CS 1678/2078: Intro to Deep Learning

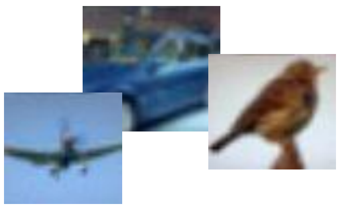
Image/Video Generation

Prof. Adriana Kovashka
University of Pittsburgh
April 10, 2024

Plan for this lecture

- Motivation and taxonomy of methods
- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)
  - Applications and variants of GANs
- Diffusion models
  - Example results and variants of diffusion models

Generative Models



Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Addresses density estimation, a core problem in unsupervised learning

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, etc.



- Generative models can be used to enhance training datasets with diverse synthetic data
- Generative models of time-series data can be used for simulation

Adapted from Serena Young

Taxonomy of Generative Models

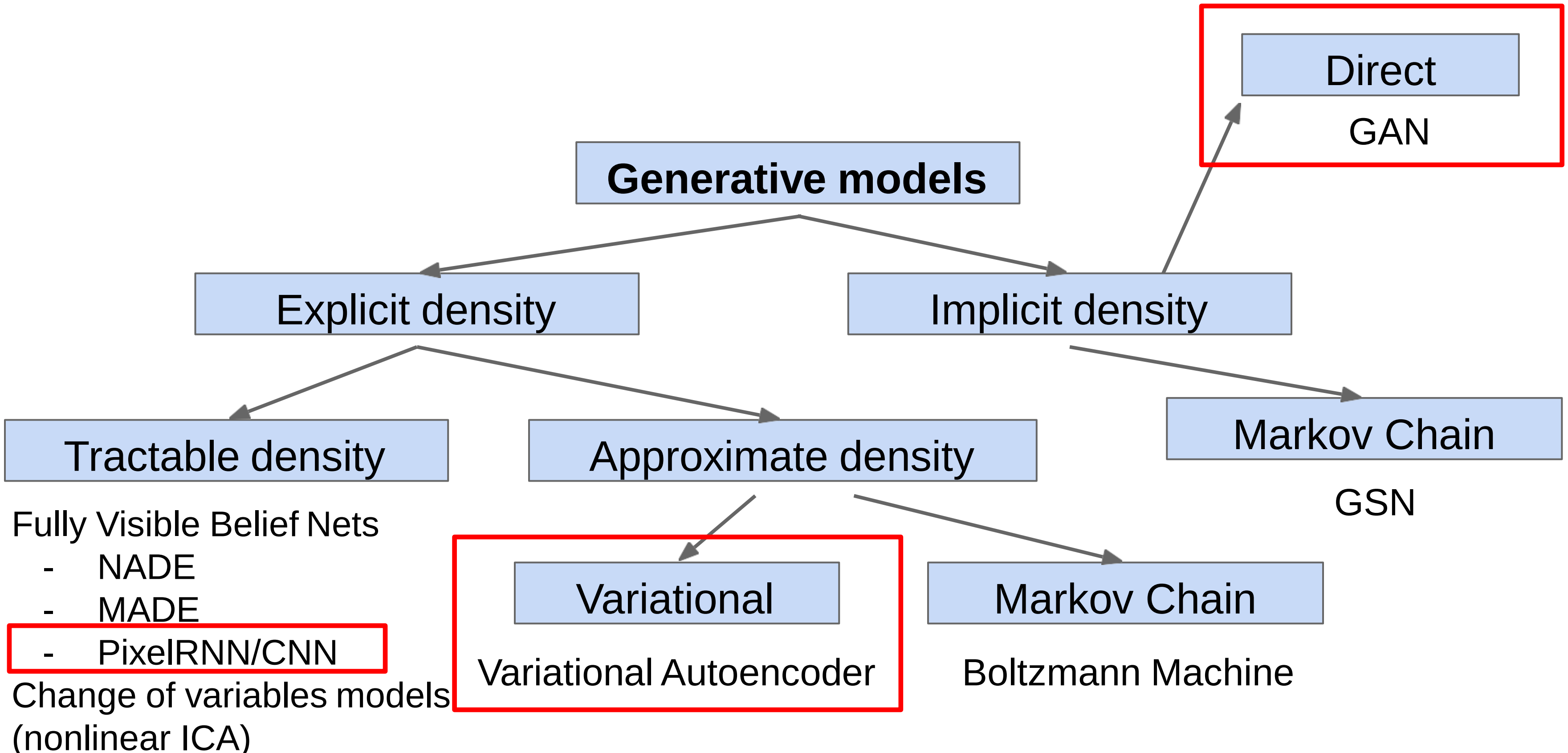


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Fully visible belief network

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Will need to define ordering of "previous pixels"

Then maximize likelihood of training data

Complex distribution over pixel values => Express using a neural network!

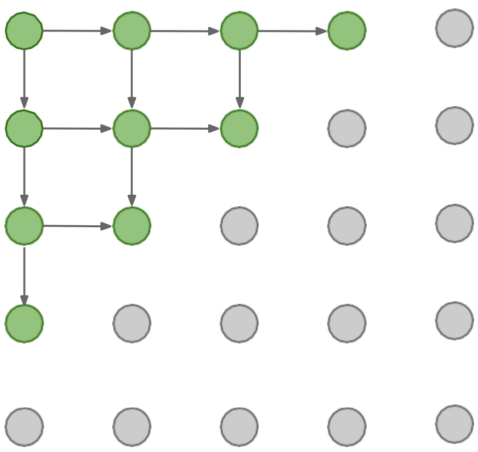
Serena Young

PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



Drawback: sequential generation is slow!

PixelCNN

[van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$$

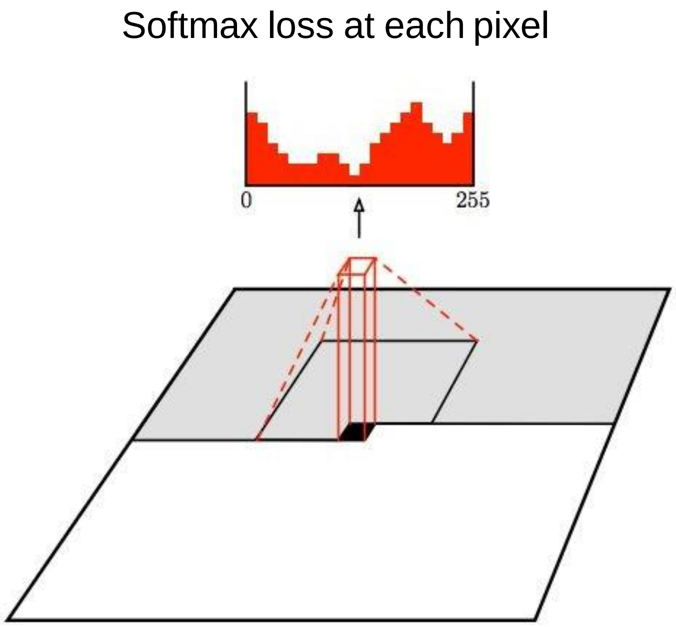


Figure copyright van der Oord et al., 2016.

Training is faster than PixelRNN (can parallelize convolutions since context region values known from training images)

Generation must still proceed sequentially => still slow

Autoencoders

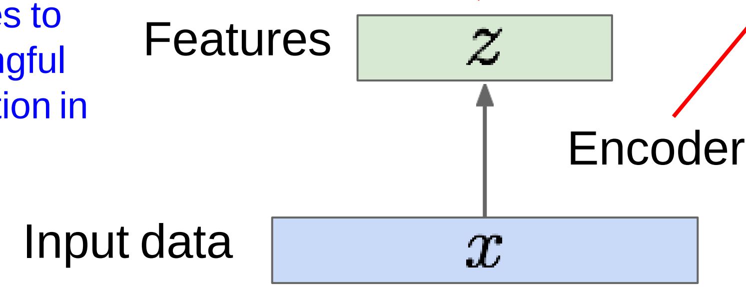
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

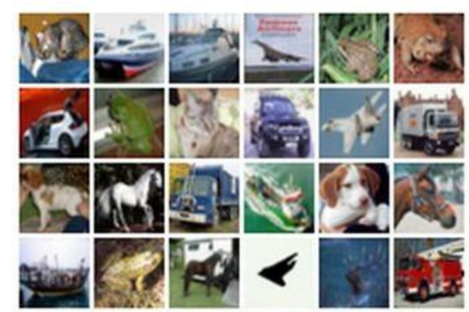
z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN



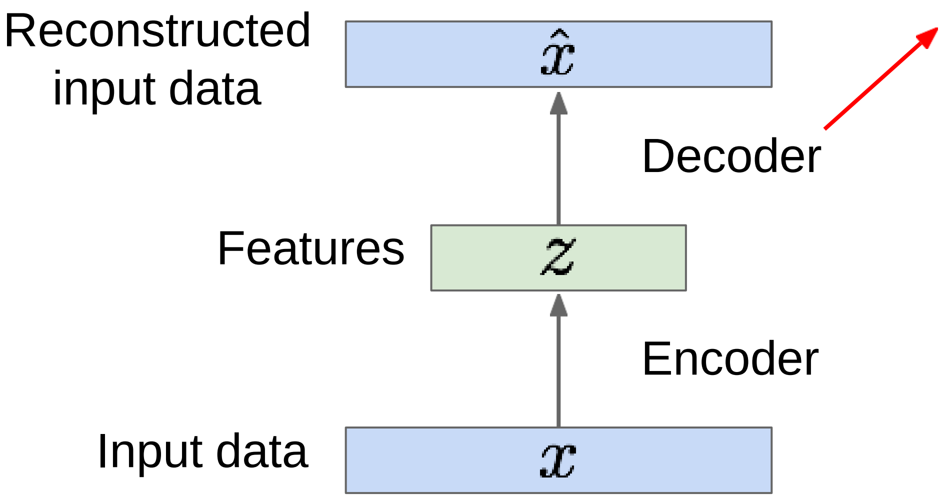


Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

“Autoencoding” - encoding itself



Originally: Linear + nonlinearity (sigmoid)

Later: Deep, fully-connected

Later: ReLU CNN (upconv)

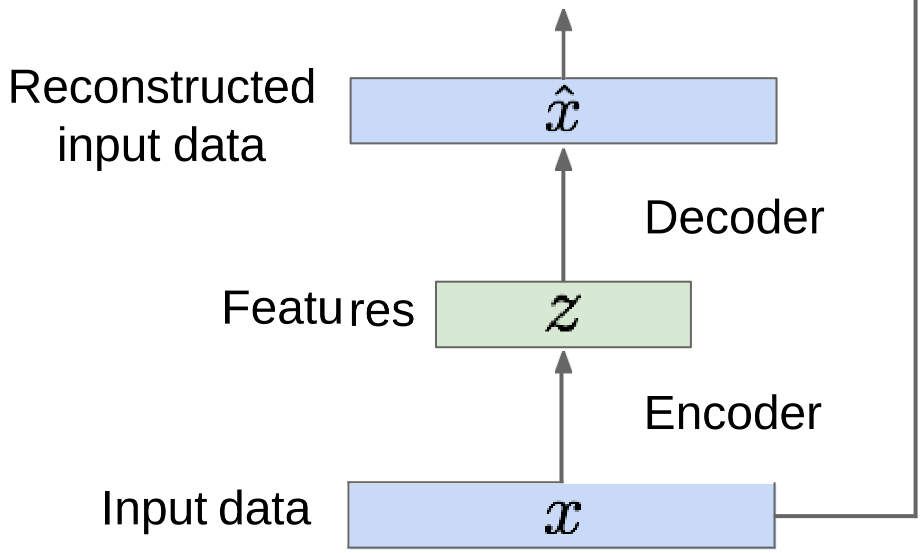
Autoencoders

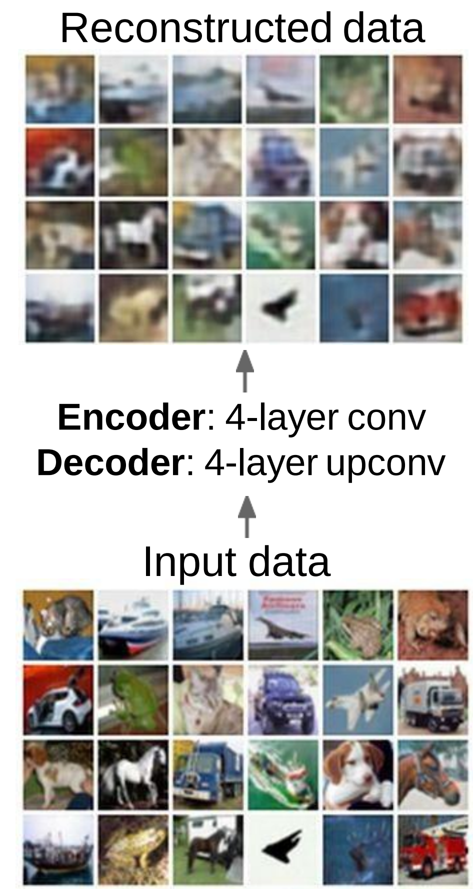
Doesn't use labels!

Train such that features can be used to reconstruct original data

L2 Loss function:

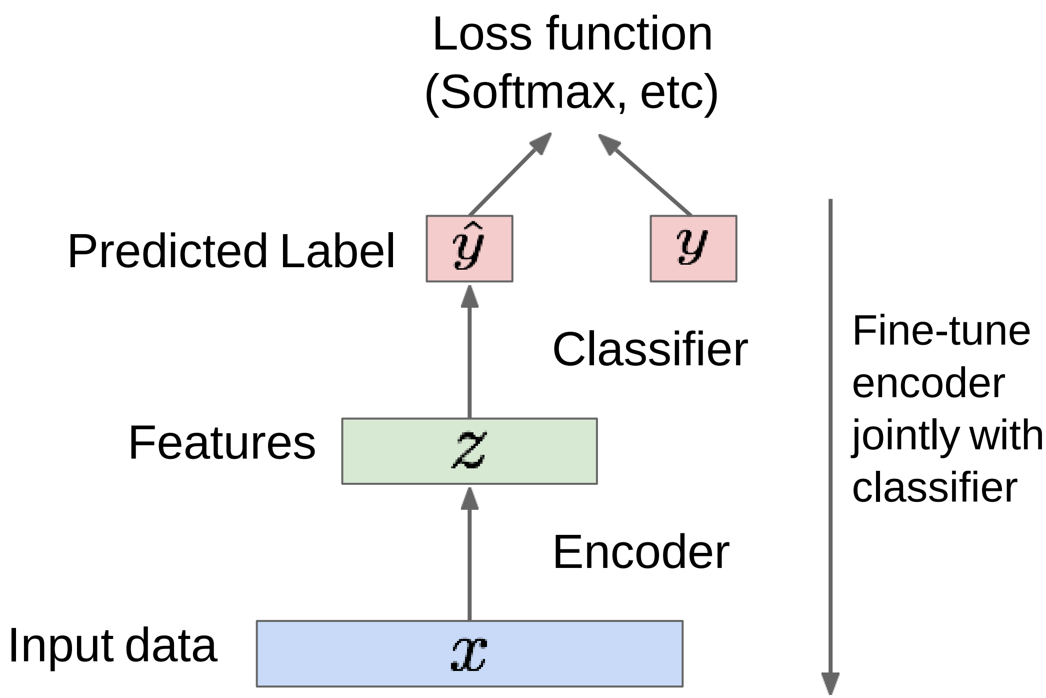
$$\|x - \hat{x}\|^2$$





Autoencoders

Encoder can be used to initialize a **supervised** model



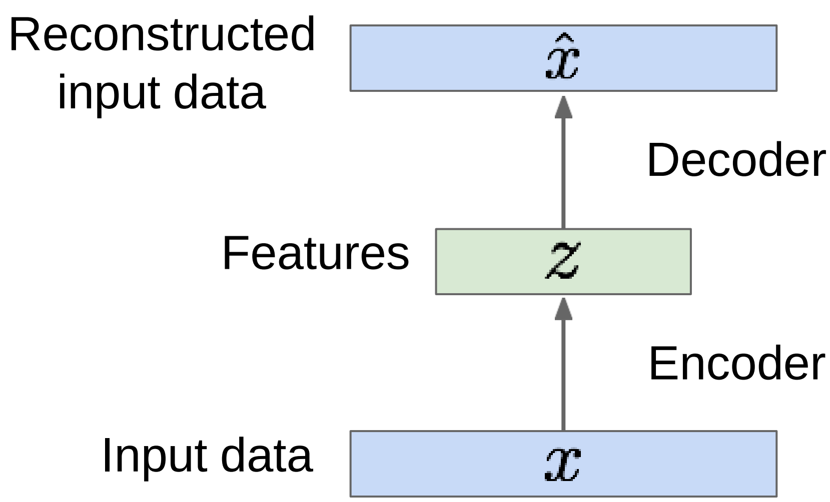
bird plane

dog deer truck

Train for final task (sometimes with small data)

Serena Young

Autoencoders



Features capture factors of variation in training data. Can we *generate* new images from an autoencoder?

Adapted from Serena Young

Variational Autoencoders

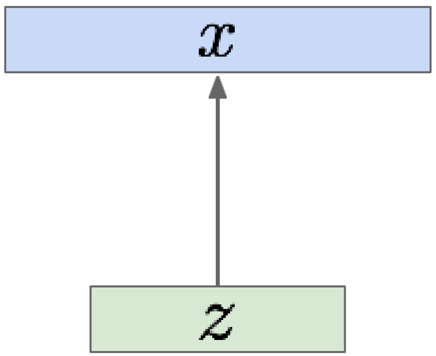
Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **z**

Intuition: **x** is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

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

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



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

Adapted from Serena Young

Variational Autoencoders

We want to estimate the true parameters θ^* of this generative model.

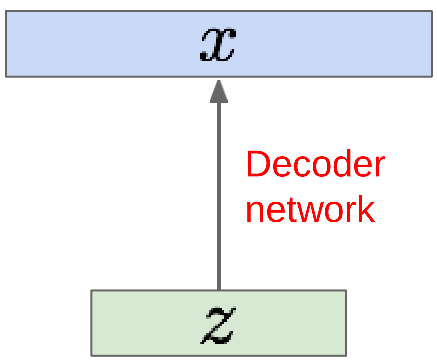
How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian.

Conditional p(x|z) is complex (generates image) => represent with neural network

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

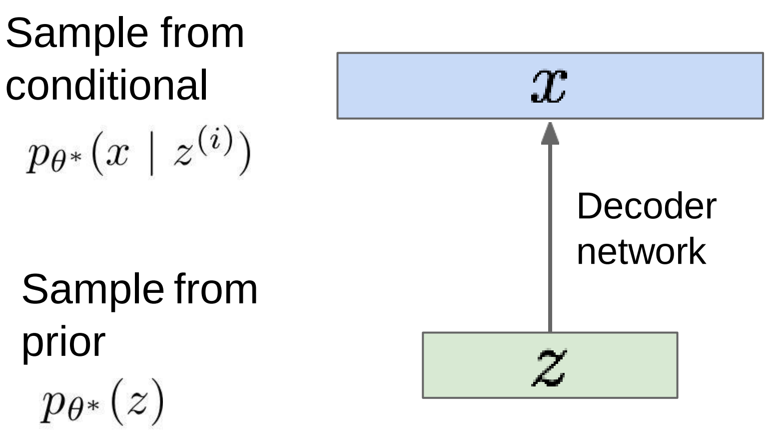
Sample from prior
$p_{\theta^*}(z)$



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

Adapted from Serena Young

Variational Autoencoders



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Learn model parameters to maximize likelihood of training data

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

Now with latent z

Q: What is the problem with this?

Intractable!

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

Adapted from Serena Young

Variational Autoencoders: Intractability

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

Simple Gaussian prior

Intractable to compute p(x|z) for every z!

Decoder neural network

Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$

Intractable data likelihood

- Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that *approximates* $p_\theta(z|x)$
- This allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize – overviewed briefly on next few slides (feel free to skip when reviewing)

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

Adapted from Serena Young

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

We want to maximize the data likelihood

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

We want to maximize the data likelihood

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

Reconstruct the input data

Make approximate posterior distribution close to prior

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

Takeaway: Variational Lower Bound

x = data
h = hidden representation

q = encoder (data to noise)
p = decoder (noise to data)

$$
\begin{aligned}
D_{\mathrm{KL}}(q(h|x) \parallel p(h|x)) &= \mathbb{E}_q[\log q(h|x) - \log p(h|x)] \\
&= \mathbb{E}_q[\log q(h|x) - \log p(x,h) + \log p(x)] \\
&= \mathbb{E}_q[\log q(h|x) - \log p(x,h)] + \log p(x)
\end{aligned}
$$

Notice that the expectation is the sign-flipped version ELBO term we derived above.

$$
\text{ELBO} = \mathbb{E}_q[\log p(x,h) - q(h|x)] \tag{6}
$$

Therefore, we have

$$
\begin{aligned}
&D_{\mathrm{KL}}(q(h|x) \parallel p(h|x)) = -\mathrm{ELBO} + \log p(x) \\
\implies &\log p(x) - \mathrm{ELBO} = D_{\mathrm{KL}}(q(h|x) \parallel p(h|x))
\end{aligned}
$$

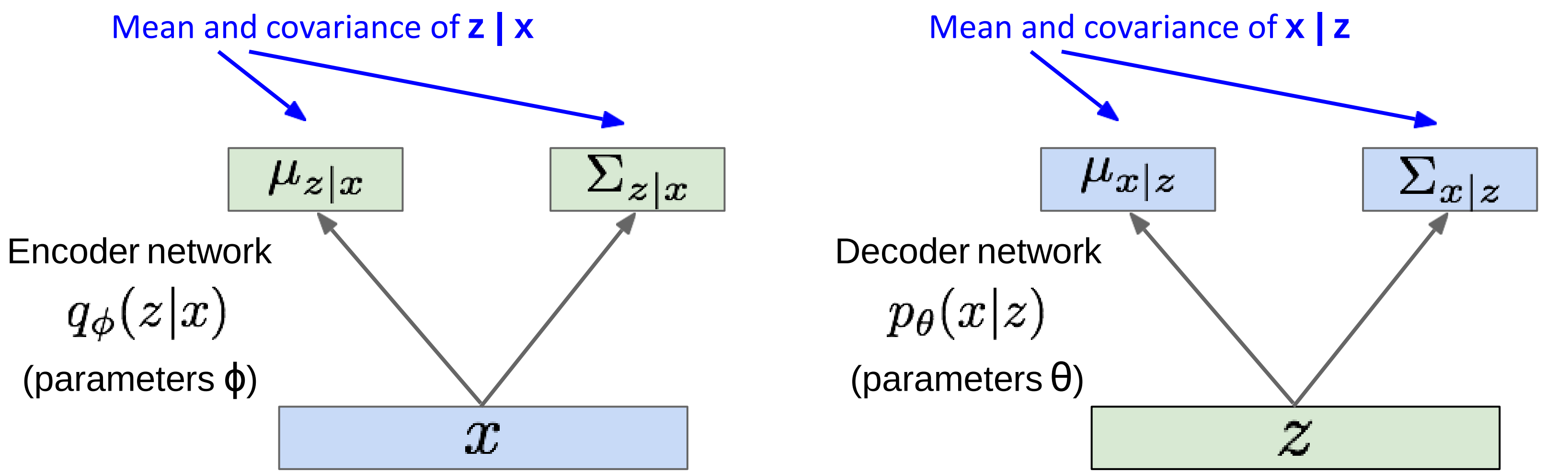
$$
\log p(x) \geq \mathrm{ELBO}
$$

$$
\begin{aligned}
\log p(x) &\geq \mathbb{E}_q[\log p(x,h) - \log q(h|x)] \\
&\geq \mathbb{E}_q\left[\log \frac{p(x,h)}{q(h|x)}\right]
\end{aligned}
$$

https://jaketae.github.io/study/elbo/

Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic

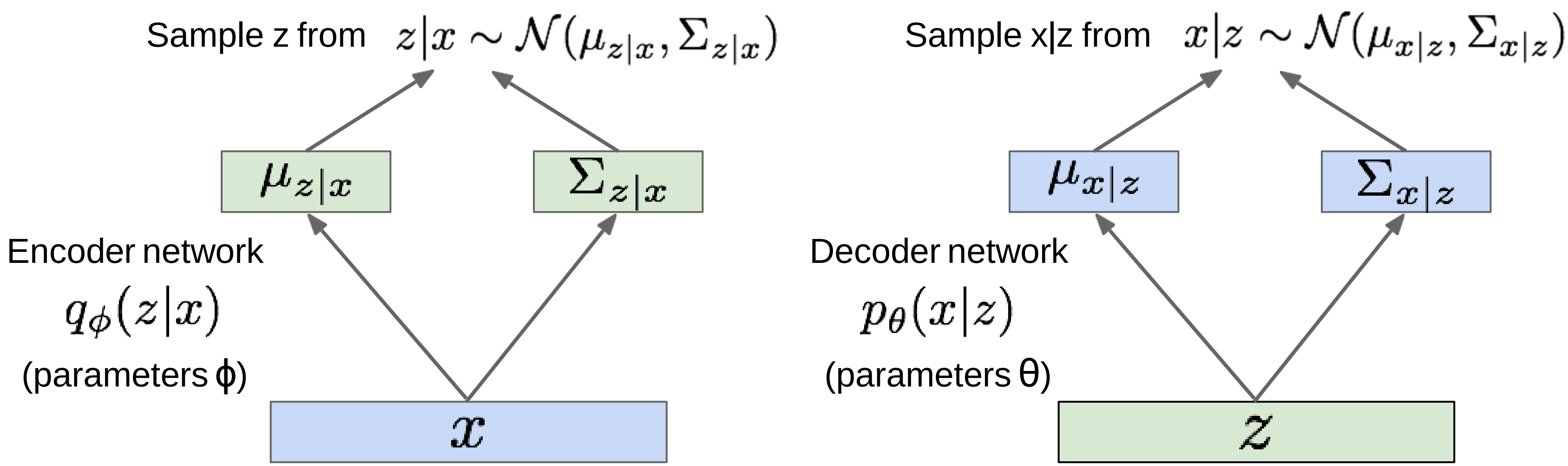


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Adapted from Serena Young

Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



Encoder and decoder networks also called "recognition"/"inference" and "generation" networks

