



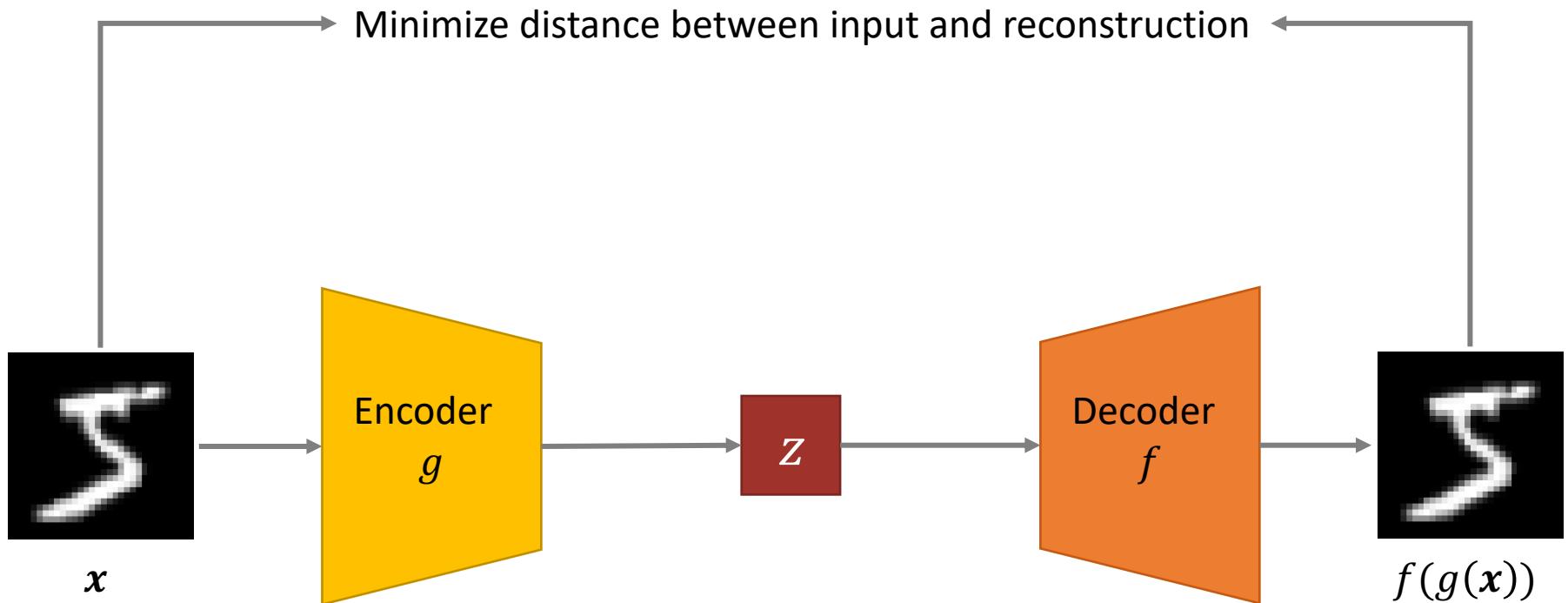
GENERATIVE MODELS

Deep Learning for Computer Vision

Arthur Douillard

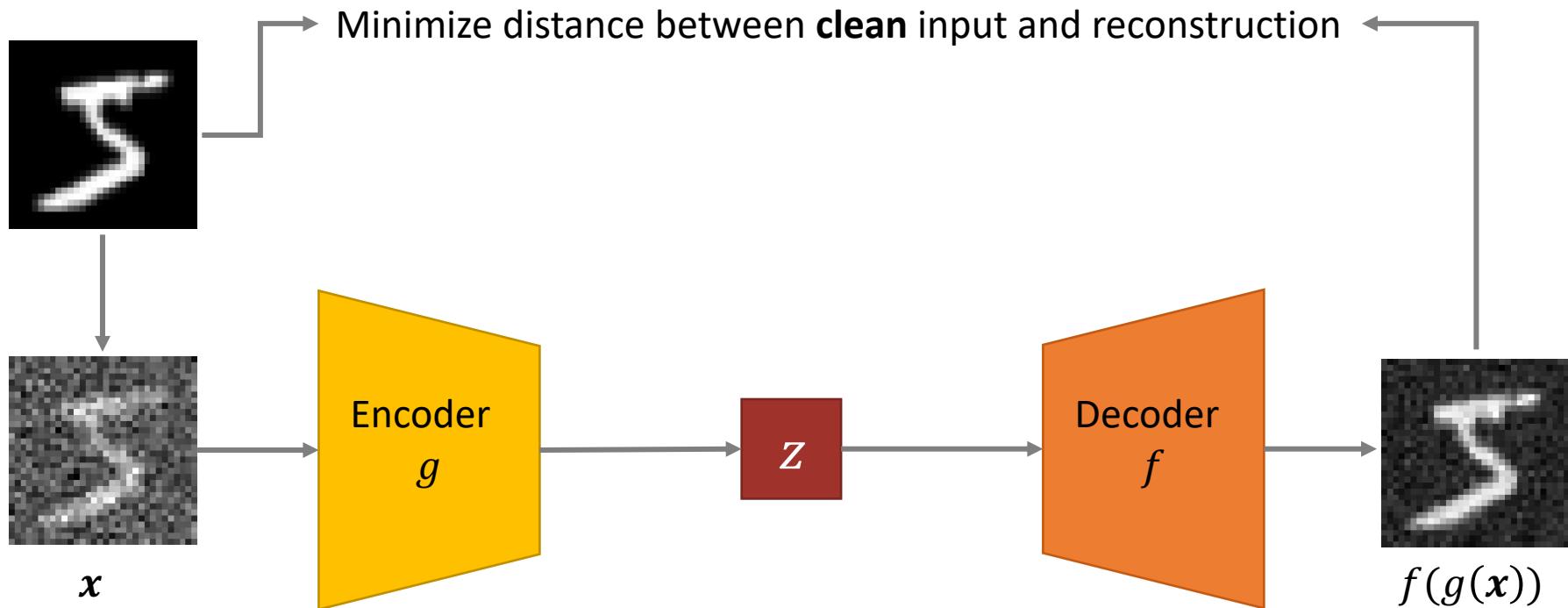
Auto-Encoder

(Under-Complete) Auto-Encoder



- The smaller is the bottleneck z , fewer important features are kept
- Similar to other methods of dimensionality reduction like PCA

Denoising Auto-Encoder



- Add noise to input, try to **denoise it** by reconstruction with clean input

Variational Auto-Encoder

Variational Auto-Encoder



Instead of producing a deterministic latent code z , can we generate a **distribution**?
→ Generate new images, not simply a reconstruction, but sampling from it

Could we force that the latent code's dimensions are **disentangled**?
→ Modify only an aspect of the image (e.g. keep the face but make hair blond)

Variational Auto-Encoder



Given the:

- Prior $p(\mathbf{z})$
- Likelihood $p(\mathbf{x} | \mathbf{z})$
- Posterior $p(\mathbf{z} | \mathbf{x})$
- Evidence $p(\mathbf{x})$

We want to estimate the **posterior**, aka what should be our latent code given \mathbf{x} .

By Bayes and then multiplication rule, we have:

$$p(\mathbf{z} | \mathbf{x}) = \frac{p(\mathbf{x} | \mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} = \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})}$$

Problem: the evidence $p(\mathbf{x})$ is **untractable** (aka it's hard to compute)

Thus, using **variational inference** we are going to approximate the posterior $p(\mathbf{z} | \mathbf{x})$ by a distribution $q(\mathbf{z})$ that we defined to be tractable.

Our goal is to minimize the divergence between them:

$$\min KL(q(\mathbf{z}) \| p(\mathbf{z} | \mathbf{x})) = - \sum q(\mathbf{z}) \log \frac{p(\mathbf{z} | \mathbf{x})}{q(\mathbf{z})}$$

Variational Auto-Encoder



$$\begin{aligned} KL(q(z) \| p(z|x)) &= - \sum q(z) \log \frac{p(z|x)}{q(z)} \\ &= - \sum q(z) \log \frac{\frac{p(x,z)}{p(x)}}{\frac{q(z)}{1}} \end{aligned}$$

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

Variational Auto-Encoder



$$\begin{aligned} KL(q(z)\|p(z|x)) &= - \sum q(z) \log \frac{p(z|x)}{q(z)} \\ &= - \sum q(z) \log \frac{\frac{p(x,z)}{p(x)}}{\frac{q(z)}{1}} \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} \cdot \frac{1}{p(x)} \\ &= - \sum q(z) [\log \frac{p(x,z)}{q(z)} - \log p(x)] \end{aligned}$$

By the properties of log

Variational Auto-Encoder



$$\begin{aligned} KL(q(z)\|p(z|x)) &= - \sum q(z) \log \frac{p(z|x)}{q(z)} \\ &= - \sum q(z) \log \frac{\frac{p(x,z)}{p(x)}}{\frac{q(z)}{1}} \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} \cdot \frac{1}{p(x)} \\ &= - \sum q(z) [\log \frac{p(x,z)}{q(z)} - \log p(x)] \quad \text{By the properties of log} \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} + \sum q(z) \log p(x) \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} + \log p(x) \sum q(z) \end{aligned}$$

Because we integrate/sum over \mathbf{z} and not x

Variational Auto-Encoder



$$\begin{aligned} KL(q(z)\|p(z|x)) &= - \sum q(z) \log \frac{p(z|x)}{q(z)} \\ &= - \sum q(z) \log \frac{\frac{p(x,z)}{p(x)}}{\frac{q(z)}{1}} \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} \cdot \frac{1}{p(x)} \\ &= - \sum q(z) [\log \frac{p(x,z)}{q(z)} - \log p(x)] \quad \text{By the properties of log} \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} + \sum q(z) \log p(x) \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} + \log p(x) \sum q(z) \\ &= - \sum q(z) \log \frac{p(x,z)}{q(z)} + \log p(x) \quad \text{Because a distribution sum to 1} \end{aligned}$$



