Neural Discrete Representation Learning (VQ-VAE)

Aaron van den Oord, Oriol Vinyals, Koray Kavukcuoglu
Google Deepmind
NeurIPS 2017

Presenter: Niharika Patil
COMP 690-204

Introduction

Vector quantization variational autoencoder (VQ-VAE)

- VAE with discrete latent space

Why discrete?

- Many important real-world things are discrete (words, phonemes, etc.)
- Learn global structure instead of noise and details
- Achieve data compression by embedding into discrete latent space

Why It's Hard?

Posterior collapse

- Powerful decoder can reconstruct x without using the latent z.
- The decoder simply ignores the latent representation.

Discrete latents are hard to train

- Categorical/discrete variables are not directly differentiable.
- Existing solutions included NVIL and VIMCO but training remained challenging.

VIMCO : Previous Approach for Discrete VAEs

Problem: Discrete latent variables cannot be directly differentiated through, making VAE training difficult.

VIMCO uses **multiple samples** of the discrete latent variable instead of just one.

For example:

$$z_1, z_2, z_3, \ldots, z_K$$

and uses all of them to estimate the learning objective.

It was one of the earlier ways to successfully train VAEs with discrete latents.

Limitation: Training is still noisy/high-variance and does not perform as well as continuous VAEs.

VQ-VAE's goal: Provide a **simpler and more effective way** to learn discrete latent representations.

Contributions

- **Introduces VQ-VAE:** simple, discrete latent representations without posterior collapse.
- **Competitive performance:** matches continuous VAEs in log-likelihood.
- **Strong generation:** coherent images, speech, and video with a learned prior.
- **Unsupervised speech learning:** discovers phoneme-like structure and enables speaker conversion.

Method : Encoder Output

The encoder takes the input x and produces a continuous vector:

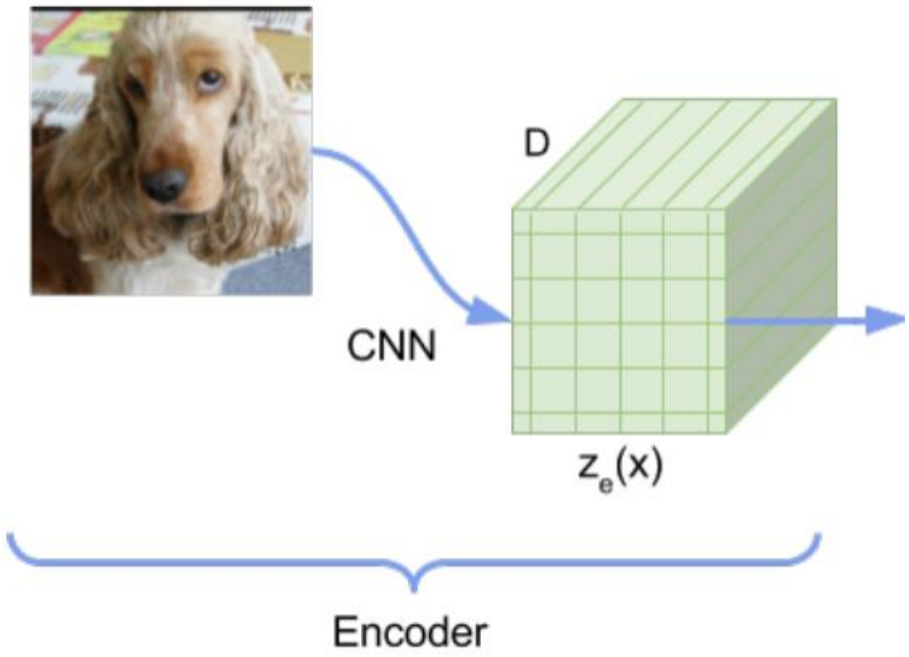
$$z_e(x)$$

For example, suppose:

$$z_e(x) = [0.2,\ 0.8,\ -0.1]$$

This is still a **continuous** representation.

The problem is that VQ-VAE wants a **discrete** representation.


