# A comparative study of Generative Adversarial Networks [*]

R Sainiranjan
BS Research (Mathematics)
rsainiranjan@iisc.ac.in

May 14, 2020

**Abstract**

In this study , we perform a comparative analysis of several Generative Adversarial Networks (GAN) architectures.Initially proposed by Ian Goodfellow several years ago ; these models are being used increasingly in practice for a wide range of tasks including being used as generative networks or to be trained as auxiliary Discriminate models . The advantages that these networks have over other generative structures like VAEs(Variational Autoencoders) lies in their great capacity to generalise out of the training space and produce impressive results . However problems such as huge efforts required in training and practical issues like Mode Collapse can offer lower the quality of results . In this study we shall be discussing the pros and cons of several architectures used in the context of image generation over a standard dataset

## 1 Introduction

Generative adversarial networks work with two neural network models simultaneously. The first is a generative model that produces synthetic examples of objects that are similar to a real repository of examples. Furthermore, the goal is to create synthetic objects that are so realistic that it is impossible for a trained observer to distinguish whether a particular object belongs to the original data set, or whether it was generated synthetically. For example, if we have a repository of car images, the generative network will use the generative model to create synthetic examples of car images. As a result, we will now end up with both real and fake examples of car images. The second network is a discriminative network that has been trained on a data set which is labeled with the fact of whether the images are synthetic or fake. The discriminative model takes in inputs of either real examples from the base data or synthetic objects created by the generator network, and tries to discern as to whether the objects are real or fake. In a sense, one can view the generative network as a "counterfeiter" trying to produce fake notes, and the discriminative network as the "police" who is trying to catch the counterfeiter producing fake notes. Therefore, the two networks are adversaries, and training makes both adversaries better, until an equilibrium is reached between them.

We can present the objective functions for the two models as follows :

[*]Project 4 of E0 250 Deep Learning Course

$$\text{Maximize}_D\, J_D = \underbrace{\sum_{\overline{X} \in R_m} \log\left[D(\overline{X})\right]}_{m \text{ samples of real examples}} + \underbrace{\sum_{\overline{X} \in S_m} \log\left[1 - D(\overline{X})\right]}_{m \text{ samples of synthetic examples}}$$

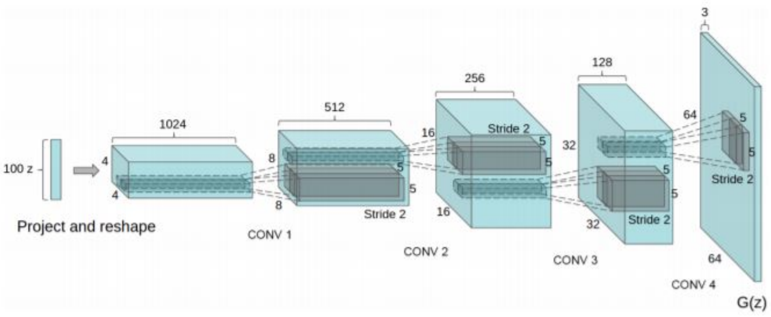
$$\text{Minimize}_G\, J_G = \underbrace{\sum_{\overline{X} \in S_m} \log\left[1 - D(\overline{X})\right]}_{m \text{ samples of synthetic example}}$$
$$= \sum_{\overline{Z} \in N_m} \log\left[1 - D(G(\overline{Z}))\right]$$

Here D and G represent the models over the training samples X

## 2 Need for a Convolutional GAN - DCGAN

While GANs are able to generalise well for simple distributions , complex distributions involving natural images prove to be a challenge for the classical vanilla architecture .In such scenarios it is reasonable to ”Augment” the GAN by adding capabilities that can capture the inherent relations in images better and use those for generation and training . Thus integration of a CNN becomes quite necessary in this pipeline . One such architecture that is used is the DCGAN (Deep Convolutional GAN) .

Figure 1: DCGAN architecture



**Our network in the implementation takes in a 100x1 noise vector, denoted z, and maps it into the G(Z) output which is 64x64x3.**

This architecture is especially interesting the way the first layer expands the random noise. The network goes from 100x1 to 1024x4x4! This layer is denoted ‘project and reshape’.

We see that following this layer, classical convolutional layers are applied which reshape the network with the (N+P — F)/S + 1 equation classically taught with convolutional layers. In the diagram above we can see that the N parameter, (Height/Width), goes from 4 to 8 to 16 to 32, it doesn’t appear that there is any padding, the kernel filter parameter F is 5x5, and the stride is 2. We see that in this case the network goes from :
100x1 to 1024x4x4 to 512x8x8 to 256x16x16 to 128x32x32 to 64x64x3

## 3 Wasserstein Loss - as an improvement for DCGAN

One of the reason GANs are so difficult to train is that the architecture involves the simultaneous training of a generator and a discriminator model in a zero-sum game. Stable training requires finding and maintaining an equilibrium between the capabilities of the two models.

The discriminator model is a neural network that learns a binary classification problem, using a sigmoid activation function in the output layer, and is fit using a binary cross entropy loss function. As such, the model predicts a probability that a given input is real (or fake as 1 minus the predicted) as a value between 0 and 1.

The loss function has the effect of penalizing the model proportionally to how far the predicted probability distribution differs from the expected probability distribution for a given image. This provides the basis for the error that is back propagated through the discriminator and the generator in order to perform better on the next batch.

