Course: Deep Learning **(19 December 2021)**

Submission Assignment #4A

Name: Bruna Aguiar Guedes, Enrico Calleris, Romy Vos *VUNetID:* bas630, ecs820, rvs249

1 Introduction

1.1 VAE

A Variational Auto Encoder (VAE) is a likelihood-based generative model from the group of latent variable models. As a main characteristic of the latent variables group, variables cannot be directly observed, but inferred from other variables, and the prescribed models are more specifically the ones in which the user presents in advance what kind of distributions are being dealt in this dataset.

The objective function of the VAE is a likelihood function lowerbounded that is constituted from a left part of a reconstruction error and the right part a Kullback-Leibler (KL) regularization that gives the distance between the variational posterior and the marginal.

$$\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\lambda(\mathbf{z})\right) \tag{1.1}$$

In practice, assuming a standard Gaussian prior we approximate expected values using a single sample

$$ELBO = \log \mathcal{N}(\mathbf{x} \mid \theta(\mathbf{z}), 1) - \left[\log \mathcal{N}\left(\mathbf{z} \mid \mu(\mathbf{x}), \sigma^2(\mathbf{x})\right) - \log \mathcal{N}(\mathbf{z} \mid 0, 1)\right] \tag{1.2}$$

where $\mathcal{N}(\mathbf{x} \mid \theta(\mathbf{z}), 1)$ is a Gaussian distribution for the Encoder, $\mathcal{N}\left(\mathbf{z} \mid \mu(\mathbf{x}), \sigma^2(\mathbf{x})\right)$ is the variational posterior with a Gaussian and finally the Gaussian prior.

We can also use different priors other than Gaussians to better estimate the density function, since Gaussians (although allow an easy and efficient calculation of derivatives for the probability distribution) may not reflect more complex distributions. Normalizing flow models can give a better distribution approximation. By applying a series (i.e chain) of invertible tranformations a simple distribution can evolve to a complex one, since we iteratively substitute the latent space variable z for a new one (defined in 1.3).

$$\begin{aligned} \mathbf{z}_{i-1} &\sim p_{i-1}\left(\mathbf{z}_{i-1}\right) \\ \mathbf{z}_i &= f_i\left(\mathbf{z}_{i-1}\right), \text{ thus } \mathbf{z}_{i-1} = f_i^{-1}\left(\mathbf{z}_i\right) \\ p_i\left(\mathbf{z}_i\right) &= p_{i-1}\left(f_i^{-1}\left(\mathbf{z}_i\right)\right) \mid \det \frac{df_i^{-1}}{d\mathbf{z}_i} \mid \end{aligned} \tag{1.3}$$

Now to be able to do the inference with the base distribution the equation should be converted into a function of z_i

$$\begin{aligned} p_i\left(\mathbf{z}_i\right) &= p_{i-1}\left(f_i^{-1}\left(\mathbf{z}_i\right)\right) \left|\det \frac{df_i^{-1}}{d\mathbf{z}_i}\right| \\ &= p_{i-1}\left(\mathbf{z}_{i-1}\right) \left|\det \left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)^{-1}\right| \\ &= p_{i-1}\left(\mathbf{z}_{i-1}\right) \left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right|^{-1} \\ \log p_i\left(\mathbf{z}_i\right) &= \log p_{i-1}\left(\mathbf{z}_{i-1}\right) - \log \left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right| \end{aligned} \tag{1.4}$$

Since we are dealing with a chain of probability density functions can can expand the equation of the output x to backpropagate until the initial distribution of the latent space (i.e z_0). For K transformations, we have that:

$$
\begin{aligned}
\mathbf{x} = \mathbf{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1 \left(\mathbf{z}_0\right) \\
\log p(\mathbf{x}) = \log \pi_K \left(\mathbf{z}_K\right) &= \log \pi_{K-1} \left(\mathbf{z}_{K-1}\right) - \log \left|\det \frac{df_K}{d\mathbf{z}_{K-1}}\right| \\
&= \log \pi_{K-2} \left(\mathbf{z}_{K-2}\right) - \log \left|\det \frac{df_{K-1}}{d\mathbf{z}_{K-2}}\right| - \log \left|\det \frac{df_K}{d\mathbf{z}_{K-1}}\right| \\
&= \ldots \\
&= \log \pi_0 \left(\mathbf{z}_0\right) - \sum_{i=1}^{K} \log \left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right|
\end{aligned}
\tag{1.5}
$$

where the full chain is formed by the successive distributions π_i.

