Deep Generative Models: A Study

Saurabh Dash, Nathan Miller and Nael Mizanur Rahman

Project summary

Generative Modelling is the method of learning data distributions in an unsupervised manner which could be used to generate new data-points. Leveraging recent advances in deep learning, generative modelling when carried out on images leads to extremely realistic looking synthetic examples. In this project, we study 3 frameworks that are popularly used for this purpose: Generative Adversarial Networks (GANs) (by Saurabh Dash) [3], Variational Auto-encoders (VAEs) [1](by Nael Mizanur Rahman) and Normalizing Flows (NFs) [5] (by Nathan Miller). We conduct experiments on MNIST/ CelebA datasets with the vanilla and modified models for each of the 3 frameworks to study and compare the outputs, strengths and weaknesses of each of the above frameworks.

1 Generative Adversarial Networks

1.1 Introduction

Generative Adversarial Networks (GANs) [3] consists of 2 Neural Networks playing a min-max game where the Generator (**G**) attempts to fool the discriminator (**D**) and the discriminator attempts to catch the synthetic generated samples.

GANs provide an elegant solution to model an unsupervised learning problem as a supervised learning problem. The discriminator is a neural network that takes an image and outputs a probability of the image being real. The generator is a neural network that takes a noise vector as an input and converts it into an image.

The discriminator attempts to increase the expected log likelihood of detecting a real example by maximizing $\mathbb{E}_{x\sim p_r(x)}[\log(D(x))]$ and increase the expected log likelihood of detecting a fake example by maximizing $\mathbb{E}_{x\sim p_g(x)}[\log(1-D(x))]$ while the generator tries to minimize the discriminator's probability of detecting a fake sample by minimizing $\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$. Ideally as the training progresses one expects both the networks to get better over time. Thus the GAN optimization can be formulated as:

$$\min_G \max_D L_{GAN}(G,D) = \mathbb{E}_{x\sim p_r(x)}[\log(D(x))] + \mathbb{E}_{x\sim p_g(x)}[\log(1-D(x))] \tag{1}$$

Once the training is over, new samples can be generated by sampling from $p_z(z)$ and passing it through the generator **G** ($\tilde{x} = G(z)$)

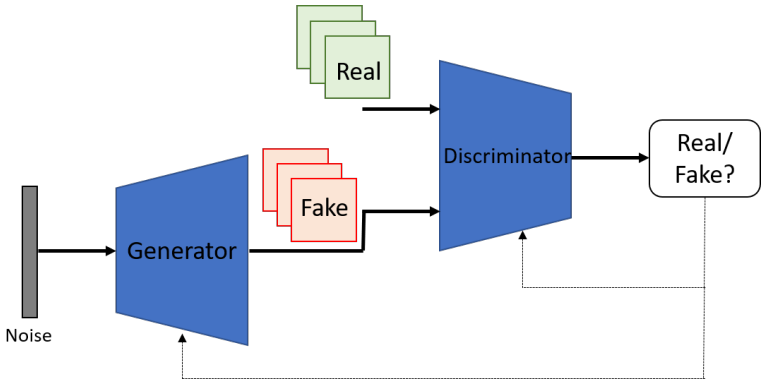


Figure 1: Architecture of GAN.

1.2 Implementation

We implemented[1] the network proposed by Radford et. al.[4] called DCGAN which implements both the Generator (G) and the Discriminator (D) as deep convolutional networks (5 convolutional and batch normalization

[1]The codes for this implementation can be found at https://github.com/saurabhdash/DeepGenerativeModels-GANandWGAN

layers). The authors present a number of techniques to improve the quality of generation:

- Instead of the standard ReLU non-linearity prevalent in image classification networks, the authors use Leaky ReLU activations in the discriminator which help in stabilizing the training of the GAN by allowing a small gradient to flow to the generator even when the output is negative.
- In the vanilla GAN loss function, once the discriminator becomes perfect, the gradient drops to 0 thus preventing any meaningful feedback to the generator. Instead of minimizing $\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$, the authors maximized $\mathbb{E}_{z\sim p_z(z)}[\log(D(G(z)))]$, thus overcoming the vanishing gradient problem.