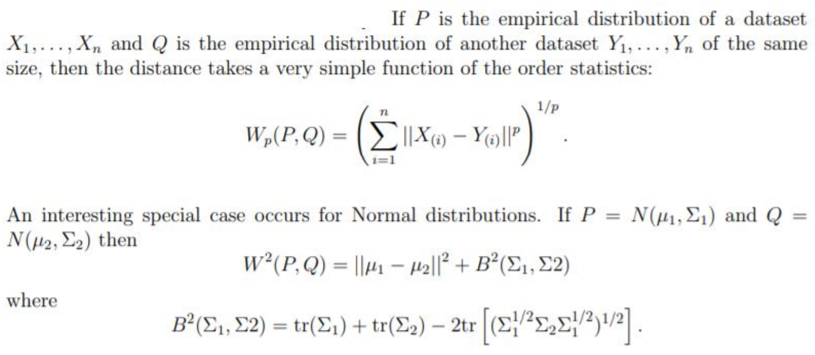
We therefore use a novel idea of ”Wasserstein GANs” to try and improve the results from our DCGAN model . Instead of using a discriminator to classify or predict the probability of generated images as being real or fake, the Wasserstein GAN changes or replaces the discriminator model with a critic that scores the realness or fakeness of a given image.

This change is motivated by a mathematical argument that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples.The Earth-Mover (EM) distance, referred to as Wasserstein distance can be used in this regard. A critic neural network can be trained to approximate the Wasserstein distance, and, in turn, used to effectively train a generator model.

Importantly, the Wasserstein distance has the properties that it is continuous and differentiable and continues to provide a linear gradient, even after the critic is well trained.This is unlike the discriminator model that, once trained, may fail to provide useful gradient information for updating the generator model.The benefit of the WGAN is that the training process is more stable and less sensitive to model architecture and choice of hyperparameter configurations.Perhaps most importantly, the loss of the discriminator appears to relate to the quality of images created by the generator.

Specifically, the lower the loss of the critic when evaluating generated images, the higher the expected quality of the generated images. This is important as unlike other GANs that seek stability in terms of finding an equilibrium between two models, the WGAN seeks convergence, lowering generator loss.

Figure 2: Wasserstein Distance Forumulation



If $P$ is the empirical distribution of a dataset $X_1, \ldots, X_n$ and $Q$ is the empirical distribution of another dataset $Y_1, \ldots, Y_n$ of the same size, then the distance takes a very simple function of the order statistics:

$$W_p(P,Q) = \left( \sum_{i=1}^{n} ||X_{(i)} - Y_{(i)}||^p \right)^{1/p}.$$

An interesting special case occurs for Normal distributions. If $P = N(\mu_1, \Sigma_1)$ and $Q = N(\mu_2, \Sigma_2)$ then

$$W^2(P,Q) = ||\mu_1 - \mu_2||^2 + B^2(\Sigma_1, \Sigma 2)$$

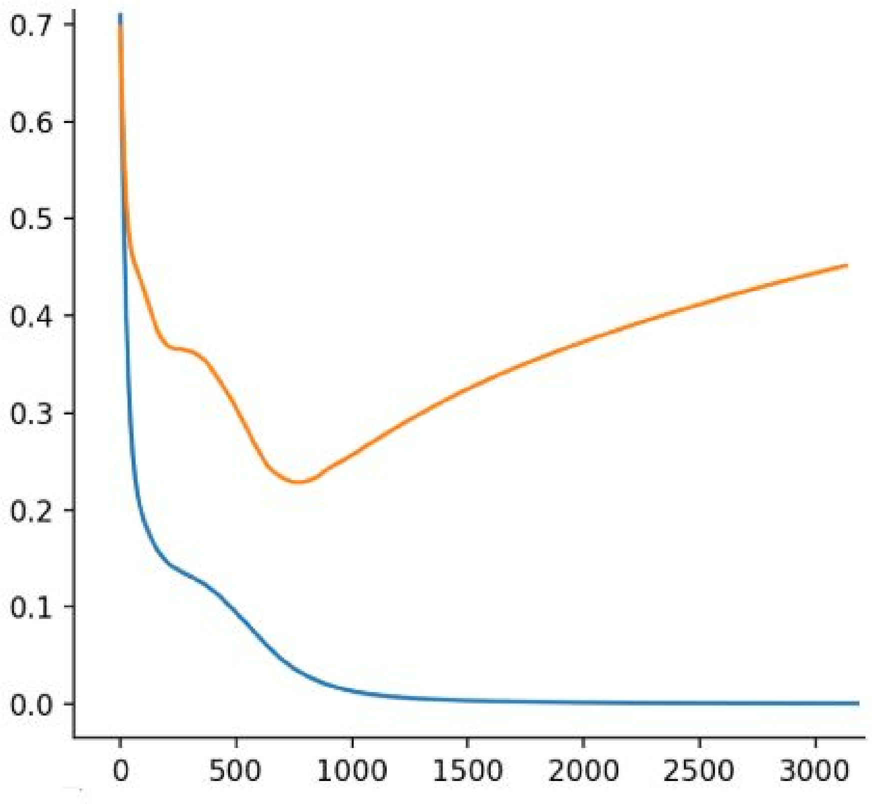
where

$$B^2(\Sigma_1, \Sigma 2) = \mathrm{tr}(\Sigma_1) + \mathrm{tr}(\Sigma_2) - 2\mathrm{tr}\left[ (\Sigma_1^{1/2} \Sigma_2 \Sigma_1^{1/2})^{1/2} \right].$$

We experimented with both a vanilla DCGAN with the standrad min-max training setup and a DCGAN augemented with wasserstein loss . The figure below shows the comparison in the rate of convergence . **In this case , the blue line represents the W-DCGAN and the orange represents the vanilla version . The y axis represents the generator loss and the x axis the training steps**

As can be seen the convergence is more rapid and better for the Wasserstein DCGAN as compared to the vanilla version.

