Pixel Convolutional Neural Networks (PixelCNN) and Vector Quantised Variational AutoEncoder (VQ-VAE)

Jia Zhao

HKUST

August 23, 2019

Machine Learning

- High-dimensional data (e.g., image data) could be viewed as random samples from an unknown high-dimensional distribution of interest.
- In the field of machine learning, we are often interested in learning probabilistic models of various natural and artificial phenomena from data.

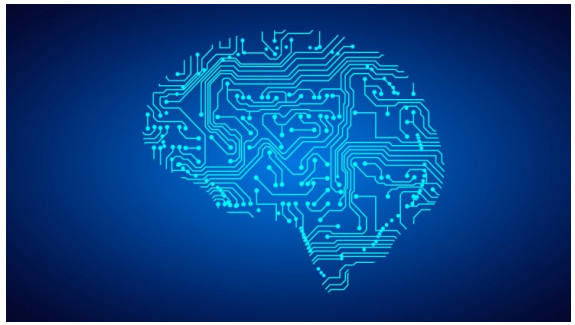


Figure 1: Machine Learning.

PixelCNN

- Pixel Convolutional Neural Networks (PixelCNN) is proposed by Google DeepMind.

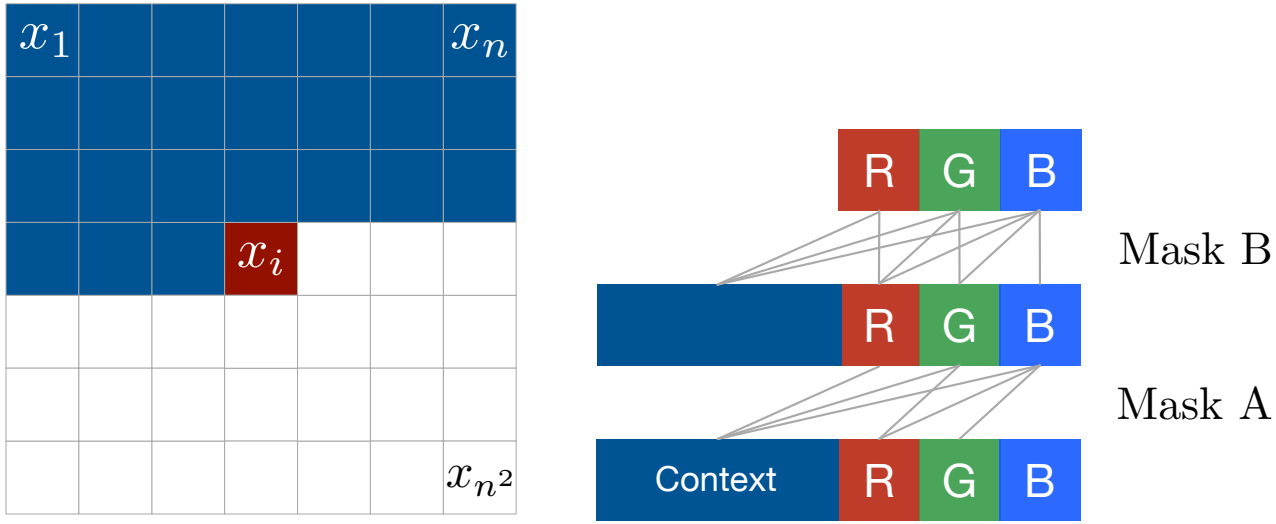


Figure 2: **Left:** To generate pixel x_i one conditions on all the previously generated pixels left and above of x_i. **Right:** Diagram of the connectivity inside a masked convolution. In the first layer, each of the RGB channels is connected to previous channels and to the context, but is not connected to itself. In subsequent layers, the channels are also connected to themselves.

Probabilistic modeling in PixelCNN

- Assume the image only has one channel for simplicity.
- Goal: To assign a probability $p(\mathbf{x})$ to each image $\mathbf{x}$ formed of $N \times N$ pixels.
- We can write the image $\mathbf{x}$ as a one-dimensional sequence $x_1, x_2, ..., x_{N^2}$ where pixels are taken from the image row by row.

Probabilistic modeling in PixelCNN

- PixelCNNs (and PixelRNNs) model the joint distribution of pixels over an image (spacial feature) $\mathbf{x}$ as the following product of conditional distributions, where x_i is a single pixel:

$$p(\mathbf{x}) = \prod_{i=1}^{N^2} p(x_i | x_1, x_2, ..., x_{i-1}). \tag{1}$$

  - +: Do not need to introduce a latent space.
  - +: Directly optimize the log likelihood (not the lower bound).

Pixels as discrete variables

- PixelCNNs model $p(\mathbf{x})$ as discrete distribution, with every conditional distribution in Eq. (1) being a multinomial distribution.
  - $+$: Each channel variables $x_{i,*}$ simply takes one of 256 distinct (discrete) values. The discrete distribution is representationally simple and has the advantage of being arbitrarily multimodal without prior on the shape.

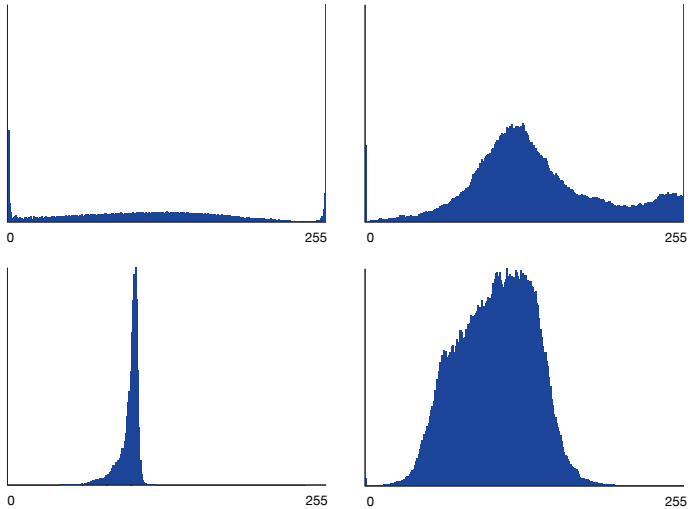
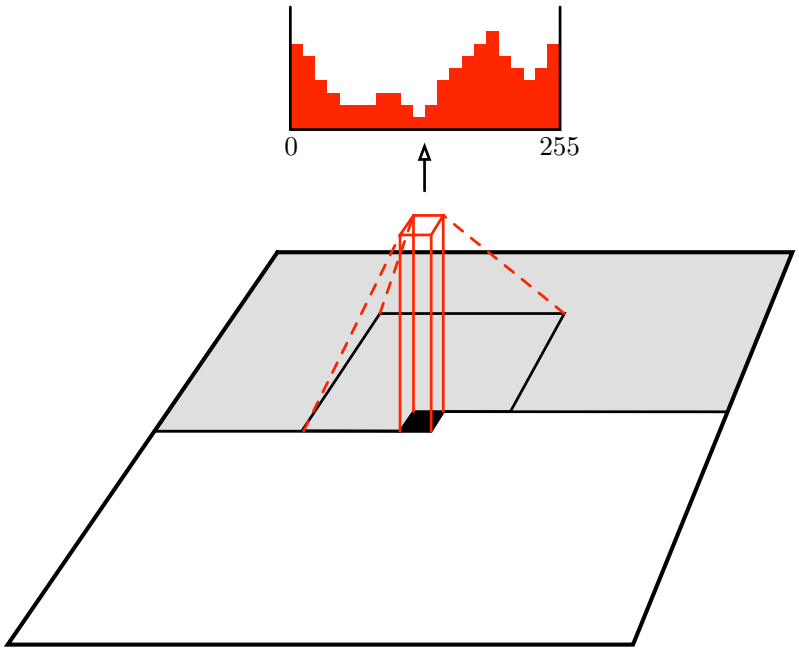


Figure 3: The discrete distributions of pixels.

