



# LECTURE 14 IMAGE GENERATION MODELS

Punnarai Siricharoen, Ph.D.

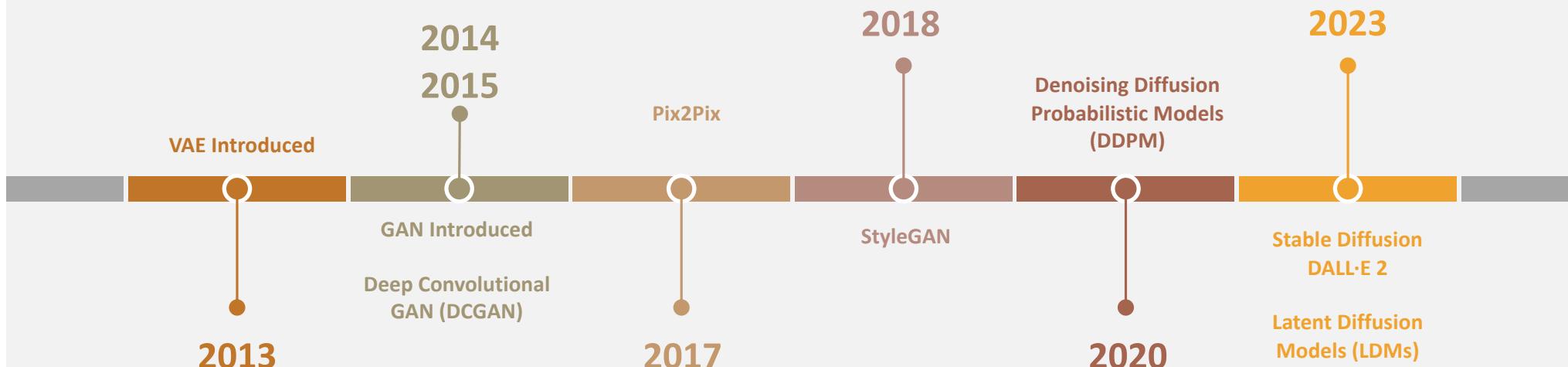
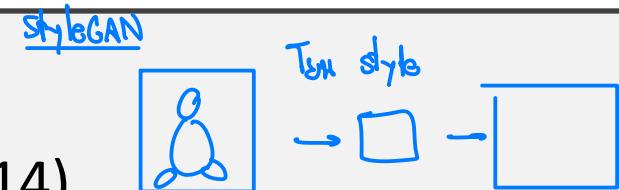
## TODAY'S CONTENT

- Image Generative model
- GAN models for various applications
- DCGAN model
- Variants of GANs
- Diffusion models

$GAN \in \underline{\text{Generation model}}$

# IMAGE GENERATION MODELS

- Variational Autoencoders (2013)
- Generative adversarial networks (GANs) (2014)
- Diffusion Models (2020)



evaluation → PSNR, SSIM ໄມໄດ້ຕາມຫຼັງ

## VARIATIONAL AUTOENCODERS (VAE)

- VAEs is based on an encoder-decoder structure
- The encoder maps input data to a latent space, and then the decoder maps these latent variables back to the data space
- VAEs assume that latent variables follow a certain prior distribution
- VAE is able to generate high-quality samples that are similar to the training data, and it is able to generate samples with a certain degree of continuity

ຂະໜາດໄປໄວ່ຈະຕອນໄວ

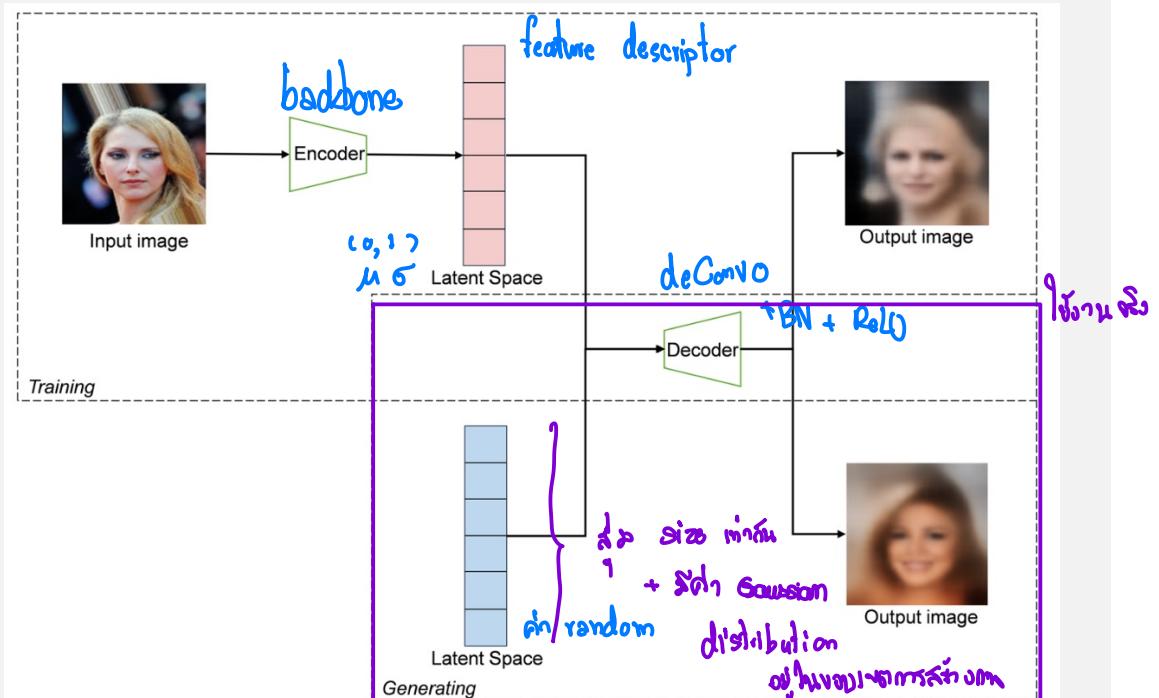


Fig. 2 The model structure of VAE consists of an encoder, which maps the image to the latent space, and a decoder, which recovers the image from the latent space to the pixel space

## VARIATIONAL AUTOENCODERS (VAE)

- Improvements :
  - CVAE - introduces condition information in order to control the generated samples under given conditions *នូវ condition នៃ encoder ដូច ពួកគា / ឈ្មោះ label*
  - alignDraw - first modern text-to-image model, It uses a recurrent variational autoencoder with an attention mechanism, enabling it to use text sequences as input.
  - IntroVAE - allowing it to self-evaluate the quality of generated samples and subsequently self-improve (similar to GAN) *evaluate ការងារអាចវិនិន័យ ?*

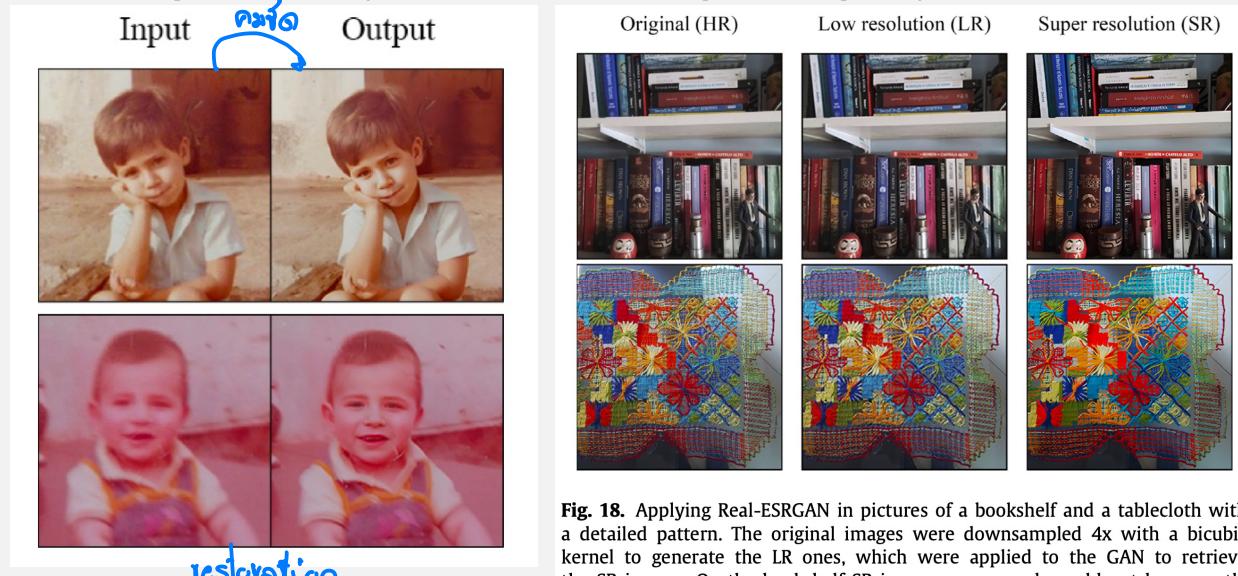
## WHAT IS THE GENERATIVE ADVERSARIAL MODEL?