Figure 3: Comparision in terms of convergence



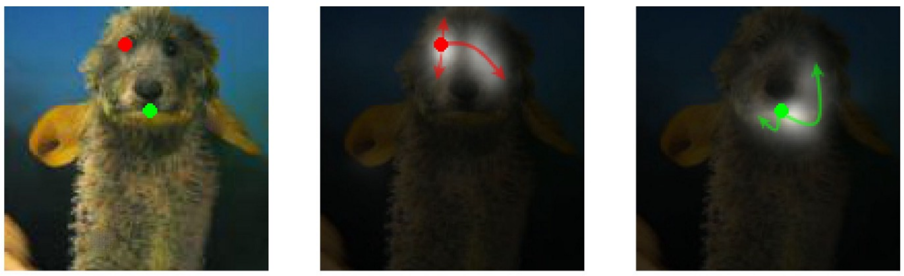
## 4 Can we do better ? - Introducing Attention

For standard convolutional based GANs such as DCGANs, they are good at classes with a lot of texture (landscape, sky) but perform much worse for structure. For example, GAN may render the fur of a dog nicely but fail badly for the dog's legs. While convolutional filters are good at exploring spatial locality information, the receptive fields may not be large enough to cover larger structures. We can increase the filter size or the depth of the deep network but this will make GANs even harder to train.For example changing the base of the Generator Network to a dense model such as Densenet-121 or Resnet -50 helps in improving the image quality but requires much more computational resources and our only suitable with powerful multi GPU systems which are themselves expensive to buy and maintain thus driving up costs greatly.

Alternatively, we can apply the attention concept.The attention concept originally from the area of Natural Language Processing helps to include **context** to various features to refine the outputs from the conventional CNNs

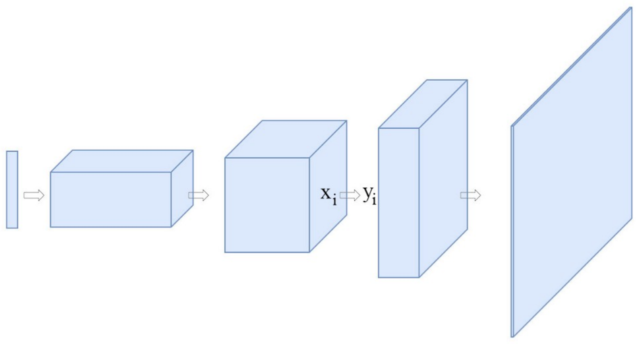
For example, to refine the image quality of the eye region (the red dot on the left figure), SAGAN only uses the feature map region on the highlight area in the middle figure. As shown below, this region has a larger receptive field and the context is more focus and more relevant. As shown below, this region has a larger receptive field and the context is more focus and more relevant. The right figure shows another example on the mouth area (the green dot).

Figure 4: Large Receptive field and context



The structure of each convolutional layer is as follows :

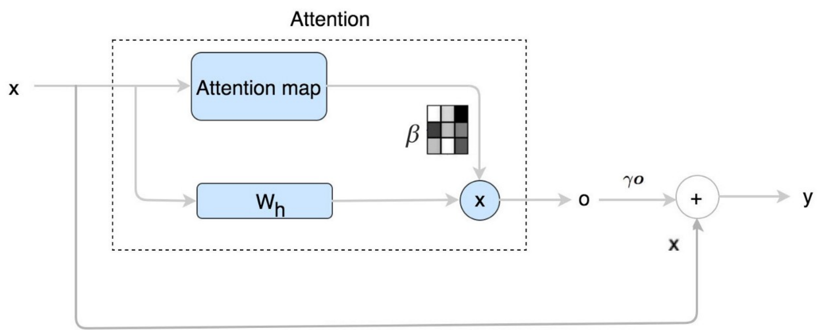
Figure 5: Design of Convolutional Layer



We refine each spatial location output with an extra term o computed by the self-attention mechanism. This extra term encodes the **context** of a particular feature and thus helps us in considering receptive fields at a micro level .

$$\boldsymbol{y_i} = \gamma \boldsymbol{o_i} + \boldsymbol{x_i}$$

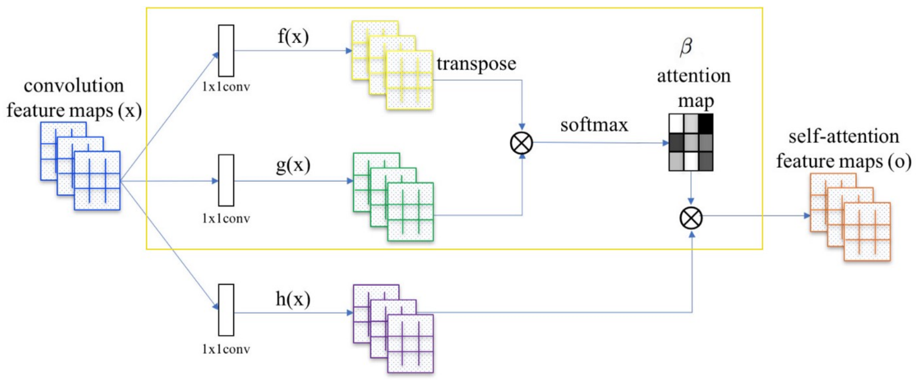
where x is the original layer output and y is the new output.
So the attention map looks like this :



The symbol β, here represents the Self Attention Map.The + operation here represents concatenation .

The full structure and equations of the attention map are as follows :

Figure 6: Attention Map architecture



$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{W_g x} \qquad \boldsymbol{x} \in \mathbb{R}^{C\times N}\ ,\ \boldsymbol{W_f} \in \mathbb{R}^{\bar{C}\times C}\ ,\ \bar{C} = C/8$$

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W_f x} \qquad \boldsymbol{W_g} \in \mathbb{R}^{\bar{C}\times C}$$

$$s_{ij} = \boldsymbol{f}(\boldsymbol{x_i})^T \boldsymbol{g}(\boldsymbol{x_j})$$

$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})} \qquad \beta \in \mathbb{R}^{N\times N}$$

$\beta_{j,i}$ indicates the extent to which the model attends to the $i^{th}$ location when synthesizing the $j^{th}$ region.

