Introduction to Reinforcement Learning & Deep Learning

CSI 5340

Generative Models: *Using VAE, GAN, and WGAN to Generate MNIST and CIFAR10 Images*

Assignment 4

Authors
Alexandra Sklokin (300010511)

Professor
Runzhi Tian

November 29, 2022

Contents

1 Introduction

In this report, I will discuss the implementation of three generative machine learning models: Variational Autoencoder (VAE), Generative Adversarial Network (GAN), and Wasserstein GAN (WGAN). These models are used to generate images from the MNIST and CIFAR10 datasets, and to analyze their training behaviours. I will also explore the effect of model complexity and latent space size on image generation.

2 Datasets

2.1 MNIST

The Modified National Institute of Standards and Technology (MNIST) dataset used in this assignment contains handwritten digits (*0* to *9*). This dataset contained 60,000 training samples, and 10,000 testing samples. Each sample is sized $(1, 28, 28)$.

2.2 CIFAR10

The Canadian Institute For Advanced Research (CIFAR10) dataset used in this assignment contains images from ten classes (*airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*). This dataset contained 50,000 training samples, and 10,000 testing samples. Each sample is sized $(3, 32, 32)$.

3 Generative Models

3.1 Variational Autoencorder

A Variational Autoencoder (VAE) consists of an encoder and decoder, and can be evaluated using the loss function. The encoder is a Neural Network (NN) which maps the input images to a latent space, smaller than the space of the original data. The decoder NN maps from the latent space back to the input space, generating new images. Both NNs are trained at the same time using the loss function.

3.2 Generative Adversarial Network

A Generative Adversarial Network (GAN) consists of a generator NN and discriminator NN, and can be evaluated using the Jensen-Shannon Distance (JSD). The generator attempts to trick the discriminator by generating 'fake' images.

The discriminator is a binary classifier trained to discern between real and generated images. The models are trained back and forth, and the outputs of these models are sequentially fed into the other. The JSD is used to calculate the 'distance' or similarity between the original images and fake images. A common issue of GANs is model collapse, where the GAN only produces a single or small subset of similar samples. This is often caused by training loss oscillation or non-convergence.

3.3 Wasserstein Generative Adversarial Network

The Wasserstein Generative Adversarial Network (WGAN) consists of a generator and critic, and can be evaluated using the Earth Mover Distance (EMD). The WGAN training algorithm is almost identical to that of GAN, but provides a stronger learning signal to the generator NN. This model attempts to resolve the issue of model collapse.

4 Methodology

For this assignment I used *jupyter notebooks* and Excel spreadsheets. Please find all of the code for this assignment in my Git repository[1].

[1] https://github.com/alexandrasklokin/CSI5340/tree/main/CSI5340_A4

5 Experimental Results

5.1 VAE

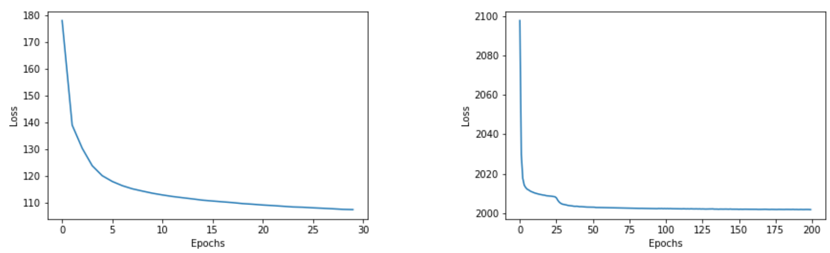
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1024]         803,840
            Linear-2                 [-1, 512]         524,800    Encoder
            Linear-3                 [-1, 256]         131,328
            Linear-4                   [-1, 5]           1,285
            Linear-5                   [-1, 5]           1,285
            Linear-6                 [-1, 256]           1,536
            Linear-7                 [-1, 512]         131,584    Decoder
            Linear-8                [-1, 1024]         525,312
            Linear-9                 [-1, 784]         803,600
================================================================
Total params: 2,924,570
Trainable params: 2,924,570
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.03
Params size (MB): 11.16
Estimated Total Size (MB): 11.19
----------------------------------------------------------------
```

(a) MNIST

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 16, 16]           3,136
            Conv2d-2            [-1, 128, 8, 8]         131,200    Encoder
            Linear-3                   [-1, 5]          10,245
            Linear-4                   [-1, 5]          10,245
            Linear-5                [-1, 2048]          12,288
   ConvTranspose2d-6           [-1, 64, 64, 64]           8,256    Decoder
   ConvTranspose2d-7         [-1, 3, 128, 128]           3,075
         MaxPool2d-8            [-1, 3, 32, 32]               0
================================================================
Total params: 178,445
Trainable params: 178,445
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.60
Params size (MB): 0.68
Estimated Total Size (MB): 3.29
----------------------------------------------------------------
```

