

# Generative Models I

Jisang Han

[onground@korea.ac.kr](mailto:onground@korea.ac.kr)

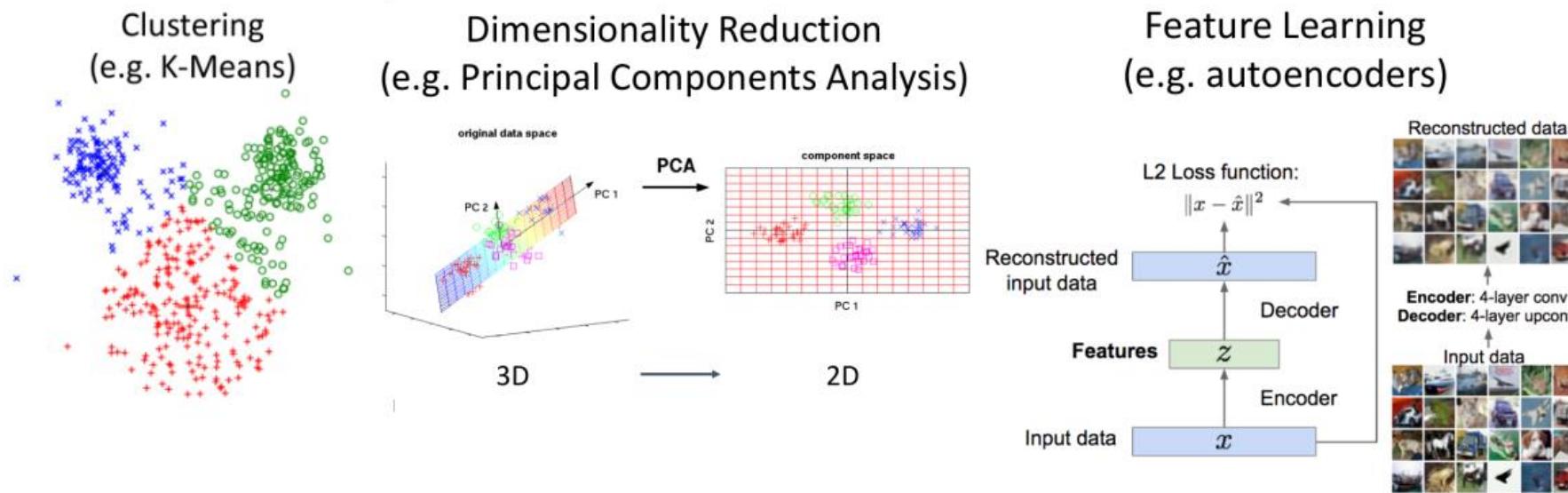
Artificial Intelligence in KU (AIKU)

Department of Computer Science and Engineering, Korea University

**AIKU**

# Supervised vs Unsupervised

	Supervised Learning	Unsupervised Learning
<b>Data</b>	$(x, y)$ : $x$ is data, $y$ is label	$x$
<b>Goal</b>	$x \rightarrow y$ 로 가는 function을 학습	Learn some underlying hidden structure of the data
<b>Examples</b>	Classification, regression, object detection, semantic segmentation, image captioning, etc.	Clustering, dimensionality reduction, feature learning, density estimation, etc.



# Discriminative vs Generative Models

---

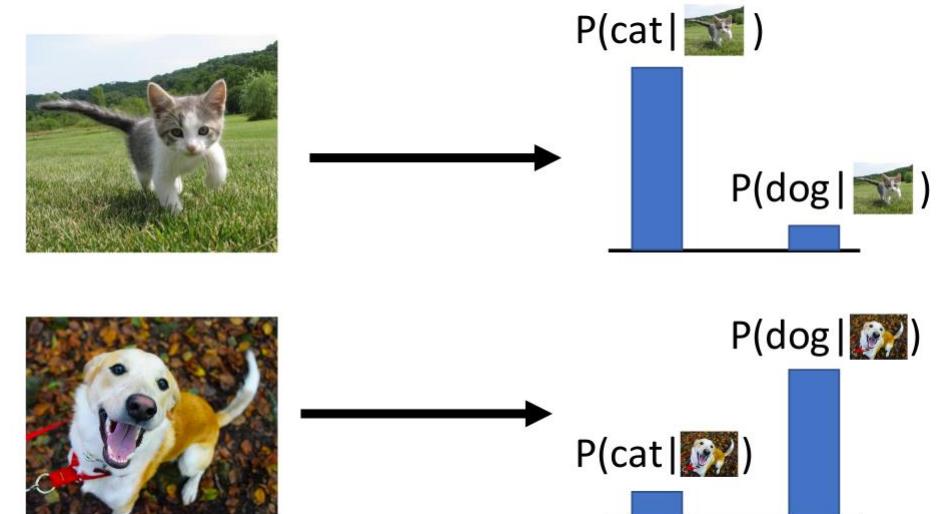
## Density Function

$$\int_X p(x) dx = 1$$

- Density Functions are normalized.
- Sum of probabilities are 1. If one is bigger then the other is smaller(compete each other).

# Discriminative vs Generative Models

## Discriminative Model

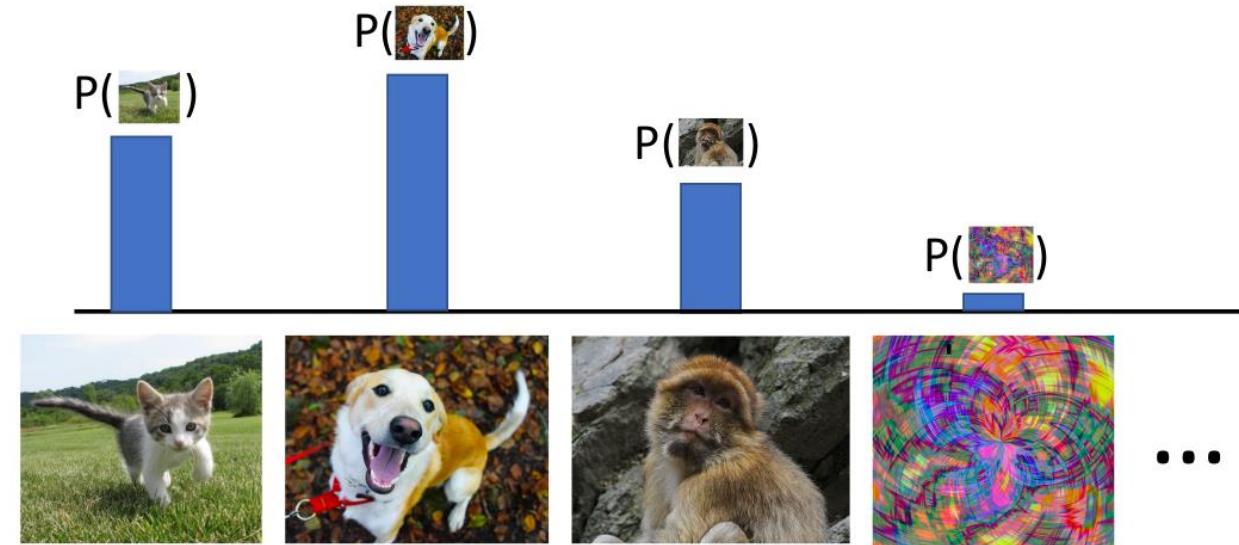


- Learn a probability distribution  $p(y|x)$  that predicts probability of the label  $y$  conditioned on the input image  $x$
- Input is  $x$  and Output is the probability. ( $x$ 에 대한 label이 나올 확률)
- The probability is calculated even if an input different from the specified labels is given.

# Discriminative vs Generative Models

---

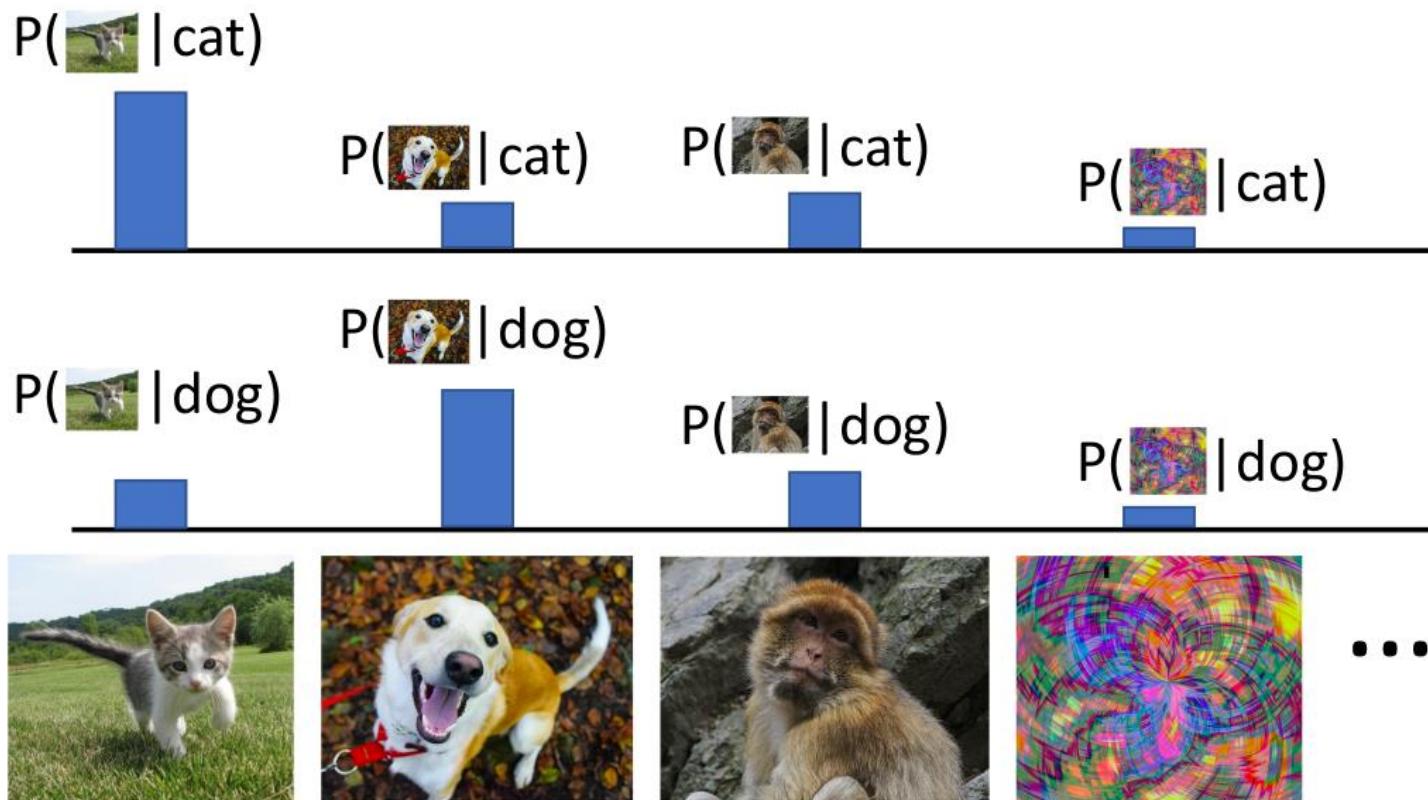
## Generative Model



- **Learn a probability distribution  $p(x)$**
- All possible images compete to the probability mass.
- A deep understanding of the image is needed. (Which is more plausible, sitting a dog or standing up?)
- The model can reject irrational inputs by giving them a very small value.

# Discriminative vs Generative Models

## Conditional Generative Model



Recall Bayes' Rule:

$$P(x | y) = \frac{P(y | x)}{P(y)} P(x)$$

Legend:  
Discriminative Model (Orange Box)  
(Unconditional) Generative Model (Purple Box)  
Conditional Generative Model (Blue Box)  
Prior over labels (Green Box)

- Learn  $p(x|y)$

# Discriminative vs Generative Models

---

## Discriminative Model:

Learn a probability distribution  $p(y|x)$



Assign labels to data  
Feature learning (supervised)

## Generative Model:

Learn a probability distribution  $p(x)$



Detect outliers  
Feature learning (unsupervised)  
Sample to **generate** new data

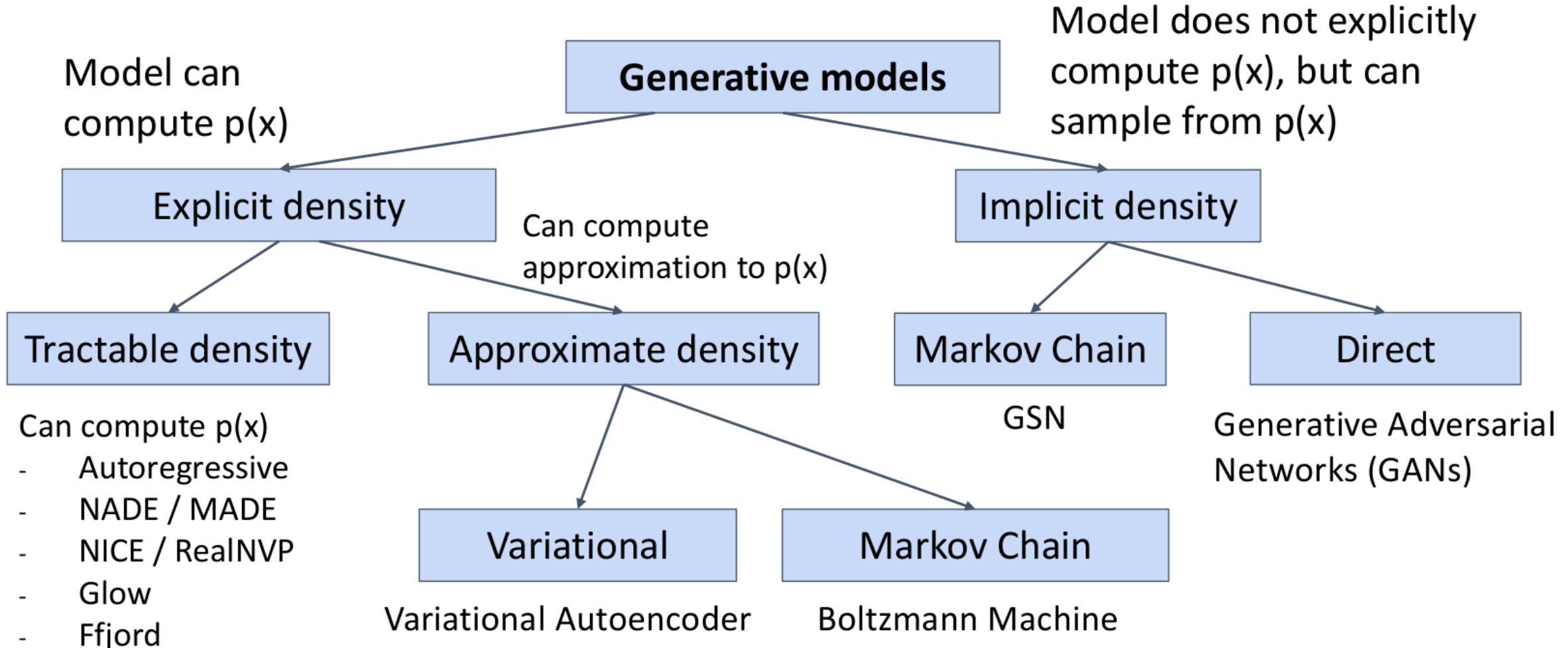
## Conditional Generative Model:

Learn  $p(x|y)$



Assign labels, while rejecting outliers!  
Generate new data conditioned on input labels

