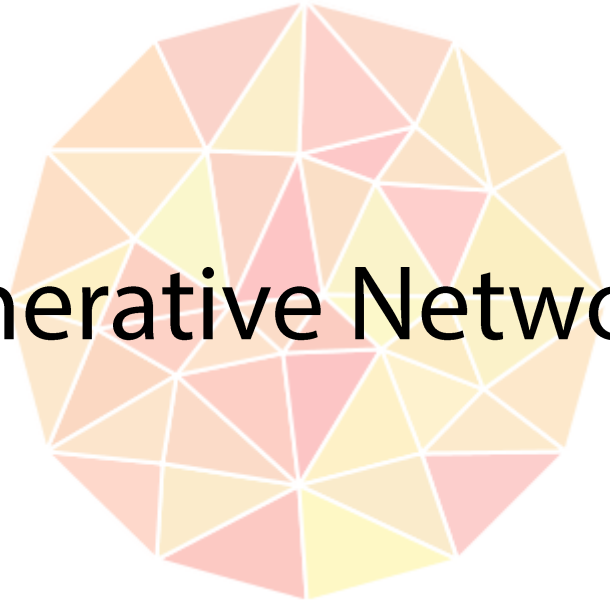
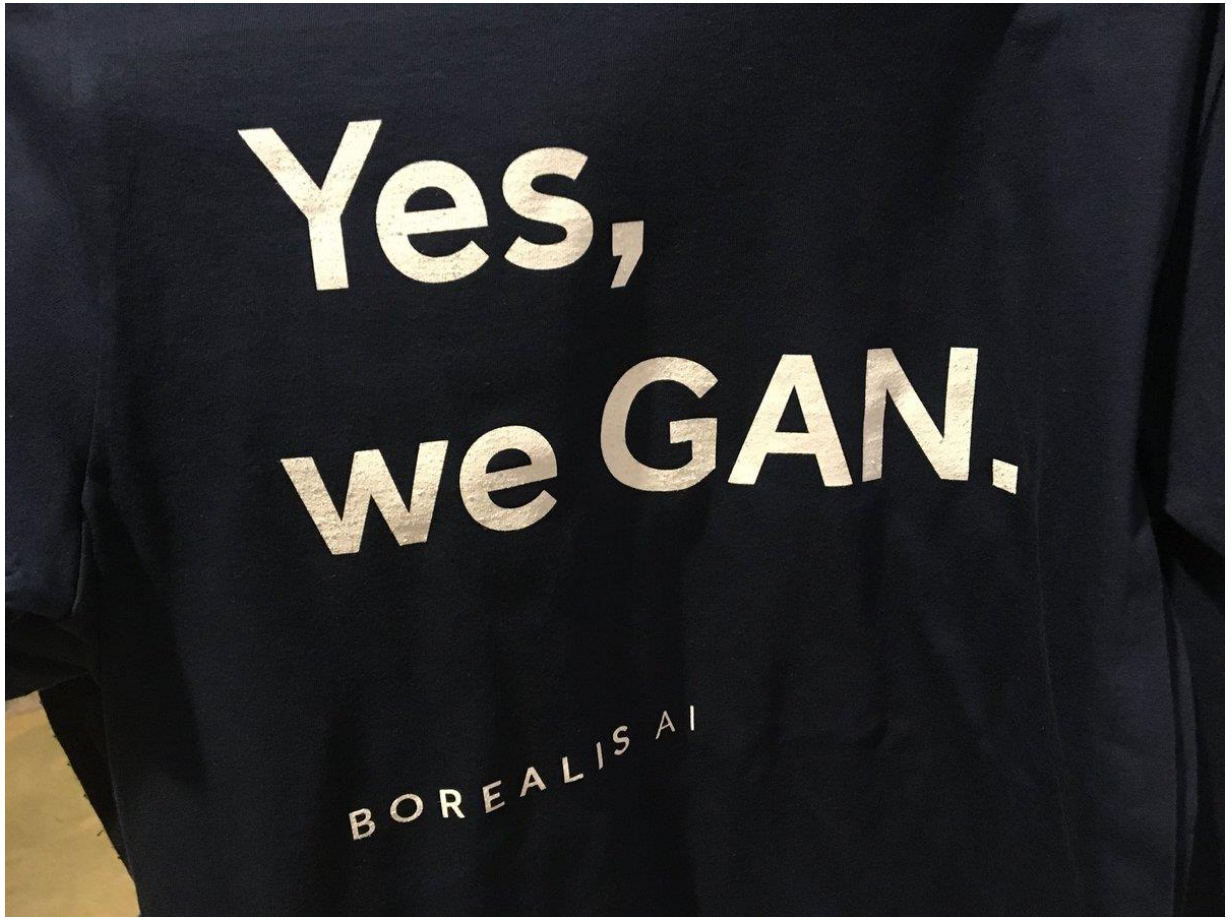


Generative Networks

Stephen Baek

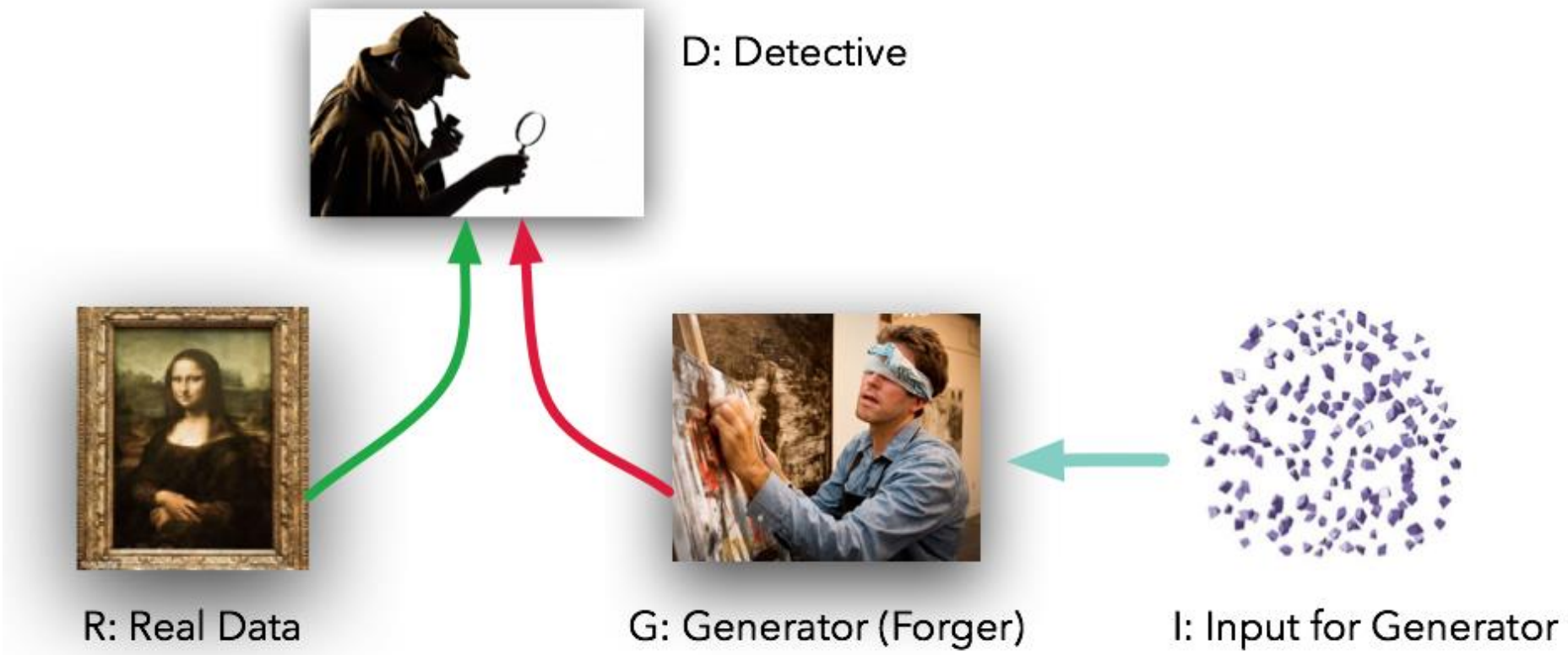
Generative Adversarial Networks (GAN)



Generative Networks

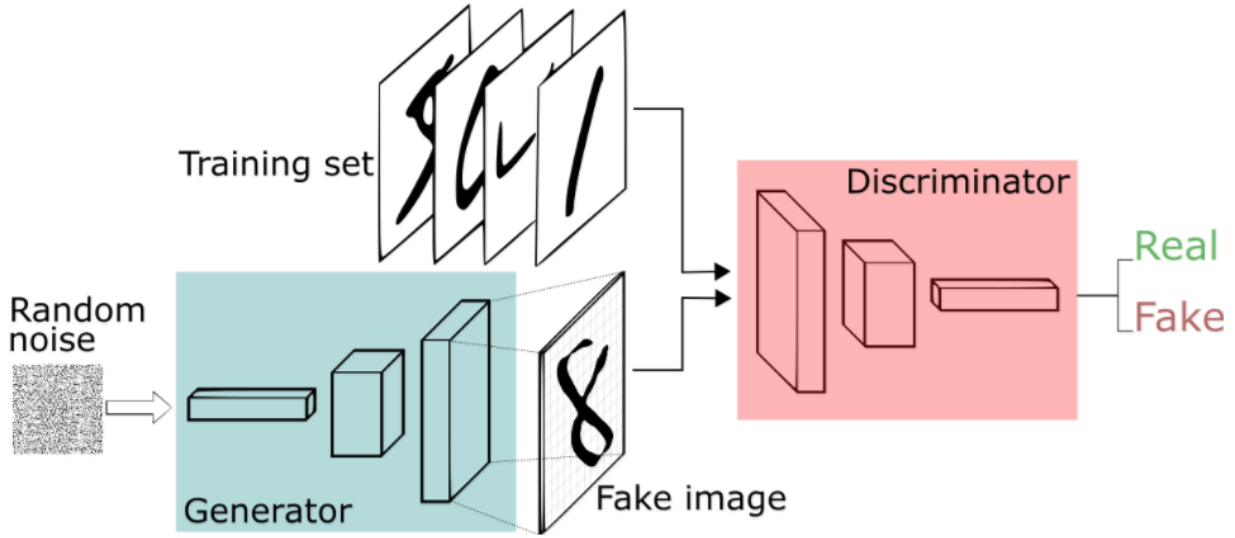
https://www.thispersondoesnotexist.com/

Generative Adversarial Networks



Generative Adversarial Networks

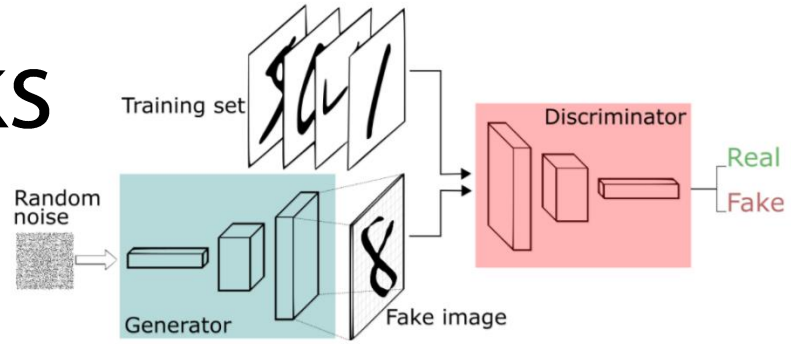
- The Minimax Game
  - Generator network: try to fool the discriminator by generating real-looking images
  - Discriminator network: try to distinguish between real and fake images



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

Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generative Adversarial Networks



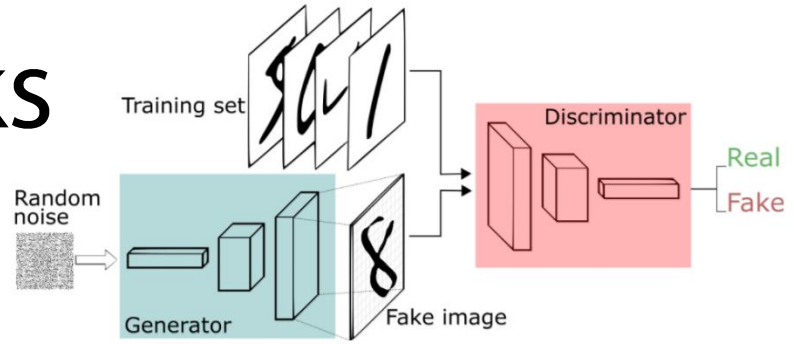
- The Minimax Game

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

  - Discriminator (θ_d) wants to maximize the objective function
    - → $D(x)$ (real) is close to 1 and $D(G(z))$ (fake) is close to 0
  - Generator (θ_g) wants to minimize the objective function
    - → $D(G(z))$ is close to 1 (i.e. discriminator is fooled into thinking $G(z)$ is real)

Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generative Adversarial Networks



- The Minimax Game

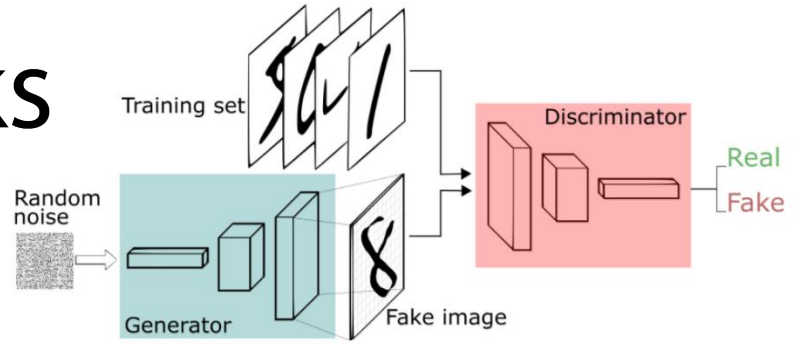
$D: I \rightarrow (0,1)$

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output on real data x Discriminator output on fake data $G(z)$

Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generative Adversarial Networks



- The Minimax Game

$D: I \rightarrow (0,1)$

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output on real data x Discriminator output on fake data $G(z)$

  - Discriminator (θ_d) wants to maximize the objective function
    - → $D(x)$ (real) is close to 1 and $D(G(z))$ (fake) is close to 0
  - Generator (θ_g) wants to minimize the objective function
    - → $D(G(z))$ is close to 1 (i.e. discriminator is fooled into thinking $G(z)$ is real)

Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Training GANs

- Training GAN is quite tricky…
  - There are several well-known fixes!

Training GANs

- For training…

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

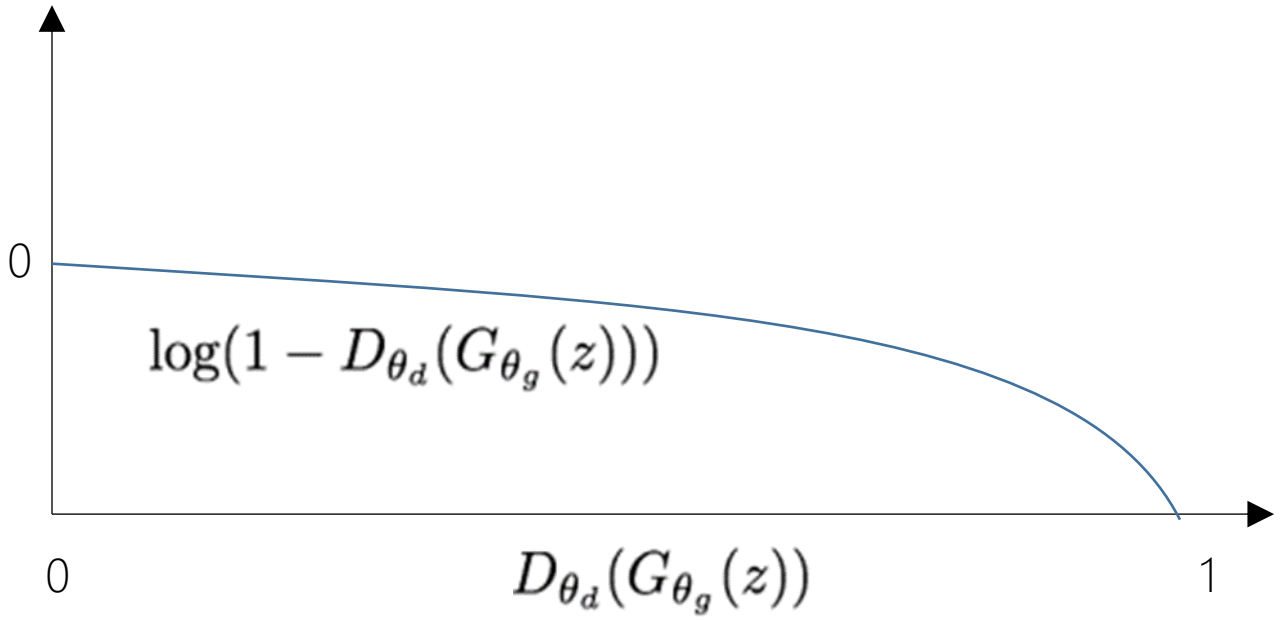
Alternate between

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs

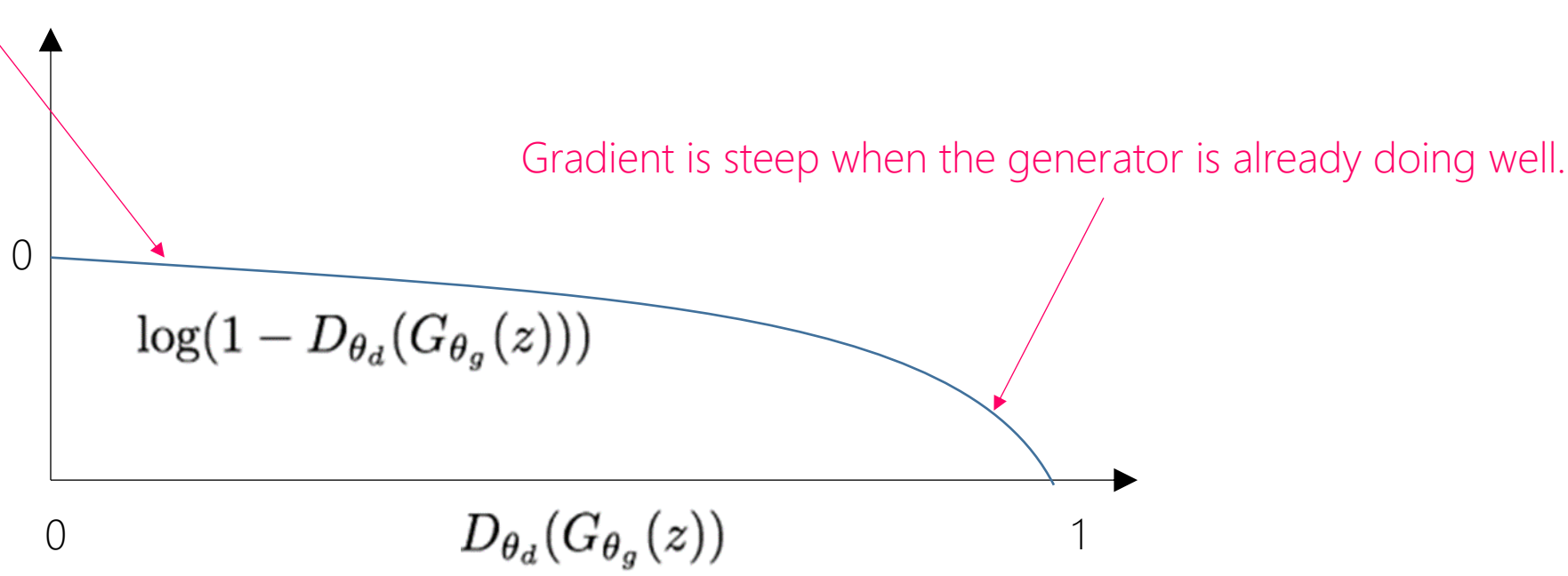
- Problem with the generator loss: $\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$



Training GANs

- Problem with the generator loss: $\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$

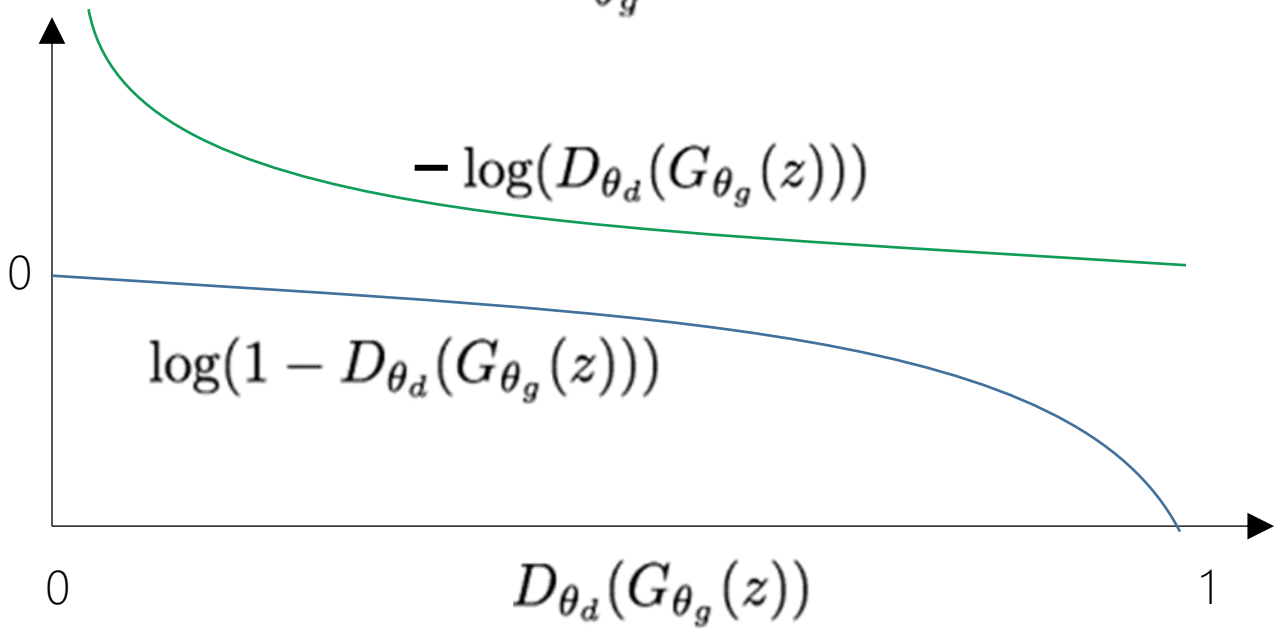
Gradient is flat when the generator is not doing well.
i.e. wants to learn something from mistakes, but actually can't!



Training GANs

- Problem with the generator loss: $\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$



Training GANs

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs

K is a magic number. Some say k=1 is better, while others find k>1 more stable.

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs

K is a magic number. Some say k=1 is better, while others find k>1 more stable.