Sampling method

- -: PixelCNNs are time-consuming when sampling large features.



1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Figure 4: Sampling method in PixelCNN

Blind spot in the receptive field

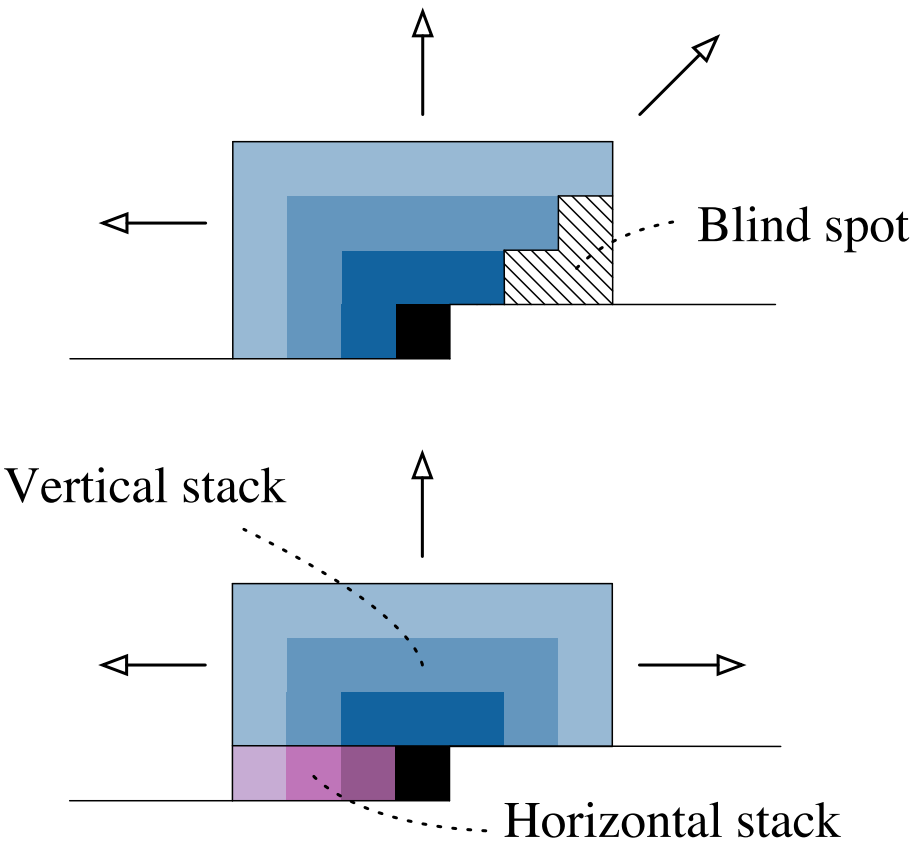


Figure 5: **Top**: PixelCNNs have a blind spot in the receptive field that can not be used to make predictions. **Bottom**: Two convolutional stacks (blue and purple) allow to capture the whole receptive field.

Solution (two CNN stacks)

- It is simple to achieve the horizontal stack.
- Here we provide a note on vertical stack component:

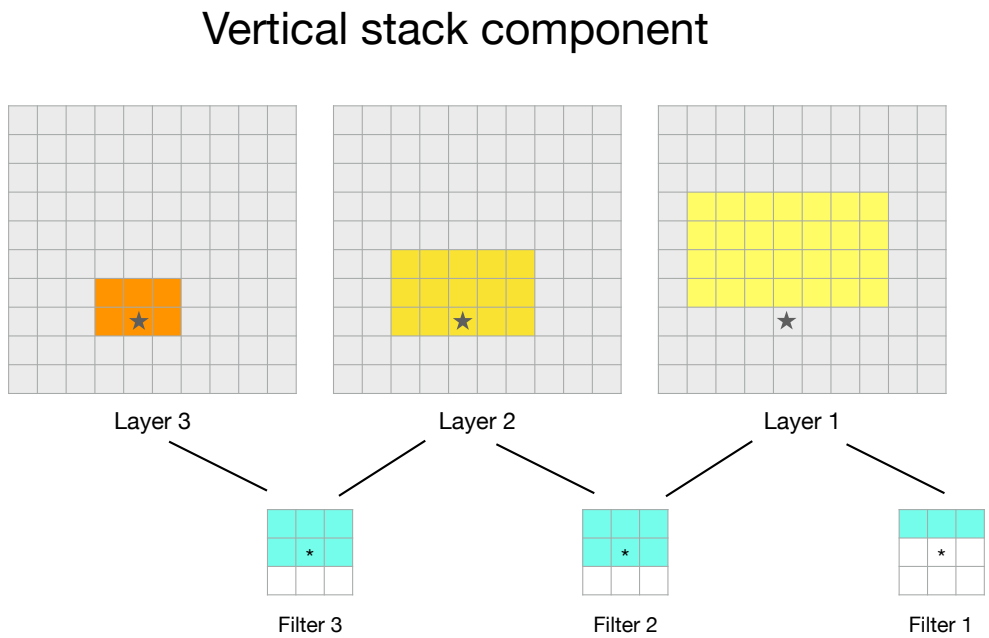


Figure 6: Note on vertical stack component.

Gated PixelCNN: Model long-term dependency and complex interactions

We replaced the rectified linear units between the masked convolutions in the original PixelCNN with the following gated activation unit:

$$\mathbf{y} = \tanh(\mathbf{W}_{k,f} * \mathbf{x}) \odot \sigma(\mathbf{W}_{k,g} * \mathbf{x}).$$

- The region of the neighbourhood available to pixelCNN grows linearly with the depth of the convolutional stack.
  However this shortcoming can largely be alleviated by using sufficiently many layers.
- Containing multiplicative units (e.g. in the form of the LSTM gates), may help to model more complex interactions.