# Taxonomy of Generative Models



# Autoregressive Models

---

## Explicit Density Estimation

**Goal :** Write down an explicit function for  $p(x) = f(x, W)$  ( $x$ : data,  $W$ : learnable weight matrix)

If dataset is  $x^{(1)}, x^{(2)}, \dots, x^{(N)}$ ,

$$W^* = \arg \max_W \prod_i p(x^{(i)})$$

Maximize the probability of training data

$$= \arg \max_W \sum_i \log p(x^{(i)})$$

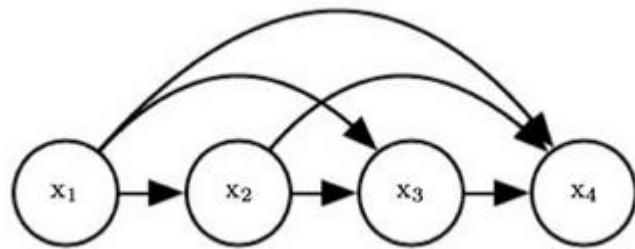
$$= \arg \max_W \sum_i \log f(x^{(i)}, W)$$

Loss Function. (Gradient Descent)

# Autoregressive Models

## Explicit Density: Autoregressive Models

- 자기 자신을 입력으로 하여 자기 자신을 예측하는 모형



- $x$ 가 여러 subparts로 이루어져 있다고 가정.  $x$ 가 이미지라고 하면 subparts는 각 픽셀이다  $x = (x_1, x_2, x_3, \dots, x_T)$
- Chain rule을 사용하여 probability를 계산한다.

$$\begin{aligned} p(x) &= p(x_1, x_2, \dots, x_T) \\ &= p(x_1) p(x_2|x_1) p(x_3|x_1, x_2) \\ &= \prod_{t=1}^T p(x_t|x_1, \dots, x_{t-1}) \end{aligned}$$

# Autoregressive Models

---

## Explicit Density: Autoregressive Models

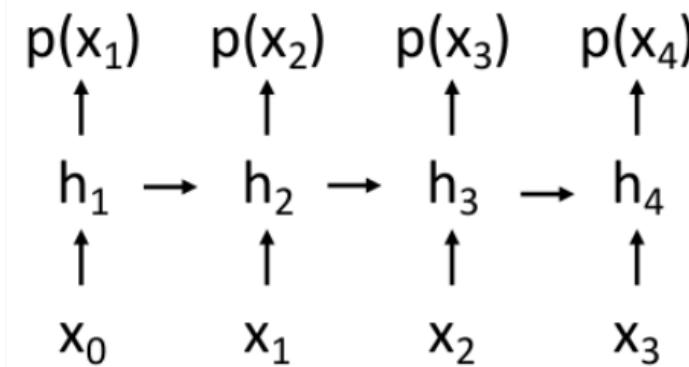
$$\begin{aligned} p(x) &= p(x_1, x_2, \dots, x_T) \\ &= p(x_1) p(x_2|x_1) p(x_3|x_1, x_2) \\ &= \prod_{t=1}^T p(x_t|x_1, \dots, x_{t-1}) \end{aligned}$$

- 즉, sequence의 확률  $p(x)$ 는 이전 sequence가 주어졌을 때 다음 sequence가 나올 확률을 전부 곱한 것이다.
- <https://wikidocs.net/22034>

# Autoregressive Models

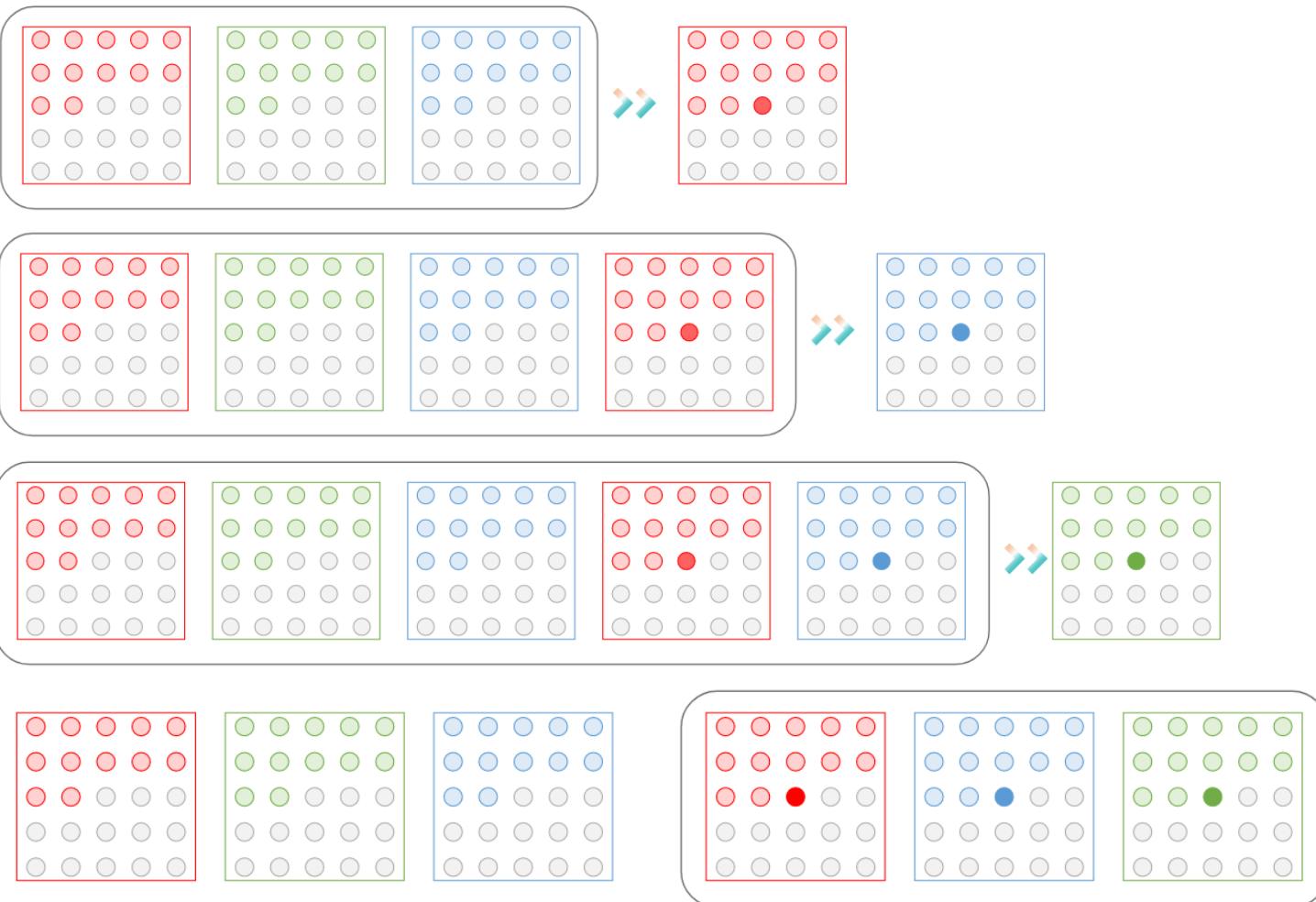
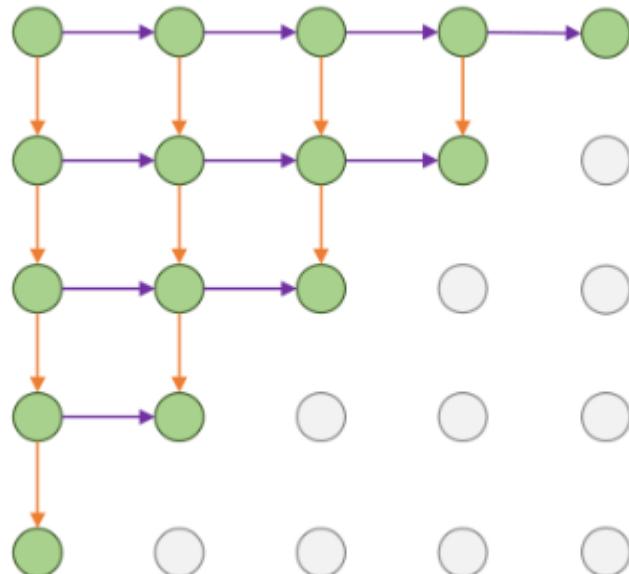
---

## Explicit Density: Autoregressive Models



# Autoregressive Models

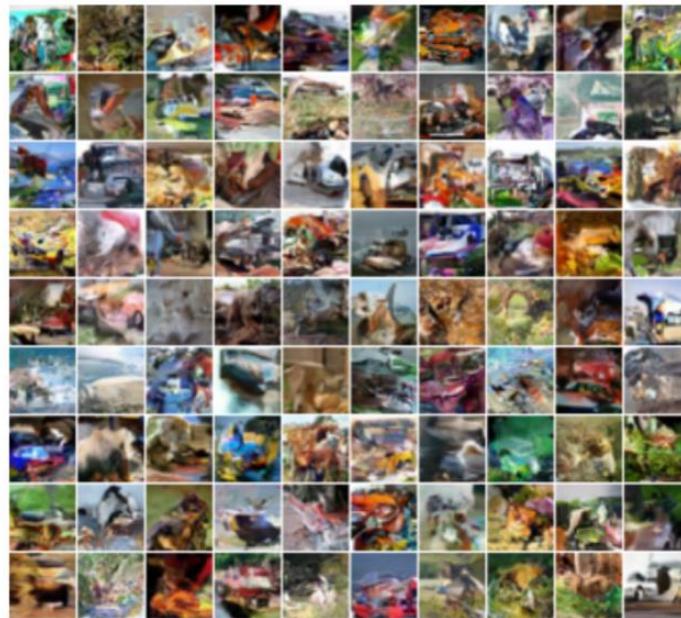
## PixelRNN



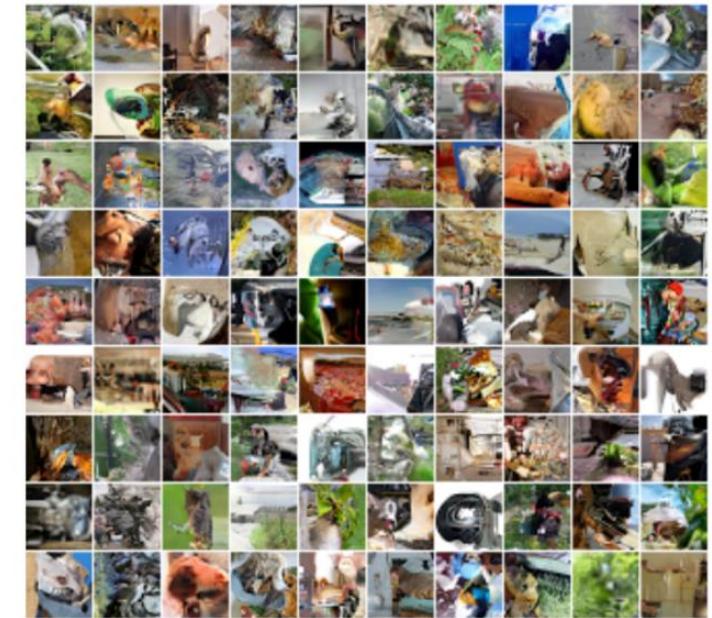
[http://dmqm.korea.ac.kr/uploads/seminar/20190705\\_Autoregressive.pdf](http://dmqm.korea.ac.kr/uploads/seminar/20190705_Autoregressive.pdf)

# Autoregressive Models

## PixelRNN



32x32 CIFAR-10



32x32 ImageNet

- 겉으로 보기에는 합리적으로 보이지만 사실 자세히 보면 거지같다..
- 그래도 edges, colors를 꽤나 그럴싸하게 생성하는 것에 의의를 가진다.
- 이는 unconditional generation이므로 test time에 내가 무엇을 생성하고 있는지 정할 수 없다

# Autoregressive Models

---

## Pros and Cons

### Pros

- Likelihood  $p(x)$ 를 명시적으로 계산할 수 있다.
- 위의 장점 덕분에 좋은 evaluation metric을 얻는다. (training data와 유사한, 즉 얼마나 그럴듯한지에 대한  $p(x)$  값을 얻을 수 있기 때문이다.)
- 나름 괜찮은 결과를 얻을 수 있다. (진짜 사진같지는 않지만..)

### Cons

- Sequential Generation 이므로 느리다.

# Variational Autoencoders (VAE)

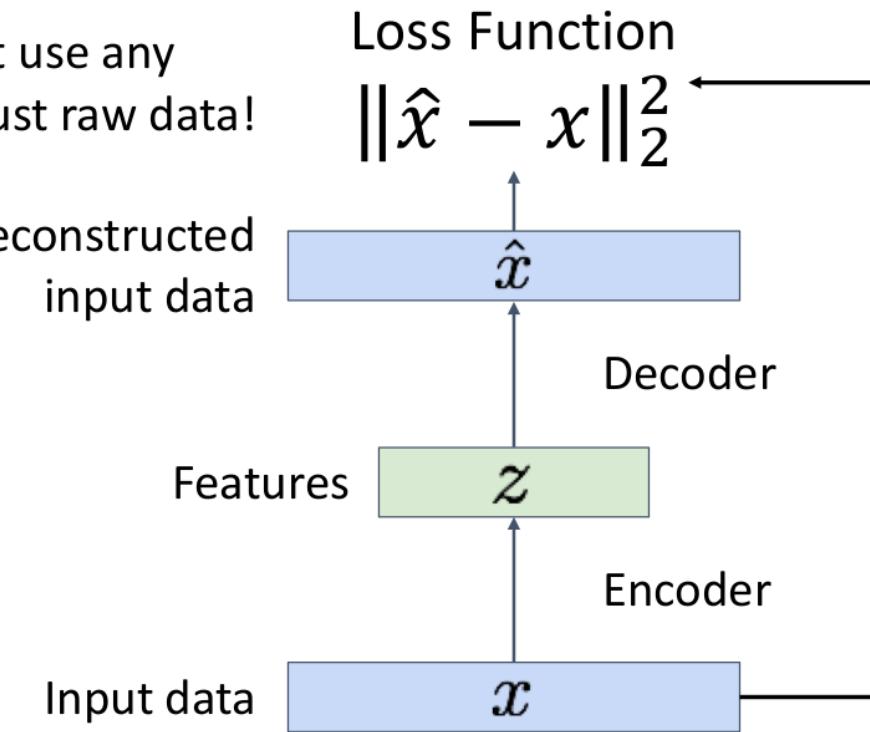
---

- In PixelRNN, PixelCNN, they defined parametric density function  $p(x) = f(x, W)$  and calculate for each input. And train the model to maximize this output.
- In VAE, Instead of maximizing the actual density value, **maximize the lower bound of the density.**
- $p_\theta(x) = \prod_{i=1}^n p_\theta(x_i | x_1, \dots, x_{i-1})$

# Variational Autoencoders (VAE)

## (Regular, non-variational) Autoencoders

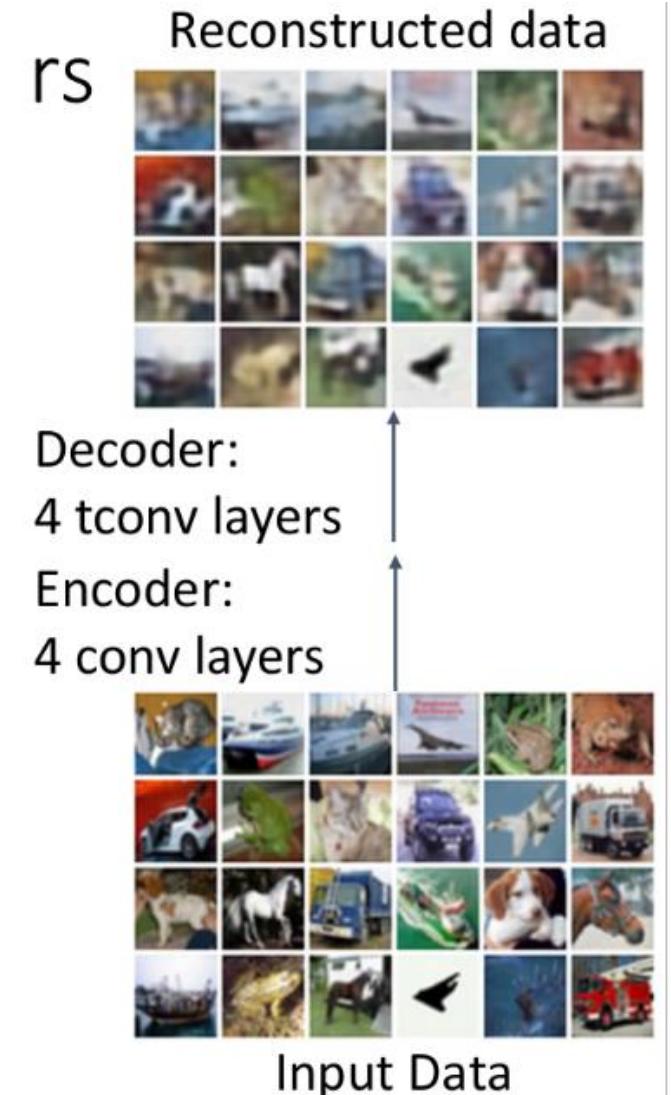
- Unsupervised method로, labels 없이 raw data  $x$ 로부터 feature vectors를 학습한다.  
(Unsupervised method for learning feature vectors from raw data  $x$ , without any labels)
  - Features** extracts useful information that can be used for downstream tasks.
  - Encoder** extracts features from input data,
  - Decoder** reconstruct the input data from the features.
  - Encoder:
    - Originally: Linear + nonlinearity (sigmoid)
    - Later: Deep, fully-connected
    - Later: ReLU CNN (upconv)
- Does not use any labels! Just raw data!
- Reconstructed input data
- Loss Function  
 $\|\hat{x} - x\|_2^2$
- $\hat{x}$
- Decoder
- $z$
- Features
- Encoder
- $x$
- Input data



# Variational Autoencoders (VAE)

## (Regular, non-variational) Autoencoders

- We expect the effect of **compressing input data through Encoder.**
- After learning, discard the decoder and use it for the downstream task using the encoder.
- **Not probabilistic** : 학습하지 않은 new data를 sampling할 수 없다.

