Neural Discrete Representation Learning

A. van den Oord, O. Vinyals, K. Kavukcuoglu
2017

Presented by: Yulia Rubanova and Eddie (Shu Jian) Du
CSC2547/STA4273

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

Algorithm

Step I: Input x is encoded into continuous $z_e(x)$

Algorithm

Step I: Input x is encoded into continuous $z_e(x)$

Step II: transforming into z -- discrete variable over K categories

Algorithm

Step I: Input x is encoded into continuous $z_e(x)$

Step II: transforming into z -- discrete variable over K categories

We define a latent embedding space $e \in R^{K \times D}$

(D is the dimensionality of each latent embedding vector)

Algorithm

Step I: Input x is encoded into continuous $z_e(x)$

Step II: transforming into z -- discrete variable over K categories

We define a latent embedding space $e \in R^{K \times D}$

(D is the dimensionality of each latent embedding vector)

To discretize $z_e(x)$: calculate a nearest neighbour in the embedding space

$$k = \text{argmin}_j \|z_e(x) - e_j\|_2$$

Algorithm

The posterior categorical distribution $q(z|x)$ -- deterministic!

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

Algorithm

The posterior categorical distribution $q(z|x)$ -- deterministic!

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

Step III: use $z_q(x)$ as input to the decoder

$$z_q(x) = e_k, \quad \text{where} \quad k = \mathrm{argmin}_j \|z_e(x) - e_j\|_2$$

Algorithm

The posterior categorical distribution $q(z|x)$ -- deterministic!

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

Step III: use $z_q(x)$ as input to the decoder

$$z_q(x) = e_k, \quad \text{where} \quad k = \mathrm{argmin}_j \|z_e(x) - e_j\|_2$$

Reconstruction loss

$$L = \log p(x|z_q(x))$$

Model is trained as a VAE in which we can bound log p(x) with the ELBO.

Training

How can we get a gradient for this?

$$z_q(x) = e_k, \quad \text{where} \quad k = \operatorname{argmin}_j \|z_e(x) - e_j\|_2$$

Training

How can we get a gradient for this?

$$z_q(x) = e_k, \quad \text{where} \quad k = \operatorname{argmin}_j \|z_e(x) - e_j\|_2$$

Just copy gradients from decoder input $z_q(x)$ to encoder output $z_e(x)$

(straight-through estimator)

Training

How can we get a gradient for this?

$$z_q(x) = e_k, \quad \text{where} \quad k = \text{argmin}_j \|z_e(x) - e_j\|_2$$

Just copy gradients from decoder input $z_q(x)$ to encoder output $z_e(x)$

(straight-through estimator)

Main idea:

Gradients from decoder contain information for how the encoder has to change its output to lower the reconstruction loss.

How do we train embeddings?

Embedding don’t get gradient from reconstruction loss $L = \log p(x|z_q(x))$

How do we train embeddings?

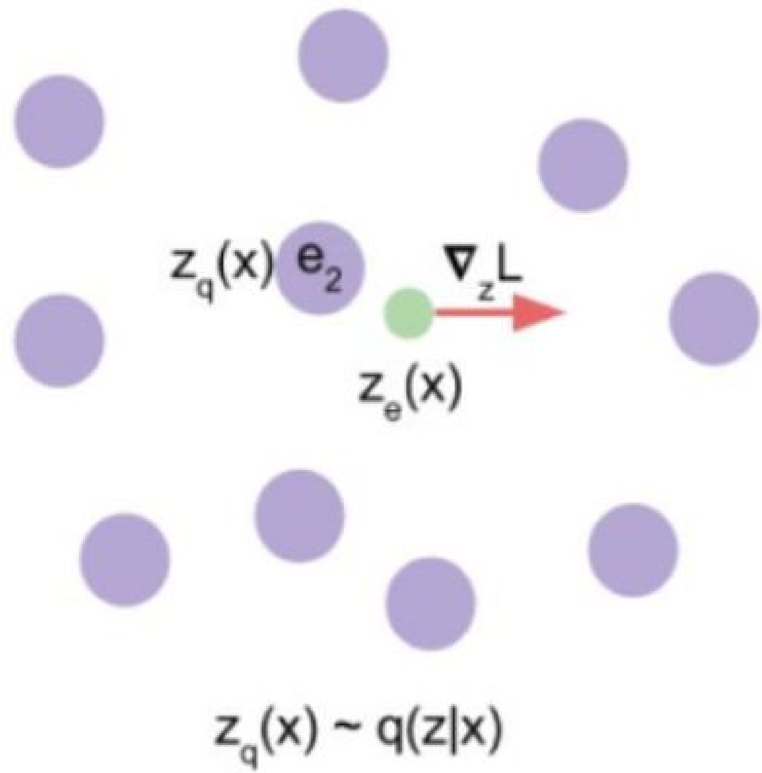
Embedding don’t get gradient from reconstruction loss $L = \log p(x|z_q(x))$

