Advanced Computer Vision

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

June 15, 2023

Autoencoders

- Family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.
- The idea is project the input into a latent space and then reconstruct the input from that latent space representation
- Consist of two parts: Encoder and decode.
  - Encoder projects the input to a latent space Z.
  - Decoder takes the encoded embedding vector and reconstructs the input from it.
  - We also use altered versions of input as output which can be even more interesting.

Autoencoders (cont.)

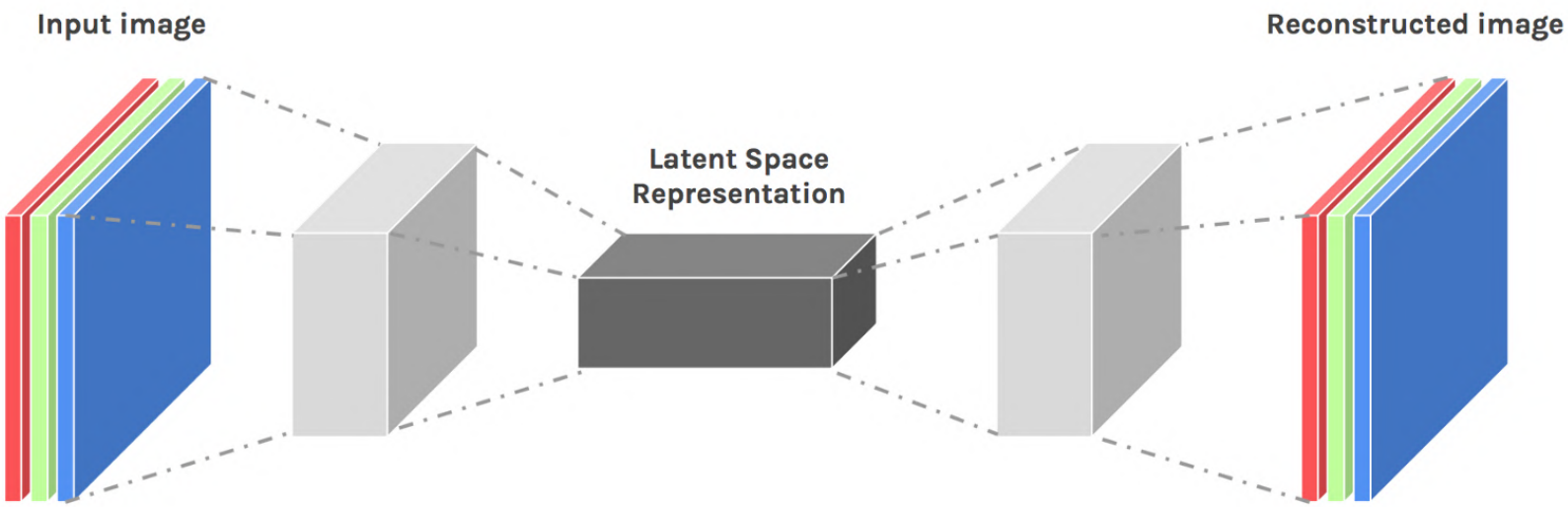


Figure 2: Autoencoder architecture

Autoencoders (cont.)

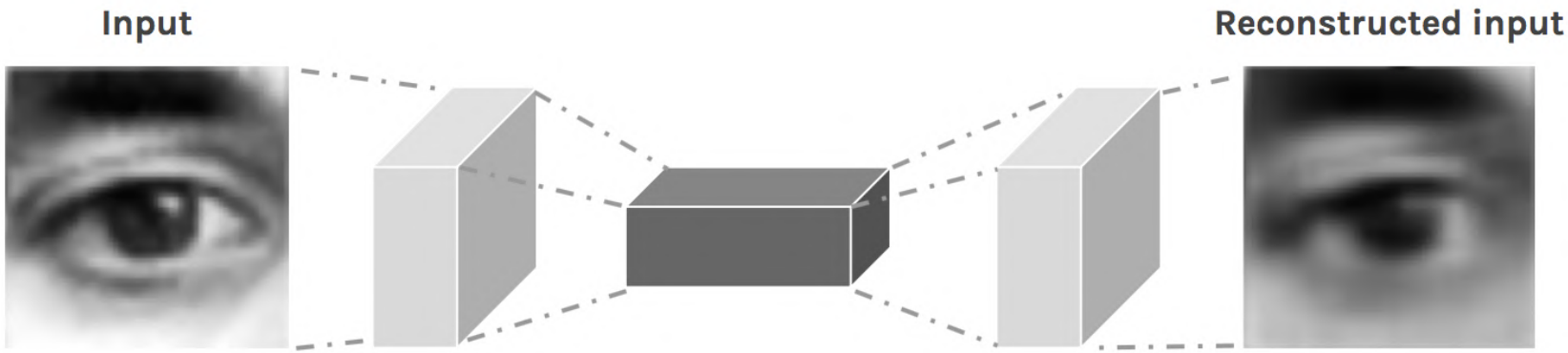


Figure 3: Sample Autoencoder

Autoencoders - Interactive Demo

https://douglasduhaime.com/posts/visualizing-latent-spaces.html

Autoencoders as generative models

- Autoencoders project data into a latent space Z.
- What if we sample a new embedding vector from Z and then have the decoder reconstruct the image from it?
- **Does not work**. Autoencoders just learn a function that maps input to output. The learned latent space is too discontinous to work as a generative model.

Autoencoders as generative models (cont.)

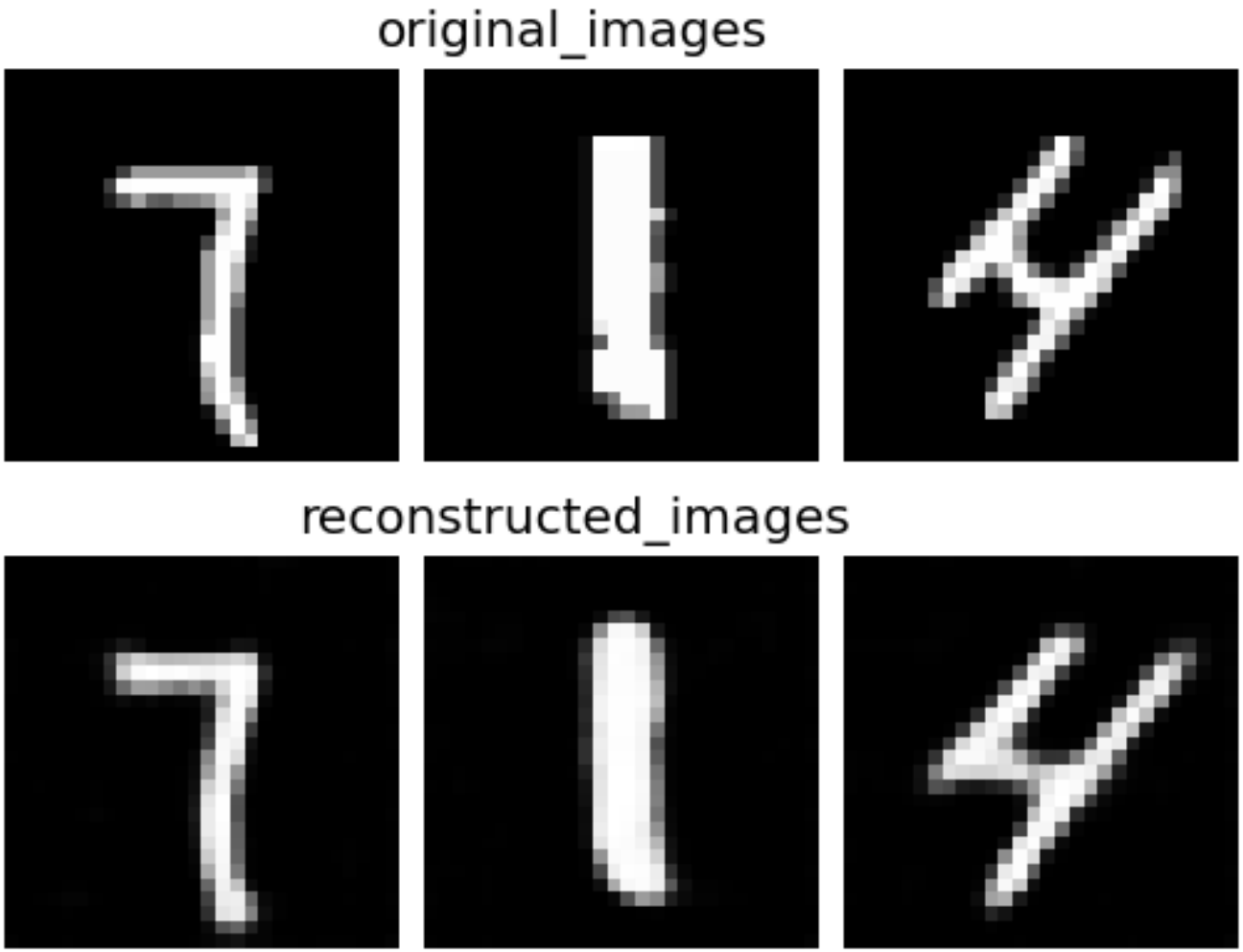


Figure 4: Image reconstruction with autoencoder trained on MNIST digits

Autoencoders as generative models (cont.)

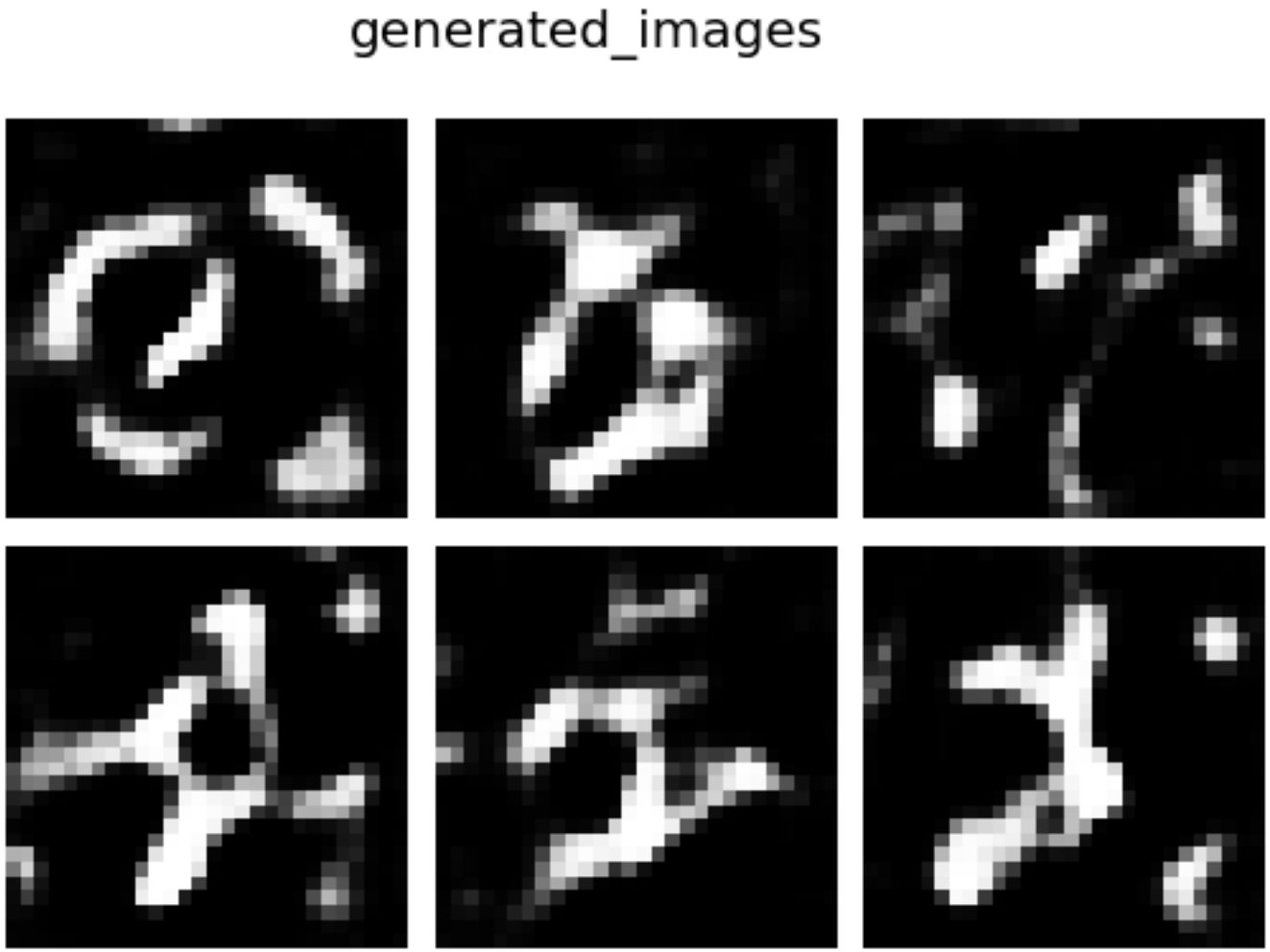


Figure 5: Image generation with autoencoder trained on MNIST digits. Encoding vector sampled from latent space Z and the passed to decoder.

Autoencoders - Applications

- While autoencoders themselves have very low generative power, we will soon talk about a type of autoencoders called **Variational Autoencoders** which are specifically designed for generative modeling.
- Other use cases of Autoencoders include:
  - Data encoding and dimensionality reduction
  - Image denoising and super-resolution
  - Image completion
  - Image colorization

Autoencoders - Applications (cont.)

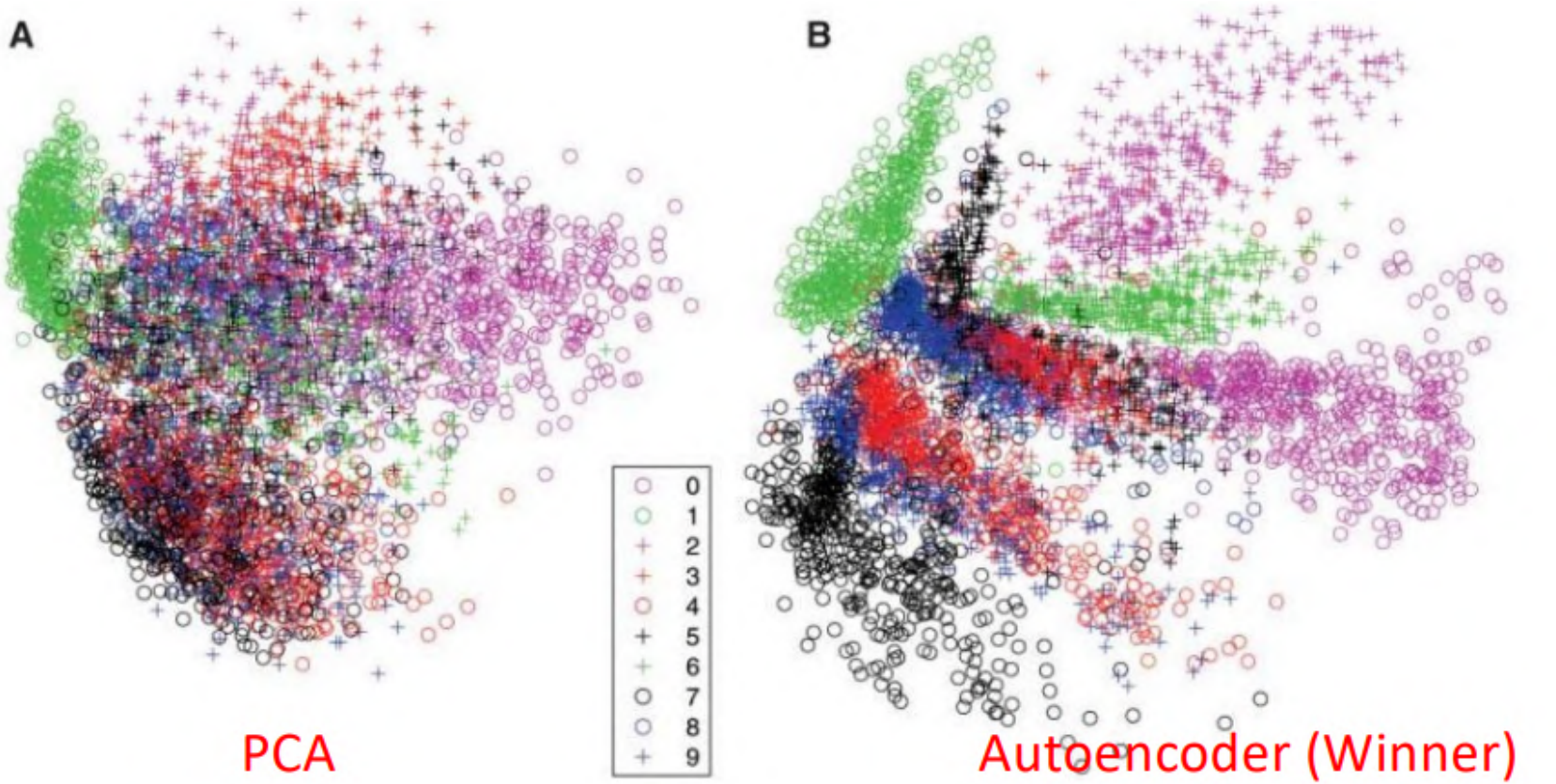


Figure 6: t-SNE visualization on MNIST digits dataset. PCA vs. Autoencoders. The image vector is projected into $\mathbb{R}^2$.

Autoencoders - Applications (cont.)

Figure 7: Image super-resolution using Autoencoders

Autoencoders - Applications (cont.)

