DCGAN and SAGAN on CIFAR-10 dataset

Name: Arun kumar S

Department: UG

S.R.No: 15656

**********Before running mani.py **********

Please download the following files from their sharable links before running the main.py.

DCGAN_discriminator_400.h5:

https://drive.google.com/file/d/1ATD1T8mi_XACity6AqE4KXMaJHU16oX6/view?usp=sharing

DCGAN_generator_400.h5:

https://drive.google.com/file/d/1R7wu_SDsy7FuO6gVqqz7tB9rOM7k2ObT/view?usp=sharing

DCGAN_GENERATOR.h5:

https://drive.google.com/file/d/17ykRMMHZPDVjwMN5_hK0uQMHIuOkhWEy/view?usp=sharing

DCGAN_DISCRIMINATOR.h5:

https://drive.google.com/file/d/1EoMfmLhKHKrF7qJVv7CvetD5kmLbh7PF/view?usp=sharing

SAGAN_generator_300.h5:

https://drive.google.com/file/d/1XkeDbOrApUDnN_kN2tLFKiB6X0Rsgpss/view?usp=sharing

SAGAN_discriminator_300.h5:

https://drive.google.com/file/d/16Z---cA6fsRvyxNZJlCl-gLolEHHP-A_/view?usp=sharing

SAGAN_GENERATOR.h5:

https://drive.google.com/file/d/1VPokEZOoXeoDcrruP7gm_Asl_ZWoT066/view?usp=sharing

SAGAN_DISCRIMINATOR.h5:

https://drive.google.com/file/d/1lW7kXXlfwl5Pr_8N1gAaBJ9y9G4ICSER/view?usp=sharing

**********Before running main.py**********

I have implemented this project using keras.

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. We have used the training set to train the GANs and test images for the evaluation of the model, i.e to calculate the FID score.

Spectral Normalization: GAN is vulnerable to mode collapse and training instability. 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. In this project, in both the models, we have applied spectral normalization to both generator and Discriminator. We have used existing implementations of the spectral normalization.

Wasserstein loss : When the generated distribution in GAN and the real distribution are disjoint, the gradient of the cost function will compose of areas of vanished and exploding gradients. WGAN claims that the Wasserstein distance will be a better cost function since it has a smoother gradient for better learning. WGAN applies a simple clipping to enforce the 1-Lipschitz constraint. In this project, we have used Wasserstein loss always in the SAGAN. In DCGAN, we have tried different loss functions like binary_crossentropy and it turned out that Wasserstein loss is the best. So, I have used Wasserstein loss in both generator and discriminator in DCGAN and SAGAN.

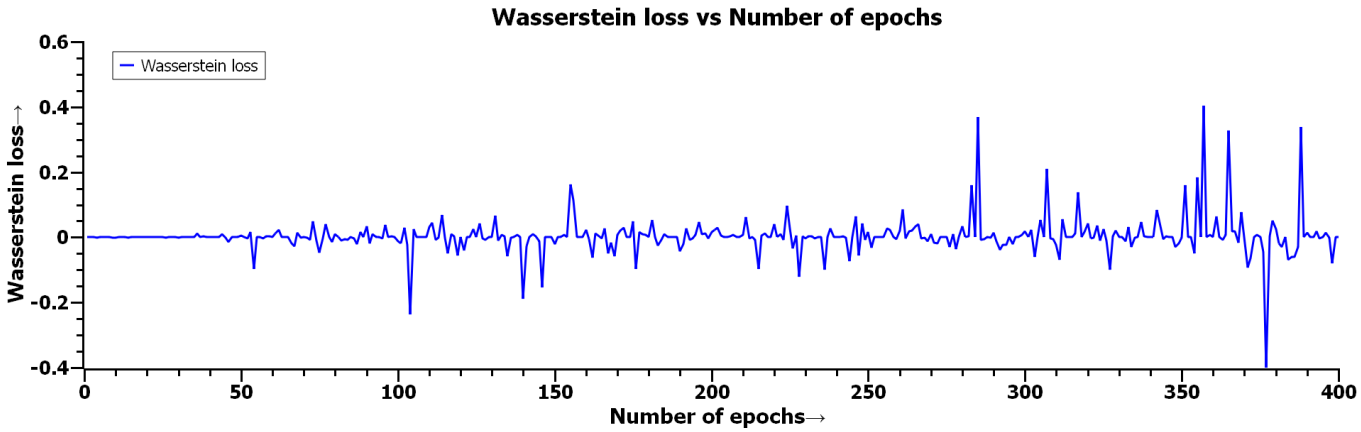
Self-attention module: Convolution operator has a local receptive field and thus it cannot capture long range dependencies in the image. We shall need larger number of convolutional layers to capture the long-range dependency and this will affect the computational efficiency. So, we have used self-attention to capture these long-range dependencies. Armed with self-attention, the generator can draw images in which fine details at every location are carefully coordinated with fine details in distant portions of the image. I have implemented the self-attention module in keras. I have just tried to mimic the mechanism given in the SAGAN paper.

