Reproducing Neural Discrete Representation Learning

Rithesh Kumar*
MILA
ritheshkumar.95@gmail.com

Tristan Deleu*
MILA
tristan.deleu@gmail.com

Evan Racah*
MILA
ejracah@gmail.com

1 Paper Summary

1.1 Motivation

Generative Modeling and Variational Inference Stochastic Variational Inference (SVI) [5, 7] is a popular method for scalable posterior inference with large datasets using stochastic gradient ascent. It was recently made highly efficient for continuous latent variables through latent-variable reparameterization, and inference (recognition) networks, amortizing the cost. This resulted in a highly scalable learning procedure termed as Variational Auto-Encoders (VAEs, [11]). Variational training of this type involves maximizing a lower bound on the log-likelihood (called the ELBO), using samples from the variational posterior to compute the required gradients.

Discrete Latent Variables Probabilistic models with discrete latent variables naturally capture datasets composed of discrete classes (such as language, audio, and images). Discrete representations are often more interpretable, and are useful for representing distributions encountered in unsupervised learning, language modeling, attention mechanisms, and reinforcement learning domains.

However, Discrete Latent Variable Models (LVMs) are notoriously difficult to train efficiently, since backpropagation through discrete variables is generally not possible. Prior methods have explored solutions in the direction of 1) applying reparametrization trick to a continuous approximation of the discrete categorical distribution (Gumbel-Softmax distribution, [8, 13]) 2) developing unbiased gradient estimator for multi-sample (importance-sampled) monte-carlo objectives [14], where multi-sample refers to the idea that several independent samples can be averaged to compute the likelihood estimates. However, these proposed methods haven't performed nearly as well as their continuous counterparts.

Vector-Quantized VAEs The paper under study [18] combines 2 popular tricks: Vector Quantization (VQ) and the Straight-Through estimator (ST, [3]) to learn discrete latent variables in the variational inference framework. The VQ method solves the issue of "posterior collapse", where powerful autoregressive decoders tend to ignore the latent variables, and the ST estimator solves the problem of backpropogation through discrete latent variables (VQ operation in this case). When these tricks are paired with a powerful learned prior (rather than static / fixed prior) such as large, high-capacity autoregressive models (ex: PixelCNN [17], WaveNet [16]), the combined model can generate very high-quality images, videos and speech. The code for our experiments has been open-sourced, and is available at `https://github.com/ritheshkumar95/pytorch-vqvae`.

1.2 Proposed approach

The authors accomplish this by training a Vector Quantized Variational Autoencoder (VQ-VAE, [18]). This model has a neural network encoder and decoder, and a prior just like a vanilla VAE. But they also have a latent embedding space that we call a *codebook* $\mathbf{e} \in \mathbb{R}^{K \times D}$ (similar to a

* Author ordering determined by sampling from an autoregressive prior
Submitted as course project for IFT6135 H2018 Representation Learning

dictionary), where K is the size of the latent space (the number of latent embeddings to pick from), and D is the dimension of each embedding e_k. In a vanilla variational autoencoder, the output of the encoder, $z_e(x)$, is used to parametrize a Gaussian distribution, that is sampled from to get a latent representation z of the input x (using the reparameterization trick). This latent representation is then passed to the decoder.

In VQ-VAEs, however, $z_e(x)$ is used as a "key" to do nearest neighbor lookup into the embedding codebook and get $z_q(x) = e_k$, the closest embedding vector in the space. This is called the *Vector Quantization* operation, and it is shown in Figure 1.2. $z_q(x)$ is then passed to the decoder, which tries to reconstruct the input x. The decoder can parameterize $p(x \mid z)$ as the mean of a Gaussian distribution, using a deconvolutional network, like in vanilla VAE, or as an autoregressive categorical distribution over the $[0, 255]$ pixel intensities using a PixelCNN. The parameters of $q(z \mid x)$ are not directly inferred by the encoder, like in vanilla VAE. Instead, the posterior approximation $q(z \mid x)$ is a one hot distribution over the K latent embedding vectors

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

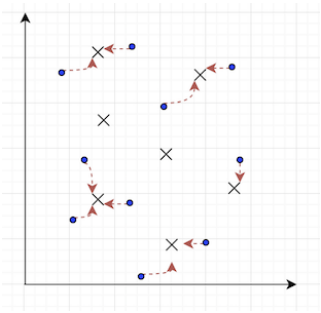

Figure 1: Illustration of Vector Quantization operation. Crosses denote the codebook entries, which can be interpreted as cluster centroids. Encoded vectors $z_e(x)$ are represented by the black circles. Vector quantization *quantizes* the continuous vectors $z_e(x)$ to the closest codebook entry (cluster centroid), as shown by the red dotted arrow. The vector representation of the cluster centroid – $z_q(x)$ is then passed as input to the decoder.

