DSCI 565: GENERATIVE DEEP MODELING

Ke-Thia Yao

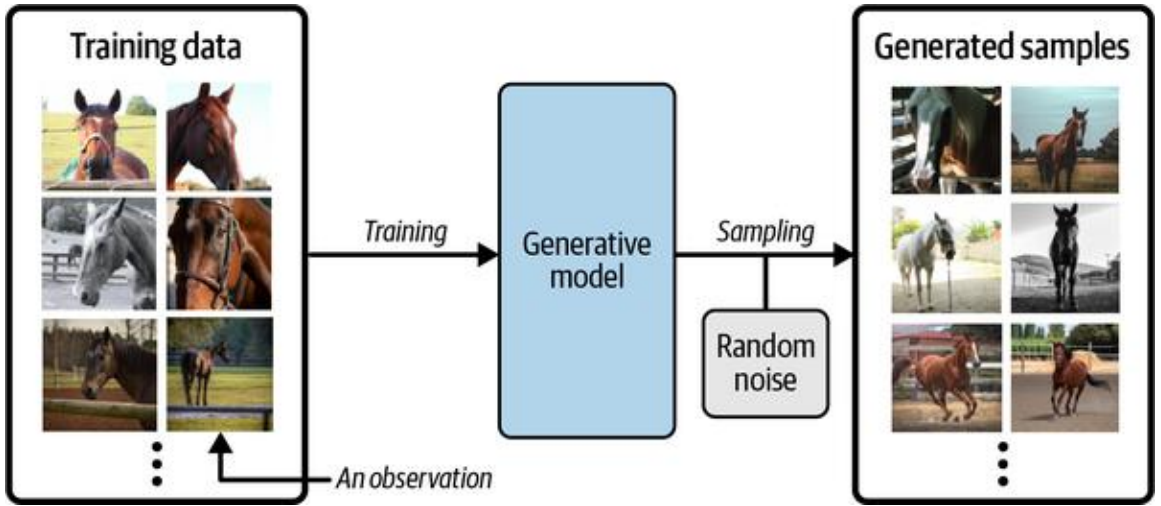
Lecture 3: 2025 November 17

Generative Modeling

discriminative $P(y|x)$

$P(x)$

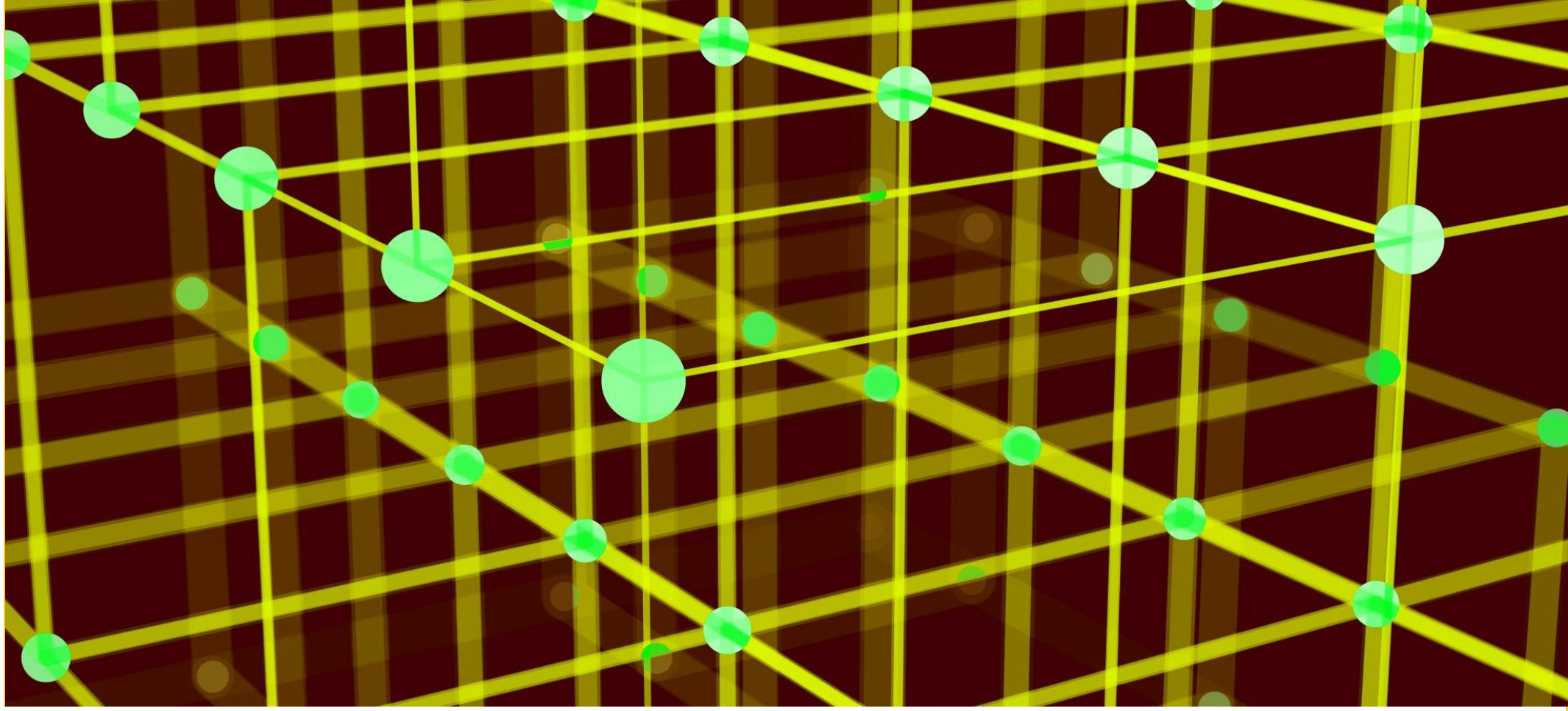
- Given a training set, learn a probabilistic model that we can sample to generate synthetic data that resemble the training set distribution
- For example, given a set of photo-realistic images, learning a model that can generate synthetic photo-realistic images
- Approaches include
  - Generative Adversarial Networks (GAN)
  - Diffusion models



References

- References

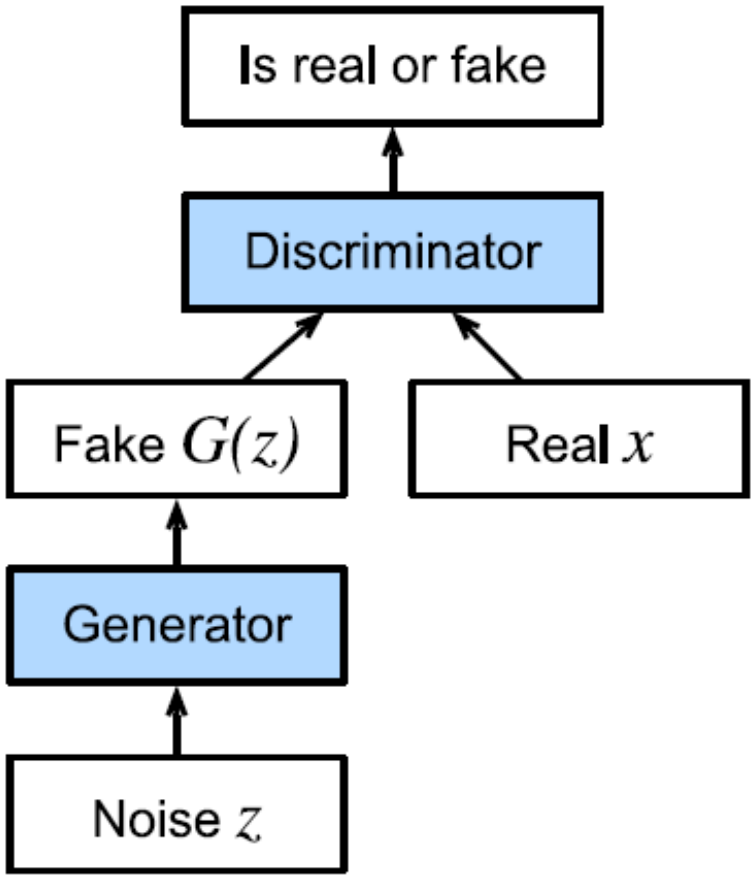
  - Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola. 2023. *Dive into Deep Learning*. 1st edition. Cambridge University Press.
  - Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play, 2nd Edition, David Foster, 2023
  - Hands-On Machine Learning with Scikit-Learn and PyTorch, 3rd Edition, Aurélien Géron
  - Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, et al. 2014. "Generative Adversarial Networks." arXiv:1406.2661. Preprint, arXiv, June 11. https://doi.org/10.48550/arXiv.1406.2661.
  - Ho, Jonathan, Ajay Jain, and Pieter Abbeel. 2020. "Denoising Diffusion Probabilistic Models." arXiv:2006.11239. Preprint, arXiv, December 16. http://arxiv.org/abs/2006.11239.
  - Nichol, Alex, and Prafulla Dhariwal. 2021. "Improved Denoising Diffusion Probabilistic Models." arXiv.Org, February 18. https://arxiv.org/abs/2102.09672v1.
  - Saharia, Chitwan, William Chan, Saurabh Saxena, et al. 2022. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding." arXiv:2205.11487. Preprint, arXiv, May 23. https://doi.org/10.48550/arXiv.2205.11487.



4 Generative Adversarial Networks

GAN Architecture

- Consists of
  - Generator network capable of generating data that looks real
  - Discriminator network capable of discriminating between fake and real data
- Iteratively train the two networks to improve them
  - Improve generator network to fool discriminator network
  - Improve discriminator network to better detect fakes



Discriminator Loss

- Discriminator is typically a binary classifier
- Training data is labelled $y = 1$ for true data, and $y = 0$ for fake
- We train the discriminator $D(x)$ to minimize cross entropy loss:

$$\min_D \{-y \log D(x) - (1 - y) \log(1 - D(x))\}$$

Generator Loss

- Objective is to train the generator $G(z)$ to fool the discriminator

$$D\big(G(z)\big) \approx 1$$

  - Where $z \in \mathcal{R}^d$ is some random vector sampled from $\mathcal{N}(0,1)$

- Train $G(z)$ to maximize cross entropy loss when $y = 0$

$$\max_G \Big\{-(1-y)\log\Big(1 - D\big(G(z)\big)\Big)\Big\}$$

$$= \max_G \Big\{-\log\Big(1 - D\big(G(z)\big)\Big)\Big\}$$

  - Ignore first term, because if $y = 1$ the image is not generated

Two Player Minimax Game

- Player D tries to minimize, and player G tries to maximize the comprehensive objective function

$$\min_{D} \max_{G} \{ -E_{x \sim Data} \log D(x) - E_{z \sim Noise} \log(1 - D(G(z))) \}$$

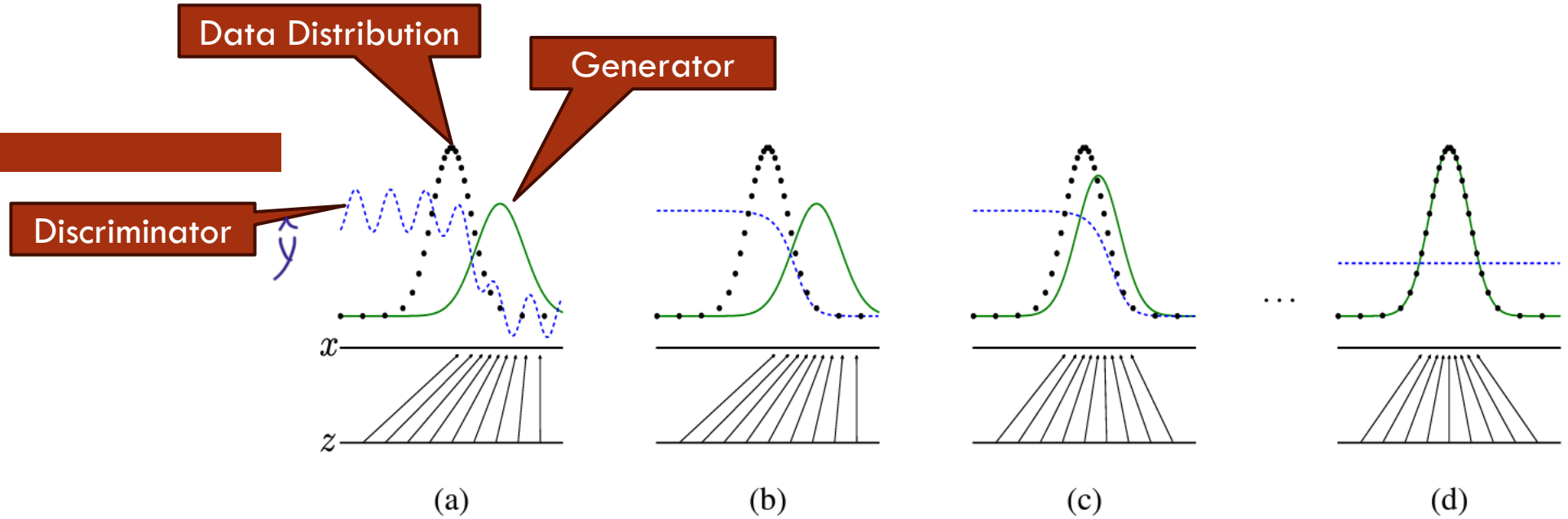


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Goodfellow *et al 2014*

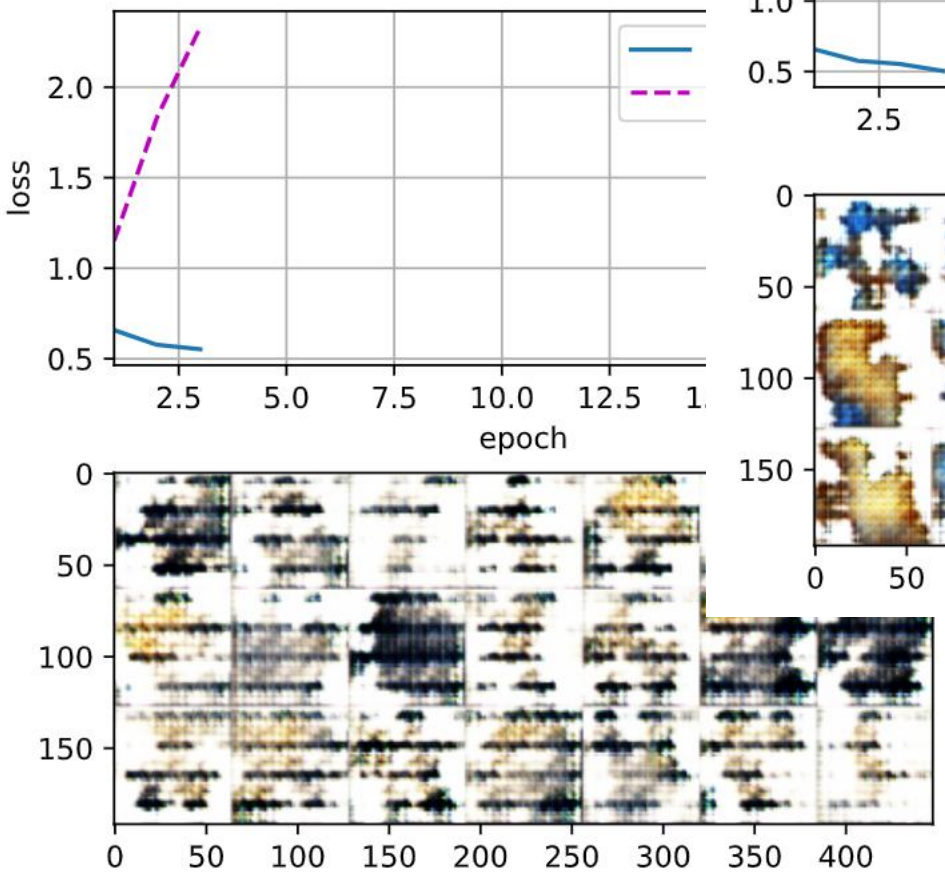
Modified Generator Object

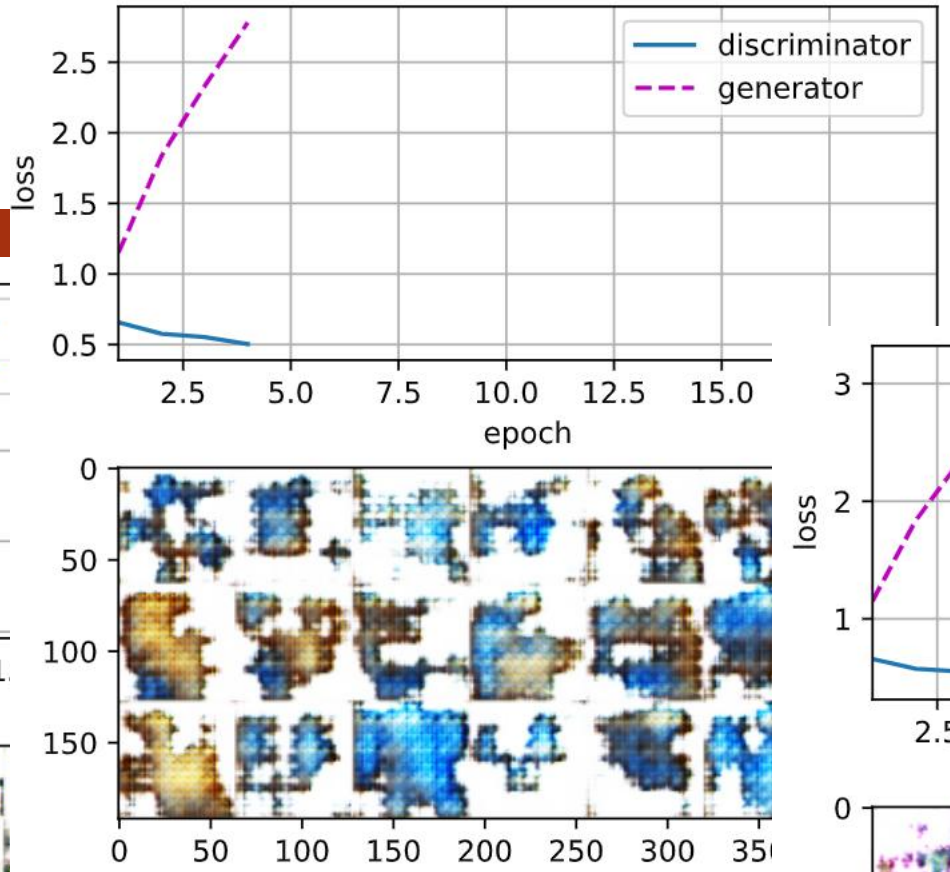
- With the current setup, the same function is being minimized by discriminator and maximized generator
- If the discriminator always successively reject generated samples with high confidence, then gradient for the generator vanishes
- Instead of $\max_G \left\{ -\log\left(1 - D\left(G(z)\right)\right) \right\}$