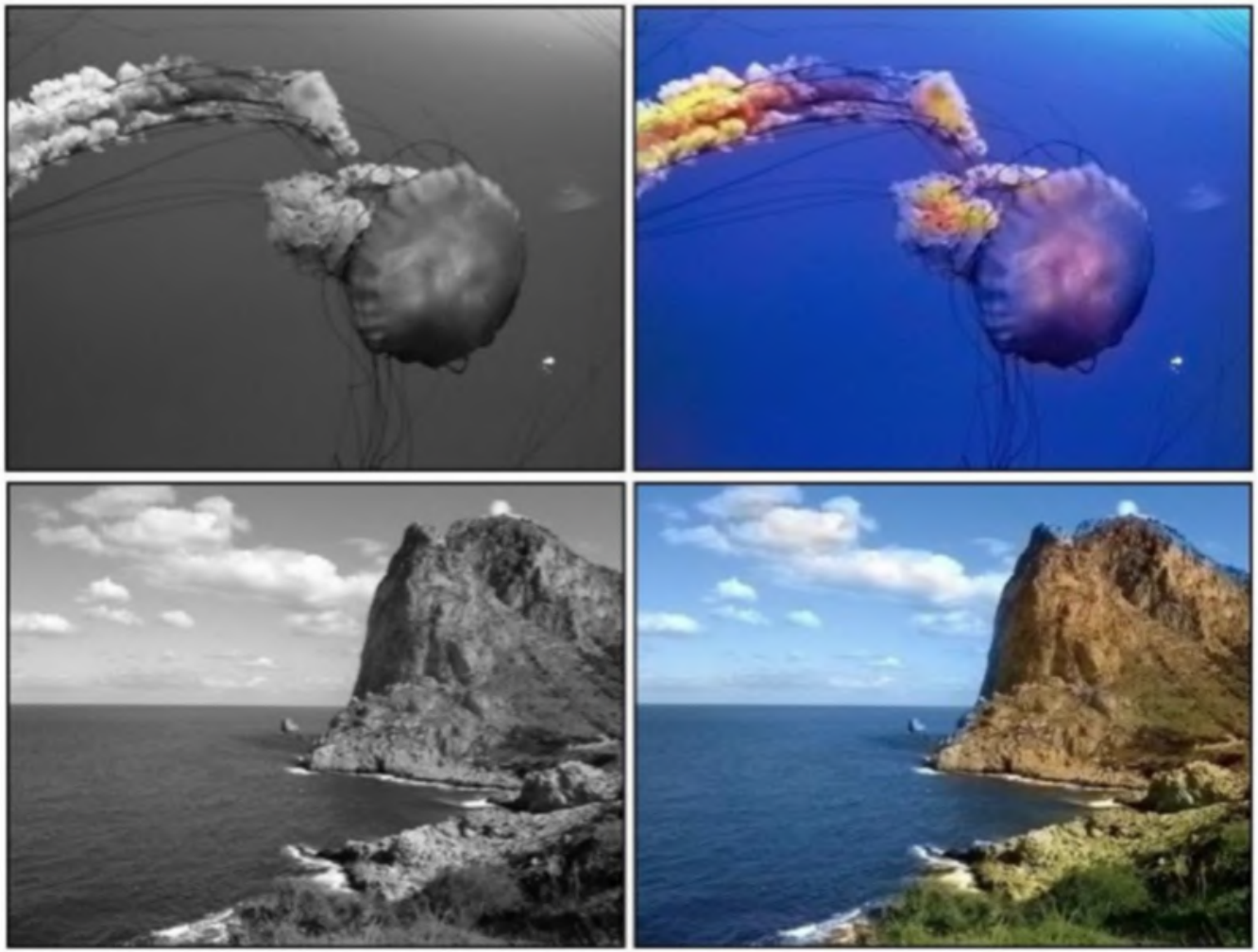


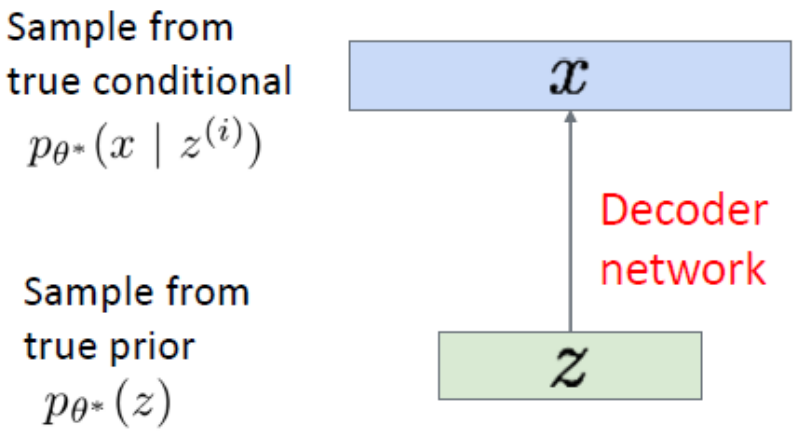
Figure 8: Image colorization using Autoencoders

Variational Autoencoders

- Autoencoders don't work as generative models because the latent space Z is too discontinous.
- **Solution**: Let's rectify that.
- **Variational Autoencoders**: Probabilistic spin on autoencders - will let us sample from the model to generate data.

Variational Autoencoders (cont.)

- We want a generative model, which given a prior **z** outputs a new sample from data.
- To make the latent space Z continuous, let's choose it to be Gaussian
- So now, we want to estimate the true parameters θ^* of this generative model.



Variational Inference

- Now, how to train this model?
- Learn model parameters to maximize the likelihood of training data.

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

- But there is a problem here. It is Intractible to compute $p(x|z)$ for every z!
- Intuitively, need to figure out which z corresponds to each x in the dataset, but such mapping is unknown.
- This also makes posterior density $p(z|x)$ intractable because it depends on $p_\theta(x)$

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

- **Solution**
  - Let's approximate $p(z|x)$ with another distribution by another distribution $q(z)$
  - If $q(z)$ is a tractable distribution e.g. Gaussian distribution
  - We can adjust parameters of $q(z)$ and make it as close to $p(z|x)$
  - **Goal**: min $KL(q||p)$

$$
\begin{aligned}
KL(q(z)||p(z|x)) &= -\sum q(z) log \frac{p(z|x)}{q(z)} \\
&= -\sum q(z) log \frac{\frac{p(x,z)}{p(x)}}{q(z)} \\
&= -\sum q(z) log \left(\frac{p(x,z)}{q(z)} \frac{1}{p(x)} \right) \\
&= -\sum q(z) \left[log \frac{p(x,z)}{q(z)} + log \frac{1}{p(x)} \right] \\
&= -\sum q(z) \left[log \frac{p(x,z)}{q(z)} - log\, p(x) \right] \\
&= -\sum_z q(z) log \frac{p(x,z)}{q(z)} + log\, p(x) \sum_z q(z) \\
&= -\sum_z q(z) log \frac{p(x,z)}{q(z)} + log\, p(x) \quad \because \sum_z q(z) = 1
\end{aligned}
$$

- $$KL(q(z)||p(z|x) = -\sum_z q(z) log \frac{p(x,z)}{q(z)} + log\ p(x)$$
- We can also write above equation as:

$$log\ p(x) = KL(q(z)||p(z|x) + \sum_z q(z) log \frac{p(x,z)}{q(z)}$$

Variational Inference (cont.)

- Given x , $log\ p(x)$ is a constant
- $KL(q(z)||p(z|x)$ is the quantity we wanted to minimize
- Assume $L = \sum_z q(z) log \frac{p(x,z)}{q(z)}$, then

$$\text{constant} = KL + L$$

$$L \leq log\ p(x) \quad \because kl \geq 0$$

- **Instead of minimizing KL we can maximise L**

Variational Lower Bound L

Looking at Lower bound L

$$
\begin{aligned}
L &= \sum_z q(z) log \frac{p(x,z)}{q(z)} \\
&= \sum_z q(z) log \frac{p(x|z)p(z)}{q(z)} \\
&= \sum_z q(z) \left[log\, p(x|z) + log \frac{p(z)}{q(z)} \right] \\
&= \underbrace{\sum_z q(z) log\, p(x|z)}_{\text{Expectation } E_{q(z)}(log\, p(x|z))} + \underbrace{\sum_z q(z) log \frac{p(z)}{q(z)}}_{-KL(q(z)||p(z)}
\end{aligned}
$$

So,

$$L = E_{q(z)}(log\, p(x|z)) - KL(q(z)||p(z))$$

Variational Lower Bound L (cont.)

- $E_{q(z)}(log\ p(x|z))$ is conceptually reconstruction
- We can assume z to be Standard Normal Distribution

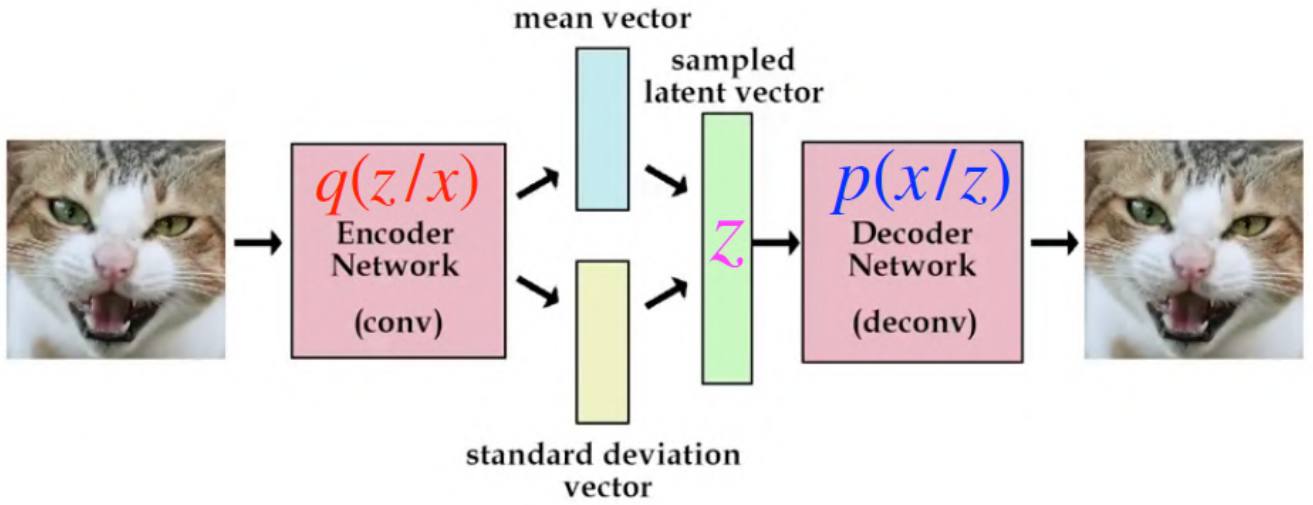
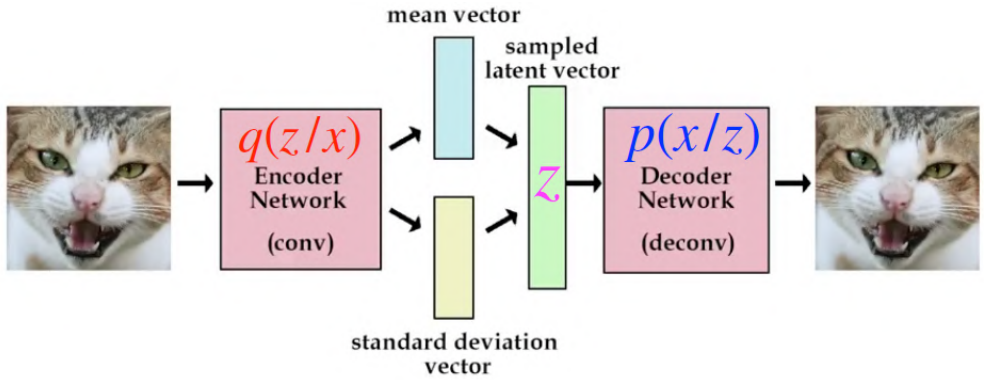


Figure 9: Variational Autoencoder Architecture

Variational Lower Bound L (cont.)



$$p(x|\hat{x}) = e^{-|x-\hat{x}|^2}$$

$$log\ e^{-|x-\hat{x}|^2} = -|x-\hat{x}|^2$$

$$L = E_{q(z)}(-|x-\hat{x}|^2) - KL(q(z)||p(z))$$

$$\min |x-\hat{x}| + KL(q(z|x)||\mathcal{N}(0,1))$$

$$\min |x-\hat{x}| - 0.5 * (1 + log\ \sigma^2 - \sigma^2 - \mu^2)$$

For full derivation of KL Loss, read here

Reparameterization Trick

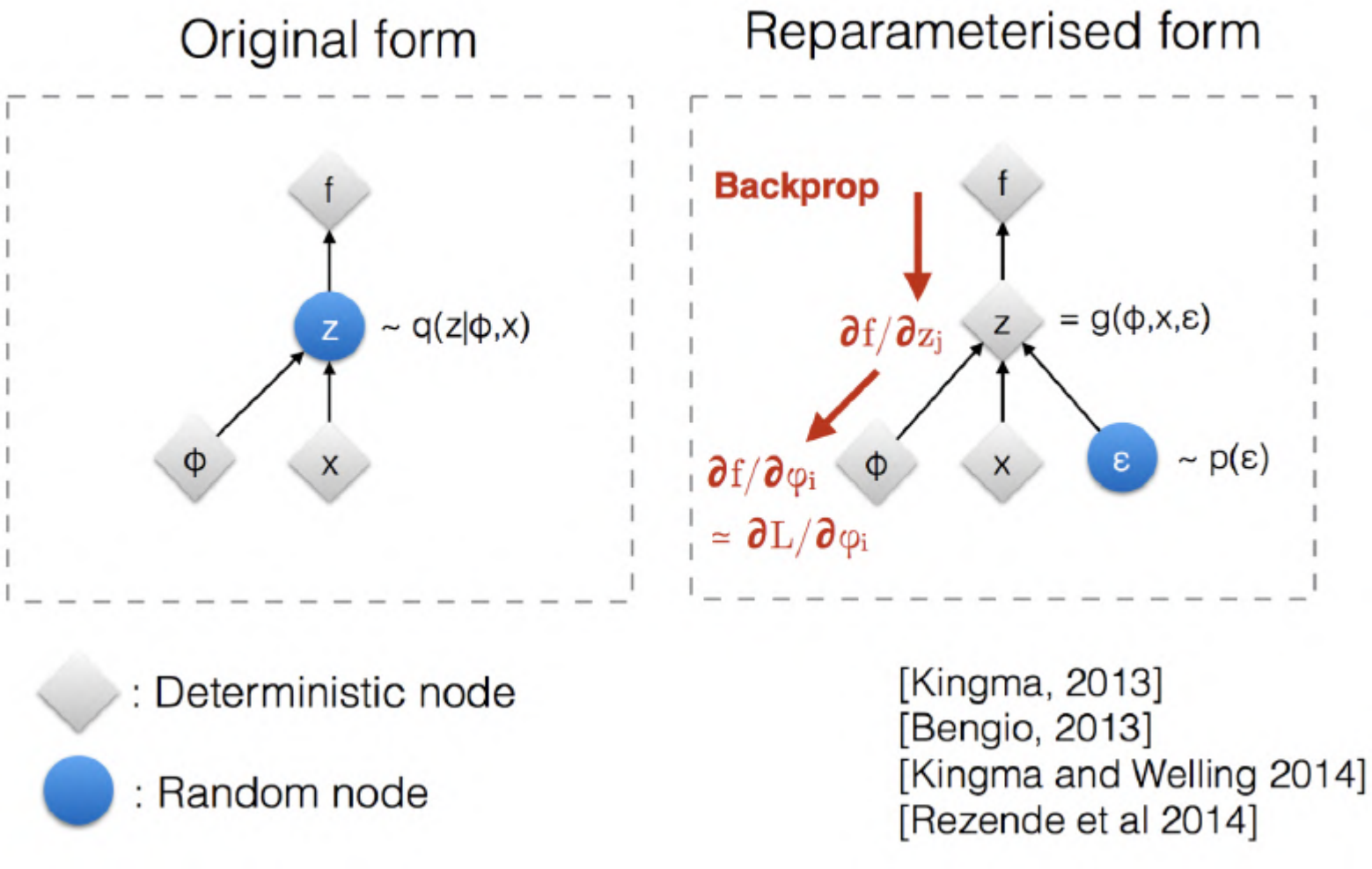


Figure 10: Reparameterization trick to make back propagation possible

Reparameterization Trick (cont.)

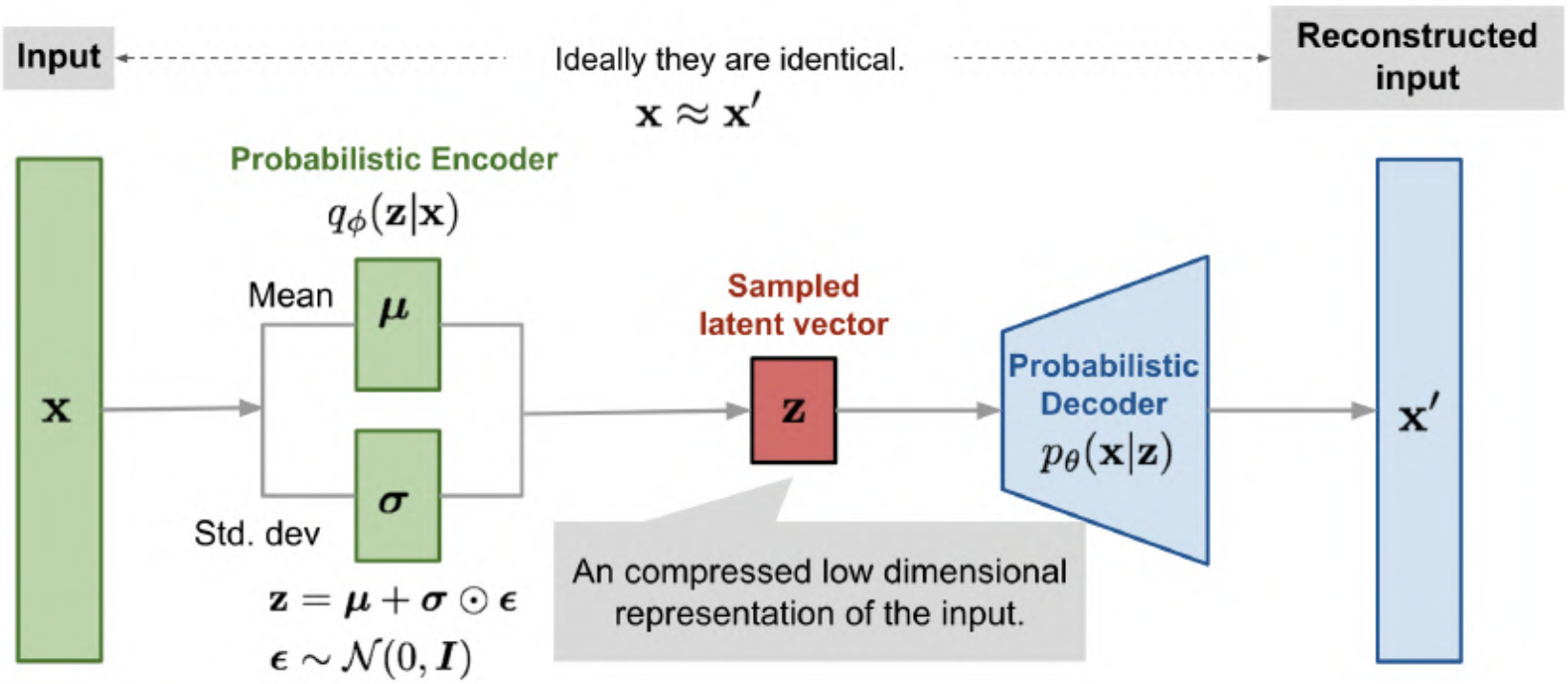


Figure 11: Variational Autoencoder with reparameterization trick

VAE - Results

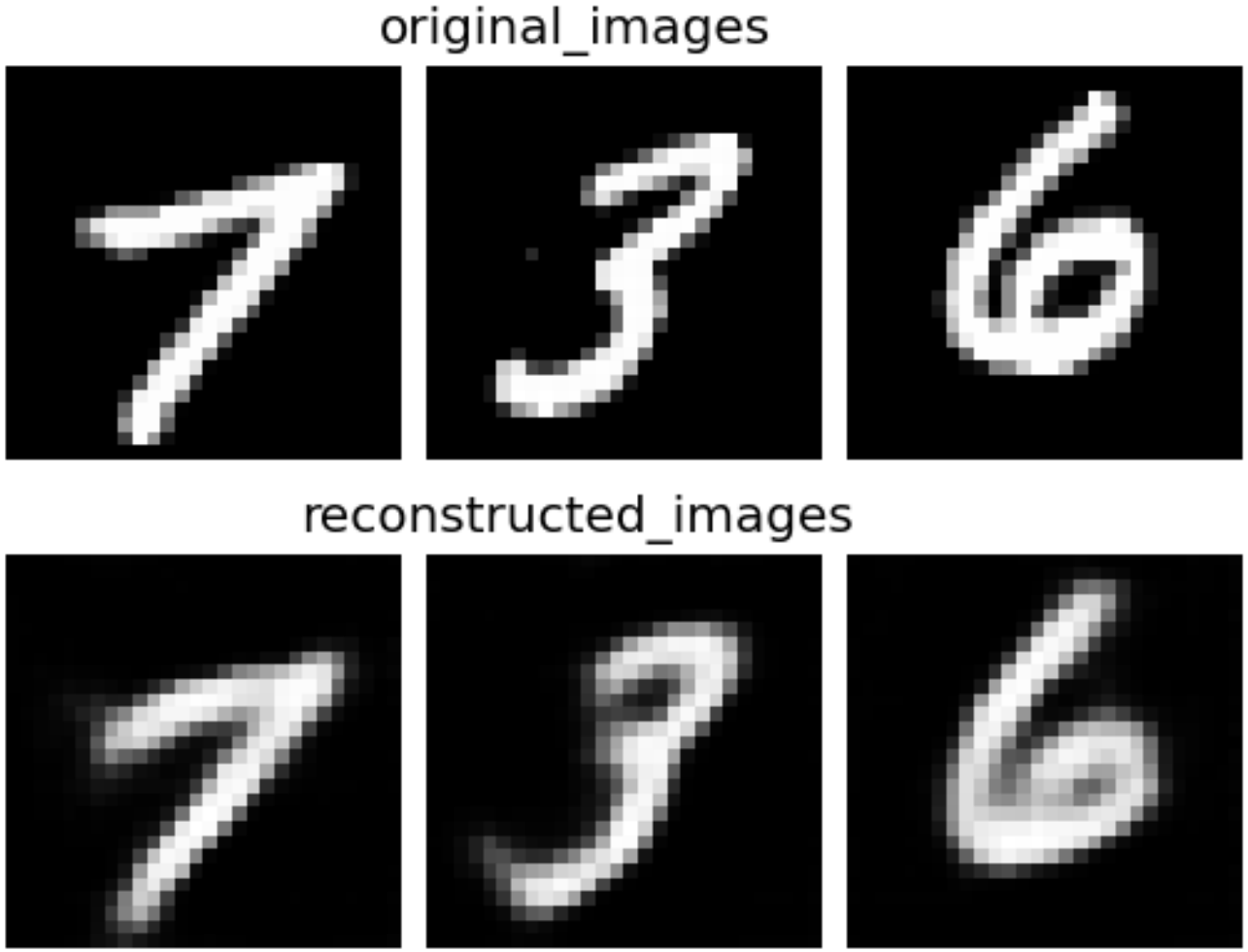


Figure 12: Image reconstruction with variational autoencoders on MNIST digits dataset

VAE - Results (cont.)