The loss function involved in the classical SAGAN architecture is the so called Hinge Loss :

$$\begin{aligned} L_D &= -\mathbb{E}_{(x,y)\sim p_{data}}[\min(0, -1 + D(x,y))] - \mathbb{E}_{z\sim p_z, y\sim p_{data}}[\min(0, -1 - D(G(z),y))], \\ L_G &= -\mathbb{E}_{z\sim p_z, y\sim p_{data}} D(G(z), y), \end{aligned}$$

Our implementation has tried other losses as well including the standard min-max loss and the Wasserstein loss . Discussion regarding these are available in the implementation section

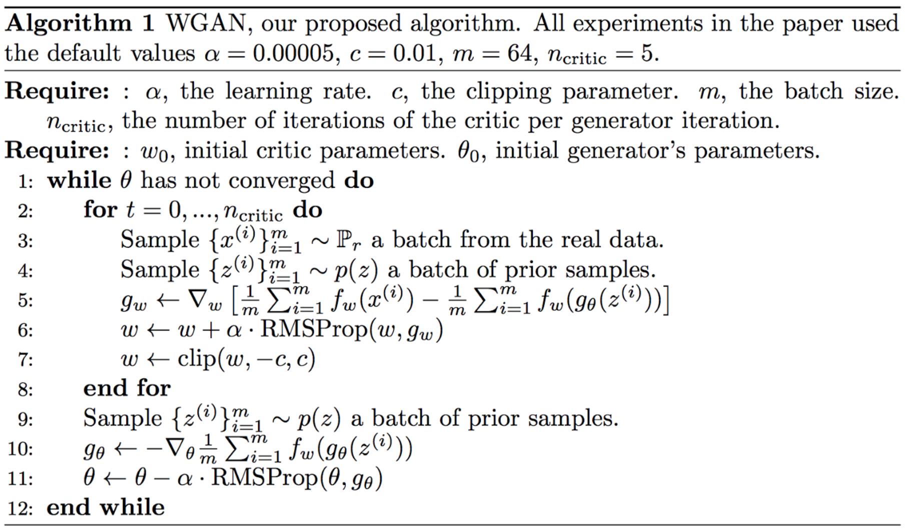
Before we go ahead , it's important to discuss the topic of **Spectral Normalization**

## 4.1 To improve Stability - Spectral Normalization

To understand the need for this let's go back to our **Wasserstein GAN** formulation that we presented earlier .Recall that WGAN claims that the Wasserstein distance will be a better cost function since it has a smoother gradient. One possibility is applying *Lipschitz constraint* to compute the Wasserstein distance. WGAN applies a simple clipping to enforce the Lipschitz-1 constraint.Lipschitz constraint refers to the fact that the loss must be *Lipschitz Continuous*. Consider the WGAN algorithm presented here :

Tuning the value of c is hard. Finding the right value is difficult. A small change can alter the gradient norm significantly. In addition, some researchers believe that the