Variational Auto-Encoder

Re-ordering the term of the equation, we have:

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

x is a constant in our case (we are training on a fixed dataset), the log of a probability is ≤ 0 , and the KL is always ≥ 0 . Therefore if we maximize the **Variational Lower Bound** $\mathcal{L} = \sum q(z) \log \frac{p(x, z)}{q(z)}$, we will minimize the KL as intended:

$$\begin{aligned}\mathcal{L} &= \sum q(z) \log \frac{p(x, z)}{q(z)} \\ &= \sum q(z) \log \frac{p(x|z)p(z)}{q(z)} \\ &= \sum q(z) [\log p(x|z) \log \frac{p(z)}{q(z)}] \\ &= \sum q(z) \log p(x|z) + \sum q(z) \log \frac{p(z)}{q(z)} \\ &= \sum q(z) \log p(x|z) - KL(q(z)\|p(z)) \\ &= E_{q(z)} \log p(x|z) - KL(q(z)\|p(z))\end{aligned}$$



Variational Auto-Encoder

Thus our variational lower bound is made of two terms:

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

Reconstruction error:

$E_{q(z)} \log p(x|z) \propto E_{q(z)} \log p(x|\hat{x})$ because the decoder is deterministic. If we choose a tractable distribution for $p(\cdot)$ such as the Gaussian distribution, our conditional probability will look like:

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

And its log by:

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

Which is the **Mean Squared Error**, aka can our model reconstruct correctly the input.

KL Divergence:

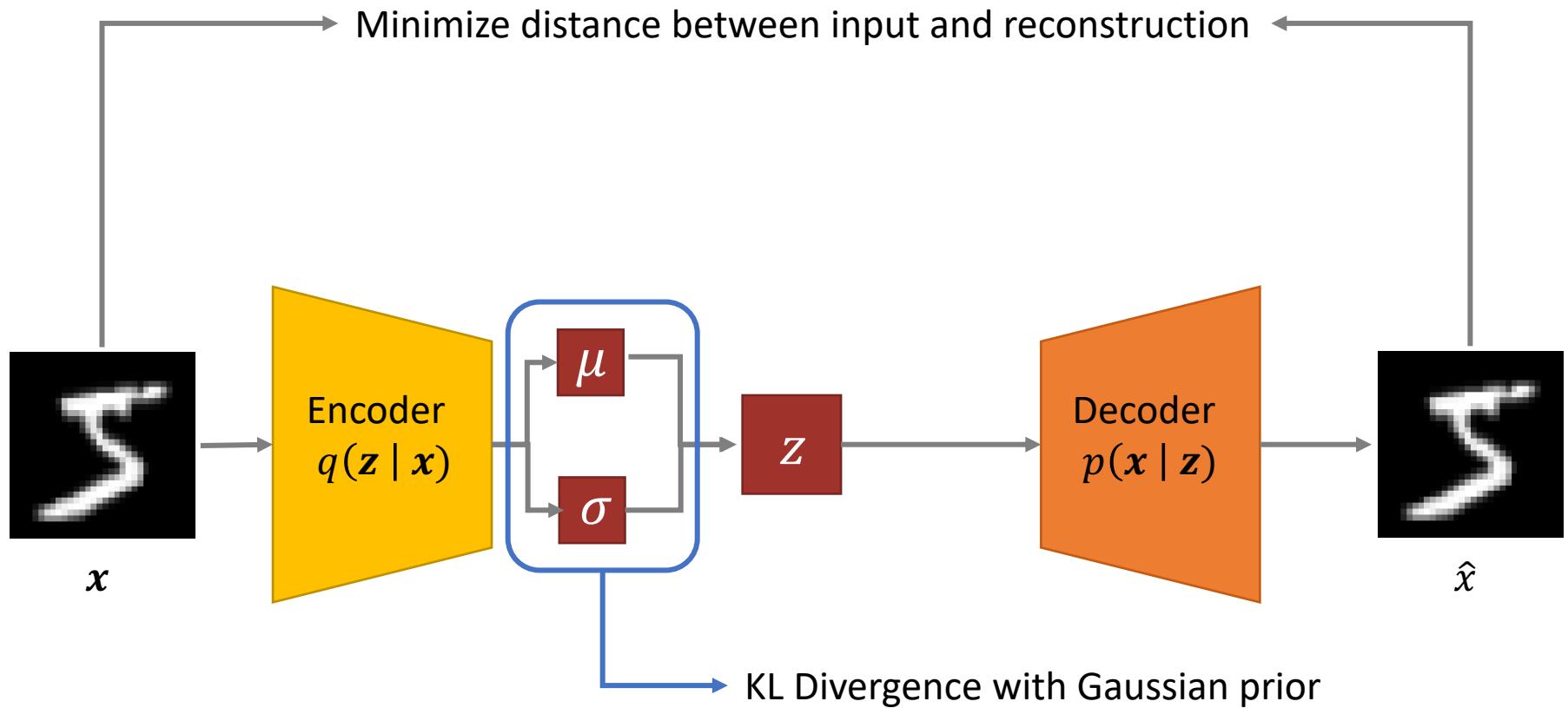
The right part says that our network-distribution $q(z)$ must match the distribution $p(z)$. Now again, we choose $p(\cdot)$ to follow the Gaussian distribution (with zero mean and unit variance $\mathcal{N}(0, I)$).

Now, we are never going to generate z directly by the encoder (it won't be a distribution), but we are going to generate the parameters of the distribution $q(\cdot)$ assuming it's gaussian.

So our KL will be:

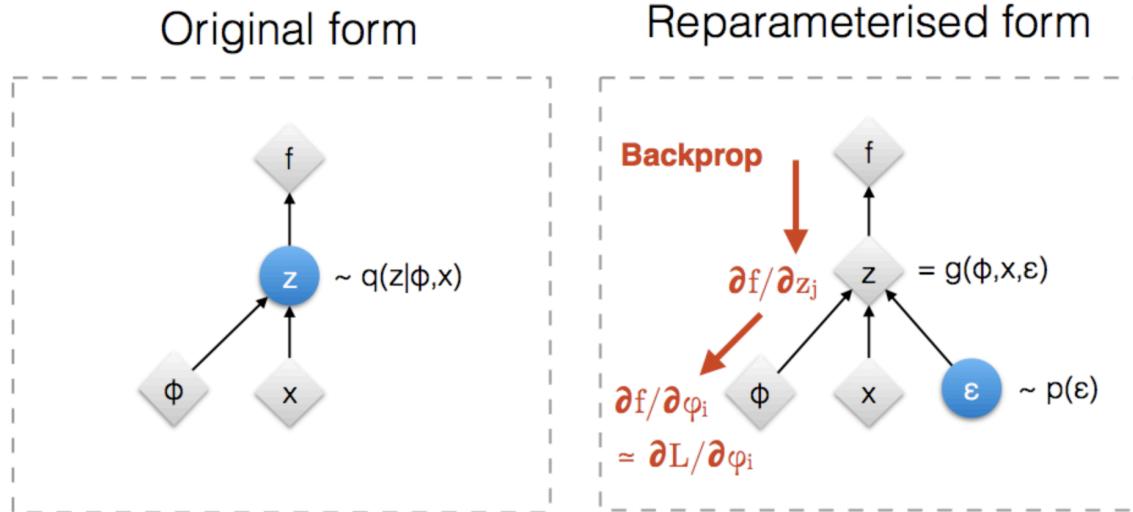
$$KL(\mathcal{N}(\mu, \Sigma)\|\mathcal{N}(0, I))$$

Variational Auto-Encoder



The KL Divergence disentangles the latent code by forcing a unique mode per dimension!

Reparametrization Trick



: Deterministic node



: Random node

[Kingma, 2013]

[Bengio, 2013]

[Kingma and Welling 2014]

[Rezende et al 2014]

- Sampling operation cannot be backpropagated
- Thus sample a random variable ϵ and multiply the predicted variance σ then add to predicted mean μ

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \times \boldsymbol{\sigma}$$