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

Train discriminator with separate batches of real and fake images?

  end for

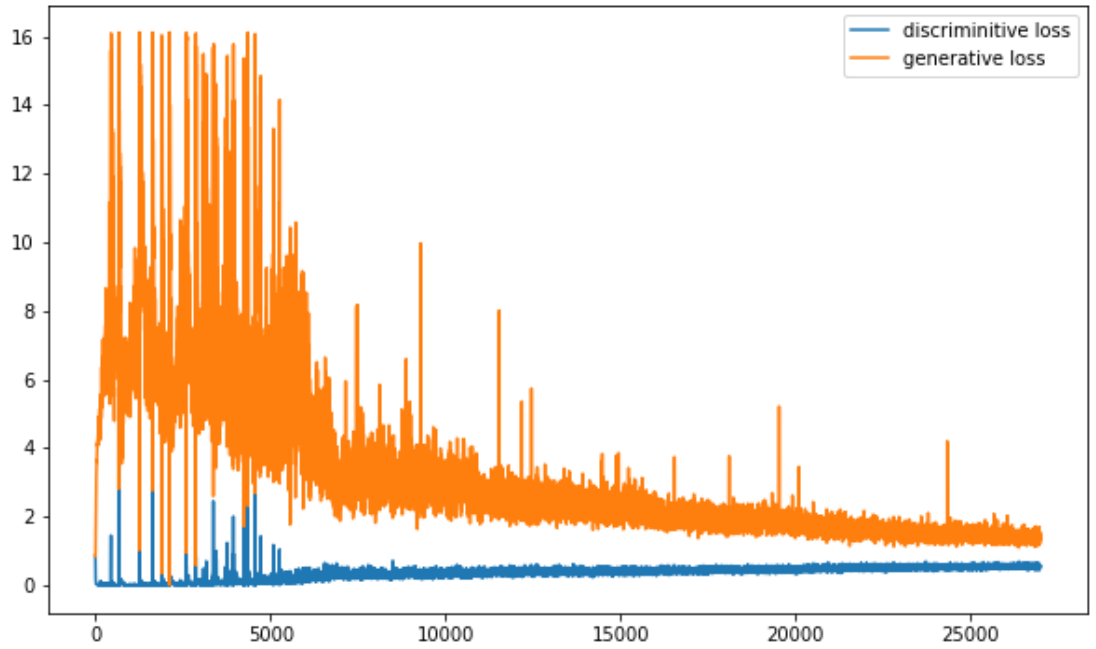
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

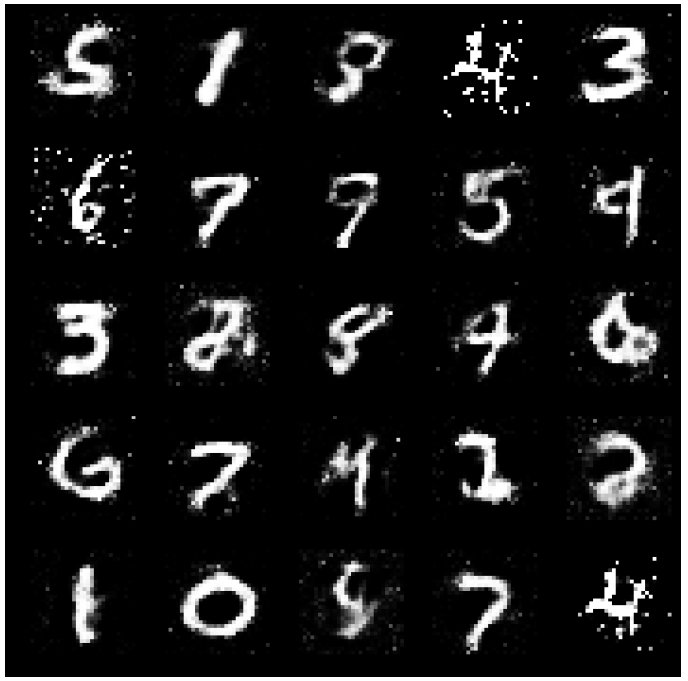
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training discriminator with separate batches of real and fake images

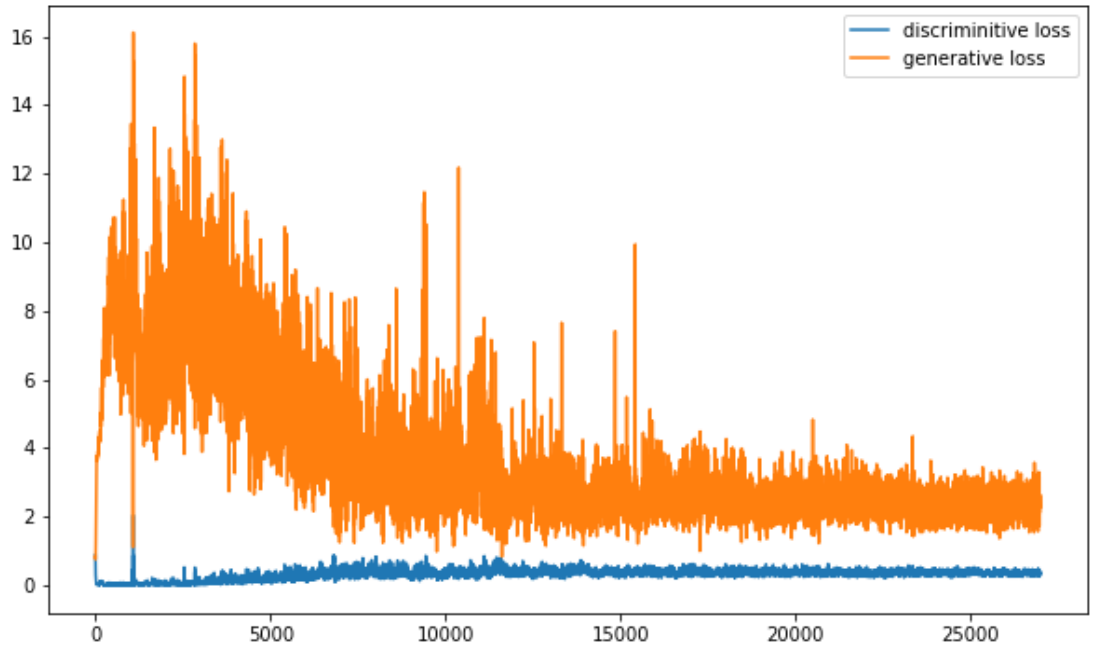
```
29990 [D loss: 0.329321, acc.: 86.72%] [G loss: 2.435551]
29991 [D loss: 0.307467, acc.: 86.33%] [G loss: 2.741055]
29992 [D loss: 0.294472, acc.: 87.11%] [G loss: 2.858673]
29993 [D loss: 0.338616, acc.: 83.98%] [G loss: 2.496999]
29994 [D loss: 0.281529, acc.: 89.45%] [G loss: 2.361624]
29995 [D loss: 0.322767, acc.: 85.55%] [G loss: 2.291273]
29996 [D loss: 0.342422, acc.: 88.28%] [G loss: 2.719167]
29997 [D loss: 0.301326, acc.: 88.28%] [G loss: 2.743211]
29998 [D loss: 0.376825, acc.: 83.20%] [G loss: 2.354978]
29999 [D loss: 0.291810, acc.: 88.28%] [G loss: 2.229078]
```

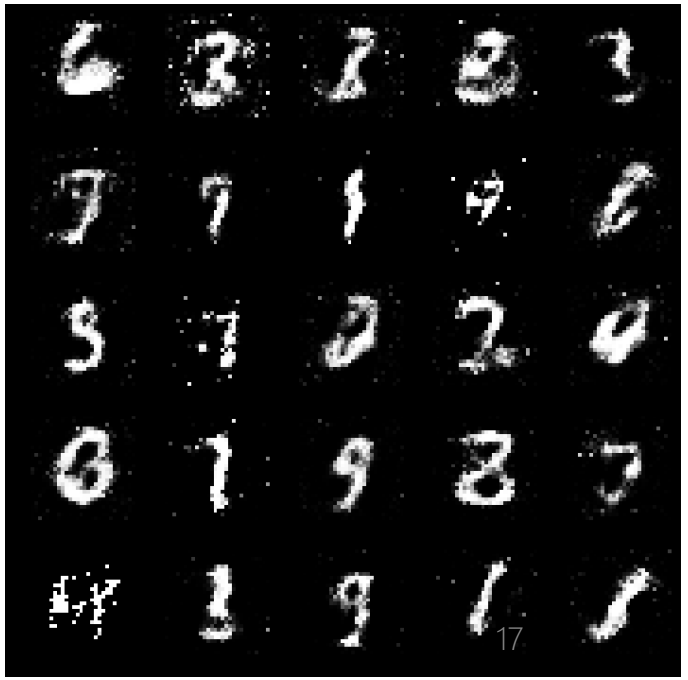




Training discriminator with real and fake images mixed up altogether

```
29990 [D loss: 0.591177, acc.: 74.61%] [G loss: 1.205729]
29991 [D loss: 0.546334, acc.: 74.22%] [G loss: 1.252525]
29992 [D loss: 0.561110, acc.: 71.88%] [G loss: 1.312339]
29993 [D loss: 0.628200, acc.: 65.62%] [G loss: 1.313918]
29994 [D loss: 0.618396, acc.: 65.23%] [G loss: 1.368492]
29995 [D loss: 0.543587, acc.: 72.66%] [G loss: 1.349365]
29996 [D loss: 0.565871, acc.: 70.70%] [G loss: 1.291717]
29997 [D loss: 0.545177, acc.: 74.22%] [G loss: 1.385682]
29998 [D loss: 0.474738, acc.: 78.52%] [G loss: 1.283363]
29999 [D loss: 0.622424, acc.: 66.02%] [G loss: 1.252099]
```





GAN in keras

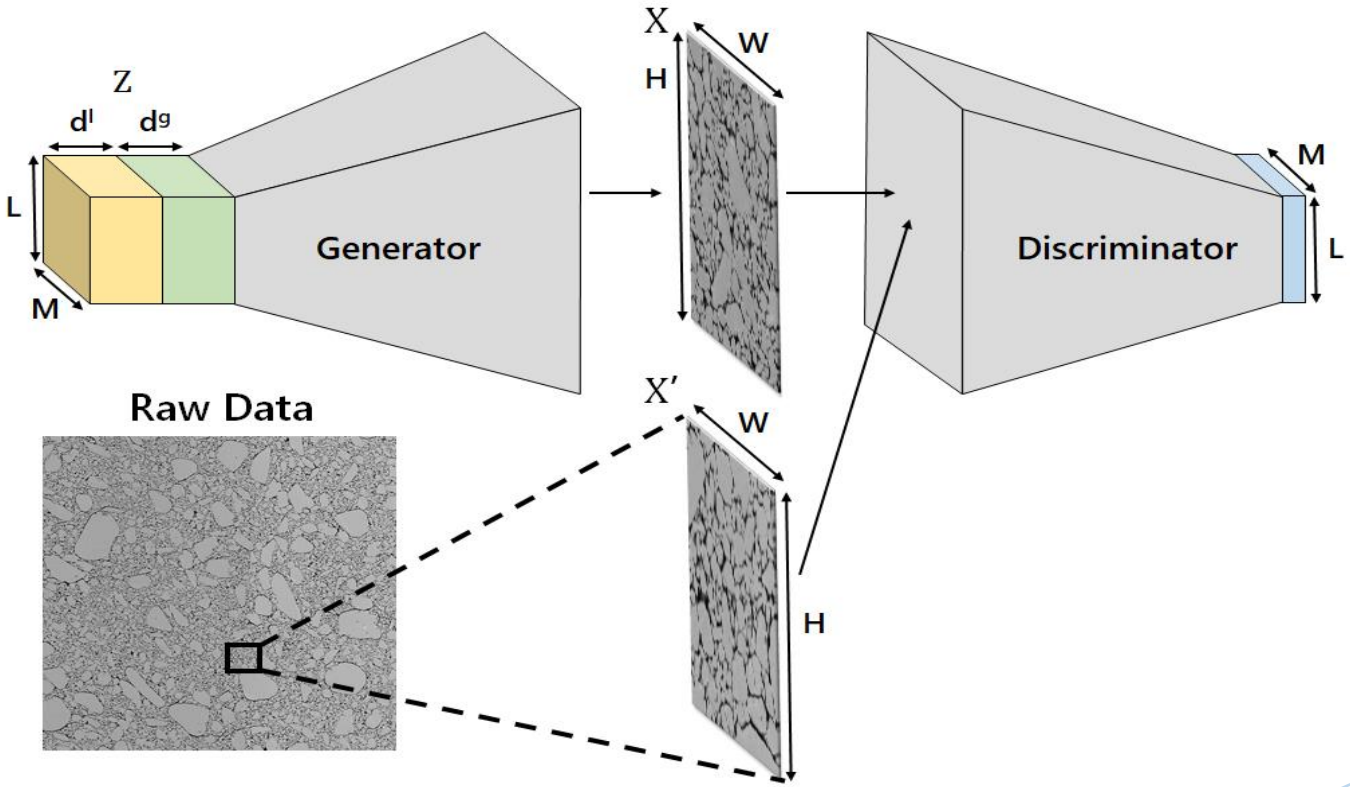
```
model = Sequential()

model.add(Dense(256, input_dim = self.latent_dim))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(1024))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
# linear embedded to output image shape
model.add(Dense(np.prod(self.img_shape),activation= 'tanh'))
model.add(Reshape(self.img_shape))

model.summary()

# Random noise as input and generate fake img
noise = Input(shape=(self.latent_dim,))
f_img = model(noise)

Generator = Model(noise, f_img)
```

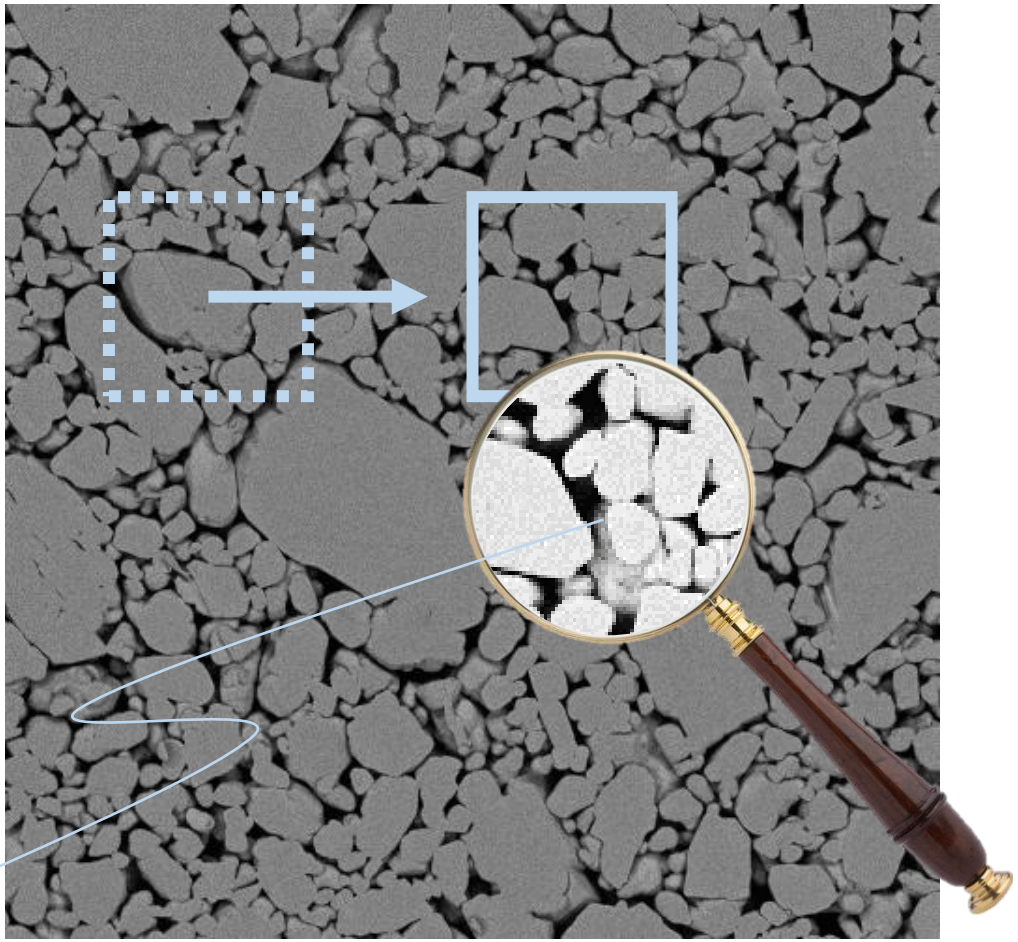
```
model = Sequential()
model.add(Flatten(input_shape = self.img_shape))
model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2))
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.2))
model.add(Dense(1,activation = 'sigmoid'))

model.summary()

# take img as input, output 0 or 1
img = Input(shape = self.img_shape)
validity = model(img)

Discriminator = Model(img, validity)
```

GAN in keras

```
for step in range(steps):
    # Make generative images
    idx = np.random.randint(0, X_train.shape[0], BATCH_SIZE)
    true_imgs = X_train[idx]
    noise_gen = np.random.normal(0,1,(BATCH_SIZE, self.latent_dim))
    fake_imgs = self.generator.predict(noise_gen)
    # Make labels
    valid = np.ones((BATCH_SIZE,1))
    fake = np.zeros((BATCH_SIZE,1))
    #---------------------------------
    # Train discriminator on generated images
    #---------------------------------
    d_loss_real = self.discriminator.train_on_batch(true_imgs, valid)
    d_loss_fake = self.discriminator.train_on_batch(fake_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
     #X = np.concatenate((true_imgs, fake_imgs))
     #Y = np.concatenate((valid,fake))
     #d_loss  = self.discriminator.train_on_batch(X,Y)
    #-------------------------------------------------------------------------
    # Train GAN: --train the generator (to have the discriminator label samples as valid)
    #-------------------------------------------------------------------------
    noise_tr = np.random.normal(0,1,(BATCH_SIZE, self.latent_dim))
    # Freeze the discriminator
    self.discriminator.trainable = False
    g_loss = self.GAN.train_on_batch(noise_tr, valid)

    # Unfreeze the discriminator
    self.discriminator.trainable = True
```

Microstructure GAN

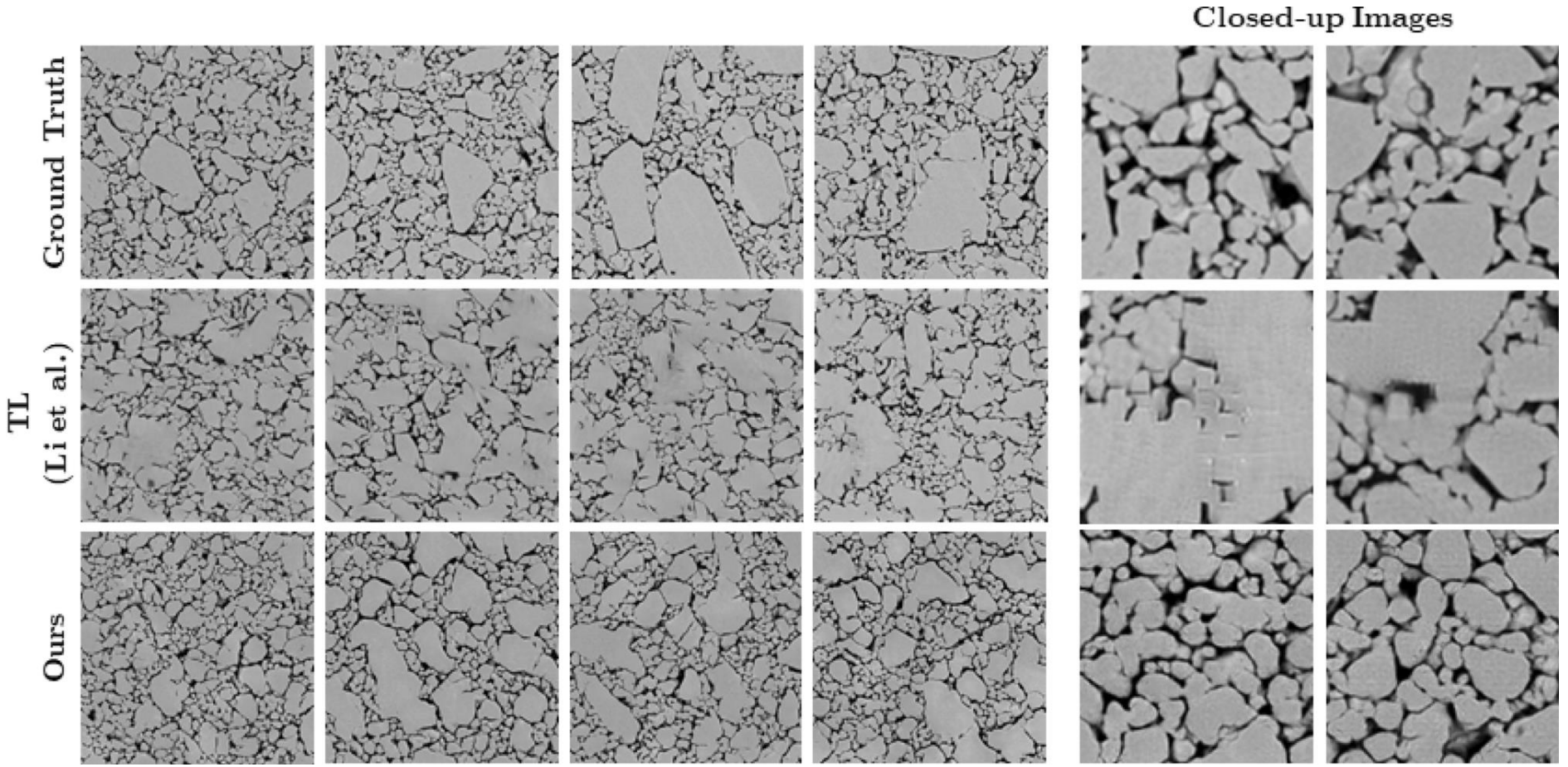




$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

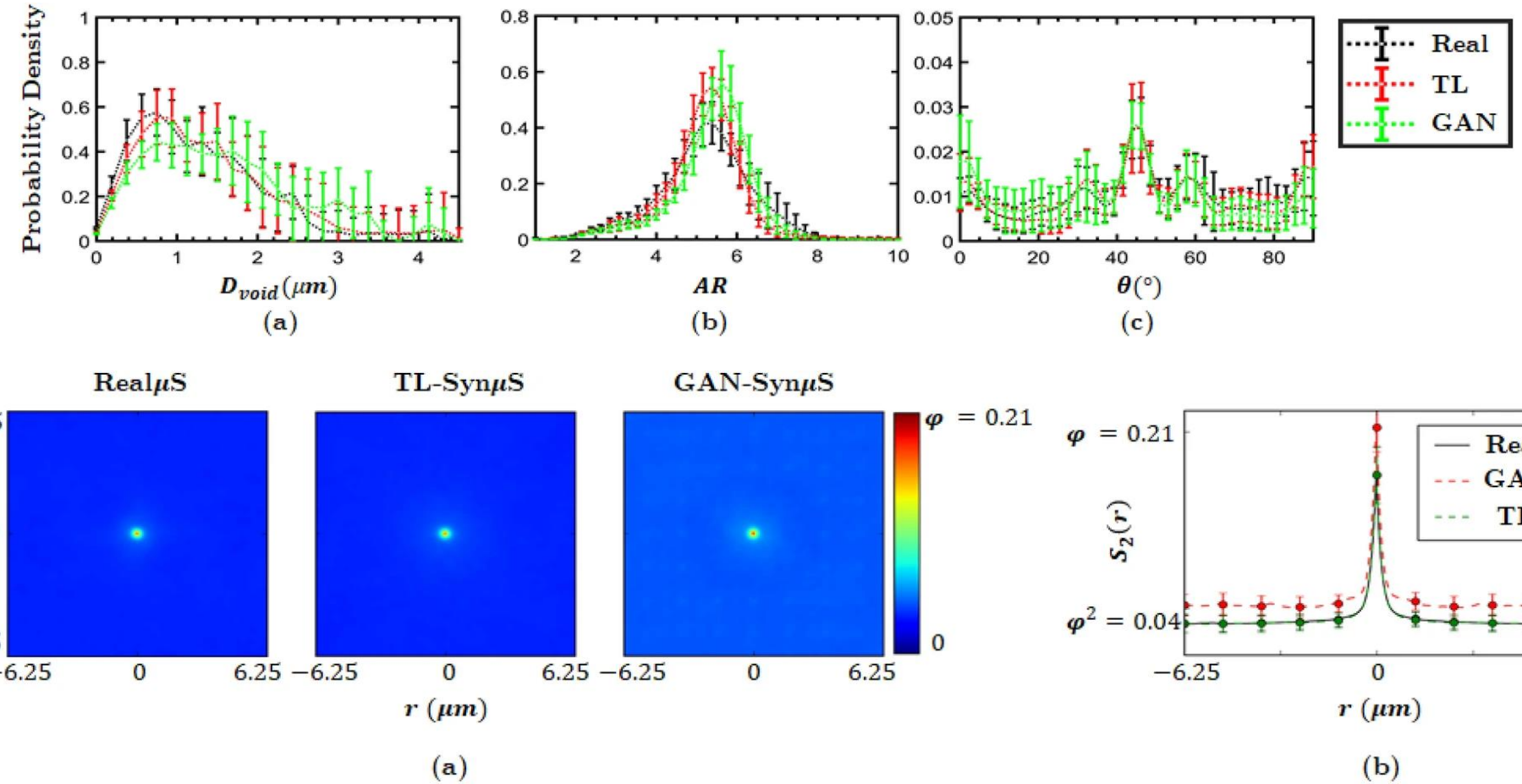
Chun et al. (2020) https://www.nature.com/articles/s41598-020-70149-0