The model was implemented on pytorch framework [9] and was trained on 2 NVIDIA GTX 1080 ti GPUs for 20 epochs. The optimizer used was ADAM [2]. As suggested by the authors, a gaussian prior was chosen to sample the noise input to the generator.

1.3 Results

Figures 2 and 3 show the progression of generated synthetic samples by the Generator network over the course of training. As we can see, in figures 2(a) and 3(a), initially the generator spits out gibberish but with proper feedback from the discriminator, the quality of images improves.

Figure 11 shows how if the noise vectors corresponding to different outputs are interpolated, there is a smooth transition between the outputs which also captures an intermediate output.

We wanted to see if the dimensions of the latent noise vector encoded human interpretable features. In our experiments, the noise vector had a size of 10 - in an effort to force the network to use few "descriptors" to generate an image. We swept each of the latent dimensions in the noise vector keeping the other dimensions fixed. In figure 12, each row shows the output when the corresponding is changed keeping the others constant. We see that some of the noise dimensions encode visual features like background color in figure 12(b) or number of loops in the digits in figure 12(a).

(a) epoch 1 (b) epoch 10 (c) epoch 20

Figure 2: Evolution of GAN generated MNIST samples as training progresses

1.4 Issues with GAN Training

It was observed that GANs are very sensitive to the choice of hyperparamters and the training in general is unstable. This is because we are trying to find a Nash equilibrium in a min-max game which might not always exist.

Another problem that was frequently encountered was Mode Collapse where the generator was stuck in one of the many modes of the real distribution and always generated the same result (explored further in the appendix). One of ways this was solved was by using one-sided label smoothening where the real samples were assigned a random value between 0.7 and 0.9 instead of 1. This prevented the discriminator from making overconfident predictions and quickly reaching the optimal value; allowing time for the generator to catch up, improving the quality of samples generated.

(a) epoch 1 (b) epoch 10 (c) epoch 20

Figure 3: Evolution of GAN generated CelebA samples as training progresses

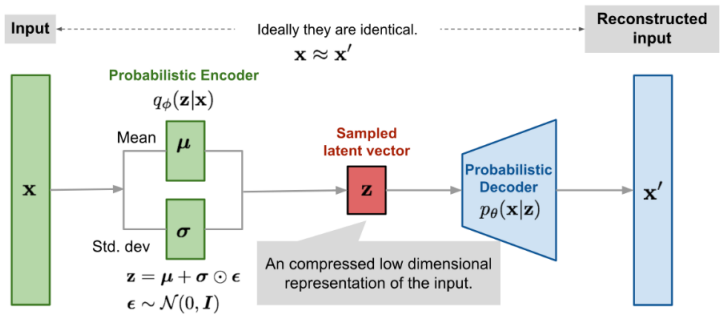


Figure 4: VAE architecture with a Gaussian prior

2 Variational Autoencoder

2.1 Introduction

An autoencoder is an unsupervised learning architecture that encodes(parametrized by(g_ϕ)) a high dimensional input into a low dimensional latent space, z and decodes (paramterized by f_θ) back into a high dimensional reconstruction of the input . ϕ and θ are learnt using Evidence Based Lower Bound (ELBO) loss function. New data is generated by sampling from the latent space z and feeding it to the decoder. Figure 5 shows a VAE architecture with a multivariate Gaussian prior on the encoder. A reparametrization methodology [1] is implemented which allows the parameters to be updated by back propagation of the gradient, thus making sampling of z trainable. In this process , the encoder determines a mean and standard deviation from the input data. A stochastic random normal variable (ε) is multiplied to each element of the std. dev. matrix (σ) which, along with the mean (μ) defines the Gaussian distribution of z.
The loss function is an *equally weighted* sum of Binary Cross Entropy loss (a reconstruction loss) and a Kullback-Liebler Divergence loss (a regularization loss) which reduces to the following for the Gaussian prior, for a dimensionality, J of the latent space z:

$$-L_{VAE} = E_{z\sim q_\phi(z|x)}[log\{p_\theta(x|z)\}] - D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = BCE_{loss} - \frac{1}{2}\sum_{j=0}^{J}(1 + log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

2.2 Implementation and Results

The VAE architecture proposed by [1] was implemented in a pytorch framework, on an NVIDIA GTX 1060 GPU. A ReLu activation function was incorporated into the encoder with a sigmoid activation function on the decoder.