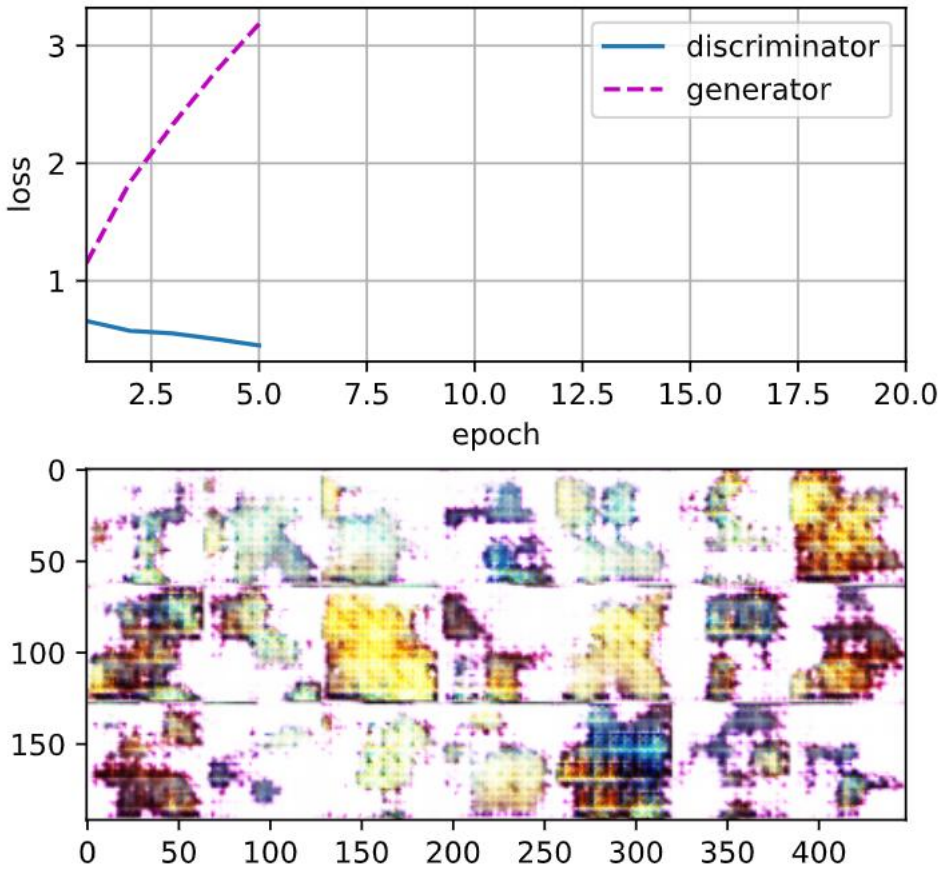
  Goodfellow suggest $\min_G \left\{ -\log\left(D\left(G(z)\right)\right) \right\}$

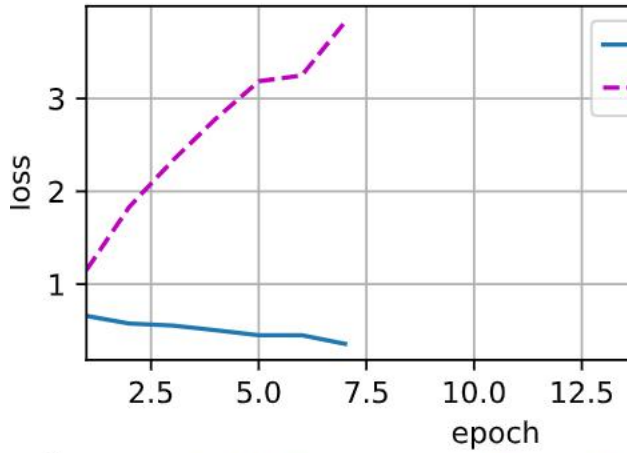
Notebook

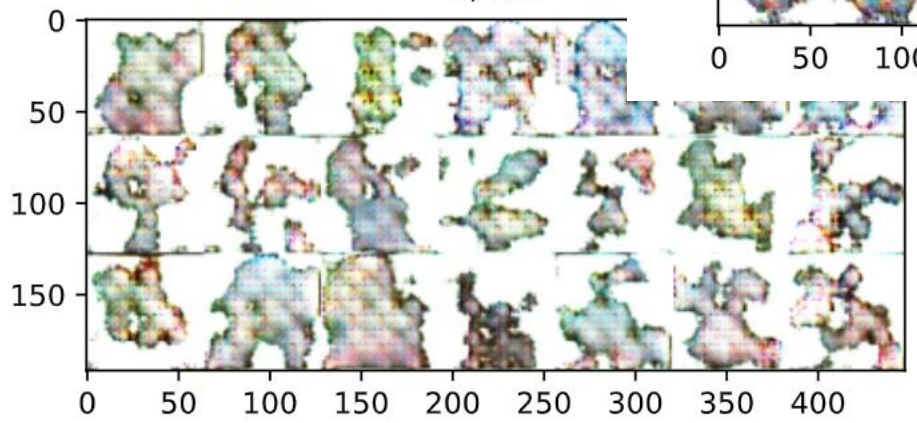
- See
  - chapter_generative-adversarial-networks/gan.ipynb
- See Deep Convolutional GAN
  - chapter_generative-adversarial-networks/dcgan.ipynb

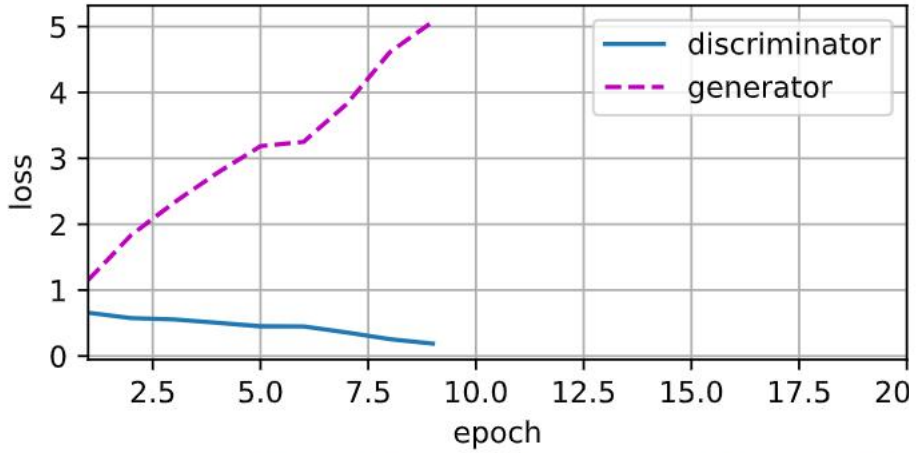

loss
epoch
0.5
1.0
1.5
2.0
2.5
5.0
7.5
10.0
12.5
0
50
100
150
200
250
300
350
400

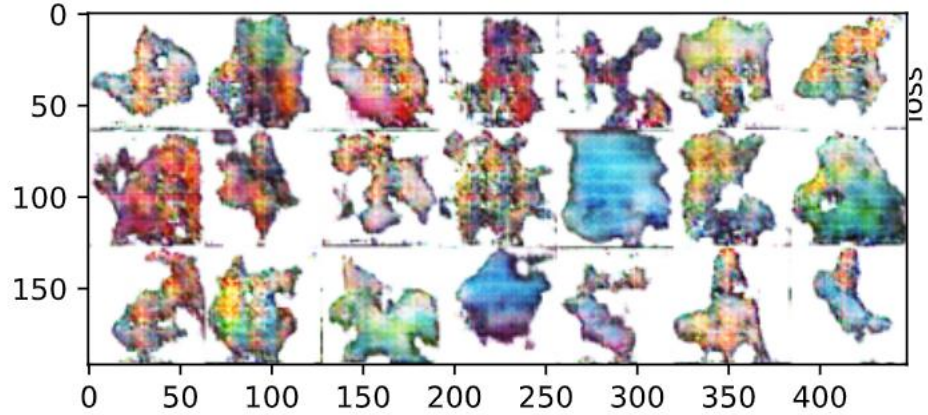

discriminator
generator
loss
epoch
0.5
1.0
1.5
2.0
2.5
5.0
7.5
10.0
12.5
15.0
0
50
100
150
200
250
300
350

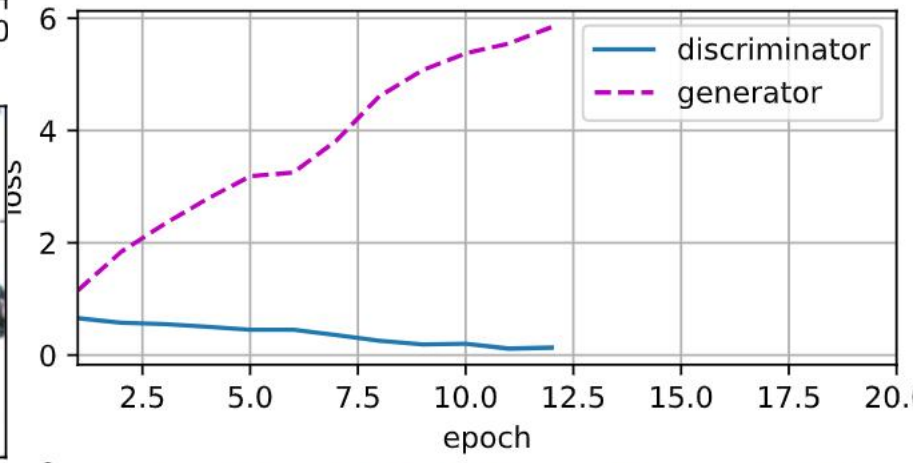

discriminator
generator
loss
epoch
1
2
3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
50
100
150
200
250
300
350
400

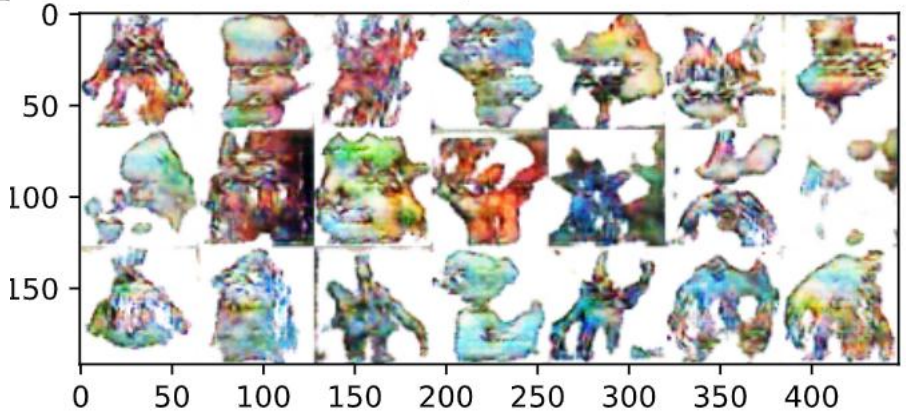

loss
epoch

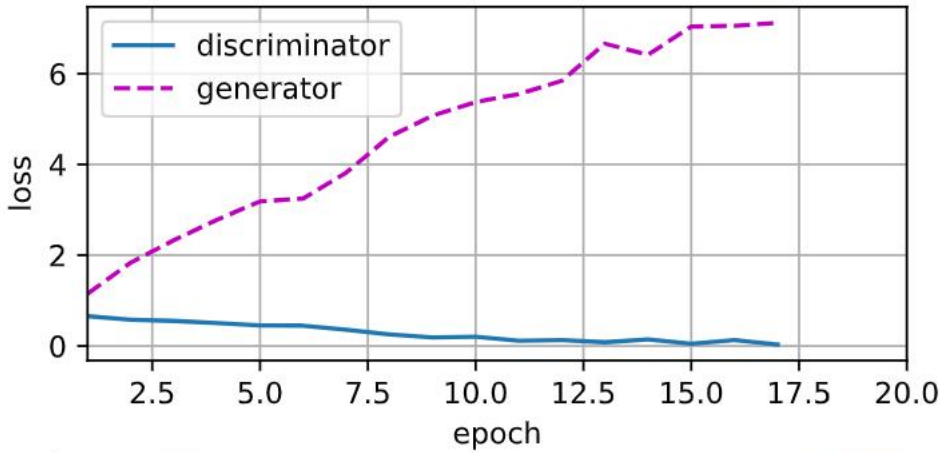


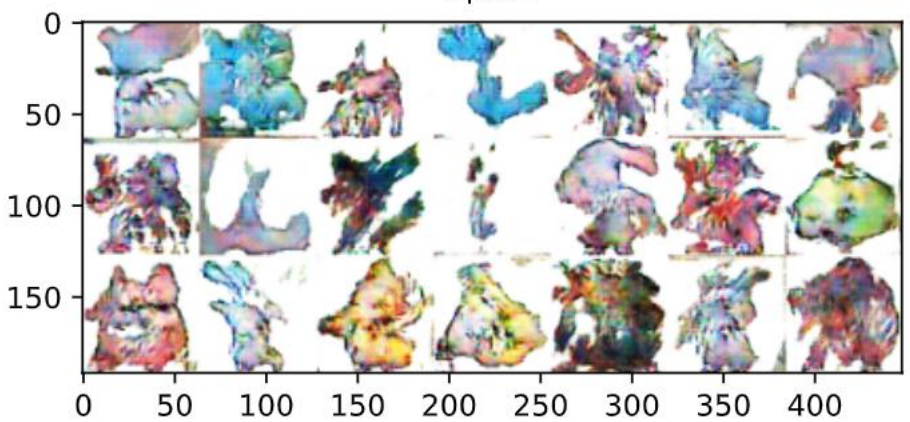

discriminator
generator
loss
epoch

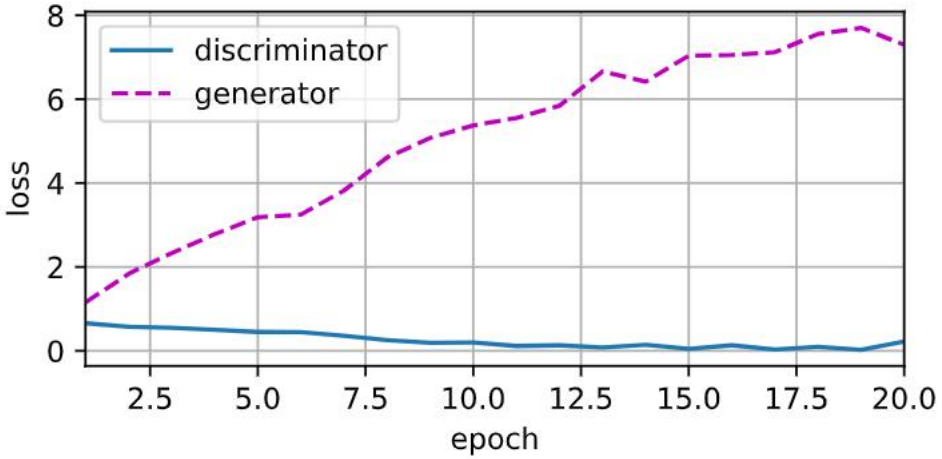


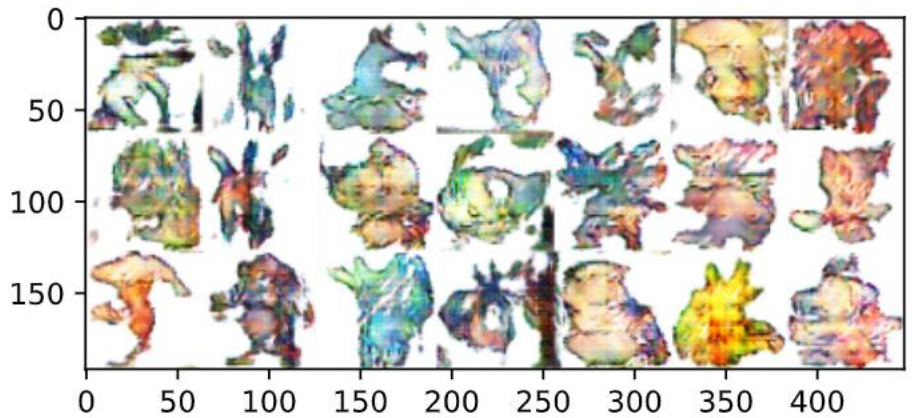

discriminator
generator
loss
epoch




discriminator
generator
loss
epoch
0
2
4
6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