Microstructure GAN



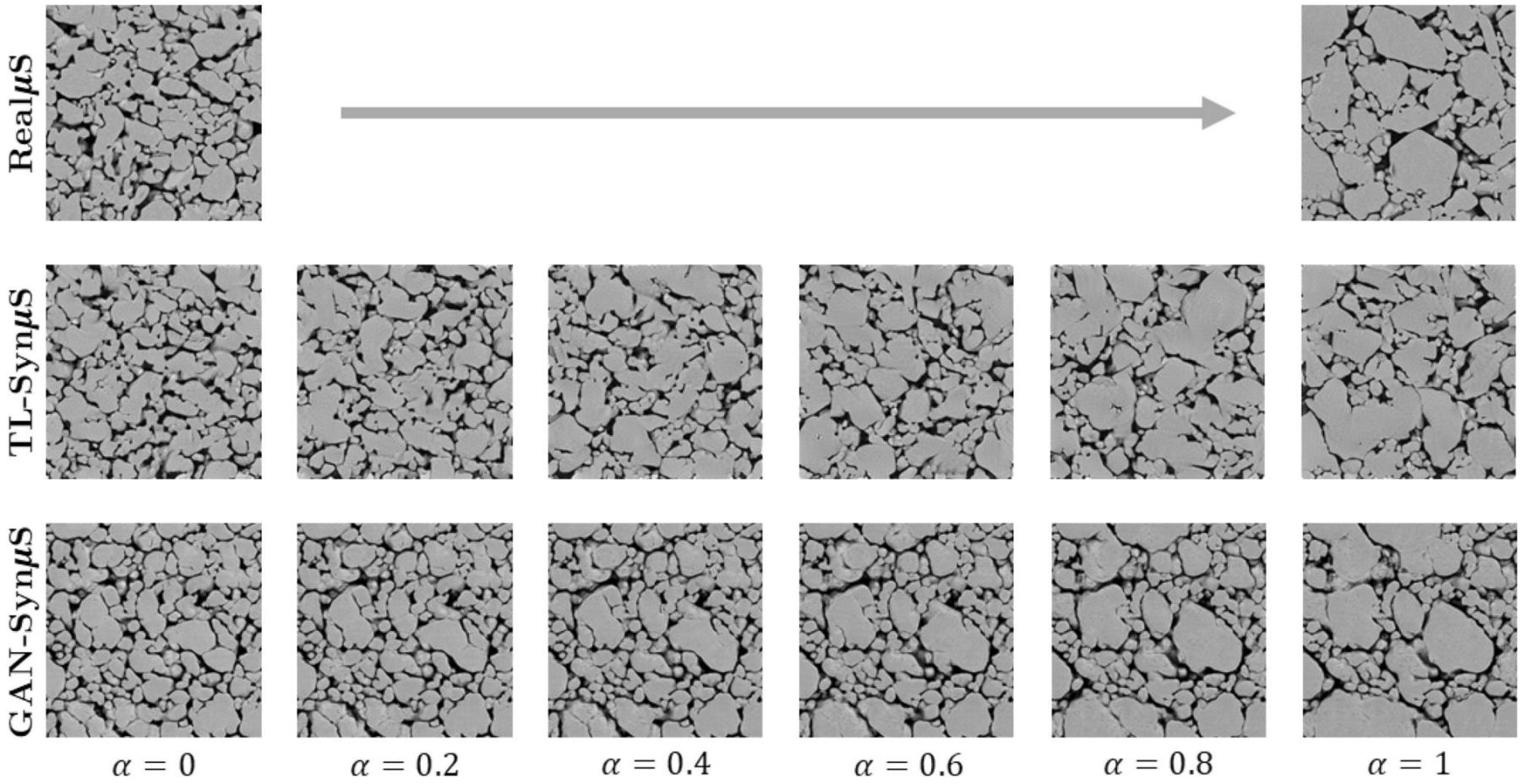
Chun et al. (2020) https://www.nature.com/articles/s41598-020-70149-0

Microstructure GAN

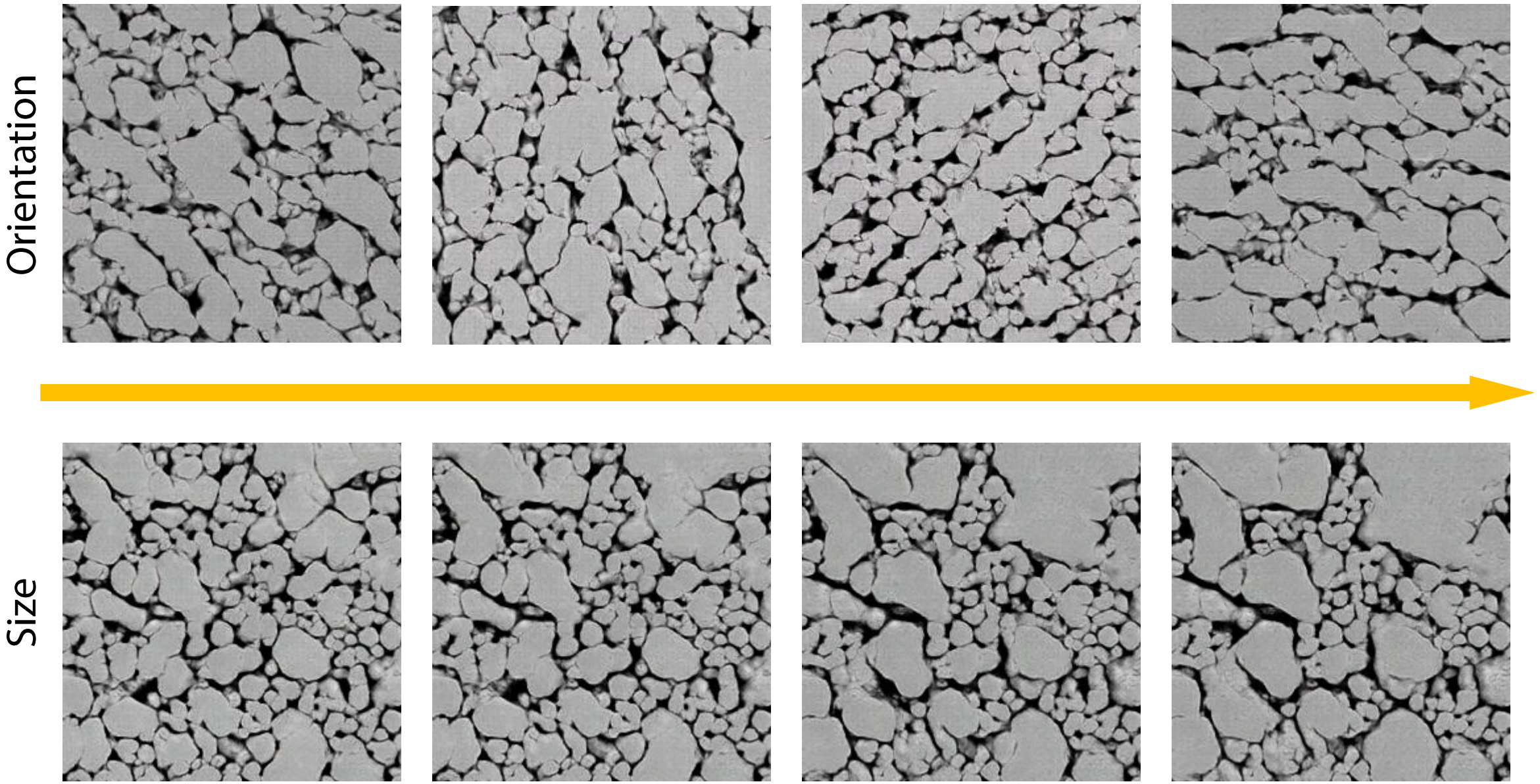


Chun et al. (2020) https://www.nature.com/articles/s41598-020-70149-0

Microstructure Interpolation

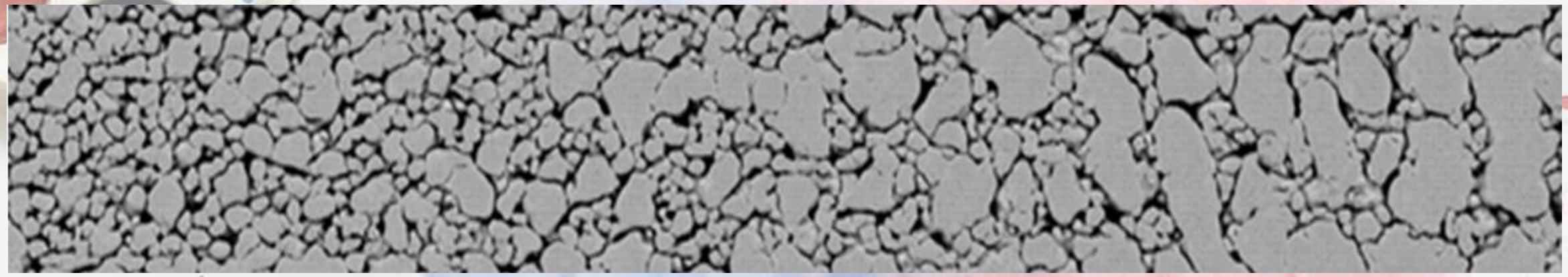


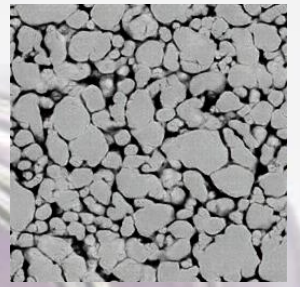
Chun et al. (2020) https://www.nature.com/articles/s41598-020-70149-0



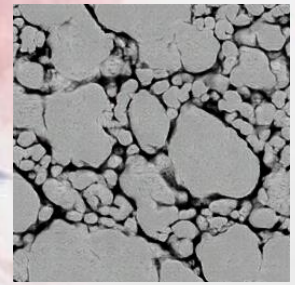
Chun et al. (2020) https://www.nature.com/articles/s41598-020-70149-0

Microstructure “Painting”



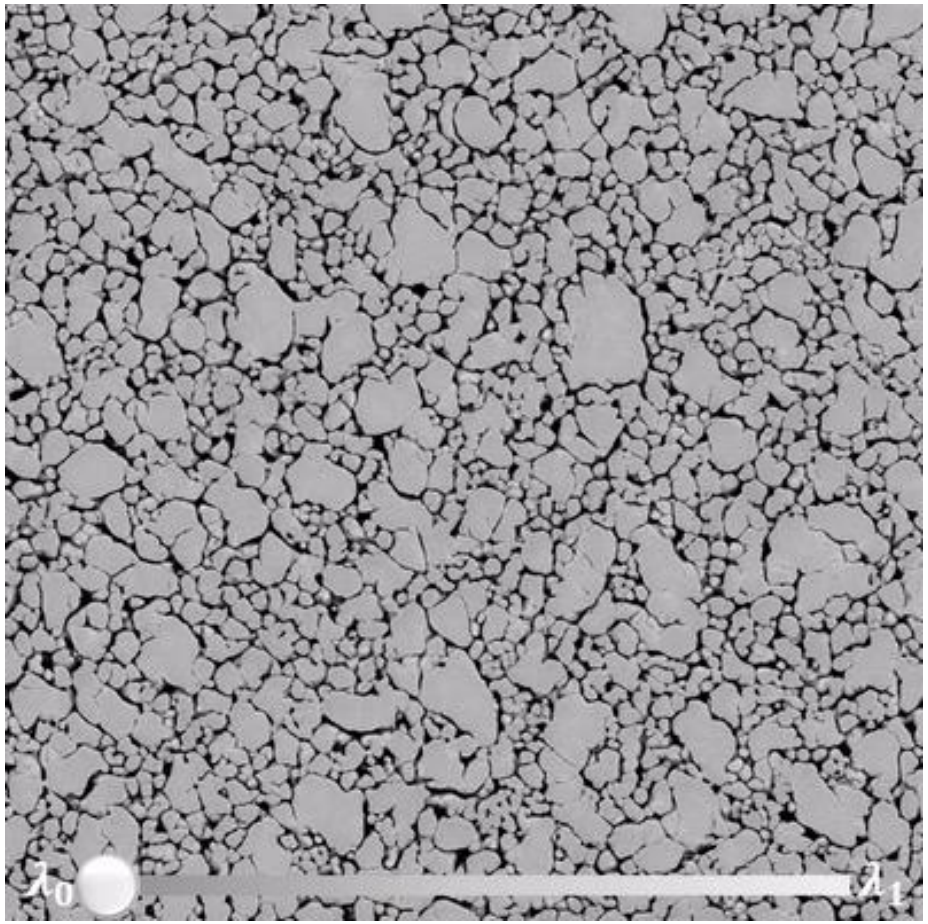


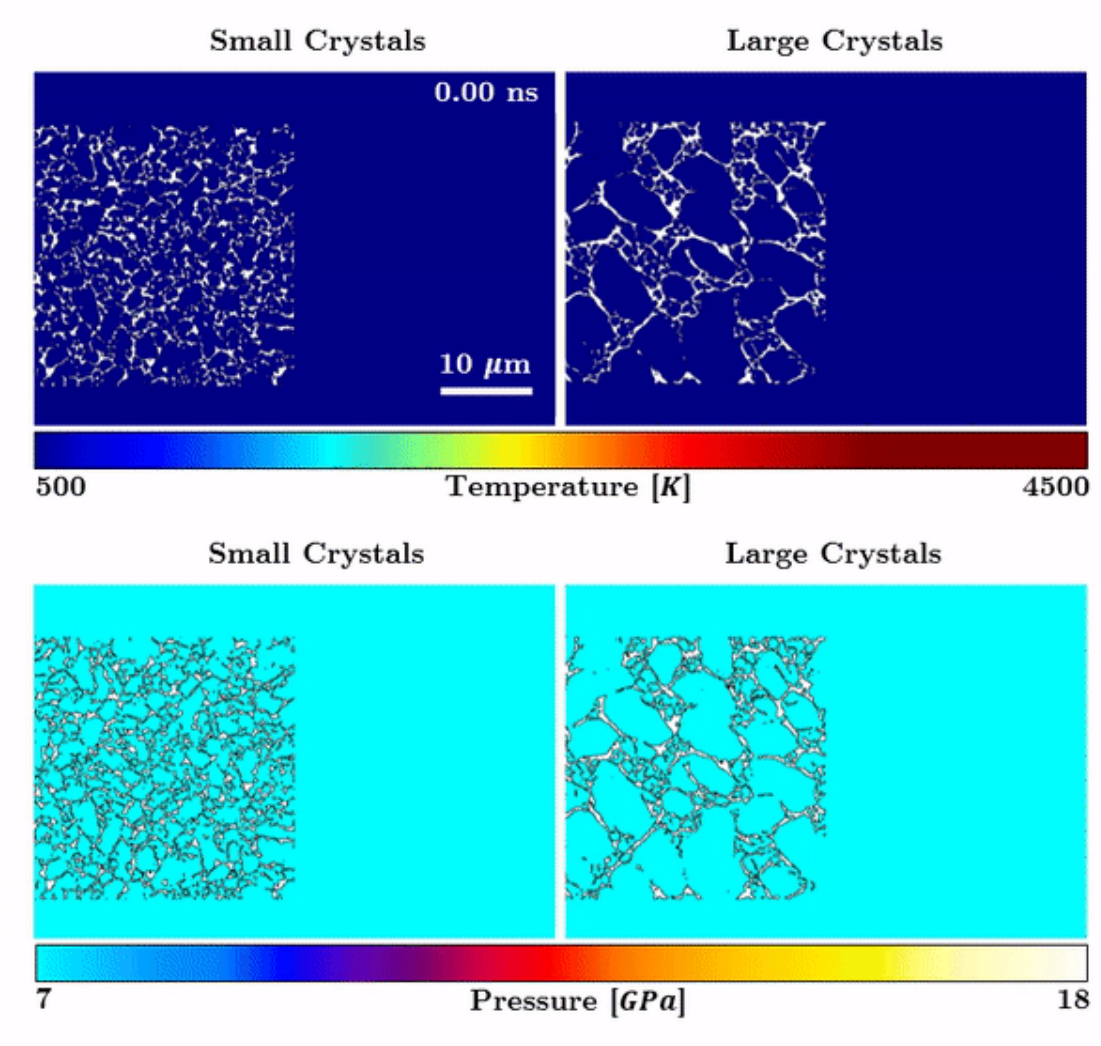
Smaller Grains



Larger Grains

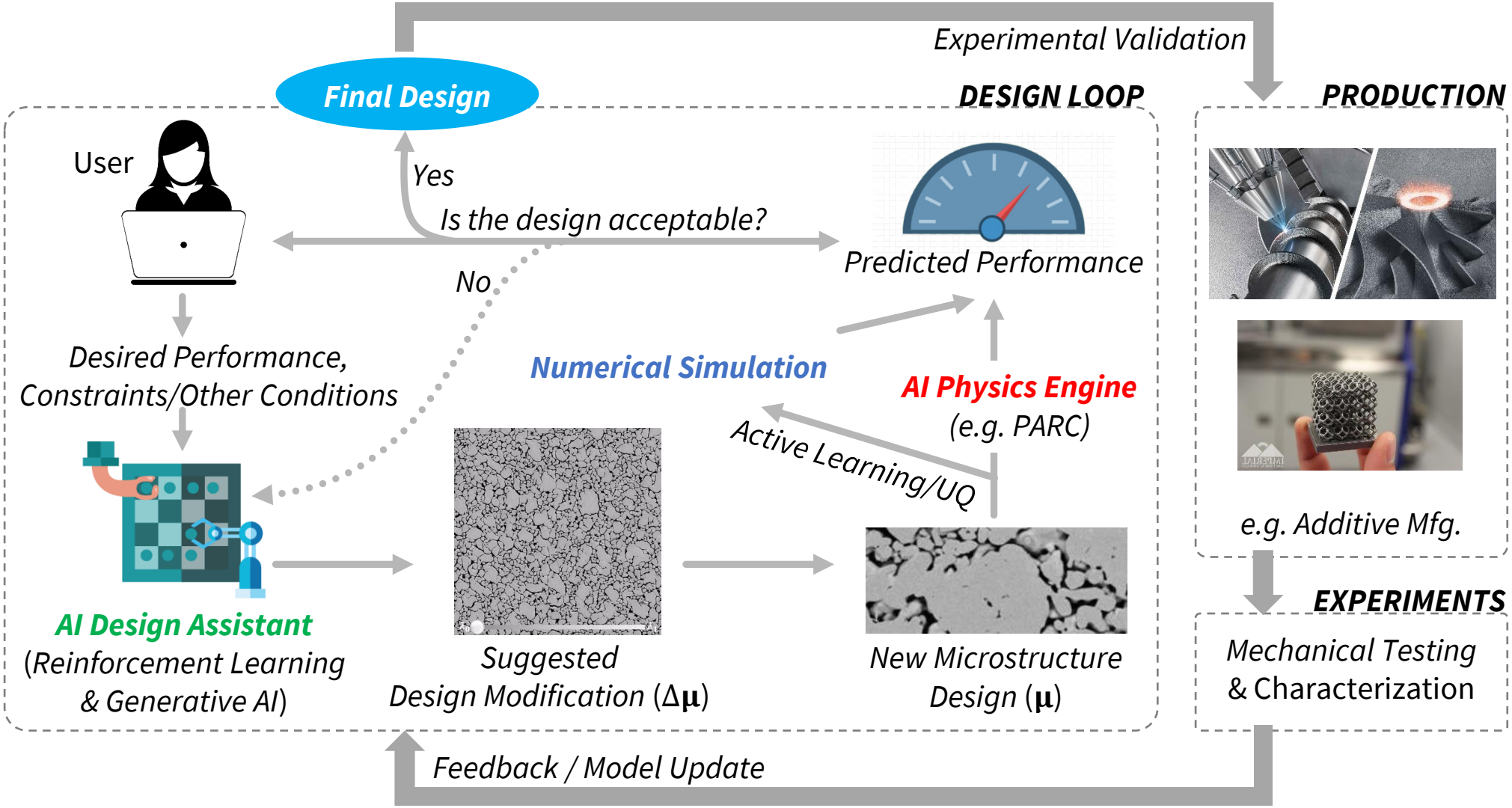
Dynamics w.r.t. Varying Microstructures



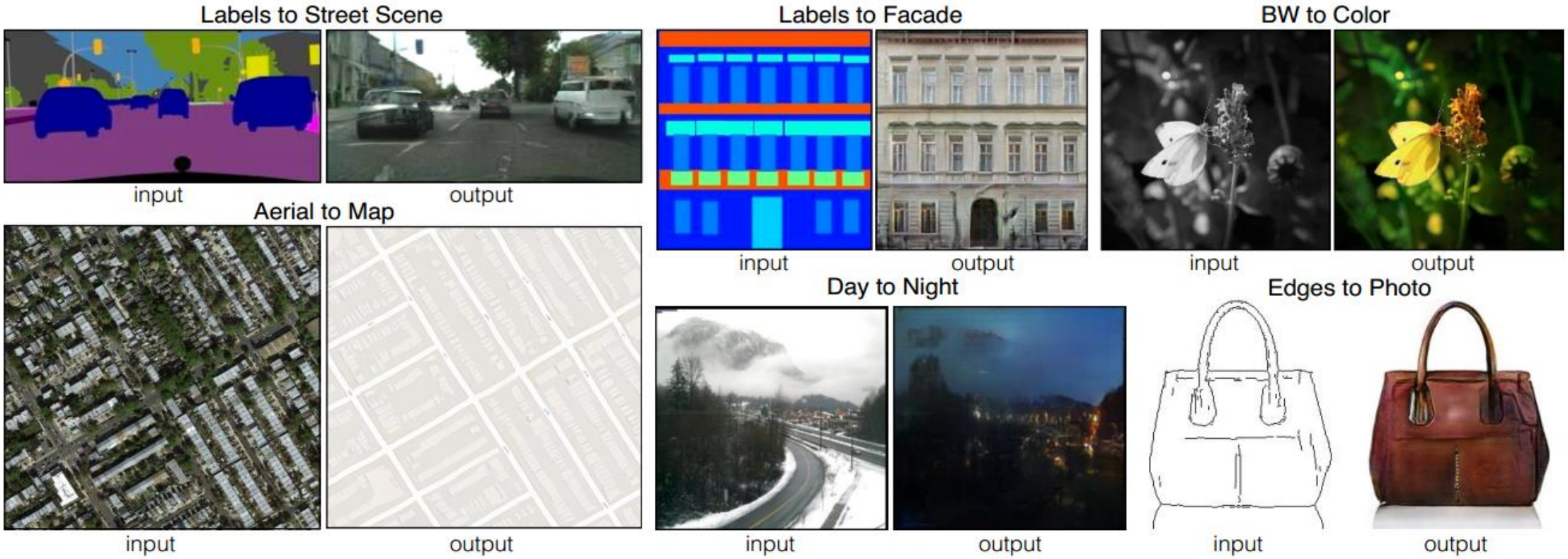


Chun et al. (2020) https://www.nature.com/articles/s41598-020-70149-0

GAN Digital Twin

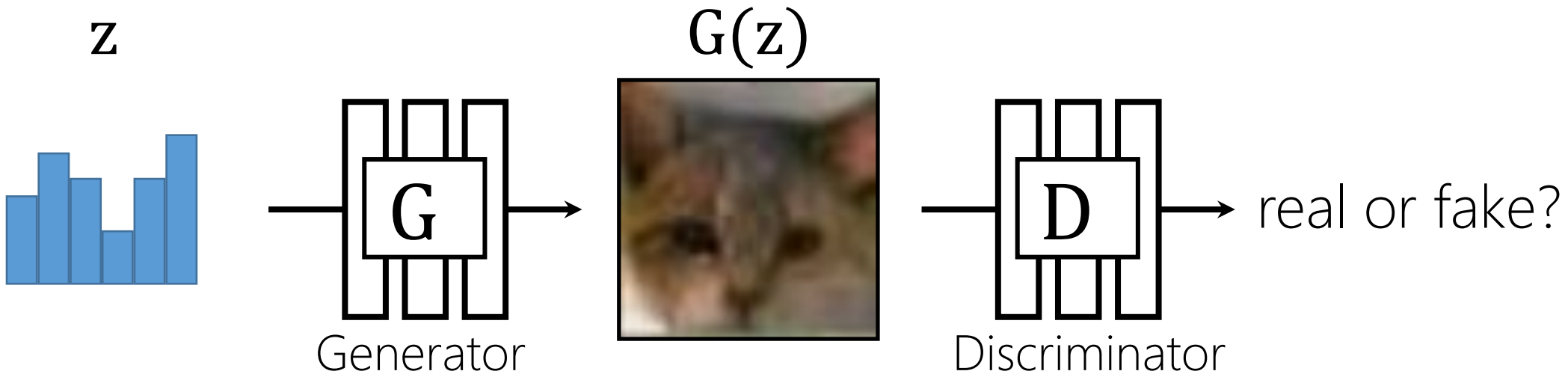


pix2pix: Image-to-image Translation



Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

Vanilla GAN



$$\min_G \max_D \mathbb{E}_{z,y}[\log D(G(z)) + \log(1 - D(y))]$$

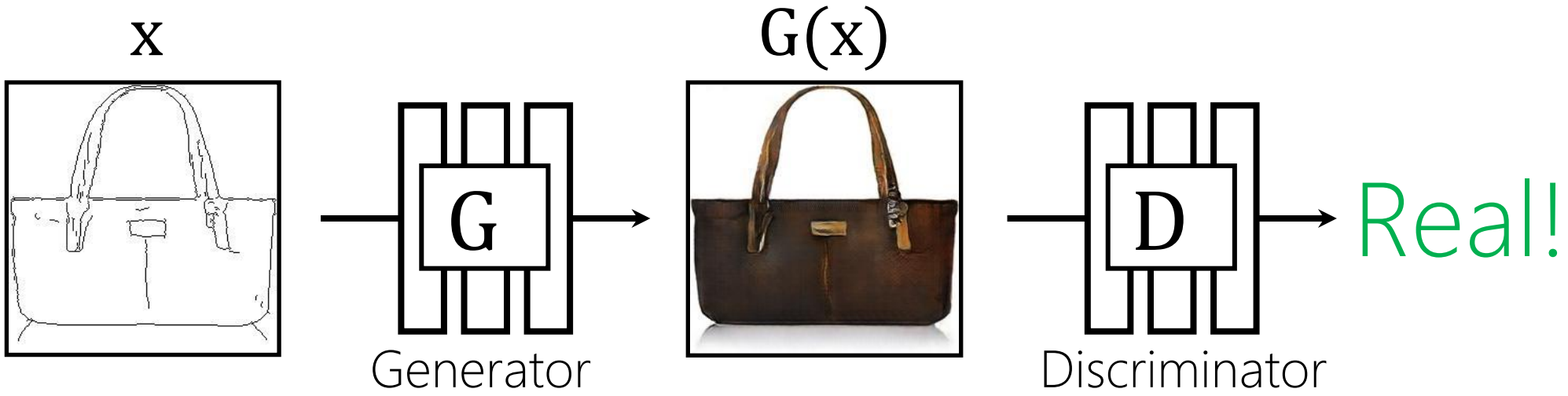
Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