At training time, $p(z)$, the prior, is a uniform distribution over the K elements in the codebook. Similar to VAEs, the loss function ends up being the ELBO

$$\log p(x \mid z_q(x)) + \mathrm{KL}(q(z \mid x) \,\|\, p(z)) \tag{2}$$

plus two extra terms discussed below. Because $q(z \mid x)$ is a one-hot distribution and the prior is a uniform distribution, the KL divergence will be a constant, $\log K$, no matter what the output of the encoder is. Indeed

$$\mathrm{KL}(q(z \mid x) \,\|\, p(z)) = \mathbb{E}_{z \sim q(z|x)}\left[\log \frac{q(z \mid x)}{p(z)}\right] \tag{3}$$

$$= \sum_{k=1}^{K} q(z = k \mid x) \log q(z = k \mid x) - \sum_{k=1}^{K} q(z = k \mid x) \log \underbrace{p(z = k)}_{= 1/K} \tag{4}$$

$$= -H(q) + \log K = \log K \tag{5}$$

where $H(q)$ is the entropy of the distribution q, which is 0 as q is deterministic. Since this term is constant, it has no influence on the optimization procedure, and can be safely ignored. The other two terms are 1) a dictionary learning term, which encourages the items in the dictionary to move closer to the encoder output $\|\mathrm{sg}[z_e(x)] - \mathbf{e}\|_2^2$, and 2) a commitment loss where we encourage the output of the encoder to be close to the embedding it picked, to *commit* to its embedding, $\beta\|z_e(x) - \mathrm{sg}[\mathbf{e}]\|_2^2$

$$L = \log p(x \mid z_q(x)) + \|\mathrm{sg}[z_e(x)] - \mathbf{e}\|_2^2 + \beta\|z_e(x) - \mathrm{sg}[\mathbf{e}]\|_2^2, \tag{6}$$

where sg means "stop-gradient", so we don't propagate the gradient with respect to that term. The gradients for the first term update the decoder and the encoder by passing the decoder gradient, $\nabla_{z_q} L$ directly to the output of the encoder, so $\nabla_{z_e} L \leftarrow \nabla_{z_q} L$: this is the *Straight Through estimator*. The gradient for the second term is just taken with respect to the codebook, and the gradient for third term is just taken with respect to the encoder.

At test time, instead of sampling from our uniform prior over z, we can fit an autoregressive distribution to z using a PixelCNN prior [17], from which we can sample using ancestral sampling. This can lead to richer, more coherent latent representations. A visual description of the training and sampling of VQ-VAE is shown in Figure 2.

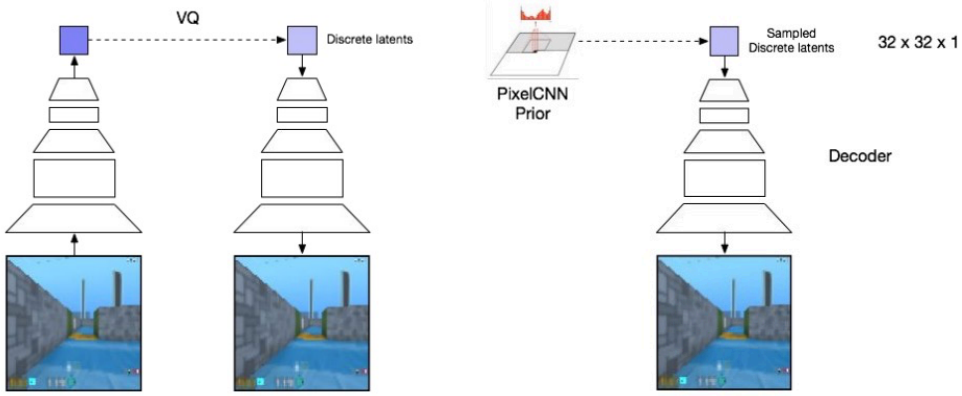

Figure 2: The VQ-VAE architecture, where the encoder is a convolutional network, and the decoder is a deconvolutional network. Left: Reconstruction from VQ-VAE. Right: Sampling from VQ-VAE.

