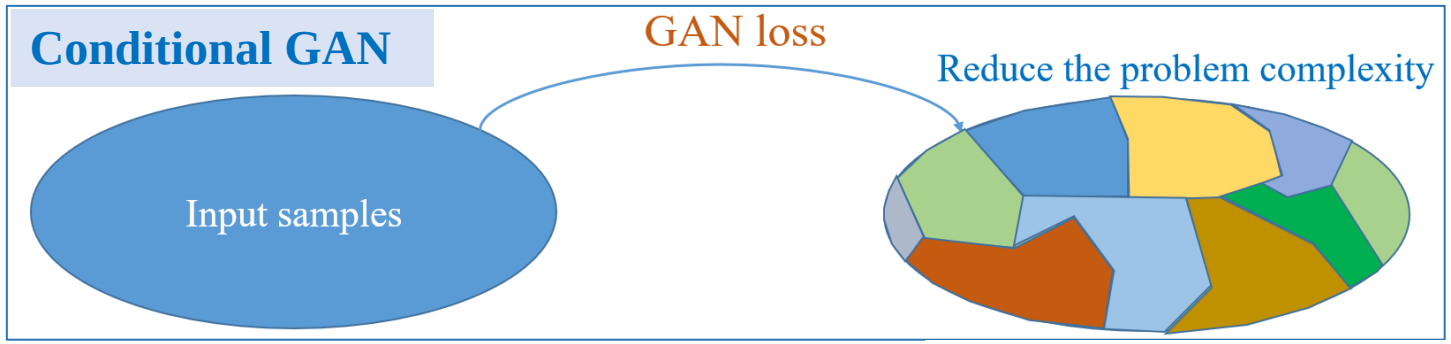
Toward to CycleGAN

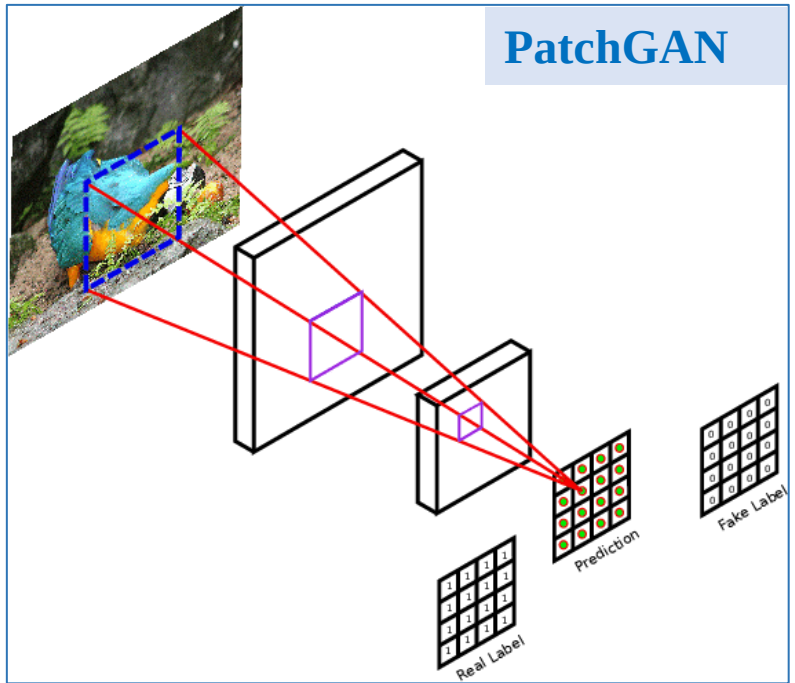
Quang-Vinh Dinh
Ph.D. in Computer Science

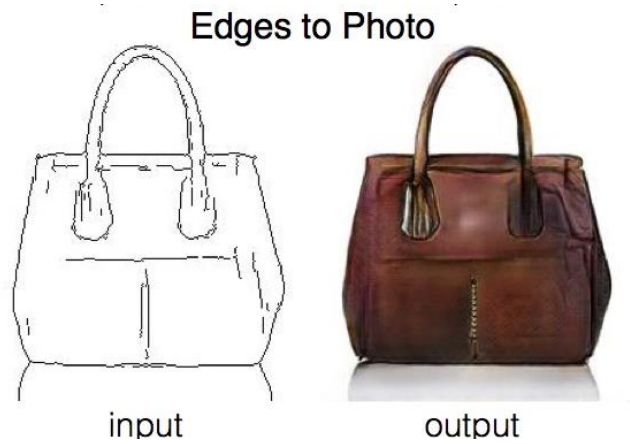
Year 2024

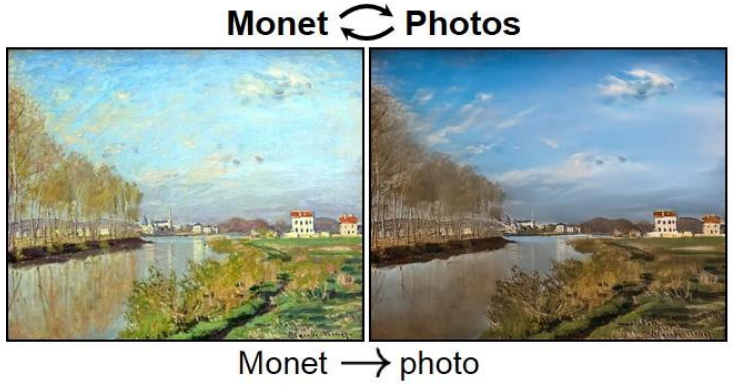
Objectives

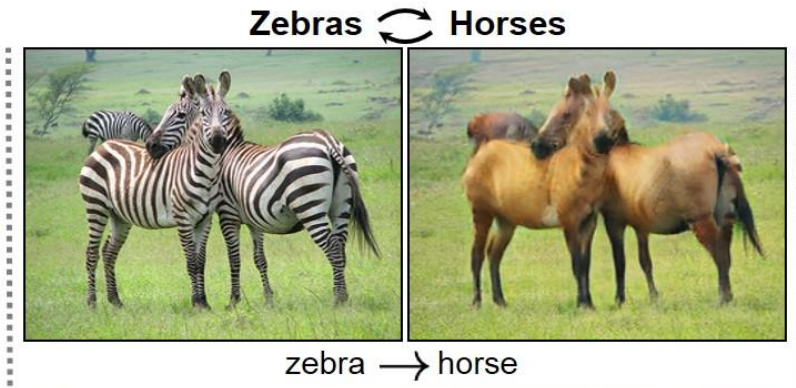
- ✓ Study conditional GAN
- ✓ Study Pix2pix
- ✓ Study PatchGAN
- ✓ Study CycleGAN











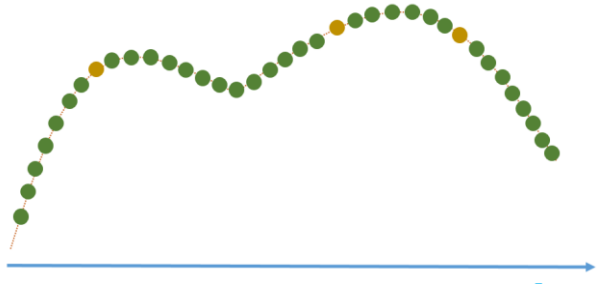
Outline

- Quick Review on GAN/DCGAN
- Conditional GAN
- PatchGAN
- Pix2Pix
- CycleGAN

Generative Adversarial Networks

❖ Input?

A perfect case: Have unlimited training

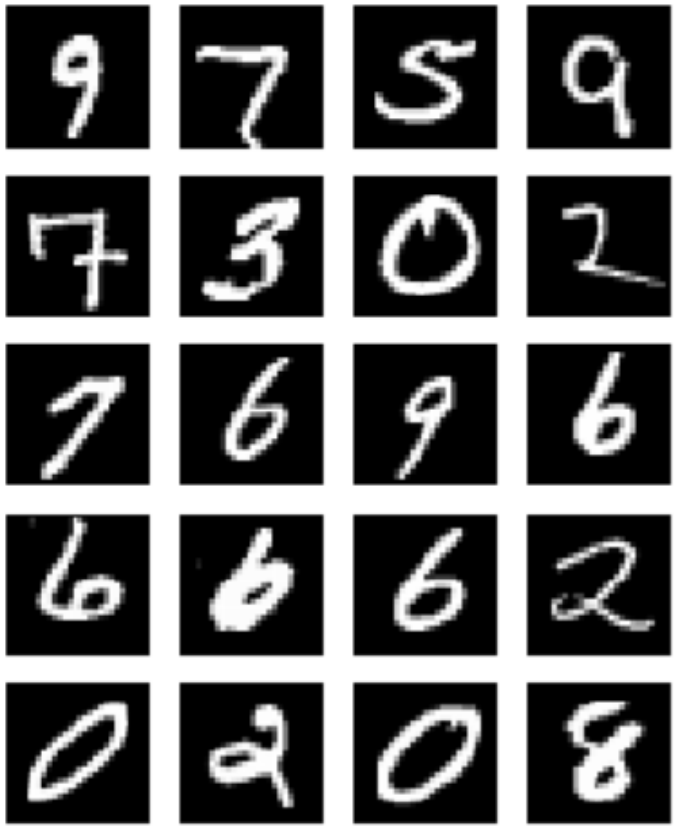


Random samples

➡

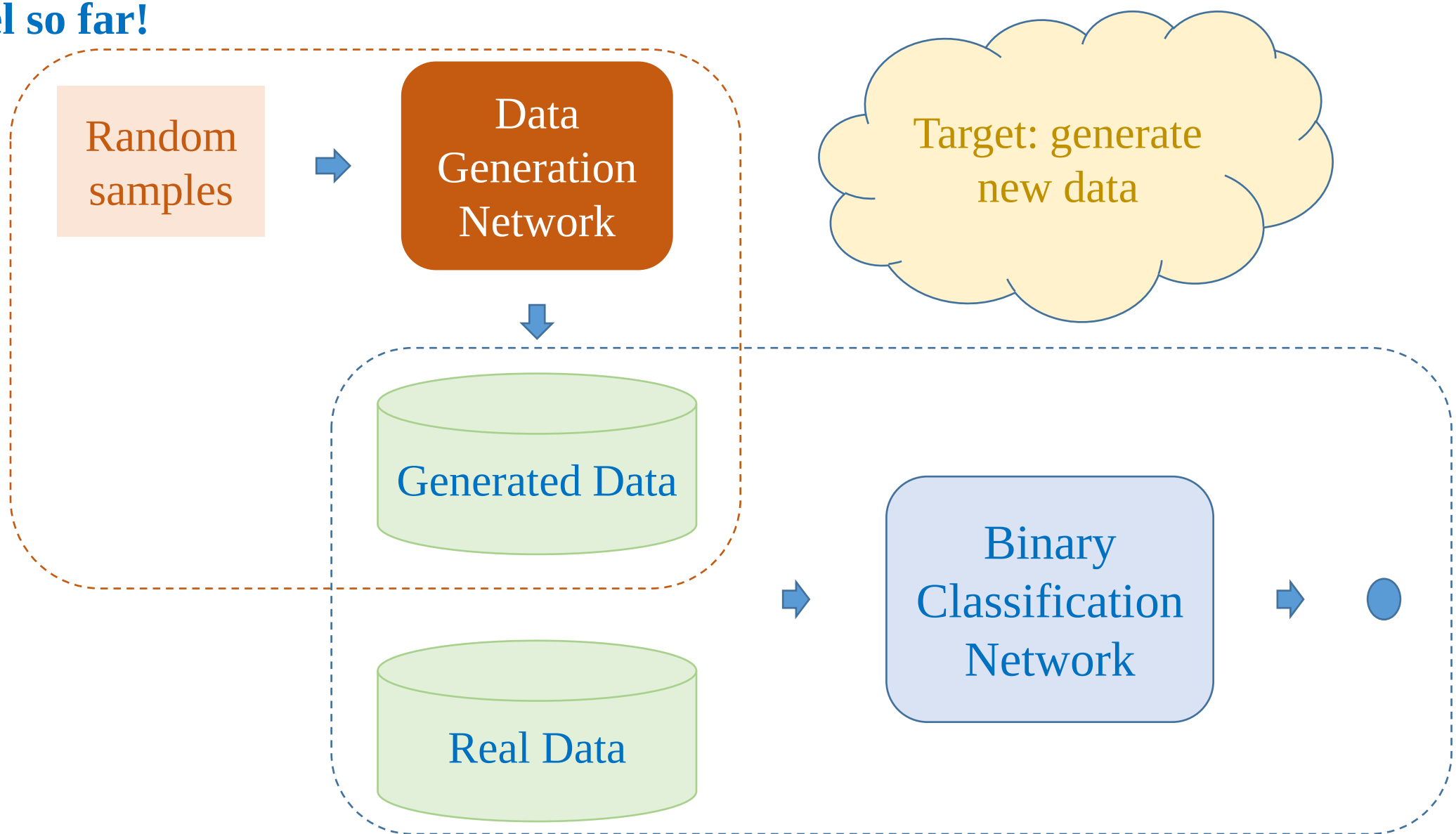
Data Generation Network

➡



Generative Adversarial Networks

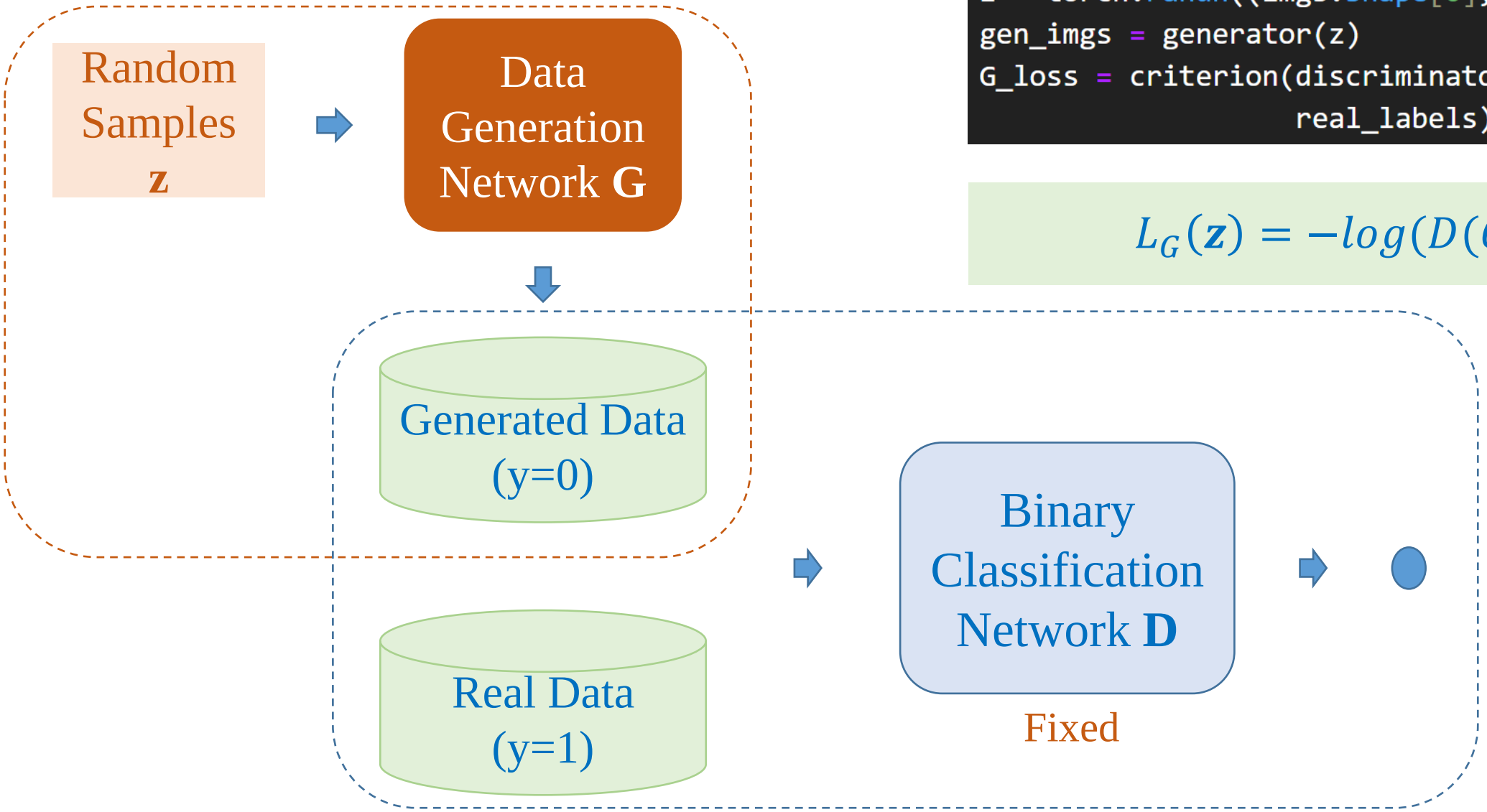
Model so far!



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)
```

Model so far!

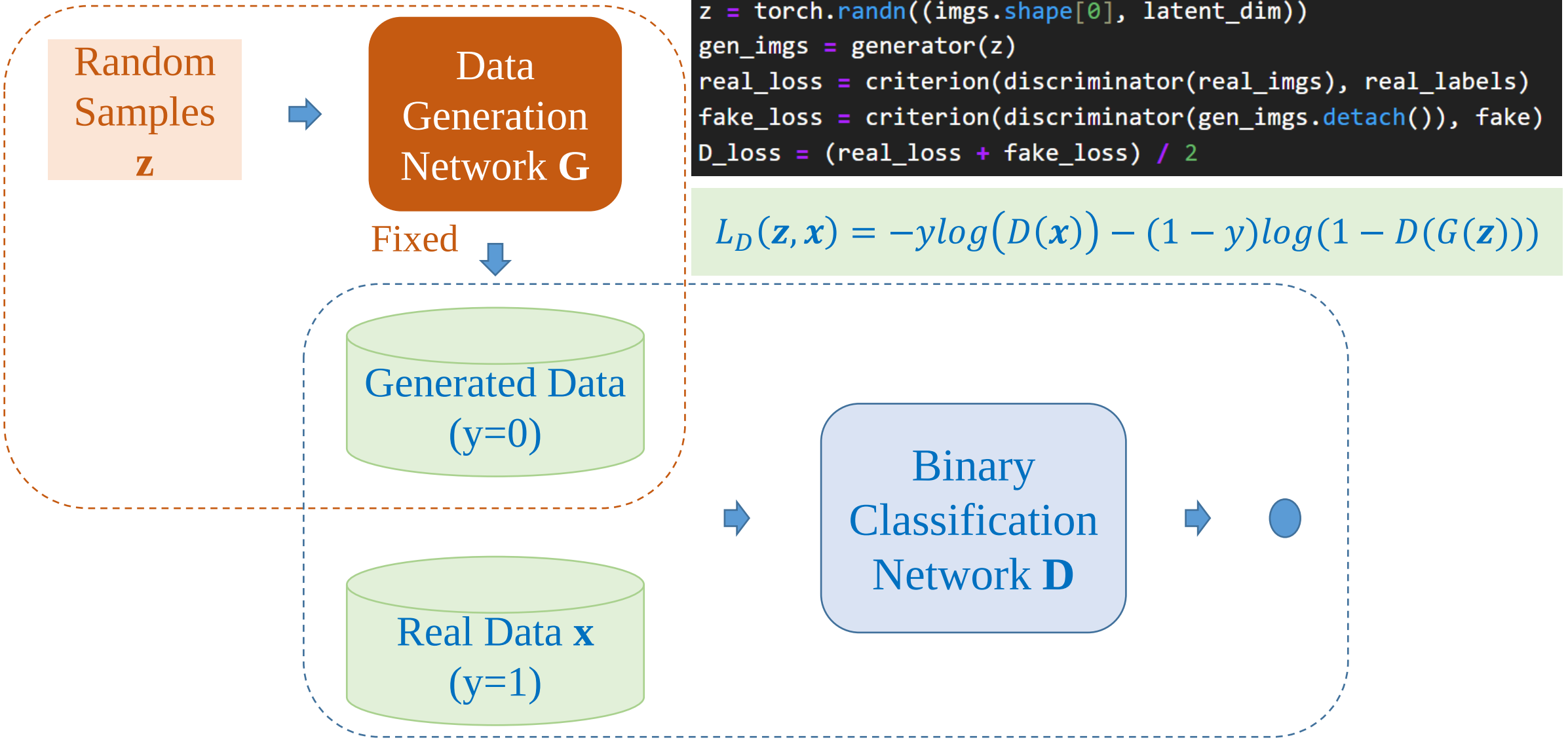


$$L_G(\boldsymbol{z}) = -log(D(G(\boldsymbol{z})))$$

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
```

Model so far!



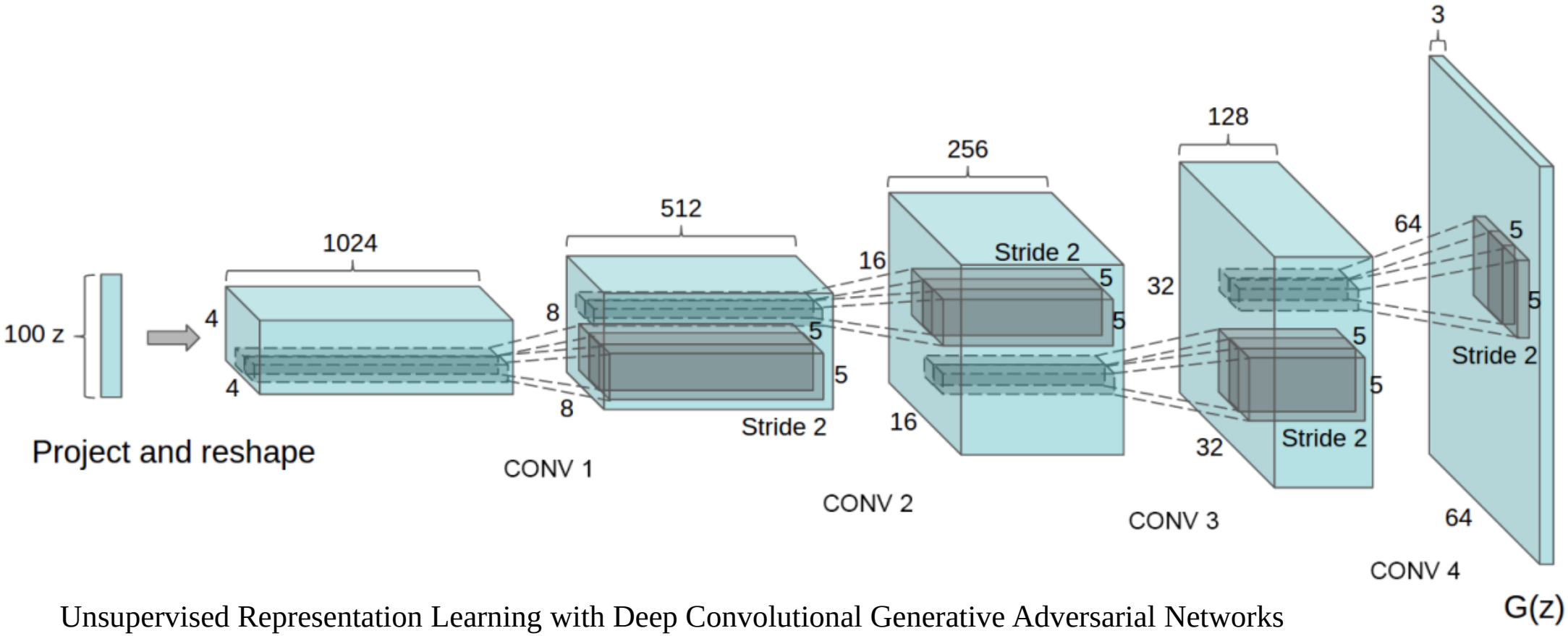
$$L_D(\boldsymbol{z}, \boldsymbol{x}) = -ylog\big(D(\boldsymbol{x})\big) - (1-y)log(1-D(G(\boldsymbol{z})))$$

Review

❖ **DCGANs (2015)**

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.