(a) epoch 1 (b) epoch 10 (c) epoch 300

Figure 5: Evolution of VAE generated MNIST samples over 300 epochs

(a) epoch 1 (b) epoch 10 (c) epoch 30

Figure 6: Evolution of VAE generated CelebA samples over 30 epochs

An Adam optimizer with a learning rate of 1e-3 was employed. The generate samples of VAE implemented on the MNIST dataset for 300 epochs is shown in Fig. 5 showing how the generated samples become clearer over epochs. The VAE model implemented on the CelebA dataset [2] (trained over 30 epochs on a GTX 1080 ti GPU) is shown in Fig 6, showing faces generated by randomly sampling the latent space

2.3 Extension to β-VAE

$\beta - VAE$ differs from VAE in the loss function, which now incurs a penalty on the KLD loss component: $-L_{VAE} = BCE + \beta KLD$. This emphasises the disentanglement of the individual features (KLD term) at the cost of reduced reconstruction accuracy. Fig. 7 compares the β-VAE loss on MNIST for 3 values of β($\beta = 1$ is normal VAE). The loss is higher and generated samples are of poorer quality for higher values of β. Feature disentanglement is not fully obvious (possibly required further compression by the encoder into lower dimensional latent space) but some intermediate outputs can be seen. Fig. 8 shows a sweep across 2 of the 10 dimensions of the latent space (each column corresponds to one dimension) for $\beta = 10$ trained over 150K iterations, on the CelebA dataset, as presented in [8] [3]. The results could have been clearer if trained for higher iterations, but was not done due to resource constraints. While some latent features were not obvious, one was clearly identifiable as orientation of the face (row 1).

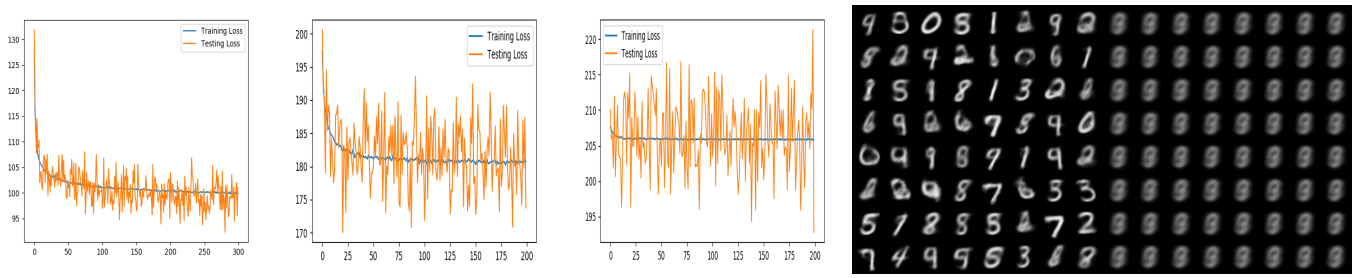
3 Normalizing Flows

3.1 Introduction

Normalizing flows is a method of density estimation which can be implemented in DNN architectures in order to explicitly learn the probability density function of the input data, $p(\mathbf{x})$. Normalizing flows involves applying a series of invertible transformation functions to the input data $\mathbf{x}$ which can then be inverted to generate new data, $\mathbf{x}'$. We specifically study RealNVP, a model which uses an affine scale and shift function to create the affine

[2] This implementation involved a slight modification of that found in https://github.com/podgorskiy/VAE

[3] The implementation is adapted from https://github.com/1Konny/Beta-VAE



(a) $\beta = 1$ (b) $\beta = 10$ (c) $\beta = 100$ (d) $\beta = 10$ (e) $\beta = 100$