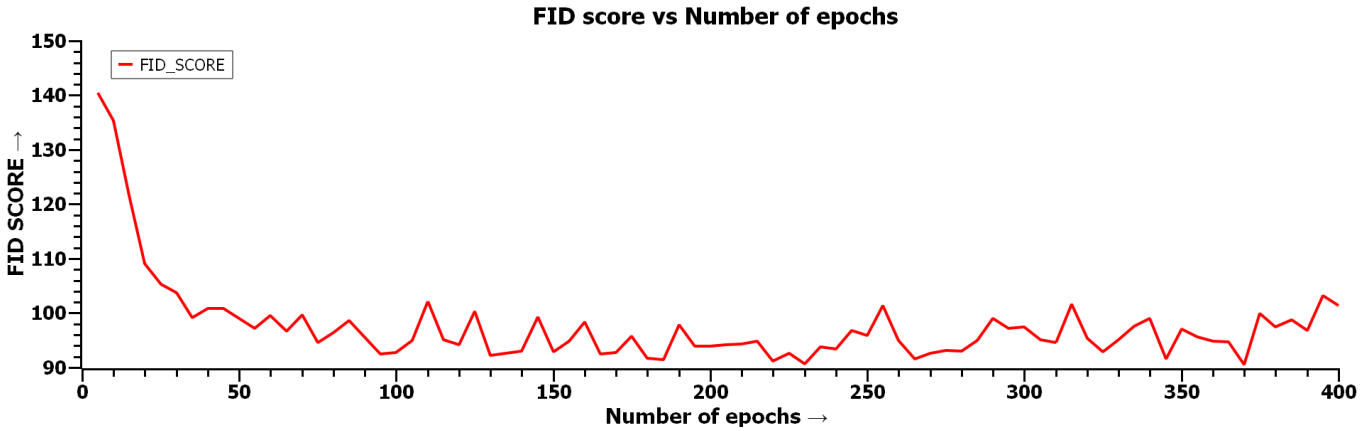
TTUR(Two Time Scale Update Rule): To address slow learning in regularized discriminators, we can use different learning rates for generator and discriminator. It is suggested in deep learning papers to keep learning rate of discriminator 4 or 5 times the learning rate used in the generator. I have kept the learning rate of Discriminator 4 times the learning rate of the Generator in most of my architectures. But when experimented, I saw that keeping the learning rate of discriminator 5 times that of the generator gave better results. Thus, my final models are having the learning rate of discriminator 5 times that of the generator.

DCGAN :

I have implemented in keras. First, I have loaded cifar-10 dataset and explored the dataset. Now, we need to implement layers with Spectral normalization. I have used existing implementation from internet for this. Now we need to build our Generator. Random noise will be the input. Keeping the input dimension of the form of power of 2 will be computationally favourable. So, I have tried different random input noise of dimensions(16,3264,128,256,512,etc). Output of the generator needs to be of the shape