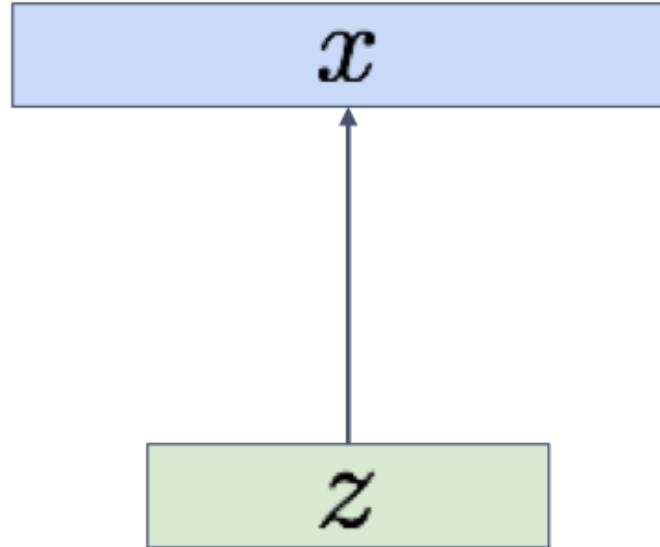


# Variational Autoencoders (VAE)

## Variational Autoencoders

Sample from  
conditional  
 $p_{\theta^*}(x \mid z^{(i)})$

Sample z  
from prior  
 $p_{\theta^*}(z)$



- Autoencoder에 확률 개념을 도입하였다
- 1. raw data로부터 **latent features z**를 학습한다.
- 2. new data를 생성하기 위해 model로부터 **sampling**한다.

# Variational Autoencoders (VAE)

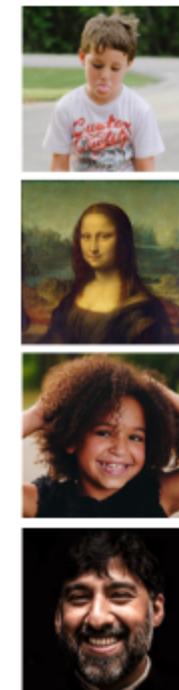
---

## Variational Autoencoders

- **Decoder:** Generating new data  $x$  from latent features  $z$  that is similar to input data but completely new.
- Sampling latent variables from prior distribution  $p_{\theta_*}(z)$ , put sampled  $z$  into the decoder to predict image  $x$
- At this time, output is not a single image but the distribution of images.
  
- $p_{\theta_*}(z)$  : prior distribution. PDF of  $x$ . (Gaussian distribution)
- $p_{\theta_*}(x|z^{(i)})$  : 주어진  $z$ 에서 특정  $x$ 가 나올 조건부 확률에 대한 PDF
- $\theta$  : parameter
- $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$  → cannot optimize directly, derive and optimize lower bound on likelihood instead.

# Variational Autoencoders (VAE)

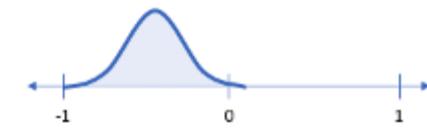
## Variational Autoencoders



Smile (discrete value)



Smile (probability distribution)

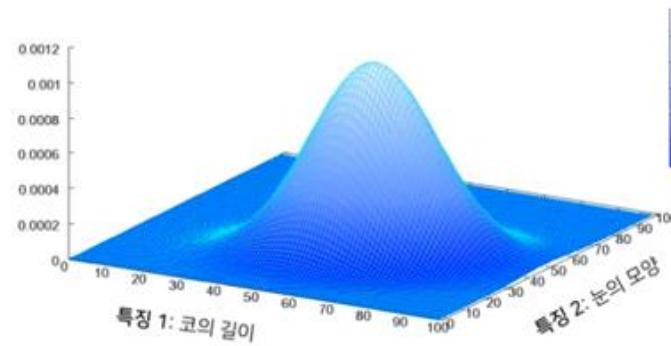


vs.

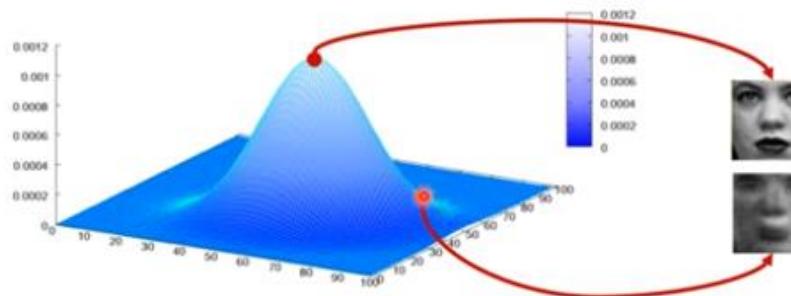


- VAE에서의  $z$ 는 AE에서의  $z$ (a value: low dimension of input data)와 다르게, 가우시안 확률분포에 기반한 확률값으로 나타낸다.
- input image가 들어오면 그 이미지의 다양한 특징들이 각각의 확률변수가 되는 확률분포를 만든다. 이 확률 분포를 잘 찾아내어 확률값이 높은 부분을 사용하면 그럴듯한 새로운 이미지를 생성할 수 있다.

# Variational Autoencoders (VAE)



입력 데이터의  
분포를 잘 근사하는 모델을 생성



- 이때 각 feature가 가우시안 분포를 따른다고 가정하고 latent z는 각 feature의 평균과 분산값을 나타낸다.
- 예를 들어 한국인의 얼굴을 그리기 위해 눈, 코, 입 등의 feature를 Latent vector z에 담고, 그 z를 이용해 그럴듯한 한국인의 얼굴을 그려내는 것이다. latent vector z는 한국인 눈 모양의 평균 및 분산, 한국인 코 길이의 평균 및 분산, 한국인 머리카락 길이의 평균 및 분산 등등의 정보를 담고 있다고 생각할 수 있다.

# Generative Models II

Jisang Han

[onground@korea.ac.kr](mailto:onground@korea.ac.kr)

Artificial Intelligence in KU (AIKU)

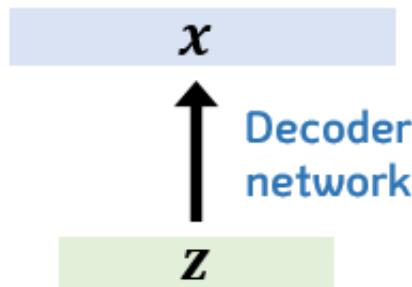
Department of Computer Science and Engineering, Korea University

**AIKU**

# Mathematical meaning

- If a model parameter  $\theta$  is given, the higher  $p_{\theta_*}(x)$  (The probability that the answer we want is  $x$ ) the better model.
- Train parameters to maximize  $p_{\theta_*}(x)$

Decoder



z latent variable의 확률분포

$$p_\theta(z)$$

z가 given일 때 x의 확률분포

$$p_\theta(x|z^{(i)})$$

어떻게 학습?

네트워크의 출력값이 있을 때  
우리가 원하는 정답  $x$ 가 나올 확률이 높길바람  
=  $x$ 의 likelihood를 최대화하는 확률분포 찾자

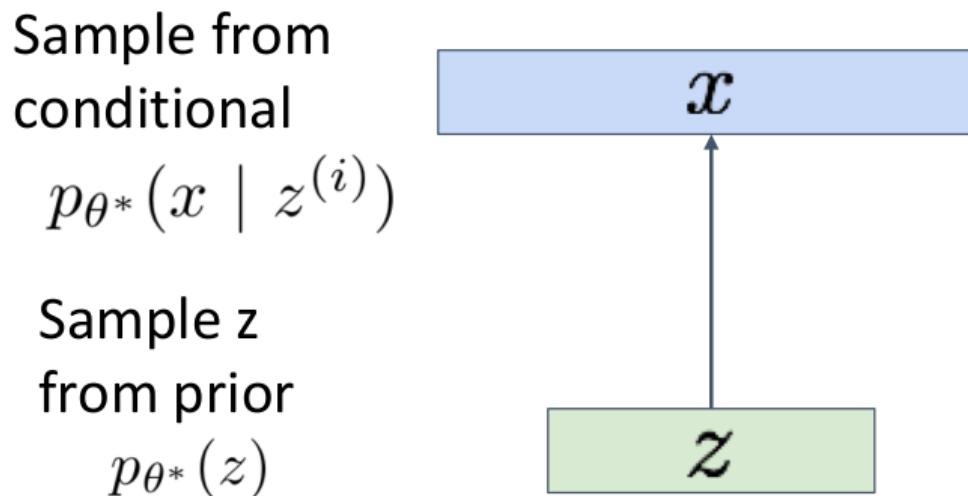


Maximize

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

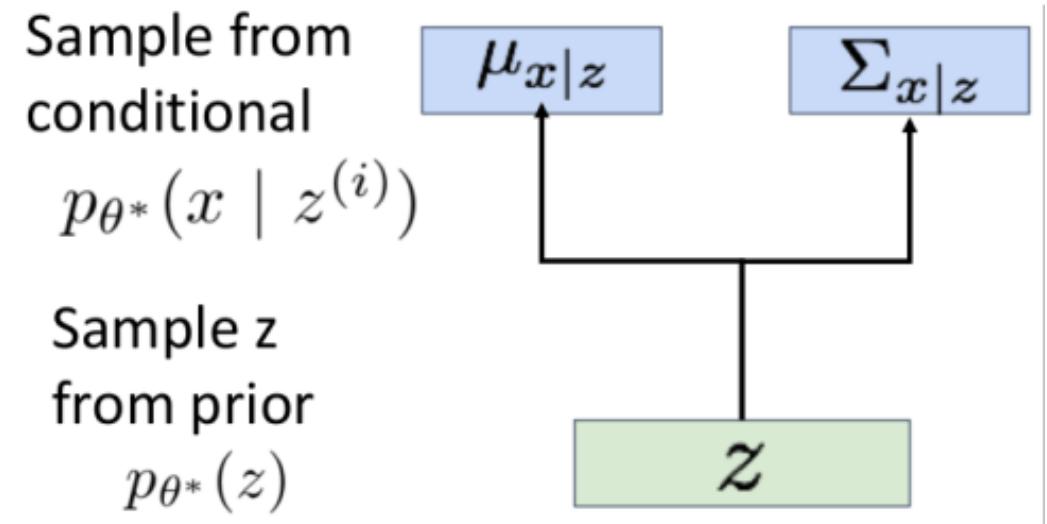
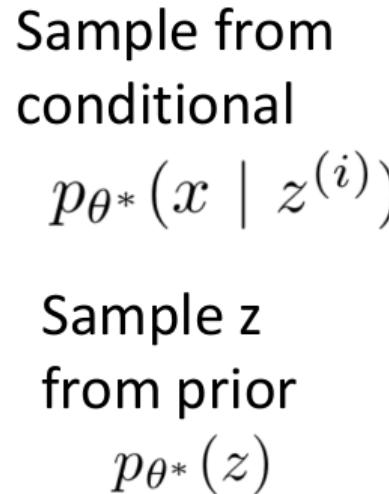
# Variational Autoencoders

---



- When image comes in,  $p(z)$  is the appropriate Gaussian distribution for each pixel of image.
- That is, each pixel has a Gaussian distribution with  $\mu, \sigma$
- But if it's a **high resolution** image, it's going to have a lot of values. Therefore, the **diagonal gaussian distribution** is used instead of the general Gaussian distribution. That is, when  $z$  is given, there is no covariance between pixels of the generated image. **Pixels are independent.**

# Variational Autoencoders



- **Decoder** outputs mean  $\mu_{x|z}$  and diagonal covariance  $\Sigma_{x|z}$  for the input  $z$
- Then sample  $x$  from the above Gaussian distribution

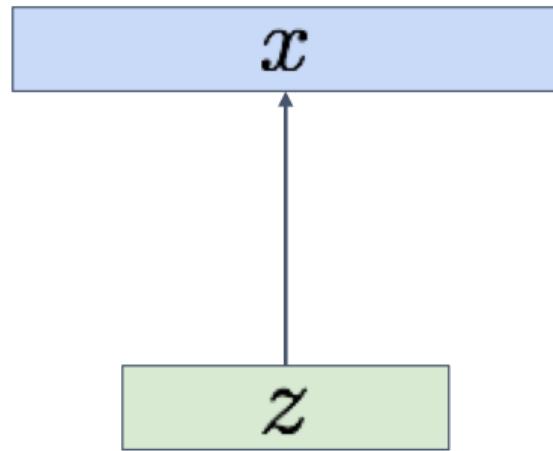
# Variational Autoencoders - Train

Sample from  
conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample  $z$   
from prior

$$p_{\theta^*}(z)$$

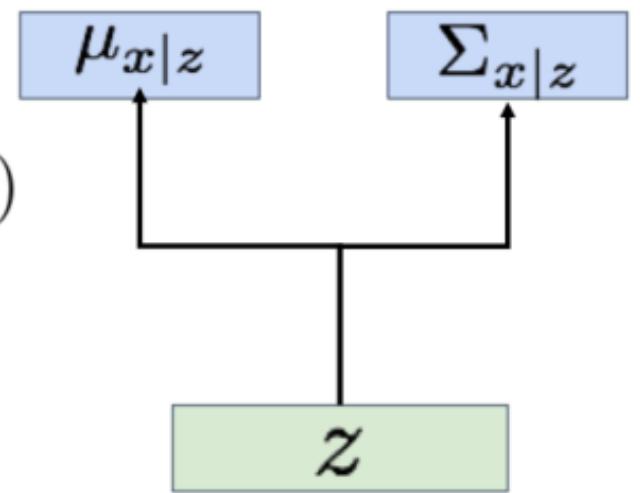


Sample from  
conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample  $z$   
from prior

$$p_{\theta^*}(z)$$



- **Maximize likelihood of data  $p_{\theta^*}(x)$**

$$p_{\theta}(x) = \frac{p_{\theta}(x|z)p_{\theta}(z)}{p_{\theta}(z|x)}$$

$$q_{\phi}(z|x) \approx p_{\theta}(z|x)$$

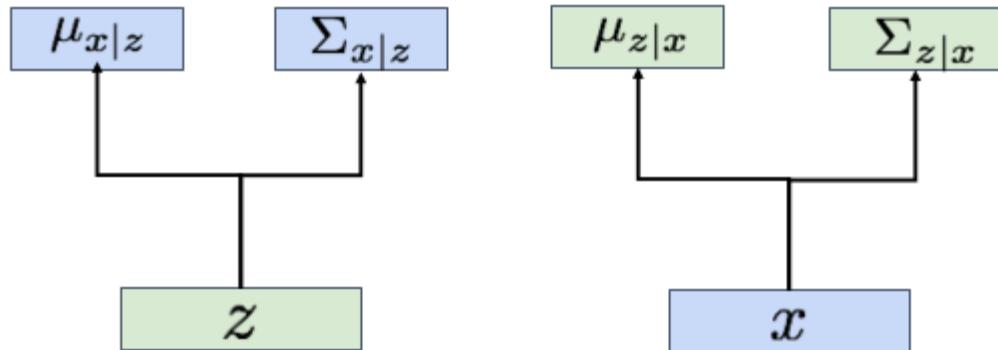
$$p_{\theta}(x) = \frac{p_{\theta}(x|z)p_{\theta}(z)}{p_{\theta}(z|x)} \approx \frac{p_{\theta}(x|z)p_{\theta}(z)}{q_{\phi}(z|x)}$$