Figure 7: β-VAE on MNIST: Loss Function and generated samples after 200 epochs

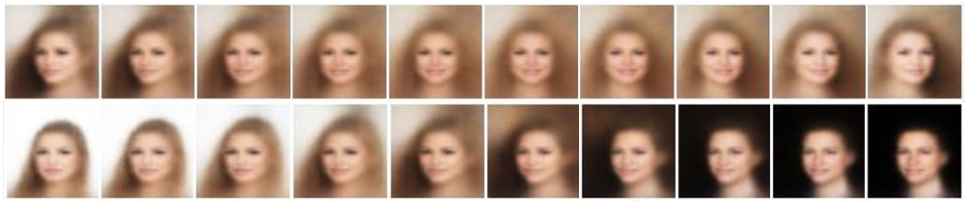


Figure 8: β-VAE on CelebA: Each row is a sweep across one dimension of the latent domain

coupling layer. Over the full dataset D, the loss function optimized by RealNVP is described as:

$$p(\mathbf{x}) = \pi(f^{-1}(\mathbf{x}))\left|det\frac{df^{-1}}{dx}\right|$$

$$L_{rnvp}(D) = -\frac{1}{|D|}\sum_{x\in D} log(p(\mathbf{x}))$$

3.2 RealNVP Implementation

The RealNVP architecture proposed by [6] produces an incredibly large network that is difficult to implement on most systems. The network uses a scale and shift transformation function as well as a ResNet architecture in the coupling layers. An Adam optimizer with learning rates varied between 1e-3 and 5e-3 was used for parameter optimization. For MNIST training, the architecture was implemented on an NVIDIA GTX 1060 GPU and trained for 100 epochs.[4] For CelebA training, the RealNVP architecure was implemented on two NVIDA GTX 1080 ti GPUs in parallel. Due to the massive nature of the model, the maximum batch size which could fit on these dual graphics cards was 32, which resulted in a training time of 45 minutes per epoch. Due to this time and memory constraint, the system was trained for 20 epochs.

3.3 Results

RealNVP takes a significant amount of training time for both MNIST and CelebA. As shown in Figure 13, the loss function produced by the training for both sets experiences massive fluctuation in the first 20-30 epochs, several orders of magnitude higher than the training loss. These tend to settle down after the initial 30 epochs, and it is hypothesized that further training would continue to settle down this highly volatile loss function. From a visual perspective, continued training on MNIST shown in Fig. 9 results in the generation of much more clear digits, most of which are fully recognizable by the 90th epoch. In the case of CelebA in Fig. 10, the images move from seemingly random colors in epoch 0 to the beginnings of recognizable facial features by epoch 15. The unique property of RealNVP that explains the way these results differ from GAN and VAE is that since RealNVP is based on a scale and shift transformation function, features which have not yet been trained well

[4]This implementation, configured for CelebA with options for MNIST implementation, can be found at https://github.com/nemiller95/rnvp and is a modification of the implementation found at https://github.com/chrischute/real-nvp

(a) Loss over 100 epochs (b) Epoch 0 (c) Epoch 10 (d) Epoch 90

Figure 9: RealNVP on MNIST: Generated samples over 100 epochs

(a) Loss over 20 epochs (b) Epoch 0 (c) Epoch 5 (d) Epoch 15

Figure 10: RealNVP on CelebA: Generated samples over 20 epochs

enough appear as distorted or misplaced (especially evident in the CelebA epoch 15 image) rather than blurry or discolored like the other models. As the model trains, the features of each generated image are moved and scaled into their proper position on the image and therefore clearer images are formed.

3.4 Issues with RealNVP Training

RealNVP encounters significant issues due to the sheer size of the model and the time it takes to train. Especially in the case of the CelebA study, the training time of 45 minutes per epoch with a maximum batch size of only 32 on a dual NVIDIA GTX 1080 ti system was prohibitive of further training and optimization. RealNVP testing loss was also seen to be highly volatile and difficult to optimize. Further training and optimization of learning rate, or potentially a different optimization method other than Adam, may have contributed to solving this issue.

References

[1] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114.

[2] Diederik P. Kingma et. al. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980.

[3] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661.

[4] Alec Radford et al. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. arXiv: 1511.06434.

[5] Danilo Jimenez Rezende et al. *Variational Inference with Normalizing Flows*. 2015. arXiv: 1505.05770.