2 Experiments

For the experiments, the authors [18] attempt to demonstrate how comparable VQ-VAEs are with continuous VAEs, as well as the general utility of VQ-VAEs for a variety of modalities (audio, images, video), for which the authors believed would benefit from discrete latents. The authors also aim to show VQ-VAEs robustness to *posterior collapse*, even when using a powerful autoregressive model for learning the prior, and as a decoder in some cases.

For all experiments in the paper, the authors used roughly the same base VQ-VAE architecture and loss function for inferring the latent representations, and the same PixelCNN architecture for learning the autoregressive prior. Also, there was a separate, smaller VQ-VAE and PixelCNN decoder used in Section 2.4, which we will further describe in that section. The main VQ-VAE used in the paper had an encoder with two convolutional layers of stride 2 and filter size of 4×4 with ReLU activation, followed by two residual blocks, which contained a 3×3, stride 1 convolutional layer with ReLU activation followed by a 1×1 convolution. The decoder was similar, with two of these residual blocks followed by two deconvolutional layers. All layers featured 256 filters. We tried to use roughly this architecture for all experiments, but we did make some changes for various reasons, notably for computational limitations. The changes to the VQ-VAE architecture, as well as our design choices for the PixelCNN are described in Section 2.1, where we used MNIST and CIFAR10 to help us select hyperparmeters.

2.1 MNIST and CIFAR10

In order to sanity check our implementation, which was non-trivial since the VQ-VAE framework has multiple moving parts (the discrete VAE, PixelCNN prior), each of which is tricky to implement on its own, we first tested the combined model on the famous MNIST dataset. Training on MNIST helped quickly tune certain hyperparameters for the VQ-VAE, which we found to work better than what is reported in the paper. Specifically, we noticed that the coefficient for the commitment term in Equation (6), β, provided much more stability in training when set to a value of $\beta = 1$, rather than when set to the author's choice of $\beta = 0.25$, which caused the model to diverge after 50 epochs of training. Moreover, we noticed that a learning rate of 3×10^{-4} worked well, and a batch size of 32 provided much faster convergence than the reported 128 in the original paper. We also note that we initialized the codebook with a $\mathrm{Uniform}(-1/K, 1/K)$ distribution, where $K = 512$ and $D = 256$.

We additionally ran our experiments on the FashionMNIST dataset, results for which are attached in the Appendix. Unfortunately, the paper does not report hyperparameters for the PixelCNN prior. We noticed that a 64-dimensional 12-layer Gated Conditional PixelCNN was strong enough to learn a powerful prior on the latents. To our surprise, Batch Normalization was essential to make VQ-VAEs work on MNIST dataset alone. We postpone investigation on this peculiar observation for the future.

We then moved to testing the model on the CIFAR10 with the above optimal hyperparameter configuration, and noticed a similar, decent performance. Results are shown in Figures 3 and 4. We used these hyperparameter settings as a starting point for the following experiments in Sections 2.2 and 2.3, though we did make a few experiment-specific changes that we describe in those sections.

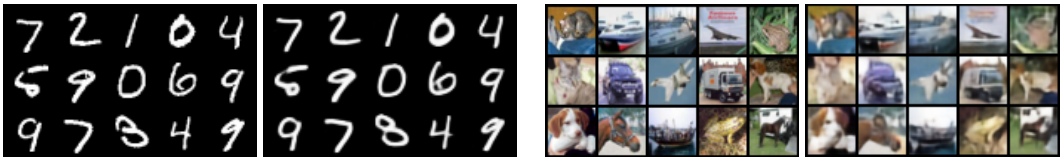

Figure 3: Examples of original vs reconstructed images with VQ-VAE trained on MNIST and CIFAR10 datasets. Left image in each pair correspond to original images. (28×28 grayscale MNIST images and 32×32 RGB CIFAR10 images) The corresponding image in each pair represents reconstructed images using the discrete latent space. ($7 \times 7 \times 1$ for MNIST and $8 \times 8 \times 1$ for CIFAR10 with $K = 512$).