(b) CIFAR10

Figure 1: VAE Model Architecture.



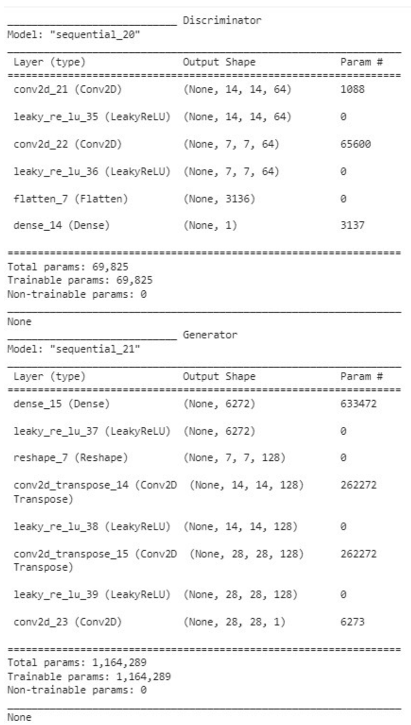
(a) MNIST (b) CIFAR10

Figure 2: VAE Model Training Loss over Epochs.

(a) MNIST (b) CIFAR10

Figure 3: VAE Model Samples of Generated Images.

5.2 GAN



Discriminator

Model: "sequential_20"

Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 14, 14, 64)	1088
leaky_re_lu_35 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_22 (Conv2D)	(None, 7, 7, 64)	65600
leaky_re_lu_36 (LeakyReLU)	(None, 7, 7, 64)	0
flatten_7 (Flatten)	(None, 3136)	0
dense_14 (Dense)	(None, 1)	3137

Total params: 69,825
Trainable params: 69,825
Non-trainable params: 0

None

Generator

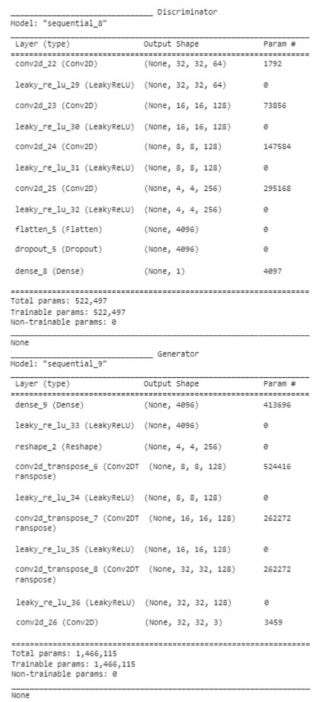
Model: "sequential_21"

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 6272)	633472
leaky_re_lu_37 (LeakyReLU)	(None, 6272)	0
reshape_7 (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose_14 (Conv2D Transpose)	(None, 14, 14, 128)	262272
leaky_re_lu_38 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_transpose_15 (Conv2D Transpose)	(None, 28, 28, 128)	262272
leaky_re_lu_39 (LeakyReLU)	(None, 28, 28, 128)	0
conv2d_23 (Conv2D)	(None, 28, 28, 1)	6273

