Experimenting With Variational Auto Encoders

MohammadReza Davari

Contents

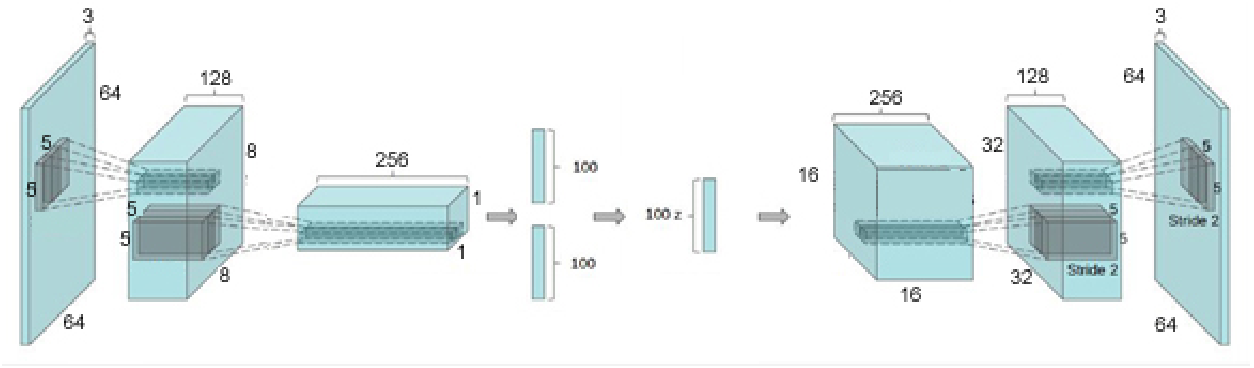

Figure 1: The symmetric VAE architecture inspired by the architecture of the generator of DCGAN [Radford et al., 2015]

Table 1: VAE Hyper-Parameters

Batch Size	**Epochs**
128	200

1 Model and Architecture

In this report we chose to work on and explore the world of **Variational Auto-Encoders (VAE)**. The architecture of our model is inspired by the architecture of the generator in DCGAN [Radford et al., 2015] in a symmetric way. An abstract visualization of our model can be found in the Figure [1].

We are training our model using the "celebA" data set. We used Google Colaboratory to train and learn the model. This made us unable to take advantage of the whole data set, since we could not upload the entire data set. As a result we are only training our model on $10,000$ data points with a validation set of 1000 data points. We have also tried to mimic the entire architecture introduced in the generator of DCGAN [Radford et al., 2015] but due to the limiting computational powers and memory space of Google Colaboratory we settled down for the model shown in the Figure [1].

We are using Adam optimizer with suggested learning rate of 0.0002 and the momentum term β_1 0f 0.5 [Radford et al., 2015]. Other hyper parameters of our model can be found in the Table [1].

2 Comparison of Methods to Increase The Feature Map Size

In this section we are going to compare the following schemes of increasing feature map size:

1. Deconvolution (transposed convolution) with padding and strides.

2. Nearest-Neighbor Up-sampling followed by regular convolution.
3. Bi-linear Up-sampling followed by regular convolution

Our comparison is based on the following two aspects:

1. Visual inspection of the difference between these schemes
2. Difference in terms of arithmetic of these operations.

2.1 Visual Inspection

In Figure [2] we can see the difference in our results for the the three models. In order to obtain these results the model has been trained once for each method of increasing feature map size. The final results shown here, clearly outline the following property of each method:

- Deconvolution: This method is adding a checkerboard effect to the output. This is due to the superimposition of the deconvolution layers. Once this superimposition happens some pixels get more overlappings than others and hence the checkerboard effect takes place. More details of what is happening in the deconvolution layer is explained in the next section [2.2].
- Nearest-Neighbor Up sampling: The outputs of the network using this method are much more pixelated and are sharper in comparison to the outputs of the other networks we are considering here. This is mainly due to how this method operates. In the Nearest-Neighbor Up-sampling, the values of the enlarged image are filled by the values of their closest neighbors (the same exact value is copied). This helps the image to be sharper overall and we can see the effect in Figure [2] part (b).
- Bi-linear Up-sampling: We can see the output of the network that used this method of up-sampling in Figure [2] part (c). The output is very smooth and blurred (for example the left eye is almost a black blob). The smoothness/blurred effect comes from the fact that the method performs an averaging operation (explained in more details in next section [2.2]) during the up-sampling to fill out the values of the new pixels. This averaging helps the network to attain smooth outputs as we can see in the figure.