Vanilla GAN

$$\min_G \max_D \mathbb{E}_{x,y}[\log D(G(x)) + \log(1 - D(y))]$$

Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

Vanilla GAN



$$\min_G \max_D \mathbb{E}_{x,y}[\log D(G(x)) + \log(1 - D(y))]$$

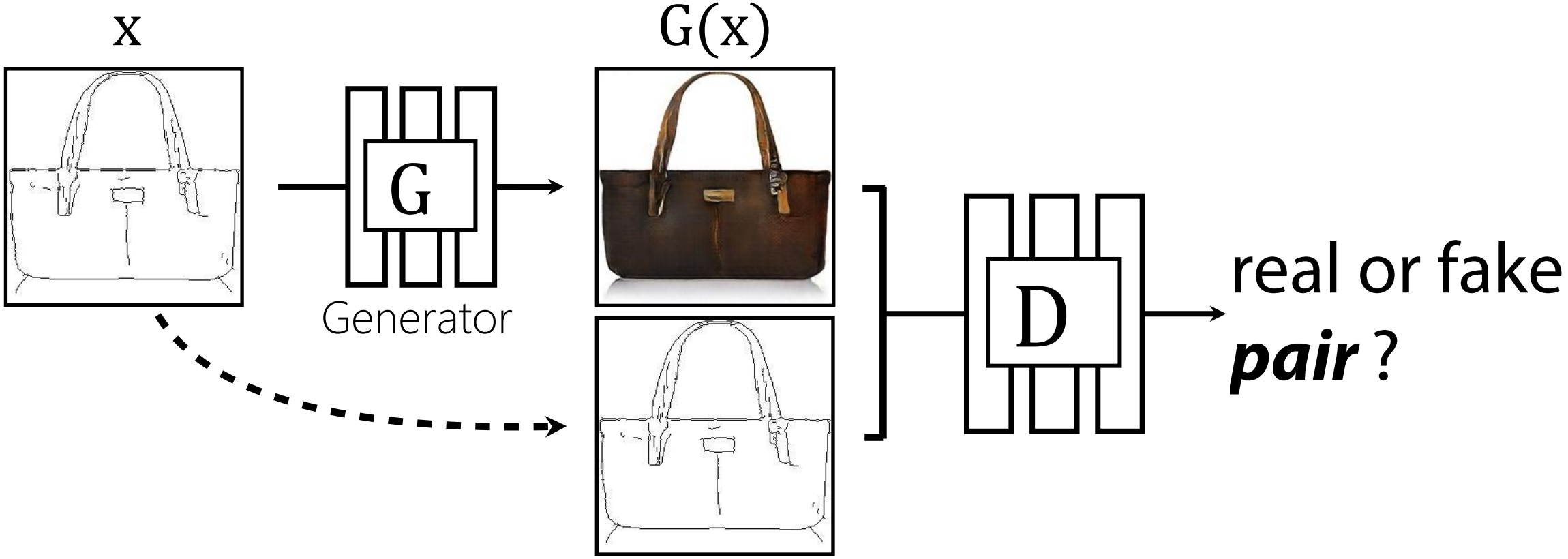
Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

Vanilla GAN

$$\min_G \max_D \mathbb{E}_{x,y}[\log D(G(x)) + \log(1 - D(y))]$$

Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

Pair-wise GAN (pix2pix)



$$\min_G \max_D \mathbb{E}_{x,y}[\log D(x, G(x)) + \log(1 - D(x, y))]$$

fake pair

real pair

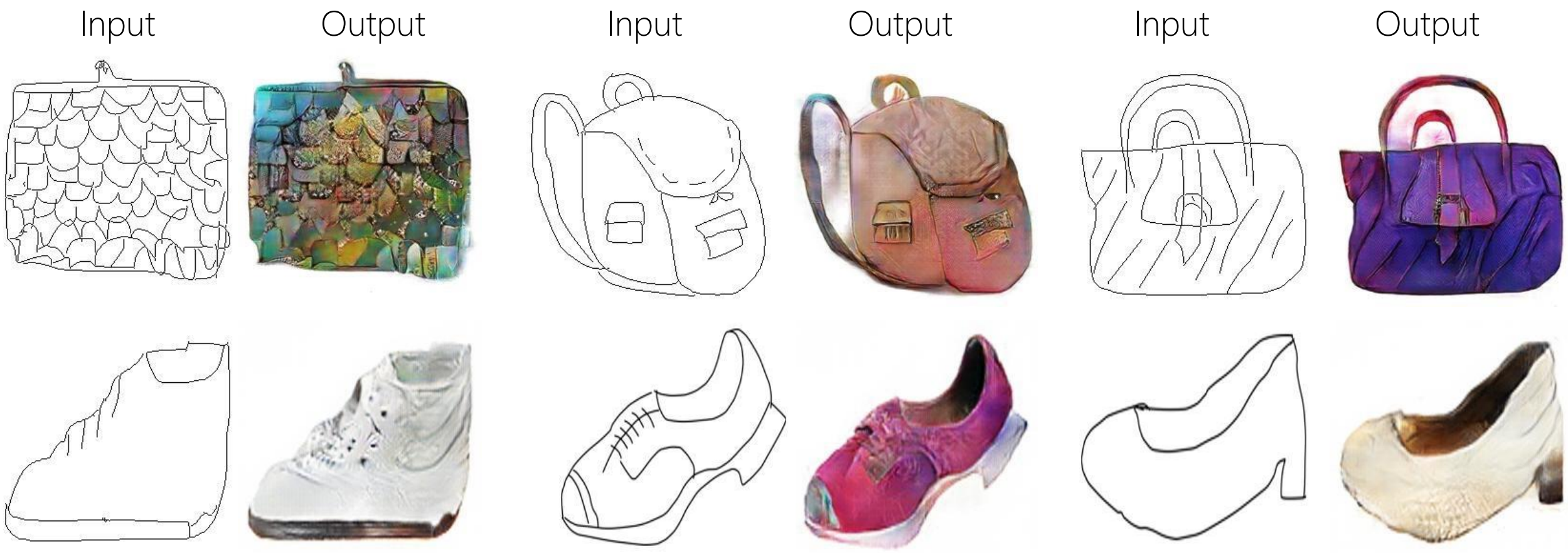
Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

Edges → Images

Input Output Input Output Input Output

Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

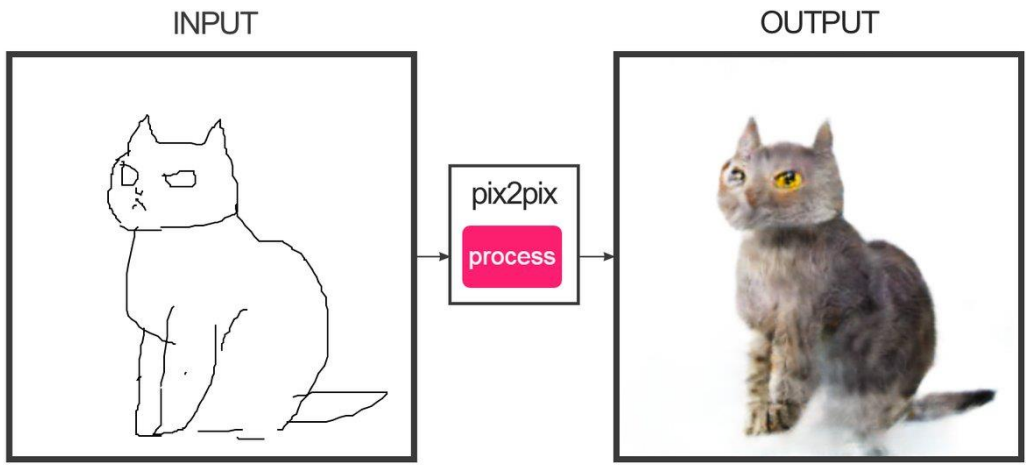
Sketches → Images

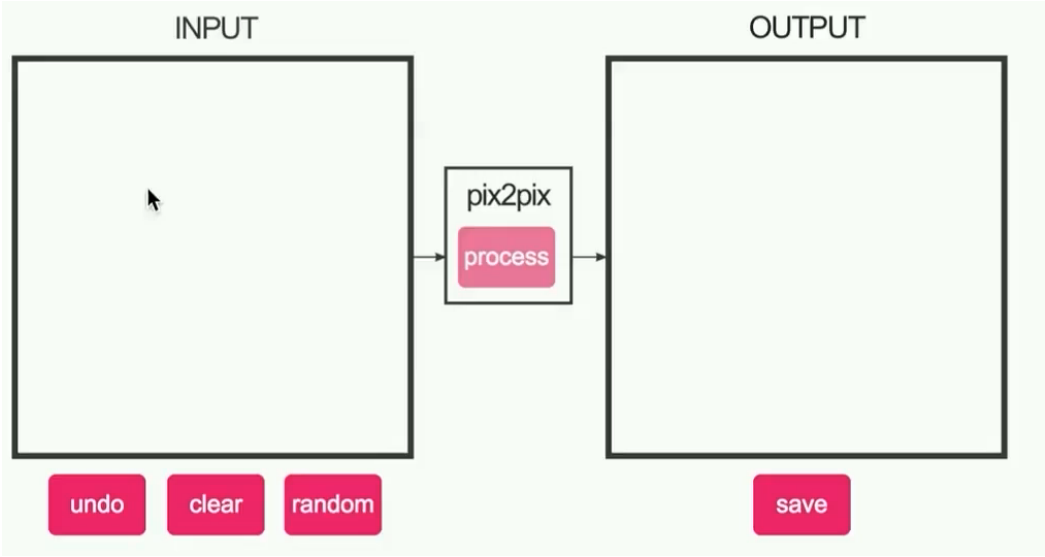


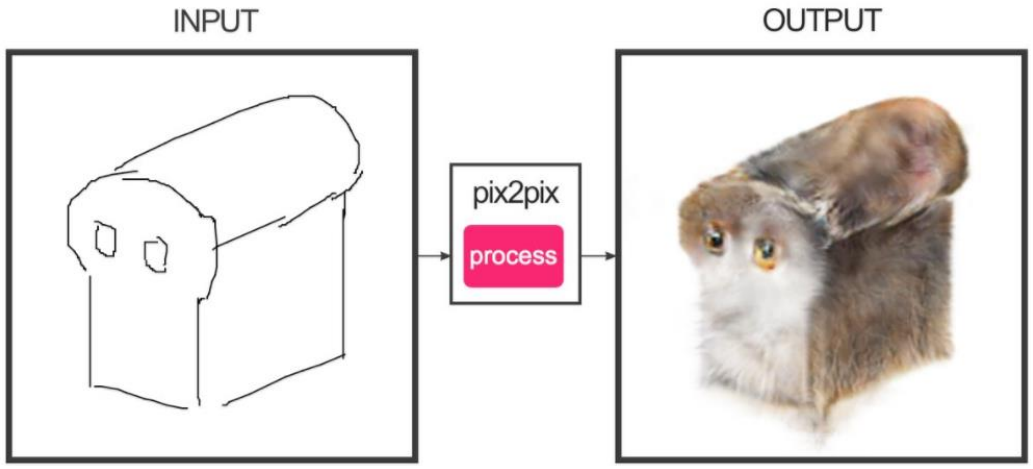
Trained on Edges → Images

Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

https://affinelayer.com/pixsrv/







Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

Map → Satellite Photo

Input

Output

Groundtruth

Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

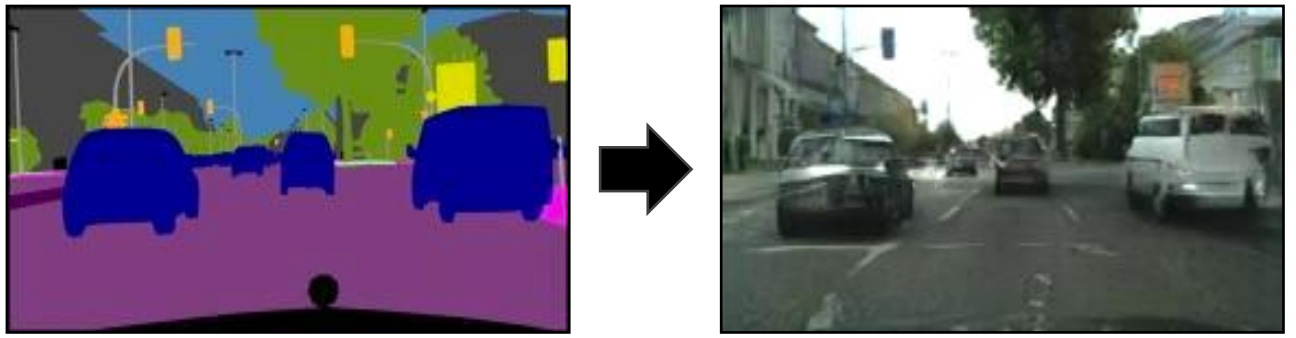
BW → Color

Input Output Input Output Input Output

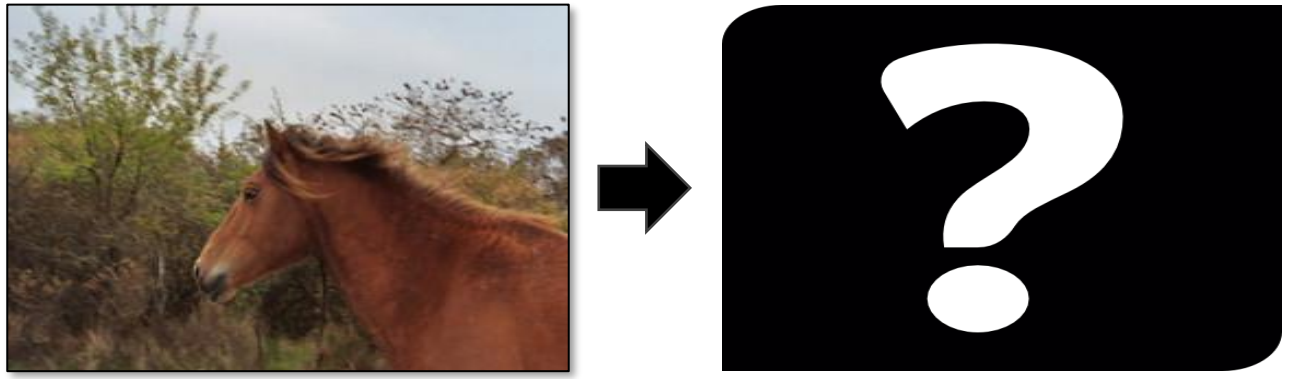
Isola et al. "Image-to-Image Translation with Conditional Adversarial Nets", CVPR2017

pix2pix is great, but training data??

Paired



Label ↔ photo: per-pixel labeling



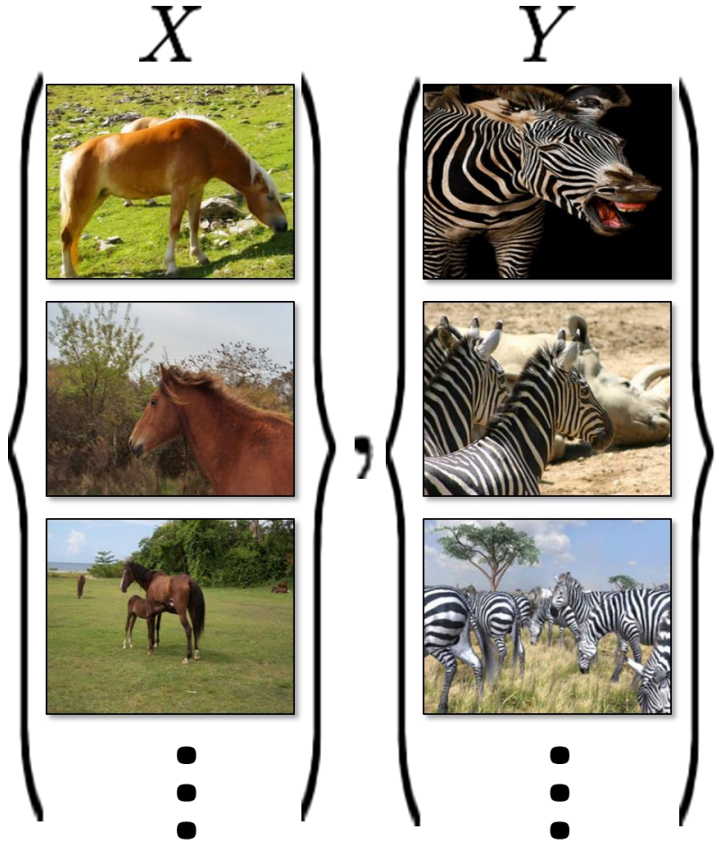
Horse ↔ zebra: how to get zebras?

Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV2017

pix2pix is great, but training data??

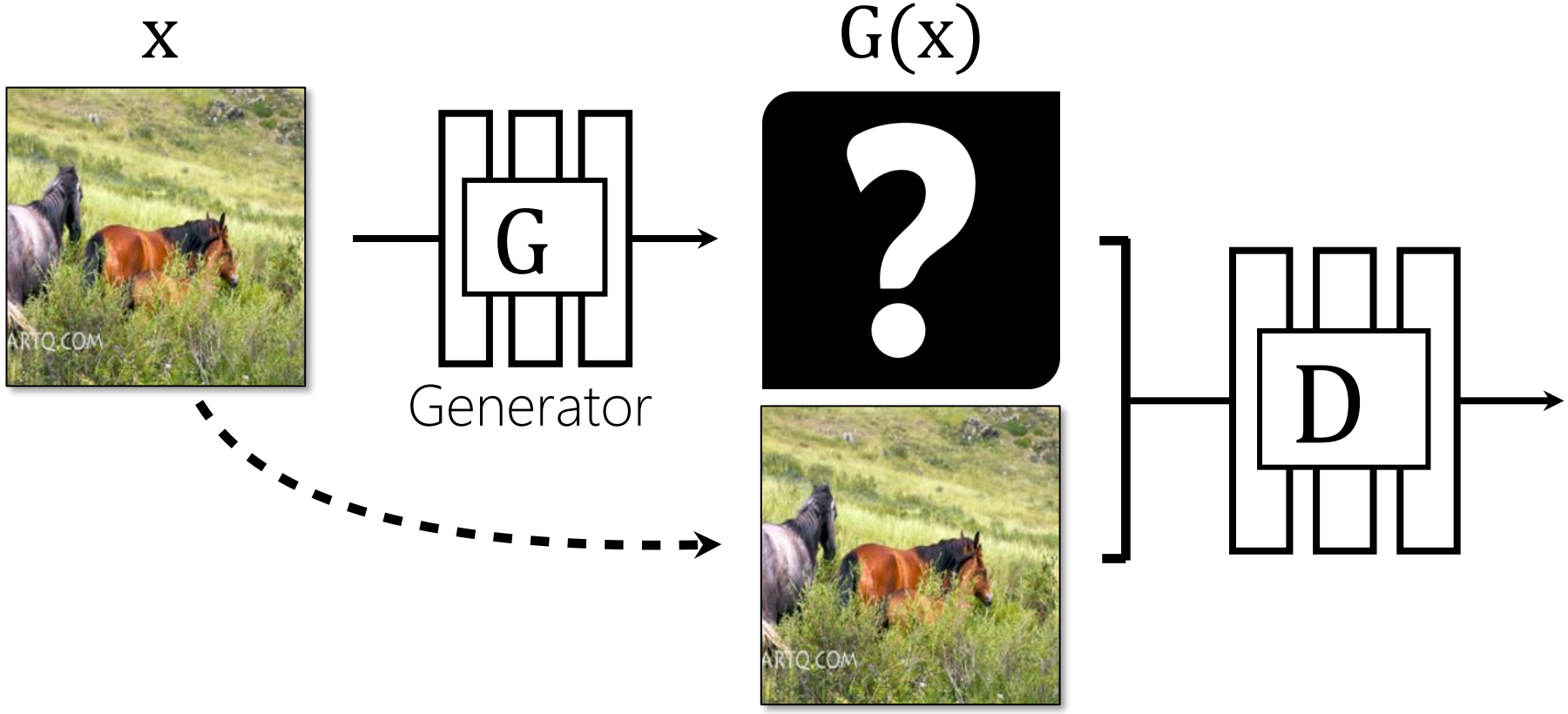
Paired

Unpaired



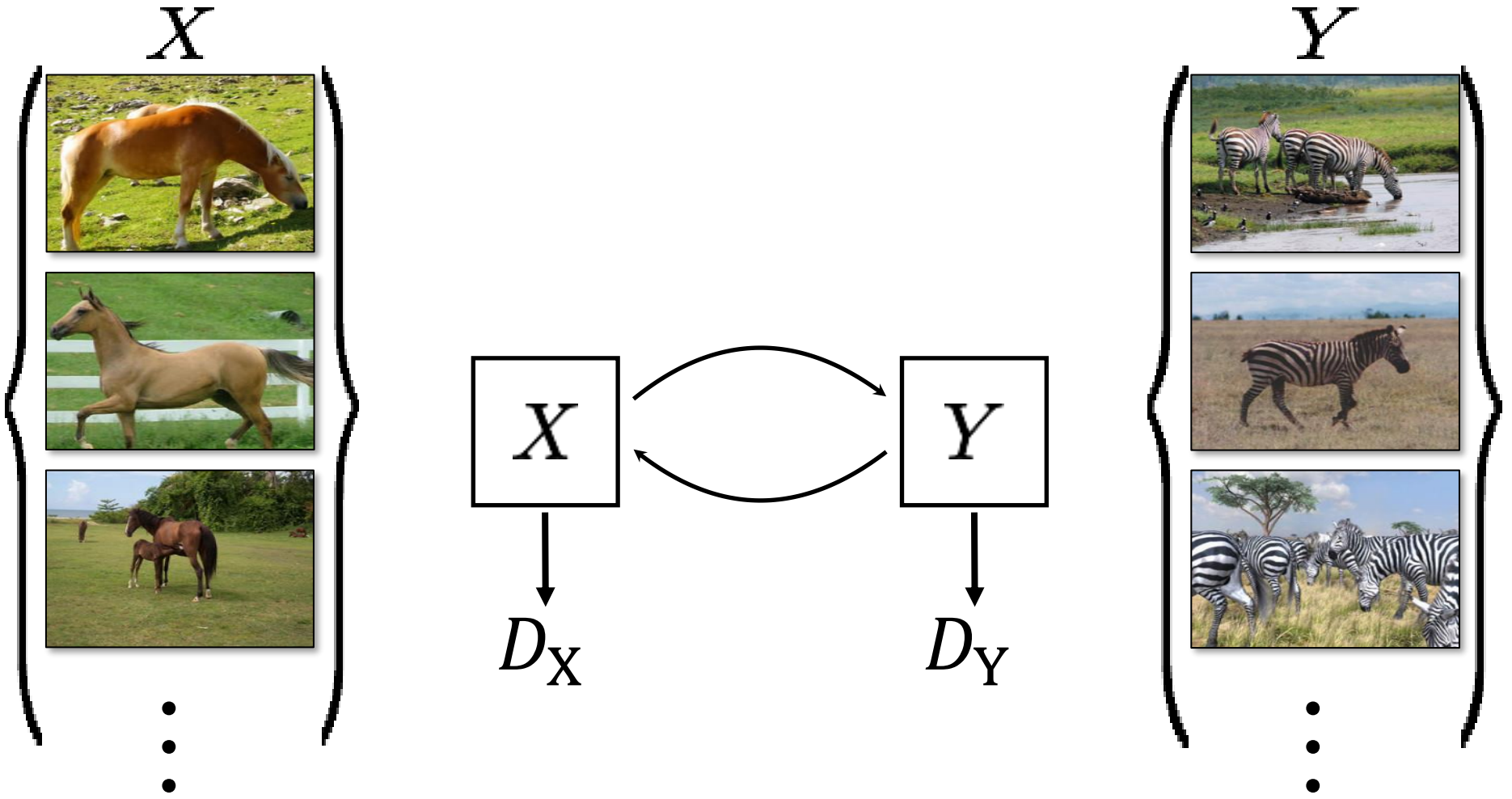
Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV2017

No Input-Output Pairs!



