Project 1 Report **Koustubh Vijay Kulkarni**
CSE 676
Prof. Sargur N. Srihari UBID: Kkulkarn
TA: Mohammad Abuzar Shaikh, Marissa Dominijanni Person Number: 50288207

Objective

The objective of this project is to learn and implement Generative Adversarial Networks (GANs) for generating samples. We will learn and implement two types of GANs- 1. Deep Convolution GAN (DCGAN) 2.Self-Attention GAN (SA-GAN). The dataset we are using is CIFAR-10 dataset.

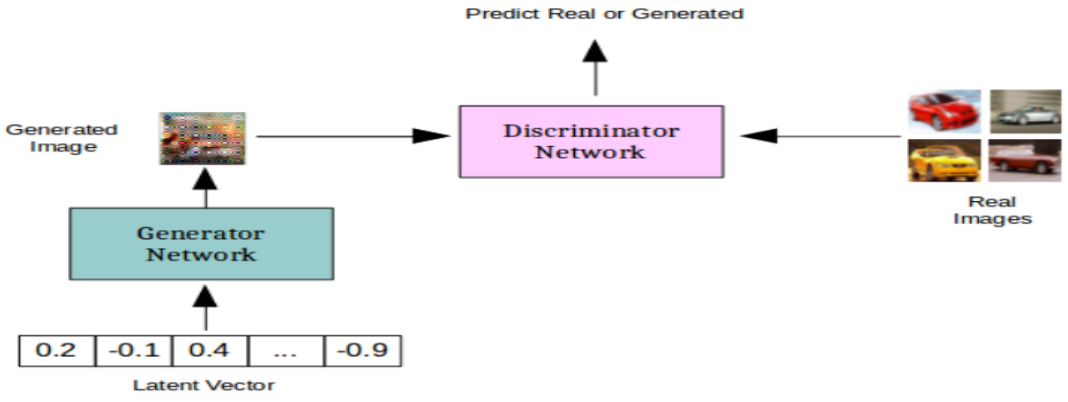
Task

Our task is to implement 2 types of GANs viz. DCGAN and SA-GAN for task of image generation. We are going to work with CIFAR-10 dataset in order to train our GAN model and try to generate images similar to the images in CIFAR-10 dataset. Here are some details about dataset:-

1. CIFAR-10 Dataset:- The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. Training set contains 50 thousand images and test set contains 10 thousand images.

What is GAN?

GANs are used to create data from scratch. Our deep learning model learns from the data and tries to produce similar data. What this means is, given a set of training data, GANs can learn to estimate the underlying probability distribution of the data.
GAN composes of two deep networks, the generator, and the discriminator which compete with each other making each other stronger at the same time. Given a training set X, the Generator Network, G(x), takes as input a random noise and tries to produce images similar to those in the training set X. A Discriminator network, D(x), is a binary classifier that tries to distinguish between the real images according the training set X and the fake images generated by the Generator. Generator's job is to fool the discriminator and discriminator's job is to get better at distinguishing images coming from generator and real images. Overview.PNG



GAN Overview

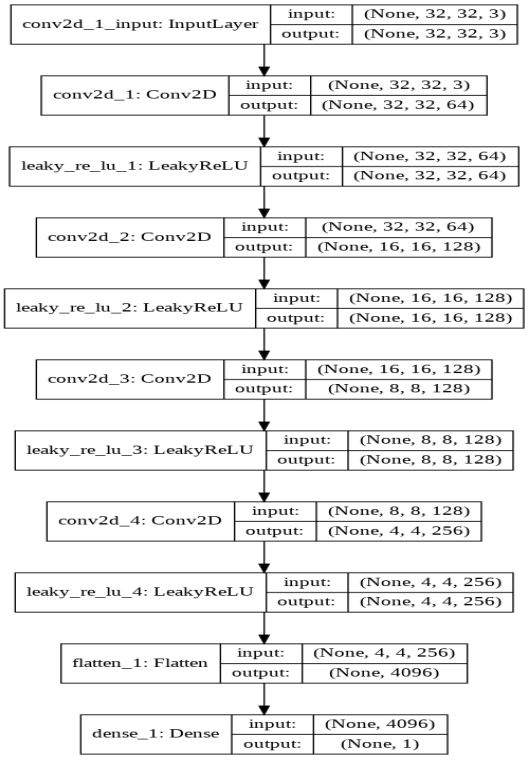
Deep Convolution GAN

One of the variants of GAN in Deep Convolutional GANs (DCGANs) in which G and D are both based on deep convolution neural network. It mainly composes of convolution layers without max pooling. It uses convolutional stride and transposed convolution for the down-sampling and the up-sampling.

