Stabilization Techniques for the Training of Generative Adversarial Networks

Vishal Pani

Indian Institute of Information Technology, Allahabad, Uttar Pradesh 211015

Guided by:

Dr. Nikhil Ranjan Pal

Indian Statistical Institute, Kolkata, West Bengal 700108

Abstract

Generative modeling has always been a fascinating field in deep learning. Its use cases range from generating realistic faces from random noise to producing high-resolution images from images of lower-resolution. Currently, models which are at the forefront of generative modeling are called Generative Adversarial Networks (GANs). GANs learn by playing a minimax game between a Discriminator Network and a Generator Network. One of the drawbacks with training a GAN is that they are difficult to stabilize. The current state of the networks, at a given moment, is highly dependent on one another. Hence, a fluctuation in the error of the Discriminator may adversely affect the Generator and vice versa. To tackle this, we introduce a regularising factor to the cost function of the Discriminator. This technique enabled the Generator network to produce qualitatively comparable if not, better results than the base model. Another issue with GAN training is that initially, both the Discriminator and Generator are poor at doing their respective tasks. Therefore, the high interdependence between the networks hampers the initial training. To mitigate this issue, the Discriminator and Generator are pre-trained separately. The Generator is pre-trained in a stacked fashion, in order to retain as much spatial information as possible. A standard GAN training follows this phase. This method allowed the Generator to produce images with much better structural integrity when compared to the base model.

Keywords: Generative Adversarial Networks, Generator, Discriminator, Regularization

---

Acknowledgments

I would like to take this opportunity to sincerely thank my project guide, Dr. Nikhil Ranjan Pal for providing me an opportunity to work on my summer research internship under his guidance. I am deeply thankful for his continuous support and guidance without which this project would not have been possible. His inputs have been instrumental in shaping and improving this project since the start of the project.

I would also like to thank my fellow interns Mr. Sudhanshu Mishra, Mr. Abinash Dutta and Mr. Utkarsh Bairolia for being honest sounding boards for my ideas. Their inputs have also played and important role in shaping my project.

Abbreviations

GAN	Generative Adversarial Network
W-GAN	Wasserstein Generative Adversarial Network
LS-GAN	Least Squares Adversarial Network
CNN	Convolutional Neural Network

---

1 INTRODUCTION

There are several issues that one may face while training a GAN. One prevalent problem is the issue of Mode Collapse. Mode Collapse occurs when the Generator starts producing similar outputs for a variety of inputs. The reason for this is that

Discriminator might get good at classifying specific modes in the data distribution. The Generator, in turn, starts to only focus on generating those modes.

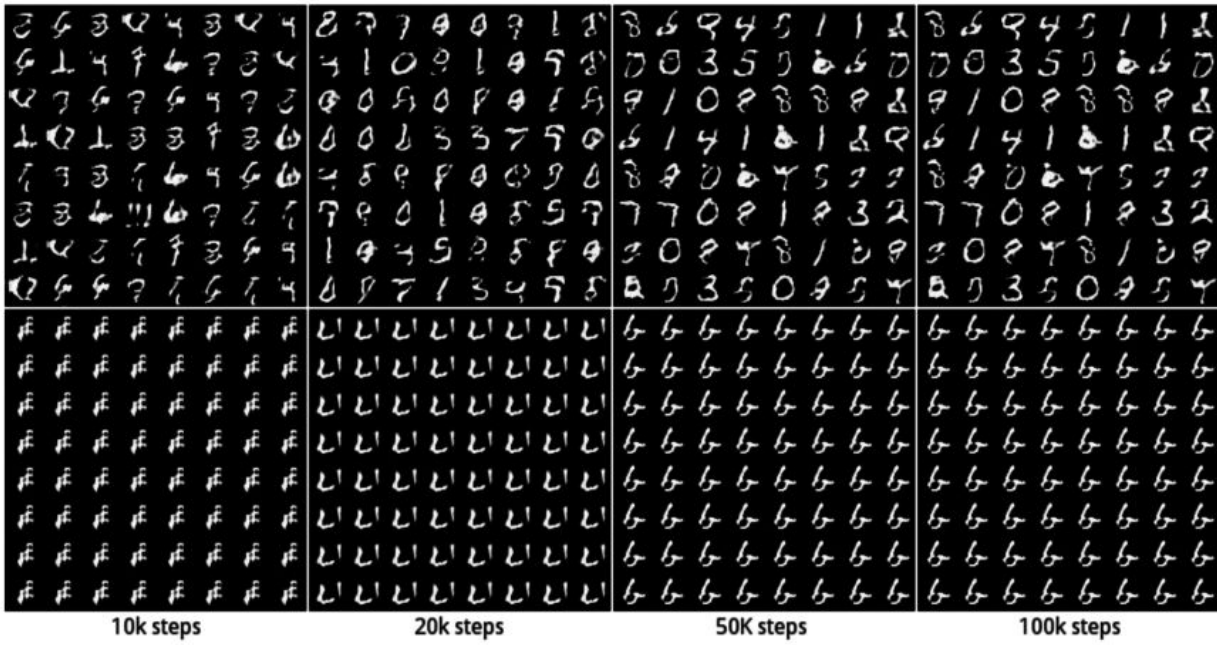

Fig 1: Mode Collapse in the MNIST Dataset

Another persistent issue is that since the Generator and Discriminator are playing a minimax game, their losses are not indicative of the quality of output. Thus there is no definite point where one should stop training the GAN. Also, during the latter half of training, the losses of both the networks start to oscillate. While a GAN is theoretically grounded and is expected to achieve the Nash Equilibrium in the end, practically that is not the case. If this had been achievable, the point of stopping the training could be easily determined.

GANs can also have a hard time figuring out the structural components for given data distribution. Let's suppose one is trying to generate images from a random distribution. While for most random vectors realistic images are generated, there are also cases when the Generator is not able to even generate the structure of the face.

There are two primary aims of this project. First, to add a regularising factor to the Generator part of the Discriminator's cost function. The intuition behind this method is that after some training, the Generator starts producing good quality images which are closer to the data distribution. Now, if the Discriminator states that this is entirely fake by labeling it as 0, it is essentially destabilizing itself by considering a point which belongs to the data distribution as not belonging to it. Hence, to regularise this, a factor

(between 0 to 1) is added so that Discriminator pushes all the fake sample to this value rather than 0.

The second aim of this project is to ensure the production of stable structural integrity by pre-training the networks on conventional loss functions and then putting them through a GAN based training phase. The intuition behind this approach is that before the GAN training starts, both the networks become competent at their respective tasks. Hence it becomes easier for the GAN phase to pick up on the structural components of the data. The implementation of this method is provided in the Experiment Methodology section.

---

2 THEORY

2.1 Introduction to GAN

A Generative Adversarial Network or GAN is a generative model which generates data whose distribution mimics the probability distribution of the input data. There are two networks in a GAN. The first one is the Discriminator, which tells whether a given input is real or not. If the input for the Discriminator is from the real data distribution, it should give an output of 1. Else, if the input is from the generated data distribution, it should give an output of 0. The other one is the Generator, which produces fake data with an aim to fool the Discriminator.