D
CNN
$z_e(x)$
Encoder

Method : Codebook / Embedding Space

The model learns a K dictionary of embedding vectors:

$$e = \{e_1, e_2, \ldots, e_K\}$$

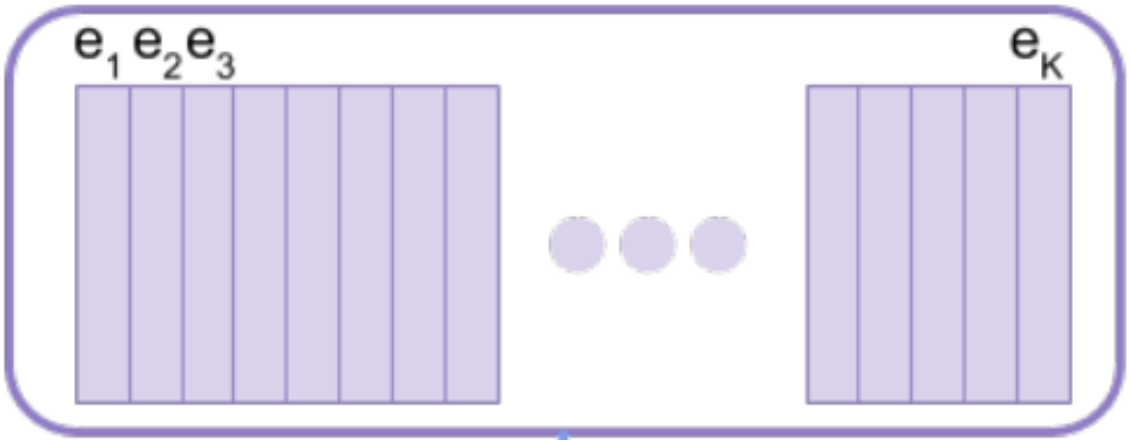
where each embedding has dimension D:

$$e \in \mathbb{R}^{K \times D}$$

So if: K=512, D=64

then the codebook contains **512 possible discrete codes**, and each code is represented by a **64-dimensional vector.**

Codebook = a learned dictionary of possible representations.



Embedding Space

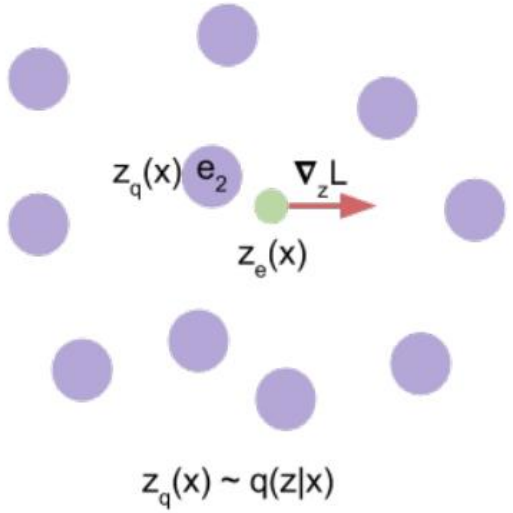
Method : Quantization Equation

The key equation is:

$$k = \arg\min_j \|z_e(x) - e_j\|_2^2$$

For the encoder output $z_e(x)$:

1. Compare it with **every codebook vector**.
2. Calculate the distance to each one.
3. Find the closest embedding.
4. Select its index K



Method : Discrete Posterior

The paper writes the posterior as:

$$q(z = k \mid x) = \begin{cases} 1, & \text{if } k = \arg\min_j \|z_e(x) - e_j\|_2^2, \\ 0, & \text{otherwise.} \end{cases}$$

The model assigns:

- probability **1** to the closest code
- probability **0** to every other code.

So this is a **one-hot categorical distribution**

Method : Quantized Representation

Once the closest code K is found:

$$z_q(x) = e_k$$

This means:

> The decoder doesn't receive the original continuous encoder output. It receives the selected codebook embedding.

So the pipeline becomes:

$$x \to z_e(x) \to \boxed{e_k} \to z_q(x) \to \text{Decoder}$$

This is the **discrete bottleneck**.

Reconstruction Loss

The first part of the VQ-VAE objective is:

$$L = \log p\left(x \mid z_q(x)\right)$$

If the decoder reconstructs x well:

$$\log p\left(x \mid z_q(x)\right) \text{ is high}$$

If reconstruction is poor:

$$\log p\left(x \mid z_q(x)\right) \text{ is low}$$

So this term teaches the model:

Choose representations that retain enough information to reconstruct the input.

Why Can't We Just Backpropagate?

The quantization operation:

$$z_e(x) \rightarrow e_k$$

involves an: argmin

This is not differentiable.

Therefore, normally we can't calculate: $\frac{\partial L}{\partial z_e}$ through the quantization step.

Straight-through estimator

The paper uses a simple approximation:

$$\frac{\partial L}{\partial z_e} \approx \frac{\partial L}{\partial z_q}$$

The gradient is essentially **copied straight through** the quantization operation.

Codebook Loss

The second term is: $\|\text{sg}[z_e(x)] - e_k\|_2^2$

This is the **codebook loss**.

What is it trying to do?