Efficient use of Residual Connections

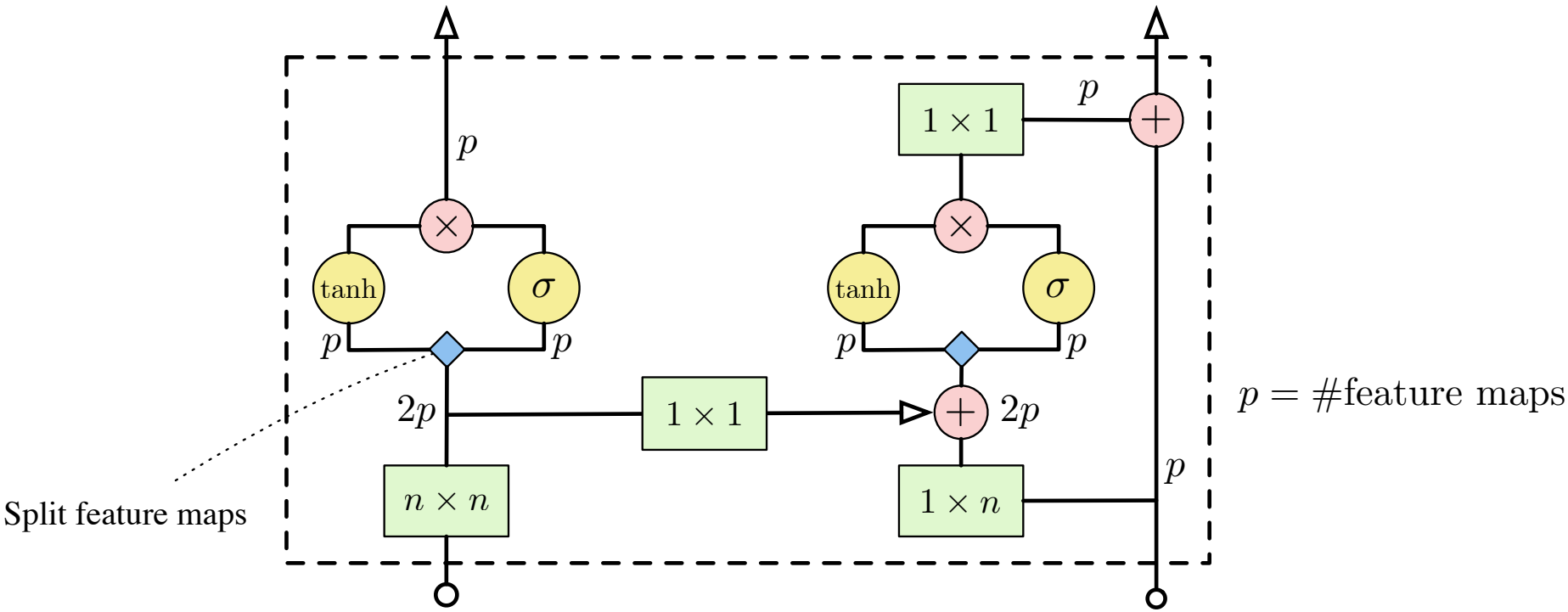


Figure 7: Gated PixelCNN with efficient use of Residual Connections.

Experiments on MNIST

Figure 8: **Left:** Generated samples in epoch 1; **Middle:** Generated samples in epoch 5; **Right:** Generated samples in epoch 50.

Motivation

- PixelCNNs are time-consuming when sampling large features. However, one may alleviate the shortcoming of PixelCNNs by the following approach:
  - First, train a model for data representation.
  - Second, train a PixelCNN to learn the output representation.
- Variational AutoEncoder (VAE) is a generative model based on AutoEncoder (AE), which is a good example.

Generative modeling: from AE to VAE

AutoEncoder (AE)

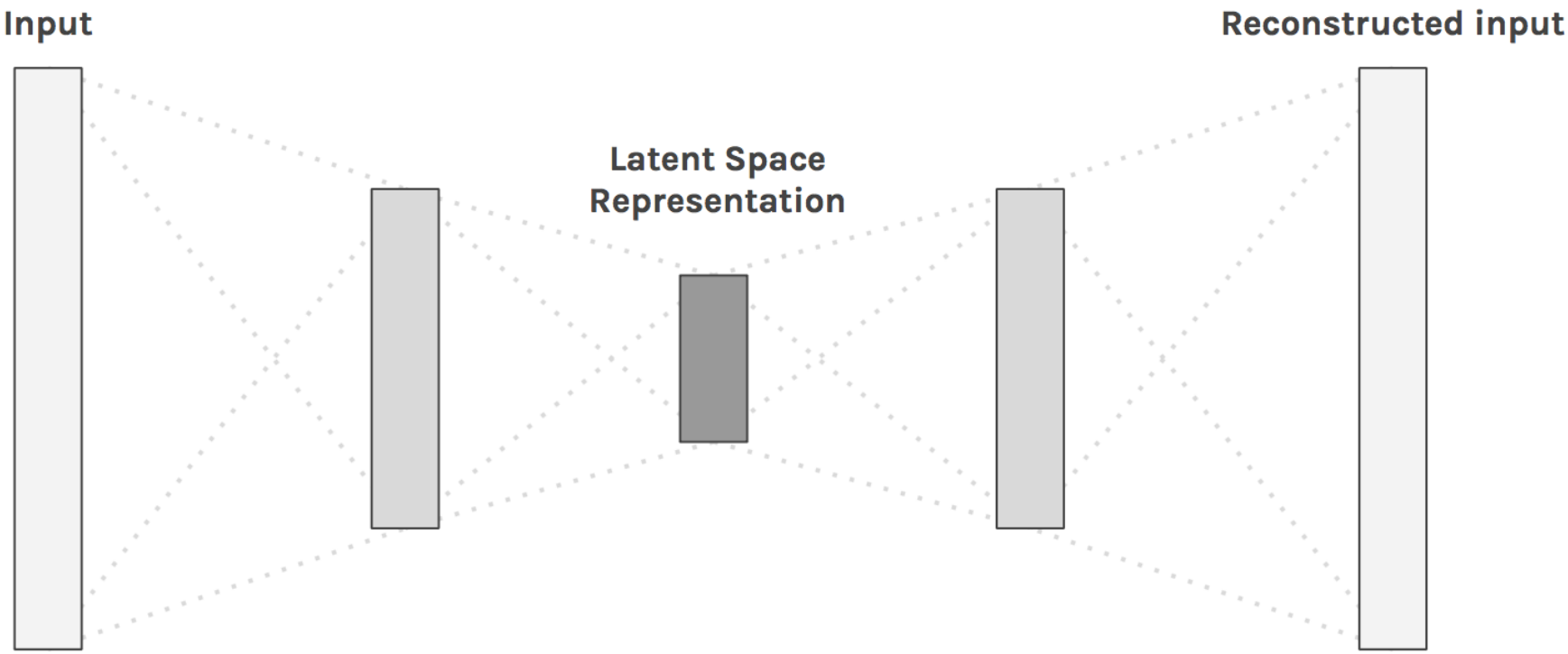


Figure 9: Simple AutoEncoder for representation learning.

Generative modeling: from AE to VAE

AutoEncoder (AE)

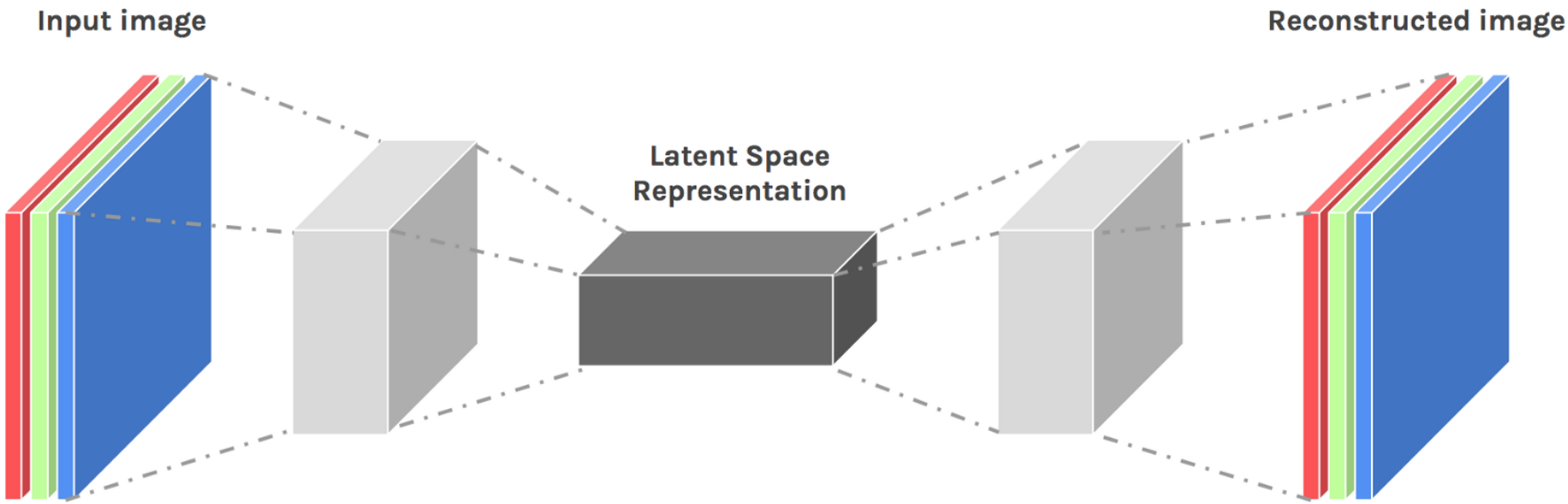


Figure 10: Convolutional AutoEncoder for representation learning.

