ROBOT INTELLIGENCE LAB

Introduction to Deep Learning

Deep Generative Models

Sungjoon Choi, Korea University

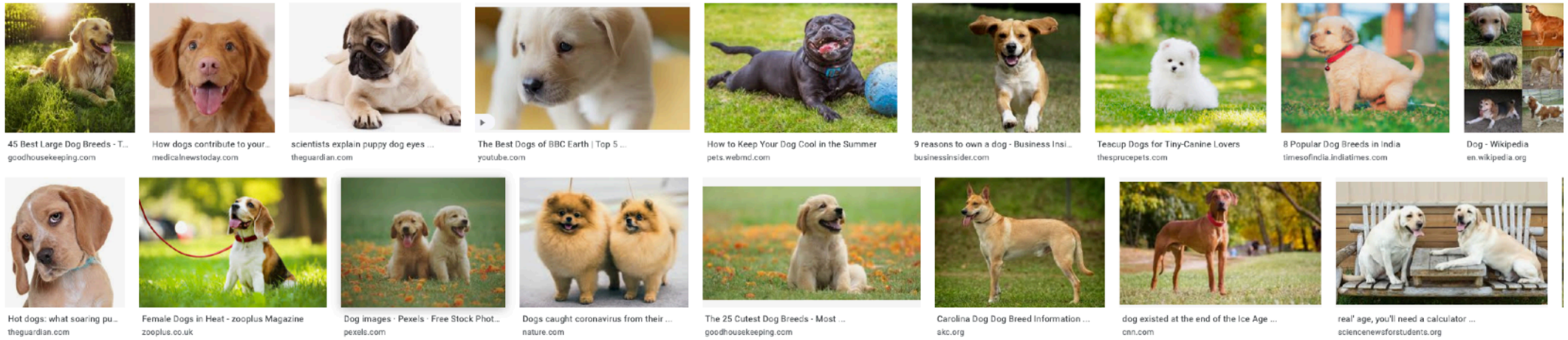
kakaobrain

Generative Model

ROBOT INTELLIGENCE LAB

What does it mean to learn a generative model?

Generative Model



Google Search: Dog

- Suppose that we are given images of dogs.
- We aim to learn a probability distribution $p(x)$ such that
  - we can sample $\tilde{x} \sim p(x)$ where $\tilde{x}$ looks like a dog (aka **generation**)
  - $p(x)$ should be high if x looks like a dog and low otherwise (aka **density estimation**)
- Then, how can we represent $p(x)$?

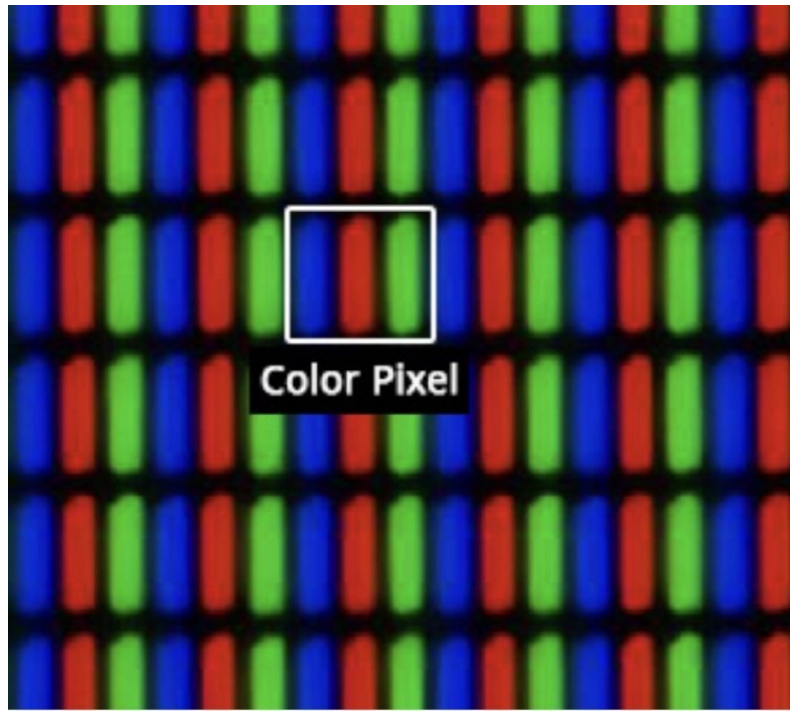
Basic Discrete Distributions

ROBOT INTELLIGENCE LAB

- Bernoulli distribution: (biased) coin flip
  - $D = \{\text{Heads}, \text{Tails}\}$
  - Specify $P(X = \text{Heads}) = p$. Then $P(X = \text{Tails}) = 1 - p$.
    - The number of parameters is one.
  - Denote $X \sim \text{Ber}(p)$
- Categorical distribution: (biased) m-sided dice
  - $D = \{1, \ldots, m\}$
  - Specify $P(Y = i) = p_i$ such that $\sum_{i=1}^{m} = 1$.
    - The number of parameters is $m - 1$.
  - Denote $Y \sim \text{Cat}(p_1, \ldots, p_m)$

Example

- Let's model **a single pixel** of an RGB image.
  - $(r, g, b) \sim P(R, G, B)$
  - Number of possible cases?
    - $256 \times 256 \times 256$
      - Each color channel is independent and has 256 possible cases.
  - How many parameters do we need?
    - $256 \times 256 \times 256 - 1$
      - We can treat a single pixel as a huge dice with $256 \times 256 \times 256$ sides.



https://www.quora.com/Why-cant-we-color-a-pixel-partially

ROBOT INTELLIGENCE LAB

(Conditional) Independence

Example

- Suppose we have a set of images with n binary pixels (binary images)
  - Number of possible cases?
    - $2 \times 2 \times \cdots \times 2 = 2^n$
  - How many parameters do we need?
    - $2^n - 1$
    - If we have an image with $28 \times 28 = 784$ pixels, the number of parameters is approximately $2^{784} \approx 10^{236}$ (where the number of total atoms in the universe is approximately 10^{82}).

Structure Through Independence

- What if each pixel $X_1, \ldots, X_n$ is independent?
  - Then, $P(X_1, \ldots X_n) = P(X_1)P(X_2)\cdots P(X_n)$.
  - Number of possible cases?
    - $2 \times 2 \times \cdots \times 2 = 2^n$
    - It is, of course, the same as before.
  - How many parameters do we need?
    - n
  - The total 2^n entries can be described by just n numbers. What does it mean?

Probability Rules

ROBOT INTELLIGENCE LAB

- Three important rules in probability theory
  - Chain rule

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$$

  - Bayes' theorem

$$p(x \mid y) = \frac{p(x, y)}{p(y)} = \frac{p(y \mid x)p(x)}{p(y)}$$

  - Conditional independence

$$\text{If } x \perp y \mid z, \text{ then } p(x \mid y, z) = p(x \mid z)$$

Chain Rule

ROBOT INTELLIGENCE LAB

- Using the chain rule,

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_1, X_2) \cdots P(X_n \mid X_1, \cdots, X_{n-1})$$

  where $X_i \sim \text{Ber}(p)$.

- How many parameters do we need?
  - $P(X_1)$: 1 parameter
  - $P(X_2 \mid X_1)$: 2 parameters (one per $P(X_2 \mid X_1 = 0)$ and one per $P(X_2 \mid X_1 = 1)$)
  - $P(X_3 \mid X_1, X_2)$: 4 parameters
  - Hence, the total number of parameters becomes:

$$1 + 2 + 2^2 + \cdots + 2^{n-1} = 2^n - 1$$

  - This was the same as before. Why?

Conditional Independence

ROBOT INTELLIGENCE LAB

- Now, suppose that $X_{i+1} \perp X_1, \ldots, X_{i-1} \mid X_i$ (aka a Markov assumption), then

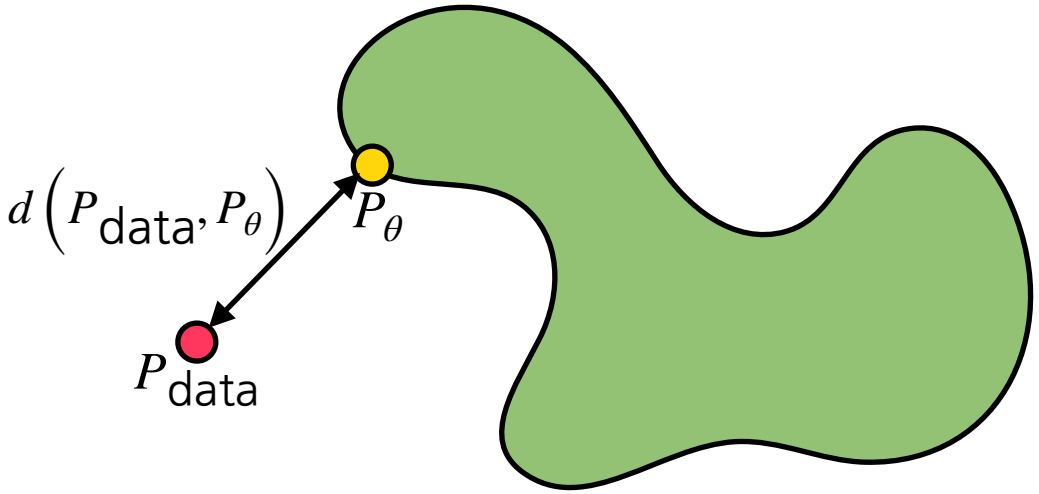
$$p(X_1, \ldots, X_n) = p(X_1)p(X_2 \mid X_1)p(X_3 \mid X_2) \cdots p(X_n \mid X_{n-1})$$

- How many parameters do we need?
  - $P(X_1)$: 1 parameter
  - $P(X_2 \mid X_1)$: 2 parameters
  - $P(X_3 \mid X_2)$: 2 parameters
  - Hence, the total number of parameters becomes $2n - 1$.
- By simply utilizing the Markov assumption, we get an exponential reduction in the number of parameters! $((2^n - 1) \rightarrow (2n - 1))$
- **Autoregressive models** leverage this conditional independence.

ROBOT INTELLIGENCE LAB

Maximum Likelihood Learning

Learning Generative Models



- Given a training set of examples, we can cast the generative model learning process as finding the **best approximating density** model from the **model family** (a set of all possible solutions).
- One important question to address is

"How can we evaluate the **goodness** of the approximation?"

Learning Generative Models

ROBOT INTELLIGENCE LAB

- One widely used metric for computing the similarity between distributions is KL divergence:

$$D_{KL}(P_{\text{data}}(x)\|P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}}\left[\log \frac{P_{\text{data}}(x)}{P_\theta(x)}\right] = \sum_x P_{\text{data}}(x)\log \frac{P_{\text{data}}(x)}{P_\theta(x)}$$

- We can simplify this with

$$D_{KL}(P_{\text{data}}(x)\|P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}}\left[\log \frac{P_{\text{data}}(x)}{P_\theta(x)}\right] = \mathbb{E}_{x\sim P_{\text{data}}}\left[\log P_{\text{data}}(x)\right] - \mathbb{E}_{x\sim P_{\text{data}}}\left[\log P_\theta(x)\right]$$

- When it comes to optimizing P_θ, the first term does not depend on θ, and minimizing the KL divergence becomes equivalent to **maximizing the expected log-likelihood**:

$$\arg\min_\theta D_{KL}(P_{\text{data}}(x)\|P_\theta(x)) = \arg\max_\theta \mathbb{E}_{x\sim P_{\text{data}}}\left[\log P_\theta(x)\right]$$

Maximum Likelihood Learning

ROBOT INTELLIGENCE LAB

- The maximum likelihood learning of a generative model:

$$\arg\max_{\theta} \mathbb{E}_{x \sim P_{\text{data}}} \left[\log P_\theta(x) \right]$$

- One can approximate the expected log-likelihood with the empirical log-likelihood:

$$\mathbb{E}_{x \sim P_{\text{data}}} \left[\log P_\theta(x) \right] = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log P_\theta(x)$$

- The **maximum likelihood learning** is then

$$\arg\max_{\theta} \mathbb{E}_{x \sim P_{\text{data}}} \left[\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log P_\theta(x) \right]$$

Empirical Risk Minimization

ROBOT INTELLIGENCE LAB

- For the maximum likelihood learning, **empirical risk minimization (ERM)** is often used.
- However, **ERM** often suffers from the overfitting.
  - Extreme case: the model remembers all training data

$$P(x) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \delta(x, x_i)$$

- To achieve a better generalization performance, we typically restrict the **hypothesis space** (model family) of distributions that we search over.
  - For example, $P_\theta(x) = \mathcal{N}(x; \mu, \Sigma)$ where $\theta = \{\mu, \Sigma\}$.
- However, it may also deteriorate the performance of the generative model.

Maximum Likelihood Learning

ROBOT INTELLIGENCE LAB

- Usually, the maximum likelihood learning is prone to under-fitting as we often use simple parametric distributions such as spherical Gaussians.

- What about other ways of measuring the similarity?
  - **KL divergence**: it leads to maximum likelihood learning (variational autoencoder or diffusion models).
  - **Jensen-Shannon divergence**: it leads to adversarial learning (generative adversarial networks).
  - **Wasserstein distance**: it leads to Wasserstein autoencoder (adversarial autoencoder).

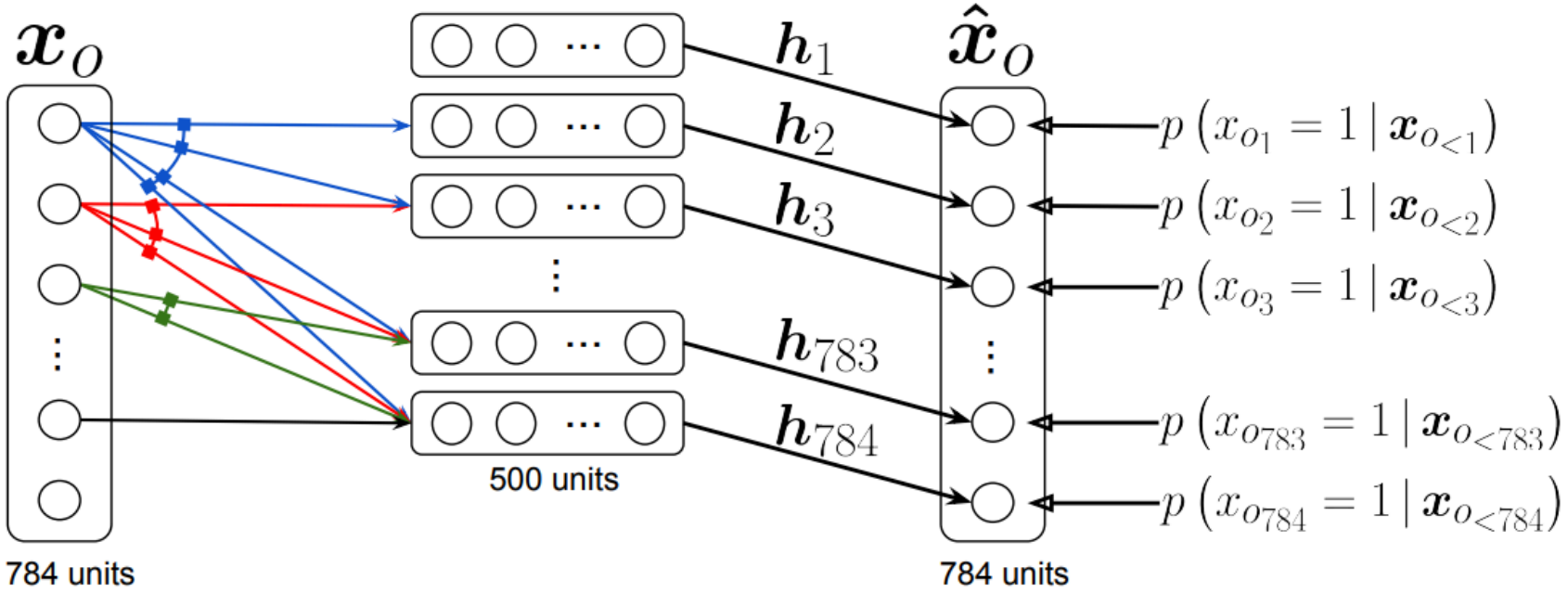
Autoregressive Models

"The Neural Autoregressive Distribution Estimator," 2011

Autoregressive Model

- Suppose we have 28×28 binary images.
- Our goal is to learn $P(X) = P(X_1, \ldots, P_{784})$ over $X \in \{0,1\}^{784}$.
- Then, how can we parametrize $P(X)$?
  - Let's use the chain rule to factor the joint distribution into conditionals.
  - In other words,
    - $P(X_{1:784}) = P(X_1)P(X_2|X_1)P(X_3|X_2)\cdots P(X_{784}|X_{783})$.
    - This is called an autoregressive model.
    - Note that we need an ordering (e.g., raster scan order) of random variables.

NADE: The Neural Autoregressive Distribution Estimator



- The probability distribution of i-th pixel is

$$p(x_i \mid x_{1:i-1}) = \sigma(\alpha_i h_i + b_i) \text{ where } h_i = \sigma(W_{<i} x_{1:i-1} + c).$$

NADE: The Neural Autoregressive Distribution Estimator

ROBOT INTELLIGENCE LAB

- NADE (or most autoregressive models) is an explicit model that can not only sample plausible outputs but also compute the density of the given inputs.
- How can we compute the probability density of the given image?
  - Suppose we have a binary image with 784 binary pixels.
  - Then, the joint probability is computed as follows:
    - $p(x_1, \ldots, x_{784}) = p(x_1)p(x_2 \mid x_1) \cdots p(x_{784} \mid x_{783})$ where each conditional probability $p(x_i \mid x_{i-1})$ is computed independently.
    - We can easily compute the log probability:

$$\log p(x_1, \ldots, x_{784}) = \log p(x_1) + \log p(x_2 \mid x_1) + \log p(x_{784} \mid x_{783})$$

- Since we factor the joint distribution into conditionals, we may use more complex probability distributions, such as a **mixture of Gaussians**.

Summary

ROBOT INTELLIGENCE LAB

- The autoregressive models have two major strengths:
- Easy to **sample** from
  - Sample $\bar{x}_0 \sim p(x_0)$
  - Sample $\bar{x}_1 \sim p(x_1 \mid x_0 = \bar{x}_0)$
  - ⋯ and so forth (in a sequential manner, with a predefined order)
- Easy to **compute** the probability $p(x = \bar{x})$
  - Compute $p(x_0 = \bar{x}_0)$
  - Compute $p(x_1 = \bar{x}_1 \mid x_0 = \bar{x}_0)$
  - ⋯ and so forth (sum their logarithms)
  - Ideally, we can compute all these terms in parallel (hence fast).
- Easy to be extended to **continuous** random variables:
  - Use a mixture of Gaussians.

ROBOT INTELLIGENCE LAB

Latent Variable Models

"Auto-Encoding Variational Bayes," 2013

"Adversarial Autoencoders," 2015

"Wasserstein Auto-Encoders," 2017

Question

ROBOT INTELLIGENCE LAB

Is an **auto-encoder** a generative model?

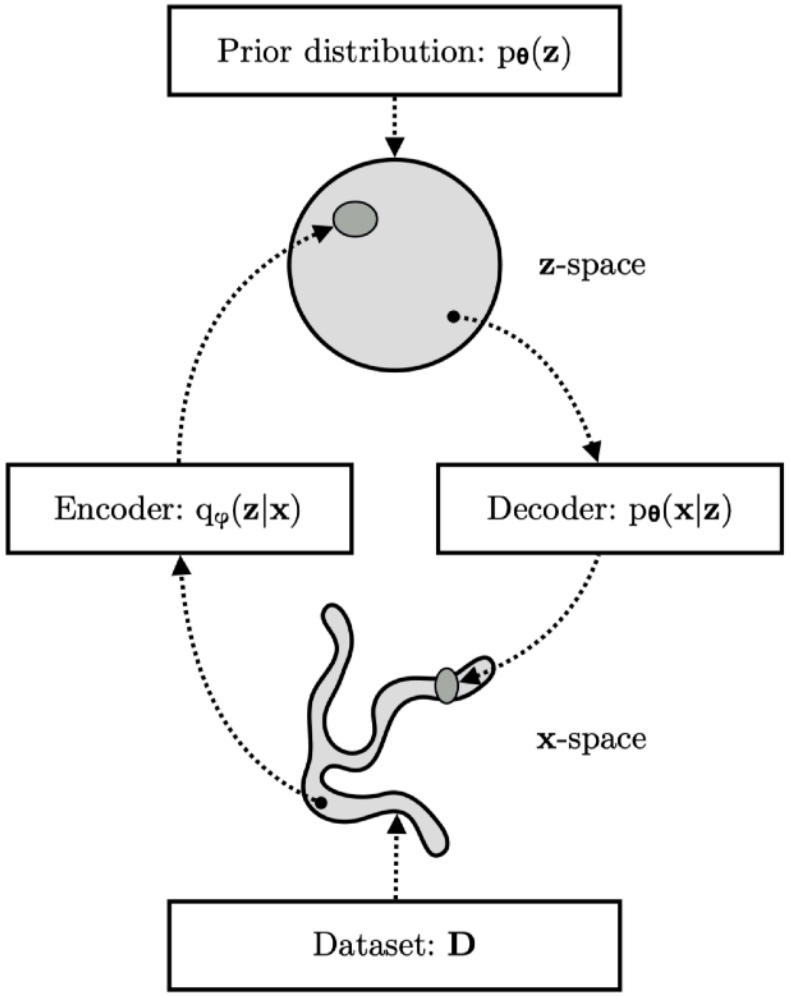
Variational Auto-Encoder

ROBOT INTELLIGENCE LAB

- The objective is simple.

$$\text{Maximize } p_\theta(\mathbf{x})$$

Variational Auto-Encoder



D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

Variational Auto-Encoder

ROBOT INTELLIGENCE LAB

$$
\begin{aligned}
\underbrace{\log p_\theta(x)}_{\text{Maximum Likelihood Learning} \uparrow} &= \int_z \underbrace{q_\phi(z|x)}_{\text{For any } q_\phi} \log p_\theta(x) dz \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x)p_\theta(z|x)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \\
&= \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{\text{ELBO} \uparrow} + \underbrace{D_{KL}\left(q_\phi(z|x)\|p_\theta(z|x)\right)}_{\text{Variational Gap} \downarrow}
\end{aligned}
$$

D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

Variational Auto-Encoder

ROBOT INTELLIGENCE LAB

$$\underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{\text{ELBO}\uparrow} = \int \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} q_\phi(z|x)dz$$

$$= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[p_\theta(x|z)\right]}_{\text{Reconstruction Term}} - \underbrace{D_{KL}\left(q_\phi(z|x)\|p(z)\right)}_{\text{Prior Fitting Term}}$$

D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

Variational Auto-Encoder

$$\underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{\text{ELBO}\uparrow} = \int \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} q_\phi(z|x)dz$$

$$= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[p_\theta(x|z)\right]}_{\text{Reconstruction Term}} - \underbrace{D_{KL}\left(q_\phi(z|x)\|p(z)\right)}_{\text{Prior Fitting Term}}$$

- Key Limitation
  - It is an **intractable** model (hard to evaluate likelihood).
  - The prior fitting term should be differentiable, hence it is hard to use diverse latent prior distributions.
  - In most cases, we use an isotropic Gaussian where we have a closed-form for the prior fitting term.