Use L2 error to move the embedding vectors e_i towards $z_e(x)$

Embedding loss = $\|\text{sg}[z_e(x)] - e\|_2^2$

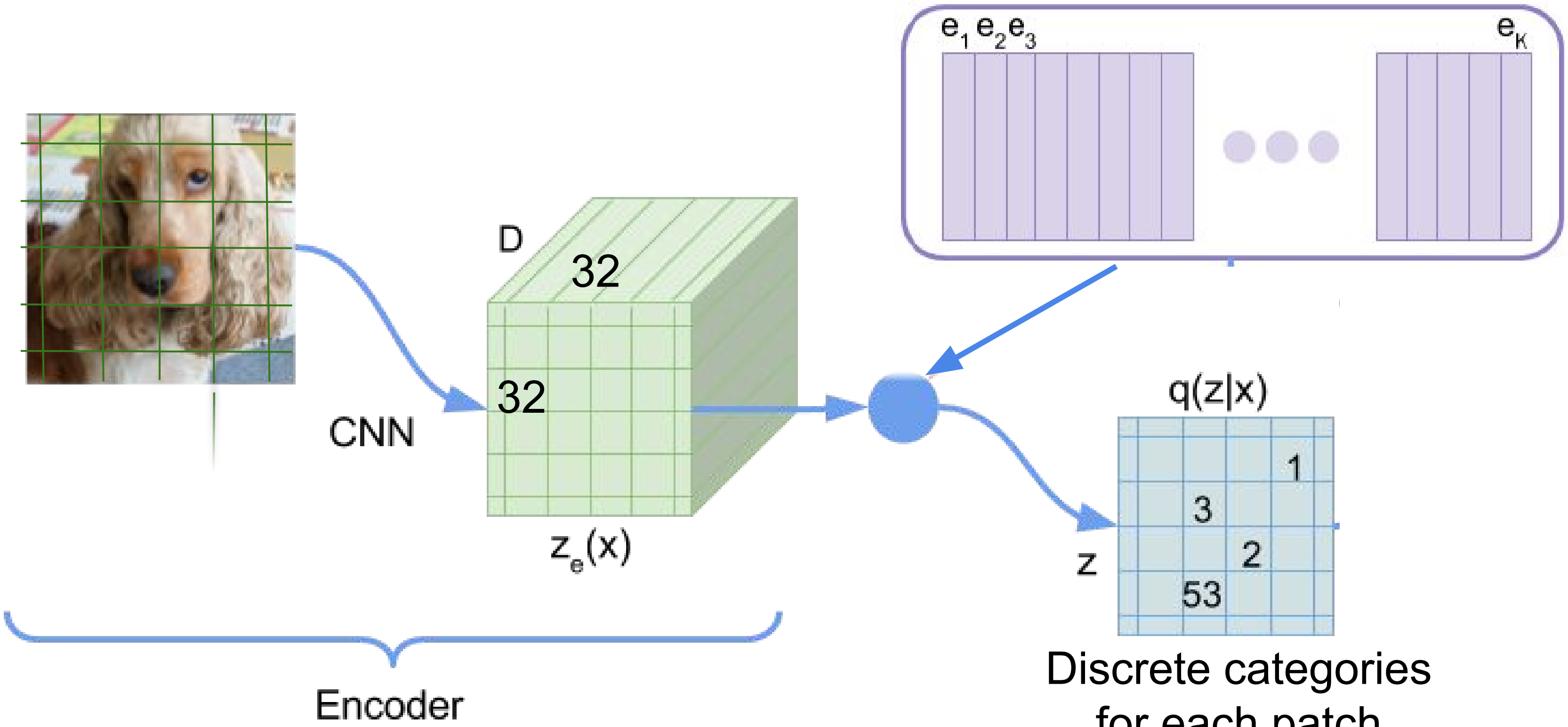
sg = stopgradient operator

Training

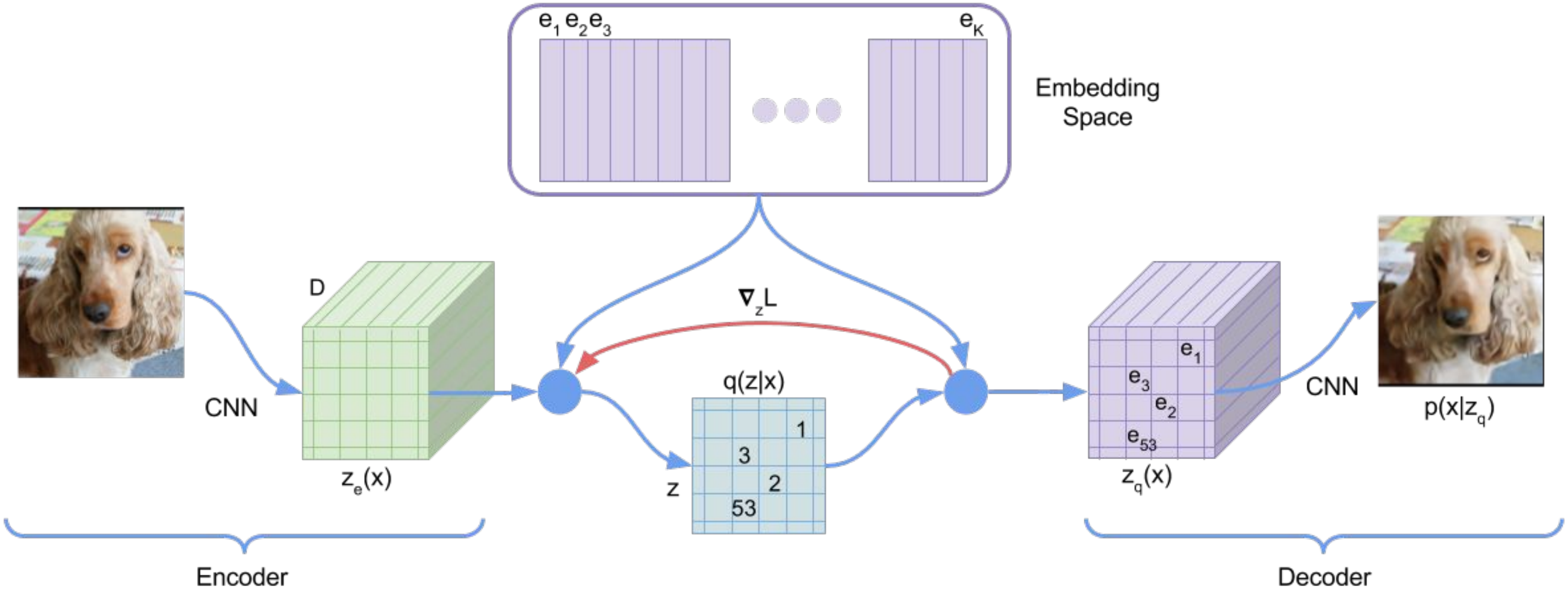


How to reconstruct an image?