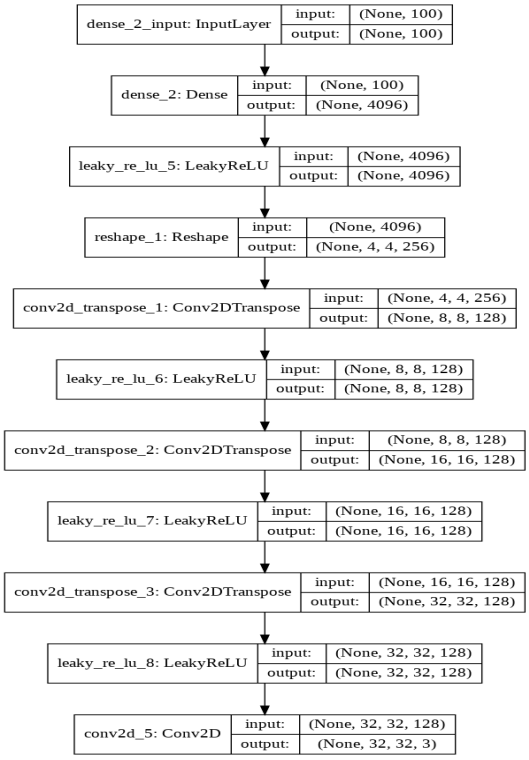
0.1 Architecture

0.1.1 Discriminator Network

Input: CIFAR-10 Image with 3 RGB channels and 32×32 pixels in size.
Output: Binary classification, 0 For fake, 1 For Real.
Our discriminator network has a normal convolutional layer followed by three convolutional layers which has a stride of 2×2. This stride is used to downsample the input image. The model has no pooling layers and a single node in the output layer. Output node has the **sigmoid** activation function to predict whether the input is real or fake. The model is trained to minimize the **'binary cross entropy'** loss function which is used for binary classification.



Discriminator Network



Generator Network

0.1.2 Generator Network

Inputs: 100-element vector of Gaussian random numbers(Noise).
Outputs: 2D square color image 3 RGB channels of 32 × 32 pixels with pixel values in the range of [-1,1].
Generator model requires that we transform a vector from the latent space with, 100 dimensions to a 2D array with 32 × 32 × 3. We need multiple parallel activation maps with different

interpretations of the input. First Dense layer needs enough nodes for multiple versions of our output image hence I have taken 256 channels. Then I used 'Conv2DTranspose' layers for deconvolution of this input.The Conv2DTranspose layer has a stride of (2×2). Which will double the height and width of the input feature maps. Sometimes in upsampling a checkerboard pattern may happen, so to avoid that is was suggested that use kernel size as multiple of stride. Hence I used kernel of size 4. The output layer of the model is a Conv2D with 3 filters for 3 RGB channels and a kernel size of 3×3 and 'same' padding, designed to create a single feature map with dimensions at 32 × 32 × 3 pixels.

0.1.3 Combined Model

In order to create GAN we need to combine Discriminator network and Generator Network.GAN model can be defined as stack of the generator and discriminator. I have used previously created two networks and combined them to create GAN model. Main thing here is, I have set discriminator model in this model as **Not Trainable**. So when gradients will be updated it will only be updated for Generator Model.

0.1.4 Hyperparameters/Model Parameters

- **Activation Function:-**I am using LeakyReLU instead of ReLU. Leaky ReLUs are very popular because they help the gradients flow easier through the architecture.In a regular ReLU, negative values are truncated to 0. This blocks the gradients to flow through the network. Instead of the function being zero, leaky RELUs allow a small negative value to pass through the network.

- **Optimizer:-** I am using 'Adam' optimization function which will try to minimize binary-crossentropy loss in discriminator as well as in generator.

- **Learning Rate:-** Learning rate affects the convergence of algorithm. I have kept learning rate of 0.0002 for both generator and discriminator. There is also Two Time-Scale Update Rule which suggests keeping discriminator learning rate 4 times high as generator's to achieve local Nash Equilibrium. So, I tried doing that and I kept learning rate of generator as 0.0001 and discriminator as 0.0004. But it produced blurry images after 20000 iterations. So, I decided to use same learning rate in generator and in discriminator.

- **Beta_1:-** This Beta_1 parameter in Adam optimizer represents, the exponential decay rate for the first momentum estimates. Best practice suggests that we should keep it as close to 1 as possible but I got better results when I kept it as 0.5.

- **No. of Iterations:-** I am training my model for 100000 iterations. As, I am using Google Colab GPU for my project, I was facing issues with iterations more than this. But I have observed good results with 10000 iterations.

0.2 How training works in DCGAN?

Data Normalization
Firstly, it is good practice to normalize the training data. This is done because it helps in faster convergence.Also, the generator model will generate images with pixel values in the range [-1,1] as it will use the **tanh** activation function. So, it is good practice to sale our real data images to same scale.

