Deep convolutional variational autoencoder to generate winged horses

Anonymous author

Abstract

Variational Autoencoders (VAE) are generative models that can learn efficient data representations and have highly interpretable latent spaces. I propose using a VAE in order to 'imagine' winged creatures not present in the data set, training using a differentiable perceptual similarity metric (SSIM) for loss over mean squared error. I evaluate these methods using the STL-10 dataset.

1 Introduction

How can we use generative models to create realistic hybrid animals not seen in the training data? Autoencoders [8] have found their use in a range of applications such as dimensionality reduction, image compression [1], denoising [5] and medical anomaly detection [4]. Autoencoders are composed of an encoder that transforms an input x to latent space z, and a decoder that attempts to transform z to an output x' where $x \approx x'$. Autoencoders are ill fitted for image generation as the latent space is highly irregular, and thus very difficult to sample from to produce realistic images. VAEs [7] solve this by imposing a structure on the latent space. I exploit this (detailed in section 3) to generate new images similar to that seen in the dataset.

Finding good metrics for differences between images is difficult, Mean Squared Error is often used however this will heavily penalize images that are not exactly correct to the pixel. Often, the rough structure of an image is more important than getting the pixels in the correct place.

2 Background

VAEs are in the family of generative models [2], which are able to learn the distribution of the observed data and generate samples from this. More precisely, if we have an observation x and target variable y, then a generative model can learn $p(x, y)$ or alternatively $p(x|y)$ in the case of conditional models, where y may be a label. This is opposed to discriminative models which usually learn $p(y|x)$, or a label given an image. Other generative models include GAN's, Normalizing Flows and Auto-regressive models such as RNN's.

VAE's give a large advantage over regular autoencoders in that their latent spaces aim to have two properties:

- **Continuity**: Vectors/samples from the latent space that are similar should result in similar outputs.
- **Completeness**: Points sampled from the latent space should give a reasonable output, not random noise but coherent structure that reflects the information in the training data.

These properties are very convenient for our purposes as it allows better interpolation resulting in a sensible transition between two images when we move between their respective points in the latent space.

VAE's [3] are directed probabilistic graphical models (See fig. 1 in [7]). They make the assumption that an image x is generated by a random process conditional on the latent z from a distribution $p'(z)$, where $x \sim p'(X|Z)$.

Finding the posterior $p'(Z|X = x)$ (the distribution of the latent variable) by Bayesian inference is intractable, so we attempt to approximate it. For this we introduce two PDFs, $q(Z|X, \theta)$ and $p(X|Z, \phi)$ which make a number of assumptions about p', q models the intractable posterior $p(Z|X)$.

We use two neural networks $e(x, \theta)$ and $d(z, \phi)$ to calculate p and q, where θ and ϕ are the parameters of our network (omitted in later sections). e is our encoder network which is used to estimate the posterior, which takes as input an image and calculates $q(Z|x, \theta) = \mathcal{N}(Z|e_\mu(x, \theta), e_\sigma(\theta, x)$. d is our decoder network, which takes a sample $z \sim p(Z|x)$ and calculates $p(X|z, \phi) = \mathcal{N}(Z|d(x, \phi), \mathbf{I})$. A normal distribution is often used here for simplicity.

In order to learn the parameters θ and ϕ, we need a clear objective function, which tries to both generate the images seen in the dataset, and regularize the latent space to be (in our case) a standard normal distribution. This means we will take an image x, feed it to our encoder e, sample from $q(Z, x)$ and give that to our decoder d which outputs the mean values for $p(X|Z = z)$. We want to get our estimate of the posterior $q(ZxX)$ to be as close as possible to the true posterior $p(Z|x)$, to do this we can use the Kullback–Leibler divergence, a measure between two continues distributions P and Q, it is defined as:

$$D_{KL}(Q||P) = \int_{-\infty}^{\infty} q(x) \log(\frac{q(x)}{p(x)} \tag{1}$$

We can use this and substitute in $q(Z|X)$ and $p(Z|X)$, then rearrange and apply Bayes' theorem to remove the intractable posterior:

$$\begin{aligned} D_{KL}(q(z|x)||p(z|x)) &= \int_{-\infty}^{\infty} q(z|x) \log(\frac{q(z|x)}{p(z|x)} \\ &= \int [q(z|x) - \log p(x|z) - \log p(z) + \log p(x) + \log q(z|x)] \end{aligned} \tag{2}$$

Then take out p(x) and rearrange again.

