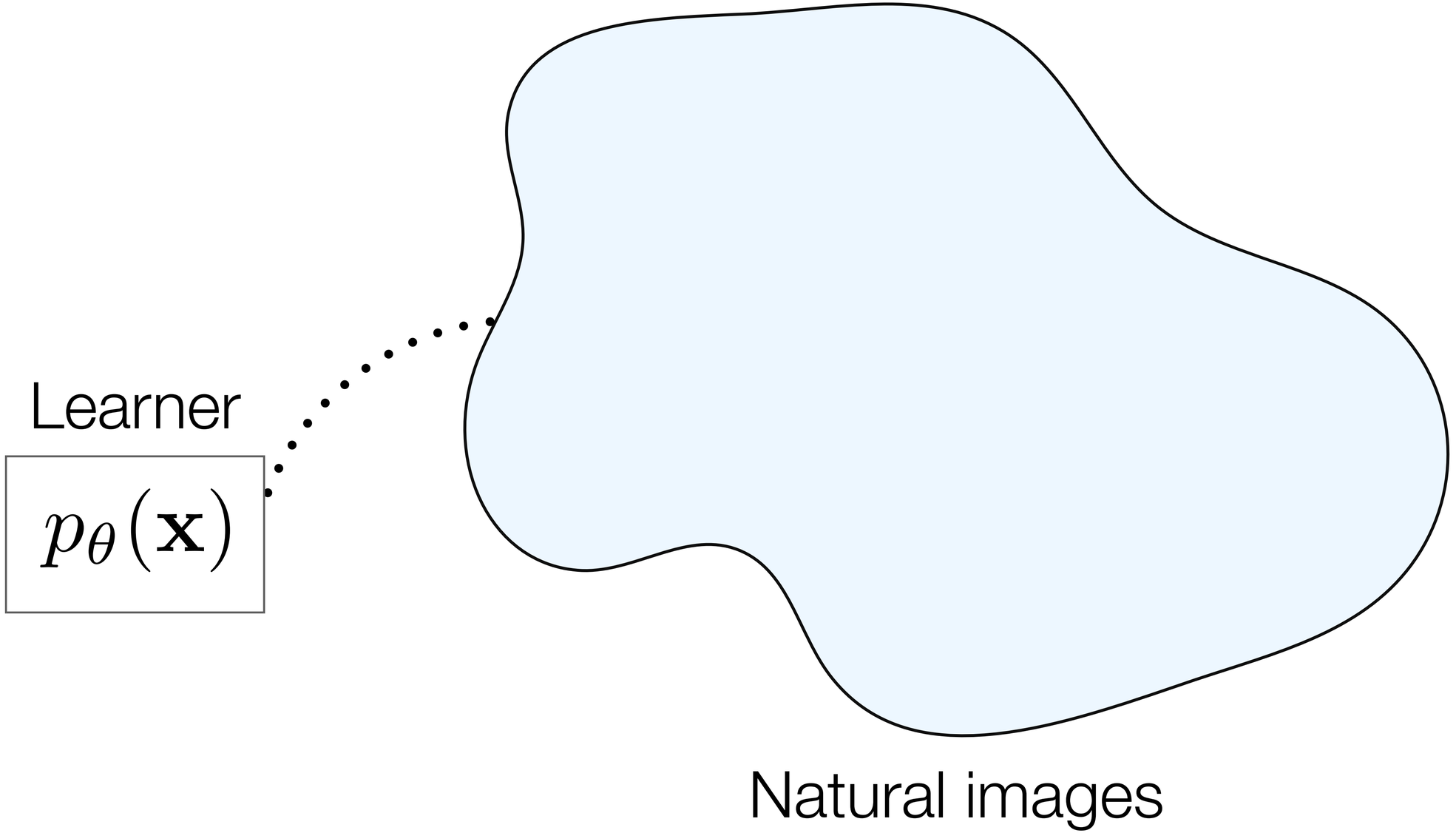
Lecture 2: Variational Autoencoders

Announcements

- First sign-up sheet is up!
- Please sign up by **Tuesday, 11:59pm**

Beginning of lectures in early sign-up period			
Lec. 4	Mon, Feb 1	**GANs**	• Donahue & Simonyan: Large Scale Adversarial Representation Learning • Bau et al:, Rewriting a deep generative model
Lec. 5	Wed, Feb 3	**Normalizing flows**	• Dinh et al.: Density Estimation using Real NVP • Kingma & Dhariwal, Glow: Generative flow with invertible 1x1 convolutions
Lec. 6	Mon, Feb 8	**Autoregressive models**	• van den Oord et al: Pixel-CNN • Chen et al.: Generative pretraining from pixels
Discriminative methods for unsupervised learning			
Lec. 7	Wed, Feb 10	**Contrastive learning**	• He et al.: Momentum Contrast for Unsupervised Visual Representation Learning • Grill et al, Bootstrap your own latent: A new approach to self-supervised Learning • Chen et al: A Simple Framework for Contrastive Learning of Visual Representations (optional) • van den Oord et al: Representation Learning with Contrastive Predictive Coding (optional) • Tian et al: Contrastive Multiview Coding (optional)
Lec. 8	Mon, Feb 15	**Pretext tasks**	• Piergiovanni et al: Evolving Losses for Unsupervised Video Representation Learning • Are Labels Necessary for Neural Architecture Search?
Learning from non-visual signals			
Lec. 9	Wed, Feb 17	**Language & vision**	• Desai & Johnson: VirTex: Learning Visual Representations from Textual Annotations • Radford et al: Learning Transferable Visual Models From Natural Language Supervision
End of lectures in early sign-up period			

Learning by generating



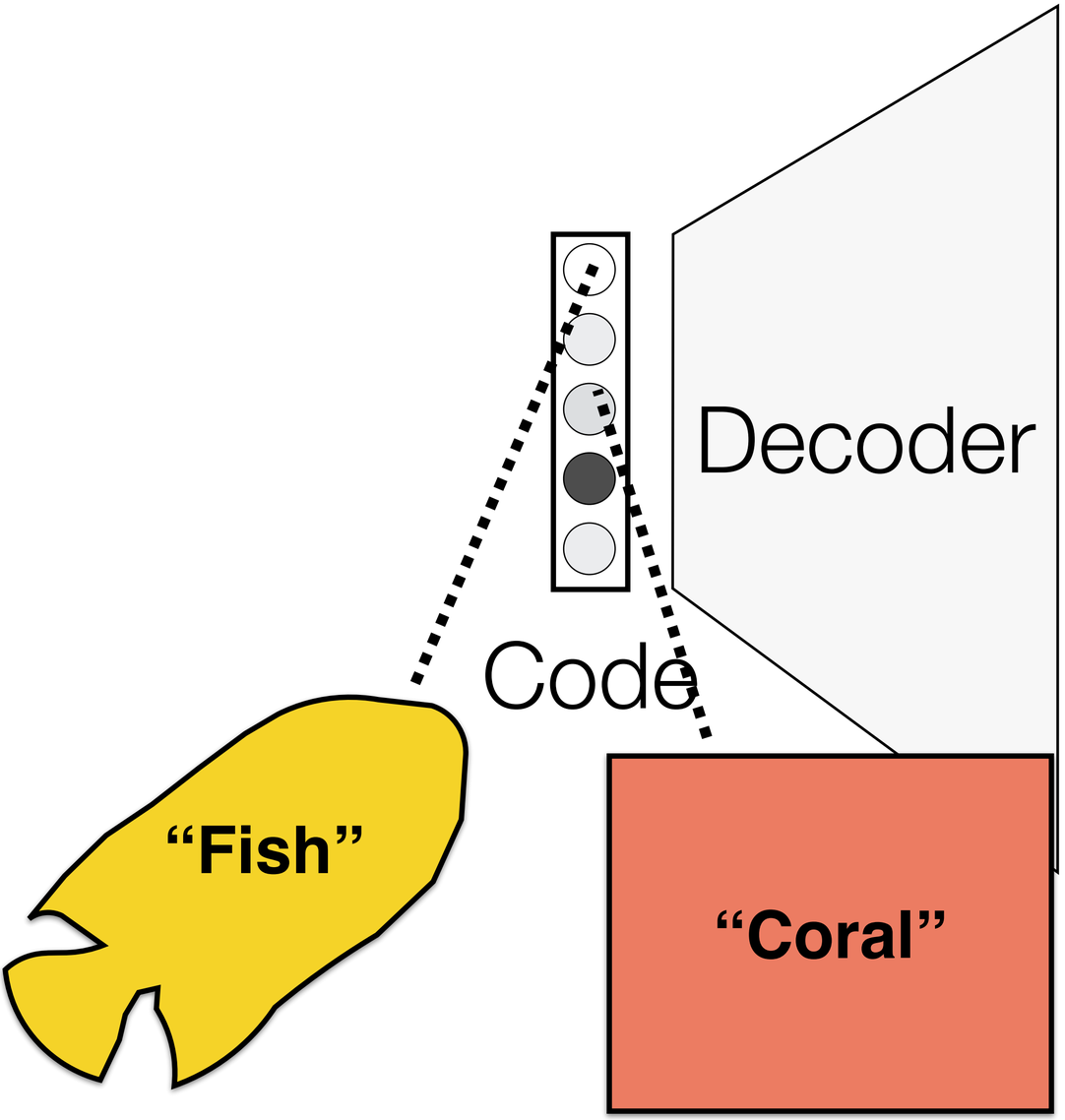
What can a model learn by generating pictures?

What can a model learn by generating pictures?

- Underlying “physical causes” of an image
  - Objects, scenes, lighting, etc.
- These physical cases can then be used for other things
- Closely related to an efficient code
  - If you could communicate images in a small number of bits, you implicitly have a probabilistic model

Source: [Hinton & Sejnowski 1999]

From codes to images



Sampled image

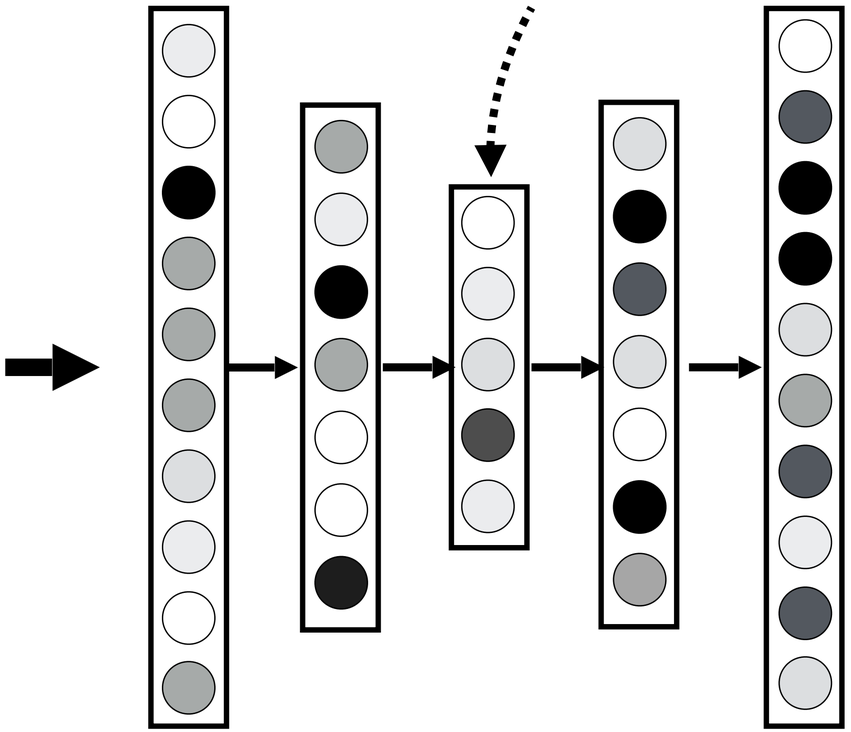
How do we get this code?