We firstly feed 64 real images to the discriminator and we will tell the discriminator model

it comes from real class and will give it label as 1.Then we will get 64 images generated by generator and feed this to Discriminator network and will give class label as 0, indicating it is a fake sample.
Then we will generate 128 images from generator and will feed this to Discriminator network. This time, instead of marking them as Fake (Class 0) we will mark them as Real(Class 1). We do this because, we want the discriminator to think that the samples output by the generator are real, not fake. Then discriminator will classify the generated samples as Fake (class 0). The back-propagation process will see this as a large error and will update the model weights-weights in the generator, as discriminator is not getting trained in this phase- to correct for this error, in turn making the generator better at generating good fake samples.

0.3 DCGAN Evaluation Metrics

There are various ways to evaluate GANs. I have used two ways in this project.

1. Inception Score:-IS uses quality of the generated images, and their diversity to measure the performance of the GAN.

$$\mathbf{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_o} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right),$$

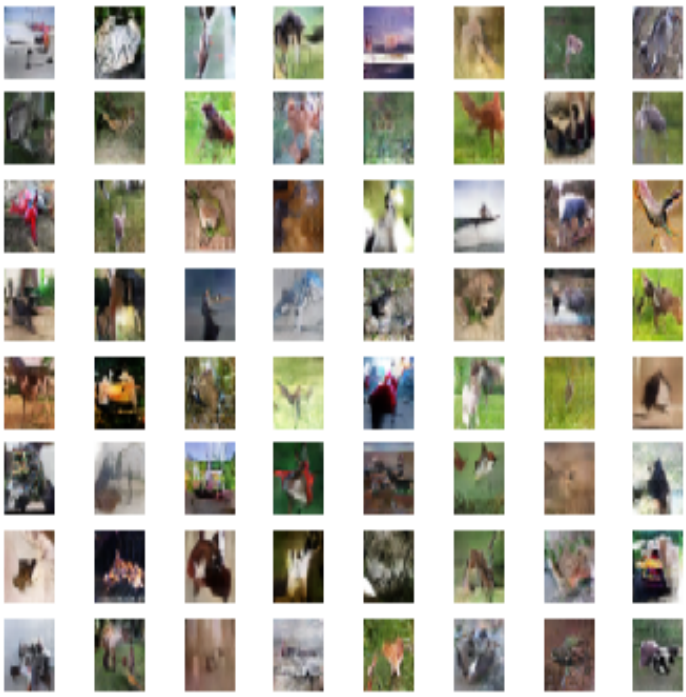
   One shortcoming of IS is that it can misrepresent the performance if model only generates one image per class. p(y) will still be uniform even though the diversity is low.

2. Fréchet Inception Distance (FID)- FID is more robust to noise than IS.Lower FID values mean better image quality and diversity.The score summarizes how similar the two groups are in terms of statistics on features of the raw images calculated using the inception v3 model used for image classification.

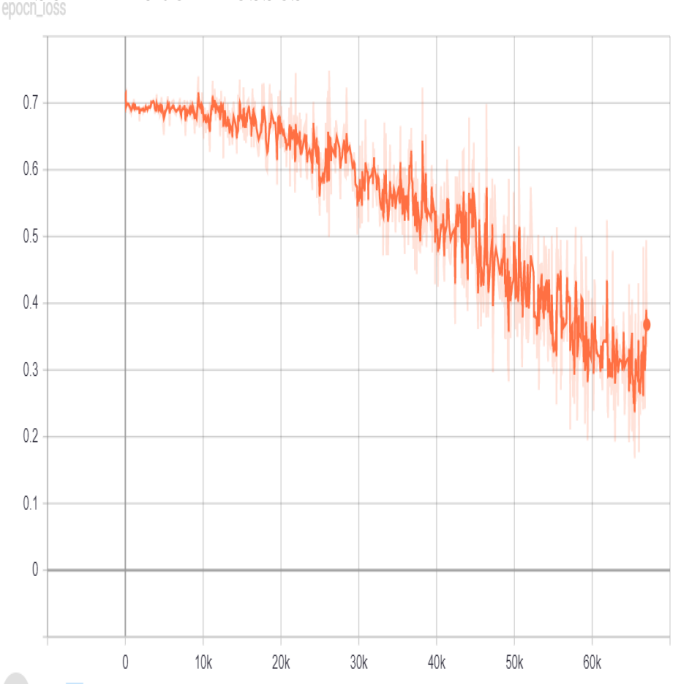
$$\texttt{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}}),$$