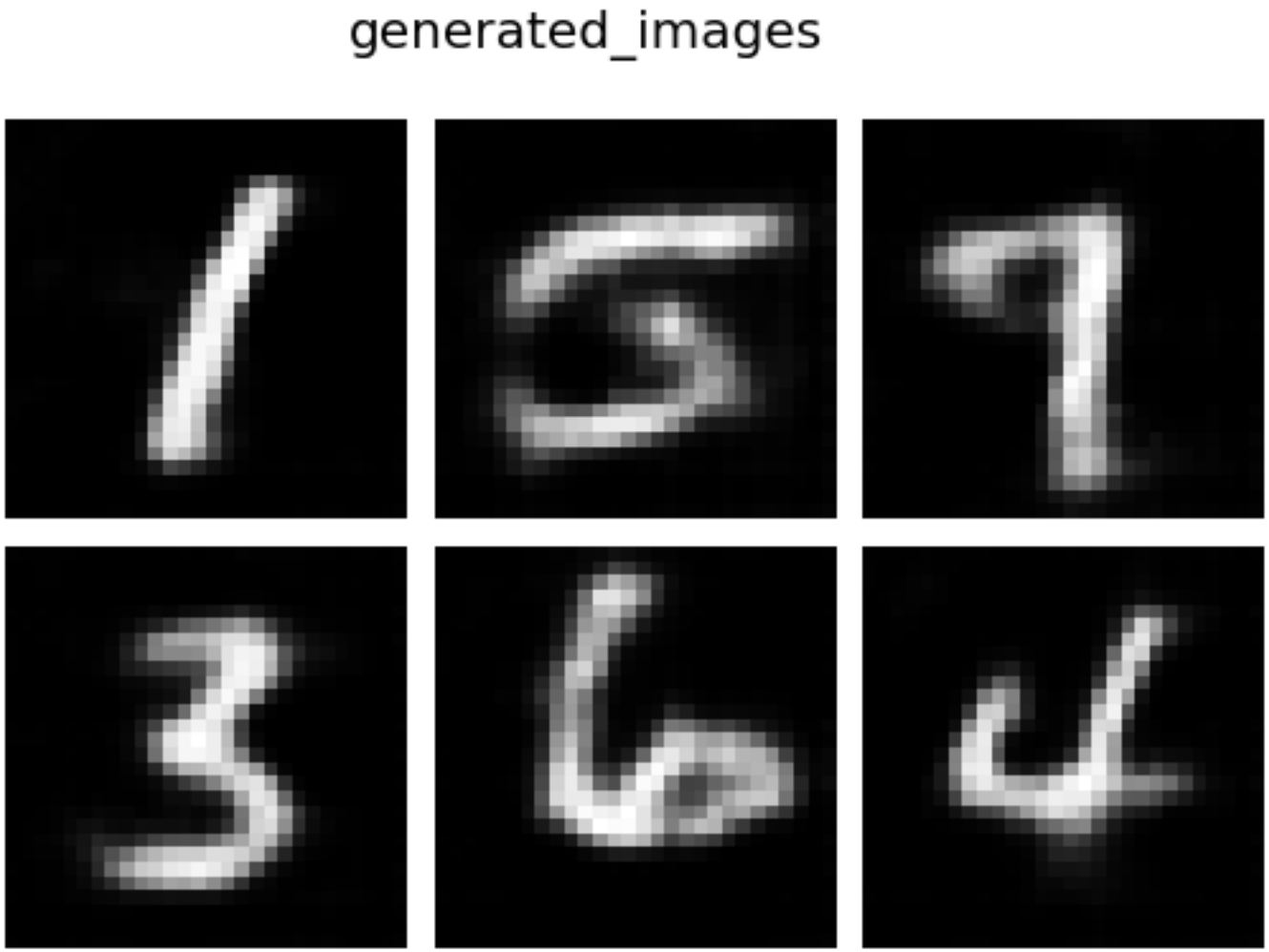


Figure 13: Image generation with variational autoencoders on MNIST digits dataset. Sample an encoding vector from $\mathcal{N}(0, 1)$ and passed it through decoder

VAE - Results (cont.)

Figure 14: Image generation with variational autoencoders on CIFAR-10 32x32 dataset

Disentangled Variational Autoencoders (β-VAEs)

- **Basic Idea**: Different neurons in latent space should be uncorrelated i.e. they all try to learn something different about input data.
- **Implementation**:

$$\mathcal{L}(\theta, \phi; x, z, \beta) = E_{q_\phi(z|x)}(log\ p_\theta(x|z)) - \beta\ KL(q_\phi(z|x)||p(z))$$

- Increasing the β is forcing variational autoencoder to encode the information in only few latent variables

Disentangled Variational Autoencoders (β-VAEs) (cont.)

Figure 15: Azimuthal rotation in β-VAEs and simple VAEs. β-VAEs produce more disentangled rotation whereas some other features also change in simple VAEs.

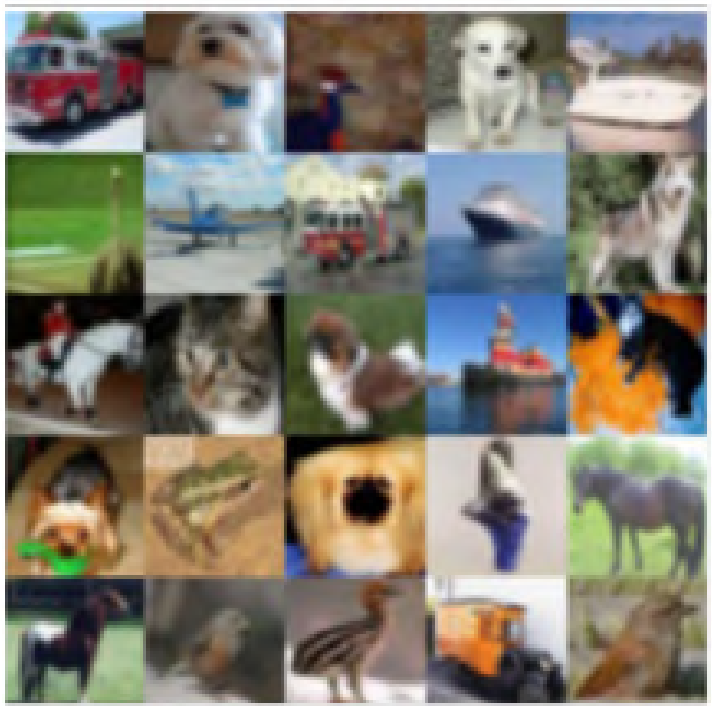
- Variational Autoencoders are based on maximizing likelihood or approximations

$$p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)$$

- 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

Comparing Distributions via Samples

Given a finite set of samples from two distributions $S_1 = \{x \sim P\}$ and $S_2 = \{x \sim Q\}$, how can we tell if these samples are from the same distribution? (i.e., $P = Q$?)



$S_1 = \{\mathbf{x} \sim P\}$

VS.

$S_2 = \{\mathbf{x} \sim Q\}$

Comparing Distributions via Samples (cont.)

- Let's consider a test statistic T for this purpose.
- Test statistic T compares S_1 and S_2 e.g., difference in means, variances of the two sets of samples.
- **Key observation**: Test statistic is likelihood-free since it does not involve the densities P or Q (only samples)

- Let $S_1 = D = \{x \sim p_{data}\}$ and $S_2 = \{x \sim p_\theta\}$
- **Idea**: Train the generative model to minimize a two-sample test objective between S_1 and S_2 i.e., a generator.
- Okay, now how do we get a two-sample test objective?
- **Another Idea**: Train another neural network to discriminate between the two samples i.e., a discriminator.
- And voila, we have a GAN. All that's left is now the training method.

GANs (cont.)

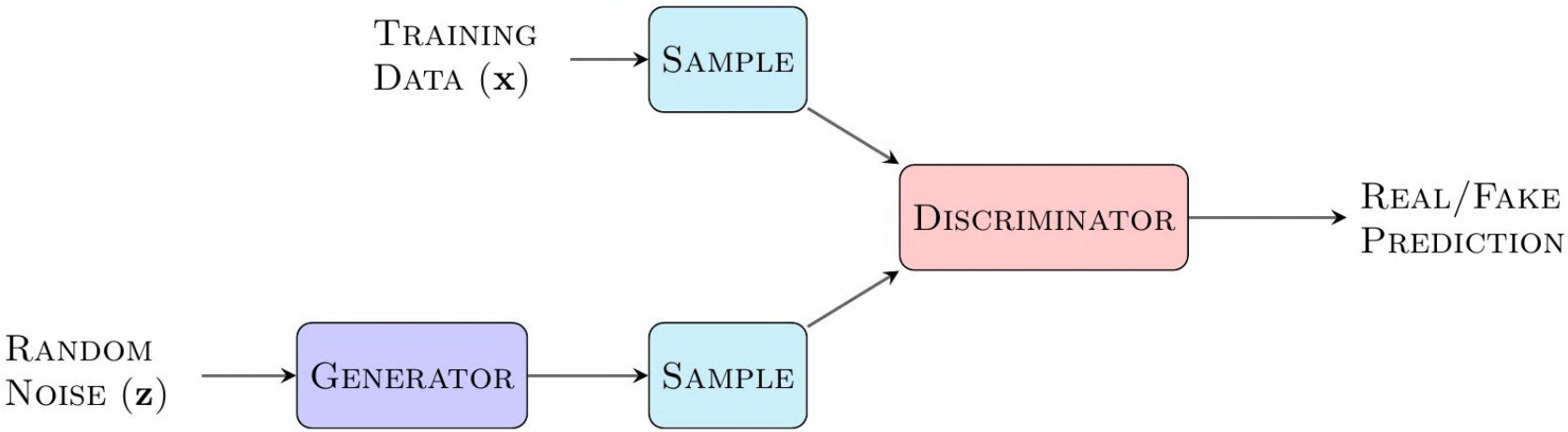


Figure 16: A GAN nodel diagram[1]

GANs (cont.)

- Generative Adversarial Networks were introduced by Ian Goodfellow et al. (2014).
- The idea behind GANs is to train two networks jointly.
- A **discriminator D** to classify samples as "real" or "fake",
- A **generator G** to map a [simple] fixed distribution to samples that fool **D**.
- The approach is **adversarial** since the two networks have antagonistic objectives.

GANs (cont.)

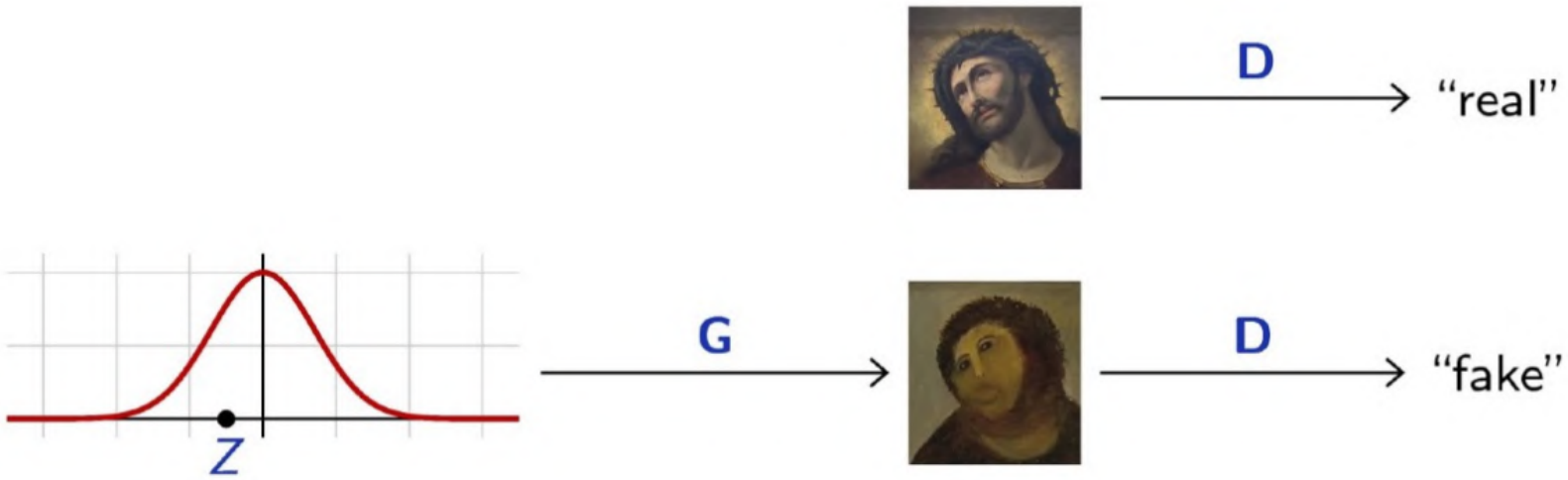


Figure 17: Generator transforms a simple distribution to a sample from data. Discriminator discriminates between real and fake samples.

[1] https://github.com/JamesAllingham/LaTeX-TikZ-Diagrams

Training GANs

- Both Discriminator and Generator are trained jointly in a min-max game.
- Minimax objective function:

$$\min_{\theta_G} \max_{\theta_D} = E_{x \sim p_{data}} log(\underbrace{D_{\theta_D}(x)}_{\substack{\text{Discriminator} \\ \text{output for} \\ \text{real data } x}}) + E_{x \sim p(z)} log(1 - \underbrace{D_{\theta_D}(G_{\theta_G}(z))}_{\substack{\text{Discriminator output} \\ \text{for generated fake} \\ \text{data } G(z)}})$$

- Discriminator θ_D wants to maximise objective such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake).
- Generator θ_G wants to minimise objective such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real).

Training GANs (cont.)

for number of training iterations **do**

for k steps **do**

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

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

end for

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

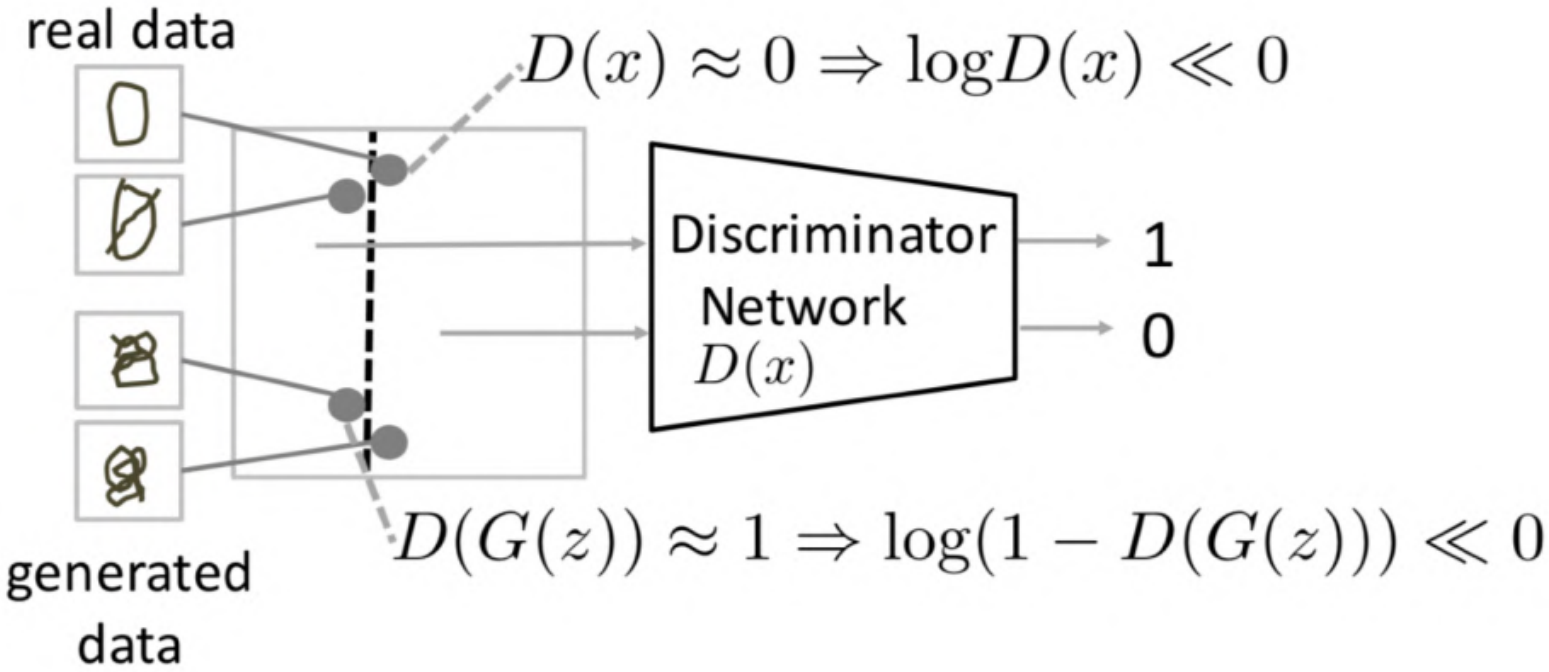
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Understanding the Objective function

