LECTURE 13 GENERATIVE ADVERSARIAL MODELS FOR IMAGES

Punnarai Siricharoen, Ph.D.

TODAY'S CONTENT

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

IMAGE GENERATION MODELS

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

PixtoPix (2017)
styleGAN (2018)

stable until now

- Banana
- SoRA
- DALL-E 2

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

CNN
FCN

VARIATIONAL AUTOENCODERS (VAE) 2013

mean

- 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

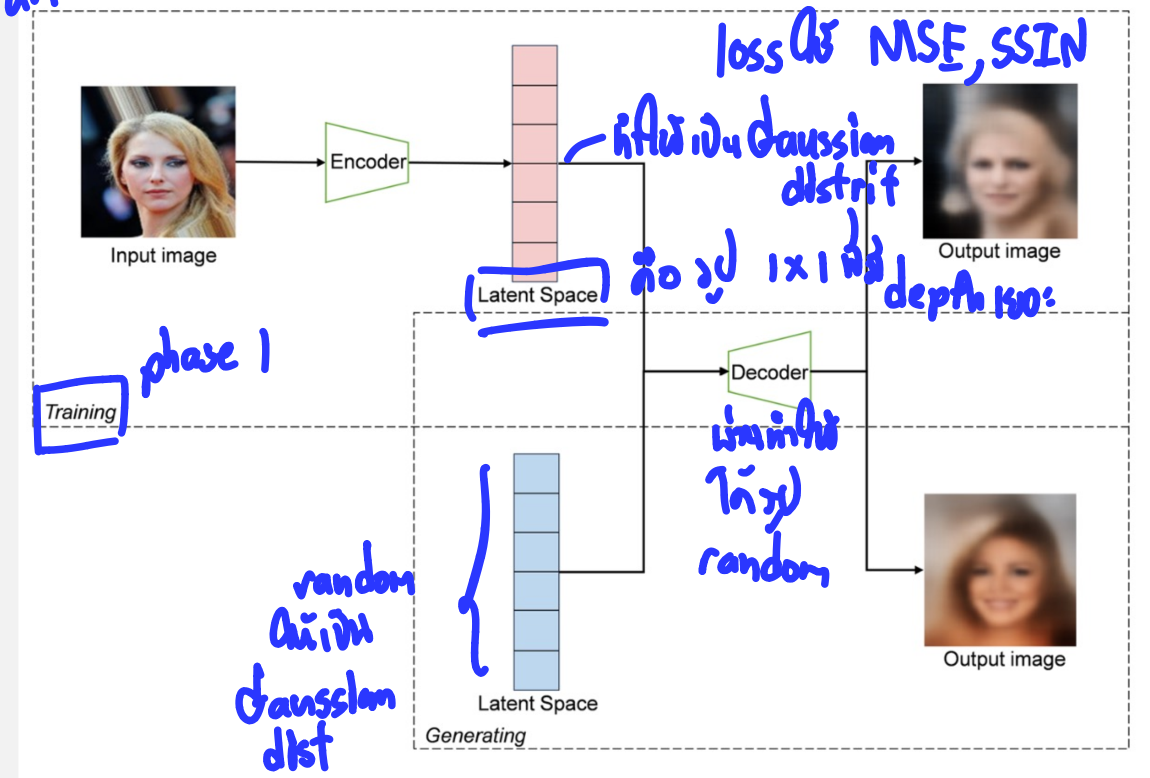


Fig. 2 The model structure of VAE consists of an encoder, which maps the image to the latent space, and a decoder, which recovers the image from the latent space to the pixel space

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

VARIATIONAL AUTOENCODERS (VAE)

- Improvements :
  - CVAE - introduces condition information in order to control the generated samples under given conditions
  - 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)

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

WHAT IS THE GENERATIVE ADVERSARIAL MODEL?

แข่งกัน

- Generative Adversarial Networks $\boxed{\text{(GANs)}}$ are a type of deep learning architecture to capture the training data distribution so we can generate new data from that same distribution.
  - 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 ~ คล้าย VAE - to generate synthetic examples based on the training data
    - a discriminator ~ แยกให้ได้ trained to distinguish synthetic (fake) instances created by the generator from real data
    - This competition leads the generator to produce very realistic samples

APPLICATIONS OF GENERATIVE MODELS FOR IMAGES

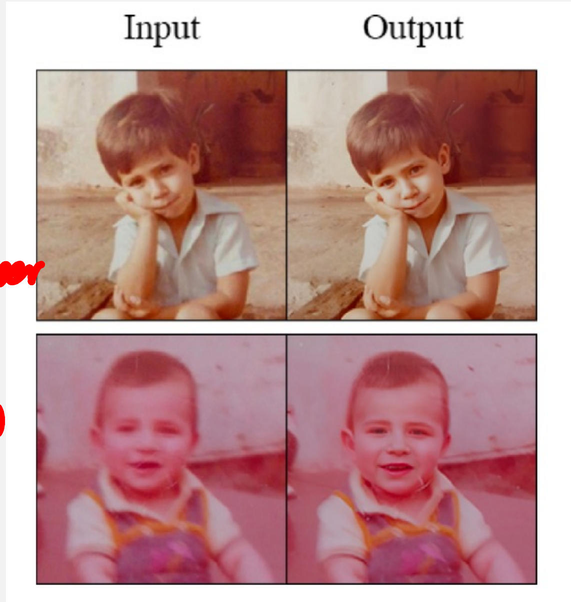
- To create a person/things that never exist - image manipulation from latent spaces, image-to-image translation, style transfer, and image repair
- To create more images for supervised learning - image synthesis

dataset มี high reso ทำให้เป็น low reso แล้วนำไปเทสเป็น ส่ง interpolation model

resteration

same resolution but look sharper

enhancement



Examples of using GFP-GAN to restore old facial photographs

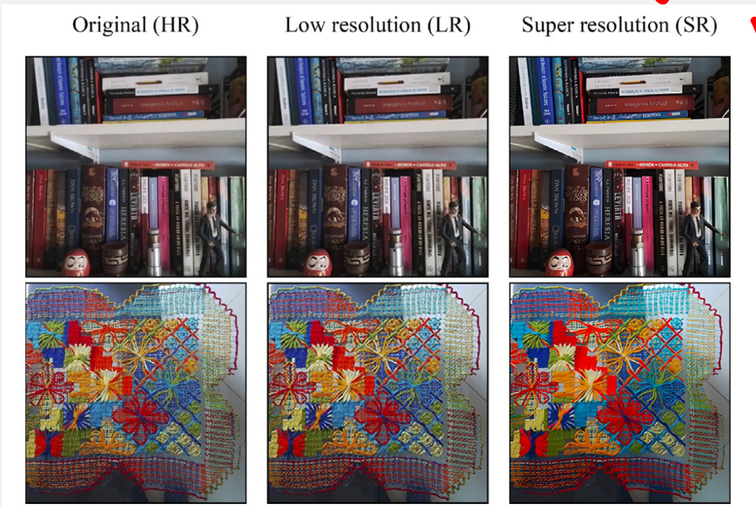


Fig. 18. Applying Real-ESRGAN in pictures of a bookshelf and a tablecloth with a detailed pattern. The original images were downsampled 4x with a bicubic kernel to generate the LR ones, which were applied to the GAN to retrieve the SR images. On the bookshelf SR image, some words could not be correctly 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.