- Generative Adversarial Networks (GANs) are a type of deep learning architecture to capture **the training data distribution** so we can generate new data from that same distribution. ພາບກັນຈົກທີ່ມີ **training data**
  - invented by Ian Goodfellow in 2014
  - uses two networks namely a **generator** and a **discriminator** that, by competing with each other, pursue to create realistic but unseen samples.
    - a **generator** - to generate synthetic examples based on the training data → **segmentation**
    - a **discriminator** trained to distinguish synthetic instances created by the generator from real data      ບອກໄລ້ລຳການ gen ດີເນີນ fake → **classification**
    - This competition leads the **generator** to produce very realistic samples

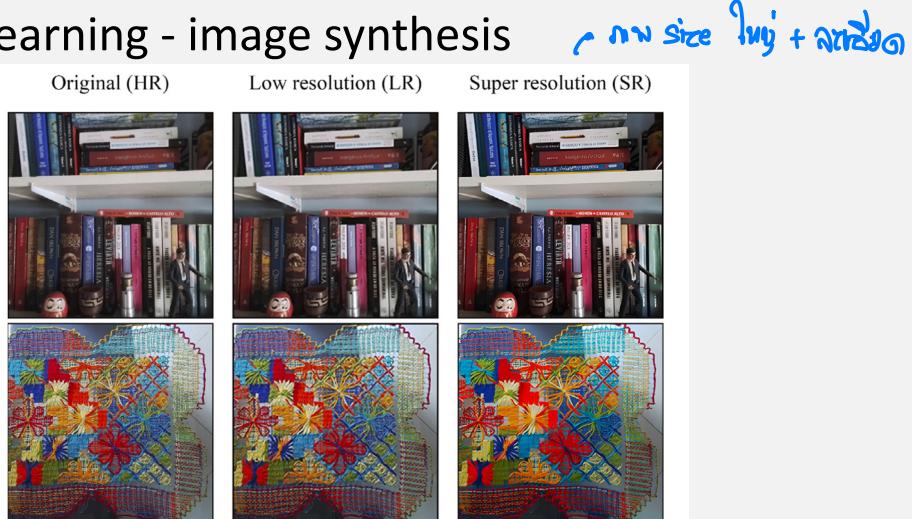
## APPLICATIONS OF GENERATIVE MODELS FOR IMAGES

- To create a person/things that never exist - image manipulation from latent spaces, image-to-image translation, style transfer, and image repair
- To create more images for supervised learning - image synthesis

නිර්මාණ  
නිර්මාණ ත්‍රයෝගීතා  
හැඳුව සංස්කරණ



Examples of using GFP-GAN to restore old facial photographs



**Fig. 18.** Applying Real-ESRGAN in pictures of a bookshelf and a tablecloth with a detailed pattern. The original images were downsampled 4x with a bicubic kernel to generate the LR ones, which were applied to the GAN to retrieve the SR images. On the bookshelf SR image, some words could not be correctly recreated. On the bottom one, the finer details of the threads are mostly not present, and the colors appear to be more vibrant. Zoom in for better view.

## APPLICATIONS OF GENERATIVE MODELS FOR IMAGES

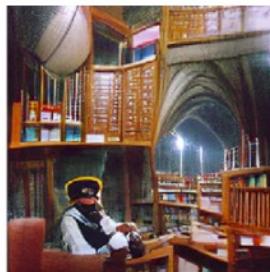
- To create a person/things that never exist - image manipulation from latent spaces, image-to-image translation, style transfer, and image repair
- To create more images for supervised learning - image synthesis



medieval castle on top  
of a mountain



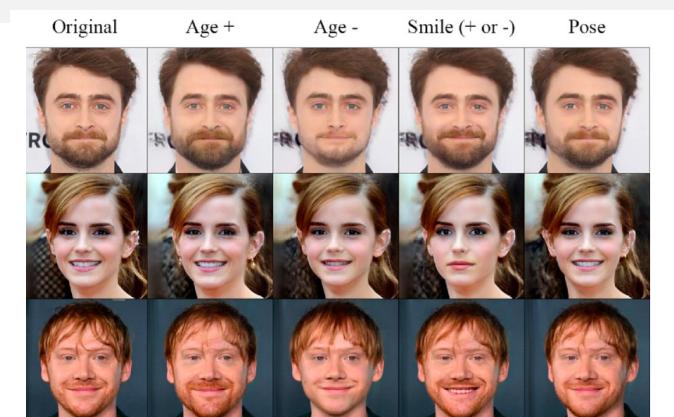
artificial intelligence  
laboratory



a scholar in a library



pirate ship in a storm

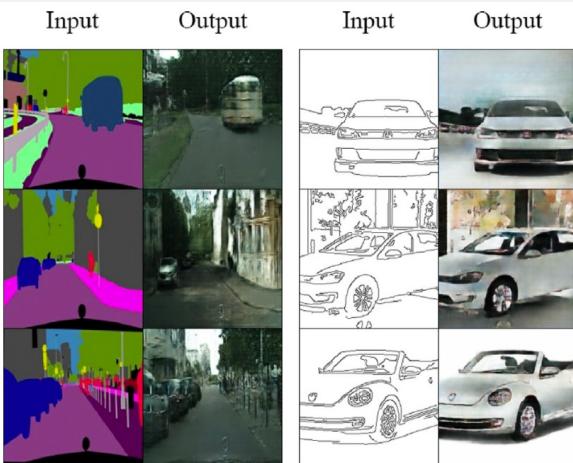


**Fig. 9.** Example of images generated by VQGAN-CLIP after 2000 iterations, with their respective text prompts. Zoom in for better view.

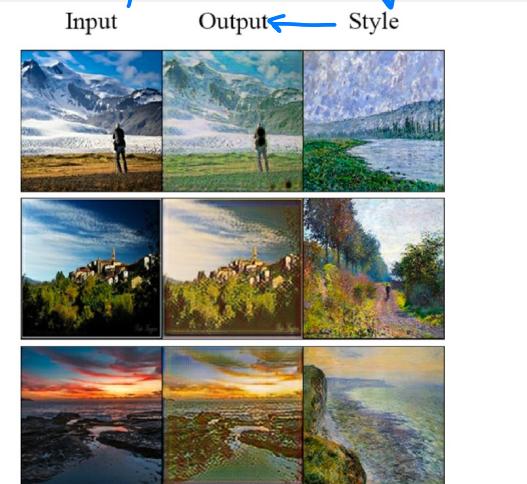
**Fig. 10.** The facial images on the first column (original) were edited by using the PTI method [32]. The following columns show the results of increasing age, decreasing age, increasing or decreasing the smile, and changing the pose on the original image, respectively.

## APPLICATIONS OF GENERATIVE MODELS FOR IMAGES

- To create a person/things that never exist - image manipulation from latent spaces, image-to-image translation, style transfer, and image repair
- To create more images for supervised learning - image synthesis



**Fig. 12.** Examples of images generated by applying pix2pix to synthesize street scenes from semantic maps (left) in the cityscapes dataset [84], and to create car images from edges (right). The input column presents the images  $X_A$  inserted into the generator, while the corresponding generated images  $G(X_A)$  are presented on the output column.

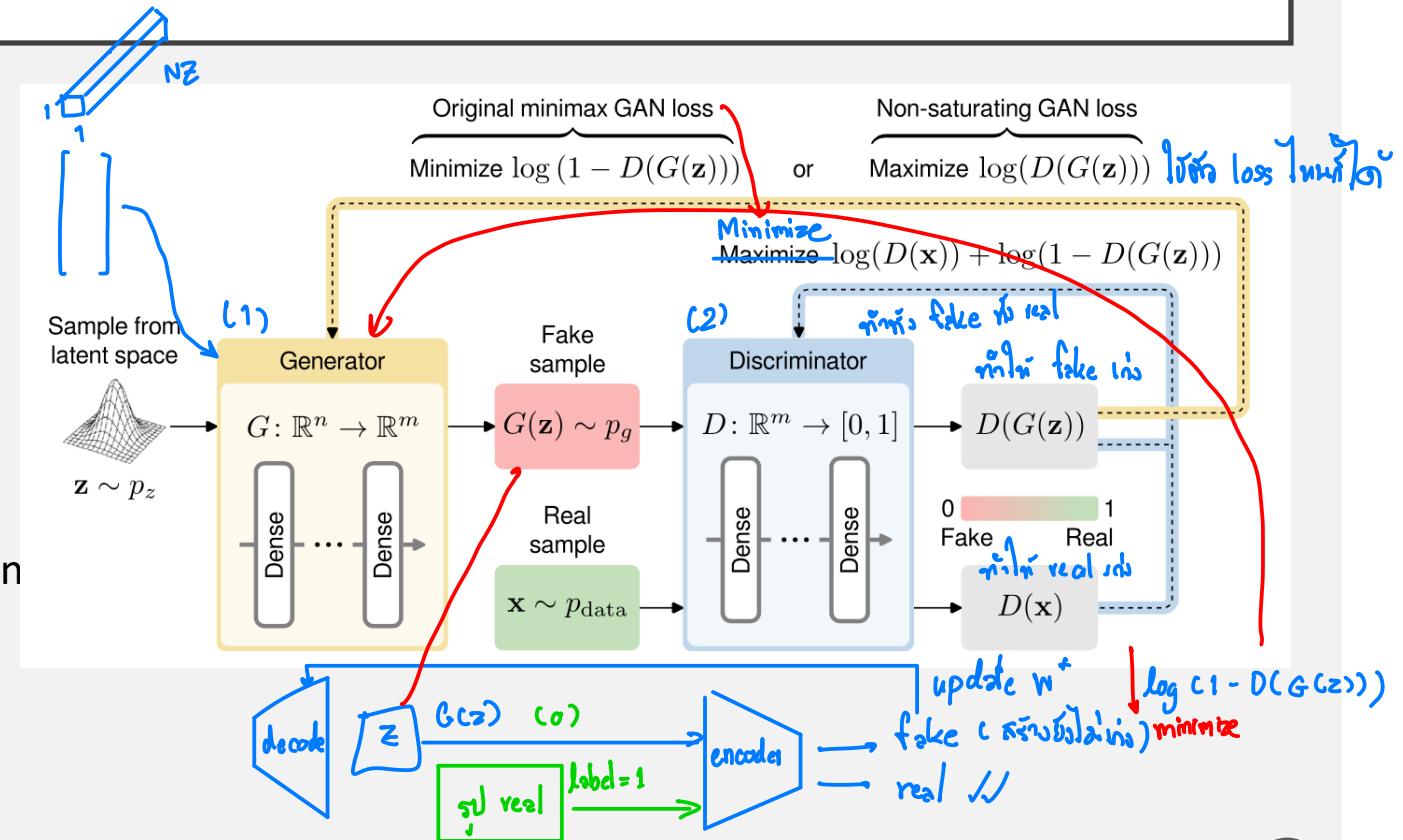


**Fig. 11.** Example of a style-transfer application with a CycleGAN trained in the monet2photo dataset [37]. The pictures (left) are transformed into paintings (center) with colors and textures similar to other samples in the painting domain (right). Zoom in for better view.