Discrete z : a field of 32 x 32 latents (ImageNet), K=512



How to reconstruct an image?



Experiments & Results

ImageNet - Reconstruction

128x128x3 images ↔ 32x32x1 discrete latent space (K=512)

Original

Reconstruction

ImageNet - Recon

128x128x3x(8 bits per pixel) / 32x32x(9 bits to index a vector)
= **42.6 times compression** in bits

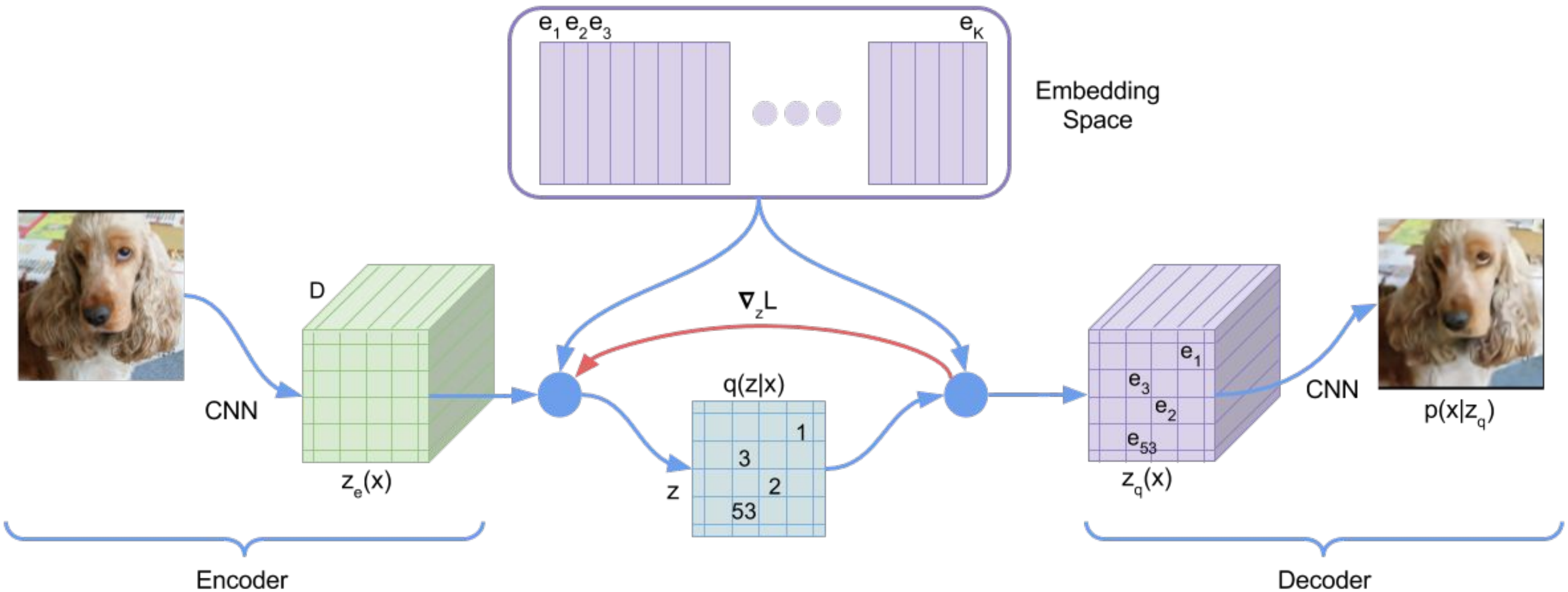
128x128x3 images ↔ 32x32x1 discrete latent space (K=512)