$$\begin{aligned} \log p(x) - D_{KL}(q(z|x)||p(z|x)) &= \int [q(z|x) - \log p(x|z) - \log p(x) + \log q(z|x)] \\ &= \mathcal{E}_{z\sim q(z|x)} \left[\log p(x|z) - \log \frac{q(z|x)}{p(z)} \right] \\ &= \mathcal{E}_{z\sim q(z|x)} \log p(x|z) - D_{KL}(q(z|x)||p(z)) \end{aligned} \tag{3}$$

We are now left on the right hand side with the Evidence Lower Bound (ELBO), which we want to maximise. We turn this into a minimization problem by negating the ELBO. We split the ELBO into the reconstruction loss $\mathcal{L}_{Rec} = -\mathbb{E}_{z\sim q(Z|x)} \log p(x|z)$, which is the likelihood of our input under p (from our decoder network) and the KL Divergence $D_{KL}(q(z|x)||p(z))$, which constrains the latent space to conform to a distribution p(z), usually $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

Finally, in order to estimate the KL divergence between our distribution $q(z|x)$ and $p(z)$ we use Monte Carlo sampling,and back-prop based on this:

$$\mathcal{L}_{KL} = \log q(z|x) - \log p(z) \approx D_{KL}(q(z|x)||p(z)) \tag{4}$$

In order to make the whole autoencoder differentiable with respect to the loss, we use the reparameterization trick. For the Gaussian case, we sample from a (usually multivariate) standard Gaussian distribution $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and then calculate $z = \mu + \epsilon * \sigma$. This simple operation allows us to back prop with respect to both the parameters of the decoder ϕ and through the latent 'bottleneck' with respect to the encoder parameters σ.

3 Methods

3.1 Architecture

I used a convolutional neural network (Figure 1) with kernel sizes 4x4, no padding and a stride of 2 for convolutions and deconvolutions. This is similar to [10]. Using a kernel size divisible by the stride helped prevent artifacts that were seen when using a kernel of 3x3. After each convolution a batch norm and leaky rely is applied to help with gradient flow. The CNN in the encoder was connected to two fully connected linear layers to output μ and $\log \sigma^2$, which hugely increased the parameter count of the model. In the decoder, a linear layer was inserted before the conv layers to interpret the latent space. Replacing the linear layers with convolutions negatively impacted performance. More complicated architectures were experimented with, such as Resnet but this gave no significant boost to performance and made training the model slow. I explored using different kinds of down/up sampling operations such as stride and max pooling. I found the differences between these to be minimal, but down/up stride was decided on due to its lower computational draw. I decided on a latent dimension of 1024 for stl-10 and 128 for cifar-10 as these gave the best results.

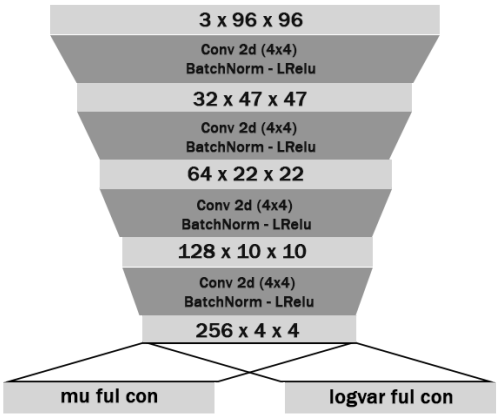


Figure 1: Encoder architecture optimized for stl-10, the fc layers output-sizes are the dimensions of the latent space. The decoder is almost identical, with deconvolution layers instead and a single fc layer.

3.2 Training

I trained on both cifar10 at 32x32 and STL-10 at 96x96 resolution. For STL-10, I first attempted to train using only labeled images of horses and birds and hoped to sample random to generate hybrid images. This failed as there are only 1000 images in these categories together, resulting in the network over-fitting, this led to very high quality reconstructions but 'fuzzy' and incoherent images when randomly sampling. Training on the full dataset gave much better results. I trained the network for six hours on a Colab GPU.

3.3 Exploring deep feature representation

Deep feature representation training [6] involves using a pre-trained network and comparing losses between layers for both the real and reconstructed version of an image, this ensures that larger details at different scales found by convolutions are consistent too, just not individual pixels. This can generally improve blurriness. As an alternative, I used the encoder network of the autoencoder and compared the loss of each layer between the output of the autoencoder $\hat{x}$ and the input image x and then back-propagated through the whole autoencoder using this. Doing this gave no significant change to image quality most likely due to the loss being only as good as the network, so i decided to omit it from training. I instead opted for

3.4 Using Perceptual Similarity Metrics

Structural similarity metric (SSIM) [11] is a useful improvement over conventional mean square error. It was designed to give a more realistic metric of similarity between two images that more closely resembles how humans view similarity. This was the metric $\mathcal{L}_{SSIM}$ that was used instead of standard MSE for reconstruction loss. It gave a lot of stability to training, allowing a higher learning rate and much better images, especially at larger resolutions such as STL-10.

