CSE 676: Deep Learning

Project 1: Implementation of GANs on CIFAR-10 Dataset

Submitted by:- Shivangi Tripathi
Person Number:- 50288218

Objective:

The aim of the project is to help us understand the concept of Generative Adversarial Networks and their functionalities. The project covers different kinds of GANs and concepts behind them.

About GAN:

GAN is model architecture for training generative model. It falls under unsupervised learning category. The GAN architecture consists of two models: Discriminator and Generator model. Discriminator model is used to classify whether generated images are real or fake. Generator model is used to create fake images using random noise. Both models are trained together. The generator generates a batch of images, and these, along with real examples from the cifar-10 dataset, are given as input to the discriminator which classifies them as real or fake.

The discriminator is then updated to get better at identifying real and fake images in the next round, and importantly, the generator is updated based on how well the generated images fooled the discriminator. During training, the weights in the generator model are updated based on the performance of the discriminator model. When the discriminator is good at detecting fake samples, the generator is updated more, and when the discriminator model is relatively poor or confused when detecting fake samples, the generator model is updated less. In this way both models are competing against each other.

The difference between GAN and DCGAN is of convolutional layer instead of fully connected layers. These convolutional layers find areas of correlation within an image which GAN is unable to find successfully.

About DCGAN:

Deep Convolutional GAN or DCGAN is a type of GAN with convolutional layers which uses convolutional strides and transposed convolution for downsampling and upsampling the images respectively. The idea behind DCGAN is to replace fully connected layers with convolutional layers which are used for upsampling in generator. The DCGAN Architecture is like GAN with difference in layers used in DCGAN.

FID Score:

FID or Frechet Inception Distance is a metric to calculate distance between feature vectors for real and generated images. The score indicates the similarity in terms of statistics of computer vision features of the raw images calculated using Inception V3 model used for image classification. It is used to evaluate the quality of images and the lower score indicates high quality images with high

similarity. The score=0 means both images are identical. This score has been calculated for all the models.

1. Architecture

1.1 The Generator:

It upsamples the noise to produce an image. It starts with Dense layer whose input is a random noise as input and upsamples it to generate 32x32x3 images. The model reshapes the output of Dense Layer to 4x4 feature maps. Then it upsamples the noise using Conv2DTranspose layers with stride 2x2. It has LeakyReLU and uses tanh activation function as images have been normalized to [-1,1]. The model has been trained using binary cross entropy loss function using Adam optimizer.

1.2 The Discriminator

It takes 32x32x3 input image and gives binary classification as output (1 for real and 0 for fake). The model has a normal convolutional layer followed by 3 convolutional layers with stride 2x2 to downsample the input images. It uses sigmoid activation function to produce binary output. The model has been trained on binary cross entropy loss function using Adam optimizer. It has LeakyReLU and Dropout as well.

2. Hyperparameter

2.1 Adam Optimizer:

This has been used for optimizing model with learning rate=0.0002 and momentum(beta_1)=0.5. It is an optimization algorithm in extension to stochastic gradient descent and has been used here as it gives best results.

2.2 Learning Rate: Although it is recommended to use two-timescale learning rates update rule that is learning rate=0.0004 for discriminator and 0.0001 for generator, while training the model, it was leading to vanishing gradients. Hence I used learning rate=0.0002 for both models.

2.3 LeakyReLU:

This has been used to avoid dying ReLU problem i.e. no zero-slope parts as well as to speed up training of the GAN model. So ReLU is not used here. It has slope=0.2 for training gan model.

2.4 Number of Iterations:

The higher the number of iterations, the more efficiently the model get trained and produces more clear images.

3 Code/Training:

In the notebook, the code has been written in following cells:

3.1 Discriminator model based on architecture defined above

3.2 Generator model based on architecture defined above

3.3 Gan model using generator and discriminator model have been defined. It is just a model to combined discriminator and generator models. In gan model, (discriminator)d_model.trainable is set False so that layers in discriminator are not updated and overtrained on fake images. It is also compiled using binary cross entropy loss function with Adam optimizer and learning rate=0.0002.

3.4 Loading the dataset to get real images. The fake images are also generated using latent dimension=100. Real images have been assigned label=1 and Fake images have been assigned label=0.

3.5 Method for calculating FID score has been defined. It is an existing implementation.

3.6 Training of DCGAN: method for training the models has been written which has number of iterations=40000 since the browser was crashing after 40000 and GPU Limitations. After every 10000 iterations, we calculate Fid score on validate dataset having 64 images. I tried to calculate FID for every 1000 iterations, however it was taking a lot of time and ultimately Hardware limitation occurred. I tried to use 1000 images as well, however, the browser was crashing for that many images. After every 10000 iterations, images have been plotted. In this training method, real images are generated and given as input to train_on_batch() of discriminator model to train it. Fake images are generated and given as input to train_on_batch() of discriminator model to train it. After that fake images are generated using latent points with labels=1 in order to confuse discriminator and given as input to train_on_batch() of gan model. Each training method generates accuracy and loss. Loss has been plotted using tensorboard. At the end of each iteration, the loss values are stored in respective tensorboard objects.

3.7 Main cell for calling all the above declared methods: here defined methods for discriminator, generator and gan models have been called. Then tensorboard objects have been declared to store event files and use these files to plot graphs for losses. Then train method has been called with datasets, latent dimension=100, tensorboard objects and list for saving fid scores. At the end of training, the models are saved and tensorboard objects call on_train_end(None).

3.8 Various cells to plot FID Score, graphs for different losses using tensorboard and plot final image grid using trained generator model have been called.