## GENERATIVE MODELS - ARCHITECTURE

generator កំសង្គមនា U-NET  
discritne អារីម U-NET

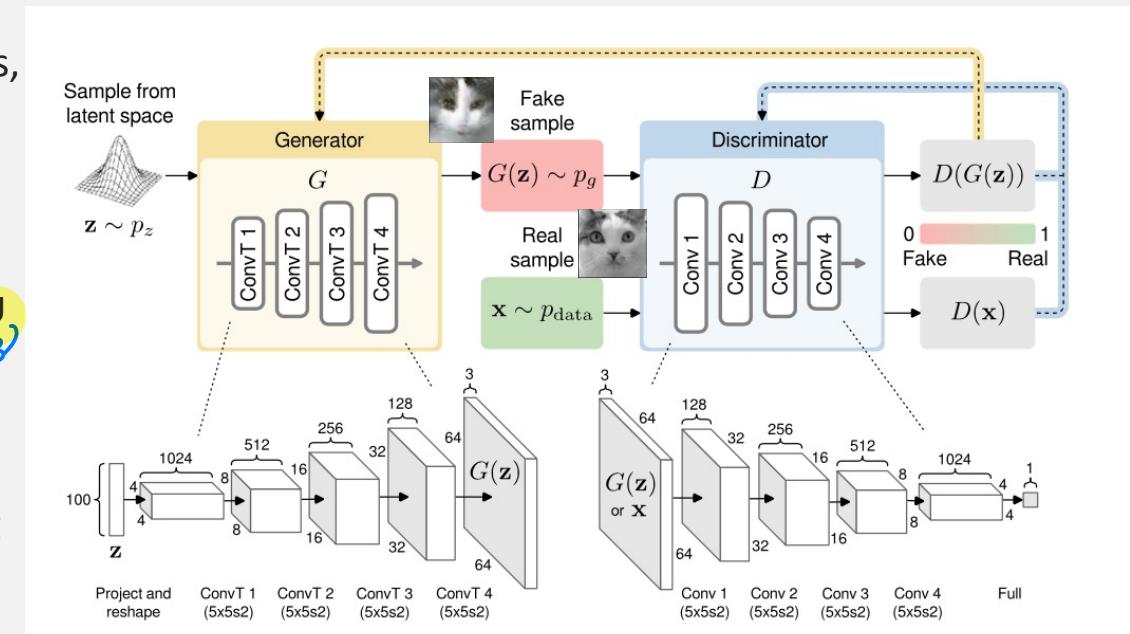
- Training steps of a GAN
  - The **generator** produces a synthetic (fake) image both real and fake are presented at the **discriminator**, that classify them as real (1) or fake (0).
  - The parameters of the **discriminator** are updated on every step (blue line)
  - while the parameters of the **generator** are updated only when a fake image is discriminated (yellow line).



# EVOLUTION OF A GENERATIVE MODELS FOR IMAGES : DCGAN

- **Deep convolutional GAN (DCGAN)** - a specialized GAN for better image synthesis, become foundation for most image-based GAN
- **DCGAN architecture**
  - The generator is composed of transposed convolutions with BatchNorm layers. ReLU activation is used on every layer of the generator except the last.
  - The discriminator is a sequence of convolution layers with leaky ReLU activations and BatchNorm layers. Pooling is not used in both networks.
  - The training process is similar to the original GAN.

*Stride = 2  
means Max Pooling*



COLAB: <https://colab.research.google.com/drive/1S4NbMV0g4nBDbr8e-ShTFjeYyTtAKMYW?usp=sharing>

## ARCHITECTURE OF A GENERATIVE MODEL : DCGAN

### • Generator

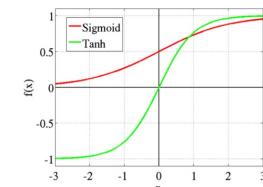
- Input : a noise vector input  $z$  is randomly sampled from a distribution  $p_z$ , referred to as the latent space  $Z$ , and typically a Gaussian distribution.
- The generator  $G$  then creates a synthetic output  $G(z)$  – output image
- The output image will be presented to the discriminator which classifies as real (1) or fake (0)

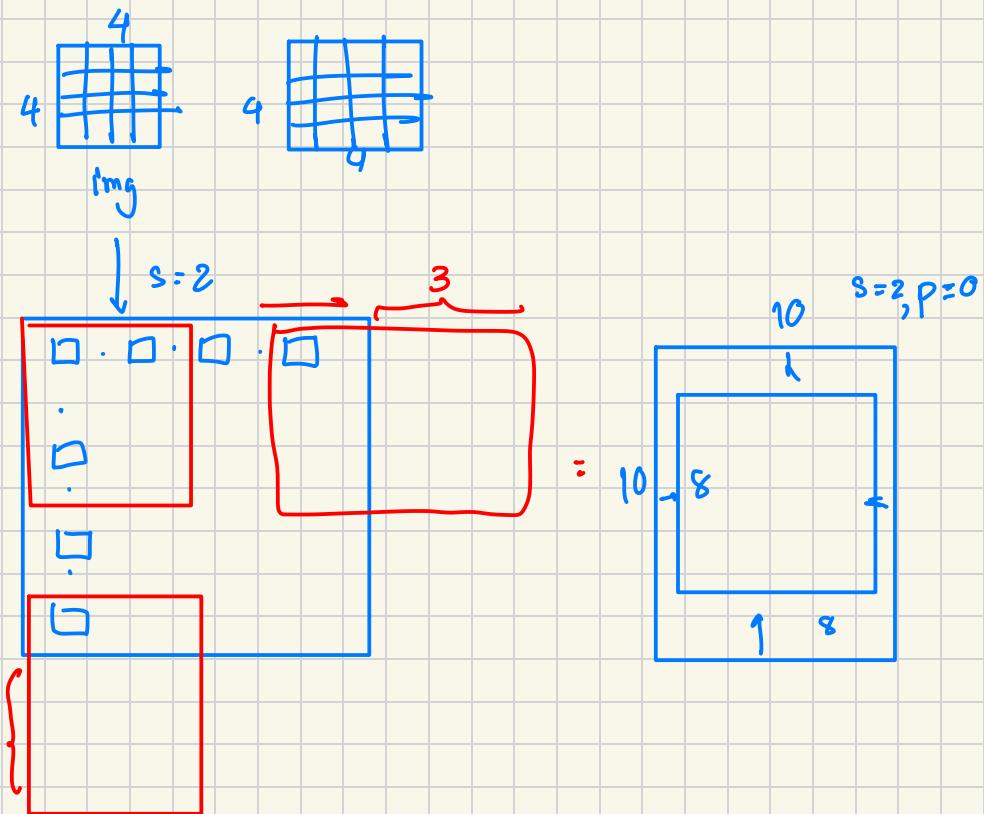
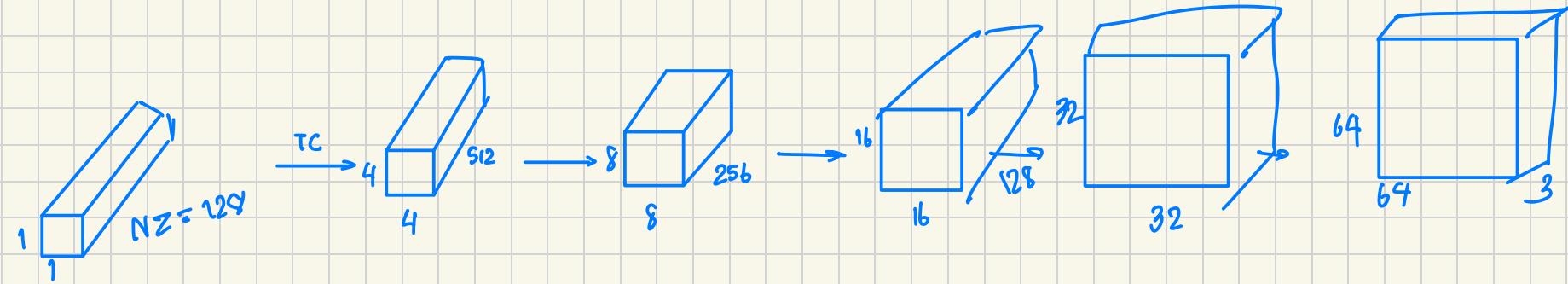
[https://pytorch.org/tutorials/beginner/dcgan\\_faces\\_tutorial.html](https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html)

```
class Generator(nn.Module):  
    def __init__(self, ngpu):  
        super(Generator, self).__init__()  
        self.ngpu = ngpu  
        self.main = nn.Sequential(  
            # input is Z, going into a convolution  
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),  
            nn.BatchNorm2d(ngf * 8), s 512 p  
            nn.ReLU(True),  
            # state size. ``(ngf*8) x 4 x 4``  
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),  
            nn.BatchNorm2d(ngf * 4),  
            nn.ReLU(True),  
            # state size. ``(ngf*4) x 8 x 8``  
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),  
            nn.BatchNorm2d(ngf * 2),  
            nn.ReLU(True),  
            # state size. ``(ngf*2) x 16 x 16``  
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),  
            nn.BatchNorm2d(ngf),  
            nn.ReLU(True),  
            # state size. ``(ngf) x 32 x 32``  
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),  
            nn.Tanh()  
            # state size. ``(nc) x 64 x 64``  
        )  
  
    def forward(self, input):  
        return self.main(input)
```

# Size of z latent vector (i.e. size of generator input)  
nz = 128  
  
# Size of feature maps in generator  
ngf = 64  
  
# Size of feature maps in discriminator  
ndf = 64

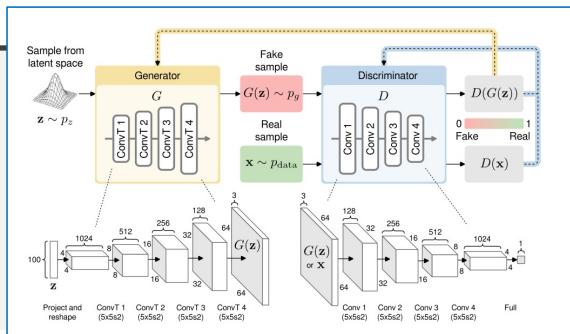
*s* *p* *kernel\_size 4x4*  
*Q* *3 channel*





```
CLASS torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True, device=None, dtype=None) [SOURCE]
```

# ARCHITECTURE OF GENERATIVE MODELS



modify the input directly, without allocating any additional output.

