CSA 250 Deep Learning

Assignment 4

Python Dependencies and Requirements

- python (v3.7.4)
- tensorflow (v2.2)
- Other libraries used: Please see “requirements.txt”

Project summary

This project is about learning a generative model from which new samples can be generated. In this project, two types of Generative Adversarial Networks (GANs) are implemented: Deep Convolution GAN (DCGAN) and Self-Attention GAN (SA-GAN) using CIFAR-10 dataset.

Typically, in a simple neural network, we never focus on the weak points of the network. But if we can focus this weak point and train specifically for those weak points, we can train a better network. This strategy is employed in GANs through an adversary network which is known as Discriminator. This network keeps hammering on these weak points and forces the other neural network to train on these points. This scenario can be thought of as a two player game where, every time, one player finds a strategy that is extremely good against the opponent player, then the opponent player also learns a way to deal with that new strategy. Both players are continuously learning and competing with each other making each other stronger at the same time.

In GANs we typically have two networks – a generator (G) and a discriminator (D). D and G compete with each other making each other stronger at the same time. The GAN model is obtained by concatenating the G and D. The latent dimension is chosen to be 100 throughout this project. A random noise vector of latent dimension from a Gaussian distribution (latent space) is provided as an input to G. The G produces an output image of dimensions (32, 32, 3), which is passed on as input to the D which does a binary classification whether the input image is real or not. The G trains on the data and learns to produce a sample from the original data distribution (here, CIFAR-10). The D trains on data and learns to classify real image and fake image from the original data distribution.

G and D are working against each other like an adversarial zero-sum game. G is constantly trying to fool the D into believing that, the image generated by G is from the real data distribution while D is constantly learns to improve its efficiency in detecting fake images. In this process, G learns a transformation

which enables one to convert the random noise input vector into a sample from the data distribution.

As the discriminator can get real and fake images as inputs and generated fake images have pixel values between -1 to 1, all images in the CIFAR-10 dataset in preprocessed to scale the pixel values [0, 255] range to [-1, 1] range.

FID

I used Fréchet Inception Distance (FID score) to evaluate how good the generative model. I used the 'inceptionV3' model to calculate the FID score following an existing implementation by Machine learning mastery. This method takes two sets of images as input and calculates the final layer activation for these sets. We then calculate the FID distance between these activation. If two sets of images are same, then the inceptionV3 model will generate same activation feature maps, and the FID score will be 0. If the image sets are different then the value would be greater than 0. To summarize, the lower the FID score, the better the model is. Due to the memory limitations in Google Colab, I was forced not to use the entire test dataset (10k) for this calculation. I chose to use 1000 images of test data and 1000 images of generator output for this purpose. Inception model expects the input images shapes to be (299, 299, 3) and hence the images are upscaled to this dimensions.