# Variational Autoencoders - Train

**Decoder network** inputs  
latent code  $z$ , gives  
distribution over data  $x$

**Encoder network** inputs  
data  $x$ , gives distribution  
over latent codes  $z$

$$p_{\theta}(x | z) = N(\mu_{x|z}, \Sigma_{x|z}) \quad q_{\phi}(z | x) = N(\mu_{z|x}, \Sigma_{z|x})$$



$$\log p_{\theta}(x) = \log \frac{p_{\theta}(x|z)p(z)}{p_{\theta}(z|x)}$$

$$= \log \frac{p_{\theta}(x|z)p(z)q_{\phi}(z|x)}{p_{\theta}(z|x)q_{\phi}(z|x)}$$

$$= \log p_{\theta}(x|z) - \log \frac{q_{\phi}(z|x)}{p(z)} + \log \frac{q_{\phi}(z|x)}{p_{\theta}(z|x)}$$

wrap in an expectation since it doesn't depend on  $z$

$$\log p_{\theta}(x) = E_{z \sim q_{\phi}(z|x)}[\log p_{\theta}(x)]$$

$$= E_z[\log p_{\theta}(x|z)] - E_z \left[ \log \frac{q_{\phi}(z|x)}{p(z)} \right] + E_z \left[ \log \frac{q_{\phi}(z|x)}{p_{\theta}(z|x)} \right]$$

$$= E_{z \sim q_{\phi}(z|x)}[\log p_{\theta}(x|z)] - D_{KL}(q_{\phi}(z|x), p(z)) + D_{KL}(q_{\phi}(z|x), p_{\theta}(z|x))$$

# Variational Autoencoders - Train

---

$$\log p_\theta(x) = E_{z \sim q_\phi(z|x)} [\log p_\theta(x)]$$

$$= E_z [\log p_\theta(x|z)] - E_z \left[ \log \frac{q_\phi(z|x)}{p(z)} \right] + E_z \left[ \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \right]$$

$$= E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z)) + D_{KL}(q_\phi(z|x), p_\theta(z|x))$$

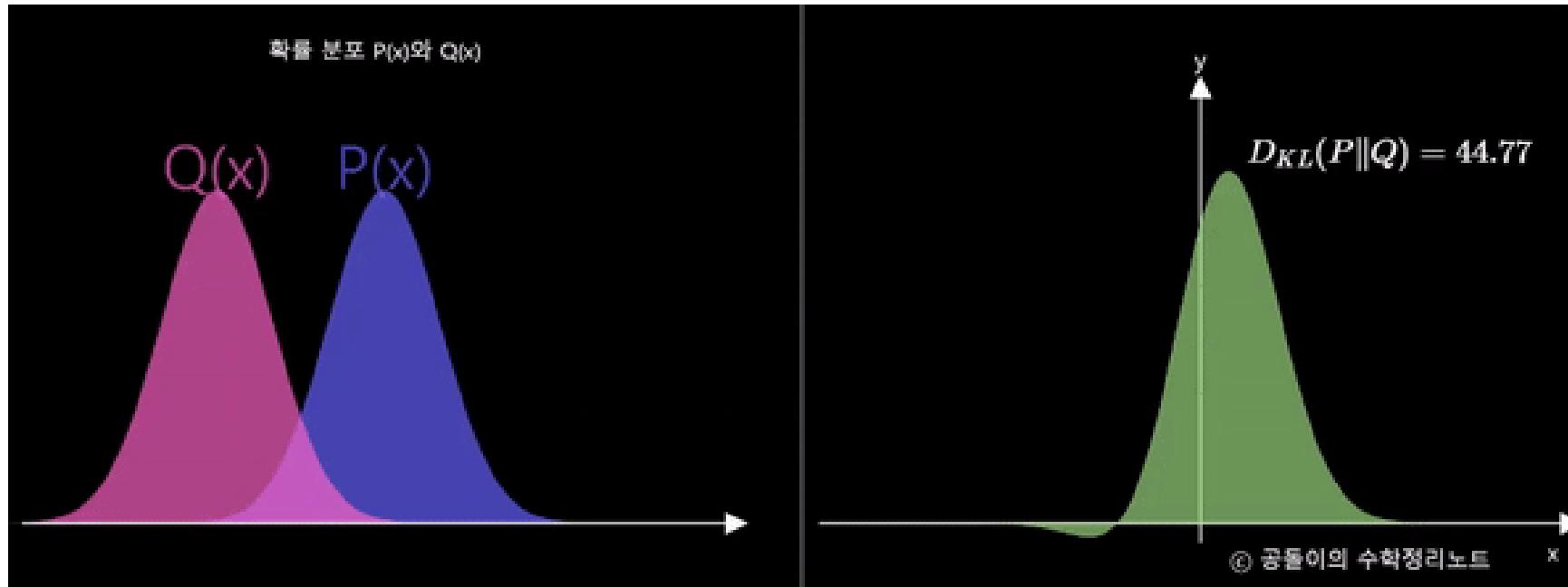
$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

Lower bound of likelihood

Through this, encoders and decoders are learned jointly to maximize the variable lower bound of data-like hood.

# Variational Autoencoders - Train

---



# Variational Autoencoders

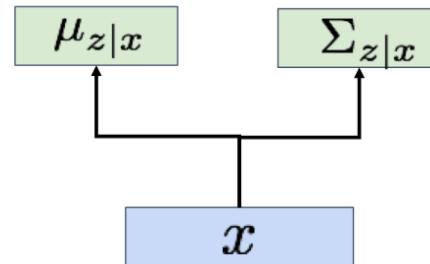
---

Jointly train **encoder** q and **decoder** p to maximize the **variational lower bound** on the data likelihood

$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

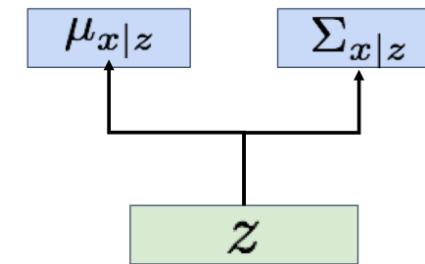
Encoder Network

$$q_\phi(z | x) = N(\mu_{z|x}, \Sigma_{z|x})$$



Decoder Network

$$p_\theta(x | z) = N(\mu_{x|z}, \Sigma_{x|z})$$



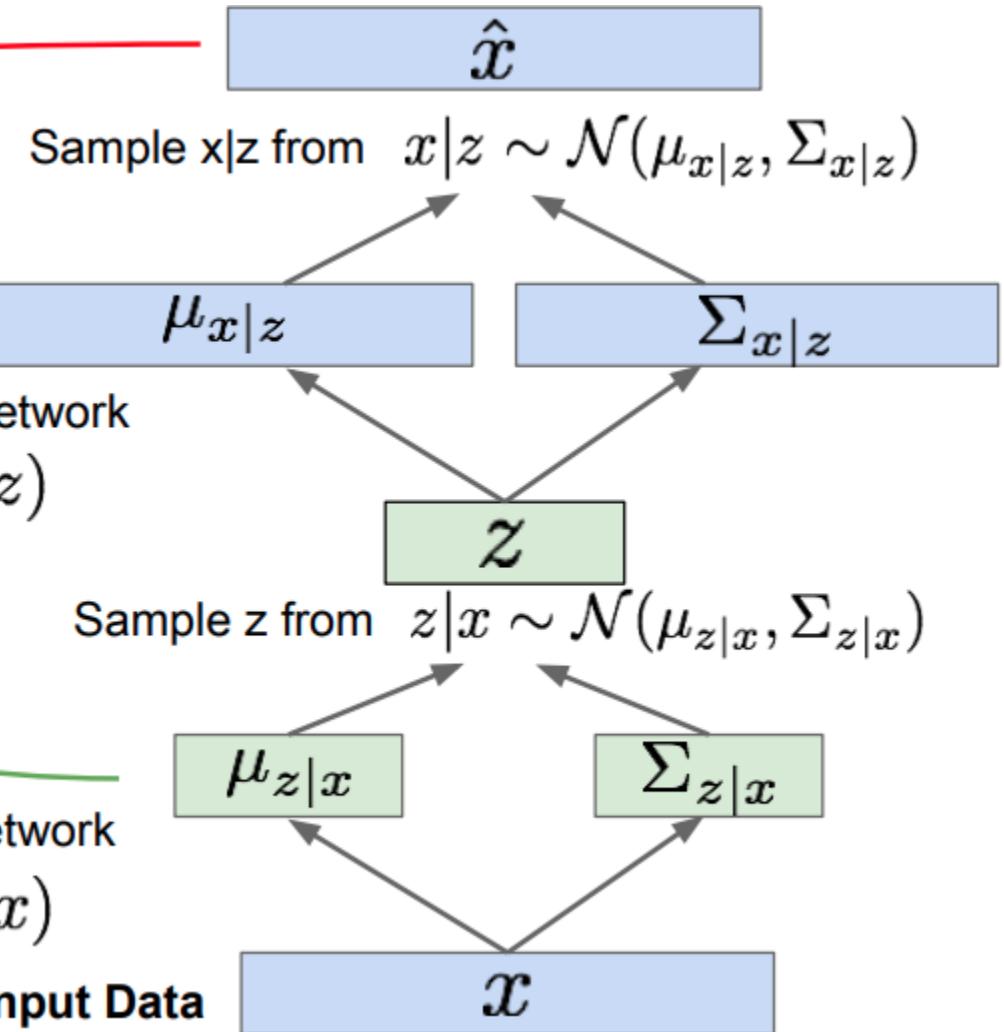
# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed

Decoder network  
 $p_\theta(x|z)$



For every minibatch of input data: compute this forward pass, and then backprop!

# Example : Fully connected VAE

MNIST Dataset

- $x = 28 \times 28$  image, flattened to 784-dim vector
- $z = 20$ -dim vector (hyper-parameter)

**Encoder Network**

$$q_{\phi}(z | x) = N(\mu_{z|x}, \Sigma_{z|x})$$

```
graph TD; x[x: 784] --> L1[Linear(784->400)]; L1 --> L2[Linear(400->20)]; L2 --> mu["μz|x: 20"]; L1 --> L3[Linear(400->20)]; L3 --> sigma["Σz|x: 20"]
```

**Decoder Network**

$$p_{\theta}(x | z) = N(\mu_{x|z}, \Sigma_{x|z})$$

```
graph TD; z[z: 20] --> L1[Linear(20->400)]; L1 --> L2[Linear(400->768)]; L2 --> mu_x["μx|z: 768"]; L1 --> L3[Linear(400->768)]; L3 --> sigma_x["Σx|z: 768"]
```

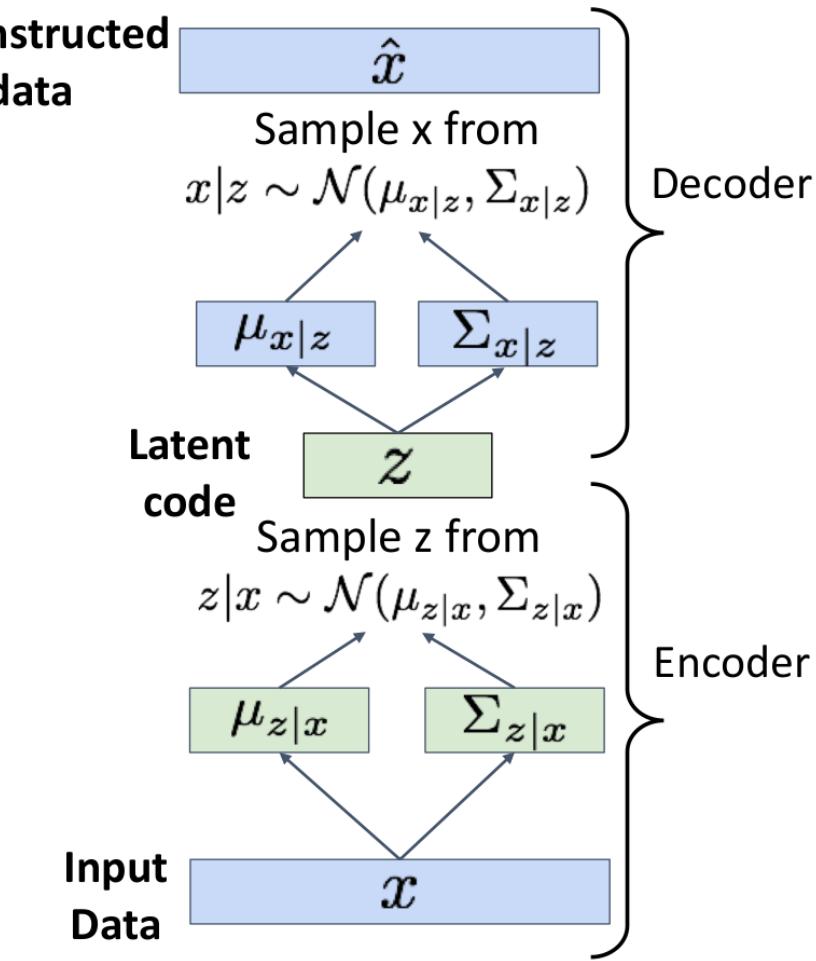
# Example : Fully connected VAE

## Variational Autoencoders

Train by maximizing the  
**variational lower bound**

$$E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior  $p(z)$ !**
3. Sample code  $z$  from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples
5. **Original input data should be likely under the distribution output from (4)!**
6. Can sample a reconstruction from (4)