## DCGAN

```
# Size of z latent vector (i.e. size of generator input)
nz = 128

# Size of feature maps in generator
ngf = 64

# Size of feature maps in discriminator
ndf = 64
```

num\_features

```
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. ``(ngf*8) x 4 x 4``
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. ``(ngf*4) x 8 x 8``
```

```
nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
nn.BatchNorm2d(ngf * 2),
nn.ReLU(True),
# state size. ``(ngf*2) x 16 x 16``
nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
nn.BatchNorm2d(ngf),
nn.ReLU(True),
# state size. ``(ngf) x 32 x 32``
nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
nn.Tanh()
# state size. ``(nc) x 64 x 64``
```

```
)
```

```
def forward(self, input):
    return self.main(input)
```

Layer (type)	Output Shape	Param #
ConvTranspose2d-1	[ -1, 512, 4, 4 ]	1,048,576
BatchNorm2d-2	[ -1, 512, 4, 4 ]	1,024
ReLU-3	[ -1, 512, 4, 4 ]	0
ConvTranspose2d-4	[ -1, 256, 8, 8 ]	2,097,152
BatchNorm2d-5	[ -1, 256, 8, 8 ]	512
ReLU-6	[ -1, 256, 8, 8 ]	0
ConvTranspose2d-7	[ -1, 128, 16, 16 ]	524,288
BatchNorm2d-8	[ -1, 128, 16, 16 ]	256
ReLU-9	[ -1, 128, 16, 16 ]	0
ConvTranspose2d-10	[ -1, 64, 32, 32 ]	131,072
BatchNorm2d-11	[ -1, 64, 32, 32 ]	128
ReLU-12	[ -1, 64, 32, 32 ]	0
ConvTranspose2d-13	[ -1, 3, 64, 64 ]	3,072
Tanh-14	[ -1, 3, 64, 64 ]	0

Total params: 3,806,080

Trainable params: 3,806,080

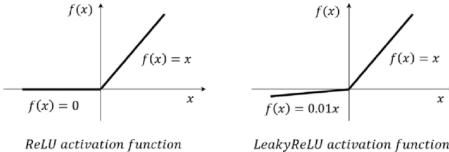
Non-trainable params: 0

[https://pytorch.org/tutorials/beginner/dcgan\\_faces\\_tutorial.html](https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html)

# ARCHITECTURE OF A GENERATIVE MODEL : DCGAN

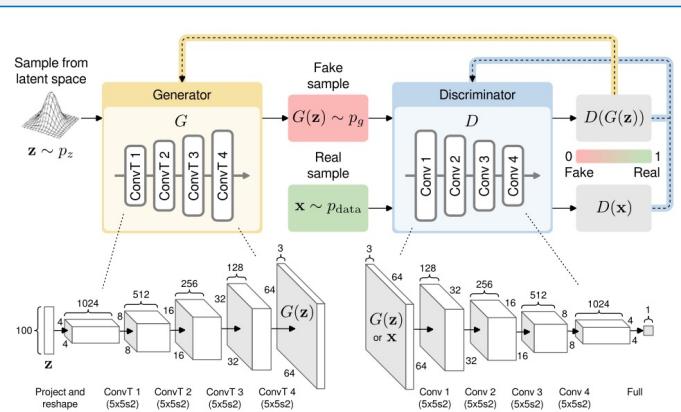
## • Discriminator

- The discriminator receives samples  $x$  from a real data distribution  $p_{\text{data}}$  and fake inputs  $G(z)$  alternately, updating its parameters  $\theta_D$  through backpropagation



`ReLU activation function`

`LeakyReLU activation function`



```
# Size of z latent vector (i.e. size of generator input)
nz = 128

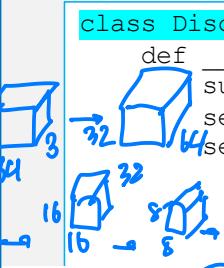
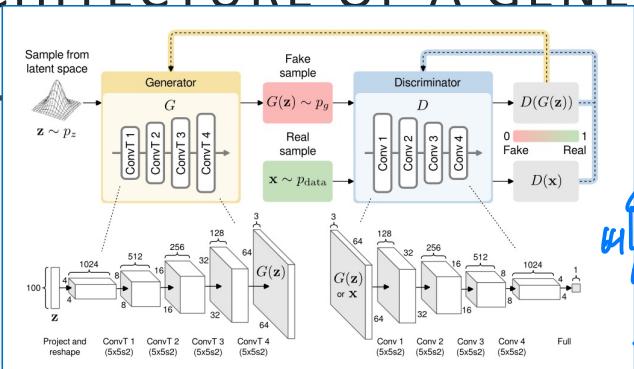
# Size of feature maps in generator
ngf = 64

# Size of feature maps in discriminator
ndf = 64

class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is ``(nc) x 64 x 64``
            nn.Conv2d(ndf, ndf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf) x 32 x 32``
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*2) x 16 x 16``
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*4) x 8 x 8``
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*8) x 4 x 4``
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

# ARCHITECTURE OF A GENERATIVE MODEL : DCGAN



Layer (type)	Output Shape	Param #
Conv2d-1	[ -1, 64, 32, 32]	3,072
BatchNorm2d-2	[ -1, 64, 32, 32]	128
LeakyReLU-3	[ -1, 64, 32, 32]	0
Conv2d-4	[ -1, 128, 16, 16]	131,072
BatchNorm2d-5	[ -1, 128, 16, 16]	256
LeakyReLU-6	[ -1, 128, 16, 16]	0
Conv2d-7	[ -1, 256, 8, 8]	524,288
BatchNorm2d-8	[ -1, 256, 8, 8]	512
LeakyReLU-9	[ -1, 256, 8, 8]	0
Conv2d-10	[ -1, 512, 4, 4]	2,097,152
BatchNorm2d-11	[ -1, 512, 4, 4]	1,024
LeakyReLU-12	[ -1, 512, 4, 4]	0
Conv2d-13	[ -1, 1, 1, 1]	8,192
Sigmoid-14	[ -1, 1, 1, 1]	0

Total params: 2,765,696  
 Trainable params: 2,765,696  
 Non-trainable params: 0

```
# Size of z latent vector (i.e. size of generator input)
nz = 128

# Size of feature maps in generator
ngf = 64

# Size of feature maps in discriminator
ndf = 64

class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is ``(nc) x 64 x 64``
            nn.Conv2d(ndf, ndf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf) x 32 x 32``
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*2) x 16 x 16``
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*4) x 8 x 8``
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*8) x 4 x 4``
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

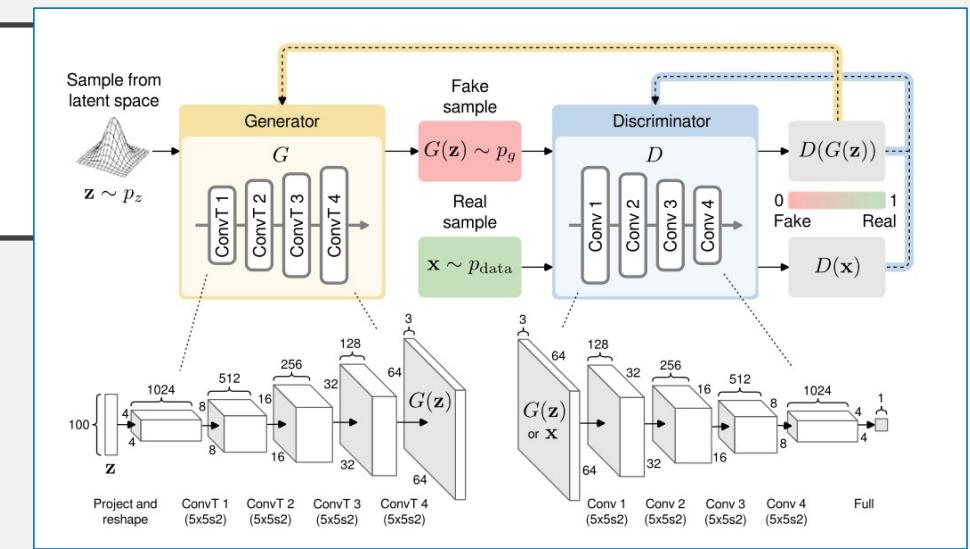
    def forward(self, input):
        return self.main(input)
```

# DCGAN : TRAINING PROCESS

```
# Create the Discriminator  
netD = Discriminator(ngpu).to(device)  
  
# Create the generator  
netG = Generator(ngpu).to(device)
```

Update D (discriminator) network

Update G (generator) network



```
# Initialize the ``BCELoss`` function  
criterion = nn.BCELoss()  
  
# Create batch of latent vectors that we will use to visualize  
# the progression of the generator  
fixed_noise = torch.randn(64, nz, 1, 1, device=device)  
  
# Establish convention for real and fake labels during training  
real_label = 1.  
fake_label = 0.  
  
# Setup Adam optimizers for both G and D  
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))  
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))
```

```

# For each epoch
for epoch in range(num_epochs):
    # For each batch in the dataloader
    for i, data in enumerate(dataloader, 0):

        #####
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        #####
        ## Train with all-real batch
        netD.zero_grad()
        # Format batch
        real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), real_label, dtype=torch.float, device=device)
        # Forward pass real batch through D
        output = netD(real_cpu).view(-1)
        # Calculate loss on all-real batch
        errD_real = criterion(output, label) BCE
        # Calculate gradients for D in backward pass
        errD_real.backward() manual Gradien
        D_x = output.mean().item()

        ## Train with all-fake batch
        # Generate batch of latent vectors
        noise = torch.randn(b_size, nz, 1, 1, device=device)
        # Generate fake image batch with G
        fake = netG(noise) return fake on G
        label.fill_(fake_label)
        # Classify all fake batch with D
        output = netD(fake.detach()).view(-1)
        # Calculate D's loss on the all-fake batch
        errD_fake = criterion(output, label)
        # Calculate the gradients for this batch, accumulated (summed) with previous gradients
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        # Compute error of D as sum over the fake and the real batches
        errD = errD_real + errD_fake
        # Update D
        optimizerD.step() JIT weight

```

## • Training process

- The discriminator receives samples  $x$  from a real data distribution  $p_{\text{data}}$  and fake inputs  $G(z)$  alternately, updating its parameters  $\theta_D$  through backpropagation

## Update D (discriminator) network

- Pass the real images with labels = 1 (**real**) into the netD model
- Generate the fake images using netG and pass these images with labels = 0 (**fake**) into the netD model
- Using BCE loss to cal **real\_error** and **fake error**

$$\text{loss} = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i)$$

- Combine **real\_error** + **fake\_error** and used that for updating the D network