In the ideal case, the Generator is perfectly able to reproduce the data distribution. Hence, in this case:

$$GeneratedDistribution(P_g) = DataDistribution(P_d) \tag{1}$$

To get this result both the networks play a minimax game and at the end of the game (theoretically), the Nash Equilibrium is attained by the system. That is Generator starts producing very realistic fakes and Discriminator can’t decide whether the input is real or not. In this situation, the Discriminator starts to output 0.5 for all inputs. The Nash Equilibrium is achieved by the GAN when it has minimized the Jenson-Shanon Divergence of the data distribution and the generated distribution.

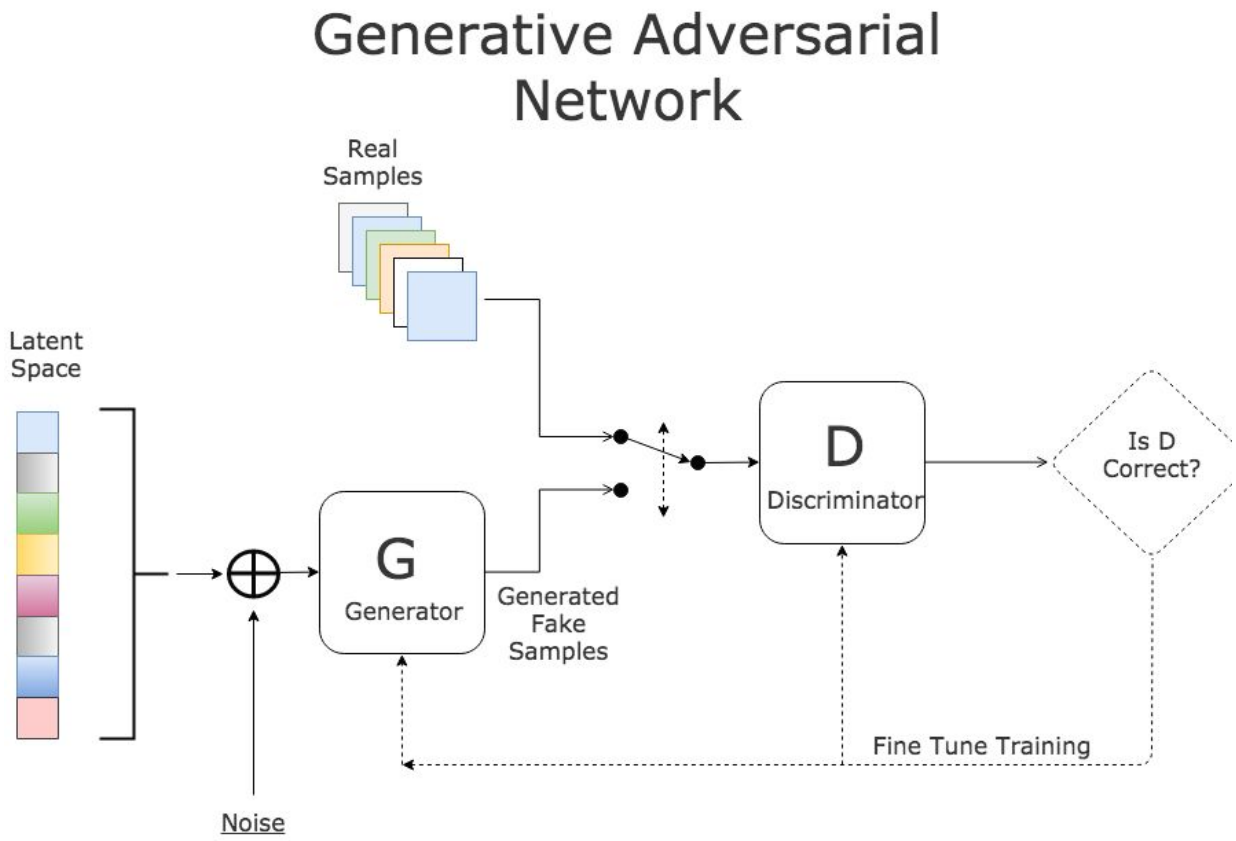

Fig 2: Generative Adversarial Network

2.2 Cost Function of a GAN

The entire aim of the GAN training is to pit both the networks in a minimax game. Hence, the cost function is also designed in such a way. As initially proposed by Goodfellow the cost functions for the Discriminator and Generator are as follows :

$$\max_D V(D) = \mathbb{E}_{x \sim p_{\text{data}}\ (x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \qquad (2)$$

$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{x}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \text{ or } \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log D(G(\boldsymbol{z}))] \qquad (3)$$

It should be noted that for the Generator an alternative cost function is chosen. As stated by Goodfellow, this choice isn't theoretically grounded but heuristically motivated. The main motivation for making this choice is that initially, the Discriminator gets very good at differentiating between fake images and real images. This happens because the fake images are just random noise. Hence, the gradient for the original cost

function of the Generator tends towards 0. Thus, initially, the learning for the Generator is very slow. To tackle this, the alternative cost function is used as it doesn't have 0 gradient initially.

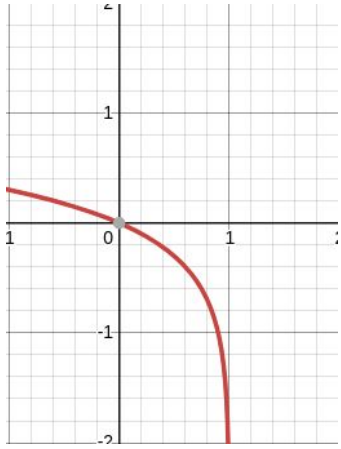

Fig 3: Graph of original cost function of the Generator

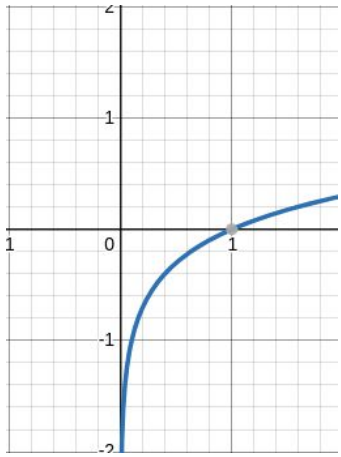

Fig 4: Graph of alternative cost function of the Generator

2.3 Types of Generative Adversarial Networks

2.3.1 Wasserstein GAN

The Wasserstein GAN (WGAN) is a GAN variant which uses the 1-Wasserstein distance (Eq 5), rather than the JS-Divergence (Eq 4), to measure the difference between the generated and target distributions. It is usually the case that the generated and target distributions do not overlap, when this happens the JS-Divergence becomes a constant (log2). It can be mathematically proven that the learning of the Generator depends on the gradient of the distance metric on which the model is based upon. Since the original