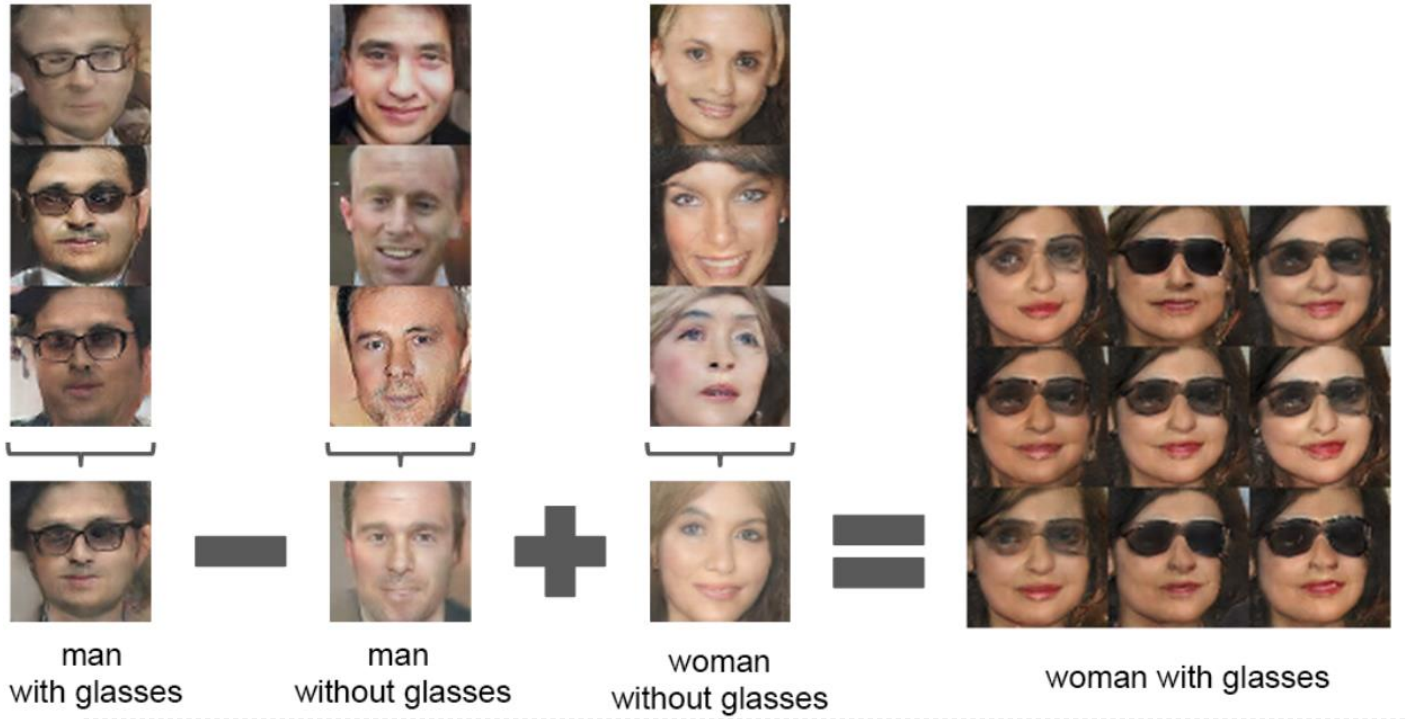
Review

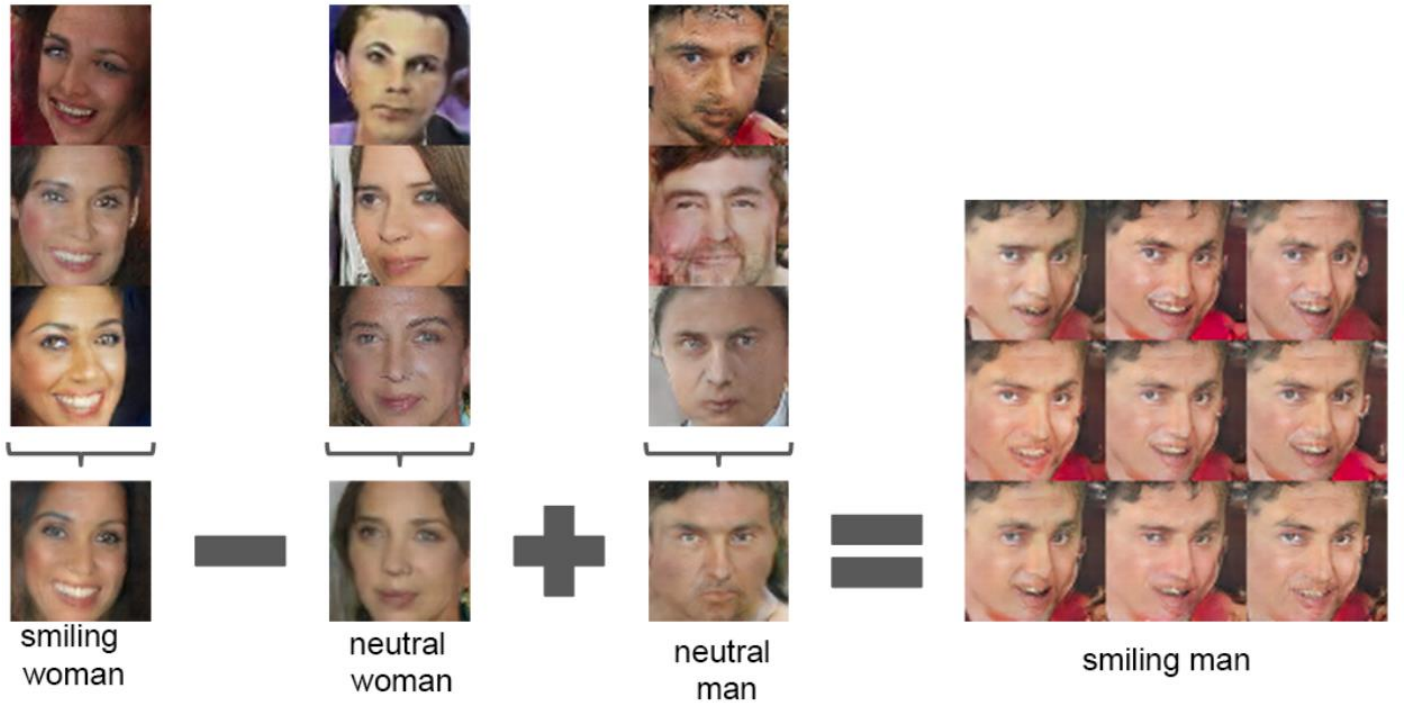
❖ DCGANs (2015)

Review

❖ DCGANs (2015)

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks



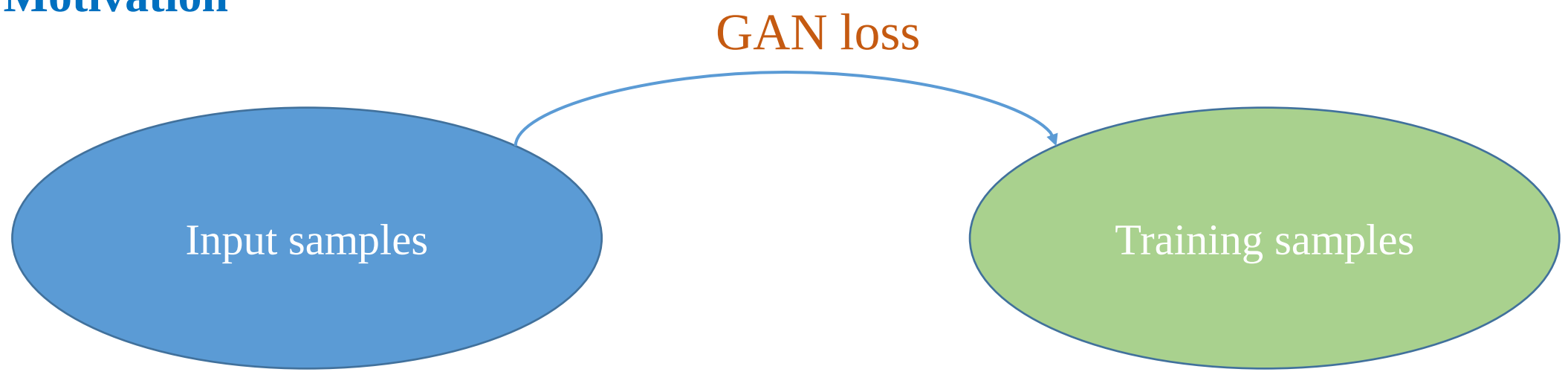


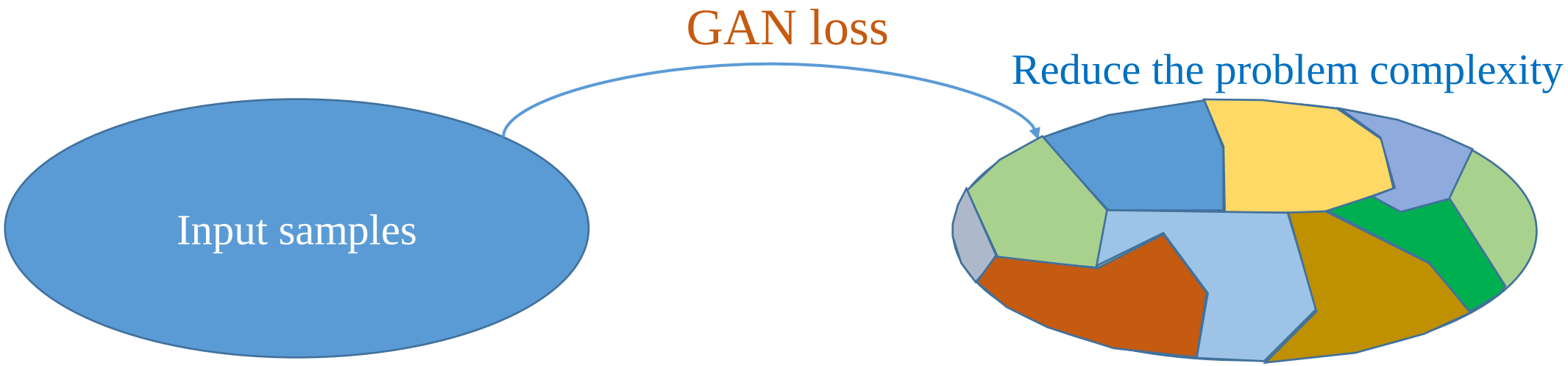
Outline

- Quick Review on GAN/DCGAN
- Conditional GAN
- PatchGAN
- Pix2Pix
- CycleGAN

Conditional GAN (cGAN)

❖ Motivation

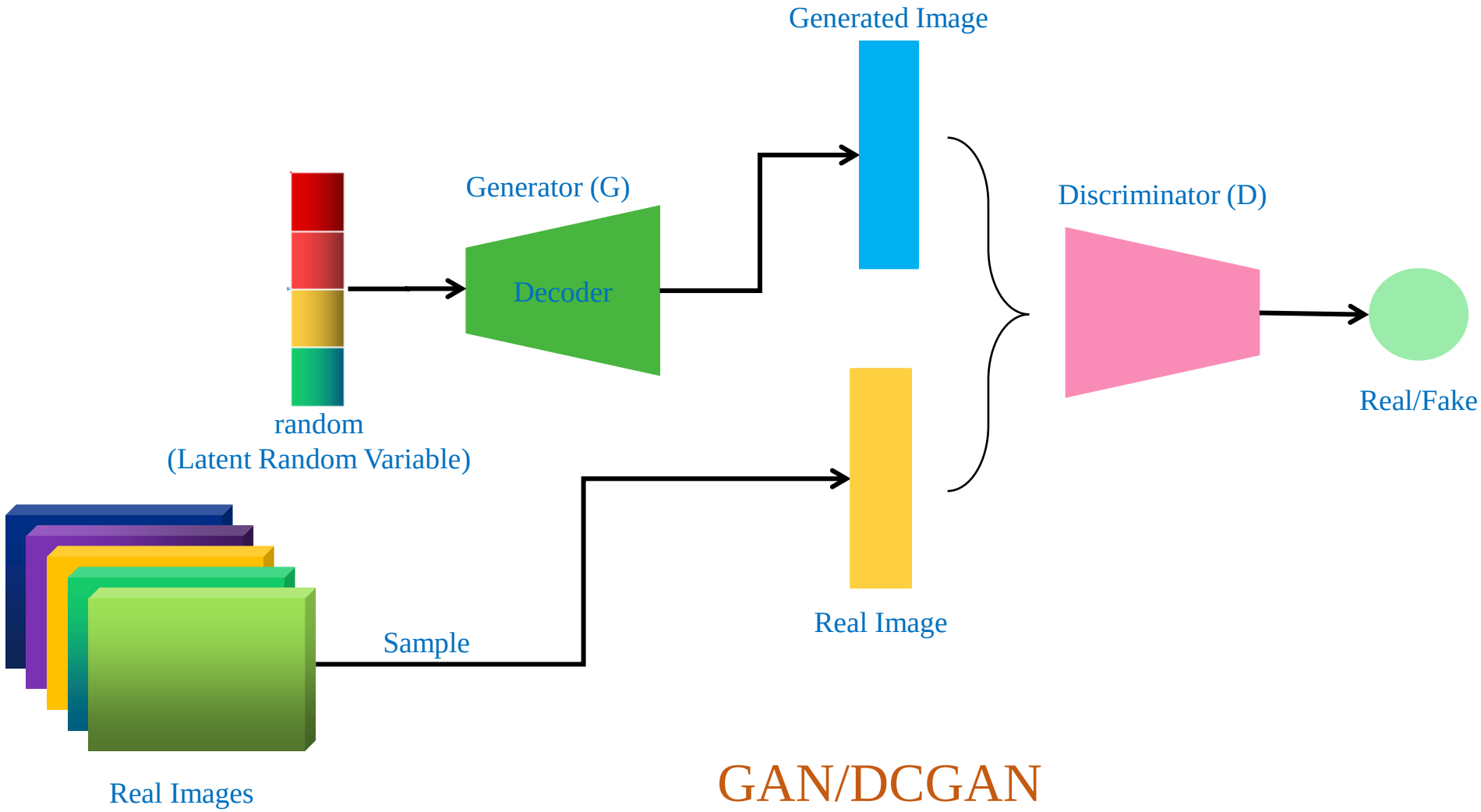




How?

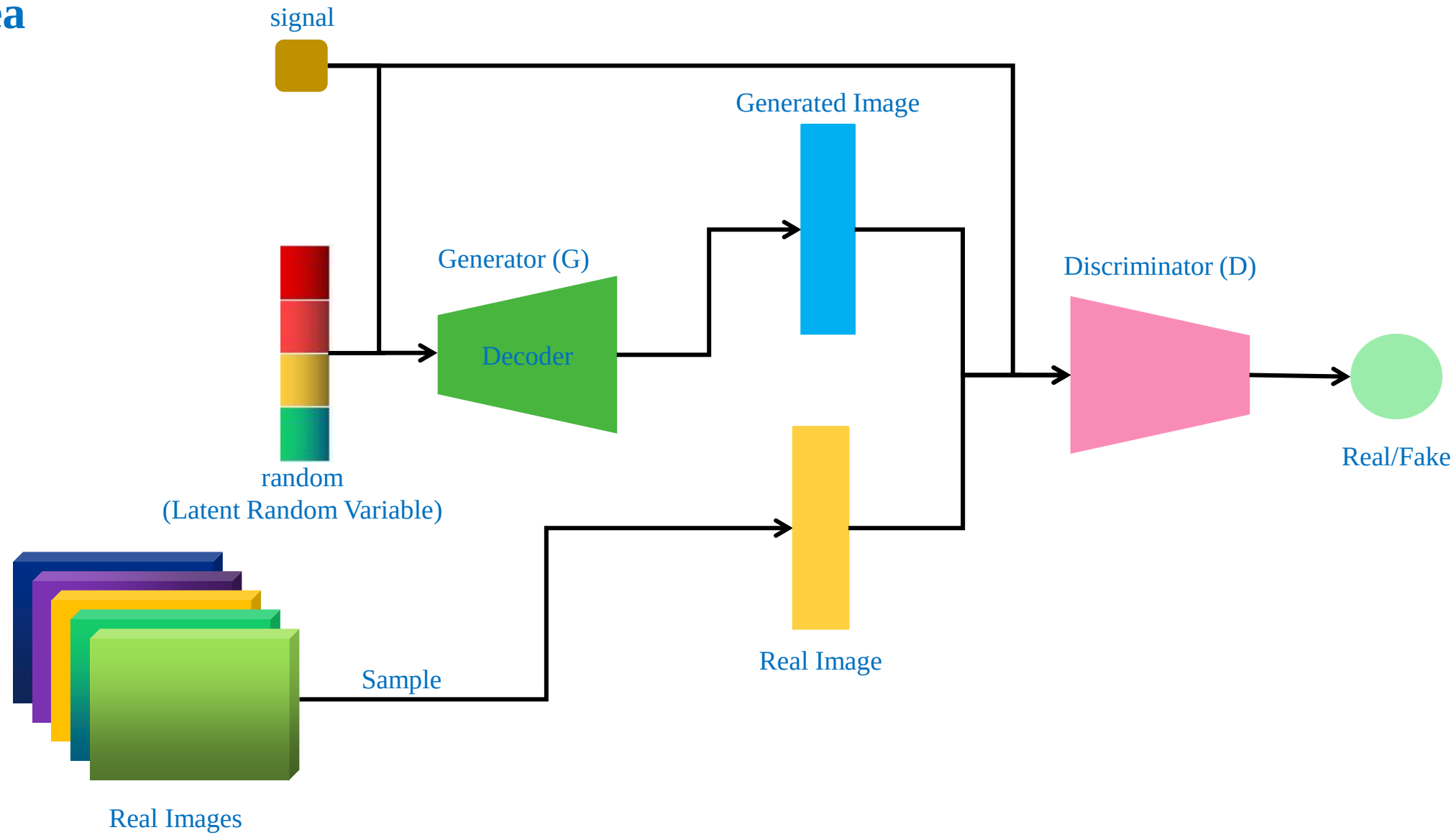
Conditional GAN (cGAN)

❖ Idea?



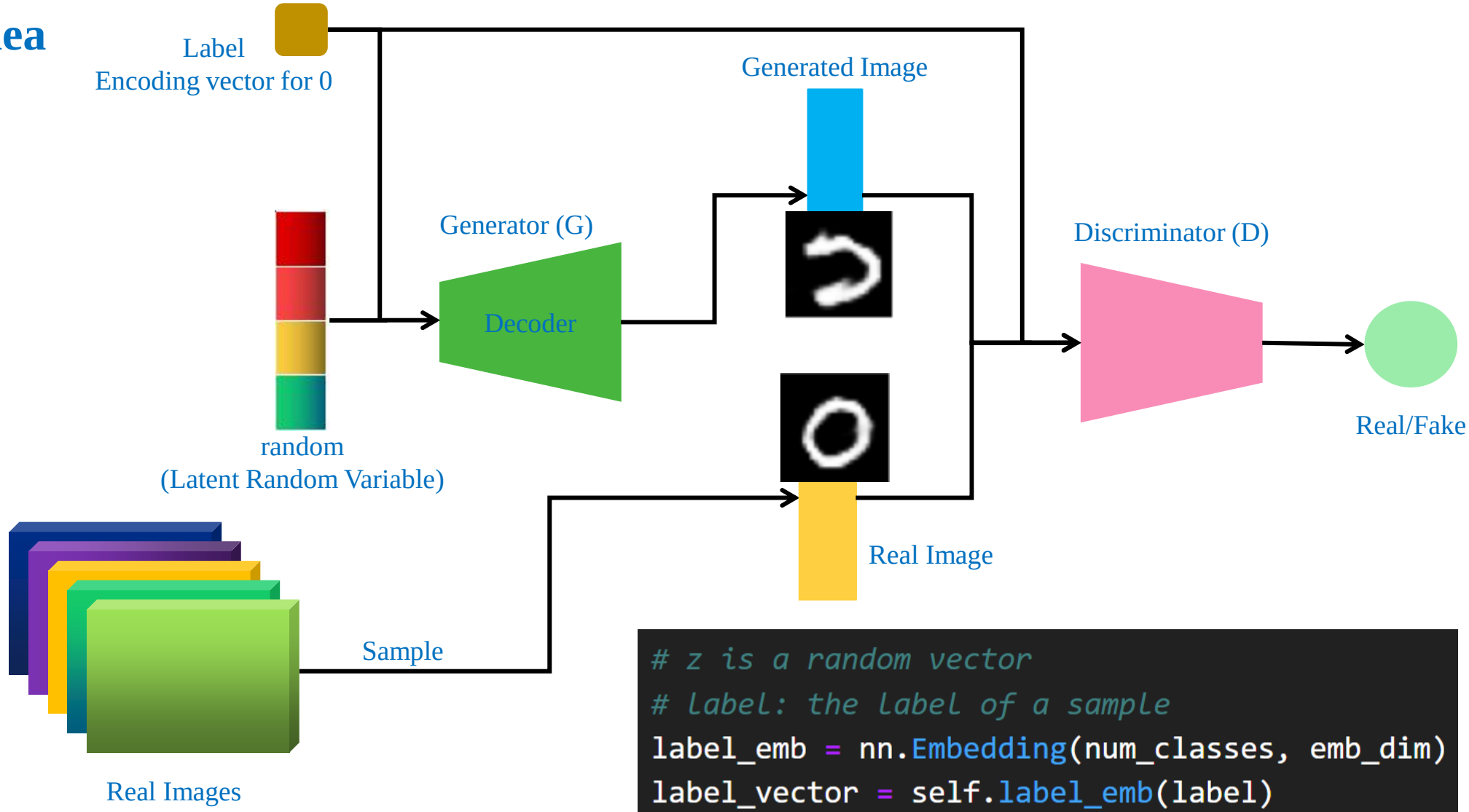
Conditional GAN (cGAN)

❖ Idea



Conditional GAN (cGAN)

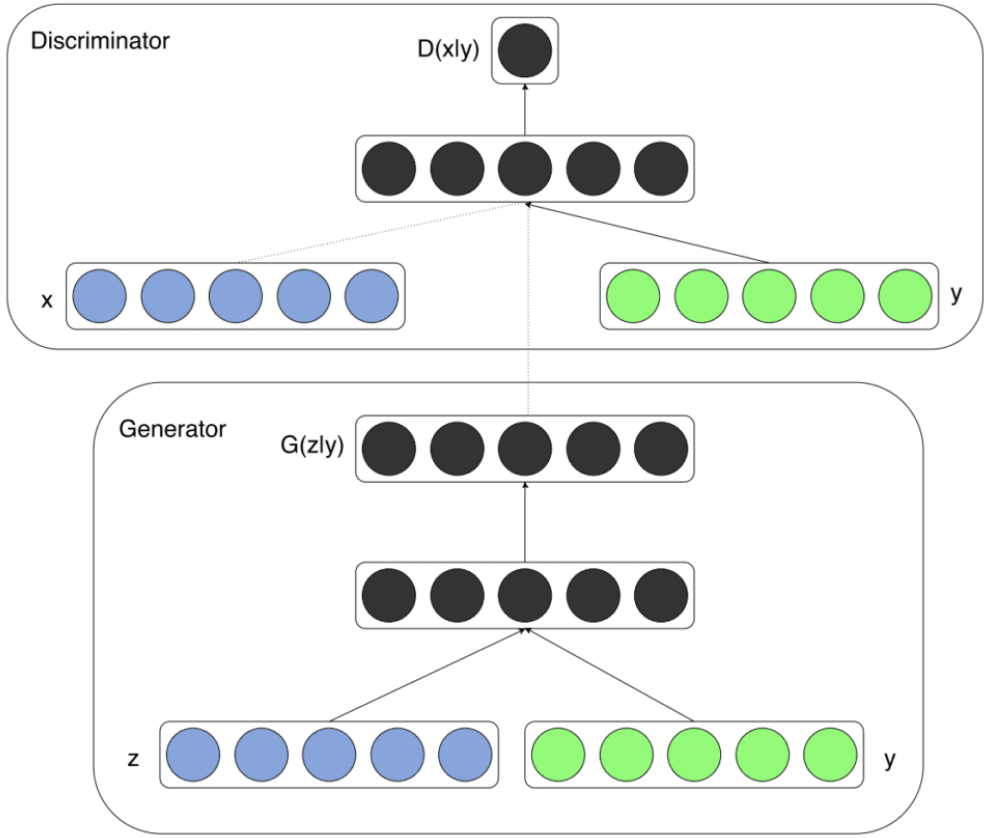
❖ Idea




```
# z is a random vector
# label: the label of a sample
label_emb = nn.Embedding(num_classes, emb_dim)
label_vector = self.label_emb(label)
x = torch.cat([z, label_vector], 1)
```