Idea #1: autoencoder

compressed image code
(vector **z**)

$\mathbf{x}$

Image



$\hat{\mathbf{x}}$

Reconstructed image

[e.g., Hinton & Salakhutdinov, Science 2006]

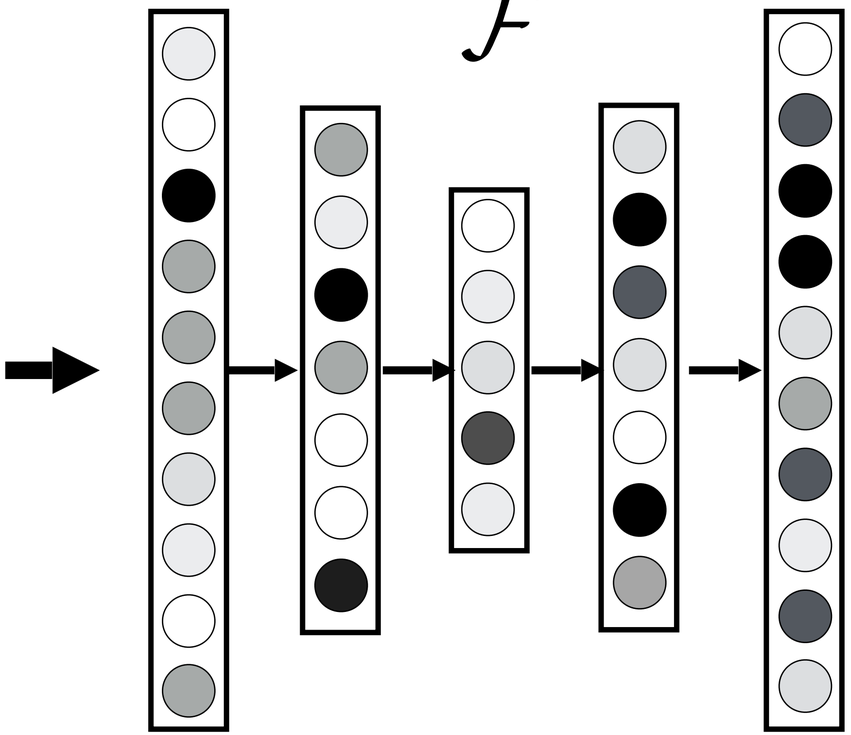
Source: Isola, Freeman, Torralba

Autoencoder

$\mathbf{X}$

Image

$\mathcal{F}$



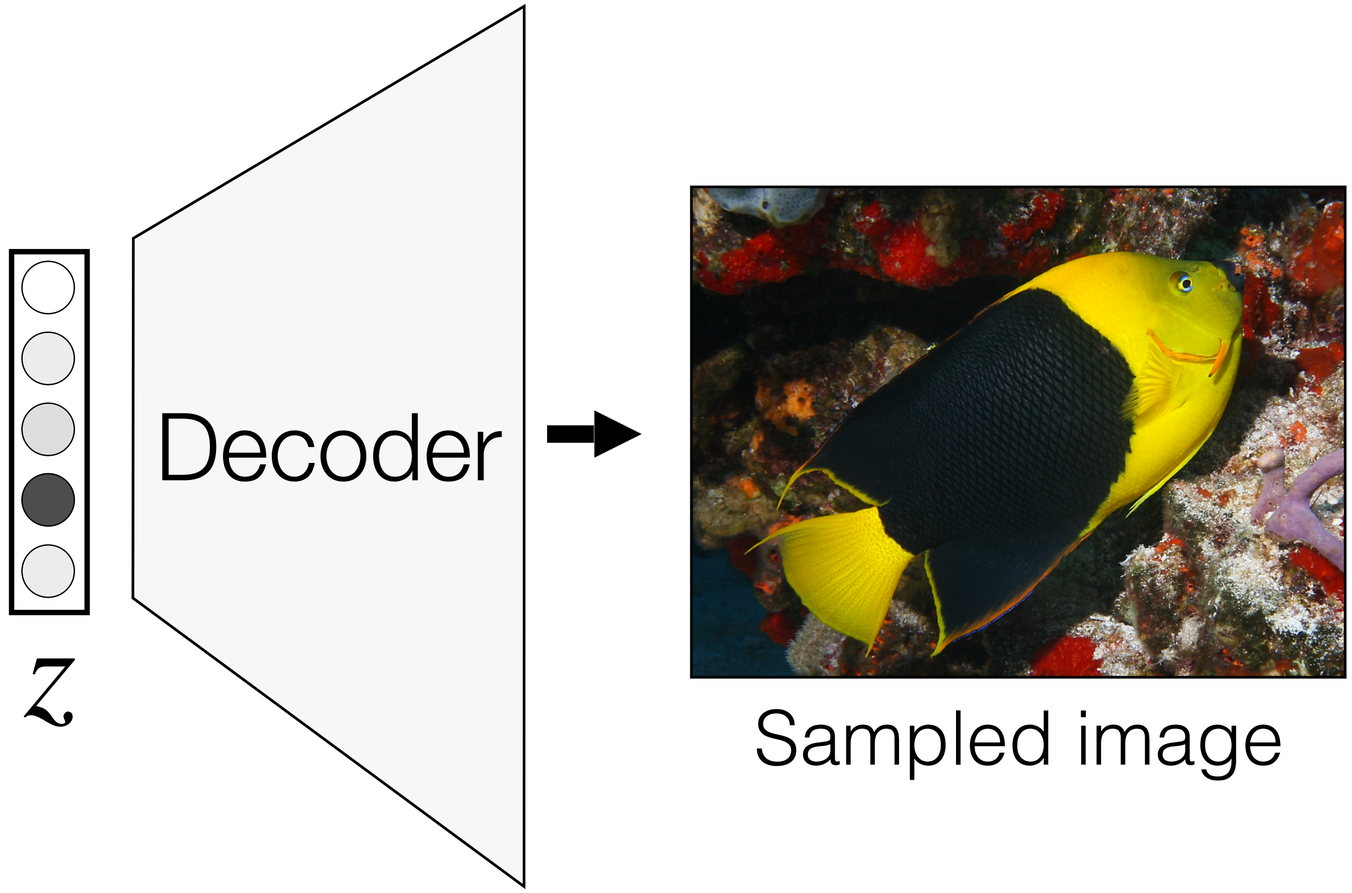
$\hat{\mathbf{X}} = \mathcal{F}(\mathbf{X})$

Reconstructed image

$$\arg\min_{\mathcal{F}} \mathbb{E}_{\mathbf{X}}[||\mathcal{F}(\mathbf{X}) - \mathbf{X}||]$$