Original

Reconstruction

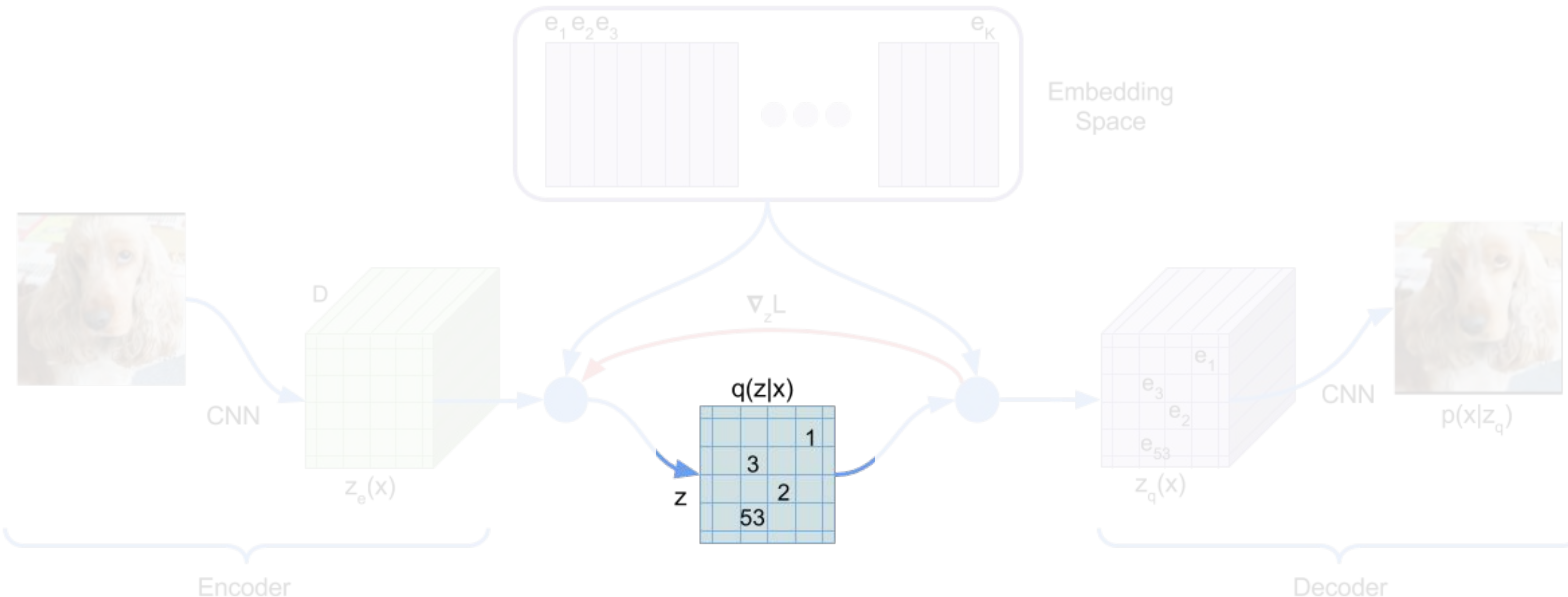
ImageNet - Samples

Train PixelCNN on the 32x32x1 discrete latent space. Sample from PixelCNN, decode with VQ-VAE decoder.



ImageNet - Samples

Train PixelCNN on the 32x32x1 discrete latent space. Sample from PixelCNN, decode with VQ-VAE decoder.

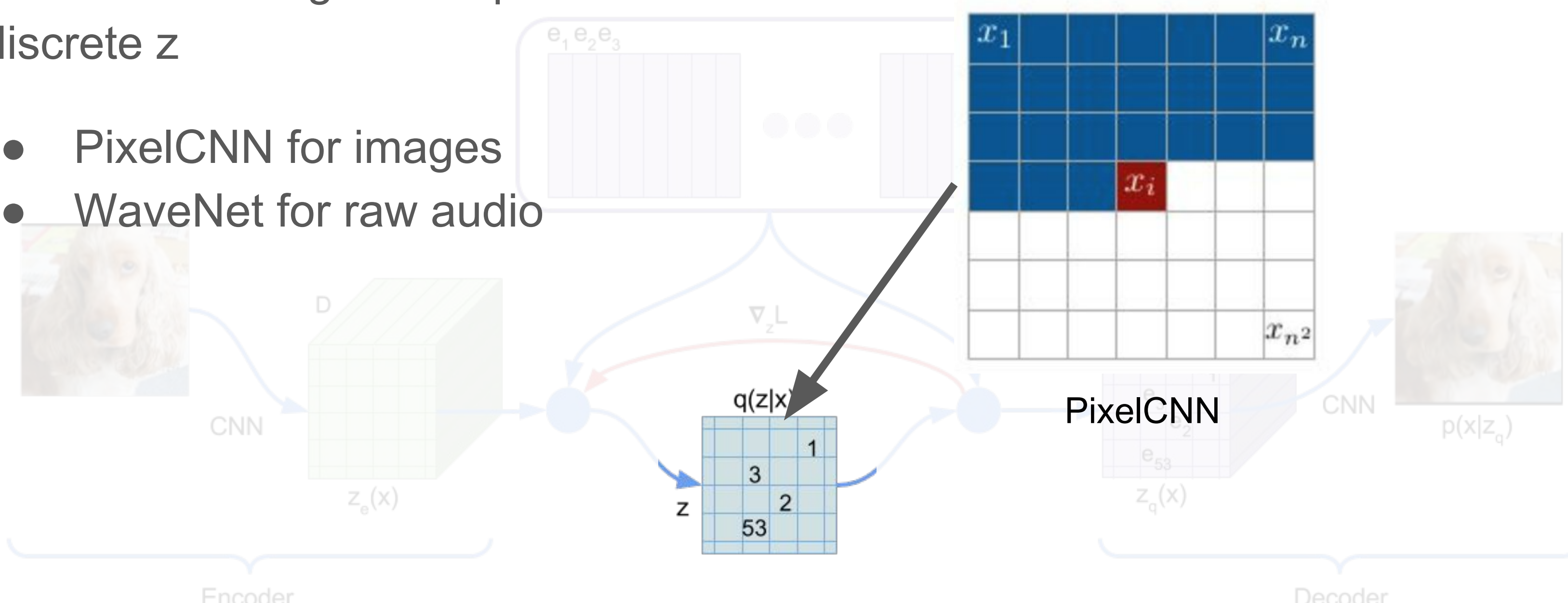


ImageNet - Samples

Train PixelCNN on the 32x32x1 discrete latent space. Sample from PixelCNN, decode with VQ-VAE decoder.

Learn an autoregressive prior over discrete z

- PixelCNN for images
- WaveNet for raw audio



PixelRNN Image Source: https://towardsdatascience.com/summary-of-pixelrnn-by-google-deepmind-7-min-read-938d9871d6d9