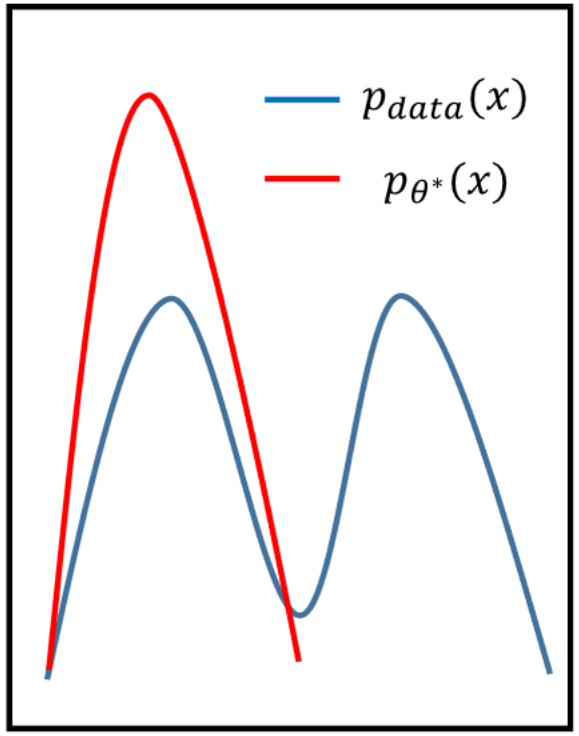

0
50
100
150
0
50
100
150
200
250
300
350
400


discriminator
generator
loss
epoch
0
2
4
6
8
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


0
50
100
150
0
50
100
150
200
250
300
350
400

Training Difficulties

- Mode collapse
  - Generator only covers one mode of the data, say cat images
- Vanishing gradients
  - Near perfect discriminator causes gradient to vanish at every data point
  - Generator cannot learn
- Convergence
  - Ideal is both networks stabilize, and generator converges to data probability distribution
  - In practice, network qualities oscillate



16

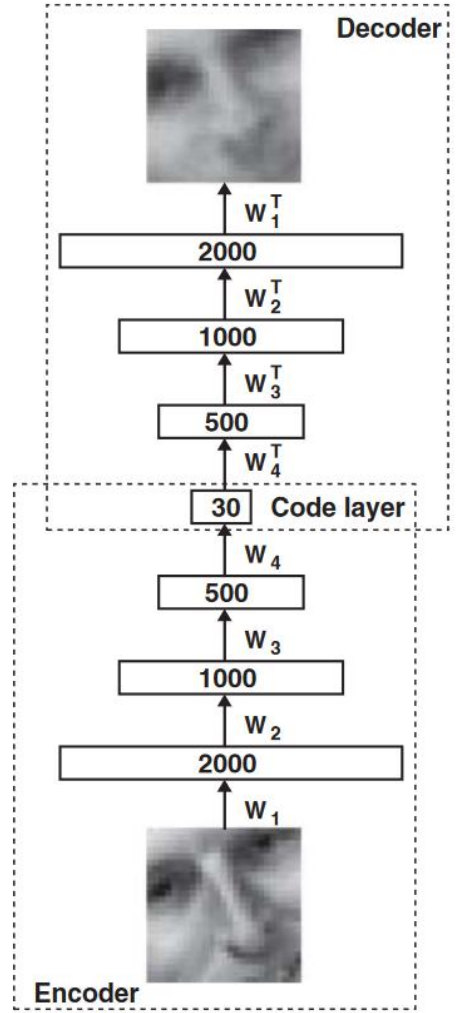
Diffusion Generative Models

Approaches Covered

- Autoencoder
- Variational autoencoder
- Denoising Diffusion
- Text-to-Image Diffusion

Autoencoder

- Autoencoder neural network learns to copy the input images to output images
- An autoencoder consists of
  - Encoder network that compress high-dimensional image into a lower-dimensional embedding space
  - Decoder network that decompress a given embedding vector into a high-dimensional image
- A bottleneck is introduced to force the network to learn latent representations
- Reconstruction loss penalize difference between input and output using either cross-entropy or RMSE



Convolutional Autoencoder

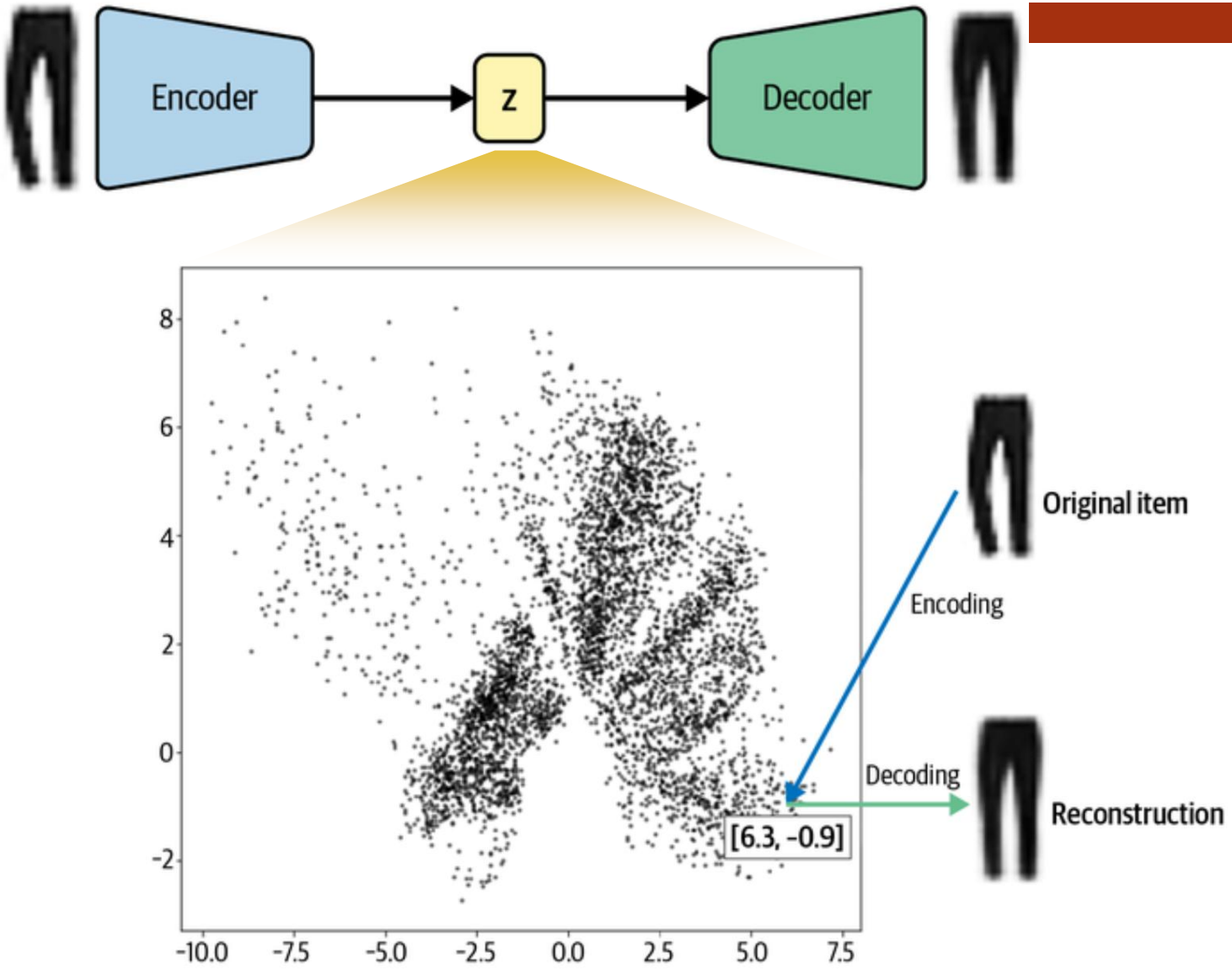
Encoder

Layer (type)	Output shape	Param #
InputLayer	(None, 32, 32, 1)	0
Conv2D	(None, 16, 16, 32)	320
Conv2D	(None, 8, 8, 64)	18,496
Conv2D	(None, 4, 4, 128)	73,856
Flatten	(None, 2048)	0
Dense	(None, 2)	4,098

Decoder

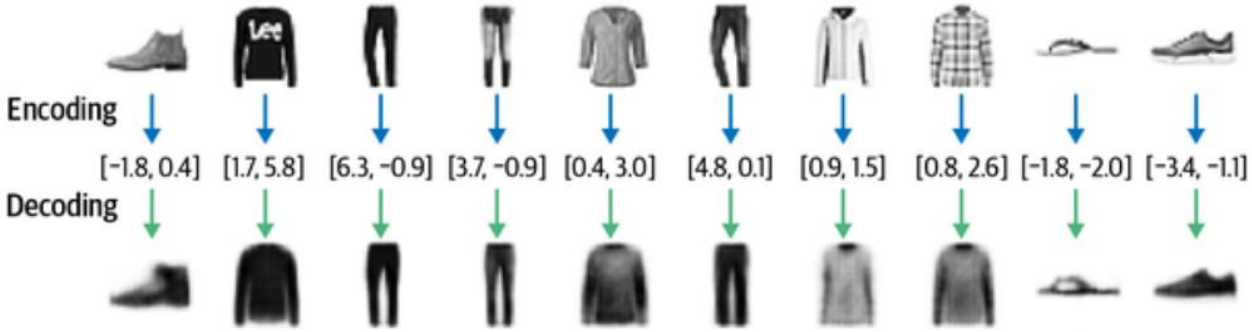
Layer (type)	Output shape	Param #
InputLayer	(None, 2)	0
Dense	(None, 2048)	6,144
Reshape	(None, 4, 4, 128)	0
Conv2DTranspose	(None, 8, 8, 128)	147,584
Conv2DTranspose	(None, 16, 16, 64)	73,792
Conv2DTranspose	(None, 32, 32, 32)	18,464
Conv2D	(None, 32, 32, 1)	289

Latent Representation

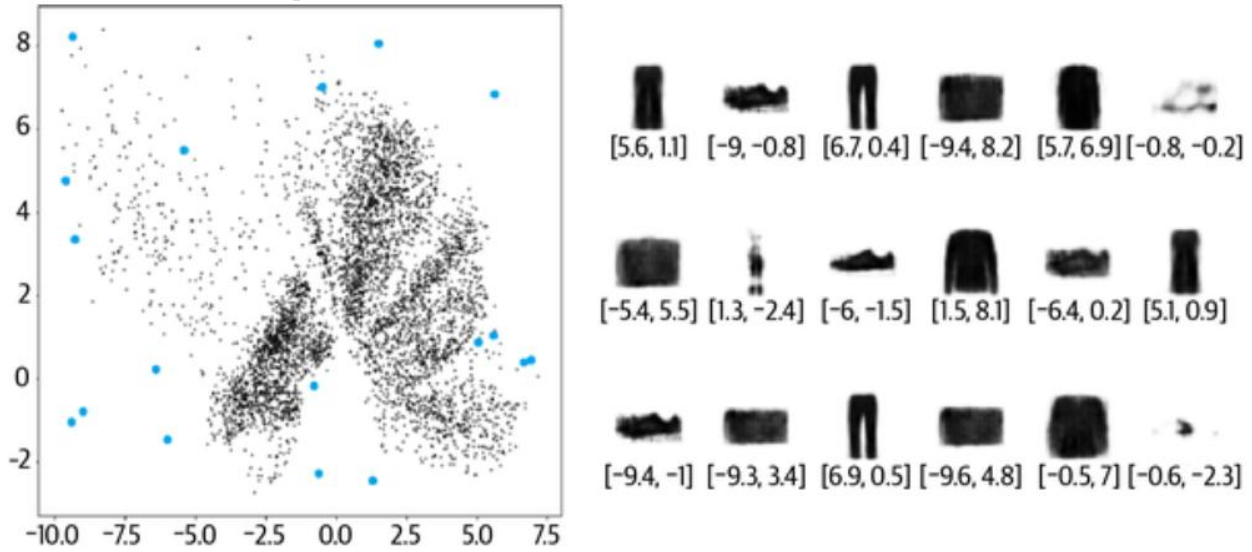


Decoding Embedding Vectors to Generate Images

- Decoding from actual encoded vectors



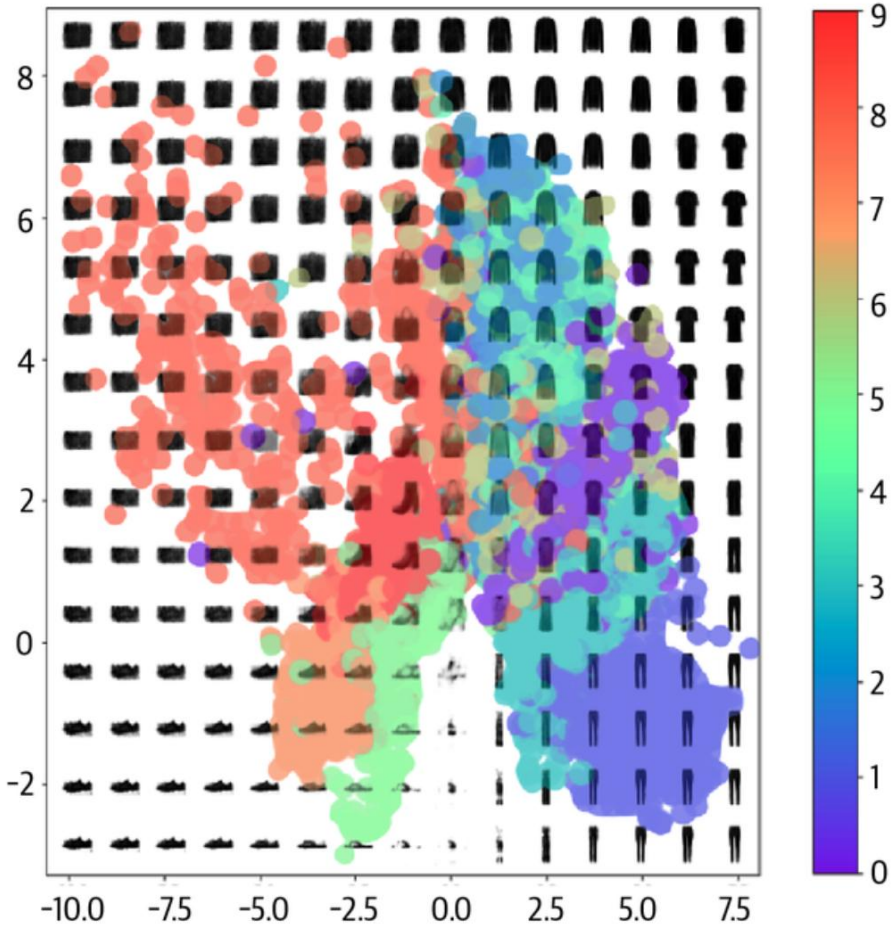
- Decoding from sampled vectors



Sampling Difficulties

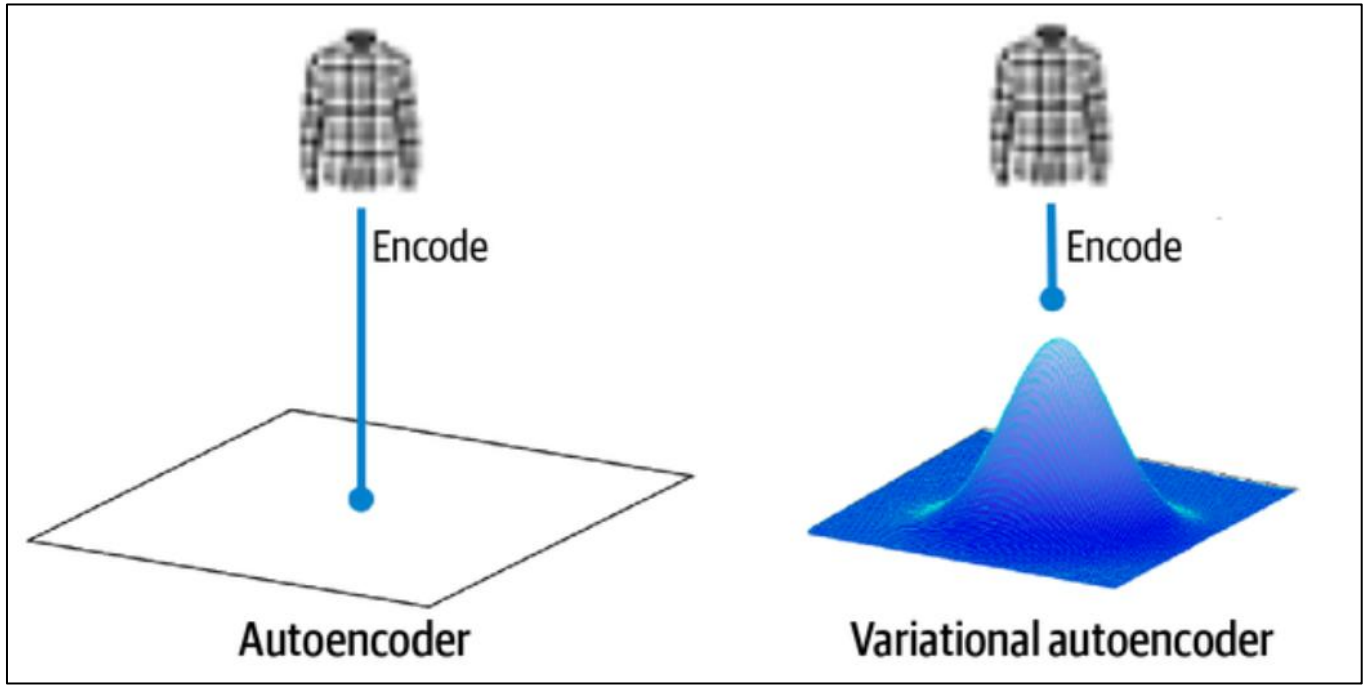
- Some clothing items are overrepresented in latent space area, e.g., bags (id=8)
- Large latent space regions with no training data, i.e., white areas
- Value range across latent space dimensions vary
- Sampling difficulties increases with higher dimensional latent space