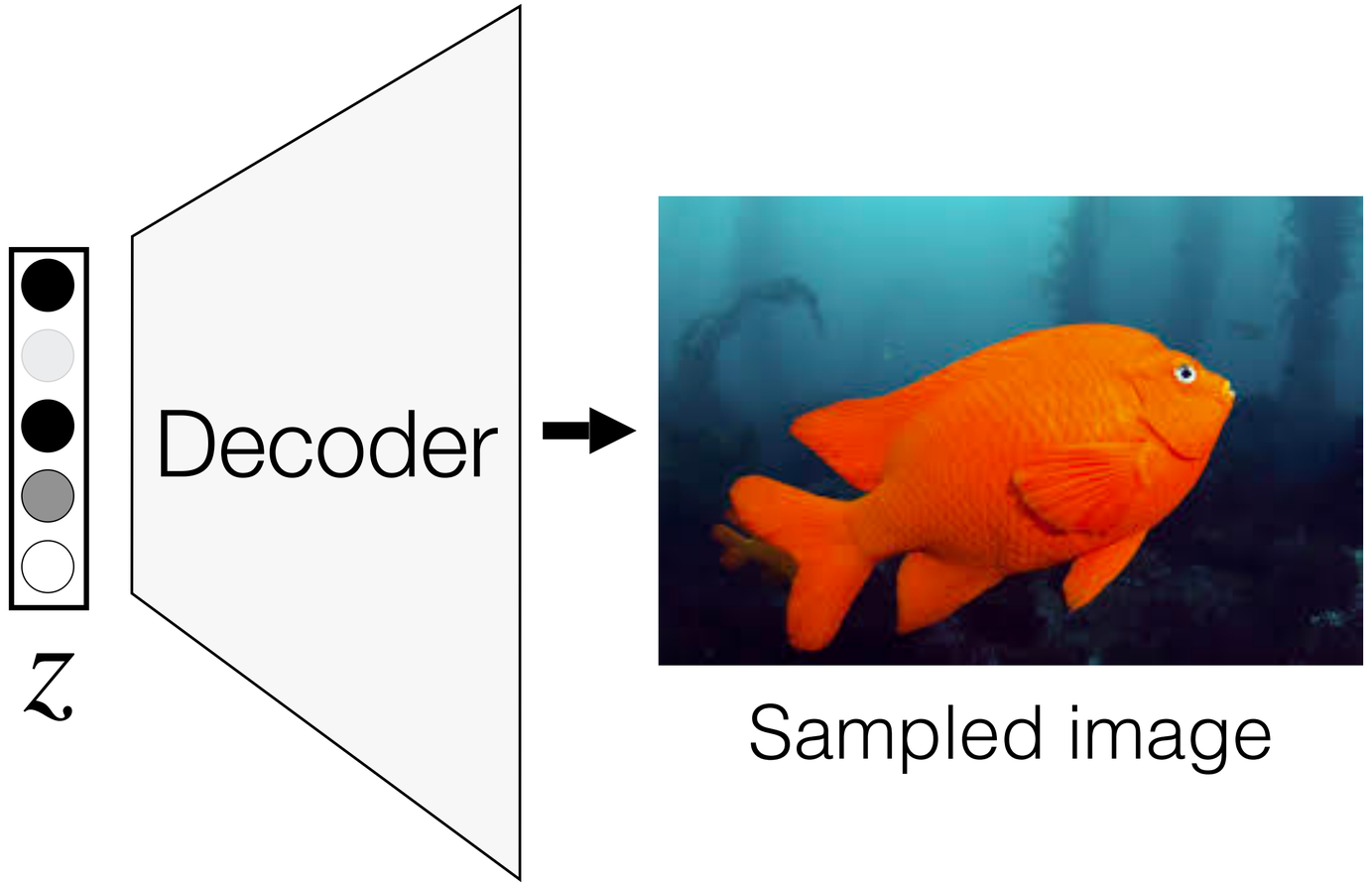
Source: Isola, Freeman, Torralba

Sampling an image



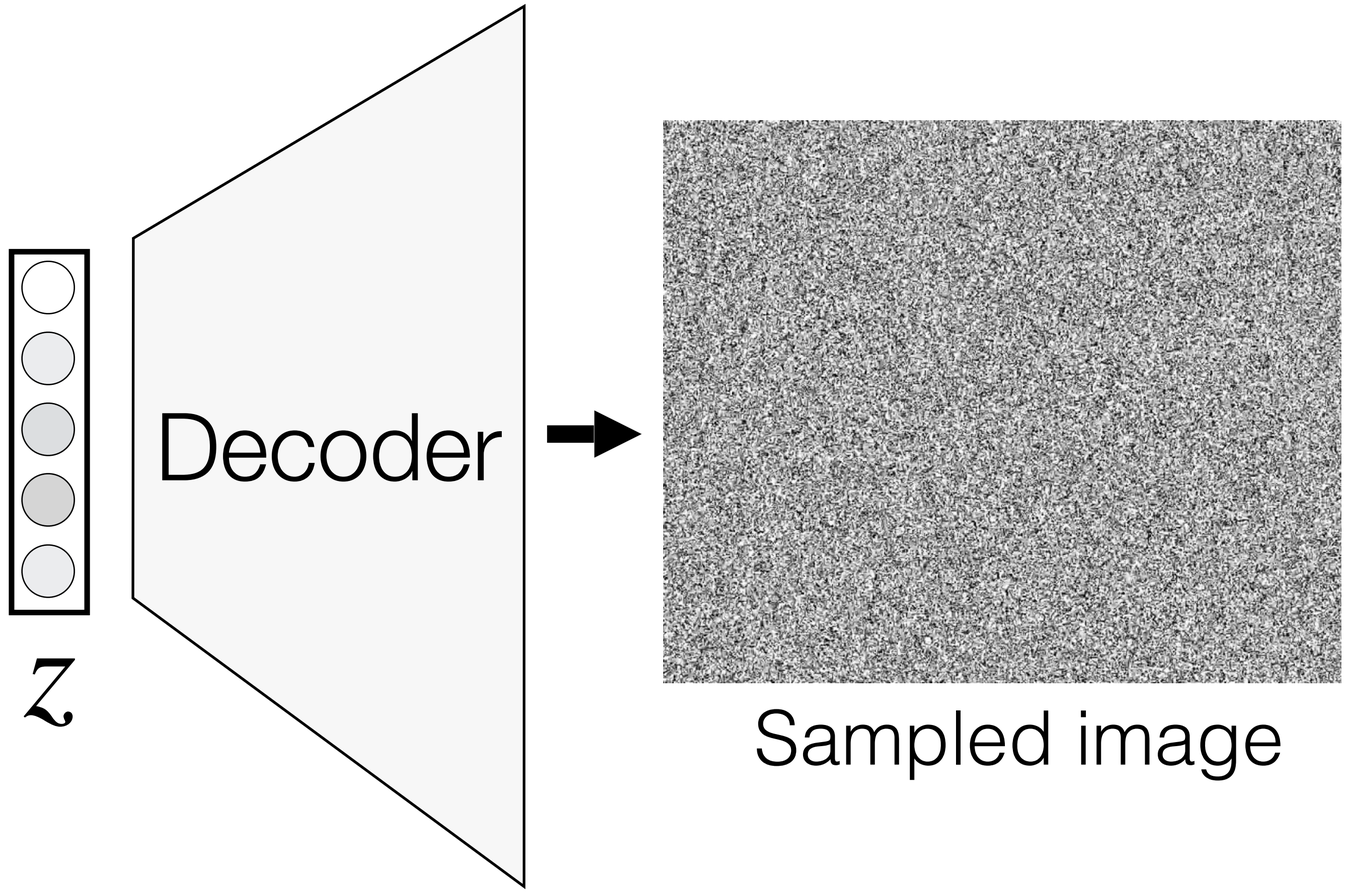
What if we just plug in random vectors?

Sampling an image



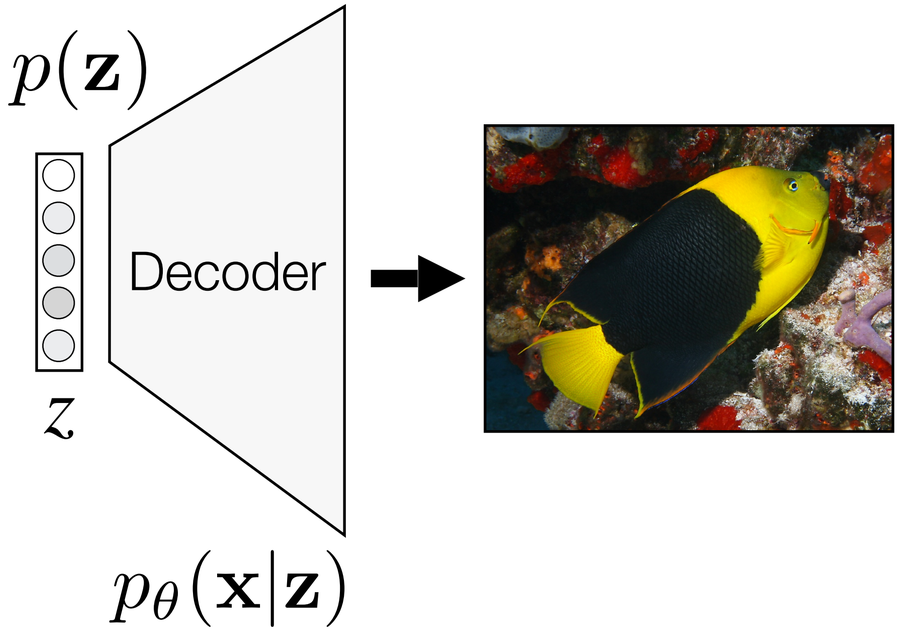
What if we just plug in random vectors?

Sampling an image



However, no guarantee that the latent space covers the space of images…

Idea #2: think probabilistically



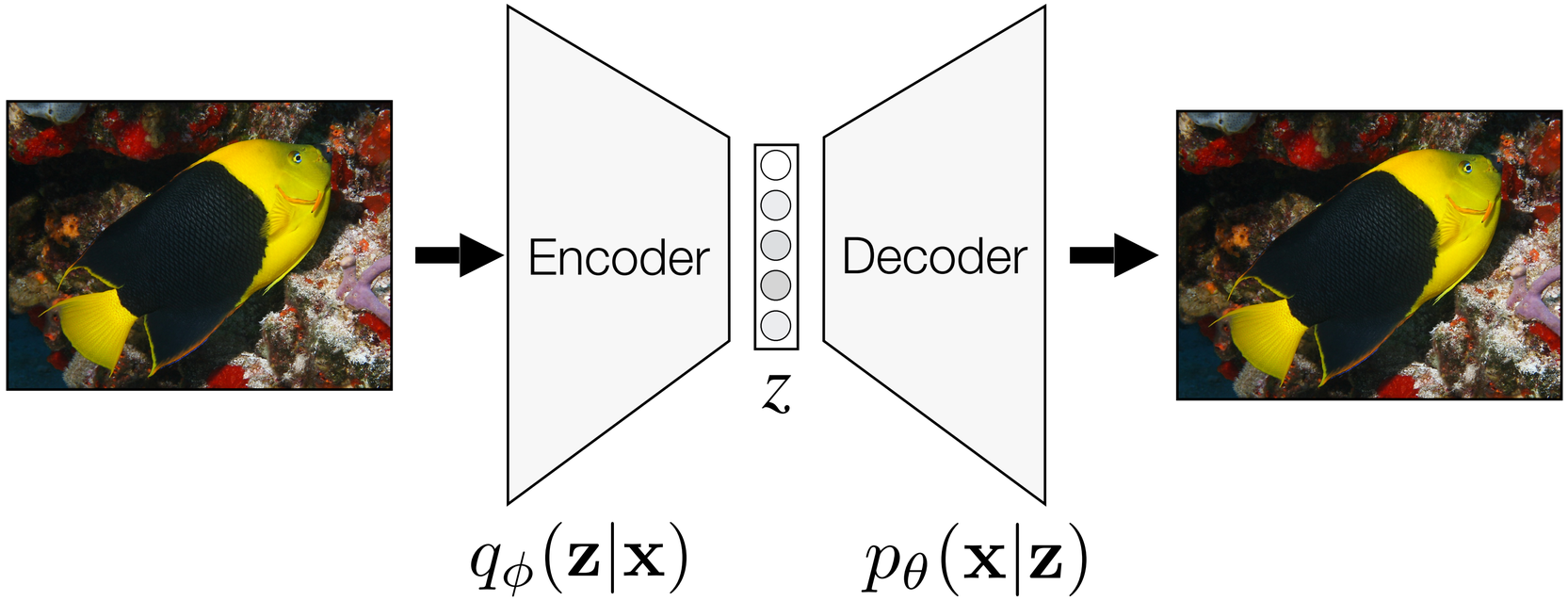
Probability of an example:

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z})dz$$

$$= \mathbb{E}_{\mathbf{z}}\left[p(\mathbf{x}|\mathbf{z})\right]$$

But this is really hard. How do you find z, given x?