Please note that to better observe the effects described above, you need to zoom in for a bit. We suggest areas such as forehead, chicks or the nose regions to be the possible candidates of observing the above effects on.

2.2 Arithmetic Difference

In terms of arithmetic approach, the Nearest-Neighbor and Bi-linear method are much more similar than the Deconvolution method. In the Deconvolution, to increase the feature map size to our desired size, we insert zeros around and in between the feature map through different values of padding and stride respectively. We then produce the output by following the steps below:

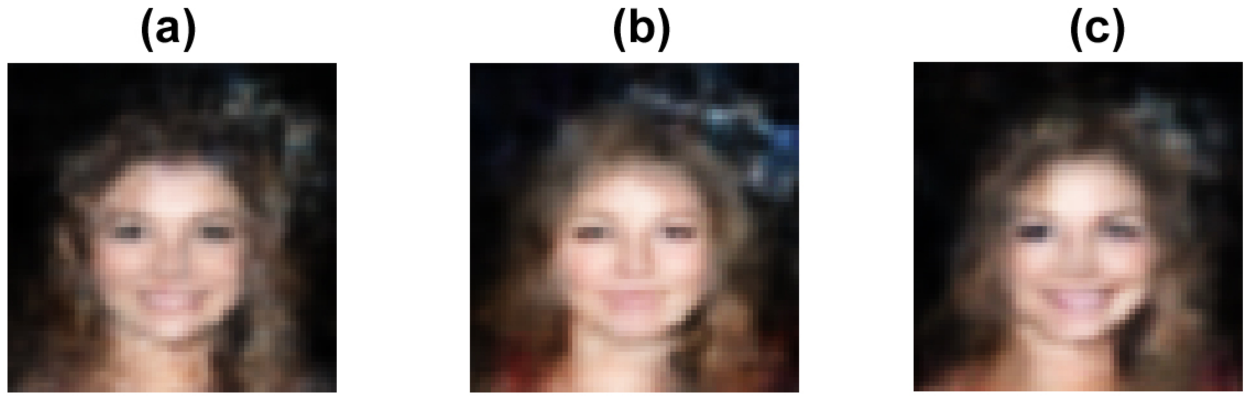

Figure 2: (a): Deconvolution (transposed convolution) with padding and strides. (b): Nearest-Neighbor Up-sampling followed by regular convolution. (c): Bi-linear Up sampling followed by regular convolution

1. Multiply the Deconvolution matrix by an element of the feature map we wish to increase its size. We can think of this step as scaling the Deconvolution matrix.
2. Then insert the values of the above operation into the output map.
3. Slide the Deconvolution matrix onto a new pixel (either to right or to the begging of the next row)
4. Do step (1) and (2) but add the values to the existing values and repeat the process.

What is happening in the Deconvolution is that we are basically superimposing scaled (by the original feature map) versions of the Deconvolution matrix to achieve the output feature map.

In the case of Nearest-Neighbor and Bi-linear method, to achieve the desired size for the feature map we first start with an output of zeros and then change the values. This can also be seen as inserting zeros around and in between pixels since as we will shortly explain the values of certain pixels will not change in this process and hence it can be viewed as just inserting zeros in between and around them. This step is exactly like the step in the Deconvolution method. What comes next however, is the main difference.

We mentioned that in the Deconvolution approach the entries of the output feature map is filled by the superimposition of the scaled Deconvolution matrix. However, in the case of Nearest-Neighbor and Bilinear method these values are filled by the values of the closest pixels to their projection in the original feature map.
The Nearest-Neighbor method simply fills up the zeros by copying the value of the nearest neighbor of these pixels in the original feature map.
The bilinear method fills up these values by a weighted average of the 4 neighbors of these pixels. The weighting is done according to the distance (the closer a pixel is the higher the weight).