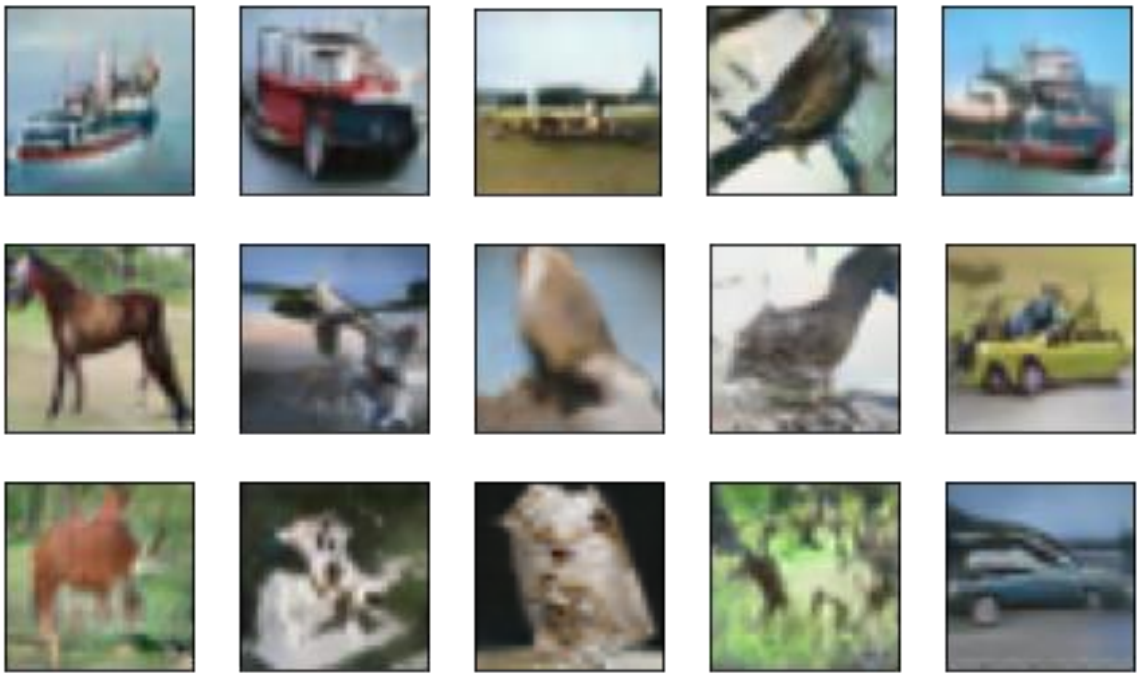
(32,32,3). I have used Deconvolutional or ConvTranspose layers to increase the dimension and bring it up to 32 X 32. I have put batch normalization layer after deconvolutional layers. In some architectures, I tried to use LeakyReLU activation function on the generator and did not use in others. Totally, I wanted to try different input noises, different ways of bringing the output to (32,32,3). I have recorded all the different architectures that I have tried in the word document Architectures.docx that I have uploaded to github in this project .
I observed one interesting result. If we keep the dimensions of the convoluted output same as the input and just vary the number of channels, then the performance is deteriorated. Even with one such layer, performance decreased by a very large amount. Whether in Generator or in discriminator, it is preferred not to have such layers which only vary the number of channels keeping the other dimensions fixed. So, in my final models, only the last layer of generator is the one in which only the number of channels is varied to bring the required output shape.
So, keeping the above result in mind, for different input noises, I have implemented the generator with different number of deconvolutional layers to obtain the output of shape (32,32,3). I found that with input noise having dimension 32, highest FID scores are obtained. So, I have used the same in the final model.
In some architectures, I have not used any activation function for the generator. In others, I have added the LeakyReLu activation function on the generator. I have obtained better FID scores with the activation function. My final model thus has the LeakyReLU activation function in the generator layers.
Discriminator will have the input of shape (32,32,3). In most of the architectures, I have tried to reduce the dimensions symmetrically with respect to the generator and give the final output which is binary. The output activation function is tanh and it will be near to 1 if the image is real and -1 if the image is fake. Discriminator has convolutional layers with spectral normalization. I have used existing implementations for spectral normalization. I have used LeakyReLU activation function following batch normalization after each convolutional layer. I have followed the standard ways in which the generator and discriminator are implemented in deep convolutional GANs. We have very few hyperparameters to tune. Learning rates for the optimizer, momentum for the batch normalization, etc were giving good results with their standard values. So, for such hyperparameters, I have used the standard values usually used in deep learning models.
I have tuned the batch size and found that with batch size of 32, results are better. Also, as explained earlier, ratio of learning rate of discriminator to learning rate of generator is kept 5. All the things said can be seen in Architectures.docx that I have attached for this project along with the FID scores obtained in every case.
I have tried different architectures for 30 epochs. I took the best model and run it over 400 epochs. I have saved the model at 370 epochs that gave the best FID score. It can be seen that there is no continuous improvement in the FID score with epochs and that FID score fluctuated with epochs without significant improvement even after very small number of epochs.
I have also plotted the Wasserstein loss and FID scores of the final DCGAN model with epochs below.