Figure 4: Class conditional samples from VQ-VAE with PixelCNN prior

Finally, we quantitatively evaluated VQ-VAE on these datasets using the values of the ELBO. Similar to the original paper, we report the lower bounds on the log-likelihood for both VAE and VQ-VAE in bits/dim on all three datasets. We noticed a large gap between the reported results and ours, which we further discuss in Section 3.4.

	As reported in [18]	Ours		
	CIFAR10	CIFAR10	MNIST	Fashion MNIST
VQ-VAE	4.670	5.852	6.221	6.228
VAE	4.510	5.681	5.705	5.685
VIMCO [14]	5.140	–	–	–

Table 1: Lower bounds on the log-likelihood for different datasets. The values are reported in bits/dim.

2.2 Mini-ImageNet

In order to test the model at a larger scale (ie. on larger input data), we looked into reproducing the experiment on the Imagenet dataset [12] from the original paper. The authors trained a VQ-VAE on 128×128 RGB images, with a latent space of $32 \times 32 \times 1$ discrete elements ($K = 512$). However, to make our experiments more practical, we decided to run the model on a smaller subset of Imagenet, called *Mini-Imagenet*. Mini-Imagenet is a dataset which was originally introduced to study few-shot learning classification problems [15, 19], whose training split contains 34k images from 64 different classes. It also contains validation and test splits, both having different subsets of classes ; this is a classic setup in the few-shot learning literature, to test the generalization capacity of these algorithms.

This is particularly handy here to test the ability of VQ-VAE to not only reconstruct accurately samples from the training set, but also completely out-of-sample images.

Our setup is similar to the one described in the original paper, the only difference being in the size of the embedding ($D = 64$ instead of $D = 256$). In particular, the generative model here (ie. the decoder) is a deconvolutional network. Figure 5 shows some samples from the test part of Mini-Imagenet, along with the reconstructions from VQ-VAE, using a discrete latent space with $K = 512$. The reconstructed images closely match their original counterparts, even with a $\frac{128 \times 128 \times 3 \times 8}{32 \times 32 \times 1 \times 9} \approx 42.6$ bits compression[1]. They are only slightly blurrier, which is particularly noticeable on smaller pieces, like text.

Figure 5: Examples of reconstruction with VQ-VAE on Mini-Imagenet (test set). Left: Original 128×128 RGB images. Right: Reconstructed images with a VQ-VAE from a $32 \times 32 \times 1$ discrete latent representation ($K = 512$).

Similar to the original paper, we then trained a 15-layer PixelCNN prior on the $32 \times 32 \times 1$ discrete latent representations given by VQ-VAE. Figure 6 shows some images generated by VQ-VAE with a PixelCNN prior. While these samples are not as appealing as the ones reported in the paper, they still display some local coherence when compared to samples generated from a uniform prior. We conjecture that the smaller capacity of our PixelCNN prior, relative to the one used in the paper, might explain this discrepancy.