4 Result:

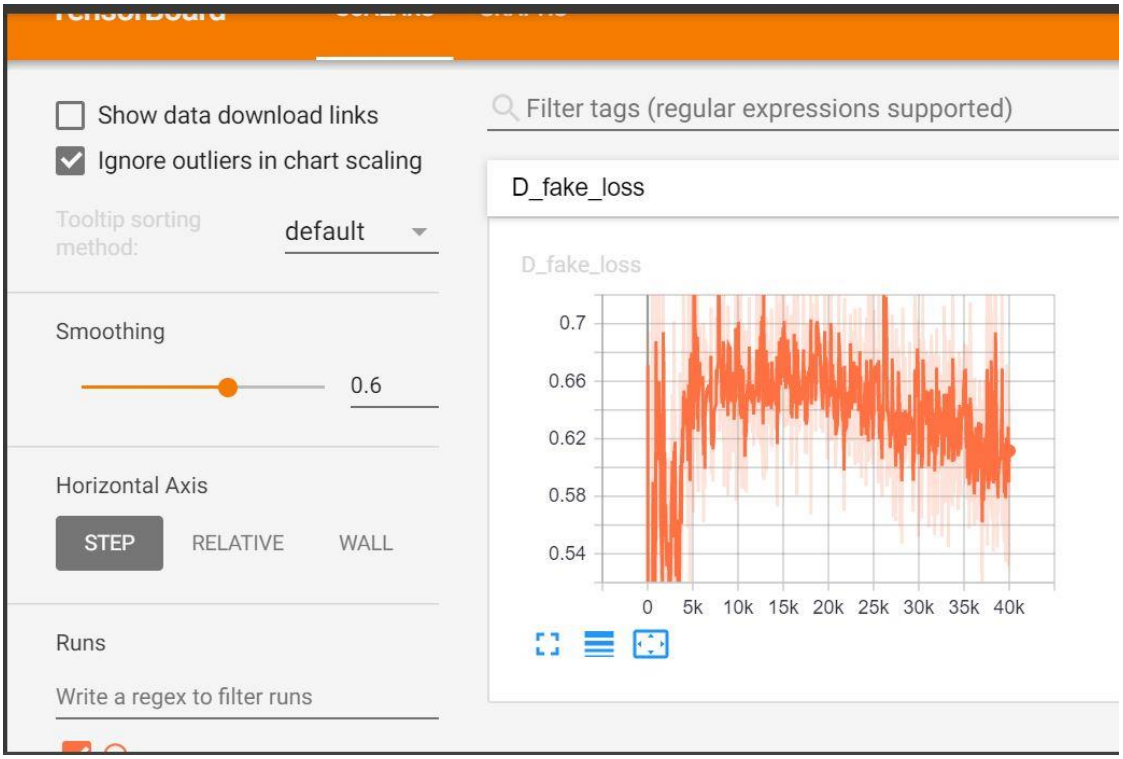
The following final images were obtained after training the generator model.

5 Evaluations:

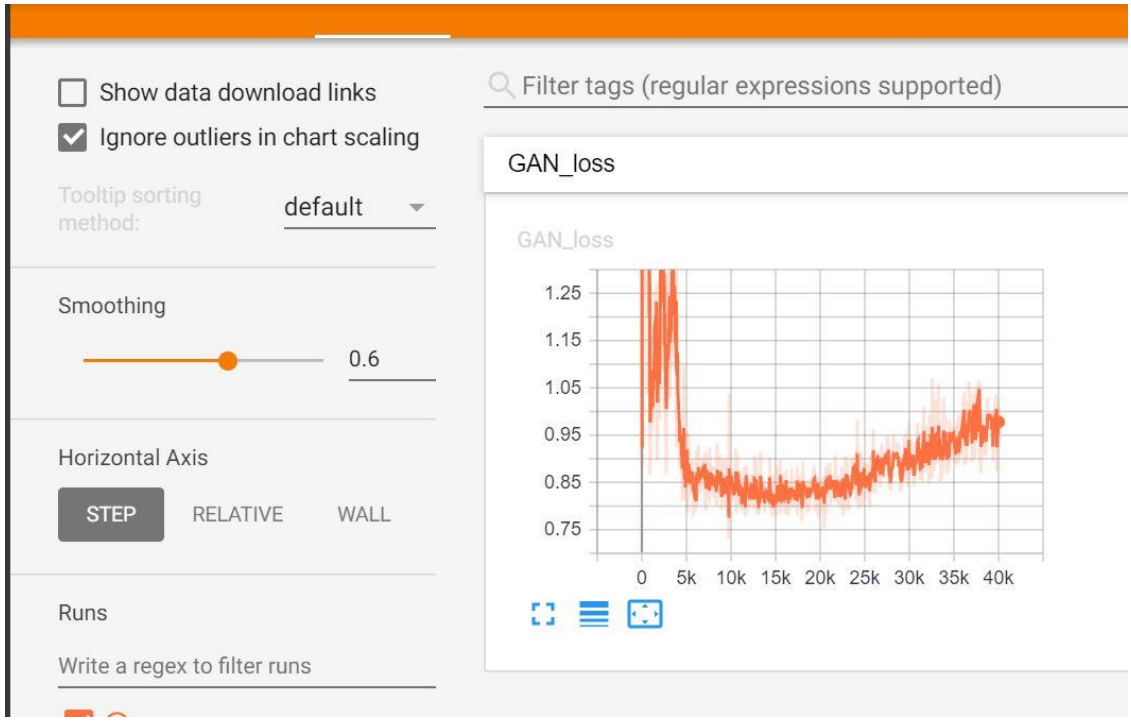
The following graphs for losses were obtained after training. Each graph has X-axis= number of iterations and Y-axis=loss value.