ImageNet - Generation

Microwave pickup tiger beetle coral reef brown bear

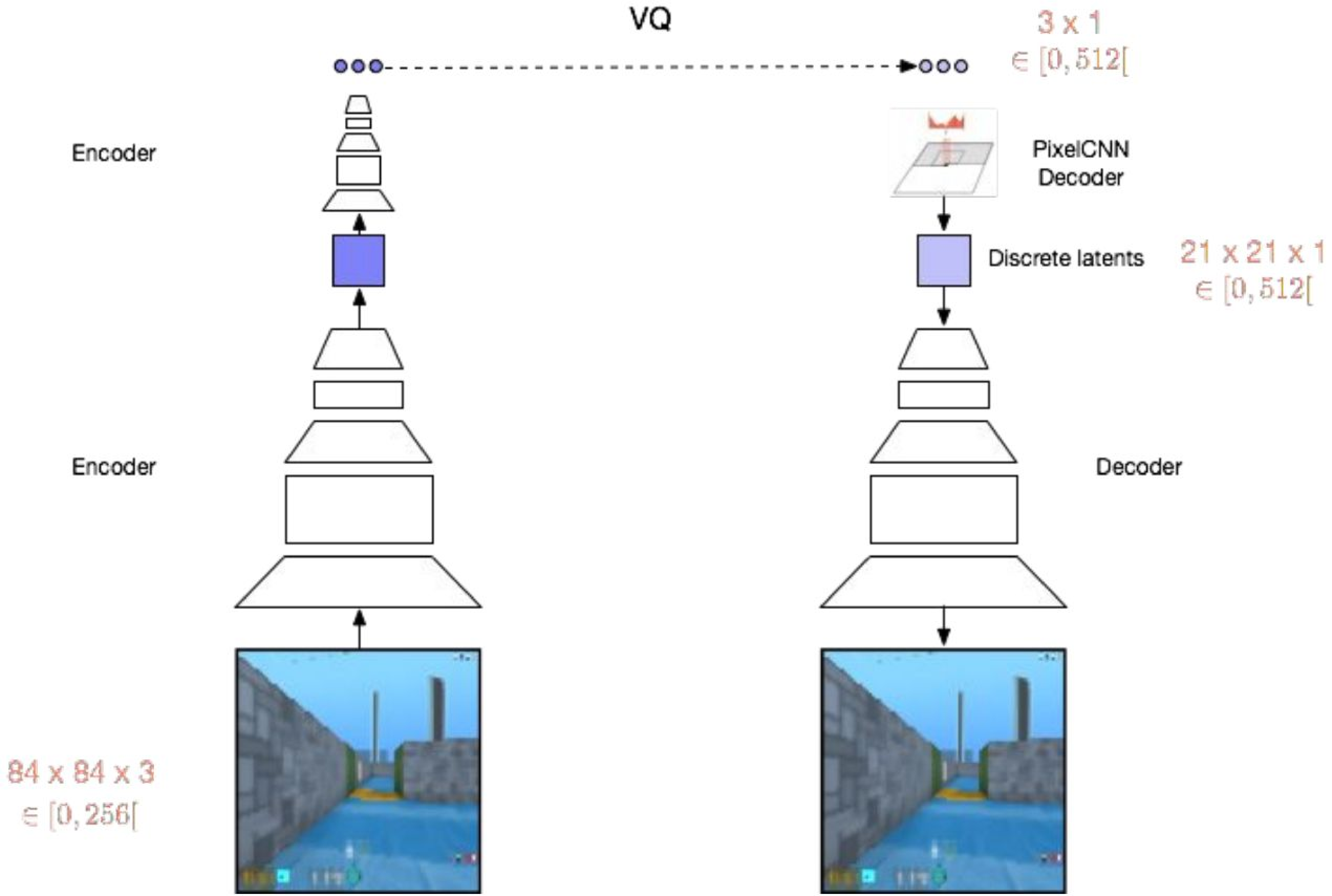
DeepMind Lab - Reconstruction

84x84x3 images
↔ 21x21x1 discrete latent space (K=512)
↔ 3x1 discrete latent space (K=512)

Two VQ-VAE layers!

3x9 = 27 bits in latent representation.

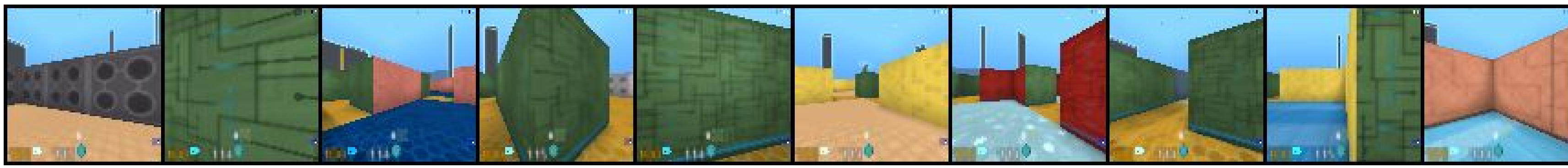
Can’t reconstruct exactly, but does capture global structure.



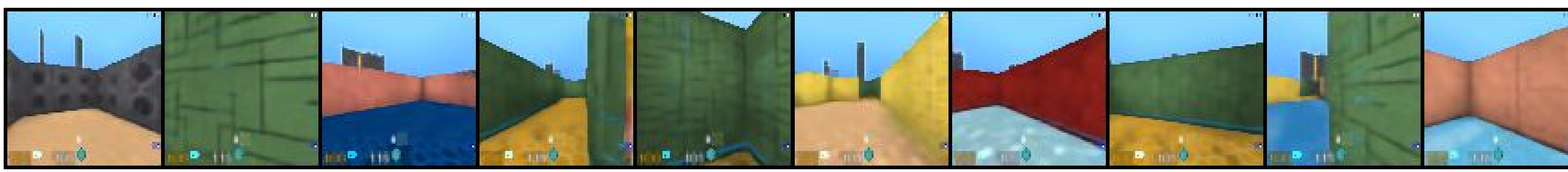
Source: https://avdnoord.github.io/homepage/slides/SANE2017.pdf