A VAE consists of a stochastic encoder (a variational posterior over latent variables), and a stochastic decoder (likelihood function), and a marginal distribution over latent variables. The latter is a prior distribution which the dataset is assumed to have. It needs to be an appropriate choice for our data type, such as a Gaussian distribution for Real-valued data.

The flow of an VAE is as follows: input x (for this study an image) is encoded by the encoder and transformed into the variable z in the latent space. This transformation is a reduction from a high-dimensional space to a low-dimensional one, and the output of the encoder are parameters mean and co-variance (i.e μ and σ). This means it returns a distribution over the latent space).

The decoder reconstructs the input $\hat{x}$ trying to reduce the quantity of information lost in the encoding-decoding process by calculating a reconstruction error. However for the learning to occur,the input for the decoding should be sampled by using a reparametrization trick, which makes the gradient descent possible despite the sampling that occurs in the middle of the architecture (i.e in the latent space).Finally this error is backpropagated through the network to update the parameters. After training, model learns a distribution in the latent space and is now able to generate new data points to be decoded.

1.2 GAN

The generative latent variable models can also be designed with implicit models. We can use a flexible transformation of a distribution unknown a priori to generate an object. For that goal, it is not necessary to assume any form (thus an implicit distribution). As a consequence we can't use a likelihood-based approach as we have previously applied to VAE models. In the GAN we have three main active components (and a passive component which provides real inputs): the generator, discriminator and the adversarial loss. The latter is the learning objective which contains the learning of both Neural Networks:

$$
\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{real}}} \left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{1.6}
$$

This is the Binary Cross Entropy loss (BCE). The generator aims to minimize the adversarial loss, as its goal is to minimize the probability that the discriminator is correct. The discriminator on the other hand aims to maximize the loss, as its goal is to maximize the probability of being correct.

The generator's objective is to minimize the error compared to a real input x (i.e make it's own x the closest as possible to a real example). Meanwhile the discriminator's objective is to maximize the likelihood of correctly predicting (since it is a binary classification the discriminator's network should end with a sigmoid function that mimics probability).

The general dynamics of the model is sampling low-dimensional representations z. The Generator (also known as Fraud) takes some noise from the samples component creates new image (for our study). This will be fed together with real inputs into the Discriminator, which function is to classify the input as real or synthetically generated.

2 Methodology

2.1 Datasets

For the experiments done next, with the goal of implementing a simple VAE, a VAE with a more complex marginal and a GAN, two datasets are chosen: MNIST, which contains 60,000 instances of 28x28x1 images and SVHN, a 32x32x3 with 73257 digits for training (from which 65,000 were used as training set and the remaining as validation) and 26032 digits for testing.

2.2 VAE with Gaussian prior

As an initial experiment a VAE implementation is done, using a Gaussian prior to represent the data distribution. The Encoder is a Linear layer (784,300) activated with a ReLU function, followed by another Linear layer (300, 32). The Decoder that will receive the variables from the latent space start with a Linear layer (16,300) activated by a ReLU function and another Linear layer (300, 784) activated by a Sigmoid function. An ADAM optimizer is used with a learning rate of 0.005 and batch sizes of 64 images on 50 training epochs.