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

Serena Young

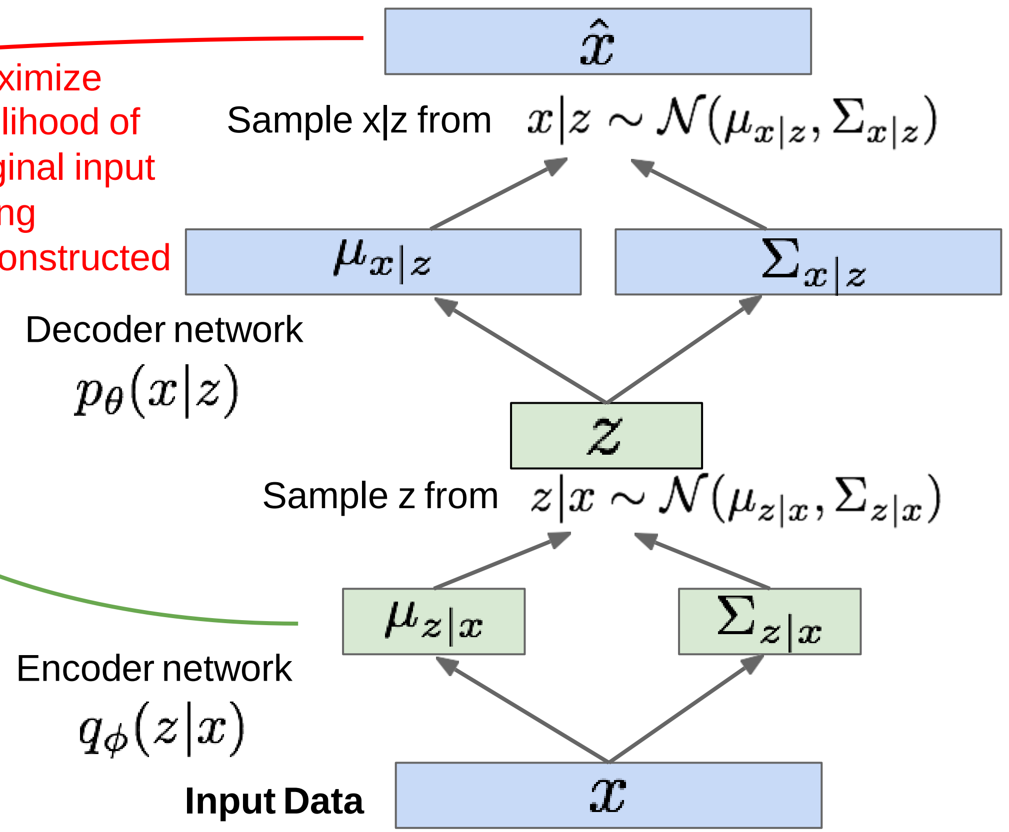
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

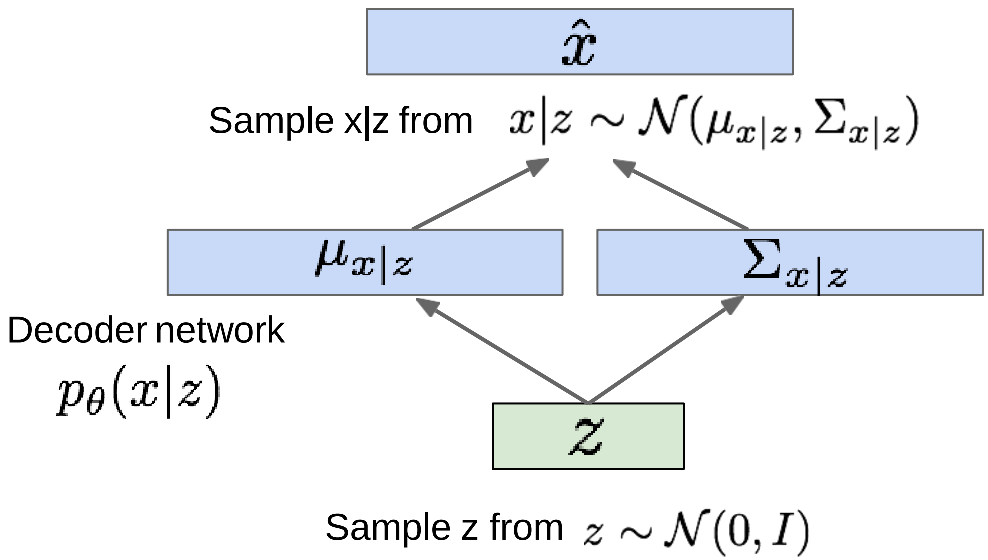
Make approximate posterior distribution close to prior



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

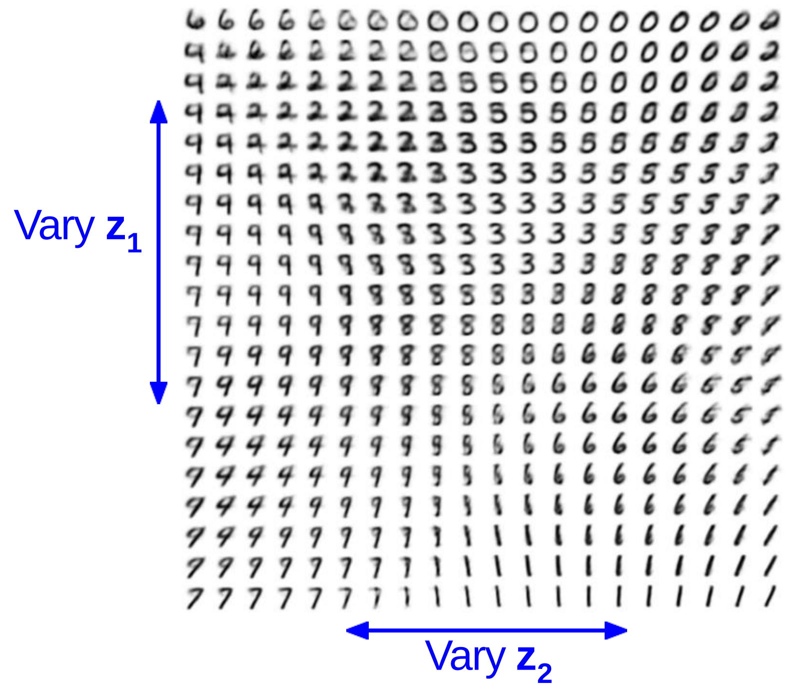
VAEs: Generating Data

Sample z from prior
Use decoder network



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

Data manifold for 2-d **z**



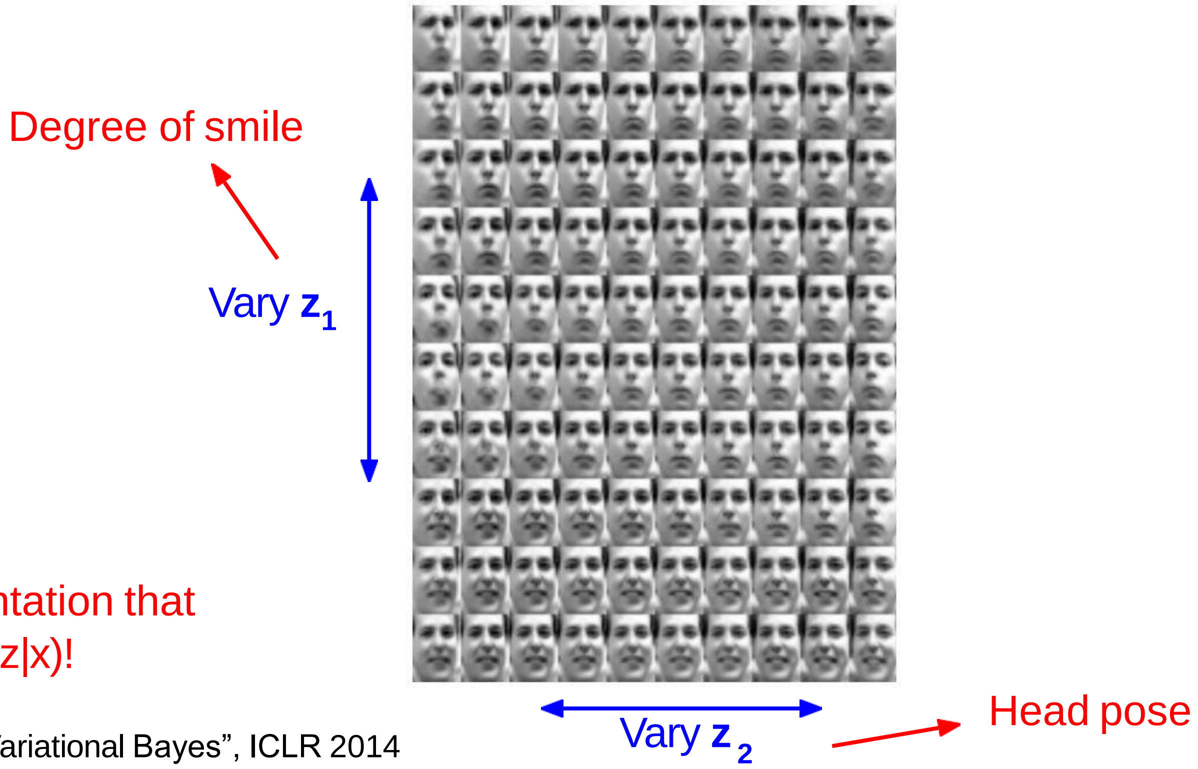
Adapted from Serena Young

VAEs: Generating Data

Diagonal prior on **z** => independent latent variables

Different dimensions of **z** encode interpretable factors of variation

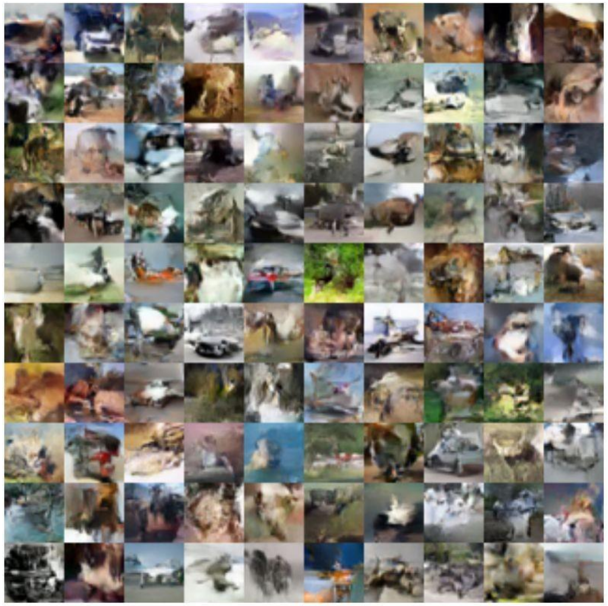
Also good feature representation that can be computed using $q_\phi(z|x)$!



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

Serena Young

VAEs: Generating Data



32x32 CIFAR-10

Labeled Faces in the Wild

Generating with little data for ads

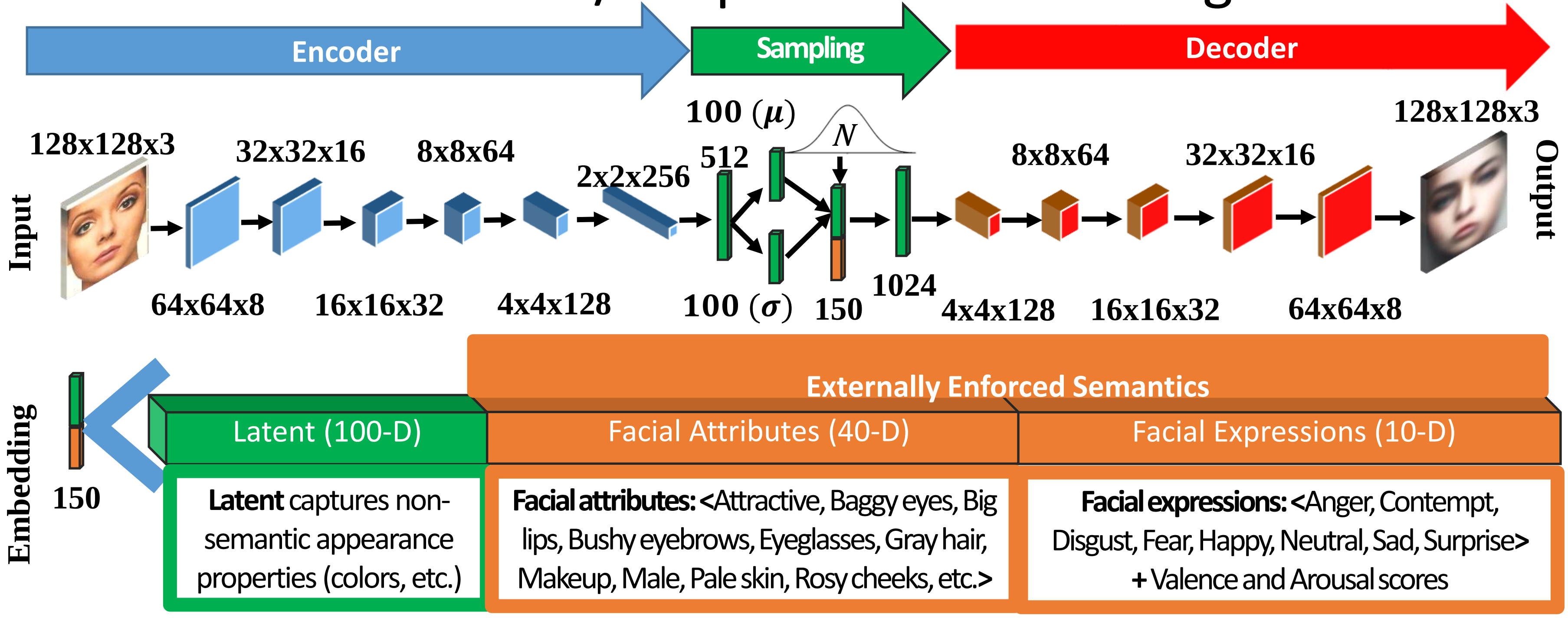
- Faces are persuasive and carry meaning/sentiment

- We learn to generate faces appropriate for each ad category
- Because our data is so diverse yet limited in count, standard approaches that directly model pixel distributions don’t work well

Thomas and Kovashka, BMVC 2018 https://people.cs.pitt.edu/~kovashka/thomas_kovashka_bmvc2018.pdf

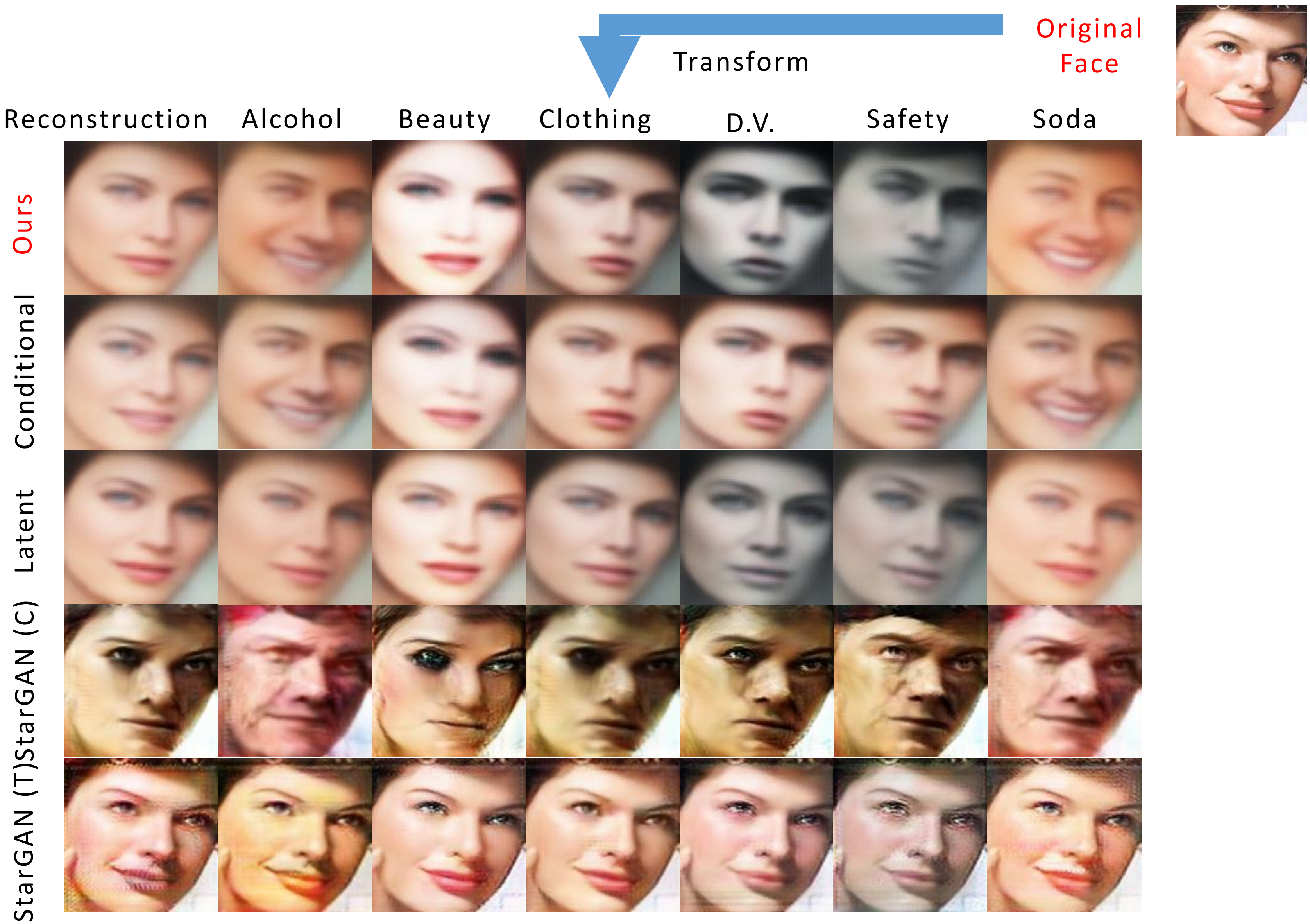
Generating with little data for ads

- Instead we model the distribution over *attributes* for each category (e.g. domestic violence ads contain "black eye", beauty contains "red lips")
- Generate an image with the attributes of an ad class
- Model attributes w/ help from external large dataset



Thomas and Kovashka, BMVC 2018

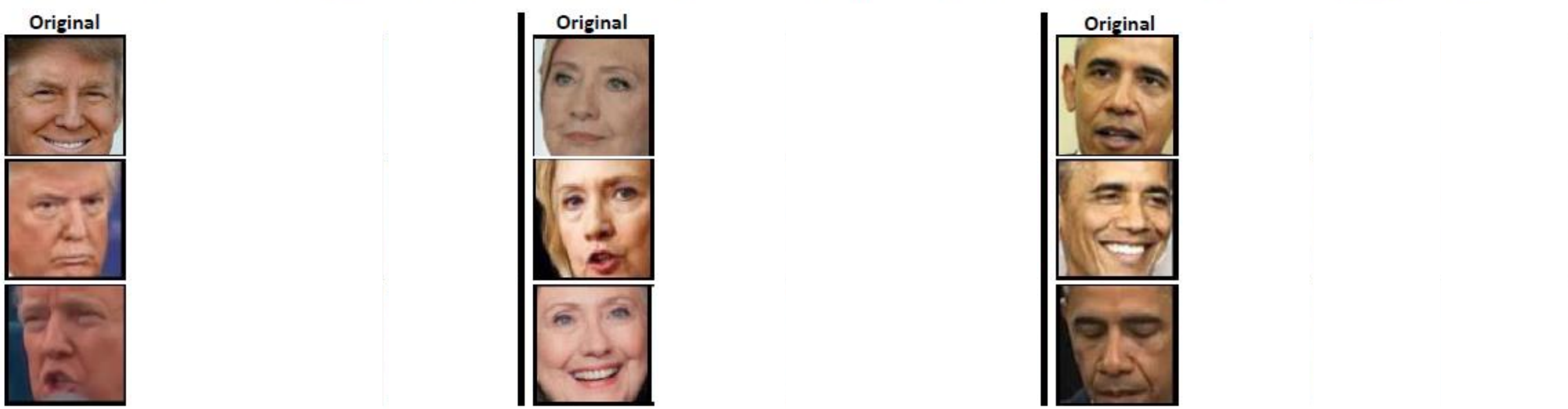
Generating with little data for ads



Thomas and Kovashka, BMVC 2018

Faces in left- and right-leaning media

- To illustrate the visual variability between left/right, we modify photos to be more left/right-leaning
- We model left/right using distributions over *attributes* (predicted using separate dataset, no extra annotations, Thomas & Kovashka BMVC 2018)
- Map attributes to pixels using large face dataset



Thomas and Kovashka, NeurIPS 2019

Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a 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)

Adapted from Serena Young

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

VAEs define intractable density function with latent **z**:

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

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: don't work with any explicit density function!
Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

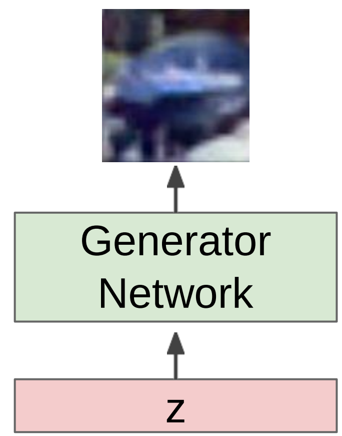
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!