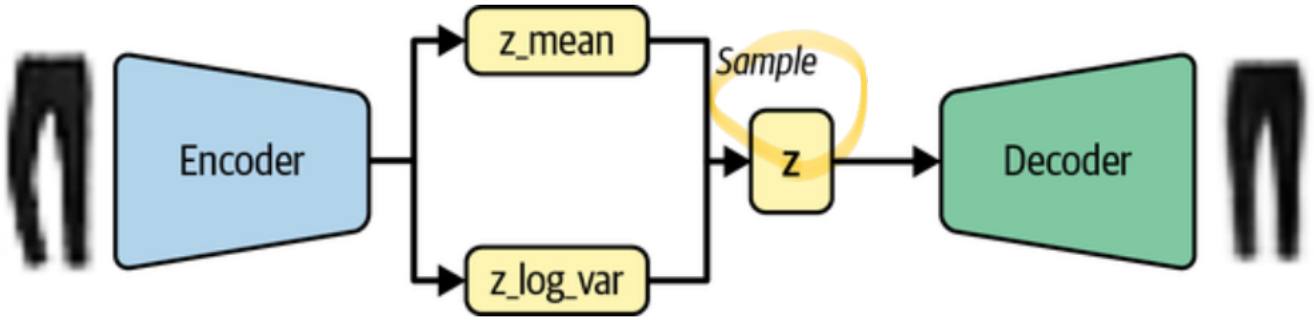
- Want simpler probability distributions for latent space



Variational Autoencoder

- Probabilistic version of autoencoders
- Encoder learns to represent each clothing type as a probabilistic region in latent space
- Give an image the encoder
  - Predicts distribution mean and (log of) variance
  - Returns a sampled point from the corresponding normal distribution
- Decoder learns to generate clothing type associate with the region

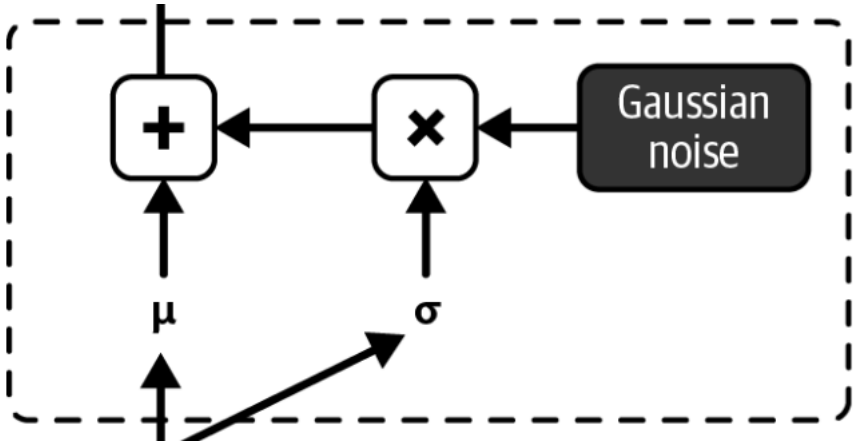




Reparameterization Trick

- Even though the encoder predicts regions (μ, σ), it returns a sampled point in the region
- From a specific point, decoder generates a specific instance of clothing
  - Decoder should not generate an average image of clothing
- However, sampling from distribution is not differentiable
- Reparameterization Trick
  - Sample Gaussian noise $\epsilon \sim \mathcal{N}(0,1)$
  - Compute embedding point: $\mu + \sigma \times \epsilon$
  - ϵ is constant with respect to automatic differentiation

↓ differentiable



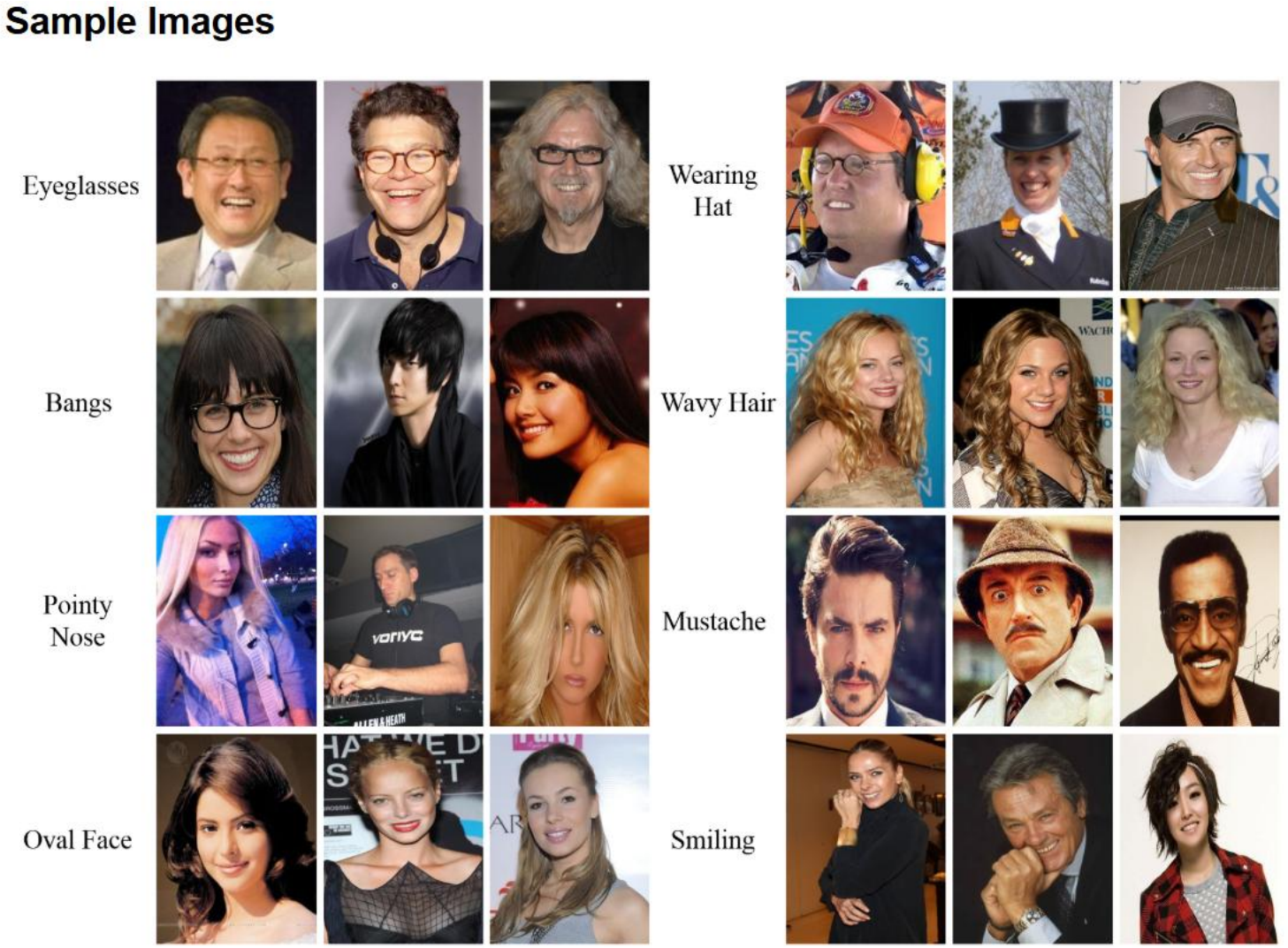
Loss Function

- In addition to reconstruction loss, add another term to force latent distributions to look like unit Gaussians
  - Unit Gaussians are easy to sample
- Add Kullback-Leibler (KL) divergence term that measures the difference between two probability distributions

$$D_{KL}[\mathcal{N}(\mu, \sigma) \parallel \mathcal{N}(0,1)] = -\frac{1}{2}(1 + \log(\sigma^2) - \mu^2 - \sigma^2)$$

Generating Faces with CelebA Dataset

- https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
- Variational Autoencoder
  - Encoder with 4 CNN layers (Batch normalization, LeakyReLU)
  - 200-dimensional latent space
  - Decoder with 4 CNN ConvTranspose layers (Batch normalization, LeakyReLU)



Network Details

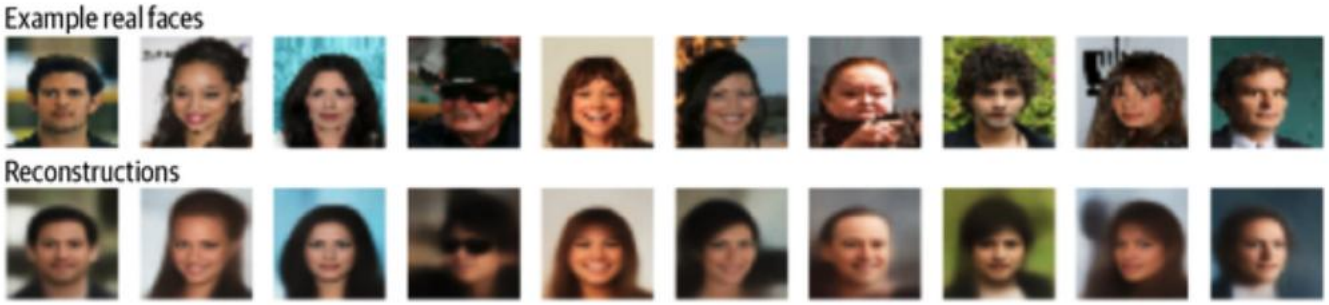
Encoder network

Decoder network

Layer (type)	Output shape	Param #	Connected to
InputLayer (input)	(None, 32, 32, 3)	0	[]
Conv2D (conv2d_1)	(None, 16, 16, 128)	3,584	[input]
BatchNormalization (bn_1)	(None, 16, 16, 128)	512	[conv2d_1]
LeakyReLU (lr_1)	(None, 16, 16, 128)	0	[bn_1]
Conv2D (conv2d_2)	(None, 8, 8, 128)	147,584	[lr_1]
BatchNormalization (bn_2)	(None, 8, 8, 128)	512	[conv2d_2]
LeakyReLU (lr_2)	(None, 8, 8, 128)	0	[bn_2]
Conv2D (conv2d_3)	(None, 4, 4, 128)	147,584	[lr_2]
BatchNormalization (bn_3)	(None, 4, 4, 128)	512	[conv2d_3]
LeakyReLU (lr_3)	(None, 4, 4, 128)	0	[bn_3]
Conv2D (conv2d_4)	(None, 2, 2, 128)	147,584	[lr_3]
BatchNormalization (bn_4)	(None, 2, 2, 128)	512	[conv2d_4]
LeakyReLU (lr_4)	(None, 2, 2, 128)	0	[bn_4]
Flatten (flatten)	(None, 512)	0	[lr_4]
Dense (z_mean)	(None, 200)	102,600	[flatten]
Dense (z_log_var)	(None, 200)	102,600	[flatten]
Sampling (z)	(None, 200)	0	[z_mean, z_log_var]

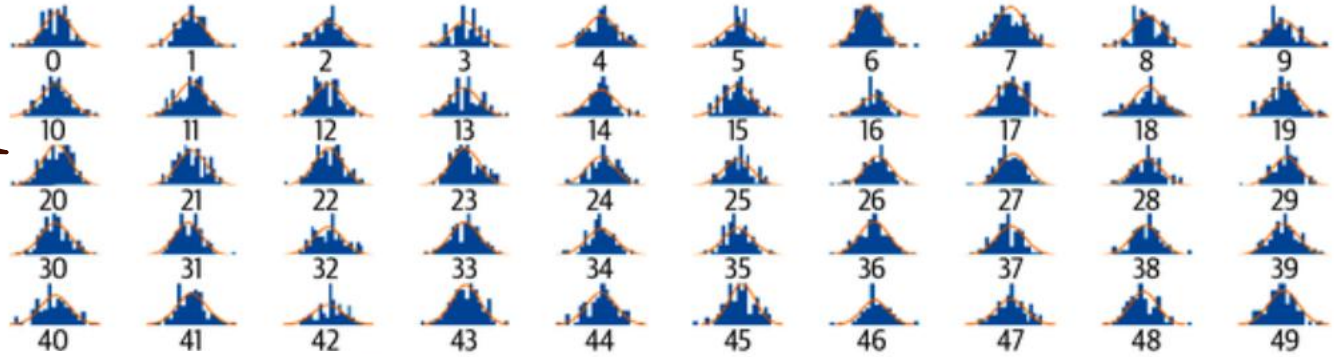
Layer (type)	Output shape	Param #
InputLayer	(None, 200)	0
Dense	(None, 512)	102,912
BatchNormalization	(None, 512)	2,048
LeakyReLU	(None, 512)	0
Reshape	(None, 2, 2, 128)	0
Conv2DTranspose	(None, 4, 4, 128)	147,584
BatchNormalization	(None, 4, 4, 128)	512
LeakyReLU	(None, 4, 4, 128)	0
Conv2DTranspose	(None, 8, 8, 128)	147,584
BatchNormalization	(None, 8, 8, 128)	512
LeakyReLU	(None, 8, 8, 128)	0
Conv2DTranspose	(None, 16, 16, 128)	147,584
BatchNormalization	(None, 16, 16, 128)	512
LeakyReLU	(None, 16, 16, 128)	0
Conv2DTranspose	(None, 32, 32, 128)	147,584
BatchNormalization	(None, 32, 32, 128)	512
LeakyReLU	(None, 32, 32, 128)	0
Conv2DTranspose	(None, 32, 32, 3)	3,459

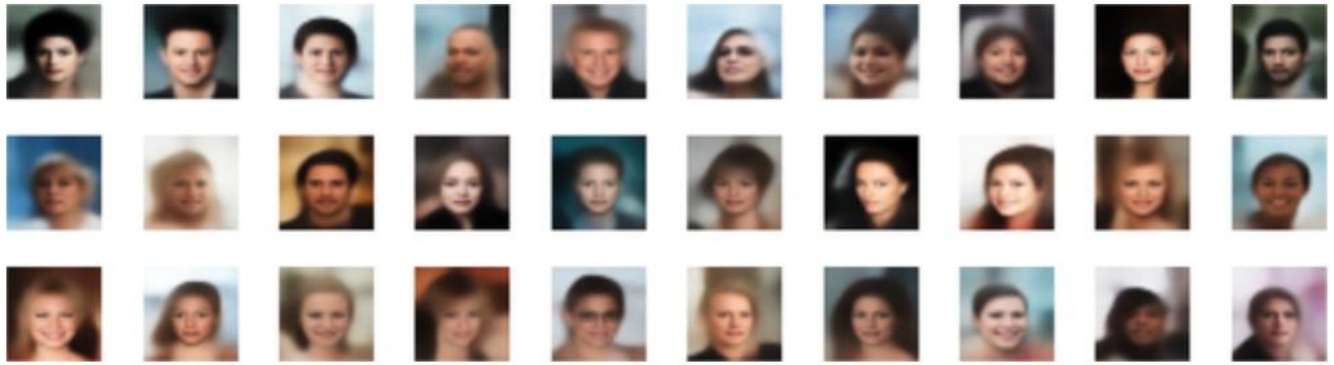
Reconstructions and Generations



Reconstructions

Histogram of latent variables
"Gaussian Noise"