DeepMind Lab - Reconstruction

Original

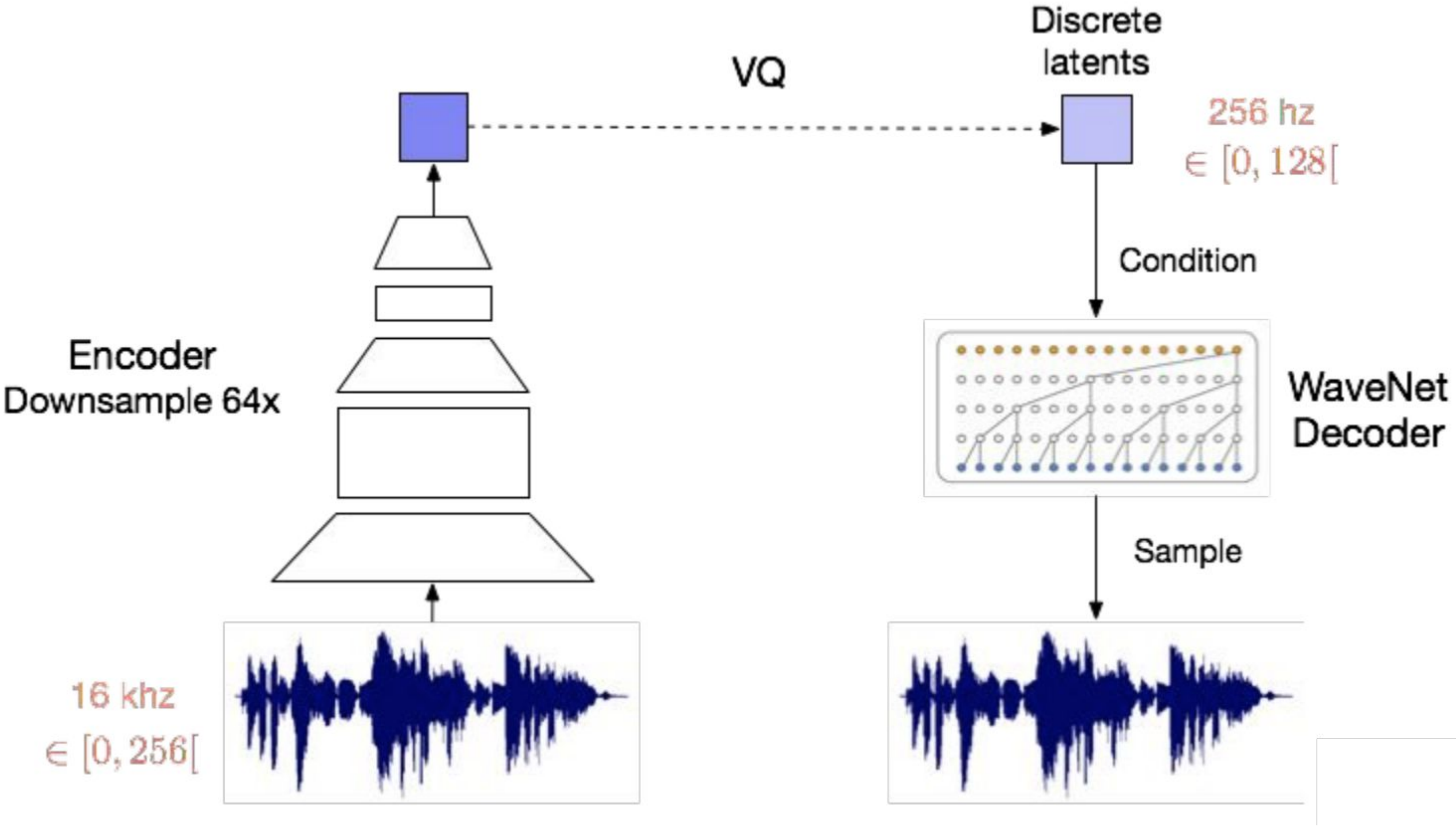


“Reconstruction”



Audio (VCTK) - Reconstruction

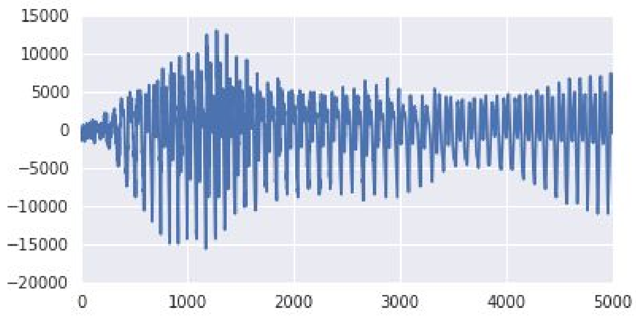
Use WaveNet decoder.



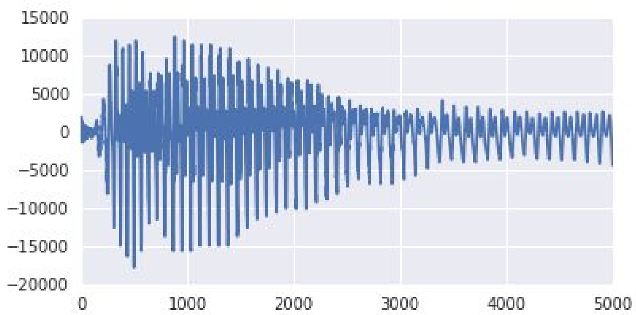
Source: https://avdnoord.github.io/homepage/slides/SANE2017.pdf