```

# For each epoch
for epoch in range(num_epochs):
    # For each batch in the dataloader
    for i, data in enumerate(dataloader, 0):

        #####
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        #####
        ...
        #####
        # (2) Update G network: maximize log(D(G(z)))
        #####
        netG.zero_grad()
        label.fill_(real_label) # fake labels are real for generator cost
        # Since we just updated D, perform another forward pass of all-fake batch through D
        output = netD(fake).view(-1) discrimine fake
        # Calculate G's loss based on this output
        errG = criterion(output, label) means real 1
        # Calculate gradients for G
        errG.backward()
        D_G_z2 = output.mean().item()
        # Update G
        optimizerG.step()

```

### • Training process

- The parameters  $\theta_G$  of the generator are only updated when a fake sample is presented to the discriminator. The goal of the training is to have  $G(z, \theta_G)$  :  $p_z \rightarrow p_g$  able to generate samples similar to the original data, i.e.,  $p_g \sim p_{\text{data}}$ .

*Base on ph fake*

### Update G (generator) network

- Pass the fake images (previously generated) with real labels (1) into the netD
- Use BCE loss to calculate netD prediction error and use that for updating the weight for G network

គំរក random នៃ ការសម្រេចរួម ដែលអត្ថបាន

## ARCHITECTURE OF GENERATIVE MODEL : DCGAN

Real images



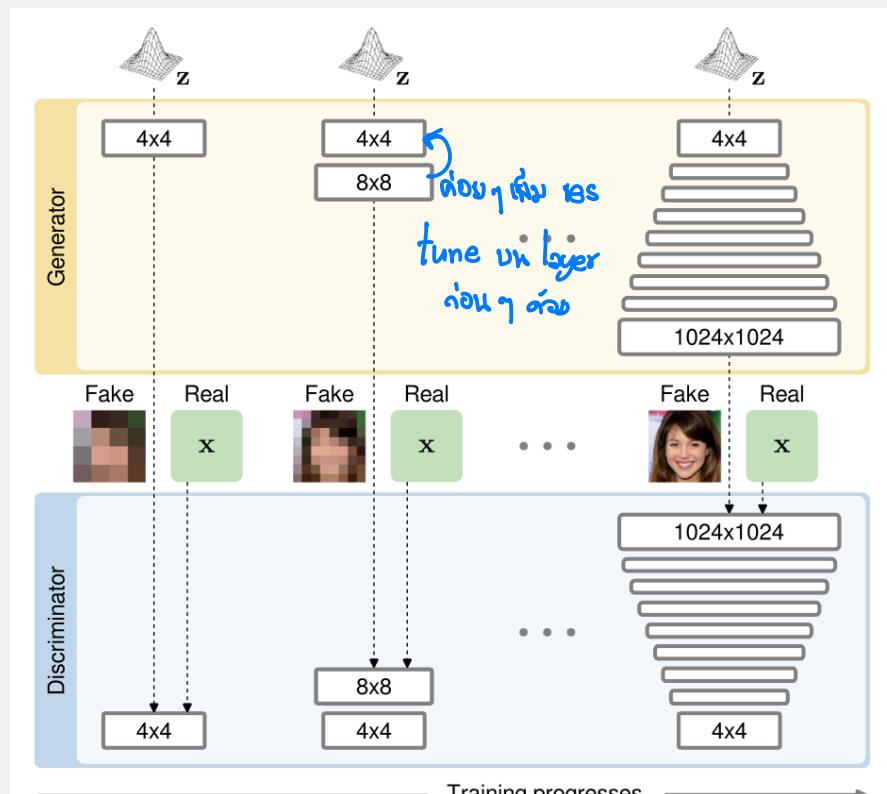
fake images



# EVOLUTION OF GENERATIVE MODELS FOR IMAGES

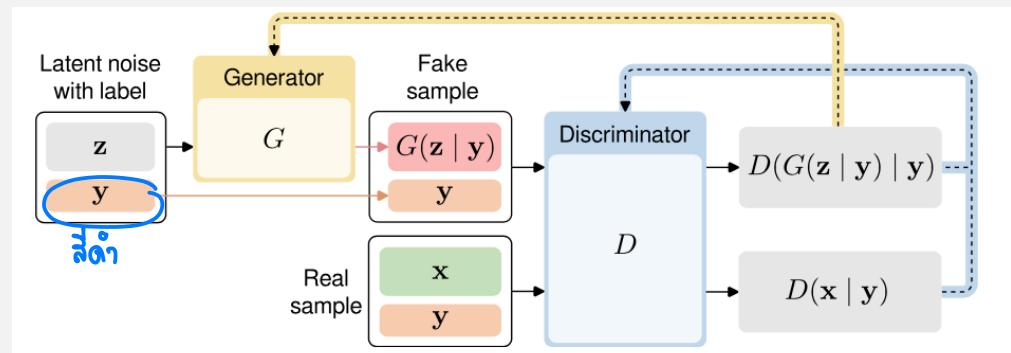
ດែលក្នុង resolution នៃការ resolution ដែលមាន

- **Progressive Growing of GANs (ProGAN)**
- The network is trained on a low resolution setting ( $4 \times 4$ ) and more layers are progressively inserted into the **generator** and **discriminator** to increase the resolution until the target dimensions are reached, in this case  $1024 \times 1024$ .



## EVOLUTION OF GENERATIVE MODELS FOR IMAGES

- **CGAN architecture**
- A condition  $y$  is concatenated with both the **generator** and **discriminator** inputs. This allows for better control of the generated images, encouraging the **generator** to create images determined by the condition  $y$ .
- Condition can be the class from which the data belongs, other image, embedded textual vector.

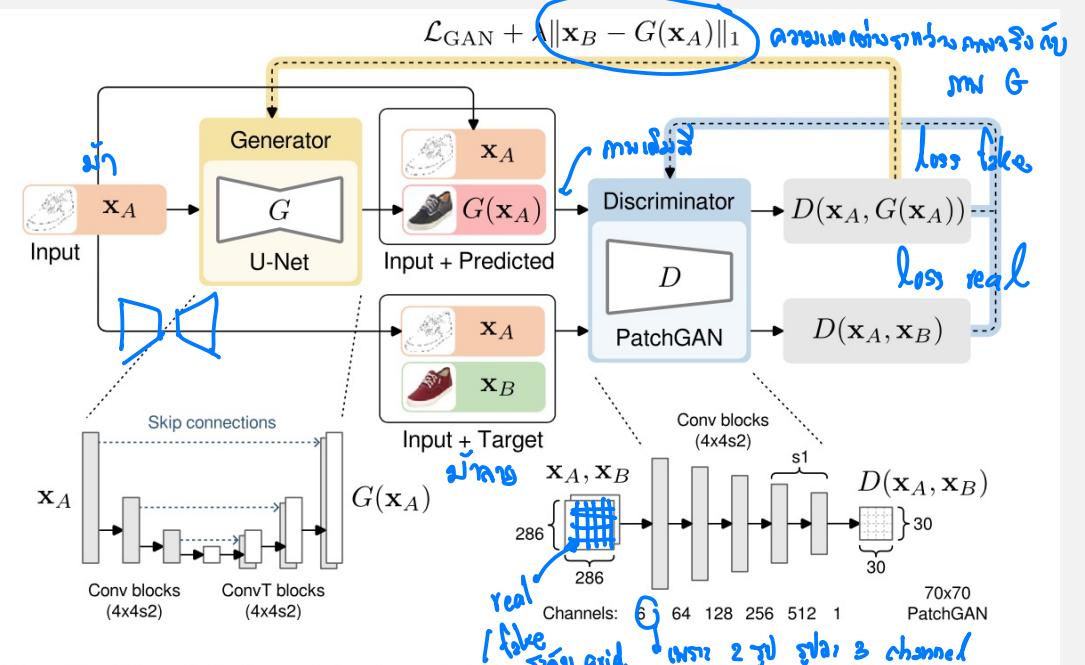


ප්‍රිංග ප්‍රකාශන → උග්

condition image

## ARCHITECTURE OF GENERATIVE MODEL : PIX2PIX

- The pioneering work for **image-to-image** translation using GANs is called pix2pix.
- to perform supervised translations across multiple domains, e.g., the creating realistic street photos from semantic annotations, detailed building facades from architectural label maps, reconstructing photographs from edges, and more.
- Input** : image pairs -  $x_A$  and  $x_B$  (input & target)
- the use of the PatchGAN discriminator, which outputs a matrix instead of a single scalar.



**Fig. 13.** The pix2pix architecture is based on the conditional GAN (CGAN). The image  $x_A$  serves as the condition for the generator and discriminator, and no noise input is used on the generator. Instead of a single scalar, the PatchGAN discriminator outputs a matrix, in which each element represents a patch of the input image.

Isola P, Zhu J-Y, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on CVPR. 2017, p. 1125–34.

# ARCHITECTURE OF GENERATIVE MODEL : PIX2PIX

**Adversarial Loss ( $L_{GAN}$ ):** Ensures

the generated images are

indistinguishable from real images

- The generator loss includes an L1 term,  $\|x_B - G(x_A)\|_1$ , evaluated from the absolute pixel-level difference between the generated image and the target. The total generator loss is then  $L_{GAN} + \lambda \|x_B - G(x_A)\|_1$ , where  $L_{GAN}$  represents the adversarial loss. The parameter  $\lambda$  is used to adjust the impact of the L1 loss on the training, and is defined by the user.
- Applications of the Pix2Pix GAN**
  - Semantic labels  $\leftrightarrow$  photo, trained on the Cityscapes dataset.
  - Map  $\leftrightarrow$  aerial photo, trained on data scraped from Google Maps.
  - Black and White  $\rightarrow$  color photo.
  - Edges / Sketch  $\rightarrow$  photo.
  - Day  $\rightarrow$  night photographs.
  - Thermal  $\rightarrow$  color photos.
  - Photo with missing pixels  $\rightarrow$  inpainted photo, trained on Paris StreetView.

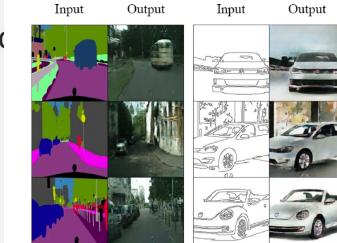
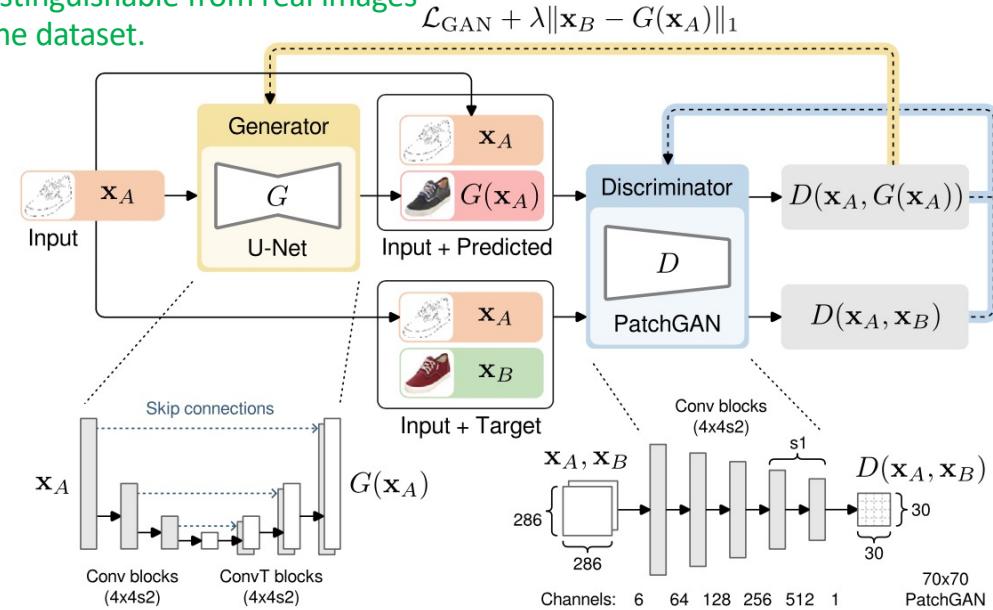


Fig. 12. Examples of images generated by applying pix2pix to synthesize street scenes from semantic maps (left) in the cityscapes dataset [84], and to create car images from edges (right). The input column presents the images  $x_A$  inserted into the generator, while the corresponding generated images  $G(x_A)$  are presented on the output column.