Output: Sample from training distribution



Input: Random noise

Training GANs: Two-player game

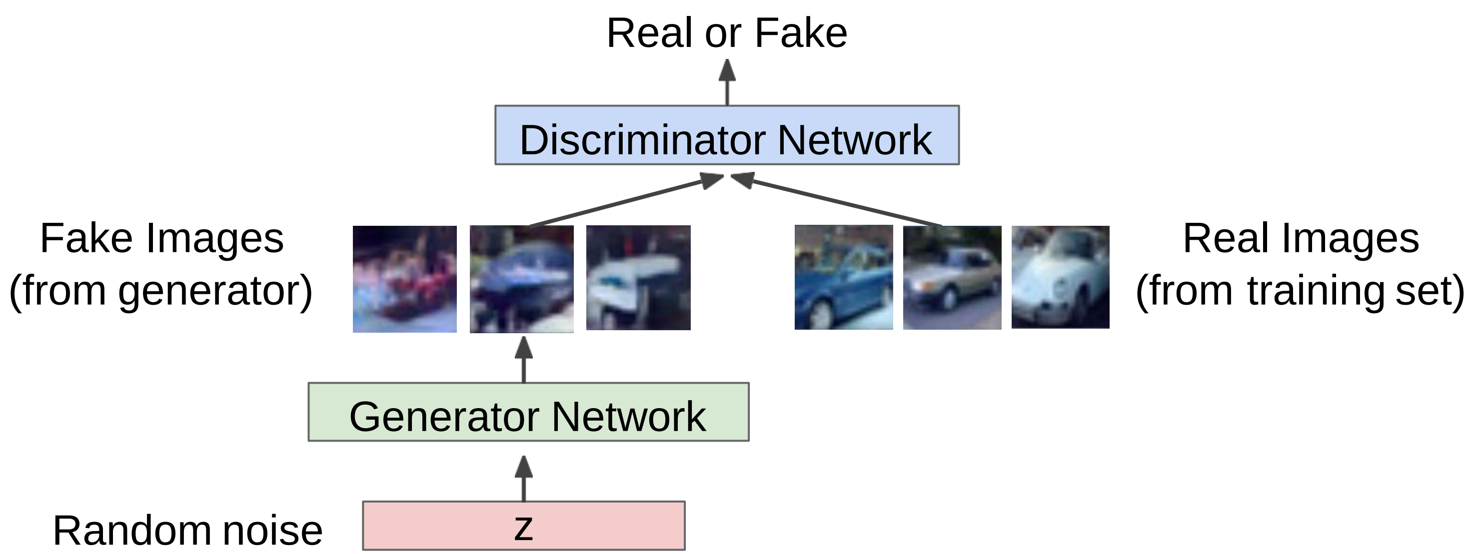
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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

Training GANs: Two-player game

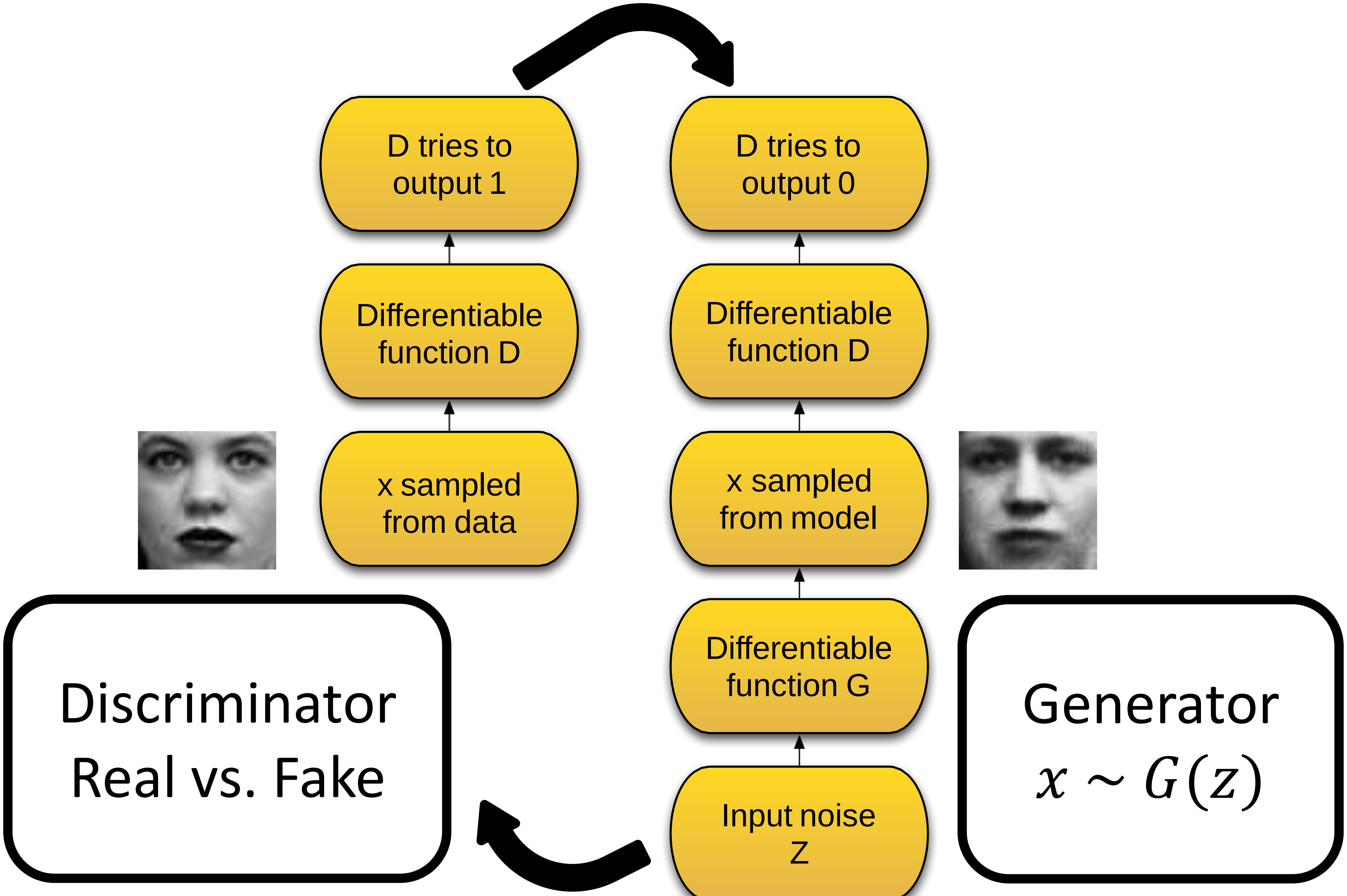
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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



Serena Young

Adversarial Networks Framework



Ian Goodfellow

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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

Train jointly in **minimax game**

Minimax objective function:

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

Training GANs: Two-player game

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

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

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

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

Discriminator output for real data x

Discriminator output for generated fake data G(z)

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two-player game

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

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

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

Training GANs: Two-player game

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

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

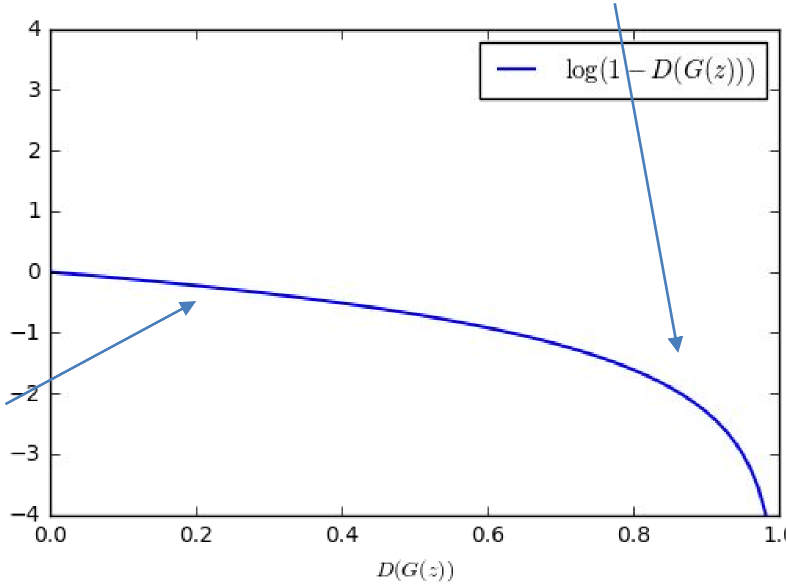
2. **Gradient descent** on generator

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

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Adapted from Serena Young

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

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

Alternate between:

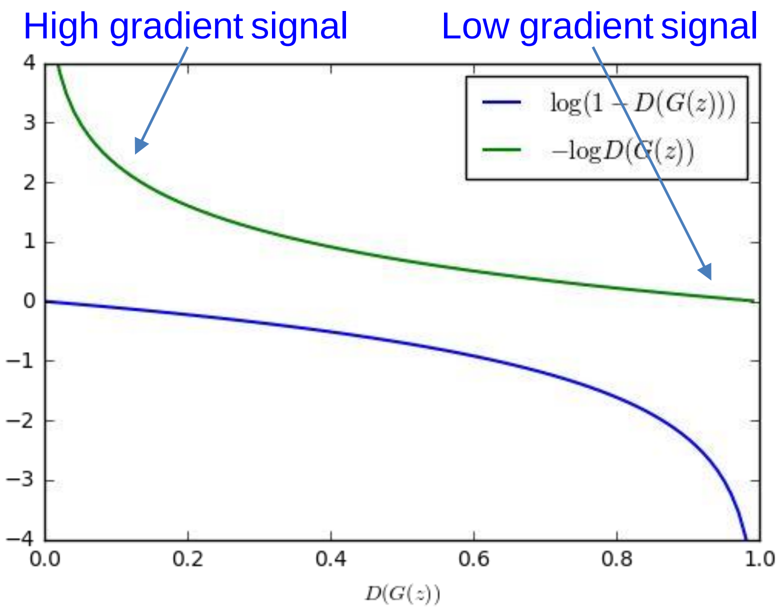
1. **Gradient ascent** on discriminator

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

2. **Instead: Gradient ascent** on generator, different objective

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

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Adapted from Serena Young

Training GANs: Two-player game

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

Putting it together: GAN training algorithm

for number of training iterations **do**

for k steps **do**

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

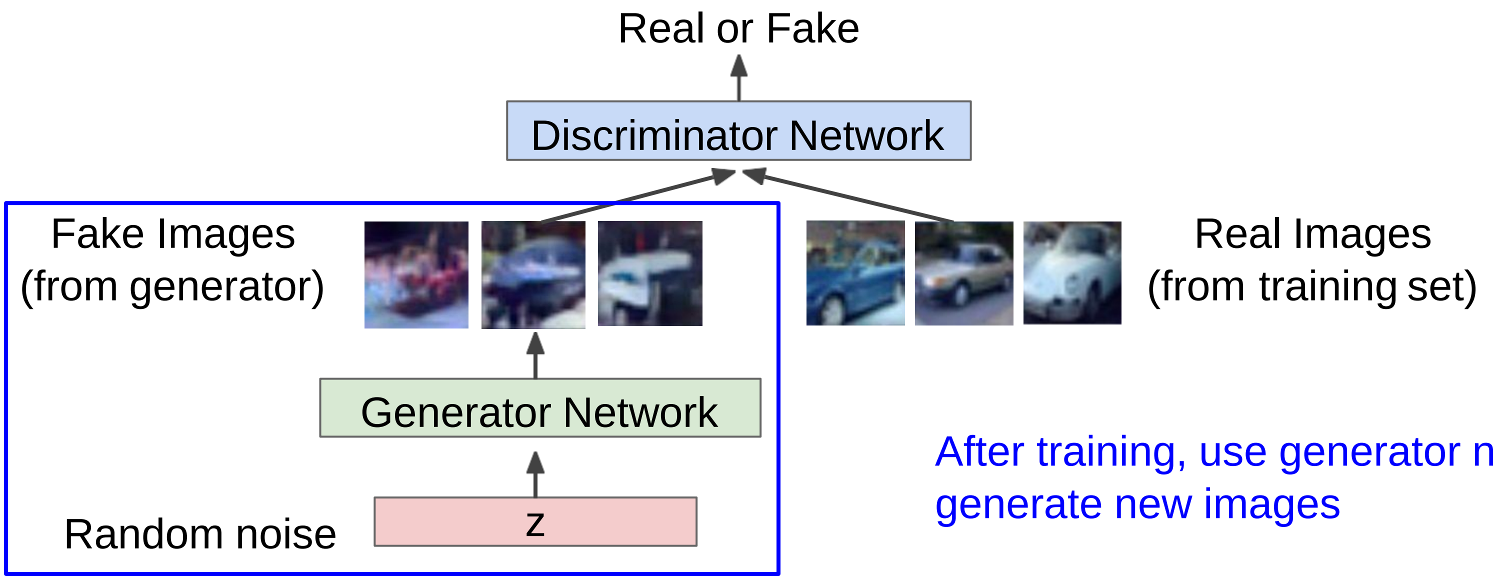
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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



After training, use generator network to generate new images

Alternative loss functions

Name	Paper Link	Value Function
GAN	Arxiv	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	Arxiv	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	Arxiv	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	Arxiv	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	Arxiv	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha x_p))\rvert - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	Arxiv	$L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$ $L_G^{CGAN} = E[\log(D(G(z), c))]$
infoGAN	Arxiv	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	Arxiv	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\mid x)] + E[P(class = c\mid G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\mid G(z))]$
EBGAN	Arxiv	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	Arxiv	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

https://github.com/hwalsuklee/tensorflow-generative-model-collections
https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

GAN training is challenging

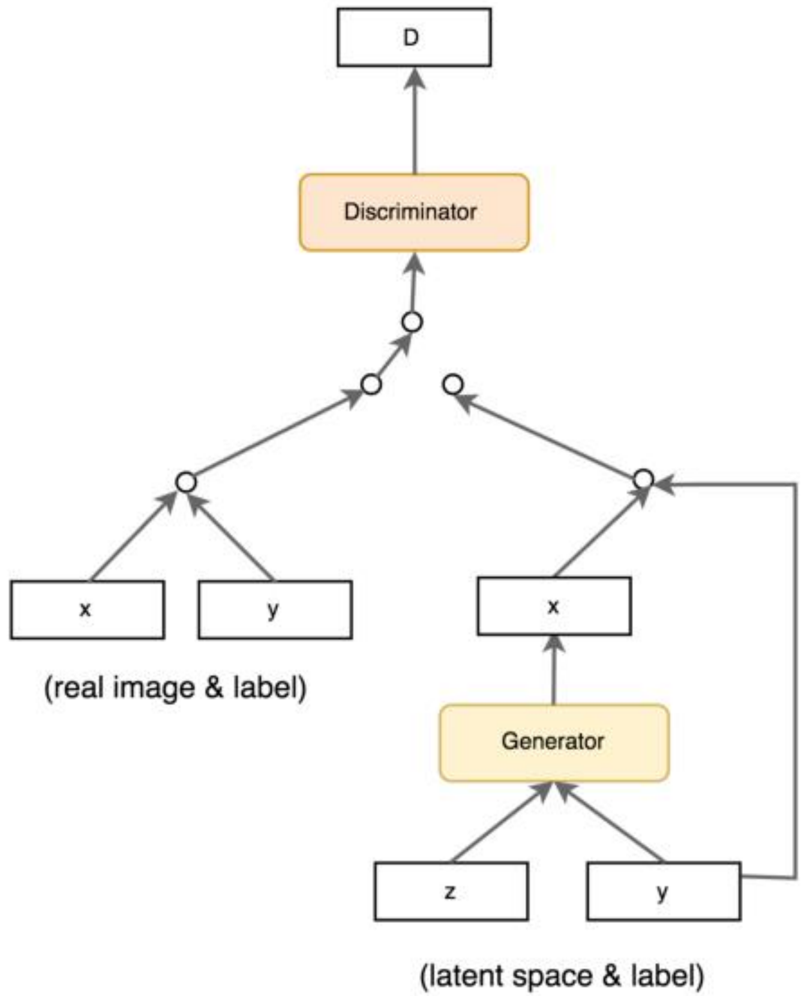
- Vanishing gradient – when discriminator is very good
- Mode collapse – too little diversity in the samples generated
- Lack of convergence because hard to reach Nash equilibrium
- Loss metric doesn’t always correspond to image quality; Frechet Inception Distance (FID) is a decent choice

Tips and tricks

- Use batchnorm, ReLU
- Regularize norm of gradients
- Use one of the new loss functions
- Add noise to inputs or labels
- Append image similarity to avoid mode collapse
- Use labels, extra info when available (CGAN)
- …

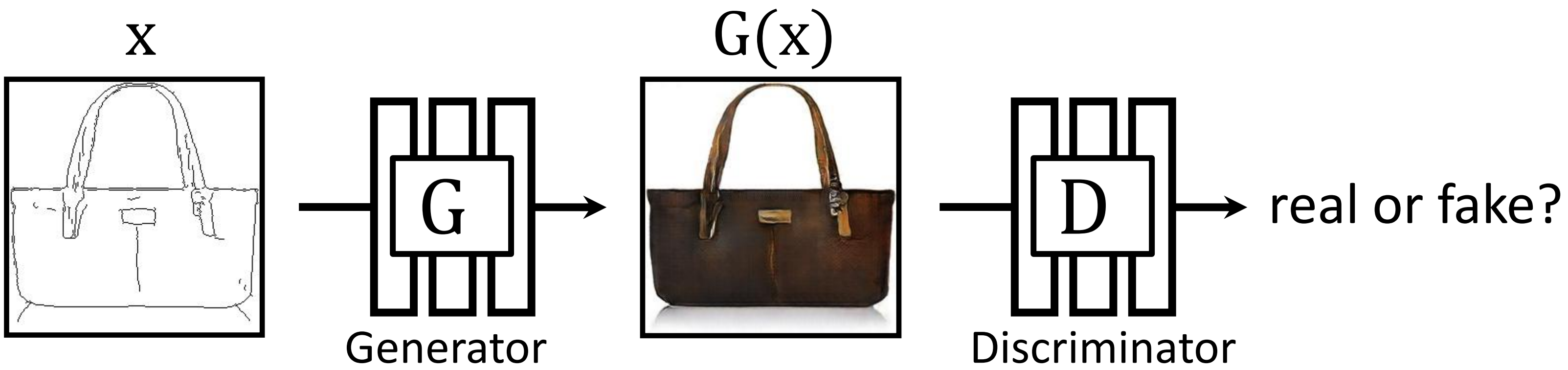
https://github.com/soumith/talks/blob/master/2017-ICCV_Venice/How_To_Train_a_GAN.pdf
https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

Conditional GANs



https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d

GANs

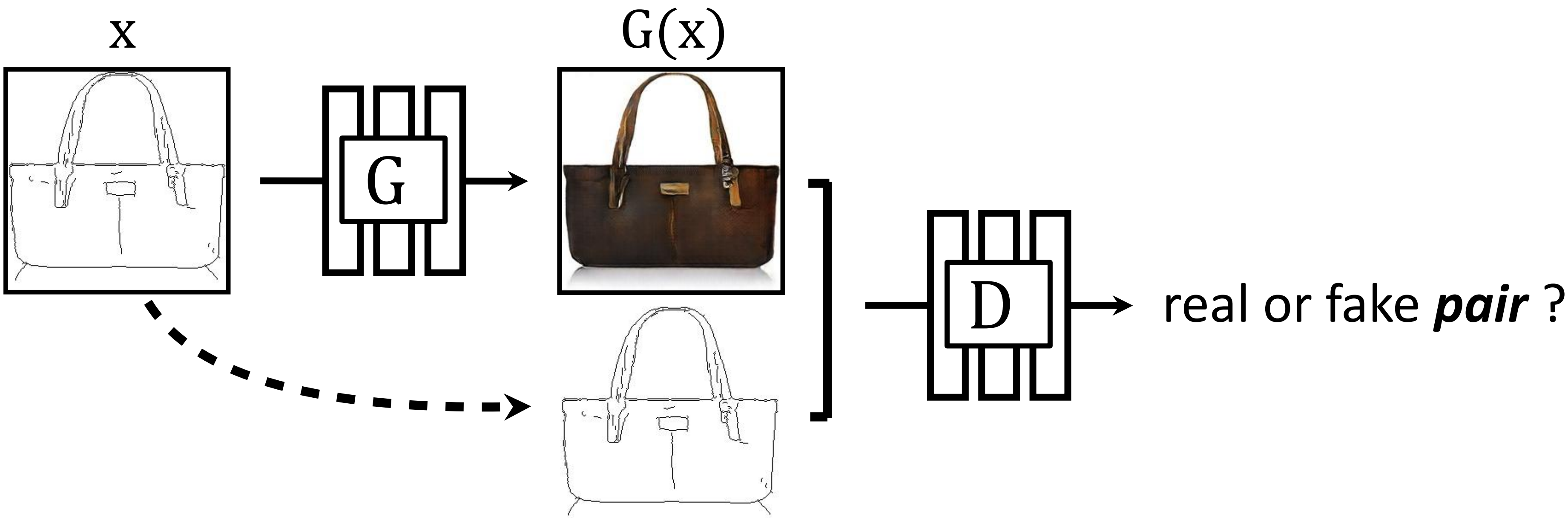


G: generate fake samples that can fool D

D: classify fake samples vs. real images

[Goodfellow et al. 2014]

Conditional GANs



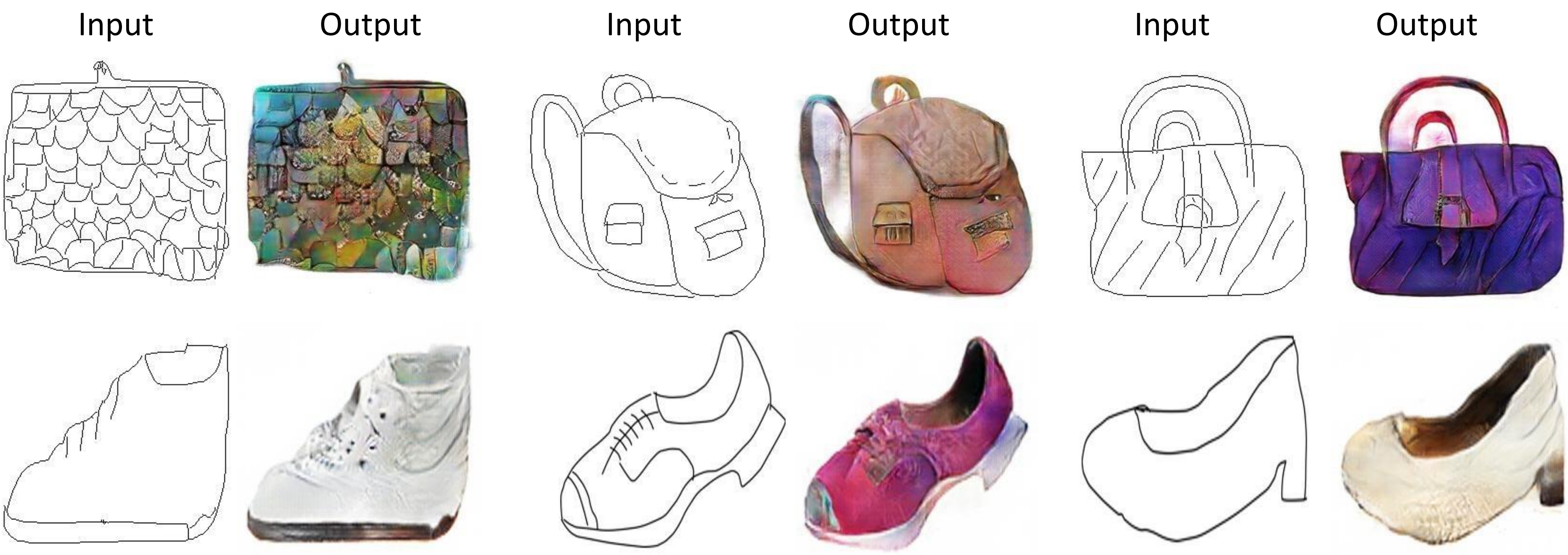
Adapted from Jun-Yan Zhu

Edges → Images

Edges from [Xie & Tu, 2015]

Pix2pix / CycleGAN

Sketches → Images

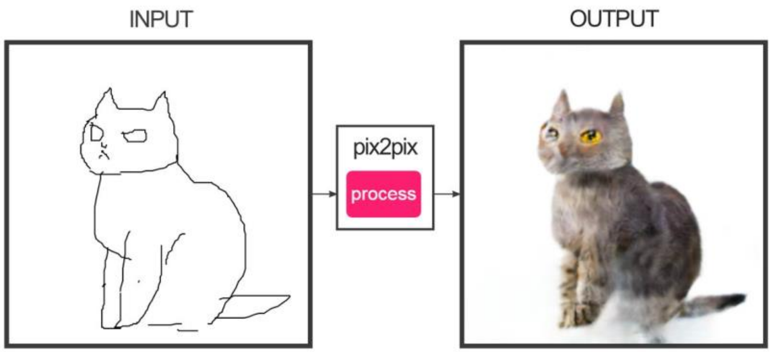


Trained on Edges → Images

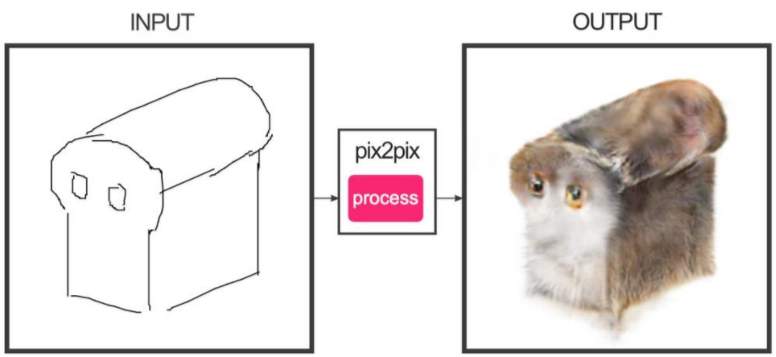
Data from [Eitz, Hays, Alexa, 2012]