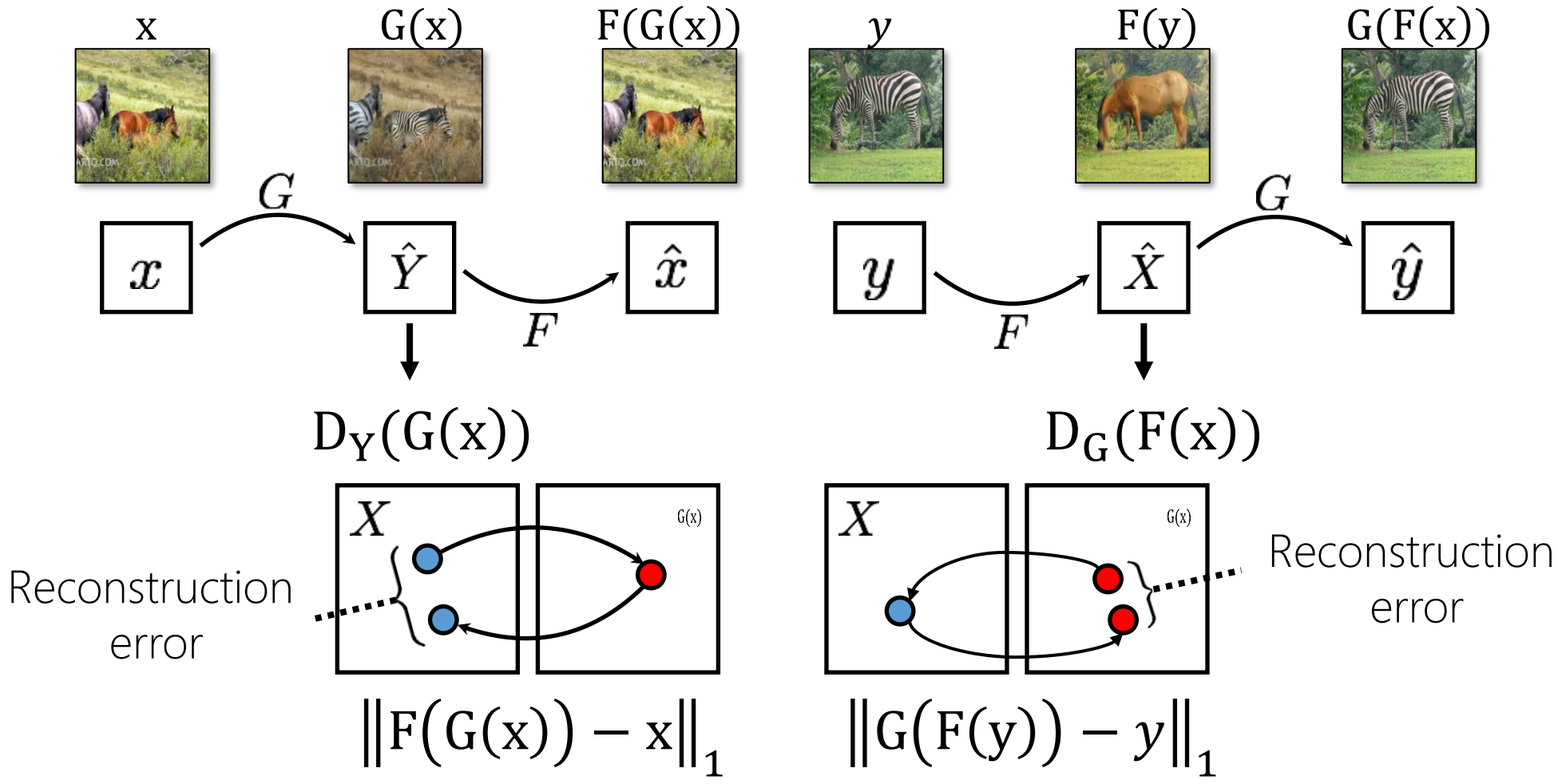
Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV2017

Cycle-Consistent Adversarial Networks



Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV2017

Cycle-Consistency Loss

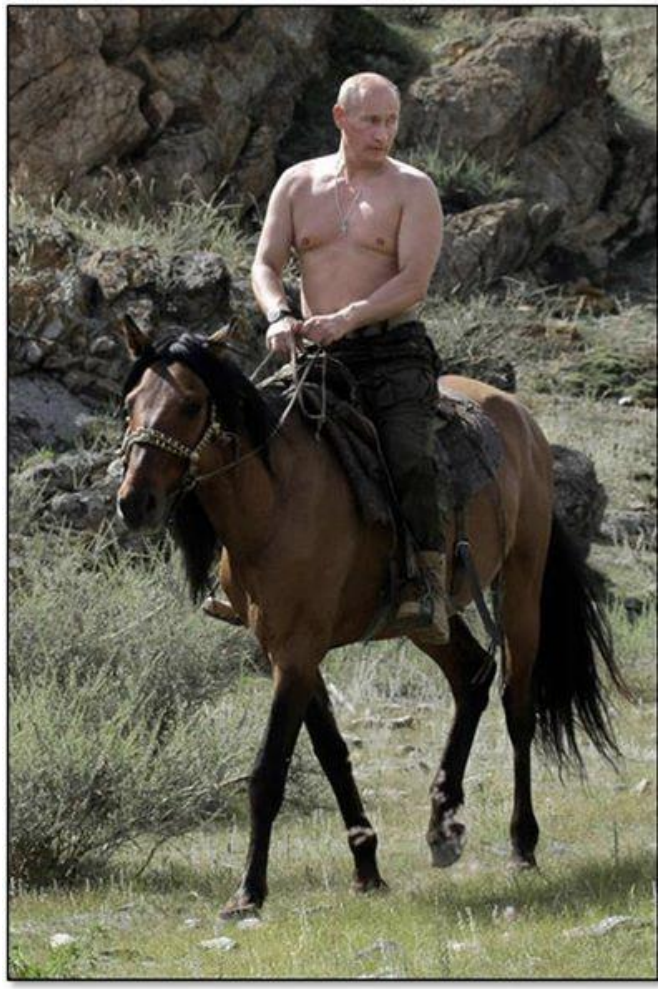


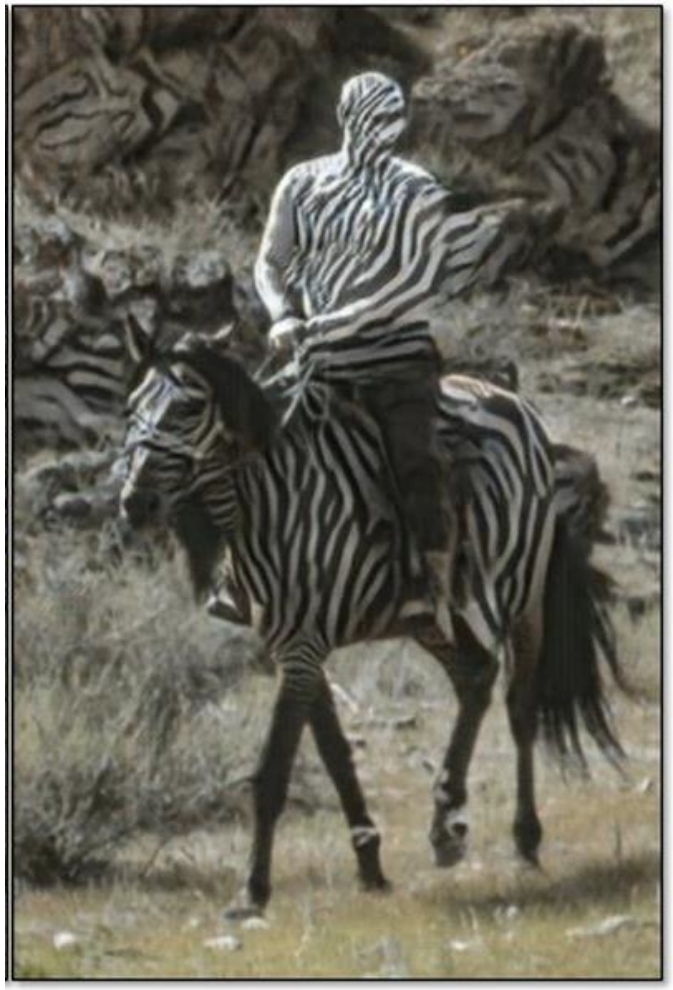
Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV2017

CycleGAN

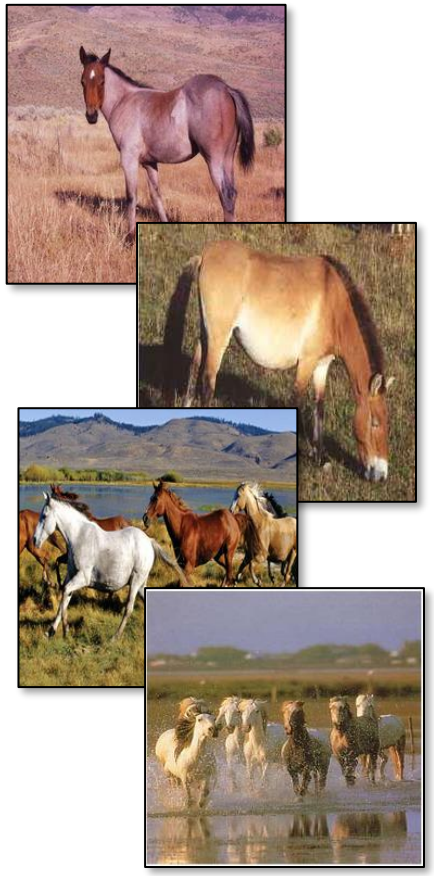
https://junyanz.github.io/CycleGAN/

Failure Case



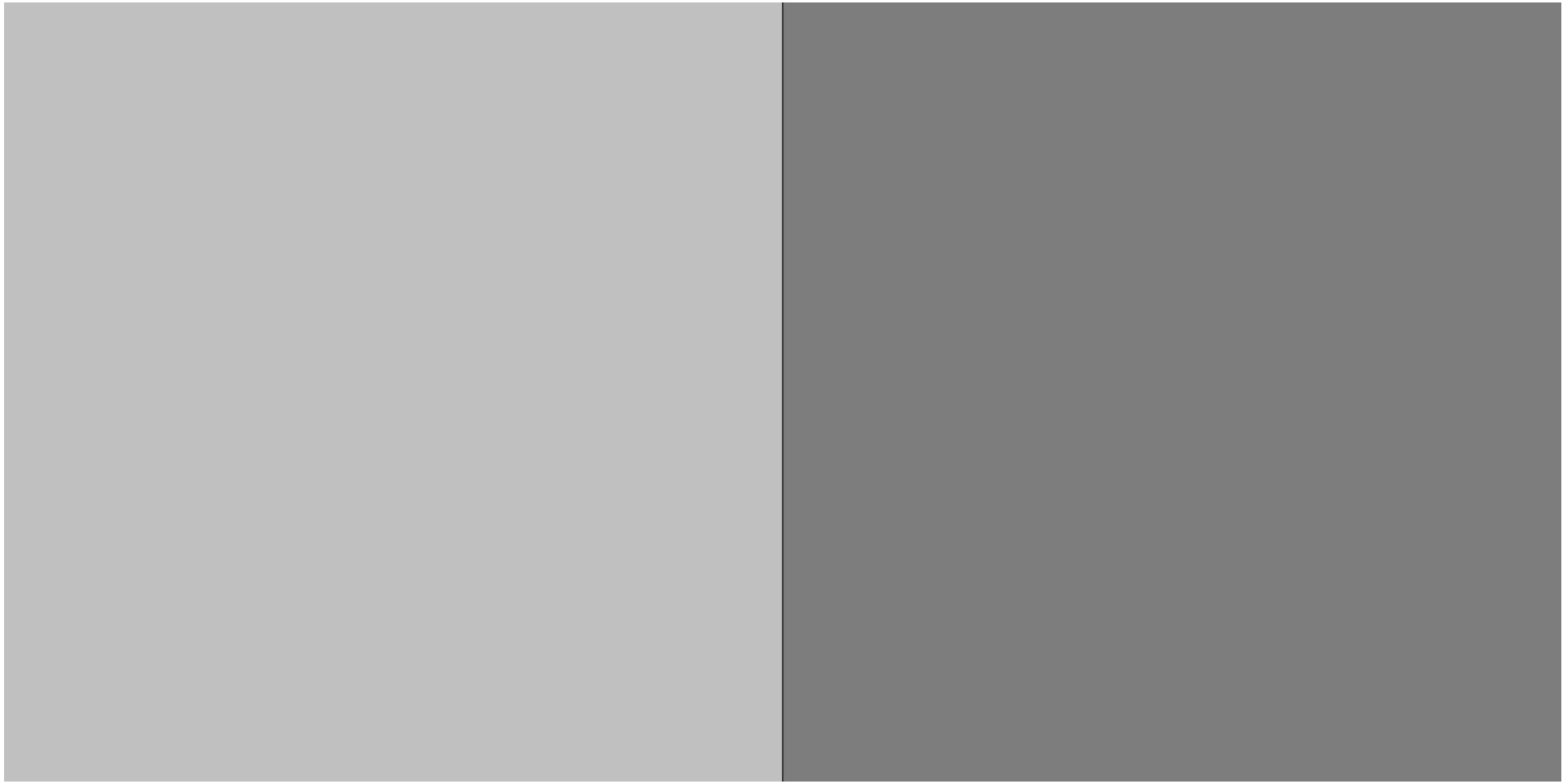


ImageNet
"Wild horse"



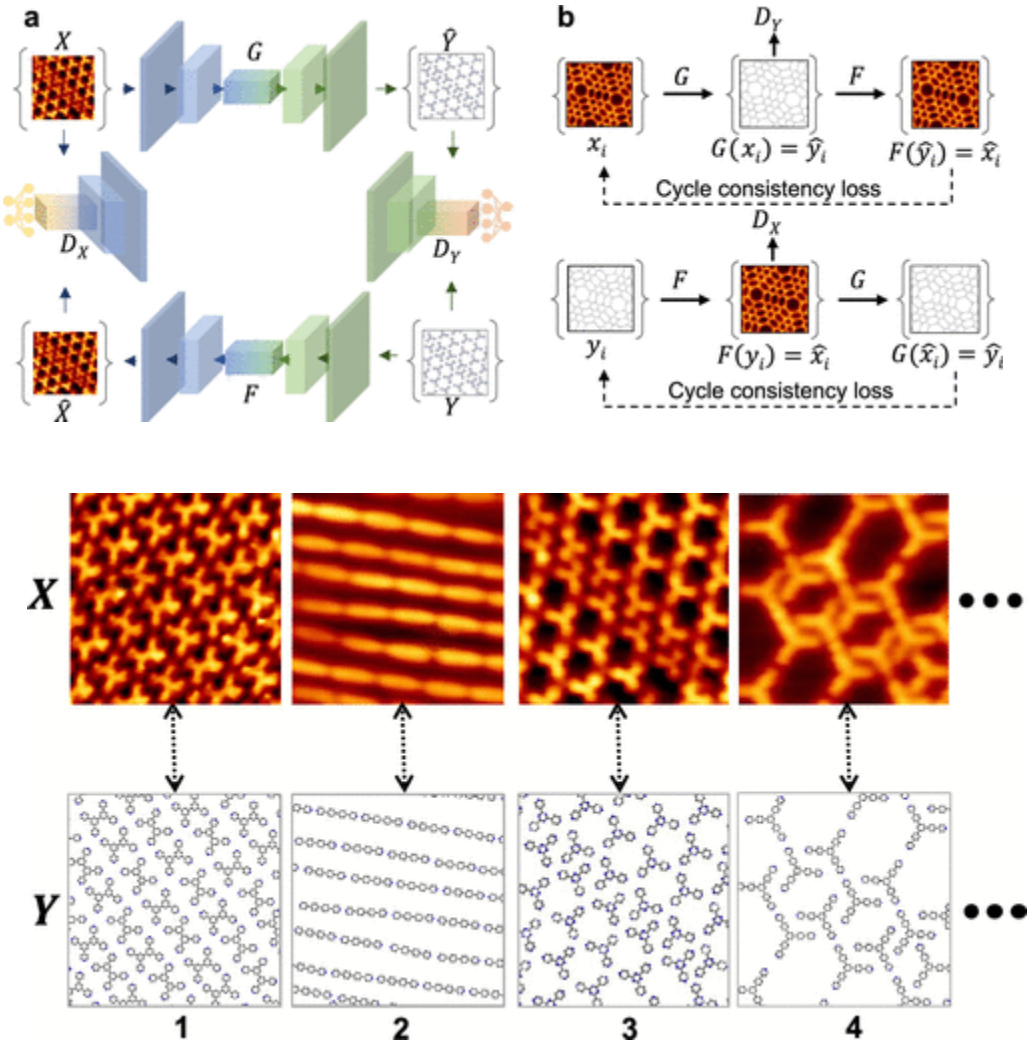
Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV2017

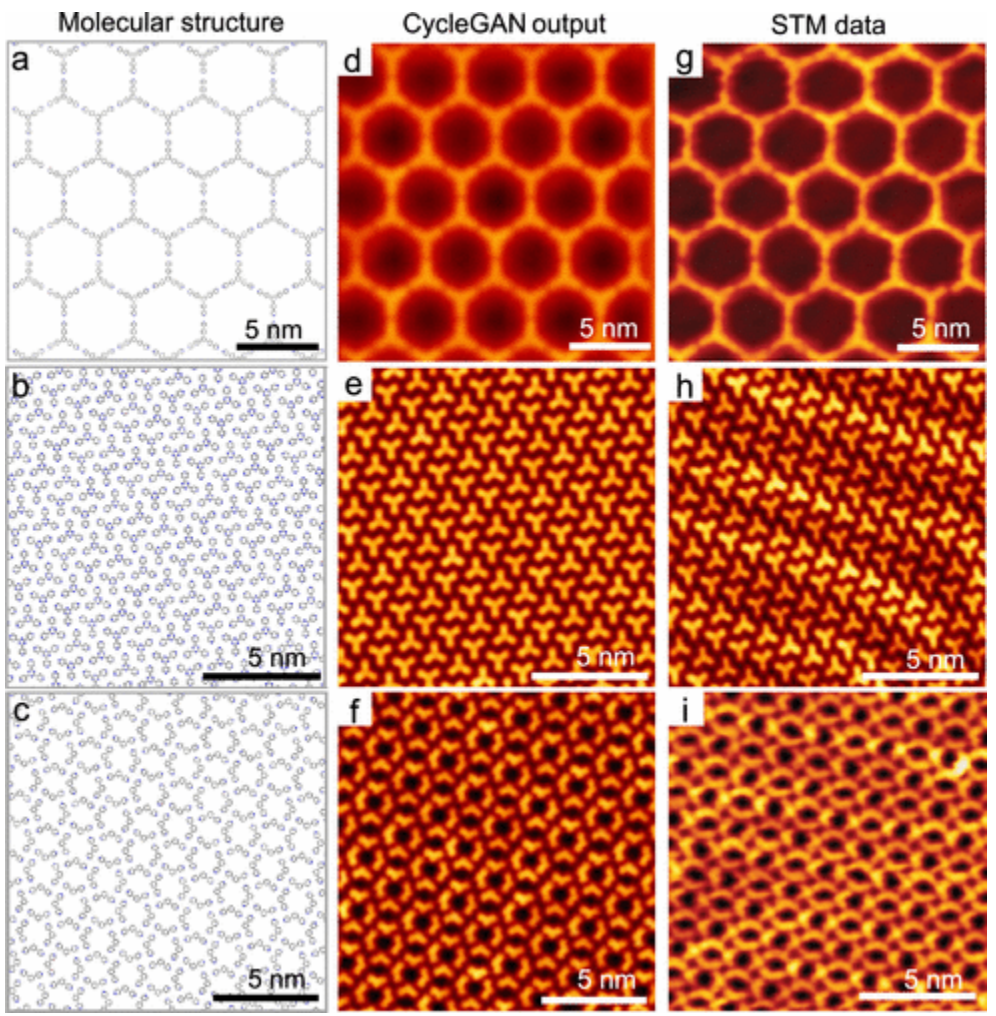
Automated Data Analysis Using CycleGAN



Zhu et al. (2024). Automated generation and analysis of molecular images using generative artificial intelligence models. *J. Phy. Chem. Letters*, *15*(7), 1985-1992.

Automated Data Analysis Using CycleGAN

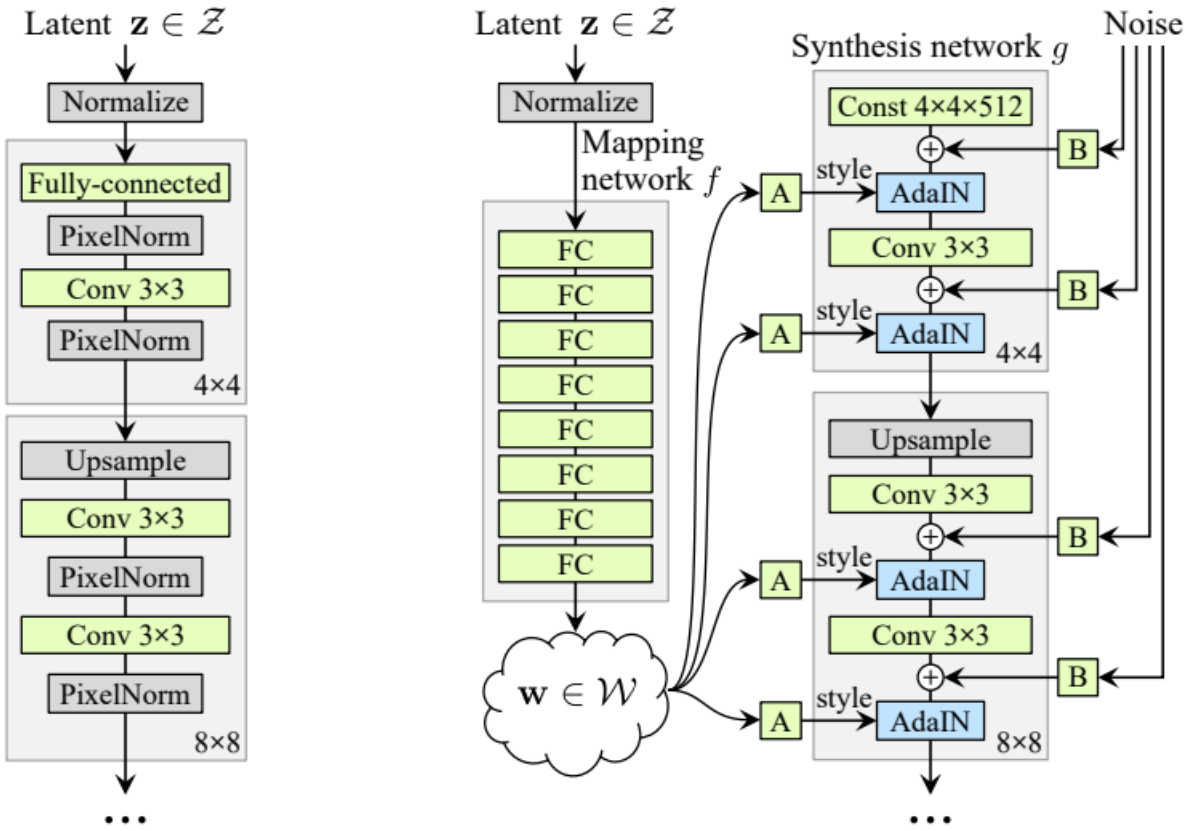




Zhu et al. (2024). Automated generation and analysis of molecular images using generative artificial intelligence models. *J. Phy. Chem. Letters*, *15*(7), 1985-1992.