2.3 VAE with Flow-Based prior (RealNVP)

Following the Gaussian prior implementation, another VAE architecture was built, this time employing a Flow-Based prior (mimicking the RealNVP architecture). The Encoder consists of two Leaky-ReLU-activated linear layers (with shape (input_size, 300) and (300, 300) respectively), followed by a final linear layer with shape (300, 200). The Decoder consists of two Leaky-ReLU activated linear layers (with respective shape of (100, 300) and (300, 300)), followed by a sigmoid-activated linear linear of shape (300, input_size). The parameter input_size is defined by the size of images in the two datasets, which are (1, 32, 32) (resized) for the MNIST and (3, 32, 32) for the SVHN. The Flow-Based Prior is implemented by using a coupling layer, a permutation layer, and a number of flows equal to 3. Two Neural Networks for the single flow are implemented. The first NN has two Leaky-ReLU-activated linear layers (with shapes (50, 300) and (300, 300)), plus a tanh-activated linear layers of shape (300, 100). The second NN is identical in structure, with the only difference of the last layer not being activated but simply linear. An ADAM optimizer is used with a learning rate of 0.001 and batch sizes of 64 images on 50 training epochs.

2.4 GAN

The GAN was implemented such that it could accept an input of images with size (3, 32, 32). For the SVHN dataset, this means that it did not need resizing. The MNIST however was resized from (1, 28, 28) to (2, 32, 32). The Generator consists of three linear layers. The first two layers are followed by a Leaky ReLU layer. The final linear layer is followed by a Tanh activation function. The input layer has the size (128, 256) and the output layer (512, 3072). The Discriminator follows the same structure, except that Sigmoid is used as activation function. Here, the input is of size (3072, 512) and the output size is (256, 1). The network is trained on 50 epochs and optimized with the ADAM optimizer.

3 Results and Discussion

All architectures were trained on MNIST and SVHN datasets, using 55'000 training instances and 5'000 validation instances for MNIST and 65'000 training instances and 8'257 validation instanced for the SVHN.

3.1 VAE

3.1.1 MNIST

For the first experiment with a VAE with a Gaussian prior in the MNIST dataset, results are as seen in **??**. After 50 epochs of training, the loss functions achieves a 0.00849 for training set in the final epoch, and the validation reaches it's lowest loss in 0.00855. The decoded images can also be observed in Figure 2.

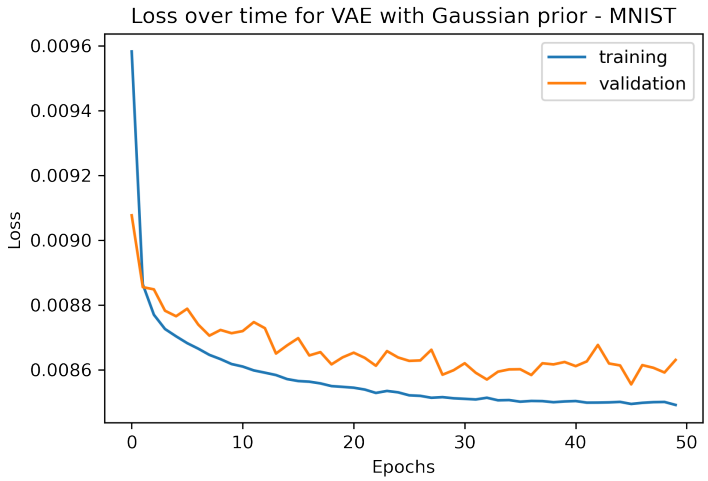


Figure 1: Loss over time for MNIST dataset fed into a VAE with Gaussian prior model.

(a) Epoch 1 (b) Epoch 25 (c) Epoch 50

Figure 2: Images of decoded output after different epochs for MNIST dataset applied to a VAE with Gaussian prior.

3.1.2 SVHN

The first experiment is also done for the SVHN dataset. in Figure 3 we see that it seems a much harder problem to minimize. After 50 epochs loss improves, however its lowest reaches only 1911.26 for training set and 1911.91 for validation. We can also observe that after 20 epochs most of the change in loss has already taken place, and that is reflected as well in the generation of the decoded images, observed in Figure 4. Here in epoch 0 the numbers from the images generated are almost indistinguishable, however on epoch 25 shades are more recognizable and not much visual improvement takes place from this epoch to the last.