$$\max_D(\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))])$$

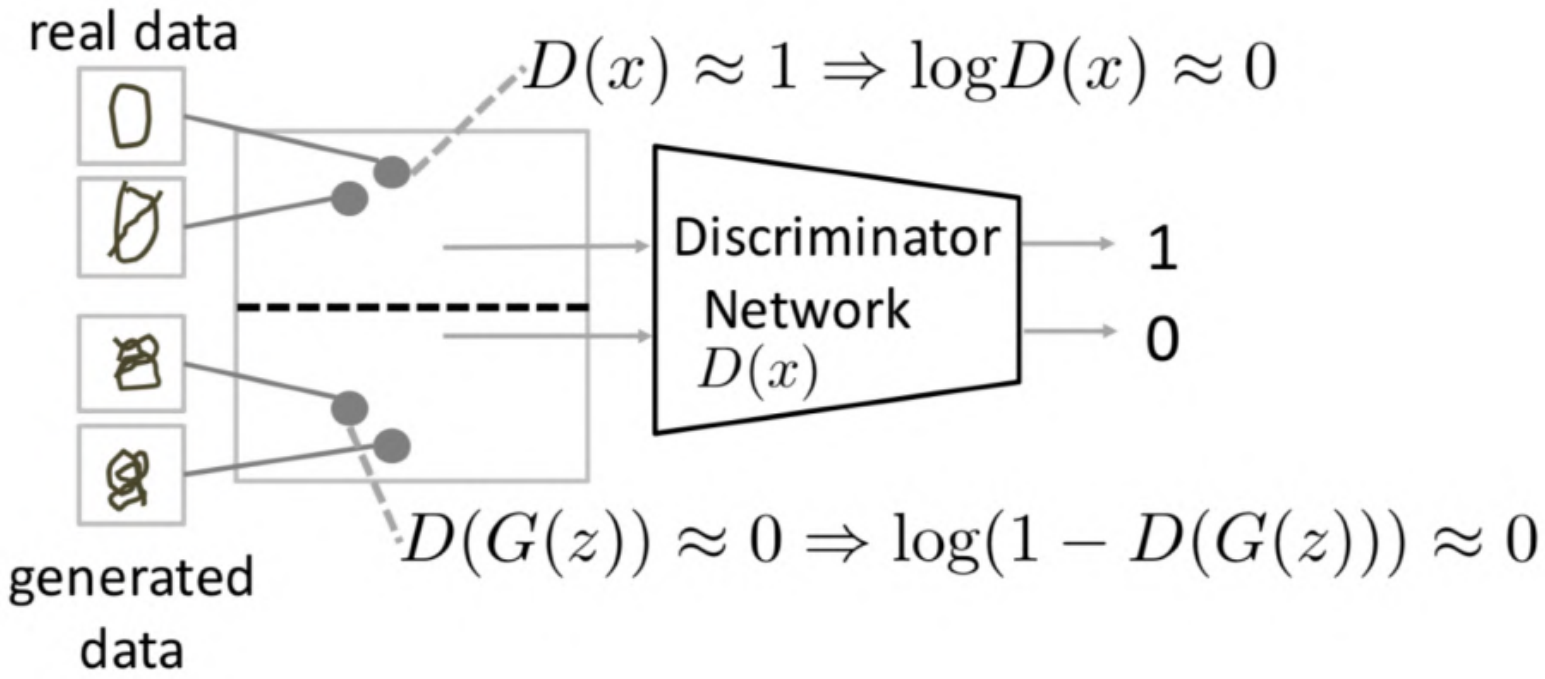
$D(x)$ should be 1 $D(G(z))$should be 0



Understanding the Objective function (cont.)

$$\max_D(\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))])$$

$D(x)$ should be 1 $D(G(z))$should be 0

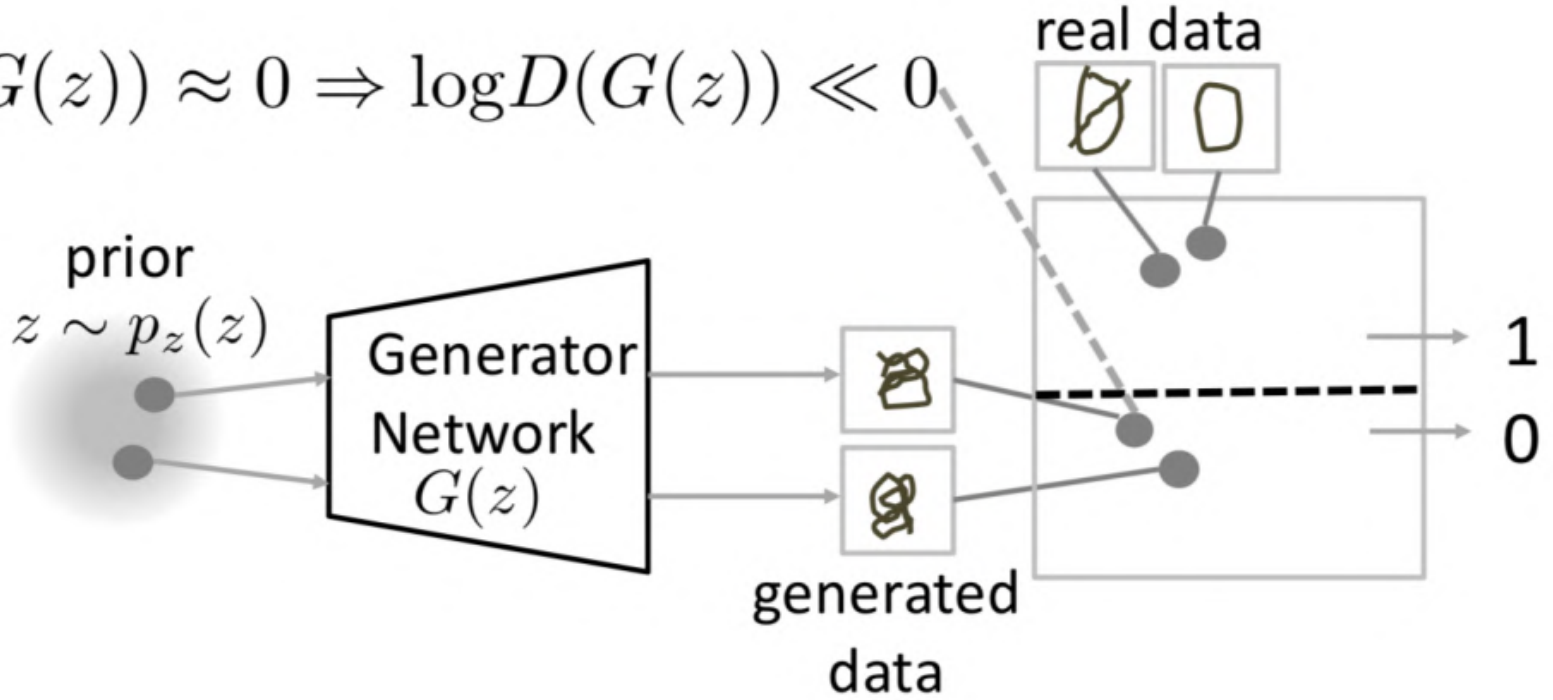


Understanding the Objective function (cont.)

$$\max_G(\mathbb{E}_{z\sim p_z(z)}[\log(D(G(z))])$$

$D(G(z))$ should be 1

$D(G(z)) \approx 0 \Rightarrow \log D(G(z)) \ll 0$

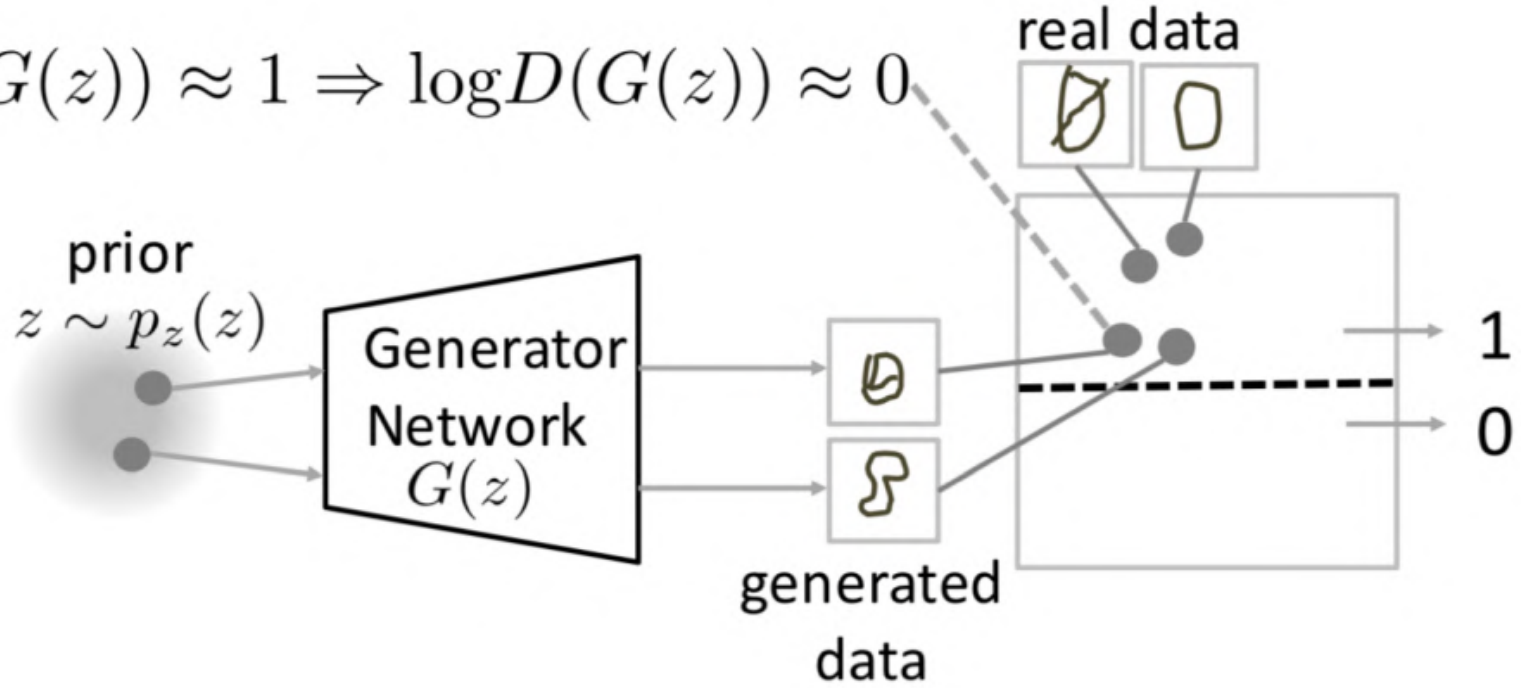


Understanding the Objective function (cont.)

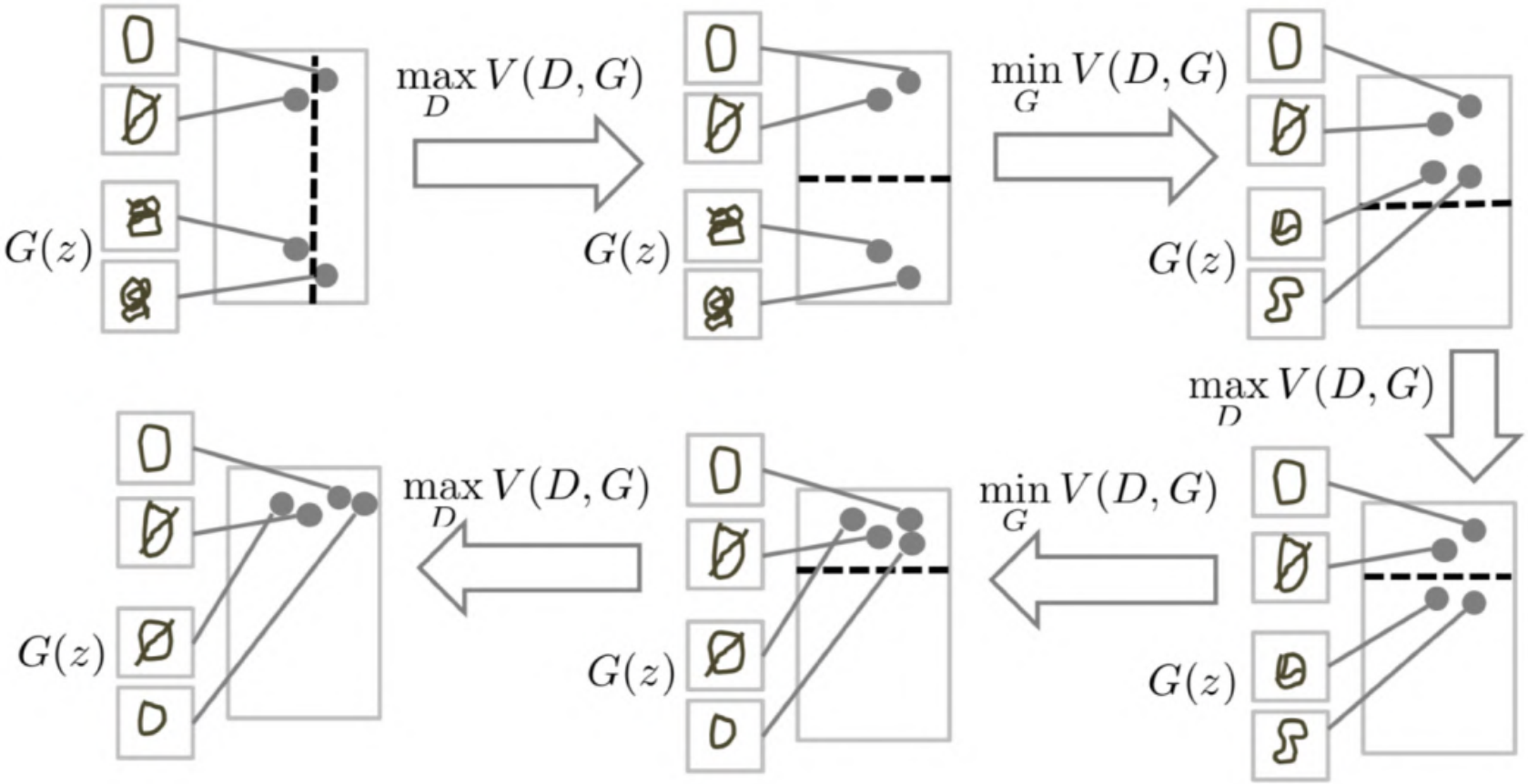
$$\max_G(\mathbb{E}_{z\sim p_z(z)}[\log(D(G(z))])$$

$D(G(z))$ should be 1

$D(G(z)) \approx 1 \Rightarrow \log D(G(z)) \approx 0$



Understanding the Objective function (cont.)



[1] https://www.slideshare.net/ckmarkohchang/generative-adversarial-networks

GANs - Interactive Demo

https://poloclub.github.io/ganlab/

GANs - Results

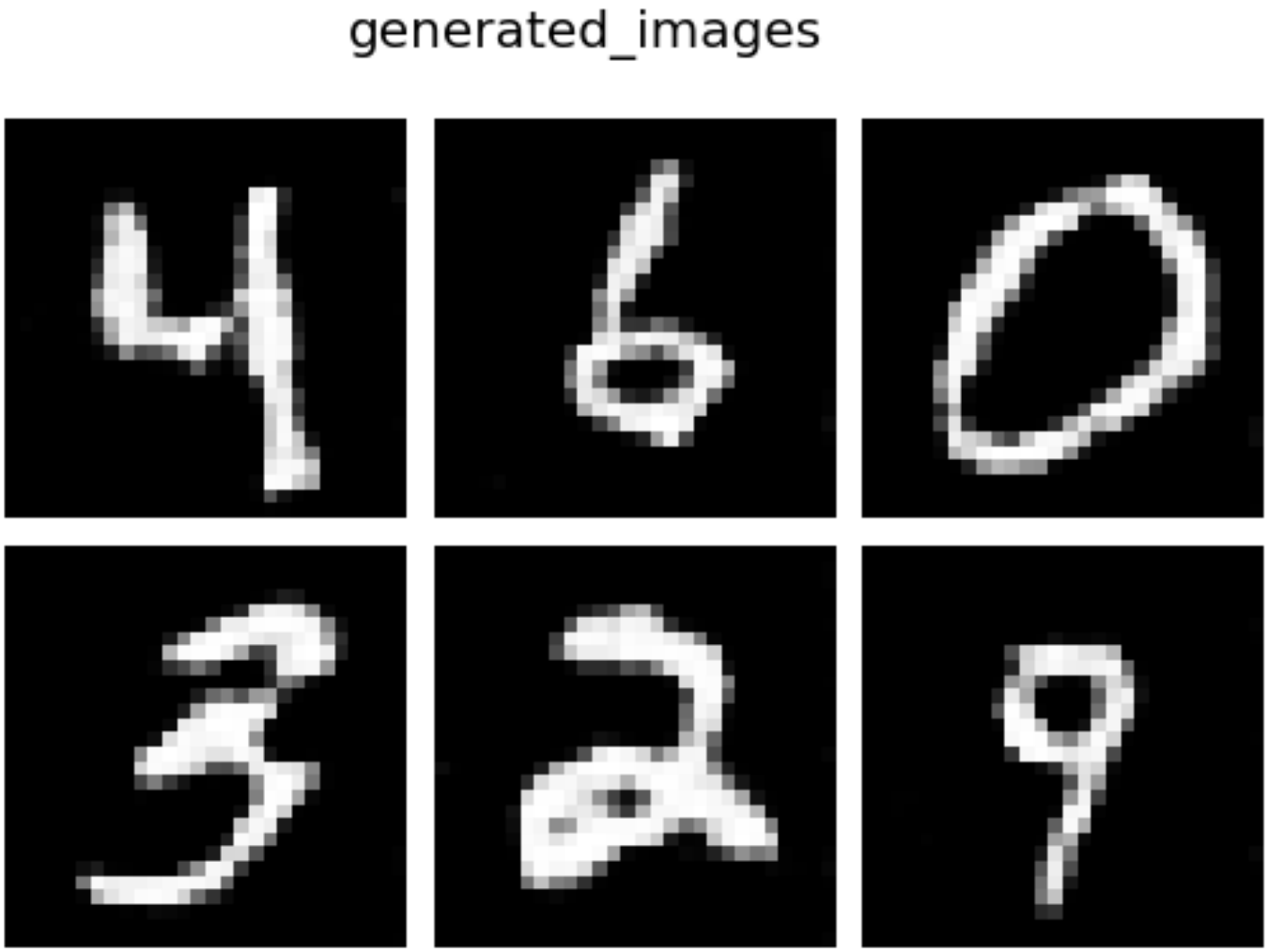


Figure 18: GAN generated samples for MNIST digits dataset

GANs - Results (cont.)

Figure 19: GAN generated samples for bedroom images

GANs - Results (cont.)