Total params: 1,164,289
Trainable params: 1,164,289
Non-trainable params: 0

None

(a) MNIST



Discriminator

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 32, 32, 64)	1792
leaky_re_lu_29 (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_23 (Conv2D)	(None, 16, 16, 128)	73856
leaky_re_lu_30 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_24 (Conv2D)	(None, 8, 8, 128)	147584
leaky_re_lu_31 (LeakyReLU)	(None, 8, 8, 128)	0
conv2d_25 (Conv2D)	(None, 4, 4, 256)	295168
leaky_re_lu_32 (LeakyReLU)	(None, 4, 4, 256)	0
flatten_5 (Flatten)	(None, 4096)	0
dropout_5 (Dropout)	(None, 4096)	0
dense_8 (Dense)	(None, 1)	4097

Total params: 522,497
Trainable params: 522,497
Non-trainable params: 0

None

Generator

Model: "sequential_9"

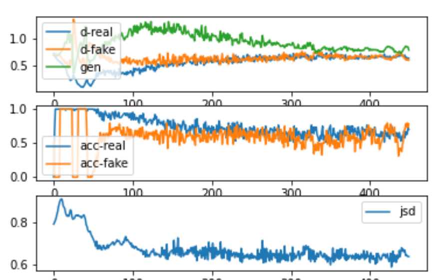
Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 4096)	413696
leaky_re_lu_33 (LeakyReLU)	(None, 4096)	0
reshape_2 (Reshape)	(None, 4, 4, 256)	0
conv2d_transpose_6 (Conv2DTranspose)	(None, 8, 8, 128)	524416
leaky_re_lu_34 (LeakyReLU)	(None, 8, 8, 128)	0
conv2d_transpose_7 (Conv2DTranspose)	(None, 16, 16, 128)	262272
leaky_re_lu_35 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_transpose_8 (Conv2DTranspose)	(None, 32, 32, 128)	262272
leaky_re_lu_36 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_26 (Conv2D)	(None, 32, 32, 3)	3459

Total params: 1,466,115
Trainable params: 1,466,115
Non-trainable params: 0

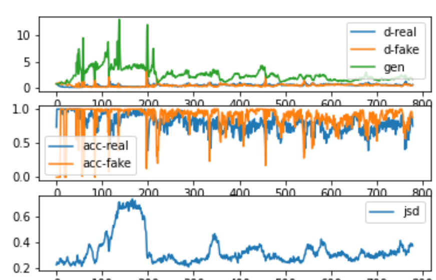
None

(b) CIFAR10

Figure 4: GAN Model Architecture.



(a) MNIST



(b) CIFAR10

Figure 5: GAN Generator and Discriminator Loss Functions, and JSD.

(a) MNIST

(b) CIFAR10

Figure 6: GAN Model Samples of Generated Images.

5.3 WGAN

Generator

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 6272)	319872
leaky_re_lu_2 (LeakyReLU)	(None, 6272)	0
reshape (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 128)	262272
batch_normalization_2 (BatchNormalization)	(None, 14, 14, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 128)	262272
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 128)	512
leaky_re_lu_4 (LeakyReLU)	(None, 28, 28, 128)	0
conv2d_2 (Conv2D)	(None, 28, 28, 1)	6273

Total params: 851,713
Trainable params: 851,201
Non-trainable params: 512

None

Critic

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1088
batch_normalization (BatchNormalization)	(None, 14, 14, 64)	256
leaky_re_lu (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 64)	65600
batch_normalization_1 (BatchNormalization)	(None, 7, 7, 64)	256
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 1)	3137