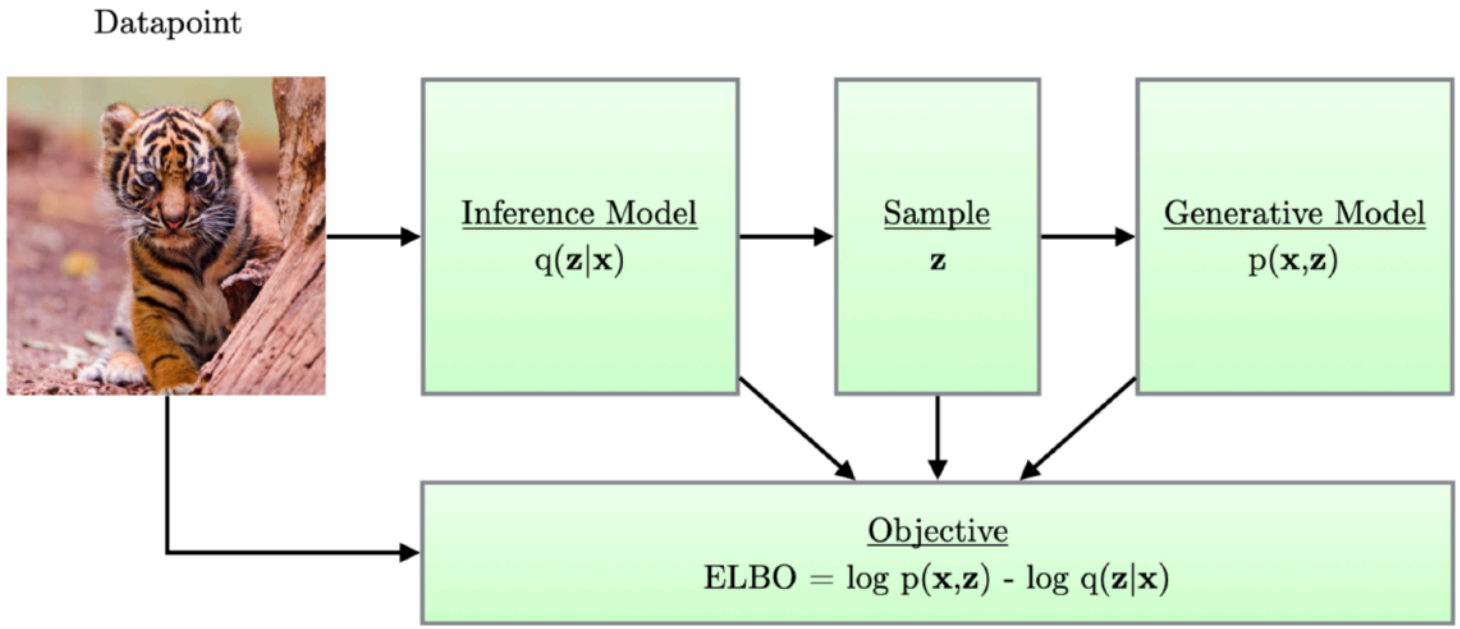
$$\underbrace{D_{KL}\left(q_\phi(z|x)\|\mathcal{N}(0,I)\right)}_{\text{Prior Fitting Term}} = \frac{1}{2}\sum_{i=1}^{D}\left(\sigma_{z_i}^2 + \mu_{z_i}^2 - \log(\sigma_{z_i}^2) - 1\right)$$

D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

Variational Auto-Encoder

ROBOT INTELLIGENCE LAB

- Key Limitation
  - It is an **intractable** model (hard to evaluate likelihood).
  - The prior fitting term should be differentiable, hence it is hard to use diverse latent prior distributions.
  - In most cases, we use an isotropic Gaussian where we have a closed-form for the prior fitting term.



D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

Re-parametrization Trick

ROBOT INTELLIGENCE LAB

- Want to compute a gradient with respect to ϕ of

$$E_{q(z;\phi)}[r(z)] = \int q(z;\phi) r(z) dz$$

  where z is continuous.

- Suppose $q(z;\phi) = \mathcal{N}(\mu, \sigma^2)$ is Gaussian with parameters $\phi = (\mu, \sigma)$. These are equivalent ways of sampling:
  - Sample $z \sim q_\phi(z)$
  - Sample $\epsilon \sim \mathcal{N}(0,1)$, then $z = \mu + \sigma\epsilon = g(\epsilon;\phi)$
- Using this equivalence, we compute the expectation in two ways:

$$E_{z\sim q(z;\phi)}[r(z)] = E_{\epsilon\sim\mathcal{N}(0,1)}[r(g(\epsilon;\phi))] = \int p(\epsilon) r(\mu + \sigma\epsilon) d\epsilon$$

$$\nabla_\phi E_{q(z;\phi)}[r(z)] = \nabla_\phi E_\epsilon[r(g(\epsilon;\phi))] = E_\epsilon[\nabla_\phi r(g(\epsilon;\phi))]$$

- It is easy to approximate the gradient by

$$E_\epsilon[\nabla_\phi r(g(\epsilon;\phi))] \approx \frac{1}{k}\sum_k \nabla_\phi r(g(\epsilon^k;\phi)) \text{ where } \epsilon^k \sim \mathcal{N}(0,1)$$

Amortization

$$\max_{\theta} l(\theta; \mathcal{D}) \geq \max_{\theta, \phi^1, \phi^2, \cdots} \sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i; \theta, \phi^i)$$

- So far, we have used a set of variational parameters (e.g., μ and σ) ϕ^i for **each** data point x^i.
  - In other words, we usually estimate $\phi^i = (\mu_i, \sigma_i^2)$ for each x^i.
- However it does not scale to large datasets.

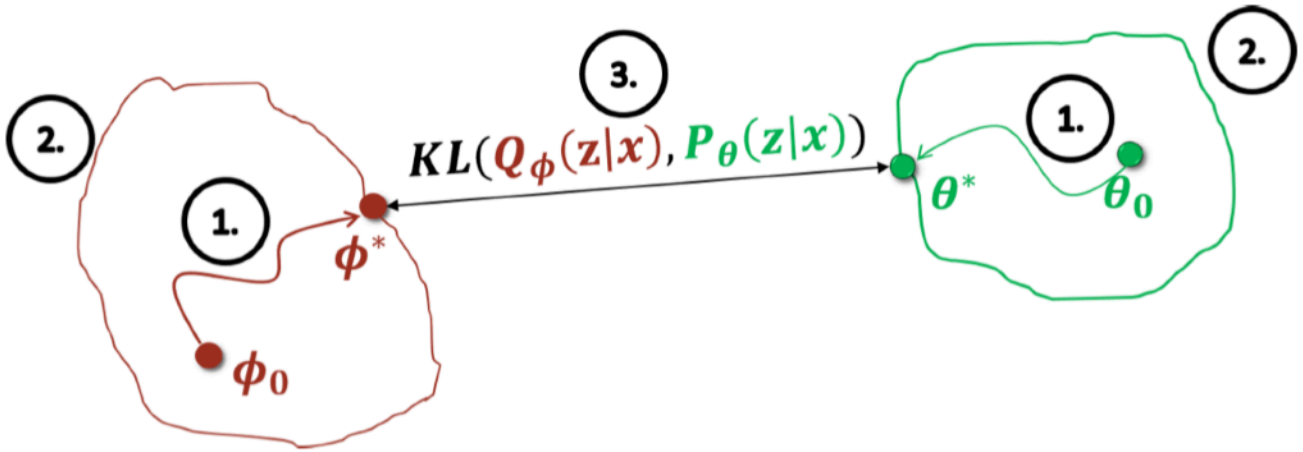
- **Amortization**: Now we learn a **single** parametric function f_λ that maps each x to a set of (good) variational parameters (i.e., $f_\lambda : x^i \mapsto \phi^i$)
  - For example, if $q(z|x^i)$ are Gaussians with different means and variances, we learn a single neural network f_λ mapping x^i to (μ^i, σ^i).
  - This is an encoder network in VAE.
- We approximate the posteriors $q(z|x^i)$ using the distribution $q_\lambda(z|x)$.

Summary of Latent Variable Models

ROBOT INTELLIGENCE LAB

1. Combine simple models to get a more flexible one (e.g., mixture of Gaussians)
2. Directed model permits ancestral sampling (efficient generation):
    1. $z \sim p(z)$
    2. $x \sim p(x \mid z; \theta)$
3. However, log-likelihood is generally intractable, hence learning is difficult.
4. Joint learning of a (decoder) model (θ) and an amortized inference component (ϕ) to achieve tractability via ELBO optimization
5. Latent representation for any x can be inferred via $q_\phi(z \mid x) = q(z; f_\lambda(x))$ (aka an encoder model).

Research Directions



1. Better optimization techniques
2. More expressive approximating (variational) families
3. Alternate loss functions

Adversarial Auto-encoder

ROBOT INTELLIGENCE LAB

- Let x be the input and z be the latent vector.
- Let $p(z)$ be the prior distribution, $q(z|x)$ be an encoding distribution, and $p(x|z)$ be the decoding distribution.
- Also let $p_d(x)$ be the data distribution, and $p(x)$ be the model distribution.
- The **aggregated posterior distribution** $q(z)$ is defined by the encoder function $q(z|x)$:

$$q(z) = \int_x q(z|x)p_d(x)dx$$

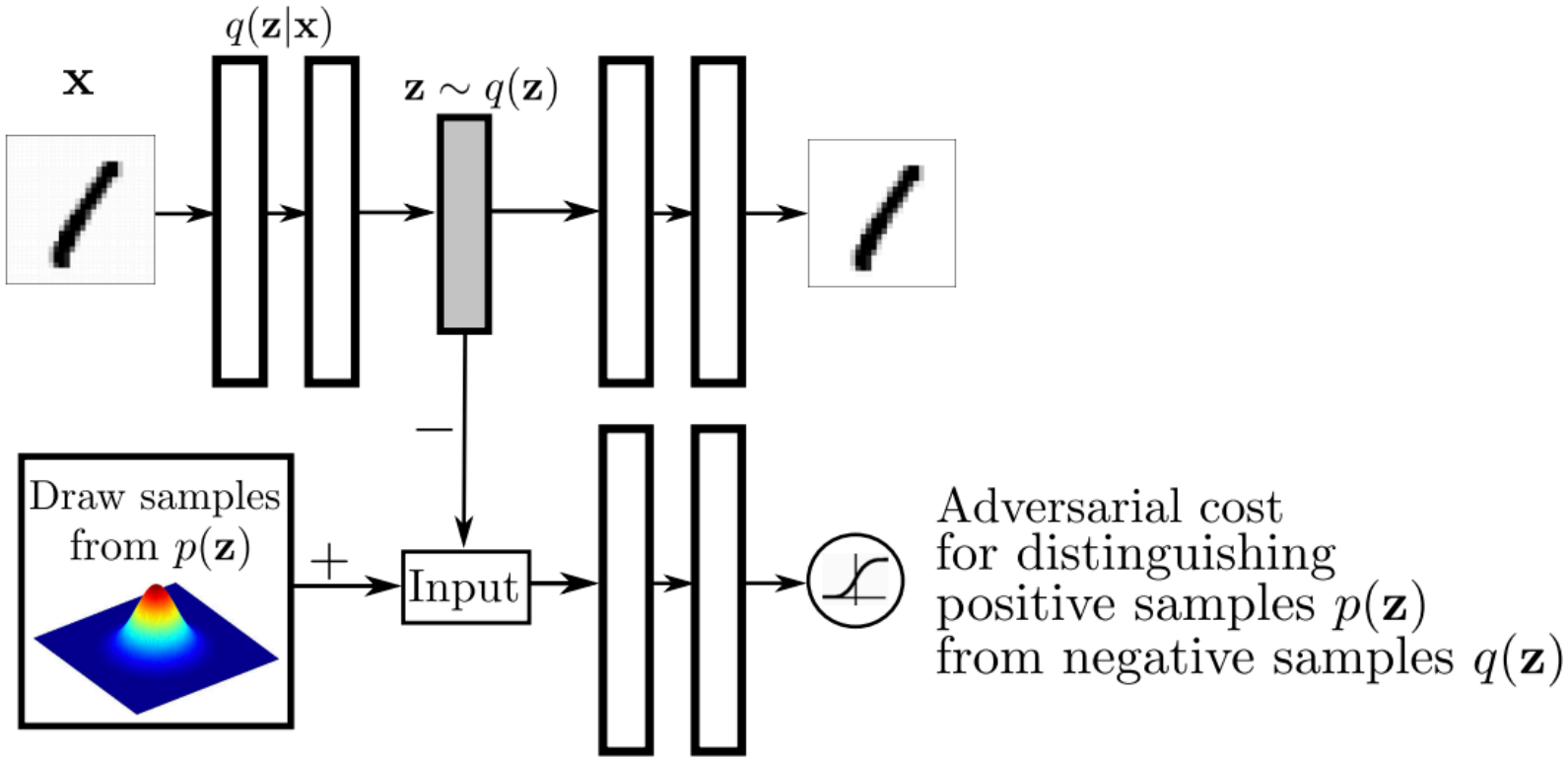
- The adversarial encoder (AAE) is an autoencoder that is **regularized by matching** the aggregated posterior, $q(z)$, to an arbitrary prior, $p(z)$.
- To achieve this, an adversarial network is attached on top of the latent vector of the autoencoder that guides $q(z)$ to match $p(z)$.

https://arxiv.org/abs/1511.05644

Adversarial Auto-encoder

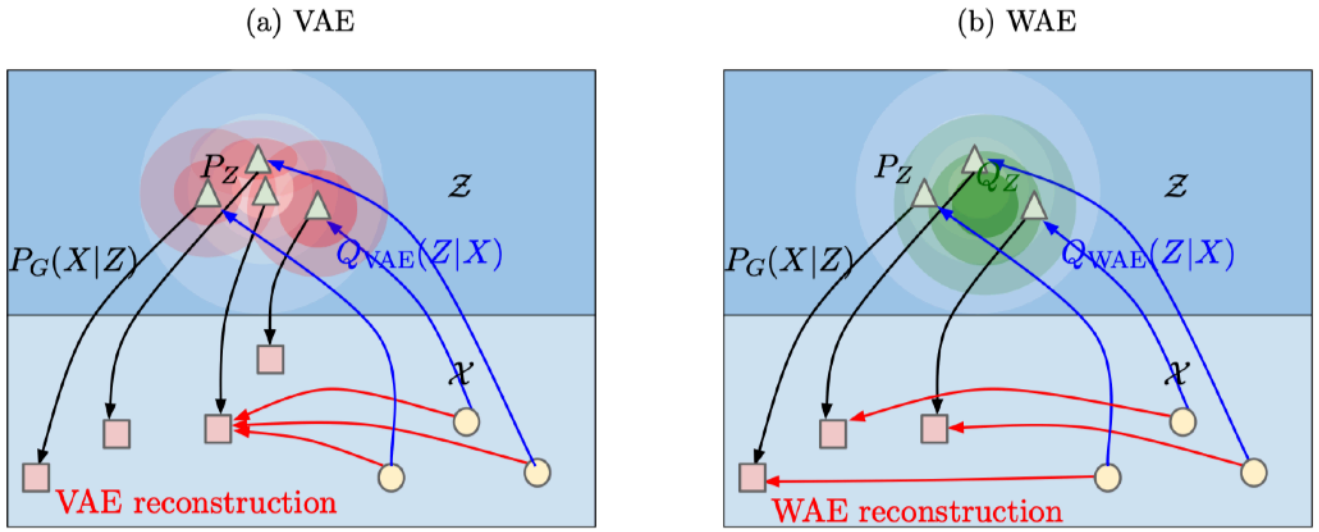
ROBOT INTELLIGENCE LAB

- AAE allows us to **use any arbitrary latent prior distributions** that we can sample.



https://arxiv.org/abs/1511.05644

Wasserstein Auto-encoders



- Both **VAE** and **WAE** minimize two terms: the reconstruction cost and the regularizer penalizing discrepancy between P_Z and distribution induced by the encoder Q.
- VAE forces $Q(Z|X=x)$ to match P_Z for all the different input examples x drawn from P_X.
- This is illustrated on picture (a), where every single red ball is forced to match P_Z depicted as the white shape. Red balls start **intersecting**, which leads to problems with reconstruction.
- In contrast, WAE forces the continuous mixture $Q_Z = \int Q(Z|X)dP_X$ to match P_Z, as depicted with the green ball in picture (b). As a result, latent vectors of different examples get a chance to **stay far way** from each other, promoting a better reconstruction.

https://arxiv.org/abs/1711.01558

Wasserstein Auto-encoders

ROBOT INTELLIGENCE LAB

- Optimal transport and its dual formulations
  - A rich class of divergences between probability distributions is induced by the optimal transport (OT) problem.

$$W_c(P_X, P_G) = \inf_{\Gamma \in \mathcal{P}(X \sim P_X, Y \sim P_G)} E_{(X,Y)\sim\Gamma}[c(X, Y)]$$

    where $c(x, y) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ is any measurable cost function and $\mathcal{P}(X \sim P_X, Y \sim P_G)$ is a set of all joint distributions of (X, Y) with marginals P_X and P_G, respectively.

  - A particularly interesting case is when $(\mathcal{X}, d)$ is a metric space and $c(x, y) = d^p(x, y)$ for $p \geq 1$. In this case, W_c is called the p-Wasserstein distance.

https://arxiv.org/abs/1711.01558

Wasserstein Auto-encoders

ROBOT INTELLIGENCE LAB

Theorem 1. *For P_G as defined above with deterministic $P_G(X|Z)$ and any function $G: \mathcal{Z} \to \mathcal{X}$*

$$\inf_{\Gamma \in \mathcal{P}(X \sim P_X, Y \sim P_G)} \mathbb{E}_{(X,Y)\sim\Gamma}\left[c(X,Y)\right] = \inf_{Q:\, Q_Z = P_Z} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\left[c(X, G(Z))\right],$$

where Q_Z is the marginal distribution of Z when $X \sim P_X$ and $Z \sim Q(Z|X)$.

$$D_{\mathrm{WAE}}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\left[c(X, G(Z))\right] + \lambda \cdot \mathcal{D}_Z(Q_Z, P_Z), \quad \text{(WAE objective)}$$

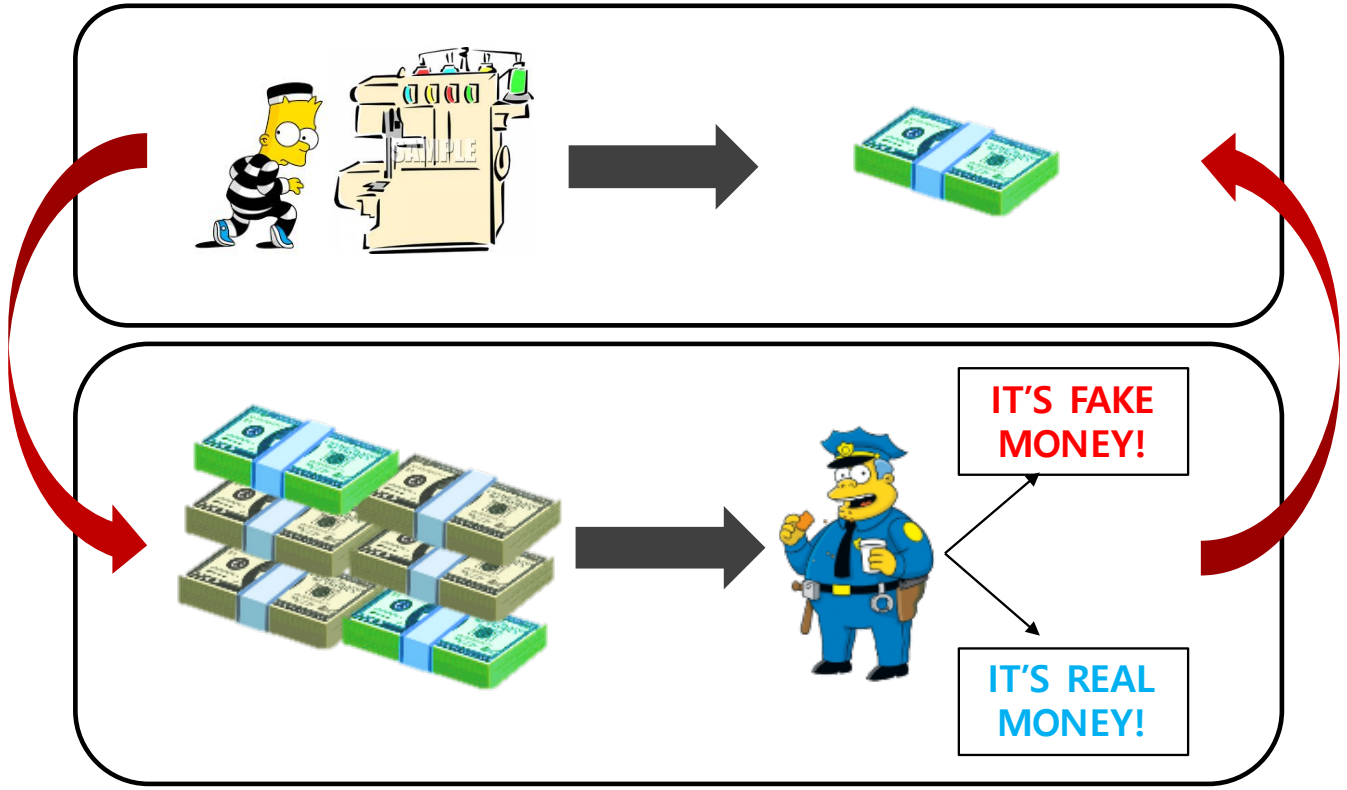
- What it means is that
  1. We can reformulate the **optimal transport problem** into the **WAE objective** using its dual formulations.
  2. The WAE objective contains a reconstruction loss and an arbitrary divergence between the aggregated posterior $Q_Z = \int Q(Z|X) dP_X$ and the prior P_Z.
  3. If we use Jensen-Shannon divergence for D_Z (i.e., $D_Z(Q_Z, P_Z) = D_{JS}(Q_Z, P_Z)$, and use the adversarial training to estimate it, it becomes the **adversarial auto-encoder**.

ROBOT INTELLIGENCE LAB

Generative Adversarial Network

"Generative Adversarial Networks," 2014

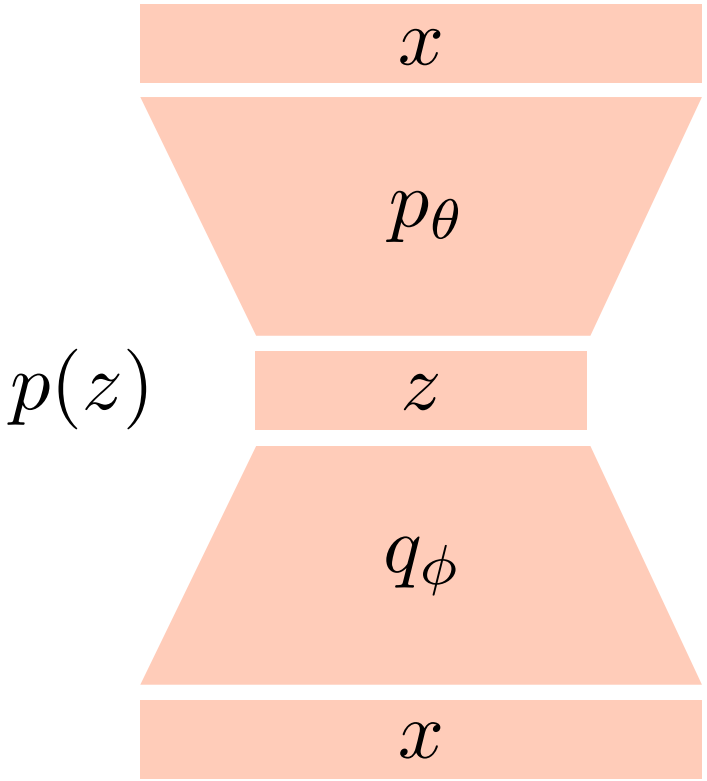
GAN



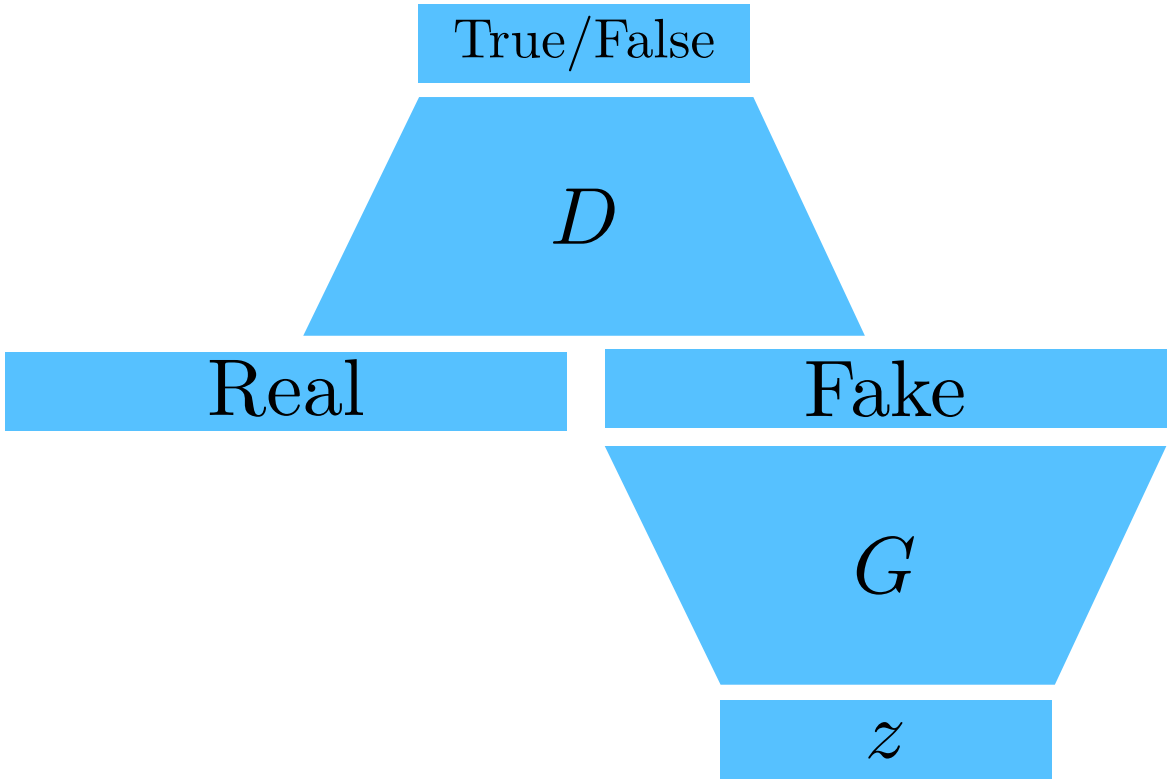
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

https://arxiv.org/abs/1406.2661

GAN vs. VAE



Variational Auto-encoder



Generative Adversarial Networks

GAN Objective

ROBOT INTELLIGENCE LAB

- GAN is a two player minimax game between **generator** and **discriminator**.
  - **Discriminator** objective

$$\max_D V(G,D) = E_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + E_{\mathbf{x}\sim p_G}[\log(1 - D(\mathbf{x}))]$$