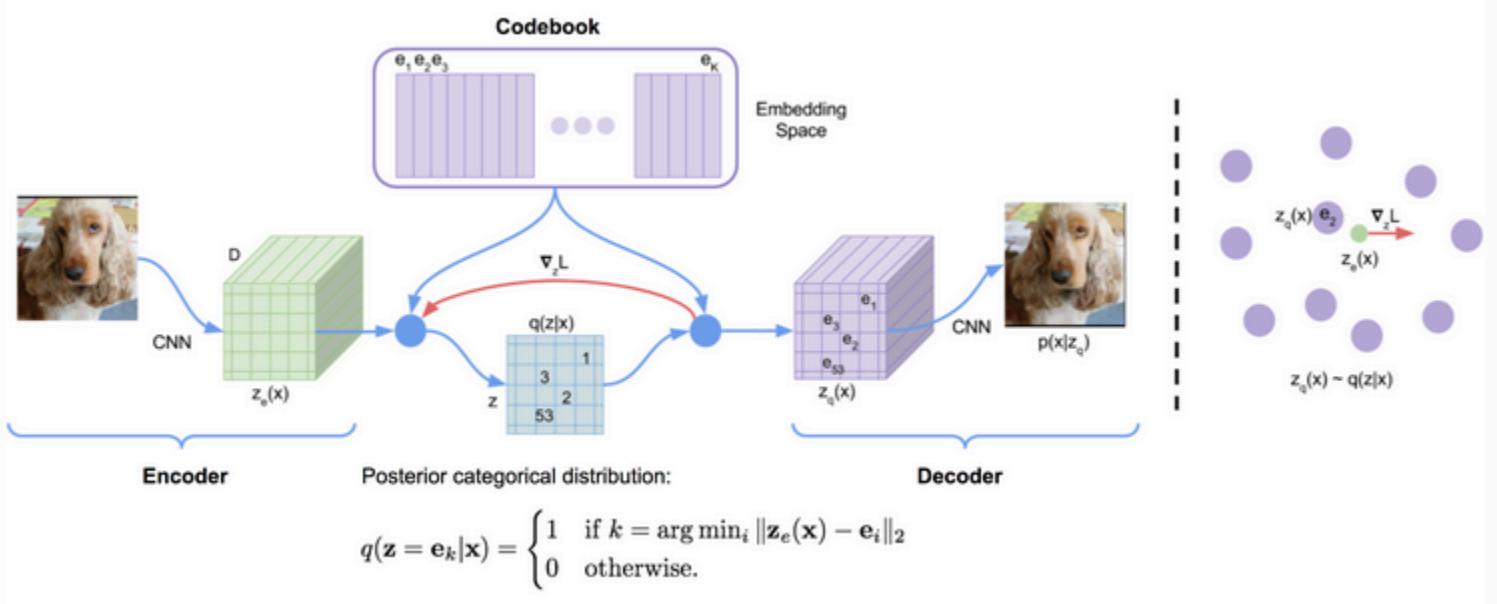
$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

Add a β factor to the KL divergence.

- High factor means better disentangling
- Low factor means better reconstruction

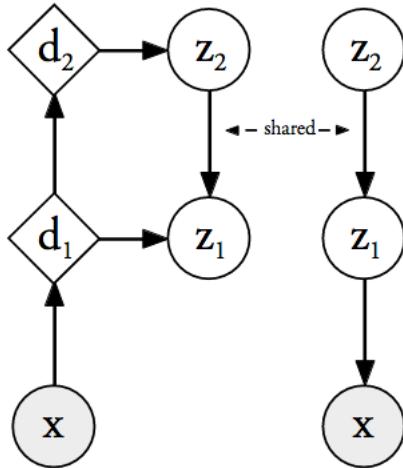
Trade-off to be made between both.

Later work propose to start with $\beta = 0$ and to increases it linearly



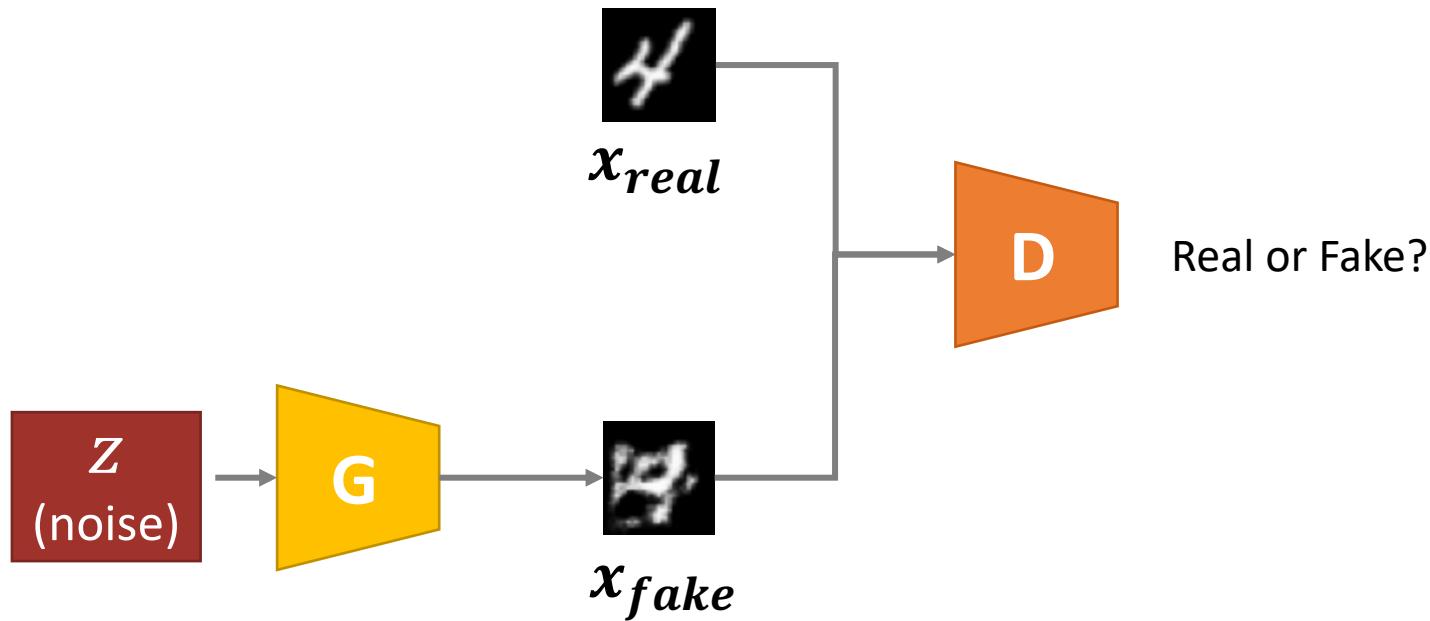
- Generate a latent code that is matched to a discrete space
- Avoid **posterior collapse** where the decoder mostly ignore the latent code
 - The sampled latent code is too weak or noisy
 - Decoder simply generates a new image from noise
- Argmin is not differentiable thus gradient is copied (similarly to STE in quantization)

Hierarchical VAE



- Increase expressiveness of the model by partitioning the latent variables into disjoint groups
- z_1 is conditioned by z_2

Generative Adversarial Networks



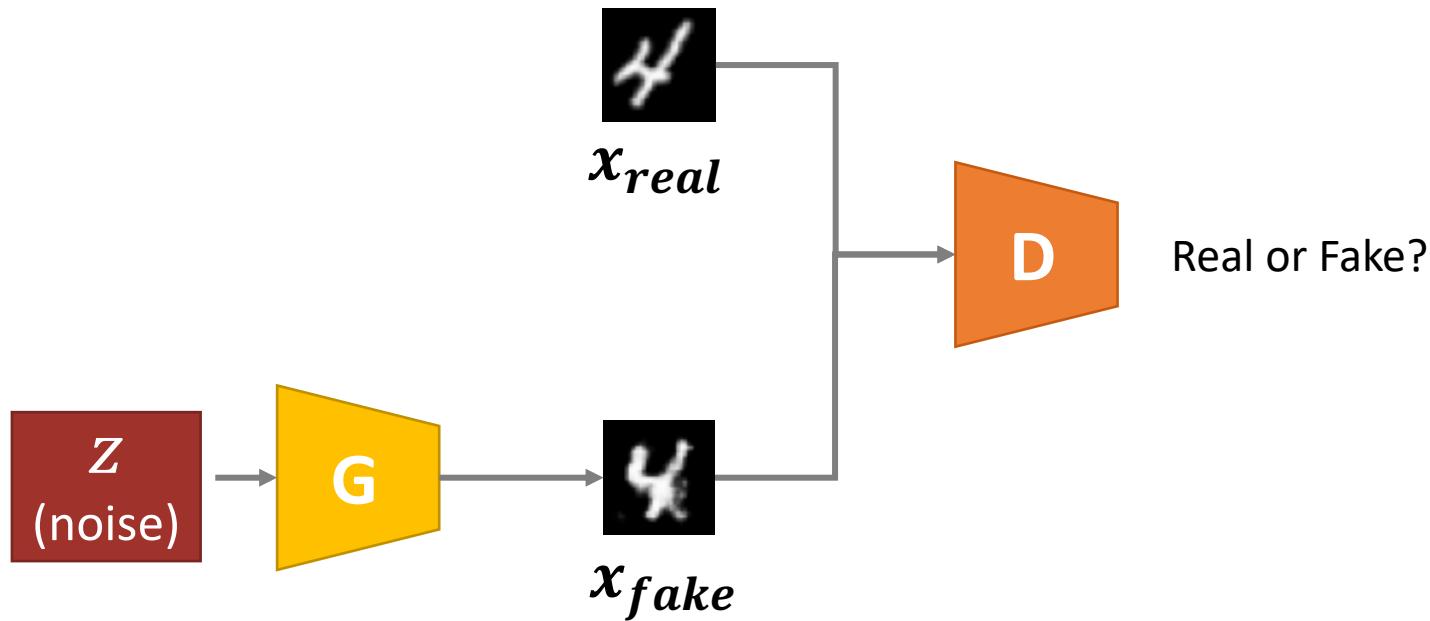
Adversarial training between a **discriminator** and a **generator**.

Discriminator has to distinguish between **real and fake images**.

$$\max_D \mathbb{E}_{\mathbf{x}^* \in \mathcal{D}_{\text{Data}}} [\log D(\mathbf{x}^*)] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))]$$

Generator must fool the discriminator.