GAN was based on JS-Divergence, it's learning might halt if such a situation occurs. This is where the Wasserstein distance is useful. Its value never becomes a constant, even when the distributions are not overlapping. The advantages of using W-GAN is that there are no signs of mode collapse during experiments. Also, the Generator can still learn when the Discriminator performs well. The change in the cost function is described in Eq 6 (Critic) and Eq 7 (Generator).

$$D_{JS}(p\|q) = \frac{1}{2}D_{KL}\left(p\|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\|\frac{p+q}{2}\right) \tag{4}$$

$$W\left(\mathbb{P}_r, \mathbb{P}_\theta\right) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{5}$$

$$\nabla_w \frac{1}{m}\sum_{i=1}^{m}\left[f\left(x^{(i)}\right) - f\left(G\left(z^{(i)}\right)\right)\right] \tag{6}$$

$$\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f\left(G\left(z^{(i)}\right)\right) \tag{7}$$

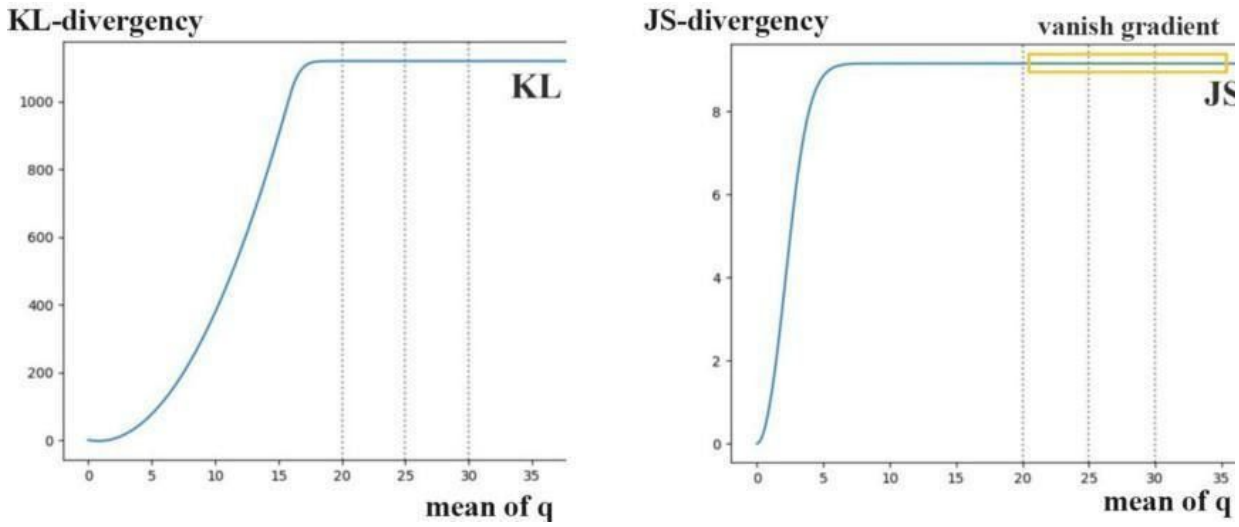

Fig 5: Vanishing gradient of KL and JS-Divergence

2.3.2 Least-Squares GAN

The main idea of LSGAN is to use loss function that provides a smooth and non-saturating gradient in is Discriminator. We want the Discriminator to “pull” data generated by Generator towards the real data $\mathbf{P_{data}(X)}$ so that the Generator generates data that are similar to $\mathbf{P_{data}(X)}$.

As we know in the original GAN, Discriminator uses log loss. The decision boundary is as given in the following figure:

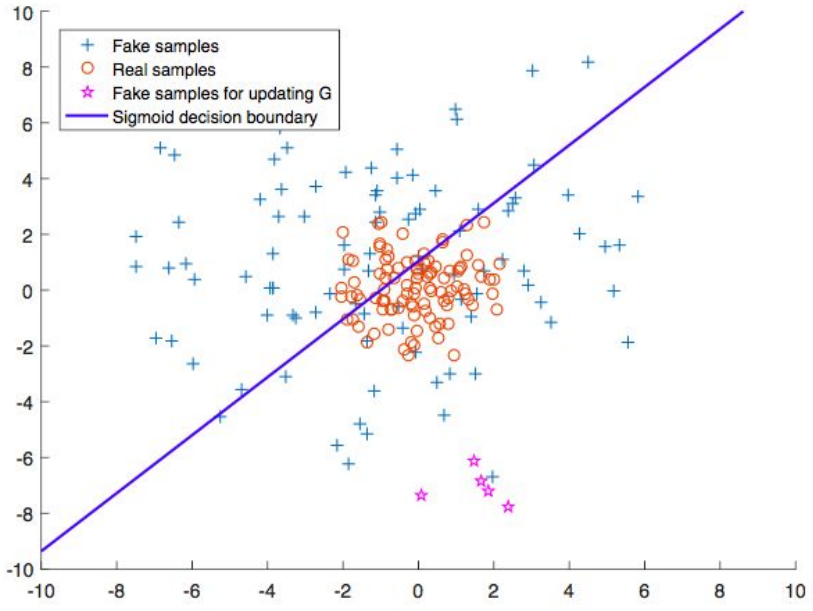

Fig 6: Decision Boundary of Original GAN

As Discriminator uses the sigmoid function, and as it is saturating very quickly, even for small data point **x**, it will quickly ignore the distance of **x** to the decision boundary **w**. What it means is that it essentially won’t penalize **x** that is far away from **w.** That is, as long as **x** is correctly labeled, the system is doing its job. Consequently, as **x** becoming bigger and bigger, the gradient of Discriminator quickly goes down to 0, as log loss doesn’t care about the distance, only the sign.

For learning the distribution of $\mathbf{P_{data}(X)}$, then log loss is not effective. The Generator is trained using the gradient of Discriminator. If the gradient of Discriminator is saturating to 0, then Generator won’t have the necessary information for learning $\mathbf{P_{data}(X)}$.

Fig 7: Decision Boundary of LS-GAN

In **L2** loss, data that are quite far away from **w**(in this context, the regression line of $\mathbf{P_{data}(X)}$) will be penalized proportionally to the distance. The gradient, therefore, will only become 0 when **w** perfectly captures all of **x**. This will guarantee Discriminator to yield informative gradients if Generator has not captured the data distribution.

The cost functions for Discriminator and Generator of LS-GAN are described as follows:

$$\min_D V_{\mathrm{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x})-1)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[D(G(\boldsymbol{z}))^2\right] \quad (8)$$

$$\min_G V_{\mathrm{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{z}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-1)^2\right] \quad (9)$$

Most of the experiments in this project are performed in LS-GAN.

2.4 Progressive Growing of GANs

The progressive growing of GANs trains the GAN network in multiple phases. In phase 1, it takes in a latent feature **z** and uses two convolution layers to generate 4×4 images. Then, we train the discriminator with the generated images and the 4×4 real images. Once the training stables, we add 2 more convolution layers to upsampling the image to 8×8 and 2 more convolution layers to downsampling images in the discriminator.