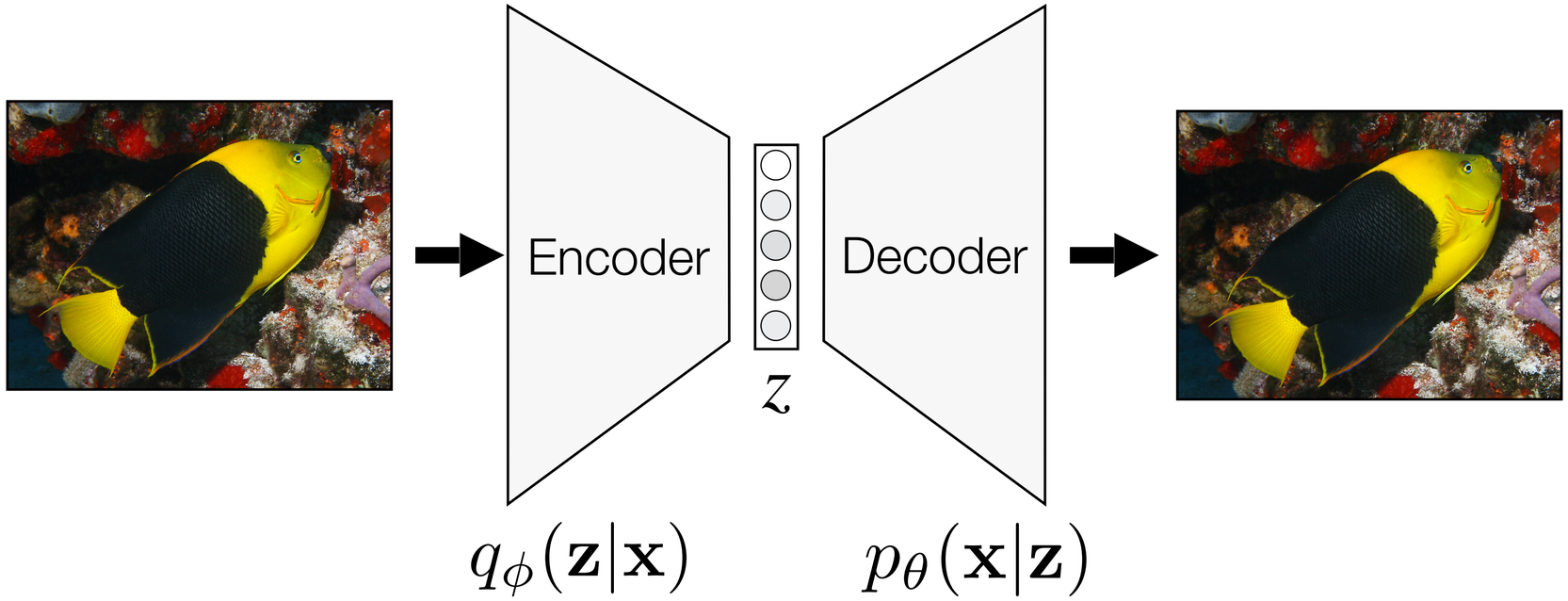
Variational autoencoder



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

ELBO: **e**vidence **l**ower **bo**und

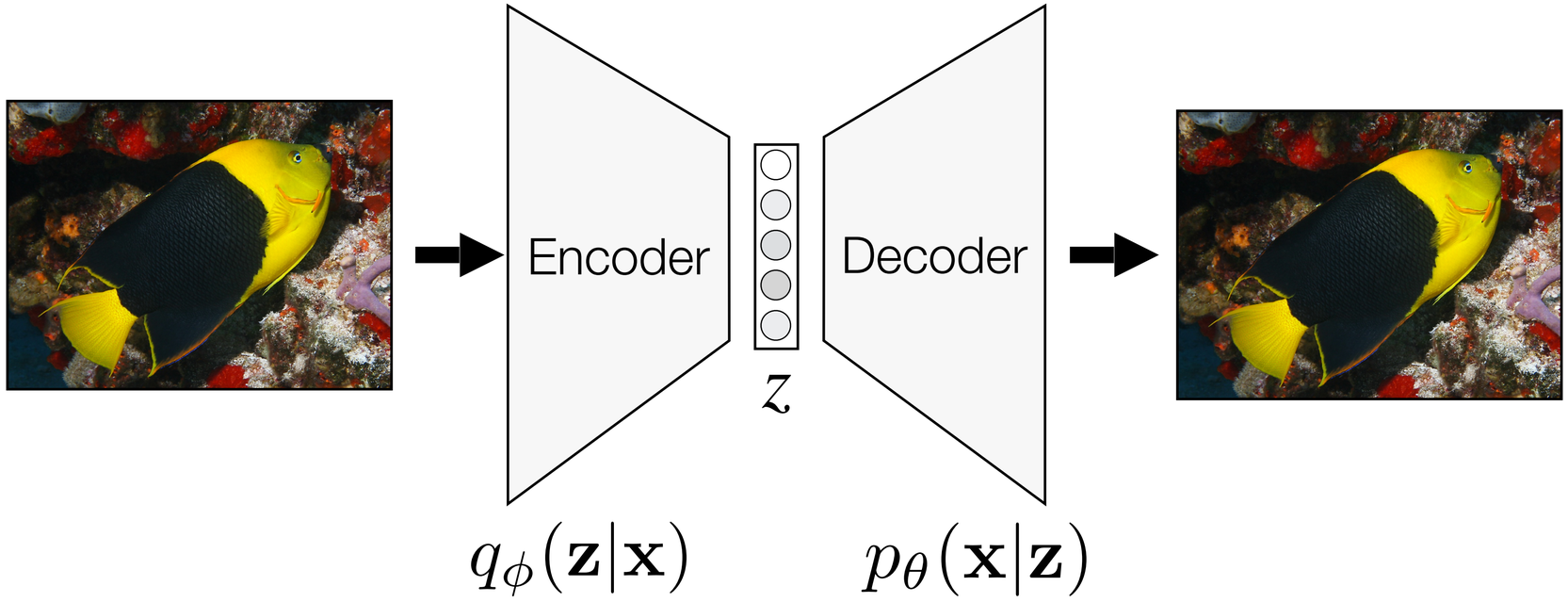
Variational autoencoder



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

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

Variational autoencoder



$q_\phi(\mathbf{z}|\mathbf{x})$ usually implemented as Gaussian with params: $\mu_\phi(\mathbf{x}),\ \sigma_\phi(\mathbf{x})$

Convenient, since we can use the **reparameterization trick** to sample z:

$\tilde{z} = g_\phi(\epsilon, \mathbf{x})$ where $\epsilon \sim \mathcal{N}(0, I)$ and $g_\phi(\epsilon, \mathbf{x}) = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x})\epsilon$

Uses for VAEs

- Image generation
  - Especially *diverse* image generation
- Image features
- An image prior

Problem: blurry images

Source: Alec Radford

Generating Diverse High-Fidelity Images with VQ-VAE-2

Ali Razavi*, Aaron van den Oord*, Oriol Vinyals

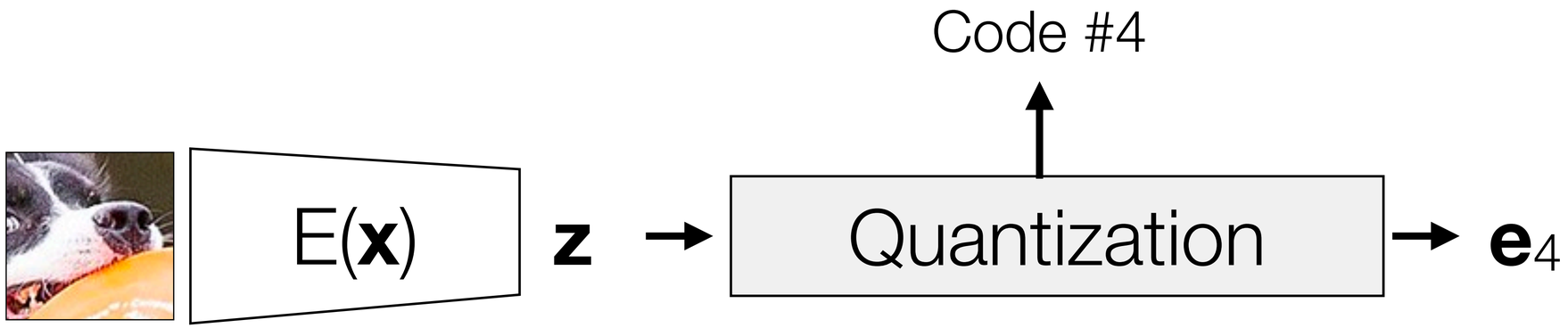
NeurIPS 2019

Discrete codes

- VAEs usually use a continuous representation for latent code, z
- But many events in the world are *discrete*.
  - E.g. can sometimes describe images concisely as collection of objects
- Key idea: replace Gaussian code with categorical code

Vector quantization

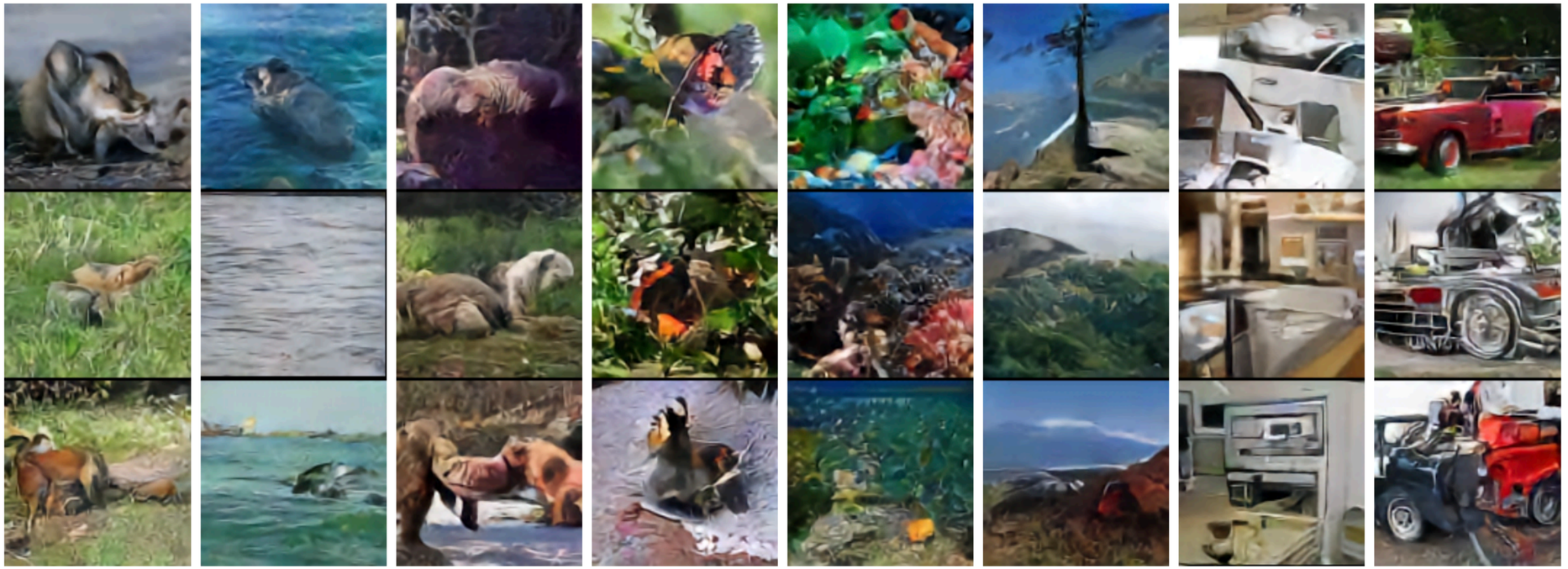
Predict a real-valued vector as usual.
Then, “snap” it to a nearest neighbor from a codebook



$$\textbf{Quantize}(E(\mathbf{x})) = \mathbf{e}_k \quad \textbf{where}\, k = \arg\min_j ||E(\mathbf{x}) - \mathbf{e}_j||$$

VQ-VAE-1

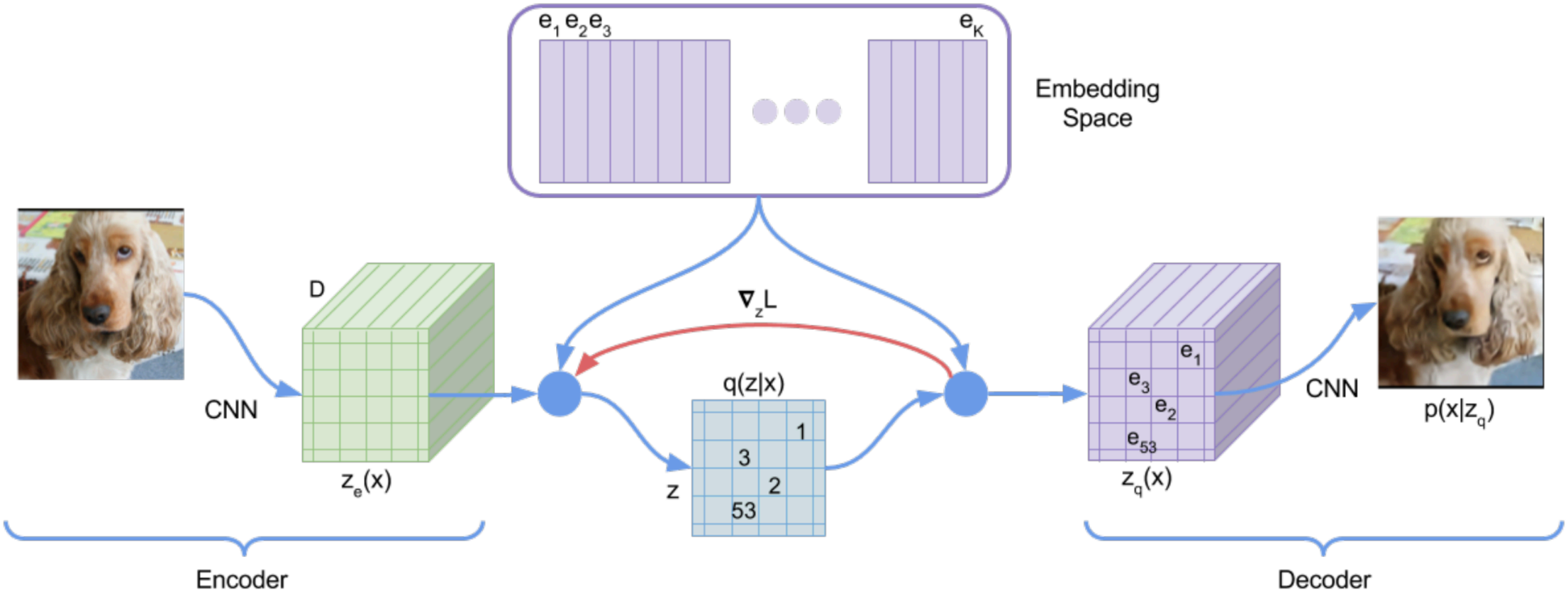
Vector quantization is key idea of previous paper [van den Oord, 2017]