Conditional GAN

❖ Model: Generator

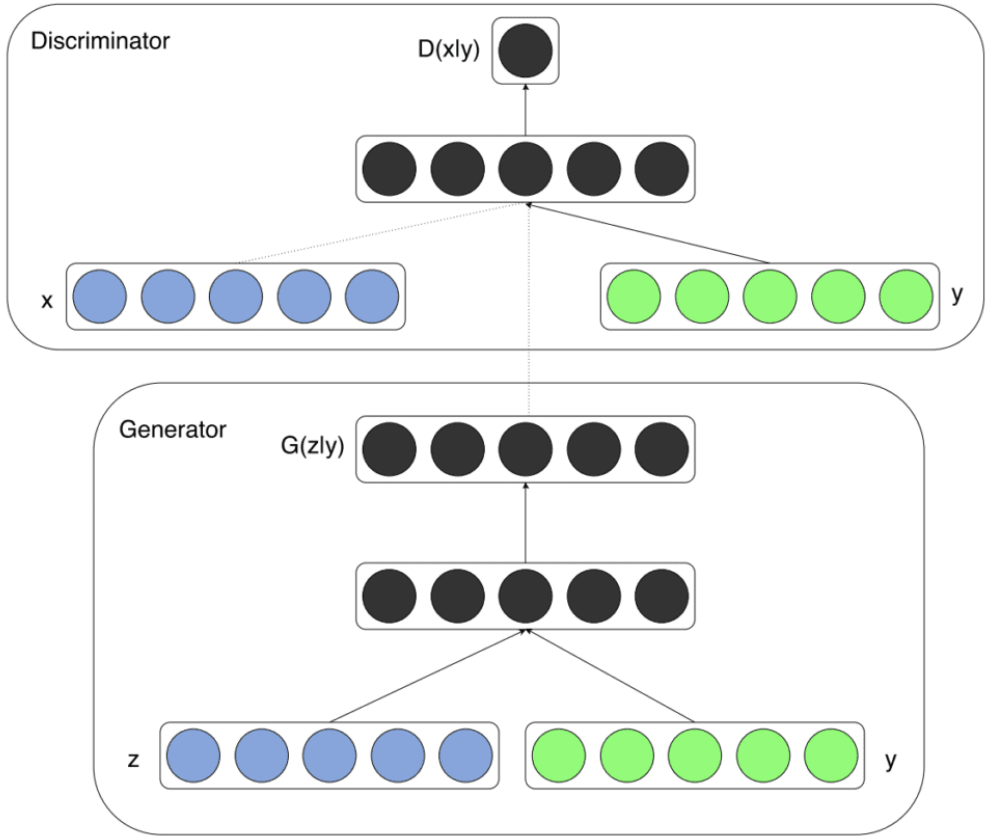


$$L_G(\boldsymbol{z}, \boldsymbol{y}) = -log(D(G(\boldsymbol{z}|\boldsymbol{y})))$$

```
# cGAN - Generator loss
# real_imgs
# labels

# loss and optimizer
criterion = nn.BCELoss()
optimizer_G = torch.optim.Adam(generator.parameters(),
                               lr=0.0001)

# prepare objective data
real_labels = torch.ones((imgs.shape[0], 1))

optimizer_G.zero_grad()

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

# train
gen_imgs = generator(z, labels)
validity = discriminator(gen_imgs, labels)
G_loss = criterion(validity, real_labels)

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

Conditional GAN

❖ Model: Discriminator

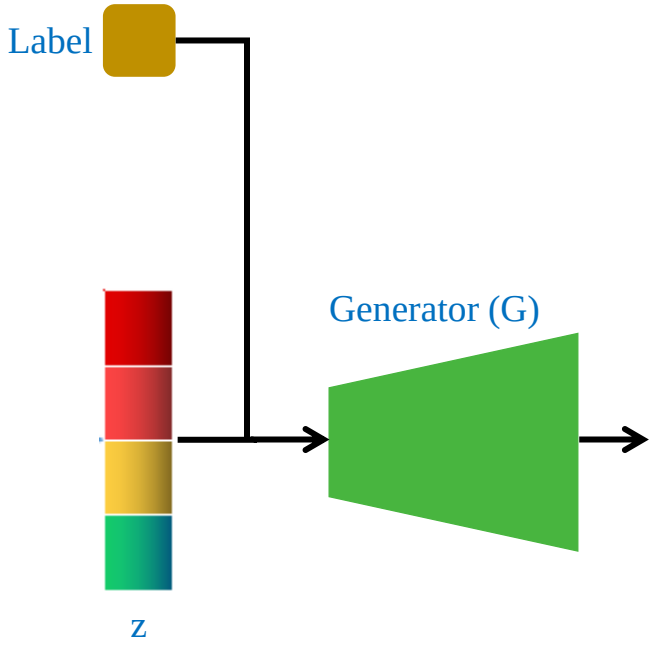


$$L_D(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y}) = -y log\big(D(\boldsymbol{x}|\boldsymbol{y})\big) - (1-y) log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))$$

```python
# cGAN - Discriminator loss
# real_imgs
# labels

# loss and optimizer
criterion = nn.BCELoss()
optimizer_D = torch.optim.Adam(discriminator.parameters(),
                               lr=0.0002)

# prepare objective data
real_labels = torch.ones((imgs.shape[0], 1))
fake_labels = torch.zeros((imgs.shape[0], 1))
optimizer_D.zero_grad()

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

# train
real_validity = discriminator(real_imgs, labels)
real_loss = criterion(real_validity, real_labels)
fake_validity = discriminator(gen_imgs.detach(), labels)
fake_loss = criterion(fake_validity, fake_labels)
D_loss = (real_loss + fake_loss) / 2

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

Conditional GAN (cGAN)

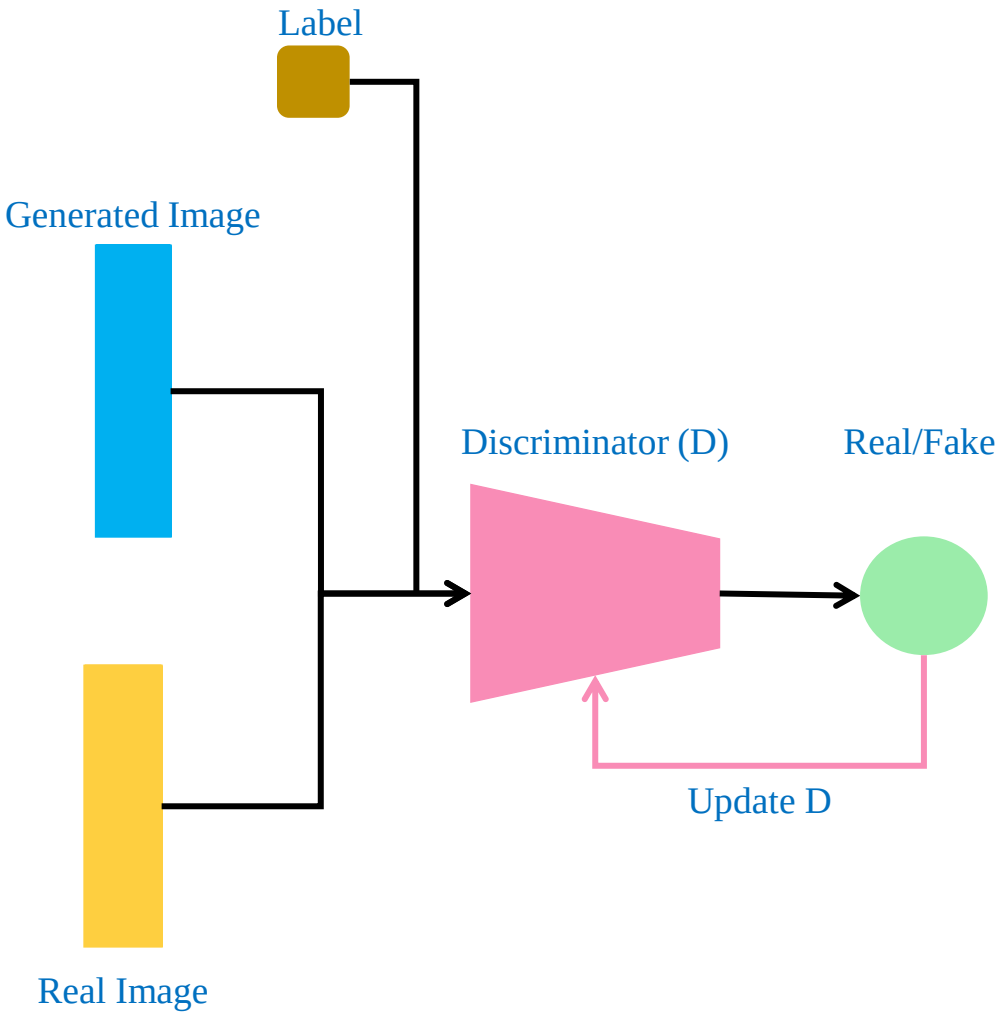
❖ Implementation




```python
class Generator(nn.Module):
    def __init__(self, num_classes, emb_dim):
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, emb_dim)

        self.model = nn.Sequential(
            nn.Linear(100+emb_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, label):
        cond = self.label_emb(label)
        x = torch.cat([z, cond], 1)
        img = self.model(x)
        img = img.view(img.size(0), *img_shape)
        return img
```

Conditional GAN (cGAN)

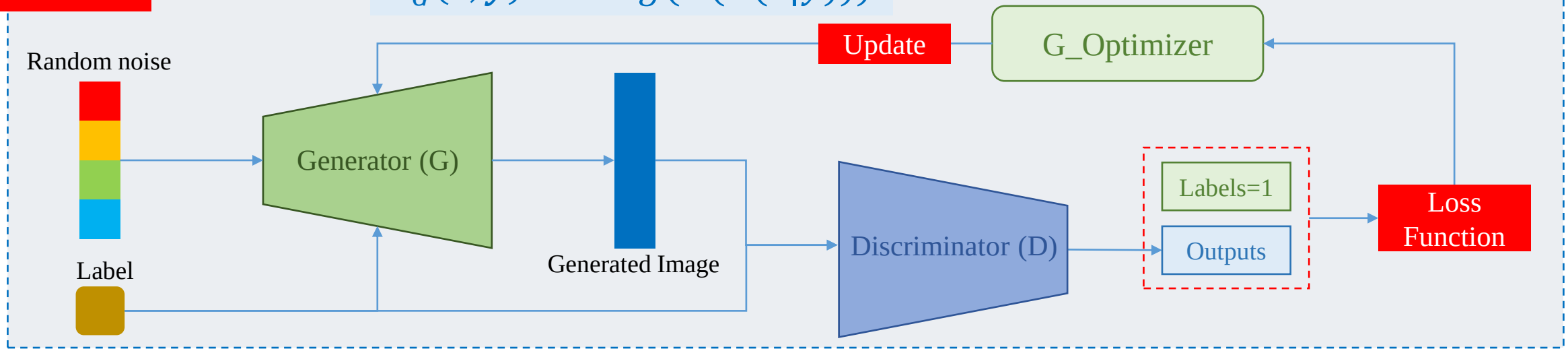
❖ Implementation



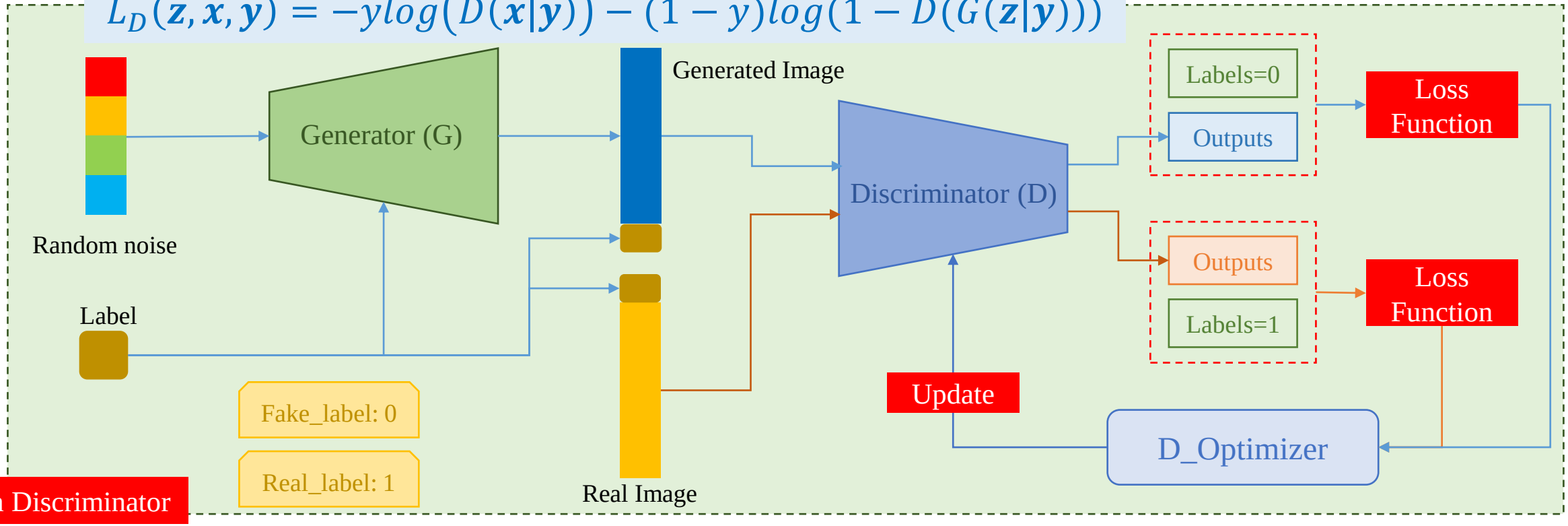

```python
class Descriminator(nn.Module):
    def __init__(self, num_classes, emb_dim):
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, emb_dim)

        self.model = nn.Sequential(
            nn.Linear(32*32 + emb_dim, 512),
            nn.LeakyReLU(0.2, inplace=True),

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

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

    def forward(self, img, label):
        img_flat = img.view(img.size(0), -1)
        cond = self.label_emb(label)
        x = torch.cat([img_flat, cond], 1)
        validity = self.model(x)
        return validity
```

Train Generator

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



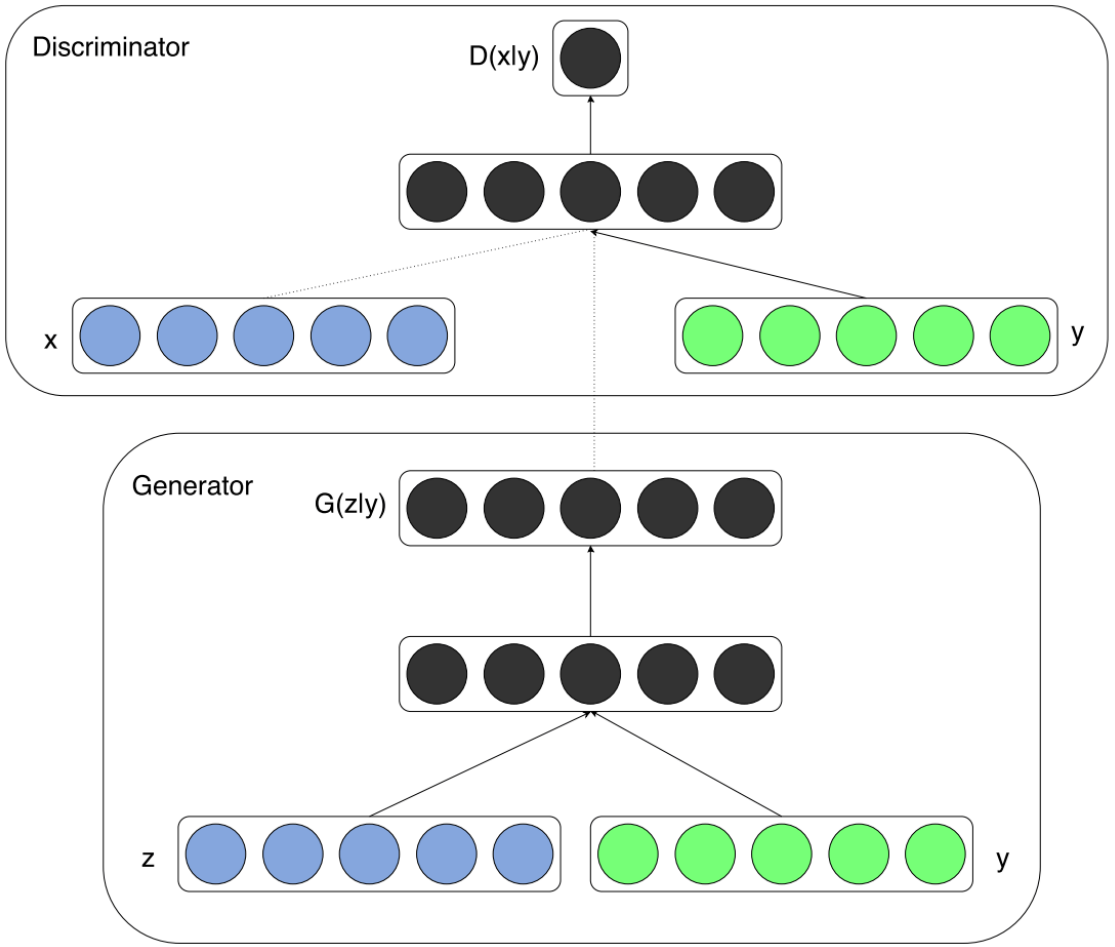
$$L_D(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y}) = -ylog\big(D(\boldsymbol{x}|\boldsymbol{y})\big) - (1-y)log(1-D(G(\boldsymbol{z}|\boldsymbol{y})))$$