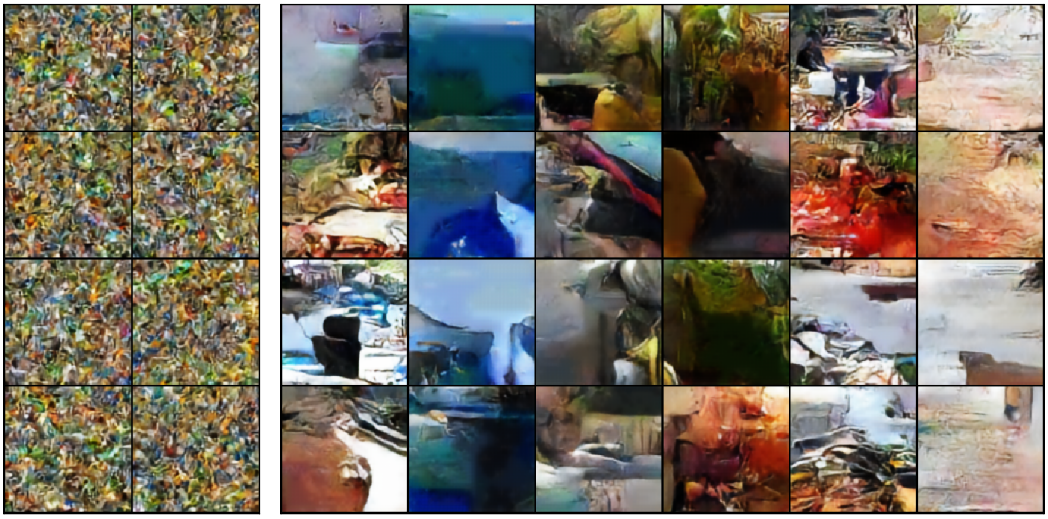

Figure 6: Generated samples with VQ-VAE, trained on Mini-Imagenet. Left: With a Uniform prior on $[1, K]$ ($K = 512$). Right: With a PixelCNN prior, conditional samples. From left to right: Newfoundland (Canadian province), snorkel, green mamba (snake), boxer, carousel, Gordon Setter (dog).

[1]The original images are $128 \times 128 \times 3$ bytes, or 8 bits, while the latent representations is $32 \times 32 \times 1$ with elements in $[1, 512]$ (9 bits of information). The compression factor does not take into account the embedding size, nor the decoder parameters.

2.3 Atari Boxing

For the final two image experiments, the authors used frames from the DeepMind Lab environment [1]. Due to time constraints, and the notorious difficulty of dealing with this environment, we decided to use frames from an Atari game instead [2, 6]. We chose Boxing due to its simple appearance and the ease at which one can acquire a representative distribution of frames from the game. We procured 60,000 frames of Boxing by stepping through the environment with a random policy. Then, we downsampled them to 84x84, which is the size of the DeepMind Lab frames used in the original paper.

2.3.1 VQ-VAE with Autoregressive Prior and Deconvolutional Decoder

For this experiment the authors run the same experiment from section 2.2, but on DeepMind Lab freames. This likely aligns with their desire to show the value of VQ-VAE on a variety of datasets, but also this experiment serves as a baseline for the experiment in section 2.4. The authors claimed to have used a VQVAE exactly like the one they used on ImageNet. As such, we used the same architecture, we determined to work from section 2.2. Using this same architecture with $84x84$ images instead of $128x128$ results in output feature maps of size 21×21 for z_e, so our latents z end up being $21 \times 21 \times 1$. Also, we use $D = 256$ instead of $D = 64$. In the spirit of Section 2.1, we attempted to use a batch size of 32. However, we had severe memory issues in training the Atari frames and could only fit a batch of 4 frames in the GPU at one time. We ended up having to use pseudo-batches, where we use a batch size of 4 and then only update the parameters every 8 batches to get an effective batch size of 32. The reconstructions from this model are shown figure 7.

Qualitatively, they are very similar to the originals, which is comparable to the high quality the original authors achieved on their DeepMind Lab samples.

For sampling, we fit a prior to the $21 \times 21 \times 1$ latents with a 15 layer PixelCNN with a hidden dimension size of 256. As there is no class for the Boxing environment, there was nothing to condition the PixelCNN on, but we tried conditioning it on 0 (no conditioning) and 1. As we can see from Figure 8, the samples with no conditioning are very good, but the ones with conditioning don't quite look like Boxing frames. It would be interesting future work to determine why this happens.

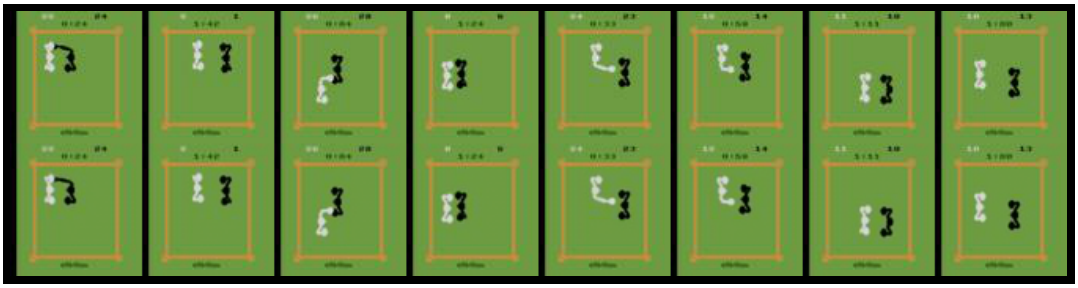

Figure 7: Examples of reconstruction with VQ-VAE on Atari Boxing (test set). Top: Original 84×84 RGB images. Bottom: Reconstructed images with a VQ-VAE from a $21 \times 21 \times 1$ discrete latent representation ($K = 512$).

