Winged Horses with a Vector Quantized Variational Autoencoder

gwwb64

Abstract

This paper proposes the use of a Vector Quantized Convolutional Variational Autoencoder with a PixelCNN prior (VQVAE) to generate images of winged horses using the CIFAR10 or STL10 datasets. Regular autoencoders are often plagued with high reconstruction loss and sampling from their latent space leads to poor results. Prior to settling on the VQVAE architecture, multiple experiments were performed with regular variational autoencoders (VAEs). While better than a standard autoencoder, their reconstructions were still barely recognisable even after 1000s of epochs and hence sampling the Gaussian latent space still produced poor results. With several limitations, VQVAE produced significantly stronger results.

1 Methodology

The method for generating images using a VQVAE is quite different to that of traditional VAEs. Firstly, as in as in [7], instead of the latent space being a continuous Gaussian distribution, it is defined as a discrete embedding $e \in R^{KxD}$ where K is the size of the latent space and D is the dimensionality of each latent vector in the embedding. In [7], $K = 512$ is used but based on implementations by others [1, 2, 3, 5], a value of $K = 256$ or $K = 512$ both work. For the purposes of this, $K = 512$ was chosen, in line with the original design. The value of D did not seem to matter much in our experiments so $D = 64$ was chosen as is it is a square number. The posterior distribution $q(z|x)$ is given using the equation:

$$q(z = k|x) = \begin{cases} 1 & for\ k = argmin_j \|z_e(x) - e_j\|_2 \\ 0 & otherwise \end{cases} \tag{1}$$

where e is the embedding space, $z_e(x)$ is the output from encoding x, e_j is the particular latent embedding vector and z represents the latent space [7]. The input of the decoder is denoted as $z_q(x)$ which consists of quantized latent vectors using the closest encodings found in the embedding lookup and is defined as

$$z_q(x) = e_k, where\ k = argmin_j \|z_e(x) - e_j\|_2 \tag{2}$$

Unfortunately the VQVAE lookup is not differentiable so the gradient cannot be backpropagated during training. To circumvent this, the encoder/decoder end/start with residual blocks respectively and the gradient is copied from the decoder input to the encoder output during backpropagation. Residual blocks are constructed as so: ReLU, 3x3 Conv2D, ReLU, 1x1 Conv2D. It is necessary to train both the encoder/decoder network as well as the embedding space which creates the final loss equation as in [7]:

$$L = \log p(x|z_q(x)) + \|\text{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \text{sg}[e]\|_2^2 \tag{3}$$

The first term is the reconstruction loss and our implementation uses mean squared error for the reconstruction loss instead of binary cross entropy. β is used to weight the commitment loss which helps ensure that the embeddings train at a similar rate to the encoder. The characters sg are simply used to denote that these terms do not use gradients as previously discussed. Our implementation uses $\beta = 0.25$ as in [7].

The architecture in Figure 1 is used to train the VQVAE and can reconstruct images from given inputs. Unlike a regular VAE where the Gaussian distribution can be sampled and

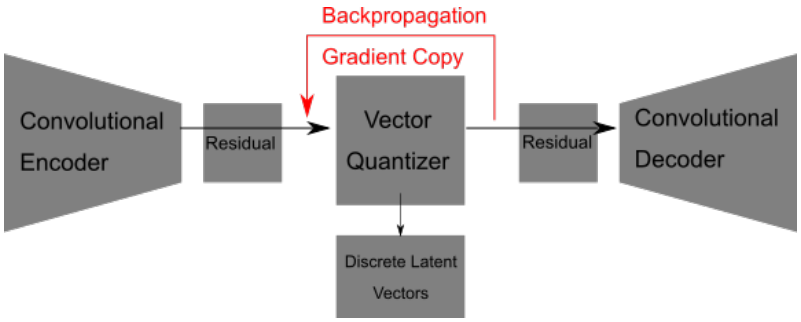

Figure 1: The overall architecture of the VQ VAE. Latent vectors are returned from the VQ as well as passed on

produce some meaningful results, random sampling of the discrete latent vector space, quantization and input into the convolutional decoder will generally output noise or highly garbled images that in no way represent anything related to the dataset. Experimentally it was found that using Sigmoid as the decoder output worked better than Tanh if for no other reason than the outputs into imshow were correctly scaled and didn't get clipped.