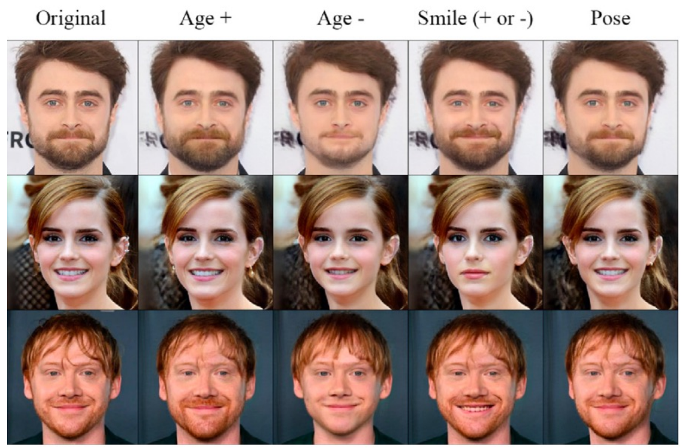
Super resolution

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)

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



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

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)

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

เปลี่ยน style

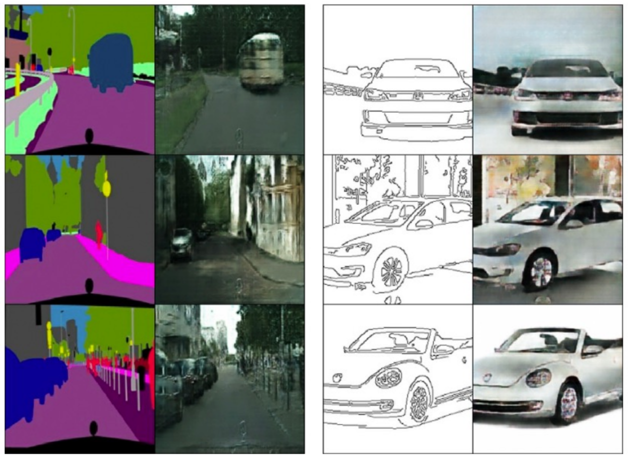


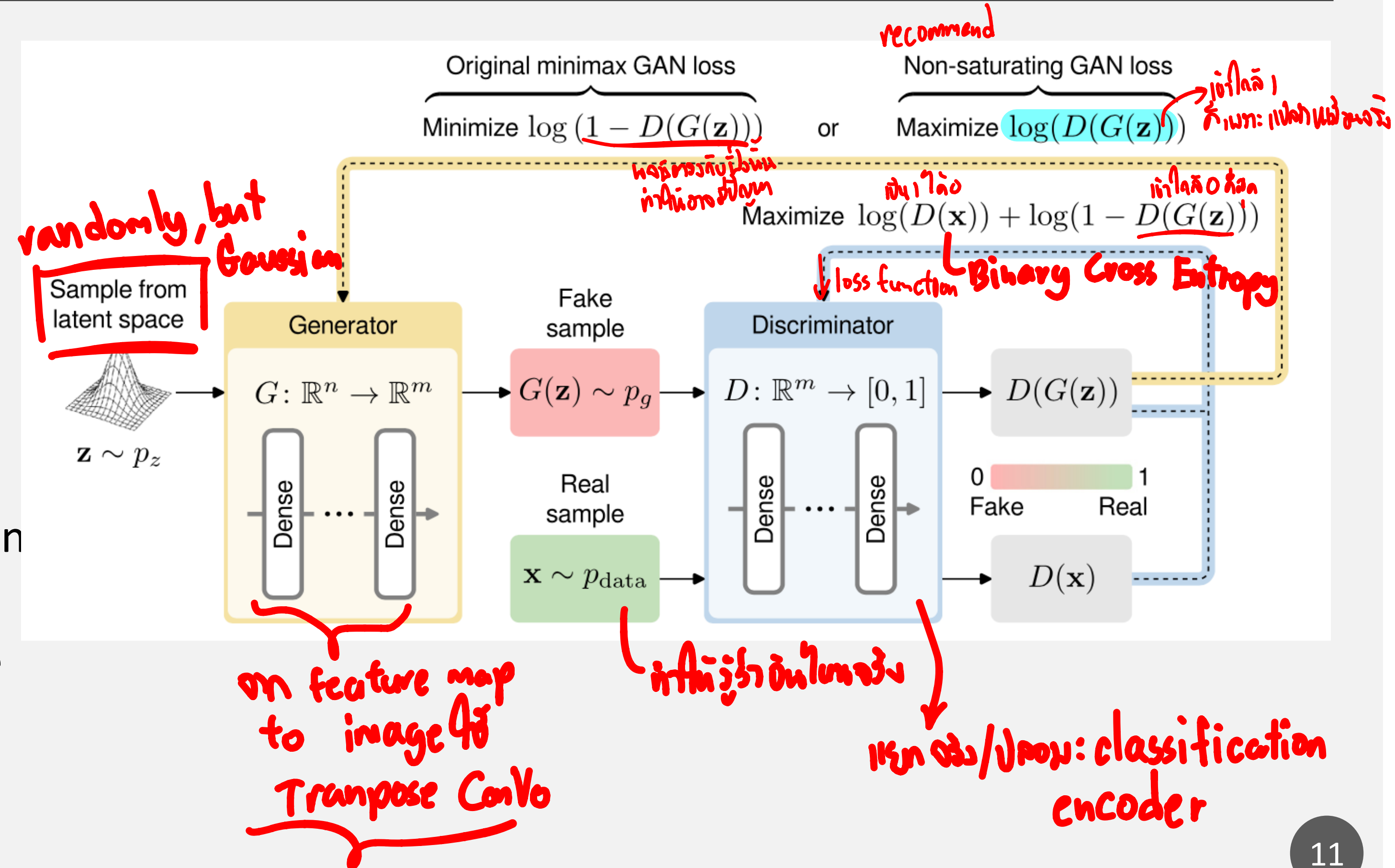
Fig. 12. Examples of images generated by applying pix2pix to synthesize street scenes from semantic maps (left) in the cityscapes dataset [84], and to create car images from edges (right). The input column presents the images X_A inserted into the generator, while the corresponding generated images $G(X_A)$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

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)

GENERATIVE MODELS - ARCHITECTURE

- 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).