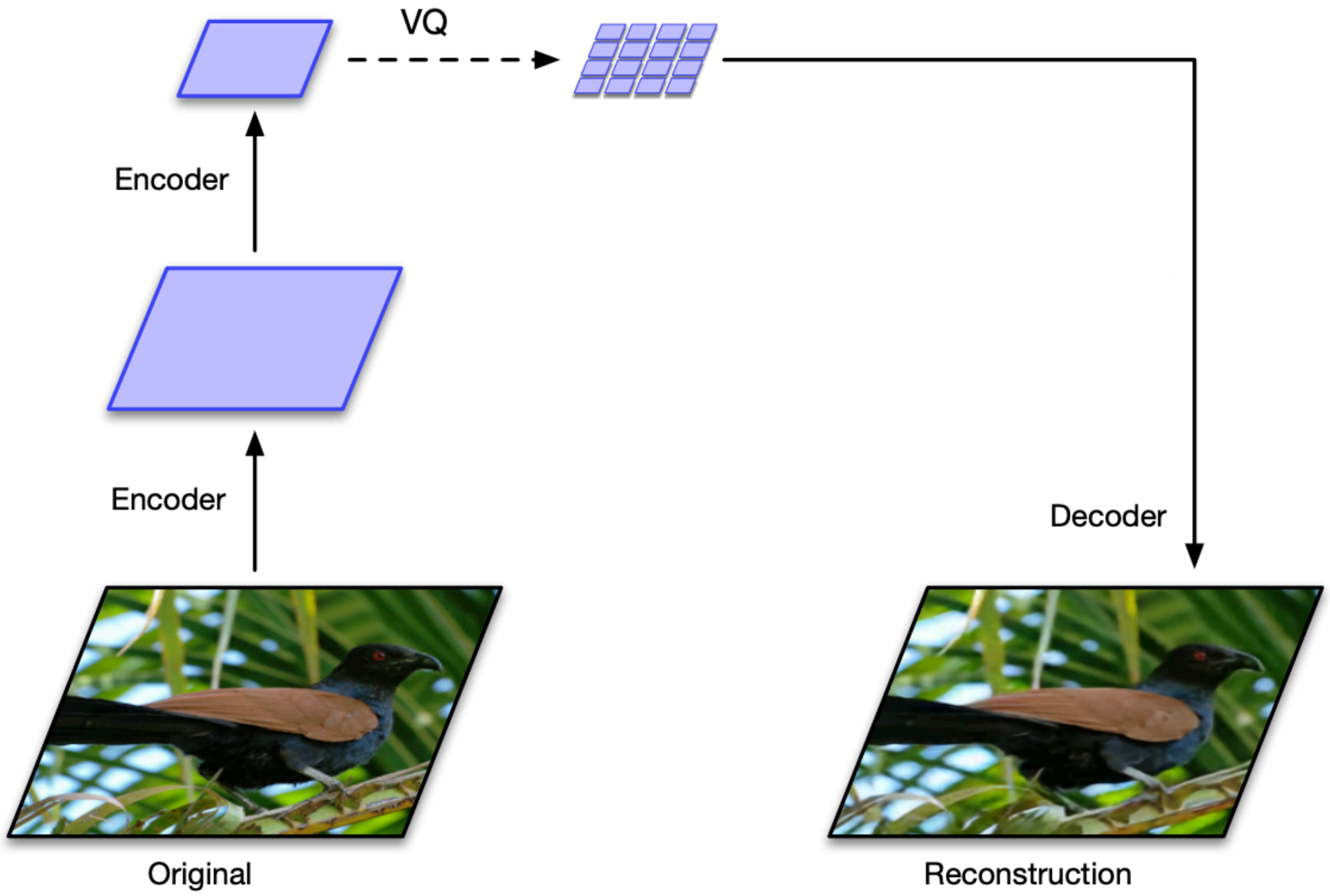
Samples from VQ-VAE-1

Pretty good samples, but not competitive with state-of-art (e.g. GANs). Similar log likelihood as standard Gaussian prior.

VQ-VAE-1


e_1 e_2 e_3
e_K
Embedding Space
D
CNN
$z_e(x)$
$\nabla_z L$
q(z|x)
1
3
2
53
z
e_1
e_3
e_2
e_{53}
$z_q(x)$
CNN
$p(x|z_q)$
Encoder
Decoder

VQ-VAE-2 architecture



VQ-VAE-2 architecture

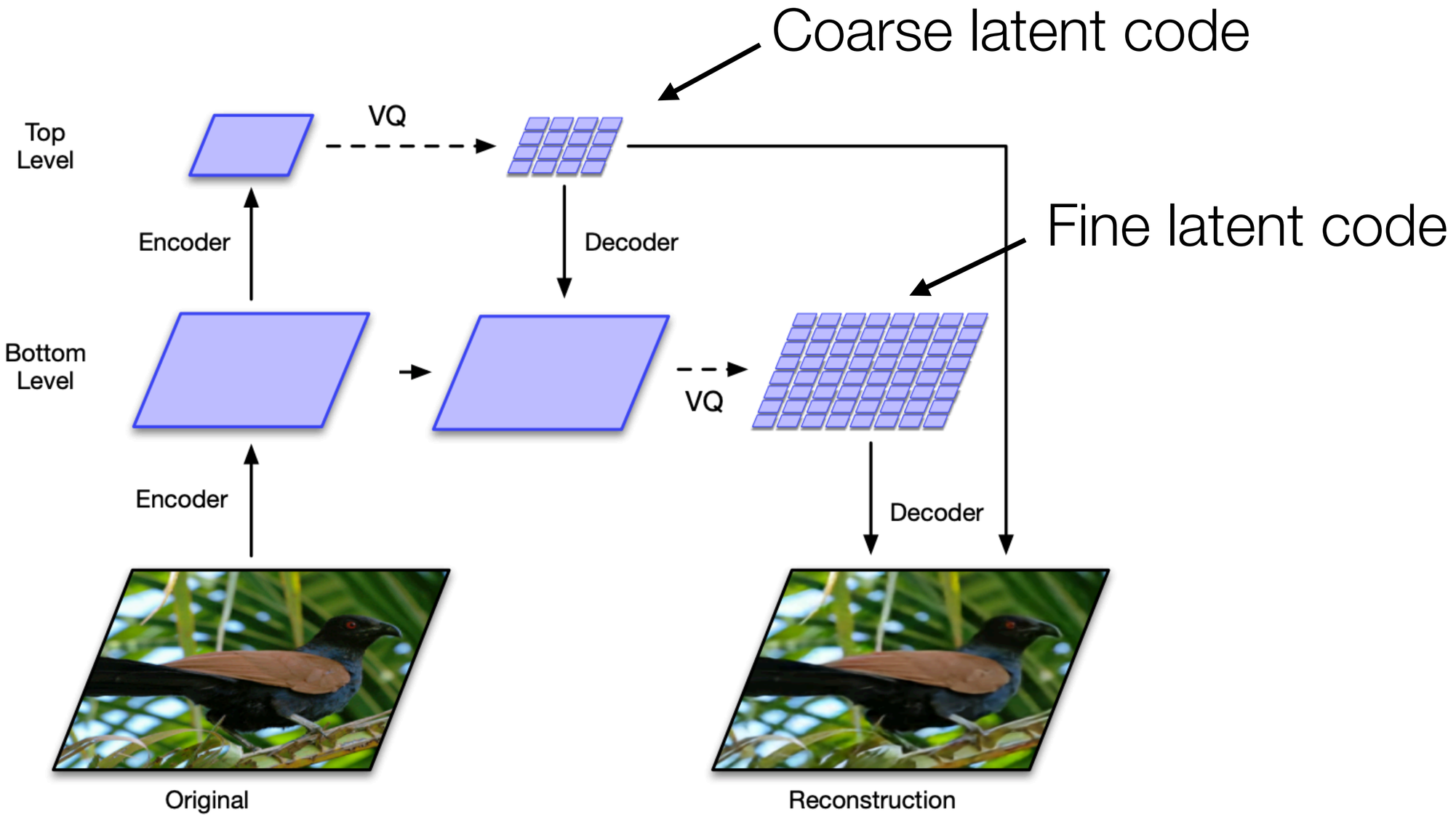


Image generation

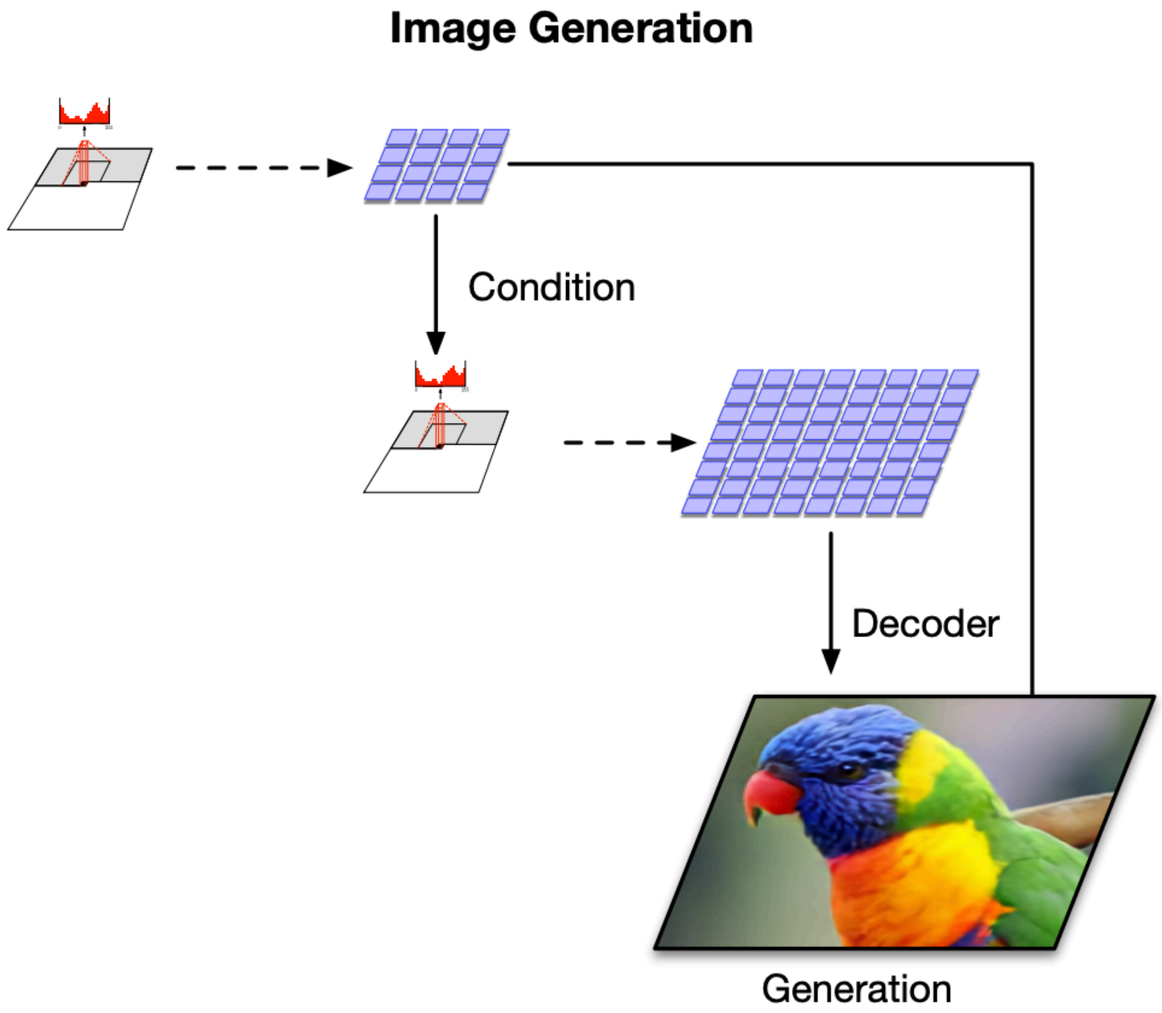


Image generation

h_{top} $h_{\text{top}}, h_{\text{middle}}$ $h_{\text{top}}, h_{\text{middle}}, h_{\text{bottom}}$ Original

Figure 3: Reconstructions from a hierarchical VQ-VAE with three latent maps (top, middle, bottom). The rightmost image is the original. Each latent map adds extra detail to the reconstruction. These latent maps are approximately 3072x, 768x, 192x times smaller than the original image (respectively).