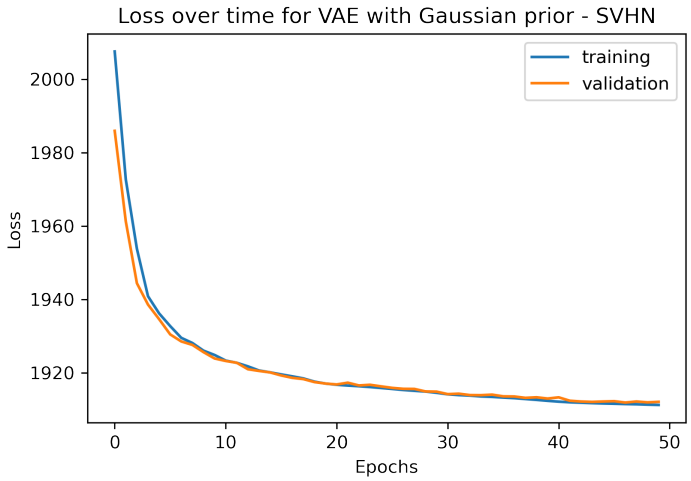
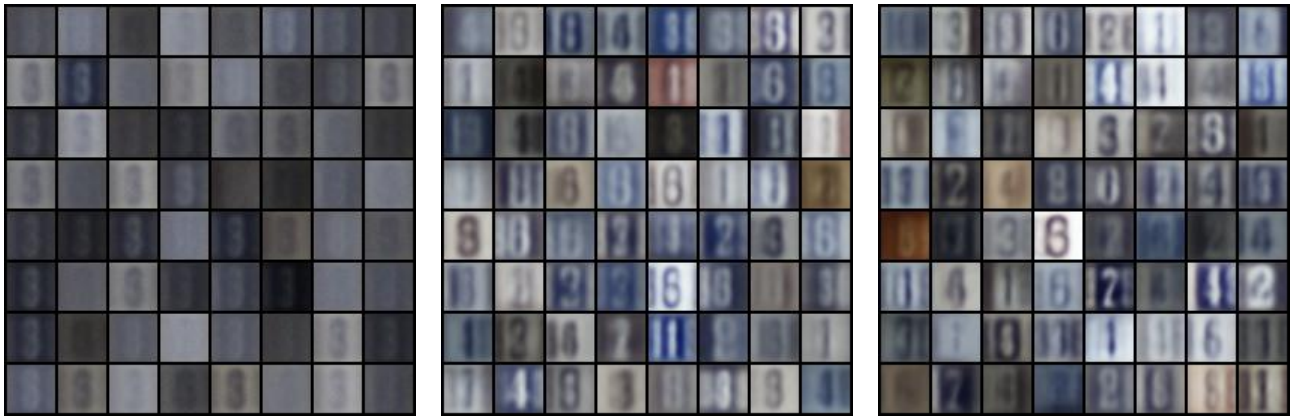


Figure 3: Loss over time for SVHN dataset fed into a VAE with Gaussian prior model.



(a) Epoch 0 (b) Epoch 25 (c) Epoch 50

Figure 4: Images of decoded output after different epochs for SVHN dataset applied to a VAE with Gaussian prior.

3.2 RealNVP

In the following section, results of experimentation with the RealNVP architecture are presented in the same fashion as before, for the MNIST and SVHN datasets.

3.2.1 MNIST

Figure 5 shows the training loss for the RealNVP. It can be observed that the loss has a much higher value compared to the Standard VAE, most likley due to its more complex architecture that needs a longer time to learn. Most of the training, however, seems to happen over the course of the first 10 epochs, with the curve showing a flattening after that point for both training and validation set. A problem that can be spotted is also the presence of a negative loss value. Figure 6 shows the generated images after 1, 25, and 50 epochs. The quality of generated images in the first iteration is not very high, with blurry lines being shown. The improvement that can be observed in Epoch 25 and 50 is substantial, but most numbers are still blurred or not completely understandable. The last two images reflect the comment made previously, not having a clear difference and therefore indicating the absence of an effective improvement during training. Most numbers appear unclear or blurry, making them difficult to discriminate also to the human eye.