The progressive training speeds up and stabilizes the regular GAN training methods. Most of the iterations are done at lower resolutions, and training is 2–6 times faster with comparable image quality using other approaches. In short, it produces higher resolution images with better image quality. Here is the final network when all phases of training are completed.

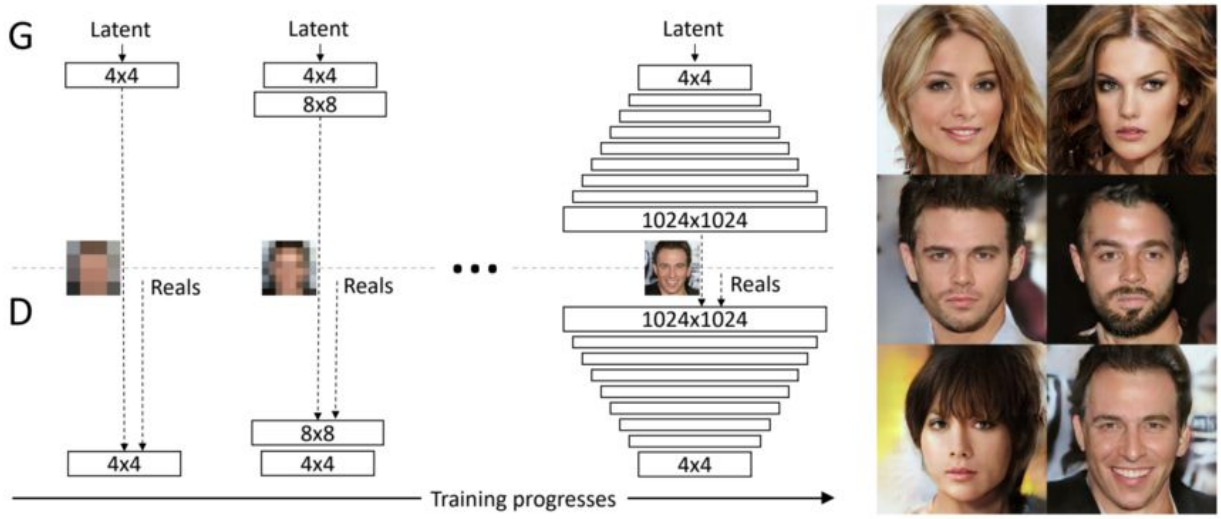

Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Fig 8: Framework of Progressive Growing of GAN

The second part of the project is inspired from this framework. Its implementation is described in the Experiment Methodology section.

---

3 LITERATURE REVIEW

There have been very few studies regarding dynamically changing the cost function of GAN in order to stabilize training. Upon reviewing several papers like the original paper on Generative Adversarial Networks (Goodfellow, et al, 2014), Wasserstein GAN paper (M. Arjovsky, et al, 2017), Improved Training of Wasserstein GANs (I. Gulrajani, et al, 2017) and the LS-GAN paper (Mao, et al, 2017) there was little to no work on such a concept. Salimans, et al, 2016 , proposed the concept of label smoothing. Label smoothing states that when the discriminator gets a real sample, its label should be assigned randomly between 0.7 to 1.2 instead of 1.0 and for fake samples, its label

should be assigned randomly between 0.0 to 0.3 instead of 0.0. This method helped the GAN training but seems rather random. Therefore, the first part of the project is based on independent intuition and has been applied to the LSGAN architecture.

The second part of the project has been inspired from the Progressive Growing of GANs (Karras T, et al, 2018) in which GAN training is done by progressively increasing the layers of the Generator and the Discriminator. The paper reports that training such a model is much more stable and around 2-6 times faster than conventional means. Through this method, they are able to produce high definition images of fake celebrity of 1024x1024 pixel dimension. This training has been performed on the CelebA-HQ dataset. To produce these results they trained the model for 20 days with an Nvidia P100 GPU. Due to limitations of GPU availability, the second part of the project is scaled up to 64x64 pixel dimension using the CelebA dataset.

The concept of pre-training is inspired by the implementation of a deep learning project called DeOldify by jantic. In this project, the task is to colourize black and white images. To get stable results, a method of NoGAN training is adopted where the Discriminator and Generator are trained separately. The Generator is trained to produce coloured images using feature loss as stated by Johnson J, et al, 2016. The Discriminator is trained as a binary classifier on real data and generated samples. This training is followed by the GAN training which helps in "increasing the realism" of the generated results.

---

4 EXPERIMENT METHODOLOGY

4.1 Model Architecture and Dataset

The model architecture for both the parts of the project is described in the following figures:

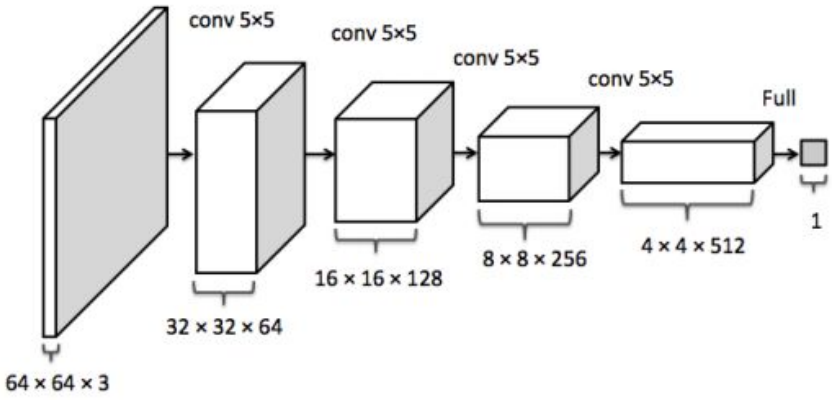

Fig 9: Discriminator/Critic Architecture

The Discriminator accepts **64x64 images** and outputs whether the given input was real (1) or fake (0). The hidden layers are convolutional layers with kernel **stride 2** and **padding 1**. Effectively, each layer is downscaling the image by a factor of 2.

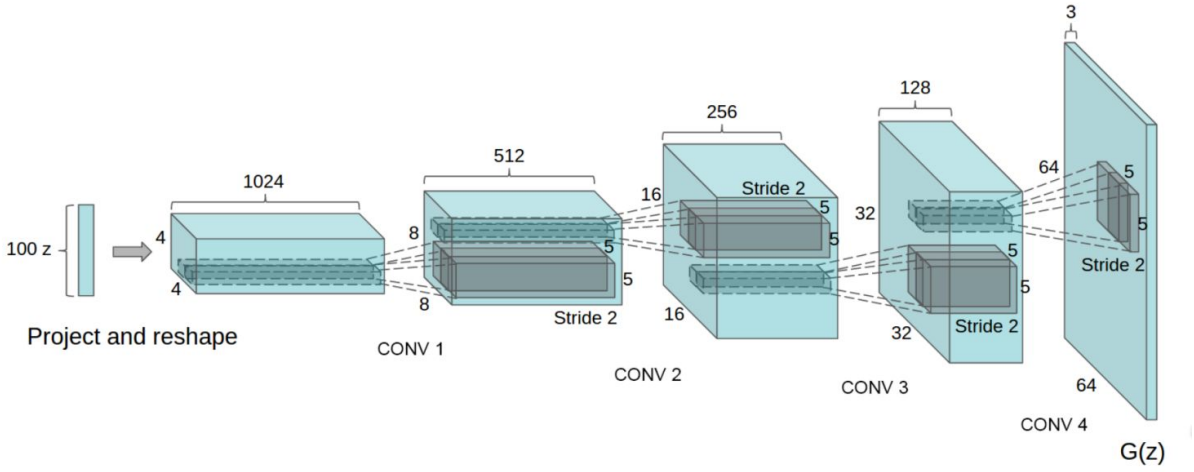

Fig 10: Generator Architecture

The **Generator accepts a 100x1 vector** and generates a **64x64** image. The hidden layers are convolution transpose layers with kernel **stride 2** and **padding of 1** which scale up the current image size by a **scale of 2**.