5.1 Discriminator Loss for Real Images:

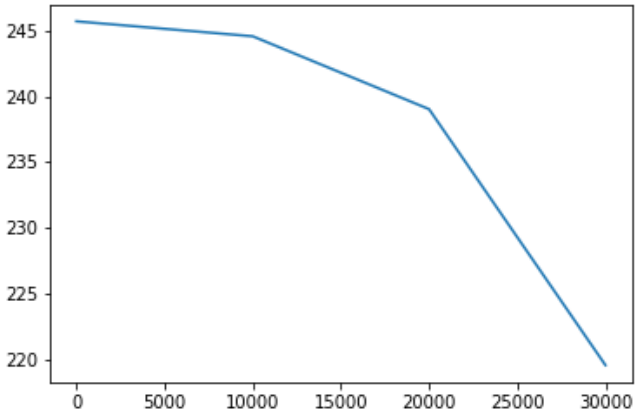
5.2 Discriminator Loss for Fake Images:



5.3 GAN Model Loss:



5.4 FID Score: the score decreases from 245 to 220 which means if more number of iterations were performed for training , the score would have decreased more, increasing quality of images.



Limitations of DCGAN:

Even though DCGAN has an advantage over GAN for image data, there are various limitations in DCGAN related to long term dependencies for image generation. It has difficulty in modeling some image classes than other while training multi-class dataset. It fails to create images with some specific geometry. Large convolutional kernels are computationally inefficient in DCGAN for long ranged dependencies. Also, it faces Mode Collapse- The generator collapses which produces limited varieties of samples. These problems exist because the convolution is a local operation

whose receptive field depends on the spatial size of the kernel. The output of convolution has no relation to any other part of except for small local region in image where output is computed. Thus, as solution to these issues, SAGAN is introduced. As well as to solve Mode collapse- Wasserstein Loss is used.

About SAGAN:

Self-Attention GAN model is same as DCGAN with additional layer of Attention to get better results. It uses Attention mechanism to create balance between efficiency and long-range dependencies in larger receptive field. The Attention mechanism can use defined using 3 vectors- Query, Key and Value. Query and Key are multiplied in such a way that they create another vector of probabilities which decide how much of Value to expose to next layer. When the query, key and value are same, it is known as self-attention.

In Self- Attention, Query and Key perform matrix multiplication and then pass through SoftMax which converts resultant vector into probability distribution and it finally gets multiplied by Value. For each spatial location, an attention map is created which acts as mask and indicates the impact of a location when rendering another location. In the given below diagram, f=Query, g=Key and h=Value, x=channel and o=output of attention layer.

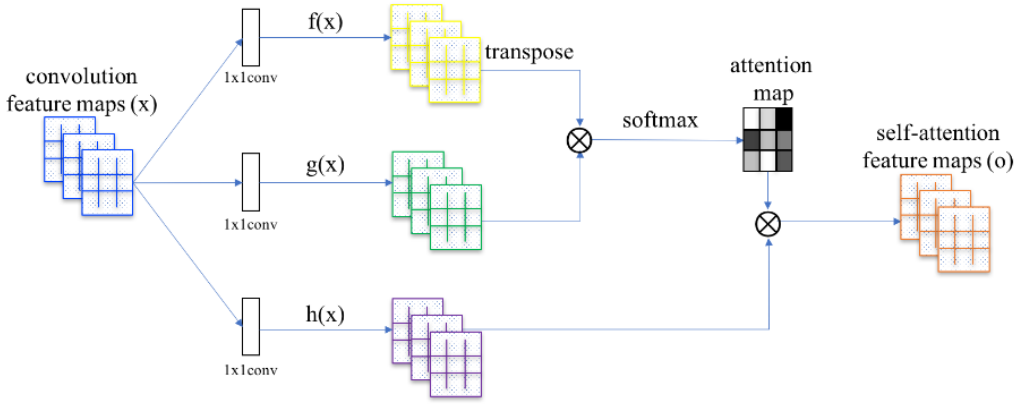


Figure 2: The proposed self-attention mechanism. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

The final output of attention layer is written as O=o*b+x, where b is a parameter which is initialized as zero and then gradually increases as model will assign value to it and use self-attention. This self-attention layer helps the model to capture fine details in long-ranged dependencies in image and remember, which makes it complementary to convolution operator.

1. Architecture:
   1.1 The Discriminator: It has same layers as used in DCGAN, except additional layer of Attention which takes 128 channels as input. This layer has been added in between the convolutional layers.
   1.2 The Generator: It has same layers as used in DCGAN, except additional layer of Attention which takes 128 channels as input. This layer has been added in between the convolutional layers.
2. Hyperparameter:

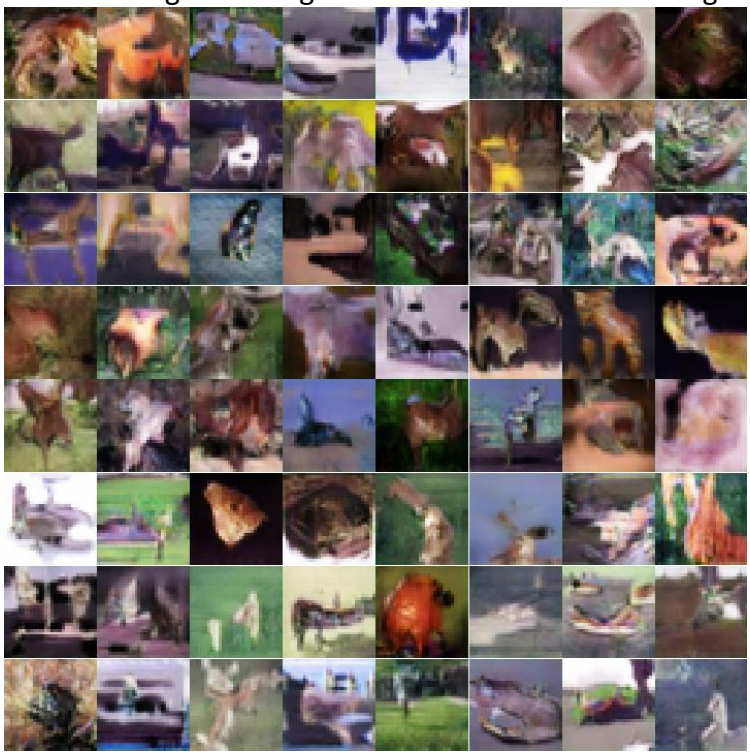
The same values as DCGAN described above, have been used to tune SAGAN.

3. Code/Training:

   In the notebook, all the cells are same as DCGAN, except the architecture of discriminator and generator model. The Self-Attention Layer is written in different python file which is imported in order to run Attention() Layer. The model has been trained for 40000 iterations only since the browser was crashing after that and GPU limitations in Colab.

4. Result

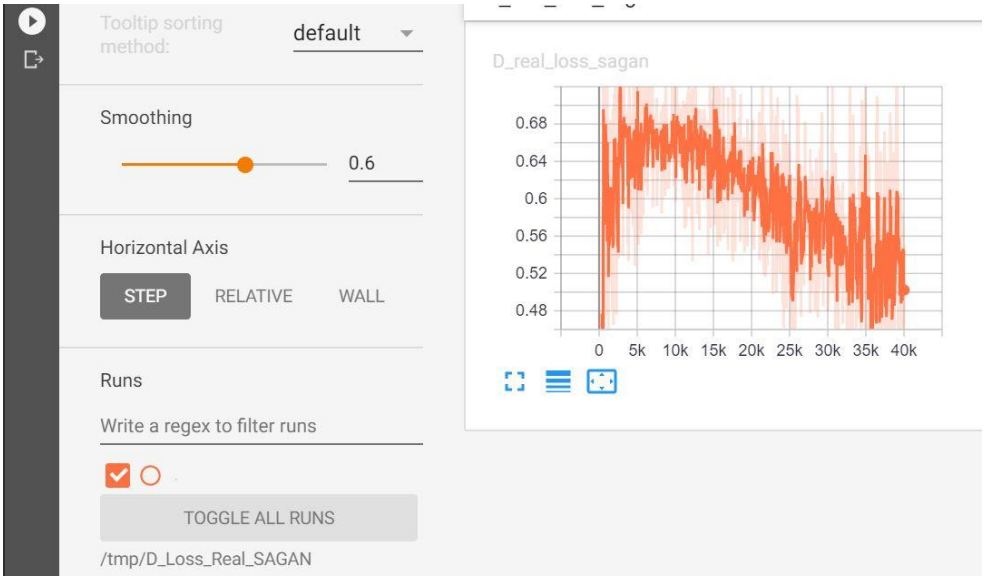
   The following final images were obtained after training the generator model.