Wasserstein loss vs Number of epochs
Wasserstein loss
Wasserstein loss →
0.6
0.4
0.2
0
-0.2
-0.4
0
50
100
150
200
250
300
350
400
Number of epochs→

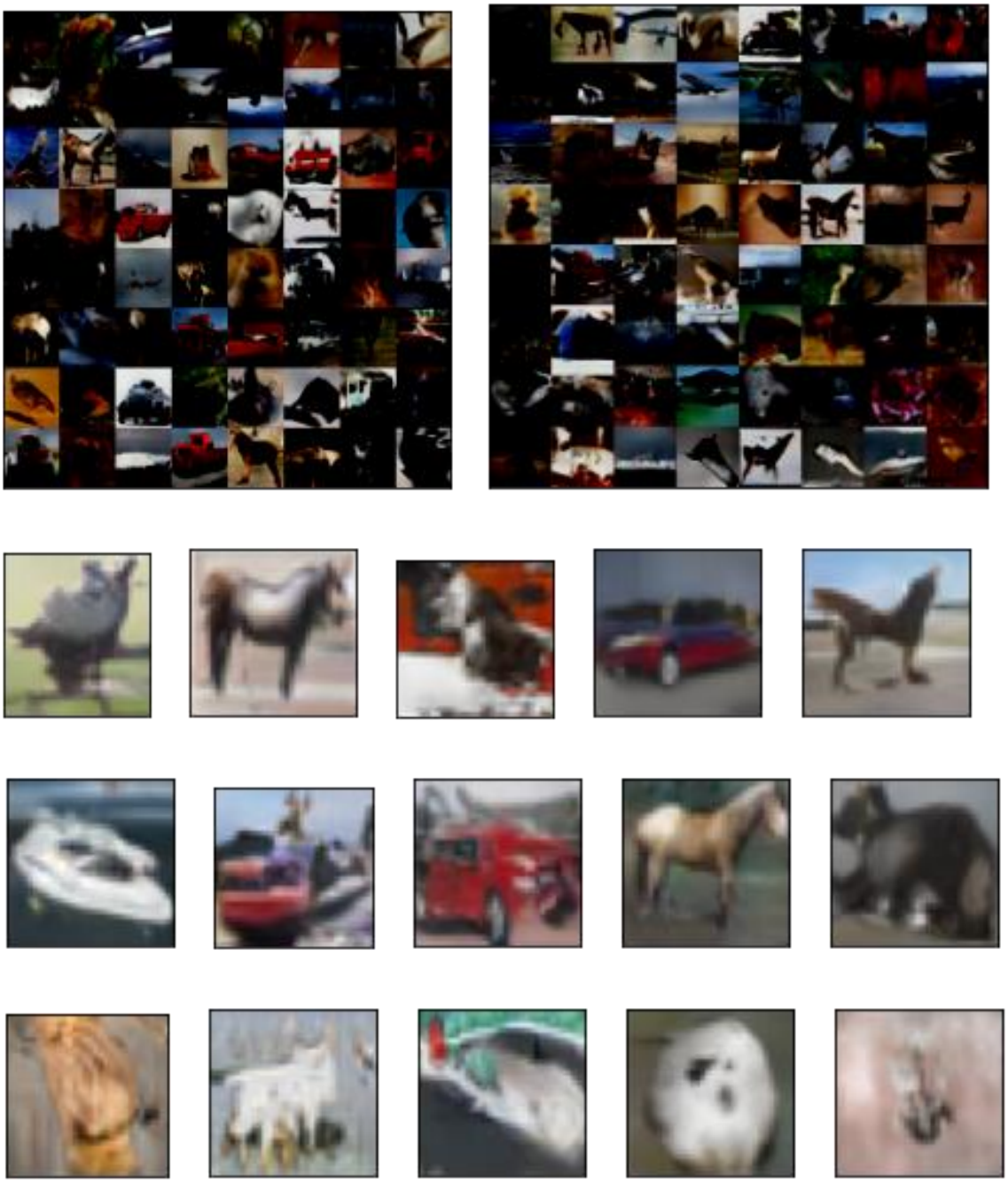

FID score vs Number of epochs
FID_SCORE
FID SCORE →
150
140
130
120
110
100
90
0
50
100
150
200
250
300
350
400
Number of epochs →

