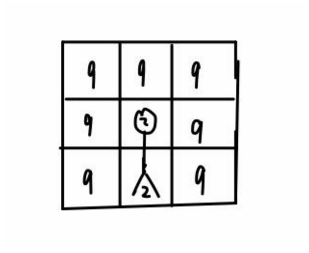
Byte-VAE: A Lightweight Alternative to VQ-VAE

Zeyuan (Faradawn) Yang
David A. McAllester

01 - Introduction

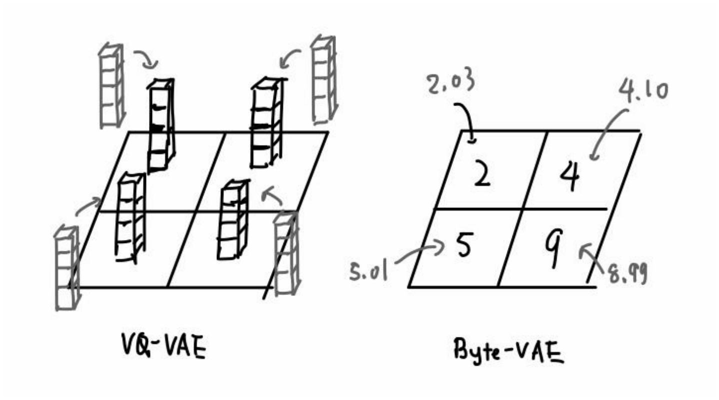
VQ-VAE is based on the idea that an image can be tokenized into discrete words. For example, the sky can be described using a token “9” and the person using token “2”.



However, VQ-VAE faces a number of challenges. 1) it finds a closest neighbor for each latent vector, which incurs a time complexity of O(n^2) and does not scale if the latent dimension increases. 2) It is hard to balance the update rate between the latent vectors and the codebook vectors, demanding a hyperparameters search on “beta”.

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

We propose Byte-VAE, a novel quantization method that rounds a floating point to an integer in [0, 255] – a “byte” – in contrast to VQ’s finding of nearest neighbors. This method avoids learning a bulky codebook while achieving a near-identical performance.

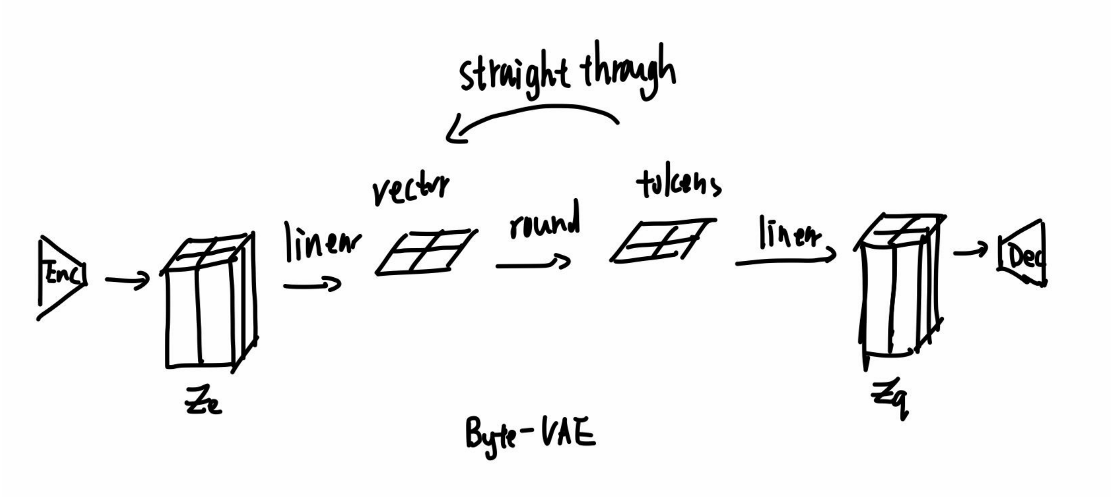


Byte-VAE offers two advantages:

1. **Fast and Memory Efficient:** rounding to the nearest integer is a O(1) operation compared to the nearest neighbor search, reducing time complexity. Also, it improves memory efficiency by removing the large codebook in cache.
2. **Simplified Training:** The codebook in VQ-VAE requires learning, which adds two additional loss terms: “commitment loss” and “nearest neighbor loss”. However, Byte-VAE removes the codebook learning and only uses a reconstruction loss.