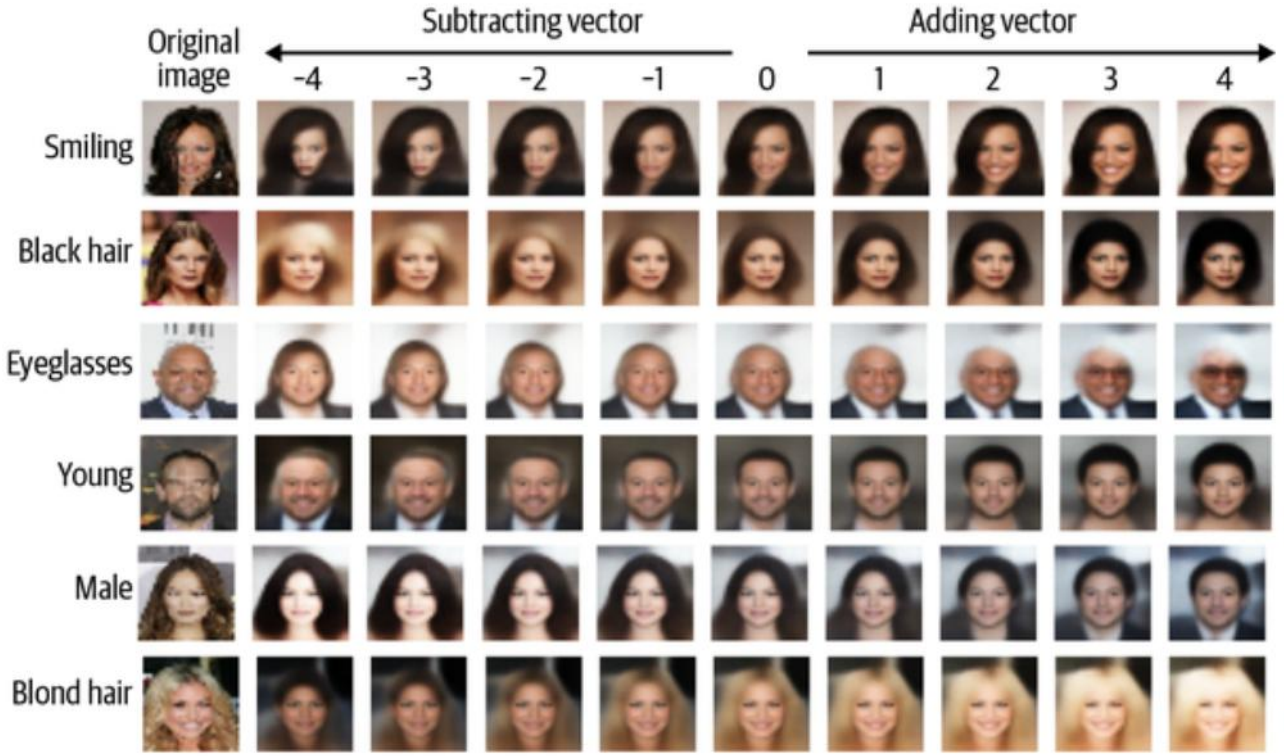


New generated Faces

Latent Space Arithmetic

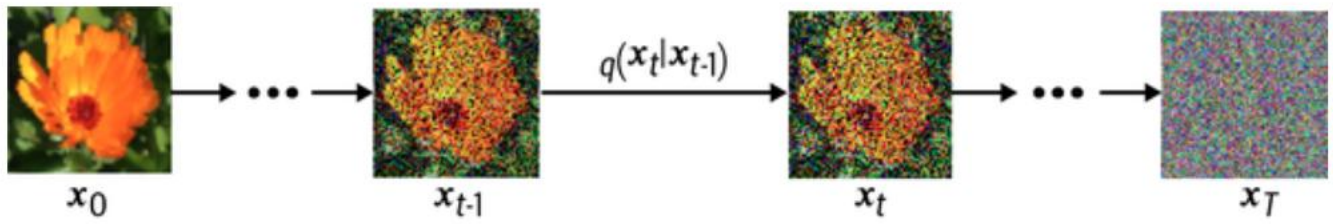
add/subtract

- Use linear interpolation between two faces
- Add/subtract vectors
  - Use dataset label to find average vector for attributes
  - Such as attribute vectors for smiling, black hair, eyeglasses, …



Denoising Diffusion Models

- Approach inspired by the diffusion process of nonequilibrium thermodynamics
- In thermodynamics diffusion is a process where particles move from higher concentration to low concentration by increase in entropy
- For images, diffusion process can be thought of as adding noise



- What if we can learn to reverse this process?

Forward Diffusion Process

- Process:
  - Start with image $\boldsymbol{x}_0$ at time zero
  - Incremental add noise to create $\boldsymbol{x}_t$
  - Until at time T, image $\boldsymbol{x}_T$ is Gaussian noise (zero mean, unit variance)
- Since we want Gaussian noise in the end
  - Normalize $\boldsymbol{x}_0$ to have zero mean and unit variance
  - At each step maintain zero mean and unit variance

Forward Diffusion Process Details

- To add noise and maintain zero mean unit variance, we take linear combination of previous image and Gaussian noise

$$\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\,\epsilon_{t-1}$$

- Above maintains zero mean unit variance, because

$$Var(X+Y) = Var(X) + Var(Y)$$
$$Var(aX) = a^2 Var(X)$$

- Let the forward diffusion process be a fist-order Markov process

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I})$$

Shortcut Forward Diffusion Process

- Sample image $\boldsymbol{x}_t$ from $\boldsymbol{x}_0$ directly without computing $\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \dots, \boldsymbol{x}_1$
- Possible because sum of independent Gaussians is Gaussian
- Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \alpha_1 \times \alpha_2 \times \cdots \times \alpha_t = \prod_{i=1}^{t} \alpha_i$
- Shortcut equation:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1 - \bar{\alpha}_t)\boldsymbol{I}\right)$$

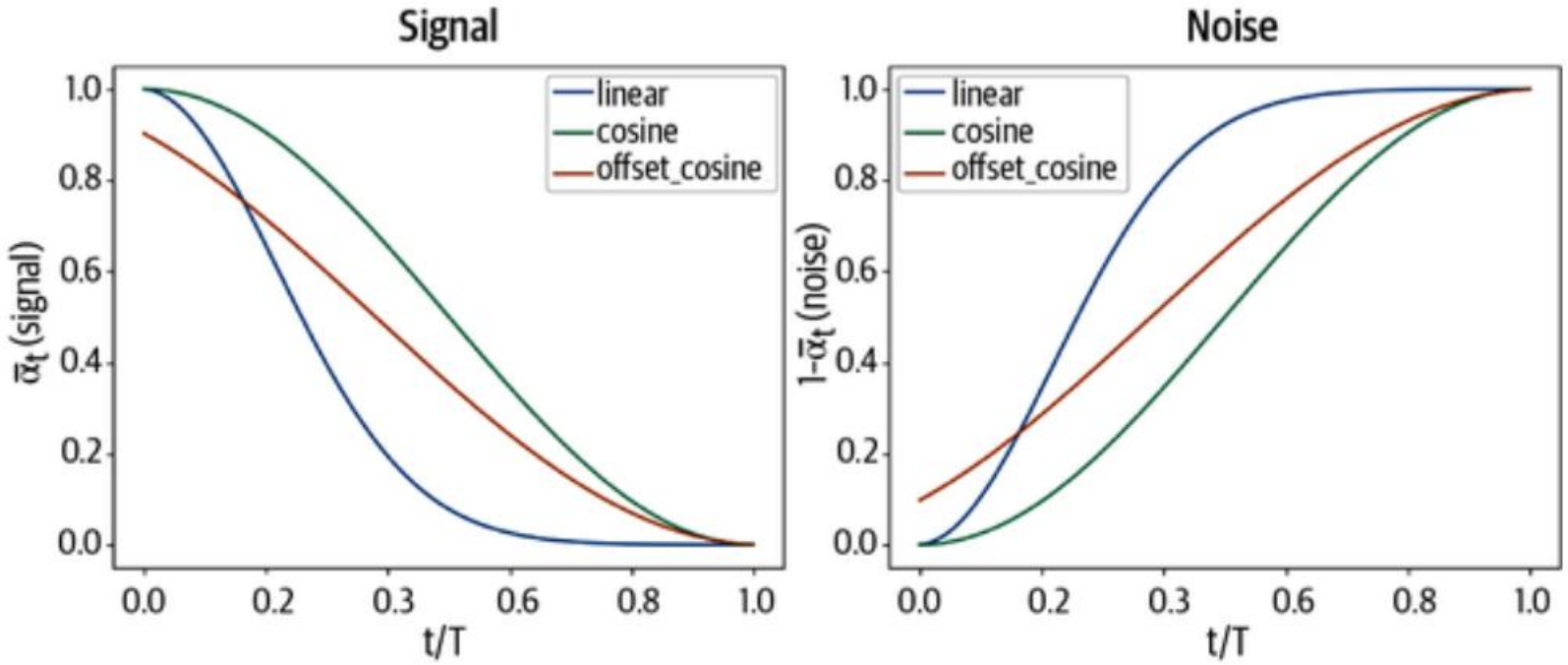
Diffusion Schedule

Linear

Cosine

- Original paper Ho et al, 2020 proposed linear schedule
  - Schedule is too aggressive in adding noise
  - Images from last quarter of the time are just noise
- Nichol and Dhariwal, 2021 proposed less aggressive cosine schedule

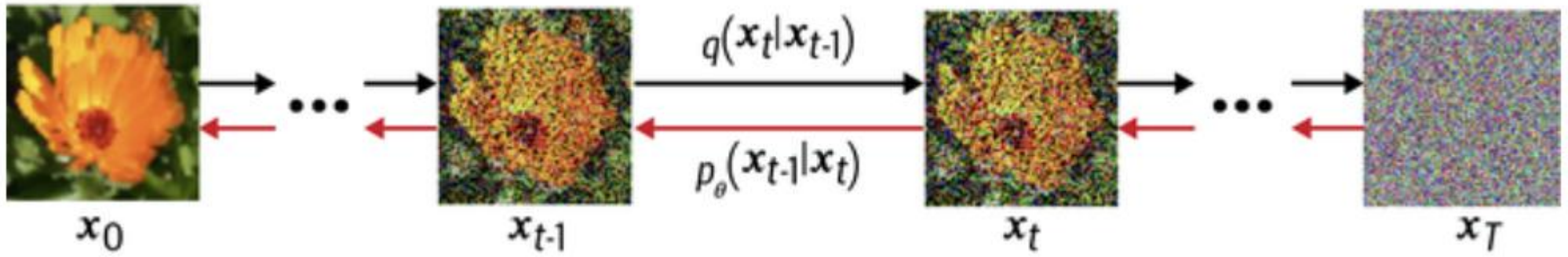
Signal and Noise Curve Comparisons



- Linear schedule adds noise the fastest
- Cosine schedule adds noise the slowest
- Foster book proposes offset cosine that is between the two schedules, but it starts with large initial noise

Reverse Diffusion Process

- Build neural network $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ to learn to reverse diffusion process



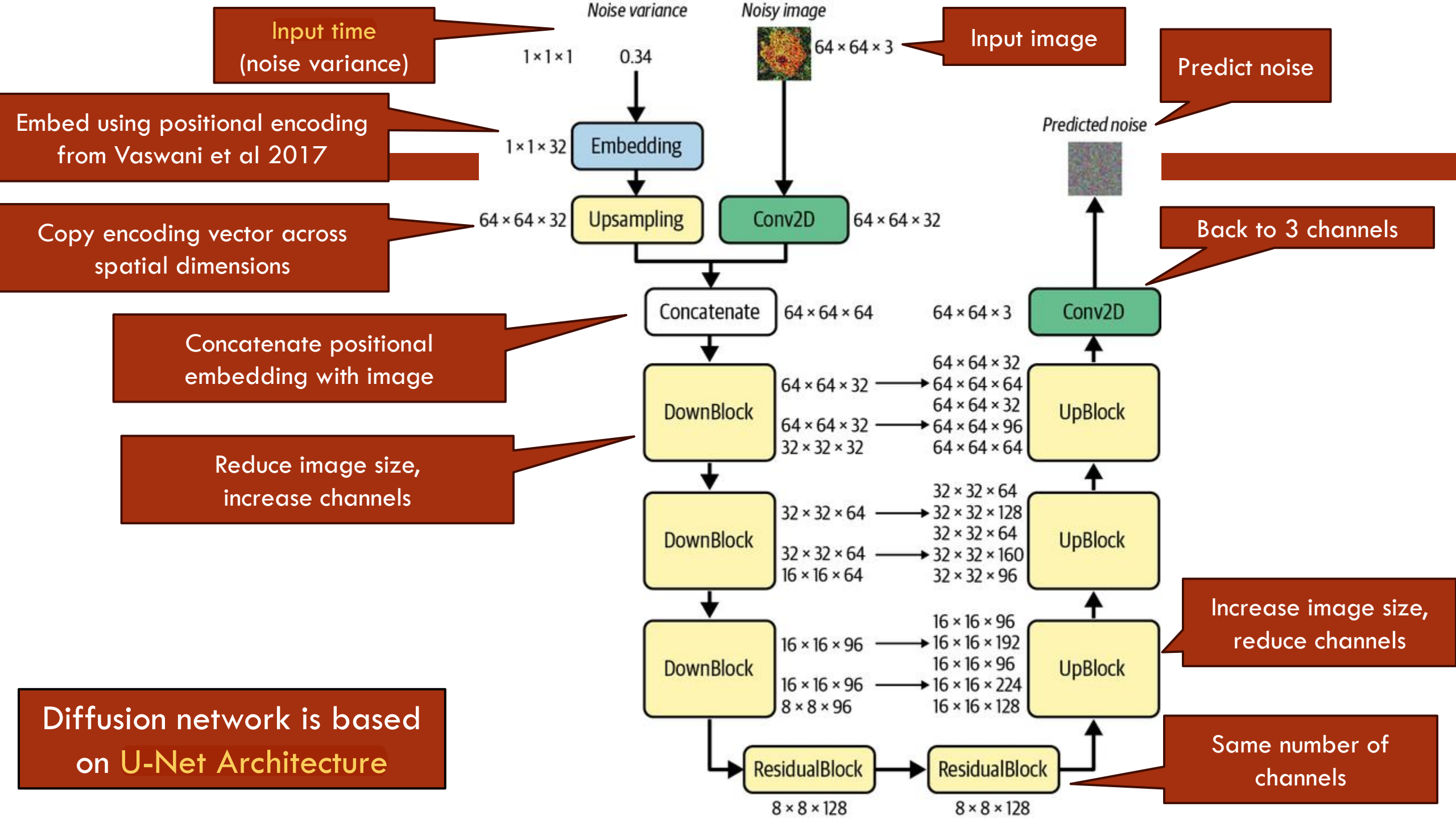
- Reverse diffusion process start with Gaussian noise image then generate picture image
- Like Variational Autoencoder, start with Gaussian latent variables then generate picture image

Training Procedure

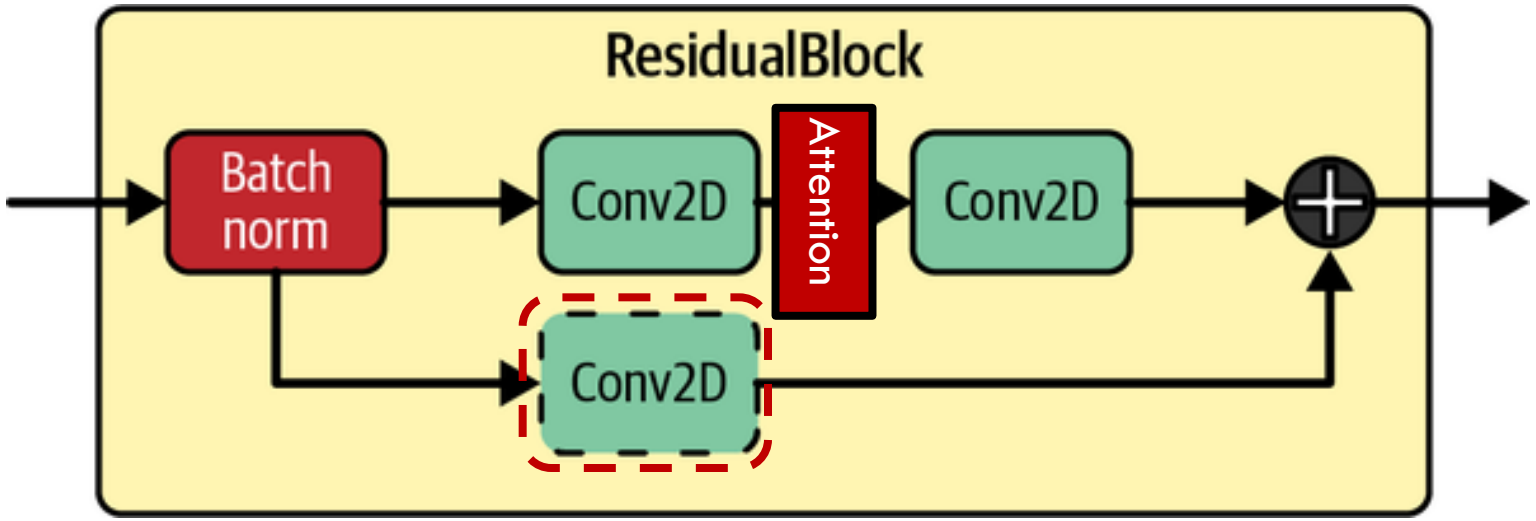
Algorithm 1 Training

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

- Training instance creation
  - Randomly select an image $\boldsymbol{x}_0$
  - Randomly select time $t \in [1, \dots, T]$, say $T = 1000$
  - Sample Gaussian noise image $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  - Create noisy image at t, $\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x_0}, +(1-\bar{\alpha}_t)\boldsymbol{\epsilon}$
  - Return $(t, \boldsymbol{x}_t, \boldsymbol{\epsilon})$
- Train network $\boldsymbol{\epsilon}_\theta$ to predict noise image with mean squared error loss
  - $\boldsymbol{\epsilon}_\theta$ is neural network with parameter θ
- Why train to predict noise $\boldsymbol{\epsilon}_\theta$ instead of original image $\boldsymbol{x}_0$?
  - Empirical, Ho et al, 2020 tried both ways. Training to predict noise image is more stable.