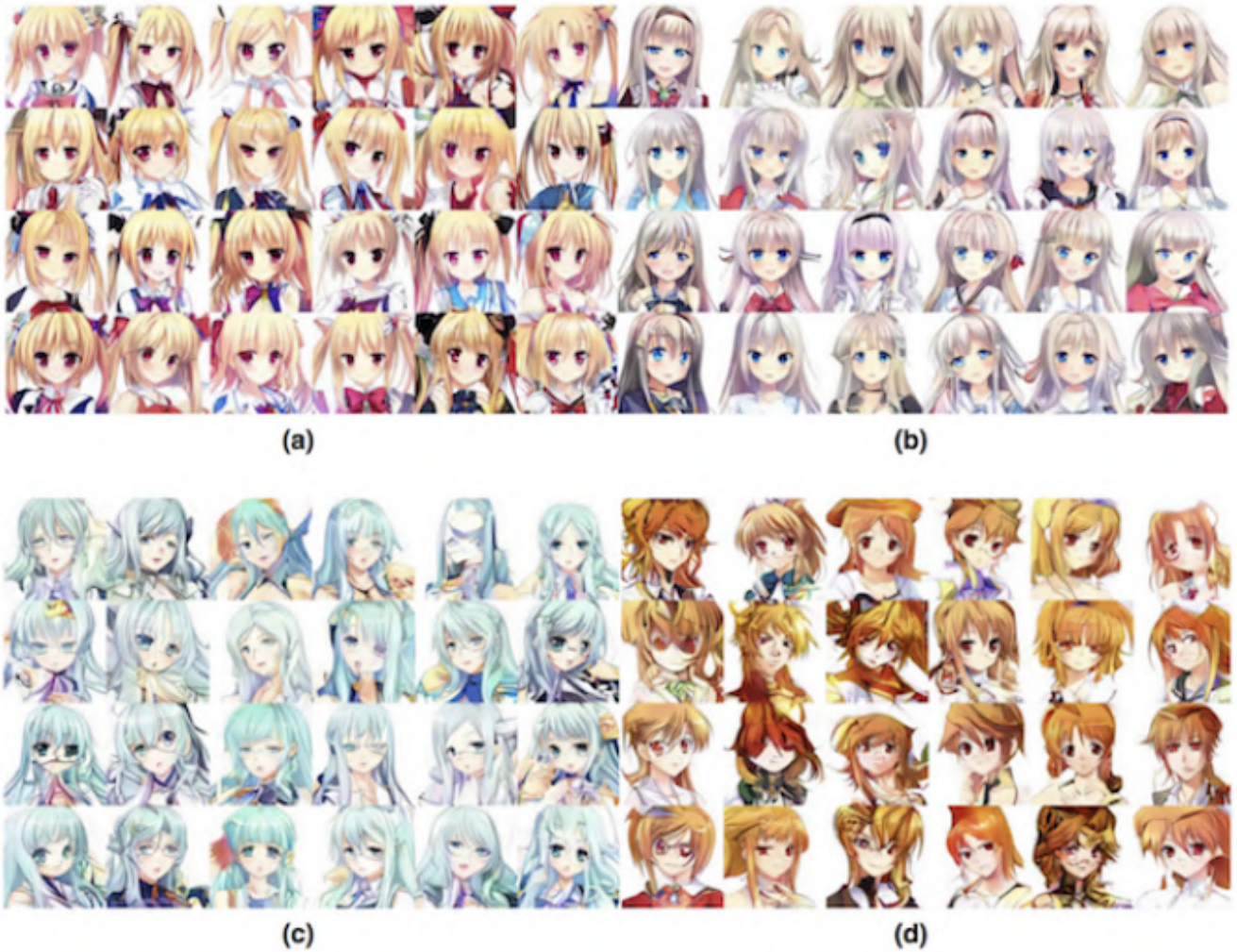


Figure 20: GAN generated samples for anime character faces

Problems with GANs

- Vanishing Gradients
  - When the discriminator is perfect, we are guaranteed with $D(x) = 1, \forall x \in p_{data}$ and $D(x) = 0, \forall x \in p_G$.
  - Loss function falls to zero and we end up with no gradient to update.
  - **Solution**: Perform gradient ascent on generator i.e., different objective.

$$\max_{\theta_G} E_{x \sim p(z)} log(D_{\theta_D}(G_{\theta_G}(z)))$$

Problems with GANs (cont.)

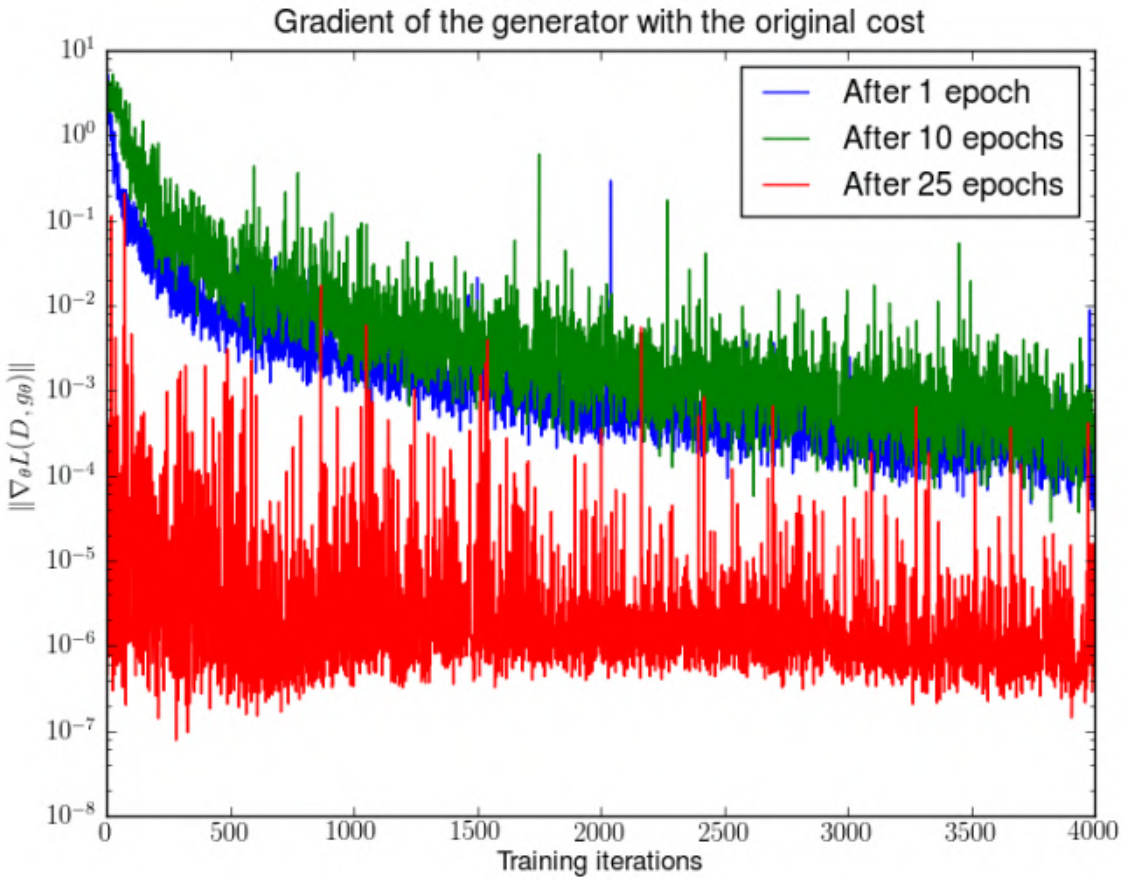


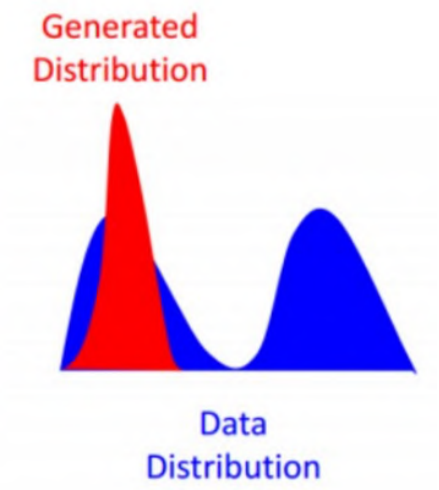
Figure 21: Vanishing gradient in GANs (Image source: Arjovsky and Bottou, 2017)

Problems with GANs (cont.)

- Hard to achieve Nash equilibrium
  - Two models are trained simultaneously to find a Nash equilibrium to a two player non-cooperative game. However, each model updates its cost independently with no respect to another player in the game. Updating the gradient of both models concurrently cannot guarantee a convergence
  - How to Train a GAN? Tips and tricks to make GANs work by Soumith Chintala
    https://github.com/soumith/ganhacks

Problems with GANs (cont.)

- Mode collapse
  - Remember the generator's goal is to trick the discriminator D into thinking that the outputs by G are real, if generator G produces one sample which is realistic to the original data, then the discriminator finds it hard to distinguish between them.
  - Fixes to mode collapse are mostly empirically driven: alternative architectures, alternative GAN loss, adding regularization terms, etc.



Problems with GANs (cont.)

Figure 22: GAN mode collapse on MNIST digits dataset

GAN Variants

- After the invention of GANs, there has been done a lot of research around them.
- A named list of GANs can be found here.
- We will discuss the following variants here:
  - Wasserstein GAN
  - Conditional GAN
  - Cycle GAN

Wasserstein GAN

- Wasserstein GAN uses wasserstein distance instead of crossentropy loss.
- Wasserstein distance that has a smoother gradient everywhere.

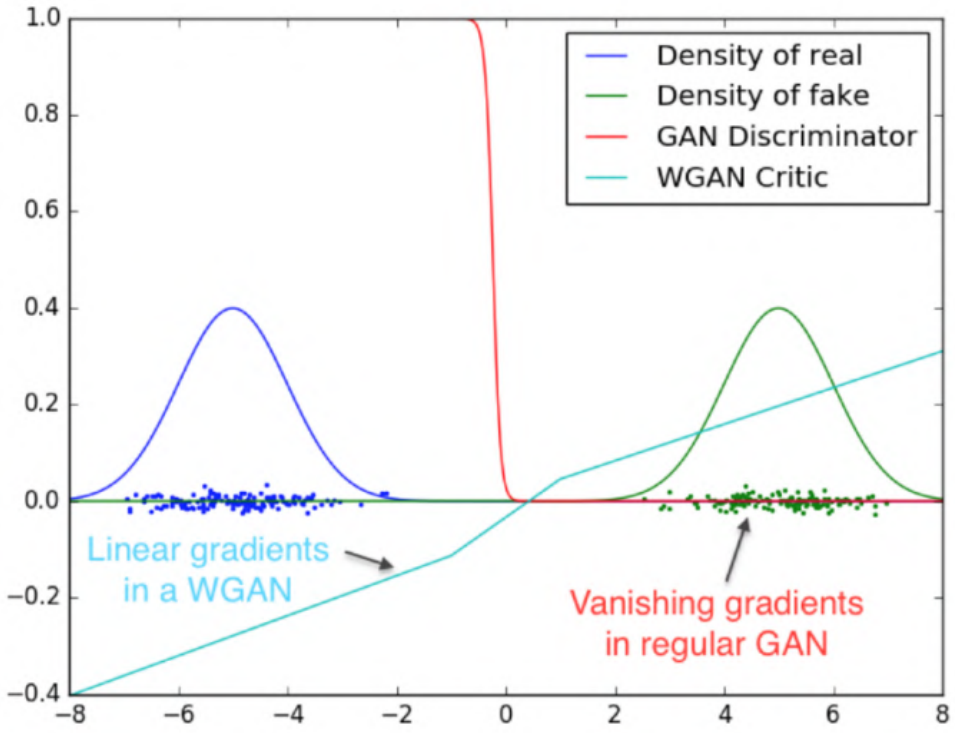
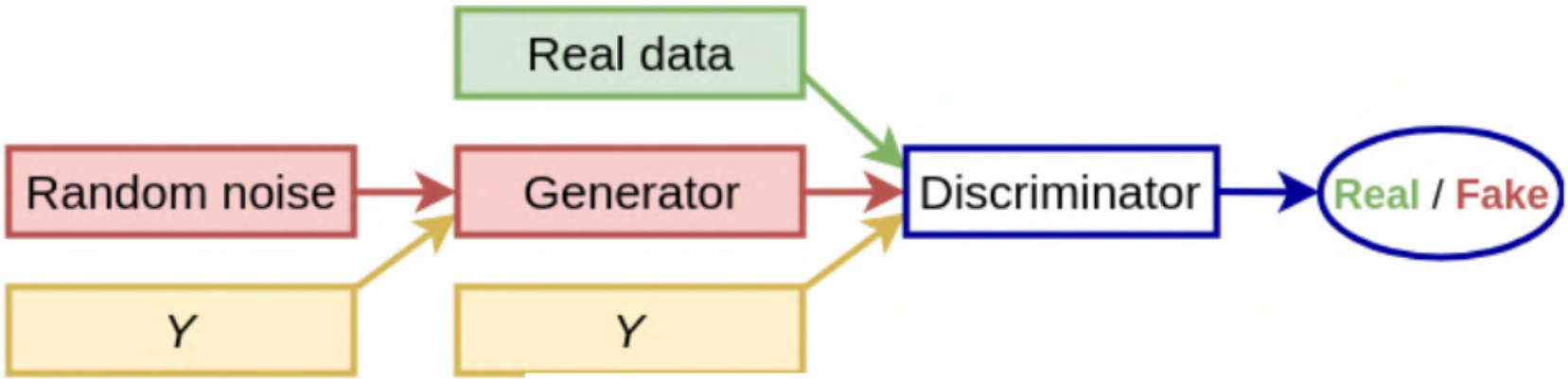


Figure 23: WGAN generation results on bedroom images

Conditional GAN

- Add a condition in addition to a random noise in the generator input. Similar case with discriminator.



Condional GAN - Image to Image

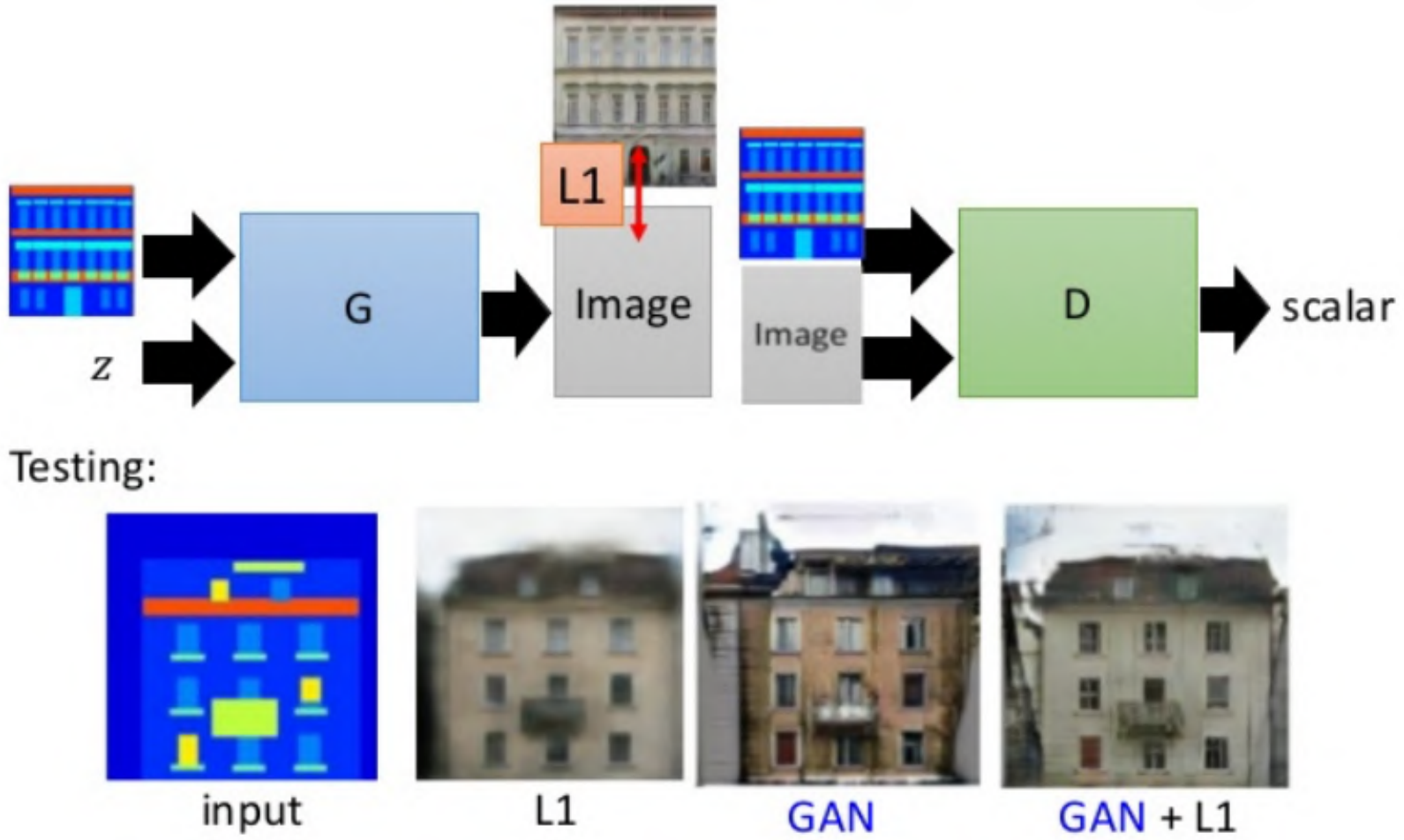


Figure 24: Using L1 loss in addition in GAN loss can help in image to image translation

Condional GAN - Image to Image (cont.)