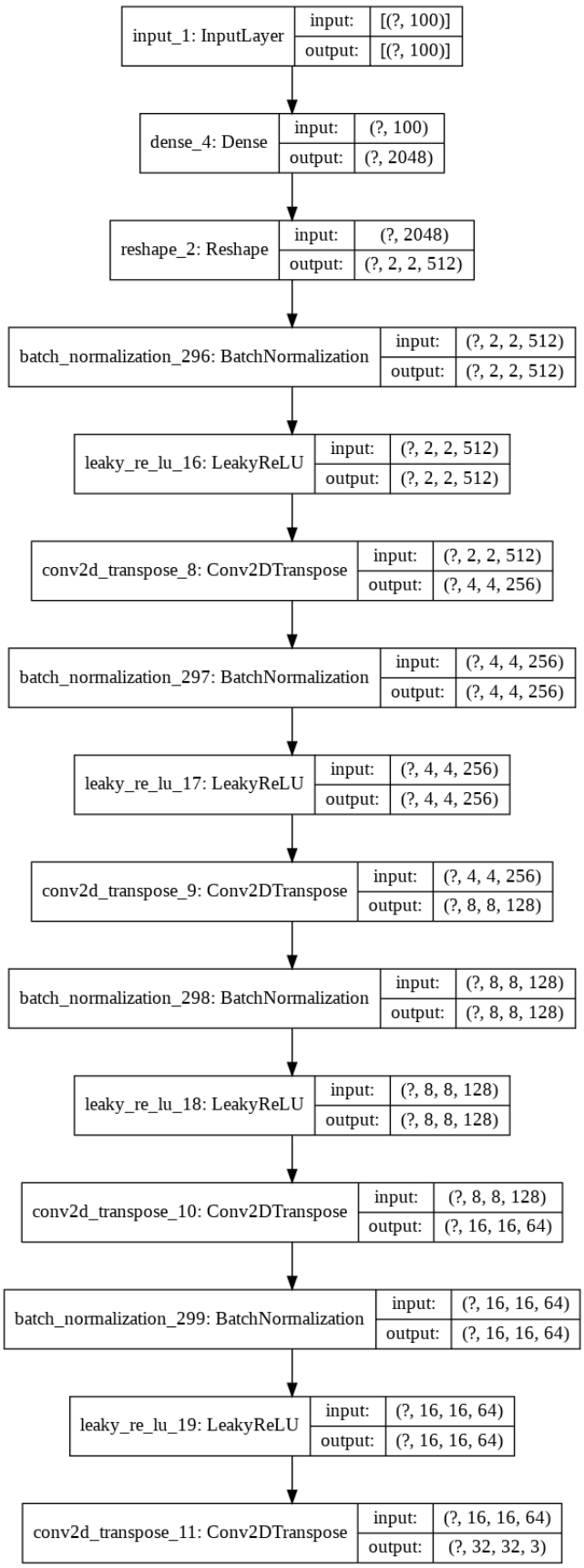
DCGAN

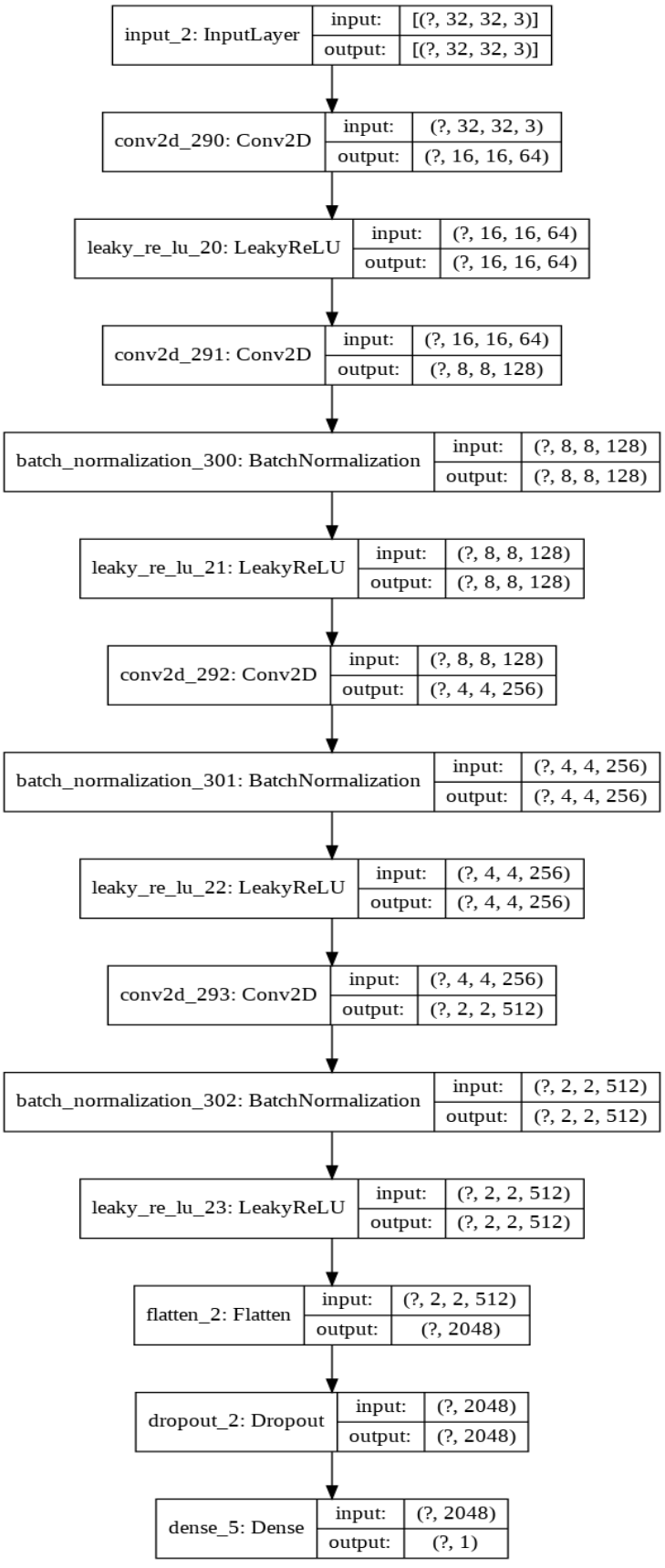
In DCGAN, I have used two convolutional neural networks (CNN) where one serves as a 'Discriminator' (D) and other serves as a 'Generator' (G). The architecture of the generator (left) and discriminator (right) is shown in image in the following page.