Now we will show some images obtained in the final model that had the best FID score and the final model run over 400 epochs.

Best DCGAN model(final model with 370 epochs):



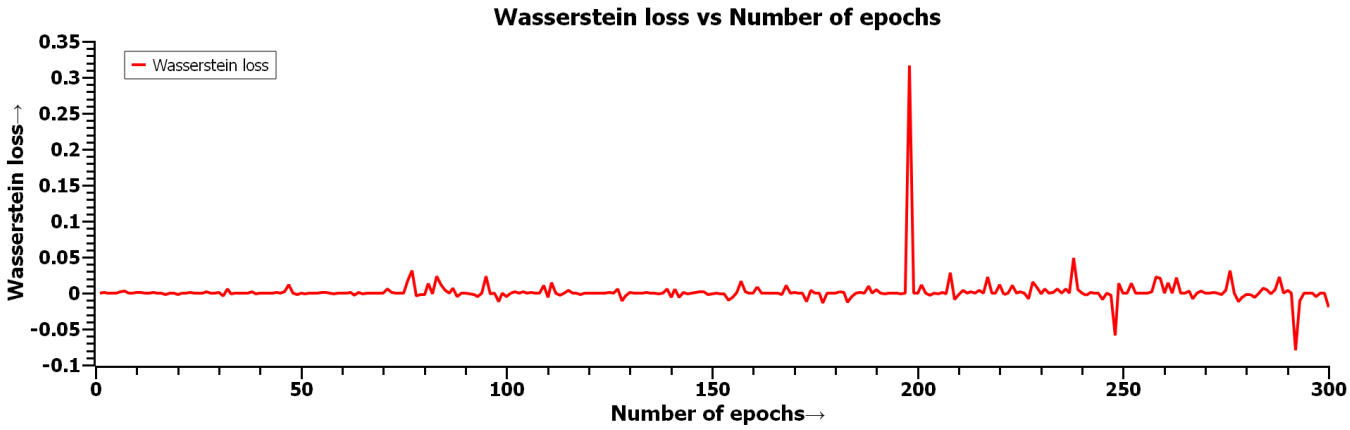
Final model after 400 epochs:

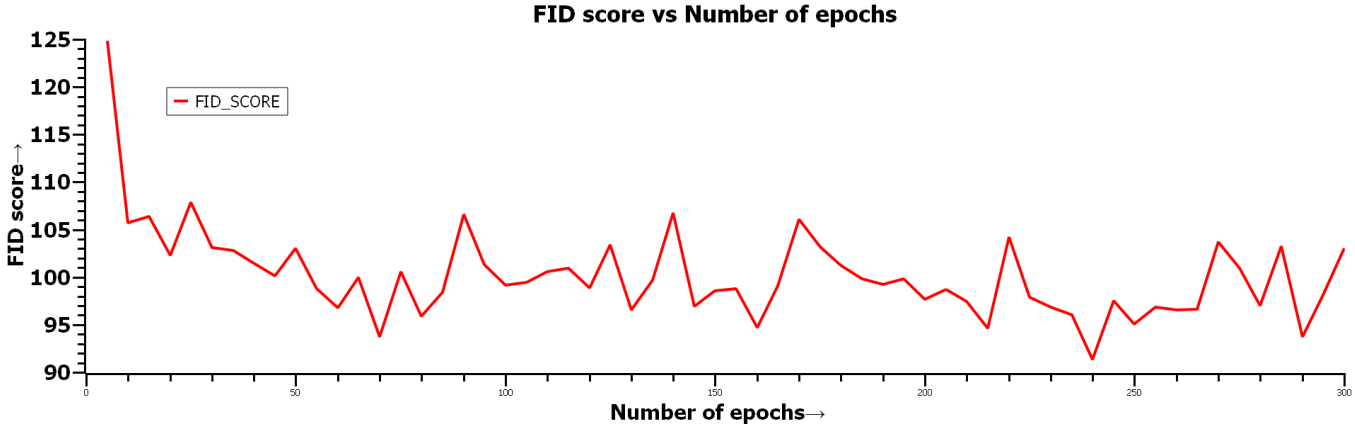


Note: I have not attached a table containing model description and the FID scores as it is difficult to contain the model description in a table. I have written all the architectures with their FID scores in Architectures.docx for both DCGAN and SAGAN.

SAGAN:

In the self attention GAN, all other implementations are same except that there is an additional attention layer added to both generator and discriminator. I have implemented the attention layer using keras. For the implementation, I have tried to mimic the implementation given in the SAGAN paper. I have taken my implementations of DCGAN and added a attention layer to both generator and disciminator. In most of the architectures and in the final model, the attention layer is added at 32 X 32 level as it is said in the project description. Almost, the trends in DCGAN and SAGAN were the same. i.e, the model that had better FID score(DCGAN model) had better FID score after having attention relatively. This trend can be easily observed in the Architectures.docx where each page has 2 models described with the top one a DCGAN and bottom model representing a SAGAN. Thus, my final SAGAN model is the final DCGAN model with attention layers included in both generator and discriminator. Because of higher computational time in case of SAGAN, I could run this over 300 epochs. The SAGAN model with best FID score was obtained at 215 epochs. I have plotted wasserstein loss and FID scores versus the number of epochs below. It can be seen that FID score of SAGAN does not improve significantly after a small number of epochs and FID score fluctuates very much with number of epochs without any significant decrease over time.