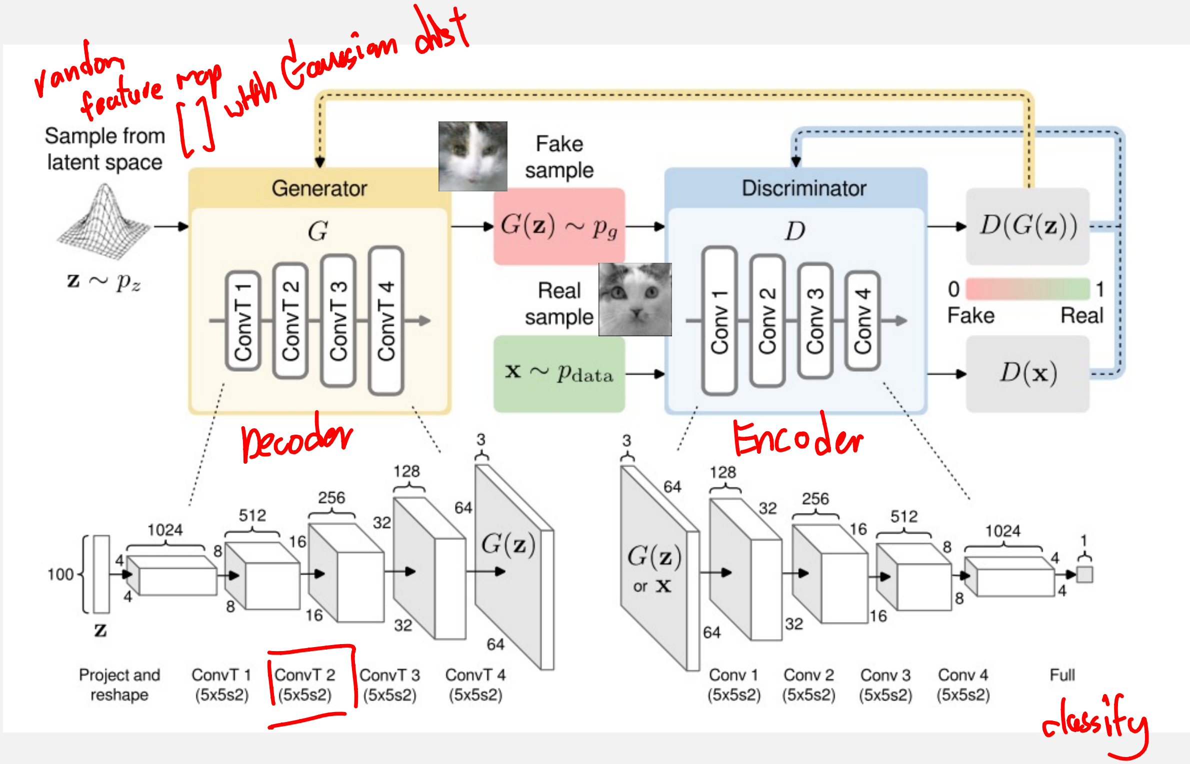


Decoder

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)

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.



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)

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

ARCHITECTURE OF A GENERATIVE MODEL : DCGAN

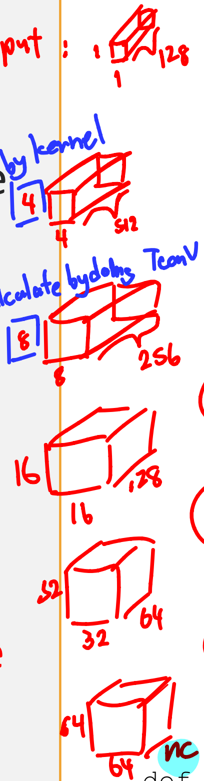
output size of TconV = (input_size − 1) · stride − 2×pad + kernel_size

- 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)

Input : 1×1×128

by kernel

calculate by doing TconV



depend on channel

```
# 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
```

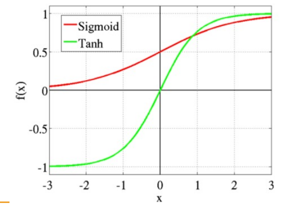
```
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf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

input

output : 64×8 = 512

1
2
3
4
5

Transpose ConVo 5 3
Bu



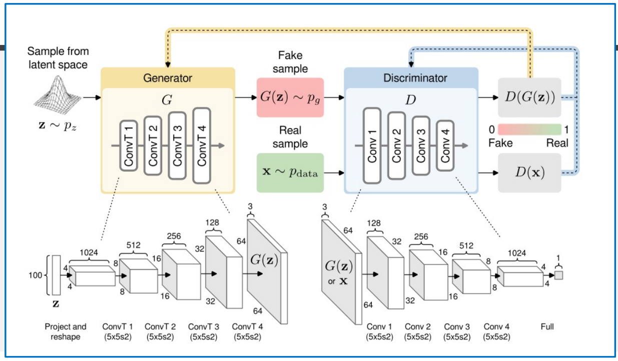
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

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 M… CGAN

modify the input directly, without allocating any additional output.