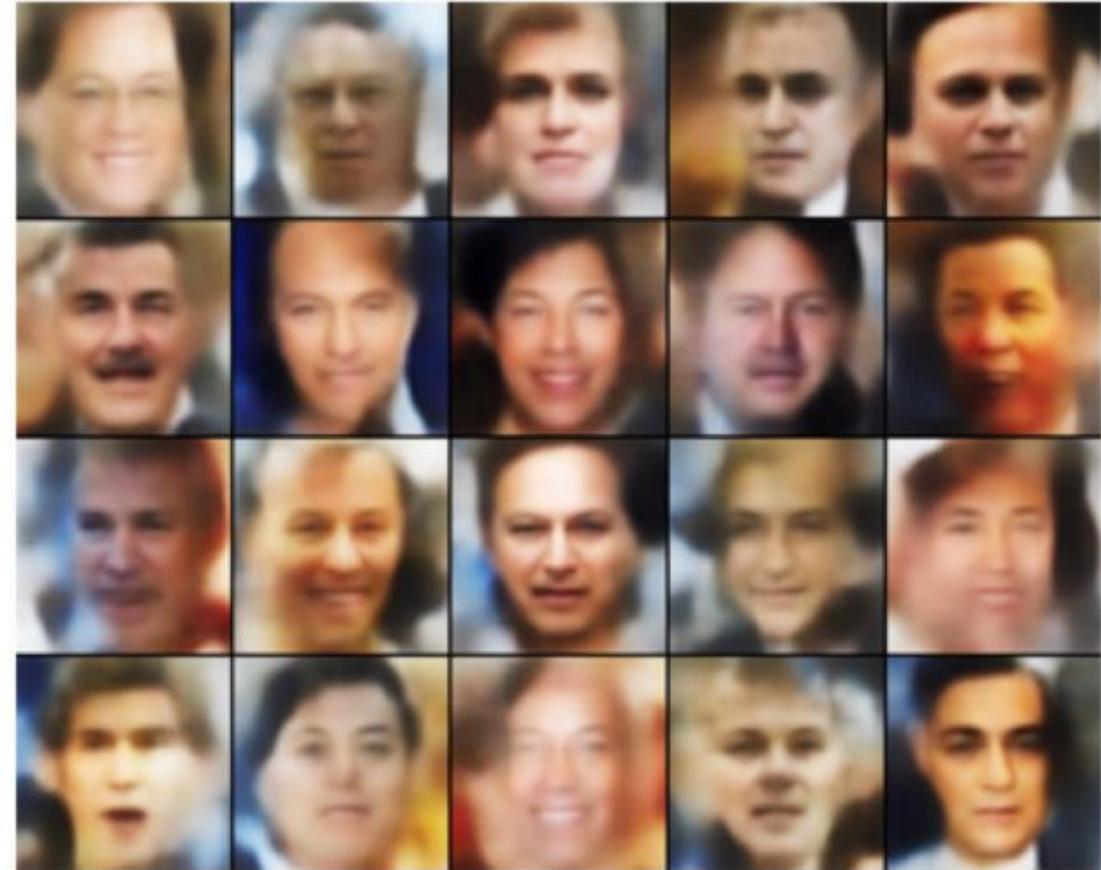
# Variational Autoencoders

---

32x32 CIFAR-10



Labeled Faces in the Wild



# Variational Autoencoders

- VAE : Editing with z

## Variational Autoencoders

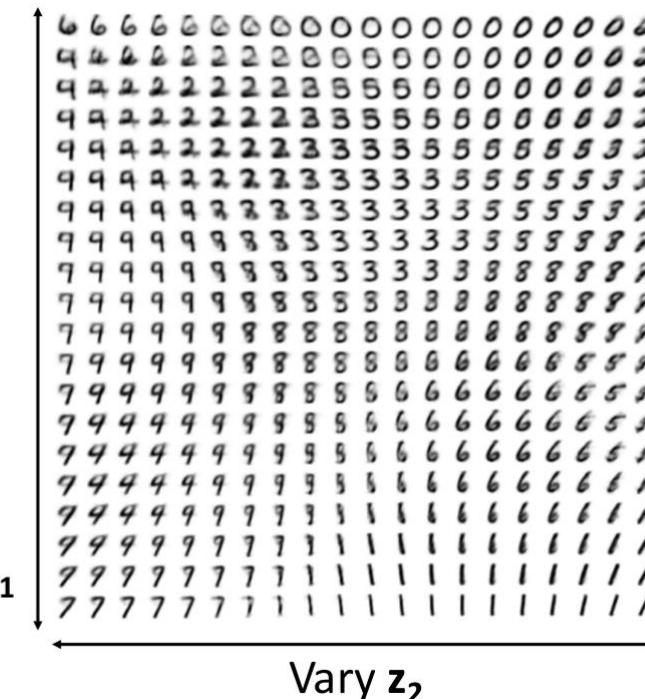
The diagonal prior on  $p(z)$  causes dimensions of  $z$  to be independent

## “Disentangling factors of variation”

Vary  $z_1$

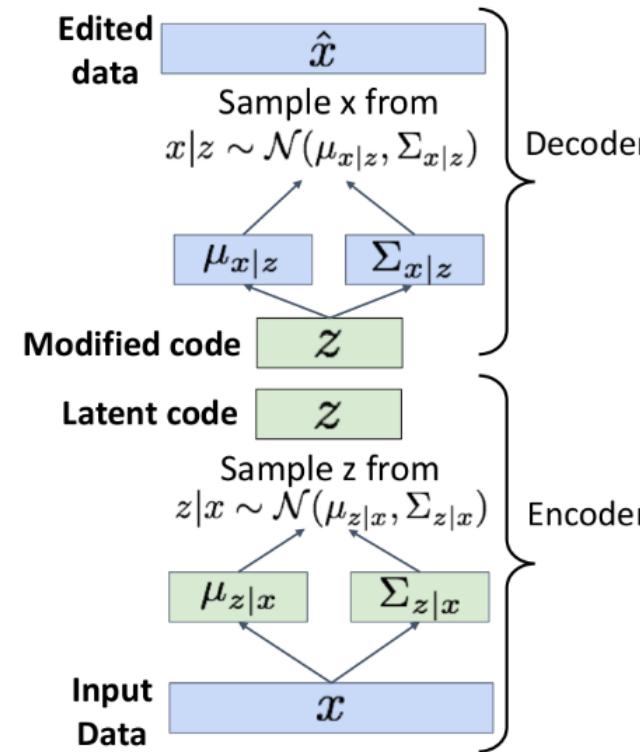
## Vary $z_2$

Kingma and Welling, Auto-Encoding Variational Bayes, ICLR 2014



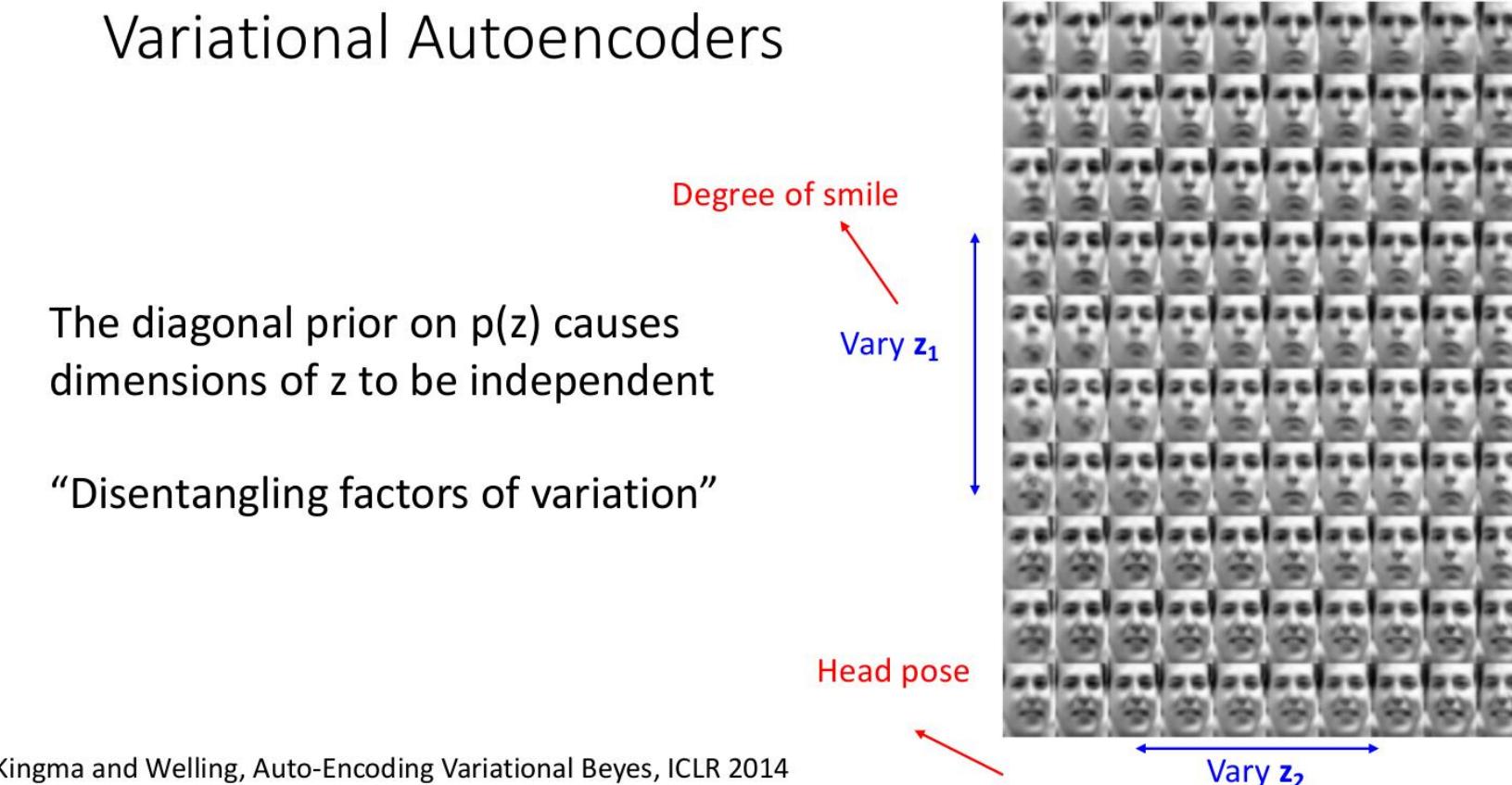
# Variational Autoencoders

- VAE : Editing with z



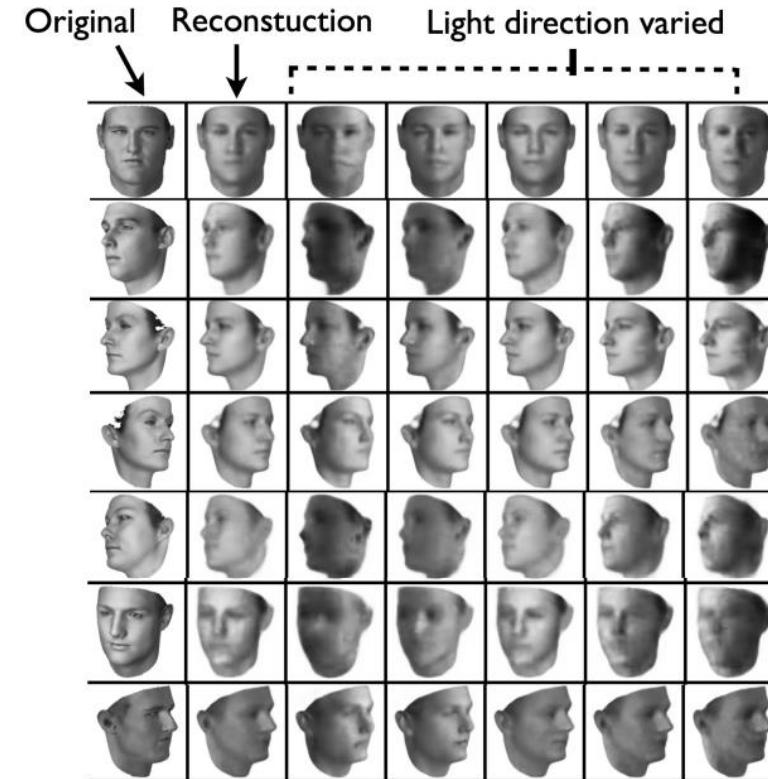
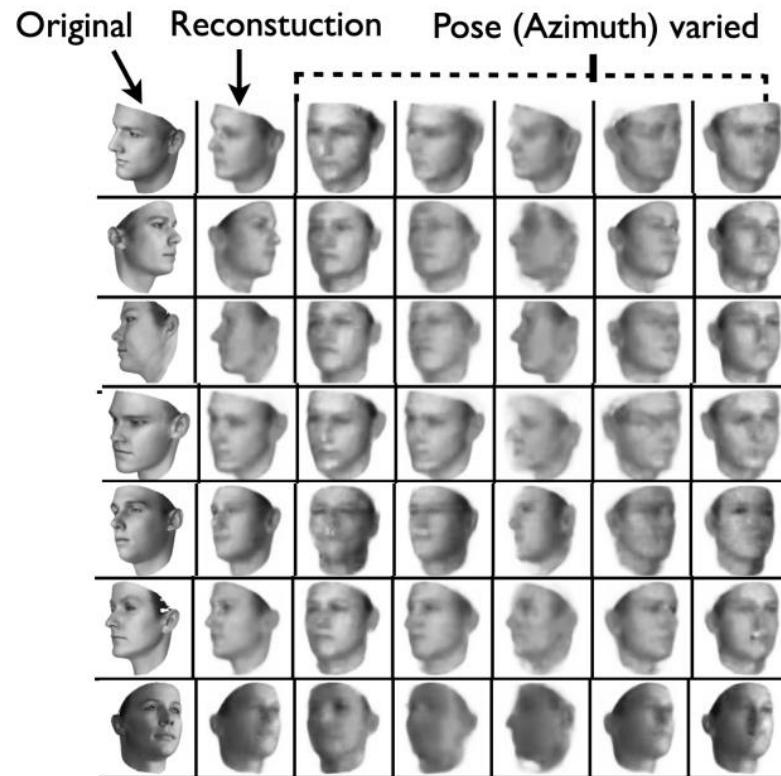
# Variational Autoencoders

- VAE : Editing with  $z$



# Variational Autoencoders

- VAE : Editing with z



# Variational Autoencoders

---

- **VAE : Summary**

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

**Pros:**

- Principled approach to generative models
- Allows inference of  $q(z|x)$ , can be useful feature representation for other tasks

**Cons:**

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

**Active areas of research:**

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs)
- Incorporating structure in latent variables, e.g., Categorical Distributions

**Autoregressive models**

- Directly maximize  $p(\text{data})$
- High-quality generated images
- Slow to generate images
- No explicit latent codes

**Variational models**

- Maximize lower-bound on  $p(\text{data})$
- Generated images often blurry
- Very fast to generate images
- Learn rich latent codes

# Generative Adversarial Networks

---

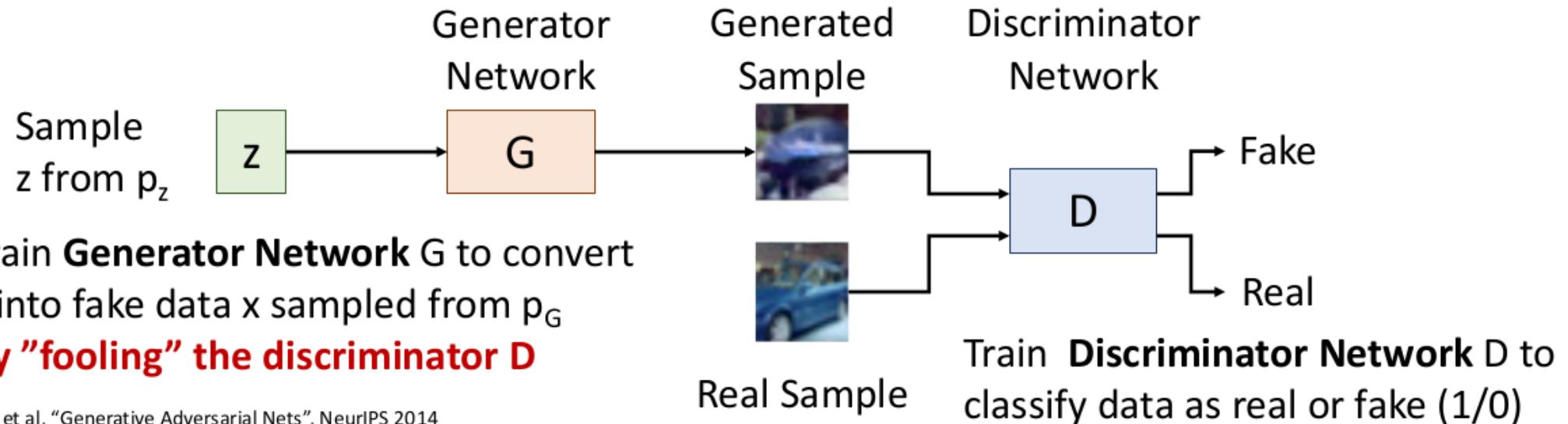
Models	설명
<b>Autoregressive models</b>	training data의 likelihood를 직접(directly) maximize한다. $p_{\theta}(x) = \prod_{i=1}^N p_{\theta}(x_i x_1, \dots, x_{i-1})$
<b>VAE</b>	latent $z$ 를 추가하였고, likelihood의 lower bound를 maximize한다. $p_{\theta}(x) = \int_Z p_{\theta}(x z)p(z) dz \geq E_{z \sim q_{\phi}(z x)}[\log p_{\theta}(x z)] - D_{KL}(q_{\phi}(z x), p(z))$
<b>GANs</b>	$p(x)$ 을 모델링하는 것을 포기한다. 그러나 $p(x)$ 로부터 샘플링할 수 있도록 한다.