Pix2pix / CycleGAN

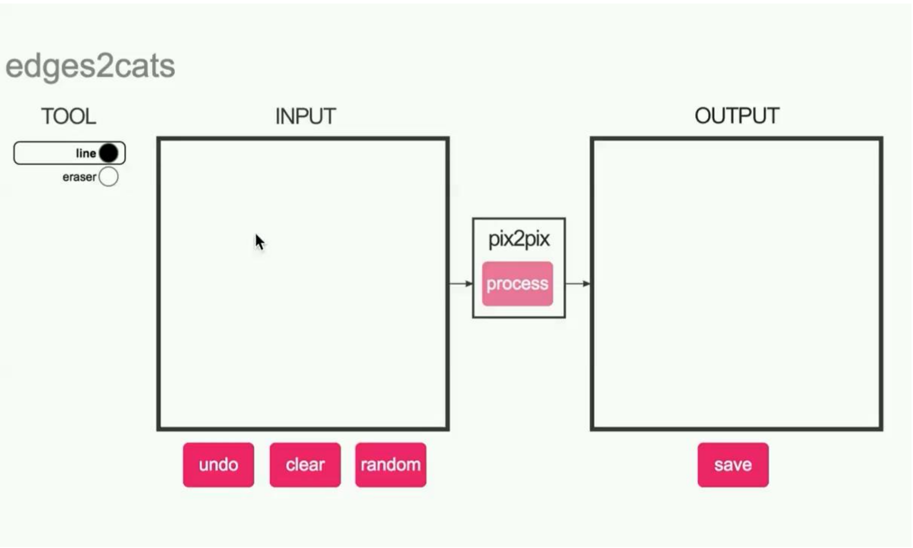
#edges2cats [Christopher Hesse]



@gods_tail



Ivy Tasi @ivymyt

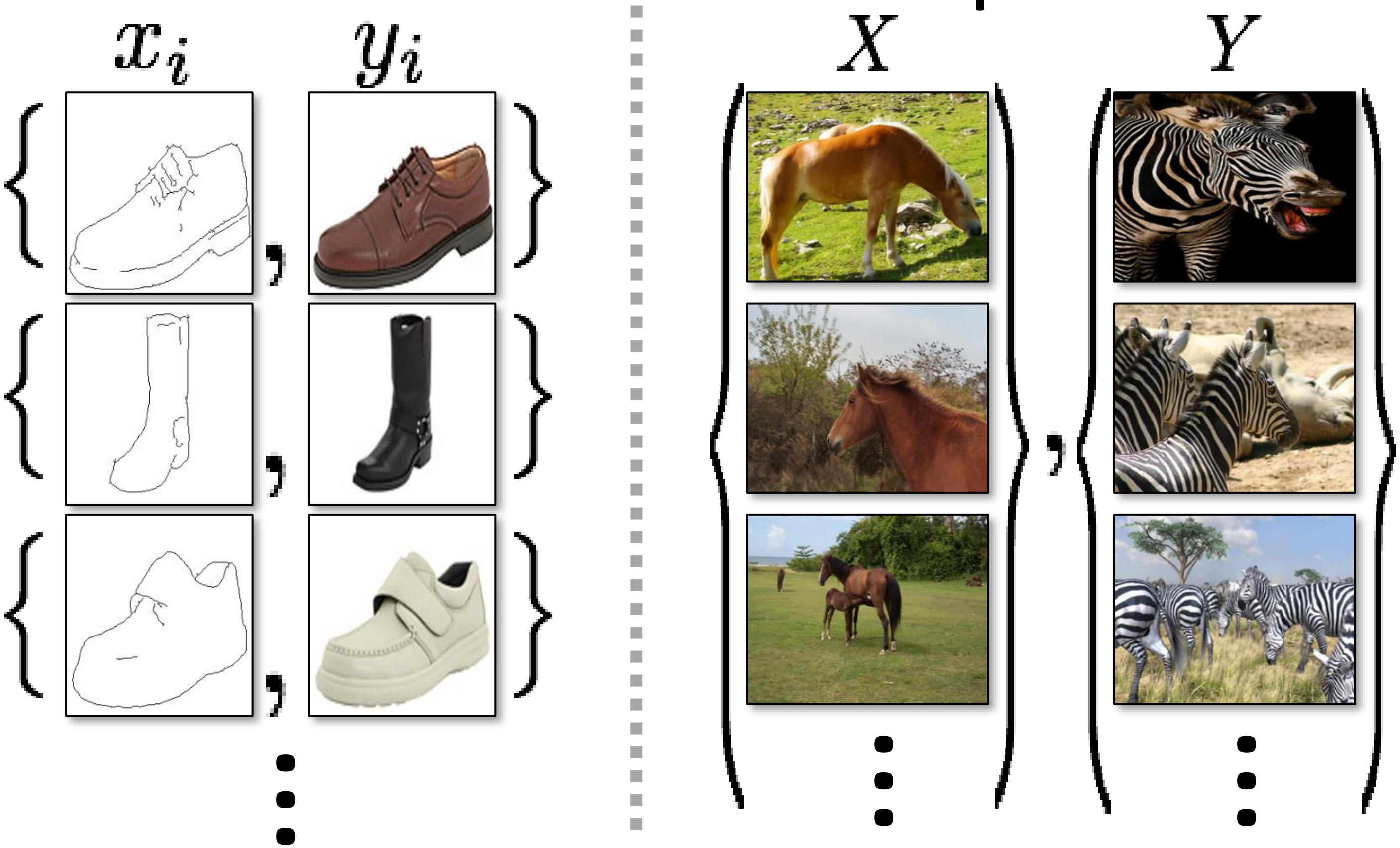


@matthematician

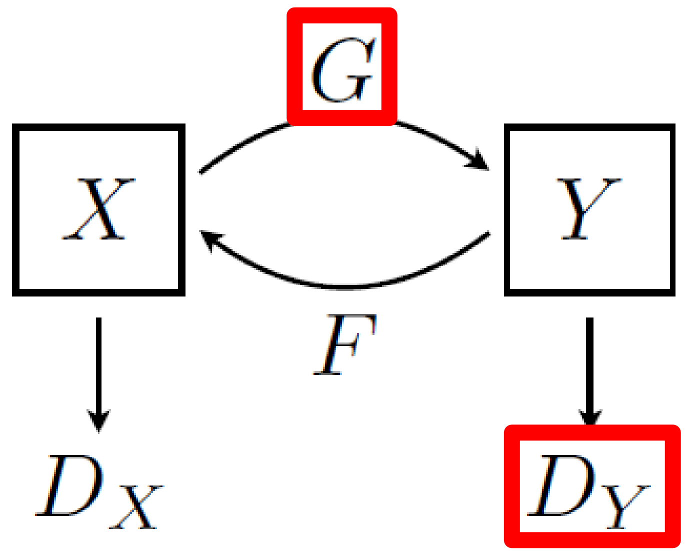
Vitaly Vidmirov @vvid

https://affinelayer.com/pixsrv/

Paired Unpaired



Cycle Consistency

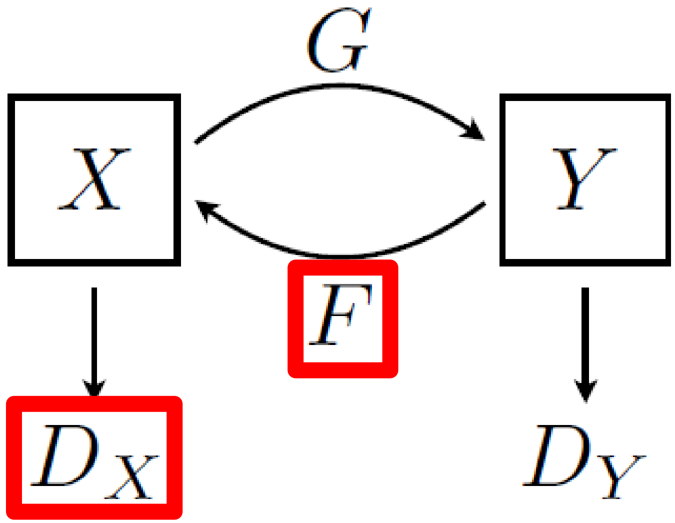


Discriminator D_Y: $L_{GAN}(G(x), y)$

Real zebras vs. generated zebras

Zhu et al., "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks", ICCV 2017

Cycle Consistency

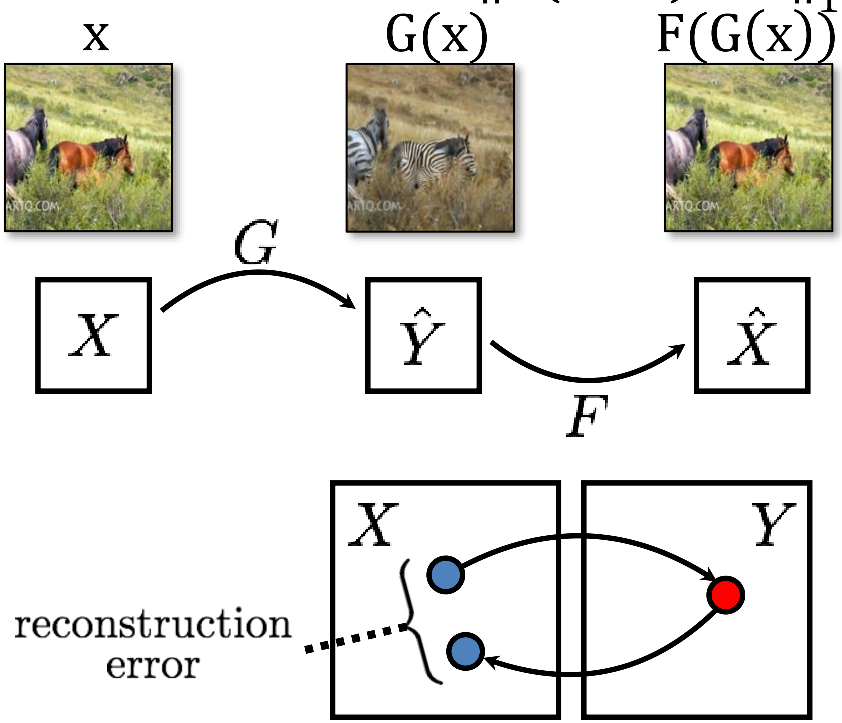


Discriminator D_Y: $L_{GAN}(G(x), y)$
Real zebras vs. generated zebras
Discriminator D_X: $L_{GAN}(F(y), x)$
Real horses vs. generated horses

Zhu et al., "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks", ICCV 2017

Cycle Consistency

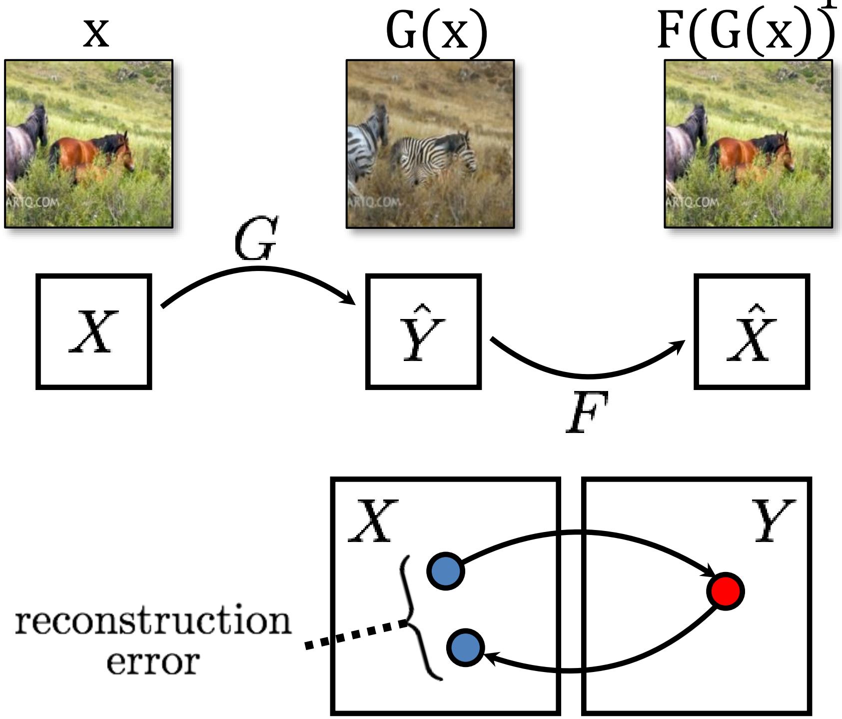
Forward cycle loss: $\left\| F(G(x)) - x \right\|_1$



Zhu et al., "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks", ICCV 2017

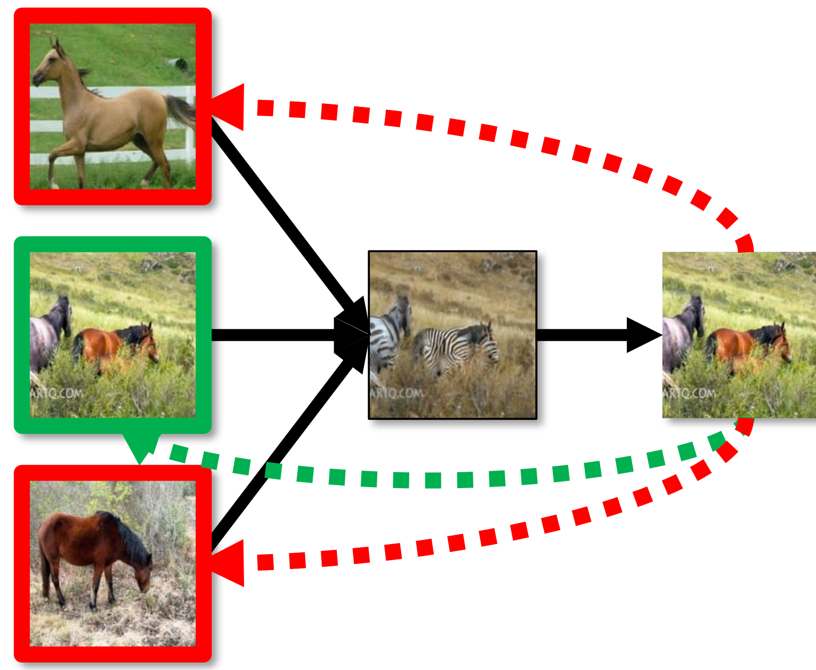
Cycle Consistency

Forward cycle loss: $\left\| F(G(x)) - x \right\|_1$



Small cycle loss

Large cycle loss



Helps cope with mode collapse

Zhu et al., "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks", ICCV 2017

Training Details: Objective

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))],$$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

$$G^*, F^* = \arg \min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y).$$

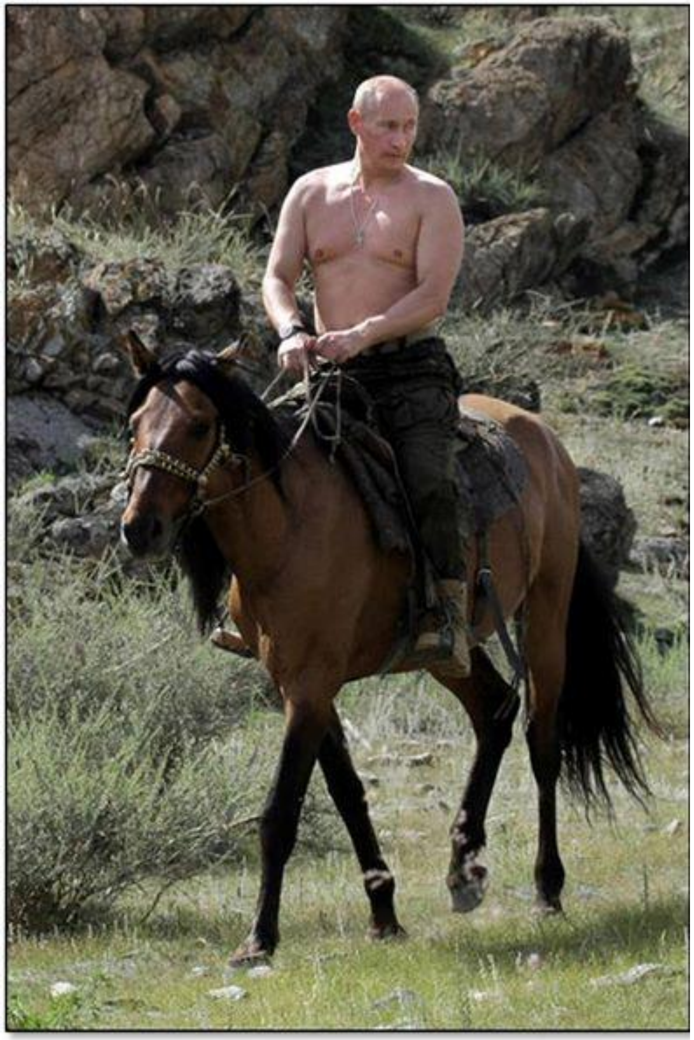
Zhu et al., "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks", ICCV 2017

Pix2pix / CycleGAN

Pix2pix / CycleGAN

Pix2pix / CycleGAN

Pix2pix / CycleGAN



Pix2pix / CycleGAN

Pix2pix / CycleGAN

Celebrities Who Never Existed

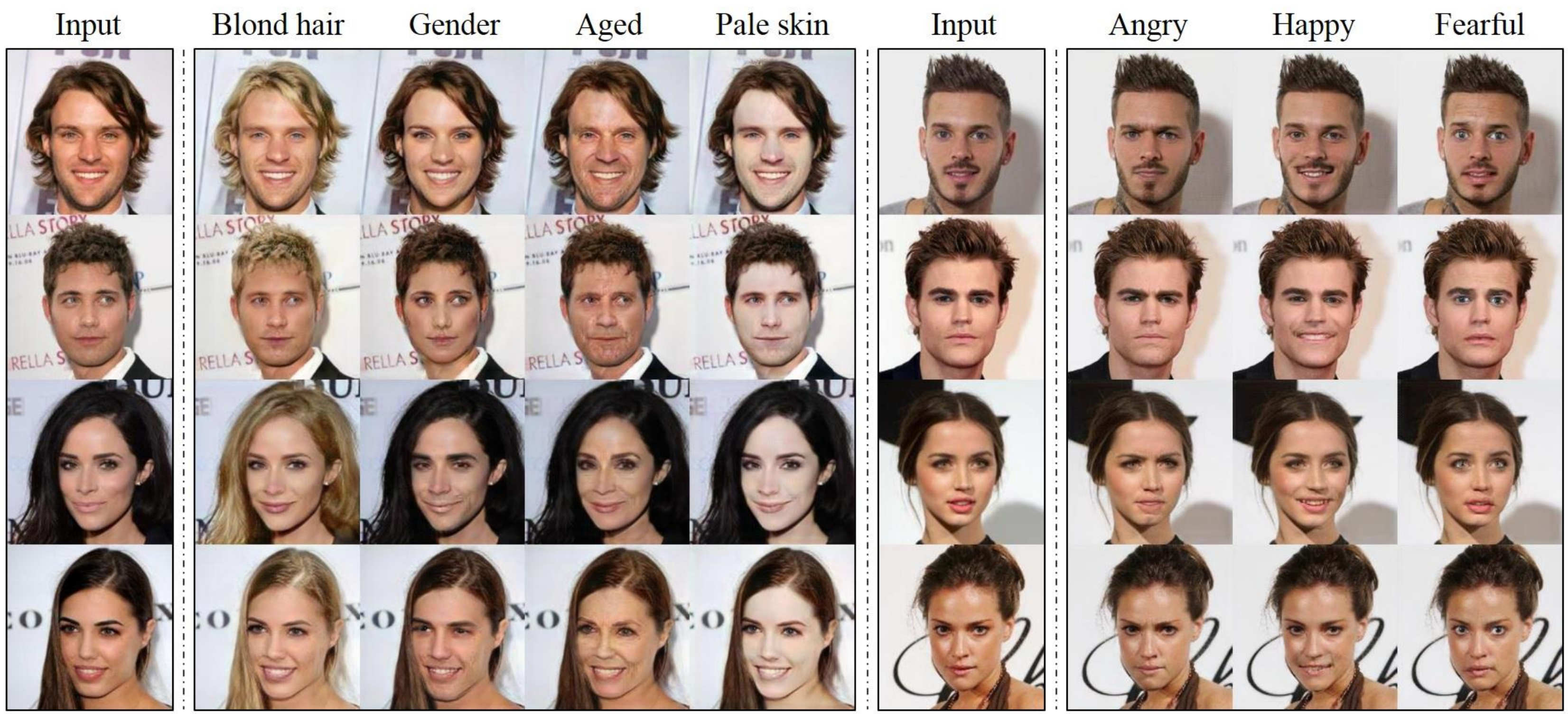
Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

Creative Adversarial Networks

CAN: Top ranked by human subjects

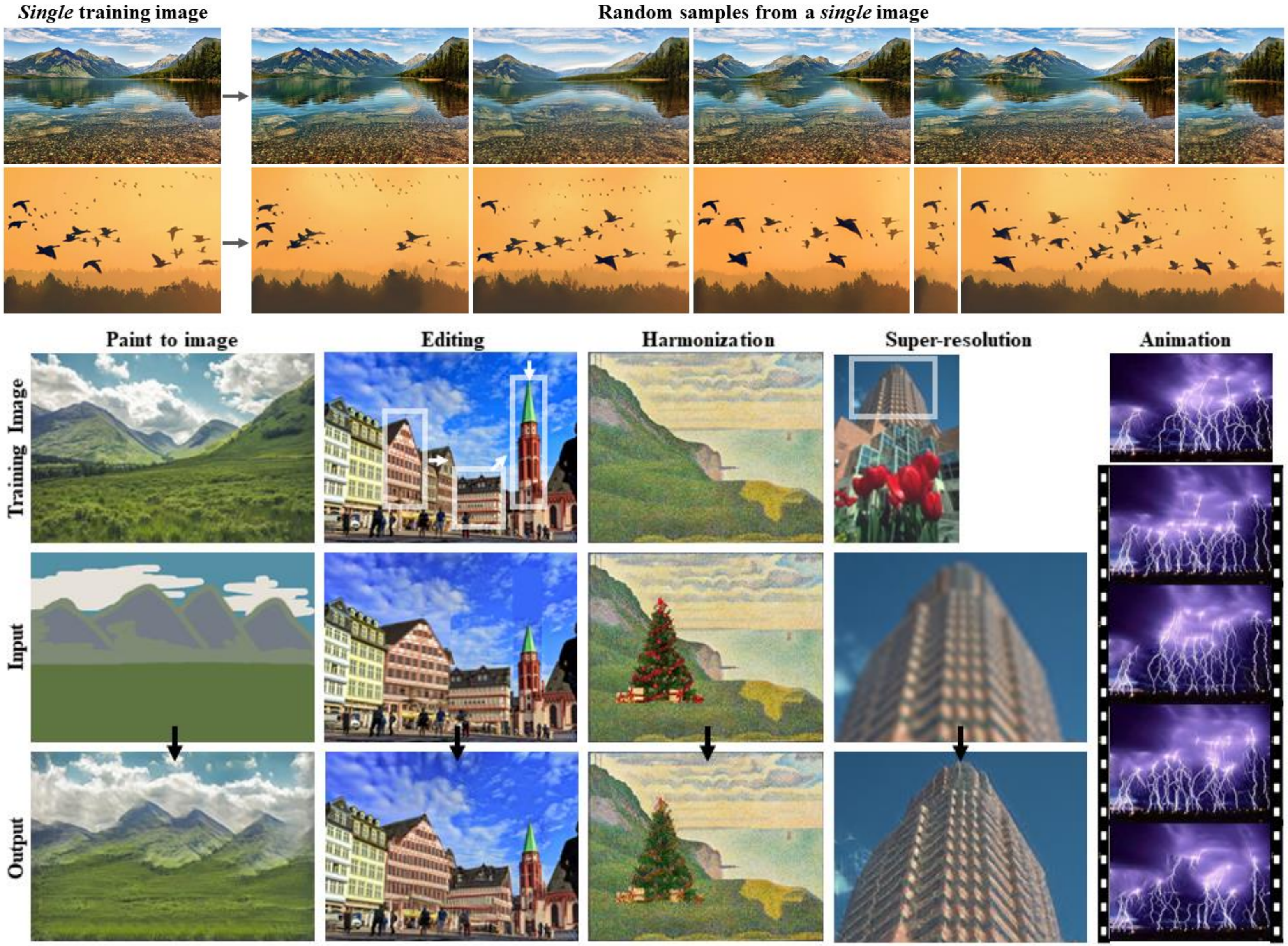
(Elgammal et al., 2017)