```
# 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 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)
```

num_features

$(4\times4\times128 + 1)\times512$

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

2×512

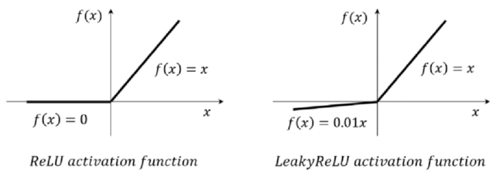
Total params: 3,806,080
Trainable params: 3,806,080
Non-trainable params: 0

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

ARCHITECTURE OF A GENERATIVE MODEL : DCGAN

```python
# 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
```

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

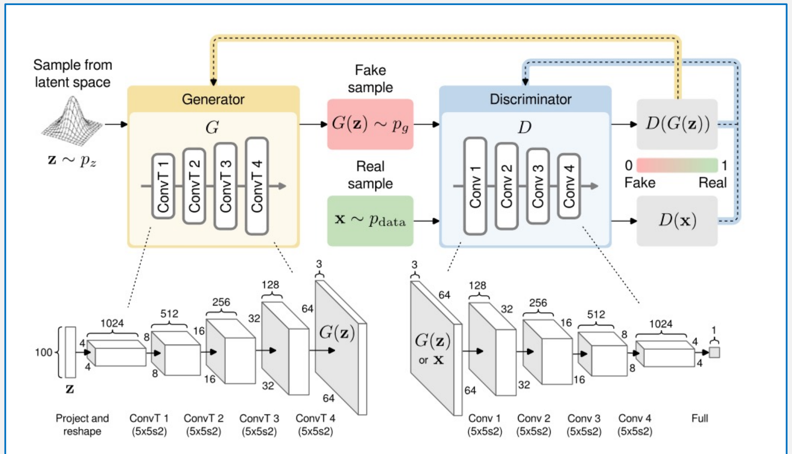



```python
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is ``(nc) x 64 x 64``
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf),
            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)
```

ทำให้ผลลัพธ์ลดลงครึ่งนึงแทน maxpool

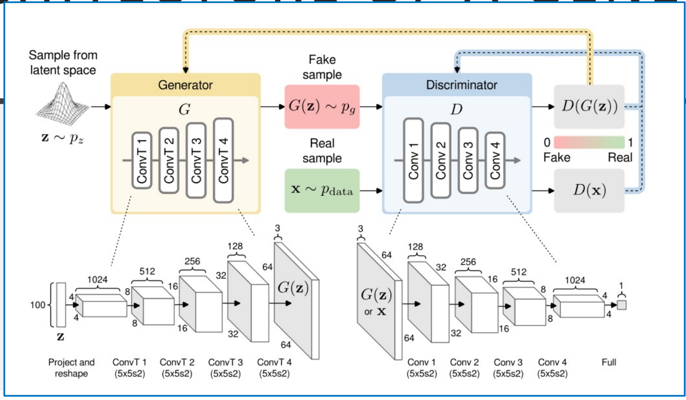
แทนผล

ได้ binary



ARCHITECTURE OF A GENERATIVE MODEL : 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
```




```
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(nc, 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)
```

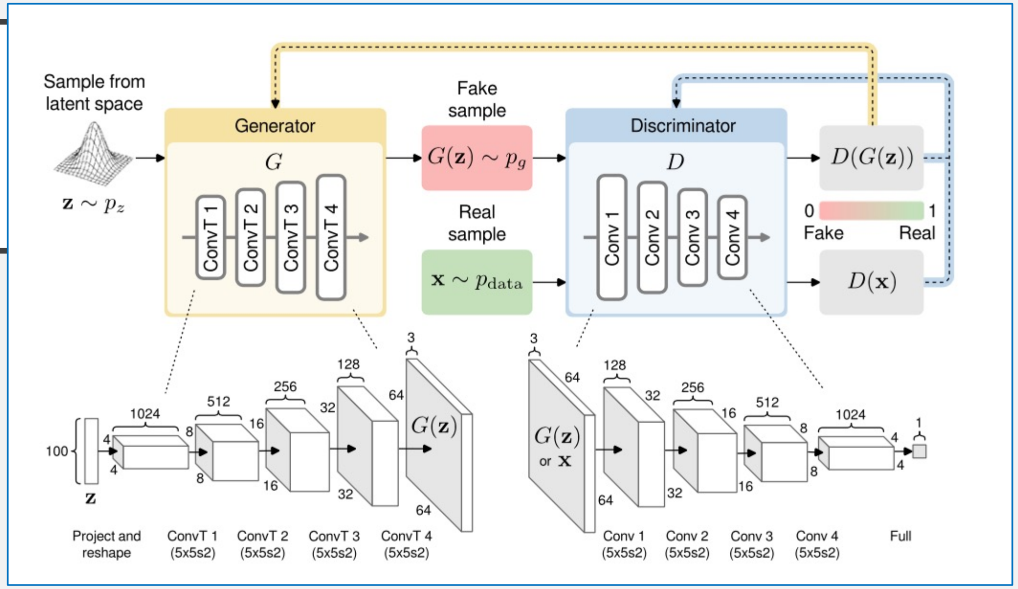
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	3,072
BatchNorm2d-2	[-1, 64, 32, 32]	128
LeakyReLU-3	[-1, 64, 32, 32]	0
Conv2d-4	[-1, 128, 16, 16]	131,072
BatchNorm2d-5	[-1, 128, 16, 16]	256
LeakyReLU-6	[-1, 128, 16, 16]	0
Conv2d-7	[-1, 256, 8, 8]	524,288
BatchNorm2d-8	[-1, 256, 8, 8]	512
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

DCGAN : TRAINING PROCESS

Train สลับกัน

```
# 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))
```

```python
# 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)
        # Calculate gradients for D in backward pass
        errD_real.backward()
        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)
        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()
```

- Training process
  - The discriminator receives samples x from a real data distribution p_{data} and fake inputs G(z) alternately, updating its parameters θ_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