  - The **optimal discriminator** is

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

https://arxiv.org/abs/1406.2661

GAN Objective

ROBOT INTELLIGENCE LAB

- GAN is a two player minimax game between **generator** and **discriminator**.
  - **Generator** objective

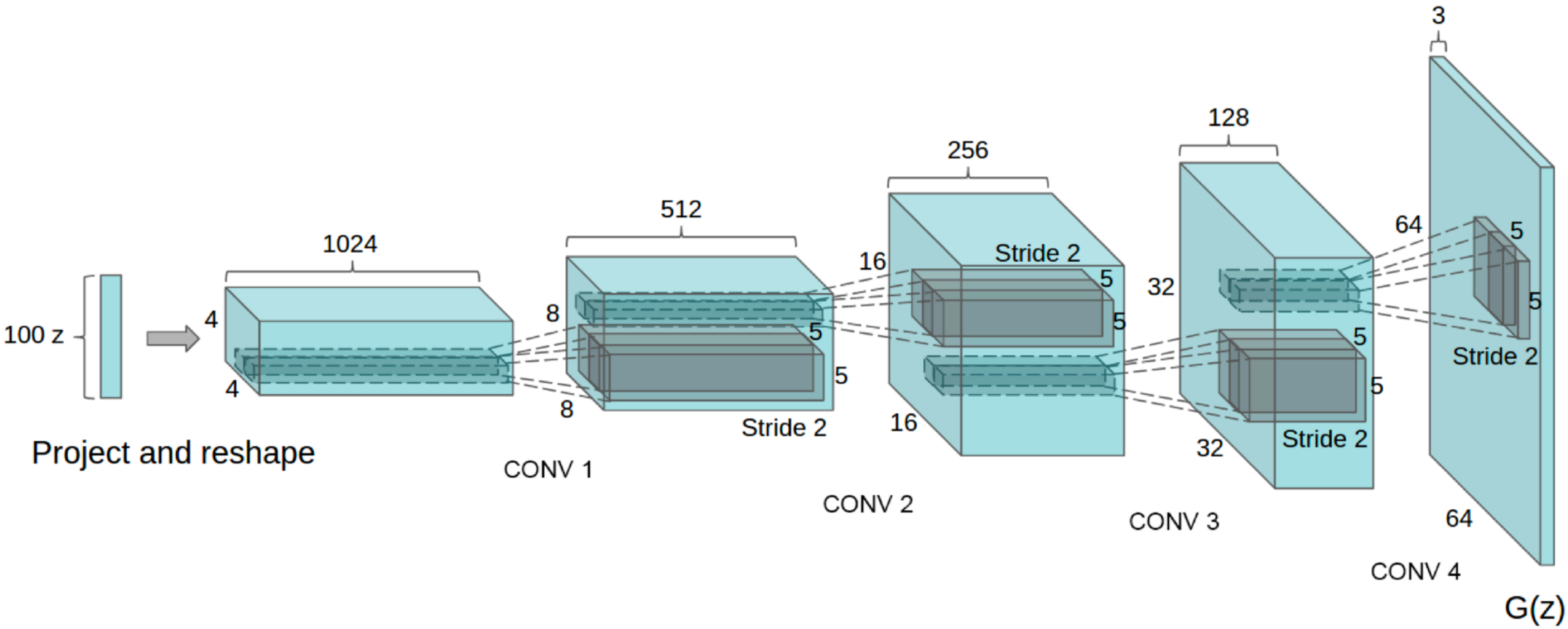
$$\min_G V(G,D) = E_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + E_{\mathbf{x}\sim p_G}[\log(1 - D(\mathbf{x}))]$$

  - Plugging in the **optimal discriminator**, we get

$$\begin{aligned} V(G, D^*_G(\mathbf{x})) &= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}\right] \\ &= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] - \log 4 \\ &= \underbrace{D_{KL}\left[p_{\text{data}}, \frac{p_{\text{data}}+p_G}{2}\right] + D_{KL}\left[p_G, \frac{p_{\text{data}}+p_G}{2}\right]}_{2\times\text{Jenson-Shannon Divergence (JSD)}} - \log 4 \\ &= 2D_{JSD}[p_{\text{data}}, p_G] - \log 4 \end{aligned}$$