Audio (VCTK) - Reconstruction

Original



Reconstruction

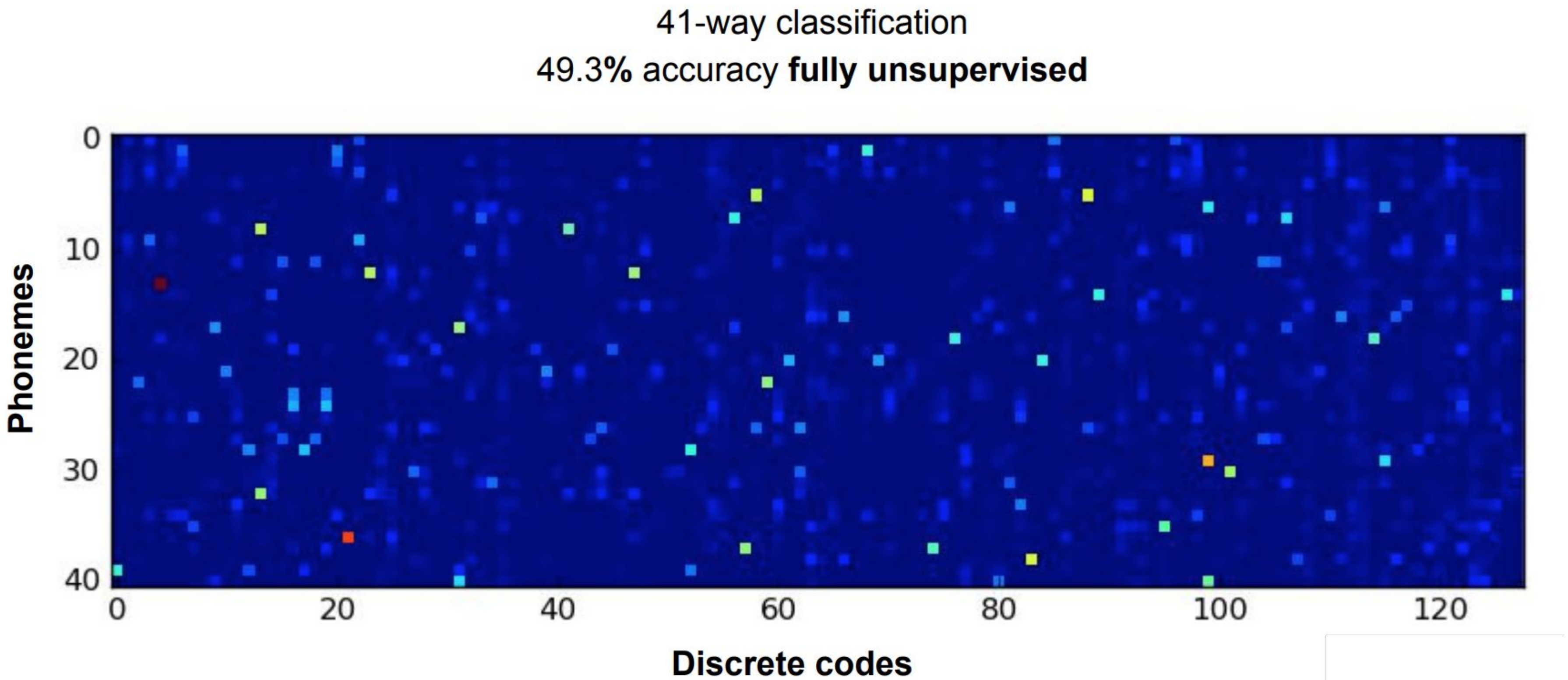


Again, not exact reconstruction, but captures global structure.

(More examples at https://avdnoord.github.io/homepage/vqvae/)

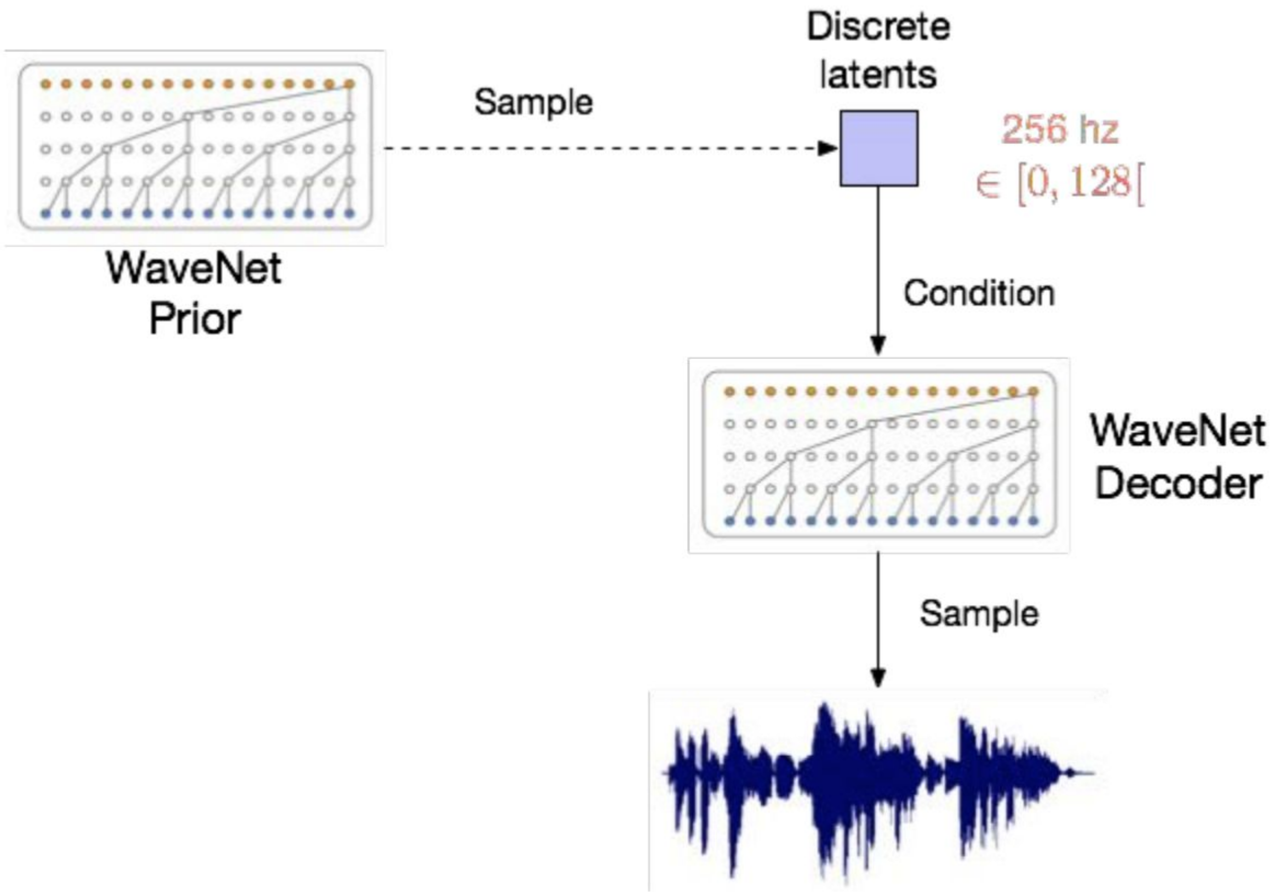
Audio (LibriSpeech) - Latents == phonemes?

