DEEP LEARNING- cse 676

DCGAN AND conditionalSAGAN

Aditya vikram Parakala - 50289171

Introduction:

- **GAN(Generative Adversarial Network)** was introduced by Ian J. Goodfellow in 2014.
- It belongs to a set of generative models**,** which means they are able to generate or produce new content.
- Given a training set, this technique learns to generate new data with the same statistics as the training set
- The generator is a neural network that models a transform function. It takes as input a simple random variable and must return, once trained, a random variable that follows the train distribution.
- The discriminator is another neural network. This neural network models a discriminative function. It takes as input an image and returns as output the probability of this image being a real or fake sample.
- GANs have also proven useful for semi-supervised learning, fully supervised learning and reinforcement learning.

DATASET: *CIFAR10*

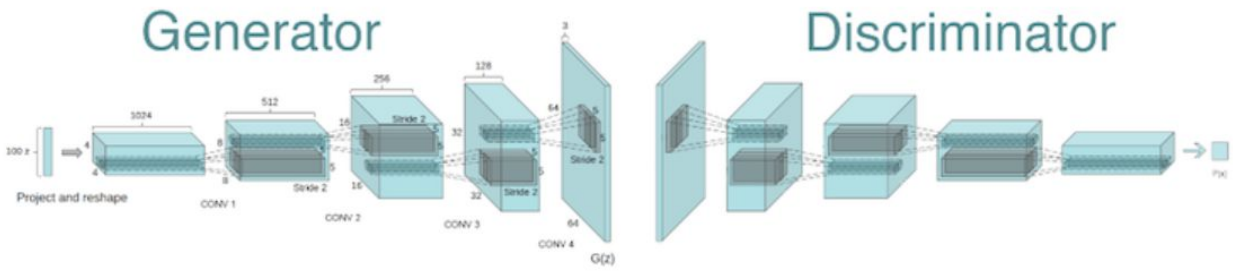
- The dataset contains 60,000 samples of images(32x32x3) that belong to 10 classes.
- The samples comprise of dogs, horses, automobiles, ship, aeroplane, birds, cats, deer, frog, truck.
- There are 6000 images in each class.

DCGAN:

- **Architecture:**

  The deep convolutional generative adversarial network has the following architecture. It consists of 2 neural networks called the **Discriminator(D)** and the **Generator(G)** network**.** Both are based on deep-convolutional neural networks.

  - **Changes to the model w.r.t vanilla GAN:**
    - All the pooling layers have been replaced by the strided convolution layers.
    - Use transposed convolution for upsampling.
    - Eliminate fully connected layers.
    - Use Batch normalization except the output layer for the generator and the input layer of the discriminator.
    - Use ReLU in the generator except for the output which uses tanh.
    - Use LeakyReLU in the discriminator.



Source: *Radford et al., 2015*

Discriminator model:

The discriminator has a series of convolution layers. The input to this network is a 32x32x3 image and the output is a probability between 0 and 1.

There are 4 convolution2D layers with spectral normalization, a dense layer with sigmoid activation as the last layer.

Generator model:

The generator has a series of transpose convolution layers. The input to this network is a random noise of some latent dimension and the output is an image of size 32x32x3, it is the generated image.

There are 4 convolution2DTranspose layers, a Dense Layer at the start of the network, and last layer has tanh as activation which enables the output values in the image to be between -1 to 1.

We combine both the models to form a generative adversarial network(DCGAN).

- **Spectral Normalization:**
    - It is a technique that normalizes spectral norm of weights in each layer so that it satisfies the **k-Lipschitz continuity constraint**.
    - The authors stress that a flat local minimum of a loss function generalizes better than a sharp one. To achieve this, they proved that it is sufficient to bound the spectral norm of the weight matrix at each layer.