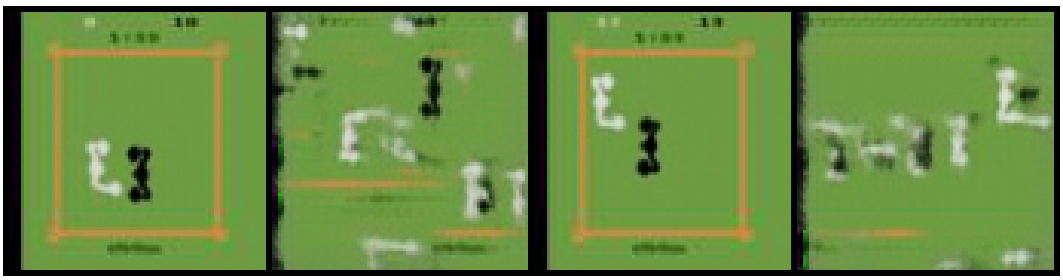

Figure 8: Generated samples from VQ-VAE, trained on Atari Boxing with a PixelCNN prior, conditioned on integers. From left to right: 0,1,0,1

2.4 Second VQ-VAE Trained on Latents of First One with PixelCNN Decoder

For this experiment, the authors took the $21 \times 21 \times 1$ latents from their DeepMind Lab VQ-VAE (we'll call this the first VQ-VAE) and use them as inputs to a second VQ-VAE with a PixelCNN decoder that compresses them to 3x1 latents using three different codebooks with $K = 512, D = 256$ (this results in compressing each frame to 3 9-bit integers). After sampling the "reconstructed" 21x21x1 latent code from the PixelCNN decoder, the corresponding z_q is passed through the deterministic deconvolutional decoder of the first VQ-VAE . This general procedure is shown in 9. This experiment is part of the author's attempt to demonstrate their architecure's robustness to *posterior collapse*, as they combine a very compressed latent with a powerful decoder (PixelCNN) and still show good results (Figure 10).

We attempted to run this exact experiment, but with Boxing frames instead of DeepMind Lab frames, so for our "first VQ-VAE", we use the pretrained one from section 2.3.1. The authors failed to describe the architecture of the second VQ-VAE, so for the encoder, we went with a similar architecture as the VQ-VAE, but we added a 3×3 convolutional layer at the end with 3 filters and an average pooling layer in order to get the desired 3x1x1 latents. We use the same codebooks as the authors (one codebook of $K = 512, D = 256$ for each of the three latents). For the PixelCNN decoder, we used a Gated PixelCNN from [17] with 5×5 filters, 16 filters per layer and 15 layers conditioned on the 3×1 latents, the design partially inspired by the PixelCNN from 2.1. We found a learning rte of 0.003 was best.

Samples from this model are shown in Figure 9. As expected, if you try compressing the Atari frames further (to 3 9-bit integers), it retains only information that minimizes the cost. So global information like background is preserved since it's 80-90% of the image, but local detail like the players are lost, hence we see mostly green squares. We are still not sure how the original authors achieved such good reconstructions (their reconstructions shown in Figure 10), but it may have had to with their PixelCNN architecture design, which they did not report in the paper.