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



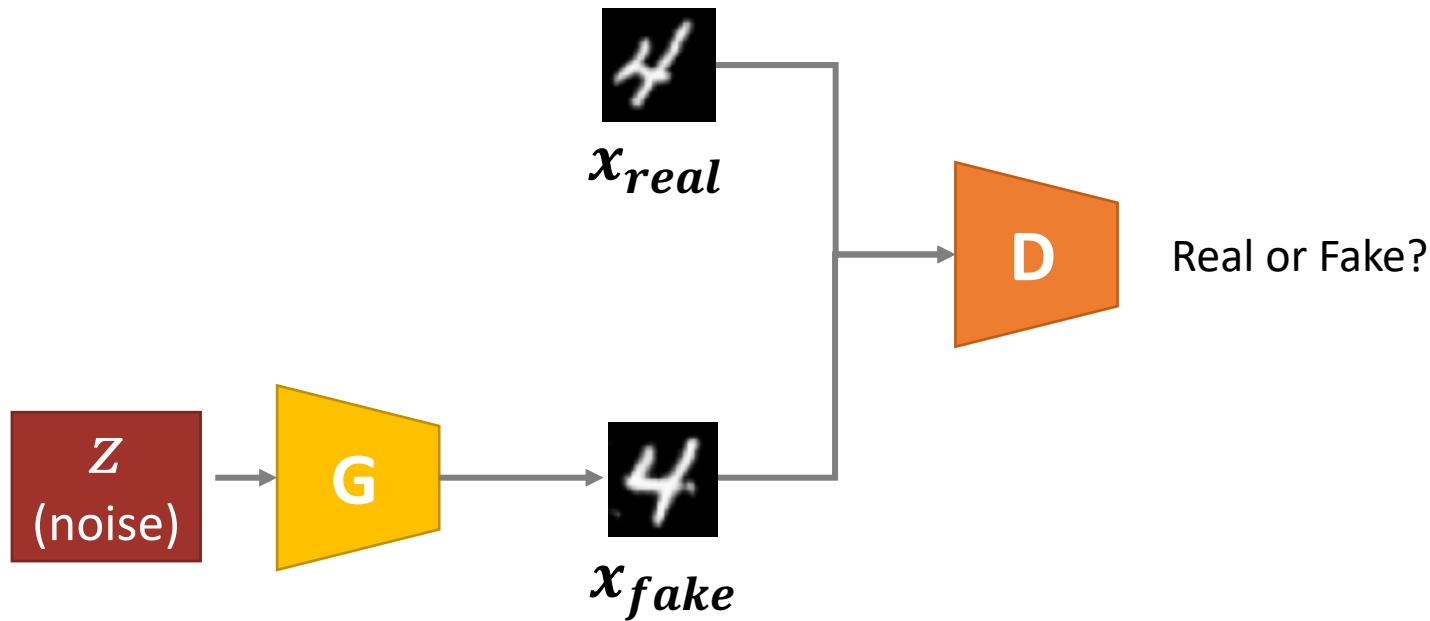
Adversarial training between a **discriminator** and a **generator**.

Discriminator has to distinguish between **real and fake images**.

$$\max_D \mathbb{E}_{\mathbf{x}^* \in \mathcal{D}_{\text{Data}}} [\log D(\mathbf{x}^*)] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))]$$

Generator must fool the discriminator.

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



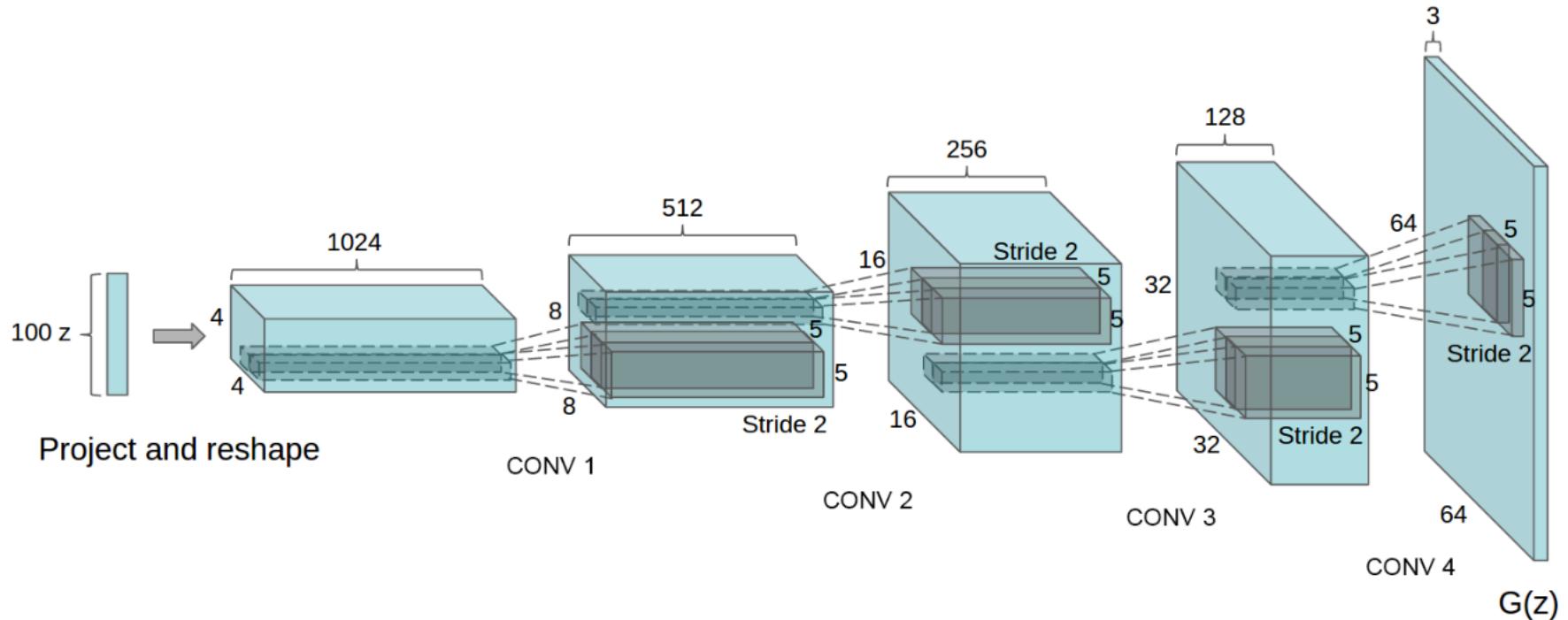
Adversarial training between a **discriminator** and a **generator**.

Discriminator has to distinguish between **real and fake images**.

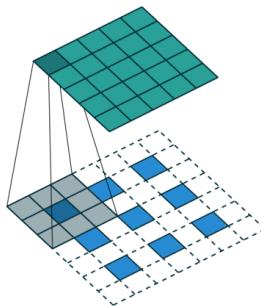
$$\max_D \mathbb{E}_{\mathbf{x}^* \in \mathcal{D}_{\text{Data}}} [\log D(\mathbf{x}^*)] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))]$$

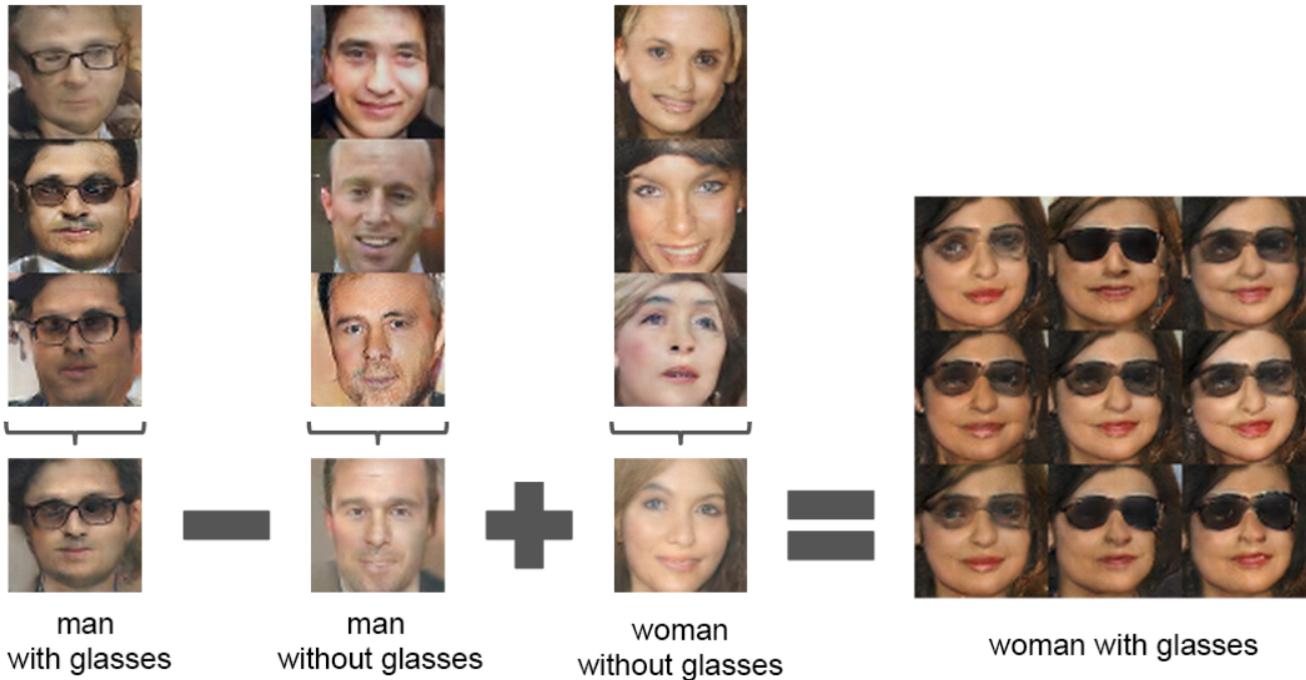
Generator must fool the discriminator.