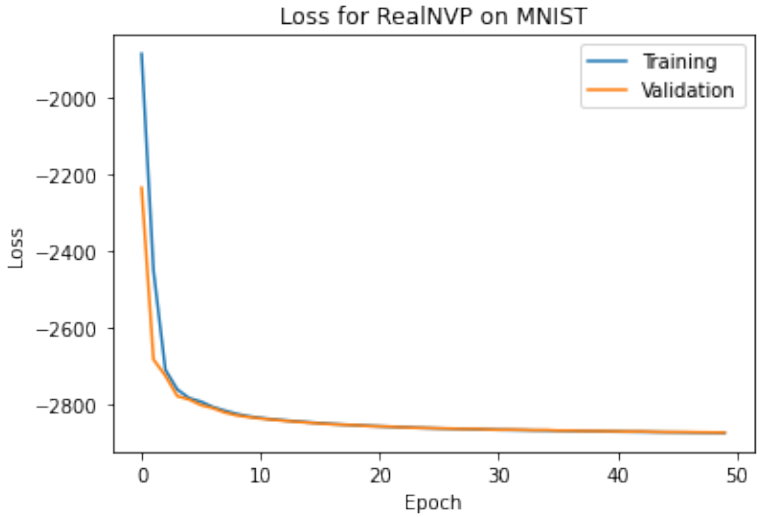


Figure 5: Loss over time for MNIST dataset fed into a VAE with Flow-Based prior.

(a) Epoch 1 (b) Epoch 25 (c) Epoch 50

Figure 6: Images of decoded output after different epochs for MNIST dataset applied to a VAE with Flow-Based prior.

3.2.2 SVHN

The same experiment was then performed on the SVHN dataset. Figure 7 shows the training loss, which present higher values compared to the MNIST plot presented in the previous section. This indicates a worse training quality, in addition to the observation that this time the convergence appears to be reached already after less than 5 epochs. No clear differences between training and validation set can be noted. The generated images are shown in Figure 8. Also in this case, the quality of the first epoch is very low, progressing to a slightly better resolution for Epoch 25 and 50. The latter generations present a more definite image, making it possible to spot some numbers, but the overall quality of the generated image remains low. This problem could be due to the nature of the SVHN dataset which, in comparison to the MNIST, has coloured images which might be more difficult to render, also with respect to the chosen arhitecture, which was not too complex due to computational and time constraints.

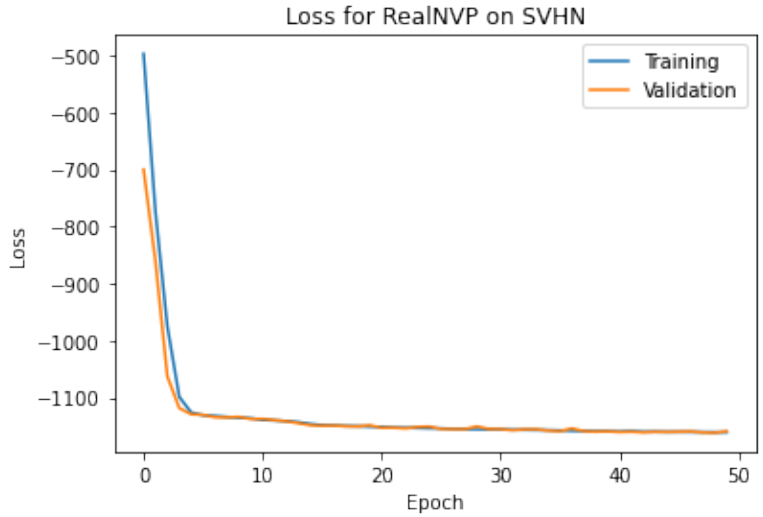


Figure 7: Loss over time for SVHN dataset fed into a VAE with Flow-Based prior.