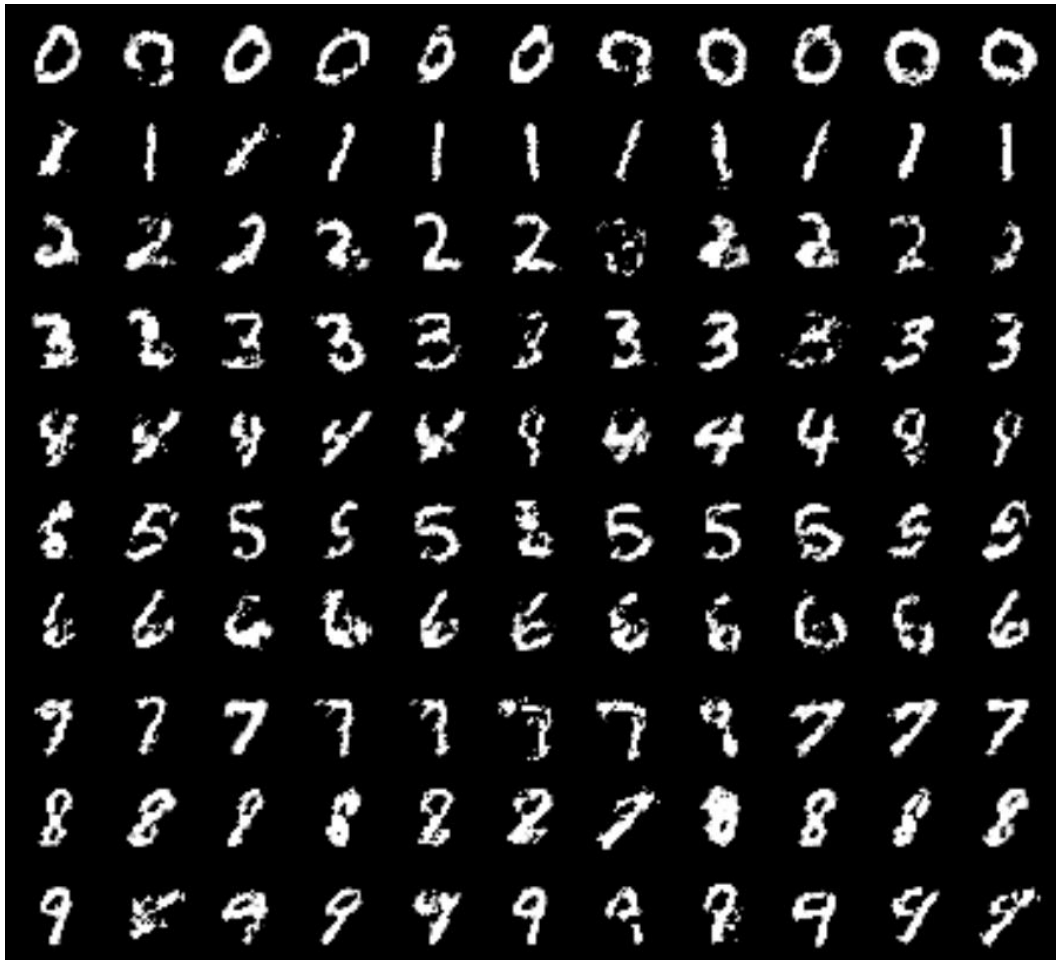


Train Discriminator

GANs Survey

❖ Conditional Generative Adversarial Nets



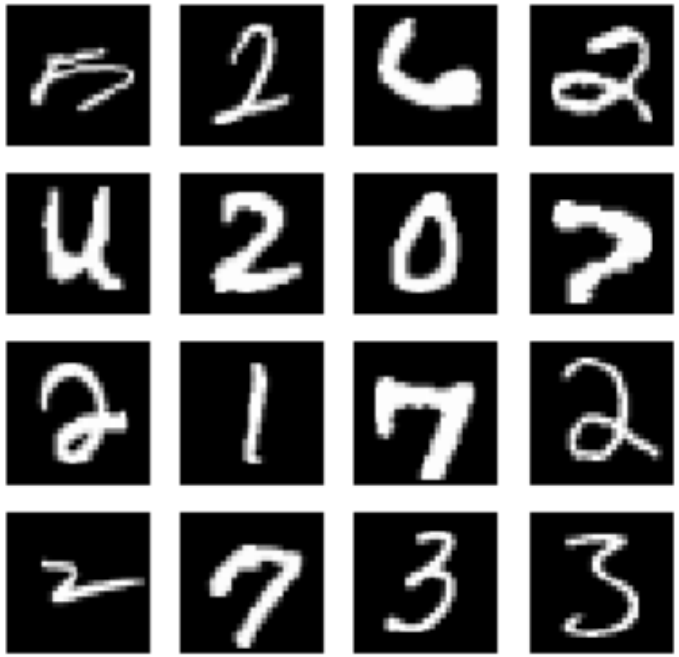


Conditional Generative Adversarial Nets, https://arxiv.org/pdf/1411.1784.pdf

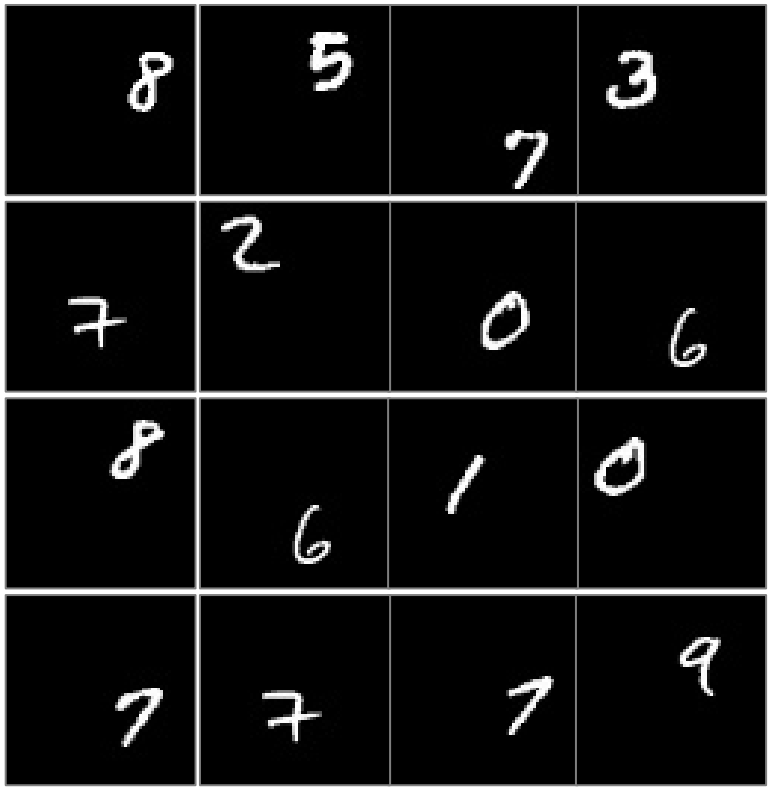
Outline

- Quick Review on GAN/DCGAN
- Conditional GAN
- PatchGAN
- Pix2Pix
- CycleGAN

❖ From detection problem



Resolution = 3x32x32



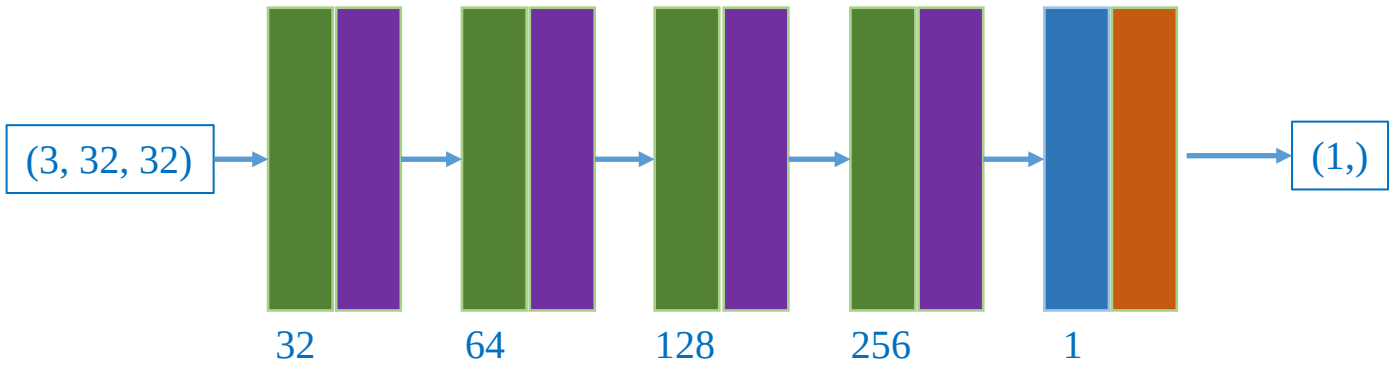
Resolution = 3x64x64

(3x3) Convolution padding='same' + BN+LeakyReLU

(2x2) Max Pooling

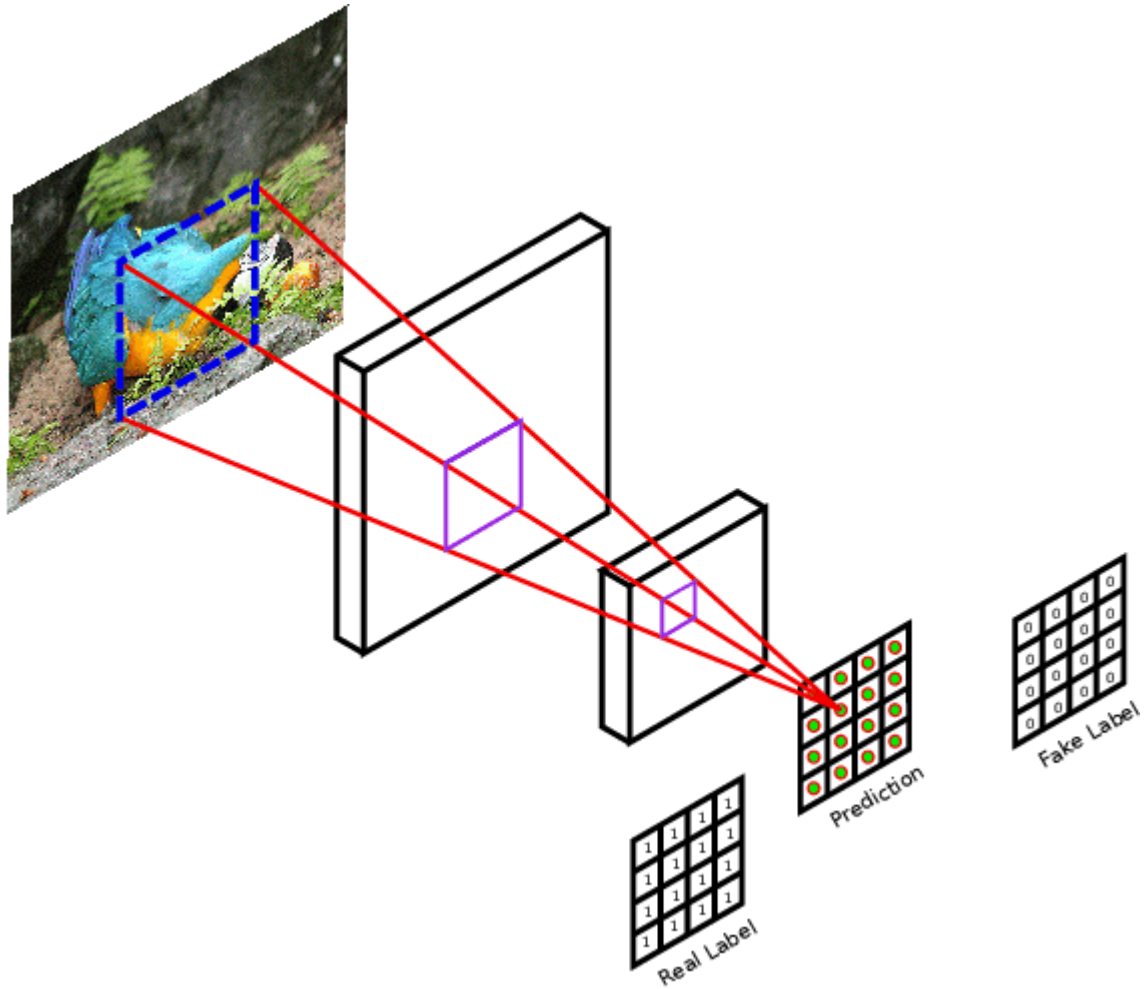
(2x2) Convolution

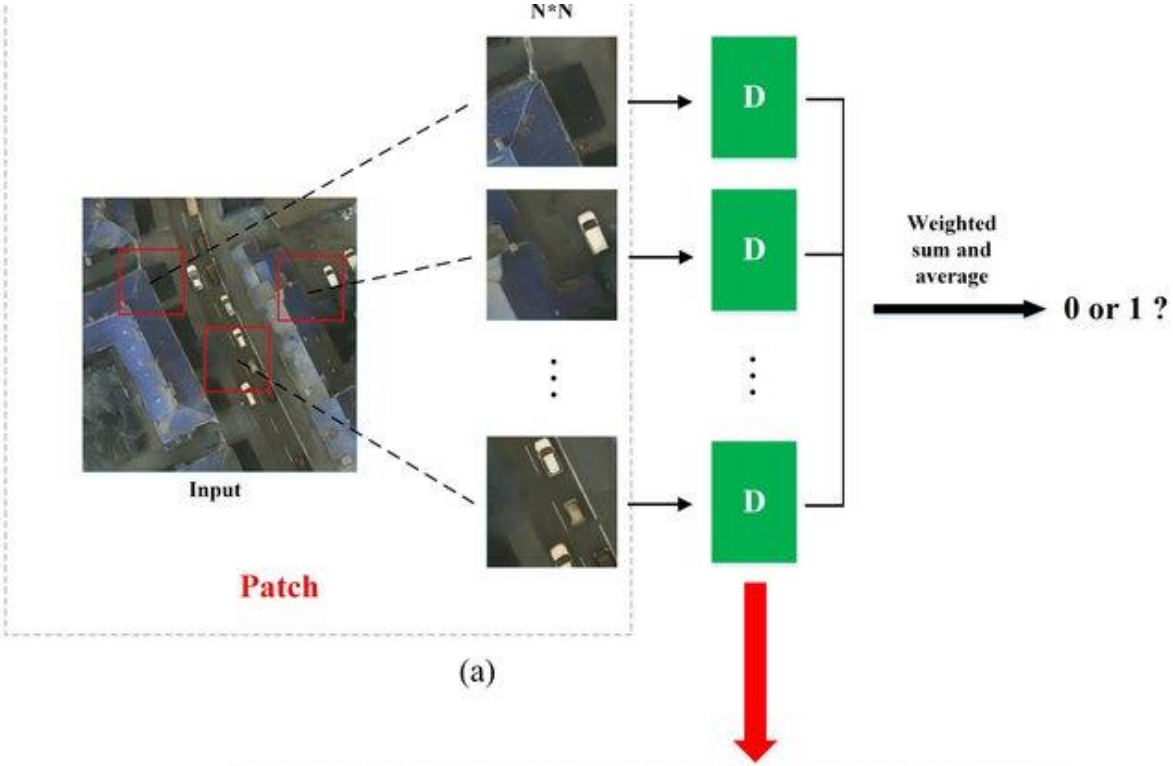
sigmoid

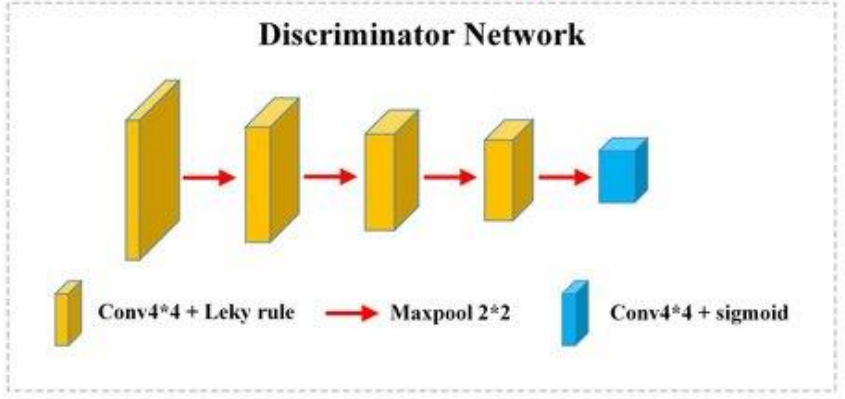


PatchGAN

❖ Idea







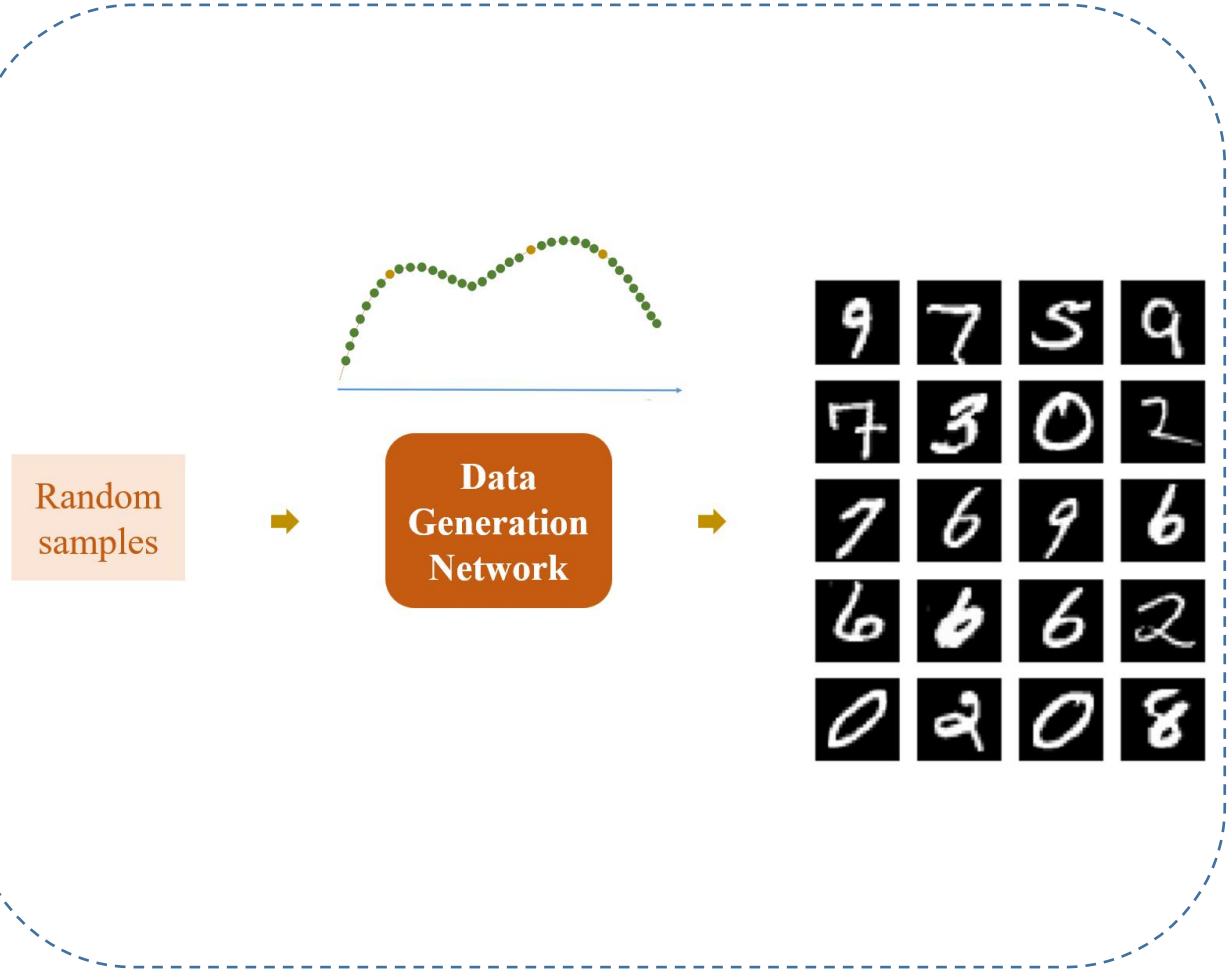
Patch-Based Image Inpainting with Generative Adversarial Networks

Outline

- Quick Review on GAN/DCGAN
- Conditional GAN
- PatchGAN
- Pix2Pix
- CycleGAN

Pix2Pix

❖ Motivation



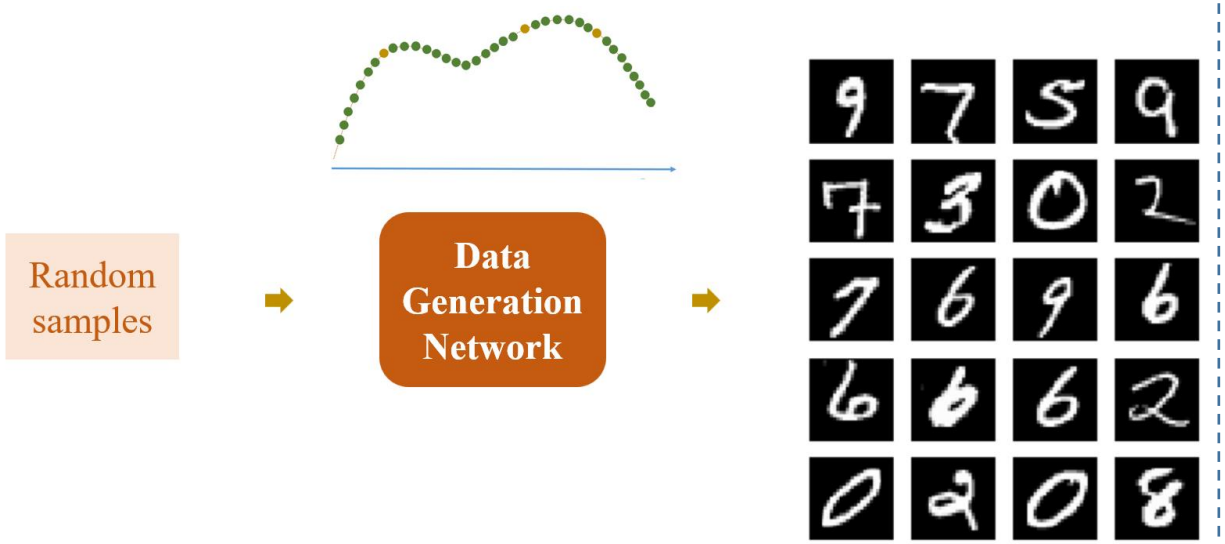
Convert from a noise space to an image space

Convert from an image space to another image space

AI VIETNAM
All-in-One Course

Pix2Pix

❖ Problems of GAN/DCGAN



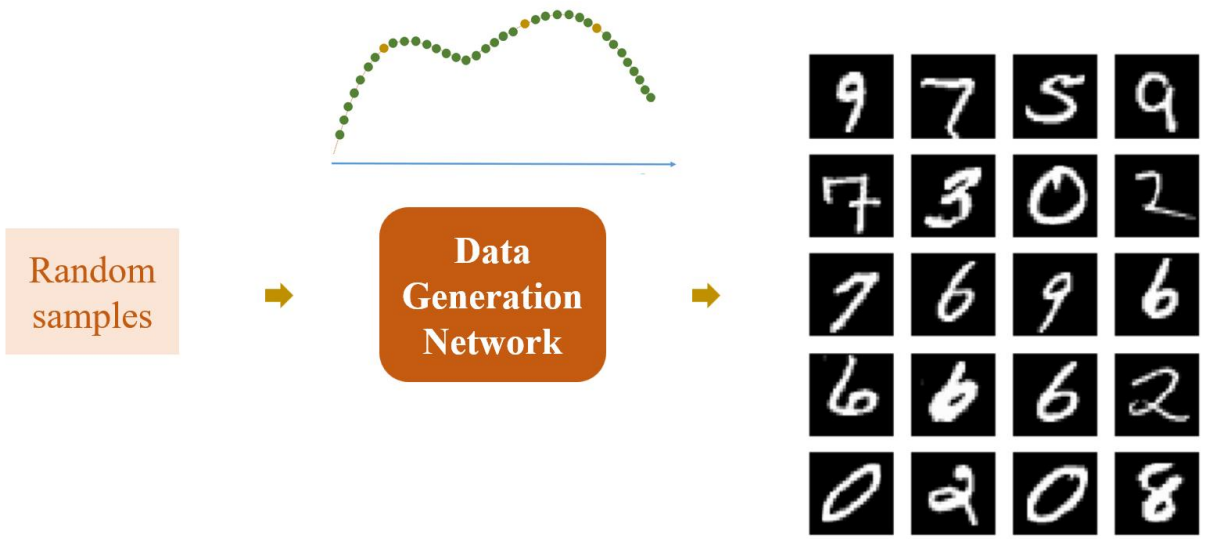
Convert from a noise space to an image space

```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
```

AI VIETNAM
All-in-One Course

Pix2Pix

❖ Problems of GAN/DCGAN

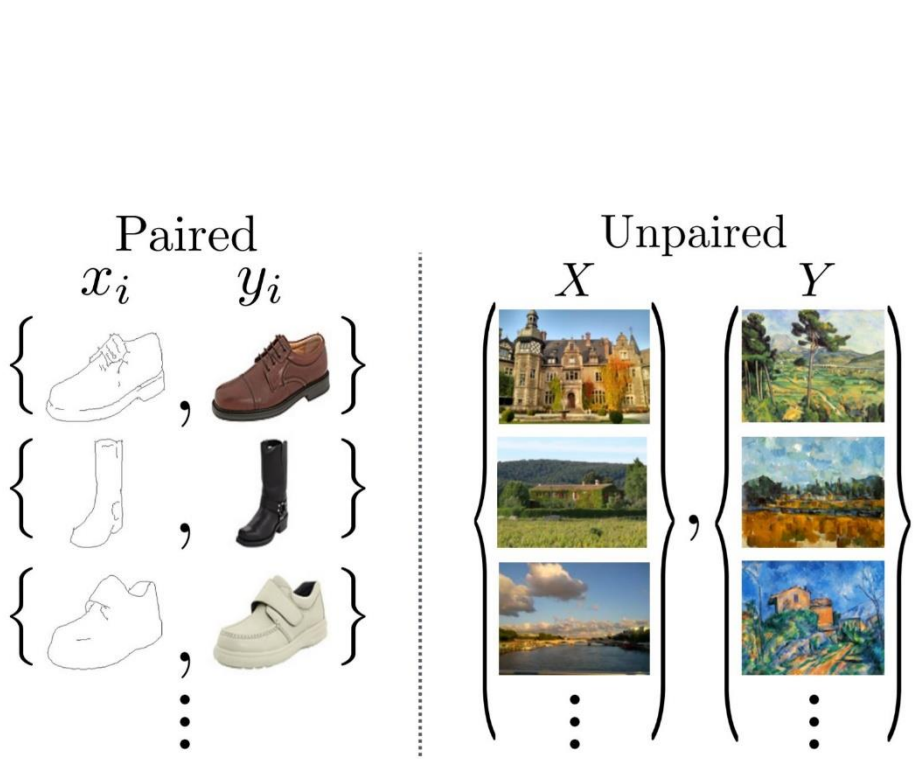


Convert from a noise space to an image space

```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
```