Generative modeling: from AE to VAE

Variational AutoEncoder (VAE):

Probabilistic generative modeling based on AE.

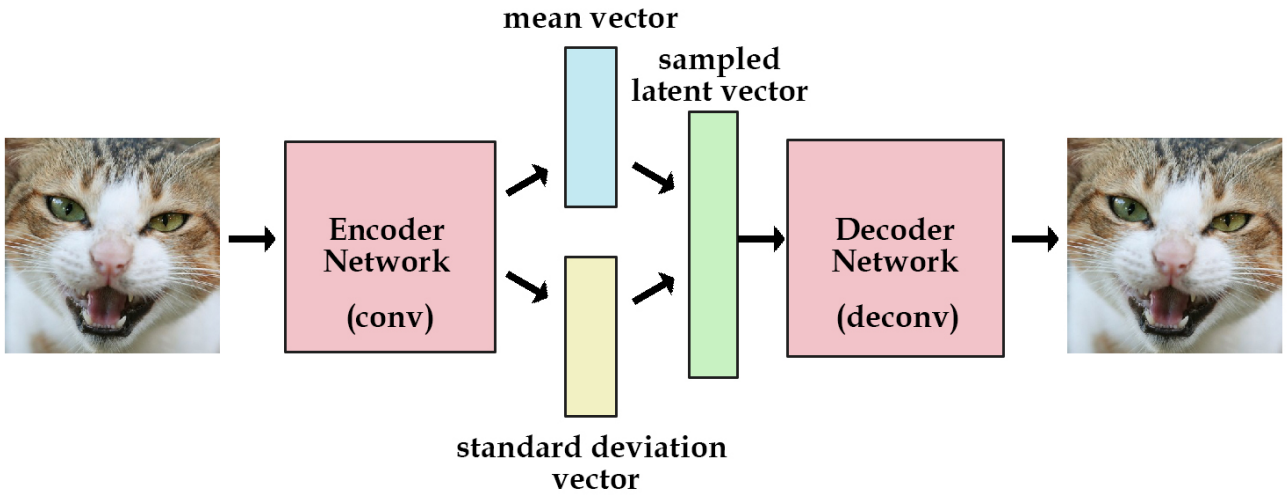
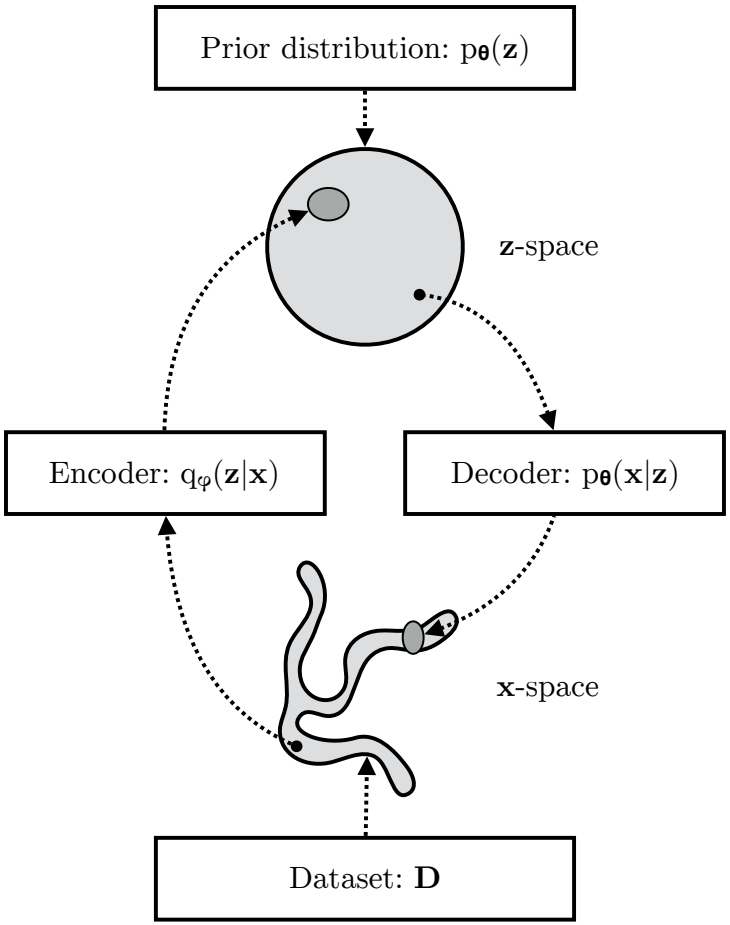


Figure 11: Variational AutoEncoder.

Generative modeling: from AE to VAE

How do VAEs work for generative modeling?



- *data* $\mathbf{x}$; *latent variables* $\mathbf{z}$.
- Data generation process (Decoder):

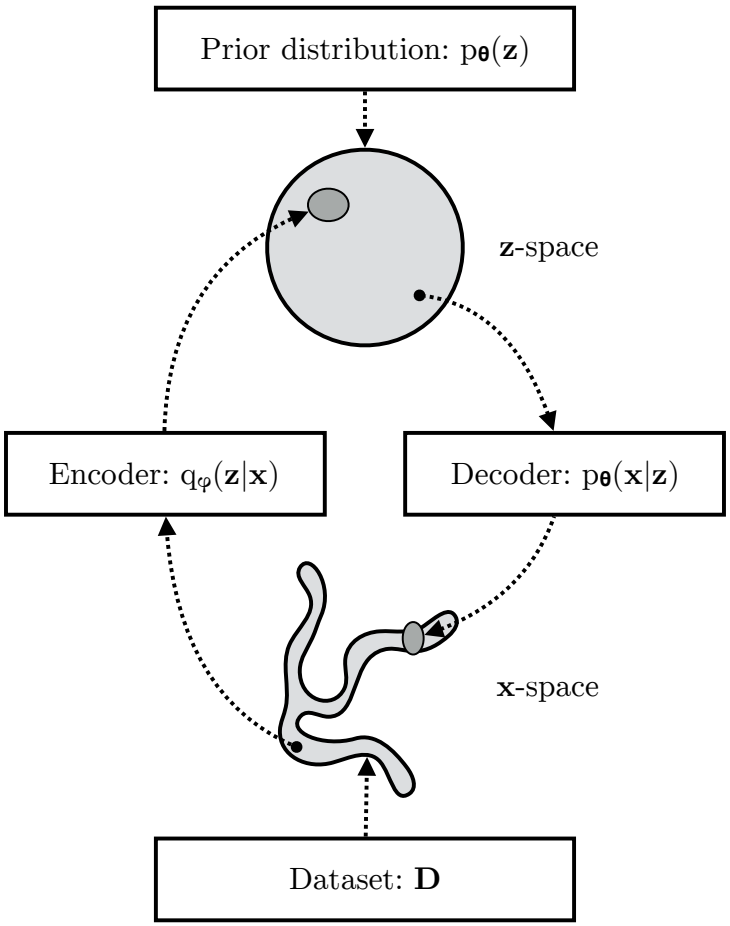
$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}),$$
$$\mathbf{p} = \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\mathbf{z}),$$
$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \text{Bernoulli}(\mathbf{x}|\mathbf{p} = \mathbf{p}(\mathbf{z}; \boldsymbol{\theta})).$$

- Posterior approximation (Encoder):

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = \text{EncoderNeuralNet}_{\boldsymbol{\varphi}}(\mathbf{x}),$$
$$q_{\boldsymbol{\varphi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\sigma})$$

Generative modeling: from AE to VAE

How do VAEs work for generative modeling?