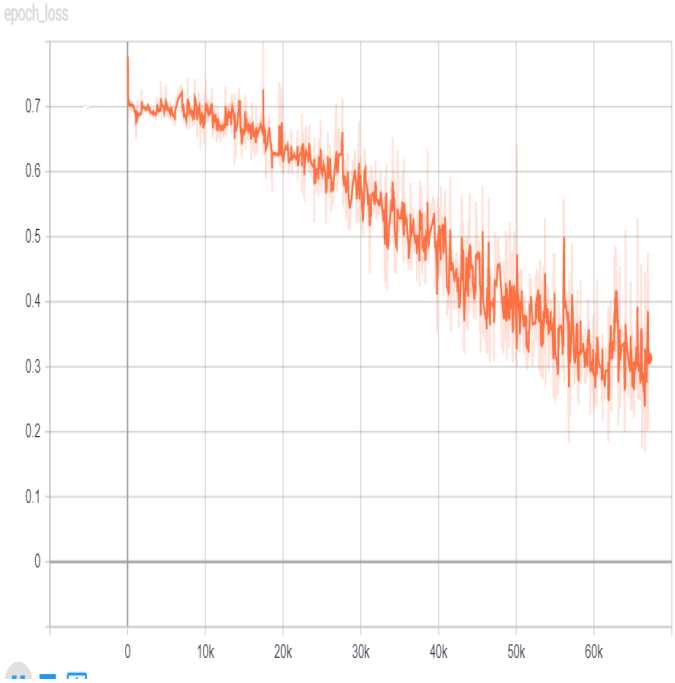
0.4 DCGAN Results

Here is the image grid generated by my DCGAN model:-

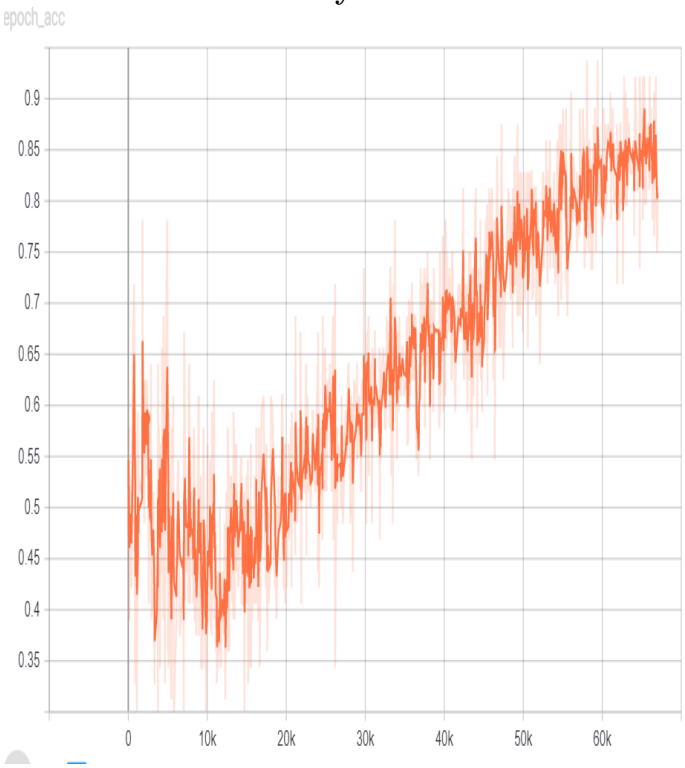


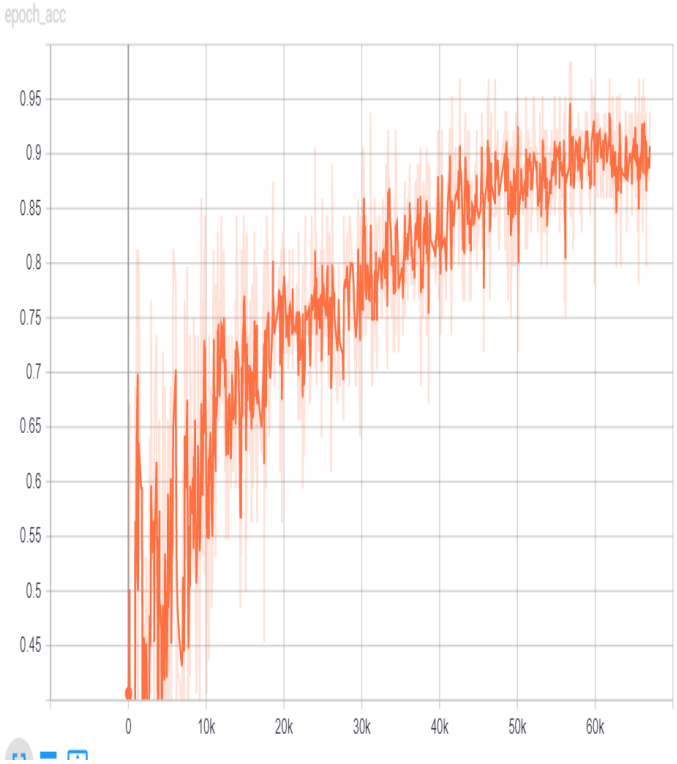
Discriminator Losses:-



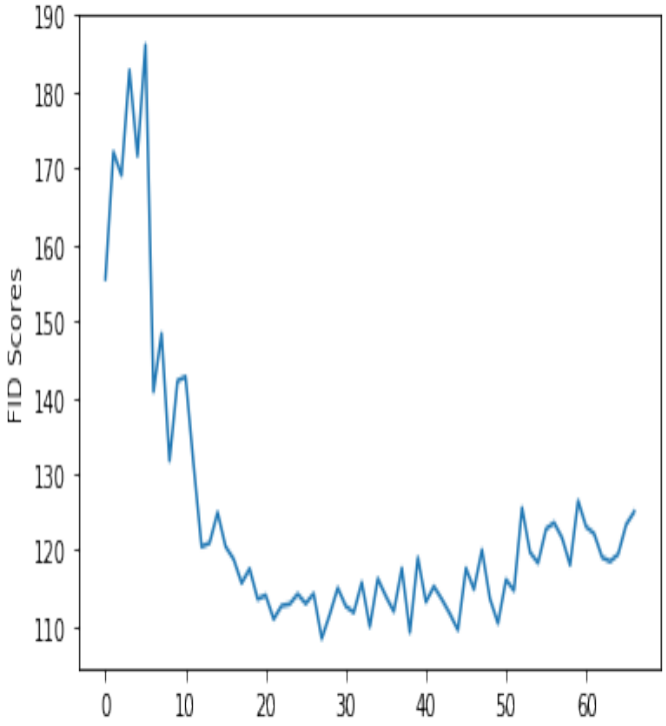


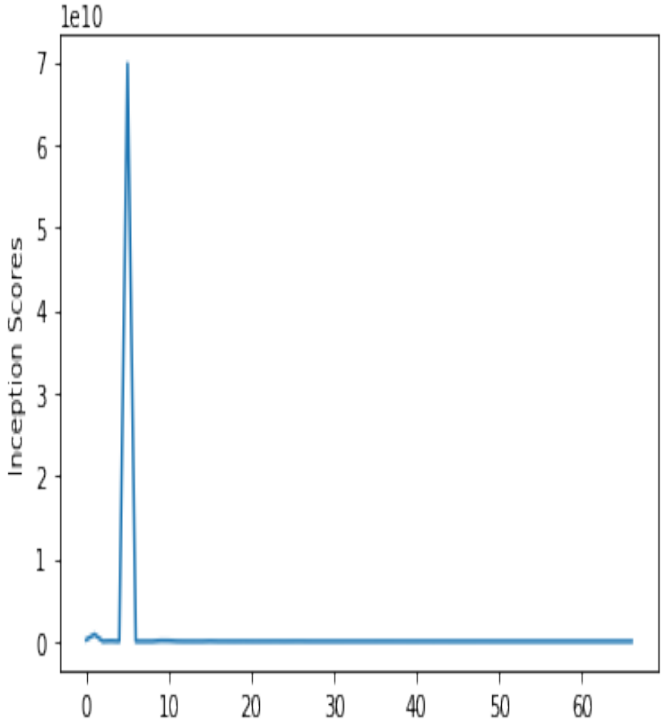
Discriminator Accuracy:-





FID and Inception Scores:-





Mean FID Score:- 120.23

Self Attention GAN

There are many Limitations with DCGAN. DCGANs exhibit limitations while modelling long term dependencies for image generation tasks.They rely heavily on convolution to model the dependencies across different image regions. But convolution operator has a local receptive field that means long ranged dependencies can only be processed after passing through several convolutional layers.
Even if we increase the number of convolution layers then it will increase the model complexity and it will be computationally inefficient.
The solutions to keeping computational efficiency and having a large receptive field at the same time is Self-Attention. It helps create a balance between efficiency and long-range dependencies by utilizing the attention mechanism.

0.5 What is Self-Attention?

We get our feature maps from the previous convolutional layer. We first pass the feature map through three 1×1 convolutions separately. We name the three filters f, g and, h. 'f' and 'g' multiply in such a way so that they create another vector of probabilities which decide how much of the 'h' to expose to the next layer. Then we multiply the final output by a learnable scale parameter **'y' (gamma)** and add back the input as a residual connection. This parameter is initially set to 0 and it will be learned as part of training.