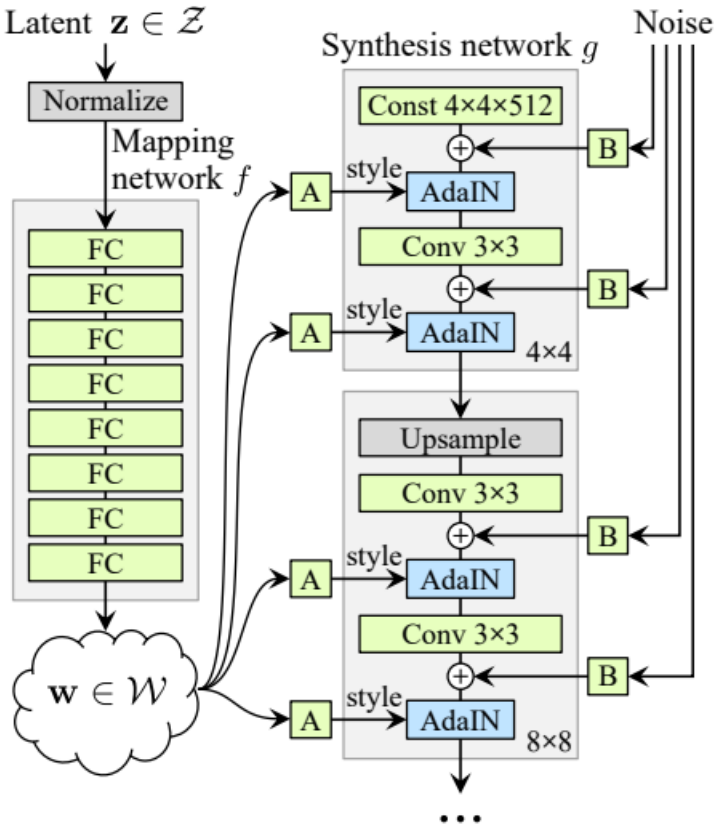
StyleGAN

- Karras et al. (2019)



StyleGAN

- Karras et al. (2019)



"Styles" $\mathbf{y} = (\mathbf{y}_s, \mathbf{y}_b)$

i-th feature map

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$

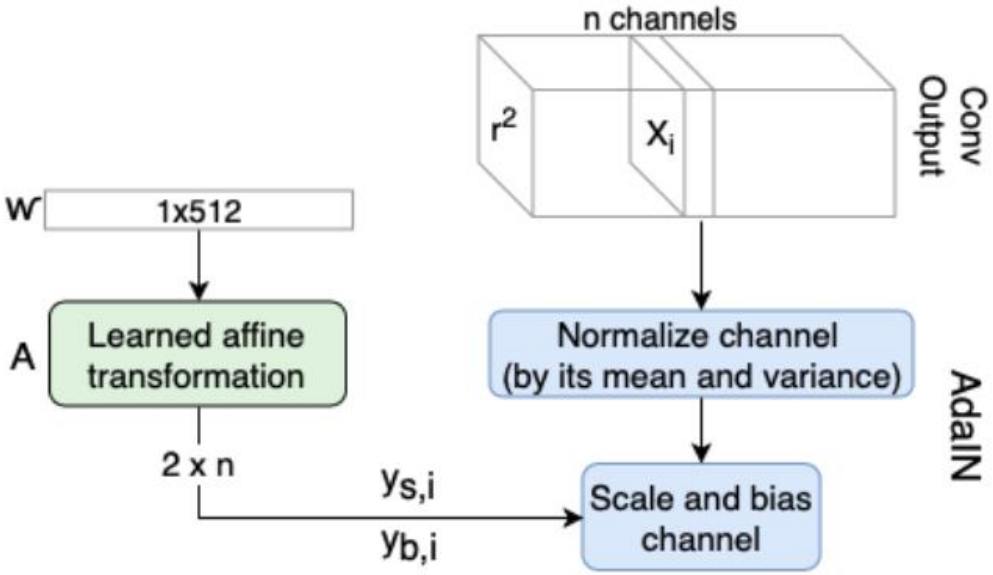
StyleGAN

DeepFake

https://www.youtube.com/watch?v=fqye4Cw0EF0

PARCel – Extending PARC to other operating conditions

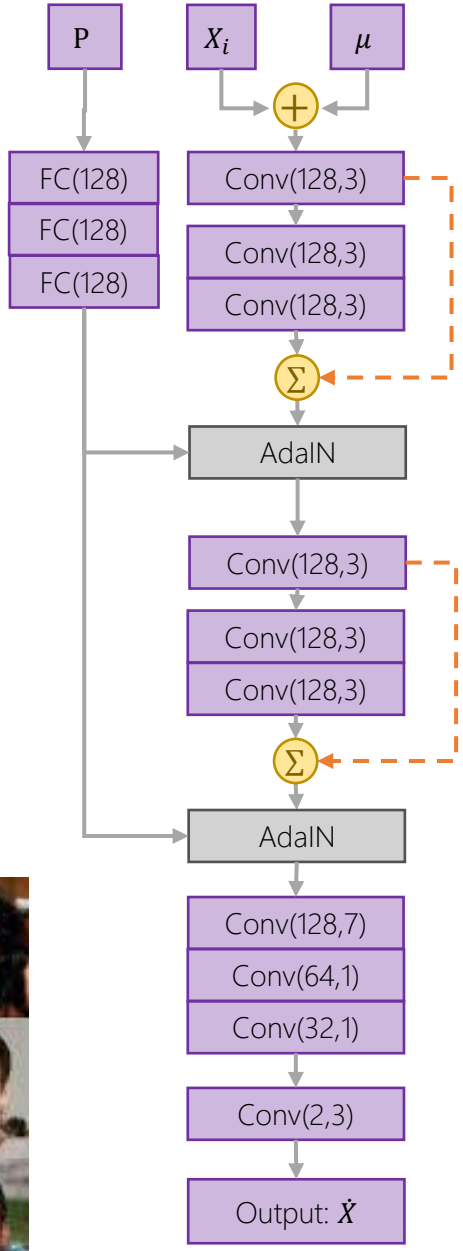
- "Style-vector" in StyleGAN



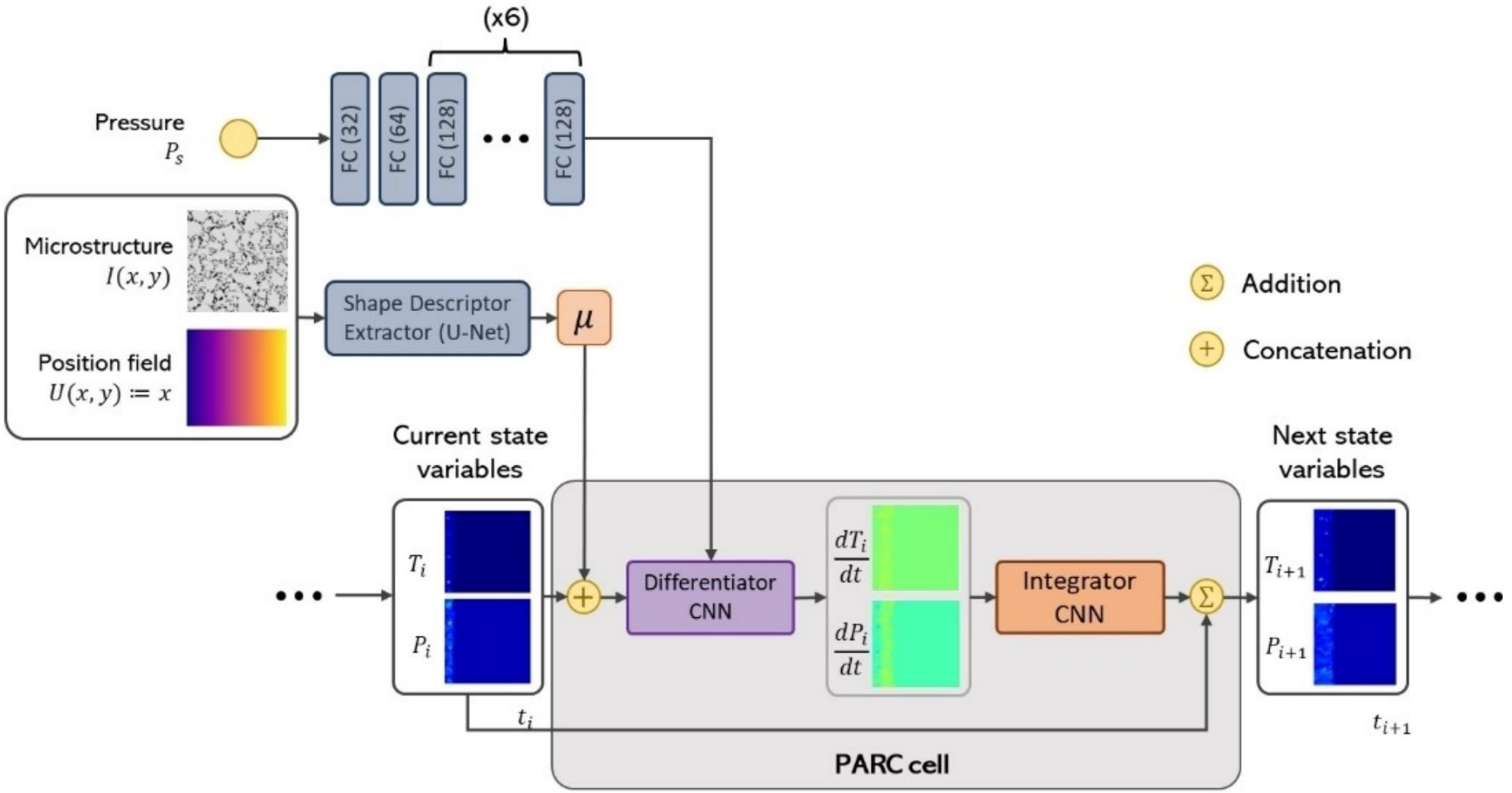
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

Karras, Laine, & Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks," CVPR 2019

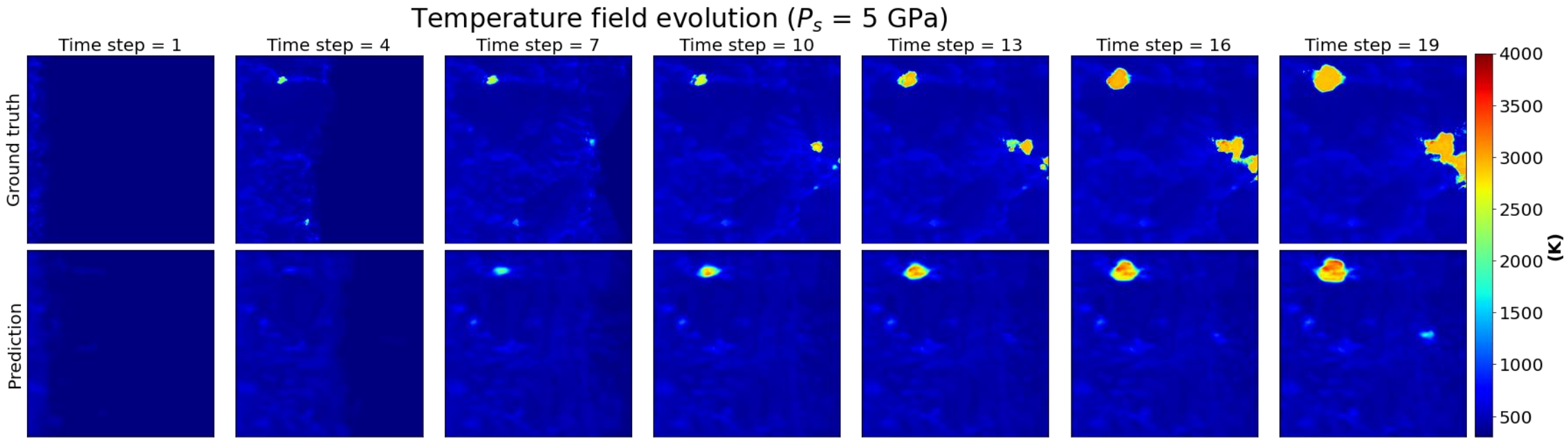
Introducing "Bias" to represent the effects of different shock loadings.



PARCel – Extending PARC to other operating conditions

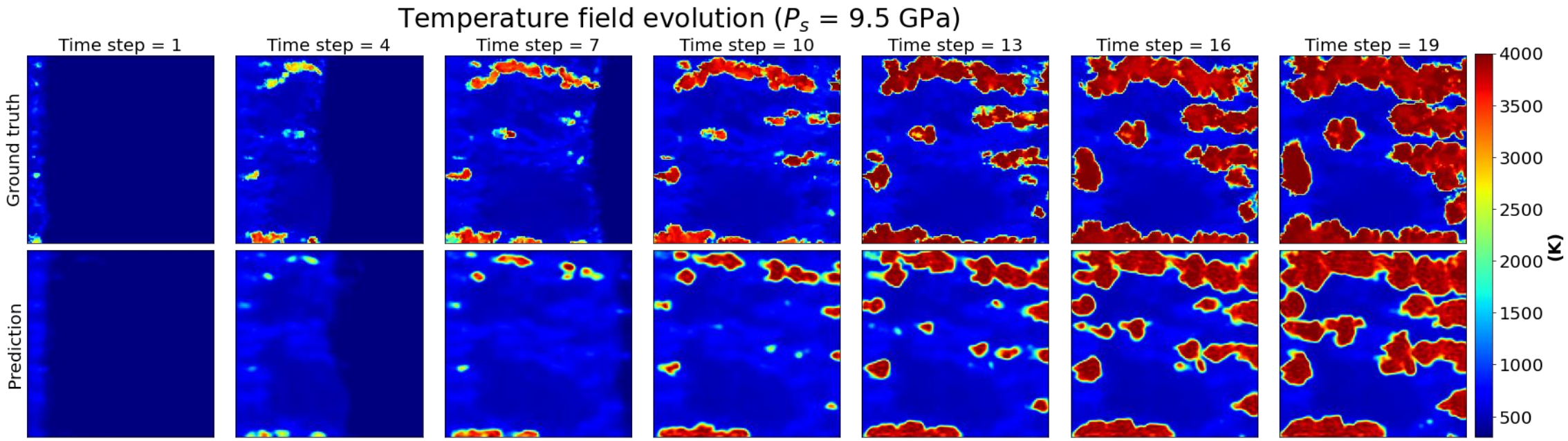


Visual Intelligence Laboratory

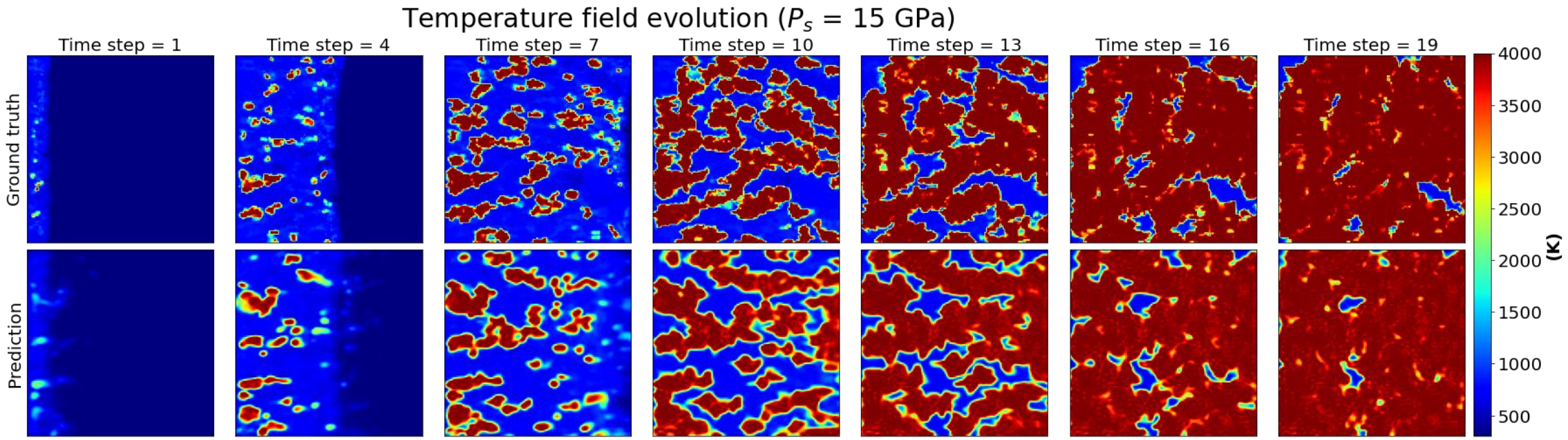


Nguyen et al. (2023). https://doi.org/10.1002/prep.202200268

Visual Intelligence Laboratory



Nguyen et al. (2023). https://doi.org/10.1002/prep.202200268

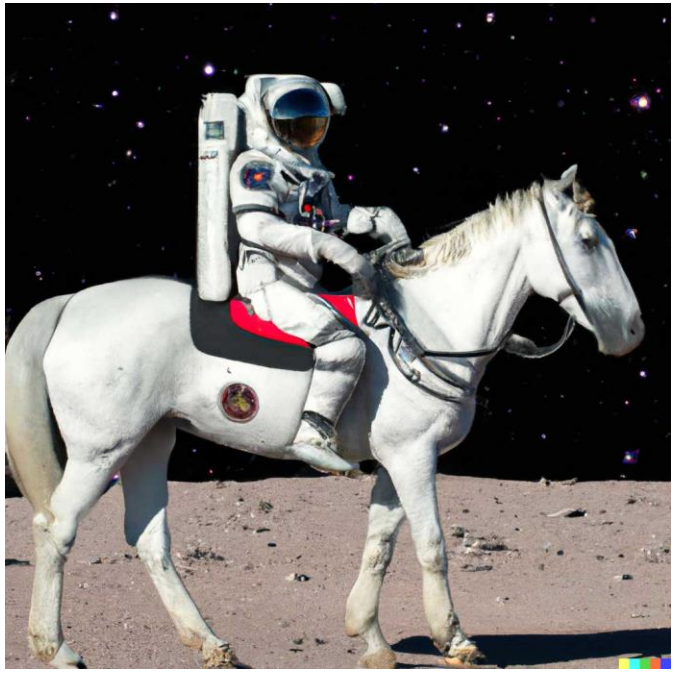


Nguyen et al. (2023). https://doi.org/10.1002/prep.202200268

Diffusion Models

Dall-E 2 (Open AI)

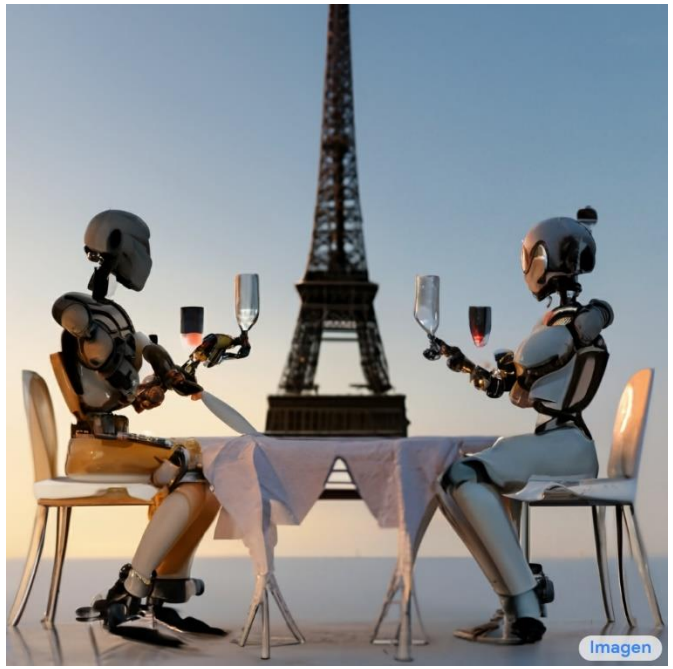
A bowl of soup as a planet in the universe



An astronaut riding a horse in a photorealistic style

Teddy bears mixing sparkling chemicals as mad scientists

Imagen (Google)



A robot couple fine-dining with the Eiffel Tower in the background

A majestic oil painting of a raccoon Queen wearing red French royal gown.

A cute corgi lives in a house made of sushi

Imagen (Google)

Imagic (Google)

Input Image | Edited Image

"A bird spreading wings"

Input Image | Edited Image

"Two kissing parrots"

Input Image | Edited Image

"A goat jumping over a cat"

"A photo of an open box"

"A photo of a sitting dog"

"A children's drawing of a waterfall"

Imagic (Google)

A vase of colorful tulips

A white wedding cake

A wooden chair

Make-A-Video (Meta)

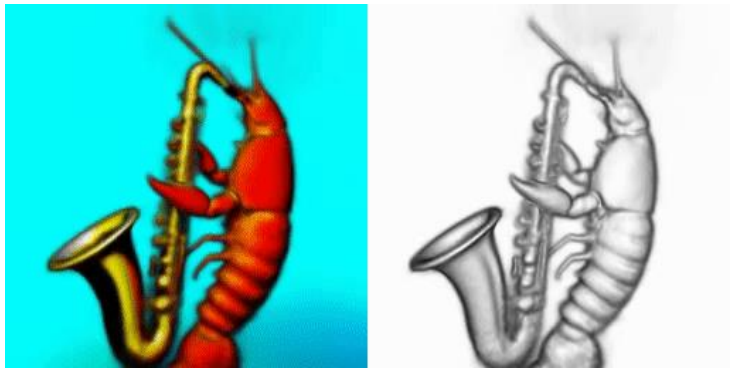
A confused grizzly bear in a calculus class

A golden retriever eating ice cream on a beautiful tropical beach at sunset, high resolution

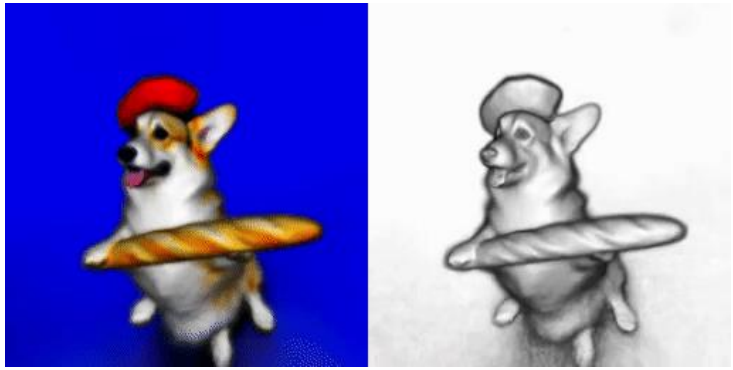
A panda playing on a swing set

DreamFusion (Google)

a fox holding a video game controller



a lobster playing the saxophone



a corgi wearing a beret and holding a baguette, standing up on two hind legs

a human skeleton drinking a glass of red wine

Recap: GAN and VAE

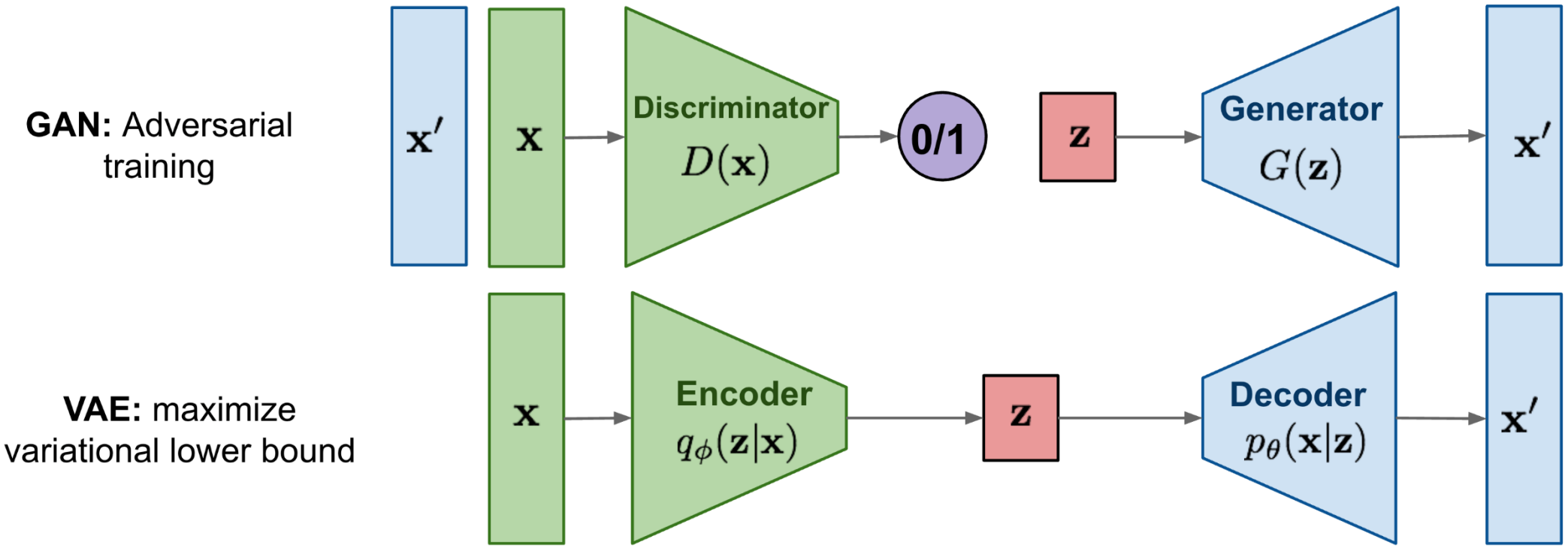
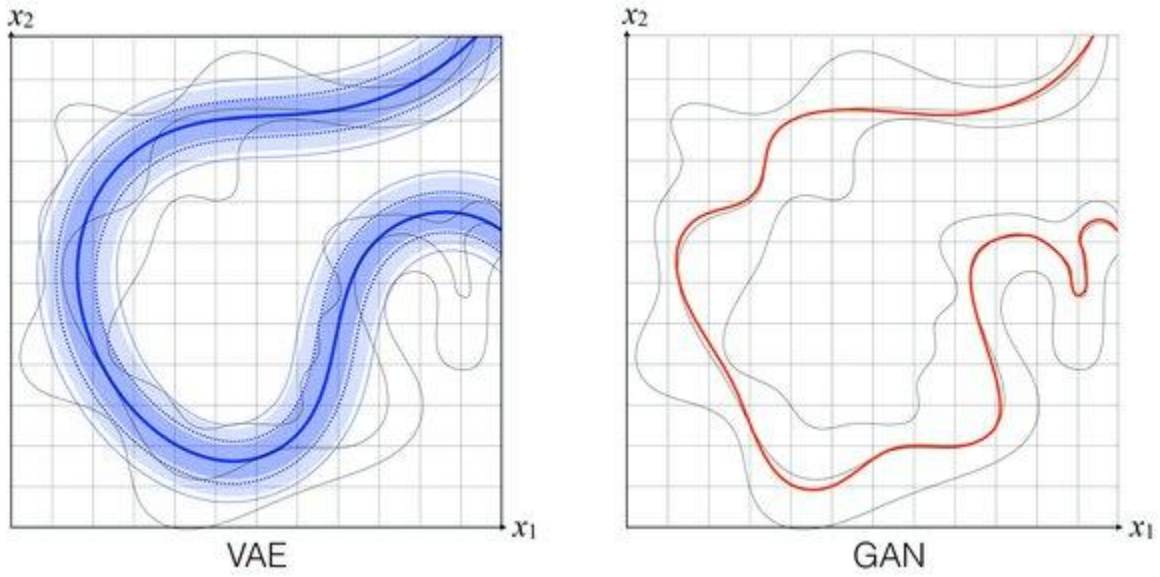