- Loss function:

$$
\begin{aligned}
\text{loss} &= -\,\text{elbo} \\
&= -\log p_\theta(\mathbf{x}) \quad (-\textit{loglikelihood}) \\
&\quad + \mathrm{KL}(p_\varphi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \\
&= -\,\mathbb{E}_{p_\varphi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \\
&\quad (\textit{reconstruction error}) \\
&\quad + \mathrm{KL}(p_\varphi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}))
\end{aligned}
$$

- Minimize loss function.

Analysis of VAEs

- +: VAEs can **learn the underlying structure in data**, but also **generate new data with similar properties**.
- -: VAEs assign a simple prior on latent code, which may cause the issue of posterior collapse.
- -: VAEs adopt the independence assumption which ignores the dependences between variables in latent codes, causing a less efficient use of latent space.
- *: Discrete representations are potentially a more natural fit for many of the modalities we are interested in (e.g., images can often be described concisely by language).

Posterior collapse of VAEs

Loss function used in VAEs:

$$\begin{aligned} \text{loss} = & -\mathbb{E}_{p_{\varphi}(\mathbf{z}|\mathbf{x})}[\log p_{\theta}(\mathbf{x}|\mathbf{z})] \quad (\textit{reconstruction error}) \\ & + \mathrm{KL}(p_{\varphi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z})) \end{aligned}$$

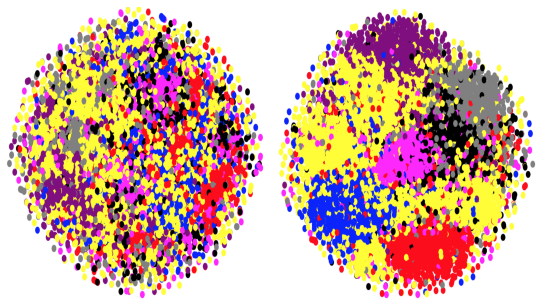


Figure 12: Posterior collapse of VAEs.

Latents are ignored in the issue of posterior collapse of VAEs probably due to the following reasons:

- Latents are paired with a powerful decoder;
- Priors assigned on latent codes are simple, and easy to be satisfied.

Discrete latent variables

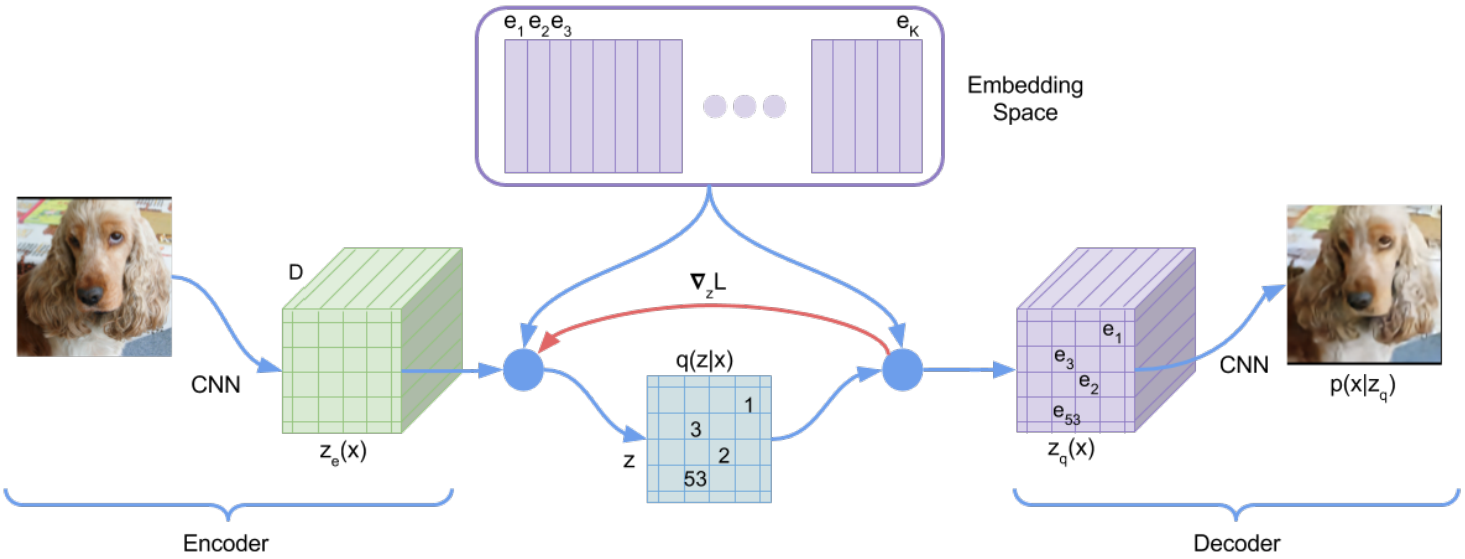


Figure 13: VQ-VAE

Notations:

- $z_e(x)$: output of encoder network;
- K: total number of latent embedding vectors;
- D: dimensionality of each latent embedding vector;
- e_i, $i = 1, 2, ..., K$: latent embedding vector.

Discrete latent variables

- The discrete latent variables are calculated by a nearest neighbour look-up using the shared embedding space e.