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



- Use convolutions instead of FC layers
- Upsample using **transposed convolutions**

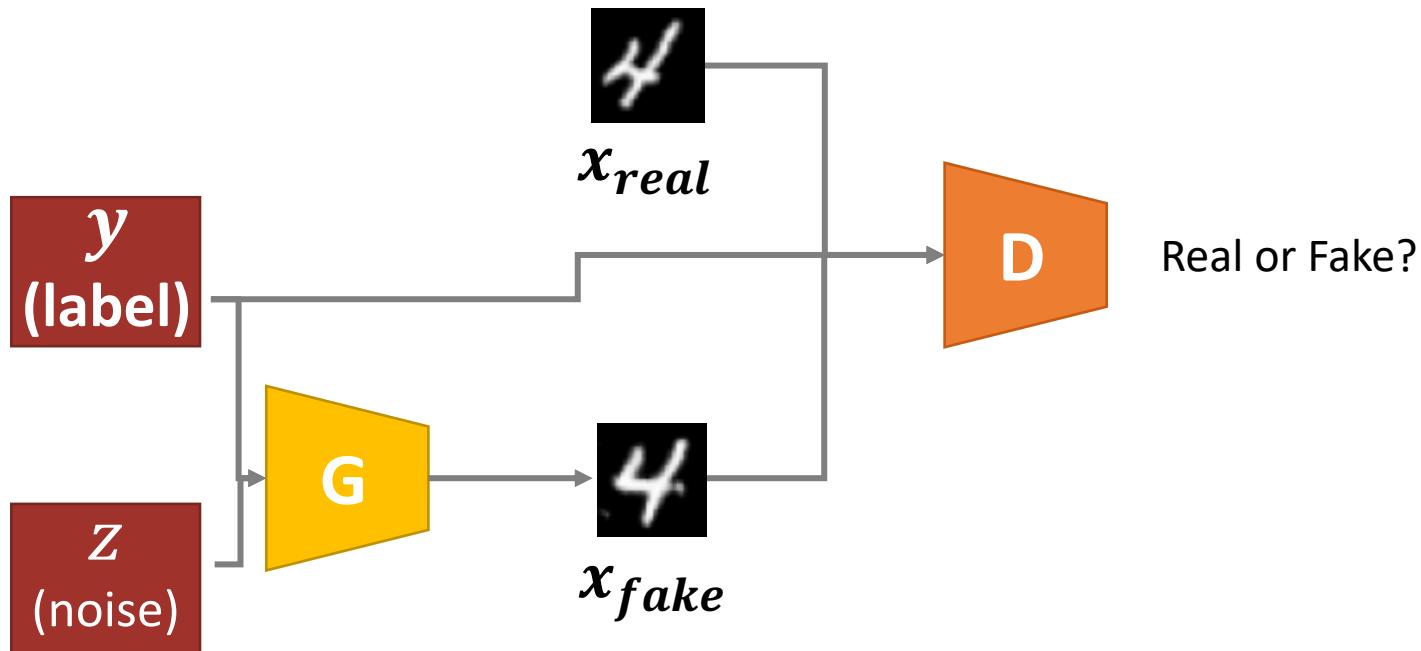




Vector arithmetic for visual concepts:

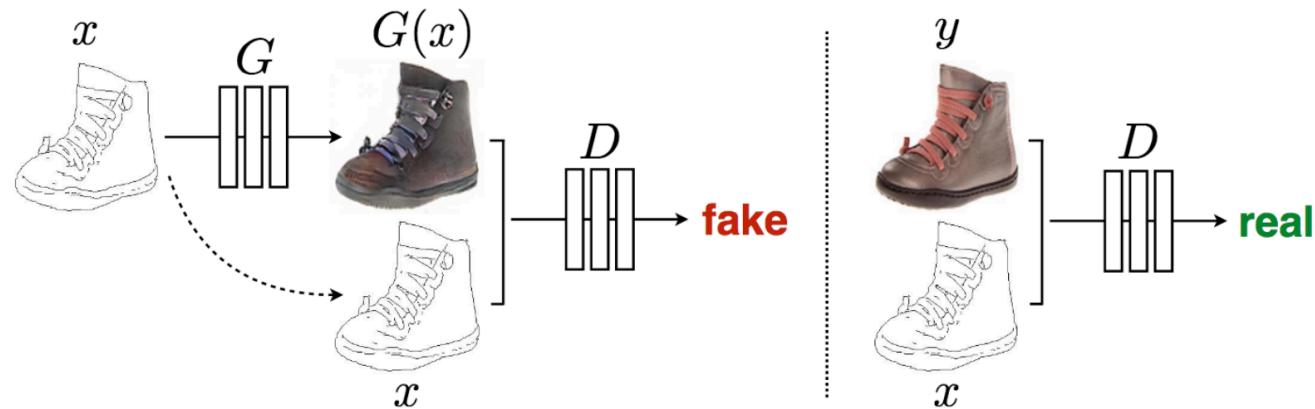
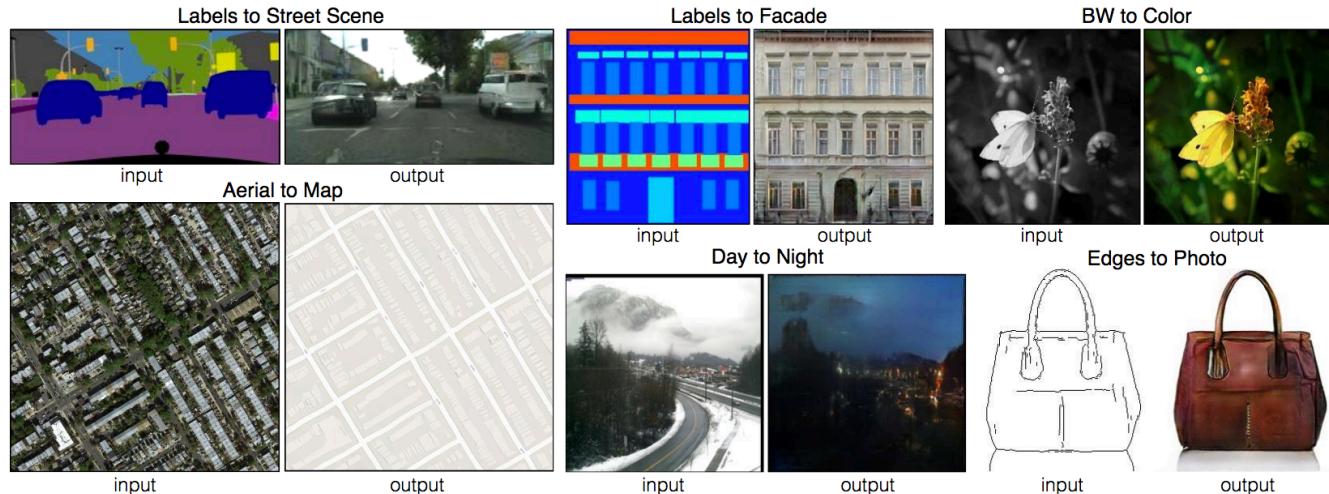
1. Find many noise vectors that produce man with glasses, man w/o glasses, etc.
2. Average noise vectors per category
3. Do some basic arithmetic with the noise vectors
4. Generate!

cGAN: Conditional-GAN



- Add label in input to both the generator and discriminator
- Now the generator, given label “4” will not tolerate a “5” even if it’s very realistic

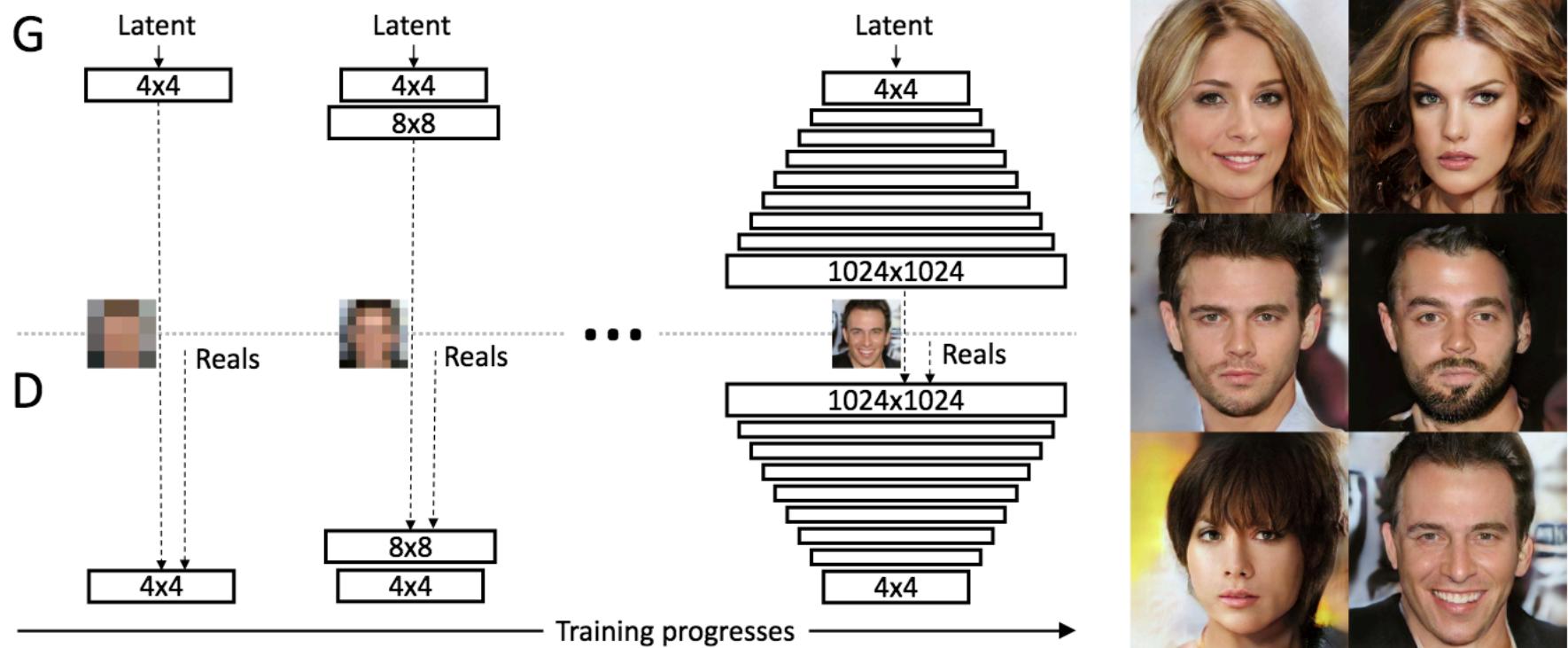
Pix2Pix



- Like cGAN but conditioned with various kind of data (segmentation, maps, drawing, etc.)