Figure 7: adapted from : `https://arxiv.org/pdf/1701.07875.pdf`



**Algorithm 1** WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
 1: while $\theta$ has not converged do
 2:     for $t = 0, ..., n_{\text{critic}}$ do
 3:         Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
 4:         Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
 5:         $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
 6:         $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
 7:         $w \leftarrow \text{clip}(w, -c, c)$
 8:     end for
 9:     Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
```

improvement of image quality in WGAN costs the mode diversity. WGAN-GP is a modification introduces the gradient penalty to enforce the Lipschitz constraint. However, this increases the computation complexity. Thus , the need for an alternative method .

Spectral Normalization is a weight normalization that stabilizes the training of the discriminator. It controls the Lipschitz constant of the discriminator to mitigate the exploding gradient problem and the mode collapse problem. To understand this procedure in detail let's go into a bit of math : The matrix or spectral norm is defined as follows :

$$\underbrace{\|A\|}_{\text{norm}} = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

But we also know from Singular Vector Decomposition (SVD) that a matrix A can be decomposed as :

$$A_{m\times n} = U_{m\times m} S_{m\times n} V^T_{n\times n}$$

such that ,

$$\begin{array}{cccc} & U & V & S \\ & \begin{pmatrix} | & & | \\ \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_m \\ | & & | \end{pmatrix} & \begin{pmatrix} | & & | \\ \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_n \\ | & & | \end{pmatrix} & \begin{pmatrix} \sqrt{\lambda_1} & & & & \\ & \sqrt{\lambda_2} & & & \\ & & \ddots & & \\ & & & \sqrt{\lambda_r} & \\ & & & & \ddots \; 0 \end{pmatrix} \\ \text{eigenvectors of} & AA^T & A^TA & \end{array}$$

U and V are chosen to be the orthogonal ($U^TU=I$ and $V^TV=I$ ) eigenvectors of $A^TA$ and

$A^TA$ respectively. Both AA and AA have the same positive eigenvalues. S contains the square root of these eigenvalues which is called singular values.

We will be using the property that *The spectral norm σ(A) is the maximum singular value of matrix A* . A short proof follows in the succeeding page :

Finding the matrix norm of *A* is the same as finding the maximum value of the Rayleigh quotient below.

$$\underbrace{\|A\|}_{\text{norm}} = \max_{x\neq 0}\frac{\|Ax\|}{\|x\|} \qquad\qquad \frac{\|Ax\|^2}{\|x\|^2} = \frac{x^TA^TAx}{x^Tx} = \overbrace{\frac{x^TSx}{x^Tx}}^{\text{Rayleigh quotient}}$$

Let's rewrite the Rayleigh quotient with the inner product.

$$\overbrace{R(x)}^{\text{Rayleigh quotient}} = \frac{\langle x, Sx\rangle}{\langle x, x\rangle} = \overbrace{\frac{\langle x, Q\Lambda Q^Tx\rangle}{\langle x, x\rangle}}^{\text{Assume S is symmetric}} = \overbrace{\frac{\langle Q^Tx, \Lambda Q^Tx\rangle}{\langle x, x\rangle}}^{\langle x, Ay\rangle = \langle A^Tx, y\rangle}$$

We substitute *x* with *Qx* above.