(a) Epoch 1 (b) Epoch 25 (c) Epoch 50

Figure 8: Images of decoded output after different epochs for SVHN dataset applied to a VAE with Flow-Based prior.

Overall, the RealNVP architecture shows worse performances as compared to the VAE with Gaussian Prior. This fact, as mentioned previouslly, could be due to the more complex architecture of the former, which might need a more appropriate tuning (and has also been slightly simplified for the sake of this project). Despite this, an appreciable training process can still be observed, by seeing that images gets progressively more resoluted and start to approximate real numbers in the last iterations.

3.3 GAN

Finally, the experiments are conducted with the GAN architecture, again on the MNIST and SVHN datasets.

3.3.1 MNIST

Figure 9 shows the training and validation loss for the GAN network on the MNIST data. The plot shows that the generator improves over the number of epochs, while the loss of the discriminator decreases. Since the generator and the discriminator essentially compete against each other in the GAN network, it makes sense that an improvement of one leads to a setback for the other. Over time, they usually stabilize. That has not happened in this plot yet, but the relative low number of epochs (GANs are usually trained on more epochs) could be the reason for that.

The output images of the generator are shown in figure 12. The images are not improving over the number of epochs: they do not look like a MNIST input image at all. This is surprising, given that the loss of the

generator was decreasing. The reason for the output could be that the images needed to be resized before feeding the images to the network and resized again before plotting: here, the images did not correctly reshape back to their original shape. Also, the 3 channels caused the output to be in color (instead of the original black-and-white color of MNIST images).

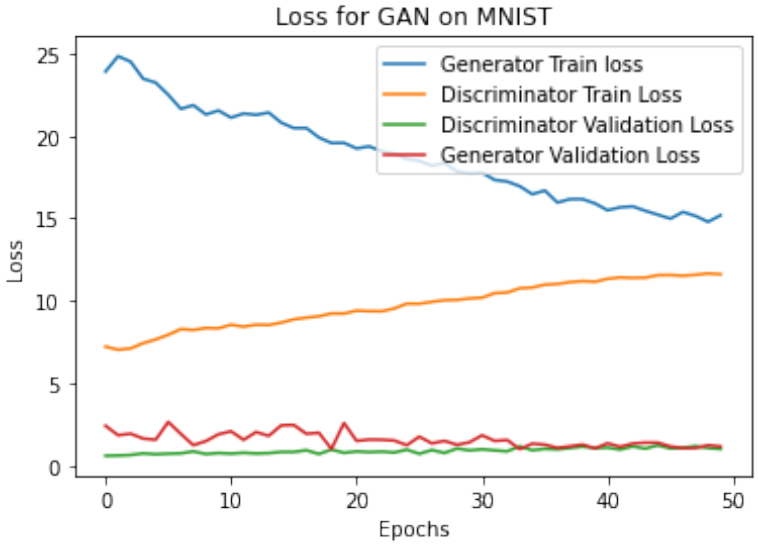
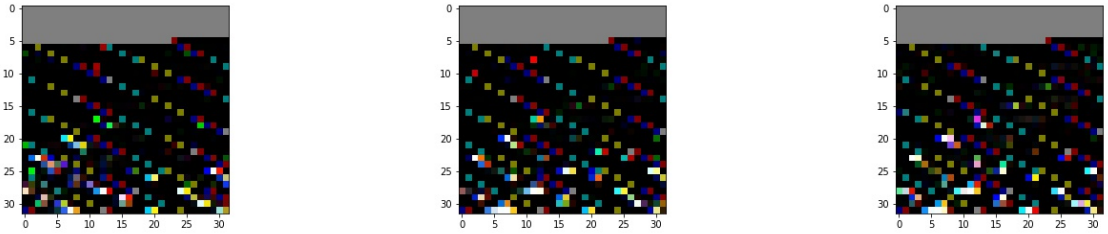


Figure 9: Loss over time for the GAN network on MNIST data.