# Generative Adversarial Networks

---

- **Setup**
  - $p_{\text{data}}(x)$  : real data distribution
  - $x_i$  : our train data from  $p_{\text{data}}(x)$
- **Idea**
  - Suppose a latent variable  $z$  with  $p(z)$  which is a simple prior (diagonal Gaussian, uniformed distribution, etc.).
  - Sample  $z$  from  $p(z)$  and pass through **Generative Network  $G$ .**
  - $x = G(z)$
  - Then the  $x$  is from Generative distribution  $p_G$ .
  - Therefore we want  $p_G = p_{\text{data}}$  (Our generative distribution to be real data distribution)

# Generative Adversarial Networks



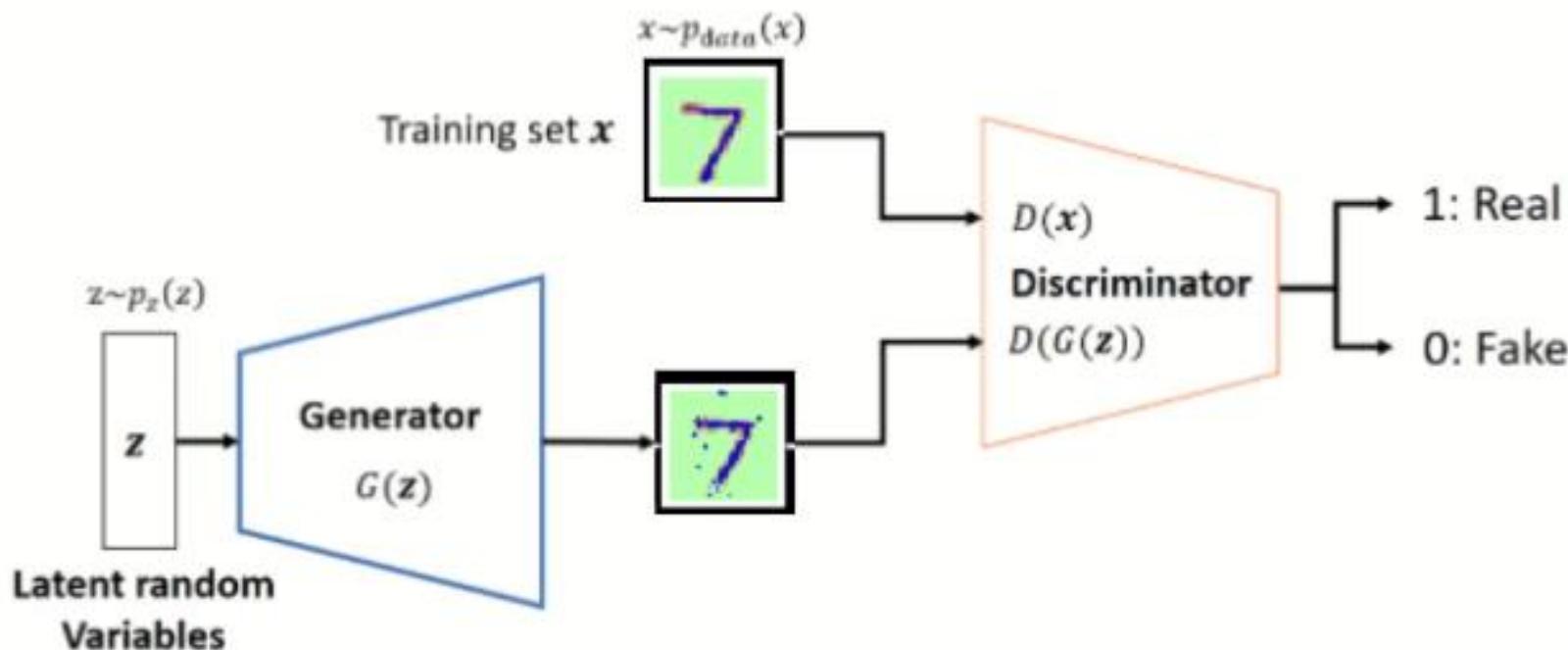
- **Generator**
  - By sampling  $x$  from  $p_G$ , train the model to generate an image that the Discriminator get fooled to think that the image was from  $p_{\text{data}}$ .
- **Discriminator**
  - Train to discriminate the generated sample and a real sample (real/fake(1/0)).
  - Jointly train the two networks. Then  $p_G$  will converge to  $p_{\text{data}}$ .

# Generative Adversarial Networks

---

## Training GANs: Two-player game

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



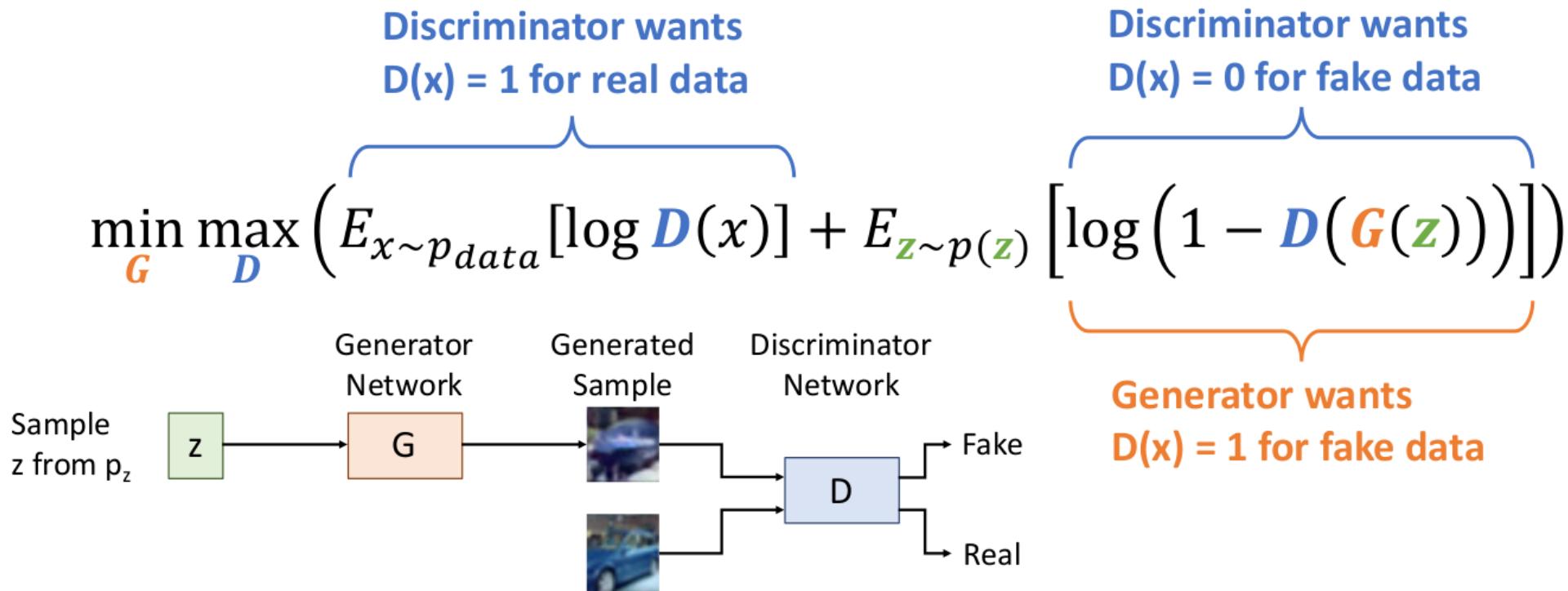
# GANs : Training Objective

---

- Train **Generator G** and **Discriminator D** jointly by **minmax game**.

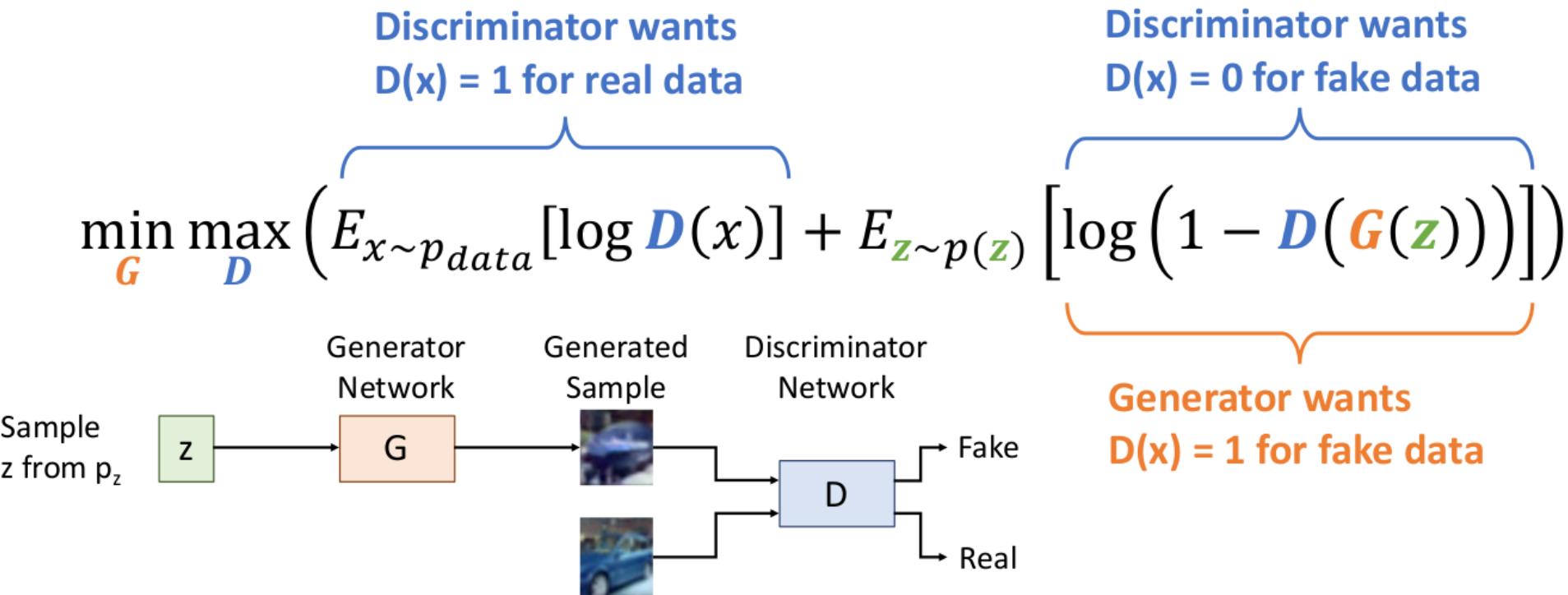
$$\min_G \max_D (E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] )$$

# GANs : Training Objective



- $x$  sampled from  $p_{data}$ , which is the real data to be REAL
- If  $D(x) < 1$ , it passes log term and becomes very small negative value. So we train  $D(x) = 1$  that the whole term can be maximized by  $D$ .

# GANs : Training Objective

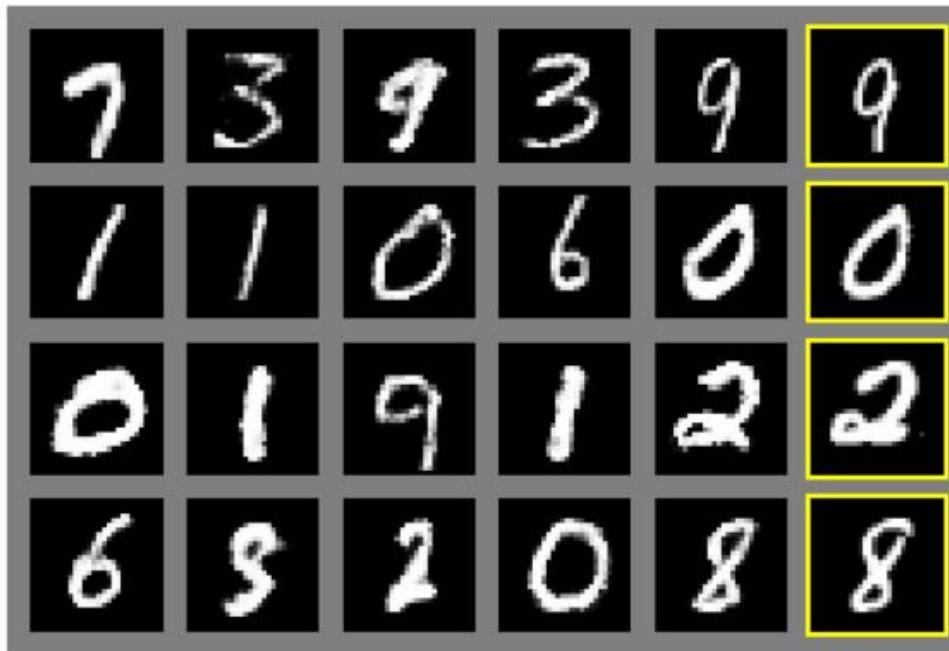


- $z$  sampled from  $p(z)$ , pass it to Generator  $G$ , and  $G$  outputs generated sample  $G(z)$ . Train Discriminator to discriminate the generated sample  $G(z)$  is fake. (fake to be FAKE)
- Generator  $G$  trains Discriminator  $D$  to discriminate  $G(z)$  is REAL. (fake to be REAL)