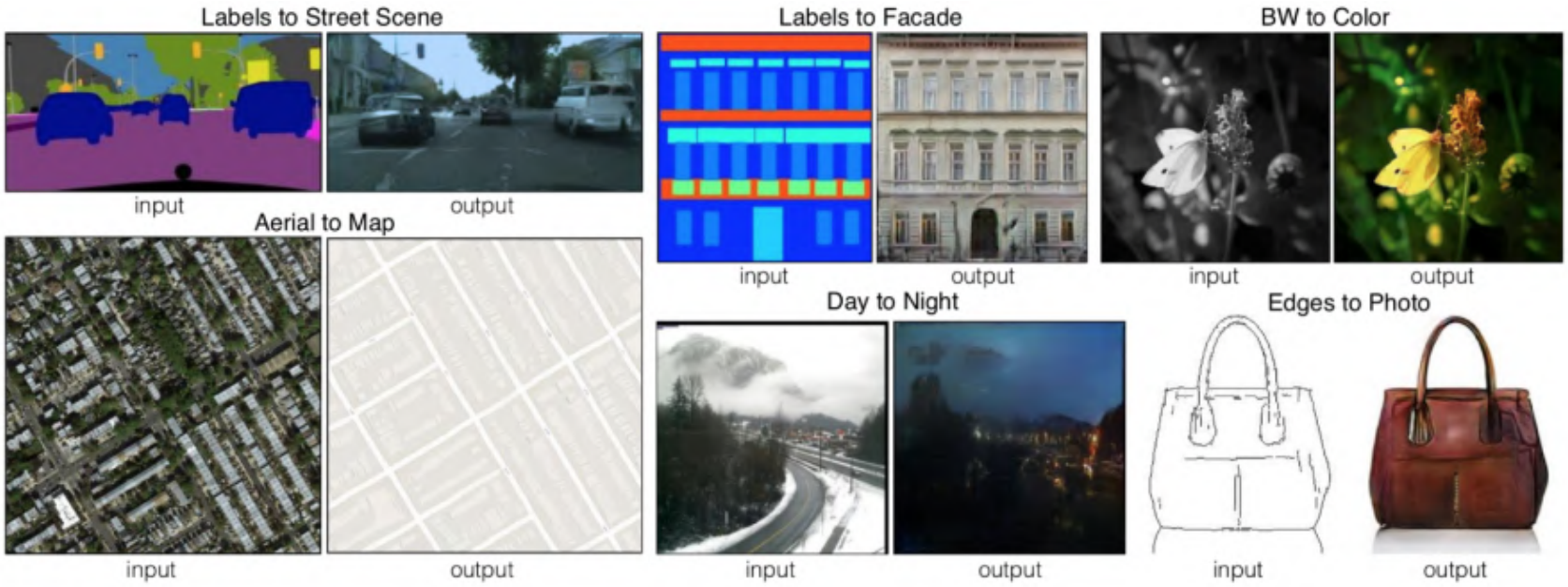
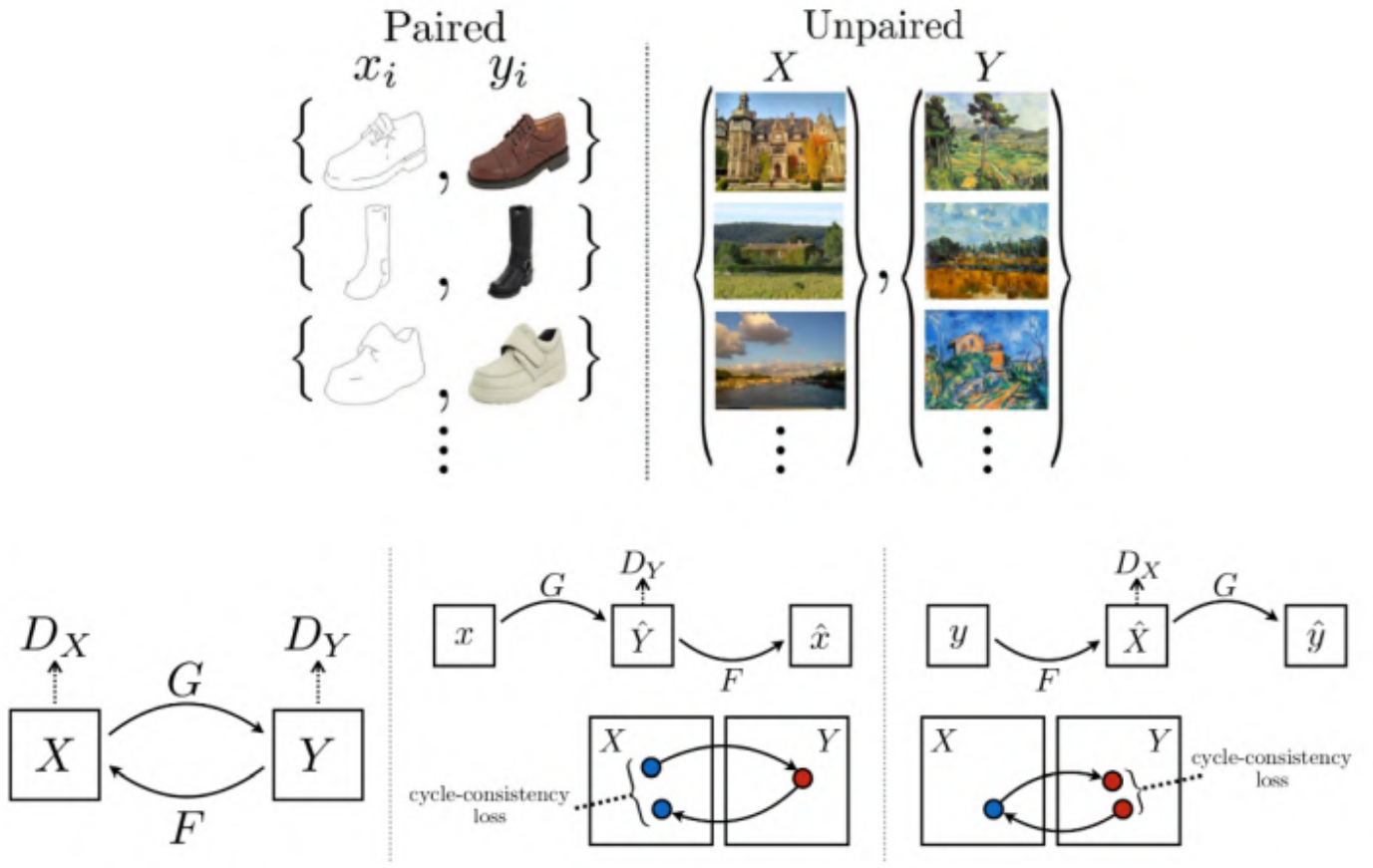


Figure 25: Image to Image translation with condional GANs

- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, Efros, ICC7 2017



Cycle GANs (cont.)

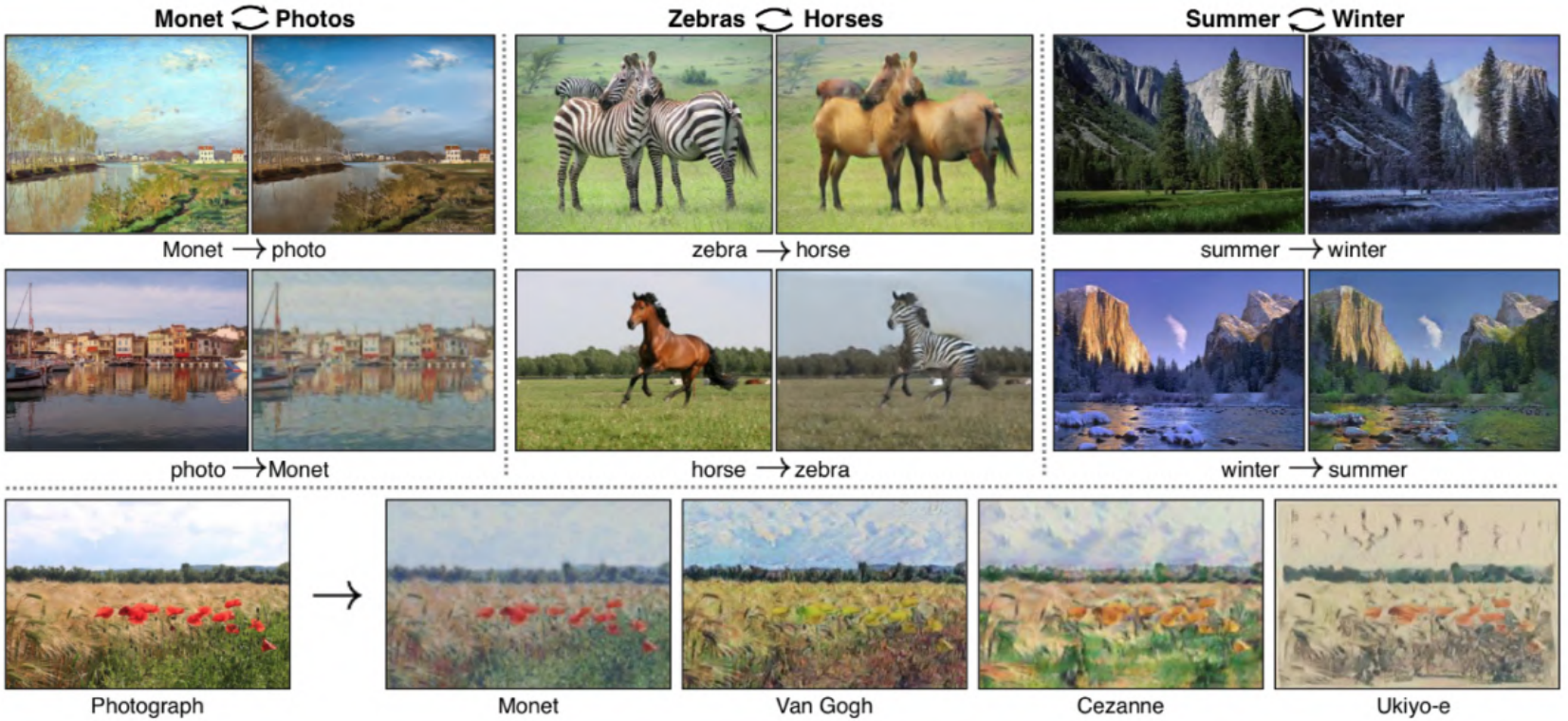


Figure 26: Image to Image translation with Cycle GANs

GANs - Latent space Interpolation

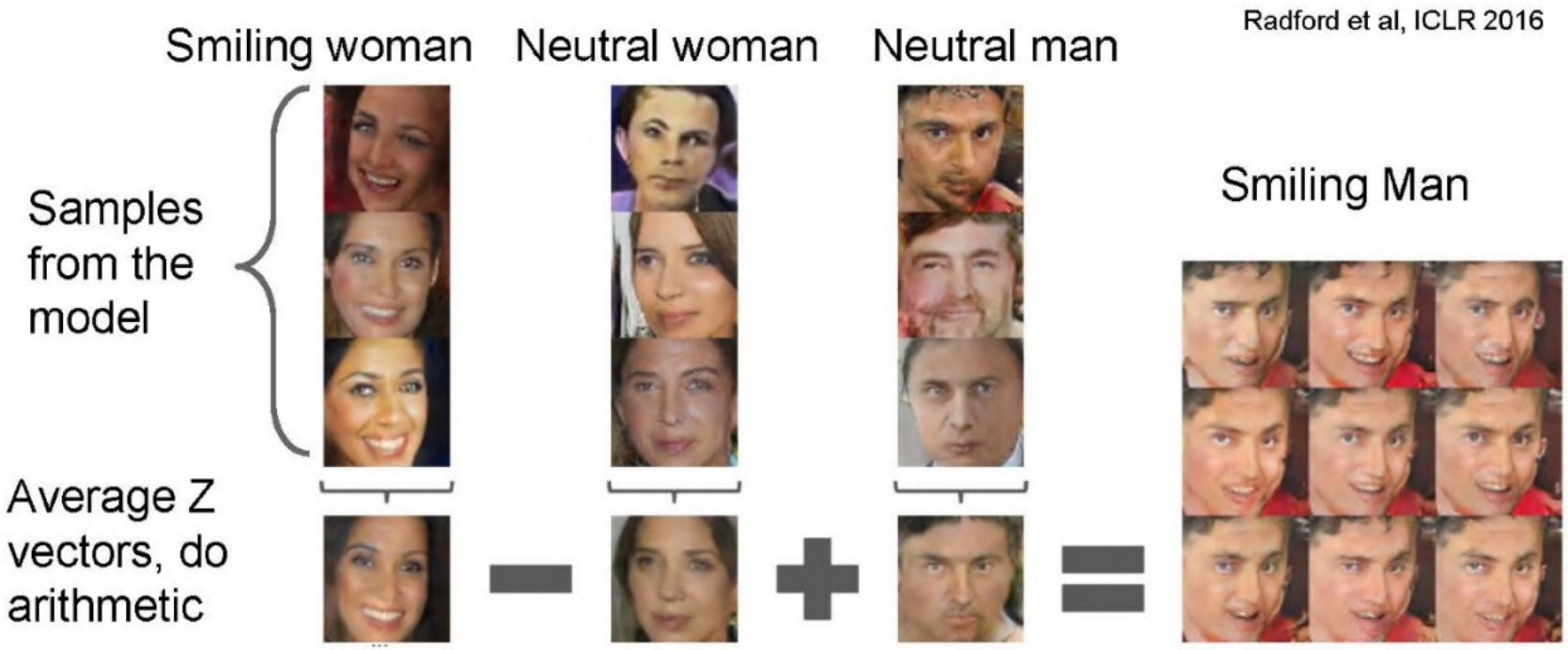


Figure 27: Latent space interploation with GANs

GANs - Latent space Interpolation (cont.)

Latent space interpolation with StyleGAN
(Demo by Xander Steenburgge)
https://colab.research.google.com/drive/1mH70YxGNInEaSOn0J8LsgkI-QOvsIb3MscrollTo=uEhxBvAR-7y3

Some more GAN results

Figure 28: Image Inpainting.
https://www.nvidia.com/en-us/research/ai-demos/

Some more GAN results (cont.)

Figure 29: Text to Image Synthesis with GANs

Some more GAN results (cont.)

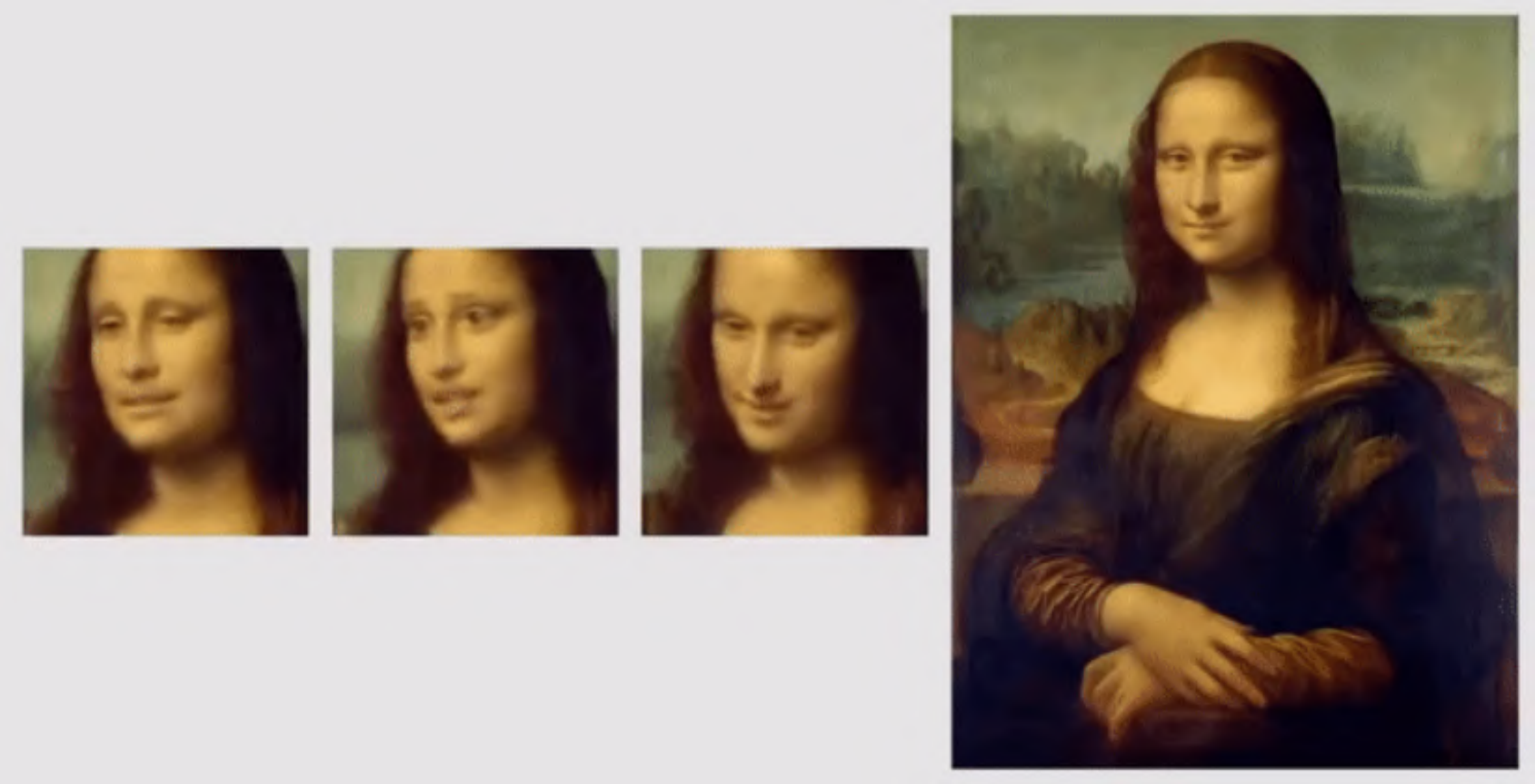


Figure 30: Living Portraits with GANs

Some more GAN results (cont.)

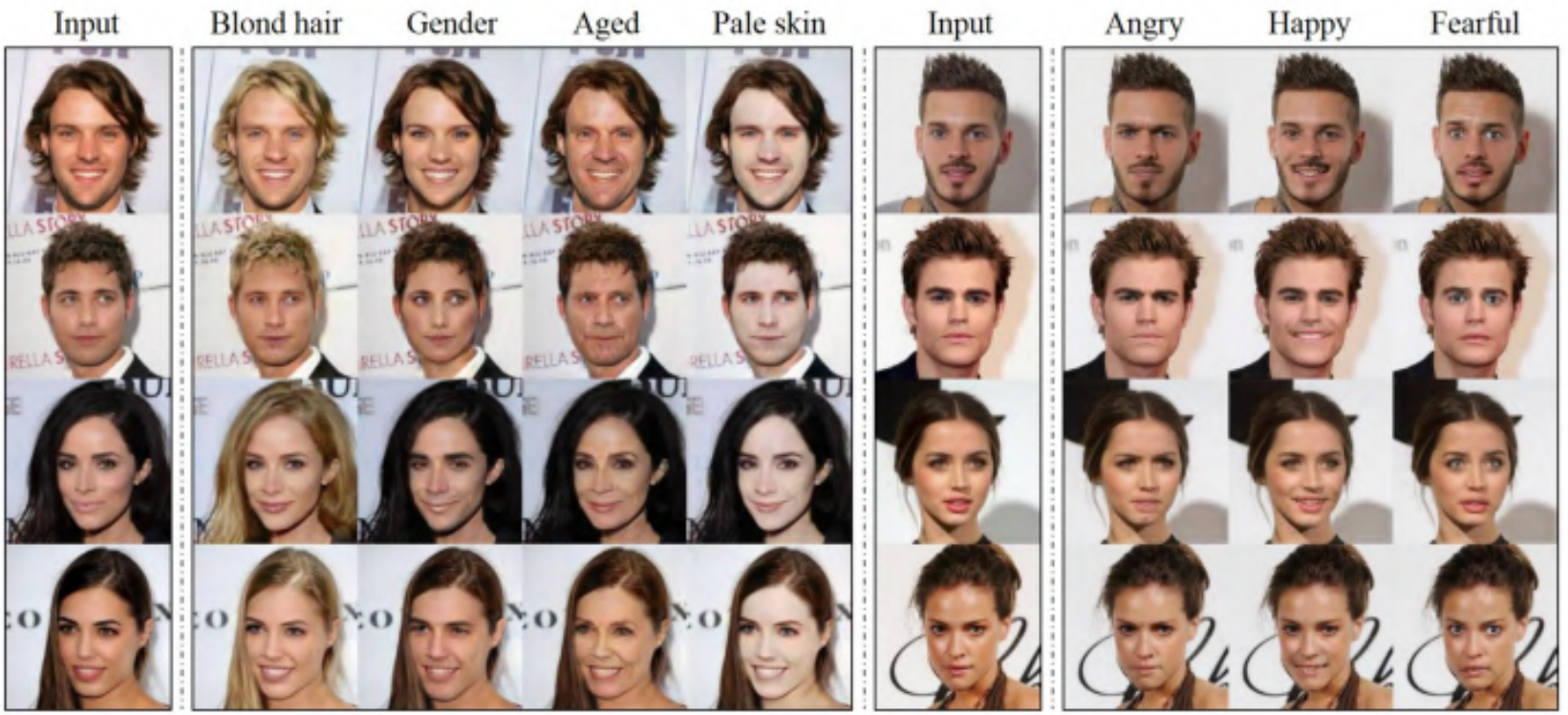


Figure 31: Image to Image translation in multiple domains with StyleGAN (Choi et al.)