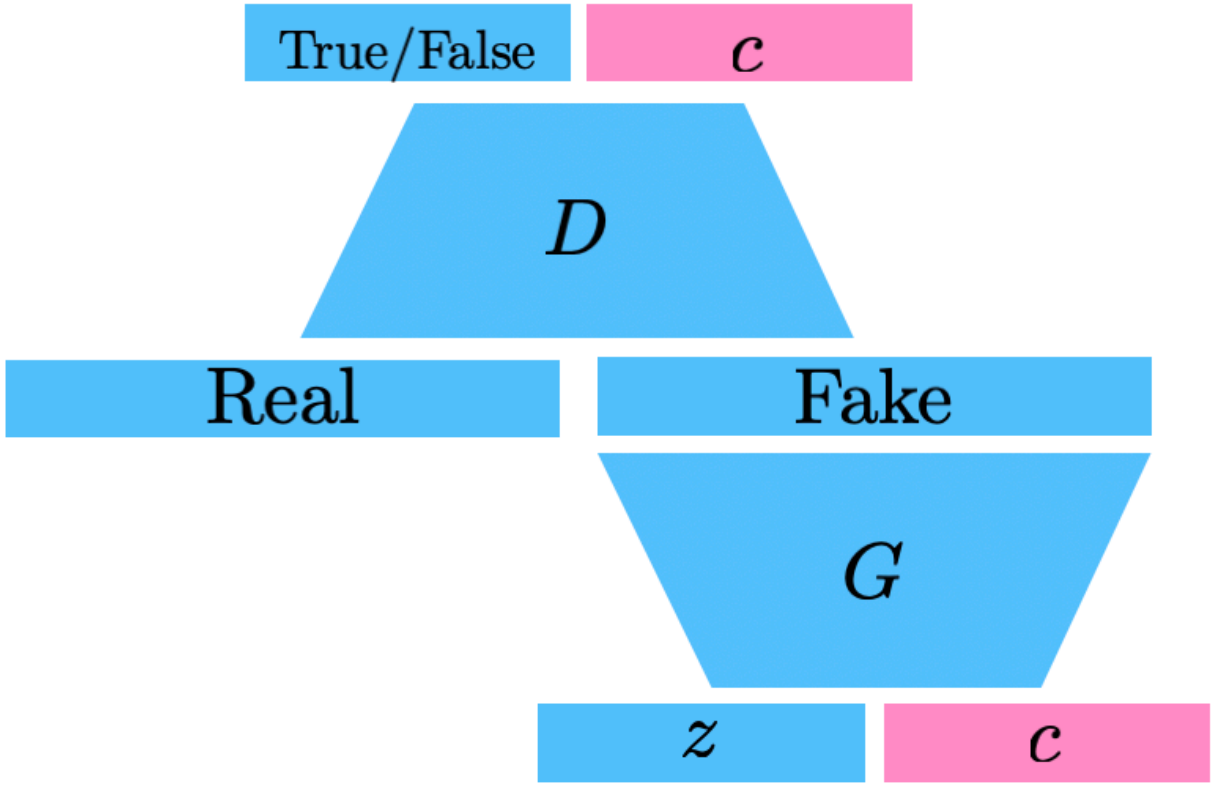
https://arxiv.org/abs/1406.2661

DCGAN



https://arxiv.org/abs/1511.06434

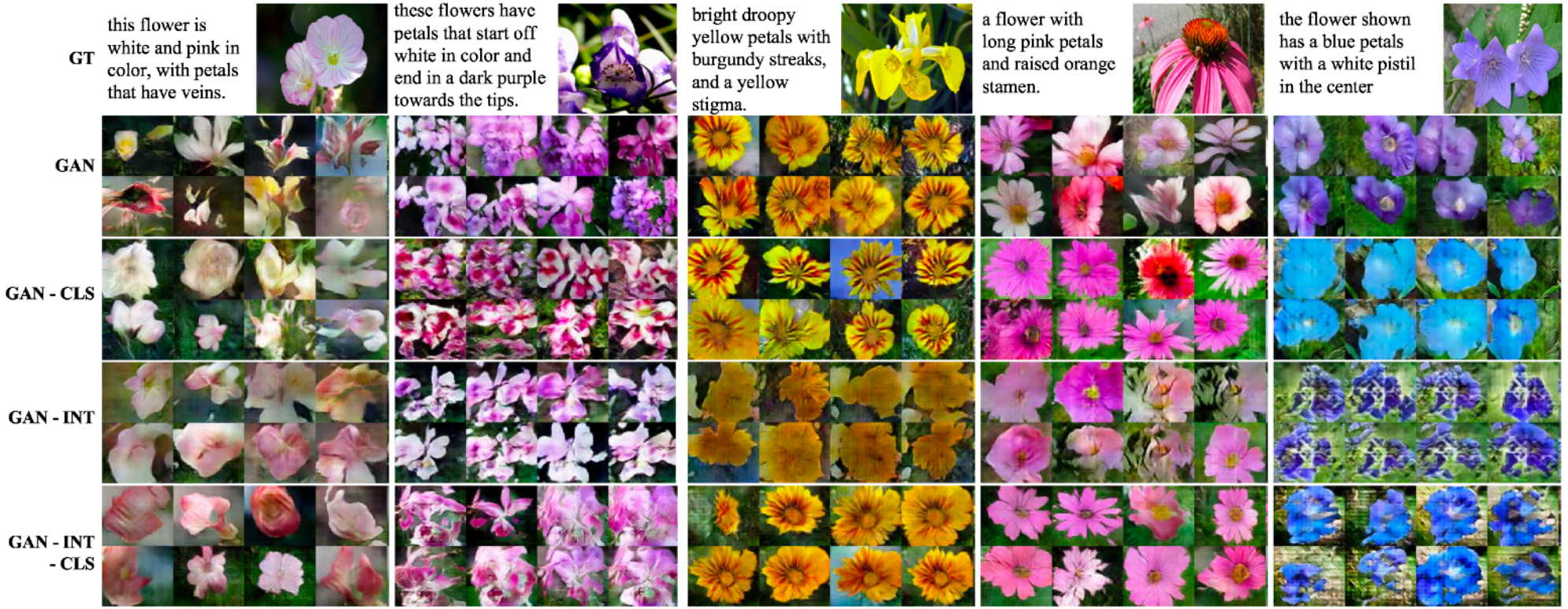
Info-GAN



https://arxiv.org/abs/1606.03657

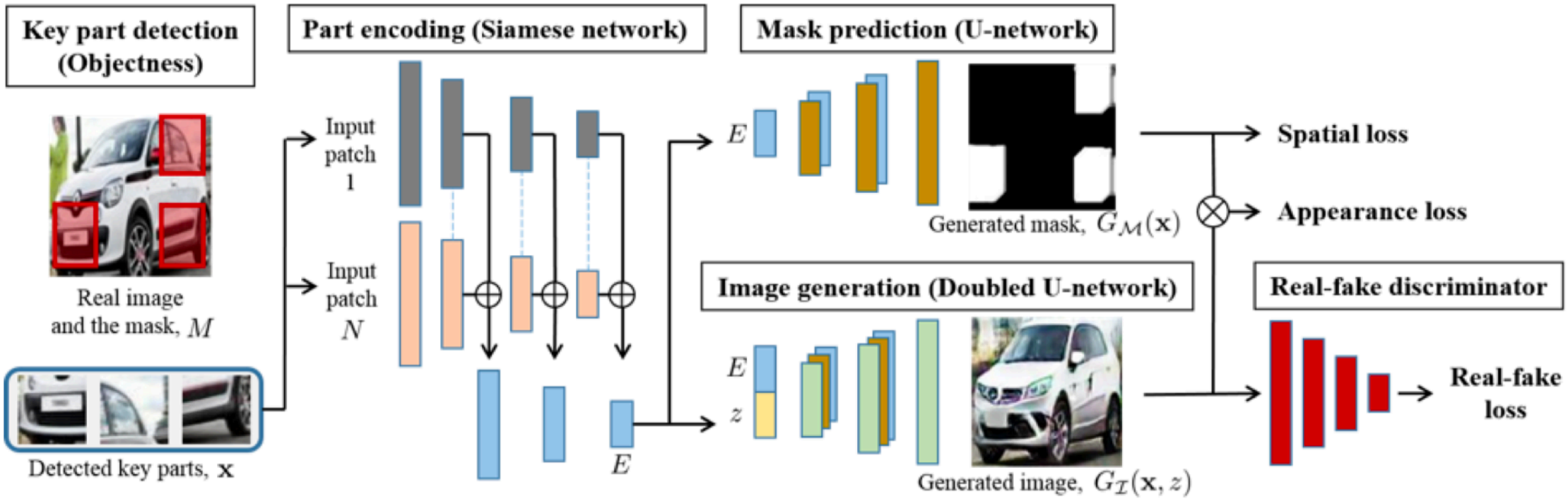
Text2Image

ROBOT INTELLIGENCE LAB



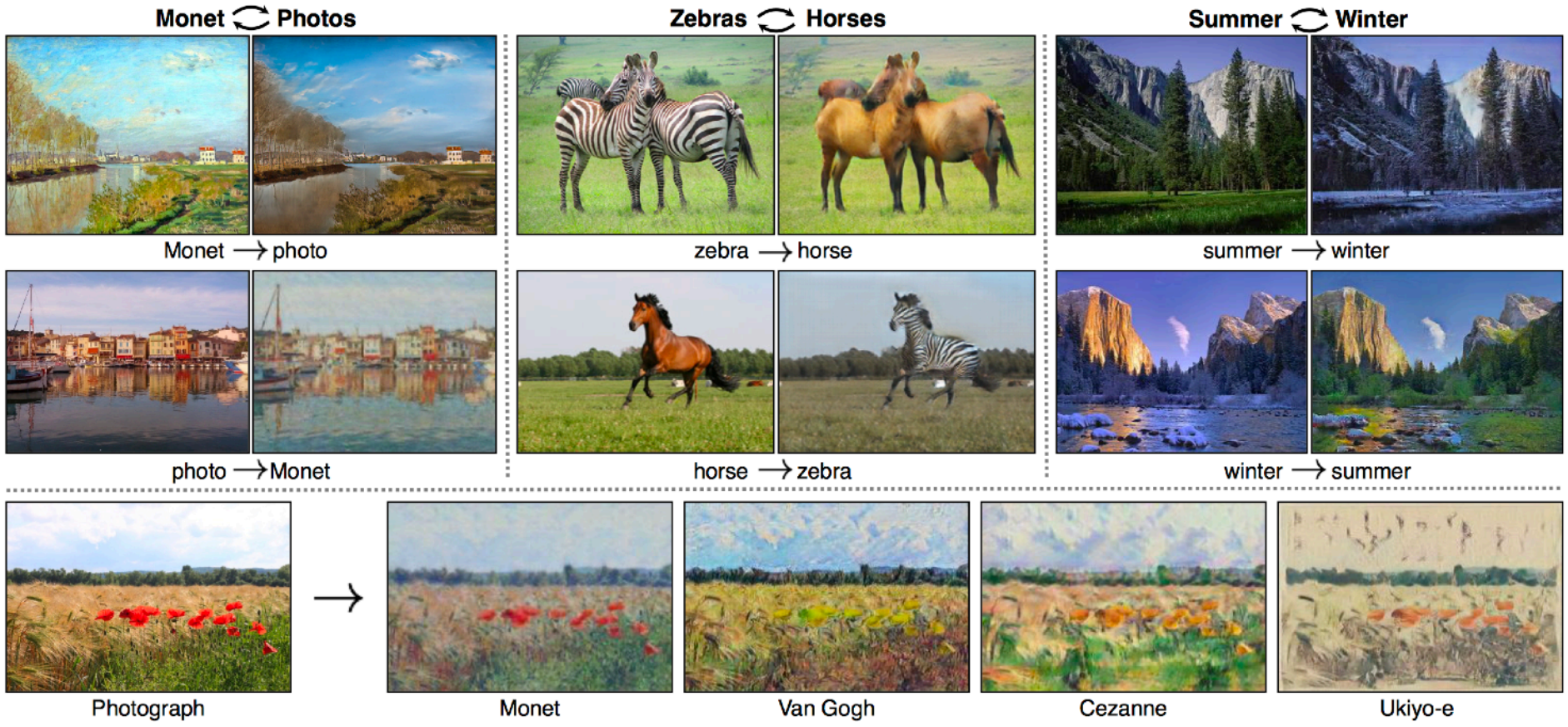
https://arxiv.org/abs/1605.05396

Puzzle-GAN



https://arxiv.org/abs/1703.10730

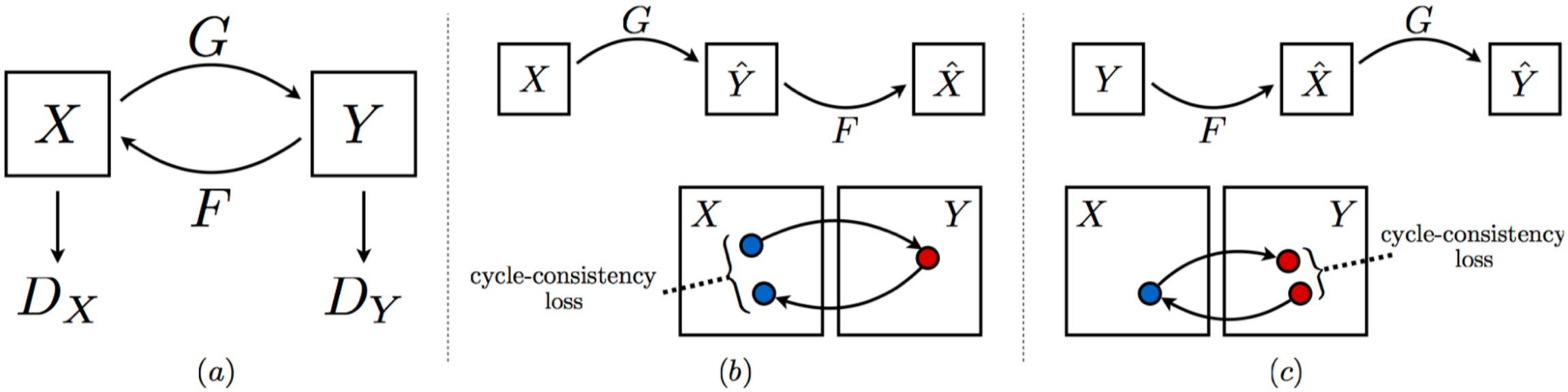
CycleGAN



https://arxiv.org/abs/1703.10593

CycleGAN

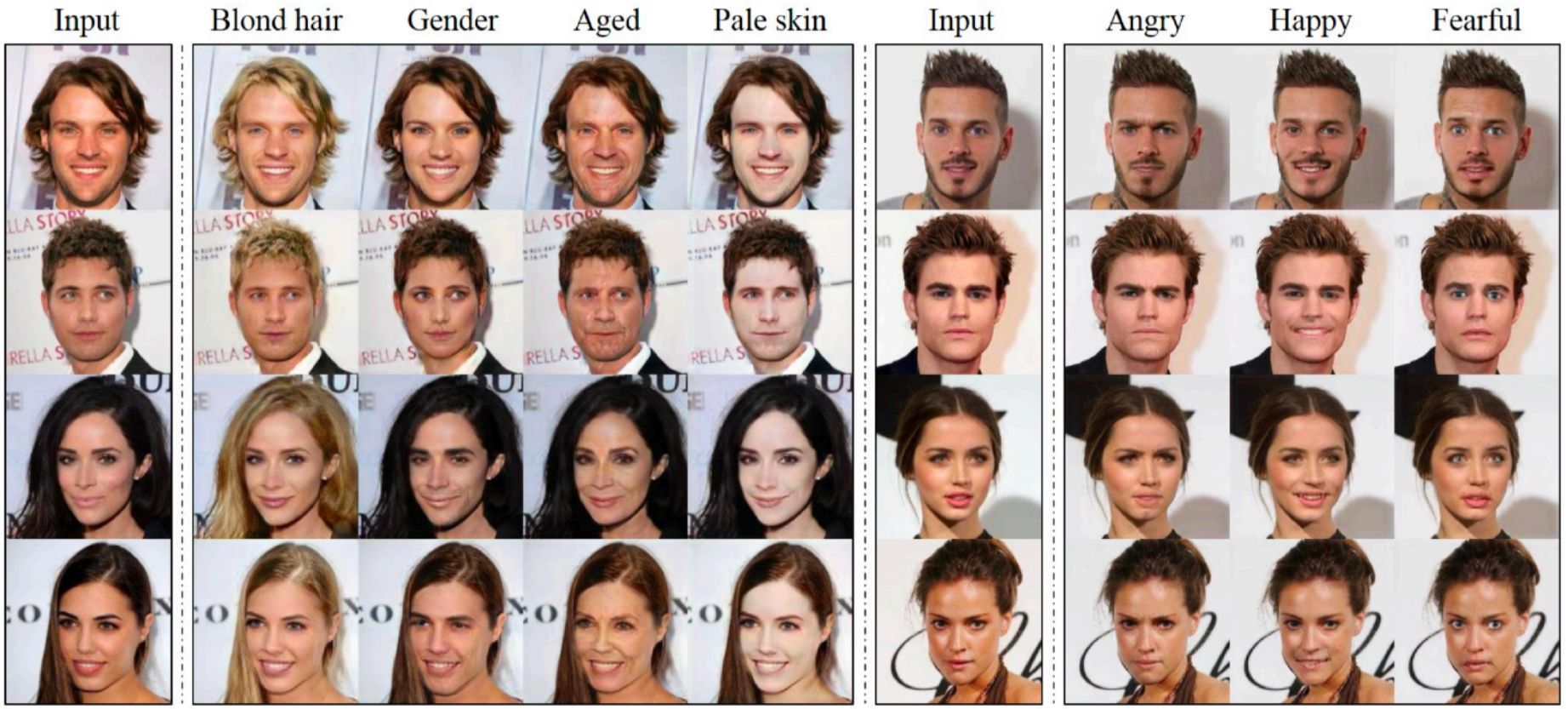
- Cycle-consistency Loss



https://arxiv.org/abs/1703.10593

Star-GAN

ROBOT INTELLIGENCE LAB

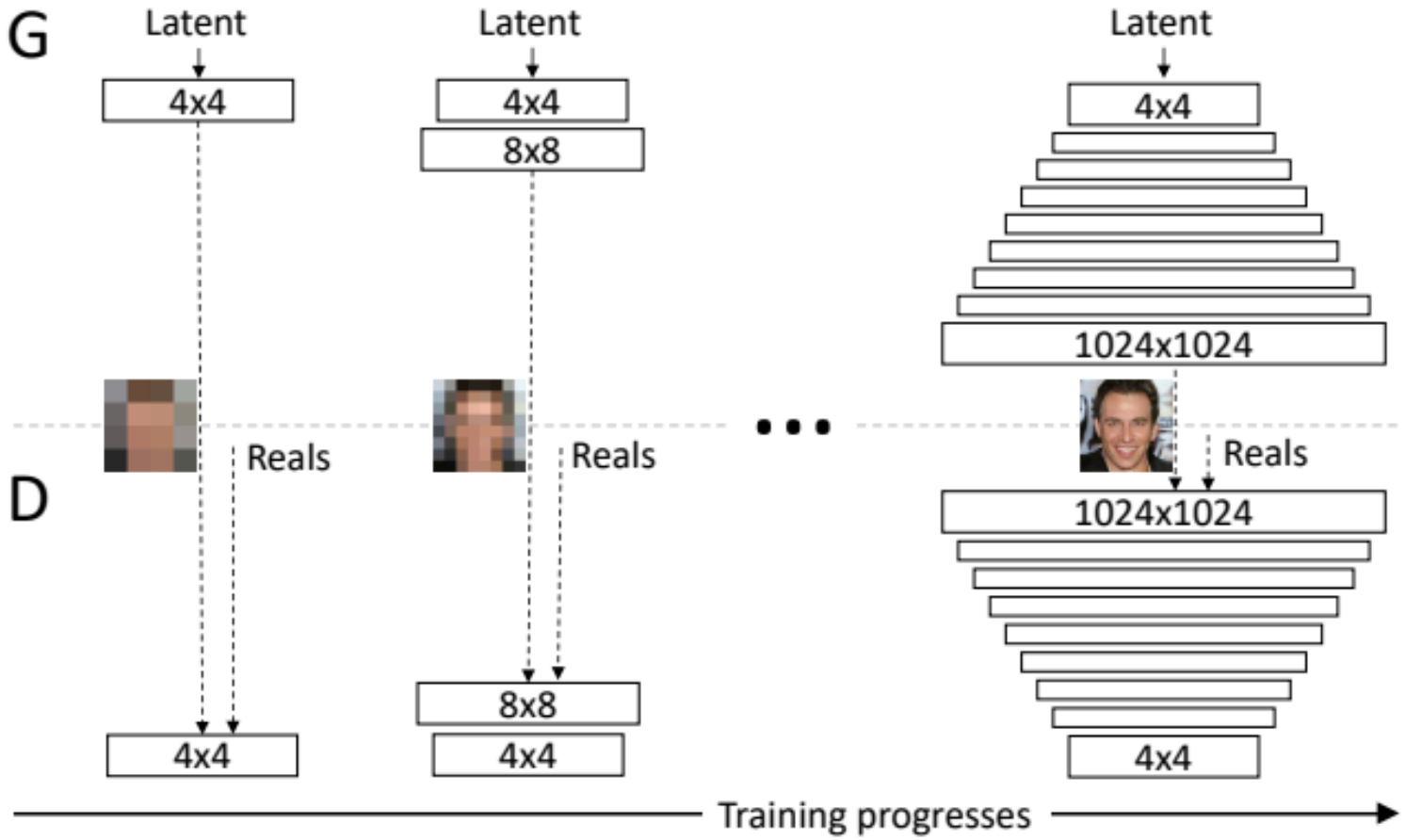


https://arxiv.org/abs/1711.09020

Progressive-GAN

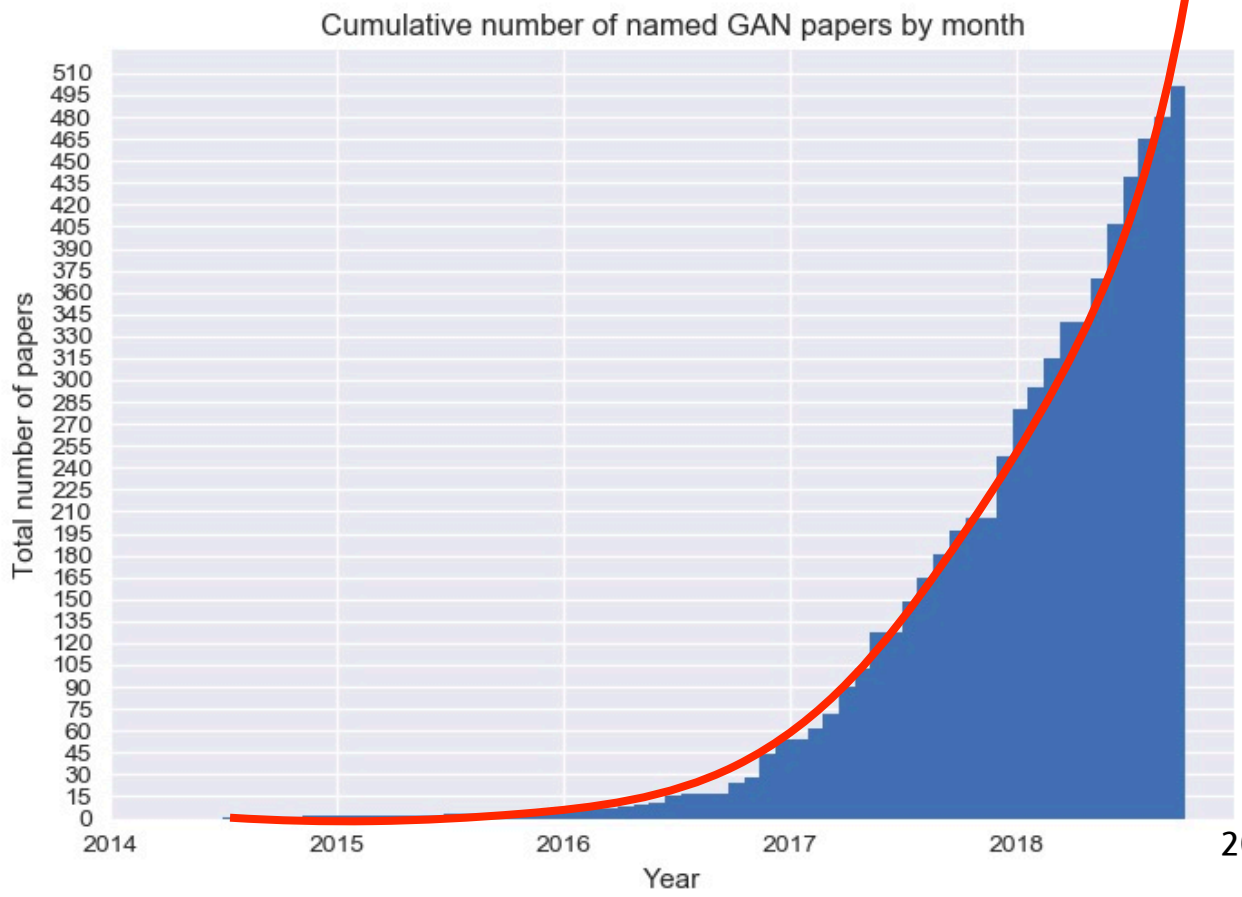
https://arxiv.org/abs/1710.10196

Progressive-GAN



https://arxiv.org/abs/1710.10196

and WAY MORE



?

https://github.com/hindupuravinash/the-gan-zoo

ROBOT INTELLIGENCE LAB

Normalizing Flow

"Variational Inference with Normalizing Flows," 2016

Normalizing Flow

ROBOT INTELLIGENCE LAB

- Limitations of existing work
  - Autoregressive models provide tractable likelihood but no direct mechanisms for learning features
  - Variational autoencoders can learn feature representations (via latent variables z) but have intractable marginal likelihoods.
- Key question
  - Can we design a latent variable model with tractable likelihoods?
- Desirable properties of any model distribution:
  - Analytic density (explicit model)
  - Easy-to-sample
- Key idea
  - Map simple distributions (easy to sample and evaluate densities) to complex distributions (learned via data) using **change of variables**.

Normalizing Flow

ROBOT INTELLIGENCE LAB

- Change of Variables Formula
  - Let Z be a uniform random variable $\mathcal{U}[0,2]$ with density p_Z. What is $p_Z(1)$?

Normalizing Flow

ROBOT INTELLIGENCE LAB

- Change of Variables Formula
  - Let Z be a uniform random variable $\mathcal{U}[0,2]$ with density p_Z. What is $p_Z(1)$?
    - The answer is $\frac{1}{2}$.
  - Let $X = 4Z$ and let p_X be its density. What is $p_X(4)$?

Normalizing Flow

ROBOT INTELLIGENCE LAB

- Change of Variables Formula
  - Let Z be a uniform random variable $\mathcal{U}[0,2]$ with density p_Z. What is $p_Z(1)$?
    - The answer is $\frac{1}{2}$.
  - Let $X = 4Z$ and let p_X be its density. What is $p_X(4)$?
    - $p_X(4) = P(X = 4) = P(4Z = 4) = P(Z = 1) = p_Z(1) = 1/2$?

Normalizing Flow

ROBOT INTELLIGENCE LAB

- Change of Variables Formula
  - Let Z be a uniform random variable $\mathcal{U}[0,2]$ with density p_Z. What is $p_Z(1)$?
    - The answer is $\frac{1}{2}$.
  - Let $X = 4Z$ and let p_X be its density. What is $p_X(4)$?
    - $p_X(4) = P(X = 4) = P(4Z = 4) = P(Z = 1) = p_Z(1) = 1/2$?
    - Unfortunately **Not**.
    - Clearly, X is uniform in $[0,8]$, so $p_X(4) = 1/8$.

Normalizing Flow

ROBOT INTELLIGENCE LAB

- **Change of variables (1D case)**
  - If $X = f(Z)$ and $f(\cdot)$ is monotone with inverse $Z = f^{-1}(X) = h(X)$, then

$$p_X(x) = p_Z(h(x))\,|\,h'(x)\,|$$

- Previous example
  - If $X = 4Z$ and $Z \sim \mathcal{U}[0,2]$, what is $p_X(4)$?
  - Note that $h(X) = X/4$ and $h'(X) = 1/4$.
  - Then, $p_X(4) = p_Z(1)h'(4) = 1/2 \times 1/4 = 1/8$.

Normalizing Flow

ROBOT INTELLIGENCE LAB

- **Generalized change of variables**
  - The mapping between Z and X, given by $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, is invertible such that $X = f(Z)$ and $Z = f^{-1}(X)$.

$$p_X(x) = p_Z\left(f^{-1}(x)\right) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

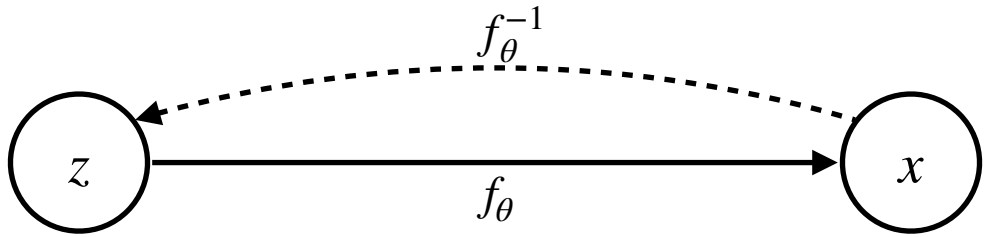
  - Note 1: x and z need to be continuous and have the same dimension. For example, if $x \in \mathbb{R}^n$ then $z \in \mathbb{R}^n$.
  - Note 2: For any invertible matrix A, $\det(A^{-1}) = \det(A)^{-1}$, then

$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial f(z)}{\partial z} \right) \right|^{-1}$$

Normalizing Flow

ROBOT INTELLIGENCE LAB

- Normalizing flow models
  - Consider a directed, latent-variable model over observed variables X and latent variables Z.
  - In a **normalizing flow model**, the mapping between Z and X, given by $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is deterministic and invertible such that $X = f_\theta(Z)$ and $Z = f_\theta^{-1}(X)$



  - Using change of variables, the marginal likelihood $p(x)$ is given by

$$p_X(x;\theta) = p_Z\left(f_\theta^{-1}(x)\right)\left|\det\left(\frac{\partial f_\theta^{-1}(x)}{\partial x}\right)\right|$$

Normalizing Flow

ROBOT INTELLIGENCE LAB

- Normalizing flow models
  - **Normalizing**: Change of variables gives a normalized density after applying an invertible transformation
  - **Flow**: Invertible transformations can be composed with each other

$$z_m \equiv f_\theta^m \circ \cdots \circ f_\theta^1(z_0) = f_\theta^m(f_\theta^{m-1}(\cdots(f_\theta^1(z_0)))) \equiv f_\theta(z_0)$$

  - Start with a simple distribution for z_0 (e.g., Gaussian)
  - Apply a sequence of M invertible transformations $x \equiv z_M$
  - By change of variables

$$p_X(x;\theta) = p_Z\left(f_\theta^{-1}(x)\right)\prod_{m=1}^{M}\left|\det\left(\frac{\partial(f_\theta^m)^{-1}(z_m)}{\partial z_m}\right)\right|$$

(Note: determinant of product equals product of determinants)

Planar flows

ROBOT INTELLIGENCE LAB

- **Planar flow**
- Invertible transformation

$$\mathbf{x} = f_\theta(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$$

parametrized by $\theta = (\mathbf{w}, \mathbf{u}, b)$ where $h(\cdot)$ is a non-linearity.

- Absolute value of the determinant of the Jacobian is given by

$$\left|\det \frac{\partial f_\theta(\mathbf{z})}{\partial \mathbf{z}}\right| = \left|\det\left(I + h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{u}\mathbf{w}^T\right)\right| = \left|1 + h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{u}^T\mathbf{w}\right|$$

(matrix determinant lemma)

https://arxiv.org/pdf/1505.05770

Planar flows

ROBOT INTELLIGENCE LAB

- **Learning** via **maximum likelihood** over the dataset $\mathscr{D}$.

$$\max_{\theta} \log p_X(\mathscr{D};\theta) = \sum_{x\in\mathscr{D}} \log p_Z\left(f_\theta^{-1}(x)\right) + \log \left| \deg\left(\frac{\partial f_\theta^{-1}(x)}{\partial x}\right)\right|$$

- **Exact likelihood evaluation** via inverse transform $\mathbf{x} \mapsto \mathbf{z}$ and change of variables formula
- Sampling via forward transformation $\mathbf{z} \mapsto \mathbf{x}$

$$\mathbf{z} \sim p_Z(\mathbf{z}) \quad \mathbf{x} = f_\theta(\mathbf{z})$$

- **Latent representations** inferred via inverse transformation (in inference network required!)

$$\mathbf{z} = f_\theta^{-1}(\mathbf{x})$$

ROBOT INTELLIGENCE LAB

InfoGAN

"InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets," 2016

InfoGAN?

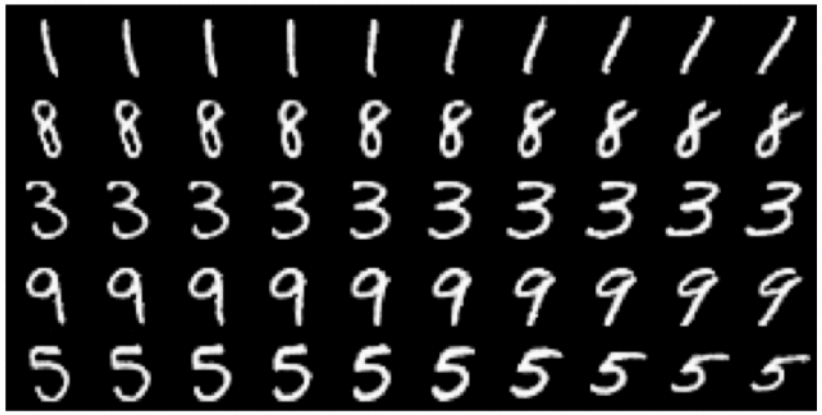
ROBOT INTELLIGENCE LAB

Information-theoretic extension to the GAN to learn **disentangled** representations

Disentangled Representation

(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)



(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

Mutual Information

ROBOT INTELLIGENCE LAB

- The input noise vector is decomposed into z, source of incompressible noise, and c, the latent code.
- In standard GAN, the generator may ignore the latent code (i.e., $P_G(x \mid c) = P_G(x)$). To handle this, latent codes c and generator distribution $G(z, c)$ should have high mutual information. Hence, $I(c; G(z, c))$ should be high.
- The mutual information between X and Y, $I(X; Y)$ can be expressed as:

$$I(X; Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X)$$

  - Intuitively, $I(X; Y)$ is the amount of reduced uncertainty in X when Y is observed.
  - If X and Y are independent, $I(X; Y) = 0$ as knowing one reveals nothing about the other.
  - If X and Y are related by a deterministic, invertible function, then maximal mutual information is attained.

InfoGAN

ROBOT INTELLIGENCE LAB

- The objective of InfoGAN is as follows:

$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c))$$

- Intuitively, given $x \sim P_G(z,c)$, we want $P_G(c|x)$ to have a small entropy.
  - However, computing the posterior $P(c|x)$ is usually intractable.

$$\begin{aligned} I(c; G(z,c)) &= H(c) - H(c|G(z,c)) \\ &= \mathbb{E}_{x\sim G(z,c)}\left[\mathbb{E}_{c'\sim P(c'|x)}[\log P(c'|x)]\right] + H(c) \\ &= \mathbb{E}_{x\sim G(z,c)}\left[D_{KL}(P(\cdot|x)\|Q(\cdot|x)) + \mathbb{E}_{c'\sim P(c'|x)}[\log Q(c'|x)]\right] + H(c) \\ &\geq \mathbb{E}_{x\sim G(z,c)}\left[\mathbb{E}_{c'\sim P(c'|x)}[\log Q(c'|x)]\right] + H(c) \end{aligned}$$

$$D_{KL}(P\|Q) = \int P \log \frac{P}{Q}$$

$$H(Y|X) = \int P(X,Y) \log \frac{P(X)}{P(X,Y)} = -\int P(X,Y) \log P(Y|X)$$

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

Variational Bound Trick

ROBOT INTELLIGENCE LAB

$$
\begin{aligned}
I(c; G(z,c)) &= H(c) - H(c|G(z,c)) \\
&= \mathbb{E}_{x\sim G(z,c)}\left[\mathbb{E}_{c'\sim P(c'|x)}\left[\log P(c'|x)\right]\right] + H(c) \\
&= \mathbb{E}_{x\sim G(z,c)}\left[D_{KL}(P(\cdot|x)\|Q(\cdot|x)) + \mathbb{E}_{c'\sim P(c'|x)}\left[\log Q(c'|x)\right]\right] + H(c) \\
&\geq \mathbb{E}_{x\sim G(z,c)}\left[\mathbb{E}_{c'\sim P(c'|x)}\left[\log Q(c'|x)\right]\right] + H(c)
\end{aligned}
$$

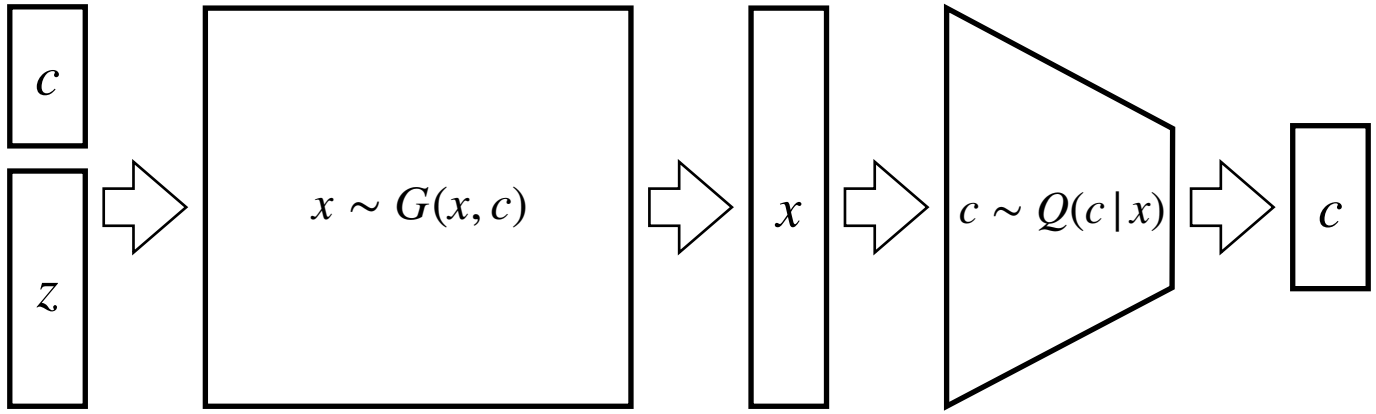
$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}\left[\log p(x)\right] &= \int p(x)\log p(x)dx \\
&= \int p(x)\log \frac{p(x)q(x)}{q(x)}dx \\
&= \int p(x)\log \frac{p(x)}{q(x)}dx + \int p(x)\log q(x)dx \\
&= D_{KL}(p(x)\|q(x)) + \mathbb{E}_{x\sim p(x)}\left[\log q(x)\right] \\
&\geq \mathbb{E}_{x\sim p(x)}\left[\log q(x)\right]
\end{aligned}
$$

Tractable Variational Lower Bound

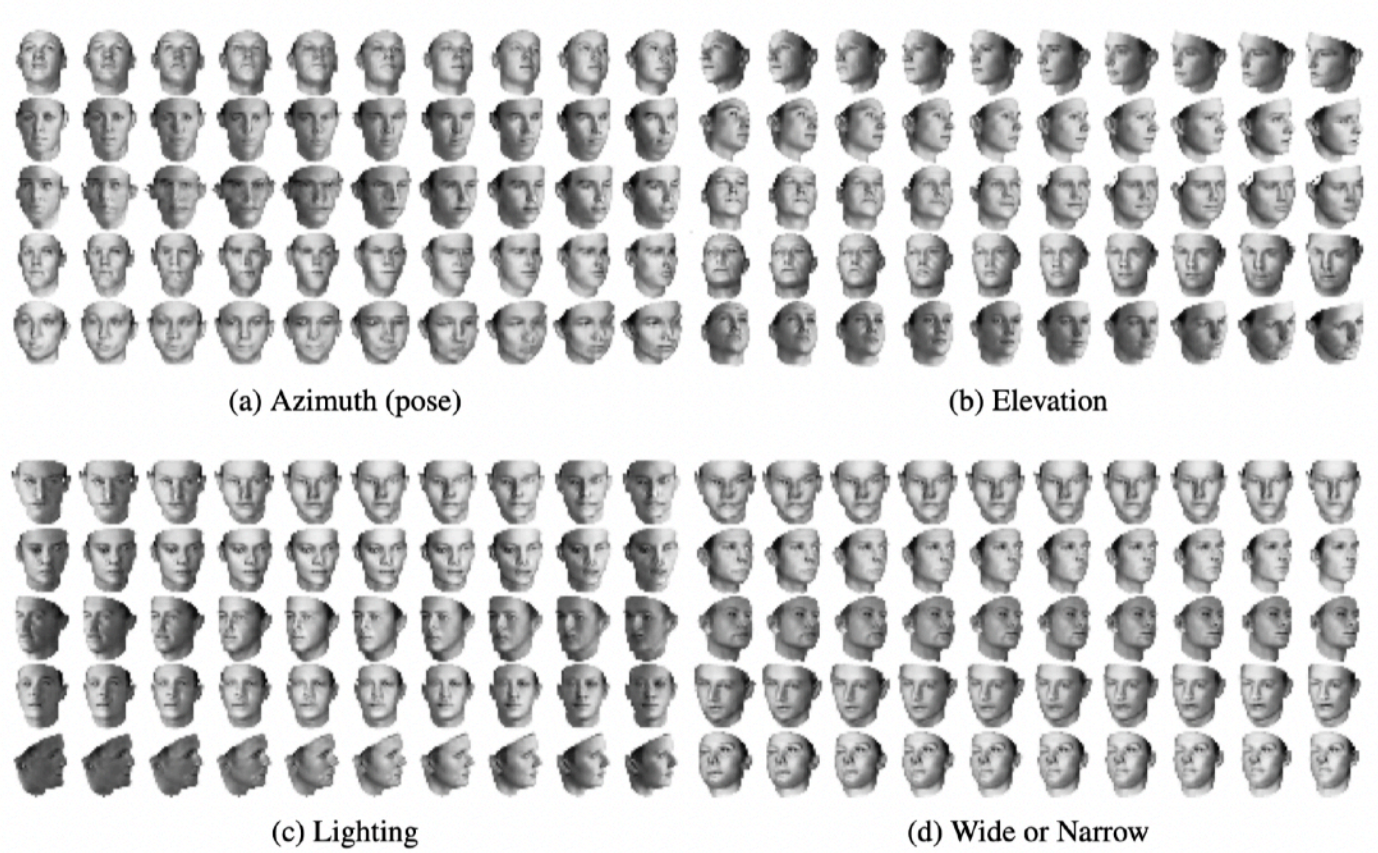
ROBOT INTELLIGENCE LAB

- Under suitable regularity conditions:

$$L_I(G, Q) = \mathbb{E}_{x \sim G(z,c), c' \sim P(c|x)}[\log Q(c' \mid x)] = \mathbb{E}_{c \sim P(c), x \sim G(z,c)}[\log Q(c \mid x)]$$



Results



(a) Azimuth (pose) (b) Elevation

(c) Lighting (d) Wide or Narrow

Figure 3: **Manipulating latent codes on 3D Faces:** We show the effect of the learned continuous latent factors on the outputs as their values vary from -1 to 1. In (a), we show that one of the continuous latent codes consistently captures the azimuth of the face across different shapes; in (b), the continuous code captures elevation; in (c), the continuous code captures the orientation of lighting; and finally in (d), the continuous code learns to interpolate between wide and narrow faces while preserving other visual features. For each factor, we present the representation that most resembles prior supervised results [7] out of 5 random runs to provide direct comparison.