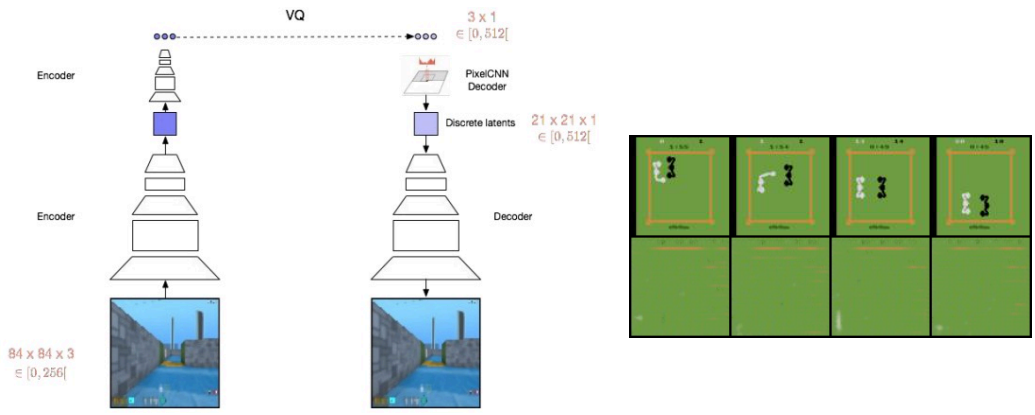

Figure 9: Examples of reconstructions created, where images are encoded into $21 \times 21 \times 1$ latents by the VQ-VAE from section 2.3.1, then these latents are encoded by a second VQ-VAE to a 3×1 latent discrete latent representation (Three codebooks, each with $K = 512$). The $21 \times 21 \times 1$ representation is then reconstructed with a PixelCNN decoder and then passed through the deconvolutional decoder from the VAE in section 2.3.1 to reconstruct the original image. Left: A diagram showing this experiment. Right: Reconstructions: top is originals, bottom is reconstructions

Figure 10: The original authors' [18] reconstructions from one VQ-VAE trained on top of another. Left: Original, Right: Reconstructions. Copied from [18].

3 Discussion

3.1 Missing Information

The paper lacked certain details for running the experiments, which we discuss in depth in the experiments section. Namely, the architecture of the PixelCNN prior used for most experiments, as well as the second VQ-VAE and the PixelCNN decoder used in the experiments in section 2.4. In addition, certain details like learning rate and initialization scheme of the codebook were missing. Furthermore, the details on how they did the video experiment were very sparse, which factored into why we did not try to reproduce it (see section 3.4). Lastly, they did not discuss how they came to compute their bits/dim measurements for log-likelihood that they used to compare VQ-VAE to its continuous VAE counterpart. This led to a lot of confusion in trying to reproduce those results. See section 3.4.

3.2 Flaws in the Experiments

It would be informative to know what types of tasks are unsuitable for discrete latents, so it would have been interesting to see the authors include an experiment where the VQ-VAE fails. To motivate discrete autoencoders, the authors point to language being discrete, so it would have also been interesting to see an experiment with text data. To be fair, the authors did do extensive experiments with audio data.

3.3 Analysis

Random embeddings Interestingly, we noticed that randomly initializing the codebook (embeddings) and never training them with the Vector Quantization objective (dropping the second term of the loss in equation 6) produces very similar overall results to actually training them. This observation was extremely peculiar at first, but upon thinking, we arrived at some plausible explanations for this. Interpreting Vector Quantization as an operation that maps a point to its closest cluster centroid (codebook entry), the codebook can be understood as a set of cluster centroids in n-dimensional space. If the cluster centroids are sufficiently spaced apart (through clever random initialization like in K-Means clustering), it may never be necessary to move these clusters around in space, since the parametric encoder can learn to place itself close to a particular centroid, rather than moving centroids towards an encoded point. While the special arrangement of codebook entries might provide some extra expressivity to the model, in practice it seems that the added expressivity is negligible.

3.4 Implementation issues

Memory Issues As described in sections 2.3.1 and 2.2, we had issues using reasonable batch sizes without running out of GPU memory in training (section 2.3.1) and in sampling (both sections). This is mainly due to the VQ operation, which requires the computation of pairwise distances. Our exact implementation of the nearest neighbor search had a space complexity of $O(K^2)$[2], which in addition to the model parameters put too much constraint on the memory of a single GPU. One solution could be to use an approximate nearest neighbors technique [4, 9] in place of our exact implementation.

Domain limitation Although the original paper featured experiments across multiple domains (images, video and audio), we decided to focus our attention on reproducing the experiments on image datasets for this project. Indeed, the experiments on the other two domains required some heavy architectures, like VPN for video [10] and WaveNet for audio [16], each of which would be a full project on their own. Moreover, the details on how they acquired their video results were lacking. That being said, in addition to the original experiments, we also included some experiments on other datasets, including MNIST, Fashion MNIST and Atari.