For this reason, the prior must be trained with a generative model such as PixelCNN [6]. This combination works well since the dimensionality of the latent vectors is 64 so they can be considered as size 8x8 grayscale images input directly into a PixelCNN (which doesn't need to consider RGB data). The [6] architecture uses stacks of ReLU and 1x1, 3x3, 1x1 Conv2d blocks over many layers to generate images. Generation happens from left to right and top to bottom and each pixel relies on the pixels before it. To ensure that pixels aren't looking ahead, masking is used. However, PixelCNNs can exhibit blind spots where some pixels do not depend on all previous pixels so instead we chose to use [8] which introduces a Gated Conditional PixelCNN which solves the blind spot problem and also allows the learning of latent spaces relating to a certain class which was desirable in the experiments we performed. Formally, a conditional PixelCNN models an image using the equation in [6]

$$p(\boldsymbol{X}|\boldsymbol{h}) = \prod_{i=1}^{n^2} p(x_i|x1, x2, ..., x_{i-1}, \boldsymbol{h}) \tag{4}$$

where X is an image, x_i is a pixel in the image and $\boldsymbol{h}$ is a condition. For this paper, $\boldsymbol{h}$ may represent the class label. The gated PixelCNN involves splitting the image generation into horizontal and vertical components which are individually processed and then combined at a gate consisting of a Tanh and Sigmoid function multiplied together with half the input going into each. We used the same sort of architecture as the original PixelCNN paper with 15 layers of blocks with the top one using Mask A and the others Mask B. The only difference was the use of the gated conditional blocks instead of residual blocks. The PixelCNN training involves minimising the cross entropy loss between the initial image (conditional on its label) and the generated image. Once training is complete, some priors can be sampled from the PixelCNN and then converted into encodings using the embedding weights as in the Vector Quantizer originally. These can then be fed directly into the decoder as suggested in [7].

Attempts were initially made to sample the latent space for different classes and then combine the sampled latents in some way. With regular VAEs, interpolation between latents is trivial. However, with the VQVAE it was found that even upon writing a function to lerp between discrete latents and keep them as integers (by converting the interpolated float tensor to long, hence cutting off decimals), interpolation did not work. The decoder output was a garbled mess of colour patterns.

The technique settled on was changing the training. Initially training was performed on horses and birds with corresponding class conditions on the PixelCNN. This time, a 3rd class was introduced and during PixelCNN training, random labels for horses or birds were switched to class 3. Each label in a batch had a 0.075 chance of being switched. This was

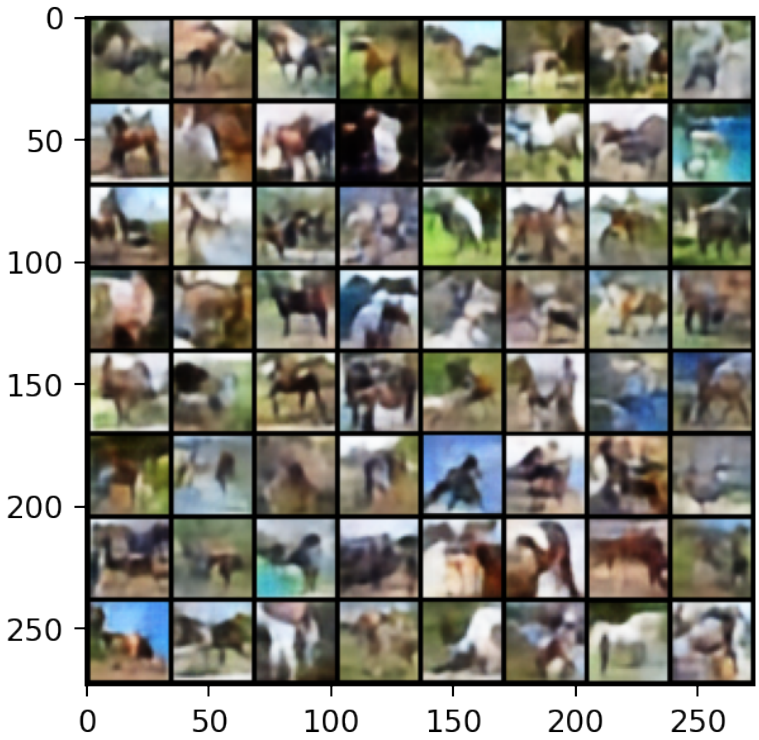

Figure 2: The best batch

arbitrarily chosen although the aim was to strike a balance between variation in the images and having a latent space that becomes strongly coupled between classes (and hence can't generate properly).