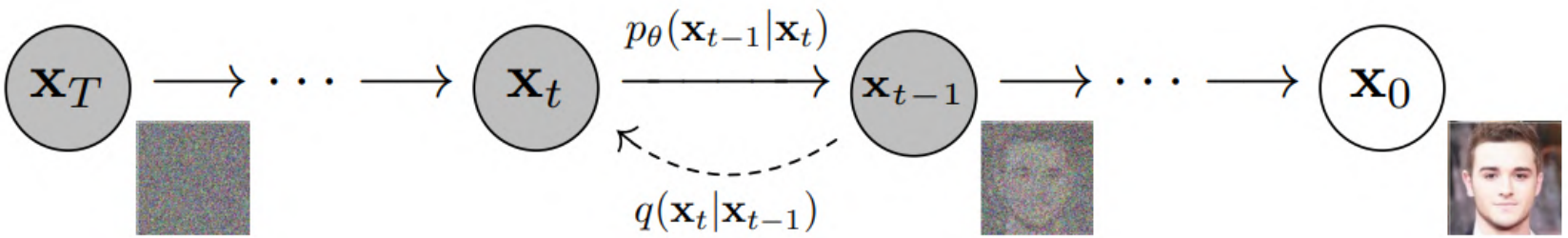
- Denoising diffusion models, now also known as score-based generative models, have recently emerged as a powerful class of generative models. They demonstrate astonishing results in high-fidelity image generation, often even outperforming GANs.

Figure 32: Diffusion Models Beat GANs on Image Synthesis Dharwal & Nichol, OpenAI, 2021

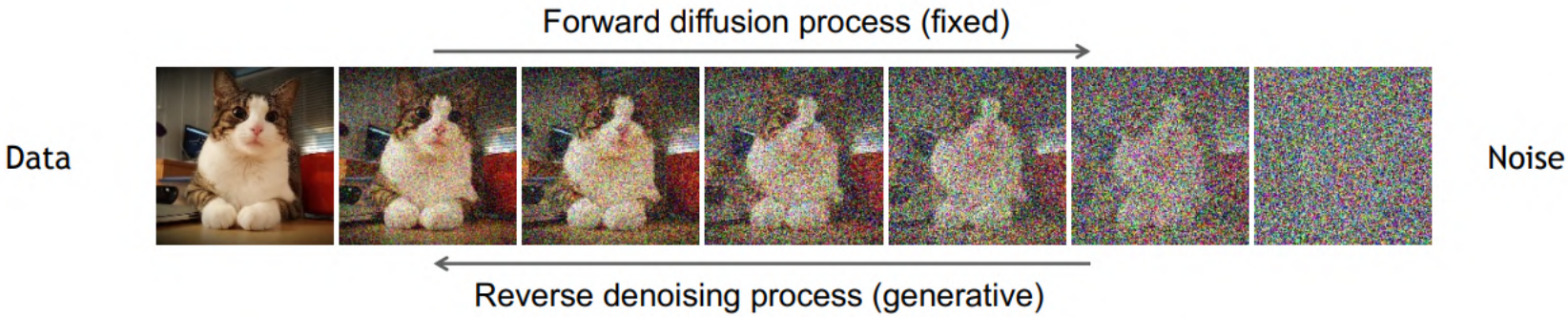
Denoising Diffusion Probabilistic Models (cont.)

Denoising diffusion models consist of two processes:

- A fixed (or predefined) forward diffusion process q of our choosing, that gradually adds Gaussian noise to an image, until you end up with pure noise
- a learned reverse denoising diffusion process p_θ , where a neural network is trained to gradually denoise an image starting from pure noise, until you end up with an actual image.



Denoising Diffusion Probabilistic Models (cont.)



Forward Diffusion Process

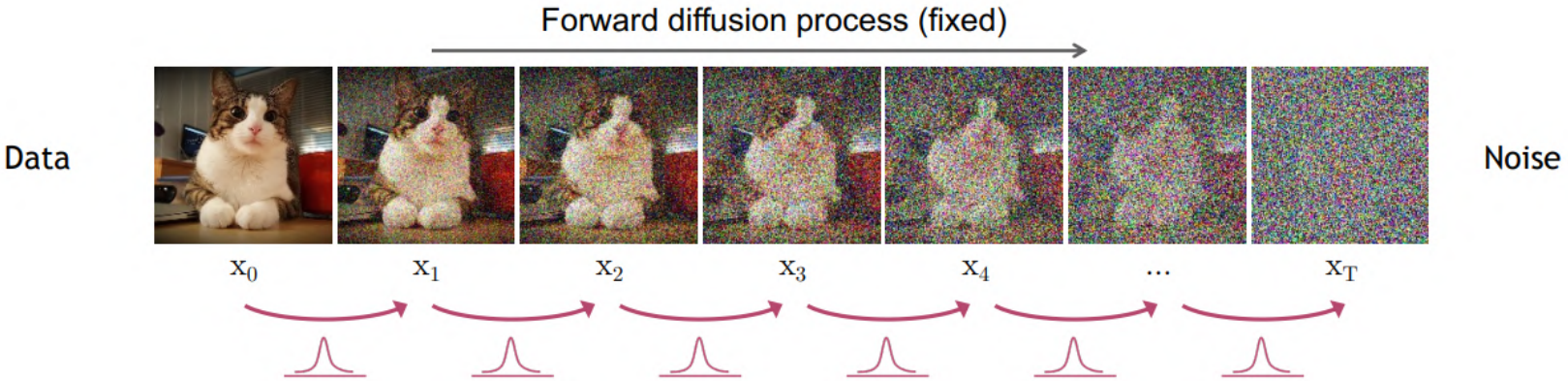
- Iteratively add Gaussian noise to the image.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \longrightarrow q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

  where β is a known variance schedule such that $0 < \beta_1 < \beta_2 < \cdots < \beta_T < 1$. Usually β is fixed but it can also be learned. This variance schedule can be linear, quadratic, cosine, etc.

- Basically, each new (slightly noisier) image at time step t is drawn from a conditional Gaussian distribution with $\mu_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1}$ and $\sigma_t^2 = \beta_t$, which we can do by sampling $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then setting $\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon$.

Forward Diffusion Process (cont.)



- What if want to go directly from q_0 to q_t?
- Let $\overline{\alpha}_t = \prod_{s-1}^{t}(1-\beta_s)$, then

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1-\overline{\alpha}_t)\mathbf{I})$$

- This allows us, during training, to optimize random terms of the loss function L (or in other words, to randomly sample t during training and optimize L_t).

Reverse Denoising process

- In the reverse denoising process, we denoise Gaassian noise to generate an image from it.
- Basically, sample $p(\mathbf{x}_T = \mathcal{N}(\mathbf{x}_T; 0, 1)$
- Then, $\mathbf{x}_{t-1} \sim p(\mathbf{x}_{t-1}|\mathbf{x}_t)$
- However, we don't know $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$.
- So, let's approximate(learn) it with a neural network.
- If β_t is small enough at each time step we can assume $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to be a Gaussian Distribution. So, we can parameterize the process as

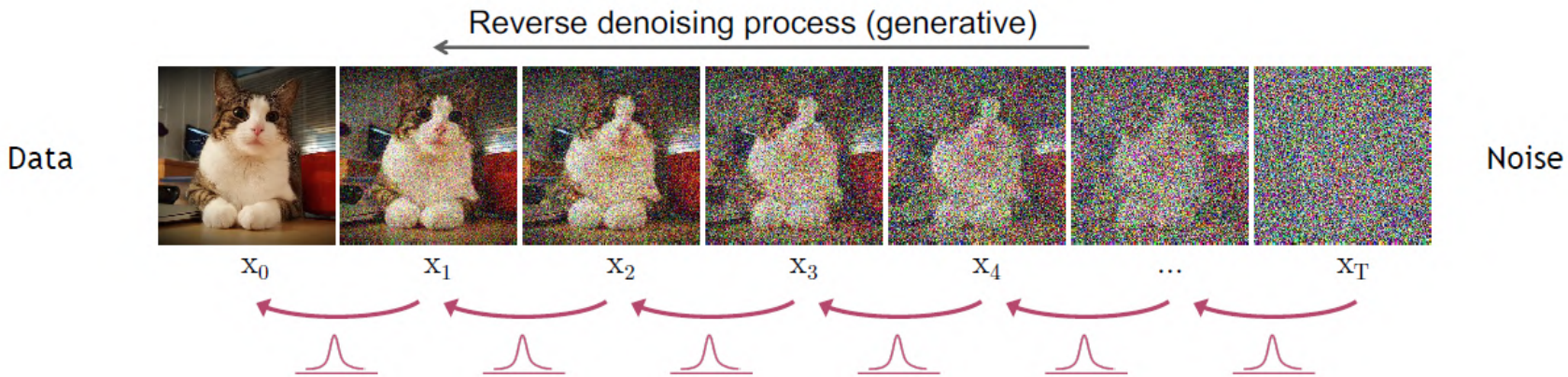
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

where μ_θ and Σ_θ are neural networks. Mean and variance are also conditioned on t.

Reverse Denoising process (cont.)

- But we fix variance for now, so we only need μ_θ. Later variants also learn Σ_θ.

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \beta_t \mathbf{I}) \longrightarrow p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_t) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$



- For training, we can form variational upper bound that is commonly used for training variational autoencoders:

$$E_{q(x_0)}[-log\, p_\theta(\mathbf{x}_0)] \leq E_{q(x_0)q(\mathbf{x}_{1:T}|x_0)}\left[-log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|x_0)}\right]$$

- It can be shown that the ELBO for this process is a sum of losses at each time step t,

$$L = L_0 + L_1 + ... + L_T$$

Learning Denoising Models (cont.)

- Lastly, we can reparametrize the mean to make the neural network learn (predict) the added noise (via a network $\epsilon_\theta(\mathbf{x}_t, t)$ for noise level t in the KL terms which constitute the losses. This means that our neural network becomes a noise predictor, rather than a (direct) mean predictor. The mean can be computed as follows:

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

- The final objective function L_t then looks as follows (for a random time step t given $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$):

$$\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2 = \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon, t)\|^2.$$

- For complete derivation, read here

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right) \right\|^2$$

6: **until** converged

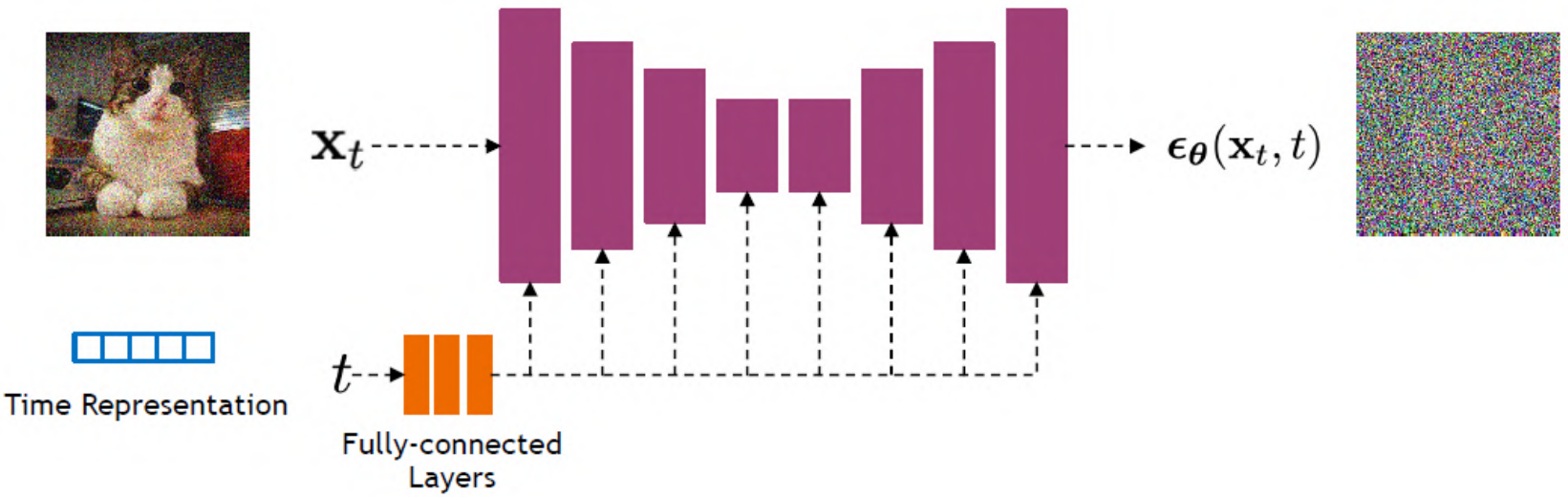
Summary (cont.)

Algorithm 2 Sampling

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

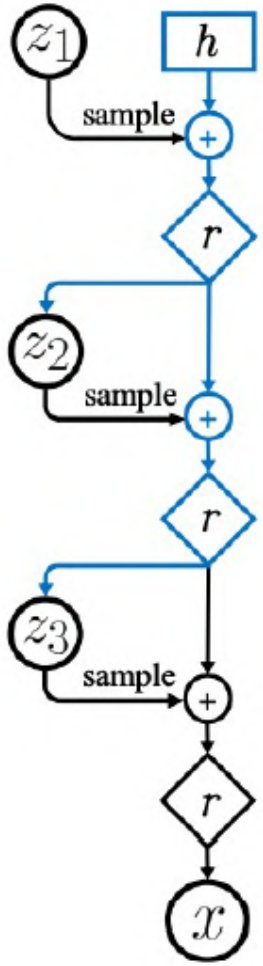
Network Architectures

- Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(x_t, t)$
- Time representation: sinusoidal positional embeddings or random Fourier features.



Connection to VAEs

- Diffusion models can be considered as a special form of hierarchical VAEs (one VAE after another).
- However, in diffusion models:
  - The encoder is fixed
  - The latent variables have the same dimension as the data
  - The denoising model is shared across different timestep
  - The model is trained with some reweighting of the variational bound.



Problems with Diffusion Models

- For now, it seems that the major limitation of the Diffusion Models is its notoriously slow sampling procedure which normally requires hundreds to thousands of time discretization steps of the learned diffusion process to reach the desired accuracy.

Trilemma of generative learning

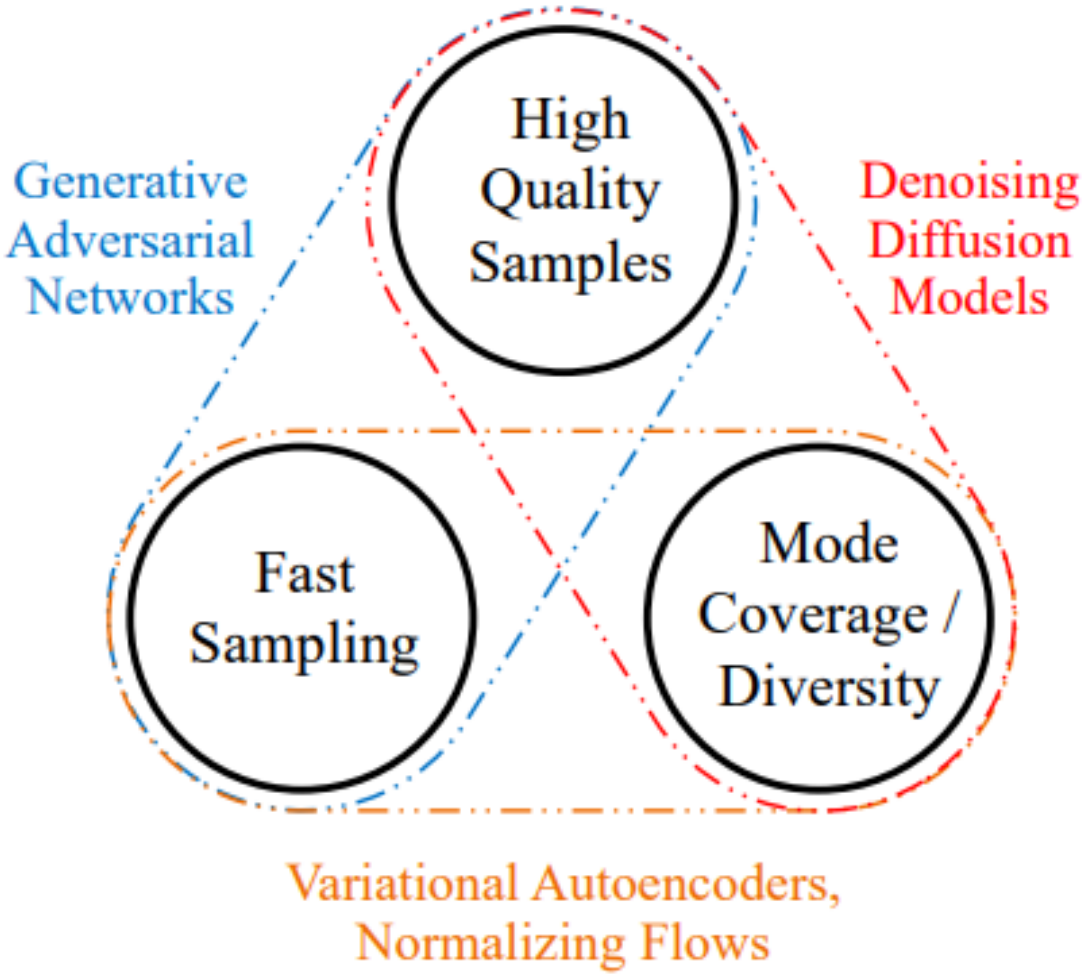


Figure 33: Generative learing Trilemma

[1] Tackling the Generative Learning Trilemma with Denoising Diffusion GANs

Diffusion Models - Current Research

- Denoising Diffusion Implicit Model
  - DDIM roughly sketches the final sample then refine it with the reverse process.
- Improved Denoising Diffusion Probabilistic Models
  - Train σ^2 while training the diffusion model instead of fixing it.
- Score-Based Generative Modeling through Stochastic Differential Equations
  - Model the gradient of the log probability density function, a quantity often known as the (Stein) score function.
- Tackling the Generative Learning Trilemma with Denoising Diffusion GANs
  - Introduce denoising diffusion generative adversarial networks (denoising diffusion GANs) that model each denoising step using a multimodal conditional GAN.
- Cascaded Diffusion Models for High Fidelity Image Generation
  - Cascaded diffusion models to boost sample quality.

GLIDE - OpenAI

Figure 34: Image Inpainting with GLIDE

[1] GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models

DALL.E 2 - OpenAI

Figure 35: Text to image generation eith DAll.E 2

[1] Hierarchical Text-Conditional Image Generation with CLIP Latents

Figure 36: Image Variations

[1] Hierarchical Text-Conditional Image Generation with CLIP Latents

Interpolate image CLIP embeddings $\mathbf{z}_i$

Use different $\mathbf{x}_T$ to get different interpolation trajectories.