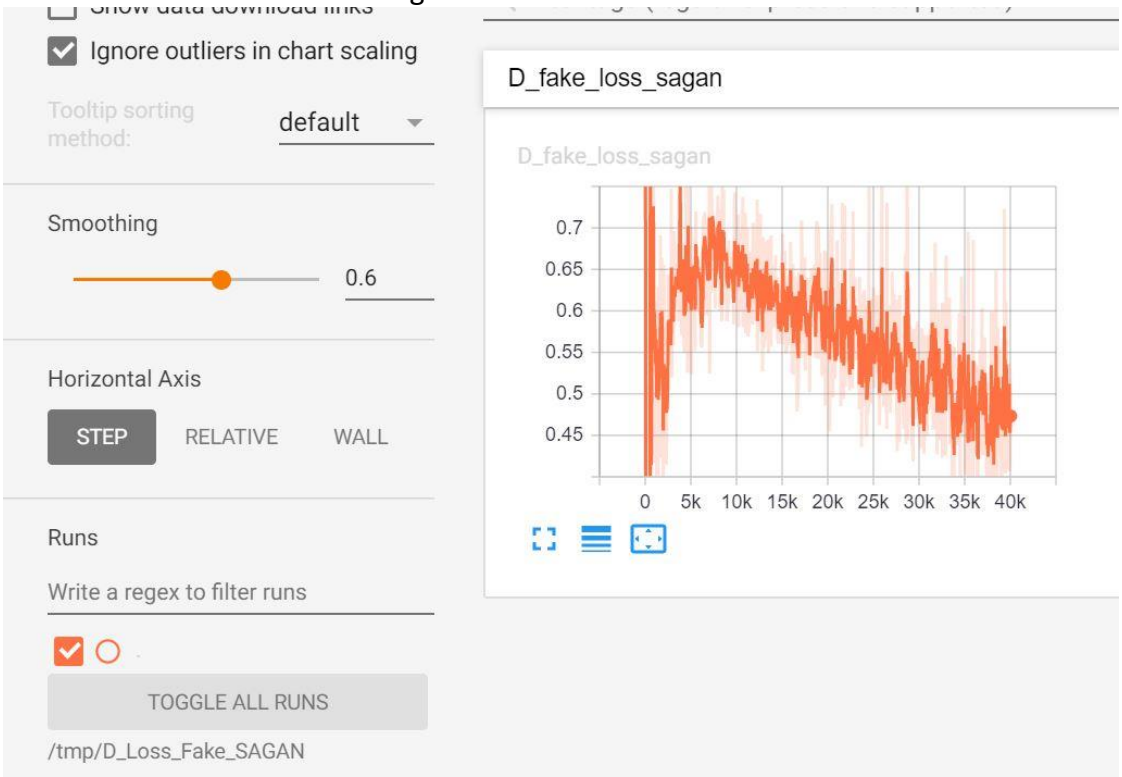
5. Evaluations

   The following graphs for losses were obtained after training. Each graph has X-axis= number of iterations and Y-axis=loss value.

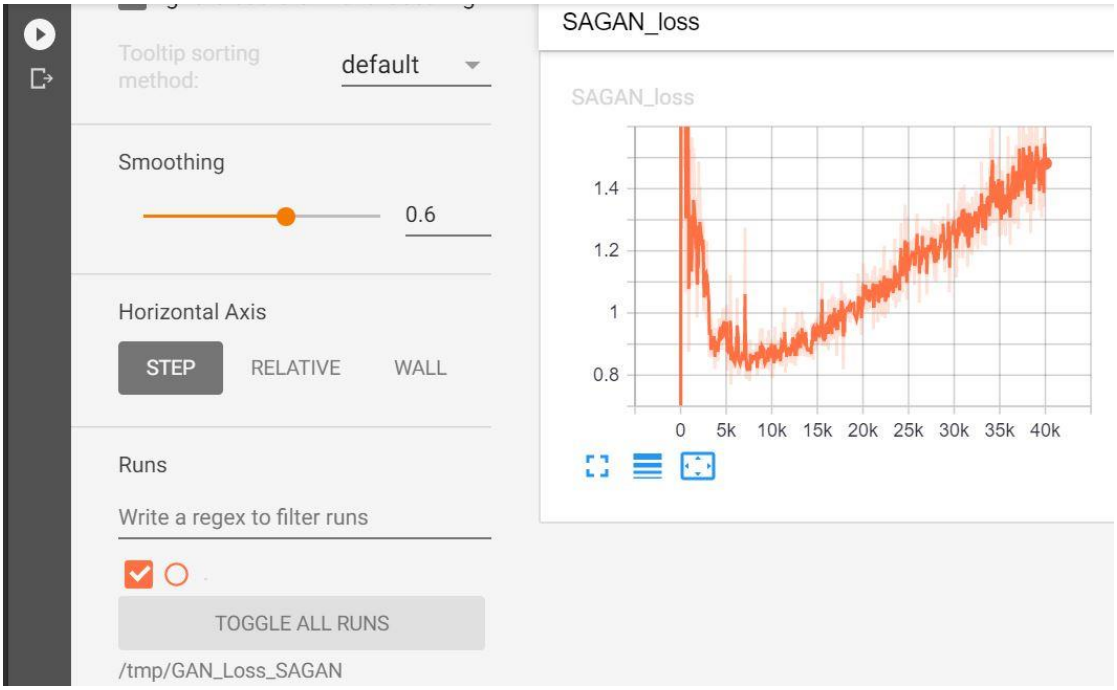
   5.1 Discriminator Loss for Real Images:



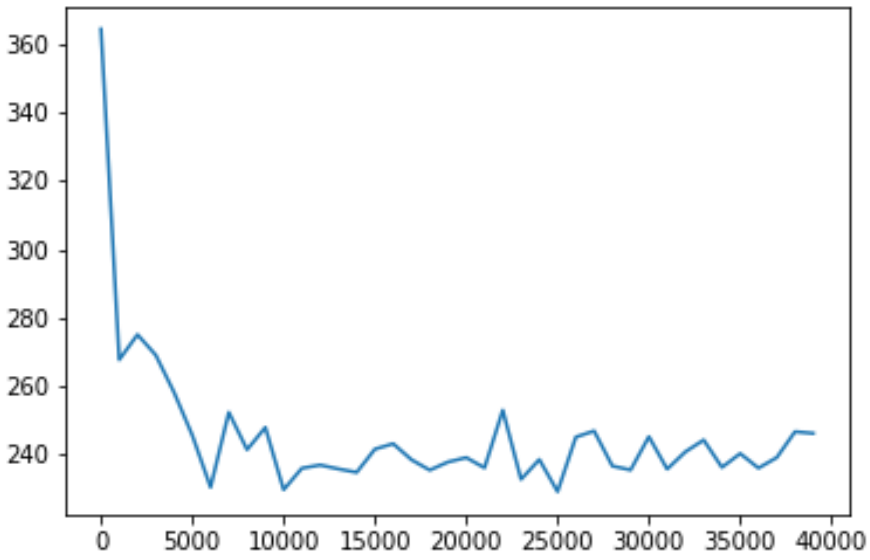
5.2 Discriminator Loss for Fake Images:



5.3 GAN Model Loss:



5.4 FID Score: The score decreases from 360 and then oscillates between 240 to 260.



SAGAN with Spectral Normalization, Conditional Input and Wasserstein Loss:

For this project, we had to implement self-attention conditional GAN with Spectral Normalization and Wasserstein Loss. The concept of self- attention has been explained in SAGAN model. Now I will explore other important concepts involved with SAGAN implementation for this project.