$$\overbrace{R(Qx) = \frac{\langle Q^TQx, \Lambda Q^TQx\rangle}{\langle Qx, Qx\rangle}}^{\text{substitute } x \text{ with } Qx} = \underbrace{\overbrace{\frac{\langle x, \Lambda x\rangle}{\langle x, x\rangle}}^{Q^TQ = I}}_{\|Qx\| = \|x\| \text{ as } Q \text{ is orthogonal}} = \frac{\lambda_1 x_1^2 + \dots + \lambda_n x_n^2}{x_1^2 + \dots + x_n^2} = \underbrace{\sum_i \overbrace{\lambda_i}^{\text{eigenvalue of } A^TA} \frac{x_i^2}{x_1^2 + \dots + x_n^2}}_{\text{weighted average of eigenvalues}}$$

So Rayleigh quotient is the weighted average of the eigenvalues of $A^TA$. Since a weighted average is smaller or equal its maximum value,

$$R(y) = \sum_i \lambda_i \frac{x_i^2}{x_1^2 + \dots + x_n^2} \leq max(\lambda_i)$$

We often reshuffle λ in descending order. So $\lambda_1$ holds the largest eigenvalue. The norm of *A* is, therefore, the root of the largest eigenvalue of $A^TA$ (the singular value $\sigma_1$ of *A*).

$$\frac{\|Ax\|^2}{\|x\|^2} = \frac{x^TA^TAx}{x^Tx} \leq \lambda_1 \quad \text{(arrange } \lambda_i \text{ to have the largest eigenvalue)}$$

$$\|A\| = \max\frac{\|Ax\|}{\|x\|} = \sqrt{\lambda_1} = \sigma_1$$

For a symmetric matrix *S*,

$$\frac{x^TSx}{x^Tx} \leq \lambda_1$$

has the largest value when *x* is the eigenvector with eigenvalue $\lambda_1$

Let's construct a deep network with the following formalization :

$$f(\boldsymbol{x}, \theta) = W^{L+1} \underbrace{a_L}_{\text{activation function}} (W^L (a_{L-1} (\underbrace{W^{L-1}}_{\text{weights for each layer}} (\dots a_1 (W^1 \boldsymbol{x}) \dots))))$$

For each layer L, it outputs y = Ax followed by an activation function a, say ReLU. The Lipschitz constant (or Lipschitz norm) of function g can be computed with its derivative as:

$$\|g\|_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h}))$$

The Lipschitz constant for g = Ax is precisely the spectral norm of W.

$$\|g\|_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h})) = \sup_{\boldsymbol{h}} \sigma(W) = \sigma(W)$$

$$\nabla g(\boldsymbol{h}) = \nabla W\boldsymbol{h} = W$$

The Lipschitz constant for RELU is 1 as its maximum slope equals one. Therefore the Lipschitz constant for the whole deep network f is

$$\begin{aligned} \|f\|_{\text{Lip}} \leq & \|(\boldsymbol{h}_L \mapsto W^{L+1}\boldsymbol{h}_L)\|_{\text{Lip}} \cdot \|a_L\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_{L-1} \mapsto W^L \boldsymbol{h}_{L-1})\|_{\text{Lip}} \\ & \cdots \|a_1\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_0 \mapsto W^1 \boldsymbol{h}_0)\|_{\text{Lip}} = \prod_{l=1}^{L+1} \|(\boldsymbol{h}_{l-1} \mapsto W^l \boldsymbol{h}_{l-1})\|_{\text{Lip}} = \prod_{l=1}^{L+1} \sigma(W^l). \end{aligned}$$

Spectral normalization normalizes the weight for each layer with the spectral norm σ(W) such that the Lipschitz constant for each layer as well as the whole network equals one.Thus , we have the following :

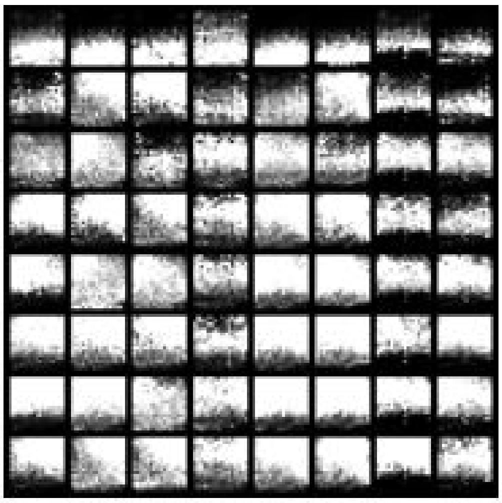
$$\bar{W}_{\text{SN}}(W) := W/\sigma(W)$$

$$\sigma\left(\bar{W}_{\text{SN}}(W)\right) = 1$$

$$\|f\|_{\text{Lip}} = 1$$

With Spectral Normalization, we can re-normalize the weight whenever it is updated. This creates a network that mitigates gradient explosion problems. The SAGAN architecture thus involves using this method to improve training

Here we show the visualization of the attention map :