Pix2Pix

❖ Network



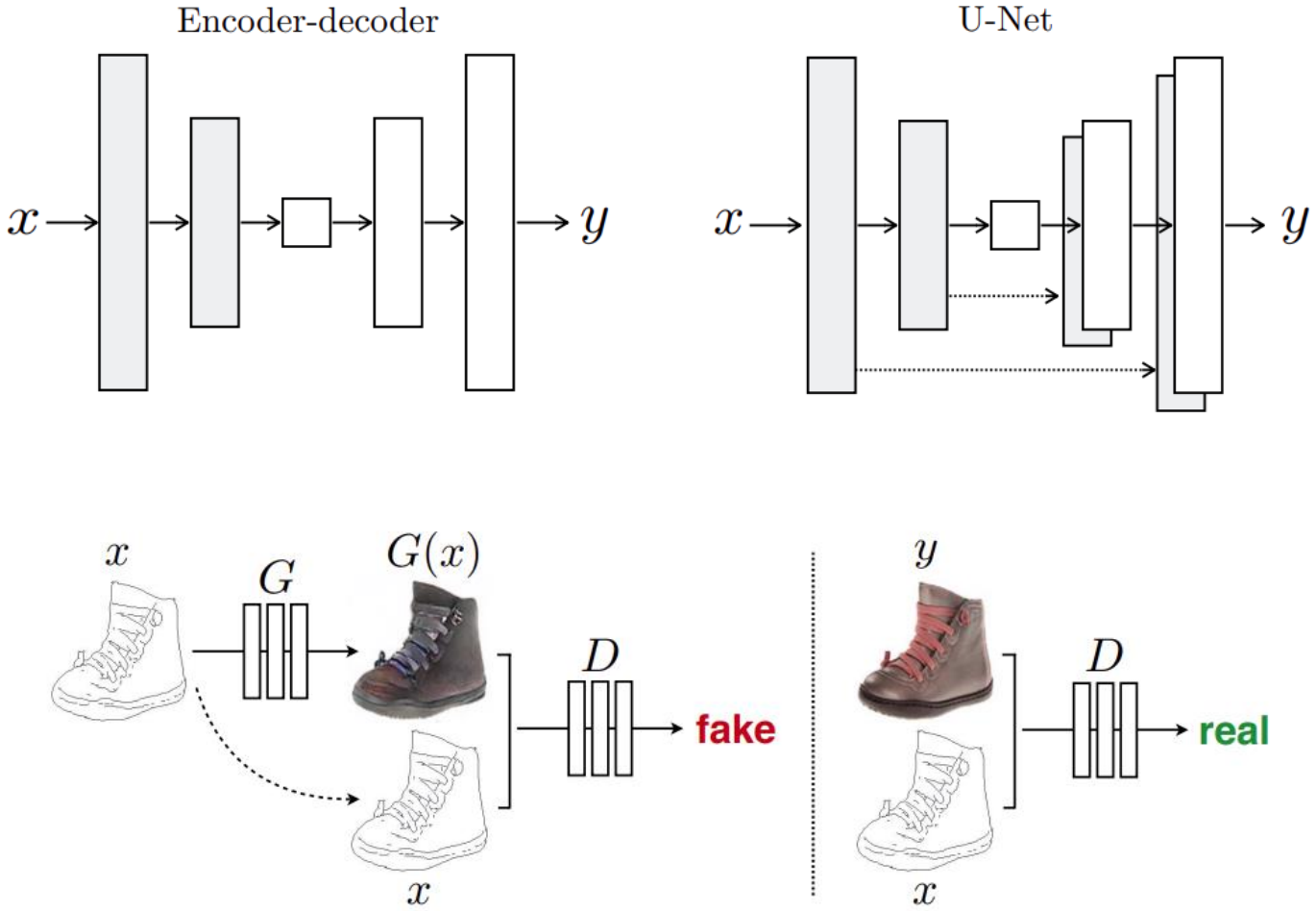
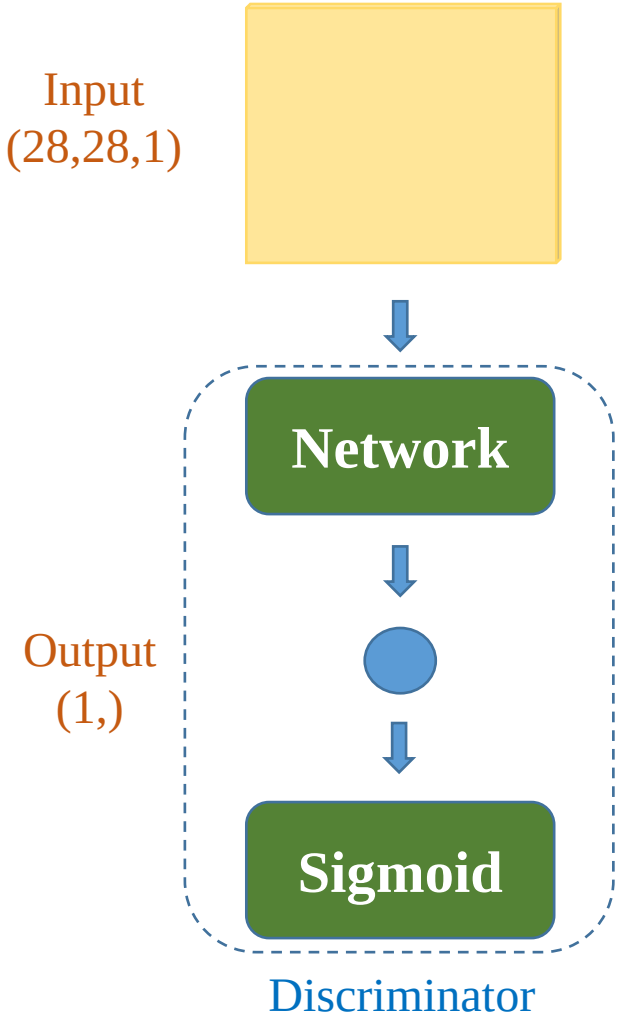
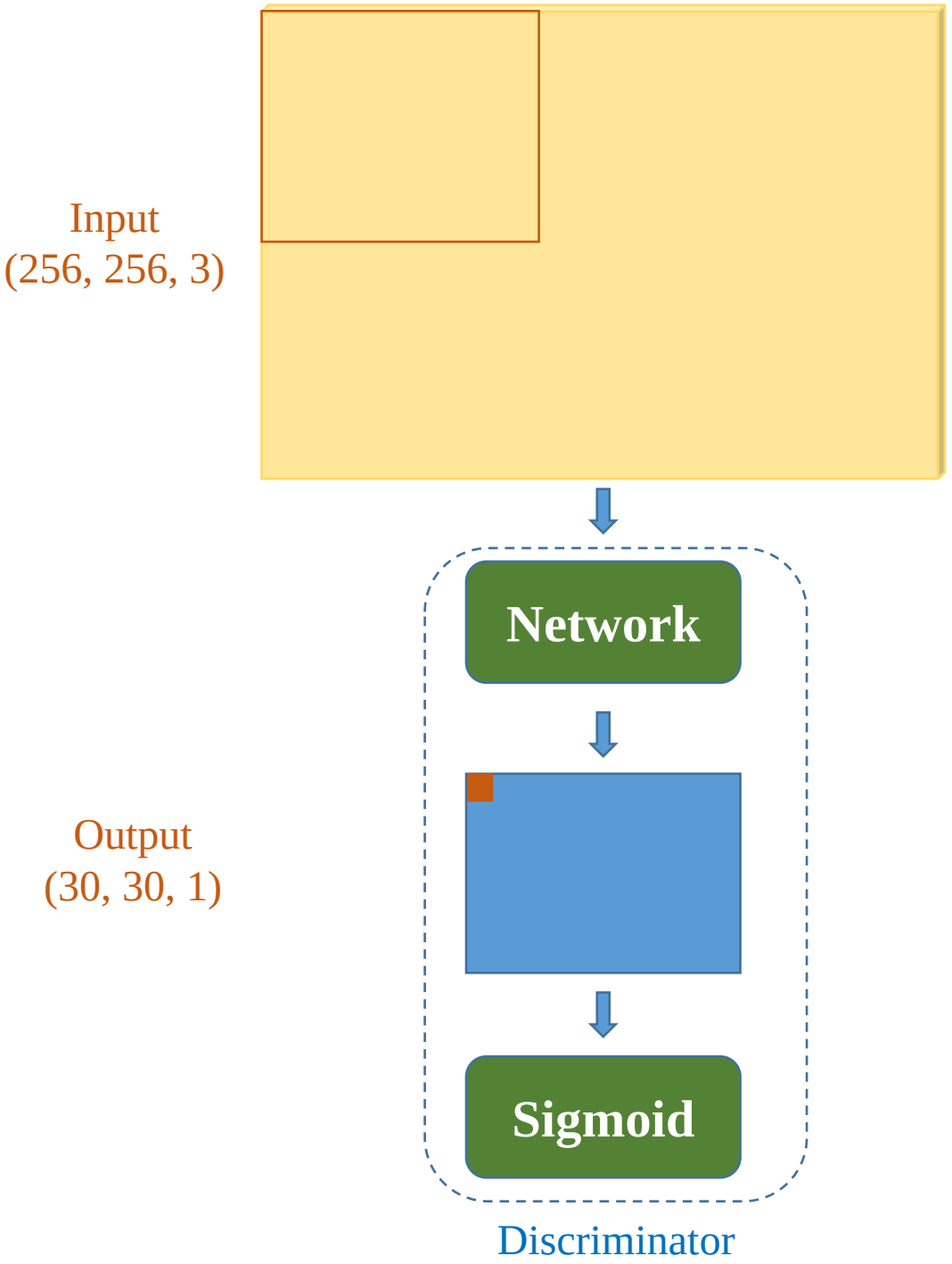


Image-to-Image Translation with Conditional Adversarial Networks, https://arxiv.org/pdf/1611.07004.pdf