The DCGAN model is trained with the following hyper-parameter setting – Adam optimizer with alpha = 0.2 and beta = 0.5, generator learning rate = 0.0001, discriminator learning rate = 0.0004. I used the two time scale update rule (TTUR) and obtained better results as major part of the gradient gets absorbed in the discriminator D compared to generator G. The model attains a minimum FID of 89.68. Though this value is attained around 30k iterations, I trained the model up to 100k iterations to observe whether the GAN model collapses – which it strangely did not. Model collapse is a scenario where the generator finds a fixed set of images to fool the discriminator, which means that the generator produces the same set of images.

In each iteration, I train the model on sampled subset (batch) of 128 data points. I also employed label smoothing for the discriminator, which is proven to be effective in preventing over fitting.

Model architecture

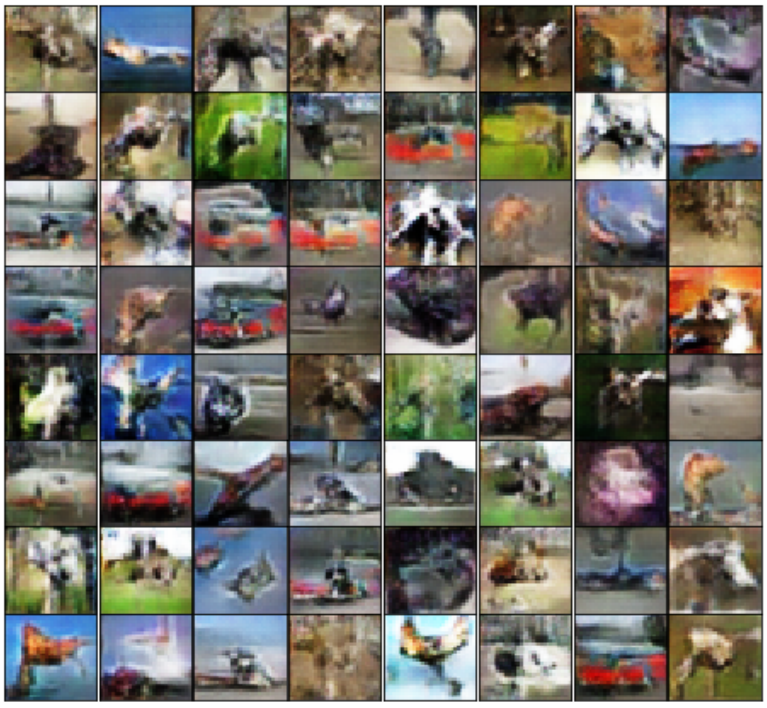




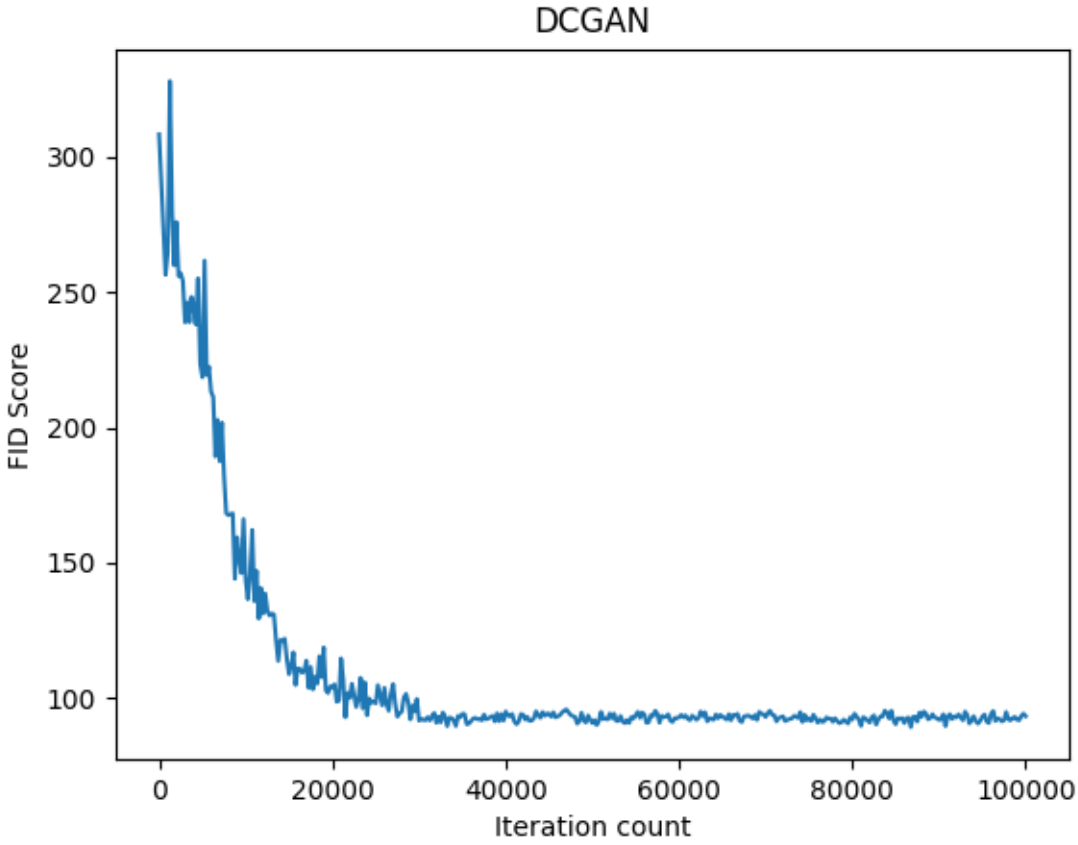
Results

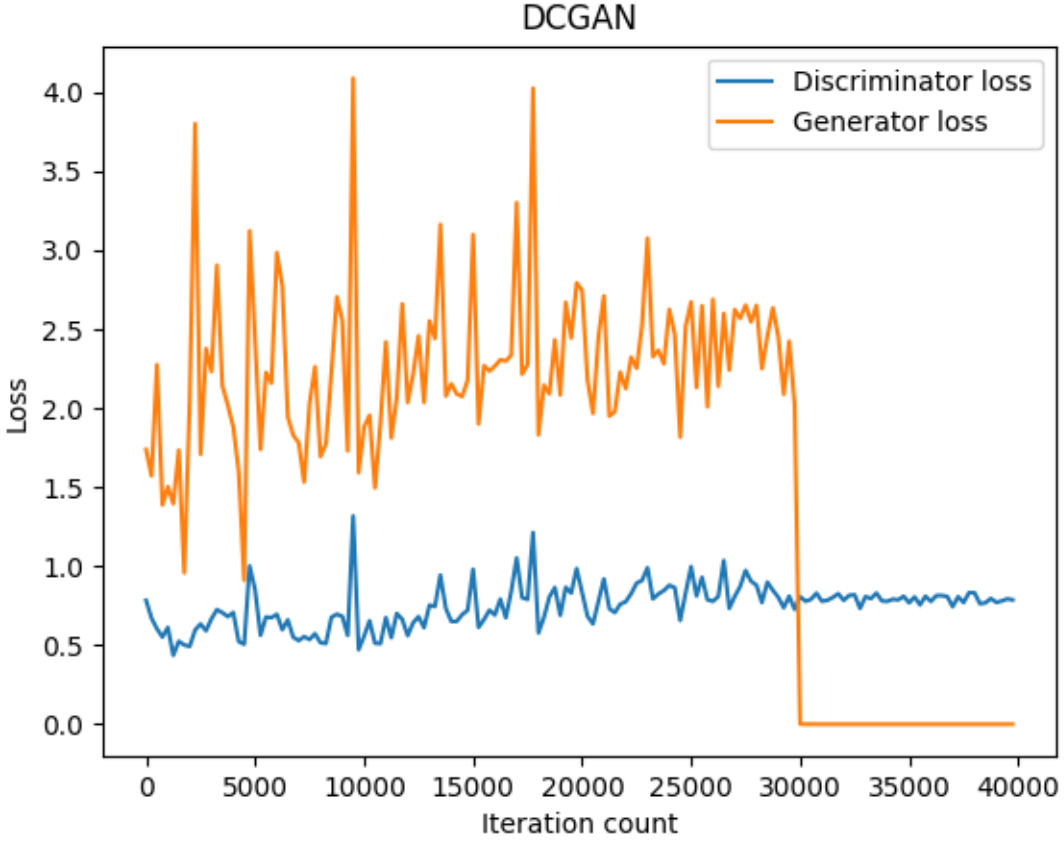
Grid of generated images from trained DCGAN is shown below. An animated GIF with grid images during training iterations is also provided. As per the project requirements, I have put ten DC-GAN generated images in the ‘./DCGAN/final_images/’ directory