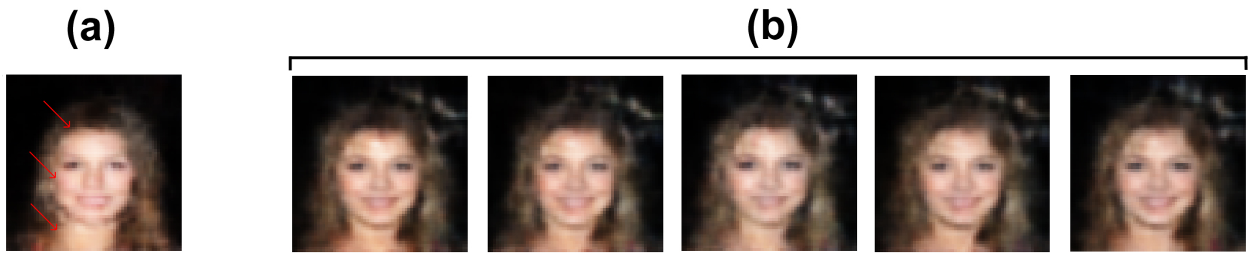

Figure 3: (a): Visual sample from the Vanilla VAE (b): Visual Sample of IWAE with 5 Monte Carlo samples

In terms of arithmetic, as explained above, the Nearest-Neighbor and Bilinear method are much alike, differing only on how they use the values of their neighbors to fill up the empty places. However, we can also look at the Deconvolution as doing a similar type of approach, if we focus on the use of neighbor notion of the operation. In that case Deconvolution can be viewed as a weighted sum of its neighbors but with the weights dictated by the Deconvolution matrix.
Hence we can say the following about these methods:

- Deconvolution: Fills out the gaps by the weighted sum of the neighboring pixels, with the weights dictated by the Deconvolution matrix.
- Nearest-Neighbor: Fills out the gaps by the value of the closest neighbor.
- Bilinear: Fills out the gaps by the weighted sum of the 4 neighboring pixels, with the weights proportional to the distances of the neighbors.

3 Qualitative Evaluations

3.1 Vanilla-VAE vs IWAE

To make the comparisons in this section we are using the "Deconvolution" approach to increase the size of the feature map. The comparison here is between our Vanilla VAE and IWAE with Monte Carlo samples of 5. A visual sample of these 2 models are presented in Figure [3]. In a first glance we can notice that the result of IWAE is smoother in comparison to Vanilla VAE. This quality makes the transition from a region to another region of the picture more natural. In Figure [3] we have pointed 3 red arrows indicating the rigidness of Vanilla VAE in transition in comparison to the IWAE samples. The reason behind this behavior lies in the approach taken in these 2 models. IWAE samples multiple times (5 times in this case) from the latent space to compute an approximation of the log likelihood, however Vanilla VAE only samples once from the latent space to approximate the log likelihood. This results in a tighter approximation of log likelihood for IWAE and hence produces a better loss function to optimize (since optimizing this tighter bound of log likelihood approximation gets the model closer to the optimal value of the real loss). This leads to the visual behaviors we are observing here.

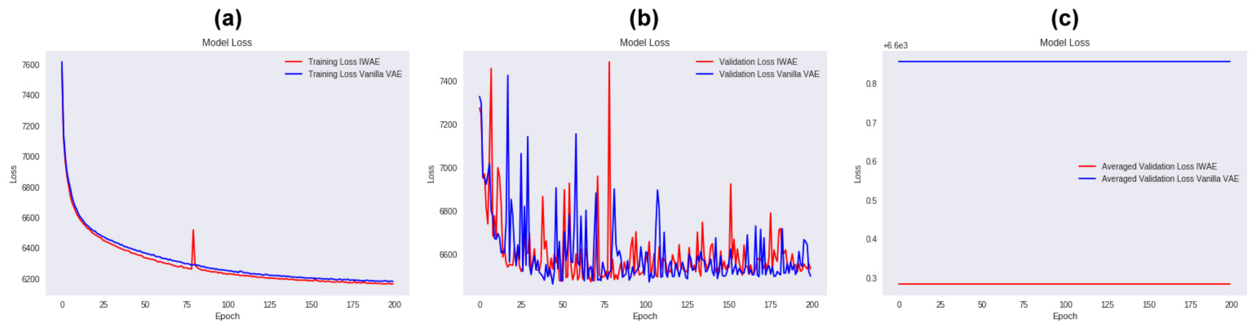

Figure 4: (a): Training error comparison of Vanilla-VAE and IWAE (b): Validation error comparison of Vanilla-VAE and IWAE (c): Averaged validation error comparison of Vanilla-VAE and IWAE.