Results

(a) Rotation (b) Width

Figure 4: **Manipulating latent codes on 3D Chairs:** In (a), we show that the continuous code captures the pose of the chair while preserving its shape, although the learned pose mapping varies across different types; in (b), we show that the continuous code can alternatively learn to capture the widths of different chair types, and smoothly interpolate between them. For each factor, we present the representation that most resembles prior supervised results [7] out of 5 random runs to provide direct comparison.

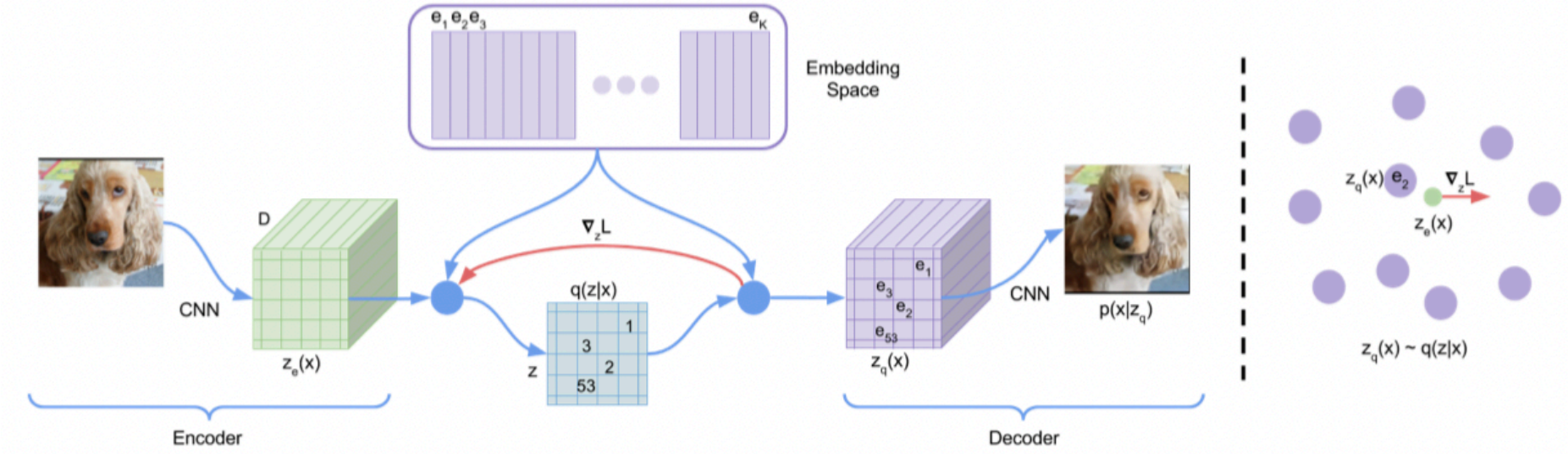
ROBOT INTELLIGENCE LAB

VQ-VAE

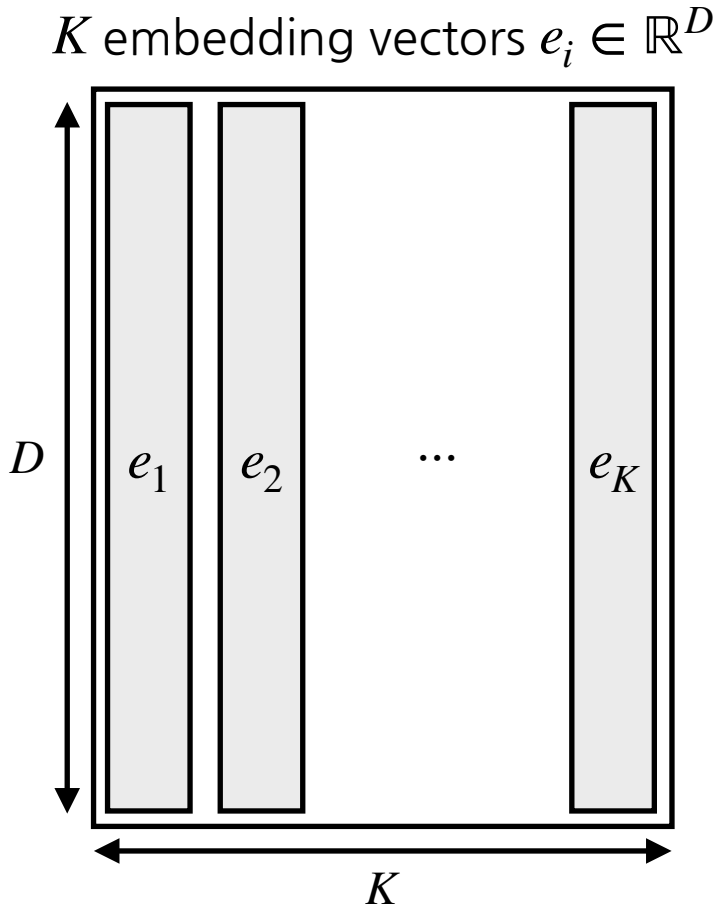
"Neural Discrete Representation Learning," 2018

VQ-VAE

- VQ-VAE combines the variational auto-encoder (VAE) framework with discrete latent representations using vector quantization (VQ).



Discrete Latent Variables



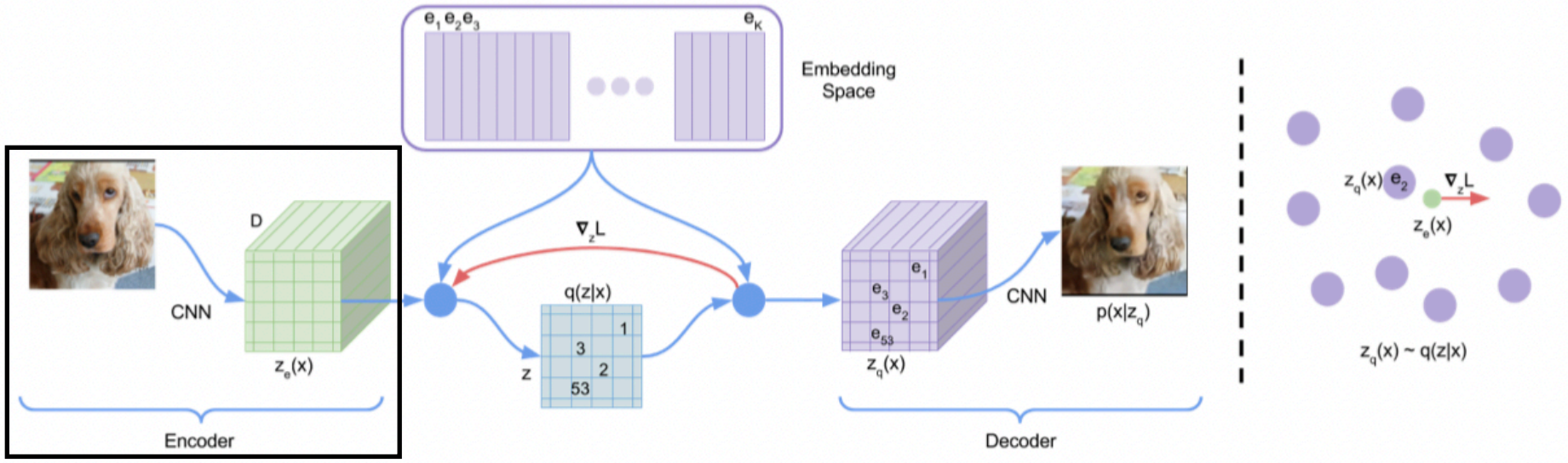
Discrete Latent Variables

ROBOT INTELLIGENCE LAB

- Latent embedding space consists of K embedding vectors $e_i \in \mathbb{R}^D$, $i \in 1,2,\ldots,K$.
- Inference
  - Given an input x, an encoder outputs $z_e(x) \in \mathbb{R}^D$ and the closest embedding vector e_k is selected.
  - In other words, $z_q(x) = e_k$ where where $k = \arg\min_j \|z_e(x) - e_j\|_2$.
- Learning
  - Since $\arg\min$ is not differentiable, the **straight-through estimator** that simply copies gradient from the decoder input $z_q(x)$ to encoder output $z(e)$.
  - To update embeddings, Vector Quantization (VQ) that uses the l_2 error to move the embedding vector towards the encoder outputs is used.

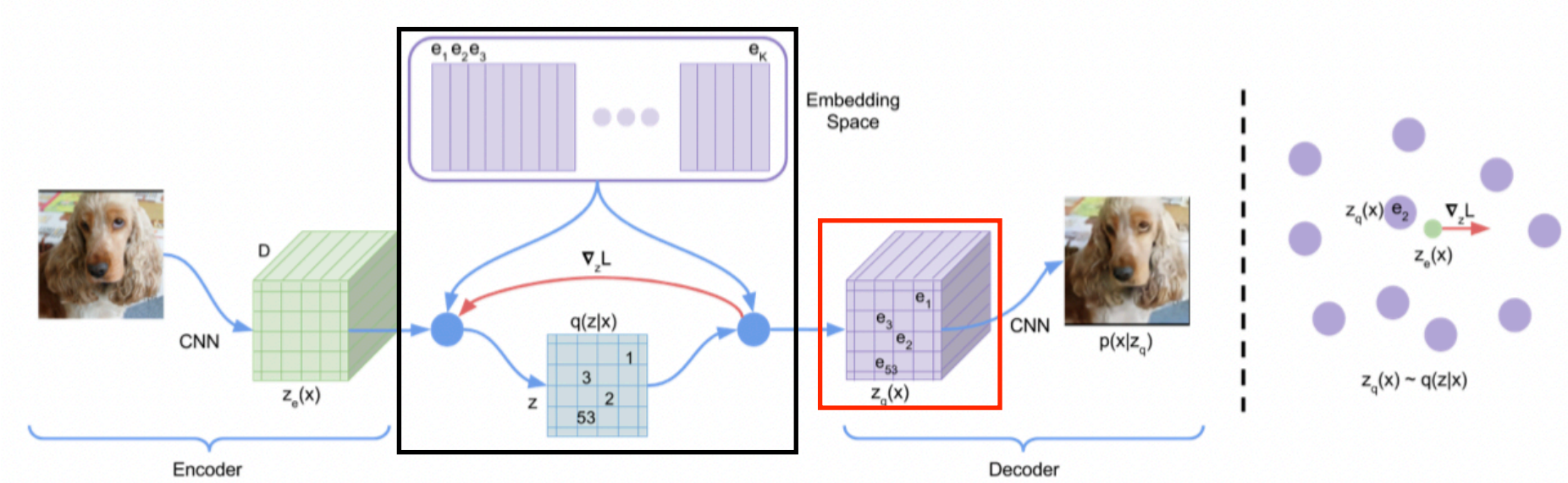
$$L = \log p(x|z_q(x)) + \|\mathrm{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \mathrm{sg}[e]\|_2^2,$$

Inference (1/3)



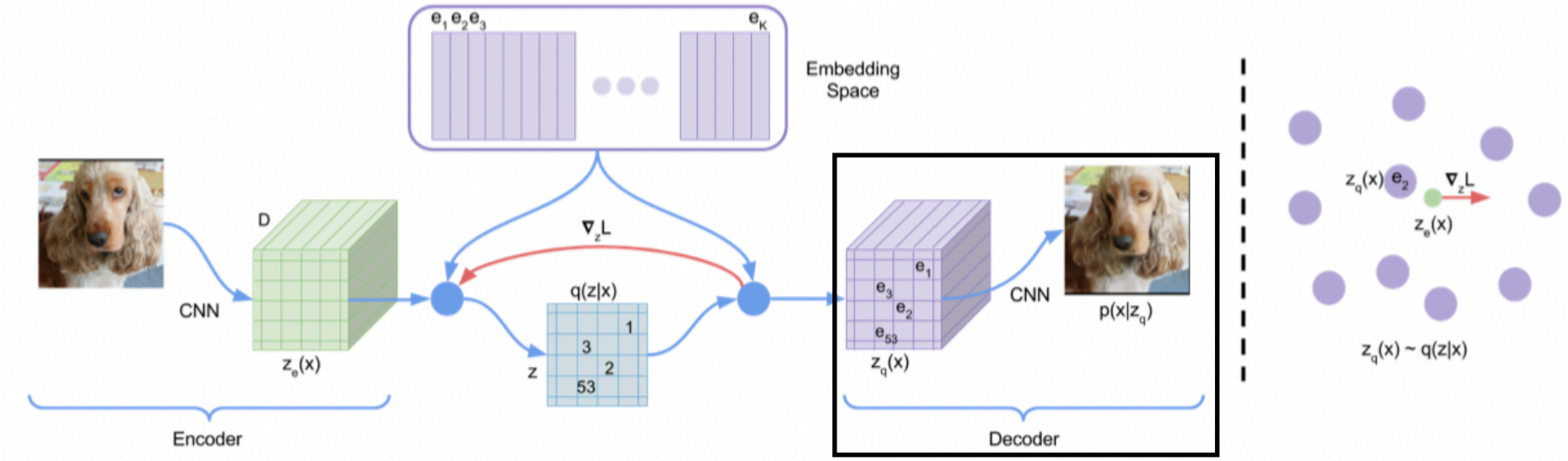
- Given an input image $x \in \mathbb{R}^{128\times128\times3}$, the encoder outputs $z_e(x) \in \mathbb{R}^{32\times32\times D}$.

Inference (2/3)



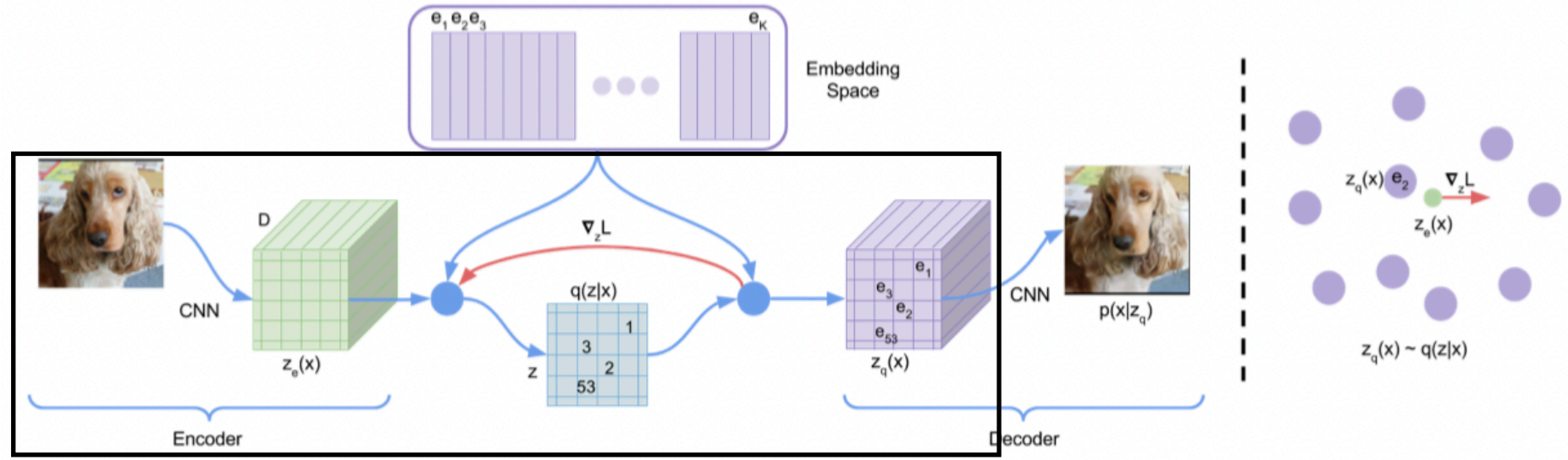
- Given an input image $x \in \mathbb{R}^{128\times128\times3}$, the encoder outputs $z_e(x) \in \mathbb{R}^{32\times32\times D}$.
- Each 32×32 embedding vector is mapped to its closest $e_k \in \mathbb{R}^D$ in the Embedding space.

Inference (3/3)



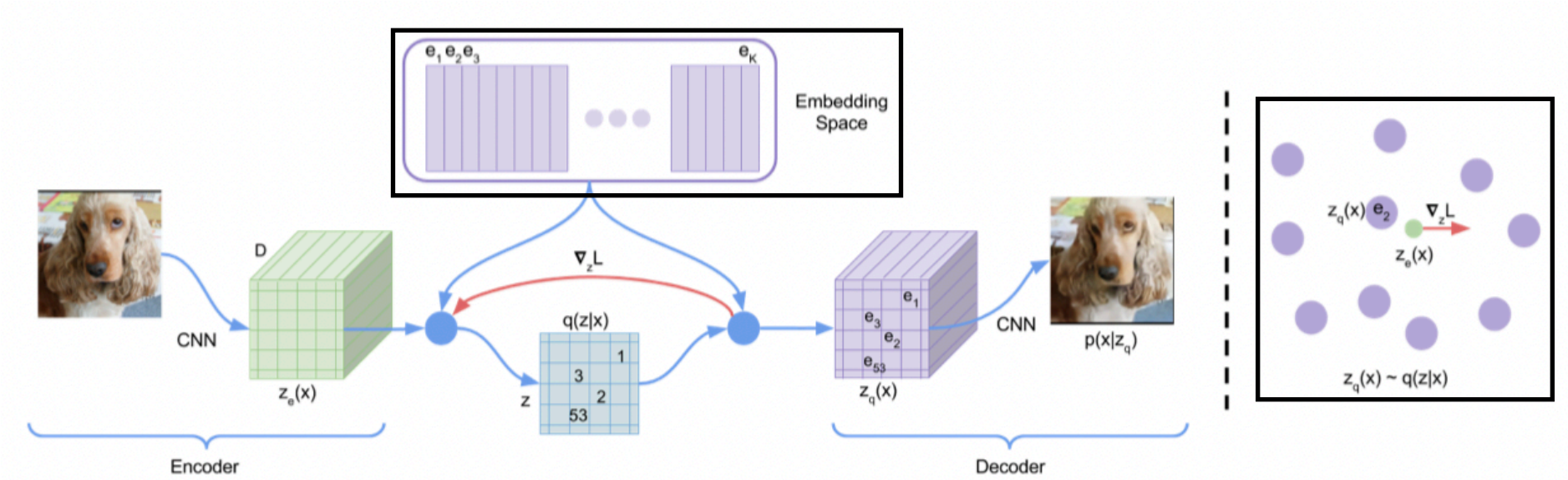
- Given an input image $x \in \mathbb{R}^{128\times128\times3}$, the encoder outputs $z_e(x) \in \mathbb{R}^{32\times32\times D}$.
- Each 32×32 embedding vector is mapped to its closest $e_k \in \mathbb{R}^D$ in the Embedding space.
- The mapped $z_q(x) \in \mathbb{R}^{128\times128\times D}$ is fed into the decoder to reconstruct the given input image.
- The bit reduction is $\frac{128 \times 128 \times 3 \times 8}{32 \times 32 \times 9} \approx 42.6$ where 8 comes from $256 = 2^8$ color resolutions and 9 comes from $K = 512 = 2^9$ embedding vectors.

Train (1/3)



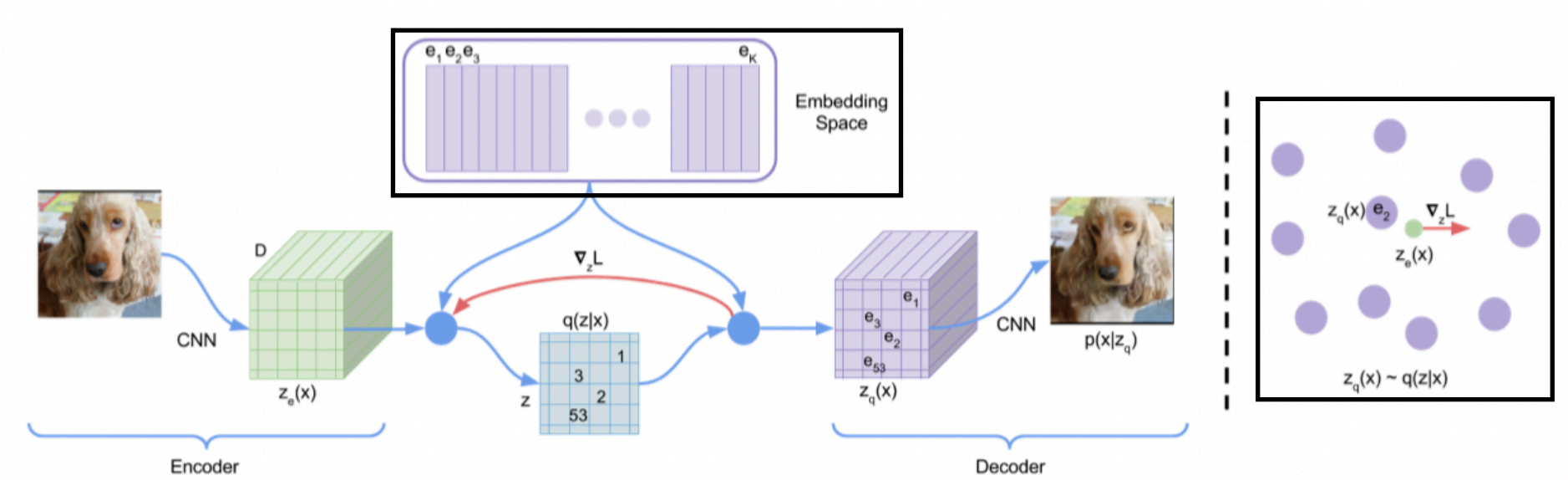
- The **straight-through estimator** that simply copies gradient from the decoder input $z_q(x)$ to the encoder output $z(e)$ is used to train the encoder part.

Train (2/3)



- The **straight-through estimator** that simply copies gradient from the decoder input $z_q(x)$ to the encoder output $z(e)$ is used to train the encoder part.
- To learn the embedding space, Vector Quantization (VQ) is used. The VQ objective uses the l_2 error to move the embedding vectors e_i towards the encoder outputs $z_e(x)$.

Train (3/3)



- The **straight-through estimator** that simply copies gradient from the decoder input $z_q(x)$ to the encoder output $z(e)$ is used to train the encoder part.
- To learn the embedding space, Vector Quantization (VQ) is used. The VQ objective uses the l_2 error to move the embedding vectors e_i towards the encoder outputs $z_e(x)$.
- After training, an autoregressive distribution over z, $p(z)$, is fitted so that we can generate z via ancestral sampling.

Overall Loss

ROBOT INTELLIGENCE LAB

$$L = \log p(x \mid z_q(x)) + \|sg[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - sg[e]\|_2^2$$

reconstruction

Vector Quantization

Results

ROBOT INTELLIGENCE LAB

Reconstruction

Figure 2: Left: ImageNet 128x128x3 images, right: reconstructions from a VQ-VAE with a 32x32x1 latent space, with K=512.

Results

ROBOT INTELLIGENCE LAB

Figure 3: Samples (128x128) from a VQ-VAE with a PixelCNN prior trained on ImageNet images. From left to right: kit fox, gray whale, brown bear, admiral (butterfly), coral reef, alp, microwave, pickup.

Results

ROBOT INTELLIGENCE LAB

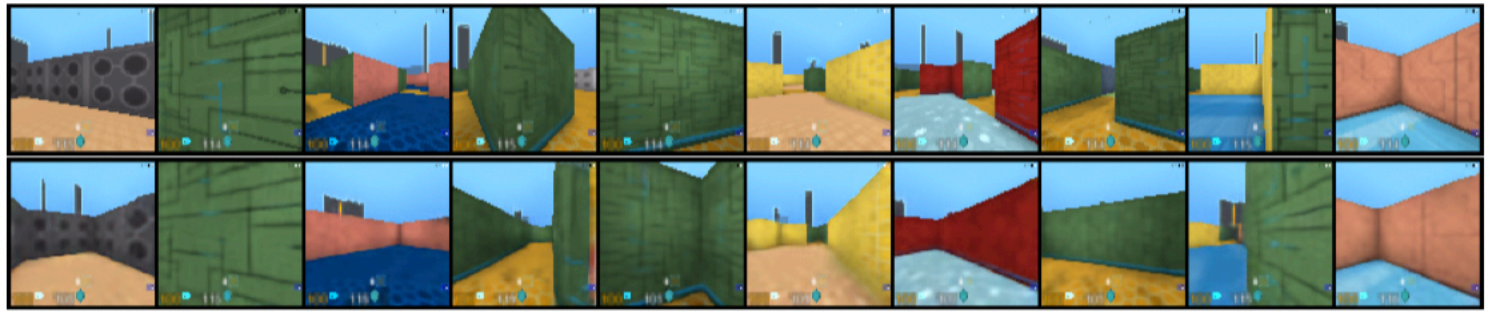


Figure 5: Top original images, Bottom: reconstructions from a 2 stage VQ-VAE, with 3 latents to model the whole image (27 bits), and as such the model cannot reconstruct the images perfectly. The reconstructions are generated by sampled from the second PixelCNN prior in the 21x21 latent domain of first VQ-VAE, and then decoded with standard VQ-VAE decoder to 84x84. A lot of the original scene, including textures, room layout and nearby walls remain, but the model does not try to store the pixel values themselves, which means the textures are generated procedurally by the PixelCNN.