Total params: 70,337
Trainable params: 256
Non-trainable params: 70,081

None

(a) MNIST

Critic

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 16, 16]	3,136
LeakyReLU-2	[-1, 64, 16, 16]	0
Conv2d-3	[-1, 128, 8, 8]	131,200
BatchNorm2d-4	[-1, 128, 8, 8]	256
LeakyReLU-5	[-1, 128, 8, 8]	0
Conv2d-6	[-1, 256, 4, 4]	524,544
BatchNorm2d-7	[-1, 256, 4, 4]	512
LeakyReLU-8	[-1, 256, 4, 4]	0
Conv2d-9	[-1, 1, 1, 1]	4,097
Linear-10	[-1, 1, 1, 1]	2

Total params: 663,747
Trainable params: 663,747
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 0.53
Params size (MB): 2.53
Estimated Total Size (MB): 3.07

None

Generator

Layer (type)	Output Shape	Param #
ConvTranspose2d-1	[-1, 256, 7, 7]	409,856
BatchNorm2d-2	[-1, 256, 7, 7]	512
ReLU-3	[-1, 256, 7, 7]	0
ConvTranspose2d-4	[-1, 128, 14, 14]	524,416
BatchNorm2d-5	[-1, 128, 14, 14]	256
ReLU-6	[-1, 128, 14, 14]	0
ConvTranspose2d-7	[-1, 64, 28, 28]	131,136
BatchNorm2d-8	[-1, 64, 28, 28]	128
ReLU-9	[-1, 64, 28, 28]	0
ConvTranspose2d-10	[-1, 3, 56, 56]	3,075
Tanh-11	[-1, 3, 56, 56]	0

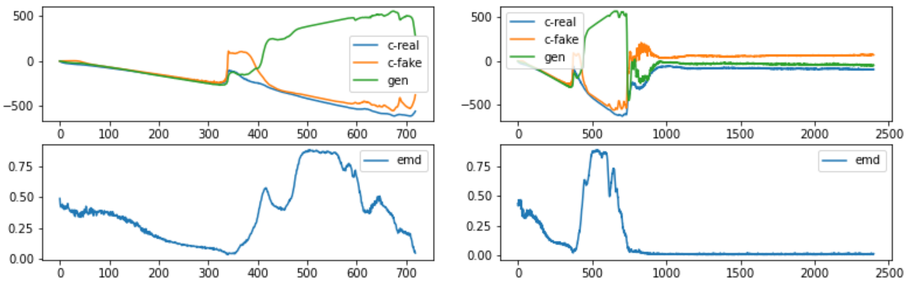
Total params: 1,069,379
Trainable params: 1,069,379
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 2.15
Params size (MB): 4.08
Estimated Total Size (MB): 6.24

None

(b) CIFAR10

Figure 7: WGAN Model Architecture.



(a) MNIST (b) CIFAR10

Figure 8: WGAN Generator and Critic Loss Functions, and EMD.

(a) MNIST (b) CIFAR10

Figure 9: WGAN Model Samples of Generated Images.

5.4 Model Complexity & Latent Space Size

(a) $layers = 2, latentSize = 2$

(b) $layers = 2, latentSize = 5$

(c) $layers = 2, latentSize = 10$

(d) $layers = 3, latentSize = 2$

(e) $layers = 3, latentSize = 5$

(f) $layers = 3, latentSize = 10$

(g) $layers = 4, latentSize = 2$

(h) $layers = 4, latentSize = 5$

(i) $layers = 4, latentSize = 10$

Figure 10: MNIST Samples for Different Model Complexity and Latent Size Settings *(*Given graphs produced using VAE models).*