StarGAN



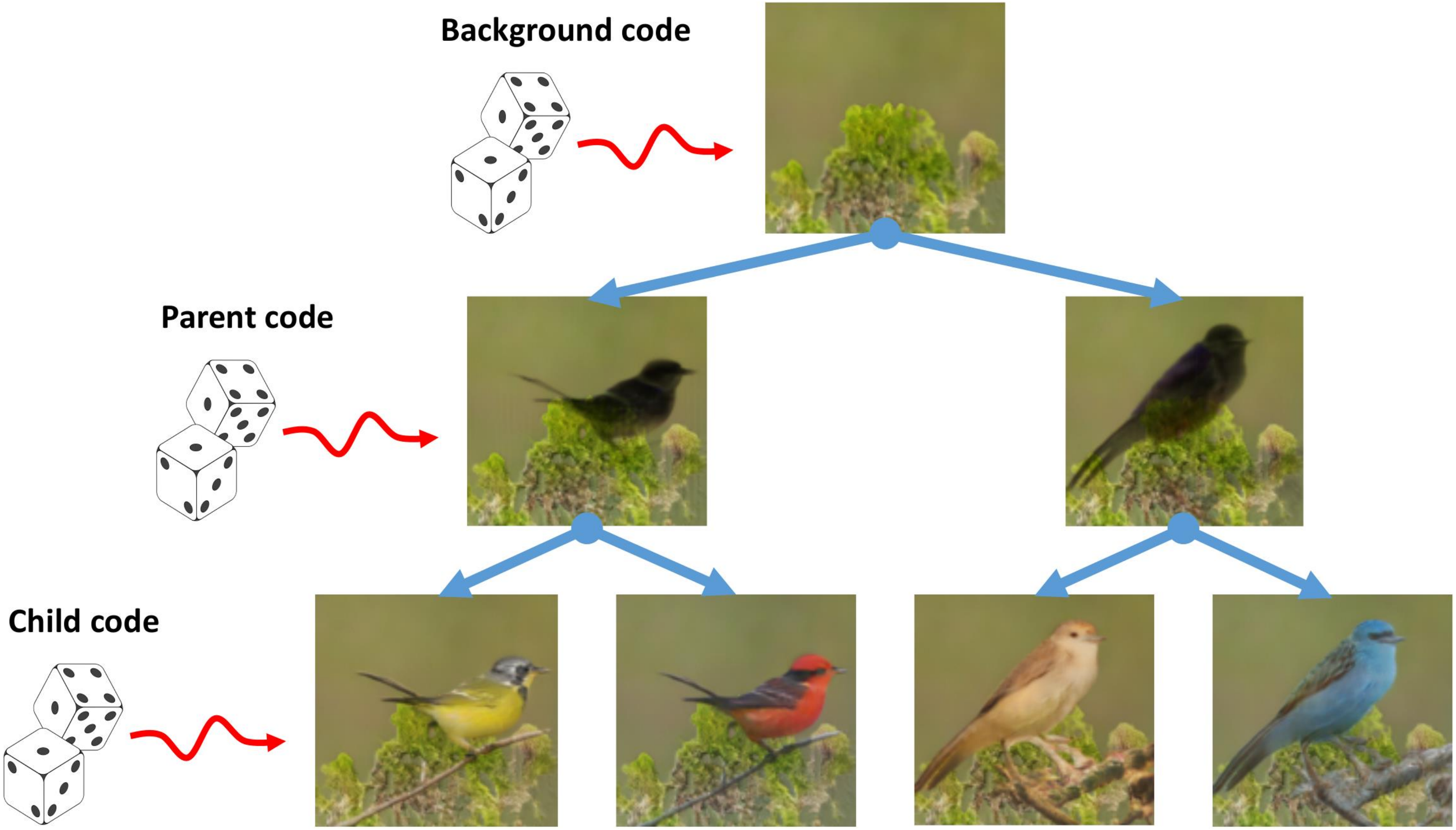
Choi et al., “StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation”, CVPR 2018

SinGAN



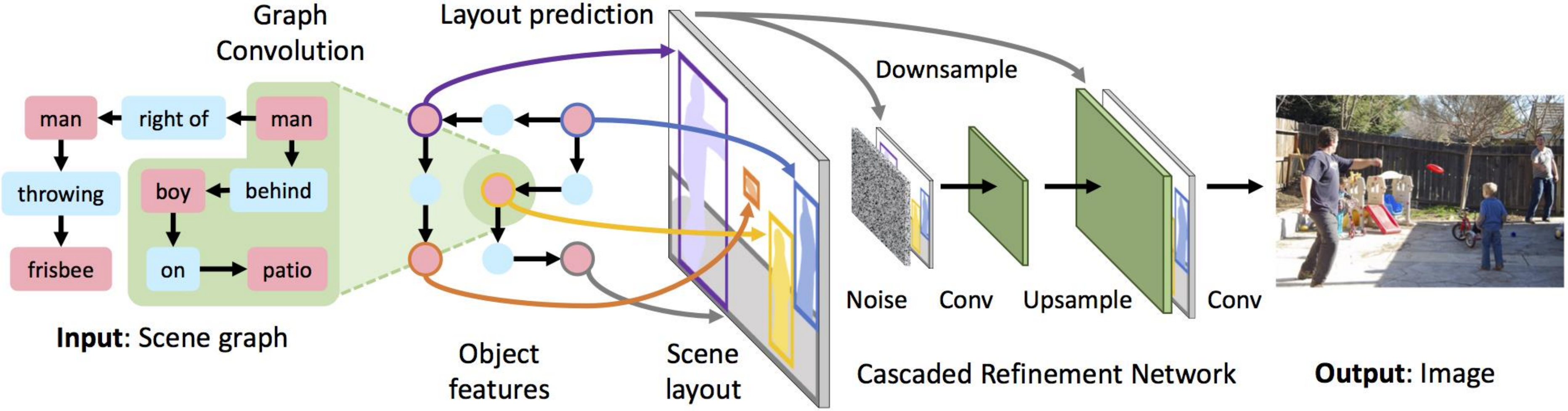
Shaham et al., "SinGAN: Learning a Generative Model from a Single Natural Image", ICCV 2019

Stagewise generation



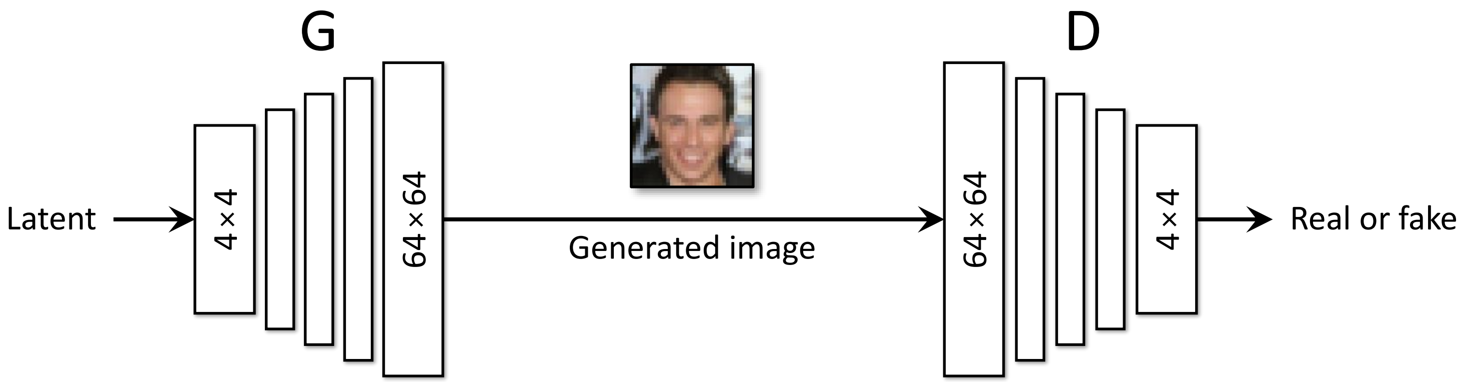
Singh et al., "FineGAN: Unsupervised Hierarchical Disentanglement for Fine-Grained Object Generation and Discovery", CVPR 2019

Stagewise generation



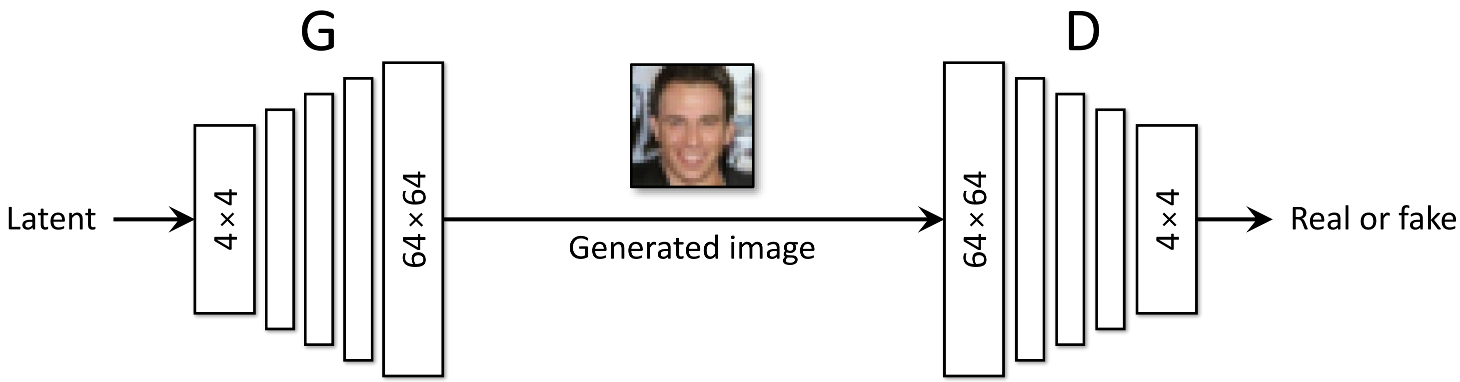
Johnson et al., "Image Generation from Scene Graphs", CVPR 2018