Conditional GAN:

Here the GAN is trained based on class labels. It means both discriminator and generator are conditioned based on class labels. This results in more stable training and high quality images for each class label. When generator is trained, it can generate images of a given type or class label.

Wasserstein Loss:

The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution q to the data distribution p. It is defined as the greatest lower bound for any transport plan. It is represented as: -

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\|x - y\| \right] ,$$

It is basically element wise multiplication of true labels and predicted labels and then calculating mean. If the value of computation is maximizing then it is real image and if it is minimizing, it is fake image. Here the discriminator does not have output sigmoid function, instead it uses the loss function which encourages it to predict a score of how real or fake a given input looks. We use weight clipping for using Wasserstein loss. Weight clipping is an important concept and is required to restrict maximum weight value in weights of discriminator controlled by some hyperparameter 'c'. Weight clipping is used as weight regulation and limits the capability of model complex functions. It is applied to discriminator only. Wasserstein loss function helps in reflecting image quality. Due to this mode collapse issue is resolved.

Spectral Normalization:

It is used to stabilize GAN training. It normalizes the spectral norm of the weight matrix in each layer so that it satisfies the Lipschitz constraint sigma(W)=1. It is given by:

$$\sigma(W_{SN}(W)) = \sigma(W)/\sigma(W) = 1$$

Spectral Normalization regularizes the gradient of W, preventing column space of W from concentrating into one particular direction. Overall, it is used for controlling gradients. This has been applied to both Discriminator and Generator.

1. Architecture:

   1.1 The Discriminator:

   Here the weight is initialized and object of ClippingClass is created in order to perform weight clipping. A new second input is defined that takes an integer for the class label of the image. This has the effect of making the input image conditional on the provided class label. The class label is then passed through an Embedding layer with the size of 50. It means that each of the 10 classes for the dataset will map to a different 50-element vector representation that will be learned by the discriminator model.
   The output of the embedding is then passed to Dense Layer with Spectral Normalization(DenseSN) . The output of DenseSN layer is reshaped into single 32×32 activation map and concatenated with the input image. This has the effect of looking like a two-channel input image to the next convolutional layer. Then normal

convolutional layers with stride 2x2 and spectral normalization are used with weight clipping constraint and initializer. Also Attention layer is added for channel=128.

1.2 The Generator:

Here the weight is initialized. The class label is passed through an embedding layer to map it to a unique 50-element vector and is then passed through DenseSN layer before being resized. In this case, the output of DenseSN layer is resized into a single 8×8 feature map. Then this resized output is concatenated with input image, which is then upsampled as in the generator used in DCGAN model with weight initializer and spectral normalization. Here Attention layer is added for channel=128.

2. Hyperparameters:

The same values as DCGAN described above, have been used to tune conditional SAGAN with Spectral Normalization and Wasserstein Loss.

3. Code/Training:

In the notebook, the code has been written in following cells:

3.1 Discriminator model based on architecture defined above

3.2 Generator model based on architecture defined above

3.3 Gan model using generator and discriminator model have been defined. It is just a model to combined discriminator and generator models. In gan model, (discriminator)d_model.trainable is set False so that layers in discriminator are not updated and overtrained on fake images. It is also compiled using Wasserstein loss function with Adam optimizer and learning rate=0.0002.

3.4 Loading the dataset to get real images. The fake images are also generated using latent dimension=100. Real images have been assigned label=-1 and Fake images have been assigned label=1 in order to generate Wasserstein loss.

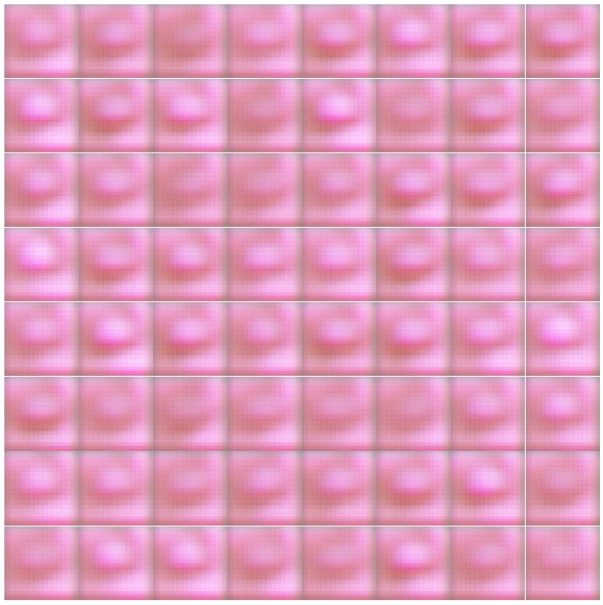
3.5 Method for calculating FID score has been defined. It is an existing implementation.

3.6 Training of DCGAN: method for training the models has been written which has number of iterations=4000 because of GPU Limitations and output was not coming good. After every 1000 iterations, we calculate Fid score on validate dataset having 64 images. I tried to calculate FID for every 1000 iterations, however it was taking a lot of time and ultimately Hardware limitation occurred. I tried to use 1000 images as well, however, the browser was crashing for that many images. After every 1000 iterations, images have been plotted. In this training method, real images are generated and given as input to train_on_batch() of discriminator model to train it. Fake images are generated and given as input to train_on_batch() of discriminator model to train it. After that fake images are generated using latent points with labels=-1 in order to confuse discriminator and given as input to train_on_batch() of gan model. Each training method generates accuracy and loss. Loss has been plotted using tensorboard. At the end of each iteration, the loss values are stored in respective tensorboard objects.