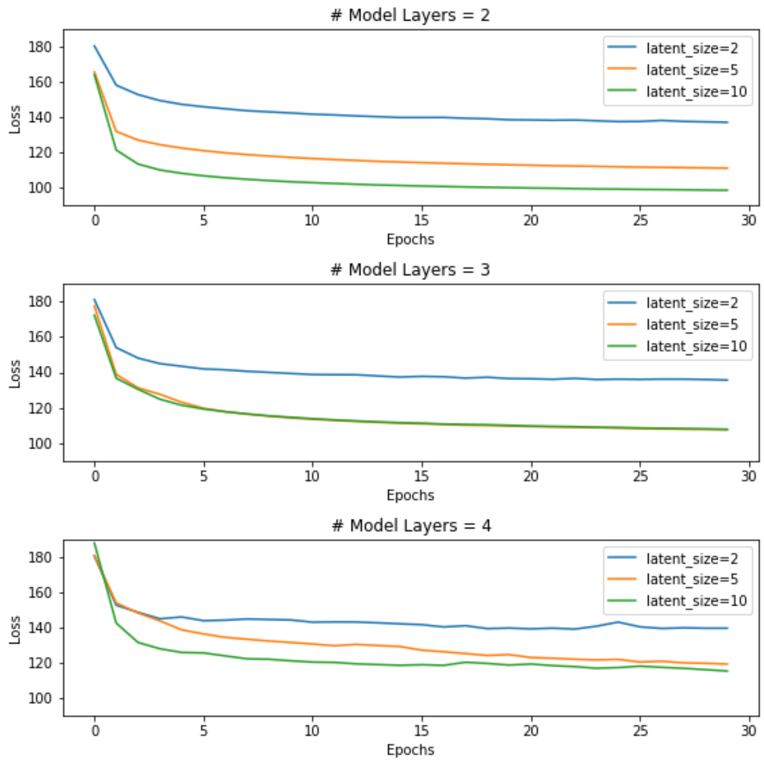


Figure 11: Training Behaviour for Different Model Complexity and Latent Size Settings *(*Given samples produced using VAE models).*

6 Discussion

In this assignment I was able to implement three types of generative Neural Networks (NNs) on the MNIST and CIFAR10 datasets. I will now discuss my investigation of the training behaviour and produced images for these models. I will also briefly discuss the effect of model complexity and latent space size on these generative models.

6.1 VAE

Figure 1 shows the model architectures used for the VAE model on both datasets. Through preliminary investigation, I determined that a simple VAE was sufficient for the MNIST dataset, and yet did not produce good images for the CIFAR10 dataset. For this reason, my CIFAR10 VAE architecture introduced convolution layers.

Figure 2 shows the loss function of the VAE models over training epochs. For the MNIST dataset, the VAE model quickly converges to a sufficiently low training loss, within 30 epochs. By contrast, despite an increased number of epochs, the proposed VAE model architecture did not perform well on the CIFAR10 dataset. The constraints of this assignment did not allow me to investigate whether the loss would continue to decreases with more epochs, however I believe it would not. I hypothesize that the CIFAR10 dataset is too complex for my model architecture. If I were to continue investigating this problem, I would try to increase model complexity by adding more layers to my network, and allowing for more training epochs.

Figure 3 shows some sample images produced from the respective VAE models. Clearly, the MNIST samples are diverse, human-recognizable, and undifferentiable from the original dataset. As expected, the generated CIFAR10 samples seem to be blurry and do not look similar to the original images in this dataset.

Thus, the VAE model is a very good generative model for the MNIST dataset, but is not suitable for the CIFAR10 dataset. It does not produce human-recognizable images in an acceptable amount of time.

6.2 GAN

Figure 4 shows the generator and discriminator architectures used for the GAN model on both datasets.

Figure 5 contains three graphs: (1) the discriminator/generator training loss; (2) the discriminator's classification accuracy for $'real'$ or $'fake'$ images; and (3) the JSD. For both the MNIST and CIFAR10 datasets, the generator's training loss converges and decreases, indicating a tendency to produce better images. Also, the predictive accuracy of the discriminator fluctuates, which is to be expected, but the variance tends to decrease. Finally, for both datasets, the JSD tends to decrease over training epochs, indicating that the 'distance' between original and generated images should be decreased. Clearly, training GANs is a difficult task, since we are simultaneously training a generator NN and discriminator NN. For this reason, we see a lot of fluctuation of the model's training behaviour, as both NNs 'battle'. In later epochs, the graphs experience less variance, which indicates a good overall trend.