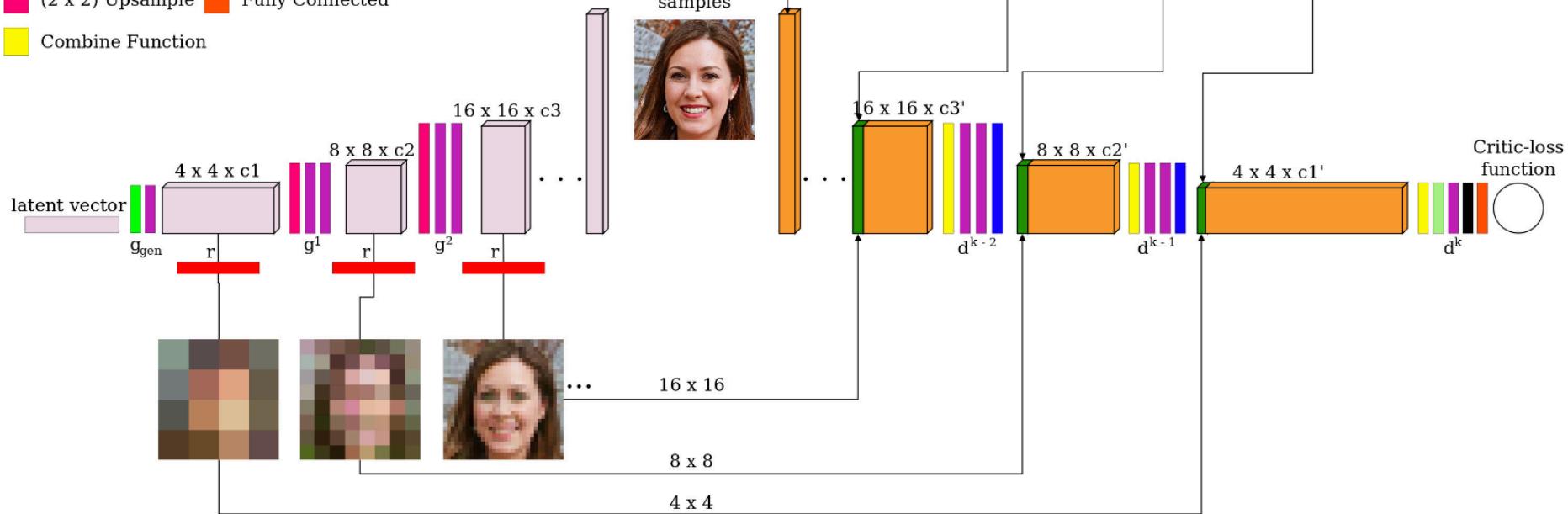
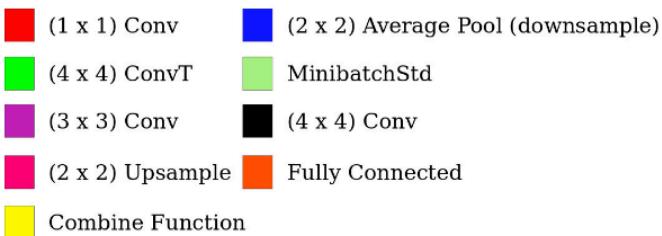


ProGAN: Progressive growing



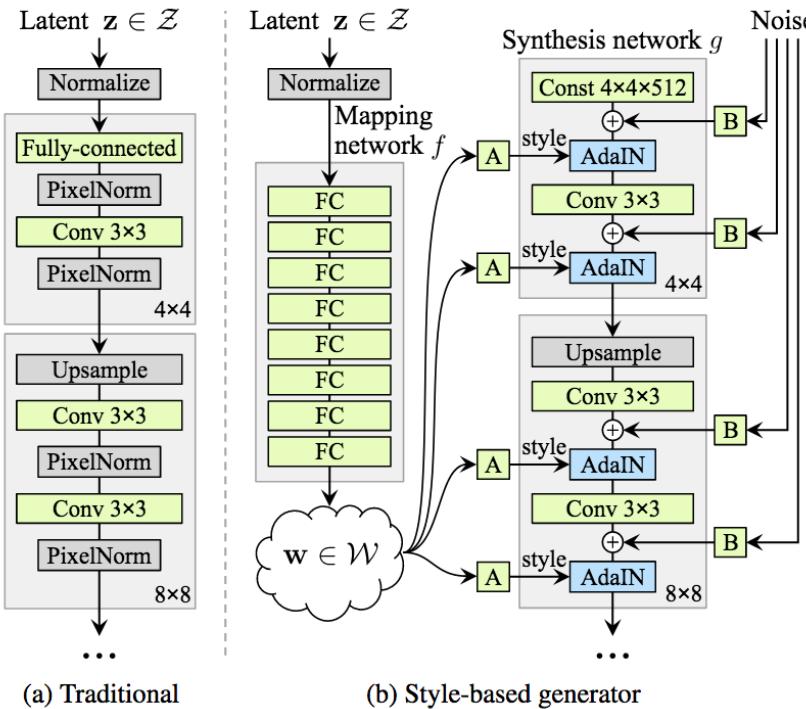
- Generate progressively higher resolution images by extending the architecture
- Akin to curriculum learning

MSG-GAN: Multi-Scale Gradients GAN



- Synthesize in the same time all resolutions
- Simpler architecture than ProGAN and much faster to converge with better results

StyleGAN



$$\text{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

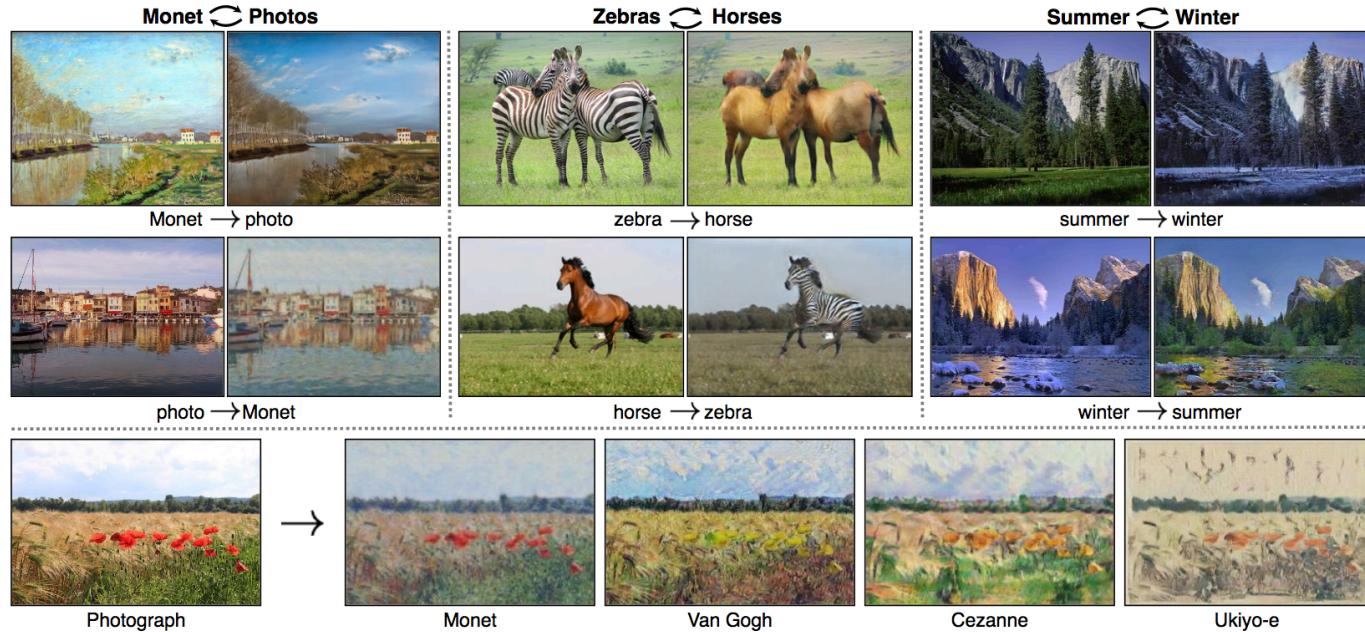
- Based on ProGAN
- Mapping network transforms latent vector noise z
- Which is then added at multiple level with AdaIN
- Latent vector is more disentangled leading to easier vector arithmetic because of the separation of style and stochastic variations