We mentioned that the IWAE has a tighter approximation bound on the true loss and hence as the training progresses it provides a better optimization on the loss, which leads to having a better behaved model in comparison to Vanilla-VAE and hence, observing higher quality samples. Figure [4] demonstrates the loss under these 2 models. As shown in part (a) of the figure the training loss of IWAE is marginally less than Vanilla-VAE all the way through the 200 epochs of training as expected. Part (b) of the figure is trying to compare the validation error of the 2 model however, since the validation error is quite jumpy it is not easy to compare them, we averaged all the values of error over 200 epochs and showed the result in part (c) of the figure. It clearly indicates a lower average validation error for IWAE. These figures confirm what we said about the effect of the number of Monte Carlo sample estimates on the loss. This is also the main reason behind the improvement in quality of the images when sampling from IWAE model.[Burda et al., 2015]

3.2 Result of Changes In The Latent Space

In this section we will make changes in the latent space (for z, on a single dimension) and observe any possible/notable changes. Although we inflicted change in different dimensions of the latent space, here we are only presenting 3 of our results. In Figure [5] we can see what changing a parameter in the latent space could result into. In part (a) of the figure we can see that the change in the latent space has resulted into a gradual change in gender, going from a female to a male with facial hairs. In part (b) we can see these changes has resulted in changes related to hair length and face orientation. The hair length is gradually decreasing and the face is slowly truing to the other way. In part (c) we can see the changes we made in the latent space has resulted in a slight change in the orientation of the face of the subject. This is happening while the subject of the picture is almost the same person (the skin tone turns a little darker than the original picture) so in a sense this dimension is responsible to turn the face around.

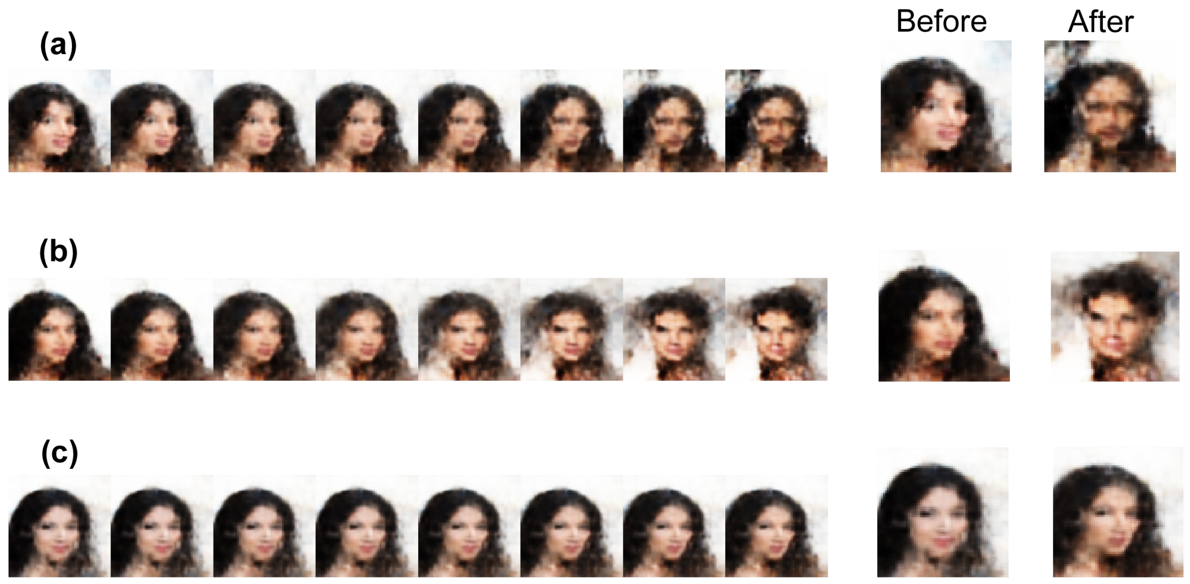

Figure 5: (a): Change in the latent space resulted in a change in gender going from a woman to a man with facial hairs (b): Change in the latent space resulted in change in length of hair going from long hairs to short hair plus a change in the direction of the head (c): Change in the latent space resulted in change in the direction of the head and keeping the subject of the picture almost the same (minor differences are skin tone being darker)