The dataset used for these experiments is the Celeba dataset. CelebFaces Attributes Dataset (CelebA)is a large-scale face attributes dataset with more than **2,00,000 celebrity images, each with 40 attribute annotations**. The images in this dataset cover large pose variations and background clutter.

4.2 Regularisation of the Cost Function

On training regular GANs like the original GAN, W-GAN, and LS-GAN it was observed that the losses for both the Generator and Discriminator oscillated during the latter half of the training. With an aim to dampen these oscillations a method was devised to regularize the cost function of the Discriminator. Most of the experiments were done on LS-GAN and some on W-GAN.

The cost functions of the Discriminator and Generator of a regular LS-GAN are specified in equation 7 and 8.

To reduce the said oscillations the following change was made to the cost function of the Discriminator:

Discriminator:

$$\min_D V_{\mathrm{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x})-1)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[D(G(\boldsymbol{z})-\boldsymbol{\alpha})^2\right] \tag{10}$$

The GAN is trained for some time, keeping **α as 0**. When the Generator starts to produce relatively good outputs, the value of α is varied by using a monotonically increasing function of iterations. Several functions like **Linear, Step and Smooth-Step functions** were tested for this. The best results were obtained when using a linear function to vary α. The **maximum values** to which **α** were varied were **0.3, 0.5 and 0.9**. The results were comparable with slightly **better results** for the case of **0.5**. The **GAN training** was done for **1,00,000 iterations for all cases** with a **learning rate of 0.0004.**

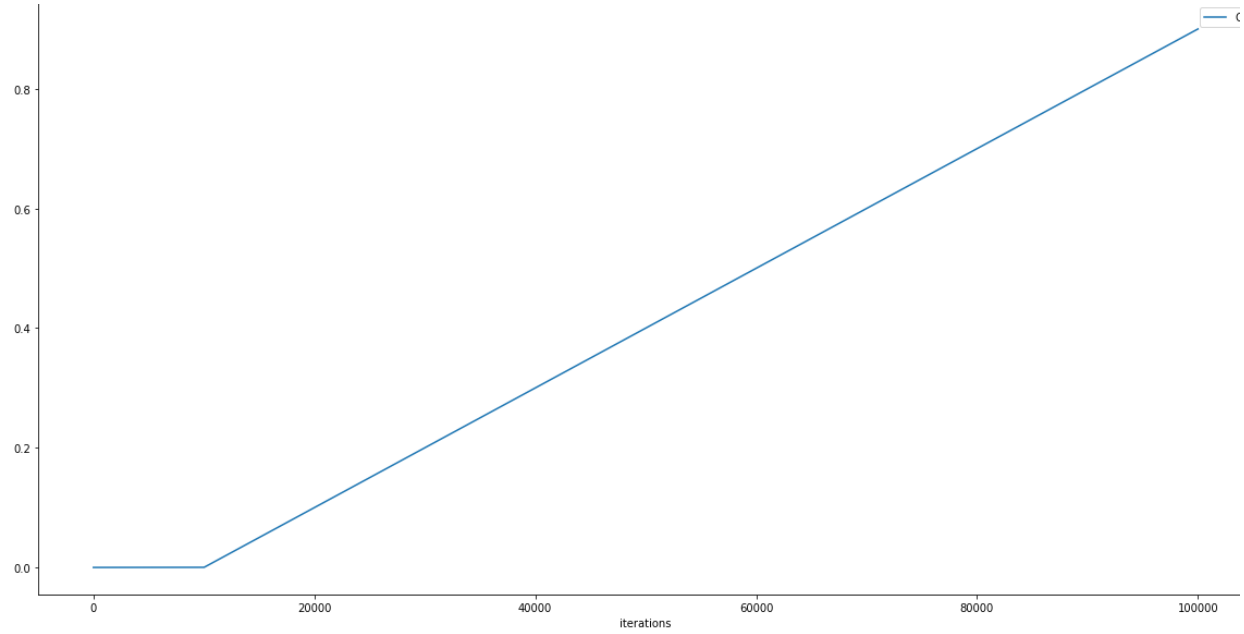

Fig 11:
Plot of α with respect to number of iterations. This is the case where α is varied as a linea r function of iterations.

4.3 Progressive Scaling and Pre-training of GAN

This part of the project focuses on pre-training of the Generator and Discriminator. The Generator is further pre-trained in a progressive manner, by adding one layer on top of another as the pre-training goes on. The proper implementation details are as follows.

4.3.1 Pre-Training of Generator and Discriminator

To pre-train the Generator, in this stage, each image of the dataset is paired with a random vector sampled from the normal distribution. Now, the **Generator is trained for producing 4x4 images with L1 loss**. Here are targets are images of the dataset (resized to 4x4) and inputs are there corresponding random vector. This training is done for **10 epochs** on the dataset. By the end of this training, the Generator is capable of generating 4x4 images. Now, additional layers are added to the Generator generating **8x8 images**. They are then trained in the exact way as mentioned above. This is scaled up so that the Generator could produce 64x64 images. For each case, the **learning rate used for pre-training the Generator was 0.001** and also, the **Adam optimizer** was used with β_1 **equal to 0.5** and β_2 **equal to 0.999**. Here, β_1 and β_2 are the hyperparameters of the Adam optimizer. Specifically, β_1 is the exponential decay rate of the running average

of the gradient, whereas, $\delta 2$ is the exponential decay rate of the running average of the square of gradient.

Once the **Generator was able to produce 64x64 images,** they were stored. These stored images would act as fake samples for the pre-training of the Discriminator. These fake samples have the target of 0 while images from the dataset are given the target of 1. With these samples and targets, the **Discriminator was pre-trained for 10 epochs.** The **learning rate used for pre-training the Discriminator was 0.001** and also, the **Adam optimizer** was used with **$\delta 1$ equal to 0.5 and $\delta 2$ equal to 0.999.** The loss function used for this classifier is described as follows:

$$\min_{D}(D(x) - y)^2 \tag{11}$$

Where x is the input of the Discriminator and y is the target for a given input. This was done because LSGAN has a similar loss function. Thus, to ensure a smooth transition from pre-training to GAN training this cost function was adopted.