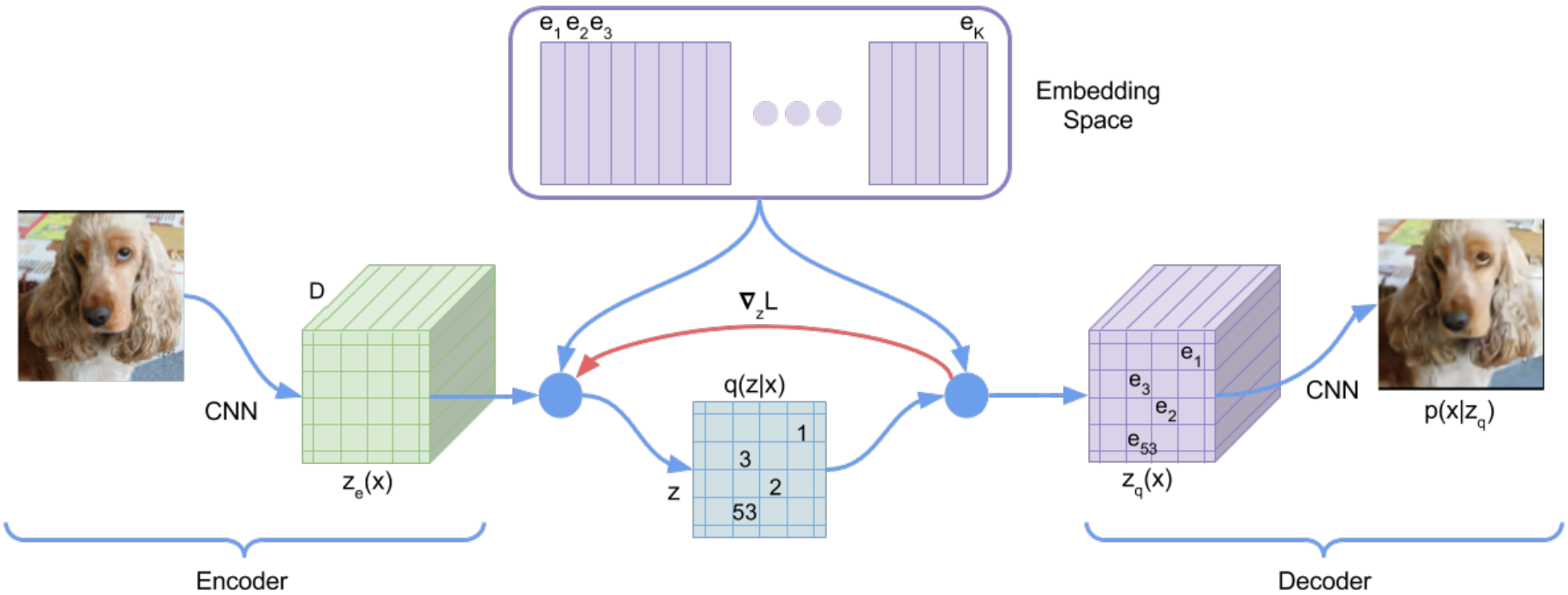


Figure 14: VQ-VAE

Discrete latent variables

- The posterior categorical distribution $q(z|x)$ probatilities are defined as one-hot as follows:

$$q(z=k|x) = \begin{cases} x=1 & \text{for } k = \arg\min_j \|z_e(x) - e_j\|_2, \\ y=0 & \text{otherwise} \end{cases}$$

- The representation $z_e(x)$ is passed through the discretisation bottleneck followed by mapping onto the nearest element of embedding e:

$$z_q(x) = e_k, \quad \text{where } k = \arg\min_j \|z_e(x) - e_j\|_2. \tag{2}$$

- Adopt tf.nn.embedding_lookup(params, ids) for Eq.(2) in practice.

Learning

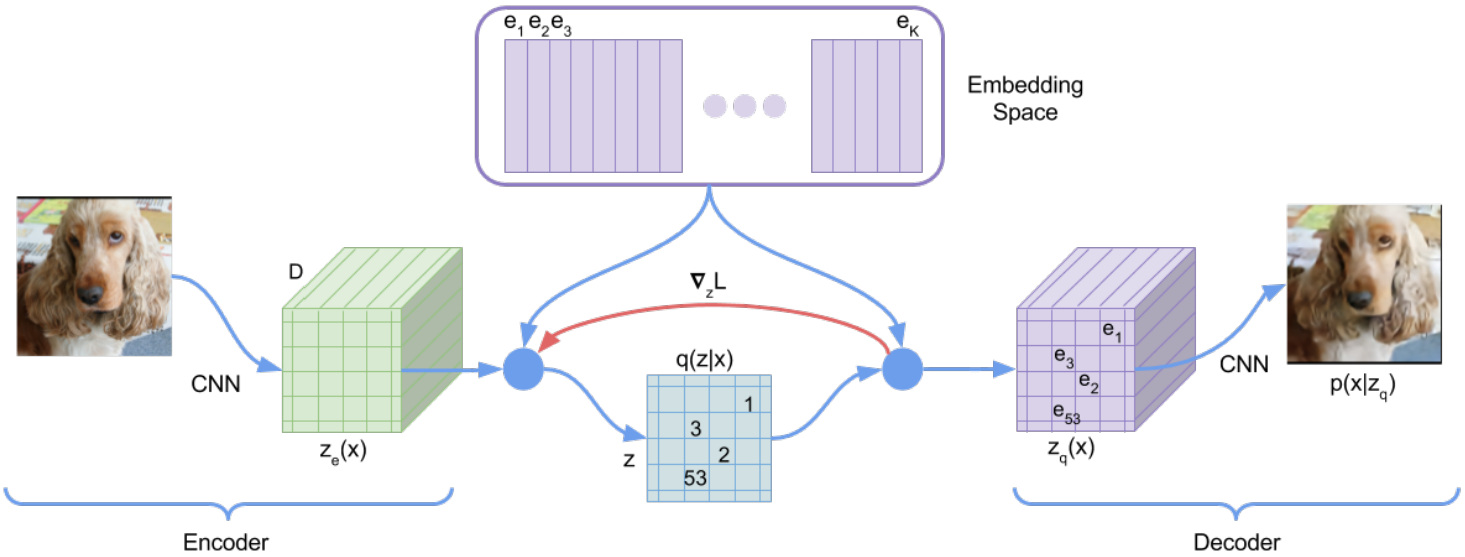


Figure 15: VQ-VAE

- The complete set of parameters for the model are union of parameters of the encoder, decoder, and the embedding space e.
- Problem: there is no real gradient defined for Eq.(2) (tf.nn.embedding_loopup(params, ids) is only differentiable w.r.t. params).

Learning

- Problem: there is no real gradient defined for Eq.(2) (tf.nn.embedding_lookup(params, ids) is only differentiable w.r.t. params).
- E.g.: Let params=$[\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4]^T$ ($\mathbf{a}_i$ is a column-vector), ids=[1], then embedding_lookup result could be written as:

$$\text{look\_up result } = [0, 1, 0, 0][\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4]^T = \mathbf{a}_2$$

Learning

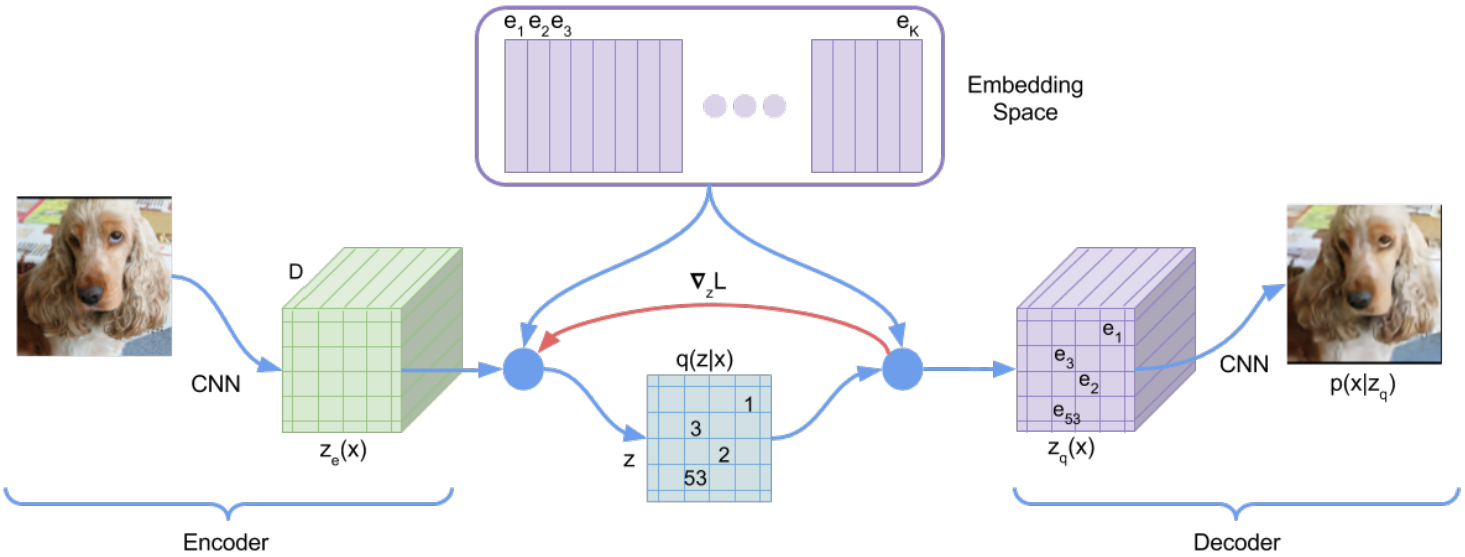


Figure 16: VQ-VAE

- Just copy gradients from decoder input $z_q(x)$ to encoder output $z_e(x)$. Implement z_q = z_e + tf.stop_gradient(z_q - z_e) in practice.
- Then if we simply use the reconstruction loss as loss function, the embeddings e_i receive no gradients from the loss, and the model would only train encoder and decoder networks as AE.