[6] Laurent Dinh et al. *Density estimation using Real NVP*. 2016. arXiv: 1605.08803.

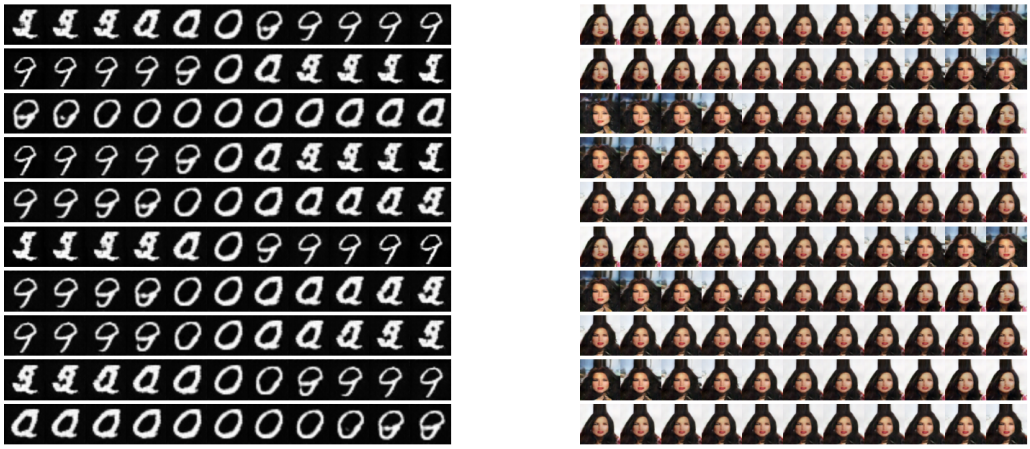
[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875.

[8] Irina Higgins et al. *beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*. 2017.

[9] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019.

A Appendix

Figure 11: Interpolation in the latent domain.



(a) MNIST (b) CelebA

Figure 12: Each row represents a sweep of one of the 10 latent values in the noise vector.

A.1 Wasserstein GAN (WGAN)

It has been observed that the support for the real distribution $\mathbb{P}_r$ actually lies in lower dimensions. This creates a major problem because it is very likely to find a perfect discriminator, leading to vanishing gradients. If the supports for real distribution $\mathbb{P}_r$ and generator distribution $\mathbb{P}_g$ do not align, the vanilla GAN loss which has been shown to minimize the Jensen-Shannon divergence becomes discontinuous [7]. This is where Wasserstein distance in superior. Wasserstein distance measures the distance between two probability measures and is a smooth measure. It is defined as:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \sim \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \tag{2}$$

$\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of all possible joint distributions between $\mathbb{P}_r$ and $\mathbb{P}_g$. We sample the joint distribution γ from this set that minimizes the expected distance between all possible points x and y over which $\mathbb{P}_r$ and $\mathbb{P}_g$ are jointly defined.

In this current from the expression in intractable. [7] used Kantorovich-Rubinstein duality obtain the equivalent expression:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq K} \mathbb{E}_{p_r}[f(x)] - \mathbb{E}_{p_g}[f(x)] \tag{3}$$

But the above equation requires that the discriminator function f be K-Lipschitz continuous. This adds an extra constraint on the parameters of the discriminator. [7] solved this by simply clamping the values of the parameters to restrict them in a range [-c,c] to enforce the K-Lipschitz constraint.

Instead of clipping, [**gulrajani2017improved**] showed that the optimal discriminator or critic f^* has gradient 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$. This could be added as an extra penalty term in the WGAN loss function leading to WGAN-GP (gradient penalty).

$$\mathscr{L}_{WGAN-GP} = \mathscr{L}_{WGAN} + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2 \tag{4}$$