$$loss = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i)$$

- 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)
        # Calculate G's loss based on this output
        errG = criterion(output, label)
        # Calculate gradients for G
        errG.backward()
        D_G_z2 = output.mean().item()
        # Update G
        optimizerG.step()
```

- Training process
  - The parameters θ_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., $p_g \sim p_{data}$.

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

ARCHITECTURE OF GENERATIVE MODEL : DCGAN

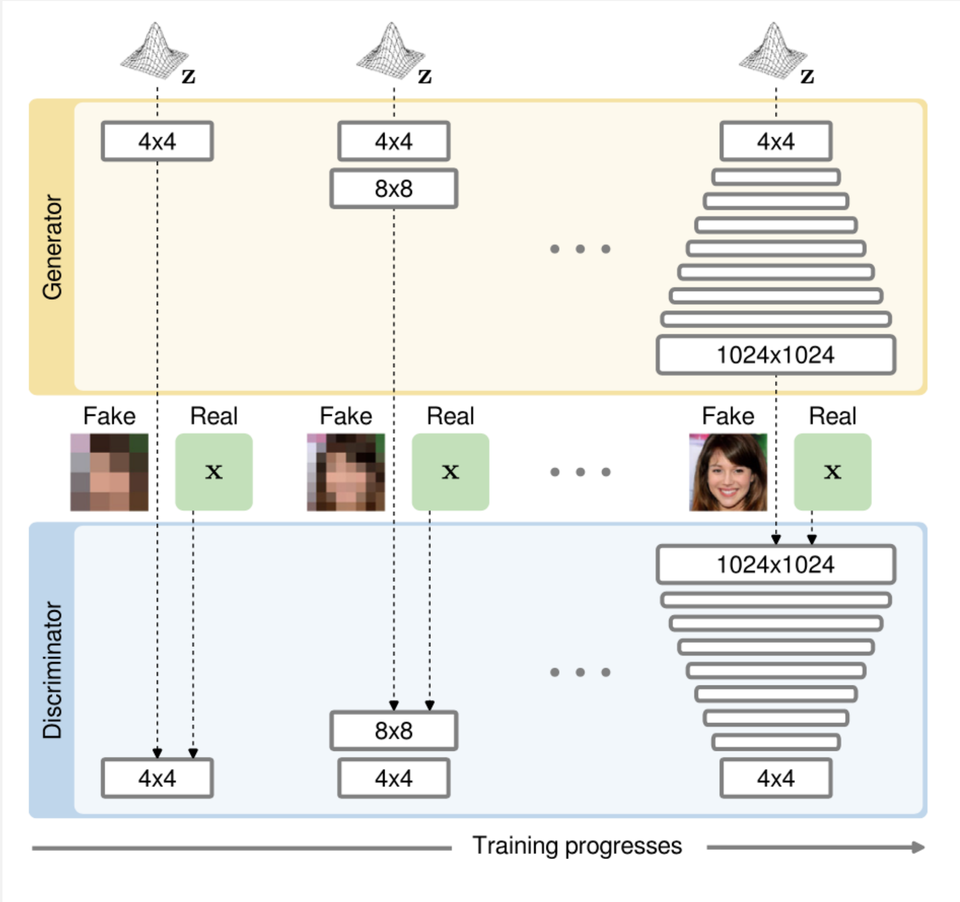
Real images

fake images

EVOLUTION OF GENERATIVE MODELS FOR IMAGES

train את low resolution קודם

- **Progressive Growing of GANs (ProGAN)**
- The network is trained on a low resolution setting (4 × 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 × 1024.

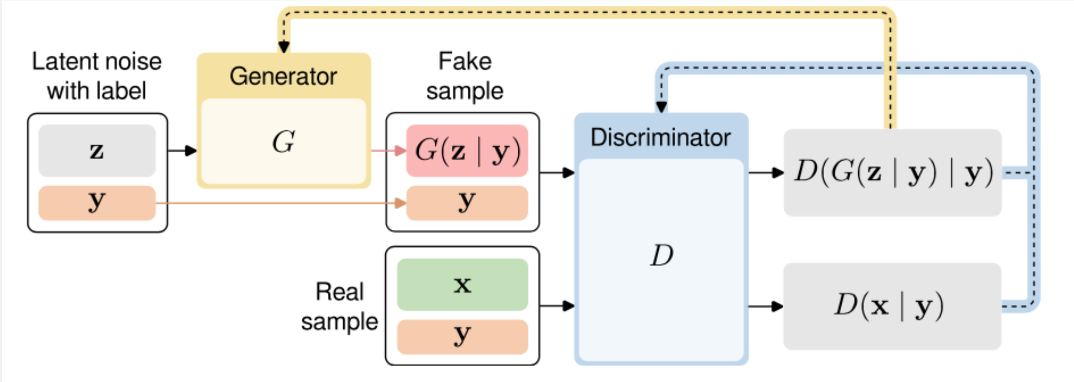


Trevisan de Souza V.L., Marques B.A.D., Batagelo H.C., Gois J.P., A review on Generative Adversarial Networks for image generation, Computers & Graphics (2023)

EVOLUTION OF GENERATIVE MODELS FOR IMAGES

คล้าย CVAE

- **CGAN architecture** Condition
- 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 beongs, other image, embedded textual vector.



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)

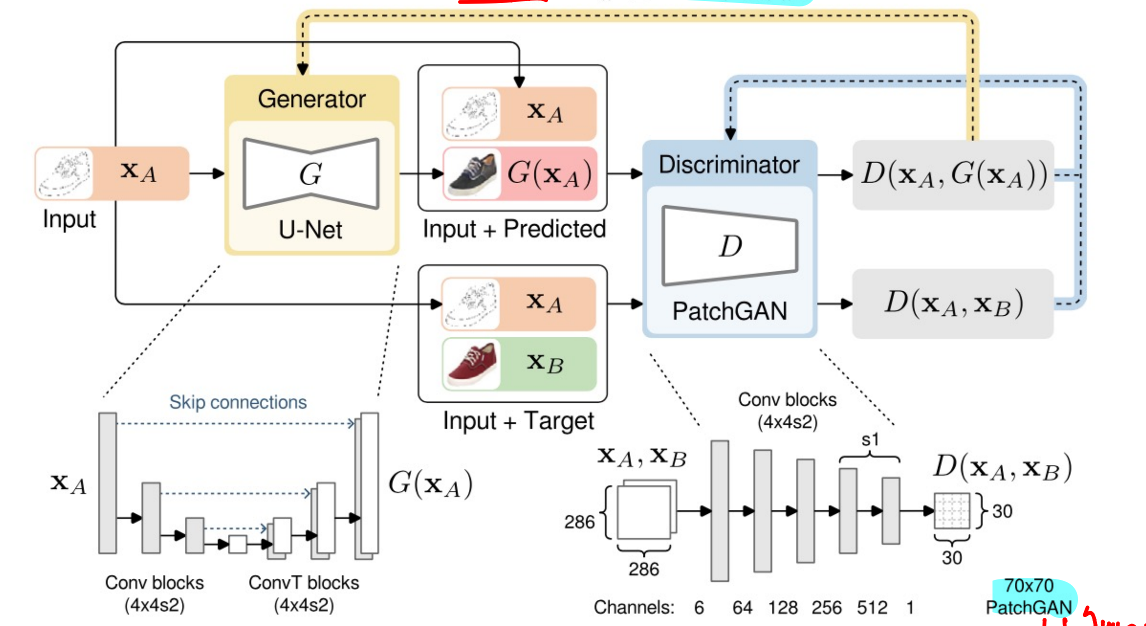
ARCHITECTURE OF GENERATIVE MODEL : PIX2PIX

- 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 archi- tectural 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.

error ของ GAN

ส่วนต่างของ ori & fake

$$\mathcal{L}_{\text{GAN}} + \lambda \|\mathbf{x}_B - G(\mathbf{x}_A)\|_1$$



จะบอกเลยว่าส่วนไหน real / fake

ได้ more specific error

Fig. 13. The pix2pix architecture is based on the conditional GAN (CGAN). The image $\mathbf{x}_A$ 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 in the dataset.

- 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 λ 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 <-> photo, trained on the Cityscapes dataset.
  - Map <-> aerial photo, trained on data scraped from Google Maps.
  - Black and White -> color photos.
  - Edges /. Sketch -> photo.
  - Day -> night photographs.
  - Thermal -> color photos.