It makes: $e_k \approx z_e(x)$

So the selected embedding moves toward the encoder output.

What does sg mean?

What Does sg Mean?

During the forward pass: $\mathrm{sg}(x) = x$

So the value is unchanged.

So gradients do not flow through it.

So:

- $z_e(x)$ is treated as fixed
- e_k moves toward $z_e(x)$

Commitment Loss

The third term is: $$\beta \left\| z_e(x) - \mathrm{sg}[e_k] \right\|_2^2$$

This looks very similar to the codebook loss, but the direction of optimization is different.

Codebook loss: $$\left\| \mathrm{sg}[z_e(x)] - e_k \right\|_2^2$$

→ **move the codebook toward the encoder**

Commitment loss: $$\beta \left\| z_e(x) - \mathrm{sg}[e_k] \right\|_2^2$$

→ **move the encoder toward the codebook**

That's the key distinction.

Why Do We Need Commitment Loss?

Without it, the encoder could keep moving its output around the embedding space.

The paper's intuition is that the embedding space can otherwise grow arbitrarily.

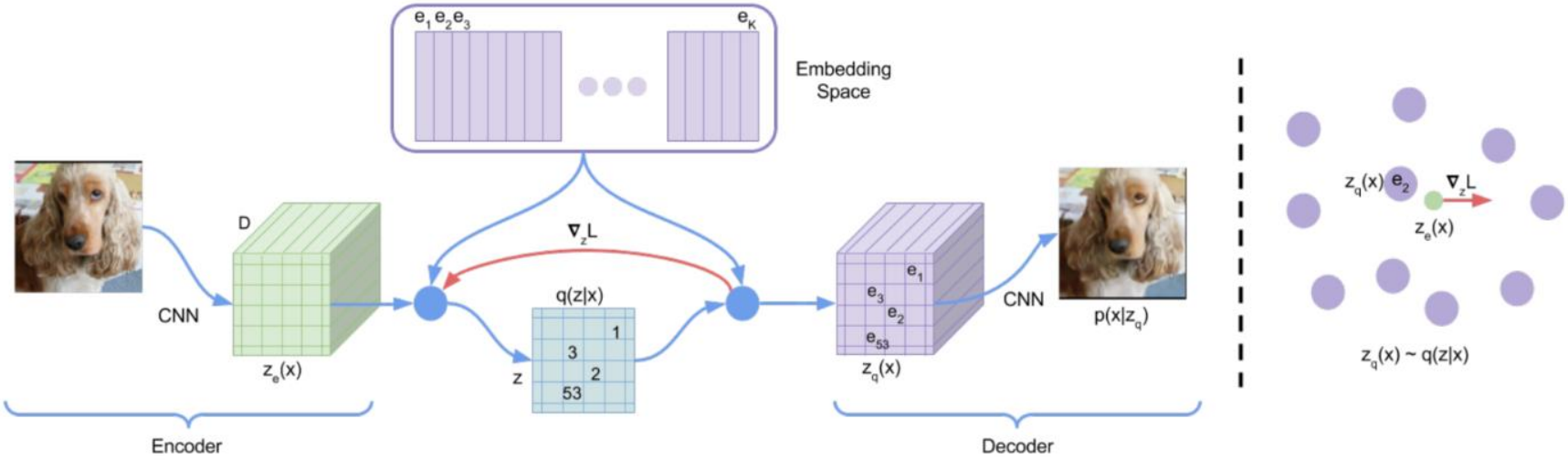
So: $z_e(x) \approx e_k$

This makes the discrete representation more stable.

Complete VQ-VAE Loss

Now the three pieces come together:

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


e1 e2 e3
eK
Embedding Space
D
CNN
ze(x)
∇zL
q(z|x)
z
1
3
2
53
e1
e3
e2
e53
zq(x)
CNN
p(x|zq)
Encoder
Decoder
zq(x) e2
∇zL
ze(x)
zq(x) ~ q(z|x)

Experiments : Comparison with Continuous VAEs

Goal: Can VQ-VAE's **discrete latents** perform as well as continuous VAEs?

Setup:
- Dataset: **CIFAR-10**
- Compared: Continuous **VAE, VQ-VAE, VIMCO**
- Same VAE architecture and varying latent capacity.

Results - Bits/dim

- **VAE:** 4.51
- **VQ-VAE:** 4.67
- **VIMCO:** 5.14

VQ-VAE comes **close to continuous VAE performance** with **compressed, discrete representation**.