## 5 Implementation

In this section , we present the details of our implementation . The dataset chosen is the **CIFAR 10** dataset . The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

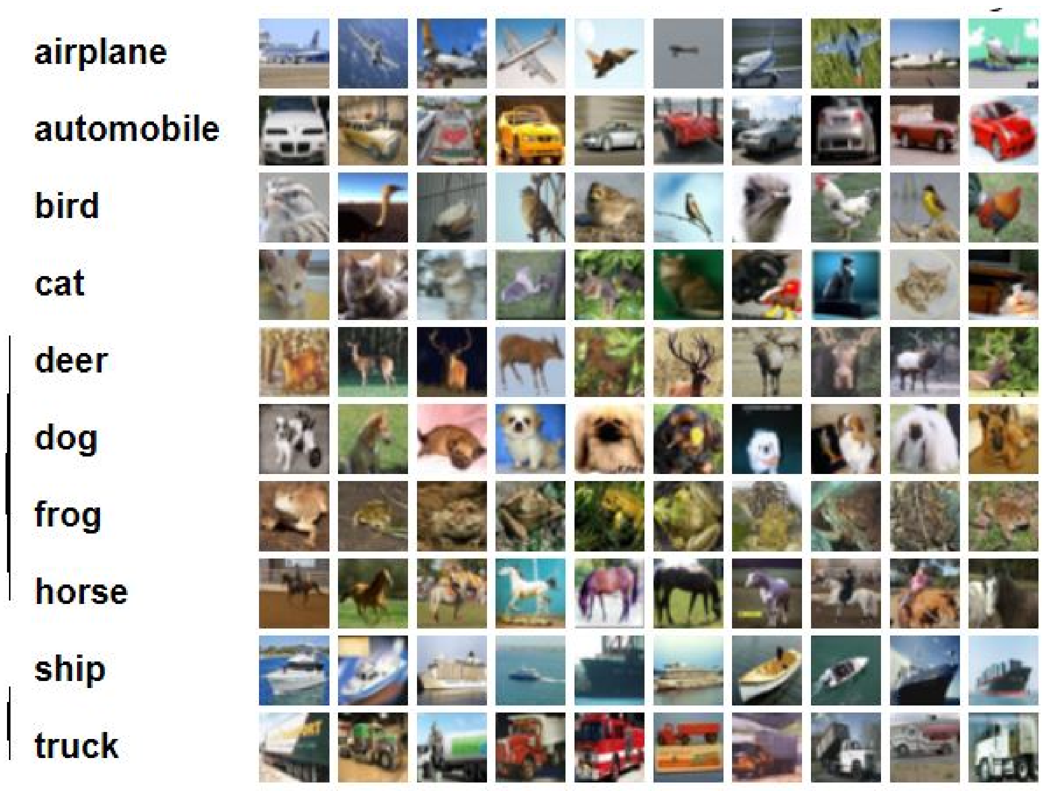
The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

We first resize images to **64x64** size to enable better feature extraction . Then apply **Normalisation**. Since our **implementation is in pytorch** we use the transforms utility to do this . Our entire implementation is trained on **GPU** based systems and we use the pytorch **visdom** utility for visualisation purposes

The latent vector is sampled from a 100 dimensional space for DCGAN and a batch size of 64 is used for training . We tried several latent dimensions , batch sizes and different optimizers ; based on the results obtained from our experiments while **increasing** the latent dimension space does lead to better results , however this correspondingly increases the cost of computation and the risk of over-fitting . A latent dimension of 100 provided a nice compromise and was suitable for training in the resources provided . Similarly while we observed increasingly better results with increase in batch size (SGD performs better as better approximations to actual gradients!) , the increasing memory utilization by the GPUs proved to be a detriment ; again , a batch size of 64 was giving good results . Finally we tried various learning rates with Adam() , RMSProp() and Adagrad() , eventually we fixed Adam() for both Generator and Discriminator with learning rates of 0.01 as it was giving stable convergence .

For the SAGAN due to unavailability of sufficient resources , we had to drastically



reduce both the size of the latent dimension and the batch size ; this led to a corresponding increase in training time and initial results weren't promising as expected . The increase in time could also have a component due to the costs involved in training the attention layers . However perhaps because of the Spectral Normalization procedure we observed better convergence . As an experiment , we tried modifying the **hinge loss** used in SAGANs with both the **vanilla min max optimization** and the **Wasserstein Loss** , surprisingly in our case we observed a marginal increase in convergence with the Wasserstein Loss and hence choose to go with it in the final implementation . The vanilla optimization however lowered convergence rate compared to the base hinge loss and gave poorer results in terms of quality of generated images . Once again we experimented with the latent dimension , batch size and the optimizers as well as we could given the constraints computational wise , to avoid running out of memory we were forced to consider the optimal values of 10 for latent dimension and 4 for batch size . We tried the TTUR(two time scale update rule) mentioned in the SAGAN paper(paper 2 in references) however didn't observe much of a appreciable difference instead our computational costs increased a lot . Thus we choose to stick with Adam() with lr =0.01 for both Generator and Discriminator models .

Here are some of the fake images generated by our two models :

Figure 8: Generated by DCGAN

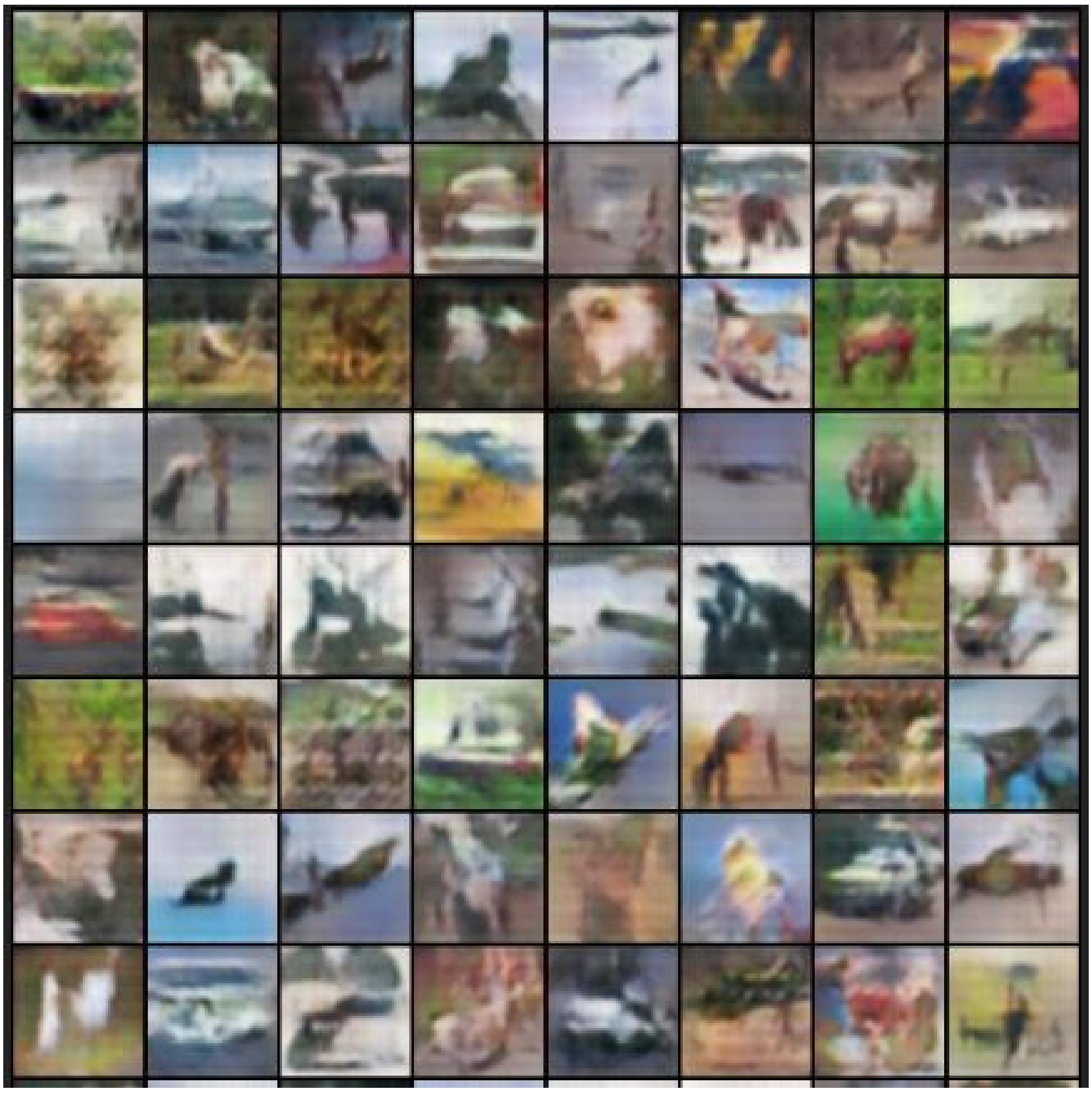
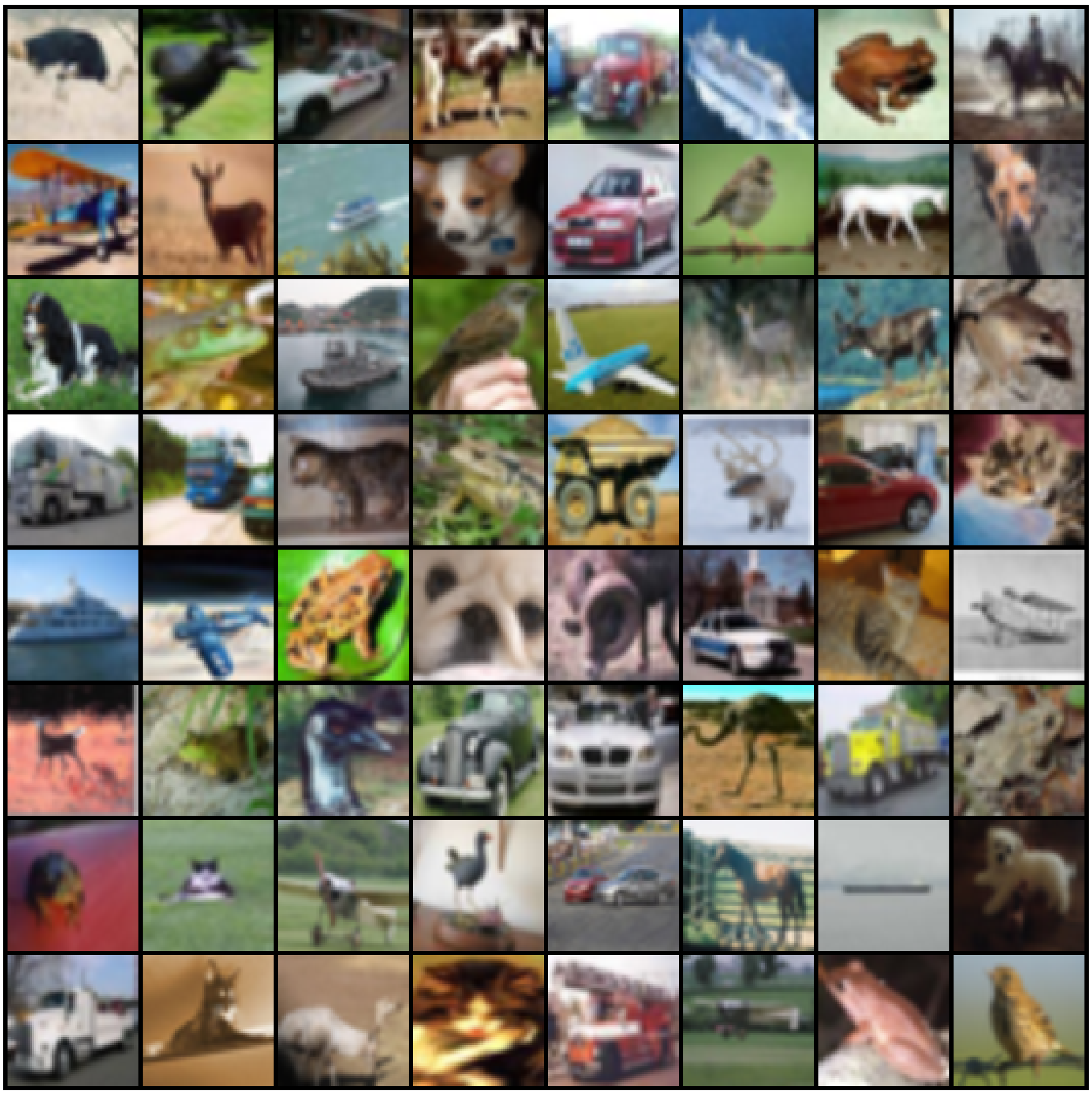