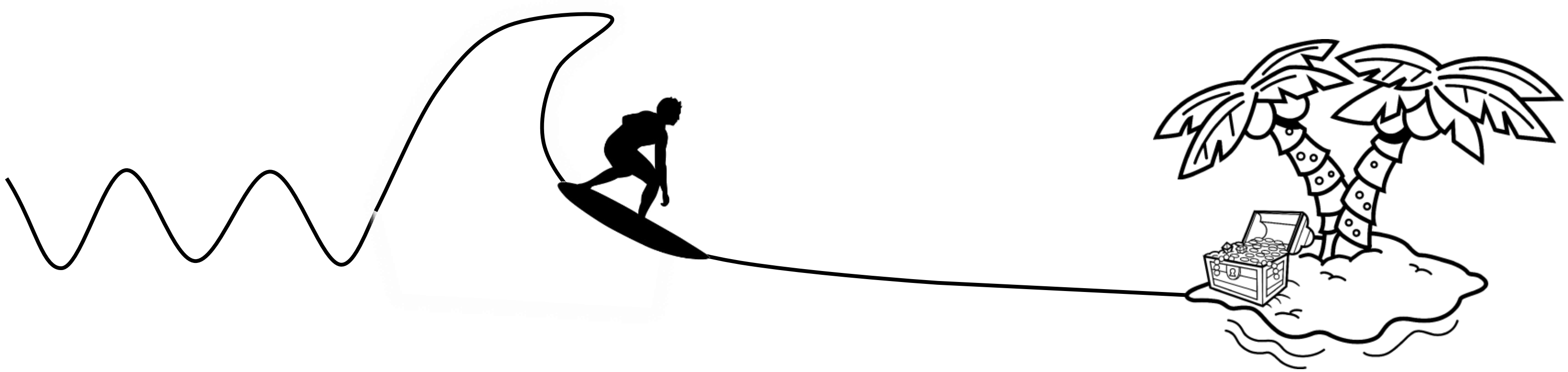
Progressive generation



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

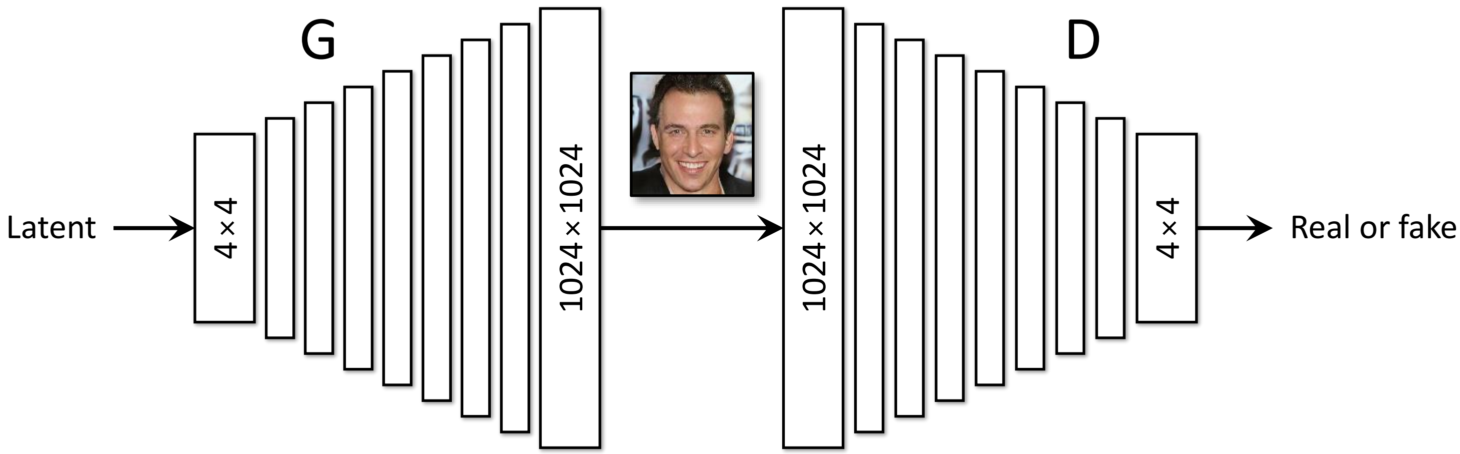
Progressive generation

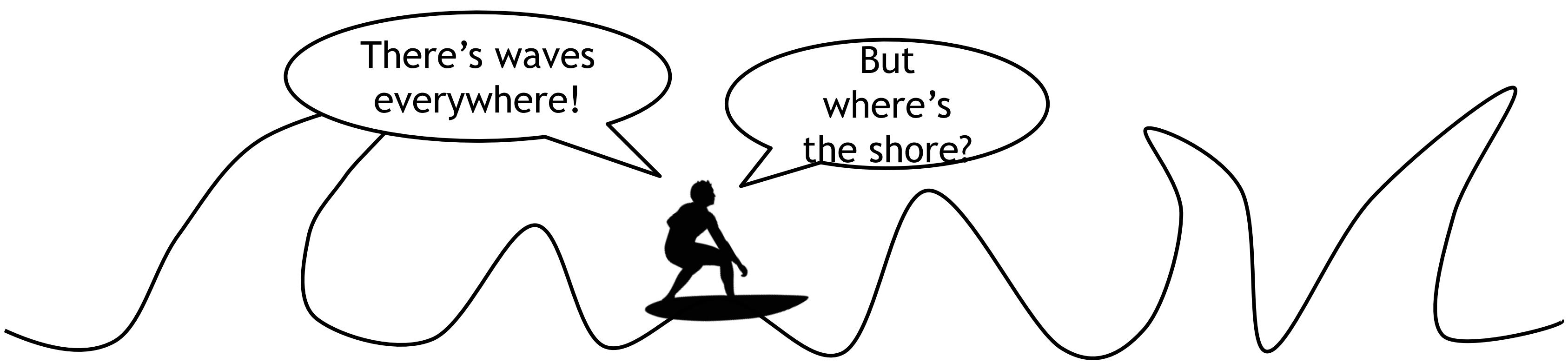




Karras et al., "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

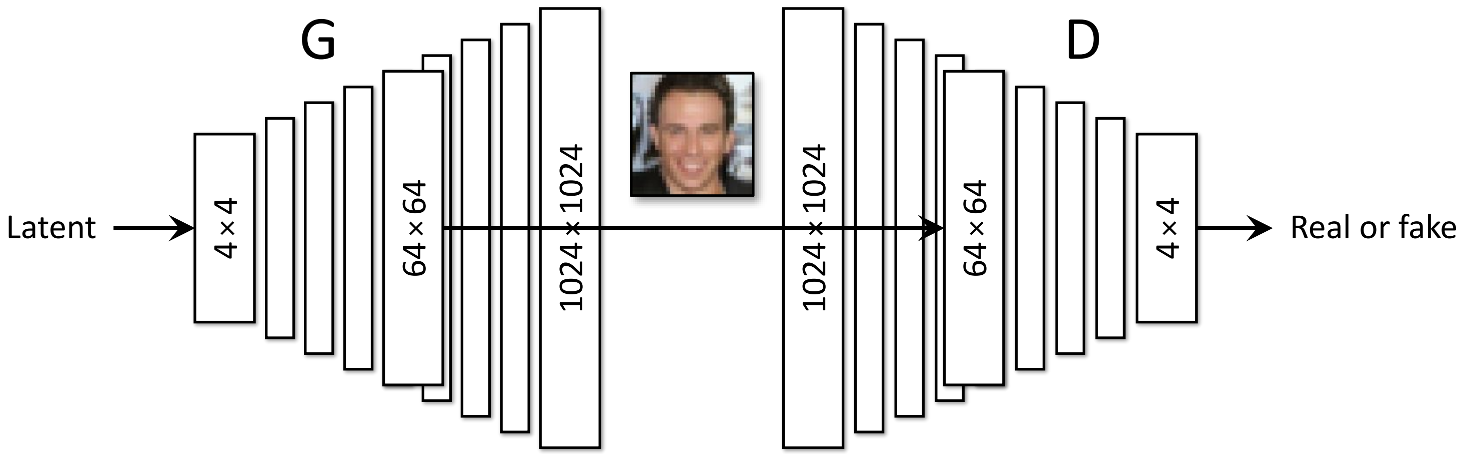
Progressive generation

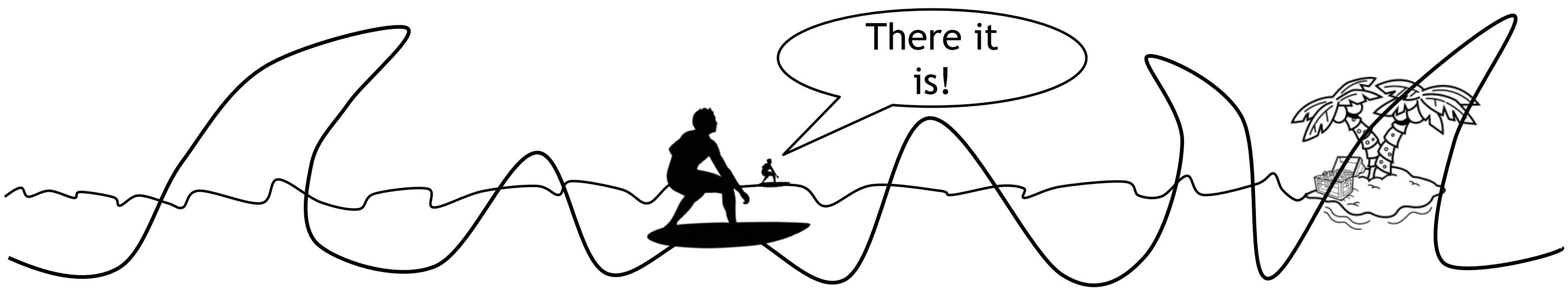




Karras et al., "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

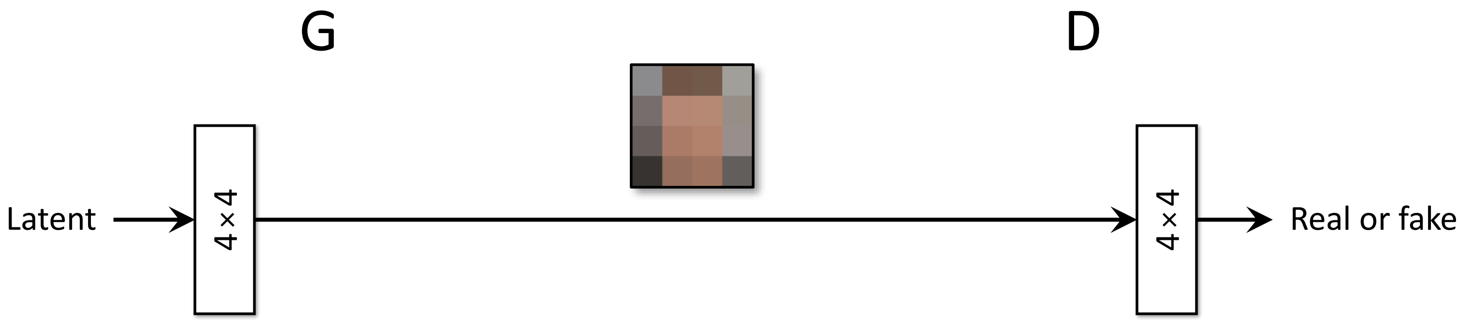
Progressive generation

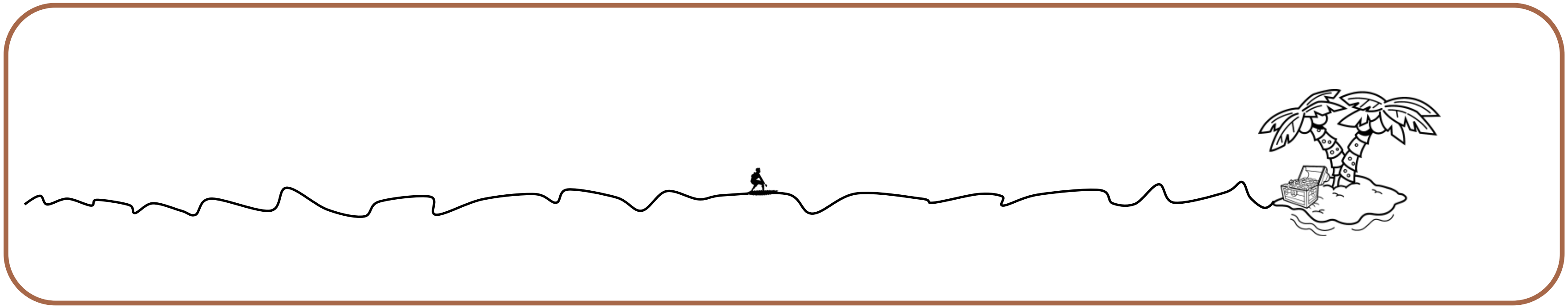




Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

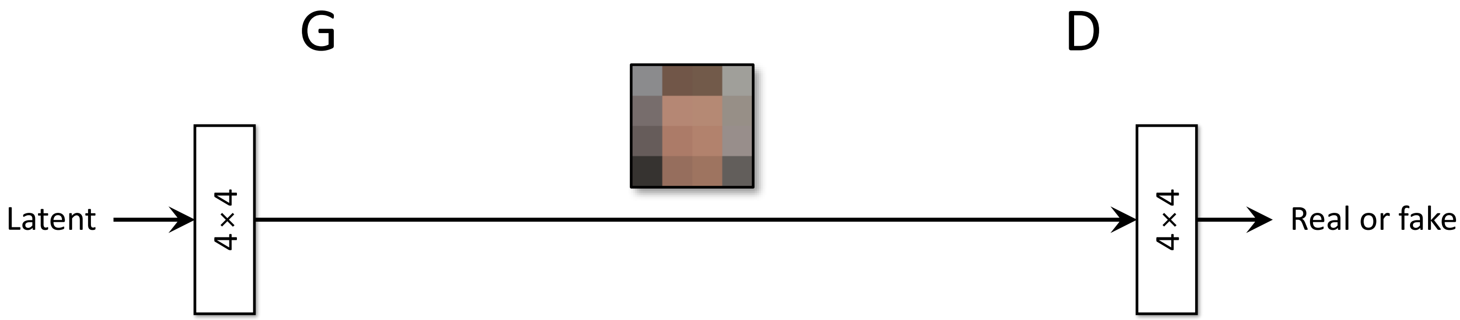
Progressive generation

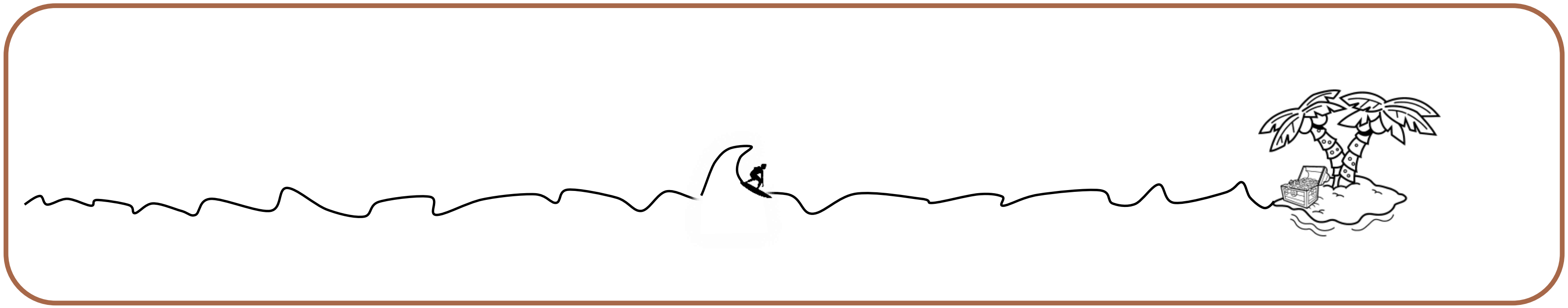




Karras et al., "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

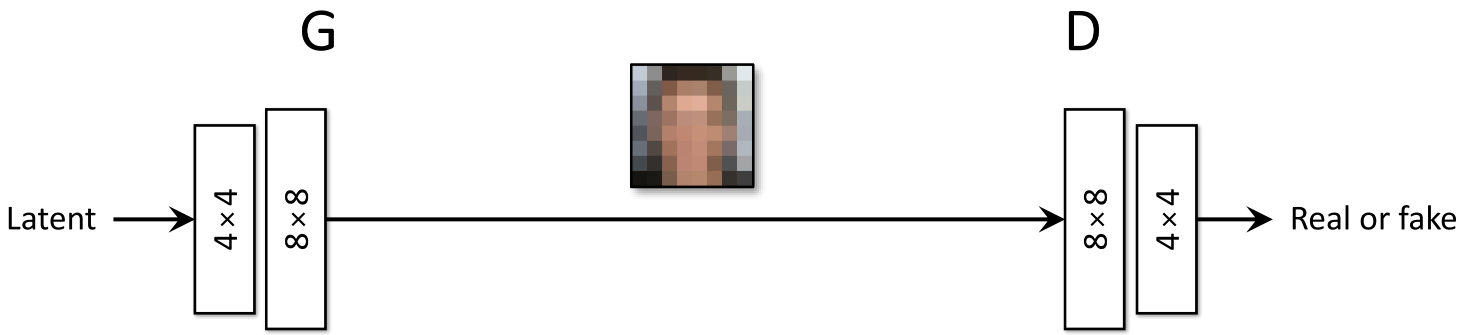
Progressive generation

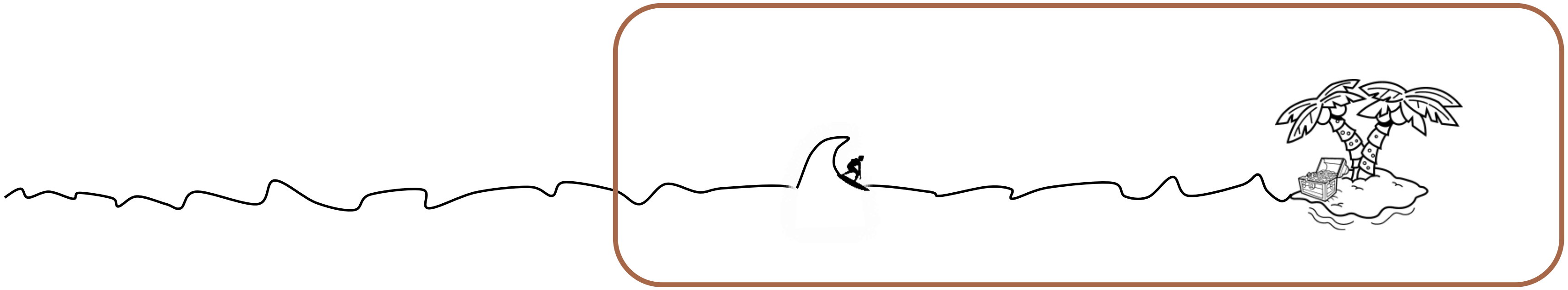




Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

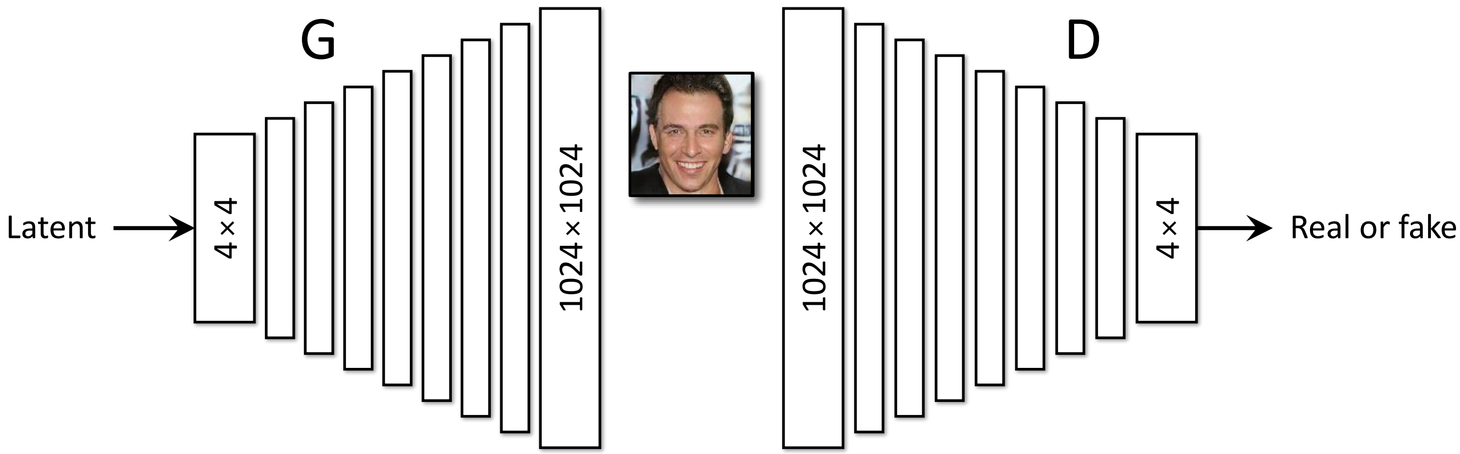
Progressive generation

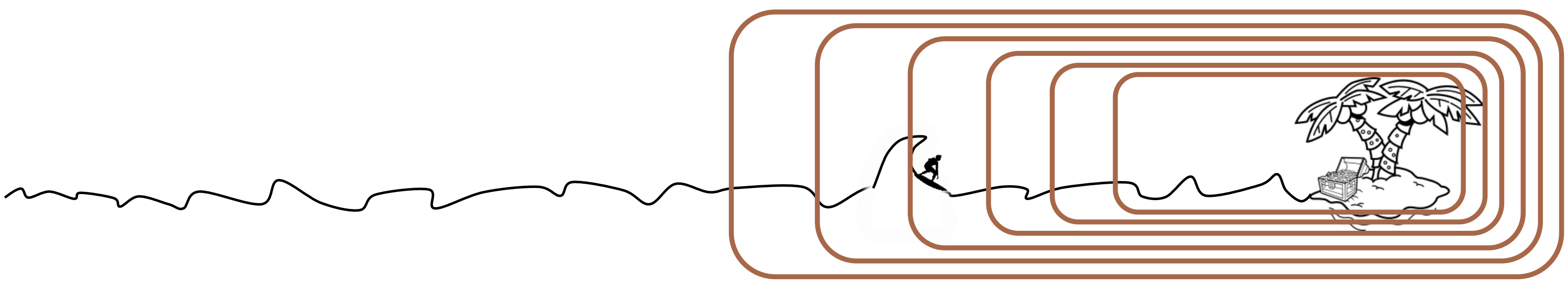




Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

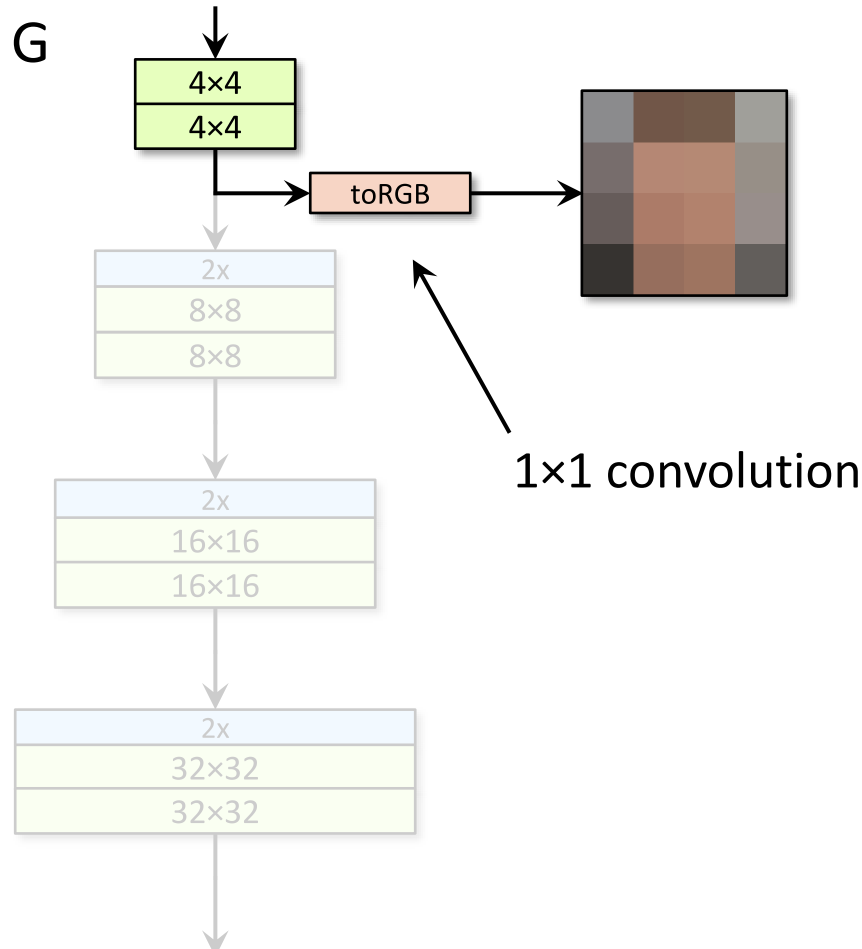
Progressive generation





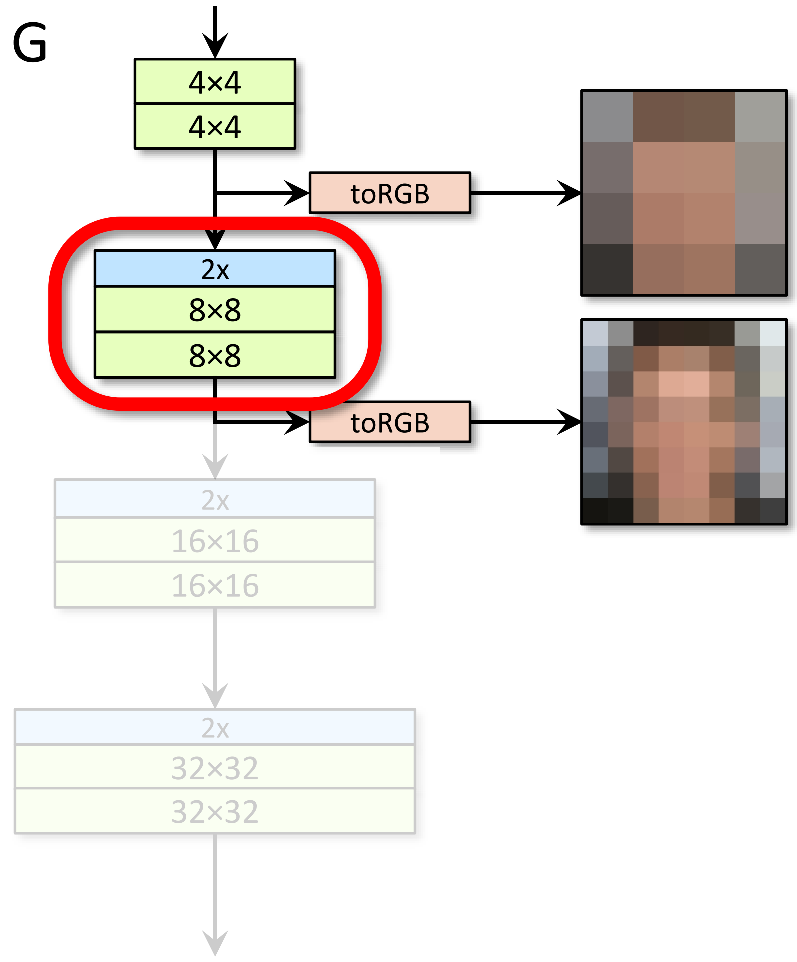
Karras et al., "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

Progressive generation



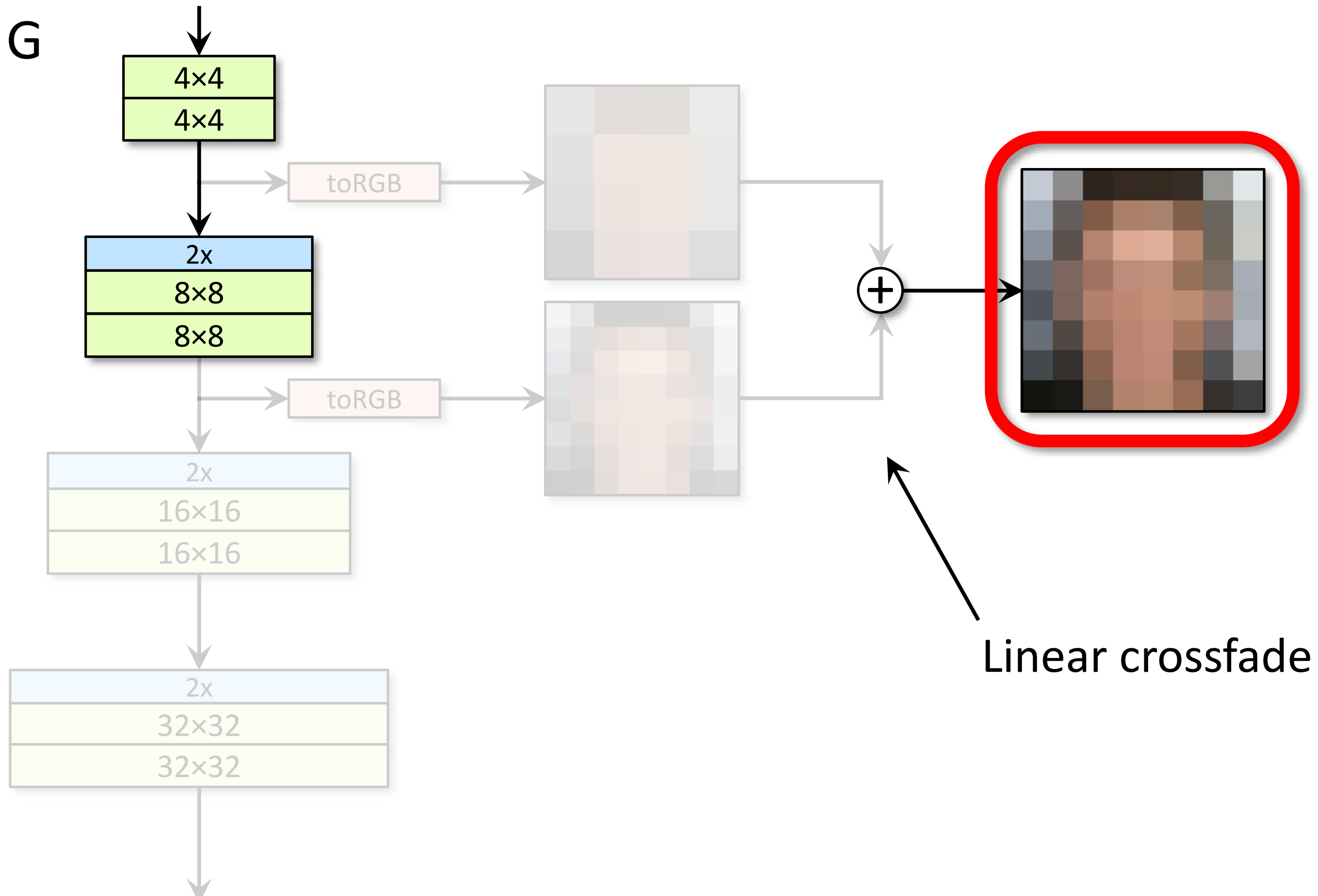
Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

Progressive generation



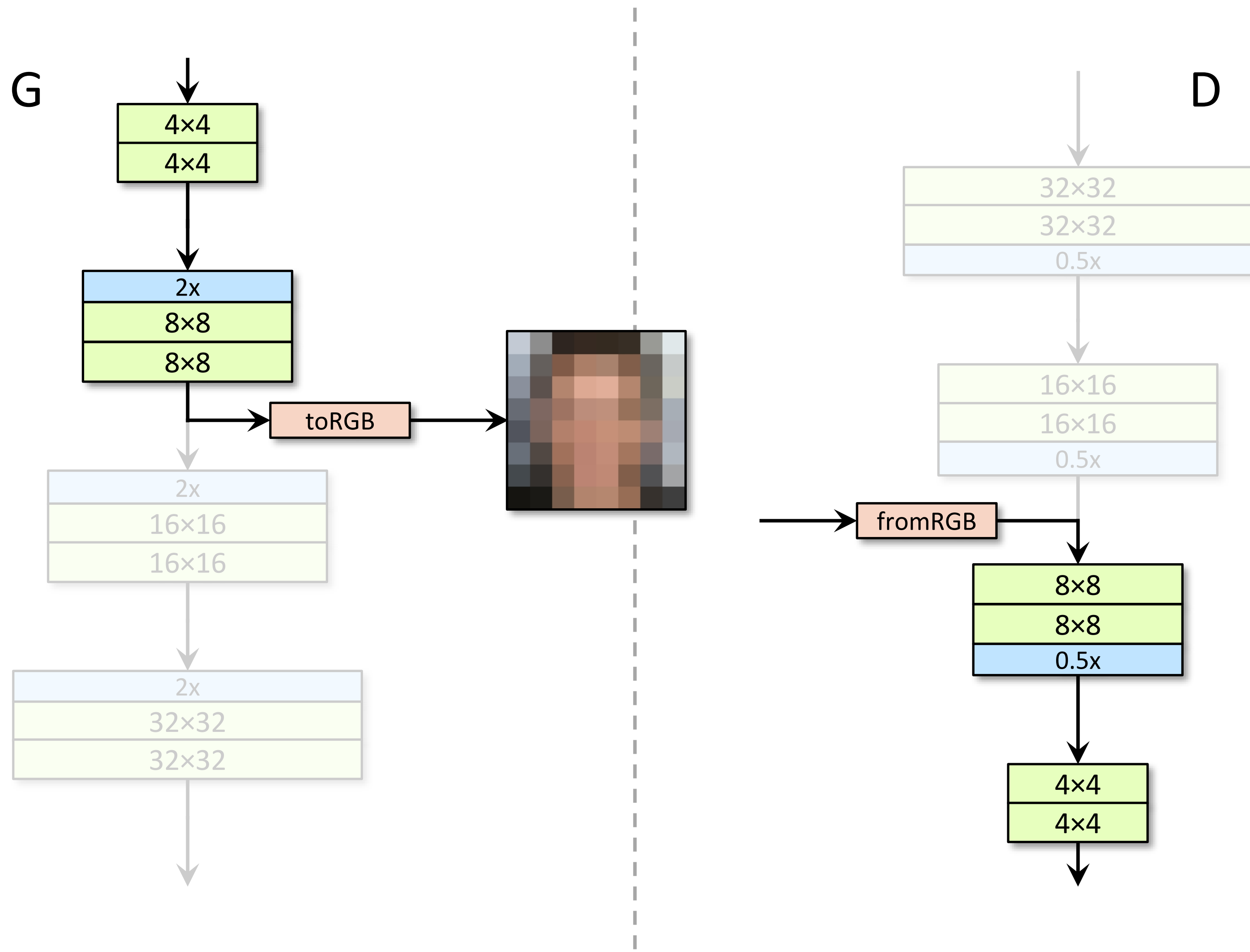
Karras et al., "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

Progressive generation



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

Progressive generation



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

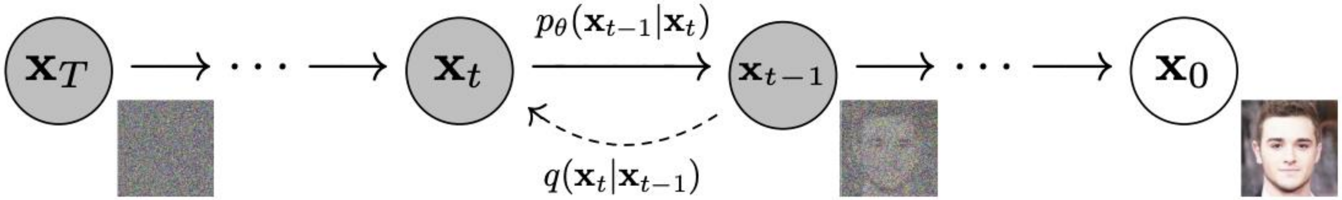
Diffusion models - Motivation

- Generative Adversarial Neural Network (GAN)
  - Difficult to optimize
- Variational Autoencoder (VAE)
  - Efficient comparing to GAN, but synthesis quality is moderate
- Denoising diffusion probabilistic models (DPPM) achieve state-of-the-art image synthesis results
  - Costly in training and inference
- Latent diffusion models (LDM) – CVPR 2022

Adapted from Haolin Zhang

Denoising Diffusion Probabilistic Model (DDPM)

- Denoising Diffusion Probabilistic Model (DDPM)
  - Diffusion/forward process $q(x_t|x_{t-1})$
  - Denoising/reverse process $p(x_{t-1}|x_t)$
  - Both processes are Markov Chain process: predictions can be made regarding future outcomes based solely on its present state



Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Denoising Diffusion Probabilistic Model (DDPM)

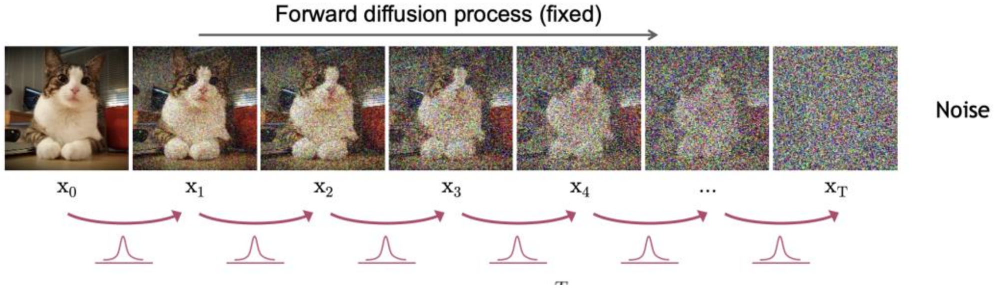
- **Model the diffusion/forward process**
  - Define:

$$q(x_t|x_{t-1}) = \mathrm{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathrm{I}),\ \beta_t$$ → hyperparameter

  - Based on Markov Chain process:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

- Based on the above definitions, $q(x_t)$ at arbitrary timestep t can be derived purely by β_t and x_0



Ref: Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in neural information processing systems* 33 (2020): 6840-6851.
CVPR 2022 Tutorial: Denoising Diffusion-based Generative Modeling: Foundations and Applications. https://cvpr2022-tutorial-diffusion-models.github.io/

Denoising Diffusion Probabilistic Model (DDPM)

$$q(x_t|x_{t-1}) = \mathrm{N}(x_t;\sqrt{1-\beta_t}x_{t-1},\beta_t\mathrm{I}) \qquad q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

- Based on the above definitions, $q(x_t)$ at arbitrary timestep t can be derived purely by β_t and x_0
  - Define: $\alpha_t = 1-\beta_t$, and $\overline{\alpha_t} = \prod_{i=1}^{T} \alpha_i$

1.Reparameterization $x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon$, ϵ is sampled from $\mathrm{N}(0,\mathrm{I})$