Learning

Two stages:

1. Train the VQ-VAE.
2. Learn a better prior, p(z)

Learning

Two stages:

1. Train the VQ-VAE.

2. Learn a better prior, p(z)

Learning the VQ-VAE

$$\mathcal{L}(\mathbf{x}, D(\mathbf{e})) = ||\mathbf{x} - D(\mathbf{e})||_2^2 + ||sg[E(\mathbf{x})] - \mathbf{e}||_2^2 + \beta||sg[\mathbf{e}] - E(\mathbf{x})||_2^2$$

where:

- E(x) is the encoder and D(e) is the decoded image
- **e** is the quantized code for **x**
- sg[x] is “stop gradient” a.k.a. “detach”

Backprop through vector quantization

Hard to run backprop through this:

$$\hat{\mathbf{x}} = D(\text{quantize}(E(\mathbf{x})))$$

Why?

$$\nabla_u \text{quantize}(\text{u}) = 0$$

Backprop through vector quantization

Hard to run backprop through this:

$$\hat{\mathbf{x}} = D(\text{quantize}(E(\mathbf{x})))$$

They use the **straight-through estimator**. Just pretend $\text{quantize}(\ .\ )$ is the identity during the backwards pass!

Learning

Two stages:

1. Train the VQ-VAE.
2. **Learn a better prior, p(z)**

How do we sample from z?

- Fit an autoregressive model to the z values you get from the encoder (e.g. [Chen et al, 2017])
- Treat the latent code as an image, and fit it using a PixelCNN.

How do we sample from z?

Algorithm 2 Prior training (stage 2)

$\mathbf{T}_{top}, \mathbf{T}_{bottom} \leftarrow \emptyset$ ▷ training set
for $\mathbf{x} \in$ training set **do**
    $\mathbf{e}_{top} \leftarrow Quantize(E_{top}(\mathbf{x}))$
    $\mathbf{e}_{bottom} \leftarrow Quantize(E_{bottom}(\mathbf{x}, \mathbf{e}_{top}))$
    $\mathbf{T}_{top} \leftarrow \mathbf{T}_{top} \cup \mathbf{e}_{top}$
    $\mathbf{T}_{bottom} \leftarrow \mathbf{T}_{bottom} \cup \mathbf{e}_{bottom}$
end for
$p_{top} = \texttt{TrainPixelCNN}(\mathbf{T}_{top})$
$p_{bottom} = \texttt{TrainCondPixelCNN}(\mathbf{T}_{bottom}, \mathbf{T}_{top})$

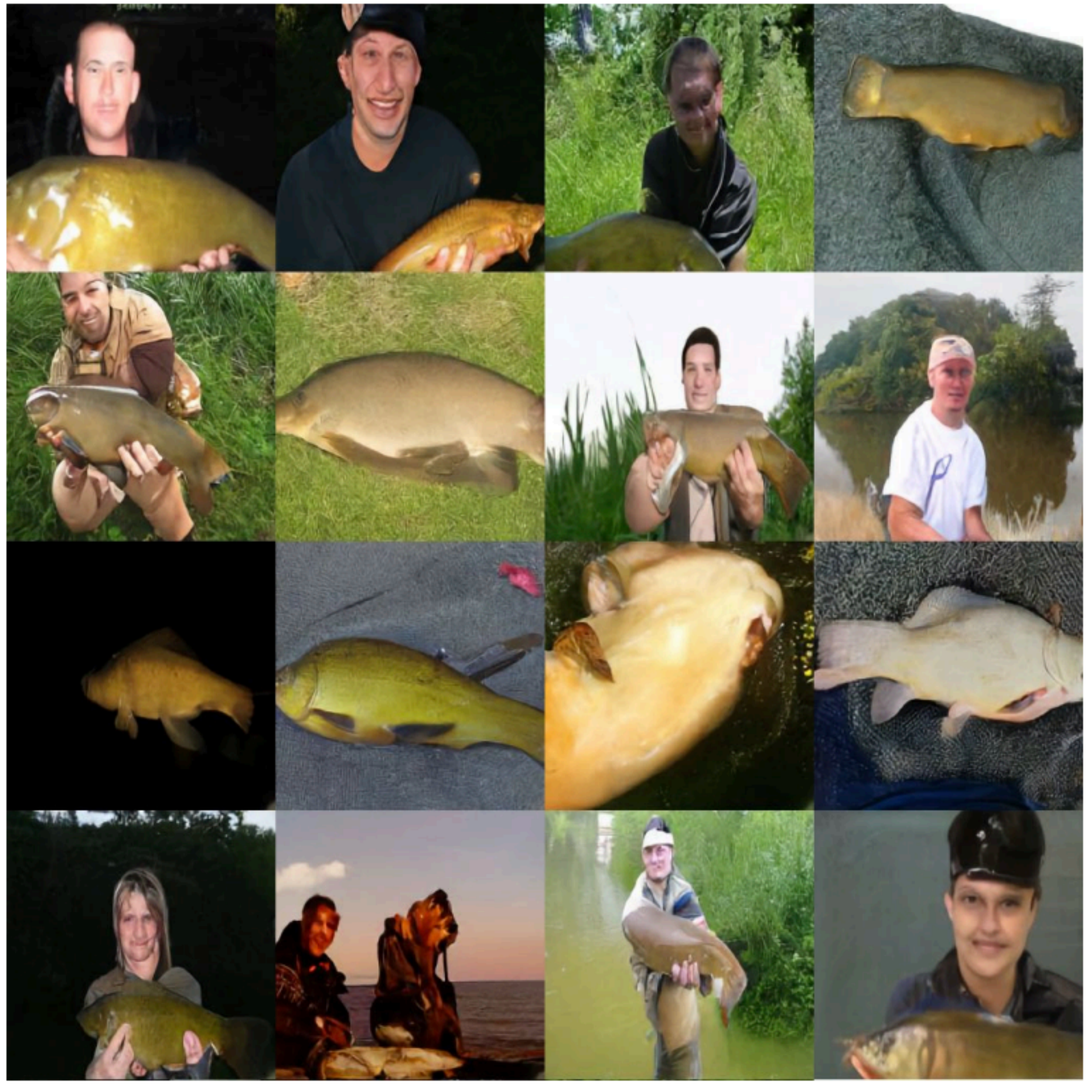
▷ Sampling procedure
while true **do**
    $\mathbf{e}_{top} \sim p_{top}$
    $\mathbf{e}_{bottom} \sim p_{bottom}(\mathbf{e}_{top})$
    $\mathbf{x} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$
end while

Qualitative results

Samples from model trained on FFHQ [Karras et al, 2018]

Qualitative results

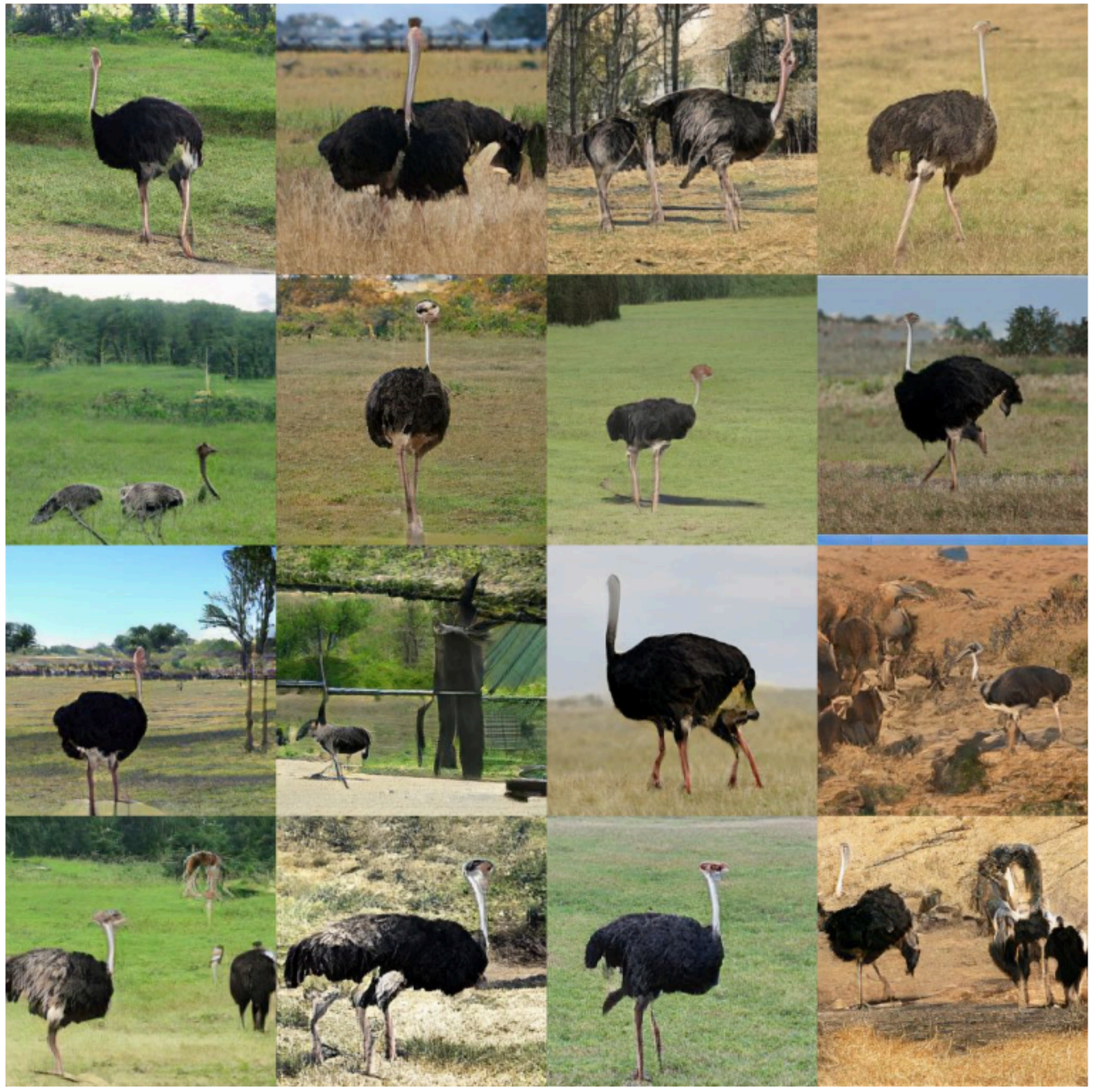
For comparison: samples from StyleGAN2 [Karras et al. 2020]