(a) Epoch 1 (b) Epoch 25 (c) Epoch 50

Figure 10: Images of decoded output after different epochs for MNIST dataset applied to a GAN.

3.3.2 SVHN

The loss of the GAN network on the SVHN images is shown in figure 11. The loss of the discriminator increases and then stabilizes over the number of epochs, meaning that it does not improve at all. The generator loss on the other hand is low. In figure 12, three images are shown per epoch. The pictures again do not resemble the original SVHN data. The reason for this could be that the network architecture (linear layers) is not fitted for more complex data: in this case, an architecture with convolution layers such as DCGAN would have been a better choice, as these are a better fit for image data.

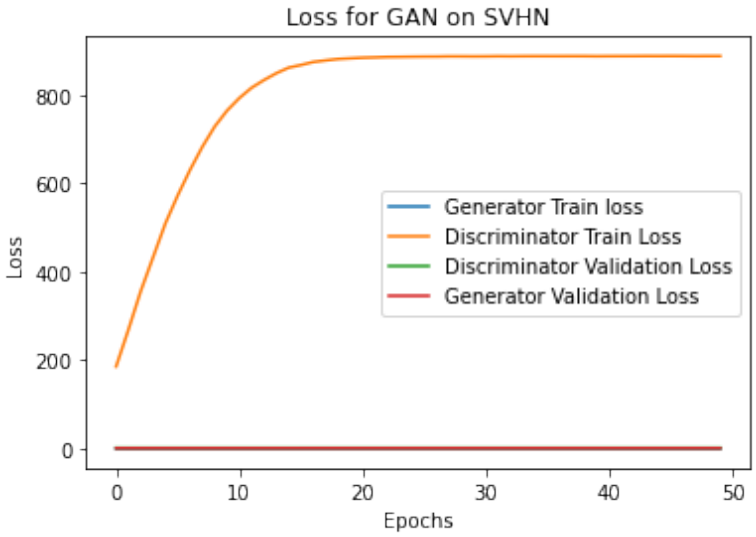
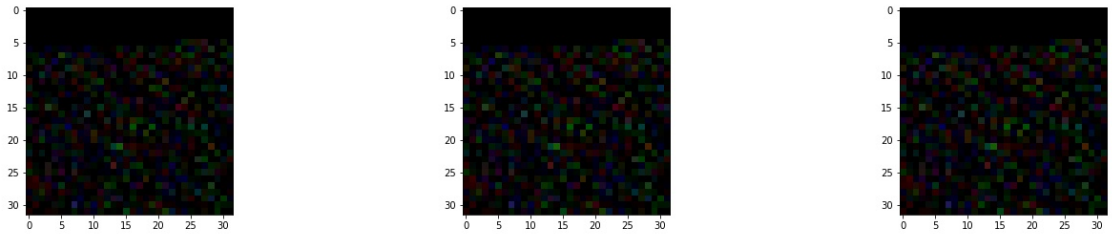


Figure 11: Loss over time for the GAN network on SVHN data.