  - Photo with missing pixels

  > inpainted photo, trained on Paris StreetView.

Urban environment

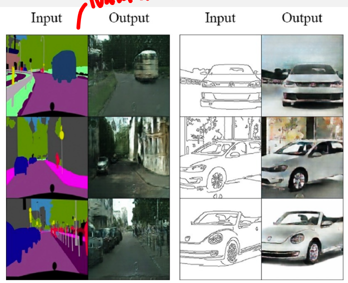


Fig. 12. Examples of images generated by applying pix2pix to synthesize street scenes from semantic maps (left) in the cityscapes dataset [84], and to create car images from edges (right). The input column presents the images X_A inserted into the generator, while the corresponding generated images $G(X_A)$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

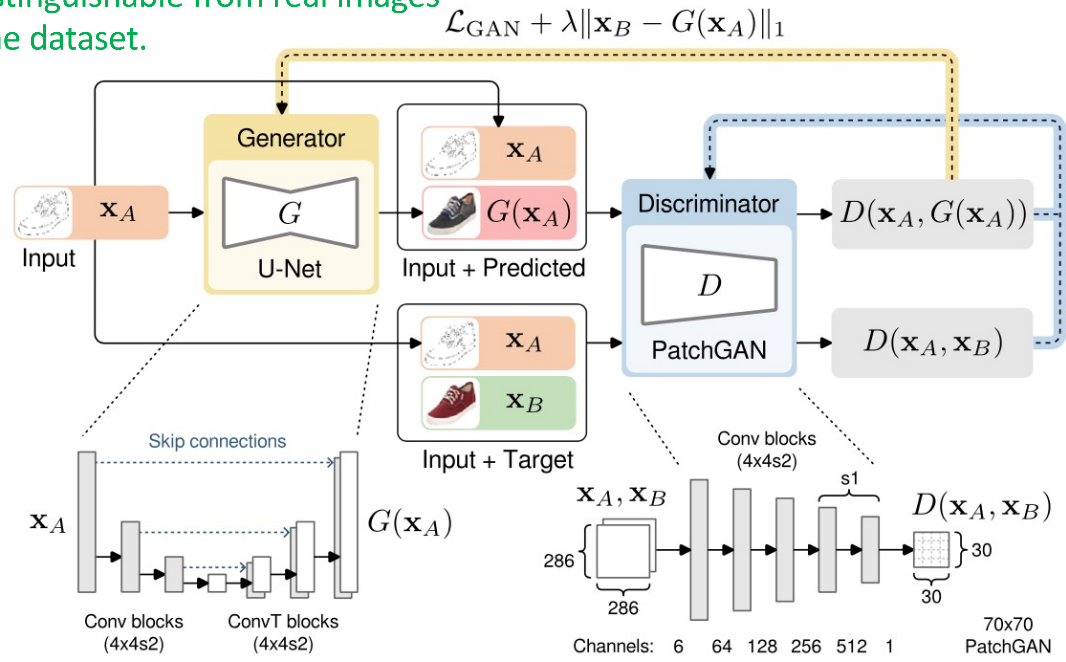


Fig. 13. The pix2pix architecture is based on the conditional GAN (CGAN). The image $\mathbf{x}_A$ serves as the condition for the generator and discriminator, and no noise input is used on the generator. Instead of a single scalar, the PatchGAN discriminator outputs a matrix, in which each element represents 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.
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.
StyleGAN2-ADA [21]	2020	Single-mode image synthesis, image editing.	Adaptive Discriminator Augmentation (ADA).
StyleGAN3 [22]	2021	Single-mode image synthesis, image editing.	Creation of translation- and rotation-equivariant layers. Remove progressive growing presented in previous StyleGANs.
SAGAN [23]	2019	Multi-modal image synthesis	Use of self-attention mechanism in the network.
BigGAN [24]	2019	Multi-modal image synthesis	Built from the SAGAN architecture. Improved stability and increased size.
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.
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.
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.
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.
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 **w** and the noise vector **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.
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.
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.
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.
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.
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.
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.
SRGAN [38]	2017	Super resolution	ResNet-based generator. Use of a perceptual loss, based on the activations of a VGG-19 pretrained network.