02 - Method

Byte-VAE uses the same bottleneck framework as VQ-VAE with the center part modified: a simple rounding process instead of a codebook search:



Starting from the left, we begin with a CNN encoder. Then, we add a linear layer (1D convolution) to squeeze the latent image's each latent vector to a single number, "a token". Next, we convert the number "x" into the rage of [0, 255] using the formula below:

$$\left(\frac{x - \min}{\text{range}}\right) \times 255$$

Then, round the result to the nearest integer. Now, each pixel becomes a "byte token" that represents the meaning of that segment. In this way, we constrain our vocabulary size (number of unique tokens) to 256. Finally, we pass the tokens to a linear layer (1D transpose convolution) and decoder to reconstruct the image.

As mentioned previously, the loss function is simplified to reconstruction loss only

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

where z_q(x) is the final reconstructed image.

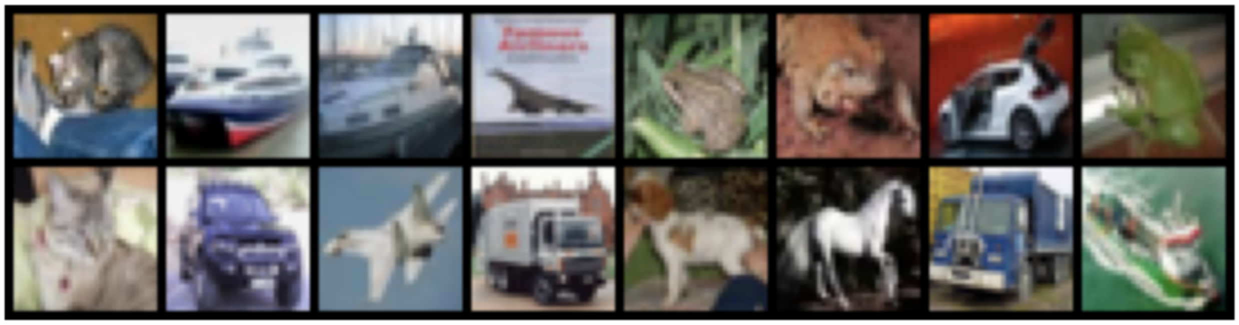
However, since the rounding operation is non-differentiable, we use a "straight-through" method to pass the gradients for tokens directly to the floating point vector. This is based on the same assumption as VQ-VAE that the tokens are close enough to the values before rounding. Although the dynamic range of floating points is high, the squeezing operation (x - min / range) is reversible, so the gradient update will be magnified or shinked accordingly.

<u>03 - Experiments</u>

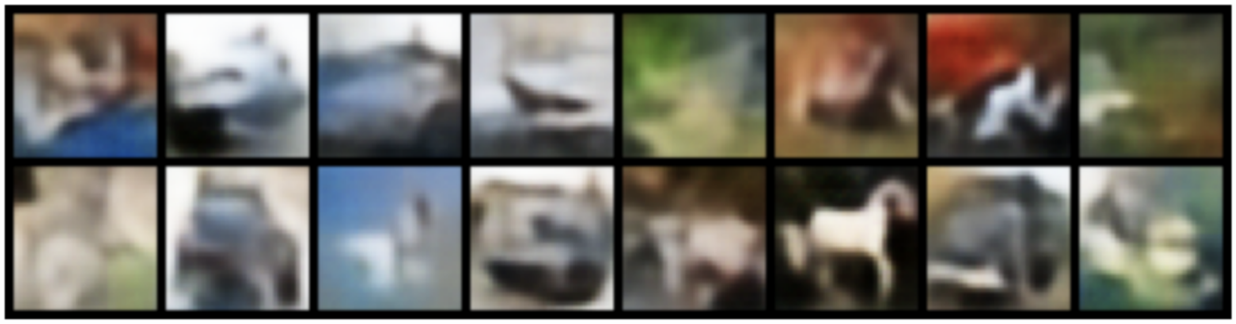
We compare our performance against VQ-VAE on the CIFAR-10 dataset (32x32x8). 1) The CNN encoder/decoder has latent dimension of 256 and space reduction of 4 times. 2) The linear layer (1D convolution) compresses a (8x8x256) latent image to (8x8x1). 3) The number of tokens is 256 for both VQ-VAE and Byte-VAE. 4) We use ADAM optimiser, a learning rate of 2e-4, a batch size of 128, and 20 epochs.