ROBOT INTELLIGENCE LAB

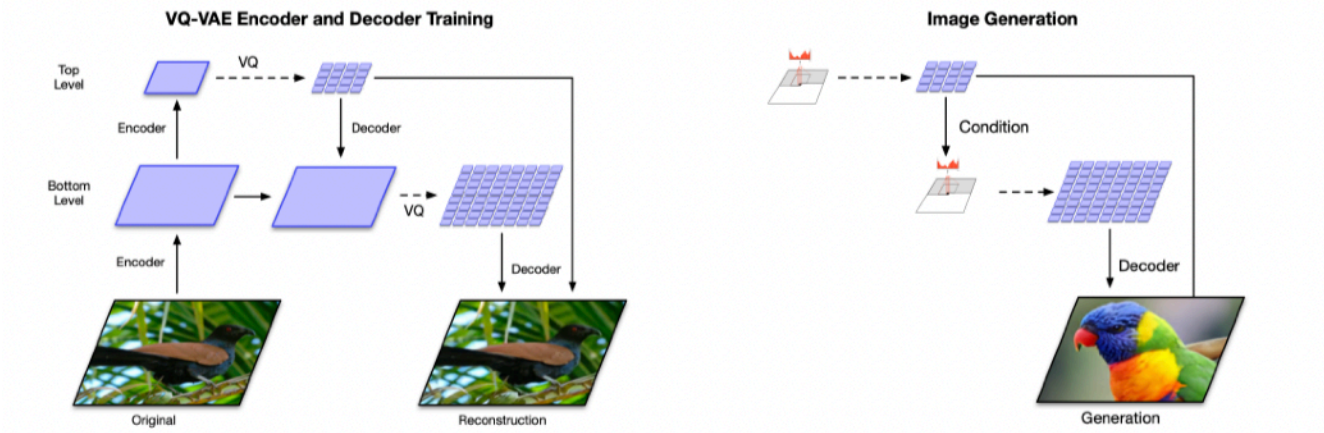
VQ-VAE2

"Generating Diverse High-Fidelity Images with VQ-VAE-2," 2019

Hierarchical Latent Codes

ROBOT INTELLIGENCE LAB

- As opposed to vanilla VQ-VAE, VQ-VAE2 uses a hierarchy of vector quantized codes.
- The encoder first transforms and downsamples the image by a factor of 4 to a 64×64 representation which is quantized to our bottom level latent map.



(a) Overview of the architecture of our hierarchical VQ-VAE. The encoders and decoders consist of deep neural networks. The input to the model is a 256×256 image that is compressed to quantized latent maps of size 64×64 and 32×32 for the *bottom* and *top* levels, respectively. The decoder reconstructs the image from the two latent maps.

(b) Multi-stage image generation. The top-level PixelCNN prior is conditioned on the class label, the bottom level PixelCNN is conditioned on the class label as well as the first level code. Thanks to the feed-forward decoder, the mapping between latents to pixels is fast. (The example image with a parrot is generated with this model).

Figure 2: VQ-VAE architecture.

Results

ROBOT INTELLIGENCE LAB

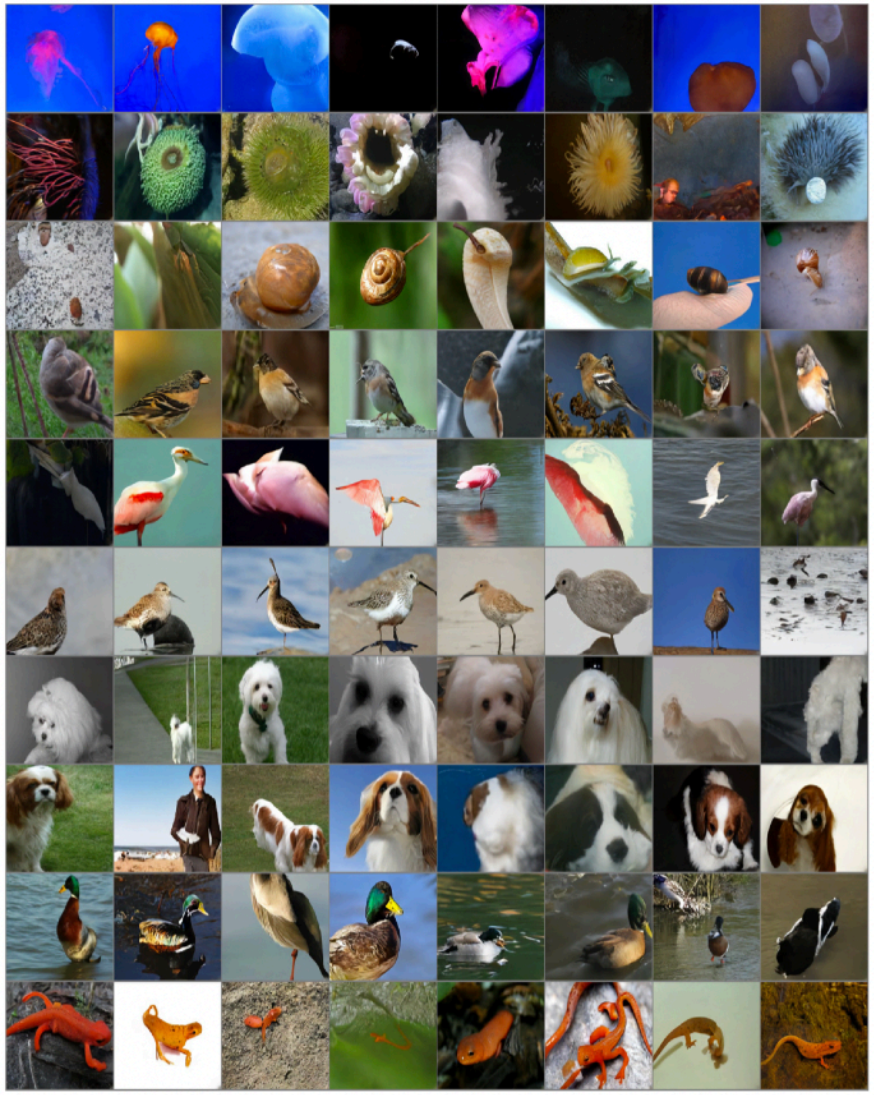


Figure 4: Class conditional random samples. Classes from the top row are: 108 sea anemone, 109 brain coral, 114 slug, 11 goldfinch, 130 flamingo, 141 redshank, 154 Pekinese, 157 papillon, 97 drake, and 28 spotted salamander.

Results

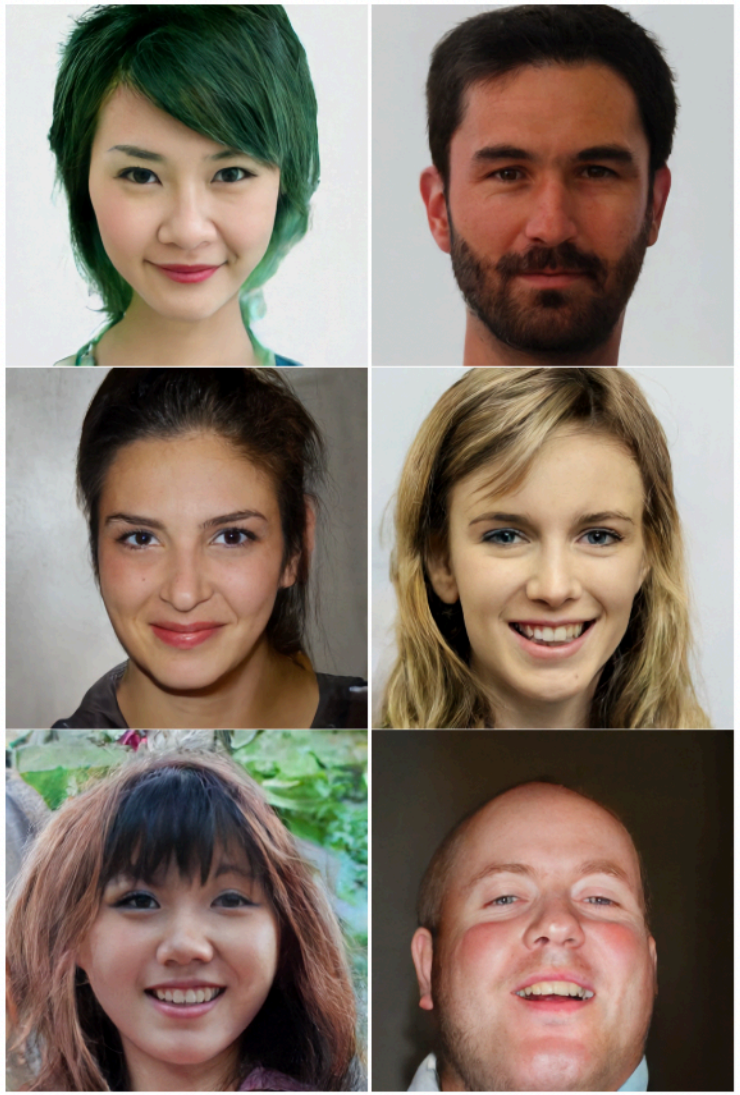


Figure 6: Representative samples from the three level hierarchical model trained on FFHQ-1024 × 1024. The model generates realistic looking faces that respect long-range dependencies such as matching eye colour or symmetric facial features, while covering lower density modes of the dataset (e.g., green hair). Please refer to the supplementary material for more samples, including full resolution samples.

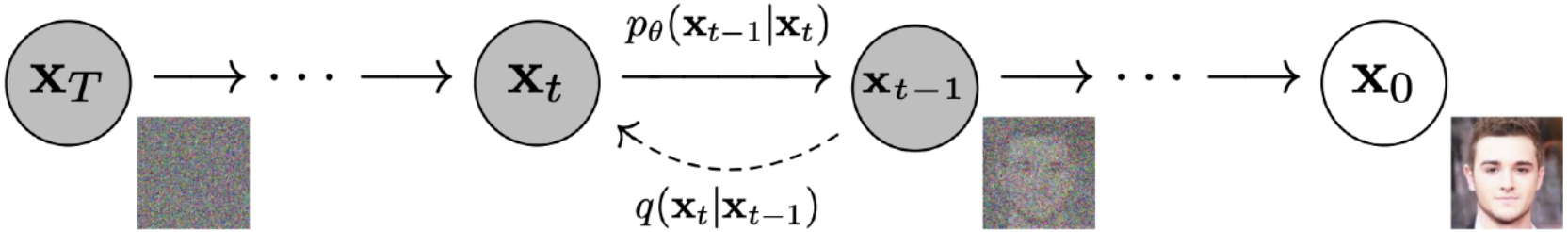
ROBOT INTELLIGENCE LAB

ROBOT INTELLIGENCE LAB

DDPM

"Denoising Diffusion Probabilistic Models," 2020

Diffusion Model



- The forward diffusion process is defined as

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t|\sqrt{1-\beta_t}x_{t-1},\, \beta_t I)$$

- Then, the jump-diffusion process can be derived as

$$\begin{aligned}
x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}z_{t-1}, \quad \text{where } \alpha_t = 1-\beta_t \text{ and } z_t \sim \mathcal{N}(0, I) \\
&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}z_{t-2}) + \sqrt{1-\alpha_t}z_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{z}_{t-2} \\
&\vdots \\
&= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_0, \quad \text{where } \bar{\alpha}_t = \prod_{s=1}^{t}\alpha_s
\end{aligned}$$

- Hence, $q(x_t|x_0) = \mathcal{N}\left(x_0; \sqrt{\bar{\alpha}_t}x_0,\, (1-\bar{\alpha}_t)I\right)$.

Maximum Likelihood Learning

ROBOT INTELLIGENCE LAB

$$
\begin{aligned}
\underbrace{\mathbb{E}_{x_0 \sim q(x_0)}\left[-\log p_\theta(x_0)\right]}_{\text{Maximum Likelihood Learning} \downarrow} &= \mathbb{E}_{x_0 \sim q(x_0)}\left[-\log \int p_\theta(x_0, x_1, \ldots, x_T) dx_1 dx_2 \ldots dx_T\right] \\
&= \mathbb{E}_{x_0 \sim q(x_0)}\left[-\log \int p_\theta(x_{0:T}) \frac{q(x_{1:T}|x_0)}{q(x_{1:T}|x_0)} dx_{1:T}\right] \\
&= \mathbb{E}_{x_0 \sim q(x_0)}\left[-\log \mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right]\right] \\
&\leq \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[\log \frac{p(x_T) p_\theta(x_{T-1}|x_T) p_\theta(x_{T-2}|x_{T-1}) \cdots p_\theta(x_0|x_1)}{q(x_1|x_0) q(x_2|x_1) \cdots q(x_T|x_{T-1})}\right] \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[-\log p(x_T) - \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}\right] \\
&\vdots \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[-\log \frac{p(x_T)}{q(x_T|x_0)} - \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} - \log p_\theta(x_0|x_1)\right] \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[\sum_{t=2}^{T} \underbrace{D_{KL}\left(q(x_{t-1}|x_t, x_0) \| p_\theta(x_{t-1}|x_t)\right)}_{L_{t-1} \downarrow}\right] + C
\end{aligned}
$$

Reverse Posterior

ROBOT INTELLIGENCE LAB

- Using the jump-forward process, $q(x_t|x_0) = \mathcal{N}\left(x_0; \sqrt{\bar{\alpha}_t}x_0,\ (1-\bar{\alpha}_t)I\right)$, we can derive the conditioned reverse posterior distribution, $q(x_{t-1}|x_t, x_0)$ as follows:

$$
\begin{aligned}
q(x_{t-1}|x_t, x_0) &= \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)} \\
&= \frac{q(x_t|x_{t-1})q(x_{t-1}|x_0)}{q(x_t|x_0)} \\
&\vdots \\
&= \mathcal{N}\left(x_{t-1};\ \underbrace{\frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{(1-\bar{\alpha}_t)}x_0 + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\sqrt{\alpha_t}x_t}_{\tilde{\mu}_{t-1}(x_t, x_0)},\ \underbrace{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}I}_{\tilde{\sigma}^2_{t-1}}\right)
\end{aligned}
$$

Loss Function (1/2)

ROBOT INTELLIGENCE LAB

- Recall that $L_{t-1} = D_{KL}\left(q(x_{t-1}\,|\,x_t, x_0)\|p_\theta(x_{t-1}\,|\,x_t)\right)$.
- Our learning objective is to optimize $p_\theta(x_{t-1}\,|\,x_t)$ that best approximates the reverse posterior:

$$q(x_{t-1}\,|\,x_t, x_0) = \mathcal{N}\left(x_{t-1};\ \underbrace{\frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{(1-\bar{\alpha}_t)}x_0 + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\sqrt{\alpha_t}x_t}_{\tilde{\mu}_{t-1}(x_t,x_0)},\ \underbrace{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}I}_{\tilde{\sigma}^2_{t-1}}\right)$$

- Assuming that the standard deviation of $p_\theta(x_{t-1}\,|\,x_t)$ is constant and the mean is $\mu_\theta(x_t, t)$, we get the following objective:

$$L_{t-1} = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\|\tilde{\mu}_{t-1}(x_t, x_0) - \mu_\theta(x_t, x_0)\|_2^2\right] + C$$

- Note that $\tilde{\mu}_{t-1}(x_t, x_0)$ can be parametrized by x_t and $\epsilon \sim \mathcal{N}(0,I)$ using $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$:

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

Loss Function (2/2)

ROBOT INTELLIGENCE LAB

- Plugging $\tilde{\mu}_t(x_t, \epsilon) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$ into $L_{t-1} - C$, we get

$$L_{t-1} - C = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,I)}\left[\frac{1}{2\sigma_t^2}\left\| \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \mu_\theta(x_t, t)\right\|_2^2\right]$$

$$\text{where } x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

- Now, if we parametrize $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$, aka denoising objectives, then, we get

$$L_{t-1} - C = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,I)}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right] \text{ where } x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

DDPM Training objective

Algorithm 1 Training

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

Algorithm 2 Sampling

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

DDPM Constants (1/2)

ROBOT INTELLIGENCE LAB

- Forward Diffusion Process

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

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I\right) \text{ where } \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$$

- Reverse Diffusion Posterior

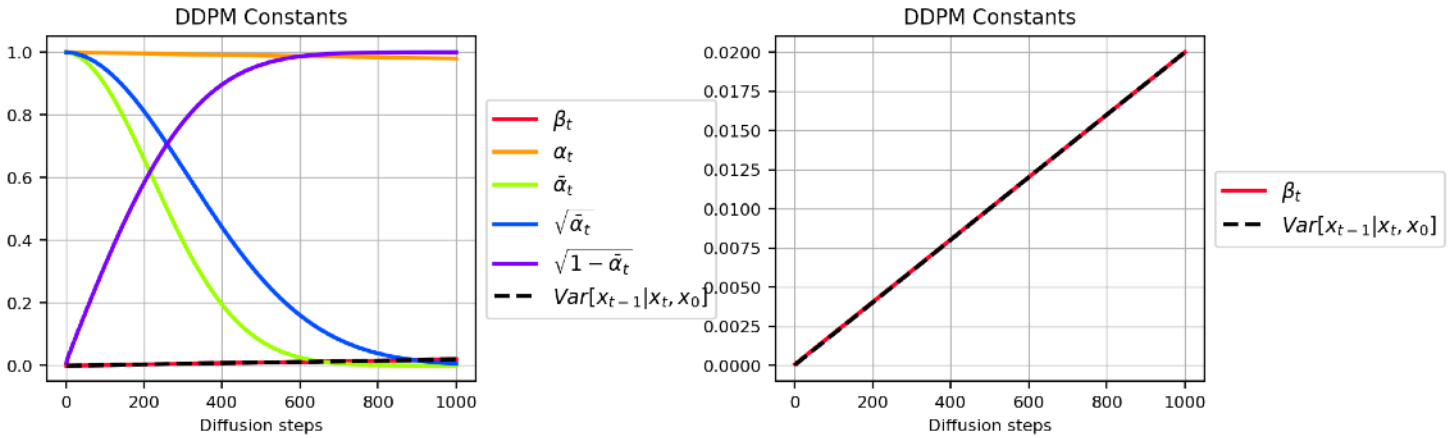
$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(x_{t-1}; \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{(1-\bar{\alpha}_t)}x_0 + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\sqrt{\alpha_t}x_t, \frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\right)$$

- Practically, scheduling $\{\beta_t\}_{t=1}^{T}$ is important.

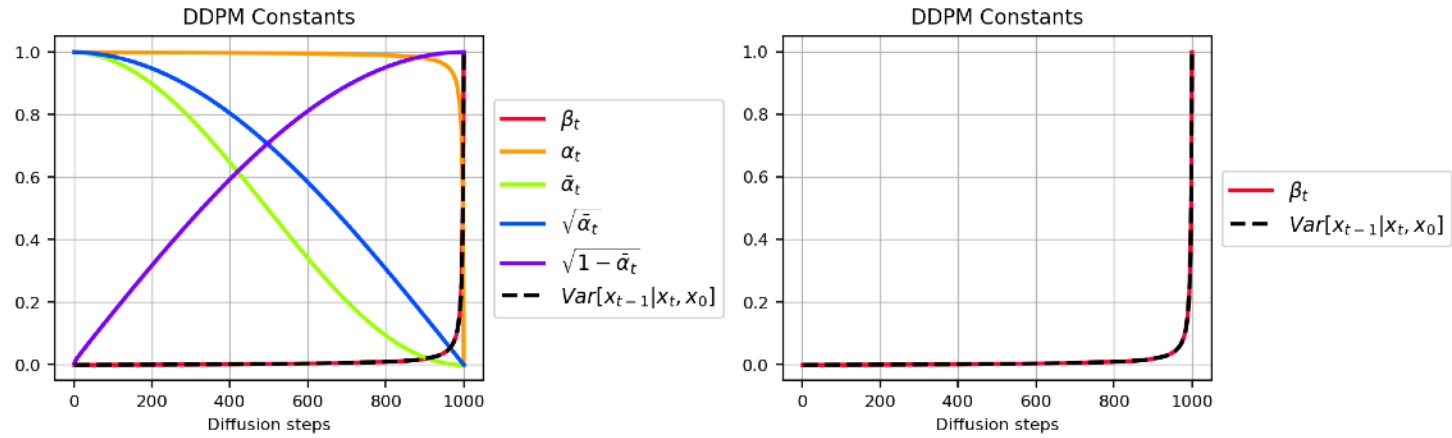
DDPM Constants (2/2)

ROBOT INTELLIGENCE LAB

- Linear scheduling



- Cosine scheduling



ROBOT INTELLIGENCE LAB

Score-based Generative Model

"Generative Modeling by Estimating Gradients of the Data Distribution," 2019

Score-based Generative Model

ROBOT INTELLIGENCE LAB

- The core of the score-based generative model (SBGM) is, instead of directly focussing on $p(x)$, we aim to model the score function:

$$s(x) \triangleq \nabla_x \log p(x)$$

where one can train $s_\theta(x)$ by the following objectives

$$\min_\theta \mathbb{E}_{x\sim p(x)} \left[\| \nabla_x \log p(x) - s_\theta(x) \|_2^2 \right]$$

- The main drawback of the original SBGM is that $s_\theta(x)$ can only properly be trained on $x \sim p(x)$, making it less accurate in low-density regions.
- To mitigate this, a noise conditional score-based model is presented:

$$\sum_{i=1}^{L} \lambda(i) \mathbb{E}_{x\sim p_{\sigma_i(x)}} \left[\| \nabla_x \log p_{\sigma_i}(x) - s_\theta(x, i) \|_2^2 \right]$$

where $p_{\sigma_i}(x) = \int p(y) \mathcal{N}(x; y, \sigma_i^2 I) dy$ is a noise perturbed distribution and $\sigma_1 < \sigma_2 < \cdots < \sigma_L$.

- Note that this noise scheduling (or noise perturbation) procedure is important.

SBGM with Stochastic Differential Equation

ROBOT INTELLIGENCE LAB

- If we have an infinite number of noise scales, $L \to \infty$, the noise perturbation procedure becomes a continuous-time diffusion process (aka a stochastic process).
- Then, how can we represent a stochastic process?
- Many stochastic processes are solutions of stochastic differential equations (SDEs)

$$dx = f(x,t)dt + g(t)dw$$

where $f(x,t)$ is a drift coefficient, $g(t)$ is a diffusion coefficient, and dw is a Brownian motion.

- The solution of an SDE is a continuous collection of random variables $\{x(t)\}_{t\in[0,T]}$ where we want $p_0(x) = p_{\text{data}}(x)$ and $p_T(x) = \pi(x)$, a prior distribution.
- Note that any SDE has a corresponding reverse SDE

(forward) $dx = f(x,t)dt + g(t)dw$

(reverse) $dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw$ where $p_t(x)$ is the distribution of x_t

- Hence, once we can approximate $\nabla_x \log p_t(x) \approx s_\theta(x,t)$ for a forward SDE, one may generate samples using the reverse SDE.
- The training objective becomes:

$\mathbb{E}_{t\in U(0,T)}\mathbb{E}_{p_t(x)}\left[\lambda(t)\|\nabla_x \log p_t(x) - s_\theta(x,t)\|_2^2\right]$ where $\lambda(t)$ is some constant to balance the magnitude.

VE-SDE

ROBOT INTELLIGENCE LAB

- In the Variance Exploding Stochastic Differential Equation (VE-SDE), we first define the forward SDE as

$$dx = \sigma^t dw \text{ where } t \in [0,1] \text{ and } \sigma \text{ as a constant (e.g., } \sigma = 25).$$

- In this case,

$$p_t(x_t \mid x_0) = \mathcal{N}\left(x_t; x_0, \frac{1}{2\log\sigma}(\sigma^{2t} - 1)I\right)$$

where $\bar{\sigma}_t^2 = \frac{1}{2\log\sigma}(\sigma^{2t} - 1)$ is the marginal probability variance and $\lambda(t) = \bar{\sigma}_t^2$.

- Then, $\nabla_{x_t} \log p_t(x_t \mid x_0) = \frac{2\log\sigma}{(\sigma^{2t} - 1)}(x_0 - x_t) = \frac{1}{\bar{\sigma}_t^2}(x_0 - x_t)$.