2.Write in form of x_{t-2} $= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t}\ \epsilon + \sqrt{\alpha_t}\sqrt{1-\alpha_{t-1}}\ \epsilon,$

3.Simplify through addition property $= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\ \epsilon$

4.Repeat till x0 $= \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}\ \epsilon$

- By addition property of gaussian distributions which states for two gaussian distributions

$$X \sim N(\mu_X, \sigma_X^2)$$
$$Y \sim N(\mu_Y, \sigma_Y^2)$$
$$Z = X + Y,$$

then

$$Z \sim N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2).$$

$\sqrt{1-\alpha_t}\ \epsilon + \sqrt{\alpha_t}\sqrt{1-\alpha_{t-1}}\ \epsilon$ can be sampled from $N(0,(1-\alpha_t\alpha_{t-1})\mathrm{I})$

Thus, given β_t, and x_0, the diffused sample at any arbitrary time step can be modeled by the above equation

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

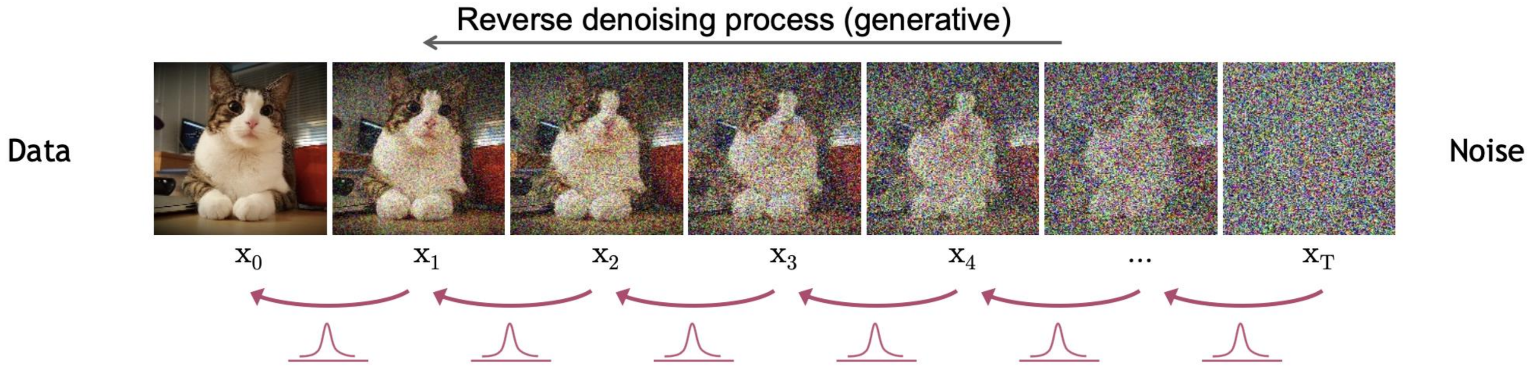
Denoising Diffusion Probabilistic Model (DDPM)

- **Reverse denoising process**
  - **After diffusion process at time step $\boldsymbol{T}$, $\boldsymbol{p(x_T) = \mathrm{N}(x_T; 0, I)}$** Noise

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$ Markov process

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

Authors propose to untrain $\Sigma_\theta(x_t, t)$ by setting $\Sigma_\theta(x_t, t) = \sigma_t^2$, where $\sigma_t^2 = \beta_t$



Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Recall: Variational Lower Bound

x = data
h = hidden representation

q = encoder (data to noise)
p = decoder (noise to data)

$$\begin{aligned} D_{\mathrm{KL}}(q(h|x) \parallel p(h|x)) &= \mathbb{E}_q[\log q(h|x) - \log p(h|x)] \\ &= \mathbb{E}_q[\log q(h|x) - \log p(x,h) + \log p(x)] \\ &= \mathbb{E}_q[\log q(h|x) - \log p(x,h)] + \log p(x) \end{aligned}$$

Notice that the expectation is the sign-flipped version ELBO term we derived above.

ELBO = $\mathbb{E}_q[\log p(x,h) - q(h|x)]$ (6)

Therefore, we have

$$\begin{aligned} & D_{\mathrm{KL}}(q(h|x) \parallel p(h|x)) = -\mathrm{ELBO} + \log p(x) \\ \implies & \log p(x) - \mathrm{ELBO} = D_{\mathrm{KL}}(q(h|x) \parallel p(h|x)) \end{aligned}$$

$$\log p(x) \geq \mathrm{ELBO}$$

$$\begin{aligned} \log p(x) &\geq \mathbb{E}_q[\log p(x,h) - \log q(h|x)] \\ &\geq \mathbb{E}_q\left[\log \frac{p(x,h)}{q(h|x)}\right] \end{aligned}$$

x_0 = data
x_1:x_T = hidden representation

Training is performed by optimizing the usual variational bound on negative log likelihood:

$$\begin{aligned} \mathbb{E}[-\log p_\theta(\mathbf{x}_0)] &\leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \\ &= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \\ &:= L \end{aligned}$$

https://jaketae.github.io/study/elbo/

Denoising Diffusion Probabilistic Model (DDPM)

$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$

$p_\theta(x_{t-1}|x_t) = \mathrm{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$

Neural Network prediction to maximize the log-likelihood of the sample generated

- **Training objective: maximizing the log-likelihood of the sample generated (at the end of the reverse process) belonging to the original data distribution.**

A variational upper bound can be formed

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

Simplified to

$$\mathbb{E}_q\Bigg[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\,\|\,p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Bigg]$$

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Loss formulation and simplification - step 1

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

↓ Simplified to

$$\mathbb{E}_q\Bigg[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\,\|\,p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Bigg]$$

Derivation:

$$L = \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \tag{17}$$

$$= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1}\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \tag{18}$$

$$= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t>1}\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)}\right] \tag{19}$$

[Bayes' rule]

$$= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t>1}\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)}\right] \tag{20}$$

[Terms in sum cancel out]

$$= \mathbb{E}_q\left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)} - \sum_{t>1}\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] \tag{21}$$

$$= \mathbb{E}_q\left[D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\,\|\,p(\mathbf{x}_T)) + \sum_{t>1} D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]$$

Adapted from Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Loss formulation and simplification - step 2

$$\mathbb{E}_q\Big[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\parallel p(\mathbf{x}_T))}_{L_T}+\sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Big]$$

Purely based on β_t (hyper parameter)

Normal distribution

Final loss is then simplified to: $L_{t-1} := D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

The posterior distribution is derived as:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)=\mathcal{N}(\mathbf{x}_{t-1};\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\mathbf{I}), \tag{6}$$

$$\text{where}\quad \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0):=\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0+\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad\text{and}\quad \tilde{\beta}_t:=\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{7}$$

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Loss formulation and simplification - step 3

With $$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}),$$

$$\text{where} \quad \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

$$p_\theta(x_{t-1}|x_t) = \mathrm{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$$

$$L_{t-1} := D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$$

$$= \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2\right] + C$$

$p \sim \mathrm{N}(\mu_0, \sigma_0)$, $q \sim \mathrm{N}(\mu_1, \sigma_1)$,

For two univariate normal distributions p and q the above simplifies to[28]

$$D_{\mathrm{KL}}(p \parallel q) = \log\frac{\sigma_1}{\sigma_0} + \frac{\sigma_0^2 + (\mu_0 - \mu_1)^2}{2\sigma_1^2} - \frac{1}{2}$$

From diffusion process, we know $x_t = \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}\,\epsilon$ (from a previous slide)

$$L_{t-1} = \mathbb{E}_q\left[\frac{1}{2\sigma^2}\left\|\tilde{\mu}_t\left(x_t, \frac{x_t - \sqrt{1-\overline{\alpha_t}}\,\epsilon}{\sqrt{\overline{\alpha_t}}}\right) - \mu_\theta(x_t, t)\right\|^2\right] + C$$

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Loss formulation and simplification - step 4

$$L_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma^2} \left\| \tilde{\mu}_t \left(x_t, \frac{x_t - \sqrt{1-\overline{\alpha_t}}\,\epsilon}{\sqrt{\overline{\alpha_t}}} \right) - \mu_\theta(x_t, t) \right\|^2 \right]$$

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right)$$

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boxed{\epsilon_\theta(\mathbf{x}_t, t)} \right)$$

Can be learned by neural network given (x_t, t)

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\frac{\beta_t^2}{2\sigma_t^2 (1-\beta_t)(1-\bar{\alpha}_t)} || \epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon}_{x_t}, t) ||^2 \right] + C$$

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1,T)} \left[|| \epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon}_{\mathbf{x}_t}, t) ||^2 \right]$$

ϵ_θ predicted by NN

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Denoising Diffusion Model – Putting it all together

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
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

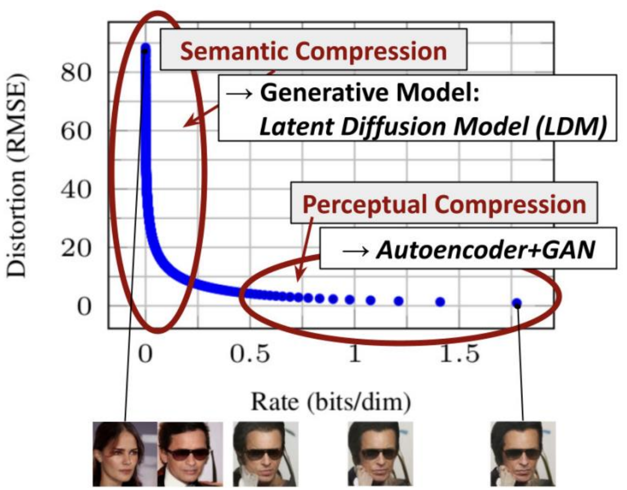
Algorithm 2 Sampling

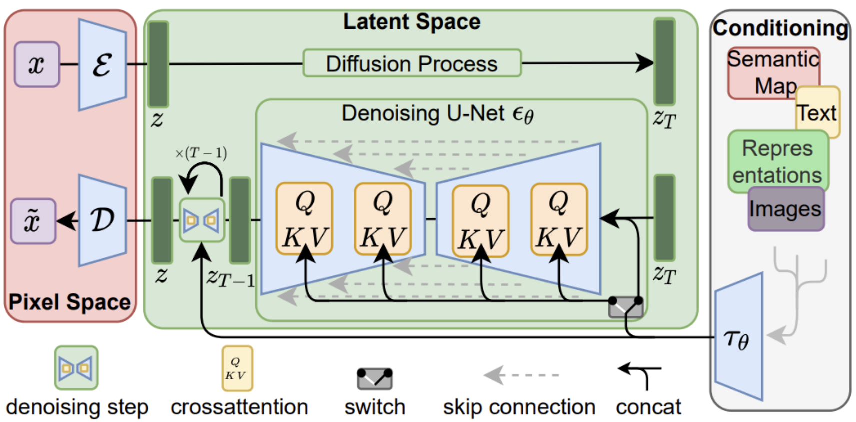
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}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

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/, https://arxiv.org/pdf/2006.11239.pdf

Latent Diffusion Model

- **Problem with DDPM**
- Both diffusion and reverse denoising process operate at pixel space --> extremely computation expensive
- Can we optimize it by diffuse and denoise in latent space?
- **Method**
- Autoencoder which learns a space that is perceptually equivalent to the image space
- Perceptual Compression: removes imperceptible high frequency details
- Semantic Compression: conceptual composition of the image





Rombach et al. "High-Resolution Image Synthesis with Latent Diffusion Models." CVPR 2022.

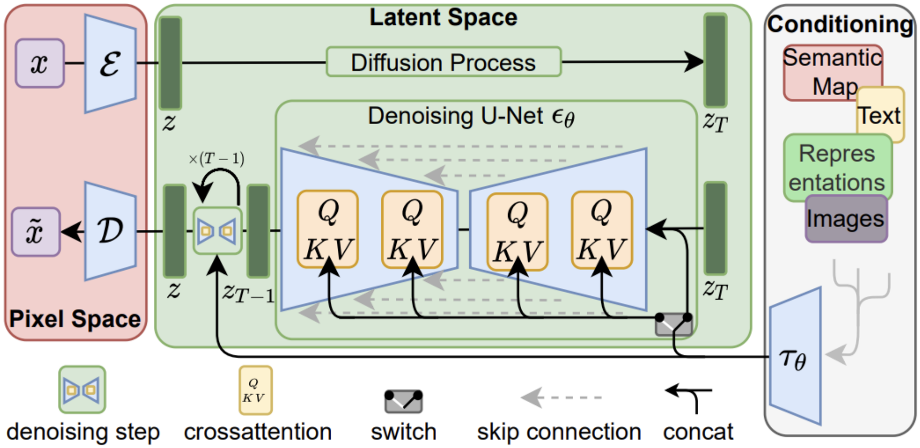
Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Diffusion Model

- **Perceptual image compression --- Autoencoder**

- Input: image $x \in \mathbb{R}^{H \times W \times 3}$
- Encoder ε encodes input x to $z = \varepsilon(x) \in \mathbb{R}^{h \times w \times 3}$
- Decoder D reconstructs the images from the latent $\tilde{x} = D(z)$

- The encoder is set to down sample the image by a factor of $f = \frac{H}{h} = \frac{W}{w}$



Loss for training the autoencoder is formulated as:

Patch-based Discriminator

$$L_{\text{Autoencoder}} = \min_{\mathcal{E},\mathcal{D}} \max_{\psi} \Big(L_{rec}(x, \mathcal{D}(\mathcal{E}(x))) - L_{adv}(\mathcal{D}(\mathcal{E}(x))) + \log D_\psi(x) + L_{reg}(x; \mathcal{E}, \mathcal{D}) \Big)$$

Regularizing Loss

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Diffusion Model

- **Diffusion model training and loss formulation**

For denoising diffusion probabilistic model (DDPM)

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1,T)} \left[|| \epsilon - \epsilon_\theta (\underbrace{\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon}_{\mathbf{x}_t}, t) ||^2 \right]$$

Pixel Space

For latent diffusion model (LDM)

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x), \epsilon \sim \mathcal{N}(0,1), t} \left[\| \epsilon - \epsilon_\theta(z_t, t) \|_2^2 \right].$$

Latent Space from the trained Autoencoder

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Diffusion Model

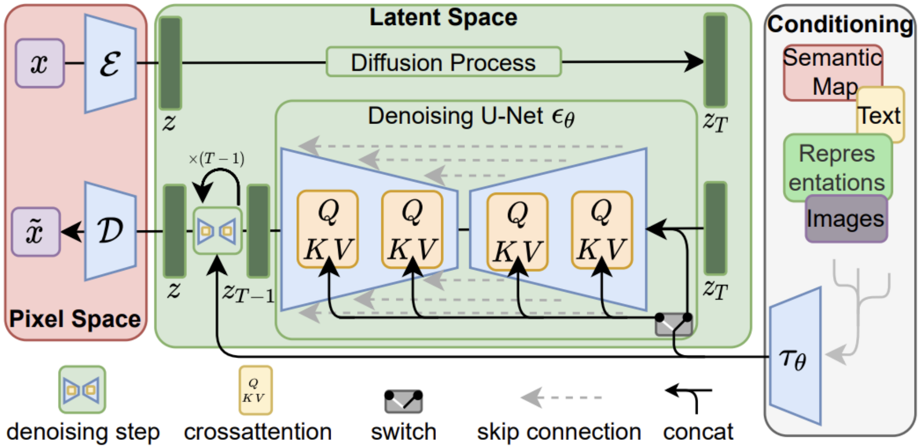
- **Conditioning mechanism**
- Flexible image generator by augmenting UNet with cross attention mechanism

- To condition the generator on y from various modalities, an encoder τ_θ is first applied to project y to the intermediate representation $\tau_\theta(y)$

- The representation is then mapped to the UNet through cross-attention mechanism which:

$$Q = W_Q^{(i)} \cdot \varphi_i(z_t),\ K = W_K^{(i)} \cdot \tau_\theta(y),\ V = W_V^{(i)} \cdot \tau_\theta(y).$$

Denoising UNet



The loss for conditional LDM is thus formulated as:

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x), y, \epsilon \sim \mathcal{N}(0,1), t}\Big[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2\Big]$$

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Diffusion Model

- **Image generation**

- State-of-the-art performance on CelebA-HQ dataset FID metrics

- Comparable performances on other datasets

- Generally better precision and recall→ better mode coverage

CelebA-HQ 256 × 256				FFHQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑	**Method**	FID ↓	Prec. ↑	Recall ↑
DC-VAE [63]	15.8	-	-	ImageBART [21]	9.57	-	-
VQGAN+T. [23] (k=400)	10.2	-	-	U-Net GAN (+aug) [77]	10.9 (7.6)	-	-
PGGAN [39]	8.0	-	-	UDM [43]	5.54	-	-
LSGM [93]	7.22	-	-	StyleGAN [41]	4.16	0.71	0.46
UDM [43]	7.16	-	-	ProjectedGAN [76]	**3.08**	0.65	0.46
LDM-4 (ours, 500-s†)	**5.11**	0.72	0.49	*LDM-4* (ours, 200-s)	4.98	**0.73**	**0.50**

LSUN-Churches 256 × 256				LSUN-Bedrooms 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑	**Method**	FID ↓	Prec. ↑	Recall ↑
DDPM [30]	7.89	-	-	ImageBART [21]	5.51	-	-
ImageBART [21]	7.32	-	-	DDPM [30]	4.9	-	-
PGGAN [39]	6.42	-	-	UDM [43]	4.57	-	-
StyleGAN [41]	4.21	-	-	StyleGAN [41]	2.35	0.59	0.48
StyleGAN2 [42]	3.86	-	-	ADM [15]	1.90	**0.66**	**0.51**
ProjectedGAN [76]	**1.59**	0.61	0.44	ProjectedGAN [76]	**1.52**	0.61	0.34
*LDM-8** (ours, 200-s)	4.02	**0.64**	**0.52**	*LDM-4* (ours, 200-s)	2.95	**0.66**	0.48

Rombach et al. "High-Resolution Image Synthesis with Latent Diffusion Models." CVPR 2022.

Latent Diffusion Model

- **Image generation (Qualitative Results)**

Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Diffusion Model

- **Conditional LDM on text to image generation**
  - 1.45B parameter KL-regularized LDM conditioned on language prompts on LAION-400M
  - Employ Bert tokenizer and set $\tau_\theta(y)$ as transformer
  - Evaluate on MS-COCO validation dataset

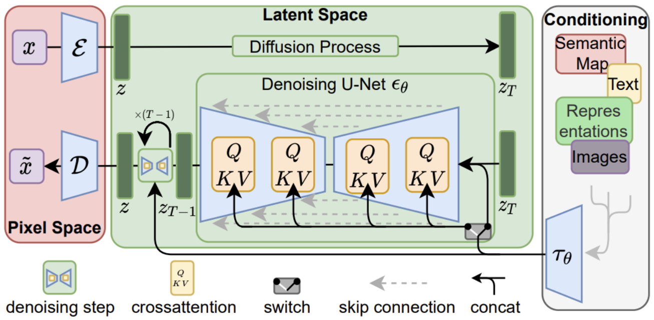
Text-Conditional Image Synthesis				
Method	FID ↓	IS↑	N_{params}	
CogView† [17]	27.10	18.20	4B	self-ranking, rejection rate 0.017
LAFITE† [109]	26.94	<u>26.02</u>	75M	
GLIDE* [59]	<u>12.24</u>	-	6B	277 DDIM steps, c.f.g. [32] $s = 3$
Make-A-Scene* [26]	**11.84**	-	4B	c.f.g for AR models [98] $s = 5$
LDM-KL-8	23.31	20.03±0.33	1.45B	250 DDIM steps
*LDM-KL-8-G**	12.63	**30.29**±0.42	1.45B	250 DDIM steps, c.f.g. [32] $s = 1.5$

- **Achieves comparable text to image synthesis results with significantly less parameters**

Not T time steps, but T transformer blocks in UNet

	input	$\mathbb{R}^{h\times w\times c}$
	LayerNorm	$\mathbb{R}^{h\times w\times c}$
	Conv1x1	$\mathbb{R}^{h\times w\times d\cdot n_h}$
	Reshape	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
$\times T$	SelfAttention	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
	MLP	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
	CrossAttention	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
	Reshape	$\mathbb{R}^{h\times w\times d\cdot n_h}$
	Conv1x1	$\mathbb{R}^{h\times w\times c}$

Architecture of transformer block for cross-attention conditioning



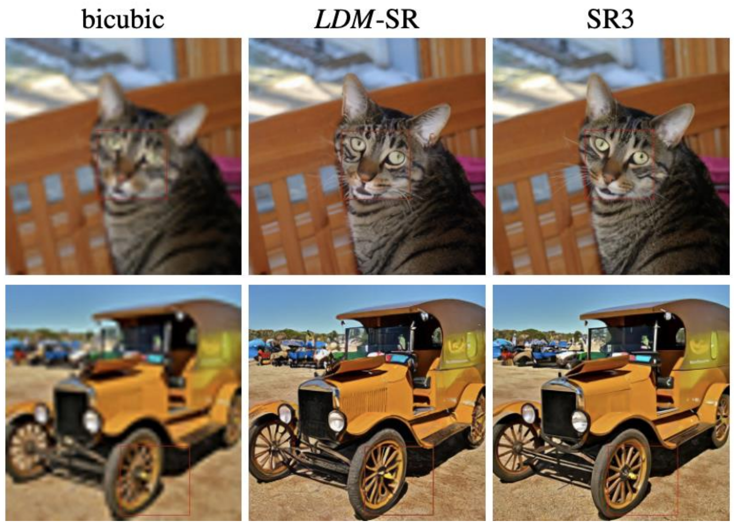
Haolin Zhang, https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Diffusion Model

- **Image super resolution ---- condition on low resolution image by direct concatenation**

- By concatenating spatially aligned conditioning information to the input of ϵ_θ, LDMs can serve as efficient general image-to-image translation model
- Trained on ImageNet. Create low resolution by first down-sampling 4× through bicubic interpolation

```
if self.conditioning_key is None:
    out = self.diffusion_model(x, t)
elif self.conditioning_key == 'concat':
    xc = torch.cat([x] + c_concat, dim=1)
    out = self.diffusion_model(xc, t)
elif self.conditioning_key == 'crossattn':
    cc = torch.cat(c_crossattn, 1)
    out = self.diffusion_model(x, t, context=cc)
elif self.conditioning_key == 'hybrid':
    xc = torch.cat([x] + c_concat, dim=1)
    cc = torch.cat(c_crossattn, 1)
    out = self.diffusion_model(xc, t, context=cc)
```



Latent Diffusion Model

- **Image super resolution ---- condition on low resolution image by concatenation**

- Regression model performs better in PSNR and SSIM because these metrics favor blurriness rather than incorrect high frequency details

Method	FID ↓	IS ↑	PSNR ↑	SSIM ↑	N_{params}	$[\frac{\text{samples}}{s}](*)$
Image Regression [72]	15.2	121.1	**27.9**	**0.801**	625M	N/A
SR3 [72]	5.2	**180.1**	26.4	0.762	625M	N/A
LDM-4 (ours, 100 steps)	$2.8^{\dagger}/4.8^{\ddagger}$	166.3	24.4 ± 3.8	0.69 ± 0.14	**169M**	4.62
emphLDM-4 (ours, big, 100 steps)	$\mathbf{2.4}^{\dagger}/\mathbf{4.3}^{\ddagger}$	174.9	24.7 ± 4.1	0.71 ± 0.15	552M	4.5
LDM-4 (ours, 50 steps, guiding)	$4.4^{\dagger}/6.4^{\ddagger}$	153.7	25.8 ± 3.7	0.74 ± 0.12	184M	0.38

- Human evaluation show generally better LDM performance over pixel-based DM

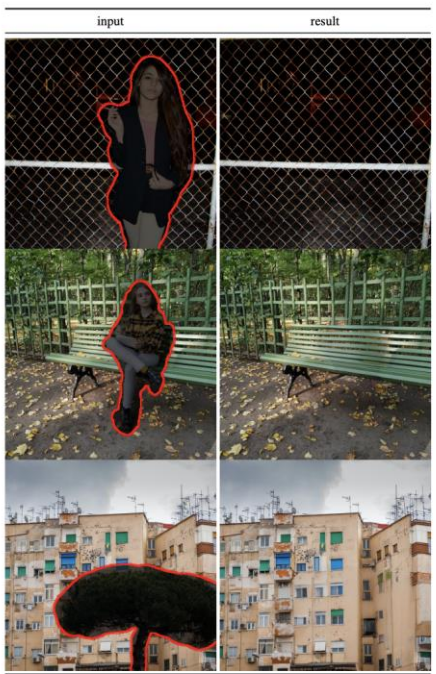
	SR on ImageNet		Inpainting on Places	
User Study	Pixel-DM ($f1$)	*LDM-4*	LAMA [88]	*LDM-4*
Task 1: Preference vs GT ↑	16.0%	**30.4%**	13.6%	**21.0%**
Task 2: Preference Score ↑	29.4%	**70.6%**	31.9%	**68.1%**

Rombach et al. "High-Resolution Image Synthesis with Latent Diffusion Models." CVPR 2022.