<u>Result 1: Reconstruction Quality</u>

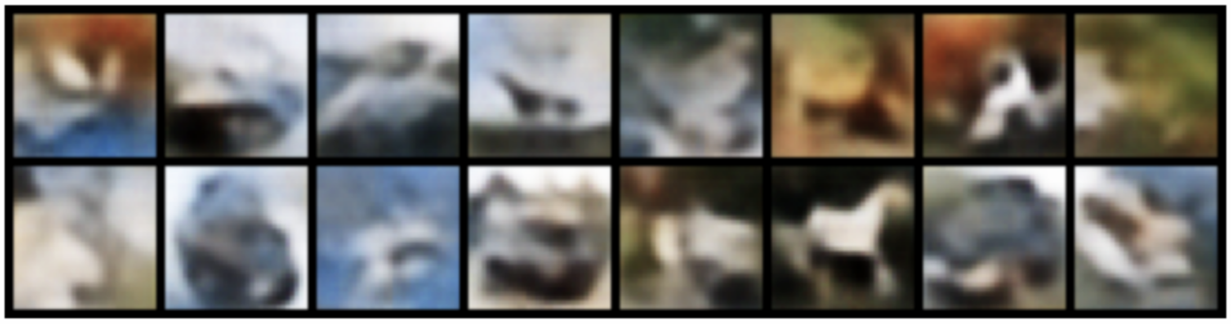
Original fixed Images



VQ VAE



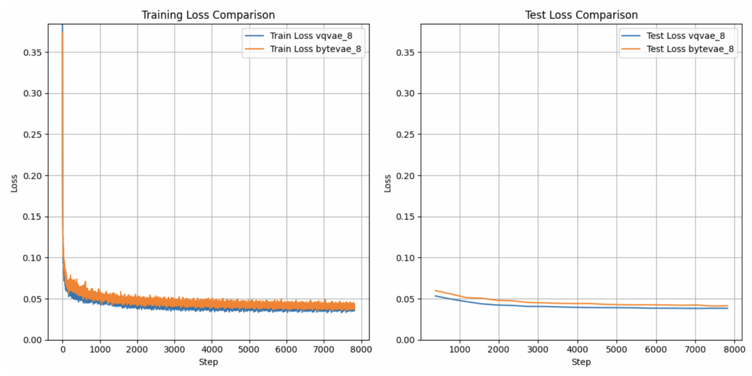
Byte VAE



We observe that Byte-VAE produces almost identical results to that of VQ-VAE in terms of structural coherence. Additionally, Byte-VAE displays a more consistent color scheme (less variance), likely due to the compression of the latent image's 256-dimension vector into a single number.

<u>Result 2: Convergence</u>

Byte-VAE achieves a near-identical loss (explained below) with VQ-VAE. In addition, the two show an identical convergence pattern.



The complete likelihood used in the VQ-VAE paper is

$$\log p(x) \approx \log p(x|z_q(x))p(z_q(x))$$

Since the same prior (pixelCNN) is used for both VQ and Byte VAE, we cancel the prior term for simplicity

$$\log p(x) = \log p(x \mid z_q(x))$$

Then, we take the negative of the log likelihood as our loss.