Figure 6 shows some sample images produced from the respective GAN models. The MNIST samples indicate model collapse! This means the GAN model which was trained is only generating a subset of the available images- an '8'. The CIFAR10 samples look significantly better than with our VAE model, with the second image looking like a $'dog'$ or $'deer'$

Thus, the GAN model has been shown to experience model collapse for the MNIST dataset. It outperforms the VAE model on the CIFAR10 dataset, where the images are much more discernible than before. Maybe we can do better...

6.3 WGAN

Figure 7 shows the generator and discriminator architectures used for the WGAN model on both datasets.

Figure 8 contains two graphs: (1) the critic/generator training loss; and (2) the EMD. Firstly, on the MNIST dataset, the generator's training loss and EMD decreases up until epoch 350, at which point it skyrockets. Interestingly, the discriminator's loss continued to decreases, indicating that it begins to 'win' over the generator. For this reason, I did not consider the final WGAN model, but rather saved the generator model produced at epoch 350. If we only look to that first half of the graphs, we see a steady decreases in EMD, indicating that the generated images are approaching the original images. Secondly, on the CIFAR10 dataset, we see some strange training behaviour at the beginning, and then the model loss and EMD converge.

Figure 9 shows some samples images produced from the respective WGAN models. Clearly, the MNIST samples are diverse, human-recognizable, and undifferentiable from the original dataset. Also, here the CIFAR10 images start to look similar to the original images in the dataset. It is possible that more training epochs would improve the images further.

Thus, the WGAN model is a very good generative model for the MNIST dataset. Although the CIFAR10 images generated by the WGAN are not quite recognizable, they do appear to approach the original images better than the VAE model.

6.4 Model Complexity & Latent Space Size

When training the previous three generative models on the two datasets, I was able to investigate the effect of model complexity and latent space size on the training behaviour. Originally, I hypothesized that increasing model complexity and increasing latent space size would usually generate better images and decrease the loss/JSD/EMD.

Firstly, I noticed that the choice of model complexity and latent space size was dependent on the dataset. As mentioned previously, the CIFAR10 dataset, which is much larger than the MNIST dataset, usually required a more complex model architecture. To increase model complexity I introducing more layers and variability of layers into the model (ex. convolutional layers). Also, CIFAR10 usually benefited from a bigger latent space size, since the original dataset is larger too. Additionally, when training the GAN model, I found that a small latent space led to model collapse!

I will now describe one specific setting (VAE with MNIST) which can be used to better highlight the effect of model complexity and latent space size on the generative model.

Figure 10 shows a matrix of samples produced by different values of model complexity and latent space size. Unfortunately, it is not immediately clear which model produces the best images, although the number of blurry digits seems to decrease as model complexity and latent space size increase.

Figure 11 clearly shows that as latent space size increases, the model training loss converges to lower values- this would indicate better generated images. Here, model complexity does not play such an important role as latent space size. Also, the third graph may show some signs of overfitting, meaning that we must be prudent when model complexity as to avoid bad generalization.

Thus, I have found that increasing model complexity and latent space size will decrease model loss/JSD/EMD (up to the point of overfitting).

7 Conclusion

I was able to (mostly) successfully utilize the Variational Autoencoder (VAE), Generative Adversarial Network (GAN), and Wasserstein-GAN (WGAN) for the purpose of image generation. I found that VAE and WGAN performed sufficiently well on the MNIST dataset, however the GAN model experienced model collapse. The CIFAR10 dataset, which is much larger, was best generated using the GAN and WGAN models.

I also investigated the effect of model complexity and latent space size on the training behaviours and found that increasing these two parameters usually produced better samples and decreased the training loss/JSD/EMD.

In conclusion, the unsupervised image generation problem is much trickier than the supervised classification problem. Yet, we are able to train Neural Network (NN) models on complex datasets to generate 'fake' images which could be mistaken, by a human, for the real ones.