# GANs

---

Generated samples



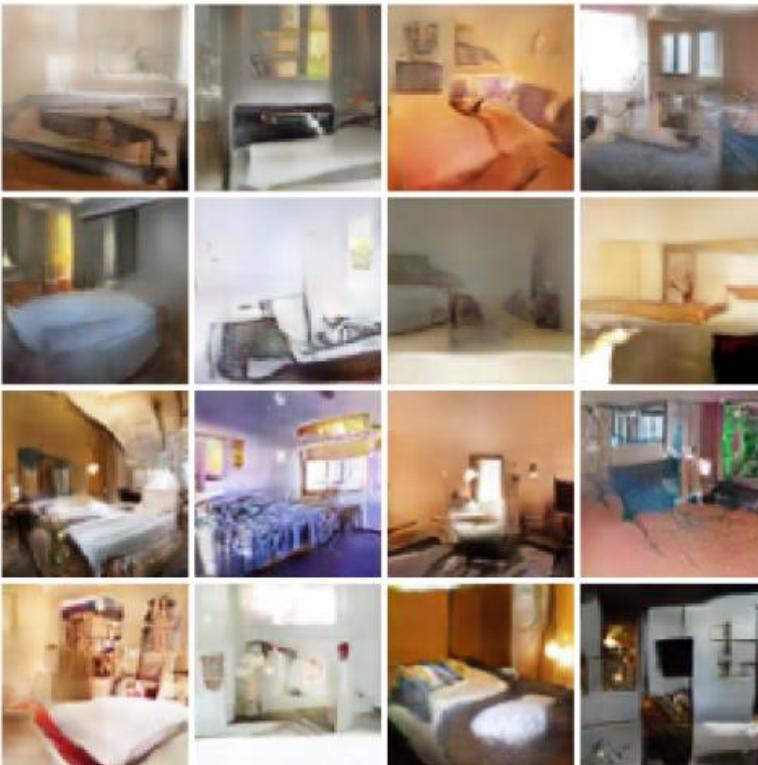
Nearest neighbor from training set

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# GANs

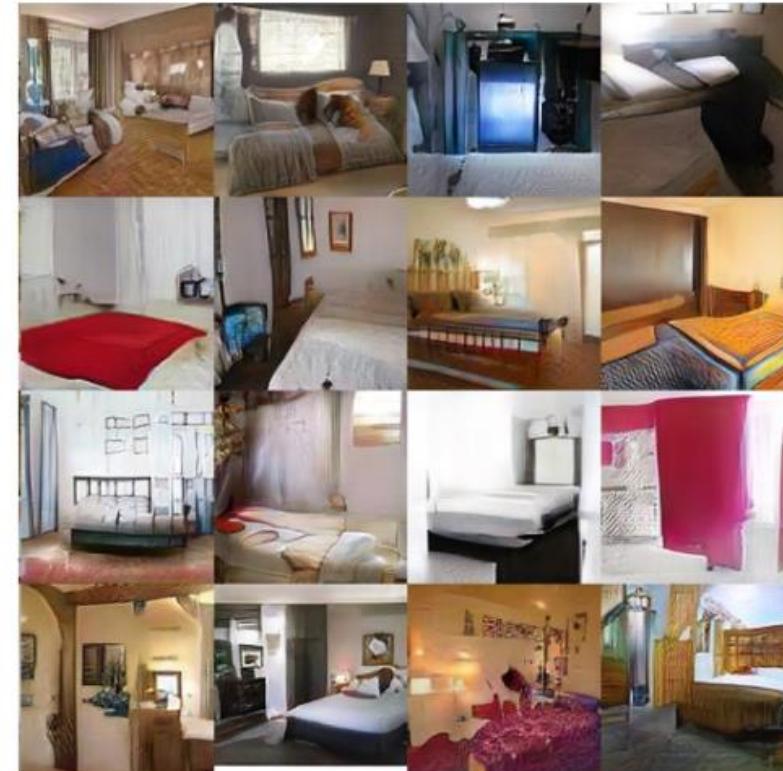
---

Wasserstein GAN (WGAN)



Arjovsky, Chintala, and Bottou, "Wasserstein GAN", 2017

WGAN with Gradient Penalty  
(WGAN-GP)



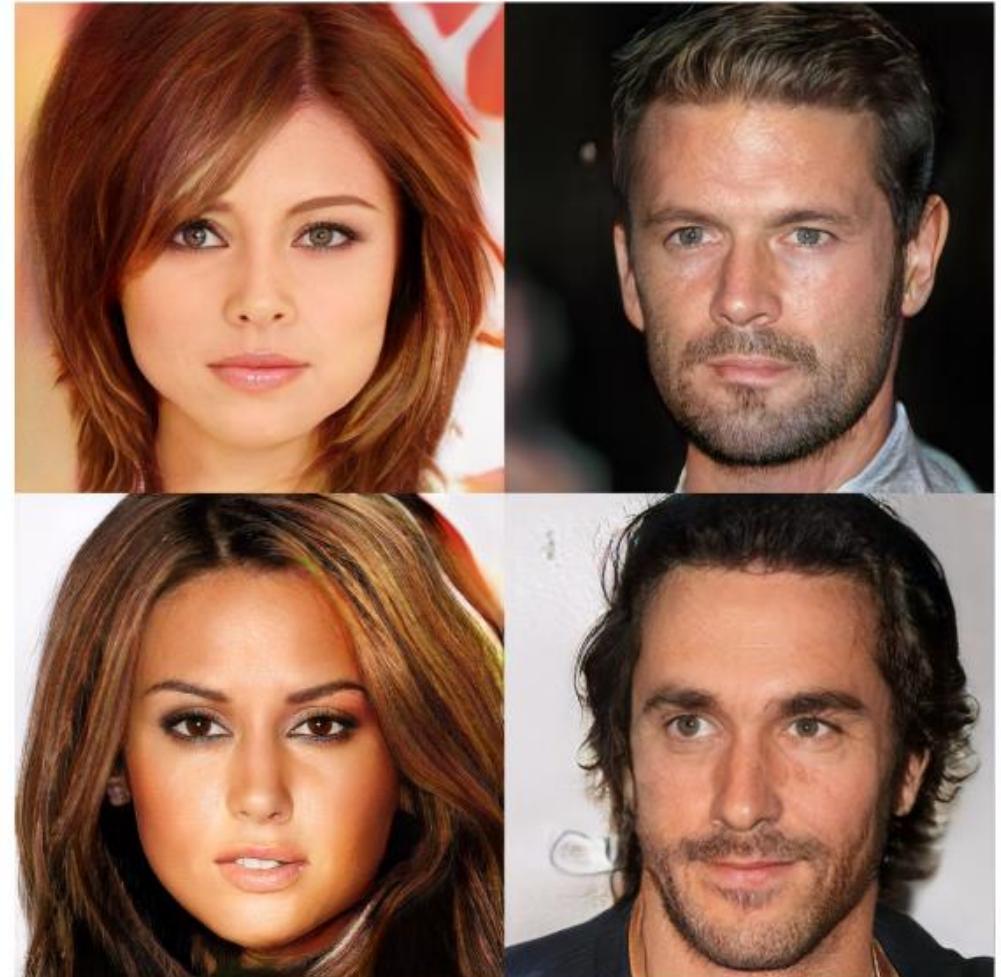
Gulrajani et al, "Improved Training of  
Wasserstein GANs", NeurIPS 2017

# GANs

256 x 256 bedrooms



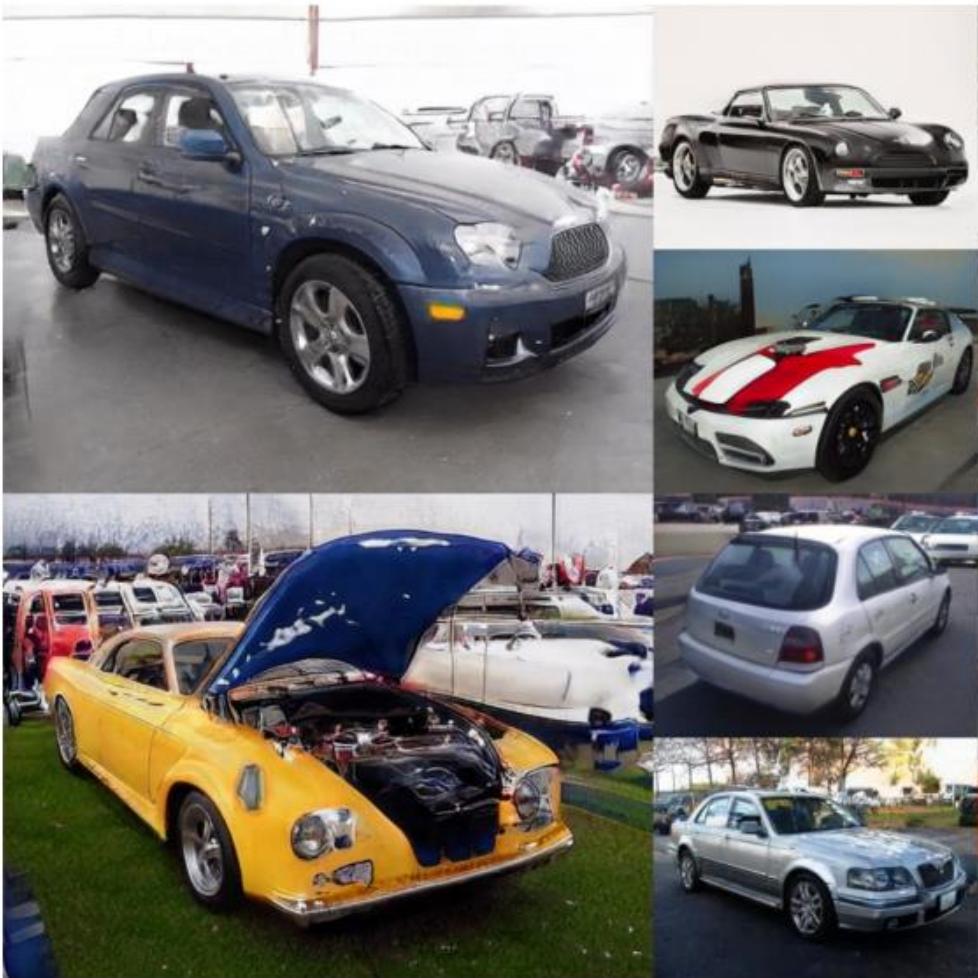
1024 x 1024 faces



Karras et al, "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

# GANs

512 x 384 cars



1024 x 1024 faces

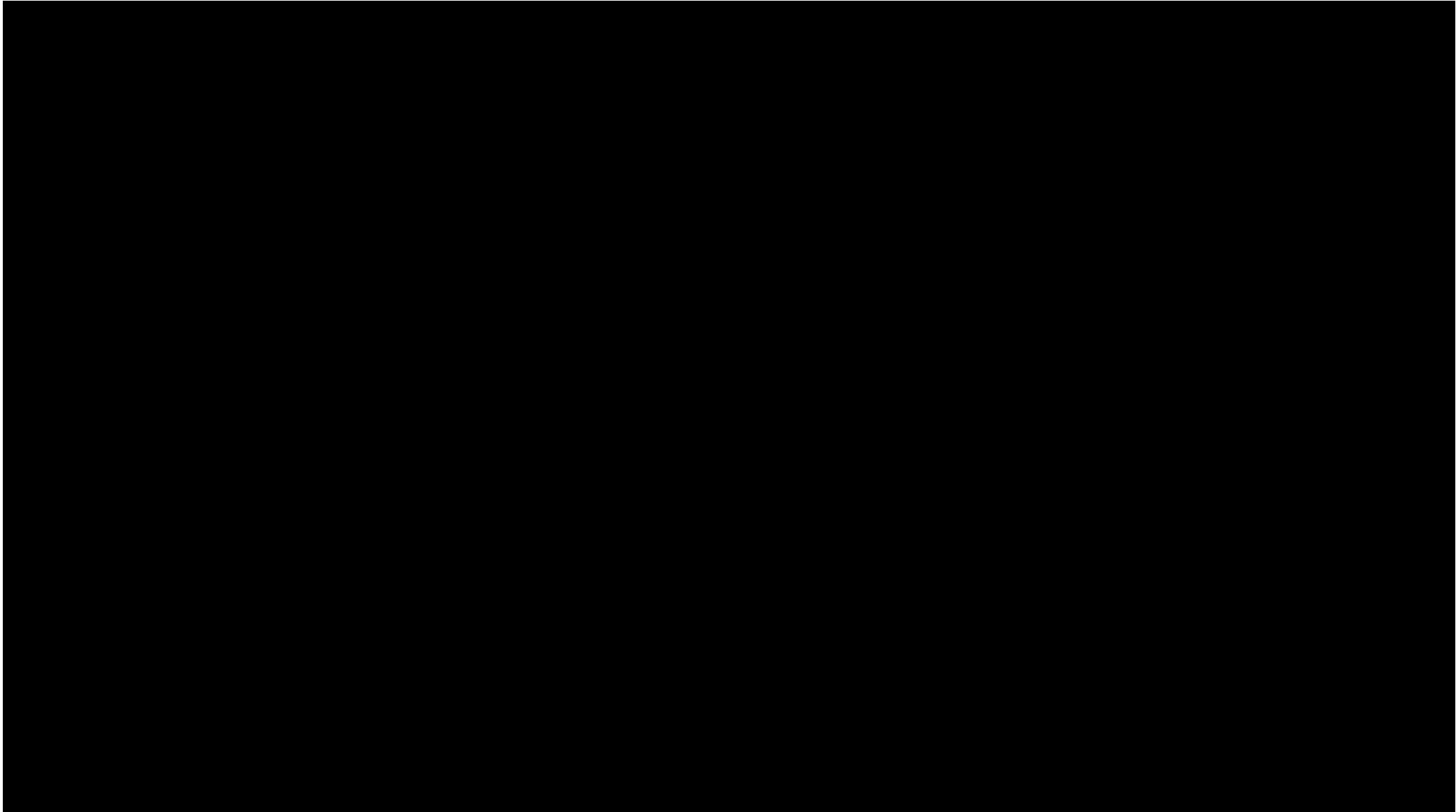


Karras et al, "A Style-Based Generator Architecture for Generative Adversarial Networks", CVPR 2019

[Images](#) are licensed under [CC BY-NC 4.0](#)

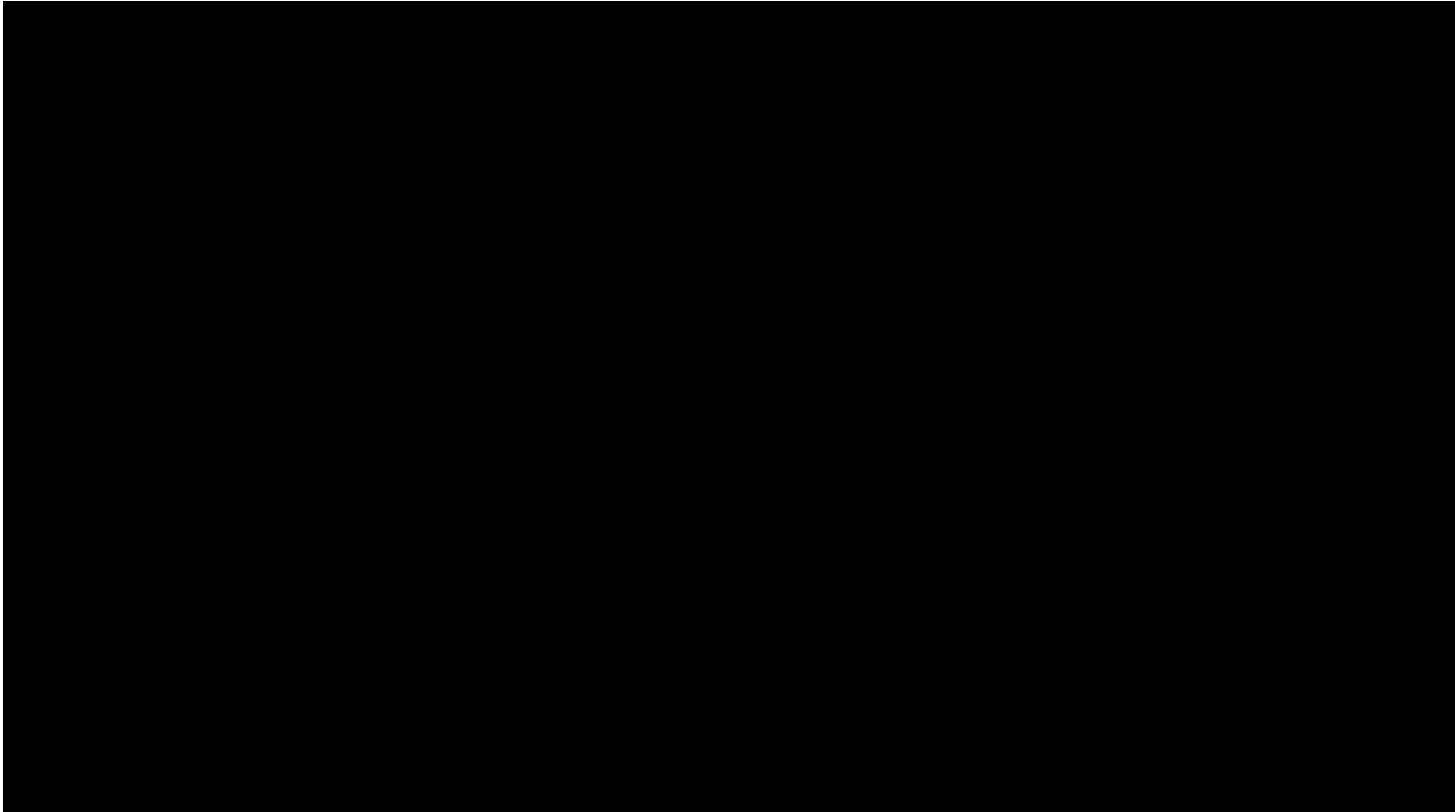
# GANs

---



# GANs

---



# GANs

---

This bird is red and brown in color, with a stubby beak

The bird is short and stubby with yellow on its body

A bird with a medium orange bill white body gray wings and webbed feet

This small black bird has a short, slightly curved bill and long legs

A picture of a very clean living room

A group of people on skis stand in the snow

Eggs fruit candy nuts and meat served on white dish

A street sign on a stoplight pole in the middle of a day



Zhang et al, "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks.", TPAMI 2018