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.
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.
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.

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)

POPULAR GANS

Methods

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

  รูปที่ได้ไม่ค่อยมี diversity

  - struggle with capturing the full diversity of the data distribution

DIFFUSION MODELS : DDPM

iterative process

- DDPM is a parametric Markov chain that generates an image by progressively removing noise from the image
- 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.

100% noise

denoise

DIFFUSION MODELS : DDPM

train

- Two processes: the forward process and the reverse process

  ↳ generate

  - The forward process is the gradual addition of Gaussian noise to the image data until it becomes random noise
  - The reverse process is a denoising procedure that generates samples from the data distribution by iteratively removing noise step by step

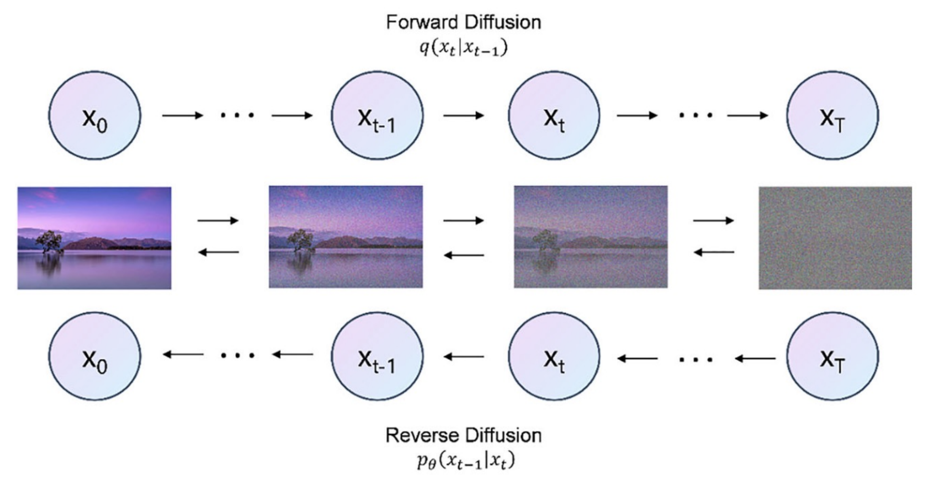


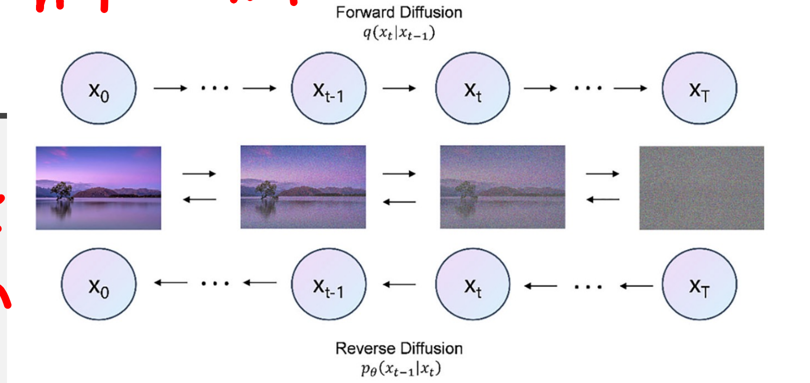
Fig. 4 Diffusion models comprise a forward process and a reverse process. The forward process incrementally adds noise to the original image, enabling the model to learn the noise distribution. Conversely, the reverse process generates images by denoising the noise-added images

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

$X_n = X_{n-1} + \varepsilon_{n-1}$ so $X_{n-1} = X_n - \varepsilon_{n-1}$

use predict



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

Unet ใช้ รูปที่มี noise มา predict ε ที่ทำให้เกิด รูปนี้

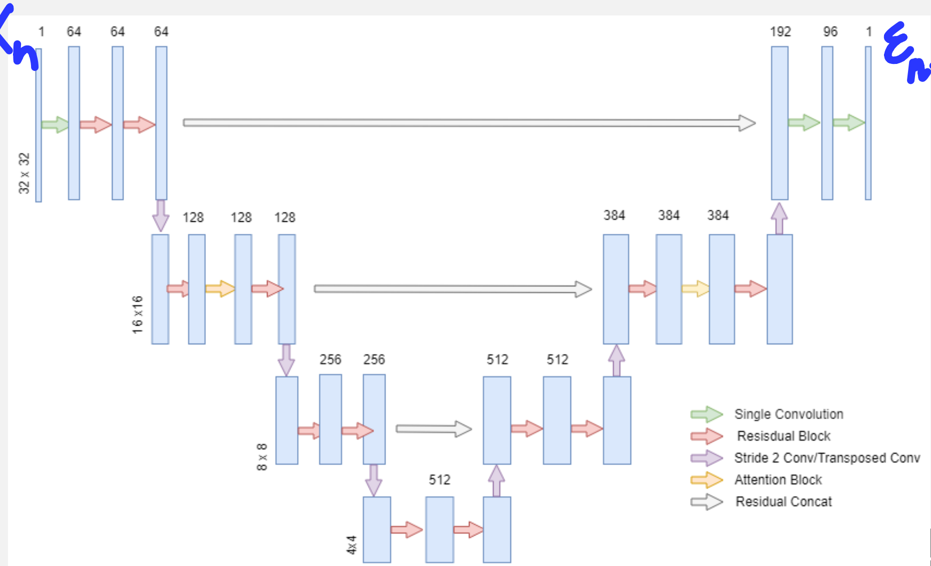
The forward process (Training):

1. The original image ($\mathbf{x_0}$) is slowly corrupted iteratively (a Markov chain) by adding (scaled Gaussian) noise.
2. This process is done for some **T** time steps, i.e., $\mathbf{x_T}$.
3. Image at timestep **t** is created by: $\mathbf{x_{t-1} + \varepsilon_{t-1}}$ **(noise)** → $\mathbf{x_t}$
4. The model, such as a U-Net or neural network, takes the noisy image $\mathbf{x_t}$ and the timestep **t** as inputs and outputs a prediction for the noise **έ**.