**Fig. 13.** The pix2pix architecture is based on the conditional GAN (CGAN). The image  $x_A$  serves as the condition for the generator and discriminator, and no noise input is used on the generator. Instead of a single scalar, the PatchGAN discriminator outputs a matrix, in which each element represents a patch of the input image.

Isola P, Zhu J-Y, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on CVPR. 2017, p. 1125–34.

# APPLICATIONS OF GENERATIVE MODELS FOR IMAGES

Summary of the methods discussed in Section 4.				
Model/Technique	Year	Application	Contribution highlights	
StyleGAN [19]	2019	Single-mode image synthesis, image editing.	Creation of a mapping network for better disentanglement. Use of AdaIN layers to input information on every resolution. Style mixing by including different vectors as inputs.	Pivotal-Tuning Inversion (PTI) [32] 2022 Image editing, GAN inversion Obtaining a pivot code $\mathbf{w} \in \mathcal{W}$ from the image $\mathbf{x}$ through inversion, and fine-tuning the StyleGAN base framework to recreate the specific image.
StyleGAN2 [20]	2020	Single-mode image synthesis, image editing.	Improvement of the progressive growing approach of StyleGAN. Use of PPL as regularization. Replacement of AdaIN with weight demodulation.	pix2pix [33] 2017 Image-to-image translation, style transfer. Use of a conditional GAN to allow the editing of image inputs with known targets. PatchGAN discriminator. Use of the L1 norm in the loss to provide supervision to the generator.
StyleGAN2-ADA [21]	2020	Single-mode image synthesis, image editing.	Adaptive Discriminator Augmentation (ADA).	pix2pixHD [34] 2018 Image-to-image translation, style transfer. The generator split into a global generator and a local enhancer. Combination of three PatchGAN discriminators in different scales. Replacement of the L1 loss with a feature-matching loss.
StyleGAN3 [22]	2021	Single-mode image synthesis, image editing.	Creation of translation- and rotation-equivariant layers. Remove progressive growing presented in previous StyleGANs.	GauGAN [35] 2019 Image-to-image translation with semantic map inputs Creation of the SPADE layer. Insertion of the semantic map input into all SPADE layers.
SAGAN [23]	2019	Multi-modal image synthesis	Use of self-attention mechanism in the network.	Swapping Autoencoder [36] 2020 Image-to-image translation, style transfer. Creation of an encoder which embeds the image input into a spatial latent tensor and a style latent vector. Style transfer by swapping codes from two different images.
BigGAN [24]	2019	Multi-modal image synthesis	Built from the SAGAN architecture. Improved stability and increased size.	CycleGAN [37] 2017 Image-to-image translation, style transfer. Use of two generator-discriminator pairs to create a cyclic network. Cycle-consistency loss. Capability of unsupervised training.
StyleGAN-XL [25]	2022	Multi-modal image synthesis	Built from the StyleGAN3-T architecture. Progressive growing. Class embedding information on the network. Increased size of the network. Use of a feature network and discriminators from Projected GAN.	SRGAN [38] 2017 Super resolution ResNet-based generator. Use of a perceptual loss, based on the activations of a VGG-19 pretrained network.
StyleCLIP [26]	2021	Text-guided image editing	Use of CLIP embeddings to allow text-guided image editing on the latent space of a StyleGAN. Mapping the text inputs into directions on the latent space.	ESRGAN [39] 2018 Super resolution Built from the SRGAN architecture. Use of Residual in residual dense blocks (RRDB) on the generator. Use of a relativistic discriminator. Pretraining of the generator with L1 loss, before the adversarial training.
VQGAN-CLIP [27]	2022	Text-to-image synthesis	Use of VQGAN encoder to create the latent vector and VQGAN decoder to create the image. Optimization of the latent vector to encourage the image to have high CLIP similarity with the text prompt.	Real-ESRGAN [40] 2021 Super resolution, image restoration. Built from the ESRGAN architecture. Trained with images passed through a synthetic degradation pipeline. Use of U-Net as a discriminator.
StyleGAN-T [28]	2023	Text-to-image synthesis	Built from the StyleGAN-XL architecture, with increased network size. Adapted to include textual information from CLIP encoders.	GFP-GAN [41] 2021 Facial image restoration Use of a U-Net as a degradation removal component and feature extractor. Use of a StyleGAN previously trained on faces to better reconstruction of facial features.
image2stylegan [29]	2019	Image editing, GAN inversion	Use of the extended latent space $\mathcal{W}^+$ . Optimized latent vector based on L2 and perceptual losses.	
image2stylegan++ [30]	2020	Image editing, GAN inversion	Built upon image2stylegan technique. Optimization in both the latent vector $\mathbf{w}$ and the noise vector $\mathbf{n}$ .	
In-Domain GAN Inversion [31]	2020	Image editing, GAN inversion	Creation of a generalist domain-guided encoder to invert the image to its latent code. Use of the domain-guided encoder in the optimization steps.	

# POPULAR GANS

## Methods

Method	Year	Papers