Experiments : Images

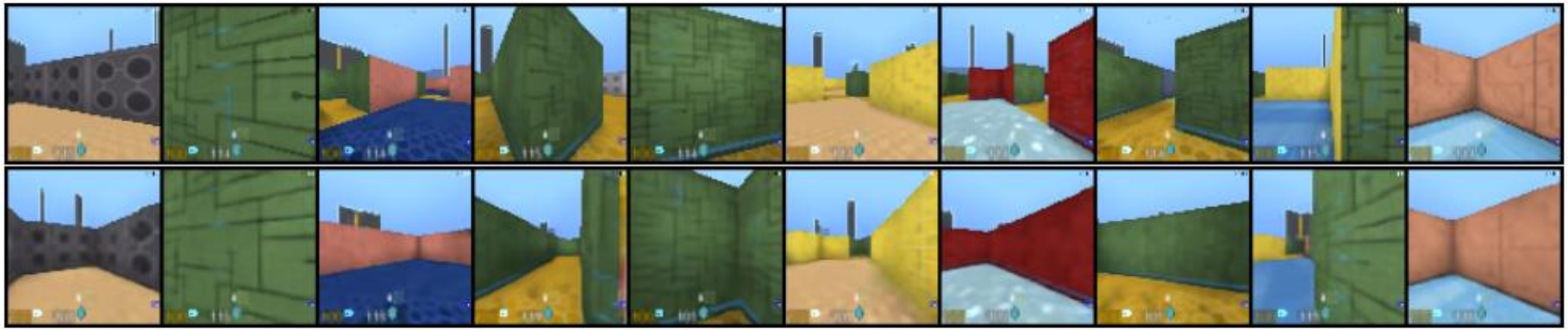
ImageNet: 128×128×3 → 32×32×1 latent space, K=512, achieving ~42.6× compression.

Experiments : Images

ImageNet generation: PixelCNN learns a prior over the **32×32×1 discrete latent space** and generates **128×128** images through the VQ-VAE decoder

Experiments : Images

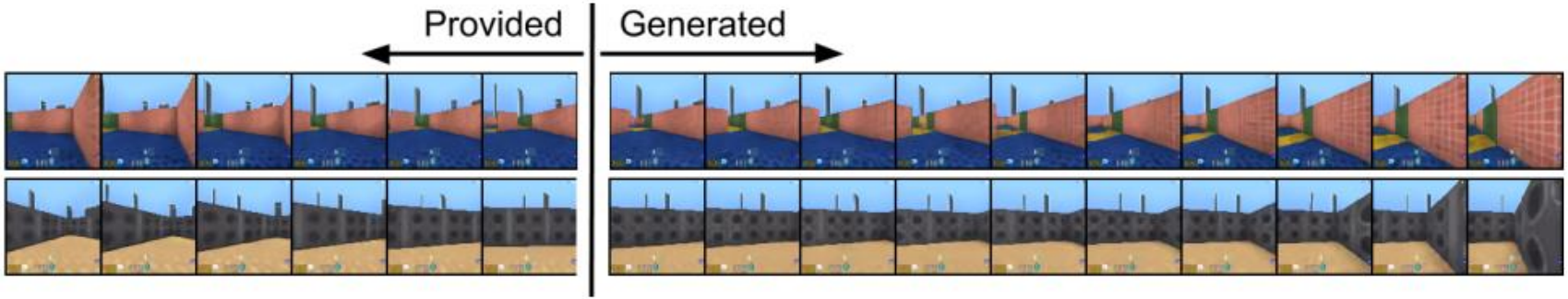
Two-stage VQ-VAE: 84×84×3 → 21×21×1 → **3 discrete latent variables (27 bits total)**, preserving global scene structure despite extreme compression.



Experiments: Audio and Video

28 discrete latent codes at 25 Hz → mapped to **41 phoneme classes**, achieving **49.3% accuracy vs. 7.2% baseline** without phoneme supervision.

DeepMind Lab → predict future frames **entirely in latent space**; given **6 input frames**, generate **10 future frames** conditioned on actions such as move forward or move right.



Summary

Pros:

- Learn meaningful representations with global information
  - Can model long range sequences
  - Fully unsupervised
  - Avoids “posterior collapse” issue
  - Model features that usually span many dimensions in data space

Cons:

- Straight-through estimator is biased
- Compression relies on large lookup tables

Questions

- Is posterior collapse avoided because of the hard bottleneck or just because the KL term is constant?

- The authors only show results with two priors in this paper, PixelCNN and WaveNet. Why is this and what would happen if we changed the priors?

- FSQ avoids issues like codebook collapse and codebook optimization. Does VQ-VAE offer any clear advantage in representation quality or flexibility that compensates for this extra complexity?