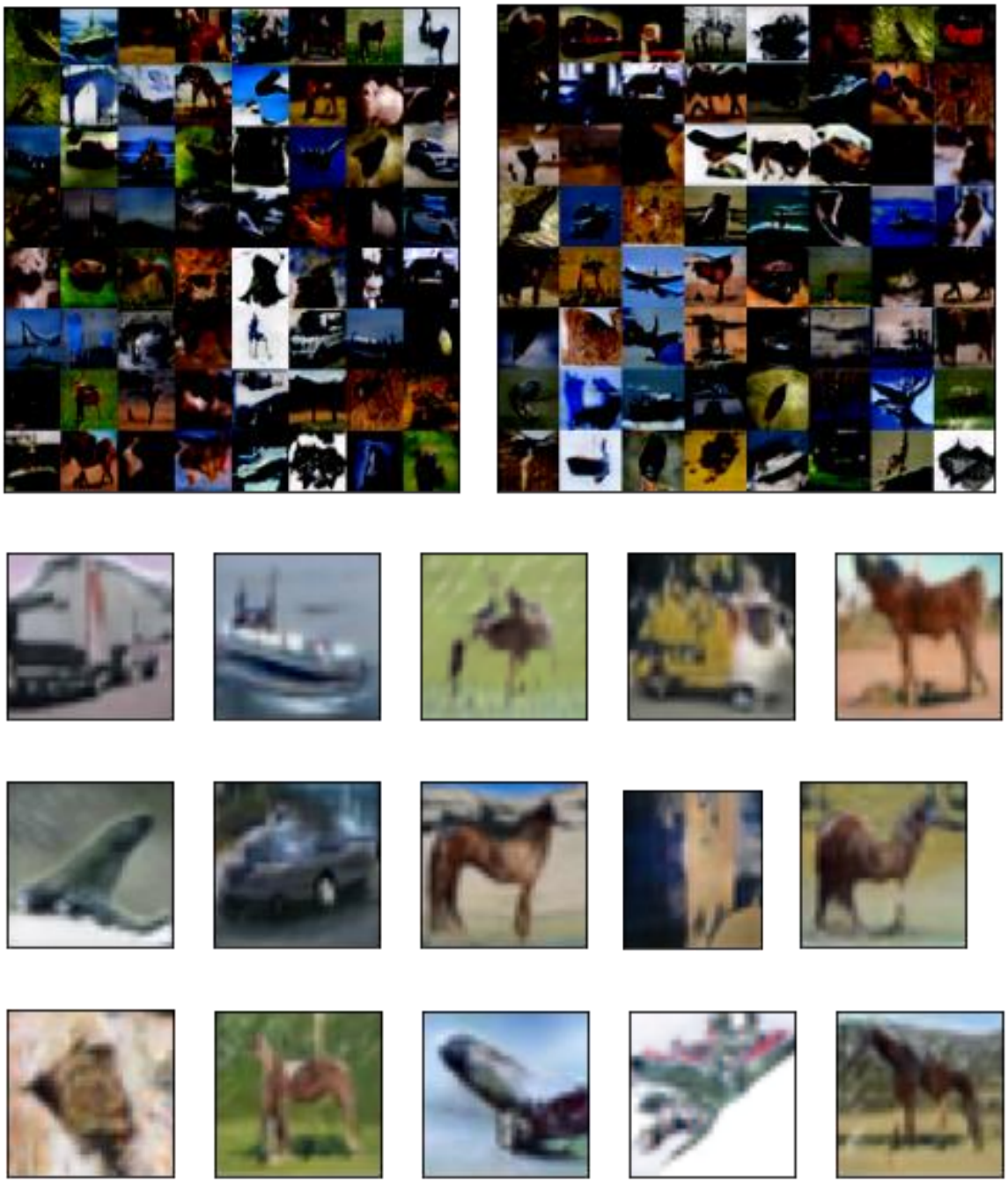




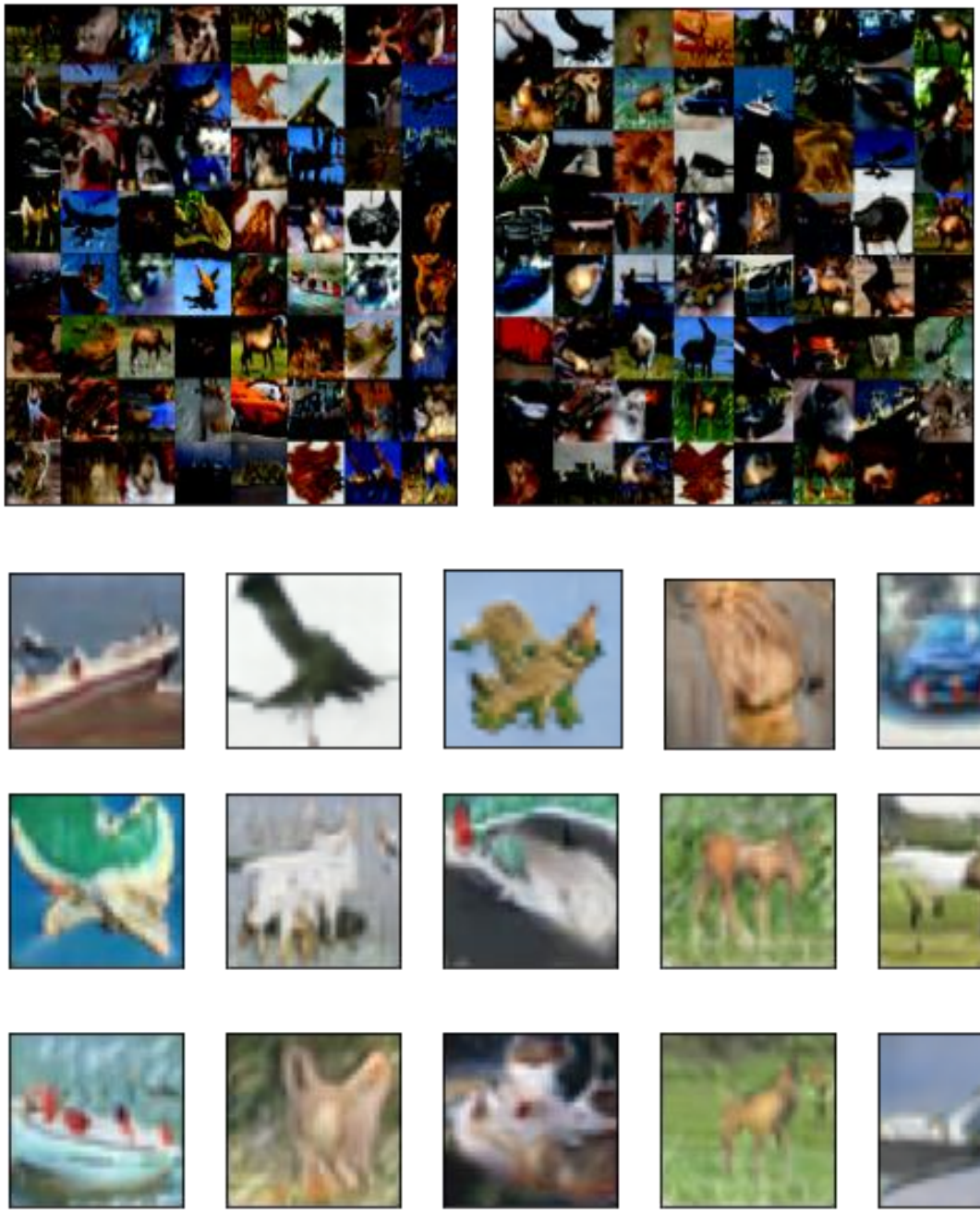
**Note: Wasserstein loss plotted is of the discriminator in both DCGAN and SAGAN. FID score was evaluated over 500 samples(of the plot) from the validation set in both DCGAN and SAGAN.