Pix2Pix

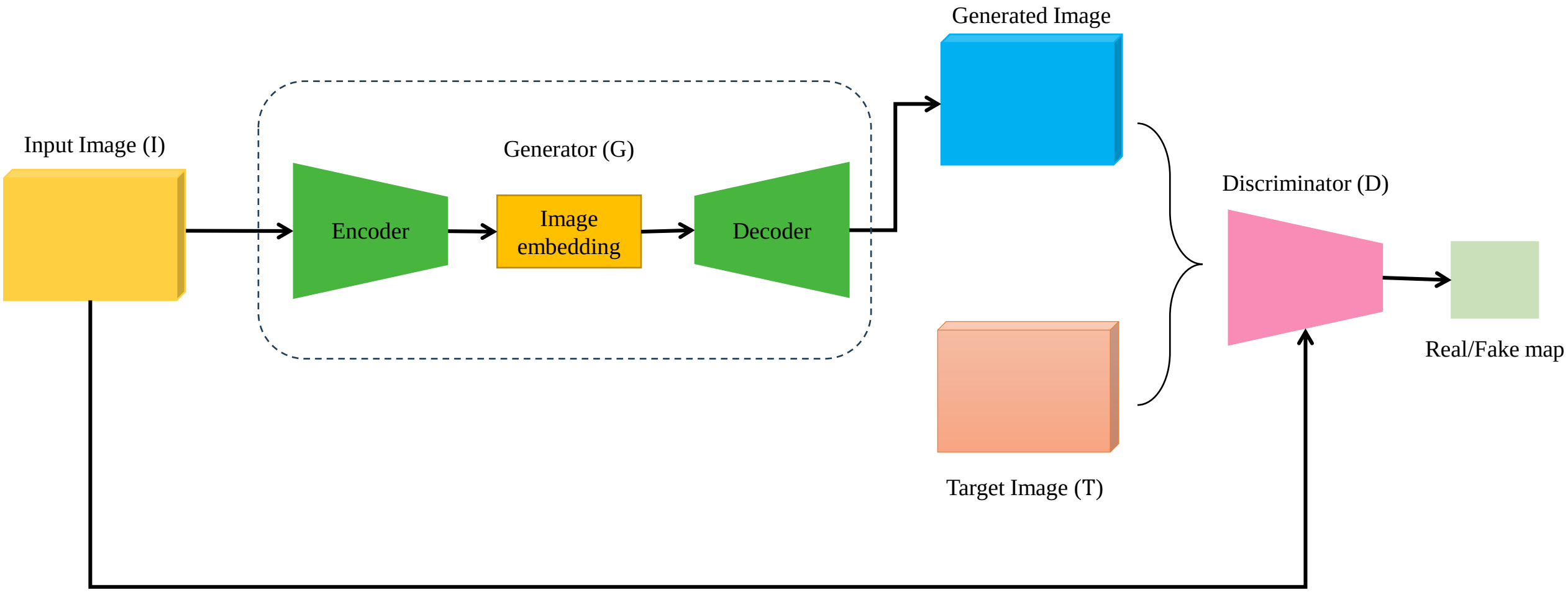
❖ Loss function



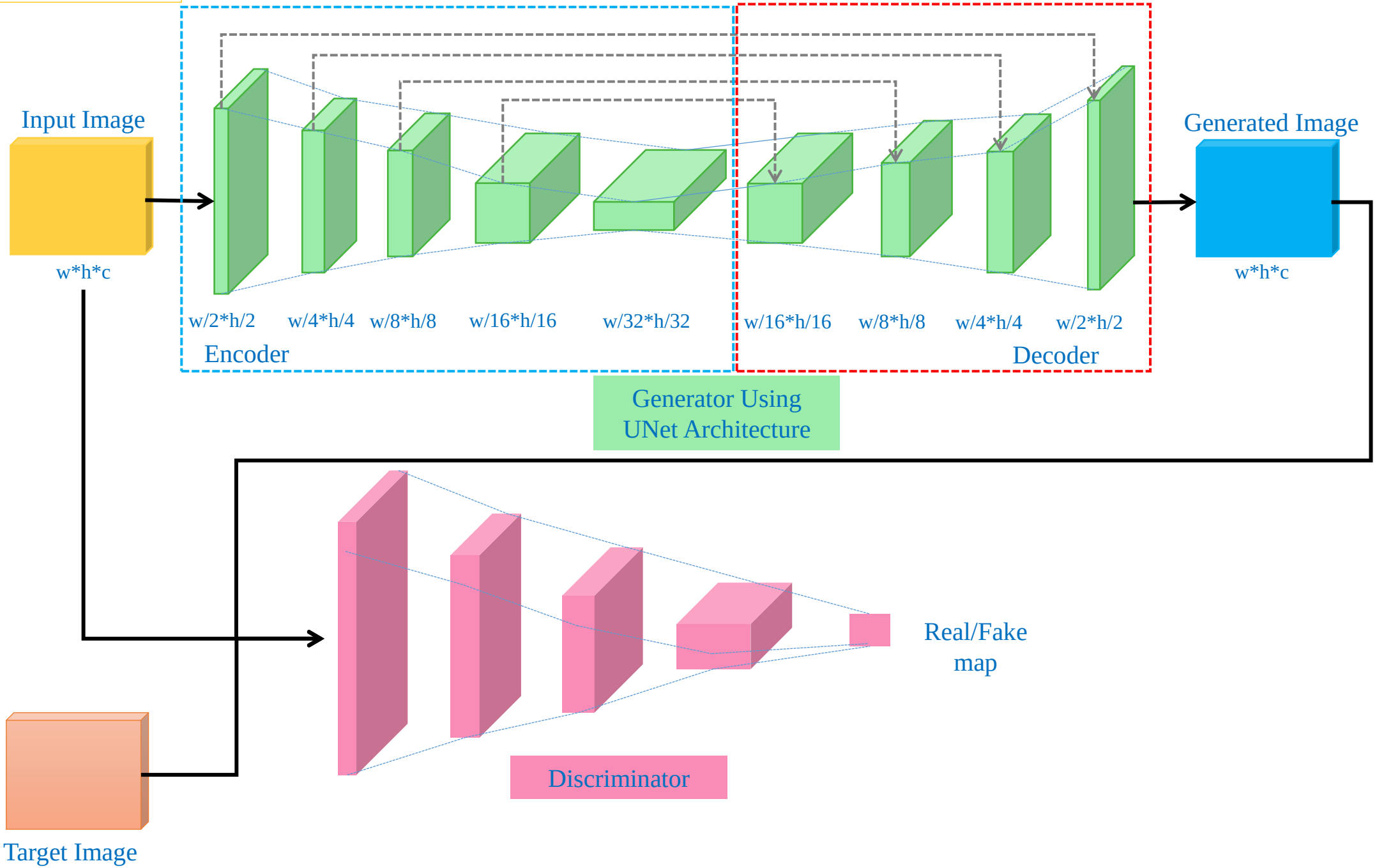


Pix2Pix

❖ Model

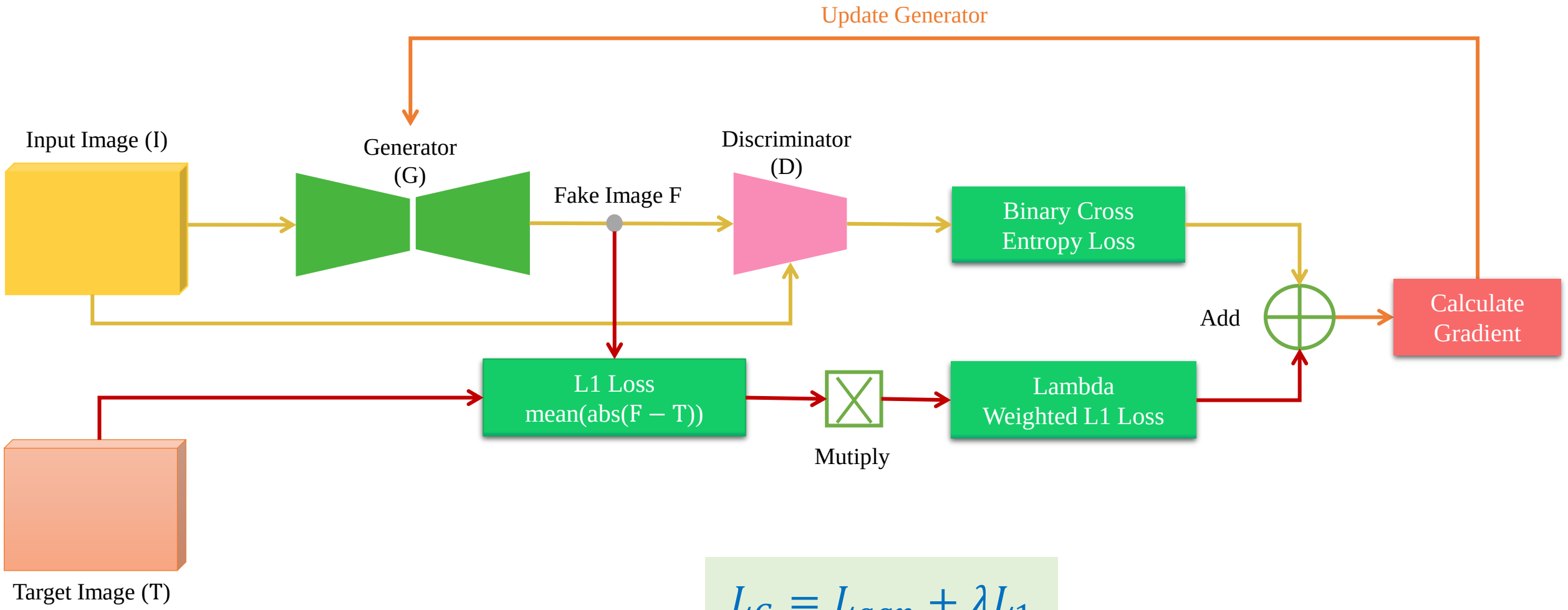


Pix2Pix Architecture

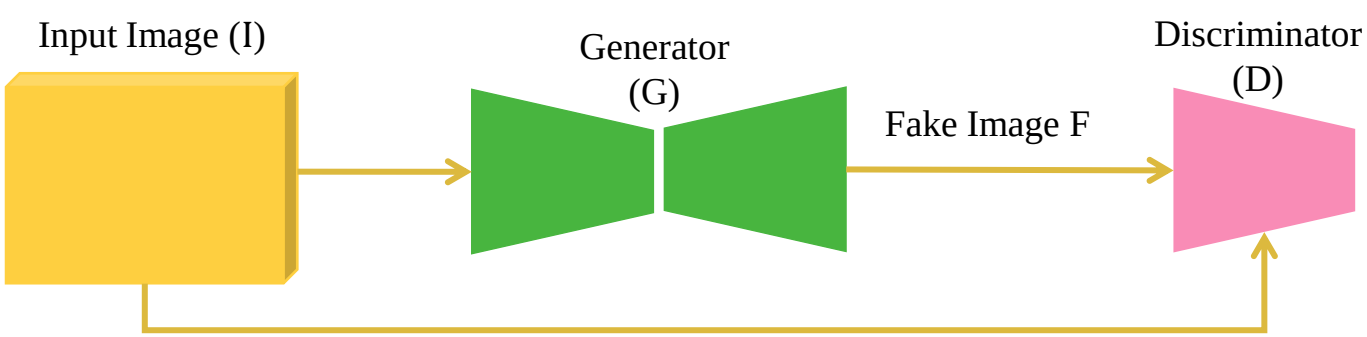


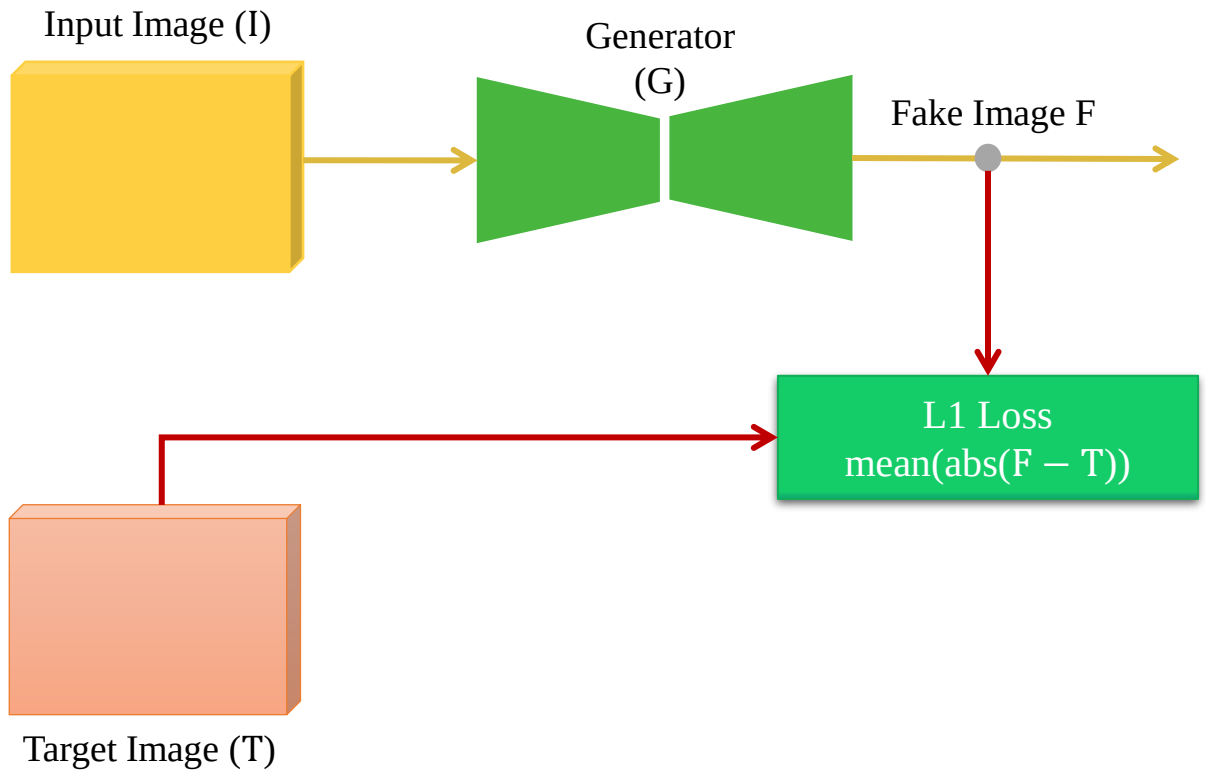
Pix2Pix

Training the Generator



$$L_G = L_{gan} + \lambda L_1$$



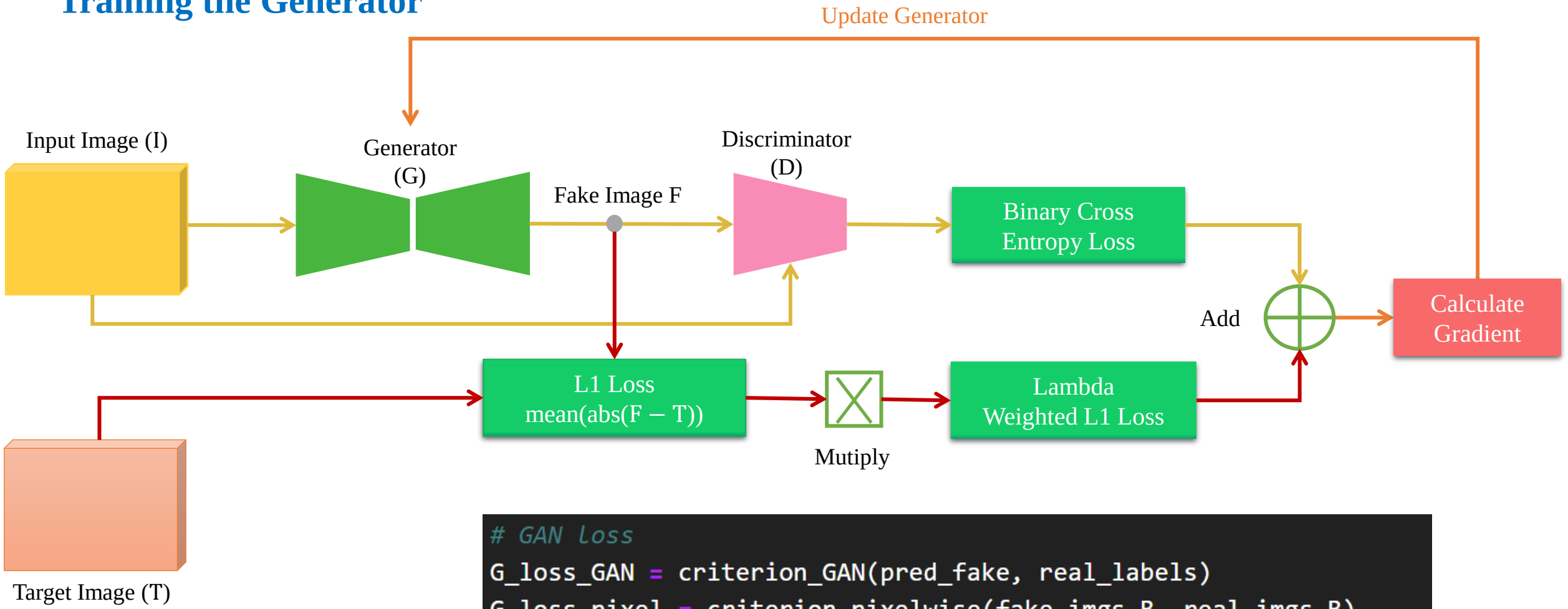



```
# real_imgs_A
# real_imgs_B
optimizer_G = torch.optim.Adam(generator.parameters(),
                               lr=0.0001)
criterion_GAN = torch.nn.MSELoss()
criterion_pixelwise = torch.nn.L1Loss()

real_labels = torch.ones(real_imgs_A.size(0), 16, 16)
fake_labels = torch.zeros(real_imgs_A.size(0), 16, 16)

# --- Train Generator ---
optimizer_G.zero_grad()
fake_imgs_B = generator(real_imgs_A)
pred_fake = discriminator(fake_imgs_B,
                          real_imgs_A)

# GAN loss
G_loss_GAN = criterion_GAN(pred_fake,
                           real_labels)
G_loss_pixel = criterion_pixelwise(fake_imgs_B,
                                   real_imgs_B)
G_loss = G_loss_GAN + lambda_pixel * G_loss_pixel


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

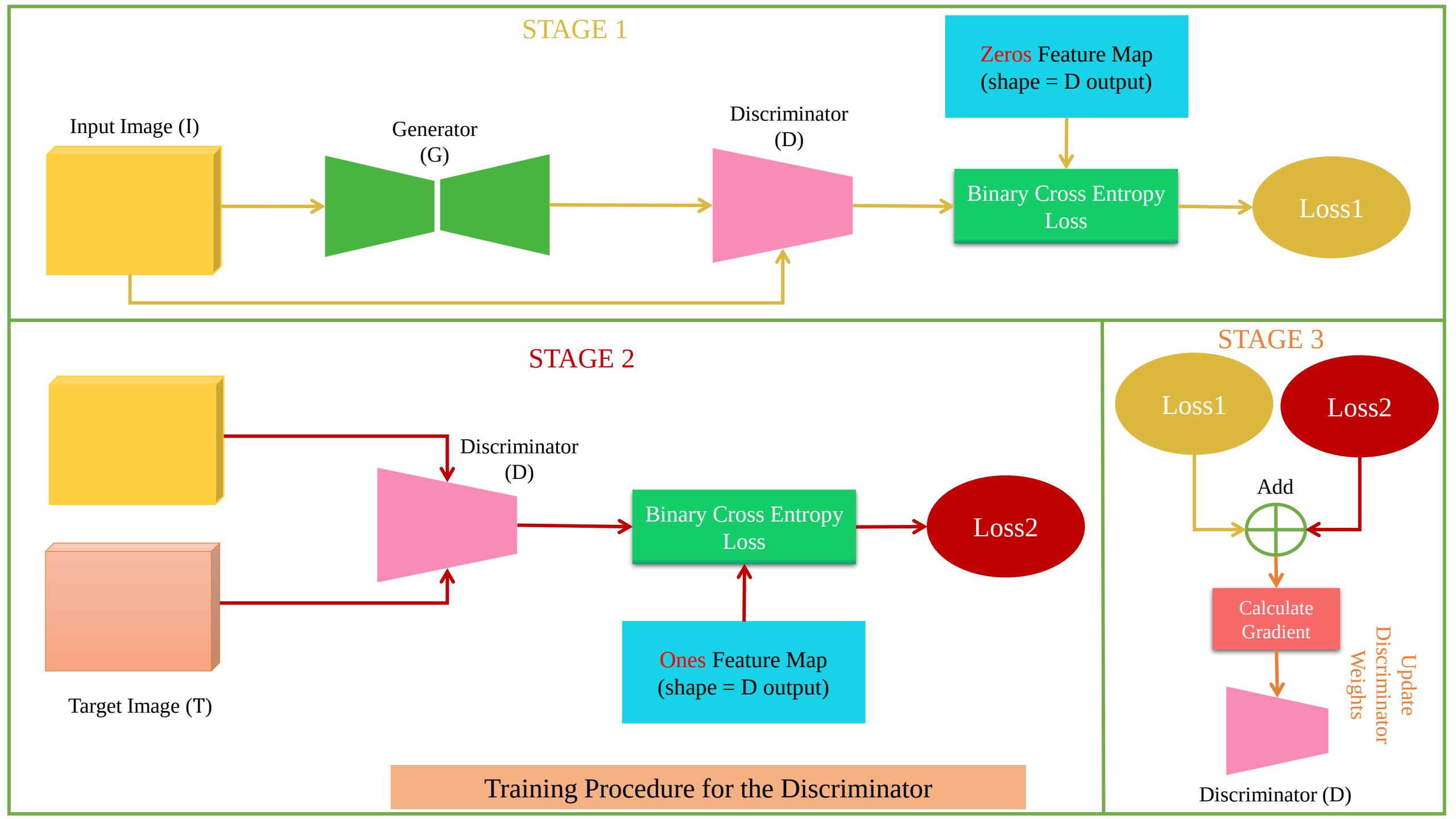
Pix2Pix

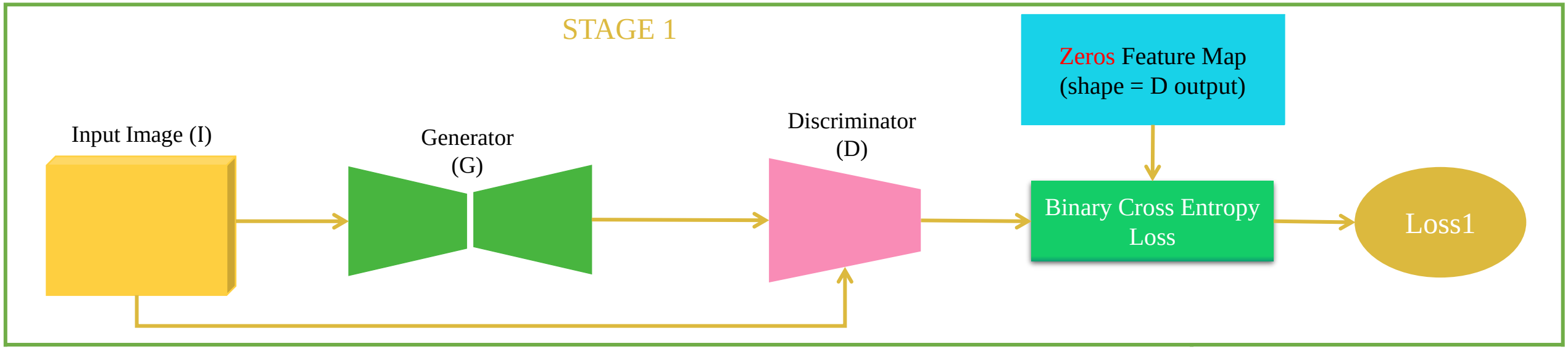
$$L_G = L_{gan} + \lambda L_1$$

Training the Generator




```
# GAN loss
G_loss_GAN = criterion_GAN(pred_fake, real_labels)
G_loss_pixel = criterion_pixelwise(fake_imgs_B, real_imgs_B)
G_loss = G_loss_GAN + lambda_pixel * G_loss_pixel
```


STAGE 1
Input Image (I)
Generator
(G)
Discriminator
(D)
Zeros Feature Map
(shape = D output)
Binary Cross Entropy
Loss
Loss1
STAGE 2
Discriminator
(D)
Binary Cross Entropy
Loss
Loss2
Target Image (T)
Ones Feature Map
(shape = D output)
STAGE 3
Loss1
Loss2
Add
Calculate
Gradient
Update
Discriminator
Weights
Discriminator (D)
Training Procedure for the Discriminator



Pix2Pix

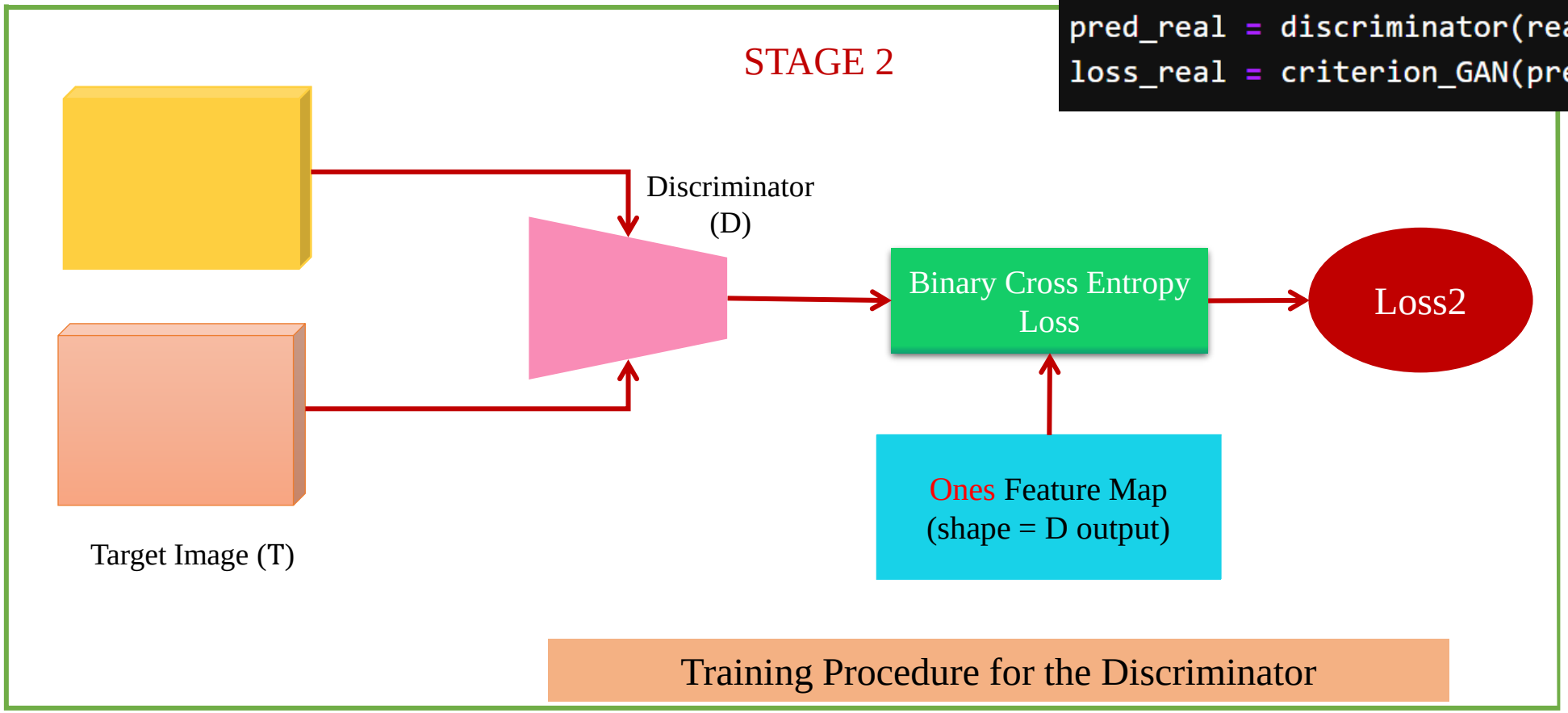
Training the Discriminator

```
# real_imgs_A
# real_imgs_B

criterion = torch.nn.MSELoss()

# prepare labels
fake_labels = torch.zeros(real_imgs_A.size(0), *patch)

# Train
pred_fake = discriminator(fake_imgs_B.detach(), real_imgs_A)
loss_fake = criterion(pred_fake, fake_labels)
```

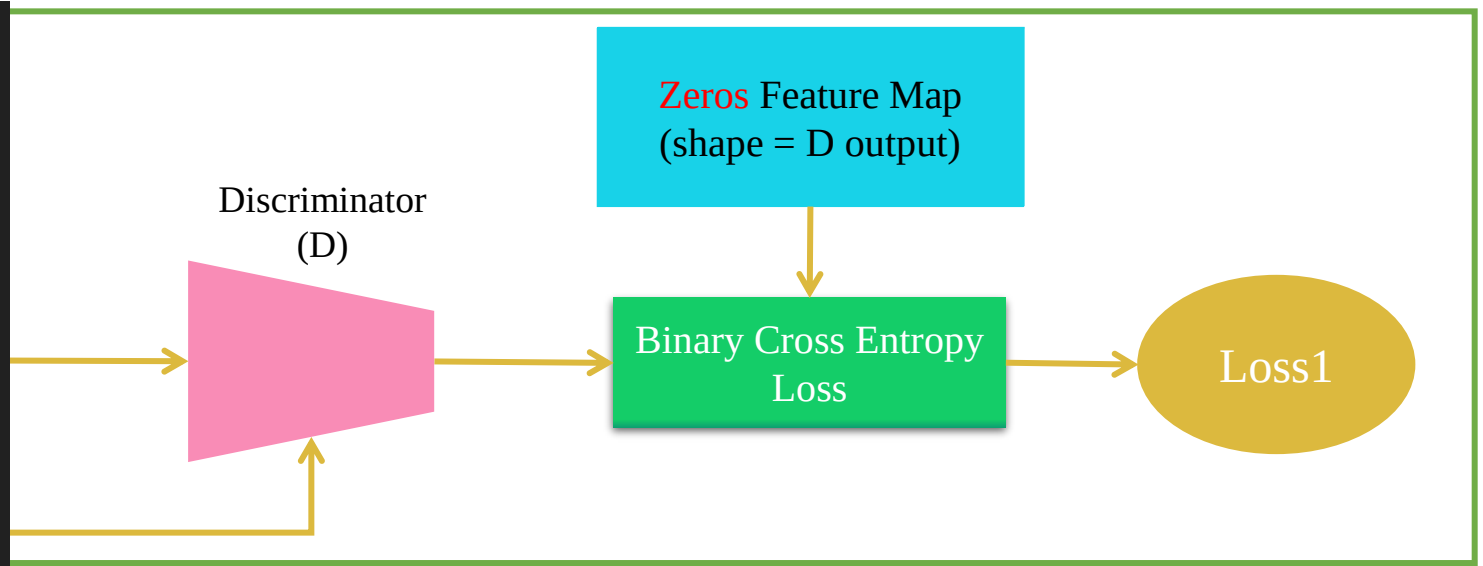
Pix2Pix

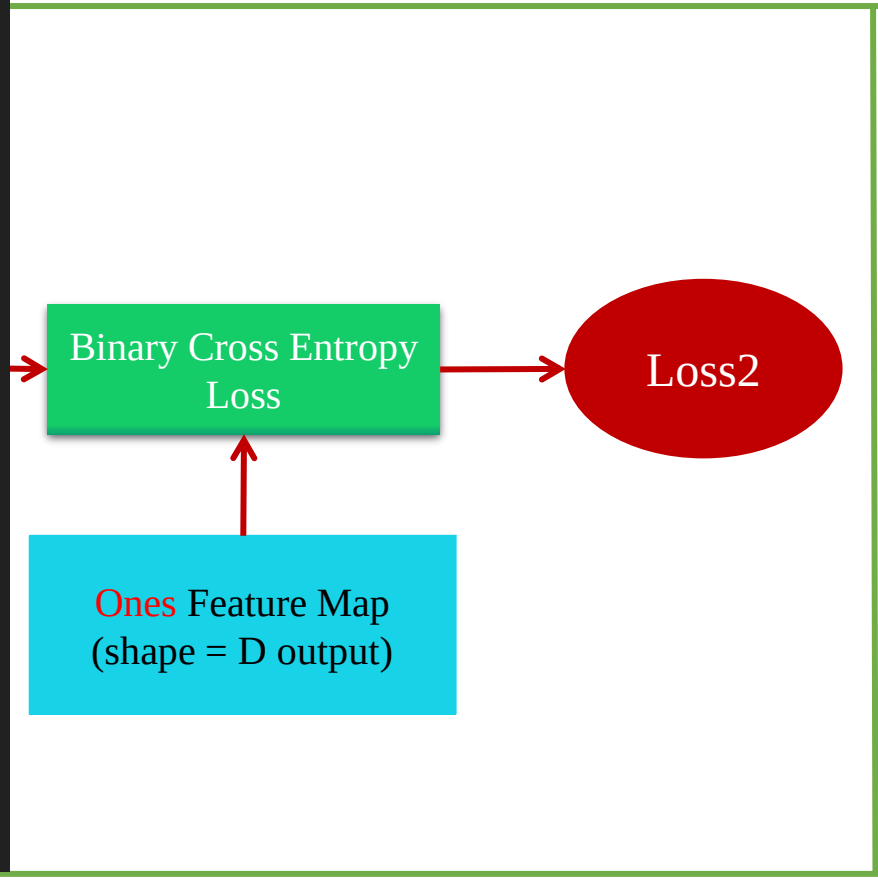
Training the Discriminator

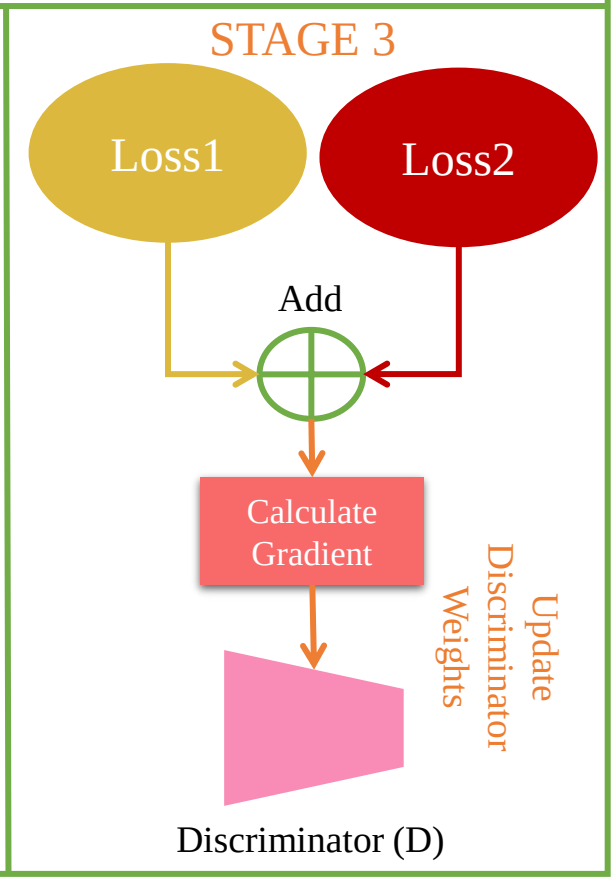
```
# real_imgs_A
# real_imgs_B

criterion = torch.nn.MSELoss()

# prepare labels
real_labels = torch.ones(real_imgs_A.size(0), *patch)

# Train
pred_real = discriminator(real_imgs_B, real_imgs_A)
loss_real = criterion_GAN(pred_real, real_labels)
```



```python
optimizer_D = Adam(discriminator.parameters(),
                   lr=0.0002)
criterion = torch.nn.MSELoss()

# prepare labels
real_labels = torch.ones(real_imgs_A.size(0),
                         *patch)
fake_labels = torch.zeros(real_imgs_A.size(0),
                          *patch)

# --- Train Discriminator
optimizer_D.zero_grad()
pred_real = discriminator(real_imgs_B,
                          real_imgs_A)
loss_real = criterion_GAN(pred_real,
                          real_labels)

pred_fake = discriminator(fake_imgs_B.detach(),
                          real_imgs_A)
loss_fake = criterion(pred_fake, fake_labels)
D_loss = (loss_real + loss_fake) / 2