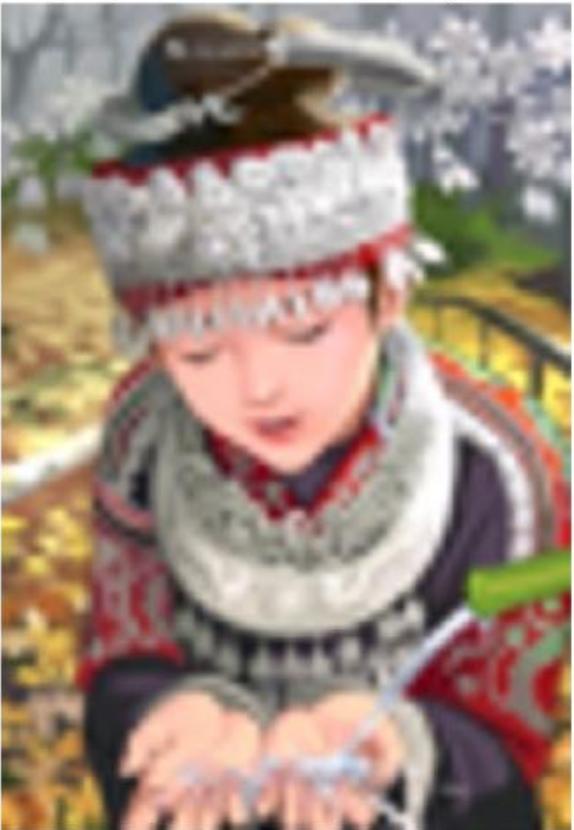
Zhang et al, "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks.", ICCV 2017

Reed et al, "Generative Adversarial Text-to-Image Synthesis", ICML 2016

# GANs

---

bicubic  
(21.59dB/0.6423)



SRResNet  
(23.53dB/0.7832)



SRGAN  
(21.15dB/0.6868)



original



Ledig et al, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", CVPR 2017

# GANs

---

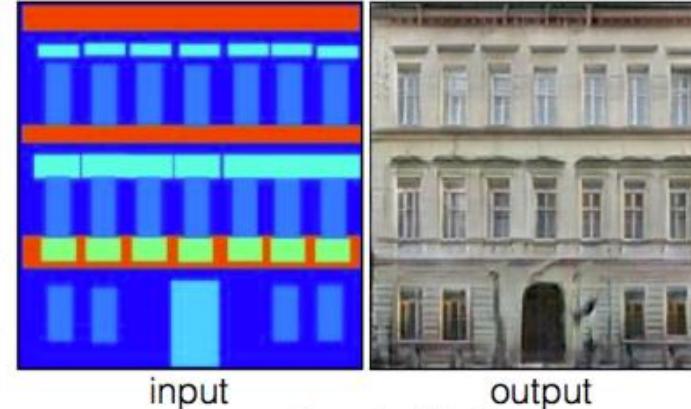
Labels to Street Scene



input

output

Labels to Facade



input

output

BW to Color



input

output

Aerial to Map



input

output

Day to Night



input

output

Edges to Photo

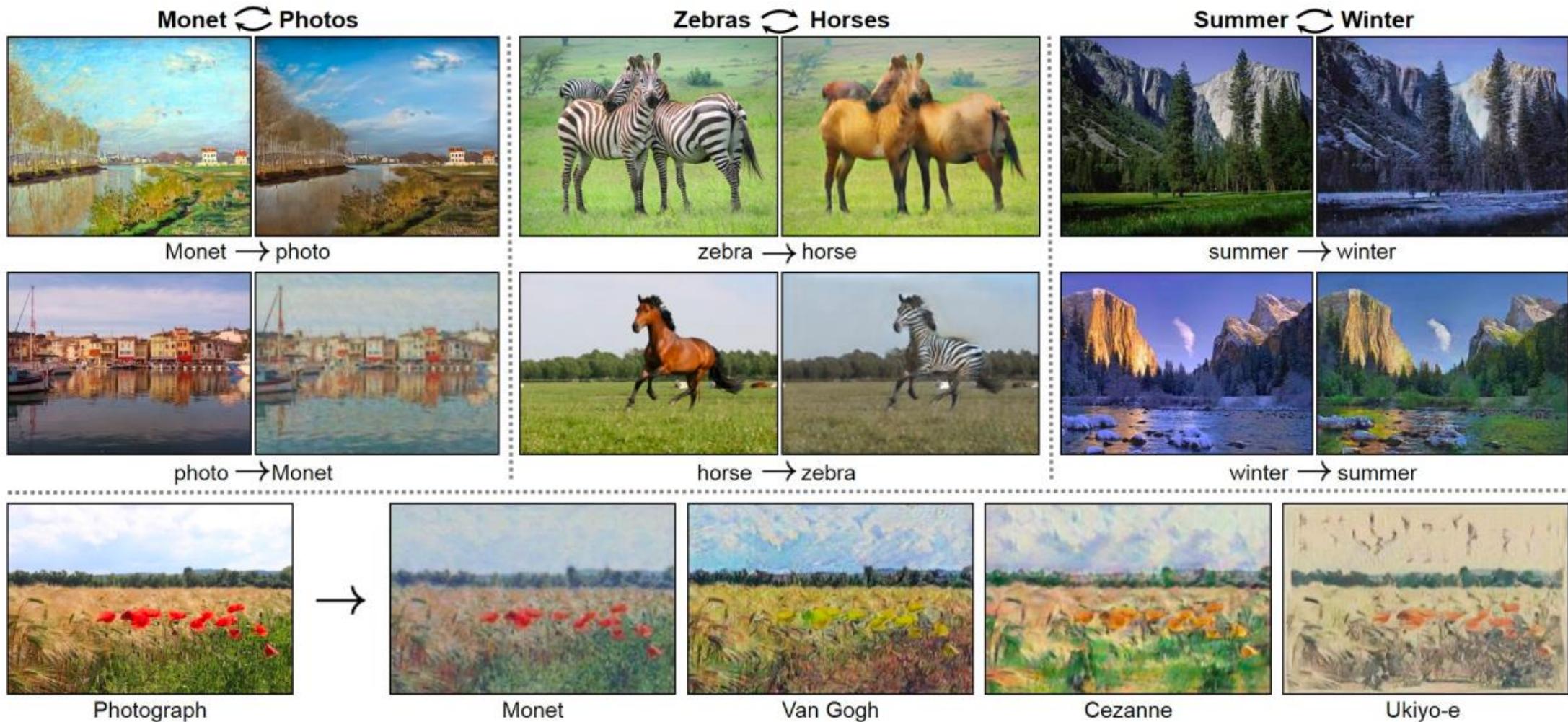


input

output

Isola et al, "Image-to-Image Translation with Conditional Adversarial Nets", CVPR 2017

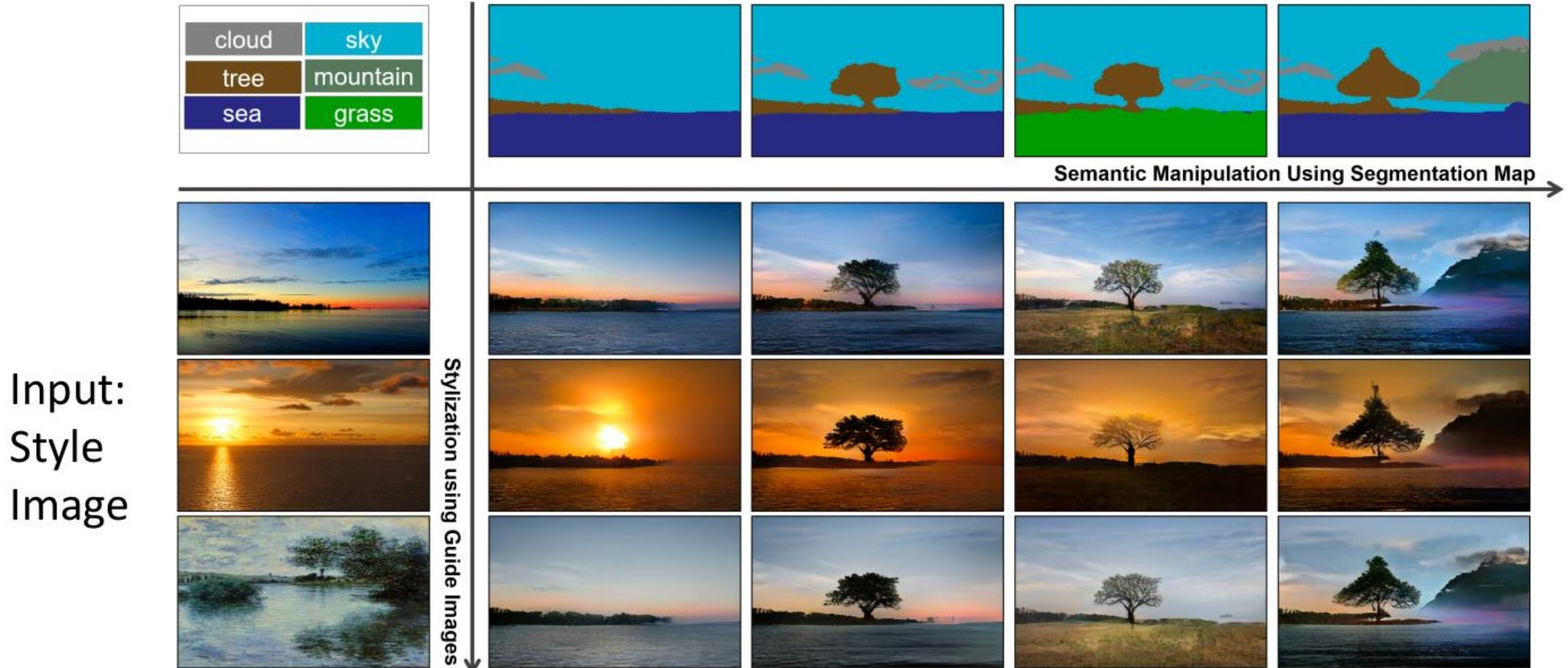
# GANs



Zhu et al, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV 2017

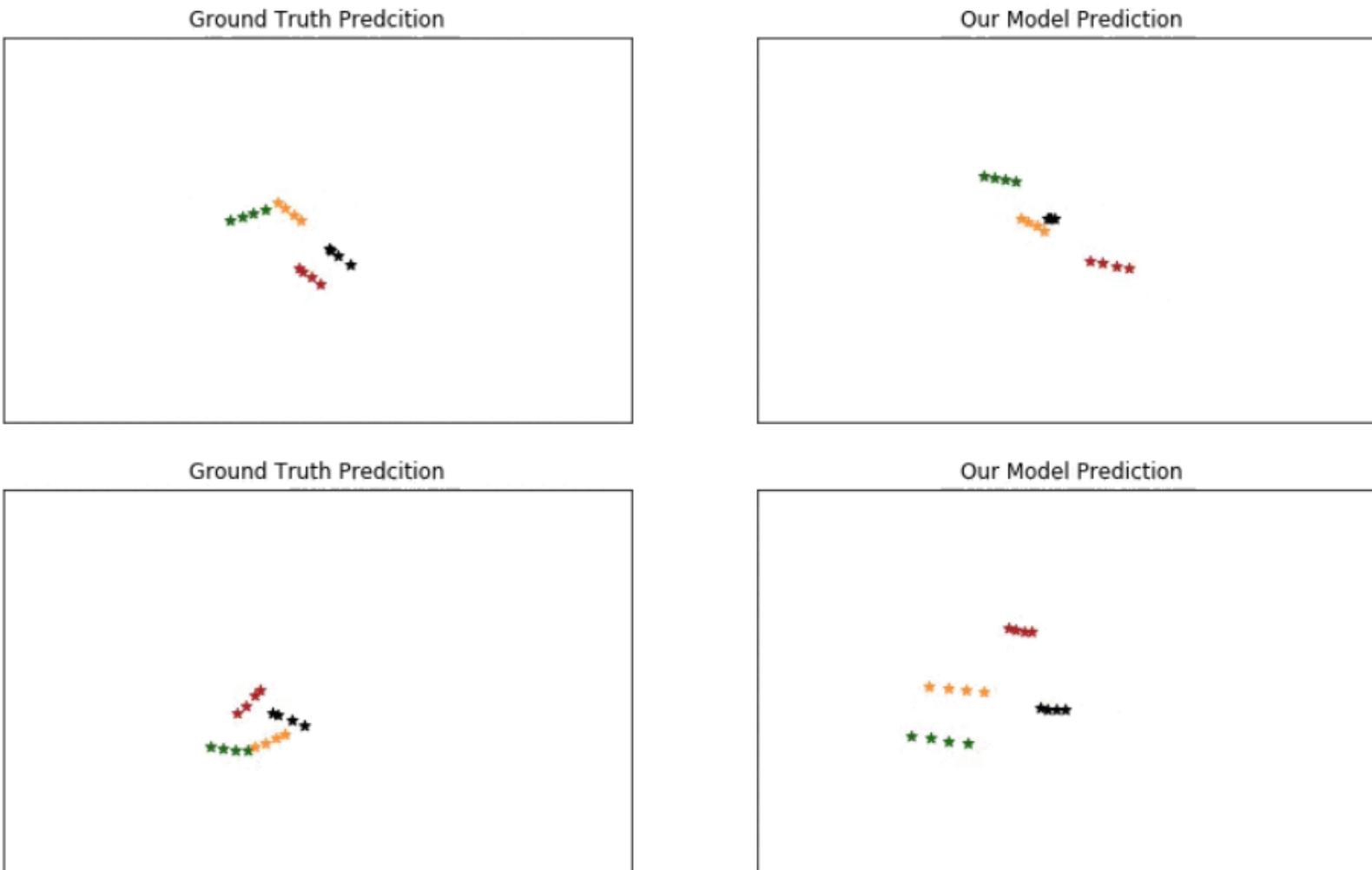
# GANs

## Label Map to Image



Park et al, "Semantic Image Synthesis with Spatially-Adaptive Normalization", CVPR 2019

# GANs



Gupta, Johnson, Li, Savarese, Alahi, "Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks", CVPR 2018

---

**Thank you!**  
**Q & A**