   ทำตอน train , $\varepsilon_{n-1} - \hat{\varepsilon}_{n-1}$: predict

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.

X_n



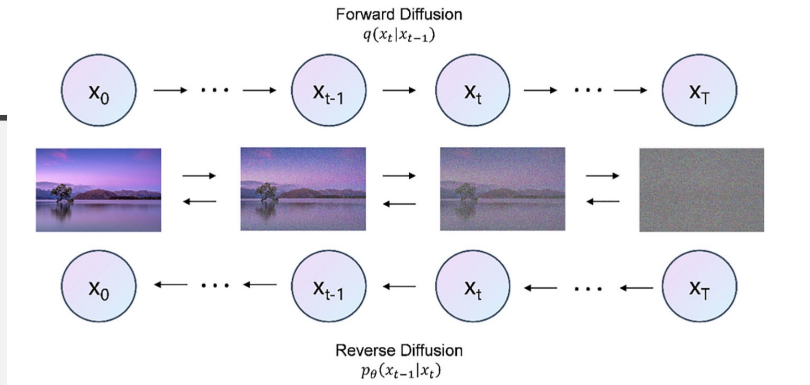
ε_{n-1}

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

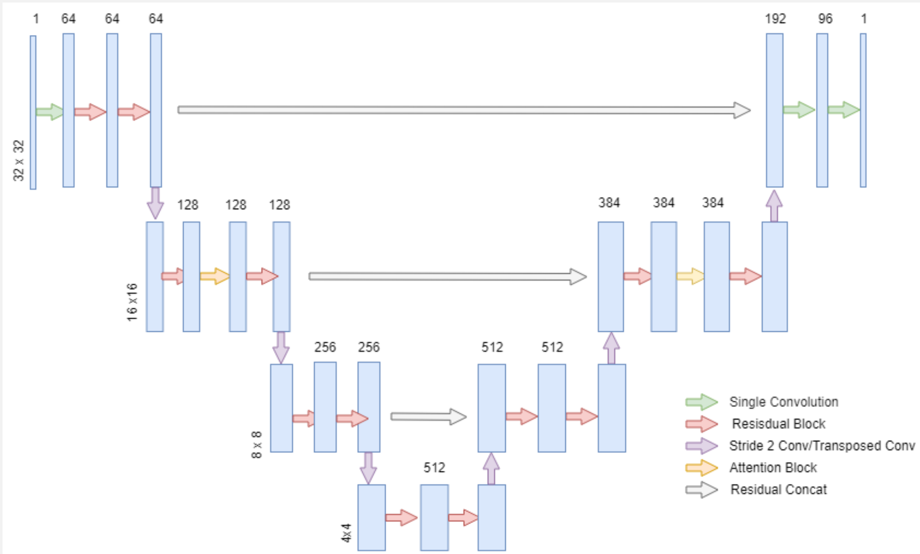
DIFFUSION MODELS : DDPM

The Reverse process (Inference):

- Initialize with Pure Noise, then Iterative Denoising from each time step T, T-1, .. to 0. progressively denoise $\mathbf{x}_t$ to generate a less noisy version $\mathbf{x}_{t-1}$. The goal is to gradually remove noise from $\mathbf{x}_t$ until you reach $\mathbf{x}_0$



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/1TrycJ3iy4RtD8LDO0aJSlhQl7-6xDI1-?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.

เพิ่ม noise ในระดับ latent

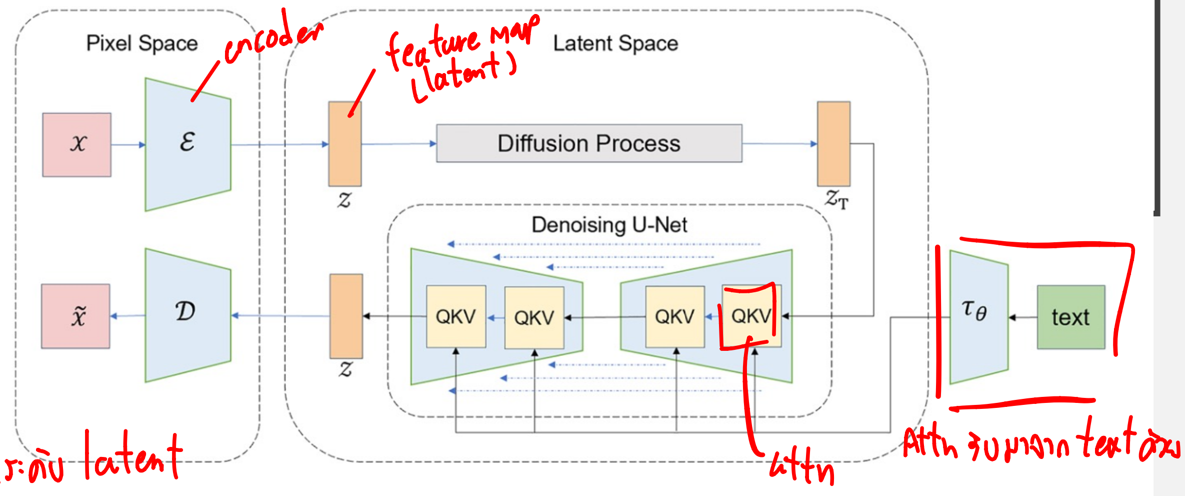


Fig. 8 In the architecture of Latent Diffusion Models, images are outputted to a latent space through an encoder, where the diffusion process takes place. The reverse process employs a U-Net architecture. Text prompts are injected into the self-attention modules of the U-Net via an encoder

  - 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 τ_θ 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

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

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,

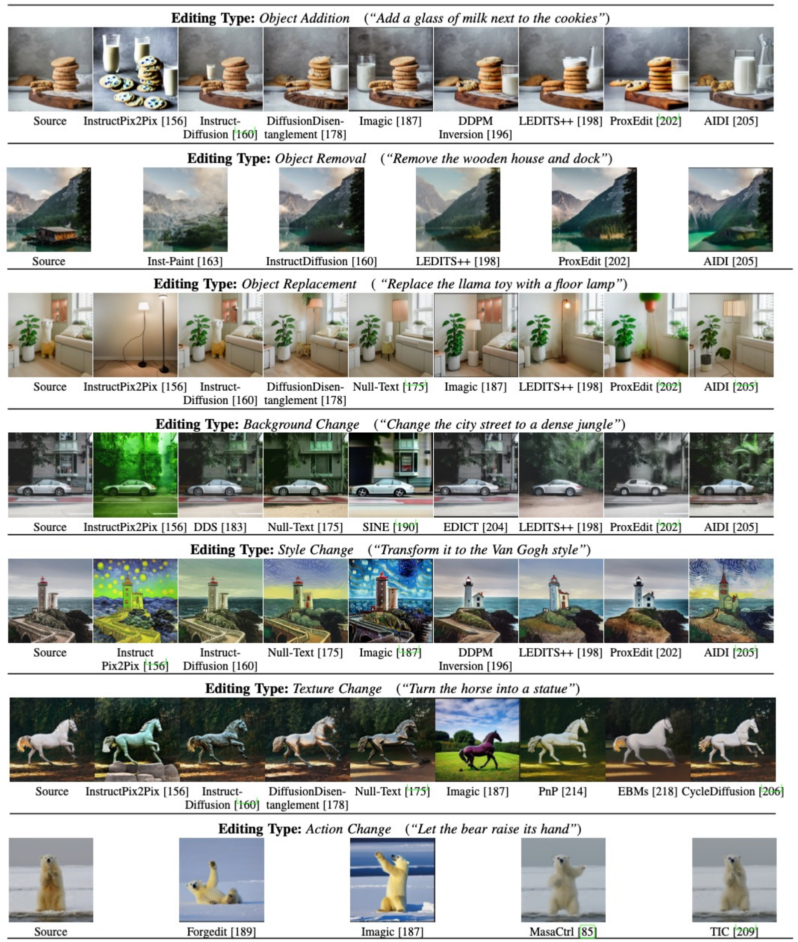


Fig. 13: Visual comparison on 7 selected editing types.

FUTURE DIRECTIONS

Duffusion ปัจจุบันนานมากๆ

- Fewer-step Model Inference : few-step or one- step generation diffusion models
- Efficient Models : backbone, top-layers
- Complex Lighting and Shadow Editing
- Unrobustness of Image Editing : not capable of accurately modeling all possible samples in the conditional distribution space
- Faithful Evaluation Metrics : PSNR, and SSIM used for a reference, mid-level similarity of two images

might be better

MSE

ดู semantic มากกว่า
not pixel-wise

เพราะมันรอบคลุมเหมือน 100%
ไม่ variety พอ

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.
- 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.
- 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.
- 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.

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.