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

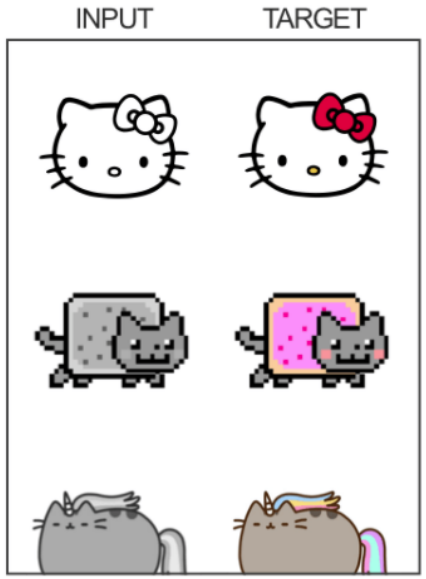


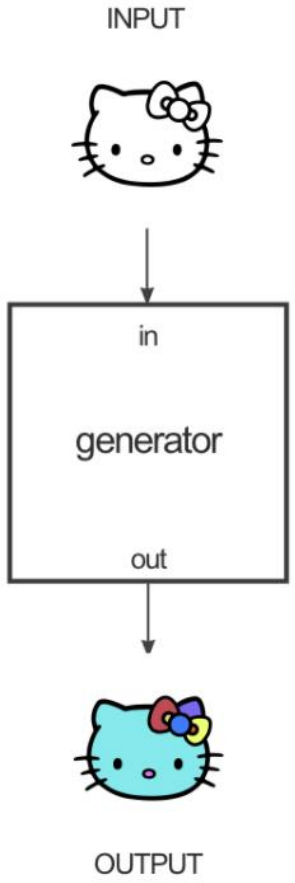


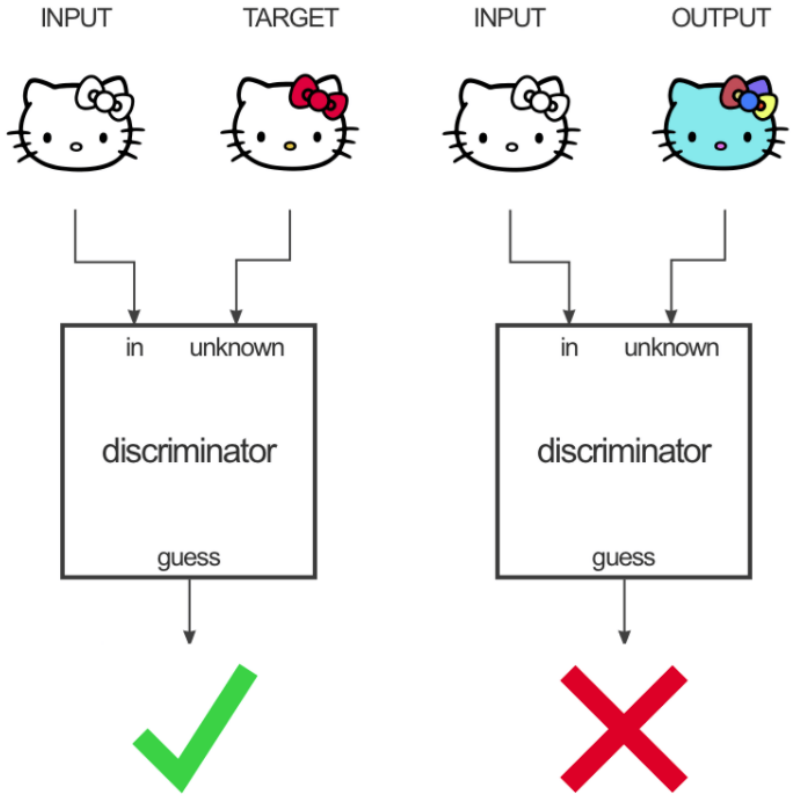


Pix2Pix

❖ Train the model





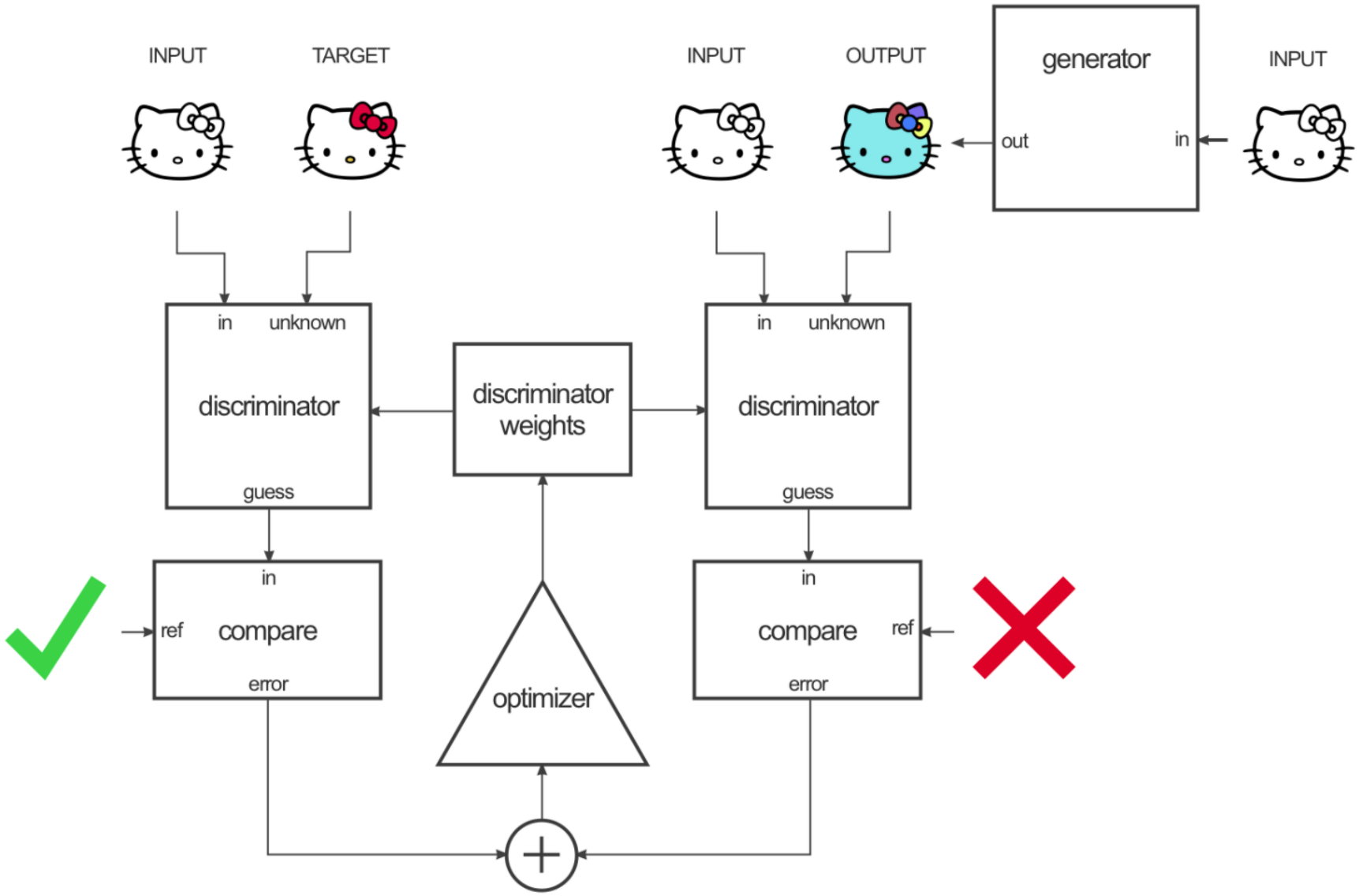


https://affinelayer.com/pix2pix/

Pix2Pix

❖ Train the model

Train the discriminator



Pix2Pix

❖ Applications

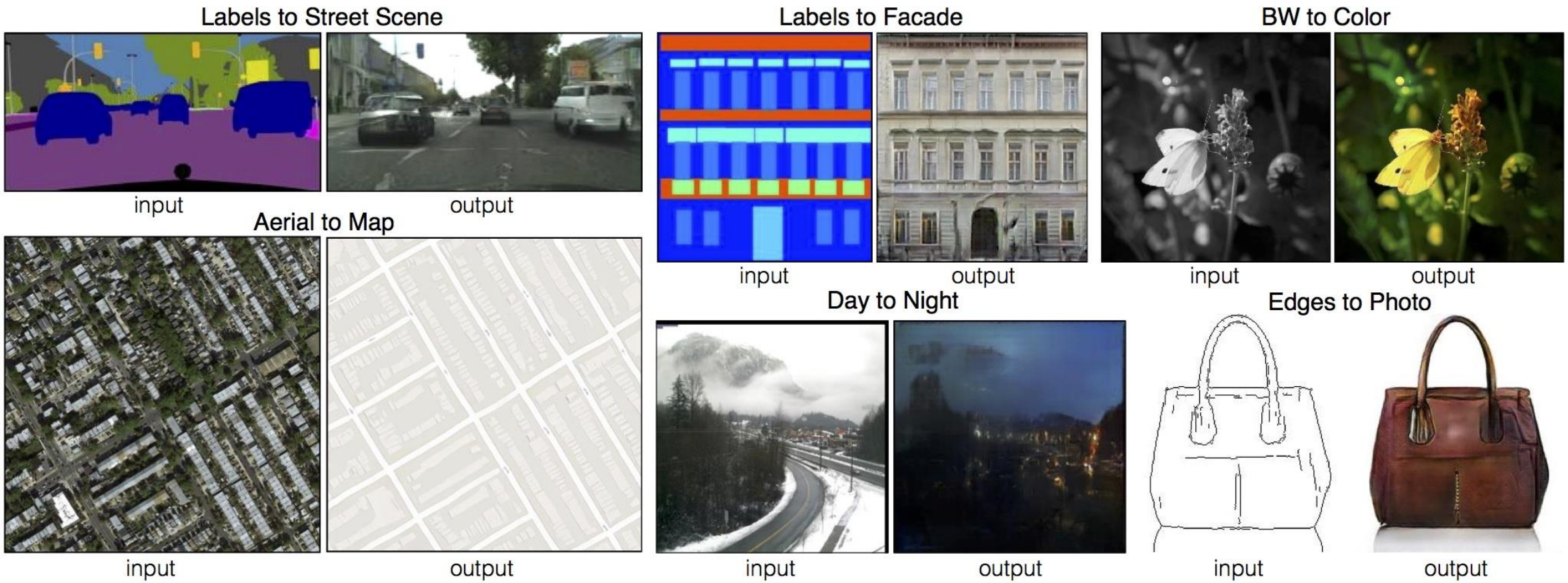
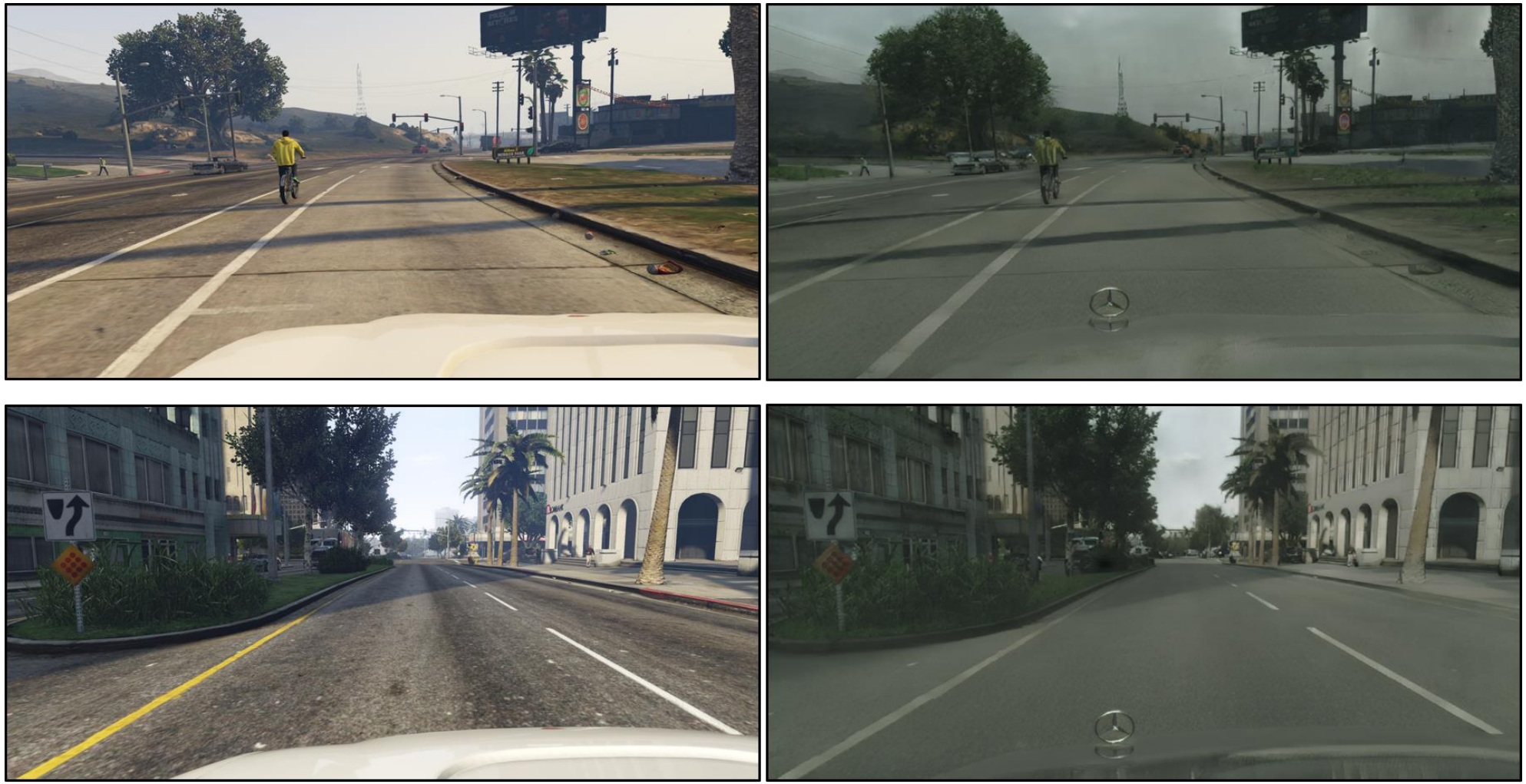


Image-to-Image Translation with Conditional Adversarial Networks, https://arxiv.org/pdf/1611.07004.pdf

Pix2Pix

❖ Applications



GTA → Cityscapes

Pix2Pix

❖ Applications

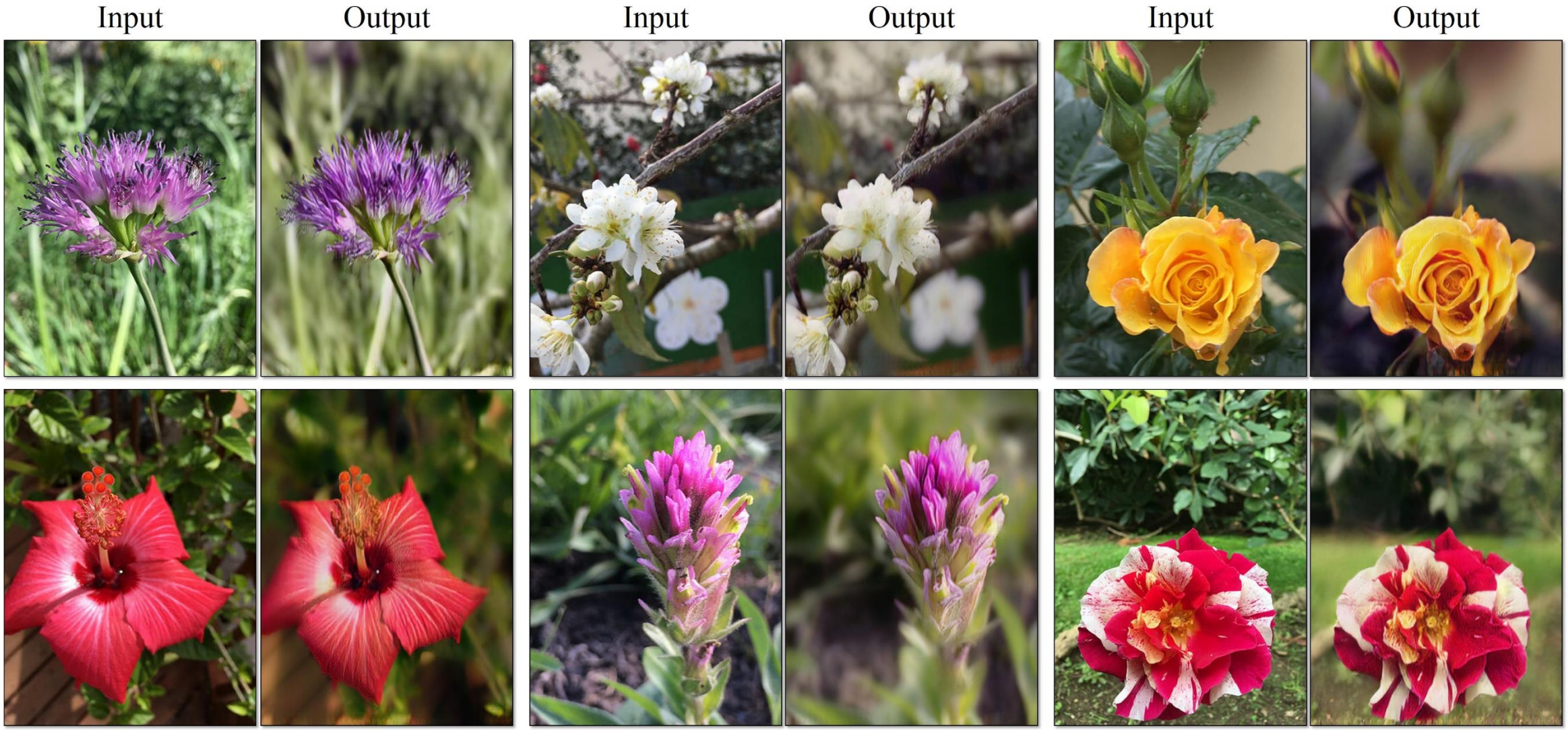
winter Yosemite → summer Yosemite

summer Yosemite → winter Yosemite

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, https://arxiv.org/pdf/1703.10593.pdf

Pix2Pix

❖ Applications

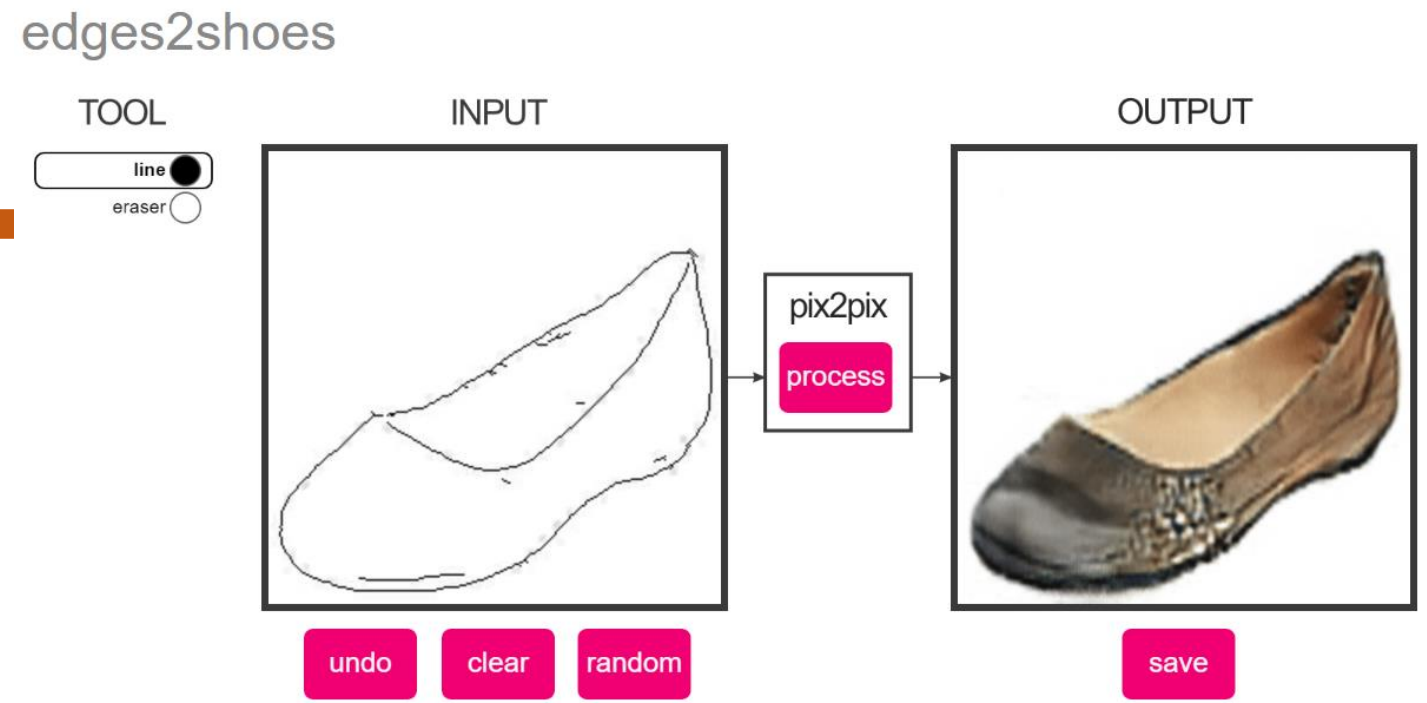


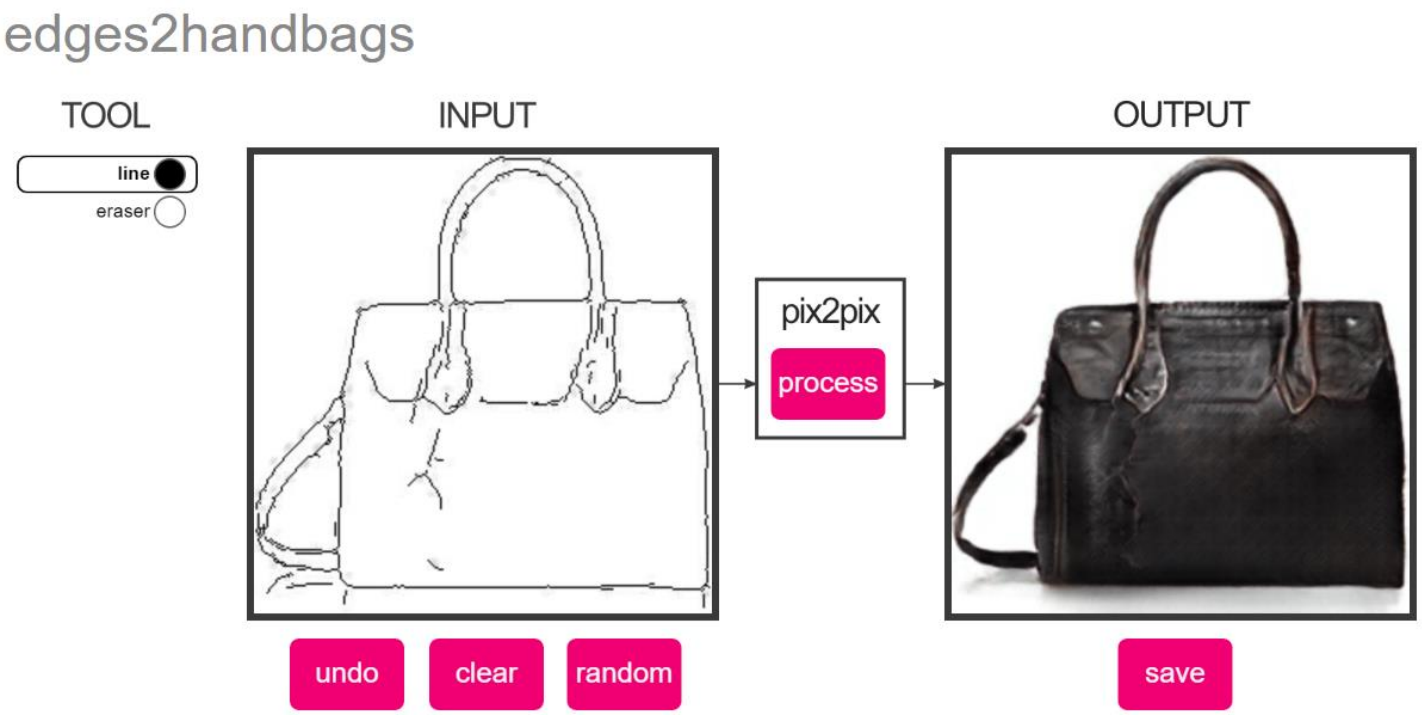
Pix2Pix

❖ Applications

Background masking

https://affinelayer.com/pixsrv/





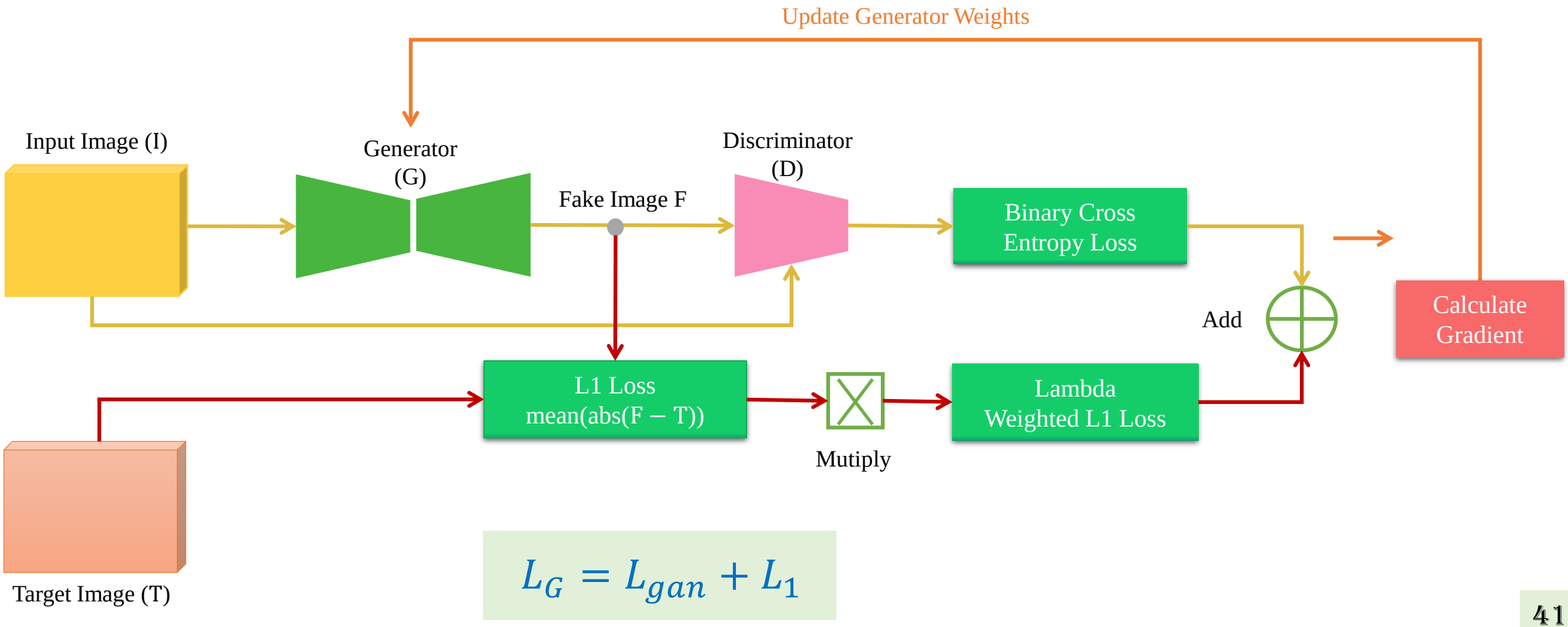

QUIZ TIME

Outline

- Quick Review on GAN/DCGAN
- Conditional GAN
- PatchGAN
- Pix2Pix
- CycleGAN

CycleGAN

❖ Pix2Pix - Problem?