Image credit: https://github.com/yzy1996/Awesome-Generative-Model

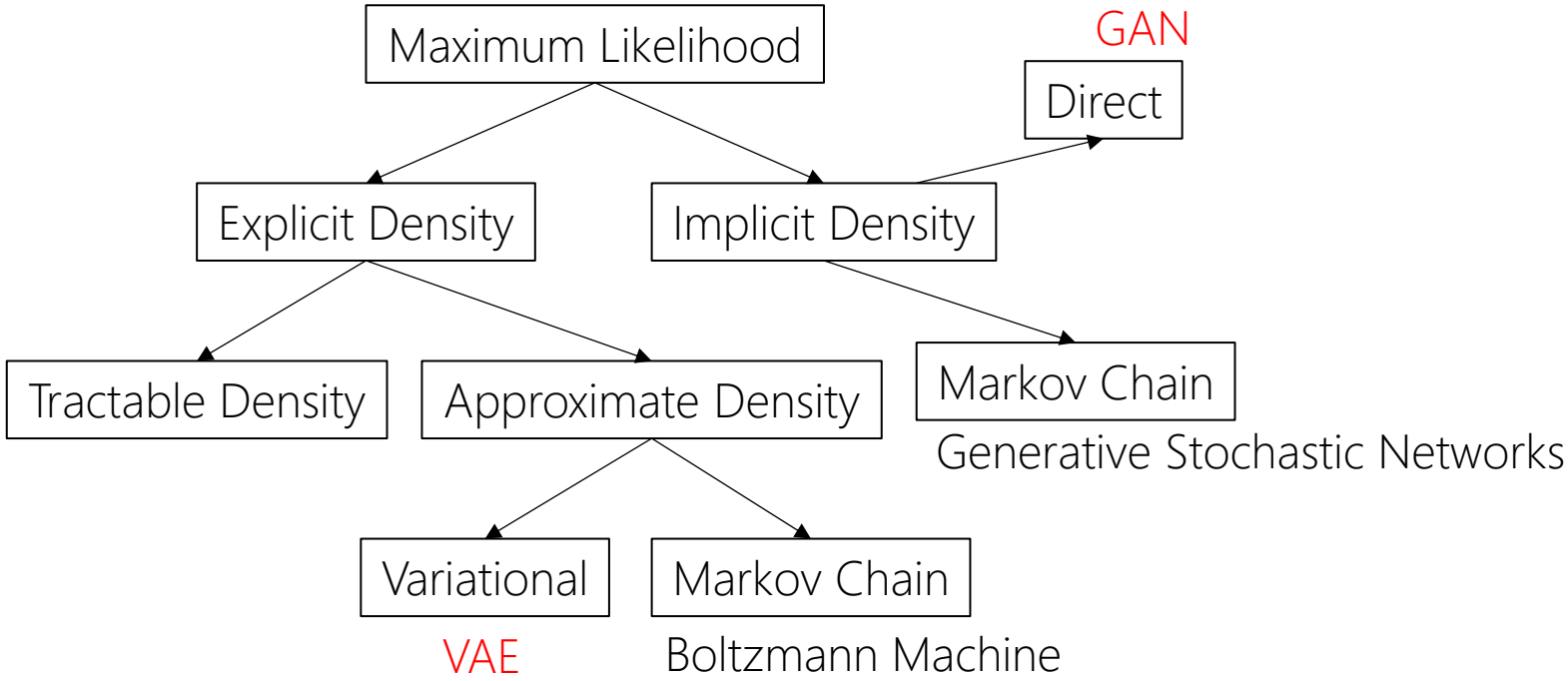
Recap: GAN and VAE

- Learning the distribution



Recap: GAN and VAE

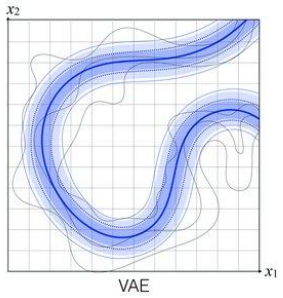
- Learning the distribution

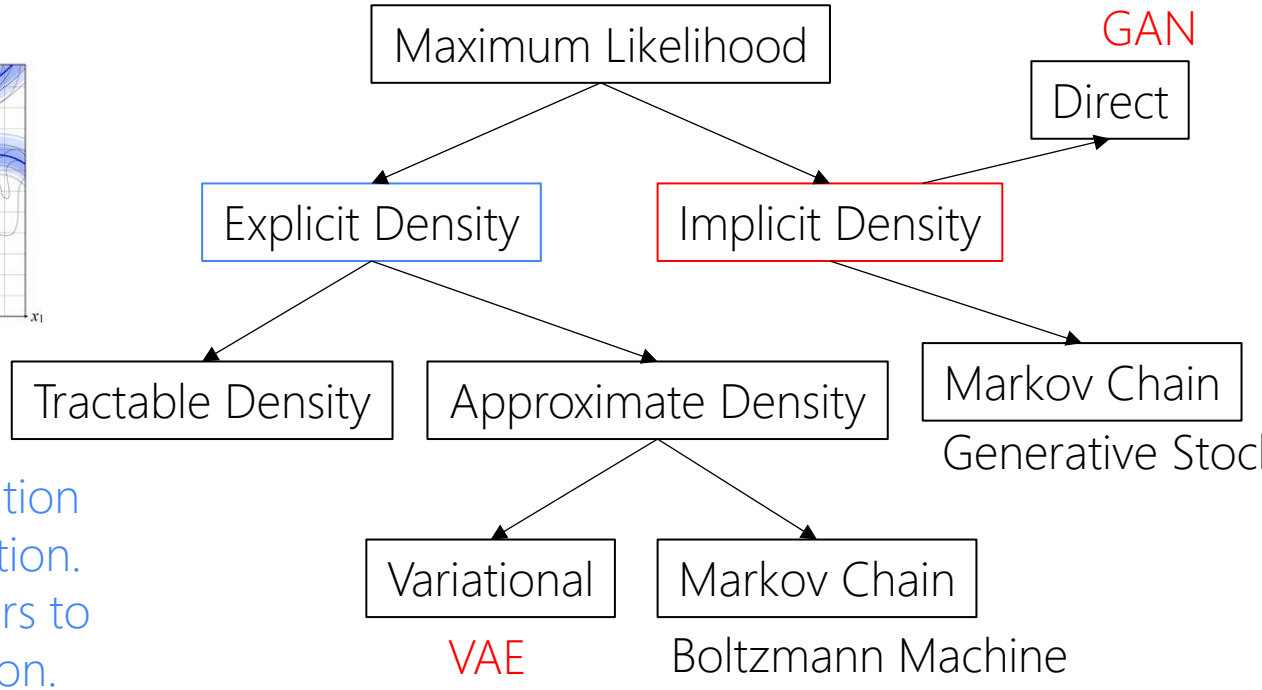


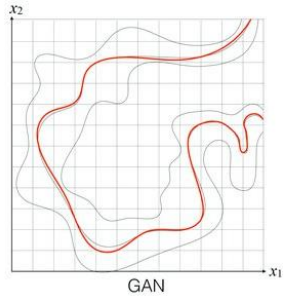
Modified from: https://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf

Recap: GAN and VAE

- Learning the distribution







Assume an explicit, parametric representation of the density distribution. Change the parameters to fit the target distribution.

Tune the model parameters directly so that the generated distribution looks similar to the target distribution.

Modified from: https://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf

Diffusion Models

- Consider a stochastic "diffusion" of an image:

Image credit: UC Berkeley, "Deep Learning for Visual Data"

Diffusion Models

- Consider a stochastic "diffusion" of an image:

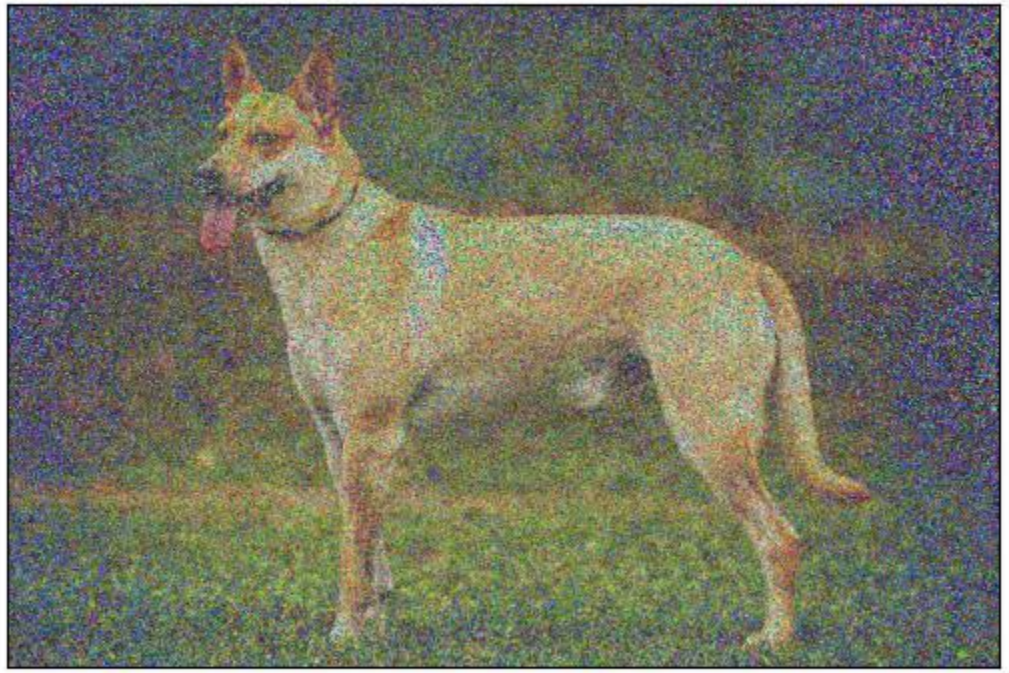


Image credit: UC Berkeley, "Deep Learning for Visual Data"

Diffusion Models

- Consider a stochastic "diffusion" of an image:

Image credit: UC Berkeley, "Deep Learning for Visual Data"

Diffusion Models

- Consider a stochastic "diffusion" of an image:

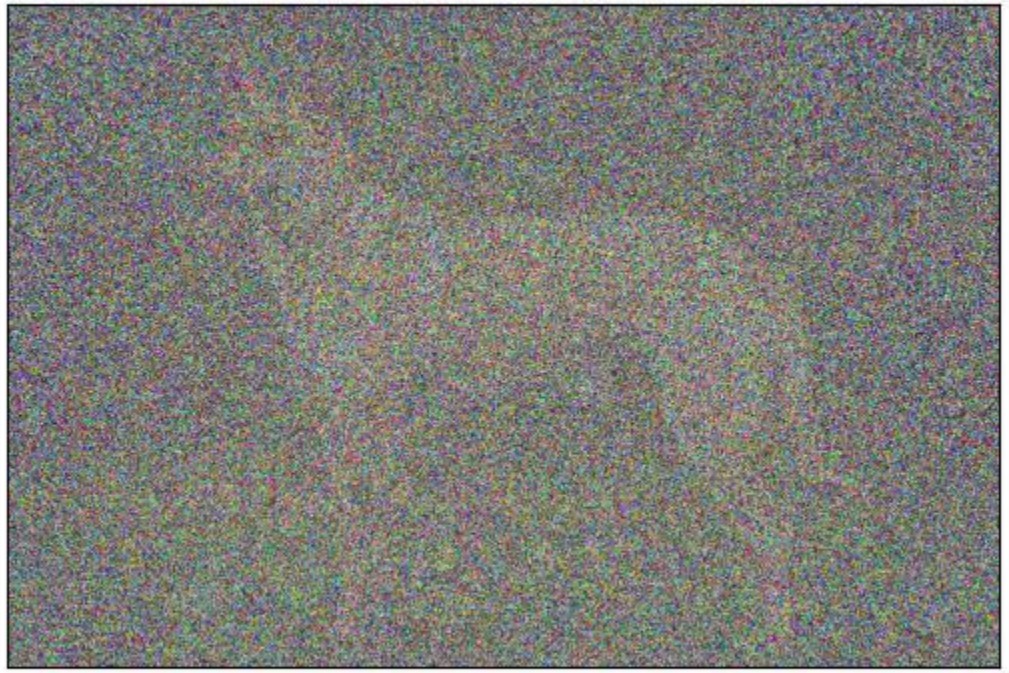


Image credit: UC Berkeley, "Deep Learning for Visual Data"

Diffusion Models

- Consider a stochastic "diffusion" of an image:

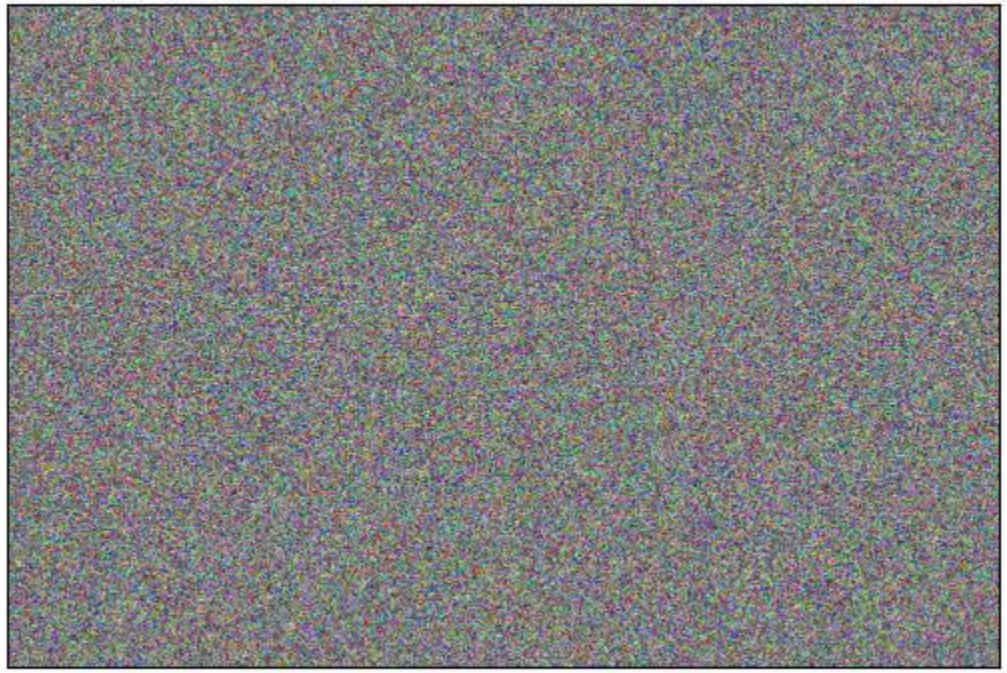


Image credit: UC Berkeley, "Deep Learning for Visual Data"

Diffusion Models

- Forward Process

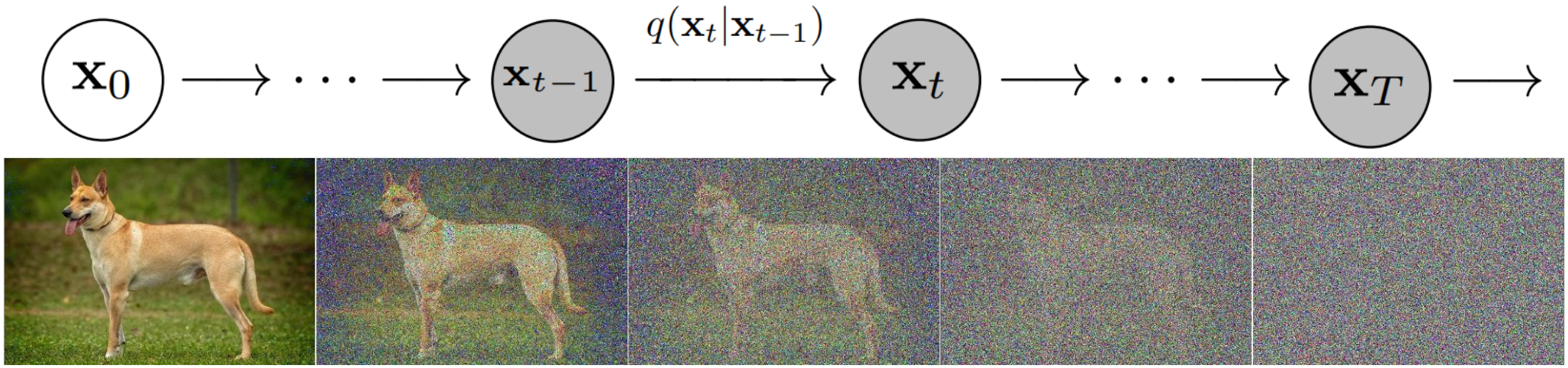


Image credit: UC Berkeley, "Deep Learning for Visual Data"

GAN, VAE, and Diffusion Models

GAN: Adversarial training

VAE: maximize variational lower bound

Diffusion models: Gradually add Gaussian noise and then reverse

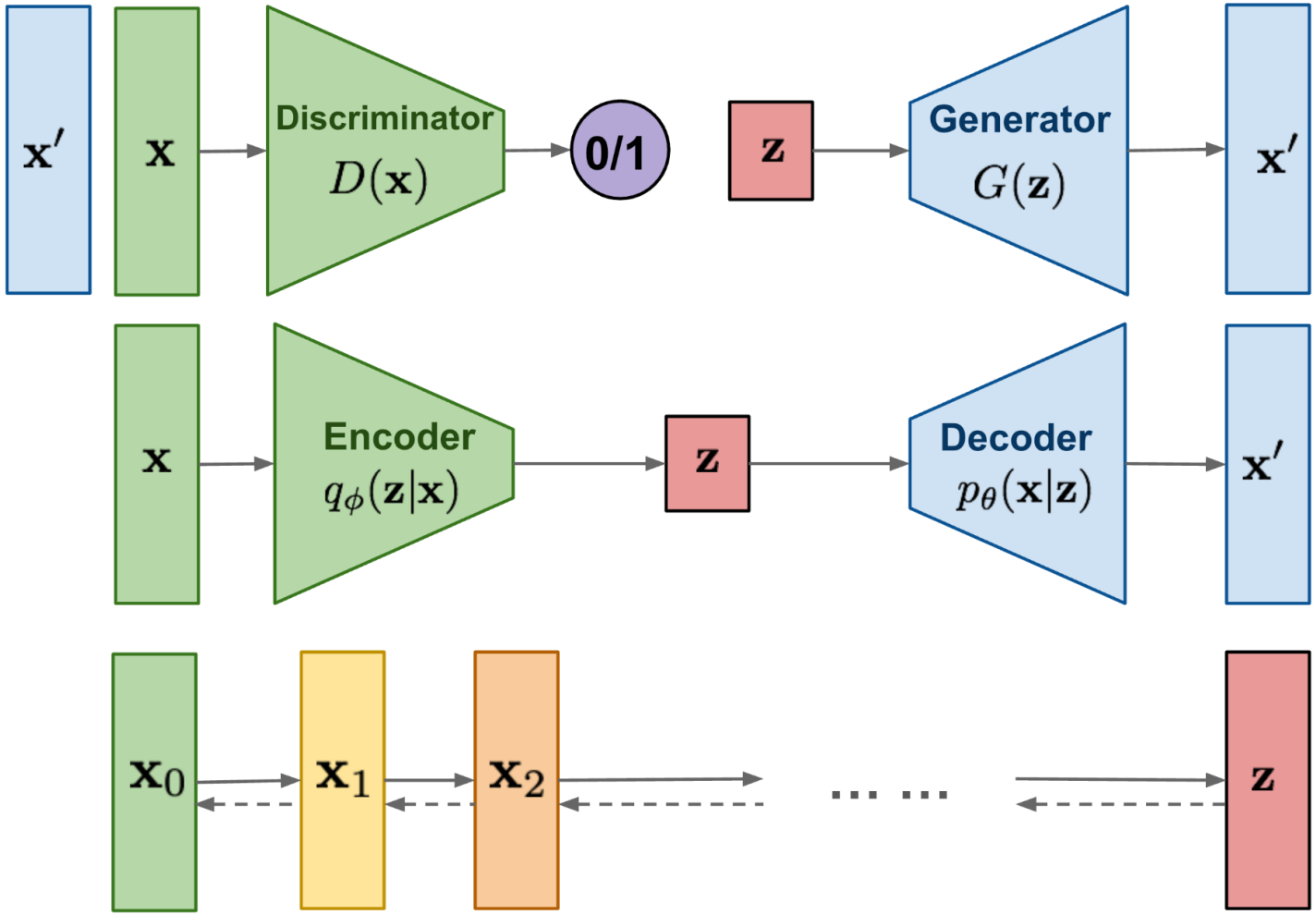


Image credit: https://github.com/yzy1996/Awesome-Generative-Model

Forward Process

- More formally:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon \qquad where \quad \epsilon \sim \mathcal{N}(0,1), \;\; \{\beta_t \in (0,1)\}_{t=1}^{T}$$

Forward Process

- More formally:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon \qquad where \quad \epsilon \sim \mathcal{N}(0,1), \;\; \{\beta_t \in (0,1)\}_{t=1}^{T}$$

"Noise schedule"
: a hyperparameter that determines how much noise is to be added at each time step.

Forward Process

- More formally:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon \qquad where \quad \epsilon \sim \mathcal{N}(0,1), \;\; \{\beta_t \in (0,1)\}_{t=1}^{T}$$

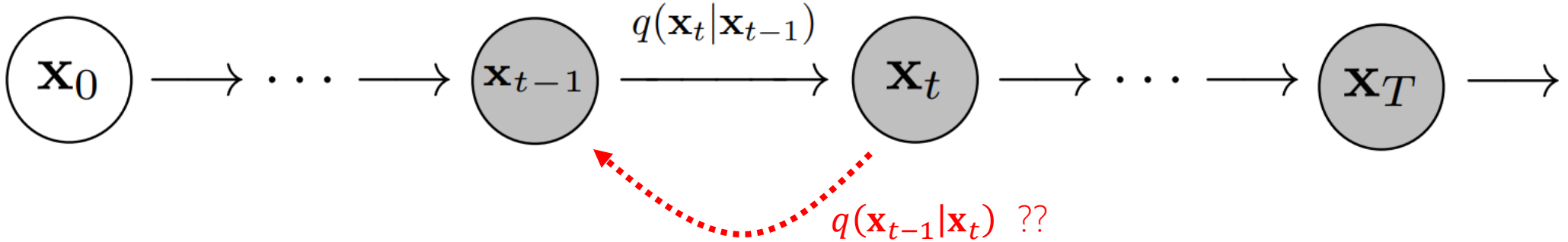
"Noise schedule"
: a hyperparameter that determines how much noise is to be added at each time step.

- Or equivalently:

$$q(x_t \mid x_{t-1}) \sim \mathcal{N}\left(\sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$$

Reverse Process

- Can we go in the other direction?



Fixed forward process

Reverse process ??

Reverse Process

- A naïve idea: the Bayes rule

$$q(x_{t-1}|x_t) = q(x_t|x_{t-1})\frac{q(x_{t-1})}{q(x_t)}$$

$$q(x_t) = \int q(x_t|x_{t-1})q(x_{t-1})dx$$

Reverse Process

- A naïve idea: the Bayes rule

$$q(x_{t-1}|x_t) = q(x_t|x_{t-1})\frac{q(x_{t-1})}{q(x_t)}$$

$$q(x_t) = \int q(x_t|x_{t-1})q(x_{t-1})dx$$

- Is this tractable?