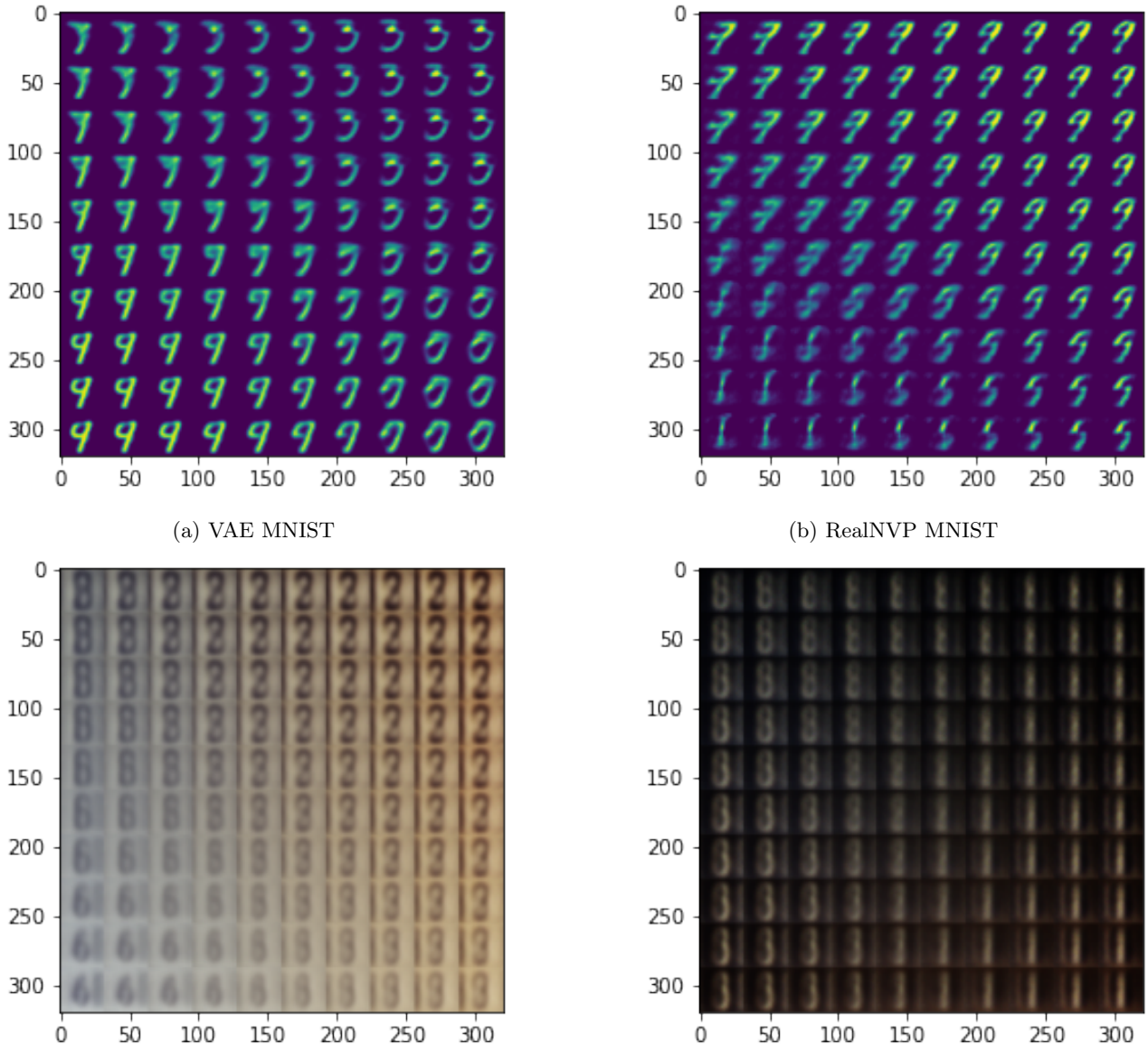


(a) Epoch 1 (b) Epoch 25 (c) Epoch 50

Figure 12: Images of decoded output after different epochs for SVHN dataset applied to a GAN.

3.4 Interpolation

Figure 13 shows the interpolated images for the 2 different architectures over the MNIST and SVHN datasets. Different generated numbers were sampled and then interpolated for 10 point total. It can be observed that the results for the VAE with Gaussian prior are satisfactory, showing clear numbers with a good morphing and clear distinction within them. The resolution of the SVHN is slightly lower than the MNIST, given the more complex nature of the dataset. The interpolated images for RealNVP are of lower quality, following the findings and discussion of the previous section. However, the level of of interpolation is still satisfactory for the MNIST dataset, by showing good morphing between generated numbers. The RealNVP on the SVHN dataset performs more poorly, also due to the lower resolution of generated images, although still showing some signs of morphing of digits.



(a) VAE MNIST

(b) RealNVP MNIST

(c) VAE SVHN

(d) RealNVP SVHN

Figure 13: Interpolated Images from VAE with Gaussian prior and VAE with Flow-Based prior (RealNVP) on MNIST and SVHN datasets.