4.3.2 GAN Training

Once the pre-training was completed, both the models were paired in an LSGAN setting. **This LSGAN was trained for 1,00,000 epochs** with **learning rate for Generator equal to 0.0004 and the learning rate for Discriminator equal to 0.0003.** The **same optimizers** were used as mentioned in the Pre-Training of Generator and Discriminator section. Further experiments were conducted by training this LSGAN by applying the regularising factor as mentioned in the Regularisation of the Cost Function section.

---

5 RESULTS

5.1 Regularisation of the Cost Function

The results of regularising the Cost Function is clearly visible in figures 12 and 13. Figure 12 represents the loss curve of the Generator and Discriminator for the base LS-GAN. It can be seen that after around 20,000 iterations the loss of both the Generator and Discriminator start to oscillate a lot. The Generator's loss oscillates between 0.1 to 0.7 while the loss of the Discriminator oscillates between 0.05 to 0.2

with rare occasions of shooting up to 1.5. This erratic behavior leads to the quality of generated images to also oscillate somewhat.

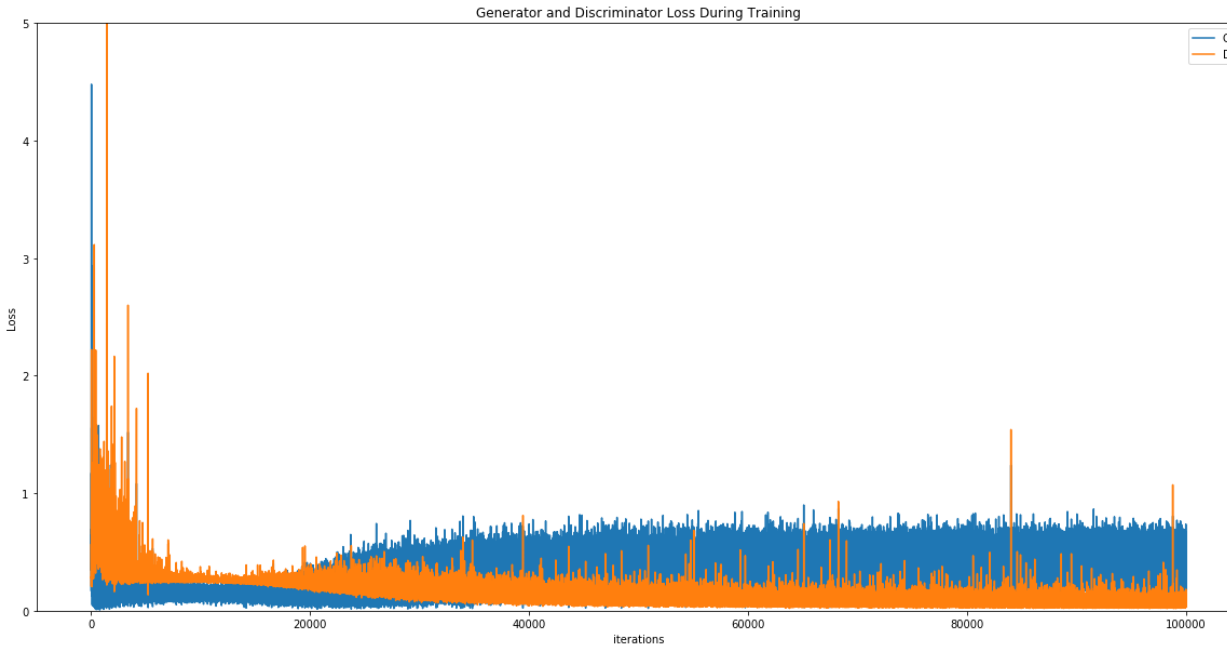

Fig 12: Oscillations of Generator (G) and Discriminator (D) losses

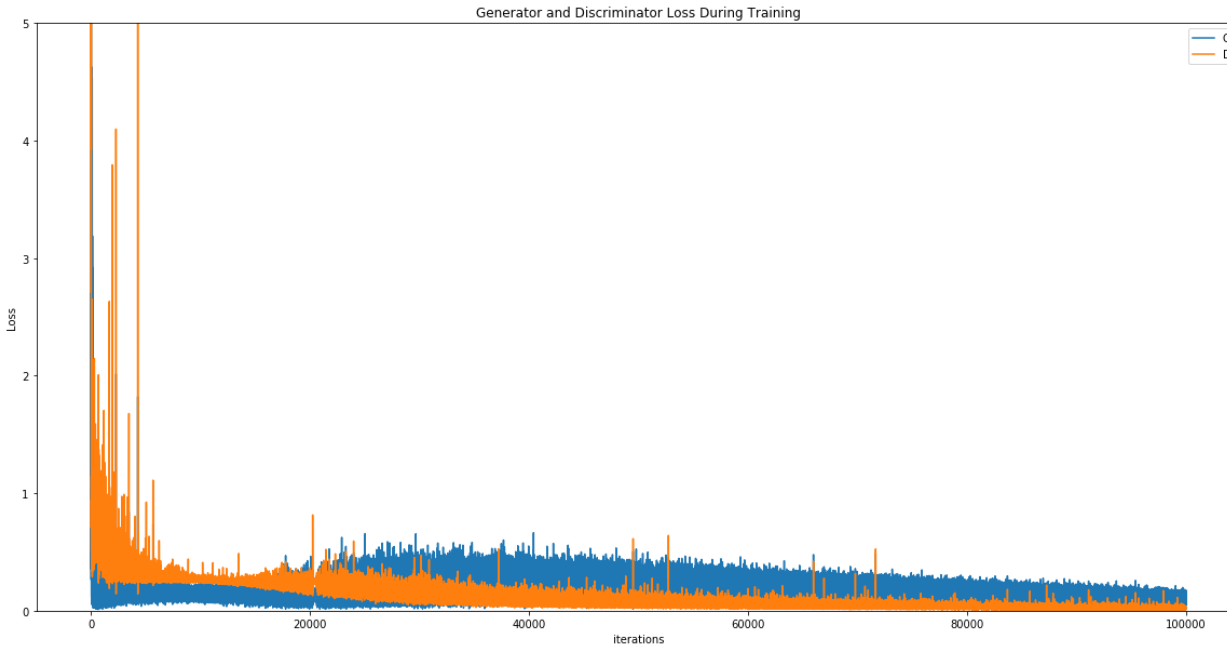

Fig 13:
Loss of Generator and Discriminator for LS-GAN when α was varied as the function given i n Fig 11.

A distinct difference in the graph of loss can be seen in figure 13. This is the graph for the regularized LS-GAN. Now, the losses do not oscillate a lot. Also, the quality of

images produced by the regularised model is comparable if not better than the regular model.