The model attains a minimum FID of 89.68. DCGANs exhibit limitations while modeling long term dependencies for image generation tasks. The problem with DCGANs exists because model relies heavily on convolution to model the dependencies across different image regions. Since convolution operator has a local receptive field, long ranged dependencies can only be processed after passing through several convolutional layers.



The following page have two plots. They show the progress of the training GAN in terms of the FID score and model loss. First plot shows the variation of FID score with respect to the iteration count. Second plot shows the model losses – generator loss and discriminator loss – with respect to the iteration count.

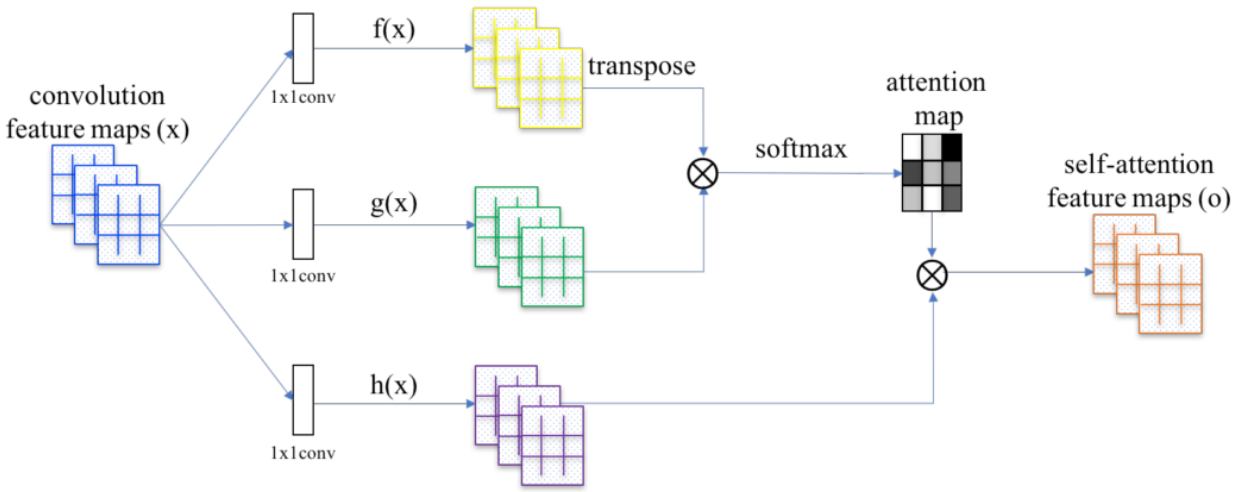

DCGAN
FID Score
300
250
200
150
100
0
20000
40000
60000
80000
100000
Iteration count


DCGAN
Discriminator loss
Generator loss
Loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
5000
10000
15000
20000
25000
30000
35000
40000
Iteration count