 <b>GAN</b> Generative Adversarial Networks	2014	1203
 <b>CycleGAN</b> Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks	2017	313
 <b>StyleGAN</b> A Style-Based Generator Architecture for Generative Adversarial Networks	2018	221
 <b>StyleGAN2</b> Analyzing and Improving the Image Quality of StyleGAN	2019	168
 <b>SAGAN</b> Self-Attention Generative Adversarial Networks	2018	136
 <b>BigGAN</b> Large Scale GAN Training for High Fidelity Natural Image Synthesis	2018	101
 <b>Pix2Pix</b> Image-to-Image Translation with Conditional Adversarial Networks	2016	92
 <b>WGAN</b> Wasserstein GAN	2017	85
 <b>DCGAN</b> Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks	2015	73

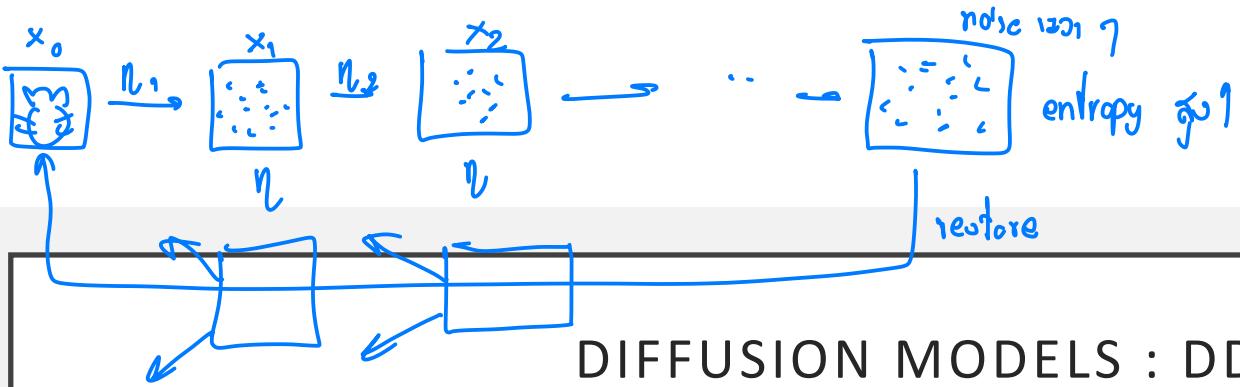
<https://paperswithcode.com/methods/category/generative-adversarial-networks>

## LIMITATIONS OF GAN

- GANs suffer from well-known issues during training, such as mode collapse (where the generator only learns to produce a limited range of outputs) and unstable gradients. The adversarial dynamics that can destabilize GANs.
- struggle with capturing the full diversity of the data distribution

ការរាយការណ៍ហាក់ហាក់

សំណង់ limited នៃ  
mr output

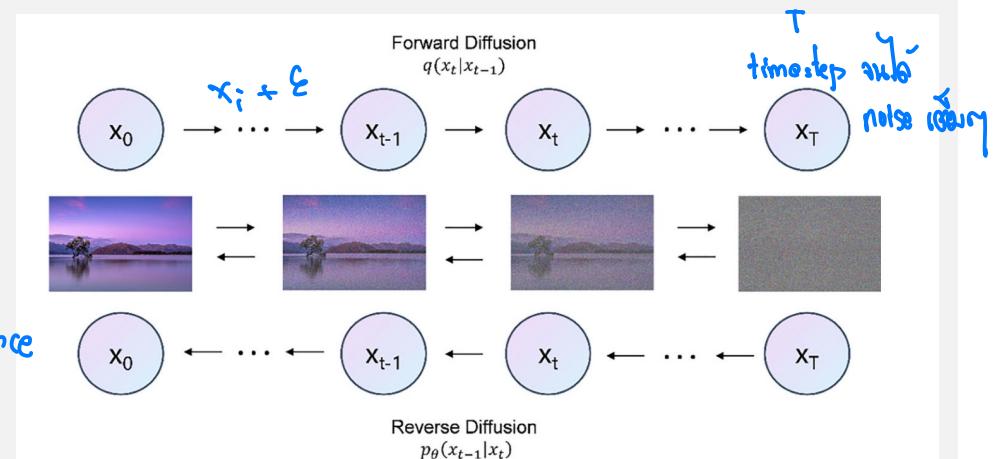


- DDPM is a parametric Markov chain that generates an image by progressively removing noise from the image 
$$x_t = x_{t-1} + \epsilon^{\text{noise}}$$
- Denoising Diffusion Probabilistic Model (DDPM) - The core idea of this model is to simulate the diffusion and restoration processes in physical systems to generate high-quality data samples. It achieves realistic image generation by introducing noise into existing images and then gradually removing it.
- The name “diffusion” comes from the fact that the model starts with a high-entropy image (i.e., a random image with no structure) and then gradually diffuses the entropy away, making the image more structured and realistic.

model រួប នៅ នៅ លើកដាក់ ក្នុង entropy

## DIFFUSION MODELS : DDPM

- Two processes: the forward process and the reverse process
  - The forward process is the gradual addition of Gaussian noise to the image data until it becomes random noise *ก้าม gaussian*
  - The reverse process is a denoising *Test / Inference* procedure that generates samples from the data distribution by iteratively removing noise step by step



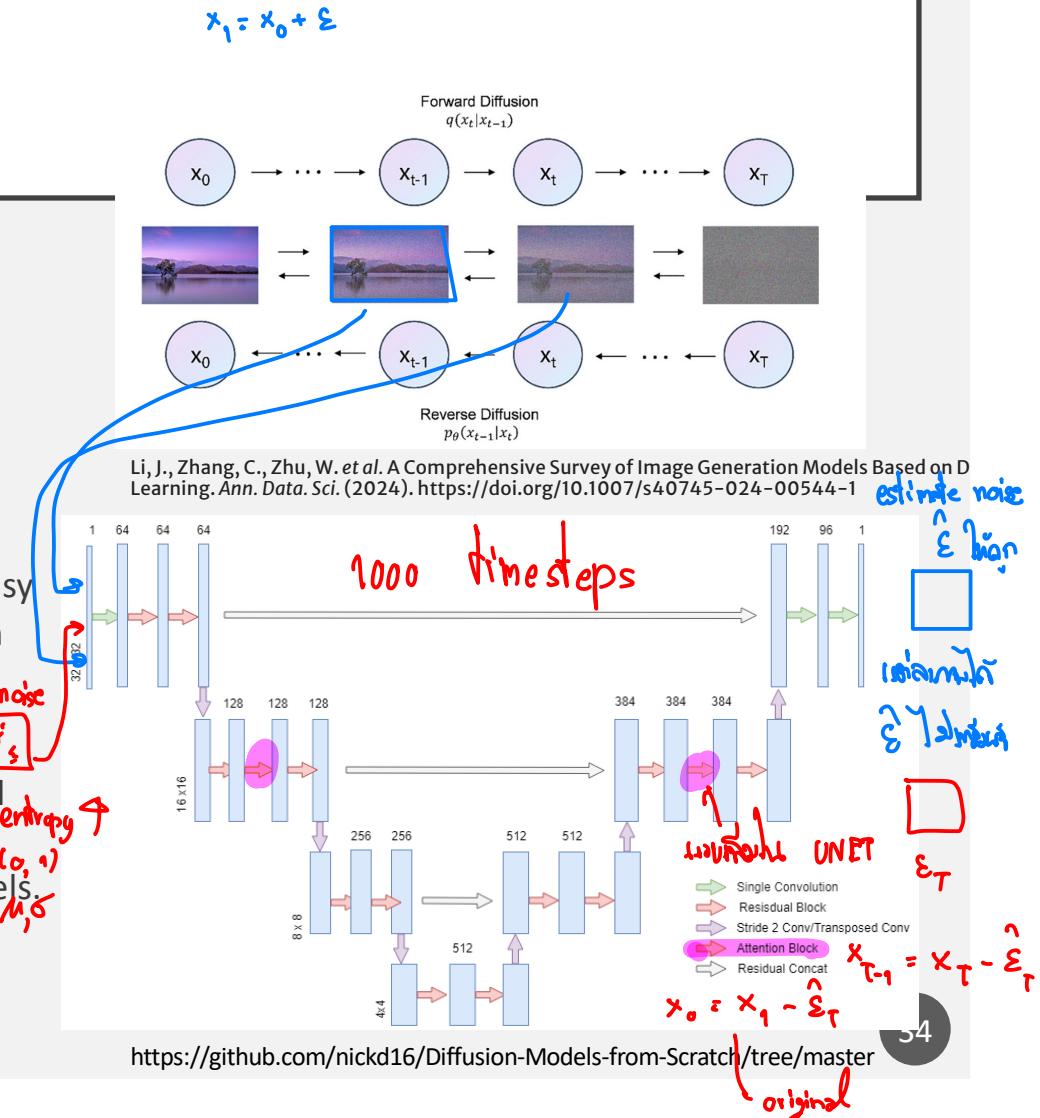
**Fig. 4** Diffusion models comprise a forward process and a reverse process. The forward process incrementally adds noise to the original image, enabling the model to learn the noise distribution. Conversely, the reverse process generates images by denoising the noise-added images

Li, J., Zhang, C., Zhu, W. et al. A Comprehensive Survey of Image Generation Models Based on Deep Learning. *Ann. Data. Sci.* (2024). <https://doi.org/10.1007/s40745-024-00544-1>

# DIFFUSION MODELS : DDPM

The forward process (Training):

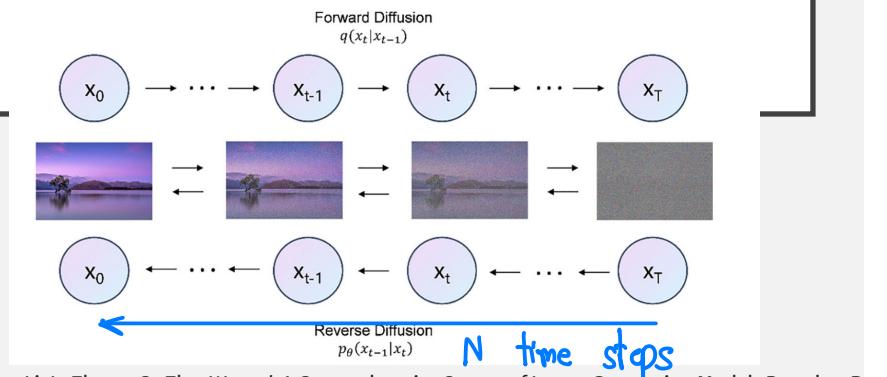
1. The original image ( $x_0$ ) is slowly corrupted iteratively (a Markov chain) by adding (scaled Gaussian) noise.
2. This process is done for some  $T$  time steps, i.e.,  $x_T$ .
3. Image at timestep  $t$  is created by:  $x_{t-1} + \epsilon_{t-1}$  (noise)  $\rightarrow x_t$
4. The model, such as a U-Net or neural network, takes the noisy image  $x_t$  and the timestep  $t$  as inputs and outputs a prediction for the noise  $\epsilon$ .
5. Mean Squared Error (MSE) used as the loss function to measure how close the model's predicted noise is to the actual noise added. Use backpropagation to update the model's parameters, minimizing the MSE loss across multiple noise levels.
6. Iterate over many epochs, adding noise to each input data point at different random timesteps  $t$  in each batch.



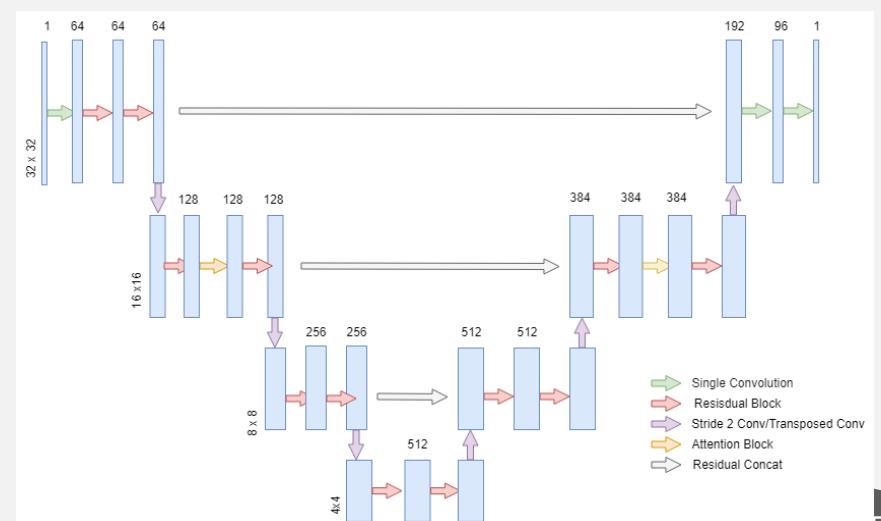
# DIFFUSION MODELS : DDPM

The Reverse process (Inference):

- Initialize with Pure Noise, then Iterative Denoising from each time step T, T-1, .. to 0. progressively denoise  $x_t$  to generate a less noisy version  $x_{t-1}$ . The goal is to gradually remove noise from  $x_t$  until you reach  $x_0$



Li, J., Zhang, C., Zhu, W. et al. A Comprehensive Survey of Image Generation Models Based on D Learning. *Ann. Data. Sci.* (2024). <https://doi.org/10.1007/s40745-024-00544-1>



<https://github.com/nickd16/Diffusion-Models-from-Scratch/tree/master>

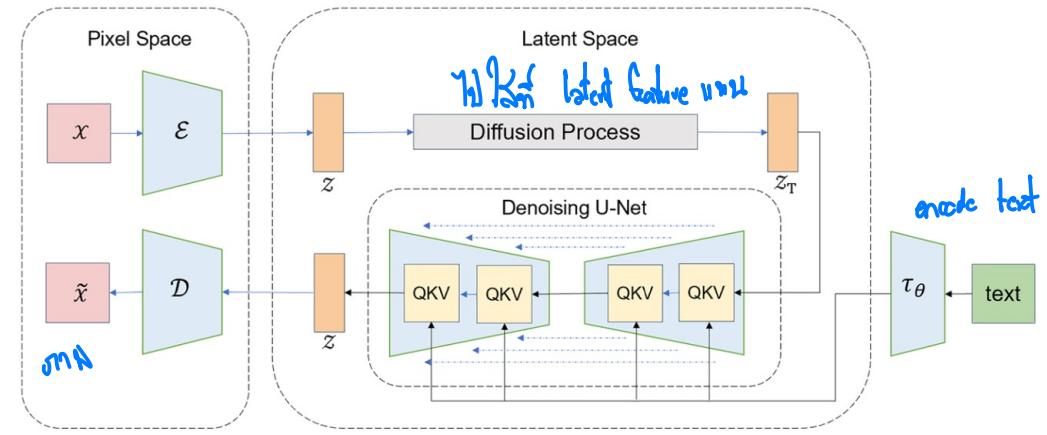
DDPM Demo:

<https://colab.research.google.com/drive/1TrycJ3iy4RtD8LDO0aJSIhQI7-6xDI1-?usp=sharing>

References : <https://github.com/nickd16/Diffusion-Models-from-Scratch/tree/master>

## DIFFUSION MODELS

- Stable diffusion - published by Stability AI, a generation model based on Latent Diffusion Models
  - LDMs perform the diffusion process in the latent space, which greatly improves the sampling speed and efficiency.
  - The core idea of LDMs is to compress the original image into the latent space using an autoencoder which reduces the computational complexity during the sampling phase.