Reverse Process

- A naïve idea: the Bayes rule

$$q(x_{t-1}|x_t) = q(x_t|x_{t-1})\frac{q(x_{t-1})}{q(x_t)}$$

$$q(x_t) = \int q(x_t|x_{t-1})q(x_{t-1})dx$$

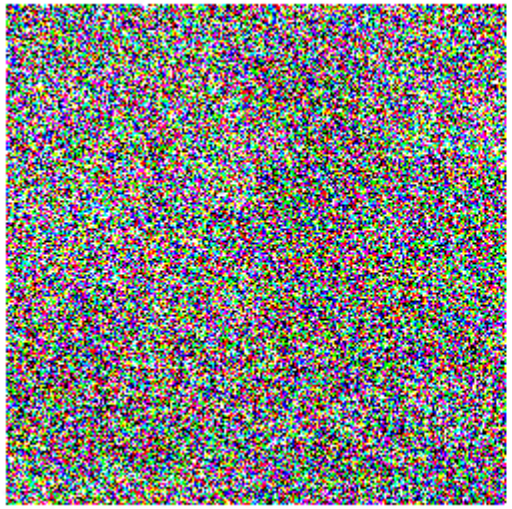
- Is this tractable?
  - The answer is of course no. Integration over the entire data distribution?!! ☹
  - A more viable solution instead would be to approximate the posterior directly.

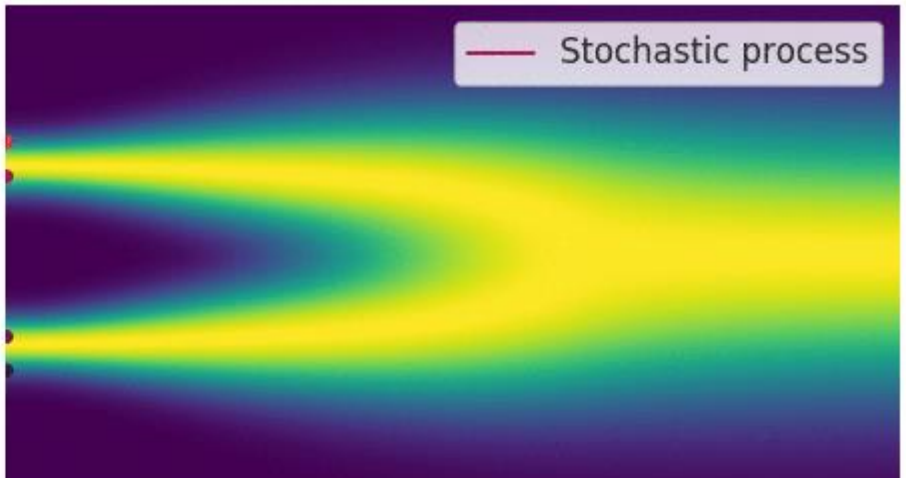
Reverse Process

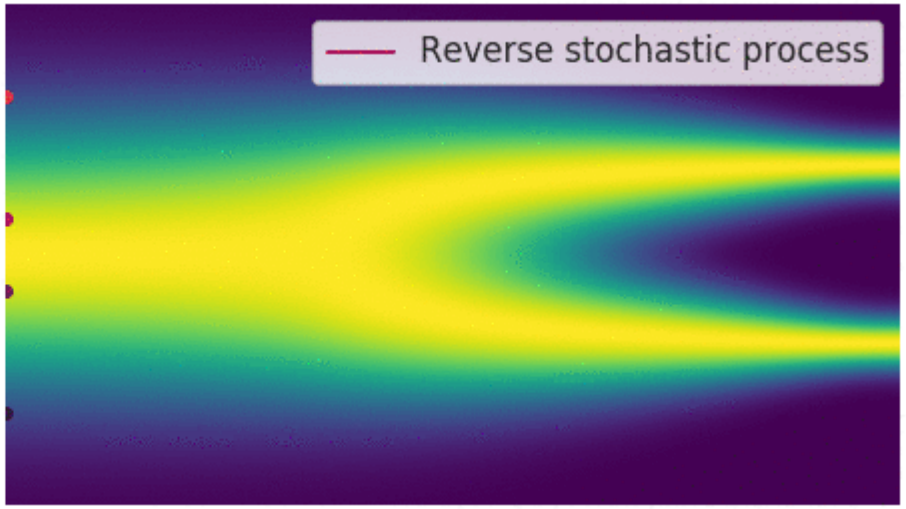
- Instead of finding the true posterior $q(x_{t-1}|x_t)$, we are now interested in finding an approximate $p_\theta(x_{t-1}|x_t)$ that is assumed to be Gaussian (assuming the noise is small):

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

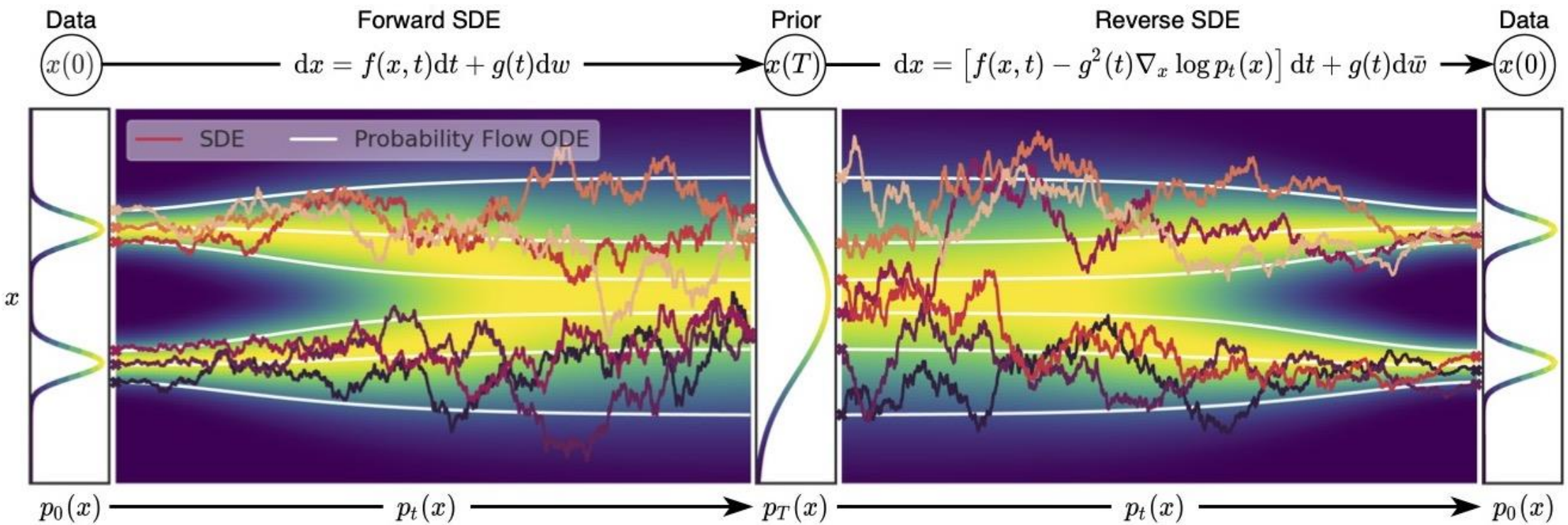
- Here, the mean μ_θ and the variance μ_θ can be learned via neural networks!







https://yang-song.net/blog/2021/score/



https://yang-song.net/blog/2021/score/

How do we train it?

- Goal: maximize the log-likelihood of data generated by a reverse process.

How do we train it?

- Goal: maximize the log-likelihood of data generated by a reverse process.

- More computationally tractable goal: Maximize the lower bound of the log-likelihood
  - Recall, ELBO or evidence lower bound, in case of VAE:

$$-L_{\mathrm{VAE}} = \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x})$$

How do we train it?

- Goal: maximize the log-likelihood of data generated by a reverse process.

- More computationally tractable goal: Maximize the lower bound of the log-likelihood
  - Recall, ELBO or evidence lower bound, in case of VAE:

$$-L_{\text{VAE}} = \log p_\theta(\mathbf{x}) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x})$$

  - Same trick for diffusion models:

$$-L = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] \leq \log p_\theta(\mathbf{x}_0)$$

How do we train it?

- Goal: maximize the log-likelihood of data generated by a reverse process.

- More computationally tractable goal: Maximize the lower bound of the log-likelihood
  - Recall, ELBO or evidence lower bound, in case of VAE:

$$-L_{\text{VAE}} = \log p_\theta(\mathbf{x}) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x})$$

  - Same trick for diffusion models:

$$-L = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] \leq \log p_\theta(\mathbf{x}_0)$$

Reconstruction term $p_\theta(x_0|x_1)$ & denoising steps $p_\theta(x_{t-1}|x_t)$, $t = 2, \ldots, T$

How close the diffused image is to the standard Gaussian (not trainable)

Simplified Objective

- Since we know the forward process:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0,\, I)$$

- … the denoising term in the ELBO can be expressed as (Ho et al. 2020):

$$\begin{aligned} L &= \mathbb{E}_{\mathbf{x}_0,\,\epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t,\, t)\|_2^2\right] \\ &= \mathbb{E}_{\mathbf{x}_0,\,\epsilon}\left[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon,\, t\right)\right\|_2^2\right] \end{aligned}$$

where

$$\alpha_t = 1 - \beta_t \quad \text{and} \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_t$$

Simplified Objective

- Since we know the forward process:

$$x_t = \sqrt{1-\beta_t} x_{t-1} + \sqrt{\beta_t}\epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, I)$$

- … the denoising term in the ELBO can be expressed as (Ho et al. 2020):

$$\begin{aligned} L &= \mathbb{E}_{\mathbf{x}_0, \epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|_2^2\right] \\ &= \mathbb{E}_{\mathbf{x}_0, \epsilon}\left[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)\right\|_2^2\right] \end{aligned}$$

where

$$\alpha_t = 1 - \beta_t \quad \text{and} \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_t$$

Main takeaway:
Instead of predicting the mean of x_{t-1}
predict the noise ϵ at each time step!

Training

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Choice of an Architecture

- Current diffusion models are essentially a U-Net on steroids…

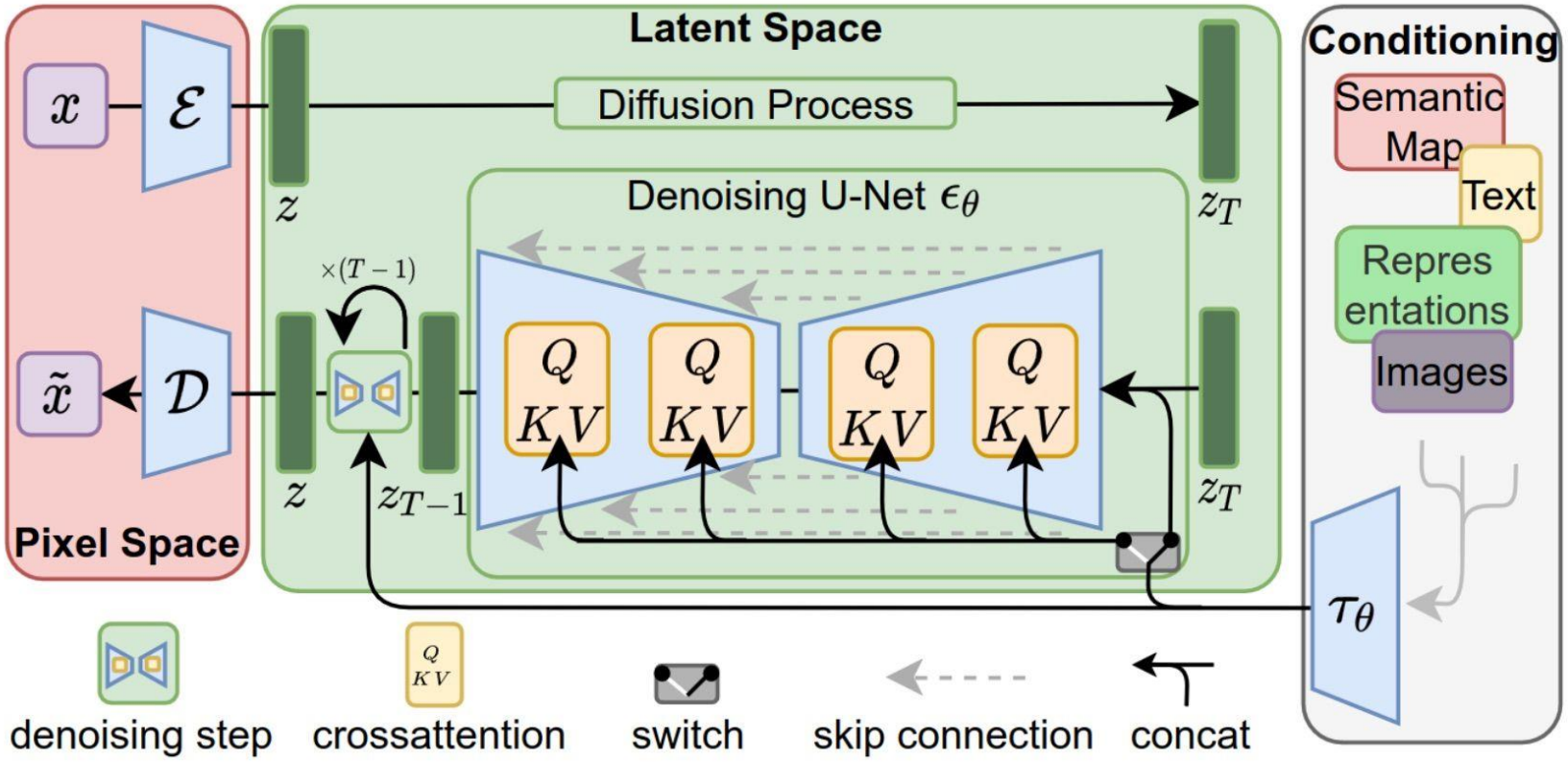
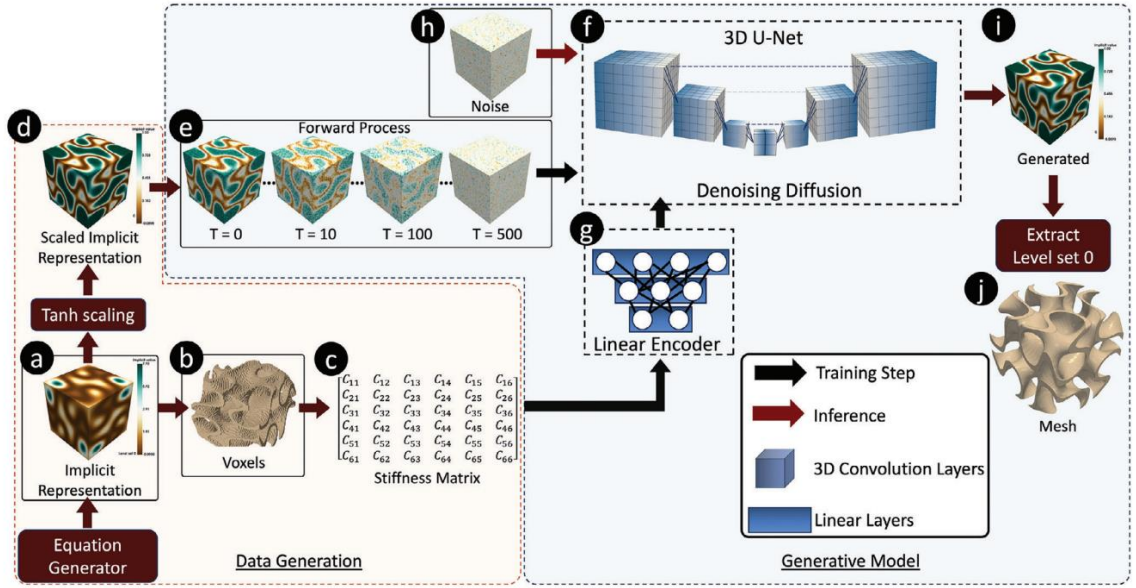
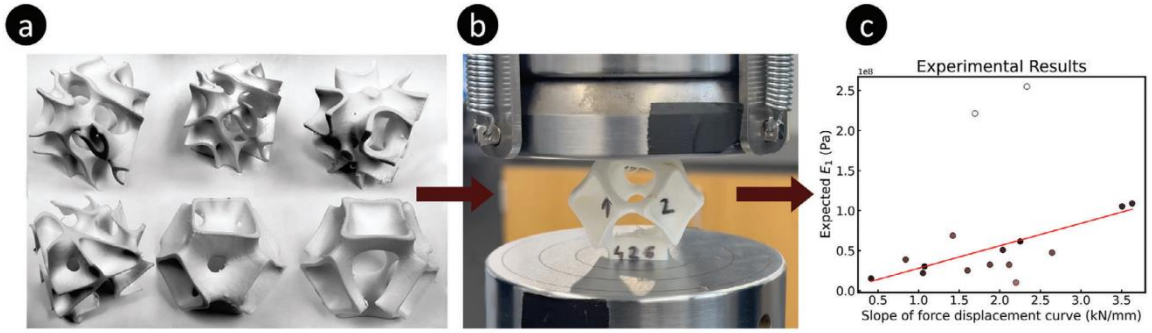


Image source: Stable diffusion paper

Applications: Structural Design





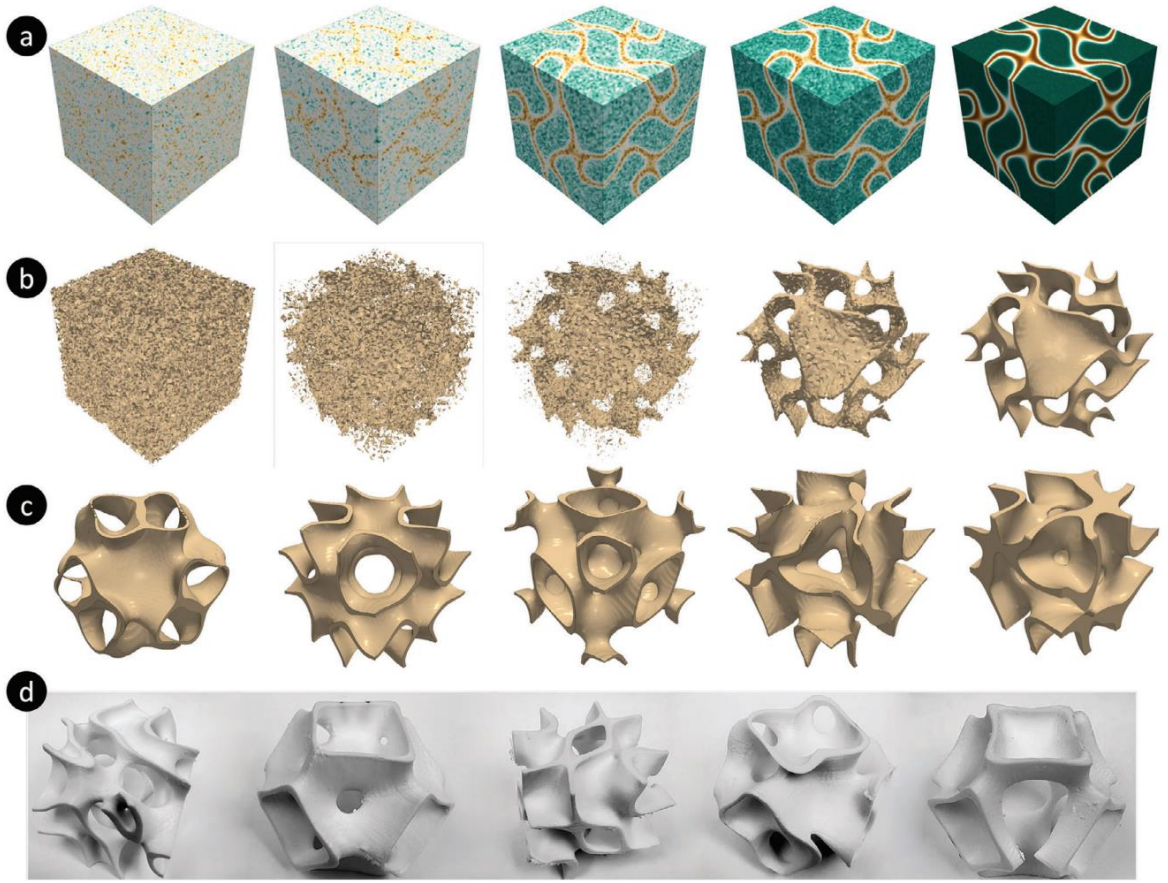
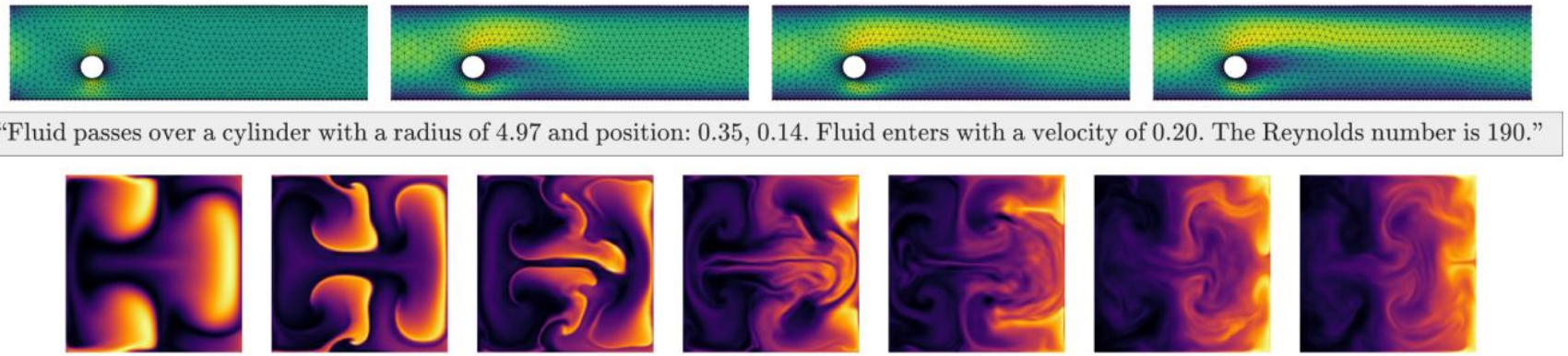


Figure 1. Generation process from noise to structure. Diffusion process converting Gaussian noise into the structure of a unit cell visualized as: a) implicit representation; b) mesh representation. c) Random generations from the proposed model. d) 3D printed lattice unit cells generated using the proposed framework.

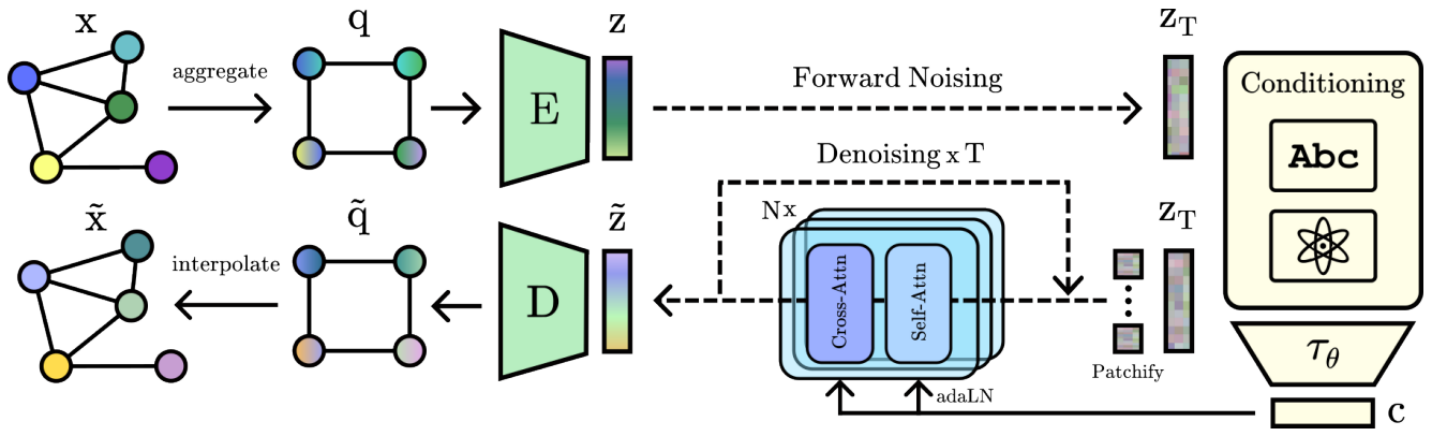
Jadhav, et al. (2024). Generative Lattice Units with 3D Diffusion for Inverse Design: GLU3D. Adv. Func. Mat., 2404165.

Applications: Text2PDE



"Fluid passes over a cylinder with a radius of 4.97 and position: 0.35, 0.14. Fluid enters with a velocity of 0.20. The Reynolds number is 190."

"Plume 1: Hook-like shape with a bulbous top and tapered bottom. Large, upper left corner, extending from top to center-left.
Plume 2: Elongated oval or pill-shape, slightly curved. Large, right side, extending from top to bottom
Plume 3: Hook-like shape, mirroring the first plume but inverted, Large, lower left corner, extending from bottom to center-left"



Zhou, et al. (2024). Text2pde: Latent diffusion models for accessible physics simulation. arXiv preprint arXiv:2410.01153.