Input time
(noise variance)
Noise variance
Noisy image
Input image
Predict noise
1 × 1 × 1
0.34
64 × 64 × 3
Embed using positional encoding from Vaswani et al 2017
1 × 1 × 32
Embedding
Predicted noise
Copy encoding vector across spatial dimensions
64 × 64 × 32
Upsampling
Conv2D
64 × 64 × 32
Back to 3 channels
Concatenate
64 × 64 × 64
64 × 64 × 3
Conv2D
Concatenate positional embedding with image
DownBlock
64 × 64 × 32
64 × 64 × 32
32 × 32 × 32
64 × 64 × 32
64 × 64 × 64
64 × 64 × 32
64 × 64 × 96
64 × 64 × 64
UpBlock
Reduce image size, increase channels
DownBlock
32 × 32 × 64
32 × 32 × 64
16 × 16 × 64
32 × 32 × 64
32 × 32 × 128
32 × 32 × 64
32 × 32 × 160
32 × 32 × 96
UpBlock
DownBlock
16 × 16 × 96
16 × 16 × 96
8 × 8 × 96
16 × 16 × 96
16 × 16 × 192
16 × 16 × 96
16 × 16 × 224
16 × 16 × 128
UpBlock
Increase image size, reduce channels
ResidualBlock
ResidualBlock
8 × 8 × 128
8 × 8 × 128
Same number of channels
Diffusion network is based on U-Net Architecture

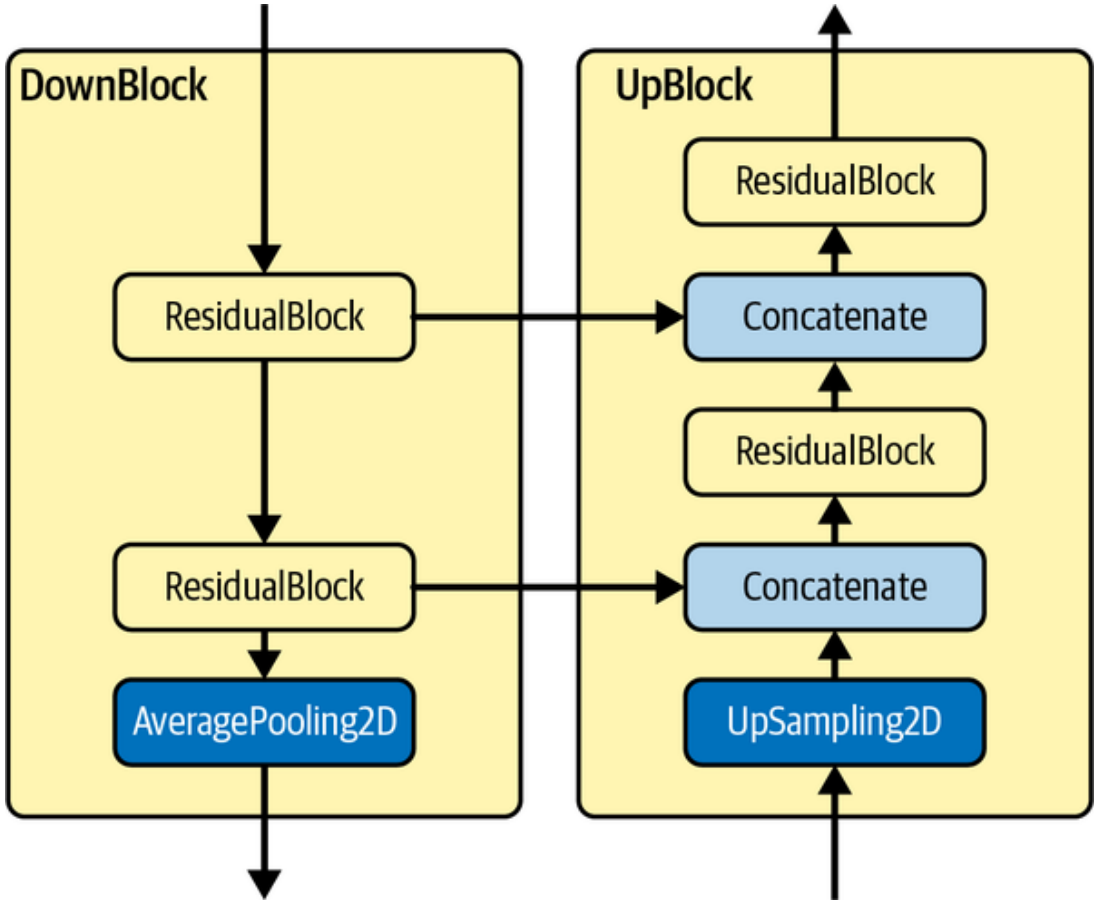
Residual Block



- Similar to ResNet residual blocks
- Dotted "Conv2D" is 1x1 convolution
- Ho et al, 2020 adds self-attention between the two Conv2D layer for some ResidualBlocks
  - "Pixels" attend to other pixels

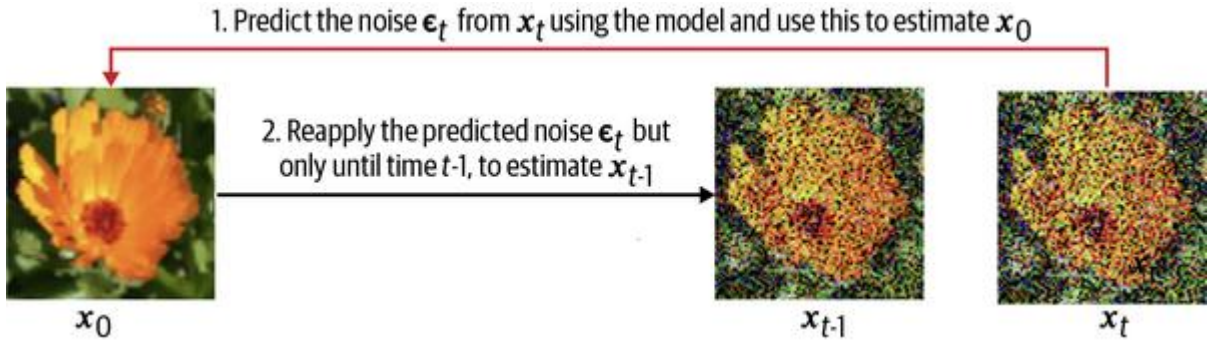
DownBlocks and UpBlocks

- DownBlocks and UpBlocks pairs connected by skip connections
- DownBlock applies "convolutions" and down samples
- UpBlock up samples and applies "convolutions"
- Before convolutions UpBlock concatenate corresponding tensor from DownBlock
- Rational
  - Reducing image size captures higher level semantic information
  - But spatial information is lost
  - Skip connection adds back spatial information



Generating Images

- Model is trained to predict all the noise and recover the original image in one step
- However, we will take multiple steps
- Each step subtracting a small fraction of the predicted noise



Algorithm 2 Sampling

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

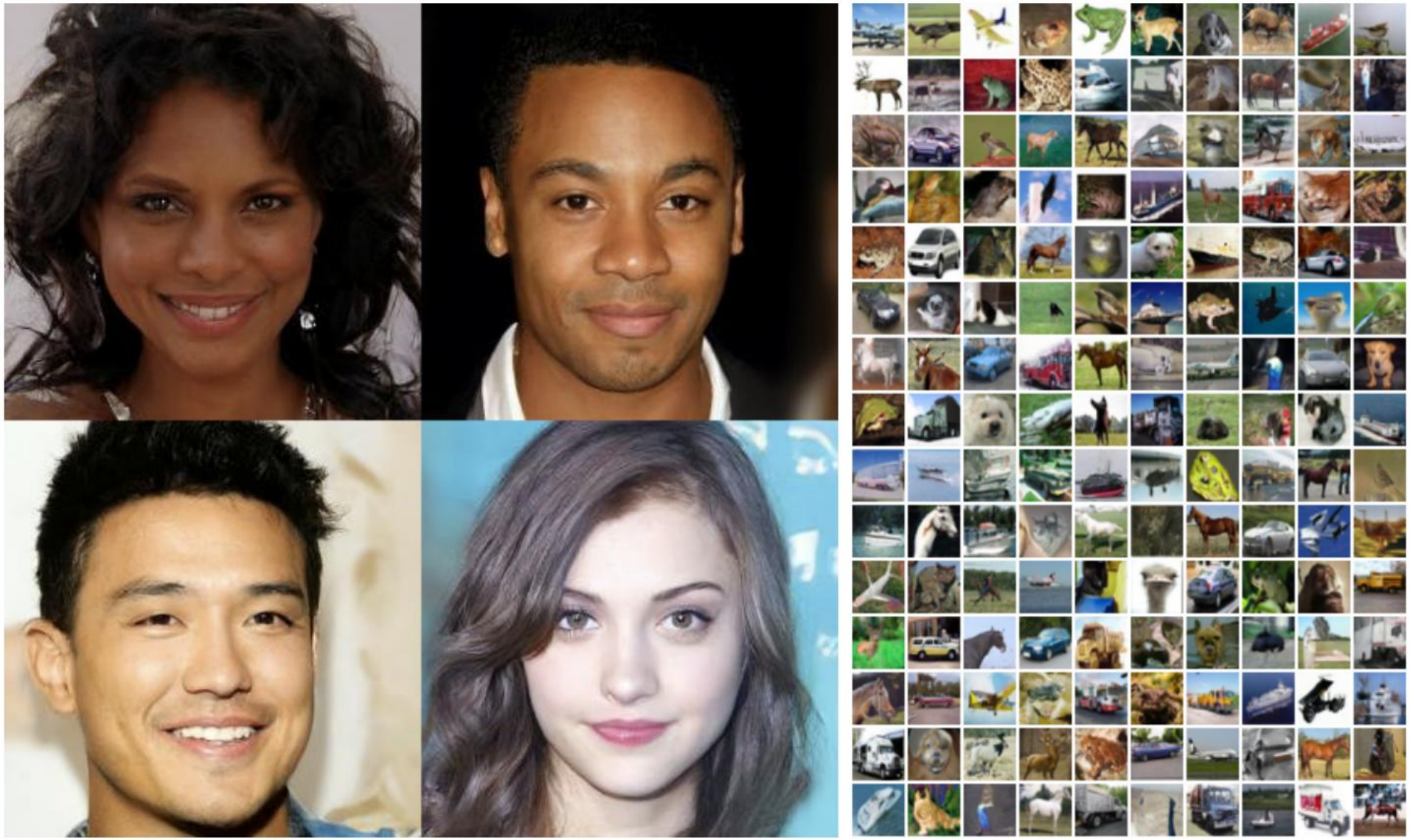


Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

Ho et al, 2020

FID (Fréchet inception distance) Metric

- Distance metric used to measure the distance between the set of training images and the set of generate images
- Encode images in each set using Inception v3 net gather statistics (mean μ and covariance Σ) on the activations of the penultimate layer (2000 features)
- Use Fréchet distance to compare the difference

$$d_F\big((\mu_1, \Sigma_1), (\mu_2, \Sigma_2)\big) = \|\mu_1 - \mu_2\|_2^2 + \mathrm{tr}\left(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{\frac{1}{2}}\right)$$

- Proposed by Heusel et al, 2018

FID between CelebA Dataset and Its Transformations

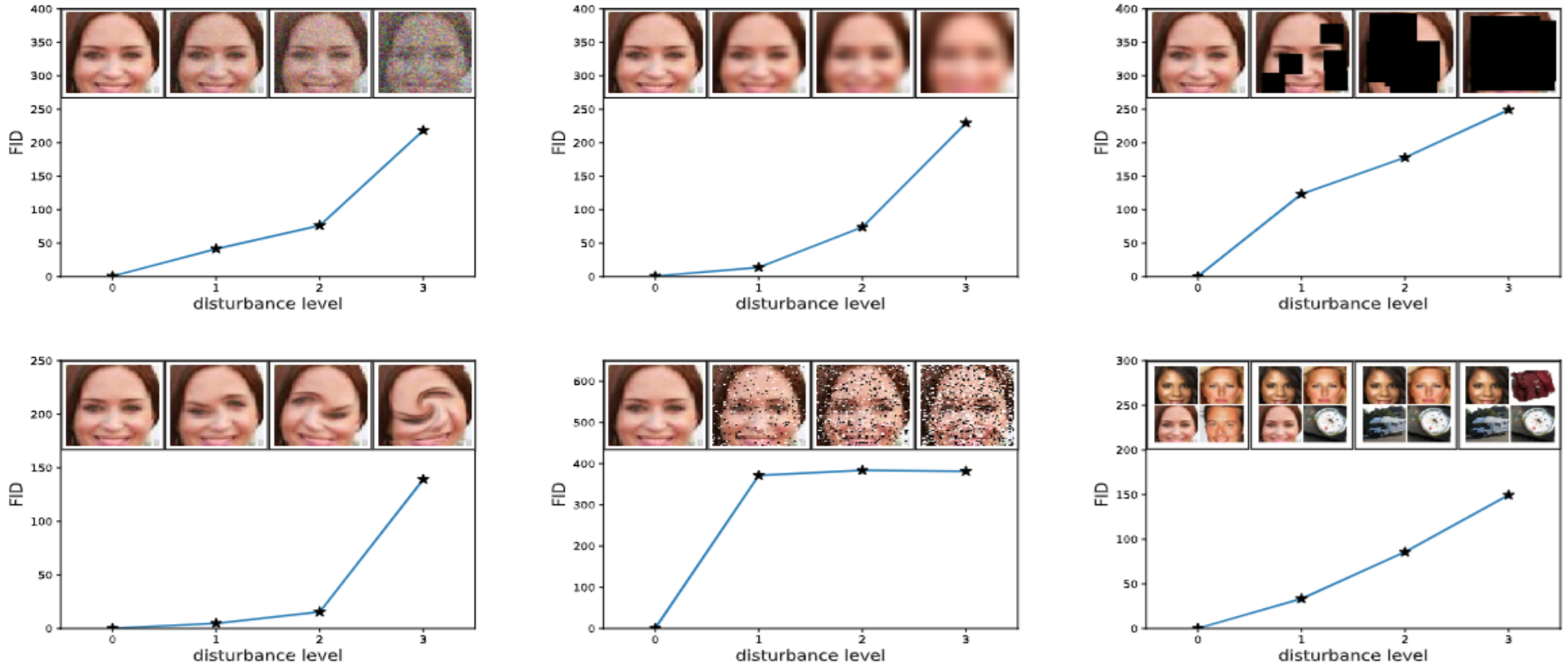


Figure 3: FID is evaluated for **upper left:** Gaussian noise, **upper middle:** Gaussian blur, **upper right:** implanted black rectangles, **lower left:** swirled images, **lower middle:** salt and pepper noise, and **lower right:** CelebA dataset contaminated by ImageNet images. The disturbance level rises from zero and increases to the highest level. The FID captures the disturbance level very well by monotonically increasing.