(a) epoch 1 (b) epoch 10 (c) epoch 20

Figure 13: Evolution of WGAN-GP generated MNIST samples as training progresses

(a) epoch 1 (b) epoch 10 (c) epoch 20

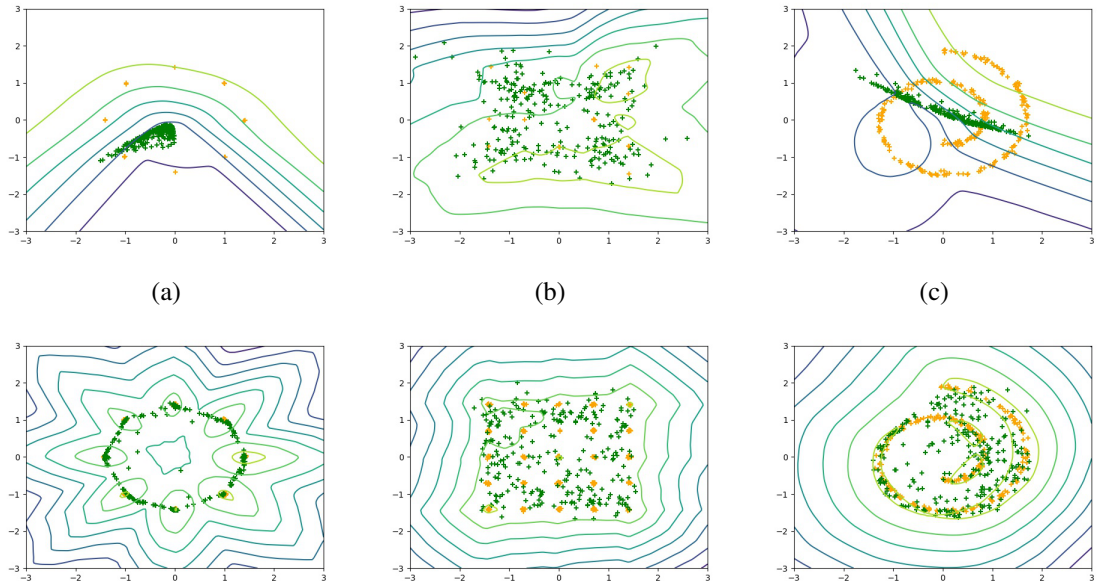
Figure 14: Evolution of WGAN-GP generated CelebA samples as training progresses

A.2 Results

Figure 15 illustrates how WGAN overcomes the limitations faced by GANs in multi-modal datasets. In figure 15(a) DCGANs collapse on a single mode while WGAN is able to learn the real distribution by generating data in all the modes.

A.3 β-VAE

Fig. 16 shows the reconstruction of the input samples to the encoder for β-VAE with different values of β. As expected, normal VAE provides the best reconstruction accuracy with the clearest picture while $\beta = 100$ proved to be too much to learn any feature of note. When implemented on the CelebA dataset, a sweep across all 10 dimensions of the latent space for $\beta = 10$ is shown in Fig.17, where each column represents a sweep across one particular feature. Trained over a higher dimension of the latent space, Fig. 18 shows a sweep across all 32 dimensions of z.



(a) (b) (c)

(d) (e) (f)

Figure 15: Actual distribution (orange) vs DCGAN generator distribution ((a)-(c)) and WGAN-GP generator distribution ((d)-(f))