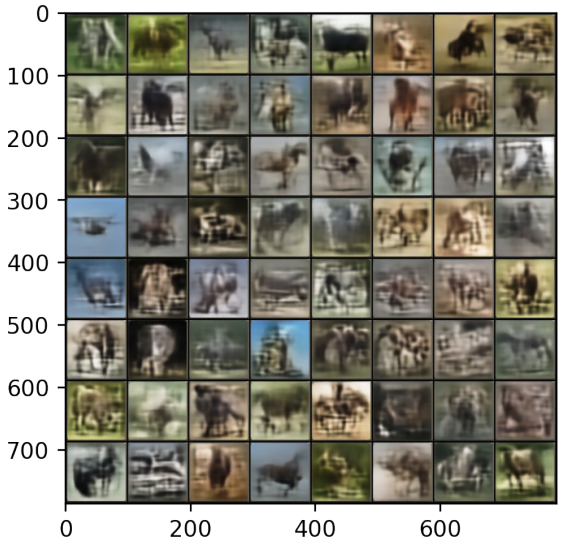
3.5 Color with Structural Similarity

The main drawback to using structural similarity is it (or the library used) under prioritized color, and often produced images that had a 'sepia' tone. Because of this, a custom fix was used which introduced a loss function that applied a large AvgPool to inputs $\hat{x}$ and x of, resulting in a 3x3 image and took the mean squared error of each color channel. This improved both the color accuracy of the reconstructions, and color diversity of the images generated with no artifacts. This gave a final loss function:

$$Loss = \mathcal{L}_{KL} + \mathcal{L}_{SSIM} + \mathcal{L}_{color} \tag{5}$$

4 Results

The best batch of images is shown below. These are result from the 96x96 stl-10, produced by interpolating between images.



From this batch, the most Pegasus-like image is:

5 Conclusion and further improvements

Although the generated images were superior over a plain VAE, they still lacked sharpness and clear structure. To further improve this model, a hierarchical VAE could be used such as NVAE [12]. Alternatively, adversarial training could be used like VAEGAN [9], the author did attempt this but found training very difficult due to poor gradient flow in the model. Additionally, tighter control could be enforced on the latent space, to try and find variables which correspond to wings.

Bonuses

Expecting a bonus of +3 for training full res on stl-10 and (possibly) white horse.

References

[1] David Alexandre et al. *An Autoencoder-based Learned Image Compressor: Description of Challenge Proposal by NCTU*. 2019. arXiv: 1902.07385 [cs.CV].

[2] Sam Bond-Taylor et al. *Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models*. 2021. arXiv: 2103.04922 [cs.LG].

[3] Carl Doersch. *Tutorial on Variational Autoencoders*. 2021. arXiv: 1606 . 05908 [stat.ML].

[4] Tharindu Fernando et al. "Deep Learning for Medical Anomaly Detection – A Survey". In: (2020). arXiv: 2012.02364 [cs.LG].

[5] Lovedeep Gondara. "Medical Image Denoising Using Convolutional Denoising Autoencoders". In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)* (Dec. 2016). DOI: 10.1109/icdmw.2016.0041. URL: http://dx.doi.org/10.1109/ICDMW.2016.0041.

[6] Xianxu Hou et al. *Deep Feature Consistent Variational Autoencoder*. 2016. arXiv: 1610.00291 [cs.CV].

[7] Diederik P Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: (2014). arXiv: 1312.6114 [stat.ML].

[8] Mark A Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243.

[9] Anders Boesen Lindbo Larsen et al. *Autoencoding beyond pixels using a learned similarity metric*. 2016. arXiv: 1512.09300 [cs.LG].

[10] Gustav Grund Pihlgren, Fredrik Sandin, and Marcus Liwicki. *Improving Image Autoencoder Embeddings with Perceptual Loss*. 2020. arXiv: 2001.03444 [cs.CV].

[11] Jake Snell et al. *Learning to Generate Images with Perceptual Similarity Metrics*. 2017. arXiv: 1511.06409 [cs.LG].

[12] Arash Vahdat and Jan Kautz. *NVAE: A Deep Hierarchical Variational Autoencoder*. 2021. arXiv: 2007.03898 [stat.ML].