Metric computation As mentioned in Section 2.1, the values of the lower bounds we computed are significantly larger than the ones reported in the original paper. This might be due to our misunderstanding in the details of the conversion from raw log-likelihood lower bound to their values

[2]The batch-size appears implictly in the multiplicative constant in the Big-O notation.

in bits/dim. The formula we used to compute the reported values is

$$\text{ELBO}(x, \tilde{x}) = \frac{1}{N}\left[\underbrace{\frac{1}{2}\left(-N\log(2\pi) - \|x-\tilde{x}\|_2^2\right)}_{=\log\mathcal{N}(\tilde{x}|x,\mathbf{I})} + \underbrace{M\log K}_{=\text{KL}}\right] + \log 128 \tag{7}$$

where $\tilde{x}$ is the reconstructed data, N is the size of the input data x, and M is the size of the latent representation. The $\log 128$ accounts for the fact that all our images are normalized in $[-1, 1]$.

4 Acknowledgements

Shoutout to the two Aarons, Courville and van den Oord. Also Chin-Wei was a great explainer of all things VAE. And let's not forget Kingma (what a beast).

References

[1] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[3] Y. Bengio, N. Léonard, and A. C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

[4] E. Bernhardsson. Annoy: Approximate Nearest Neighbors Oh Yeah, 2013.

[5] D. M. Blei, M. I. Jordan, et al. Variational inference for dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143, 2006.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[7] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[8] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[9] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017.

[10] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. Video pixel networks. *CoRR*, abs/1610.00527, 2016.

[11] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[13] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016.

[14] A. Mnih and D. J. Rezende. Variational inference for monte carlo objectives. *CoRR*, abs/1602.06725, 2016.

[15] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. 2016.

[16] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[17] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.

[18] A. van den Oord, O. Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6309–6318, 2017.

[19] O. Vinyals, C. Blundell, T. P. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016.

Appendix A Additional Experiments: Fashion MNIST

Along with our experiments on MNIST and CIFAR10, we also experimented with the Fashion MNIST dataset. This dataset has the same characteristics as the original MNIST dataset (same input dimensions, same sizes of train/test sets), but features more complex shapes with smaller details. Figure 11 shows some samples from this dataset, along with their reconstruction with VQ-VAE. Figure 12 shows some samples generated with VQ-VAE and a PixelCNN prior.

Figure 11: Examples of original vs reconstructed images with VQ-VAE trained on Fashion MNIST. Left: Original 28×28 grayscale images. Right: Reconstructed images using the $7 \times 7 \times 1$ discrete latent space ($K = 512$).

Figure 12: Class conditional samples from VQ-VAE with PixelCNN prior

Appendix B Additional Figure

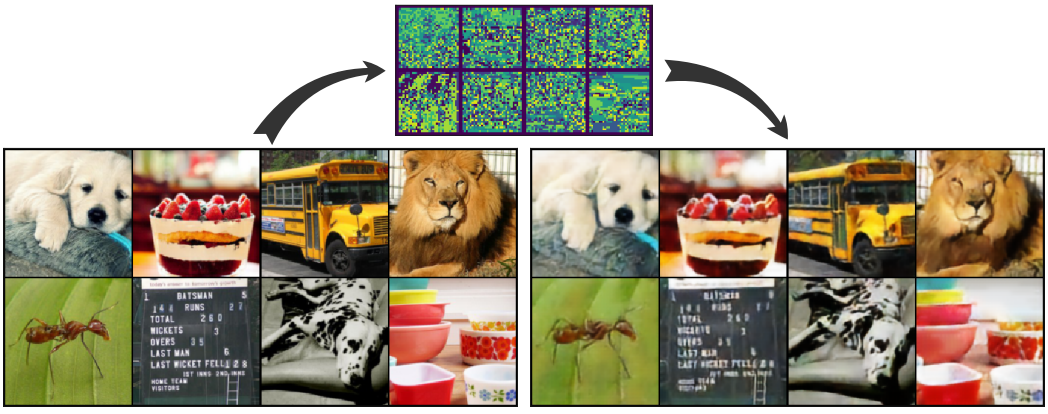

Figure 13: Examples of reconstruction with VQ-VAE on Mini-Imagenet (test set). Left: Original 128×128 RGB images. Right: Reconstructed images with a VQ-VAE from a $32 \times 32 \times 1$ discrete latent representation (top, $K = 512$). Similar to Figure 5.