VE-SDE

ROBOT INTELLIGENCE LAB

- In the training phase,
  - $\min_\theta \mathbb{E}_{t\sim U(0,1)}\left[\lambda(t)\mathbb{E}_{x_0\sim p(x_0)}\mathbb{E}_{x_t\sim p(x_t|x_0)}[\|s_\theta(x_t,t)-\nabla_x \log p_t(x_t\,|\,x_0)\|_2^2]\right]$
- By plugging in, $x_t = x_0 + \bar{\sigma}_t z$,
  - $\min_\theta \mathbb{E}_{t\sim U(0,1)}\mathbb{E}_{x_0\sim p(x_0),z\sim\mathcal{N}(0,I)}\mathbb{E}_{x_t=x_0+\bar{\sigma}_t z}\left[\lambda(t)\|s_\theta(x_t,t)-\nabla_x \log p_t(x_t\,|\,x_0)\|_2^2\right]$
- By using the following property: $f(x)=\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-u)\right)$, $\nabla_x \log f(x)=\Sigma^{-1}(\mu-x)$ and plugging $\lambda(t)=\bar{\sigma}_t^2$:
  - $\min_\theta \mathbb{E}_{t\sim U(0,1)}\mathbb{E}_{x_0\sim p(x_0),z\sim\mathcal{N}(0,I),x_t=x_0+\bar{\sigma}_t z}\left[\bar{\sigma}_t^2\left\|s_\theta(x_t,t)-\frac{1}{\bar{\sigma}^2{}_t}(x_0-x_t)\right\|_2^2\right]$
  - $\min_\theta \mathbb{E}_{t\sim U(0,1)}\mathbb{E}_{x_0\sim p(x_0),z\sim\mathcal{N}(0,I),x_t=x_0+\bar{\sigma}_t z}\left[\left\|\bar{\sigma}_t s_\theta(x_t,t)-\frac{1}{\bar{\sigma}_t}(x_0-x_t)\right\|_2^2\right]$
- Denote $\bar{s}_\theta(x_t,t)=\bar{\sigma}_t x_\theta(x_t,t)$ and using $x_t = x_0 + \bar{\sigma}_t z$,
  - $\min_\theta \mathbb{E}_{t\sim U(0,1)}\mathbb{E}_{x_0\sim p(x_0),z\sim\mathcal{N}(0,I),x_t=x_0+\bar{\sigma}_t z}\left[\left\|\bar{s}_\theta(x_t,t)+z\right\|_2^2\right]$

VE-SDE

ROBOT INTELLIGENCE LAB

- To summarize, the training phase becomes:
  1. Sample $t \sim U(0,1)$, $x_0 \sim p(x_0)$, $z \sim \mathcal{N}(0,I)$
  2. Compute $x_t = x_0 + \bar{\sigma}_t z$
  3. Train: $\min_\theta \|\bar{s}_\theta(x_t, t) + z\|_2^2$
- For sampling, we can utilize Euler-Maruyama sampling:
  - $x_T \sim \mathcal{N}(0, \bar{\sigma}_T I)$ where $T = 1.0$
  - loop $t = T, T - \Delta t, T - 2\Delta t, \ldots, 0$

$$x_{t-\Delta t} = x_t + \sigma^{2t} s_\theta(x_t, t)\Delta t + \sigma^t \sqrt{\Delta t} z_t$$

$$= x_t + \frac{\sigma^{2t}}{\bar{\sigma}_t} \bar{s}_\theta(x_t, t)\Delta t + \sigma^t \sqrt{\Delta t} z_t \quad \text{where } z_t \sim \mathcal{N}(0,I)$$

ROBOT INTELLIGENCE LAB

ADM

"Diffusion Models Beat GANs on Image Synthesis," 2021

Ablated Diffusion Model

ROBOT INTELLIGENCE LAB

- This paper presents **Ablated Diffusion Model** (ADM):
  - Several improvements techniques over DDPM
  - Conditioned generation via classifier-guidance

Ablated Diffusion Model

ROBOT INTELLIGENCE LAB

- ADM improves existing DDPM via:
  - Non-constant variance:

$$\Sigma_\theta(x_t, t) = \exp(v \log \beta_t + (1 - v)\log \tilde{\beta}_t)$$

where $v_\theta(x_t, t)$ is the output of the neural network,

β_t (upper bound) is a scheduled constant ($\beta_1 = 10^{-4}$ to $\beta_T = 0.02$)

,and $\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t$ (lower bound), $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=0}^{t} \alpha_s$

  - Fewer steps sampling using DDIM

Architecture Improvements

ROBOT INTELLIGENCE LAB

- Following architectural changes are considered:
  - Increasing depth versus width, holding model size relatively constant
  - Increasing the number of attention heads
  - Using attention at 32×32, 16×16, and 8×8 resolutions rather than only 16×16
  - Using the BigGAN residual block for upsampling and downsampling the activations
  - Rescaling residual connections with $\frac{1}{\sqrt{2}}$.

Classifier Guidance

ROBOT INTELLIGENCE LAB

- For conditioned generation,
  - an additional classifier $p_\phi(y \mid x_t, t)$ is trained and use gradients $\nabla_{x_t} \log p_\phi(y \mid x_t, t)$ to guide the diffusion sampling process towards an arbitrary class label y.
  - Note that the classifier $p_\phi(y \mid x_t, t)$ should be trained on noisy images x_t (how?)

Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return x_0

Samples from ImageNet

Samples from LSUN (bedroom)

ROBOT INTELLIGENCE LAB

Samples from LSUN (horse)

Samples from LSUN (cat)

ROBOT INTELLIGENCE LAB

CFG

"Classifier-Free Diffusion Guidance," 2021

Classifier-Free Guidance

ROBOT INTELLIGENCE LAB

- Classifier guidance mixes a score estimate with a classifier gradient during sampling

$$x_{t-1} \leftarrow \mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y \mid x_t), \Sigma) \text{ where } s \text{ is gradient scale}$$

- A downside classifier guidance is that it requires an additional classifier model and thus complicates the training pipeline.
- CFG only modifies the existing diffusion model by making the reverse process function approximator receives c an input (i.e., $e_\theta(z_\lambda, c)$).

$$\tilde{\epsilon}_\theta(z_\lambda, c) = (1 + w)\epsilon_\theta(z_\lambda, c) - w\epsilon_\theta(z_\lambda)$$

where c is the class label,$\epsilon_\theta(z_\lambda) = \epsilon_\theta(z_\lambda, c = 0)$, and w is the implied-classifier weights

Tradeoffs

- **FID**: sample variety
- **IS**: individual sample fidelity

Method	FID (↓)	IS (↑)
ADM [3]	2.07	-
CDM [6]	**1.48**	67.95
Ours, no guidance	1.80	53.71
Ours, with guidance		
$w = 0.1$	1.55	66.11
$w = 0.2$	2.04	78.91
$w = 0.3$	3.03	92.8
$w = 0.4$	4.30	106.2
$w = 0.5$	5.74	119.3
$w = 0.6$	7.19	131.1
$w = 0.7$	8.62	141.8
$w = 0.8$	10.08	151.6
$w = 0.9$	11.41	161
$w = 1.0$	12.6	170.1
$w = 2.0$	21.03	225.5
$w = 3.0$	24.83	250.4
$w = 4.0$	26.22	**260.2**

Figure 1: ImageNet 64x64 results

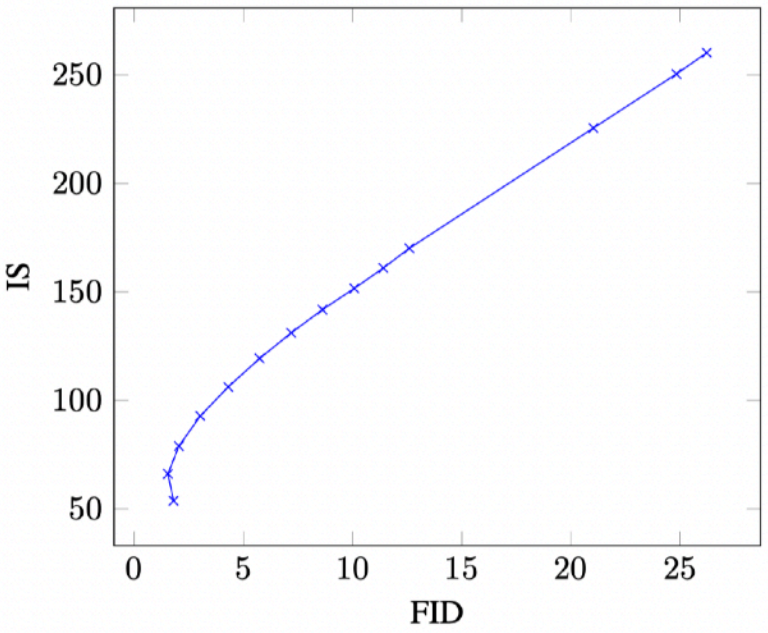
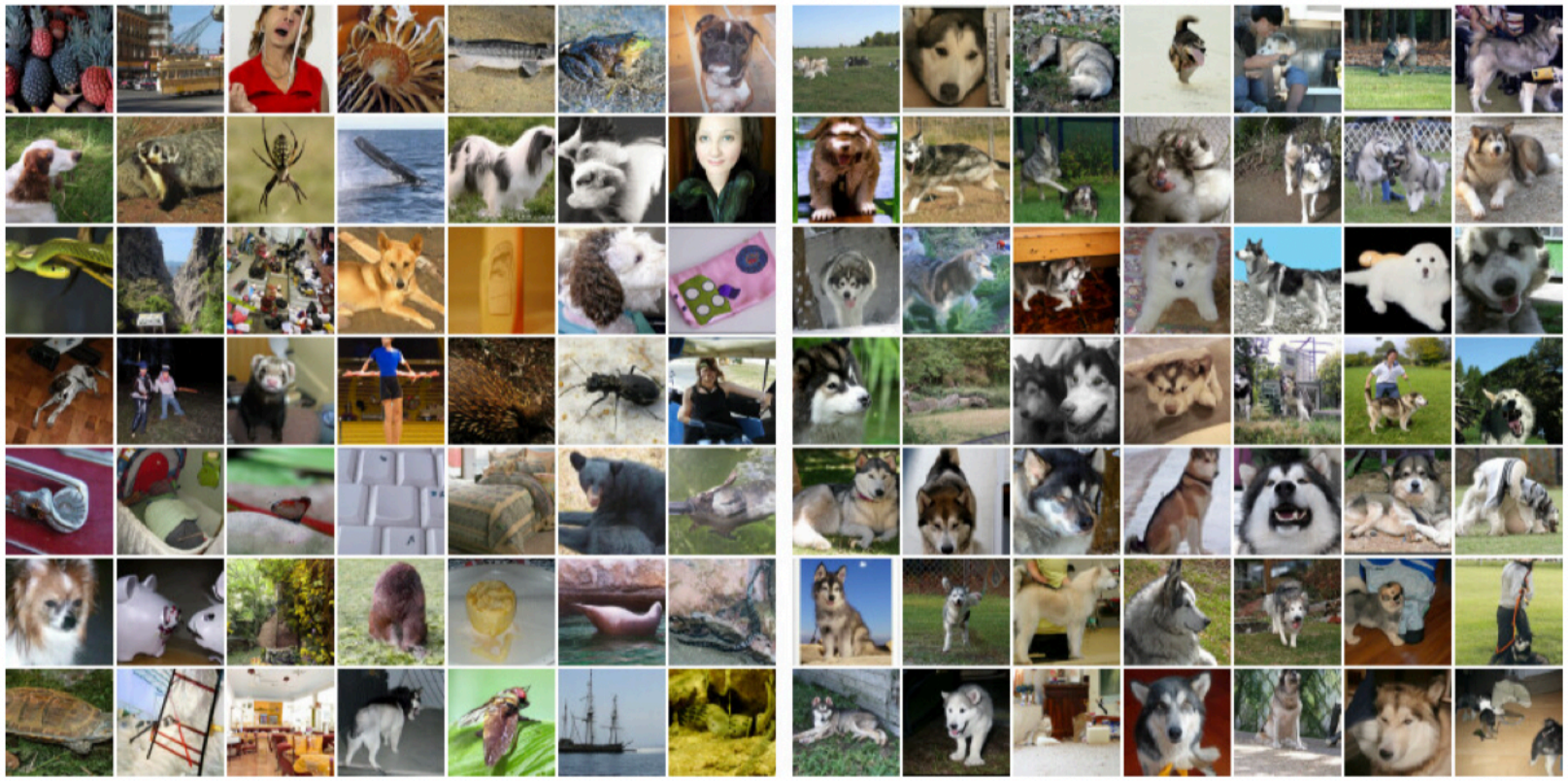


Figure 2: ImageNet 64x64 FID vs. IS

Non-guided

- Low FID (the lower the better) -> High Variability
- Low IS (the higher the better) -> Low Image Quality

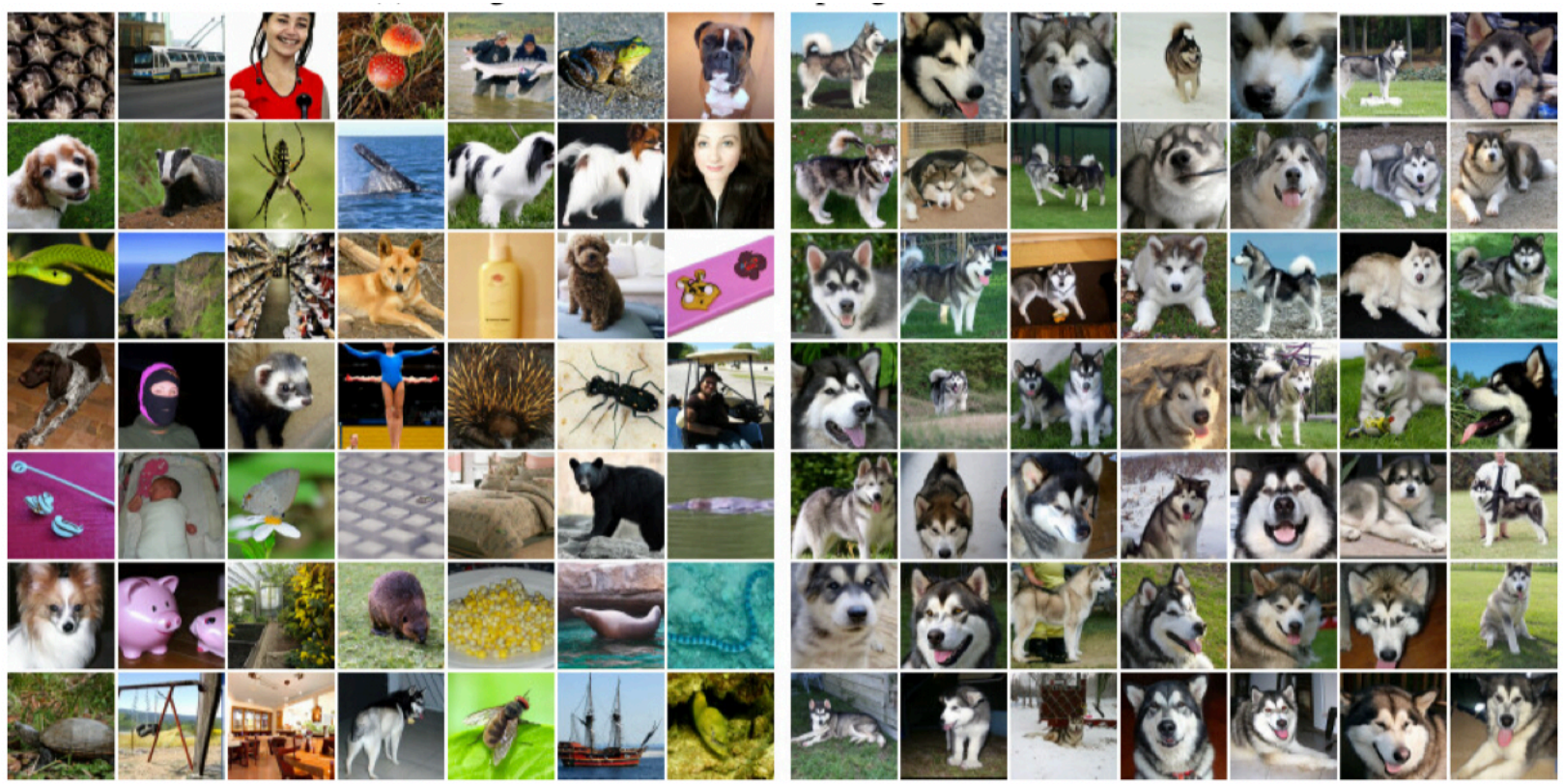


(a) Non-guided conditional sampling: FID=1.80, IS=53.71

CFG with Small Guidance

ROBOT INTELLIGENCE LAB

- Middle FID (the lower the better) -> Middle Variability
- Middle IS (the higher the better) -> Middle Image Quality

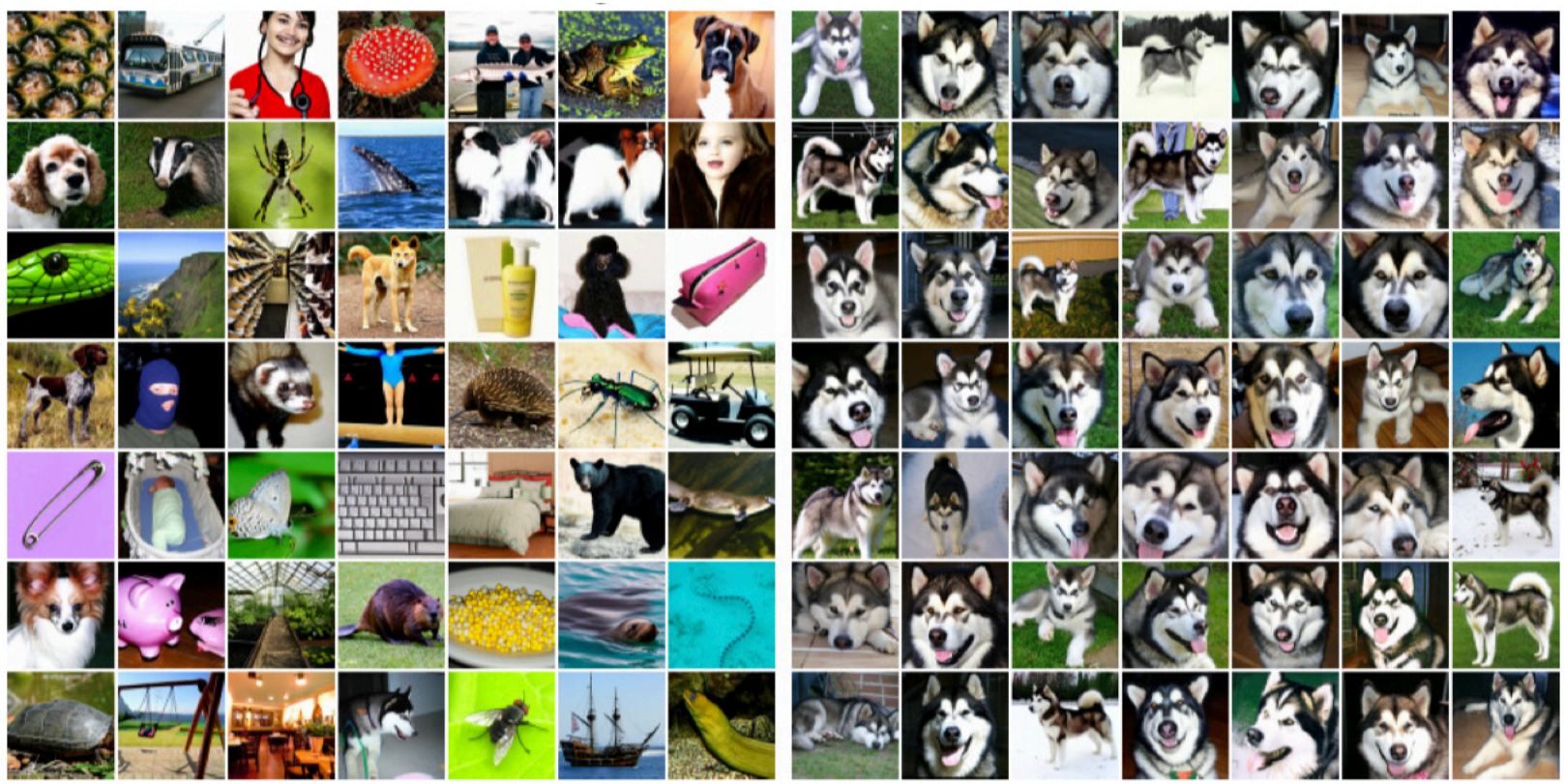


(b) Classifier-free guidance with $w = 1.0$: FID=12.6, IS=170.1

CFG with High Guidance

ROBOT INTELLIGENCE LAB

- High FID (the lower the better) -> Low Variability
- High IS (the higher the better) -> High Image Quality



(c) Classifier-free guidance with $w = 3.0$: FID=24.83, IS=250.4

ROBOT INTELLIGENCE LAB

GLIDE

"GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models," 2022

GLIDE

ROBOT INTELLIGENCE LAB

Guided Language to Image Diffusion for Generation and Editing (GLIDE)

Diffusion version of **DALL-E** (or precursor of **DALL-E2**)

GLIDE

ROBOT INTELLIGENCE LAB

Two different approaches are presented: CLIP guidance and classifier-free guidance
+
Editing capabilities (e.g., image inpainting)

Background

ROBOT INTELLIGENCE LAB

- Diffusion Models
  - Forward diffusion: $q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t} x_{t-1}, (1-\alpha_t)I)$
  - Backward model: $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t), \Sigma_\theta(x_t))$
  - Objective: generate samples $x_t \sim q(x_t \mid x_0)$ by applying Gaussian noise ϵ to x_0, then train a model ϵ_θ to predict the added noise using a standard mean-squared error loss

Background

ROBOT INTELLIGENCE LAB

- Guided Diffusion
  - Perturbed mean: $\hat{\mu}_\theta(x_t \mid y) = \mu_\theta(x_t \mid y) + s \cdot \Sigma_\theta(x_t \mid y) \nabla_{x_t} \log p_\phi(y \mid x_t)$
  - Note that $p_\phi(y \mid x_t)$ is a classifier trained with noisy images and s is the guidance scale
- Classifier-free Guidance
  - Class-conditional diffusion model: $\epsilon_\theta(x_t \mid y)$
  - Extrapolated error prediction: $\hat{\epsilon}_\theta(x_t \mid y) = \epsilon_\theta(x_t \mid \varnothing) + s(\epsilon_\theta(x_t \mid y) - \epsilon_\theta(x_t \mid \varnothing))$ where $s \geq 1$ is the guidance scale
- CLIP Guidance
  - CLIP: $f(x) \cdot g(c)$ is high for matching image x and caption c
  - Perturbed mean: $\hat{\mu}_\theta(x_t \mid c) = \mu_\theta(x_t \mid c) + s \cdot \Sigma_\theta(x_t \mid y) \nabla_{x_t}(f(x_t) \cdot g(c))$
  - Note that the CLIP $f(x_t) \cdot g(c)$ should be trained on noisy images

Text-Conditional Diffusion Models

- ADM model with text conditioning information (CLIP guidance and classifier-free guidance)
  - Visual part: ImageNet 64×64 model of ADM with 512 channels -> 2.3B parameters
  - Text part: Transformer encoder with 24 residual blocks of width 2048 -> 1.2B parameters
  - Upsampler: 64×64 resolution to 256×256 resolution -> 1.5B parameters
- Trained on the same dataset as DALL-E.. ~~(not public)~~

Image Inpainting

ROBOT INTELLIGENCE LAB

- Inpainting with a diffusion model can be done by replacing the known region of the image with a sample from $q(x_t | x_0)$ after each sampling step.
  - It is **NOT** replacing the known region of the image directly.
  - But what about the unknown region of x_0?
  - It has the disadvantage that the model cannot see that entire context but only a noised version sampled from $q(x_t | x_0)$.
- Glide explicitly **fine-tune** the model to preform inpainting.
  - During fine-tuning, random regions of training images are erased, and the remaining portions are fed into to model with a mask channel as additional conditioning information.
  - The input of the model has four additional channels: a second set of RGB channels and a mask channel.