It turns out discrete latent variables roughly correspond to phonemes.
Note that the semantics of discrete codes could be dependent on previous codes;
so it's interesting that individual discrete codes actually hold meaning!



Source: https://avdnoord.github.io/homepage/slides/SANE2017.pdf

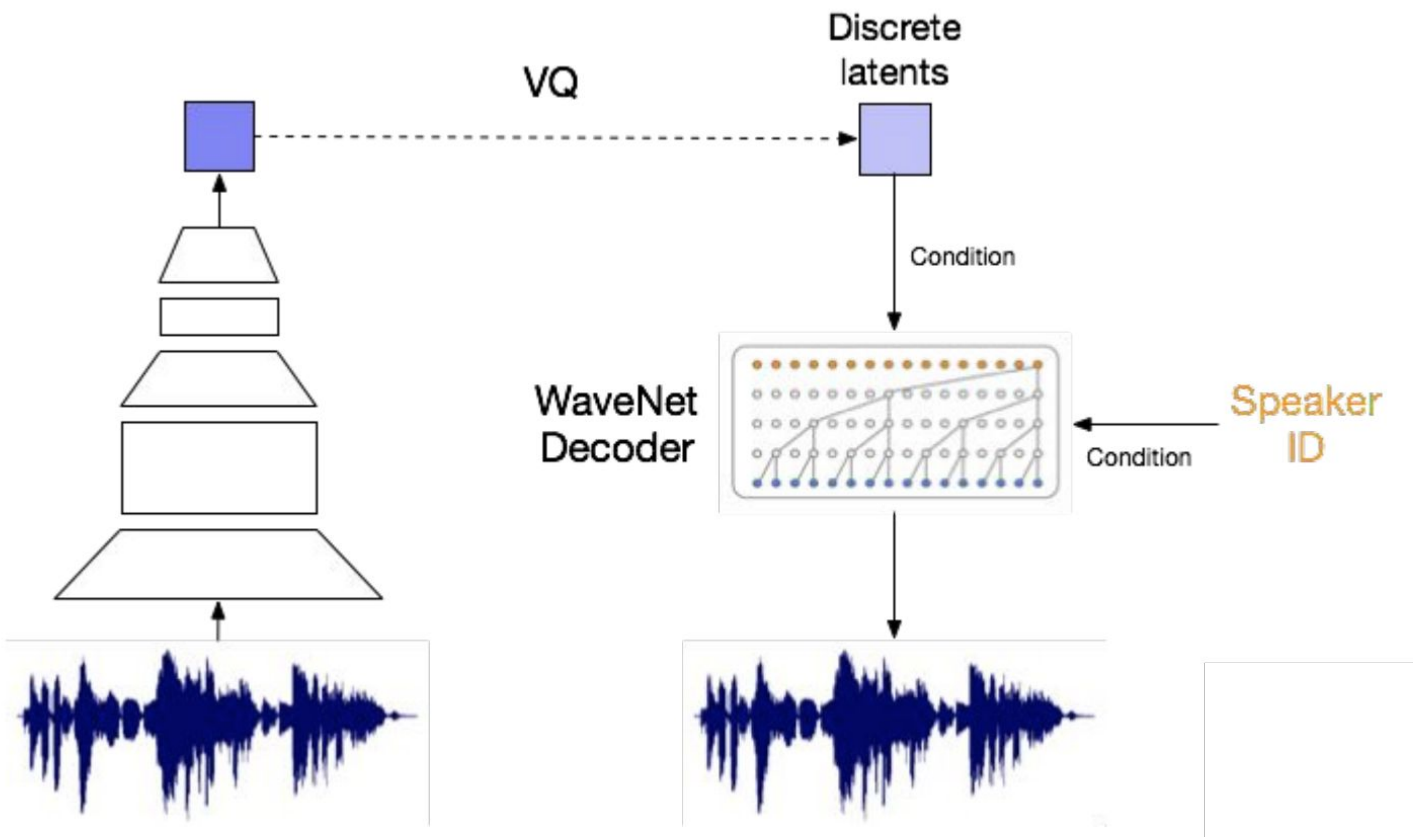
Audio (LibriSpeech) - Sampling



[Example]()

Source: https://avdnoord.github.io/homepage/slides/SANE2017.pdf

Audio (LibriSpeech) - Change Speaker Identity



[Original](#)

[Transferred](#)

=> Discrete latent variables are not speaker-specific!

Source: https://avdnoord.github.io/homepage/slides/SANE2017.pdf

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