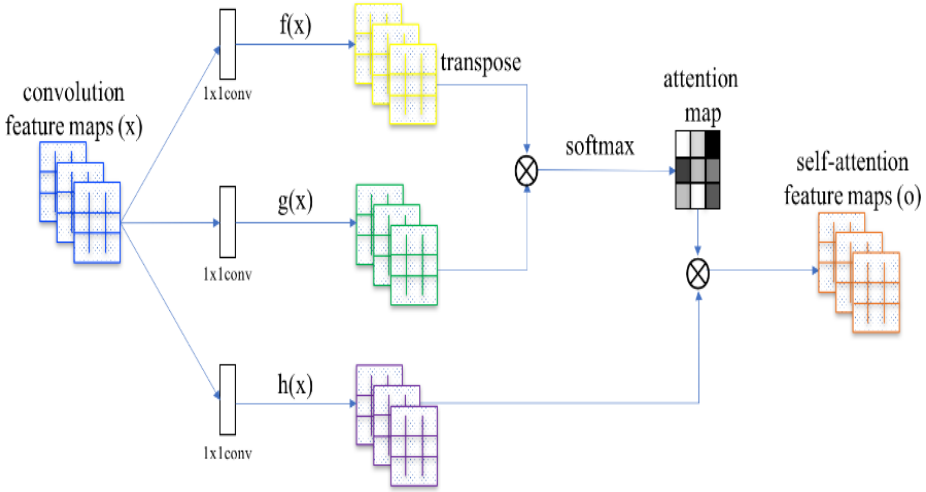


Figure 2: The proposed self-attention mechanism. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

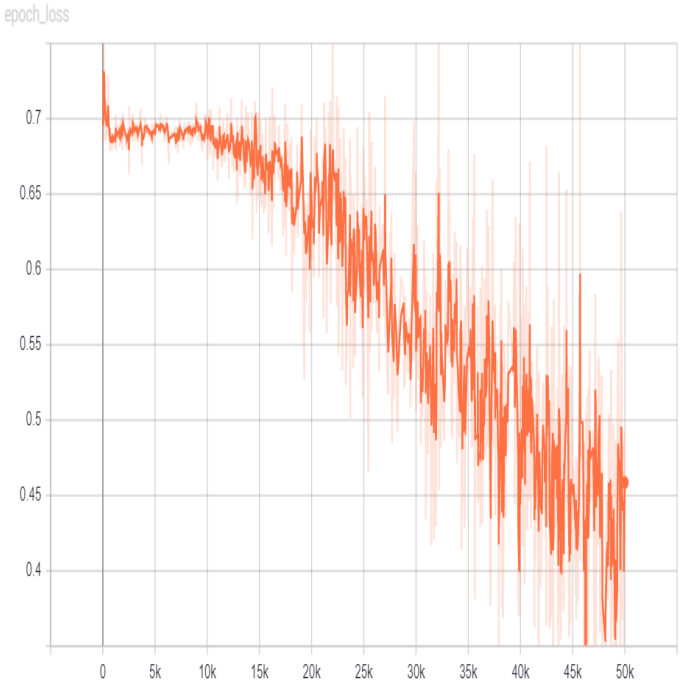
0.6 Architecture

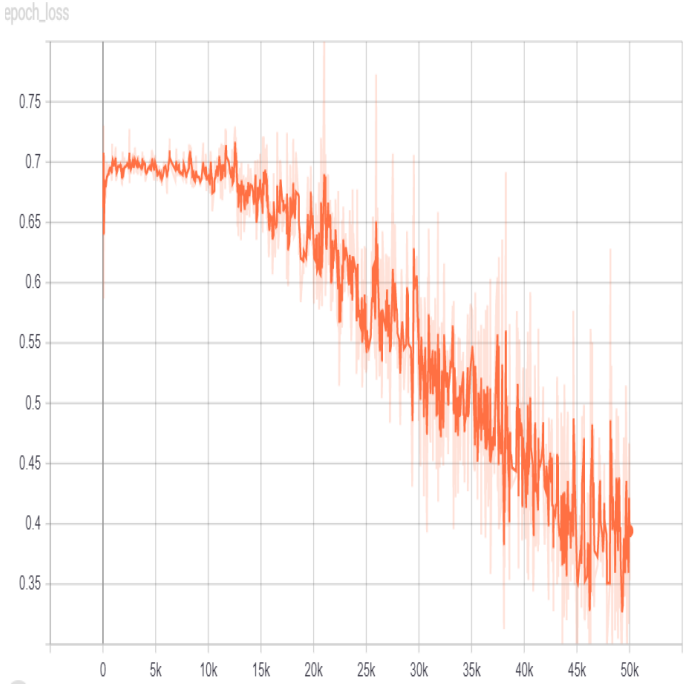
I have kept the same architecture as of my DCGAN and just added Self attention Layer after two deconvolution layers in generator and after 3 convolution layers in Discriminator.