Results

ROBOT INTELLIGENCE LAB

"a hedgehog using a calculator"

"a corgi wearing a red bowtie and a purple party hat"

"robots meditating in a vipassana retreat"

"a fall landscape with a small cottage next to a lake"

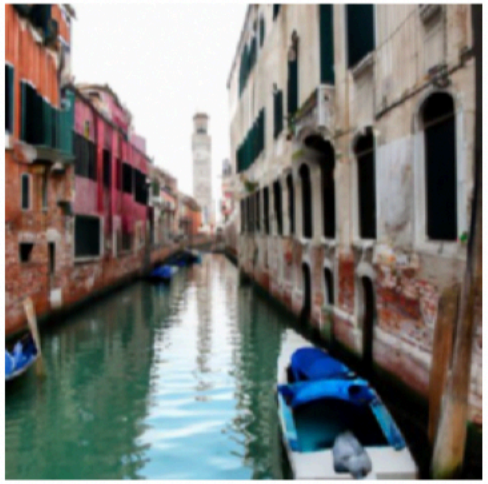
"a surrealist dream-like oil painting by salvador dalí of a cat playing checkers"

"a professional photo of a sunset behind the grand canyon"

"a high-quality oil painting of a psychedelic hamster dragon"

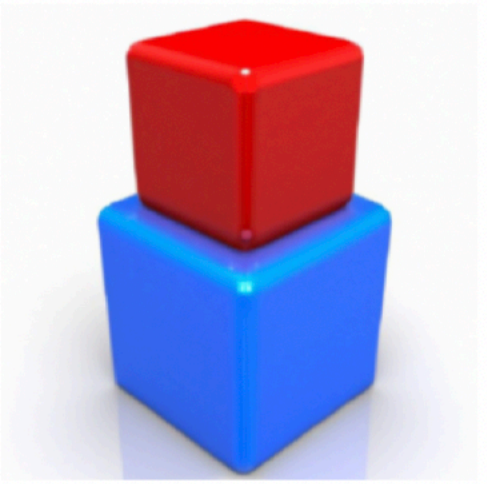
"an illustration of albert einstein wearing a superhero costume"

Results

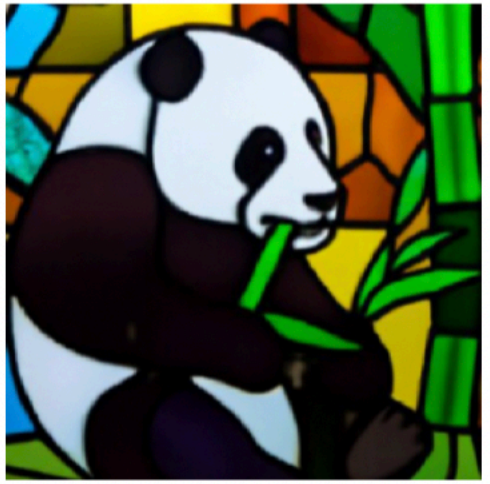


"a boat in the canals of venice"

"a painting of a fox in the style of starry night"



"a red cube on top of a blue cube"



"a stained glass window of a panda eating bamboo"

"a crayon drawing of a space elevator"

"a futuristic city in synthwave style"

"a pixel art corgi pizza"

"a fog rolling into new york"

Results

"a cozy living room"

"a painting of a corgi on the wall above a couch"

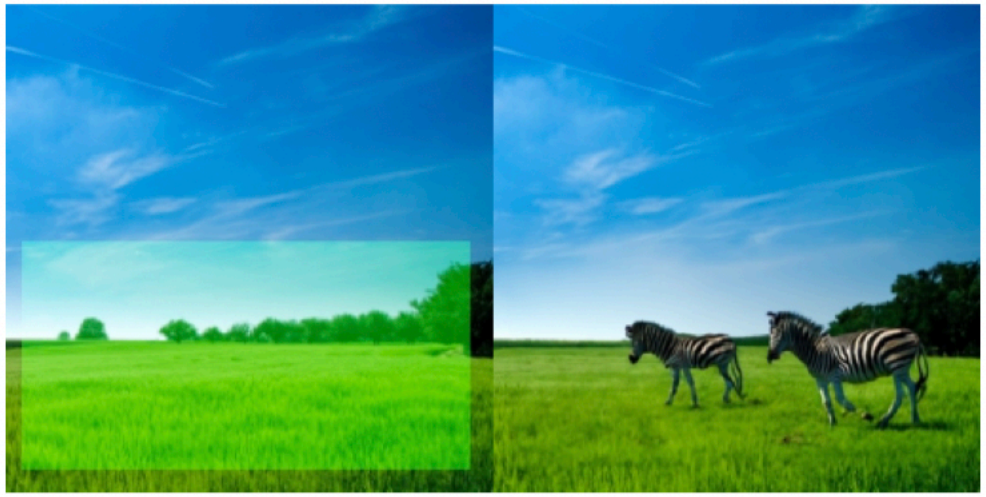
"a round coffee table in front of a couch"

"a vase of flowers on a coffee table"

"a couch in the corner of a room"

Results

ROBOT INTELLIGENCE LAB



"zebras roaming in the field"

"a girl hugging a corgi on a pedestal"

"a man with red hair"

"a vase of flowers"

ROBOT INTELLIGENCE LAB

DALL-E 2

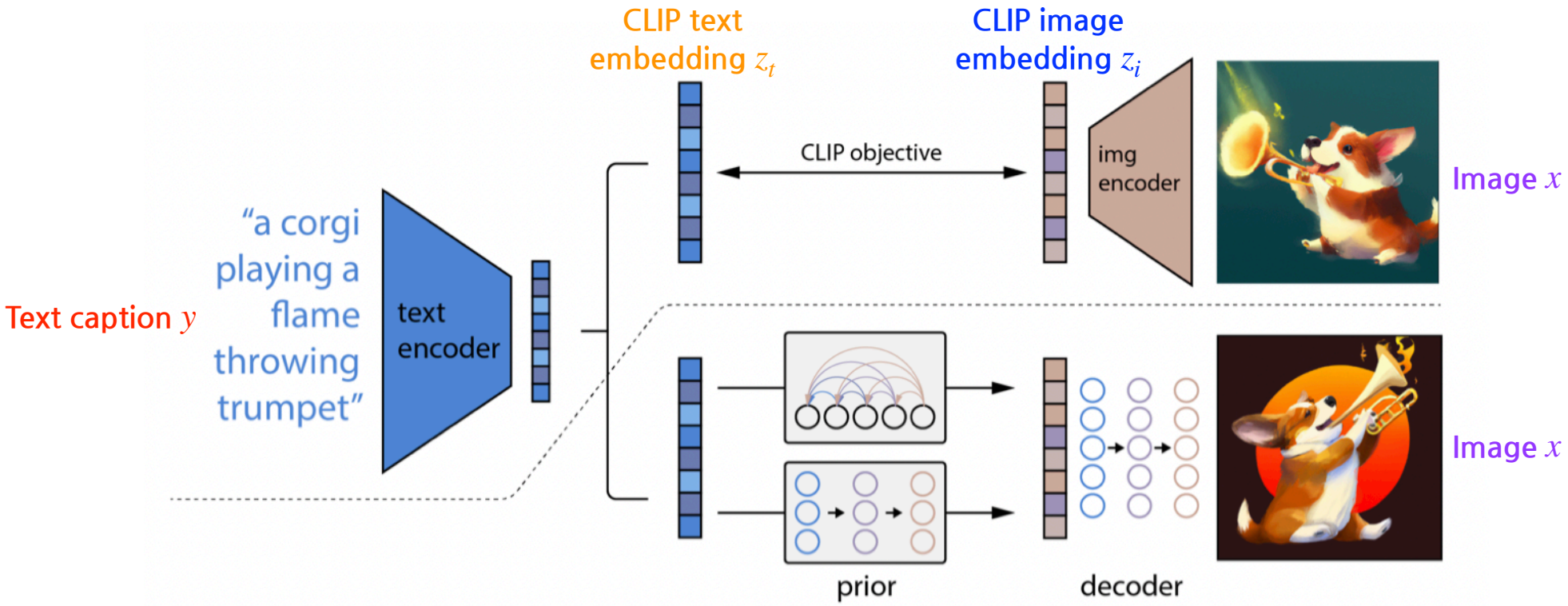
"Hierarchical Text-Conditional Image Generation with CLIP Latents," 2022

unCLIP

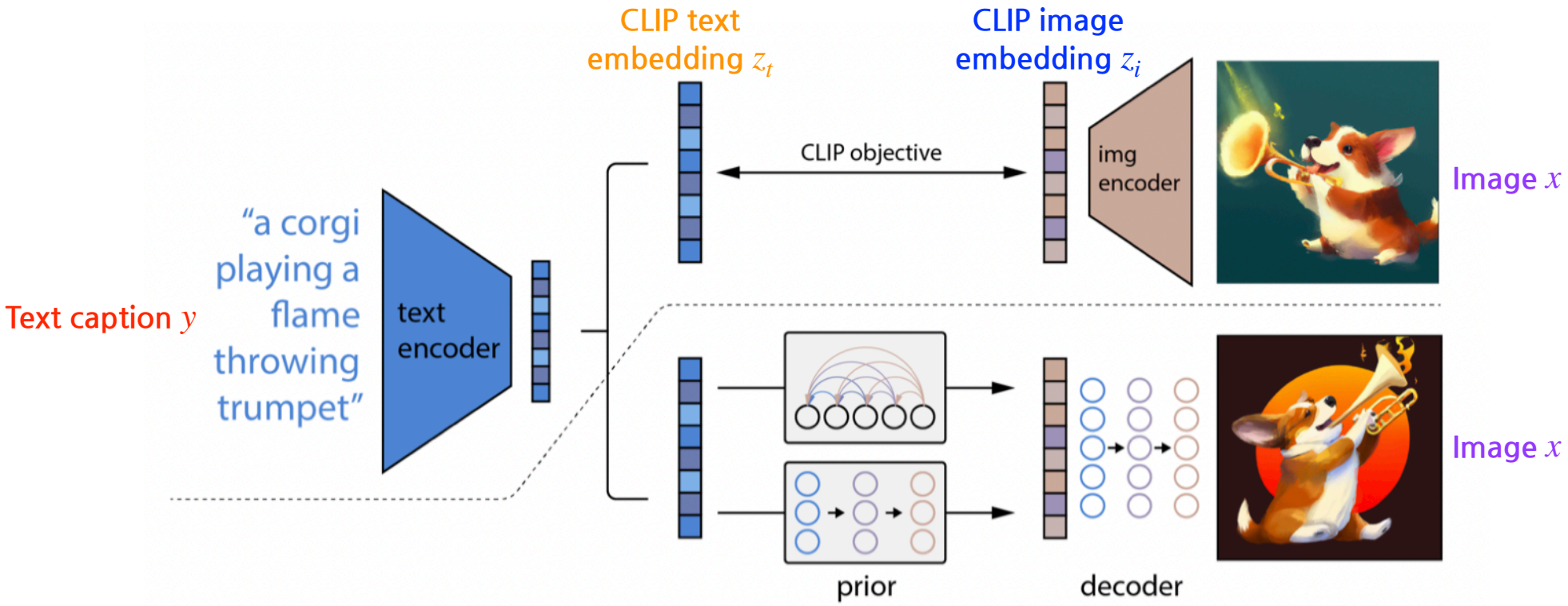
ROBOT INTELLIGENCE LAB

This is the famous **DALL-E 2** paper
(the name of the proposed text-to-image stack is **unCLIP**)

unCLIP

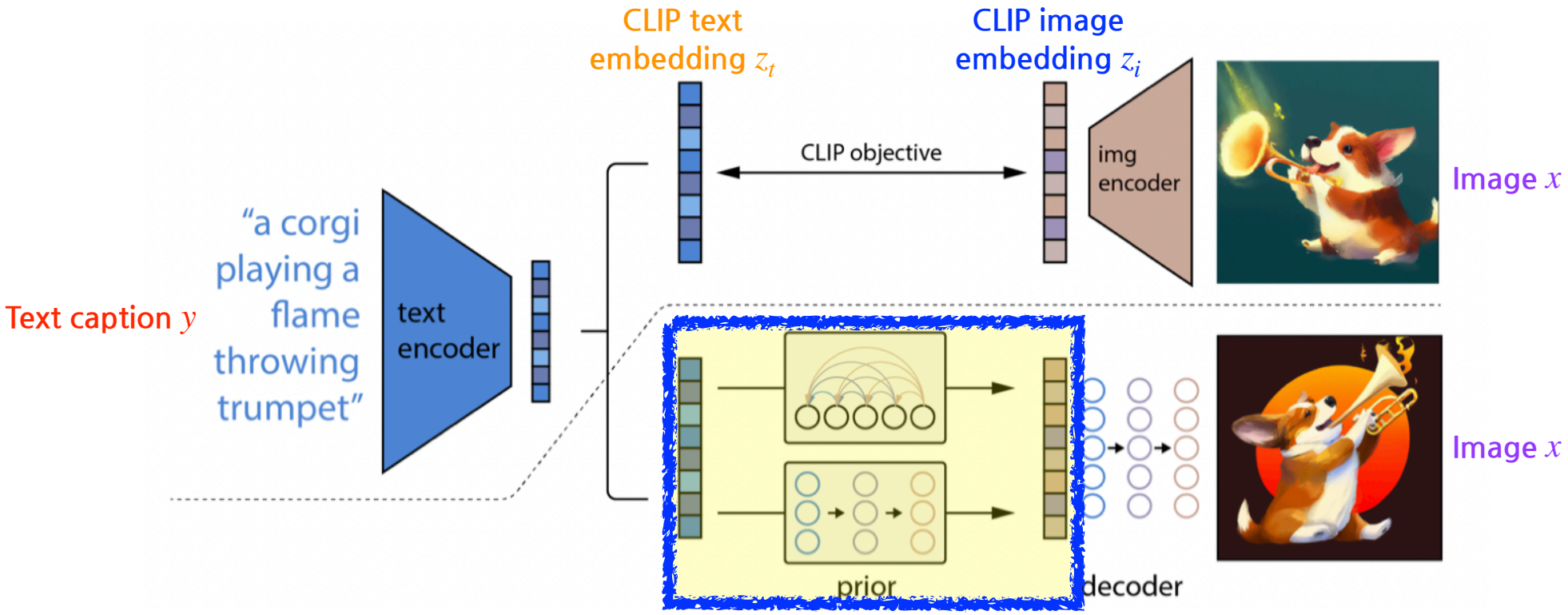

CLIP text embedding z_t
CLIP image embedding z_i
CLIP objective
img encoder
Image x
Text caption y
"a corgi playing a flame throwing trumpet"
text encoder
prior
decoder
Image x

unCLIP



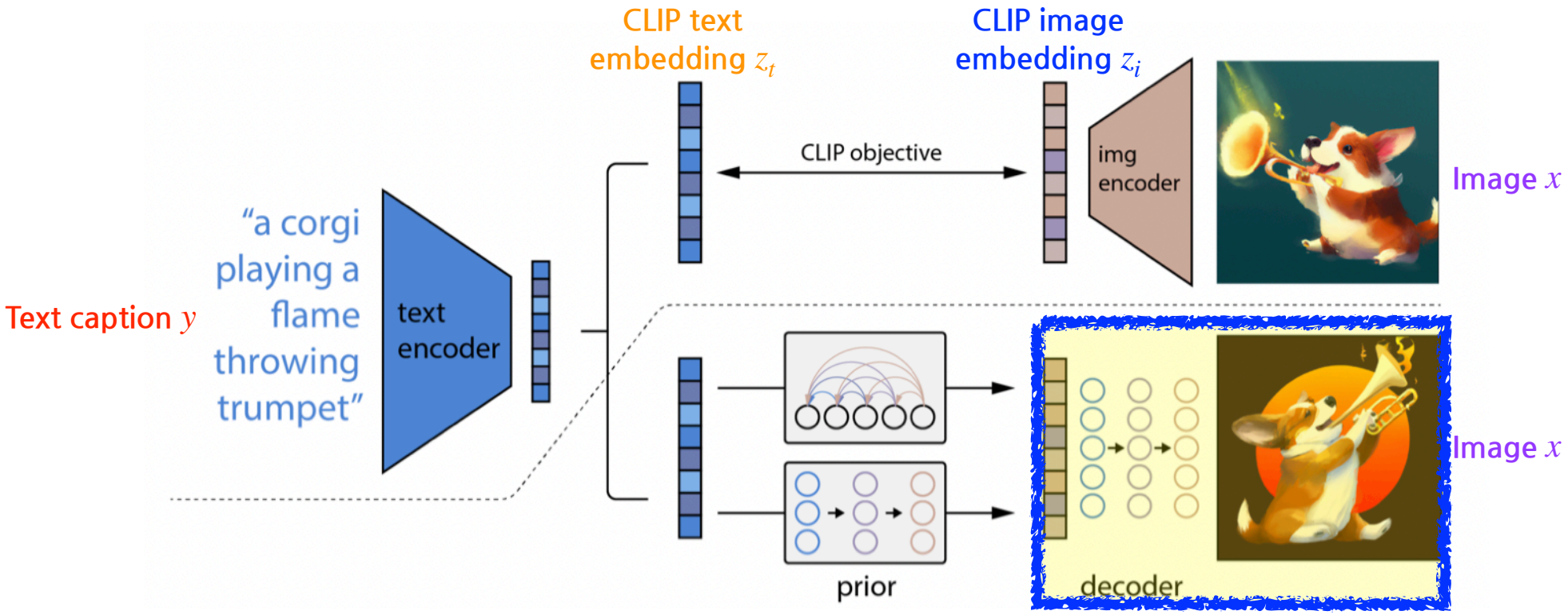
This part is the same as the original CLIP

unCLIP



Prior: CLIP text embedding z_t -> CLIP image embedding z_i

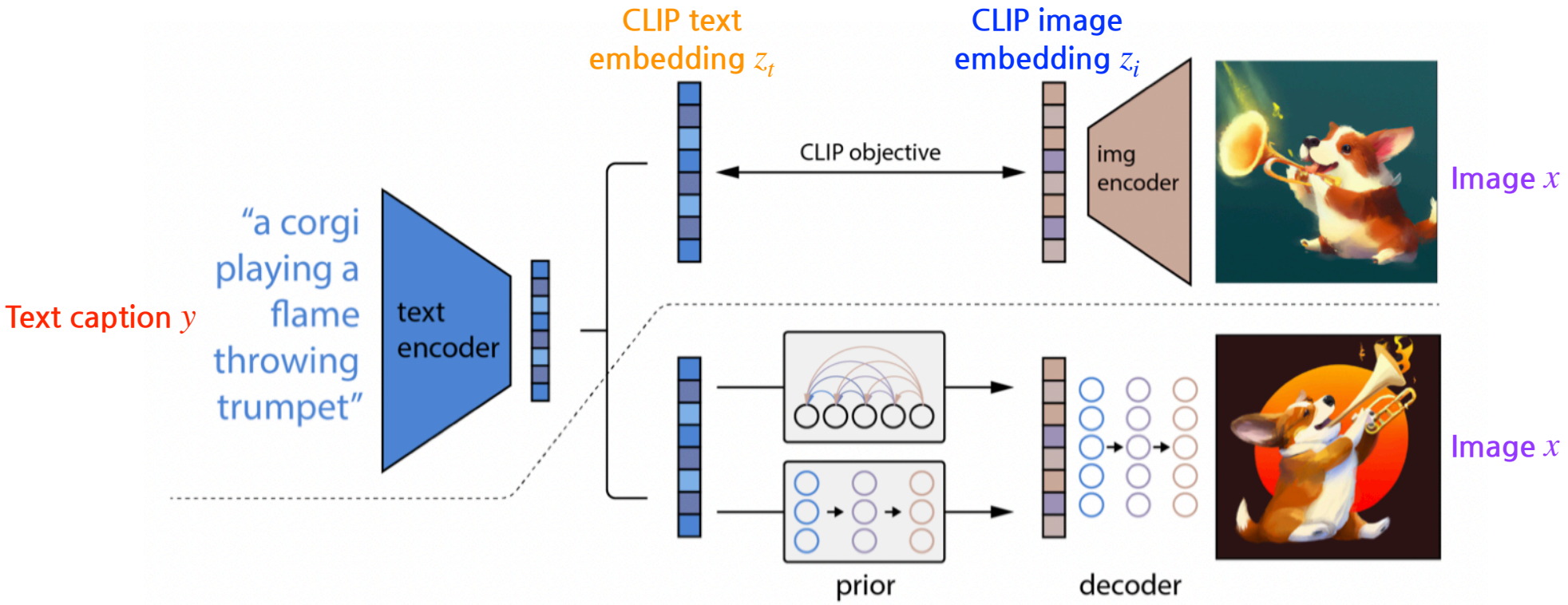
unCLIP



Decoder: CLIP image embedding z_i-> Image x

unCLIP

- unCLIP: text caption => text embedding => image embedding => image



Results

vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

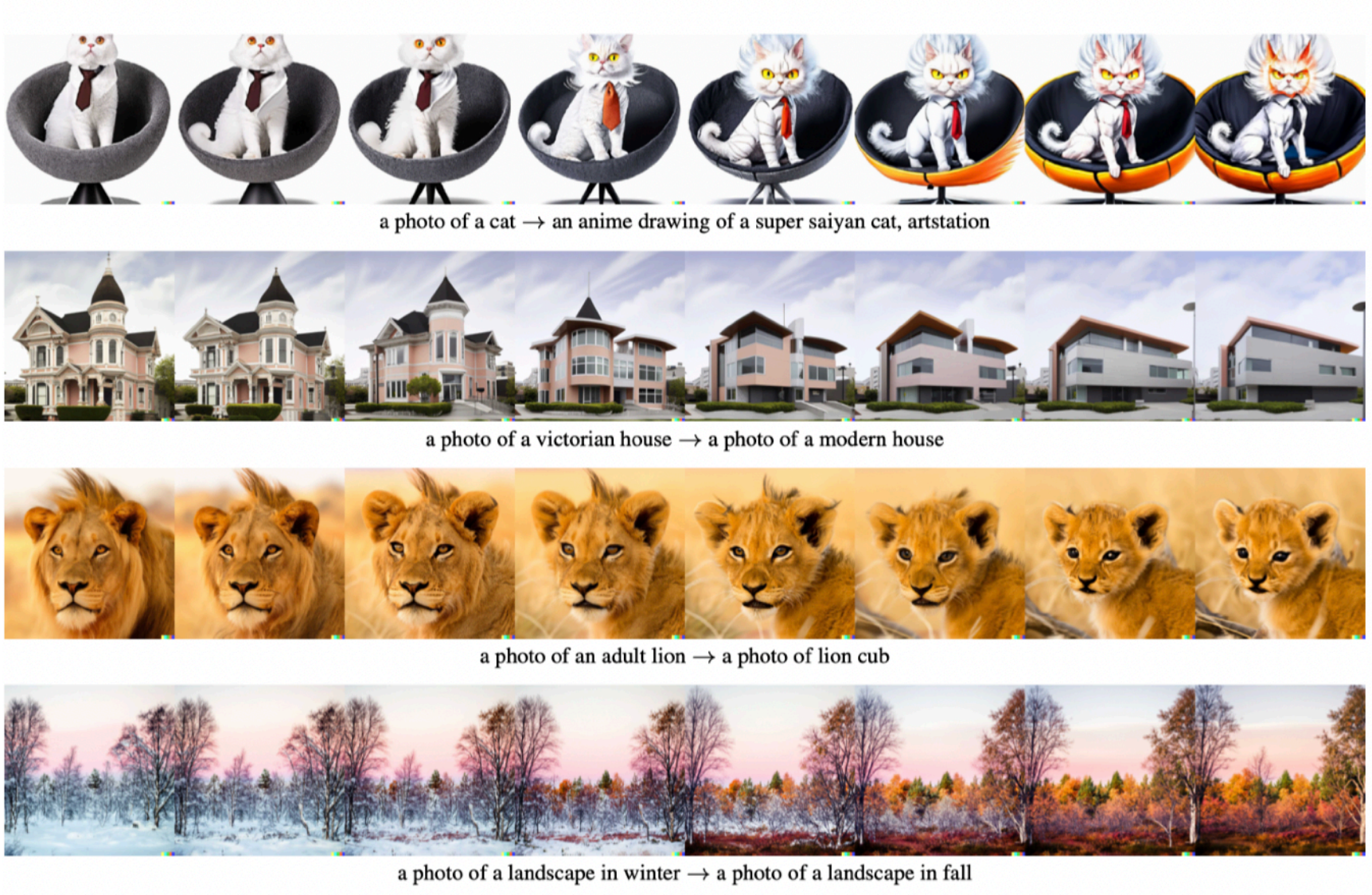
a close up of a handpalm with leaves growing from it

an espresso machine that makes coffee from human souls, artstation

panda mad scientist mixing sparkling chemicals, artstation

a corgi's head depicted as an explosion of a nebula

Interpolation



Failures

Figure 16: Samples from unCLIP for the prompt, “A sign that says deep learning.”

Failures

(a) A high quality photo of a dog playing in a green field next to a lake.

(b) A high quality photo of Times Square.

Figure 17: unCLIP samples show low levels of detail for some complex scenes.

ROBOT INTELLIGENCE LAB