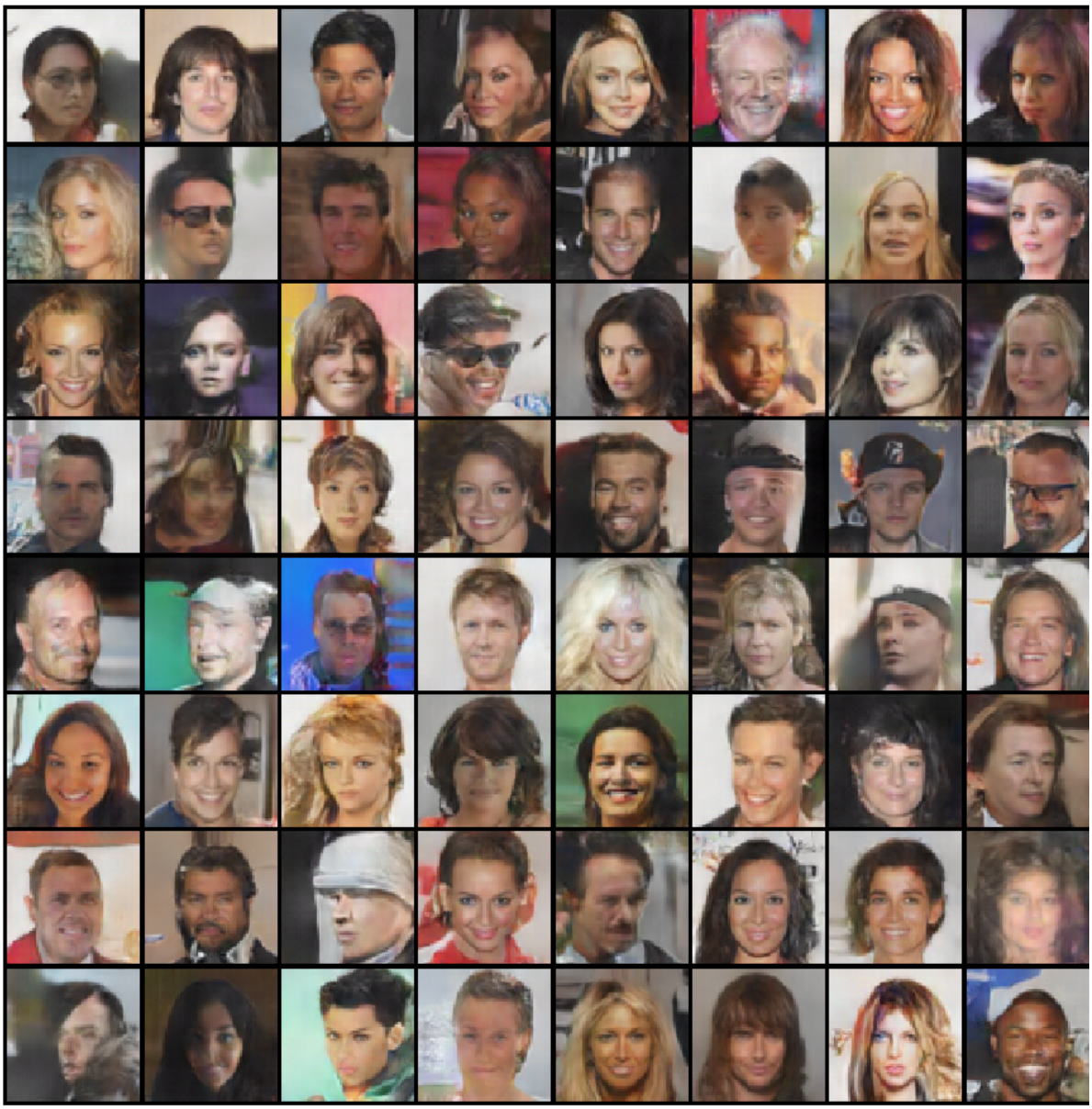

Fig 14: Final results of base model after 1,00,000 iterations

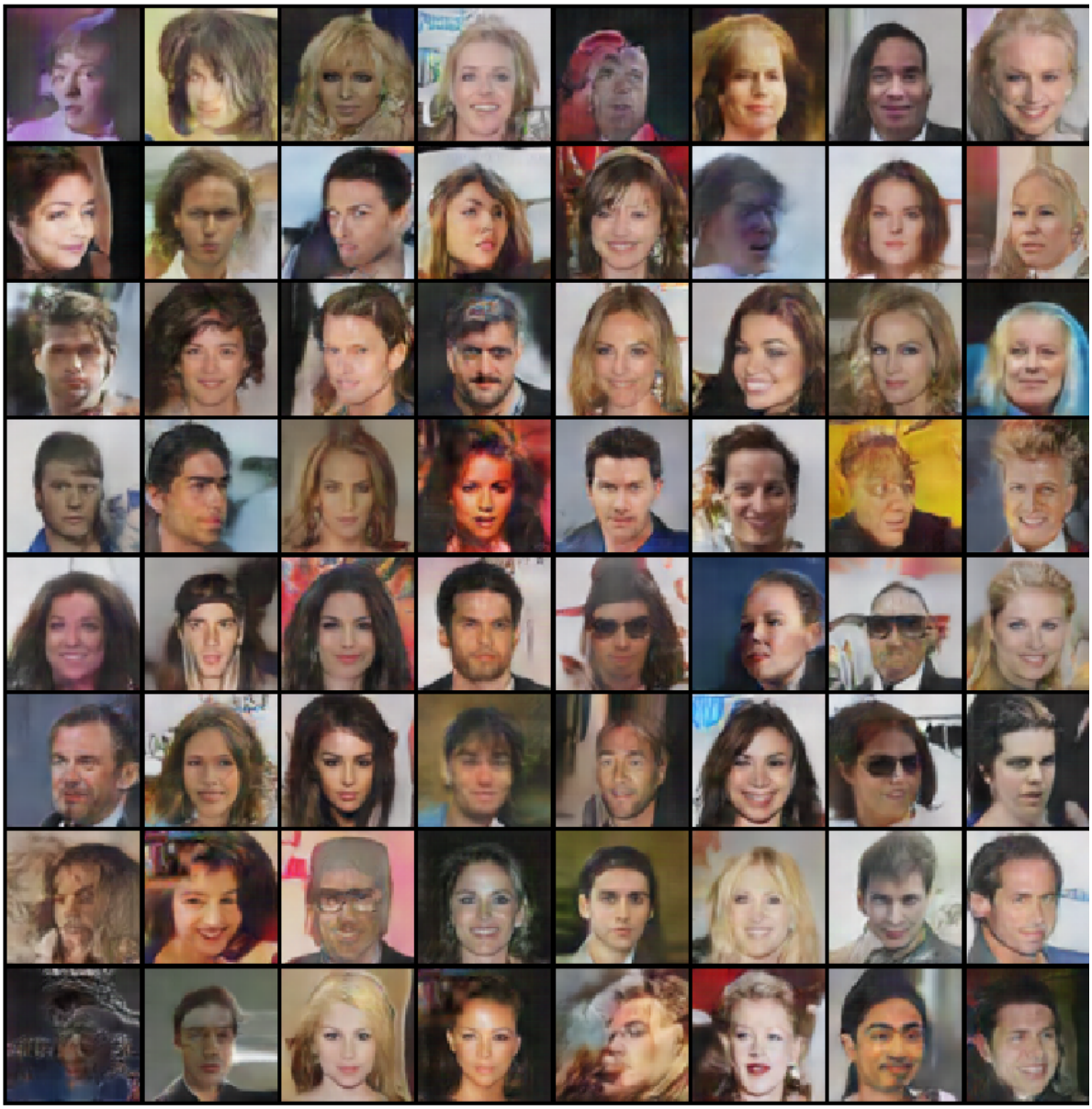

Fig 15: Final output of regularised model after 1,00,000 iterations

5.2 Progressive Scaling and Pre-training of GAN

This experiment also showed some promising results. First of all, the images produced by the pre-trained generator were convincing enough to move onto the GAN training. Although these images weren't realistic they had captured the structural integrity of the faces, thus they had performed their task.