0.7 SAGAN Results

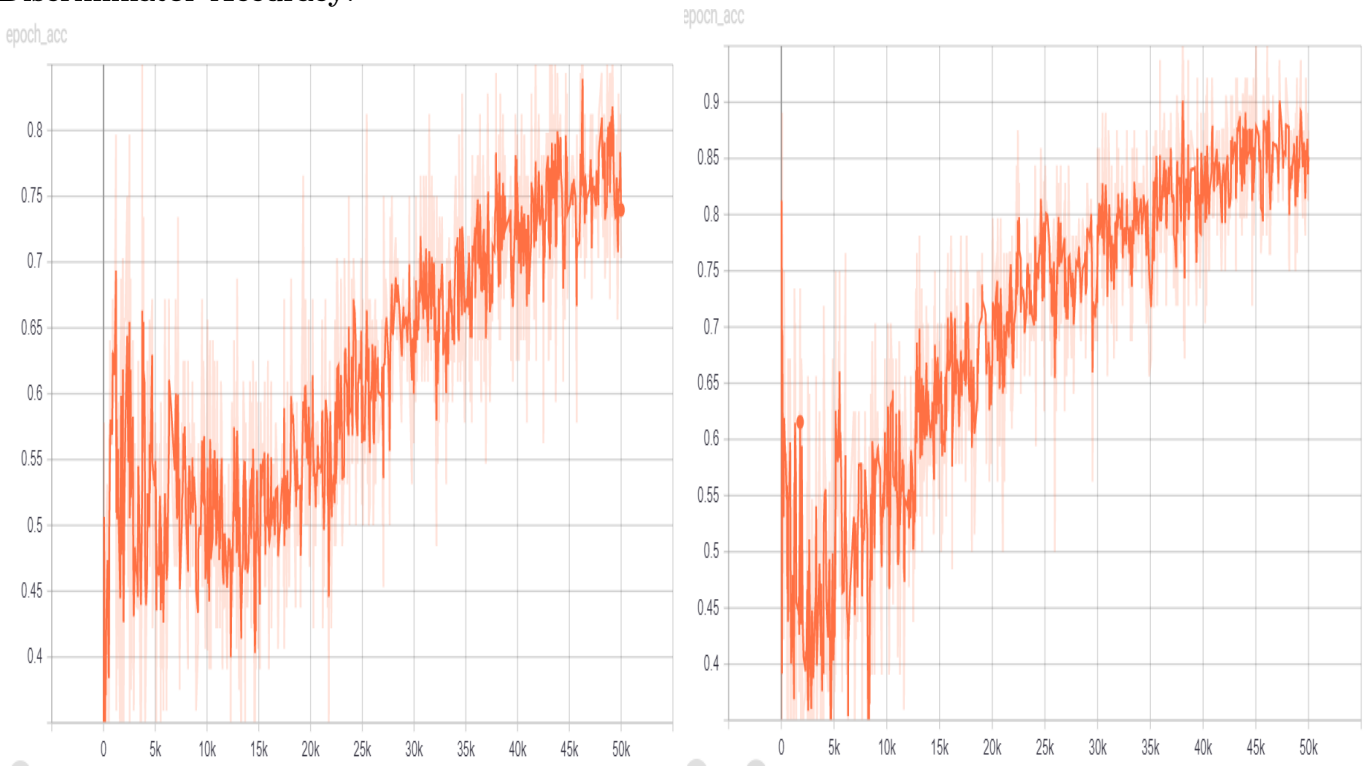
Here is the image grid generated by my SAGAN model:-