(a) $\beta = 1$ (b) $\beta = 10$ (c) $\beta = 100$

Figure 16: Reconstruction of input images for β-VAE on MNIST after 200 epochs

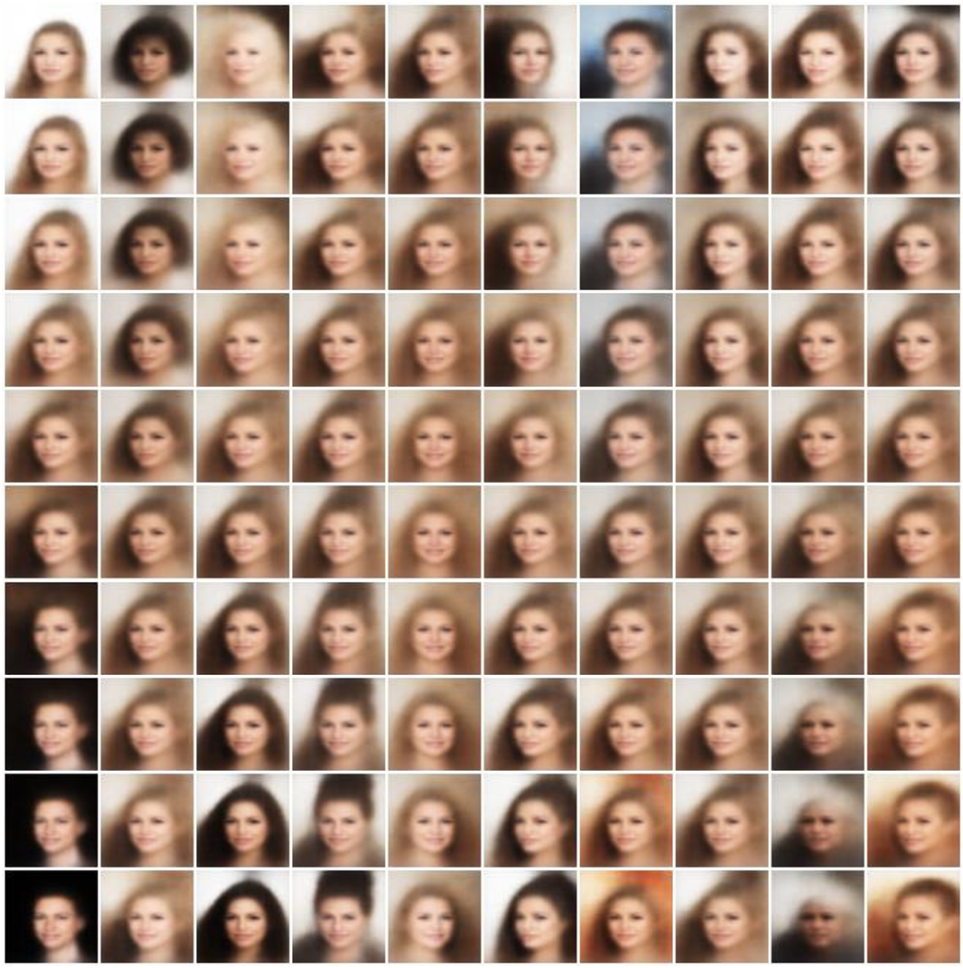


Figure 17: A sweep across all 10 dimensions of the latent space with $\beta = 10$

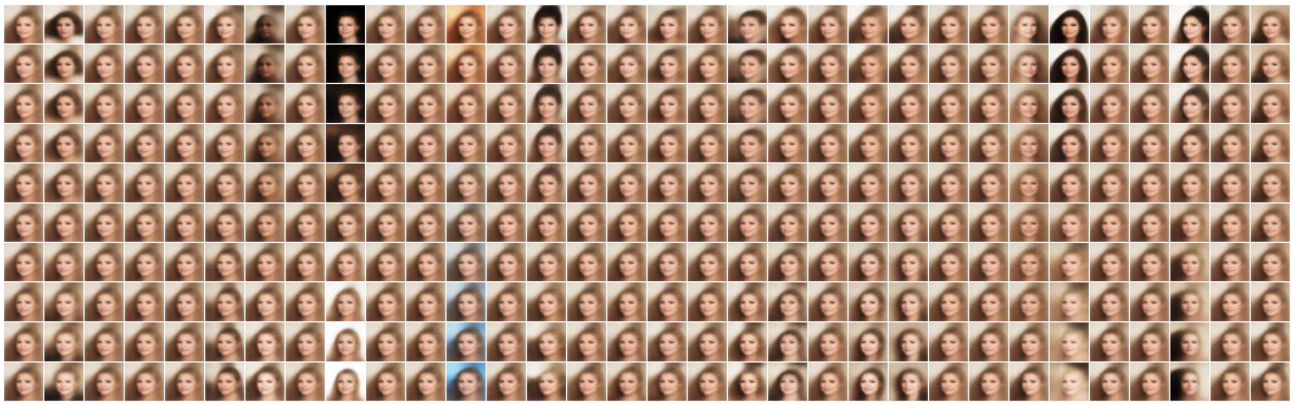


Figure 18: A sweep across all 32 dimensions of the latent space with $\beta = 10$