VQ-VAE

BigGAN deep

VQ-VAE



BigGAN deep

VQ-VAE

BigGAN deep

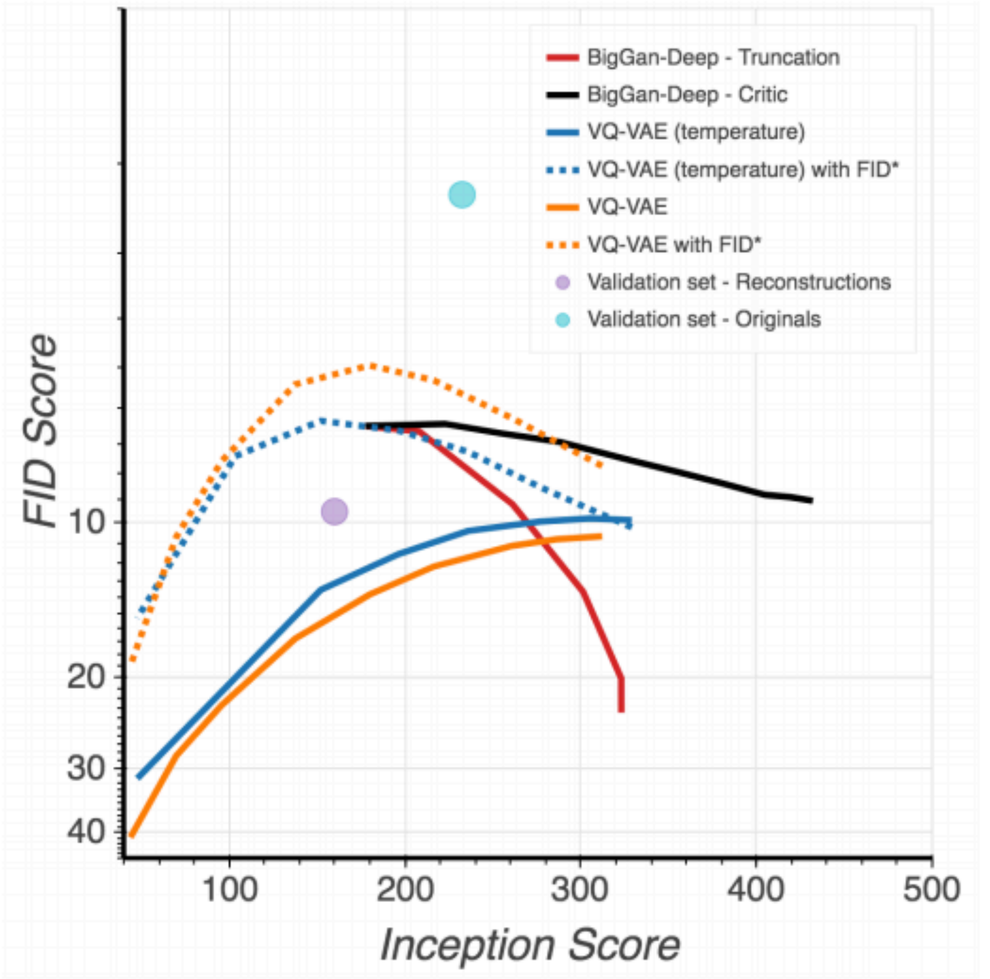
Recall: BigGAN [Brock et al., 2018]

Not sure if any of the VQ-VAE samples look quite this good…

Quantitative evaluation

	Train NLL	Validation NLL	Train MSE	Validation MSE
Top prior	3.40	3.41	-	-
Bottom prior	3.45	3.45	-	-
VQ Decoder	-	-	0.0047	0.0050

Quantitative evaluation



Quantitative evaluation

	Top-1 Accuracy	Top-5 Accuracy
BigGAN deep	42.65	65.92
VQ-VAE	54.83	77.59
VQ-VAE after reconstructing	58.74	80.98
Real data	73.09	91.47

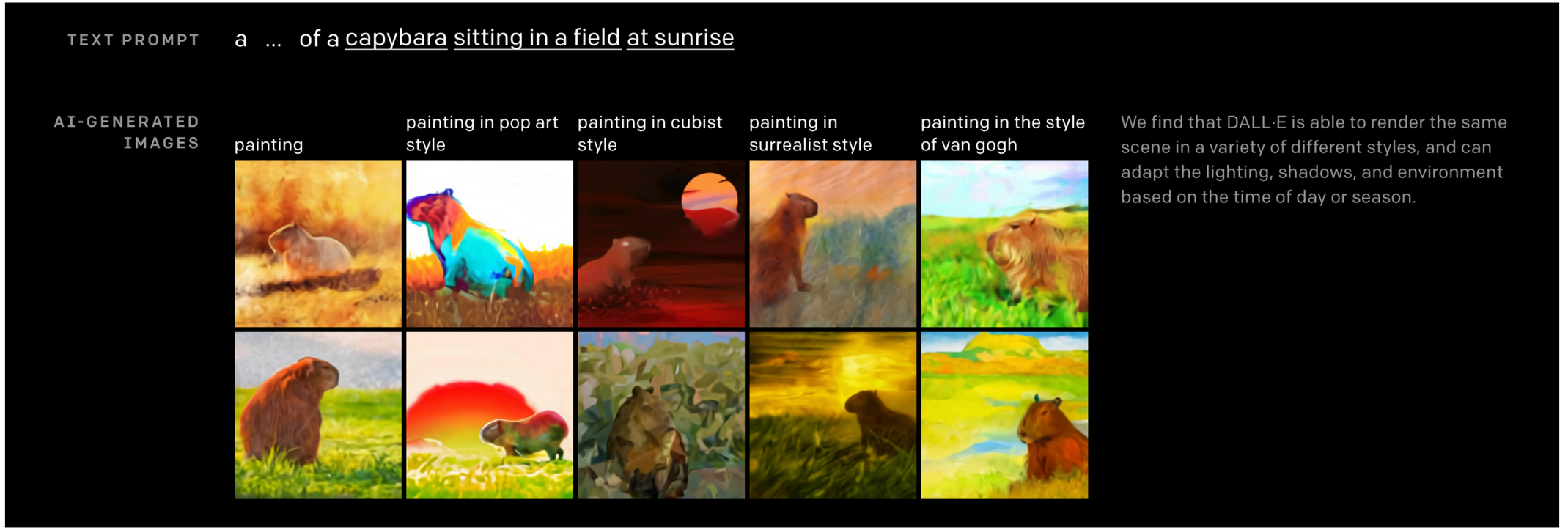
Train classifier on generated images. How well does it do?

No speed evaluation, despite claims of 30x faster than VQ-VAE-1

Version 2.0 change log

- Lots of architectural changes
- Larger images
- Hierarchical latent code sampling
- Improving the PixelCNN
- What makes it work? Really hard to say!
  - Paper doesn’t have any ablation experiments

More recent application of VQ-VAEs



https://openai.com/blog/dall-e/

Trade-offs to this approach

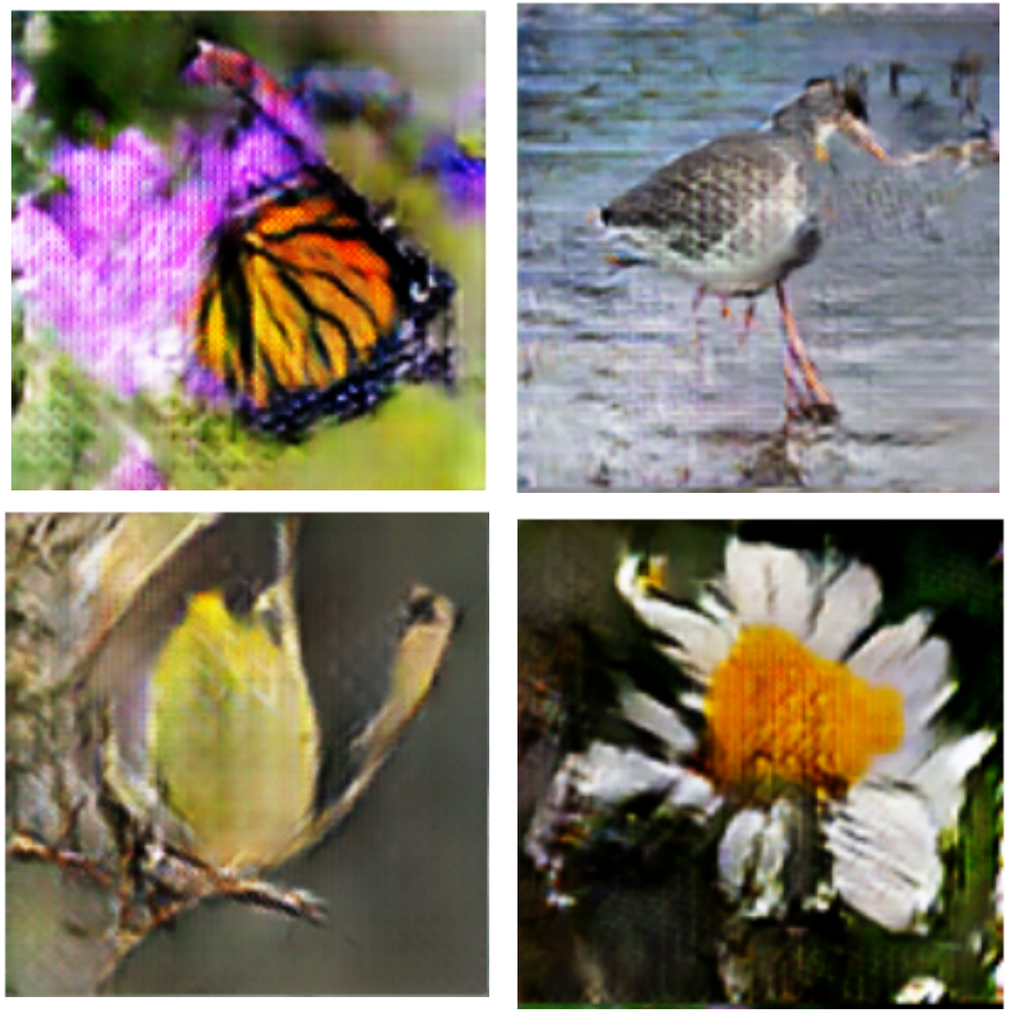
NVAE: A Deep Hierarchical Variational Autoencoder

Arash Vahadt & Jan Kautz

NeurIPS 2020

Rapid progress in GANs due mostly to better architectures and compute

ACGAN [Odena et al. 2016]



BigGAN [Brock et al. 2018]

Both trained on ImageNet

Source: Isola, Freeman, Torralba

Architecture design for VAEs

Maybe the problem is the architecture, not the loss:

1. New architecture based on depthwise convolution
2. New parameterization for approximate posteriors
3. Optimization tricks for stabilizing training
4. Tricks for reducing memory

Hierarchical VAEs

Partition z into L groups:

$$p(\boldsymbol{z}) = \prod_l p(\boldsymbol{z}_l|\boldsymbol{z}_{<l})$$

Encoders:

$$q(\boldsymbol{z}_{<l}|\boldsymbol{x}) := \prod_{i=1}^{l-1} q(\boldsymbol{z}_i|\boldsymbol{x}, \boldsymbol{z}_{<i})$$

z variables arranged spatially and multiscale (2x size each level)

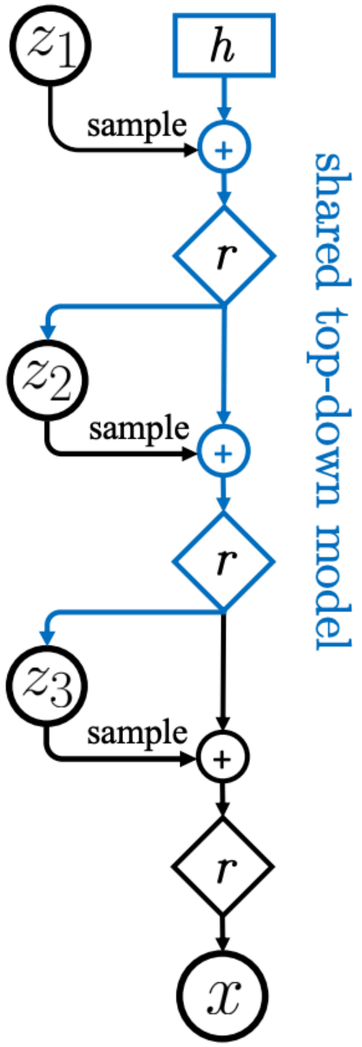


Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and h is a trainable parameter.