SAGAN

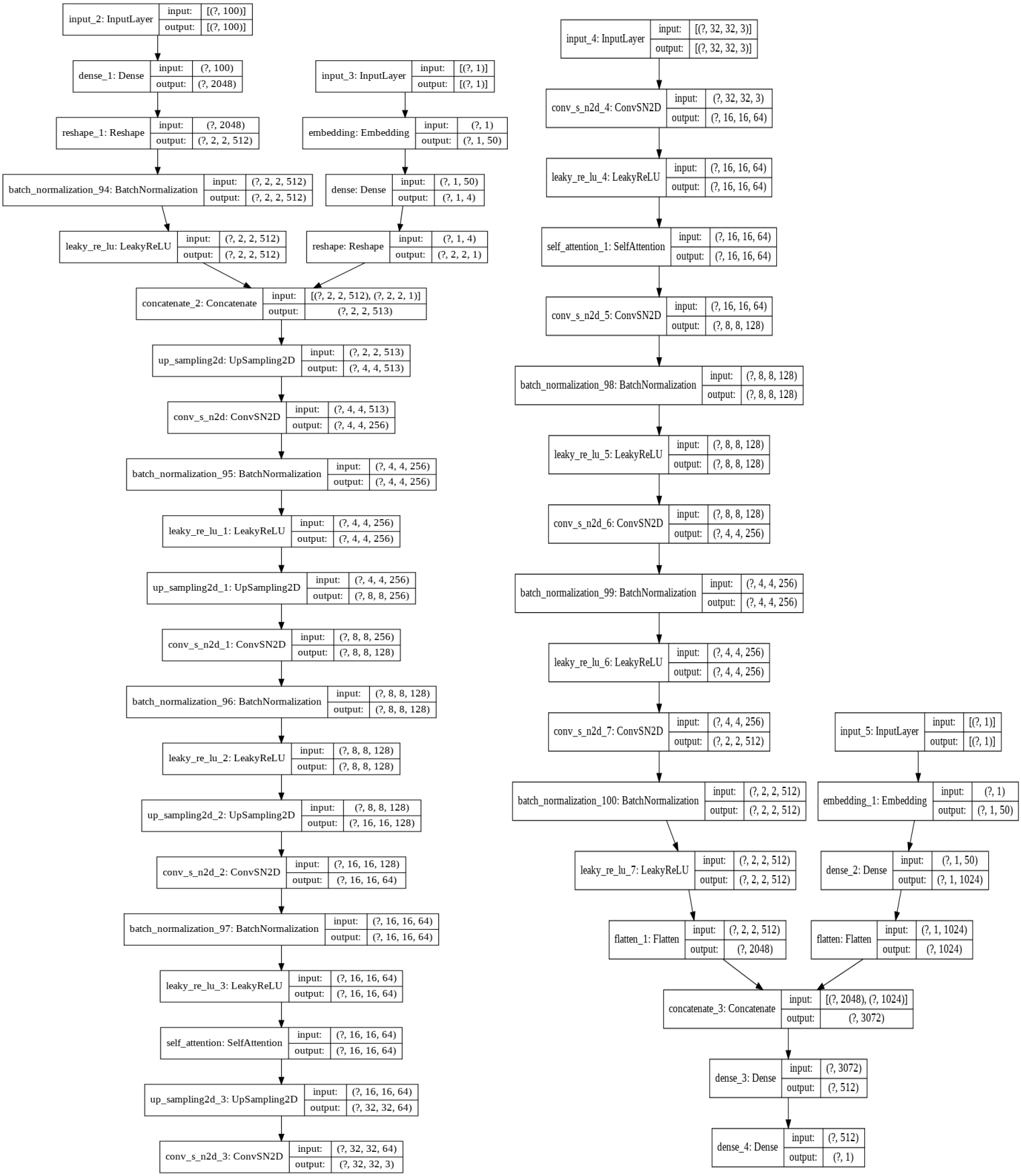
The DCGAN is able to produce new unseen images but has some inherent problems due to the convolutional structure they employ. Convolution architecture can only learn the local structure of an image since their receptive field is small. Hence it cannot model long-term dependencies in image generation tasks. We can observe from the model outputs that, DCGAN is able to produce textures very well but fails to produce accurate geometric shape due to the limiting local view. The convolution architecture can model long-term dependencies if the images are passed through several layers of convolution or by increasing the size of the convolution kernel – both resulting in the loss of computational efficiency of CNN.