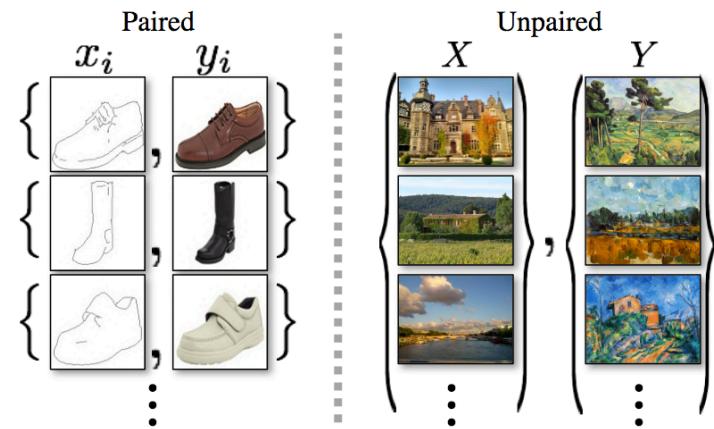
Which face is real?

whichfaceisreal.com

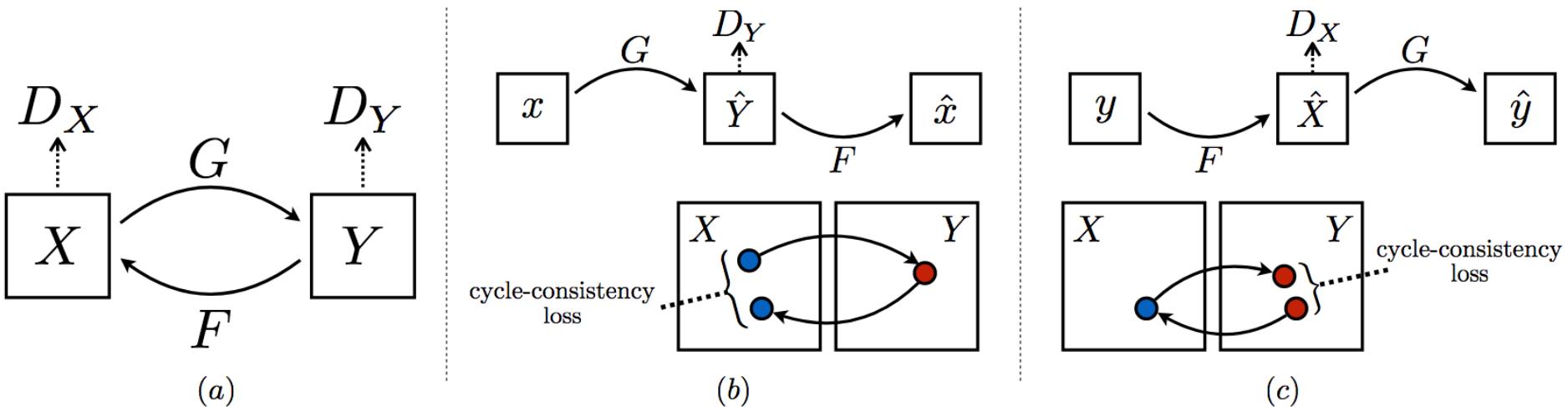
CycleGAN



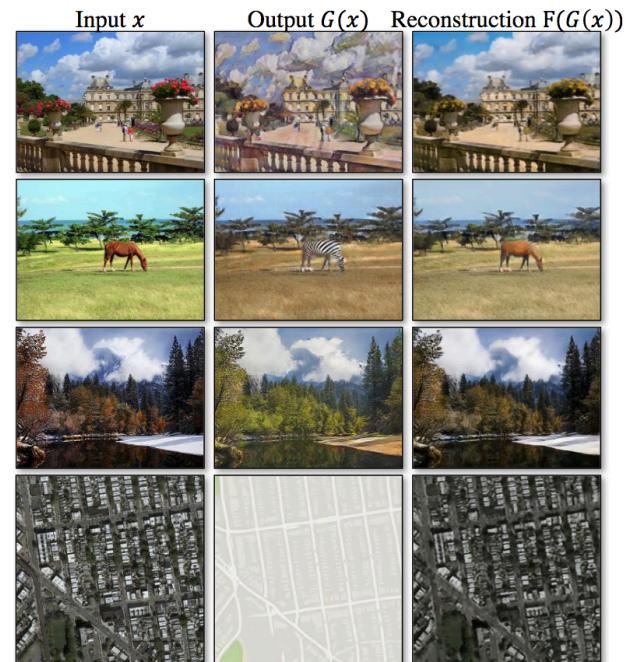
- Unpaired Image-to-Image Translation
- No need to have matching domains!

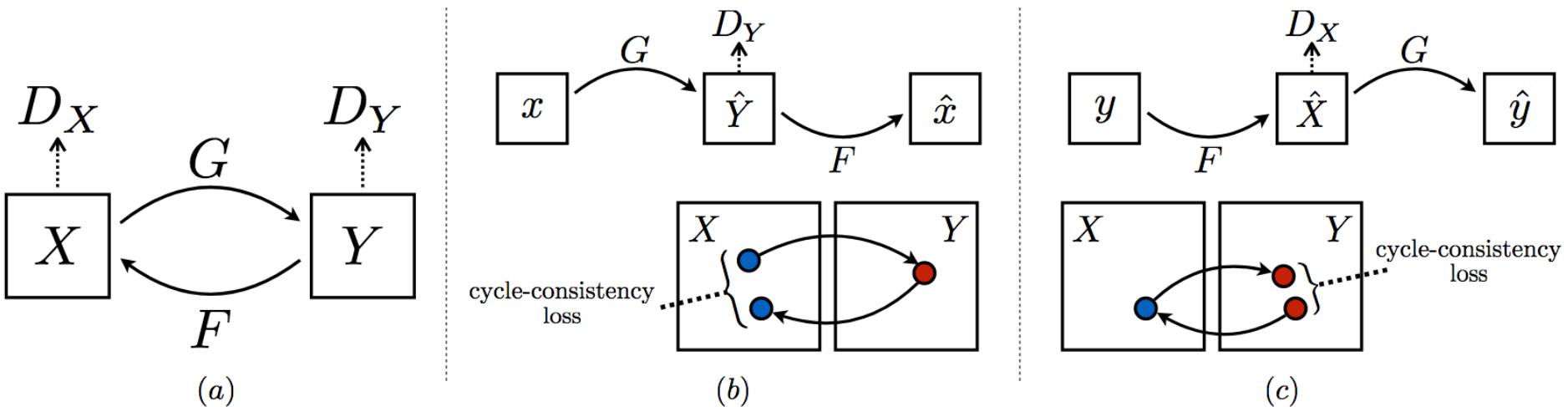


CycleGAN

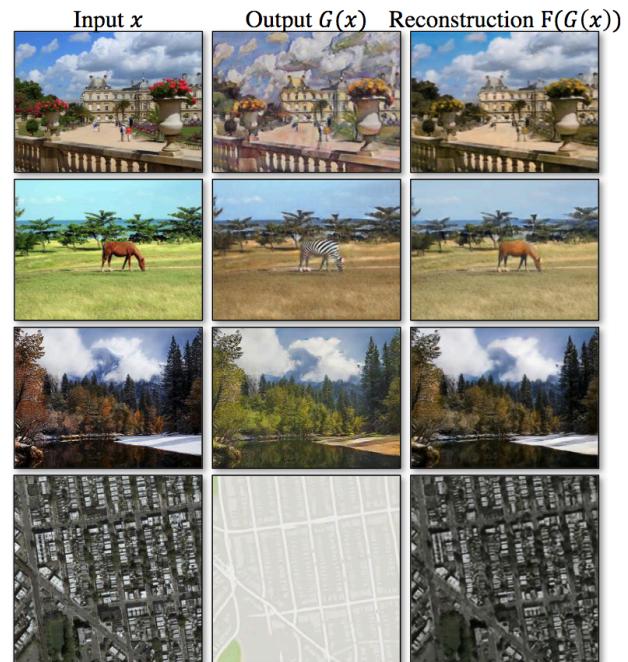


- Translate image from domain X to Y then back to X
 - And vice-versa
- When in domain Y, a discriminator determines if domain is correct



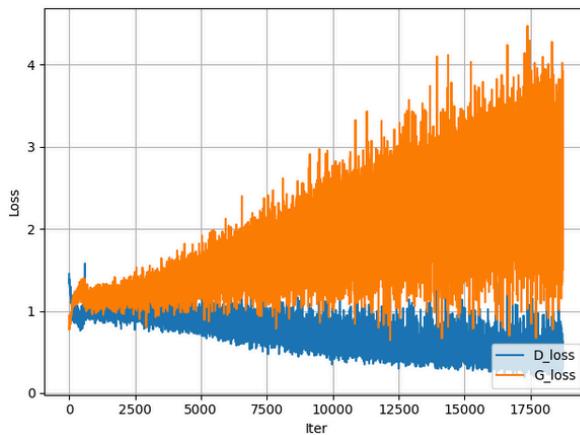


- Translate image from domain X to Y then back to X
 - And vice-versa
- When in domain Y, a discriminator determines if domain is correct





Training difficulty



Training GANs is much more challenging than your common classification model:

- Losses, by definition, do not converge to zero
 - The discriminator and the generator needs to be balanced, of equal “efficiency”
 - The optimal number of updates per model is not necessarily the same
- Hard to know when to stop (more on that in a few slides)
- The optimizer (often Adam) hyperparameters are super sensitive
- Losses can explode
- Models can collapse:
 - **Mode collapse:** same image is always generated
 - **Mode dropping:** some factor of variations are never generated



Spectral Normalization

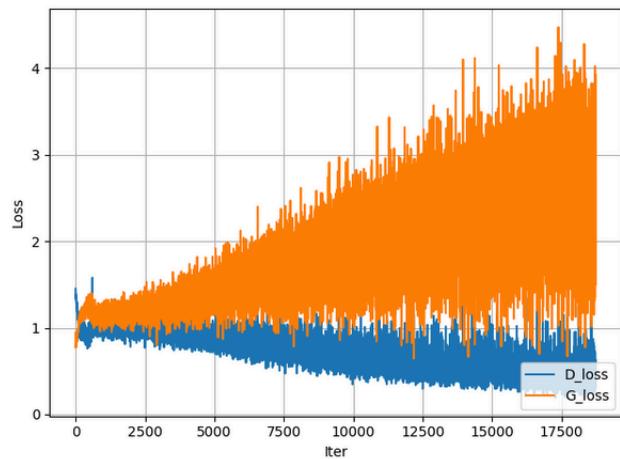
To facilitate training, it helps that the discriminator is K -Lipschitz for a small K .