Hierarchical VAEs

Partition z into L groups:

$$p(\boldsymbol{z}) = \prod_l p(\boldsymbol{z}_l|\boldsymbol{z}_{<l})$$

Encoders:

$$q(\boldsymbol{z}_{<l}|\boldsymbol{x}) := \prod_{i=1}^{l-1} q(\boldsymbol{z}_i|\boldsymbol{x}, \boldsymbol{z}_{<i})$$

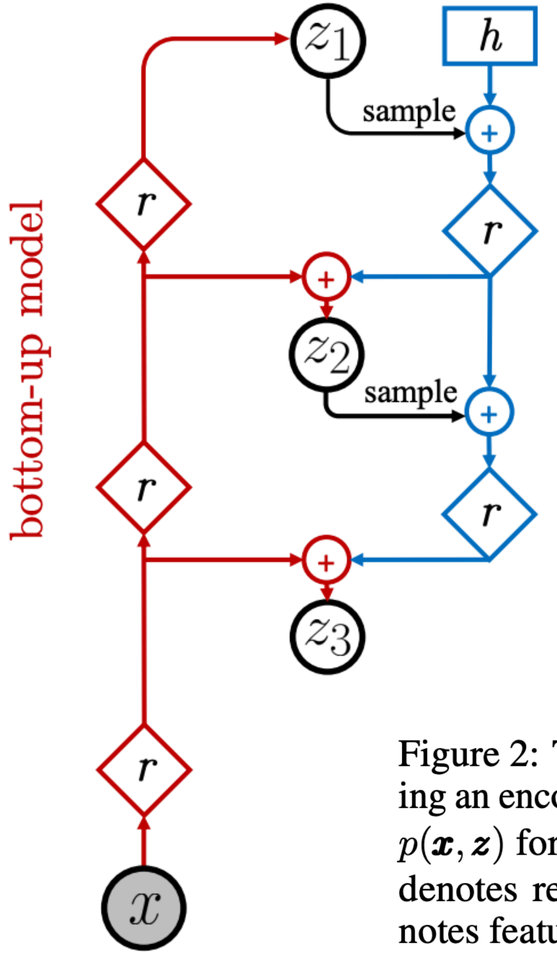
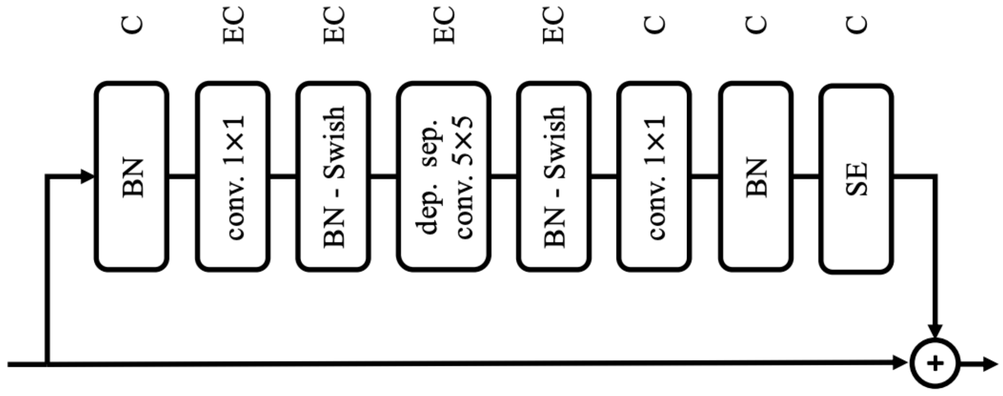


Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and h is a trainable parameter.

Residual cells

- **Want:** large receptive field to model long-range dependencies
- **Problem:** large convolutions are expensive
- **Solution:** depthwise convolutions (i.e. operate on each channel separately)
- Not very expressive, so (like in MobileNetV2), expand number of channels first



(a) Residual Cell for NVAE Generative Model

Other tricks

Batch normalization:

- Other papers complain that batch norm (BN) hurts performance due to noise
- They observe: BN good during training, bad during testing
- Need to set hyperparameters carefully (momentum and scaling regularization) so that the results aren't outdated or mismatches amplified by scaling

Memory:

- Use mixed precision (reduces memory usage by 40%)
- Gradient checkpointing for BN+activation (recompute BN in backward pass), approx. 18% memory reduction
- Together, doubles batch size (64 images / sec afterwards)

KL divergence problems

- **Problem:** encoder/decoder get out of sync, causing $\mathbf{KL}\big(q(z^i|\boldsymbol{x})||p(z^i)\big)$ to explode
- **Residual normal distributions**: Use residual normal distribution. Mean and standard deviation of **z** at level **l** are added to those of ***l-1***
- **Spectral regularization:** Each network layer should not change the inputs too much. Add a term minimizing singular values after power iteration [Yoshida & Miyato, 2017]
- Use normalizing flows (see lecture next week), though only small improvement

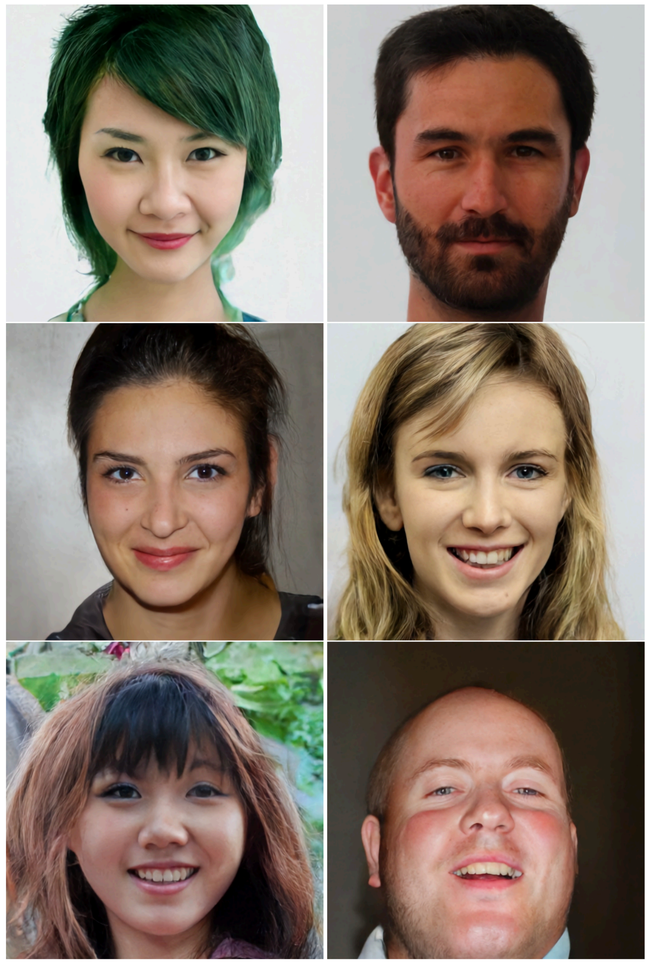
Overall architecture

Hyperparamter	**FFHQ** 256×256
# epochs	200
batch size per GPU	4
# normalizing flows	4
# latent variable scales	5
# groups in each scale	4, 4, 4, 8, 16
spatial dims of $\boldsymbol{z}$ in each scale	8^2, 16^2, 32^2, 64^2, 128^2
# channel in $\boldsymbol{z}$	20
# initial channels in enc.	30
# residual cells per group	2
λ	0.1
GPU type	32-GB V100
# GPUs	24*
total train time (h)	160

Approx. 36 layers for largest model

Qualitative results

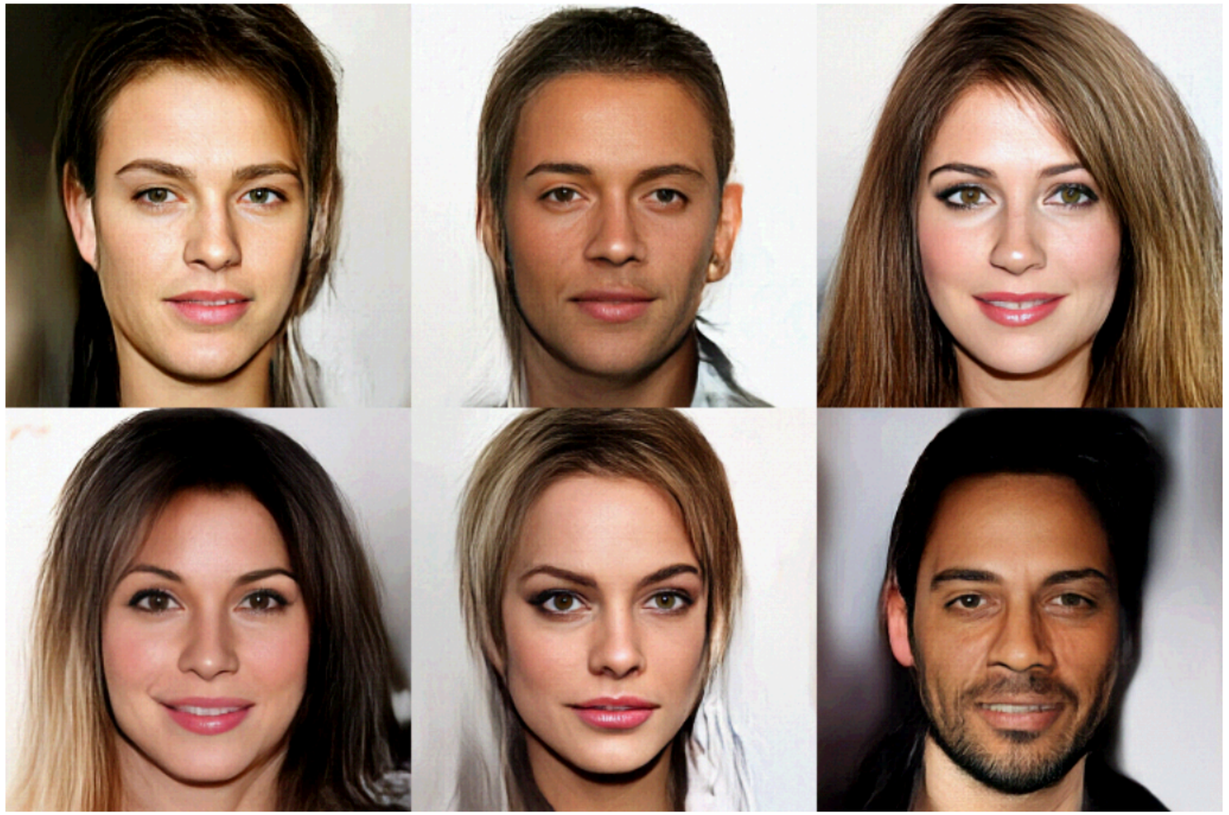
(e) FFHQ ($t = 0.5$)



For comparison: VQ-VAE-2 (high-res)

Qualitative results

(d) CelebA HQ ($t = 0.6$)



(g) Glow [62] trained on CelebA HQ ($t = 0.7$)

For comparison: GLOW

Qualitative results

(a) MNIST ($t = 1.0$)

(b) CIFAR-10 ($t = 0.7$)

(c) CelebA 64 ($t = 0.6$)

Quantitative results

- Log likelihood competitive with autoregressive models.

Method	MNIST 28×28	CIFAR-10 32×32	ImageNet 32×32	CelebA 64×64	CelebA HQ 256×256	FFHQ 256×256
NVAE w/o flow	**78.01**	2.93	-	2.04	-	0.71
Autoregressive Models						
SPN [68]	-	-	3.85	-	0.61	-
PixelSNAIL [34]	-	2.85	3.80	-	-	-
Image Transformer [69]	-	2.90	3.77	-	-	-
PixelCNN++ [70]	-	2.92	-	-	-	-
PixelRNN [41]	-	3.00	3.86	-	-	-
Gated PixelCNN [71]	-	3.03	3.83	-	-	-

Adding even more layers

Generated by a network with 75 layers (vs. 36 for NVAE)
Negative log likelihood: 0.61 vs. 0.68 for NVAE on FFHQ

Child. “Very Deep VAEs Generalize Autoregressive Models And Can Outperform Them On Images" ICLR 2021

What are the pros and cons of generative unsupervised learning?

Are we evaluating these models correctly?

Next class: energy-based models