FID Distance on Generate Churches and Bedrooms

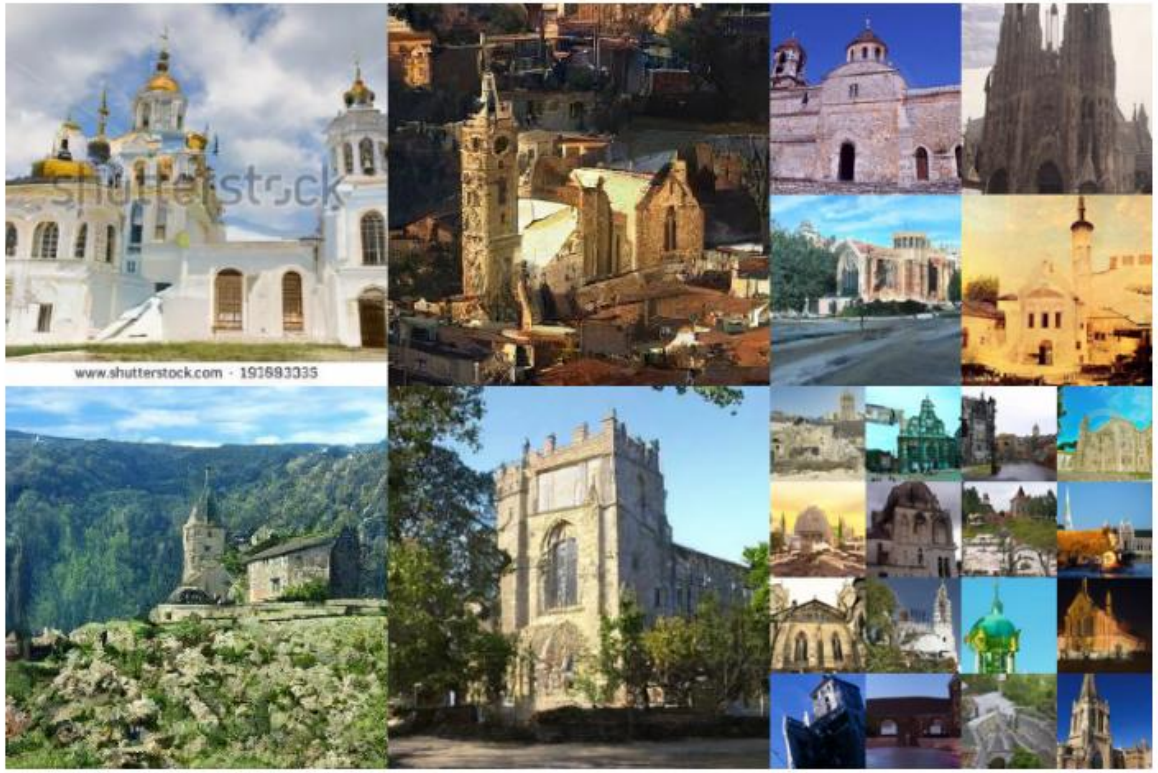


Figure 3: LSUN Church samples. FID=7.89

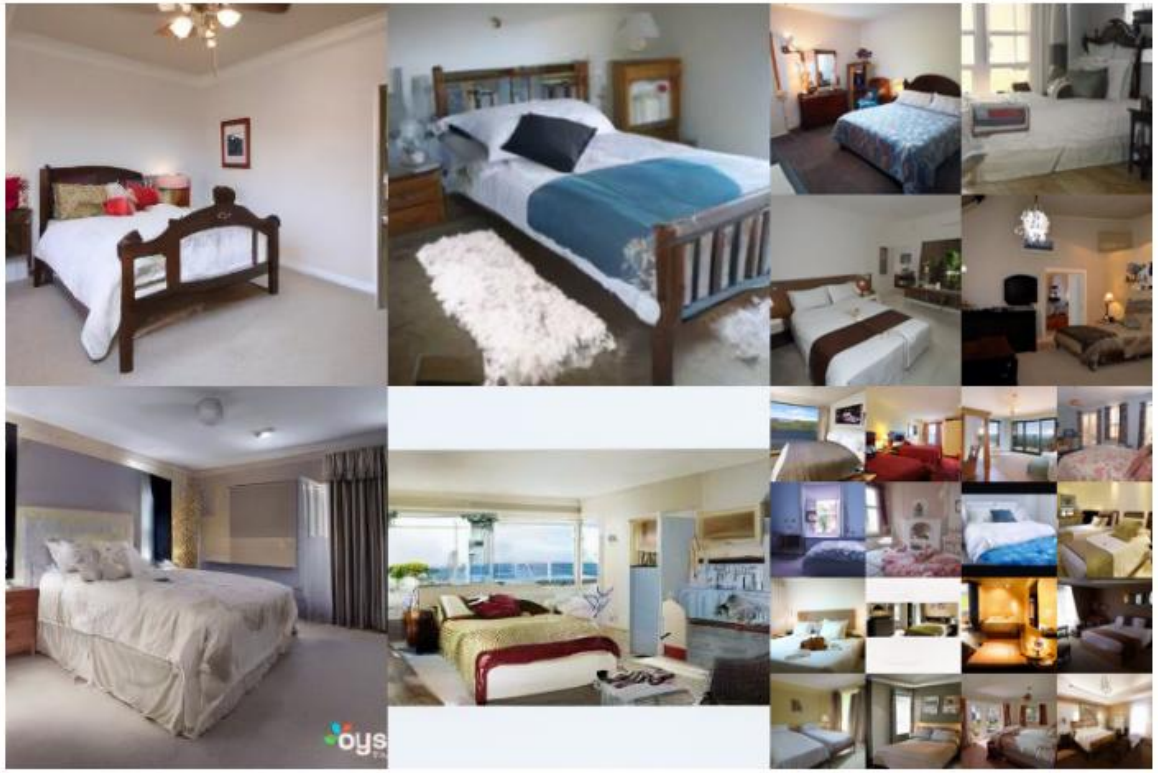


Figure 4: LSUN Bedroom samples. FID=4.90

Text-to-Image Diffusion Models

- Sampling from a diffusion model generates a random image from the distribution from with the model is training on
- In text-to-image generation we wish guide the generation process using text prompts
- These models are called conditional diffusion models, since generation probability distribution is conditioned on the text prompts
- Example models include Stable Diffusion from Stability AI, DALL-E from OpenAI, Imagen from Google

Imagen: Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

- Saharia et al, 2022
- Approach
  - Use pretrained transformer-based LLM (T5) to encode text prompt
  - Use conditional diffusion model to generate small images
  - Use cascaded diffusion models to generate larger images
- Can generate photorealistic images as well as images various styles, such as oil painting, illustration, surreal, …

Sample Imagen Images

Sprouts in the shape of text 'Imagen' coming out of a fairytale book.

A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.

A high contrast portrait of a very happy fuzzy panda dressed as a chef in a high end kitchen making dough. There is a painting of flowers on the wall behind him.

Samples From the Same Prompt

A brown bird and a blue bear.

One cat and two dogs sitting on the grass.

A sign that says 'NeurIPS'.

A small blue book sitting on a large red book.

A blue coloured pizza.

A wine glass on top of a dog.

A pear cut into seven pieces arranged in a ring.

A photo of a confused grizzly bear in calculus class.

A small vessel propelled on water by oars, sails, or an engine.

Imagen Styles

Add style to pixelated imnages

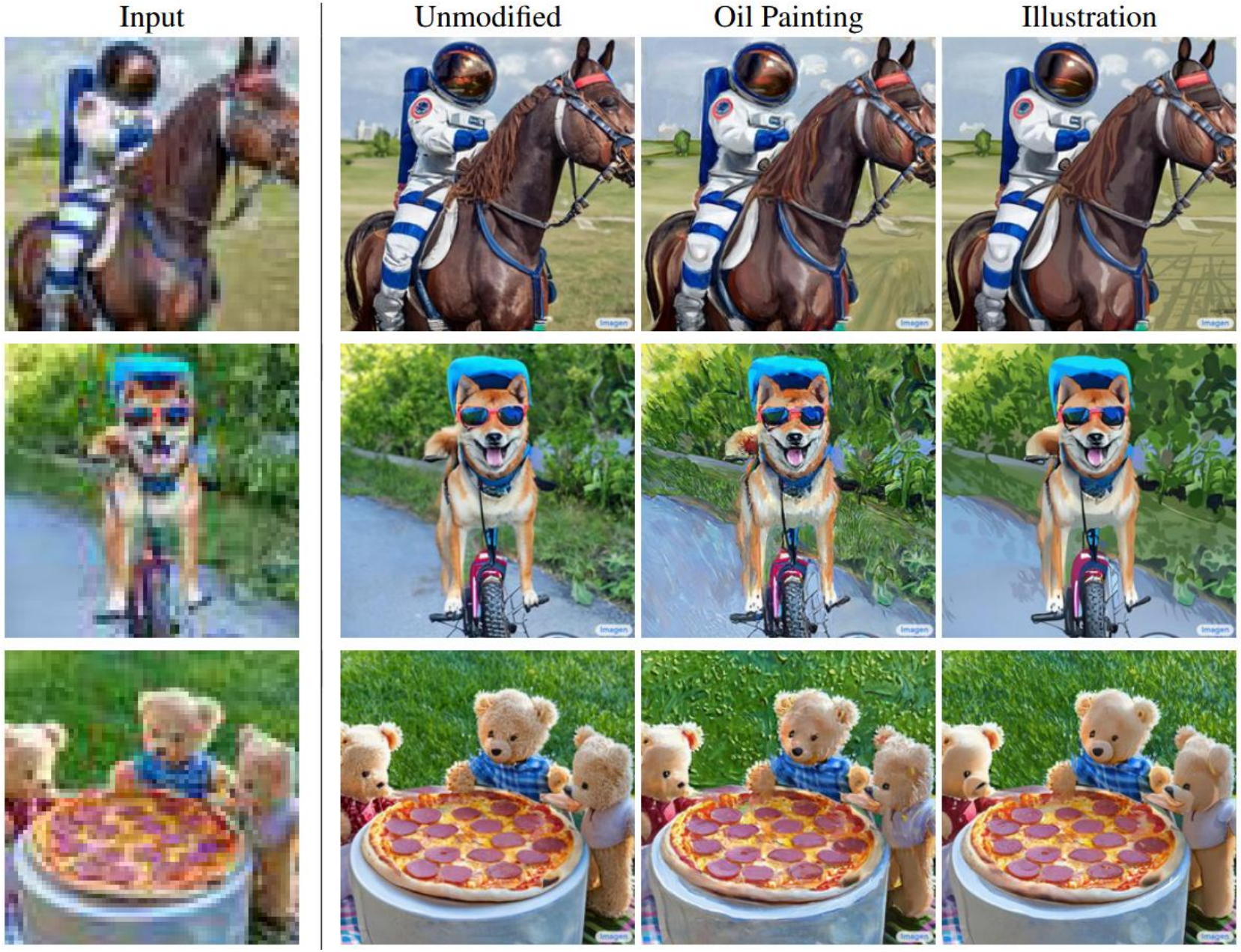
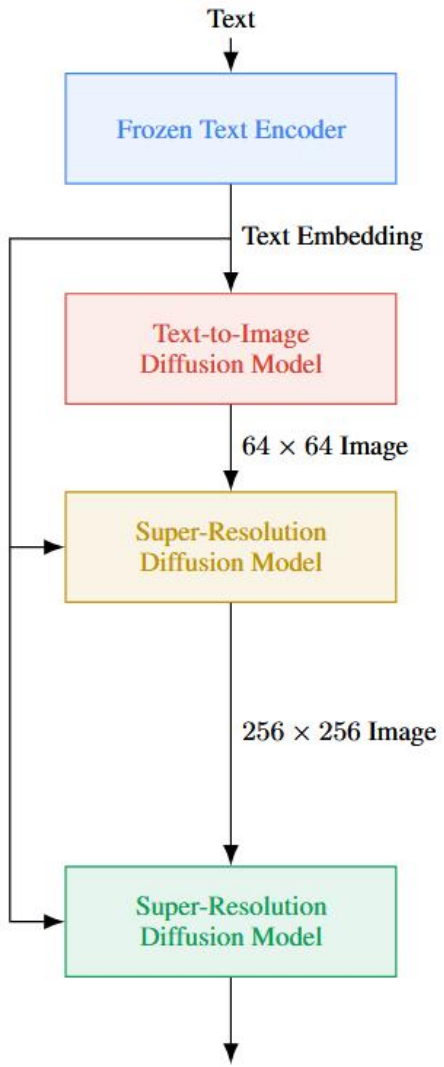


Imagen Generation Process

- Encode text prompt as an embedding vector with frozen text encoder
- Generate small 64x64 image using text-to-image diffusion model
- Generate larger images using super-resolution diffusion model



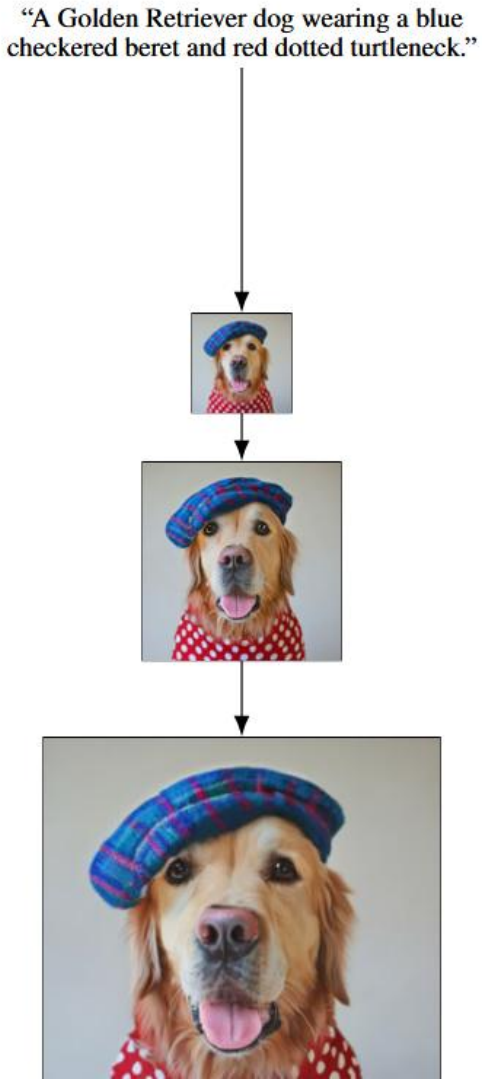
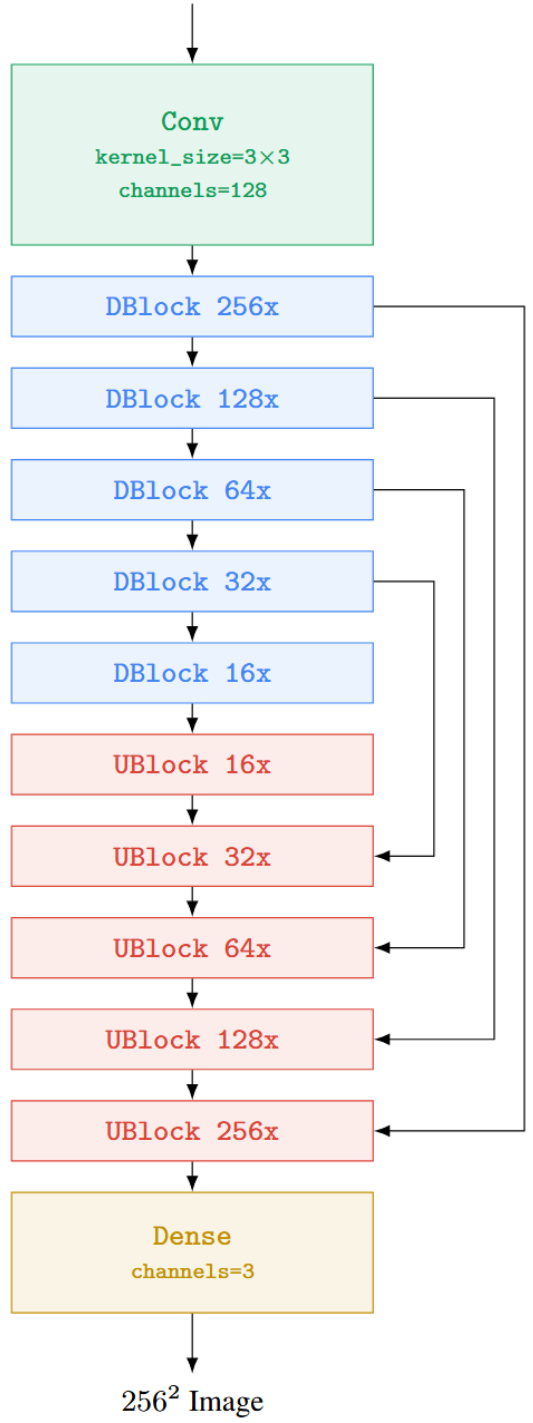