- Wasserstein GAN (W-GAN)
- Spectral Normalization GAN (SN-GAN)

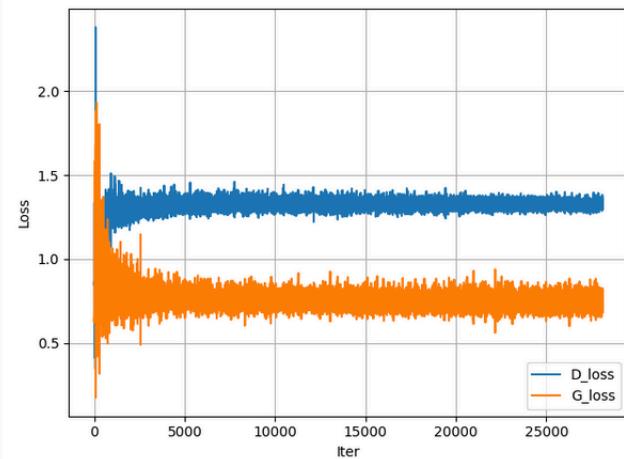
SN-GAN is the simplest and most efficient:

→ Divide each weight of the discriminator by its **spectral norm**, aka the largest singular value:

$$\mathbf{W}_{SN} = \frac{\mathbf{W}}{\sigma(\mathbf{W})}, \sigma(\mathbf{W}) = \max_{\mathbf{h}: \mathbf{h} \neq 0} \frac{\|\mathbf{Wh}\|_2}{\|\mathbf{h}\|_2}$$



without SN



with SN

Is My Generative Model Good?



IS: Inception Score

$$\text{IS}(G) = \exp \left(\mathbb{E}_{\mathbf{x} \sim p_g} D_{KL}(\ p(y|\mathbf{x}) \parallel p(y)) \right)$$

1. Produce likelihoods $p(y|x)$ with a pre-trained Inception network
2. Average likelihoods to have marginal probability $p(y)$
3. Compute KL divergence between them + average over multiple split + exp

Higher is better, minimum score is 0.

$$p(y) = \int_{\mathbf{x}} p(y|\mathbf{x})p_g(\mathbf{x})$$

We want:

- A low-entropy conditional probability $p(x|y)$ (aka high confidence on a class label)
- A high-entropy marginal probability $p(y)$ to have more diversity

<https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a>



$$\text{FID} = |\mu - \mu_w|^2 + \text{tr}(\Sigma + \Sigma_w - 2(\Sigma \Sigma_w)^{1/2})$$

1. Extract features at a deep but not last layer for both real and generated images
2. Minimize this distance between on the mean and covariance activations

Lower is better, minimum score is 0.



Amazon Mechanical Turk

Access a global, on-demand, 24x7 workforce

Get started with Amazon Mechanical Turk

Amazon Mechanical Turk (MTurk) is a crowdsourcing marketplace that makes it easier for individuals and businesses to outsource their processes and jobs to a distributed workforce who can perform these tasks virtually. This could include anything from conducting simple data validation and research to more subjective tasks like survey participation, content moderation, and more. MTurk enables companies to harness the collective intelligence, skills, and insights from a global workforce to streamline business processes, augment data collection and analysis, and accelerate machine learning development.

- Often papers also use humans to validate if a model X produces images more realistic than a model Y
- Platform like Mechanical Turk can do that, but there is a cost!

Style Transfer
Auto-Regressive
Normalizing Flows
NeRF



Style Transfer



Given pretrained VGG16 network:

1. Clone content image as X
2. Minimize **content loss** (MSE) between X and Content image
Minimize **style loss** (MSE) between gram matrices of X and Style image

All that is done at the features level not at the pixels level.

Auto-Regressive



$$p_{\theta}(\mathbf{x}) = \prod_{i=1}^N p_{\theta}(x_i | \mathbf{x}_{<i})$$

- Image generation as a sequence modeling task, akin to language model in NLP
- Can be slow to generate because of a limited parallelizability
- Can suffer from very long sequences



Auto-Regressive (dall-e)

TEXT PROMPT

an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED
IMAGES



a small red block sitting on a large green block



a stack of 3 cubes. a red cube is on the top, sitting on a green cube.
the green cube is in the middle, sitting on a blue cube. the blue cube is
on the bottom.

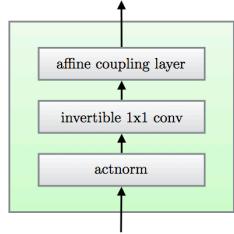


an emoji of a baby penguin wearing a blue hat, red gloves, green shirt,
and yellow pants

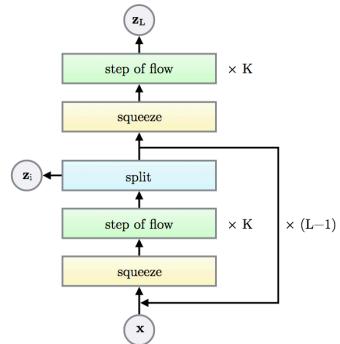


- Autoregressive model based on GPT-3 (transformer for NLP)
- Predict pixels distribution given image textual description

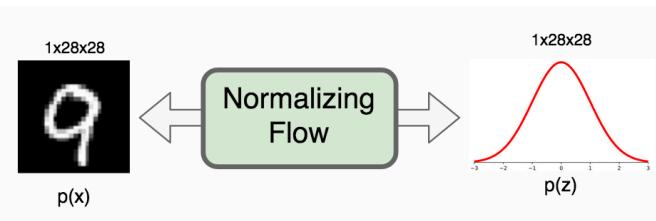
Normalizing Flows



(a) One step of our flow.

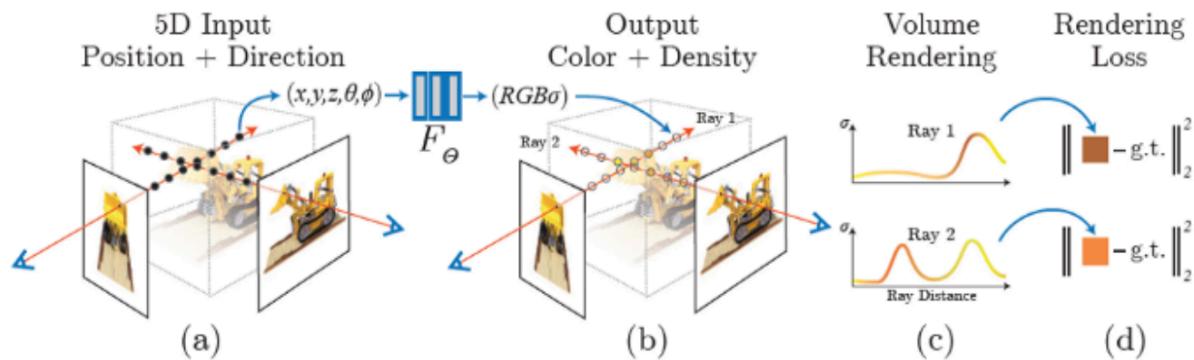
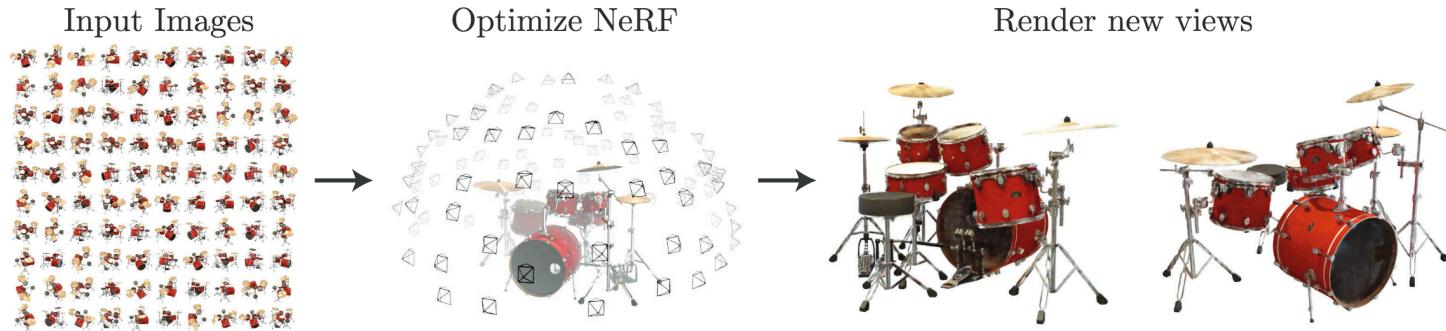


(b) Multi-scale architecture (Dinh et al., 2016).



- Whole network is invertible (thus no pooling, need to have same dimension through the network!)
- During training learn the mapping $f(x) = z$, with z a multivariate Gaussian
- During inference, generate images by sampling z and doing the inverse mapping $f^{-1}(z)$

NeRF: Neural Radiance Fields



- Volume rendering similar to ray tracing, learn to generate a pixel color and a volume density given a 3D location and 2D viewing direction
- Can be used extrapolate new views
- Use 9 fully connected layers + ReLU

Small break,
then coding session!