3.3 Interpolation Between Images

In this section we will investigate the result of interpolation between 2 images in the following schemes:

1. Interpolating between 2 randomly picked points z_0 and z_1 in the latent space computed by $z' = \alpha z_0 + (1-\alpha)z_1$ for $\alpha \in \{0, 0.1, 0.2, \dots, 1\}$
2. Interpolating between 2 sample points x_0 and x_1 corresponding to z_0 and z_1 using $x' = \alpha x_0 + (1-\alpha)x_1$ with α defined as in the above case

The result of these 2 interpolations are shown in Figure [6] with the label (a) and (b) corresponding to the first and second method of interpolation respectively.

The difference between these 2 method is that, in the first method we are interpolating between the images in the data manifold and hence, every intermediate step of this interpolation corresponds to an image on where the data lives. In the second method, we are interpolating by averaging the 2 images (a weighted average), which results in a superposition of one image over the other.

The first approach in interpolation produces smooth and meaningful intermediate images whereas the second approach does not necessary do well on these 2 aspects. In Figure [6] we

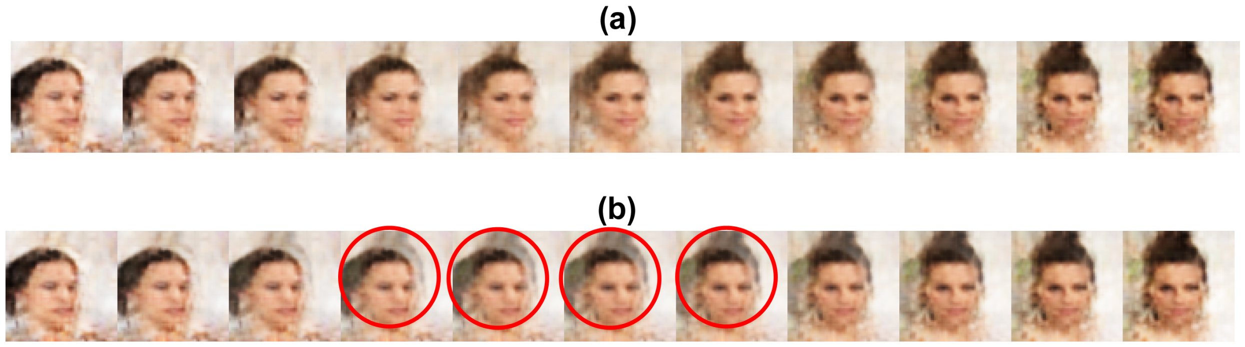

Figure 6: (a): Interpolation result between 2 randomly picked points z_0 and z_1 in the latent space computed by $z' = \alpha z_0 + (1-\alpha)z_1$ for $\alpha \in \{0, 0.1, 0.2, \dots, 1\}$. (b) Interpolation result between 2 sample points x_0 and x_1 corresponding to z_0 and z_1 using $x' = \alpha x_0 + (1-\alpha)x_1$ with α the same as before.

have circled with red one these areas. The first approach is styling the hair in a meaningful way, but the second image is producing a shadow artifact as emphasized by the red circles.

4 Quantitative Evaluations

In this part we are going to evaluate our model based on bits-per-pixel (bpp) using the following importance sampling estimation of the log likelihood with $K = 2000$:

$$\log p(x_i) \approx \log \sum_{k=1}^{K} \frac{p(x_i, z_k)}{q(z_k|x_i)}$$
$$z_k \sim q(z|x_i)$$

The main calculation for this importance sampling estimate is done in our code by using the function *nll*. Then by following the method in section 2.4 of [Danihelka et al., 2017] and changing the log function to a base 2 log function and dividing it by the number of pixels $(64 * 64)$ we will achieve the bits-per-pixel (bpp) for our model as:

$$\begin{aligned} \text{bbp} &= \frac{7133 \times \log_2(e)}{64 * 64} && (1) \\ &= 2.51 && (2) \end{aligned}$$

References

[Burda et al., 2015] Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

[Danihelka et al., 2017] Danihelka, I., Lakshminarayanan, B., Uria, B., Wierstra, D., and Dayan, P. (2017). Comparison of maximum likelihood and gan-based training of real nvps. *arXiv preprint arXiv:1705.05263*.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.