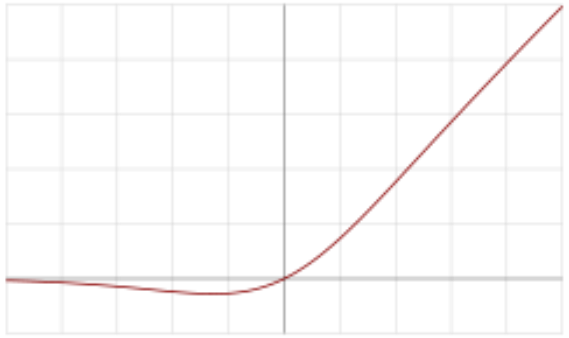
Imagen Architecture

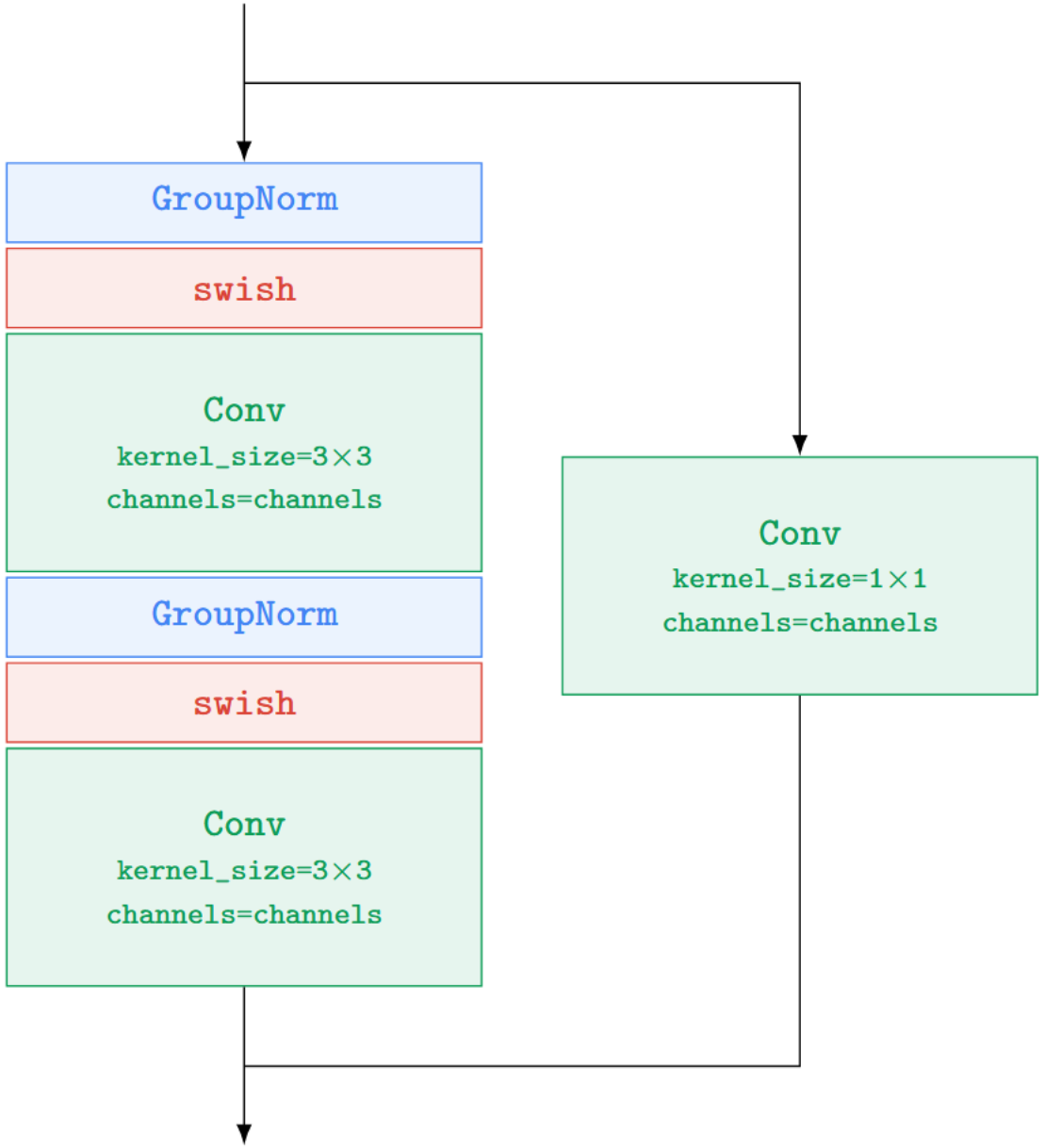
- Text-to-Image is based on U-Net
- Four pairs of down/up blocks
- Blocks have lower resolution, which speed up computation



ResNet Blocks

- Down/up blocks are based on ResNet blocks
- Use group normalize, which is designed for small batches
- Use swish activation function

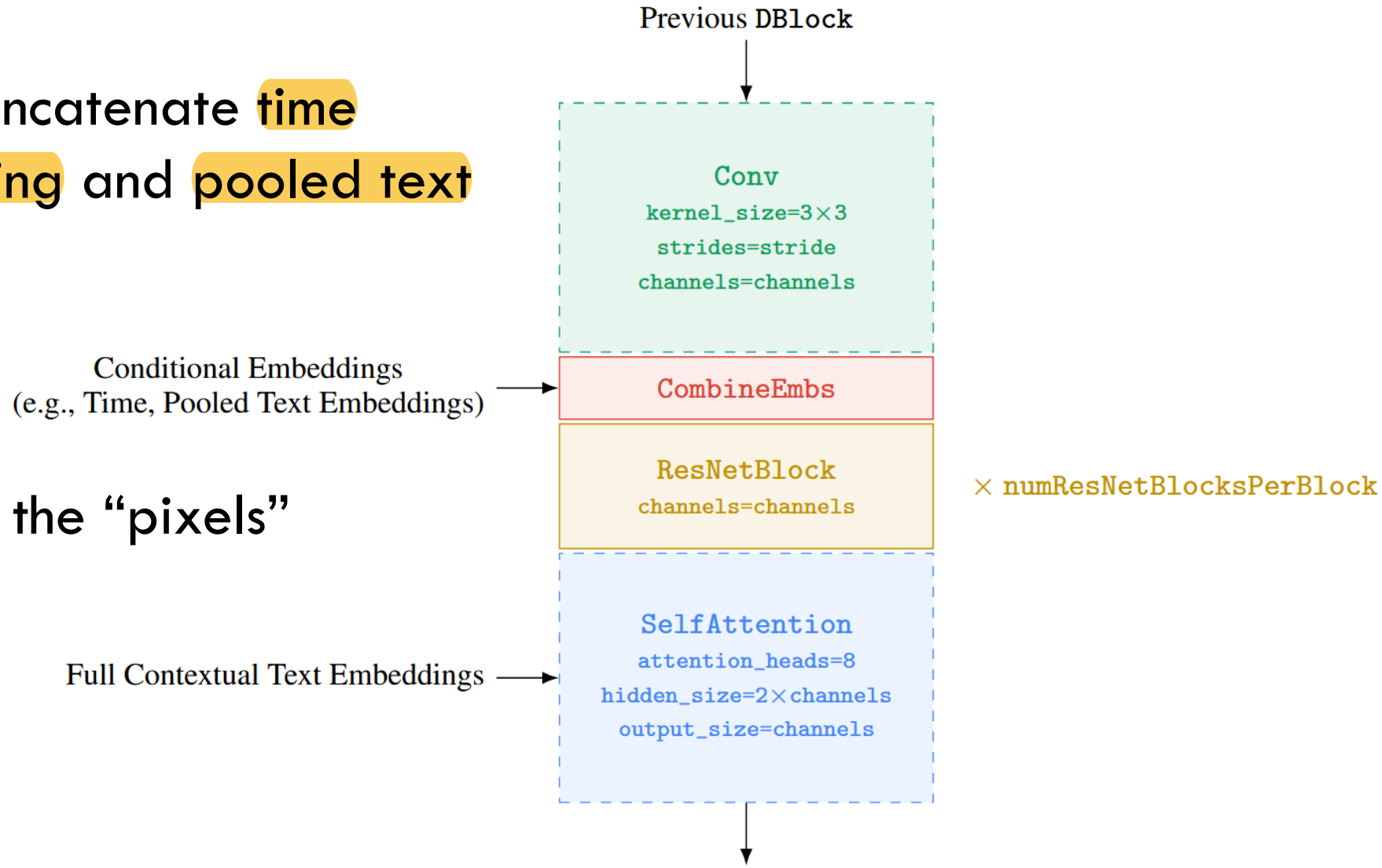




Down Block

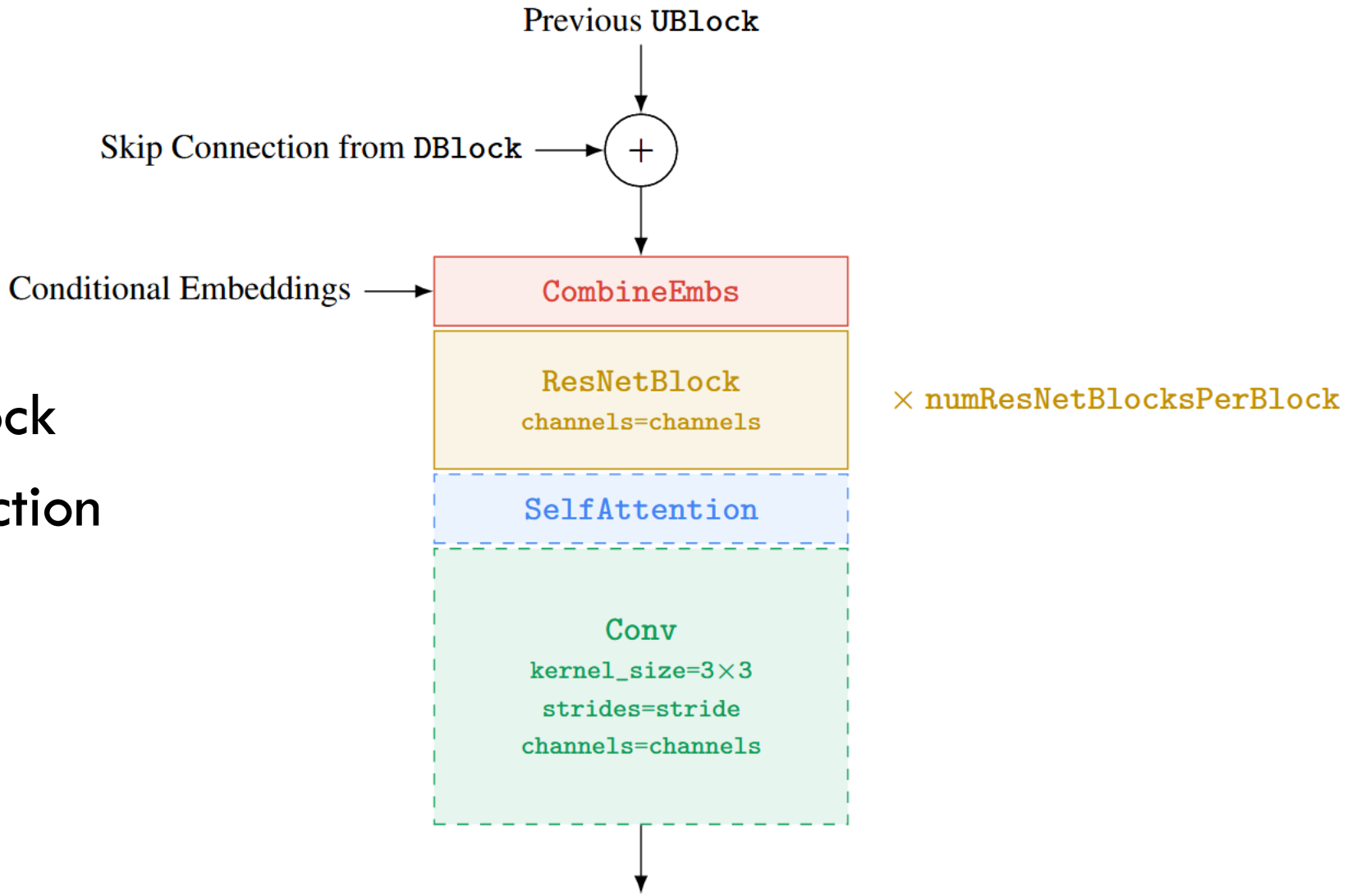
- CombineEmbs concatenate time positional encoding and pooled text embedding

- Self attention on the “pixels”



Up Block

- Similar to Down Block
- Includes skip connection



Normalization Methods

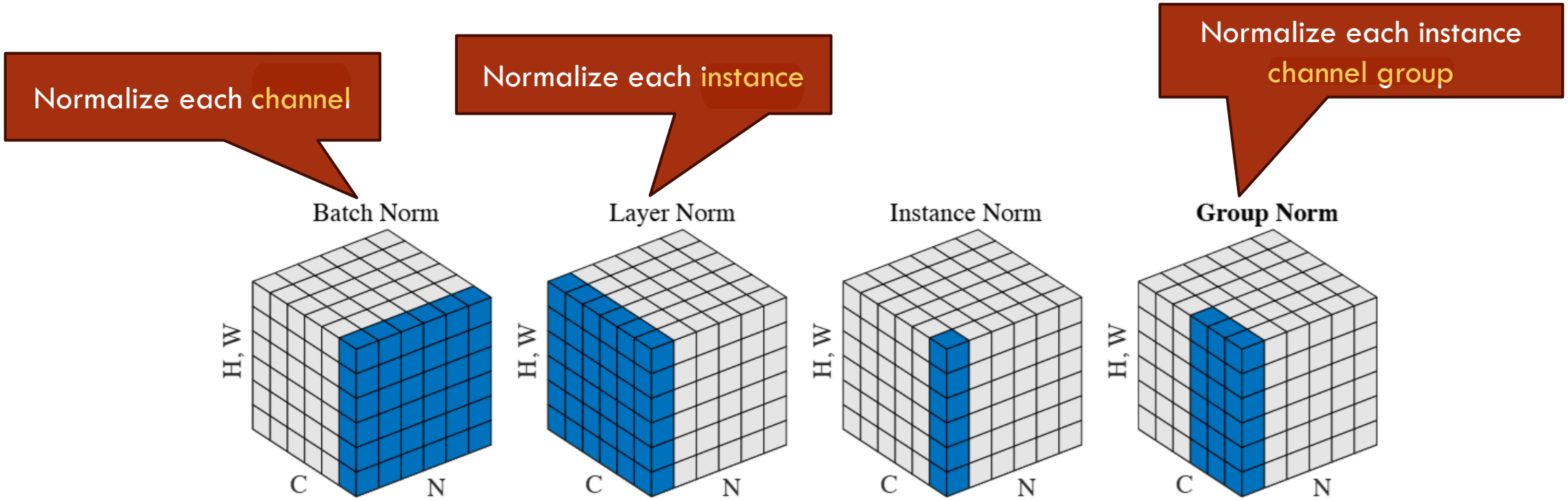


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Wu and He, 2018

Normalization Method Comparison

Batch size: 32 Images

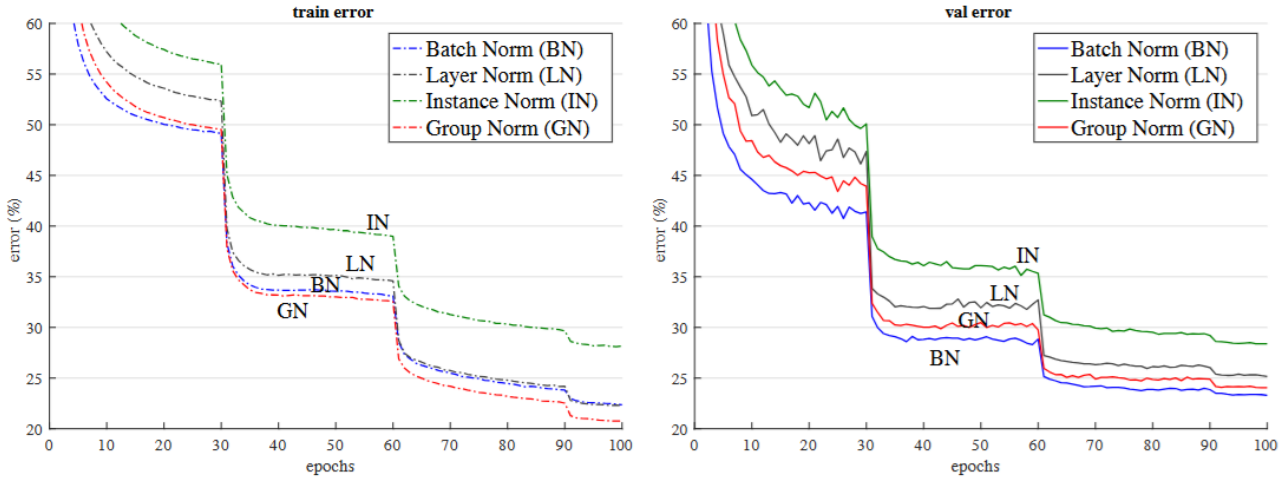


Figure 4. **Comparison of error curves** with a batch size of **32 images**/GPU. We show the ImageNet training error (left) and validation error (right) *vs.* numbers of training epochs. The model is ResNet-50.

Deceasing batch size

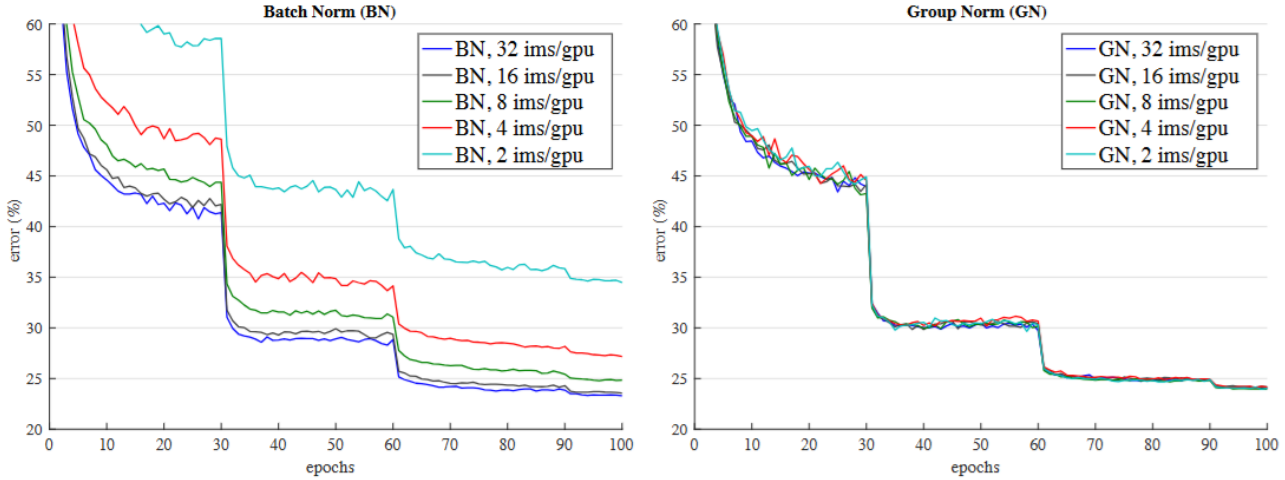


Figure 5. **Sensitivity to batch sizes**: ResNet-50's validation error of BN (left) and GN (right) trained with 32, 16, 8, 4, and 2 images/GPU.