Now we show some images generated by both best SAGAN model and final model run over 300 epochs.

Best SAGAN model:



SAGAN model after 300 epochs:



Implementation of FID: Implementing FID score calculation is fairly simple and I have implemented it. We first need to rescale the images into shape of (299,299,3) and then pass on to the InceptionV3 network. Values obatined in the final layer of the inception network is used to calculate the mean and covariance between generated images and images of validation set. Then using the definition of FID score, we have calculated the FID score.

Final FID scores:

While training, I could at max calculate the FID score over 500 samples because of constraints on RAM memory. RAM used to crash after that. Initially in some architectures, I have estimated FID over 100 samples as well. Calculation over 500 samples is fair enough as repeated estimations give very near results. In most of the architectures I have tried, FID score is calculated over 500 samples and even the plots of FID scores we have made in both SAGAN and DCGAN are calculated over 500 samples.

After saving all the models, I finally calculated the FID score over 1000 samples.

Between training images themselves, FID score was around 47.65.

Between validation or test images themselves, FID score was around 47.59.

Between training and validation images, FID score was around 47.66.

You can clearly see that training and validation images belong to almost the same distribution.

For the best DCGAN model, FID score was around 62.44.

For the DCGAN model with 400 epochs, FID score was around 70.654.

For the best SAGAN model, FID score was around 66.87.

For the SAGAN model with 300 epochs, FID score was around 72.9.

(I have taken the mean of few estimates to get the above answers over 1000 samples)

Best FID score for SAGAN is higher compared to best FID score for DCGAN though their architectures are very similar other than the attention layer. We expected an improvement in the performance but the performance has decreased. Maybe, we should have tried a larger network for both generator and discriminator giving the model a higher representation capacity and then SAGAN might have worked better compared to the DCGAN. We might have obtained better results with SAGAN if he had tried more number of architectures.

Some images from best DCGAN:

Some images from best SAGAN:

Though we obtained lesser FID score for SAGAN, we can still see the effect of self attention. Long range dependencies are to be learnt to learn the shape of legs. SAGAN model has learnt the same while DCGAN has not learnt long range dependency since it lacks the attention module.

Results:

SAGAN can be used to learn long range dependenices.

Both DCGAN and SAGAN saturate and fluctuate after a small number of epochs.

Wasserstein loss is a better loss function in GANs.

Two time scale Update rule gives better results as it addresses the slow learning of Discriminator.

Best FID score found in case of DCGAN is 62.44.

Best FID score found in case of SAGAN is 66.87.

****Note: Training GANs took huge amount of time. A single epoch took several minutes many times. Training SAGNs is more time consuming compared to DCGANs. Many times, it took many hours to evauate a particular architecture. To train for larger epochs, it took too many hours.