Discriminator Losses:-

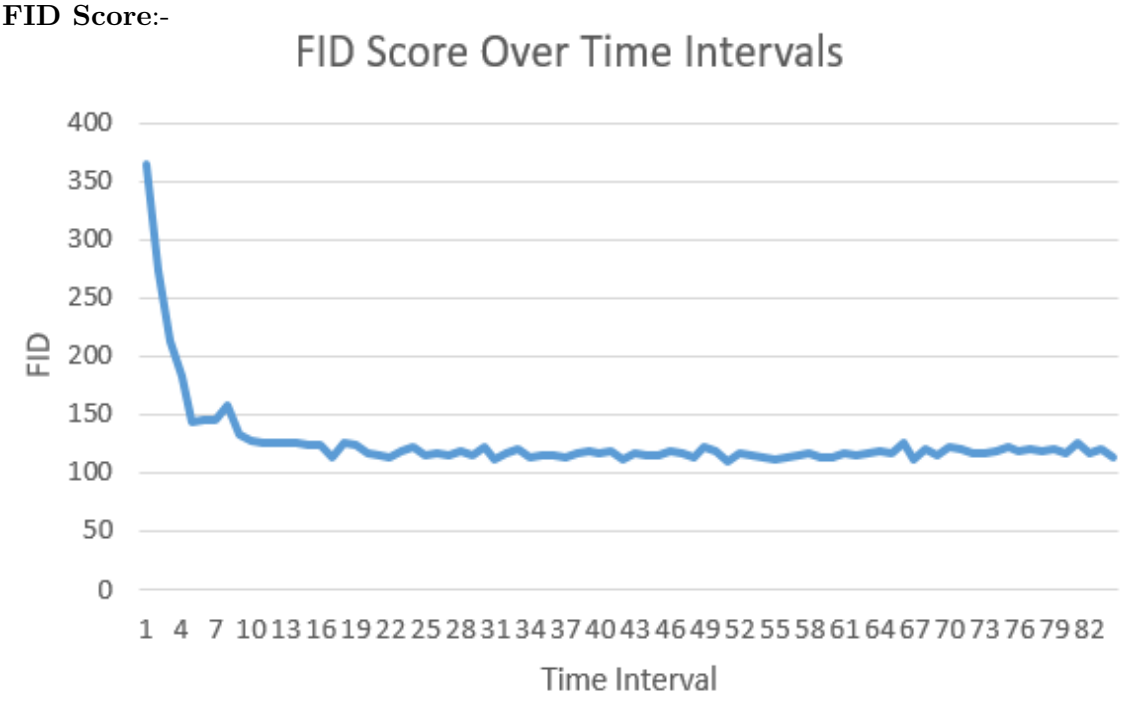




Discriminator Accuracy:-



FID Score:-



Mean FID Score:- 126.30

SAGAN With Wasserstein Loss Spectral Normalization

Why to use Wasserstein loss?

One of the disadvantage of DCGAN is the issue of Mode collapse.To alleviate this issue, GANs

are trained with Wasserstein Loss. The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution q to the data distribution p.

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

0.8 Changes done in SAGAN Model

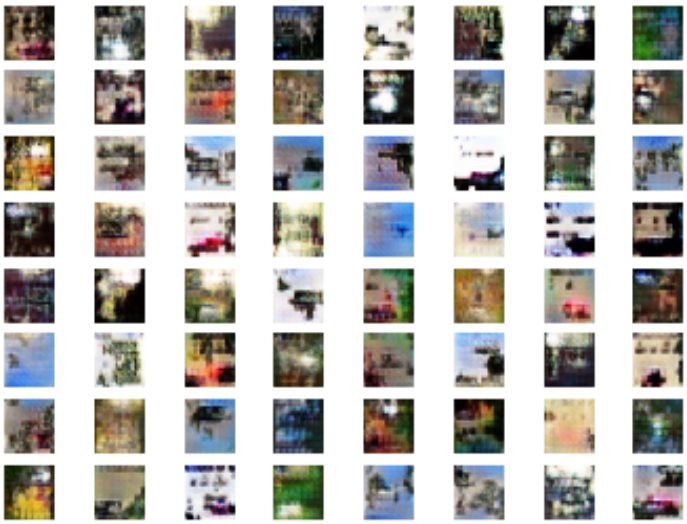
In order to implement WGAN with Self attention I had to make couple of changes in my model.

1. Using a linear activation function in the output layer of the discriminator model instead of sigmoid.
2. Implemented weight clipping and constrained discriminator to restrict it's weights between [-0.1,0.1]. WGAN is sensitive to weight clipping parameter 'c', I kept it as 0.01 initially and then I faced **Vanishing Gradient** problem.
3. Updating the discriminator model more times than the generator each iteration. I am updating discriminator 5 times and generator once per iteration.
4. Using the RMSProp version of gradient descent with small learning rate and no momentum.

Why to use Spectral Normalization?
Spectral normalization is used as a way to stabilize the training of the discriminator network.Basically SN precludes the transformation of each layer from becoming sensitive in one direction.

0.9 SAWGAN Results

I could not run SAWGAN for 100k iterations. But I ran it for 10k iterations and here is the image I got after 10k iterations.



If I could have ran this till 100k iterations then I think results would have been better.

Learnings and Conclusion

I observed that performance of Self-Attention DCGAN is better than the DCGAN. Though I could only run the models till 100k iterations, some of the images were very good for SAGAN and DCGAN.

Initially I faced issue with SAWGAN and I faced issue of vanishing gradients and my model was not working properly. Very late I made some changes and that model showed some kind of good output in the beginning. Overall, I think,if that model would have worked then results of that model would have been more promising.
There were couple of learnings for me in this projects that I like to share here:-

- Learned about various types of GAN and how to write code to implement them in Keras.
- Learned how to write your own layer in Keras.
- Learned how to implement Kernel_constraints in Keras for the implementation of weight clipping.
- Learned about Self-Attention, Spectral Normalization, Batch Normalization.

References

[1] Class and Recitation notes.

[2] https://medium.com/towards-artificial-intelligence/techniques-in-self-attention-generative-adversarial-networks-22f735b22dfb

[3] https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

[4] https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd529764aa9

[5] https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/

[6] https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/

[7] https://medium.com/@jonathanhui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

[8] https://github.com/sthalles/blog-resources/tree/master/sagan

[9] https://github.com/IShengFang/SpectralNormalizationKeras