Learning

The total training objective (loss L) is given as:

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

where sg stands for the stopgradient operator. Three components are used to train different parts of VQ-VAE.

- **Term 1**: Reconstruction loss optimizes the decoder and the encoder as AE.
- **Term 2**: To update the dictionary (move the embeddings e_i towards the encoder outputs z_e), Vector Quantisation (VQ) is adopted.
- **Term 3**: Since the volumn of the embedding space is dimensionless, it can grow arbitratrily if the embeddings e_i do not train as fast as the encoder parameters. To make sure the encoder commits to an embedding and its output does not grow, we add a commitment loss.

Prior

- The prior distribution over the discrete latents $p(z)$ is a categorical distribution, and can be made autoregressive by depending on other z in the feature map.
  - +: Prevent the model from posterior collapse.
  - +: Capture the dependencies between the variables in latent codes.
- Whilst training the VQ-VAE, the prior is kept constant.
- After training, we fit a PixelCNN for $p(z)$, so that we can generate x via ancestral sampling.

Experiments on MNIST

Training data reconstruction results from VQ-VAE.

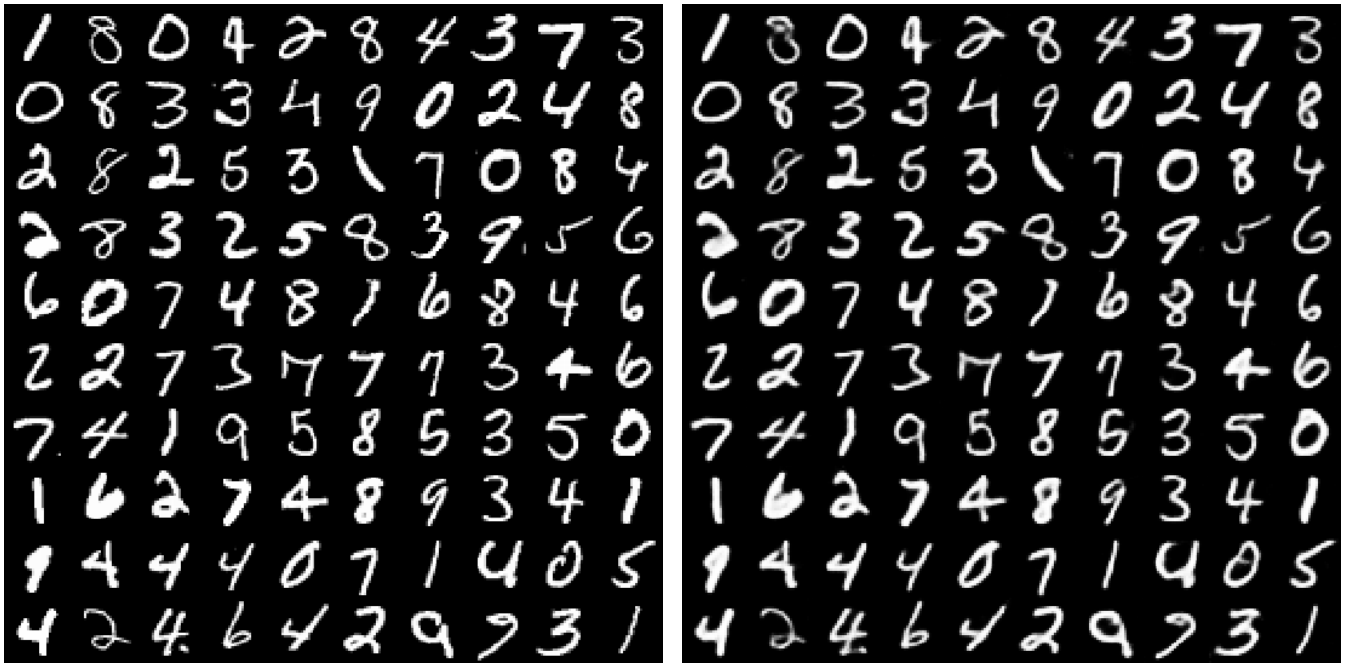


Figure 17: **Left:** Training data. **Right:** Reconstruction of training data.

Experiments on MNIST

Testing data reconstruction results from VQ-VAE.

Figure 18: **Left:** Testing data. **Right:** Reconstruction of testing data.

Experiments on MNIST

Generated images from VQ-VAE with priors learned by using PixelCNN networks.

Figure 19: **Left:** Samples generated by using the codes which are close to the codes from testing data. **Middle:** Samples generated by using random codes. **Right:** Samples generated by using codes learned by PixelCNN.

Experiments on cifar-10

Training data reconstruction results from VQ-VAE.

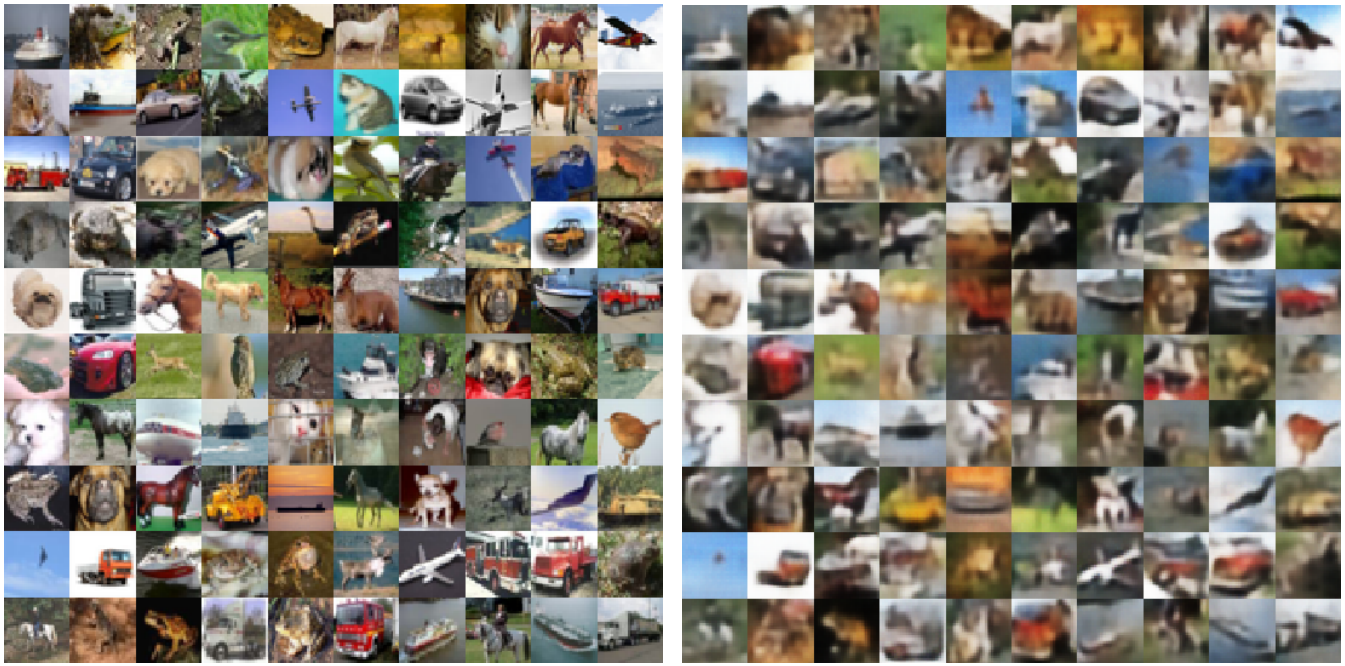


Figure 20: **Left:** Training data. **Right:** Reconstruction of training data.