It can be observed from the generated images that DCGANs are able to produce images with a simpler geometry like ocean, sky etc. but failed to produce images with specific geometry like animals and automobiles. To mitigate this issue, self-attention was introduced in the GAN model which is called as self-attention GAN (SA-GAN). Self-Attention can model long-term dependencies while maintaining the computational efficiency. A general architecture for SA-GAN is shown in the image below.



Another problem in DCGAN is mode collapse. To solve this issue I used Wasserstein loss function during the training. The architecture of the generator (left) and discriminator (right) is shown in image in the following page. I have used conditional generator and conditional discriminator for SAGAN. Conditional models takes an additional condition/constraint (here, class label) as an input to the model.

Model Architecture



In every convolutional layer, I have used batch normalization to avoid the imbalance (vanishing/exploding) of the gradients and spectral normalization to constrains the Lipschitz constant of the convolutional filters. Spectral normalization is used as a way to stabilize the training of the discriminator network. I have used a self-attention layer with 64 feature maps after the first convolution layer in discriminator and a a self-attention layer with 64 feature maps before the last convolution layer in the generator.

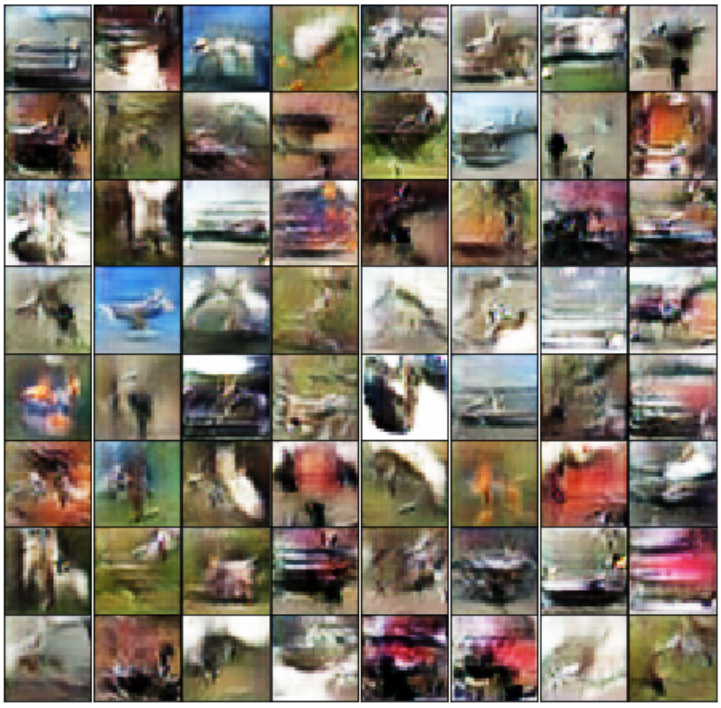
The SAGAN model is trained with the following hyper-parameter setting – Adam optimizer with alpha = 0.2 and beta = 0.5, generator learning rate = 0.0001, discriminator learning rate = 0.0004. I used the two time scale update rule (TTUR) and obtained better results as major part of the gradient gets absorbed in the discriminator D compared to generator G. The model attains a minimum FID of 107.25. Though this value is attained around 30k iterations, I trained the model up to 100k iterations to observe whether the GAN model collapses – which it did not.

In each iteration, I train the model on sampled subset (batch) of 128 data points. I also employed label smoothing for the discriminator, which is proven to be effective in preventing over fitting.

Results

Grid of generated images from trained SAGAN is shown below. An animated GIF with grid images during training iterations is also provided. As per the project requirements, I have put ten SA-GAN generated images in the ‘./SAGAN/final_images/’ directory

The model attains a minimum FID of 107.25.



The following page have two plots. They show the progress of the training GAN in terms of the FID score and model loss. First plot shows the variation of FID score with respect to the iteration count. Second plot shows the model losses – generator loss and discriminator loss – with respect to the iteration count.