Figure 9: Generated by SAGAN



## 6 Results

In this section we mainly use the **Frechet Inception Distance** or FID Metric as a means of quantifying the results of the models .First a short discussion on the FID metric itself :

Usually whilst training GANs we only observe the 'loss' of the Generator with respect to the Discriminator and vice versa which tell us how well these two models are learning to dominate over their opponent . This metric however can be deceiving , often times losses can converge rapidly yet the Generated quality of images is relatively poor this happens when the Generator learns to exploit a loophole and learns to circumvent the investigation of the Discriminator . In this case to keep ourselves aware of how well the actual generalisation is working we need a certain metric .

In this regard , the Frechet Inception Distance(FID) metric was proposed . The aim of this metric is to quantify 'distance' between two image distributions . Intuitively it is obvious that similar to other difference metrics such as KL Divergence if the intention is to quantify differences between distributions then the actual distribution or a good enough approximation to it must be obtained in some manner . We can think of the base level features extracted from the two images models by a sufficiently deep CNN to be a good enough approximate in this regard .

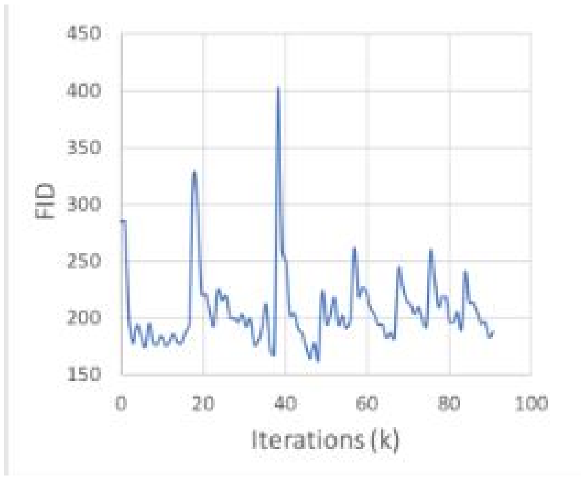
In FID, we use the Inception network to extract features from an intermediate layer. Then we model the data distribution for these features using a multivariate Gaussian distribution with mean μand co-variance Σ. The FID between the real images x and generated images g is computed as:

$$\texttt{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \mathrm{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}}),$$

Our FID implementation is adapted from the one presented at this site : machinelearningmastery.com . Our distributions are the CIFAR 10 dataset and the stored generated images during training and validation . We randomly sample images from both datasets and then use the **Keras pretrained Inception v3 model** to compute prediction vectors which are then used to compute means and covariances finally we apply the equation to obtain the FID score .

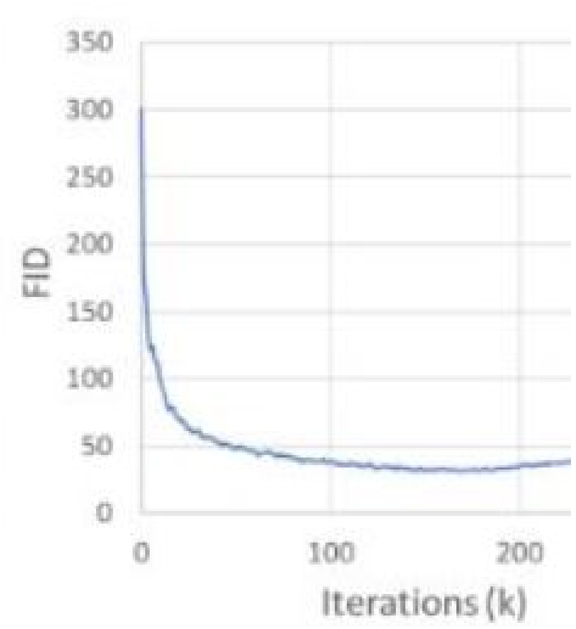
Here we present graphs showcasing the FID results :

Figure 10: FID curve for DCGAN



FID has been computed for every 100 epochs when total training was set for close to 100000 epochs

Figure 11: FID curve for SAGAN



FID has been computed for every 40 epochs when total training was set for close to 100000 epochs

As can be seen here from the FID curves it is apparent that SAGAN shows better performance as compared to DCGAN which is not quite so obvious based on visual inspection of the generated images .

## References

*Wasserstein GAN* Martin Arjovsky, Soumith Chintala, and Leon Bottou.
*Self-Attention Generative Adversarial Networks* Han Zhang , Ian Goodfellow , Dimitis Metaxas , Augustus Odena
*Are GANs Created Equal? A Large-Scale Study* Mario Lucic , Karol Kurach , Marcin Michalski , Olivier Bousquet , Sylvain Gelly
FID implementation adapted from : `https://machinelearningmastery.com/how-to-implement-`