Figure 37: Image interpolation

[1] Hierarchical Text-Conditional Image Generation with CLIP Latents

a photo of a cat → an anime drawing of a super saiyan cat, artstation

a photo of a victorian house → a photo of a modern house

a photo of an adult lion → a photo of lion cub

Change the image CLIP embedding towards the difference of the text CLIP embeddings of two prompts.

Decoder latent is kept as a constant.

Figure 38: Text Difference Image interpolation

[1] Hierarchical Text-Conditional Image Generation with CLIP Latents

Imagen - Google

A brain riding a rocketship heading towards the moon.

[1] Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Imagen - Google

A dragon fruit wearing karate belt in the snow.

[1] Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Imagen - Google

A relaxed garlic with a blindfold reading a newspaper while floating in a pool of tomato soup.

[1] Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Diffusion Autoencoders

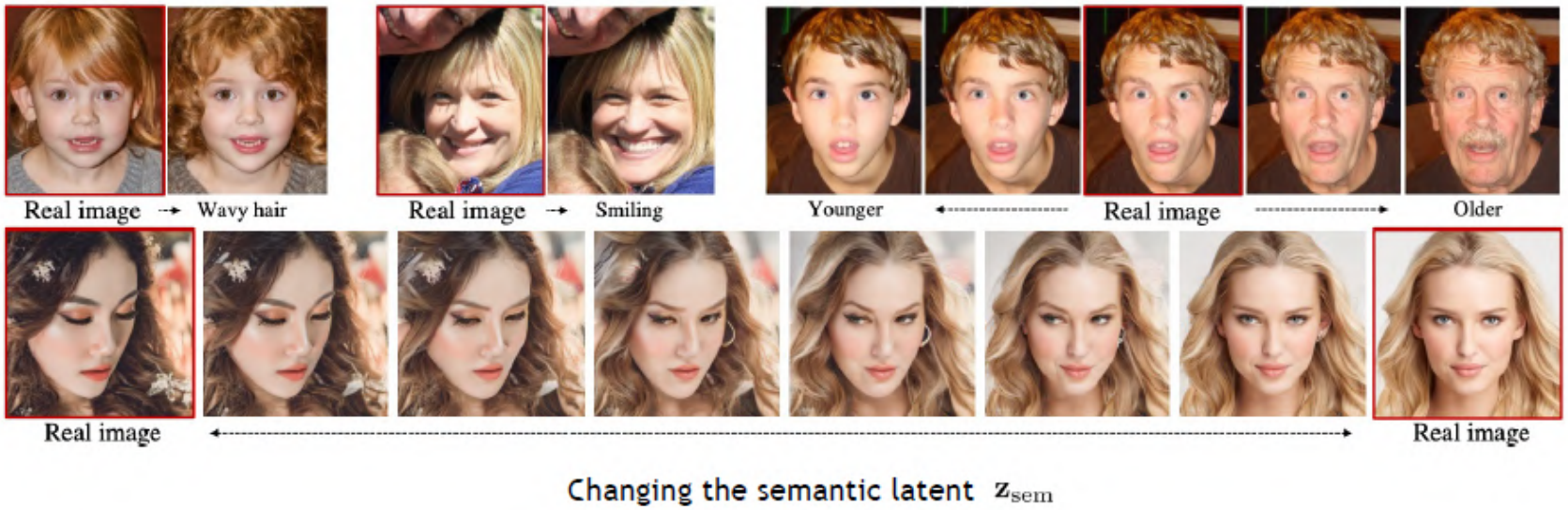


Figure 39: Learning semantic meaningful latent representations in diffusion models

[1] Diffusion Autoencoders: Toward a Meaningful and Decodable Representation

Super Resolution

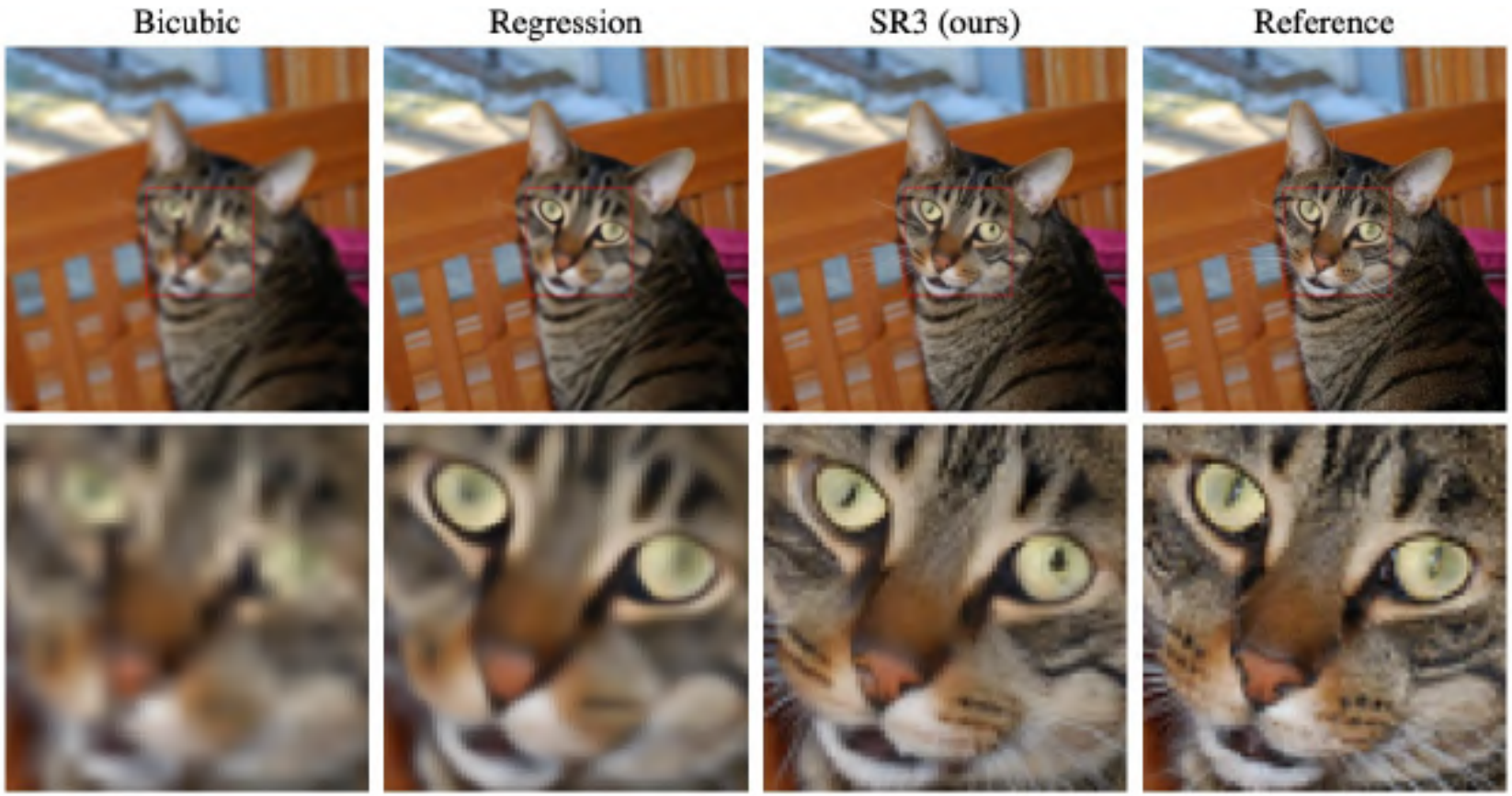
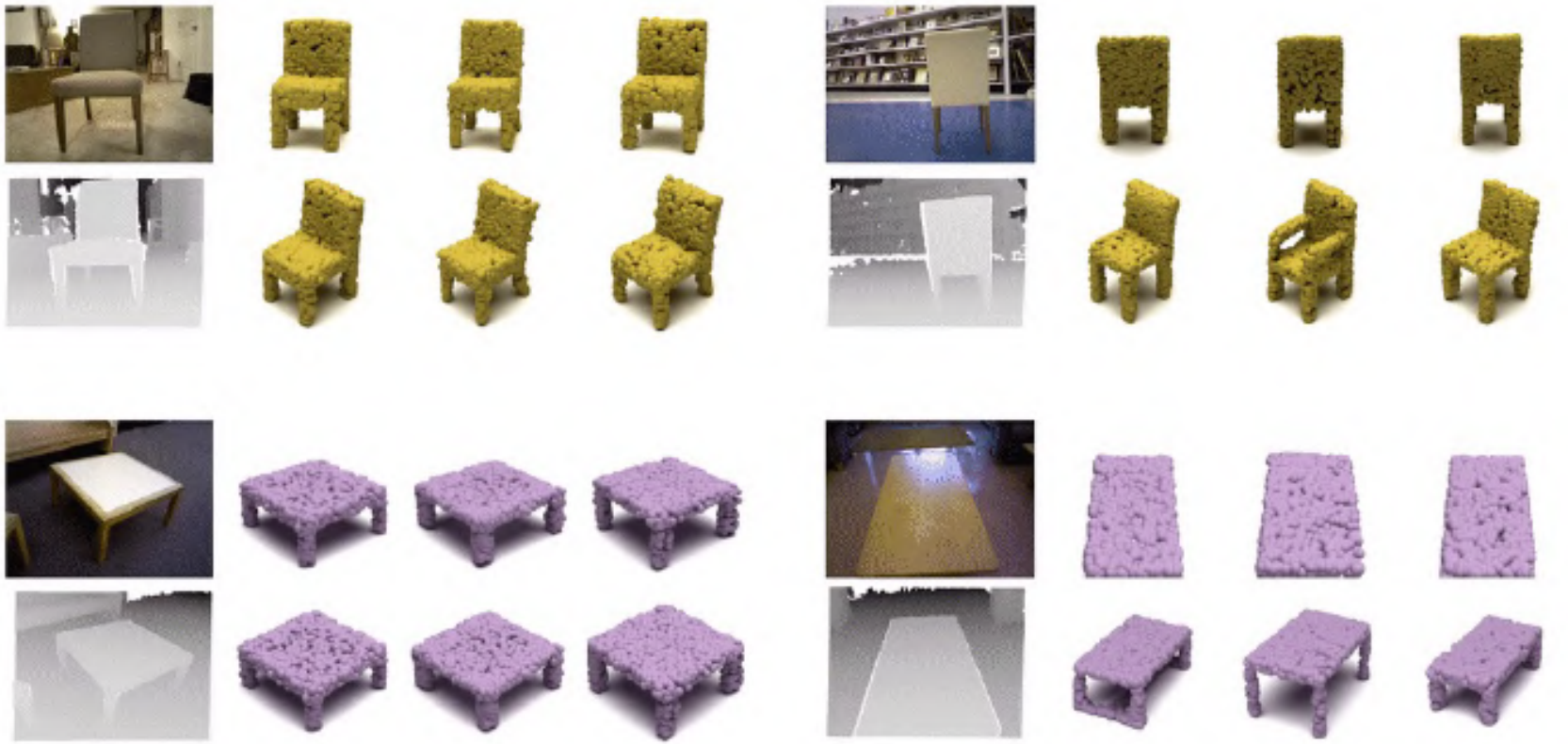


Figure 40: Natural Image Super Resolution $64 \times 64 \longrightarrow 256 \times 256$

[1] Image Super-Resolution via Iterative Refinement

3D Shape Generation



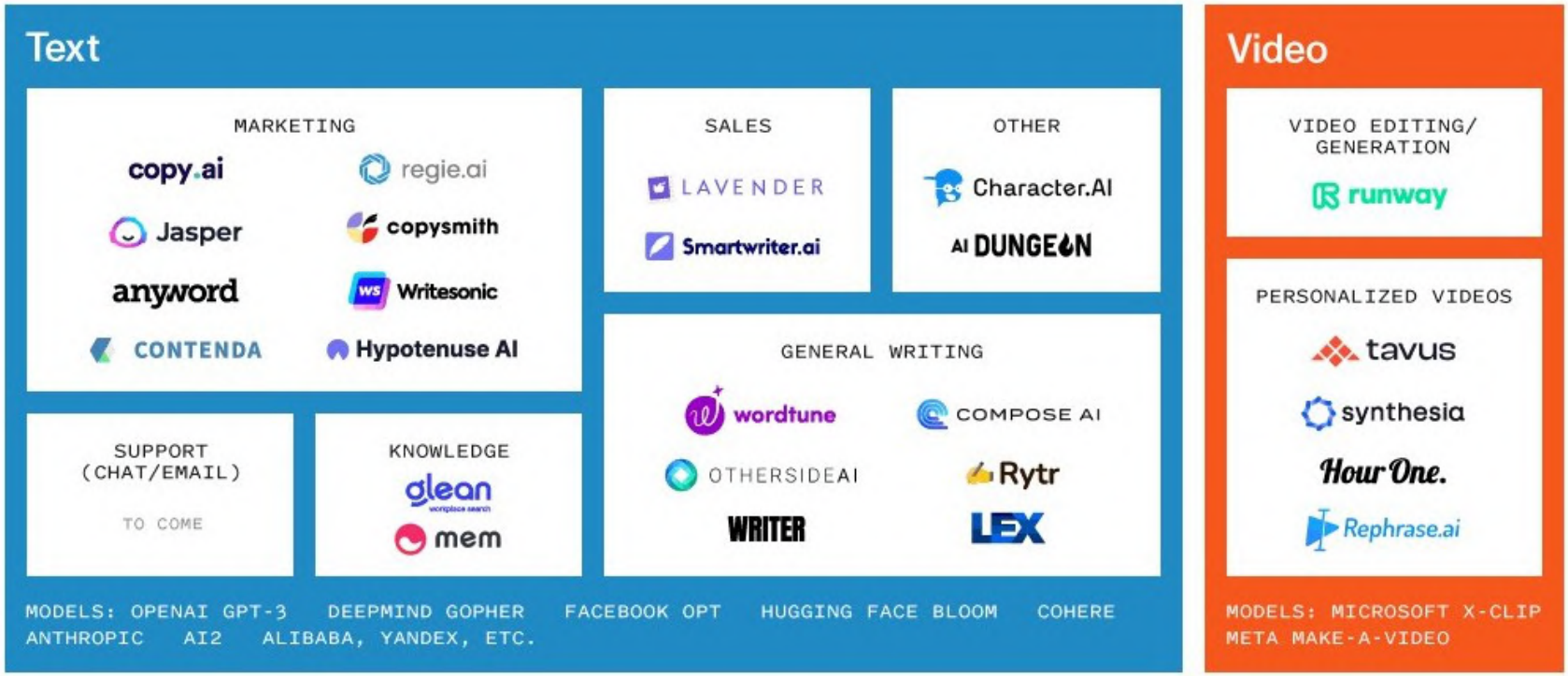
[1] 3D Shape Generation and Completion through Point-Voxel Diffusion

Try It Yourself!

Text to Image generation with stable diffusion.
https://huggingface.co/spaces/stabilityai/stable-diffusion

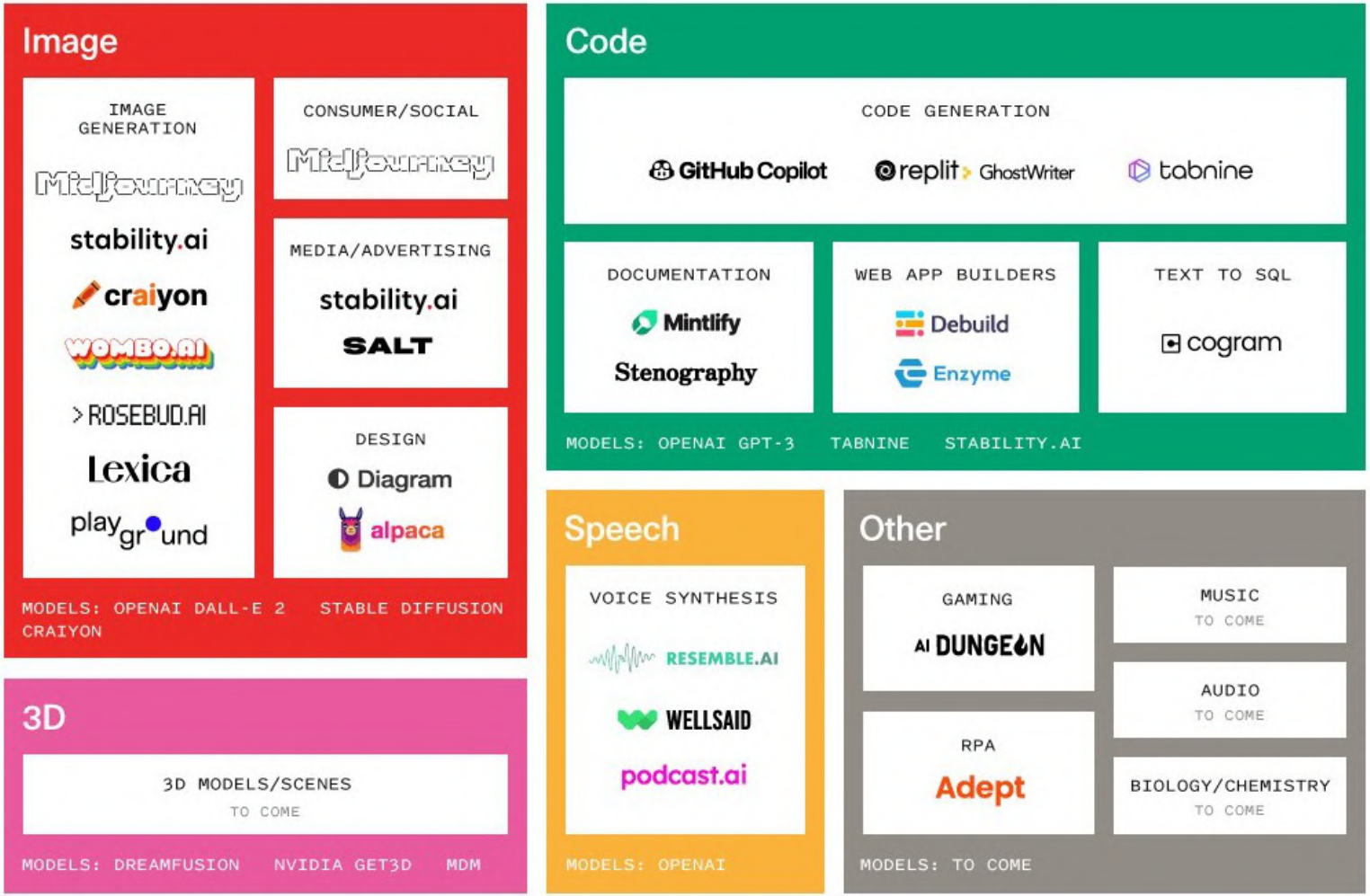
[1] High-Resolution Image Synthesis with Latent Diffusion Models

The Generative AI Landscape



[1] Sonya Huang (@sonyatweetybird)

The Generative AI Landscape



[1] Sonya Huang (@sonyatweetybird)

References

Reference Slides

- Fei-Fei Li "Generative Deep Learning" CS231
- Hao Dong "Deep Generative Models"
- Hung-Yi Lee "Machine Learning"
- Francois Fleuret "Deep Learning" EE559
- Murtaza Taj "Deep Learning" CS437
- Kreis, Gao & Vahdat CVPR 2022 Tutorial
- Rogge & Rasul The Annotated Diffusion Model