3.7 Main cell for calling all the above declared methods: here defined methods for discriminator, generator and gan models have been called. Then tensorboard objects have been declared to store event files and use these files to plot graphs for losses. Then train method has been called with datasets, latent dimension=100, tensorboard objects and list for saving fid scores. At the end of training, the models are saved and tensorboard objects call on_train_end(None).

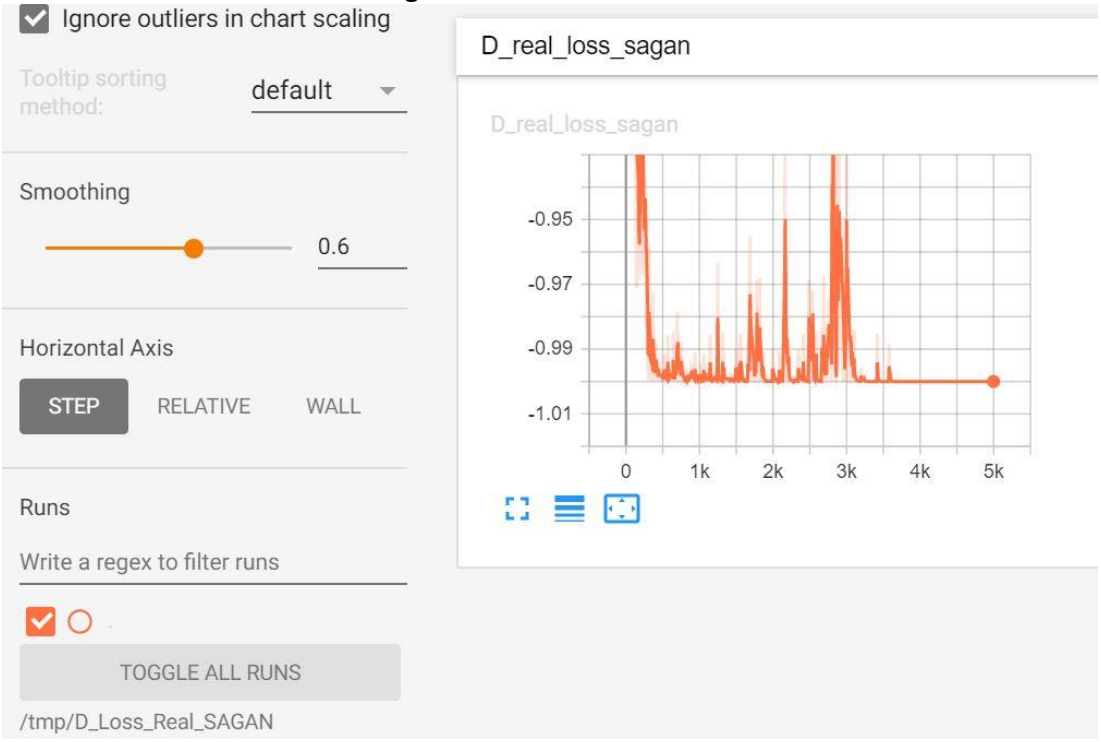
3.8 Various cells to plot FID Score, graphs for different losses using tensorboard and plot final image grid using trained generator model have been called.

4. Result:



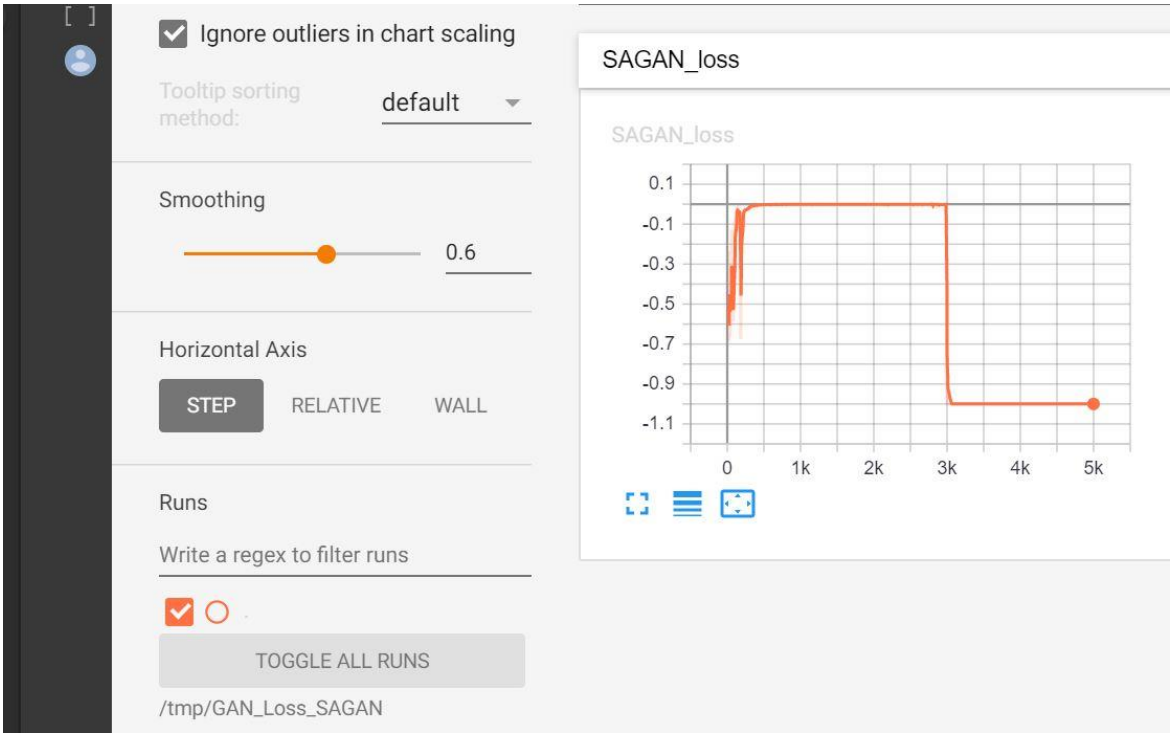
5. Evaluations:

5.1 Discriminator Loss for Real Image:

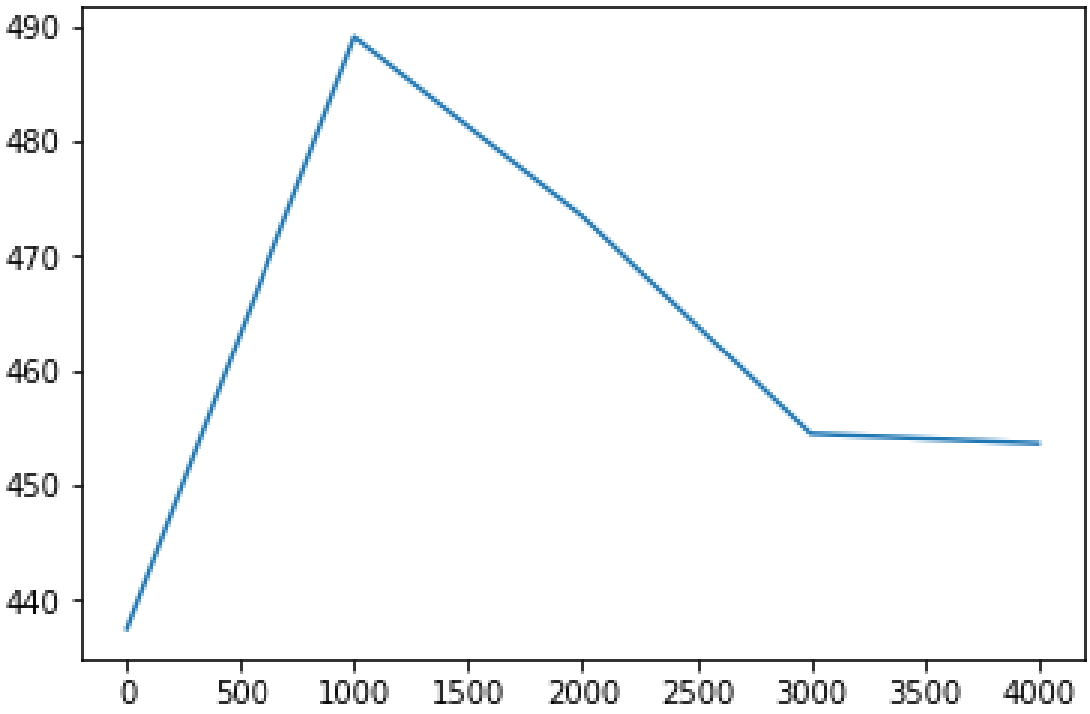


5.2 Discriminator Loss for Fake Image:

5.3 GAN Model Loss:



5.4 FID Score: The score first increases from 440 to 490 and then decreases till 455 around.

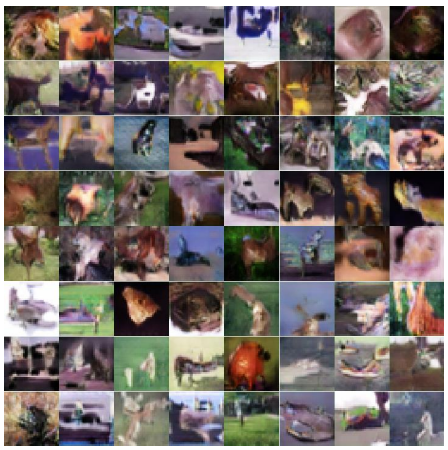


Comparisons of Models: DCGAN vs SAGAN

Here we will compare image quality of images generated by DCGAN and SAGAN respectively.

DCGAN:

SAGAN:



Here you observe that SAGAN has more good quality image than DCGAN due to addition of self-attention mechanism.

Concepts Learned:

Various concepts learned such as Self-Attention, Spectral Normalization, Wasserstein Loss, Weight Clipping, DCGAN, Conditional GAN, Hyperparameters used and functionalities of different types of GANs.

References:

1. https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f
2. https://skymind.ai/wiki/generative-adversarial-network-gan
3. https://www.tensorflow.org/tutorials/generative/dcgan
4. https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/
5. https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/
6. https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/
7. https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b
8. https://github.com/IShengFang/SpectralNormalizationKeras/blob/master/SpectralNormalizationKeras.py
9. https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/
10. https://medium.com/@jonathan_hui/gan-how-to-measure-gan-performance-64b988c47732
11. https://github.com/kiyohiro8/SelfAttentionGAN
12. https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490
13. https://towardsdatascience.com/not-just-another-gan-paper-sagan-96e649f01a6b
14. https://medium.com/@jonathan_hui/gan-self-attention-generative-adversarial-networks-sagan-923fccde790c
15. https://medium.com/towards-artificial-intelligence/techniques-in-self-attention-generative-adversarial-networks-22f735b22dfb