Experiments on cifar-10

Testing data reconstruction results from VQ-VAE.

Figure 21: **Left:** Testing data. **Right:** Reconstruction of testing data.

Experiments on cifar-10

Generated images from VQ-VAE with priors learned by using PixelCNN networks.

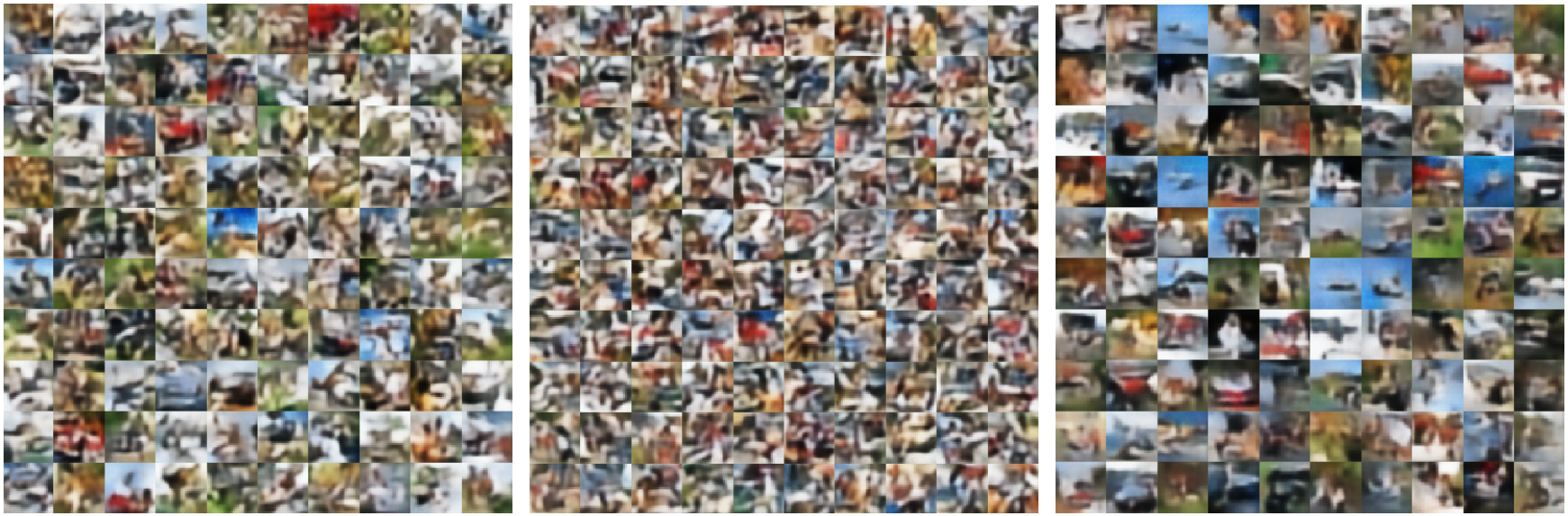


Figure 22: **Left:** Samples generated by using the codes which are close to the codes from testing data. **Middle:** Samples generated by using random codes. **Right:** Samples generated by using codes learned by PixelCNN.

Code and Acknowledgement

- Codes for these experiments could be found at: https://github.com/jiazhao97/VQ-VAE_withPixelCNNprior.
- Acknowledgement
  - VQ-VAE is originally mentioned in the paper [Neural Discrete Representation Learning].
  - PixelCNN is proposed in the papers [Pixel Recurrent Neural Networks] and [Conditional Image Generation with PixelCNN Decoders].
  - Implementation of VQ-VAE (without priors) is based on [the official codes from Google DeepMind]. Note: Different from the official codes, the implementation here does not rely on the [Sonnet library].
  - Implementation of PixelCNN is based on [this repo] with little modify.

VQ-VAE-2: Build the structure to generate diverse high-fidelity images with VQ-VAE

- The main motivation behind hierarchical VQ-VAE (VQ-VAE-2) is to model local information, such as texture, separately from global information such as shape and geometry of objects.

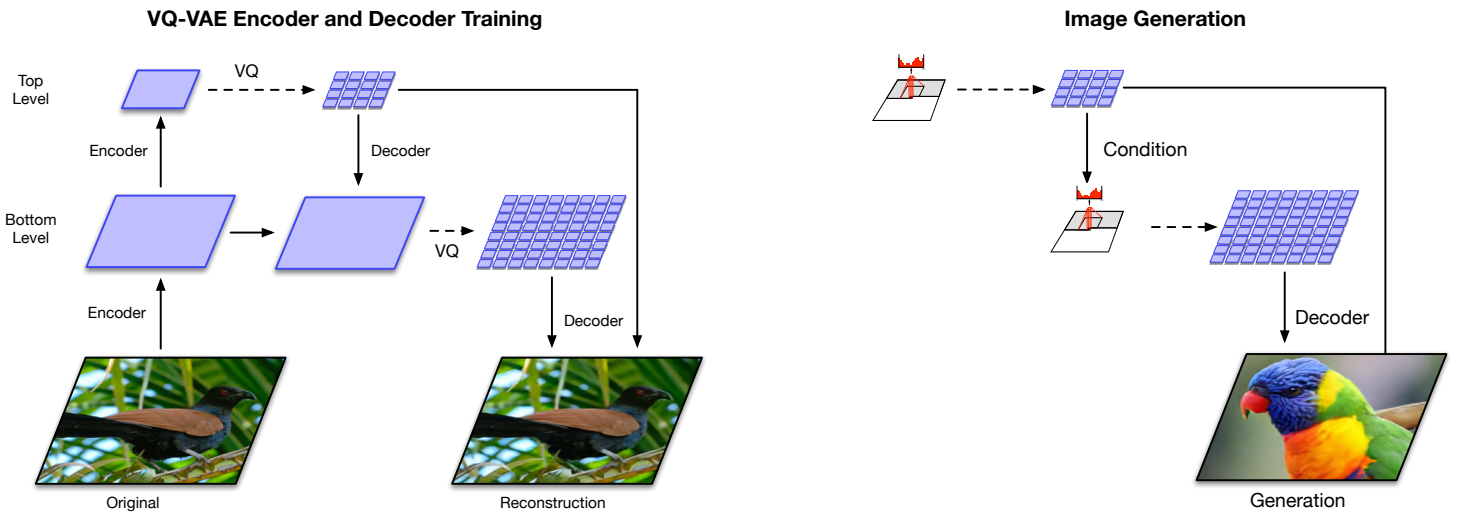


Figure 23: **Left:** Overview of the architecture of the hierarchical VQ-VAE (VQ-VAE-2). **Right:** Multi-stage image generation.