  - The diffusion process in the latent space is similar to the standard diffusion model, specifically implemented as a U-Net.
  - a domain-specific encoder  $\tau_\theta$  to map conditional information into an intermediate representation which is then integrated into the U-Net's intermediate layers through a cross-attention layer mapping



**Fig. 8** In the architecture of Latent Diffusion Models, images are outputted to a latent space through an encoder, where the diffusion process takes place. The reverse process employs a U-Net architecture. Text prompts are injected into the self-attention modules of the U-Net via an encoder

**Table 6** Major multimodal image generation large-scale models

Year	Method	Code	Forks	Stars
2023	Stable Diffusion XL [82]	<a href="https://github.com/Stability-AI/generative-models">https://github.com/Stability-AI/generative-models</a>	2.3 k	18.8 k
2023	DALL-E 3	—	—	—
2022	Muse [64]	—	—	—
2022	Parti [80]	—	—	—
2022	Imagen [78]	<a href="https://github.com/lucidrains/imagen-pytorch">https://github.com/lucidrains/imagen-pytorch</a>	715	7.6 k
2022	DALL-E 2 [77]	<a href="https://github.com/lucidrains/DALLE2-pytorch">https://github.com/lucidrains/DALLE2-pytorch</a>	1 k	10.6 k
2022	Stable Diffusion [12]	<a href="https://github.com/CompVis/stable-diffusion">https://github.com/CompVis/stable-diffusion</a>	9.6 k	62.9 k
2021	GLIDE [76]	<a href="https://github.com/openai/glide-text2im">https://github.com/openai/glide-text2im</a>	479	3.4 k
2021	DALL-E [74]	<a href="https://github.com/openai/DALL-E">https://github.com/openai/DALL-E</a>	1.9 k	10.6 k

# DIFFUSION MODELS

## Popular Diffusion tools

- DALL-E, Midjourney

## Diffusion vs. GAN

- Diffusion models are more stable. GANs can be unstable, and they can sometimes generate images that are not realistic or that are not consistent with the training data.
- diffusion models are more versatile than GANs. Diffusion models can be used to generate images from a variety of different domains,

Li, J., Zhang, C., Zhu, W. et al. A Comprehensive Survey of Image Generation Models Based on Deep Learning. *Ann. Data. Sci.* (2024). <https://doi.org/10.1007/s40745-024-00544-1>

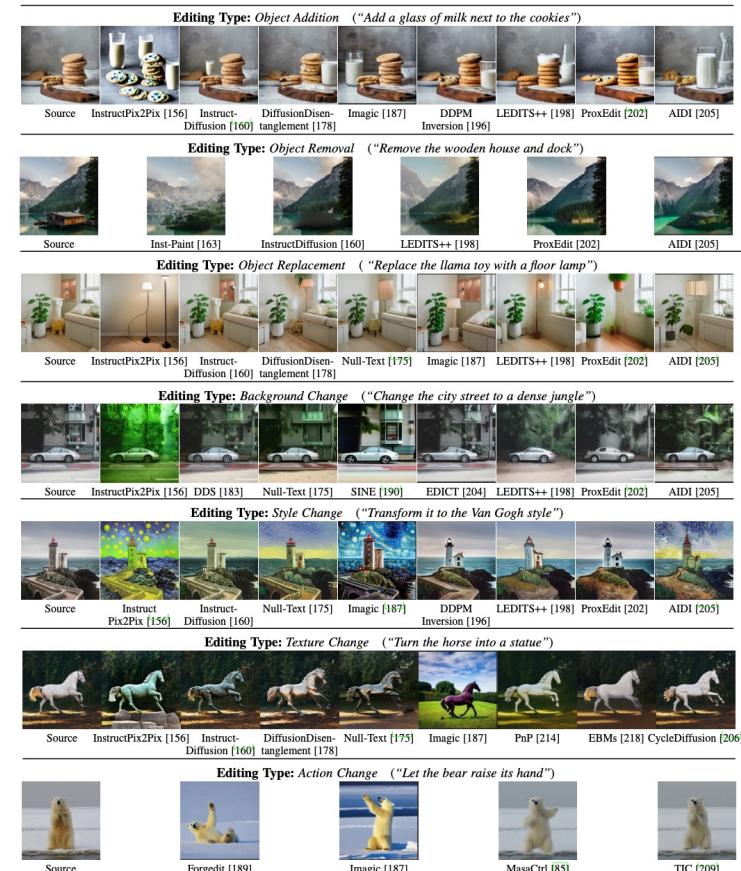
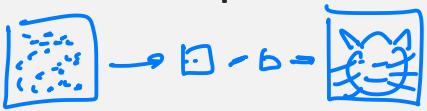


Fig. 13: Visual comparison on 7 selected editing types.

## FUTURE DIRECTIONS

- Fewer-step Model Inference : few-step or one- step generation diffusion models នូវ time step ងាយ ក្នុង 
- Efficient Models : backbone, top-layers
- Complex Lighting and Shadow Editing សម្រេច / គាំទ្រ correlate ពីកម្មភាព
- Unrobustness of Image Editing : not capable of accurately modeling all possible samples in the conditional distribution space ការណើតសែរ នឹងពេល (នៅ cond; ត្រូវបាន)
- Faithful Evaluation Metrics : PSNR, and SSIM used for a reference, mid-level similarity of two images ការវិនិច្ឆ័យថ្មី ពេកវិនិច្ឆ័យសំខាន់

## ETHICAL CONSIDERATIONS AND FUTURE TRENDS

- Deepfakes and Misinformation: One of the most prominent concerns with GANs is their ability to create realistic but fake images, videos, and audio recordings, known as deepfakes. These can be used to spread misinformation, manipulate public opinion, or defame individuals, posing a significant threat to personal reputation, political integrity, and even national security.
- Consent and Privacy Violations: GANs can generate realistic images of people who never consented to have their likeness used, including generating images of people in compromising or controversial situations. This raises serious privacy concerns and questions about the ethics of using someone's likeness without their permission. *ការប្រើប្រាស់រឹងរាល់របស់វា] generate*
- Bias and Discrimination: Like many AI models, GANs can inherit and amplify biases present in their training data. If the training data is biased, the generated outputs can perpetuate stereotypes and discriminatory practices. This is particularly concerning in applications like facial recognition, where biased data can lead to unfair or prejudiced outcomes. *តាមពេល bias model និង bias នៅក្នុងពេការណ៍នៃការ data*
- Artistic and Intellectual Property Rights: GANs can replicate the style of artists or generate new artworks, raising questions about intellectual property rights and the originality of art. Determining the ownership of AI-generated content and the ethical implications of AI mimicking human creativity are ongoing debates. *គីឡូទីតិចៗ និង AI ?*
- Economic Impact: The ability of GANs to generate realistic images and videos could impact industries like photography, modeling, and film, potentially displacing jobs. While they can also create new opportunities and efficiencies, the displacement of traditional roles raises concerns about economic impacts and the future of work. *ការរករាយ sketch និង disrupt តម្លៃ ?*

## REFERENCES

- Trevisan de Souza V.L., Marques B.A.D., Batagelo H.C., Gois J.P., A review on Generative Adversarial Networks for image generation, *Computers & Graphics* (2023)
- Isola P, Zhu J-Y, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, p. 1125–34.