2 Results

The results are unfortunately quite unimpressive and even then, getting a reasonable batch as in Figure 2 took a few attempts and was quite hit-or-miss. There are a couple of candidates for best horse from the batch but all of them are a bit mutant. One with a white body, cream coloured wings on the right side pointing up and black legs was chosen as in Figure 3. Unfortunately, some imagination is required because the head has become quite deformed. On the plus side, the VQVAE performed better than all other models attempted for the purpose of generating Pegasi. Regular VAEs with multiple configurations of hidden dimension depth, sizes and latent space dimensions were tried. Additionally, modifying the VAE loss function in various ways was attempted as was adding warm-up to the KL-divergence term to try force reconstruction accuracy in early epochs. None of these gave good results, even after 1000 epochs with a couple of horse reconstructions looking vaguely horse-like but most bird reconstructions looking like blurry potatoes. An attempt was also made to implement a Ladder-VAE [10] but this took a lot longer to train and for some reason performed worse than the regular VAE. The VQVAE was very good at reconstruction (even after only 100 epochs), as expected with a discrete neighbourhood latent space so it is definitely the PixelCNN prior training that limits this technique. One final attempt at creating a VAEGAN [4], which may work better than the VQVAE + PixelCNN combination, was made but it ran into severe Runtime Errors with some inline operation but since none were directly used in the code it appears there was some strange interaction between components happening.

Figure 3: The best winged horse

Figure 4: STL-10 Pegasus sampling failure (not part of submission)

3 Limitations

Most of the examples in the batch do look like winged horses with varying degrees of imagination required. It was disappointing that it wasn't possible in the time constraints to generate better images. There are likely 2 causes for this. One of them is that the mixing technique to create a 3rd class causes the prior to learn a messy amalgamation between horses and birds. Since the prior interpolation technique wasn't viable, it could potentially be possible to improve results by training only on specific images that are already of the correct sort of rotation/positioning so that when merged together the Pegasus images will always look somewhat decent. However, this would be a lot of manual work and also is not a replacement for just a better model. The work in [7] mentions the global structure of images and this is probably the bigger issue with the PixelCNN prior. This prior is capable of accurately capturing local structure in the images and generating images with these structures but when it comes to generating images with accurate global structure, it is less capable. Intuitively, this means it understands what makes up a horse, but doesn't truly understand how these elements should be arranged and hence some of the more "spaghettified" images. Using a hierarchical prior such as in [9] would be capable of capturing global structure and hence would probably have better results. However, creating a PixelSNAIL + PixelCNN hierarchy as in the paper was a bit too complex for this paper. The PixelCNN was trained between 100 and 200 epochs but it doesn't seem likely that increasing the number of training epochs would improve the structural deficiencies to any great extent.

One major advantage of the VQVAE + PixelCNN technique is that it can easily process input images of any size since the latent space size is based on the fixed size embedding dimensions. A version of the VQVAE was trained on STL-10 at 96x96 but unfortunately the results for the Pegasus generation using the 3rd class were unusable. All generated images looked incredibly messy, severely lacking coherence and global structure as in Figure 4. This may be due to the lack of depth in the encoder/decoder layers but given that the reconstructions are still impressive, this is probably not the issue. It is more likely that the lack of global structure is exacerbated when the image dimensions are larger. It is also possible that the ratio between 2D prior size and image size increasing contributes to this.

Bonuses

This submission has a total bonus of -2 marks (a penalty) as it is trained only on CIFAR-10 (technically it was trained on STL-10 at full size too but this may not count given the degradation of the technique making the output meaningless).

References

[1] AntixK. *PyTorch VAE*. 2020. URL: https://github.com/AntixK/PyTorch-VAE.

[2] Google Deepmind. *Sonnet*. 2019. URL: https://github.com/deepmind/sonnet/blob/v2/sonnet/src/nets/vqvae.py.

[3] Rithesh Kumar. *Reproducing Neural Discrete Representation Learning*. 2018. URL: https://github.com/ritheshkumar95/pytorch-vqvae.

[4] Anders Larsen, Søren Sønderby, and Ole Winther. "Autoencoding beyond pixels using a learned similarity metric". In: (Dec. 2015).

[5] Misha Laskin. *Vector Quantized Variational Autoencoder*. 2019. URL: https://github.com/MishaLaskin/vqvae.

[6] Aaron Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel Recurrent Neural Networks". In: (Jan. 2016).

[7] Aaron Oord, Oriol Vinyals, and Koray Kavukcuoglu. "Neural Discrete Representation Learning". In: (Nov. 2017).

[8] Aaron Oord et al. "Conditional Image Generation with PixelCNN Decoders". In: (June 2016).

[9] Ali Razavi, Aaron Oord, and Oriol Vinyals. "Generating Diverse High-Fidelity Images with VQ-VAE-2". In: (June 2019).

[10] Casper Sønderby et al. "Ladder Variational Autoencoders". In: *Arxiv* (Feb. 2016).