$$L_G = L_{gan} + L_1$$

CycleGAN

❖ Motivation

Horses2Zebra Dataset






CycleGAN

❖ Motivation

Apple2Orange Dataset



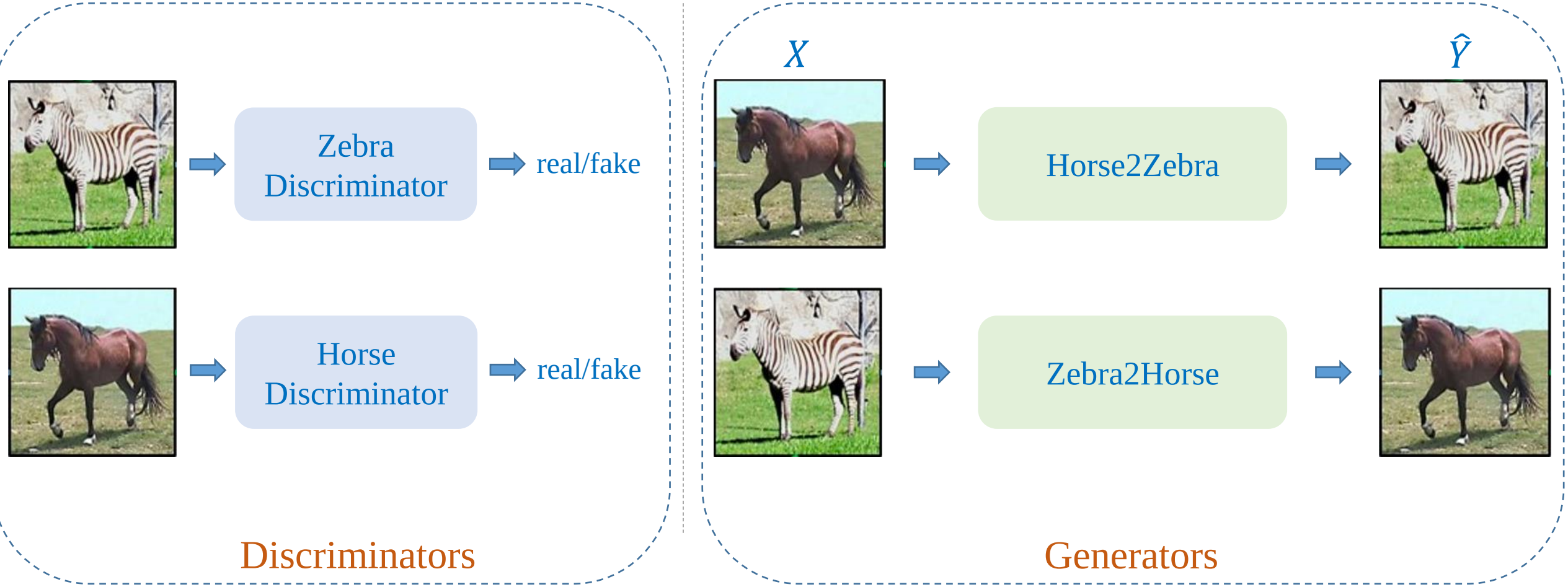



CycleGAN

❖ Idea

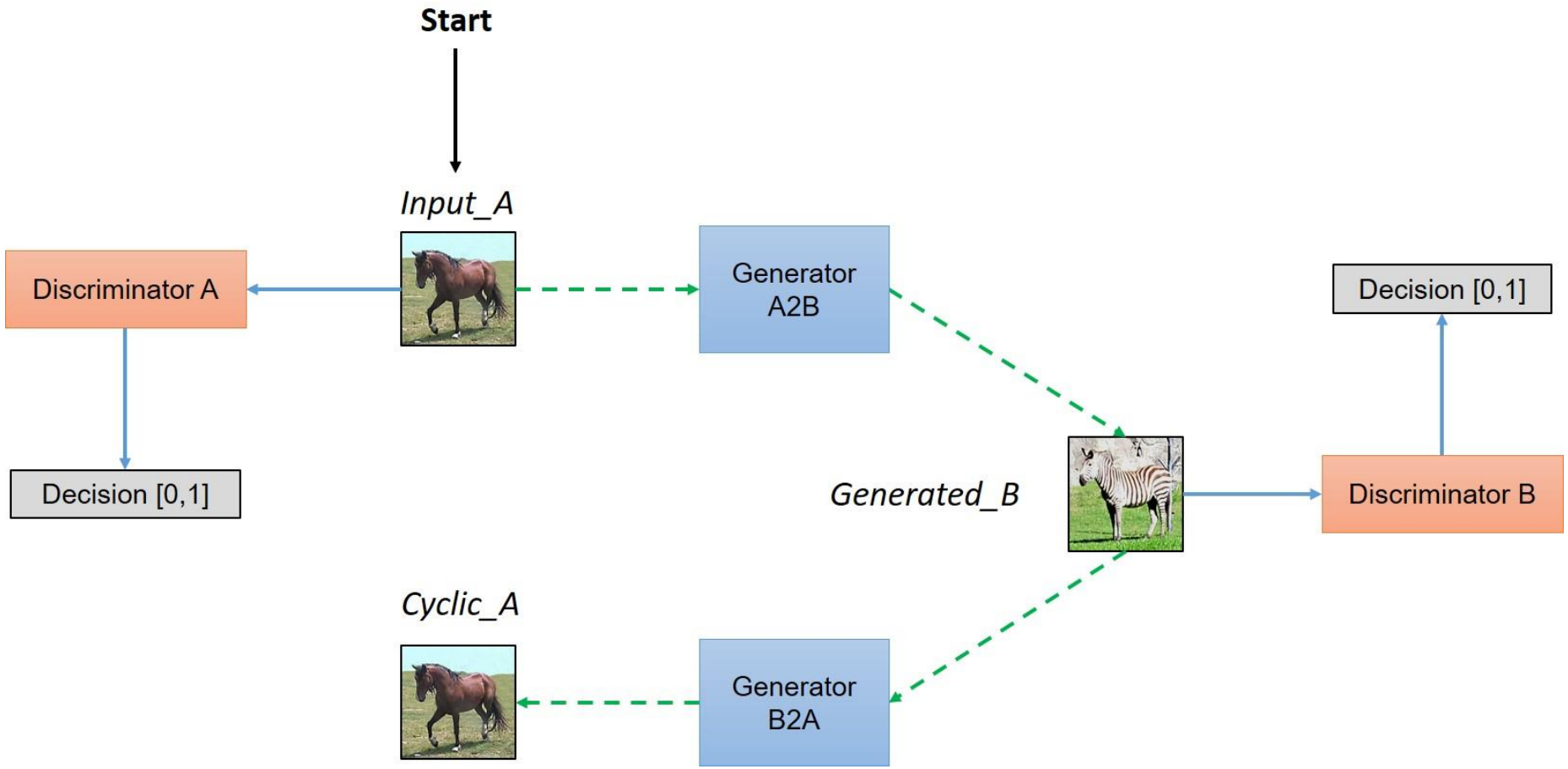
Using L1 loss?

Using GAN loss?



CycleGAN

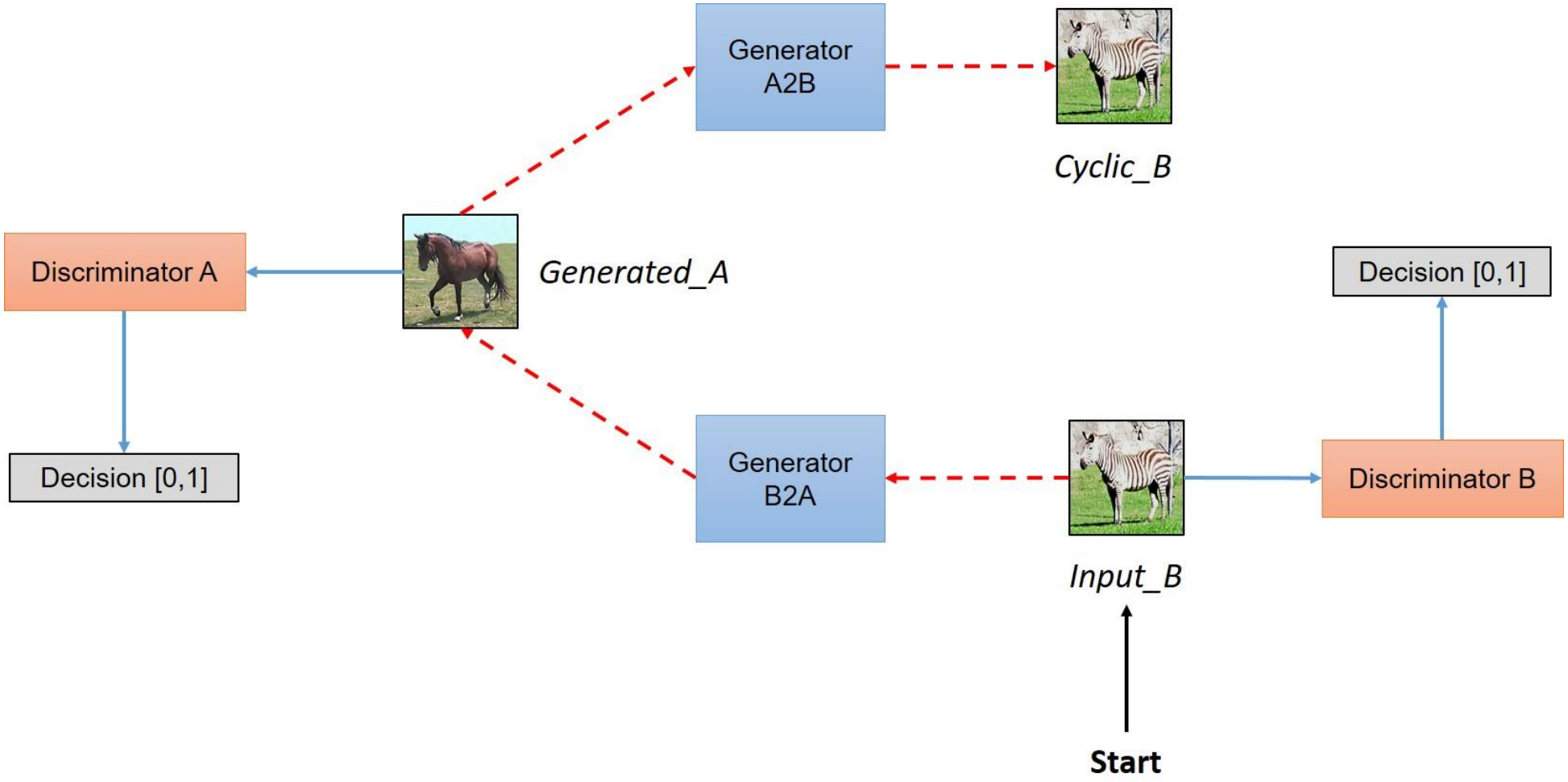
❖ Model



Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, https://arxiv.org/pdf/1703.10593.pdf

CycleGAN

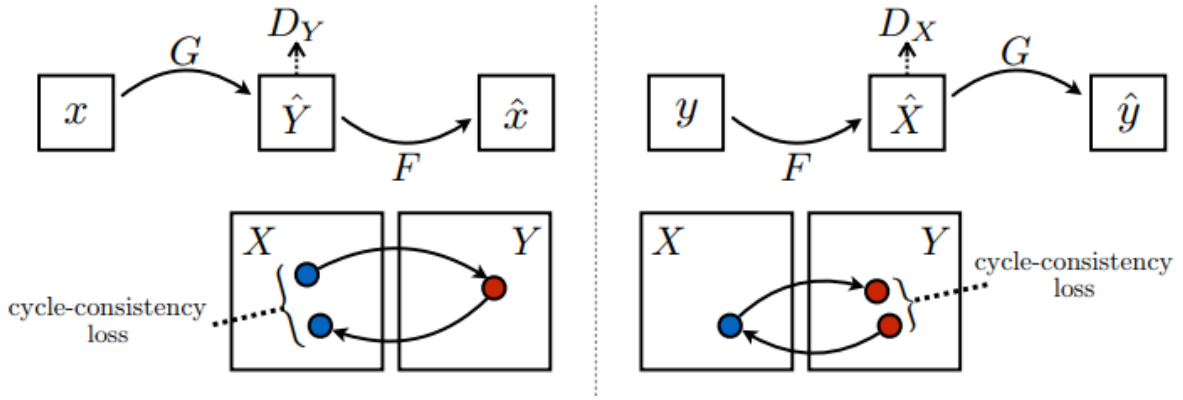
❖ Model

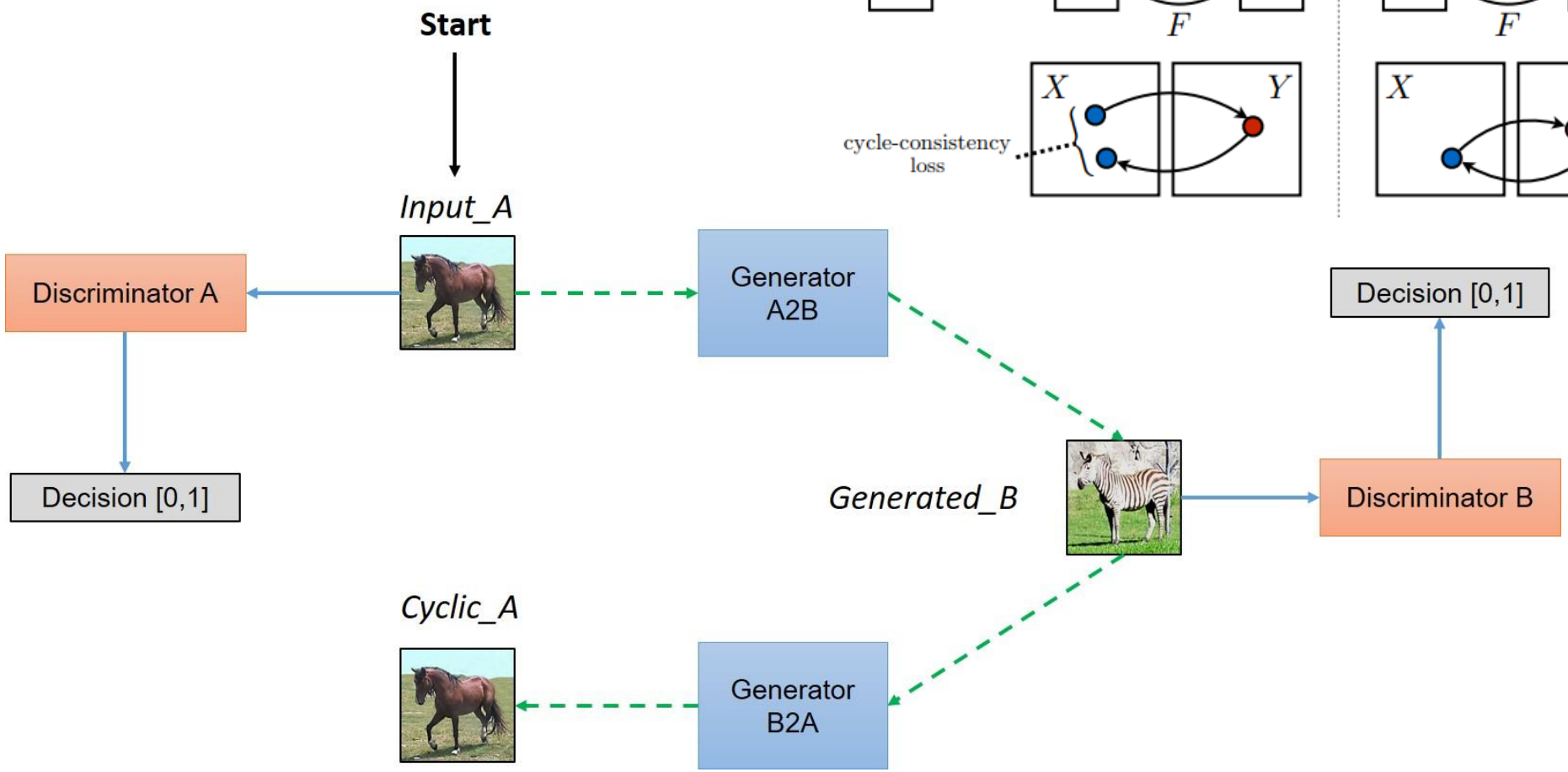


Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, https://arxiv.org/pdf/1703.10593.pdf

CycleGAN

❖ Loss

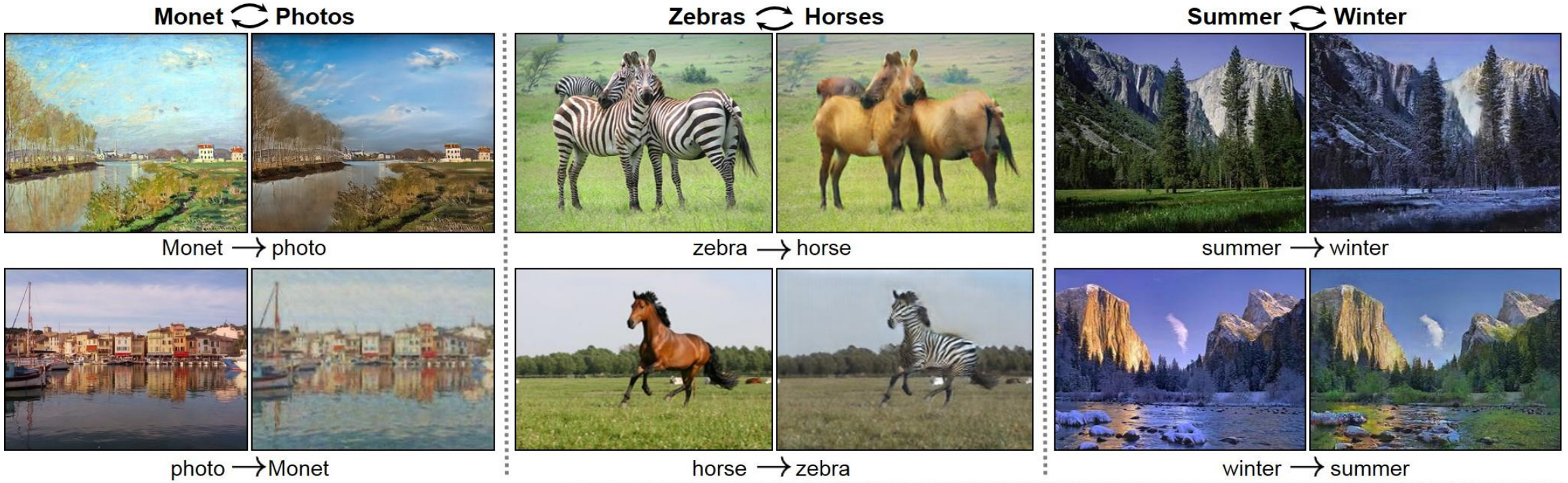




Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, https://arxiv.org/pdf/1703.10593.pdf

CycleGAN

❖ Applications



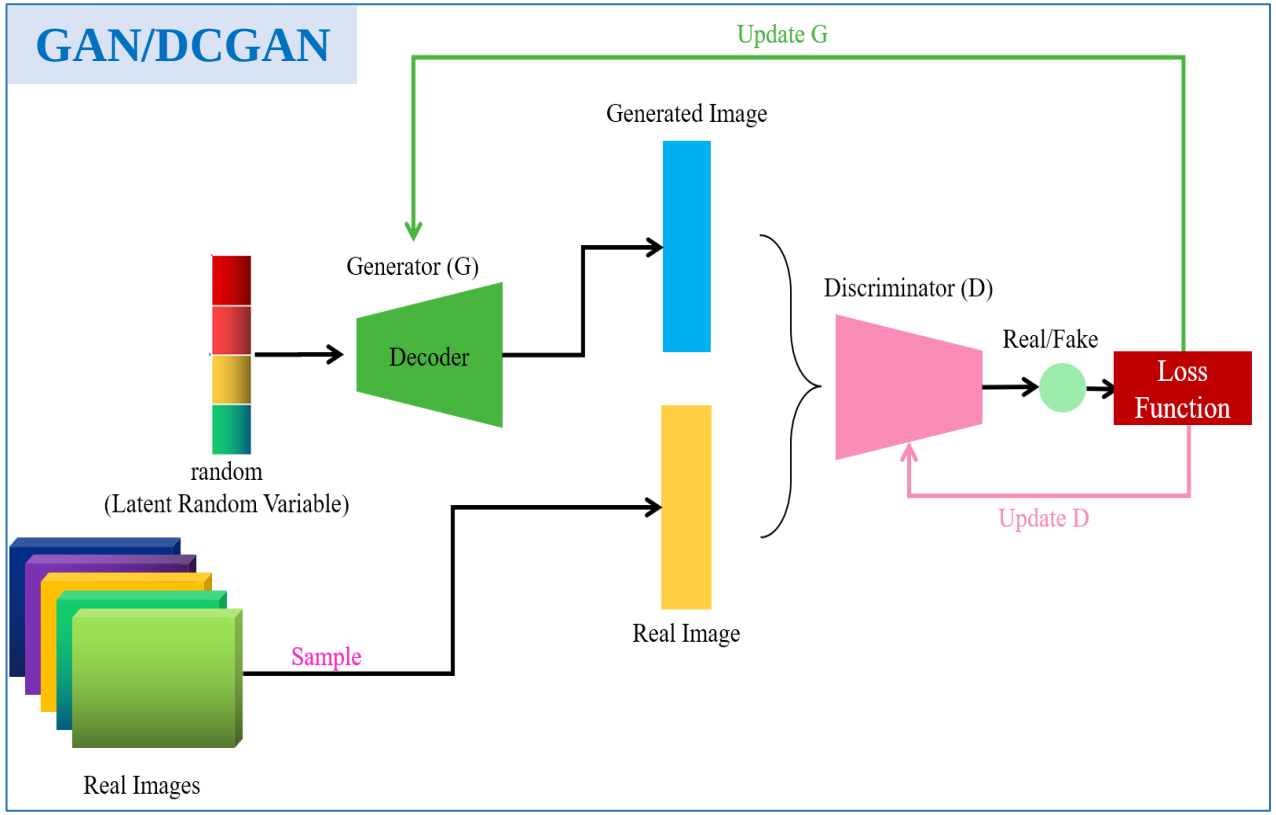
Summary

- Quick Review on GAN/DCGAN
- Conditional GAN
- PatchGAN
- Pix2Pix
- CycleGAN

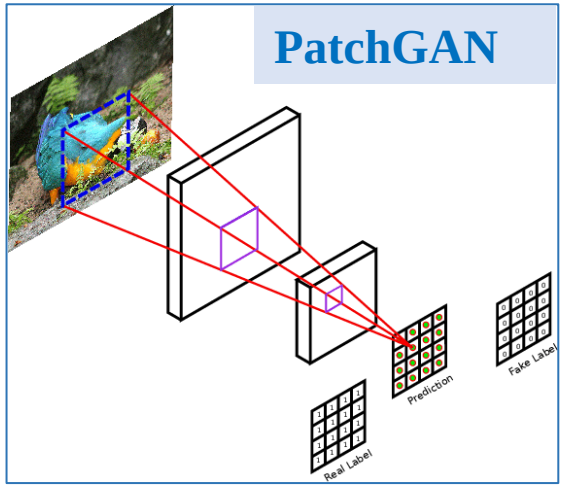
Objectives

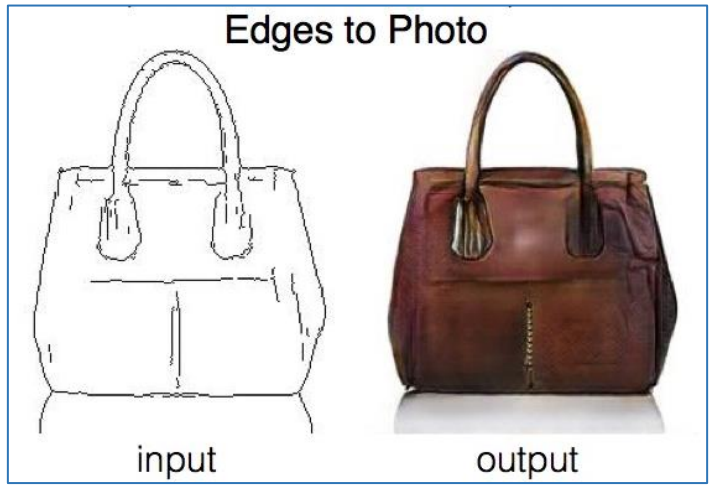
- ✓ GAN/DCGAN
- ✓ Studied conditional GAN
- ✓ Studied Pix2pix
- ✓ Studied PatchGAN
- ✓ Studied CycleGAN

GAN/DCGAN

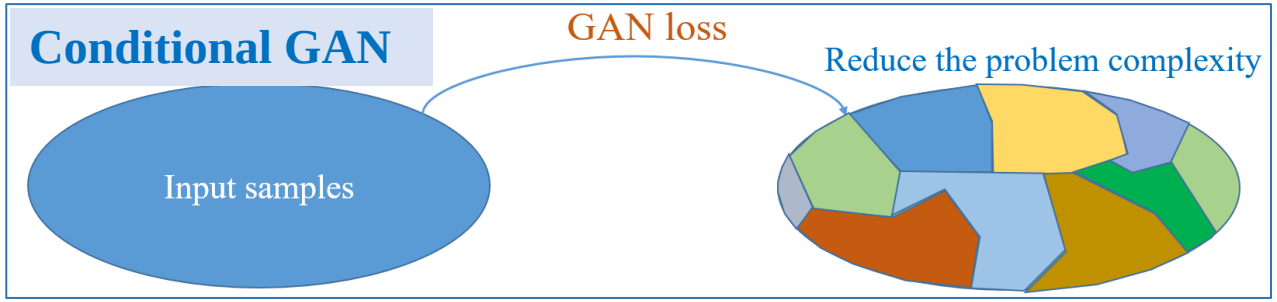


PatchGAN





Conditional GAN



On Sunday

❖ Text to Image

Generative Adversarial Text to Image Synthesis

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran
Bernt Schiele, Honglak Lee

REEDSCOT[1], AKATA[2], XCYAN[1], LLAJAN[1]
SCHIELE[2],HONGLAK[1]

[1] University of Michigan, Ann Arbor, MI, USA (UMICH.EDU)
[2] Max Planck Institute for Informatics, Saarbrücken, Germany (MPI-INF.MPG.DE)

Generative Adversarial Text to Image Synthesis

this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.

the flower has petals that are bright pinkish purple with white stigma

this white and yellow flower have thin white petals and a round yellow stamen