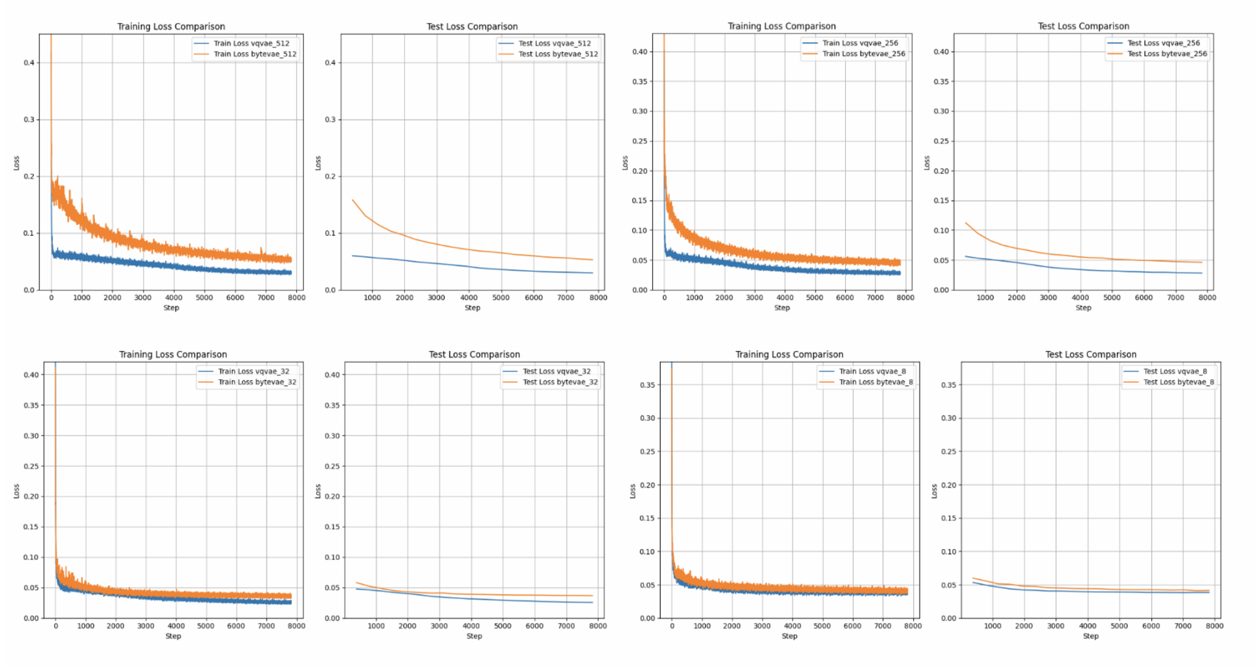
04 - Conclusion

We propose Byte-VAE, a novel quantization method that rounds each latent vector into a “byte token”. It 1) removes the nearest-neighbor searching, 2) simplifies the loss function, 3) reduces the number of parameters by removing the codebook. Yet, it achieves a near-identical performance as VQ-VAE for all values of token size: k = [512, 256, 32, 8] (appendix). In the future, we plan to create a hierarchy of byte-quantization, i.e., quantizing at every resolution during encoding.

Acknowledgement: special thanks to David A. McAllester for the Byte-VAE idea and Jose Marcelo Sandoval-Castañeda for the straight-through gradient.

05 - Appendix

We test different vocabulary sizes (number of tokens) using k = [512, 256, 32, 8]. We observe that the smaller the vocabulary size, the better performance of Byte-VAE. Therefore, in a memory constrained environment, Byte-VAE is likely to shine.


Training Loss Comparison
Train Loss vqvae_512
Train Loss bytevae_512
Test Loss Comparison
Test Loss vqvae_512
Test Loss bytevae_512
Training Loss Comparison
Train Loss vqvae_256
Train Loss bytevae_256
Test Loss Comparison
Test Loss vqvae_256
Test Loss bytevae_256
Training Loss Comparison
Train Loss vqvae_32
Train Loss bytevae_32
Test Loss Comparison
Test Loss vqvae_32
Test Loss bytevae_32
Training Loss Comparison
Train Loss vqvae_8
Train Loss bytevae_8
Test Loss Comparison
Test Loss vqvae_8
Test Loss bytevae_8
Loss
Step