Latent Diffusion Model

- **Image inpainting**

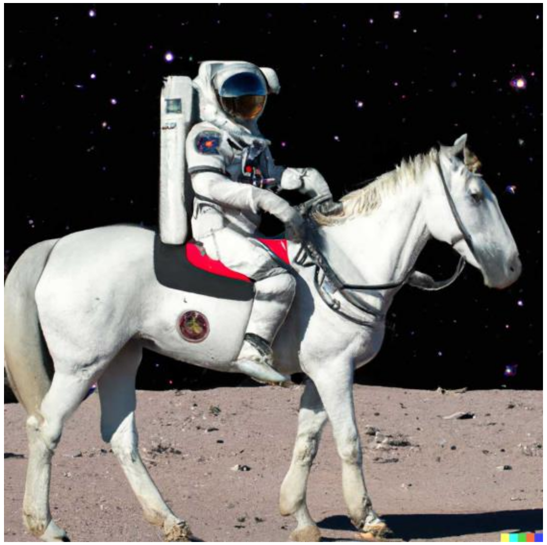
- LDM achieves better or comparable performances



Method	40-50% masked FID ↓	40-50% masked LPIPS ↓	All samples FID ↓	All samples LPIPS ↓
LDM-4 (ours, big, w/ ft)	**9.39**	0.246 ± 0.042	**1.50**	0.137 ± 0.080
LDM-4 (ours, big, w/o ft)	12.89	0.257 ± 0.047	2.40	0.142 ± 0.085
LDM-4 (ours, w/ attn)	11.87	0.257 ± 0.042	2.15	0.144 ± 0.084
LDM-4 (ours, w/o attn)	12.60	0.259 ± 0.041	2.37	0.145 ± 0.084
LaMa [88]†	12.31	**0.243** ± 0.038	2.23	**0.134** ± 0.080
LaMa [88]	12.0	**0.24**	2.21	0.14
CoModGAN [107]	10.4	0.26	1.82	0.15
RegionWise [52]	21.3	0.27	4.75	0.15
DeepFill v2 [104]	22.1	0.28	5.20	0.16
EdgeConnect [58]	30.5	0.28	8.37	0.16

DALLE 2 (Text-to-Image)

Teddy bears mixing sparkling chemicals as mad scientists



An astronaut riding a horse in a photorealistic style

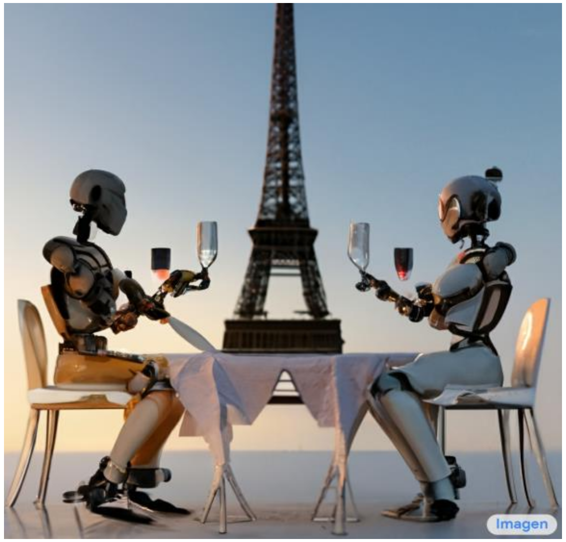
A bowl of soup as a planet in the universe

Aryan Jain, https://openai.com/dall-e-2/

Imagen (Text-to-Image)

A cute corgi lives in a house made of sushi

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



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

Aryan Jain, https://imagen.research.google/

Make-A-Video (Text-to-Video)

An artist's brush painting on a canvas close up

A young couple walking in heavy rain

Horse drinking water

Aryan Jain, https://makeavideo.studio/

Make-A-Video (Text-to-Video)

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

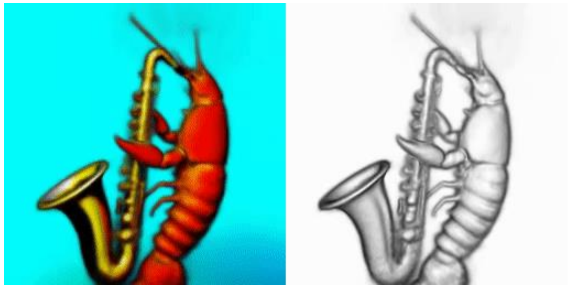
Aryan Jain, https://makeavideo.studio/

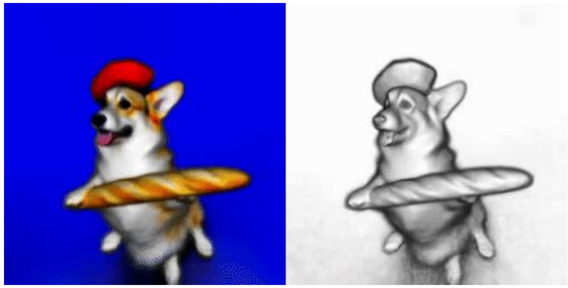
Imagen Video (Text-to-Video)

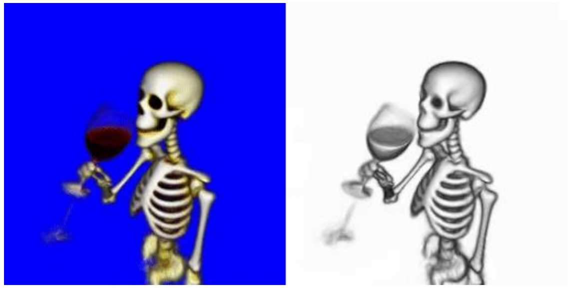
Aryan Jain, https://imagen.research.google/video/

DreamFusion (Text-to-3D)


a fox holding a video game controller

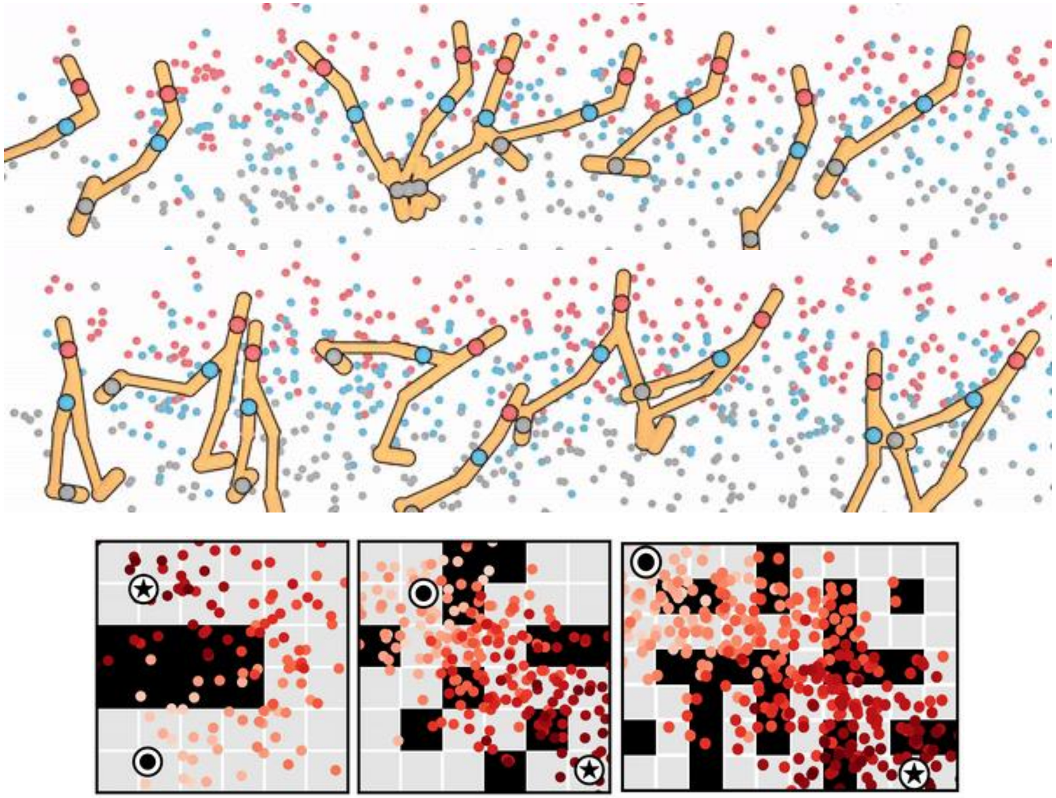

a lobster playing the saxophone


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


a human skeleton drinking a glass of red wine

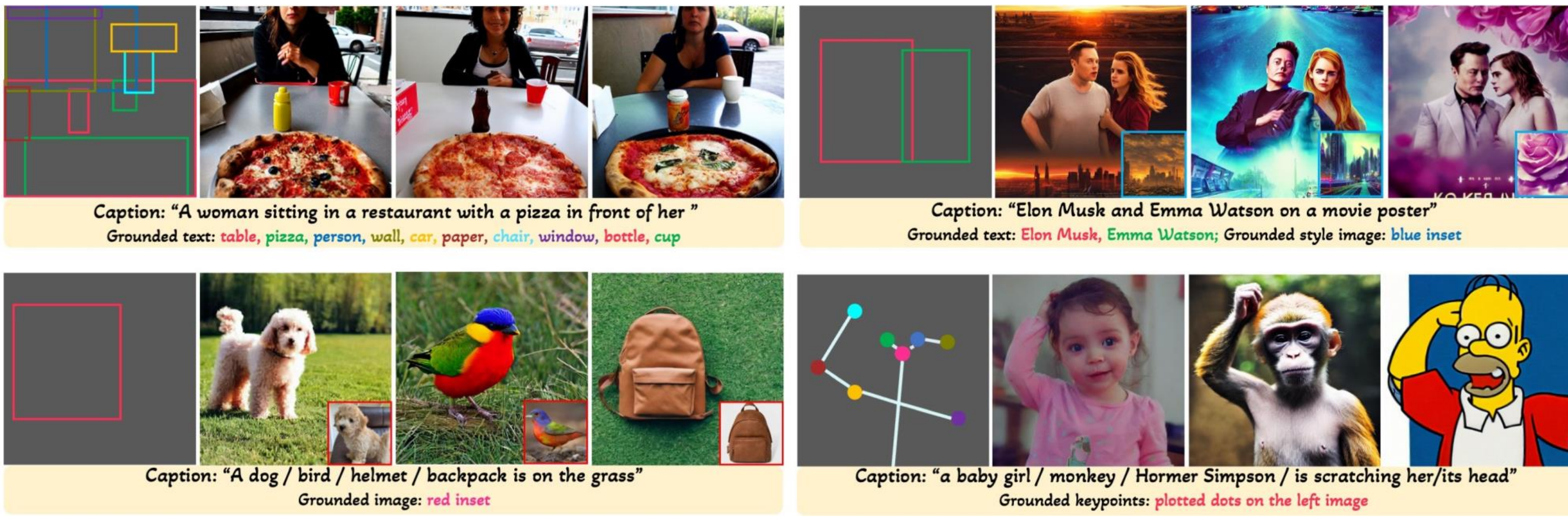
Aryan Jain, https://dreamfusion3d.github.io/gallery.html

Diffuser (Trajectory Planning)



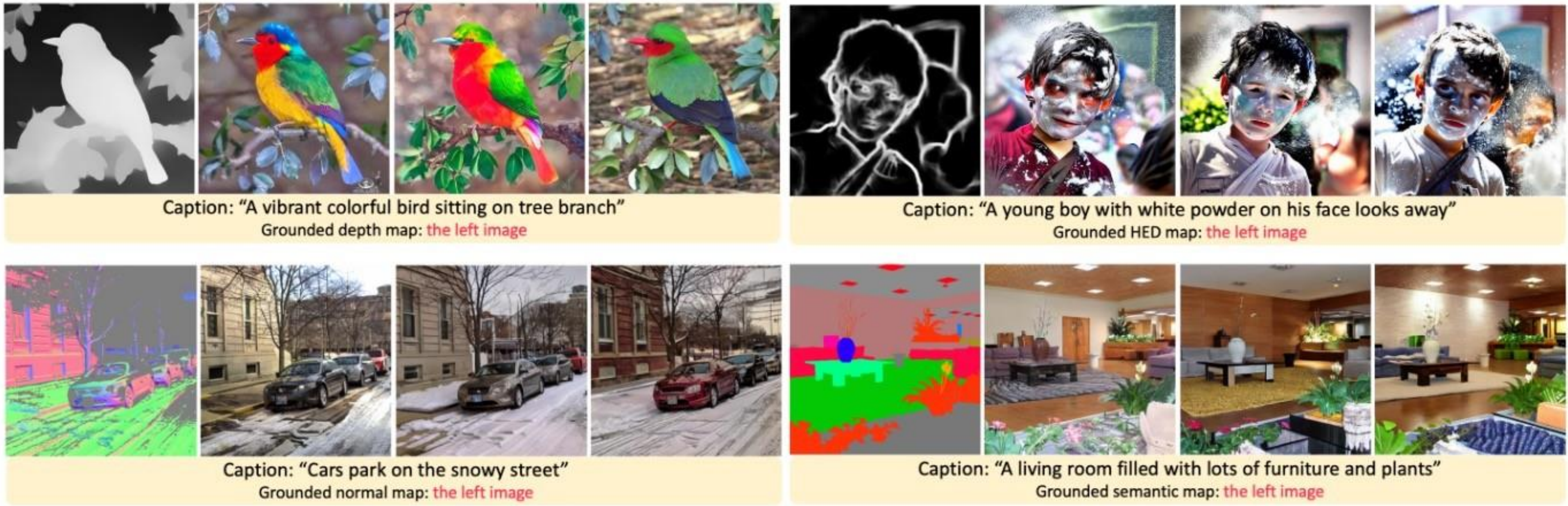
Aryan Jain, https://diffusion-planning.github.io/

GLIGEN: Open-Set Grounded Text-to-Image Generation



Yong Jae Lee, https://huggingface.co/spaces/gligen/demo, https://gligen.github.io/

GLIGEN: Open-Set Grounded Text-to-Image Generation



Talk by Yong Jae Lee on April 12, 2pm, Sennott Square 5317

Yong Jae Lee, https://huggingface.co/spaces/gligen/demo, https://gligen.github.io/

DemoCaricature: Democratising Caricature Generation with a Rough Sketch

Dar-Yen Chen Subhadeep Koley Aneeshan Sain Pinaki Nath Chowdhury
Tao Xiang Ayan Kumar Bhunia Yi-Zhe Song
SketchX, CVSSP, University of Surrey, United Kingdom.
{s.koley, a.sain, p.chowdhury, t.xiang, a.bhunia, y.song}@surrey.ac.uk

https://democaricature.github.io

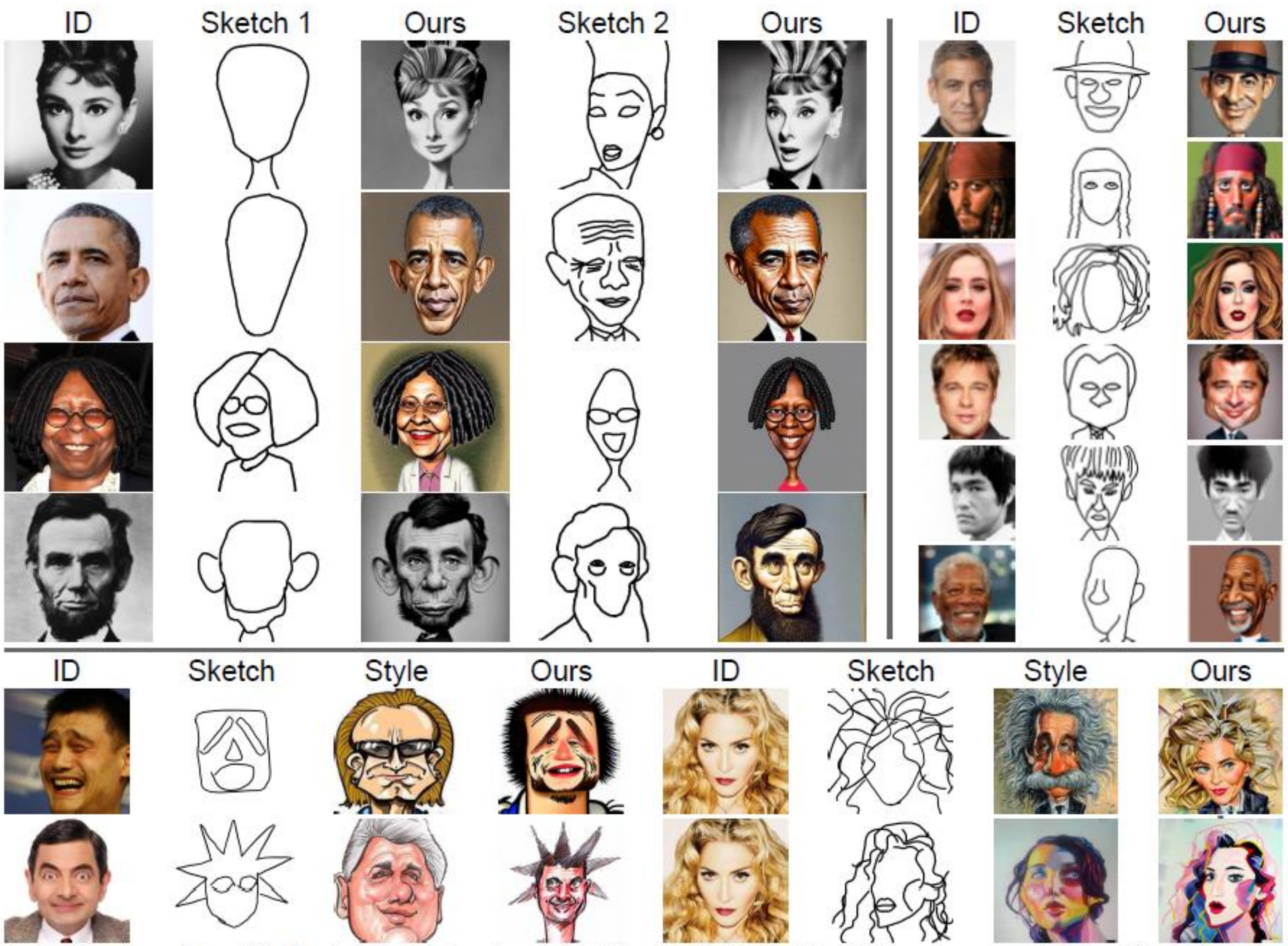


Figure 1. Given an *abstract freehand* sketch and an image depicting the facial identity of a person, our method transforms the deformed sketch into a plausible-looking caricature while maintaining *identity-fidelity* and imitating the *exaggerations* portrayed in the input sketch. Additionally, it can seamlessly transmit the *look-and-feel* of a given *style-image* into the output caricature.

CVPR
2024

It's All About *Your* Sketch: Democratising Sketch Control in Diffusion Models

Subhadeep Koley[1,2] Ayan Kumar Bhunia[1] Deeptanshu Sekhri[1] Aneeshan Sain[1,2]
Pinaki Nath Chowdhury[1,2] Tao Xiang[1,2] Yi-Zhe Song[1,2]
[1]SketchX, CVSSP, University of Surrey, United Kingdom.
[2]iFlyTek-Surrey Joint Research Centre on Artificial Intelligence.
{s.koley, a.bhunia, d.sekhri, a.sain, p.chowdhury, t.xiang, y.song}@surrey.ac.uk

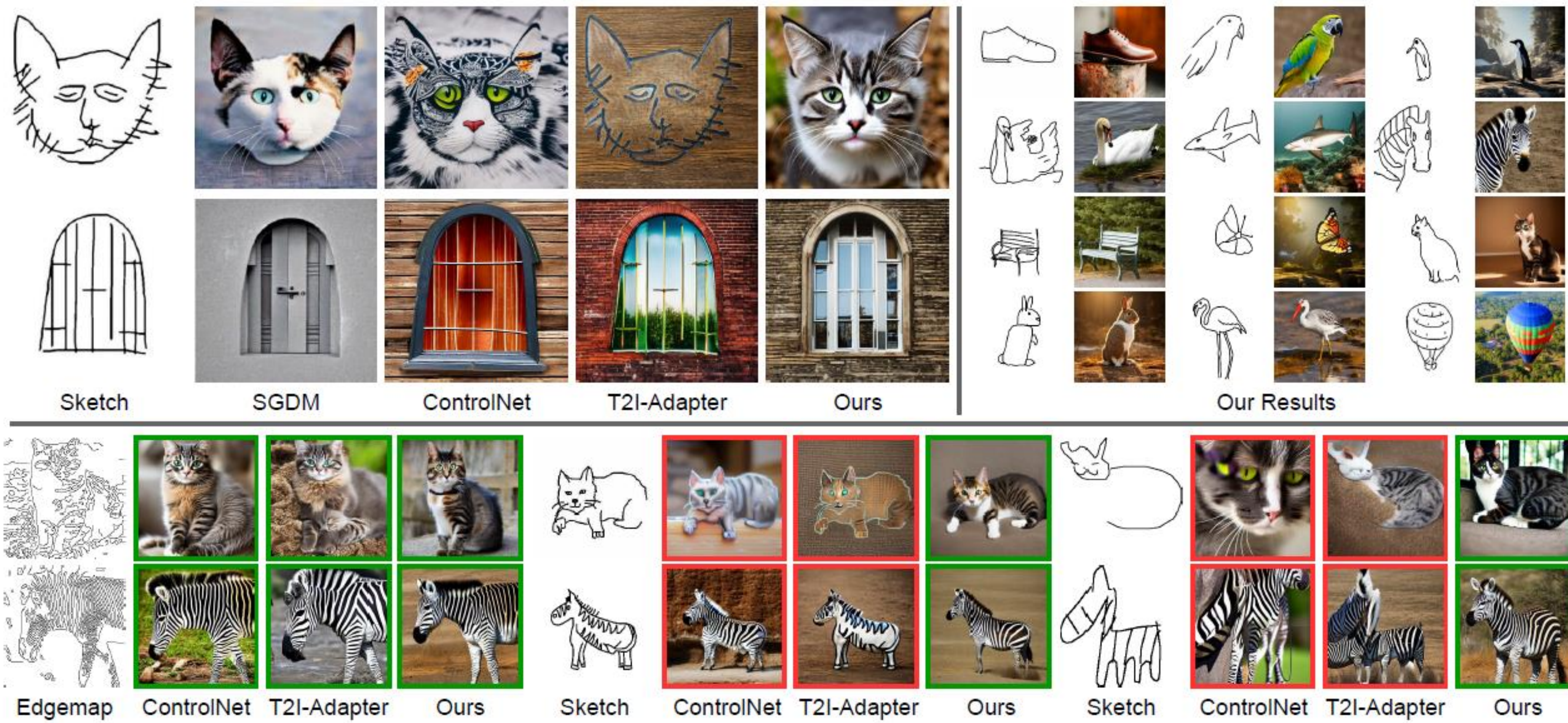


Figure 1. *Top-left*: Comparison of images generated by our method with SGDM [80], ControlNet [90], and T2I-Adapter [54]. *Top-right*: A set of photos generated by our method. *Bottom*: While existing methods [54, 90] generate realistic images from *pixel-perfect edgemaps*, they perform sub-optimally for *freehand abstract sketches*. *(Best view when zoomed in.)*

CVPR 2024

RAVE: Randomized Noise Shuffling for Fast and Consistent Video Editing with Diffusion Models

Ozgur Kara[1*] Bariscan Kurtkaya[2*†] Hidir Yesiltepe[4] James M. Rehg[1,3] Pinar Yanardag[4]

[1]Georgia Tech [2]KUIS AI Center [3]UIUC [4]Virginia Tech

okara7@gatech.edu, bkurtkaya23@ku.edu.tr, hidir@vt.edu, jrehg@uiuc.edu, pinary@vt.edu

Project Webpage: https://rave-video.github.io

Figure 1. **RAVE** is a lightweight and fast video editing method that enhances temporal consistency in video edits, utilizing pre-trained text-to-image diffusion models. It is capable of modifying local attributes, like changing a person's *jacket* (bottom right), and can also handle complex shape transformations, such as turning a *wolf* into a *dinosaur* (bottom left).

CVPR 2024