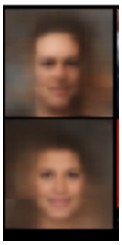

Fig 21: Output of pre-trained Generator (64x64)

The final outputs of this networks are given below:

Fig 16: Output after GAN training for 1,00,000 iterations

It was inferred that all the structural features of all the faces are intact. Even for the bad cases, one can clearly see that the model has generated the structure of the face, if not the finer details. This wasn't the case in the base model, where the bad cases didn't have any facial features at all.

For this model also the losses oscillated a lot. So, this model was also regularized and the loss curves are given below along with the output of this regularised model. There were few spikes in the losses but the oscillations did decrease as the training progressed

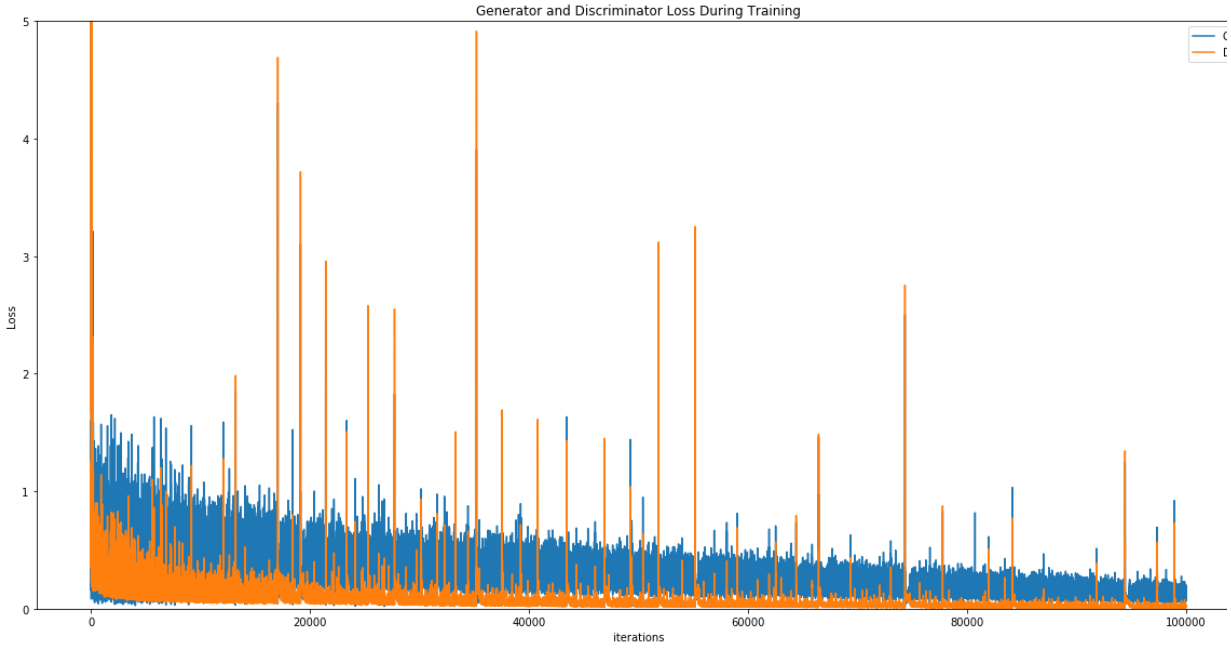

Fig 23:
Loss curves of Generator and Discriminator for the regularised GAN training after the pre-tr aining the networks.

Fig 17: Output after regularised GAN training for 1,00,000 iterations.

6 SUMMARY AND CONCLUSION

In this project, various methods of stabilizing GAN training was investigated. Firstly, an understanding of GANs was carried out, along with different procedures to training such GANs. It was found out that very less attention was given to the stabilization of the Discriminator during training, hence, experiments were carried out to stabilize it.

In order to do this, a regularizing factor was subtracted from the Generator part of the Discriminator's cost function. This factor was varied according to a monotonically increasing function of iterations. It was observed, that the oscillations in the losses of the Generator and Discriminator did indeed decrease. This technique didn't hamper the image quality, in fact, produced images of quality at par with the base model.

A second experiment was performed with an aim to preserve structural components of the generated images. In order to achieve this, the Discriminator and Generator were pre-trained separately. The Generator was pre-trained in a stacked fashion. After the pre-training, they were then put through the conventional GAN training. On completing this experiment it was observed that the produced images had better structural integrity than the base model, owing to the pre-training. you

It can be concluded that these methods indeed lead to a more stable training of GANs. Further study needs to be done on these methods. Specifically, the pre-training loss function of the Generator can be changed from the regular L1 loss to a more robust perceptual loss as mentioned by Johnson J, et al, 2016.

References

1. Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. NIPS, 2014.
2. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. arXiv preprint arXiv:1701.07875, 2017.
3. I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. arXiv preprint arXiv:1704.00028, 2017.
4. Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Smolley, S. P. Least squares generative adversarial networks. In International Conference on Computer Vision, 2017.

5. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X., 2016. “Improved techniques for training gans”. In Advances in Neural Information Processing Systems, 2016.
6. Karras T, Aila T, Laine S, Lehtinen J. Progressive growing of gans for improved quality, stability, and variation. Proceedings of the International Conference for Learning Representations (ICLR), 2018.
7. Johnson J, Alahi A, Li FF. Perceptual losses for real-time style transfer and super-resolution. In European conference on computer vision, 2016.

Sources

1. Fig 1: https://arxiv.org/pdf/1611.02163.pdf
2. Fig 2: https://iconix.github.io/dl/2018/07/28/lcgan
3. Fig 5: https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490
4. Fig 6: https://wiseodd.github.io/techblog/2017/03/02/least-squares-gan/
5. Fig 7: https://wiseodd.github.io/techblog/2017/03/02/least-squares-gan/
6. Fig 8: https://arxiv.org/pdf/1710.10196.pdf
7. Fig 9: http://bamos.github.io/2016/08/09/deep-completion/
8. Fig 10: https://arxiv.org/pdf/1511.06434.pdf