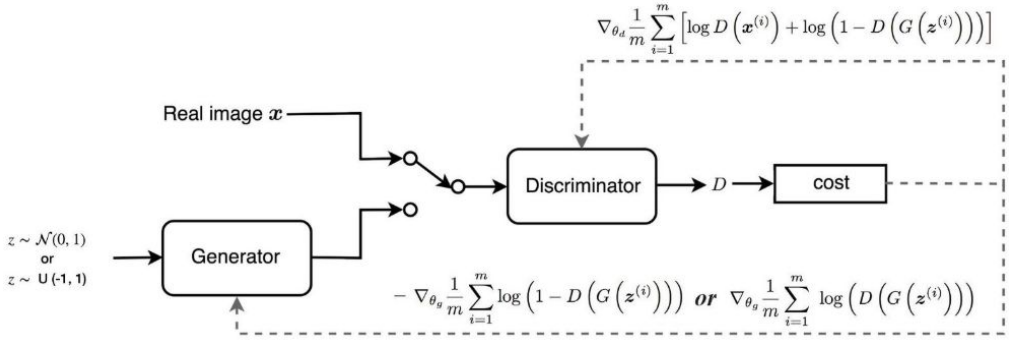
$$W_{SN}(W) := W/\sigma(W)$$

- Spectral normalization regularizes the gradient of W, preventing the column space of W from concentrating into one particular direction. This enables transformations of each layer to avoid becoming sensitive in one direction.
- For every weight in our network, we randomly initialize vectors u and v. we only need to perform a single power iteration on the current version of these vectors for each step of learning, this is why spectral normalization is more computationally efficient.

$$\sigma(W) = ||Wv|| = u^T W v.$$

- If a D(discriminator) is **Lipschitz** continuous, then we can draw a cone centered at every point on its graph such that the graph lies outside of that cone.

- **Training:**



The training happens in the following way:

- The generator is kept constant i.e. no training takes place on the generator model of the GAN.
- The discriminator is trained by passing samples from training and telling those are real(with label=1), passing samples generated by the generator and telling those are fake(with label=0).
- The discriminator model weights are updated accordingly.
- Then, we make the discriminator constant and train the generator based on the discriminators loss.
- The generator is trained to maximize the probability of the discriminator being incorrect.
- The discriminator is not updated in this operation but provides the gradient information required to update the weights of the generator model.
- The above process continues as a minimax game, each one trying to win over the other network.

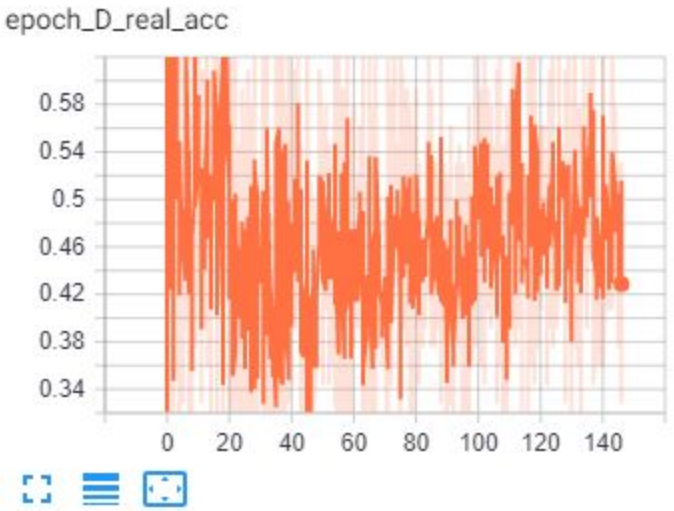
★ The goal of the discriminator is to tell accurately from where the input image has come from i.e. either real samples or fake samples.

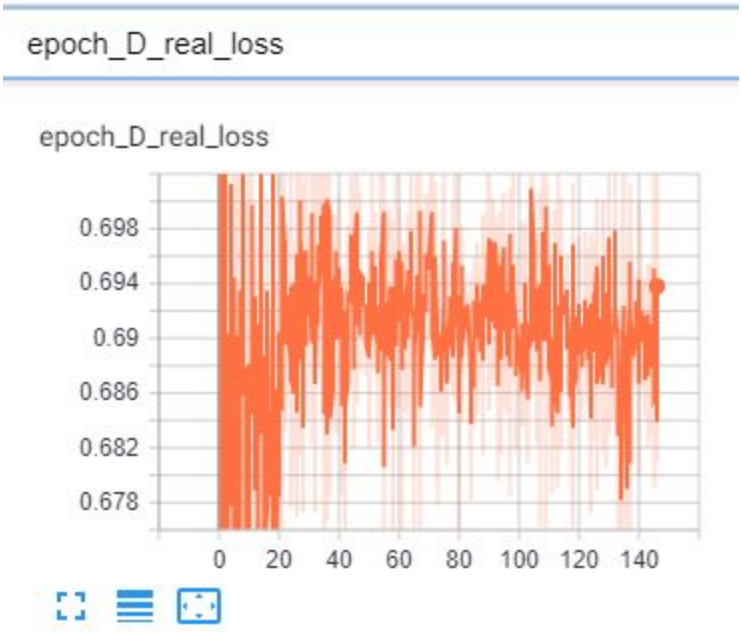
★ The goal of the generator is to produce images as real as the training samples, which can fake the discriminator.

The loss function used is called the **binary cross-entropy** loss function.

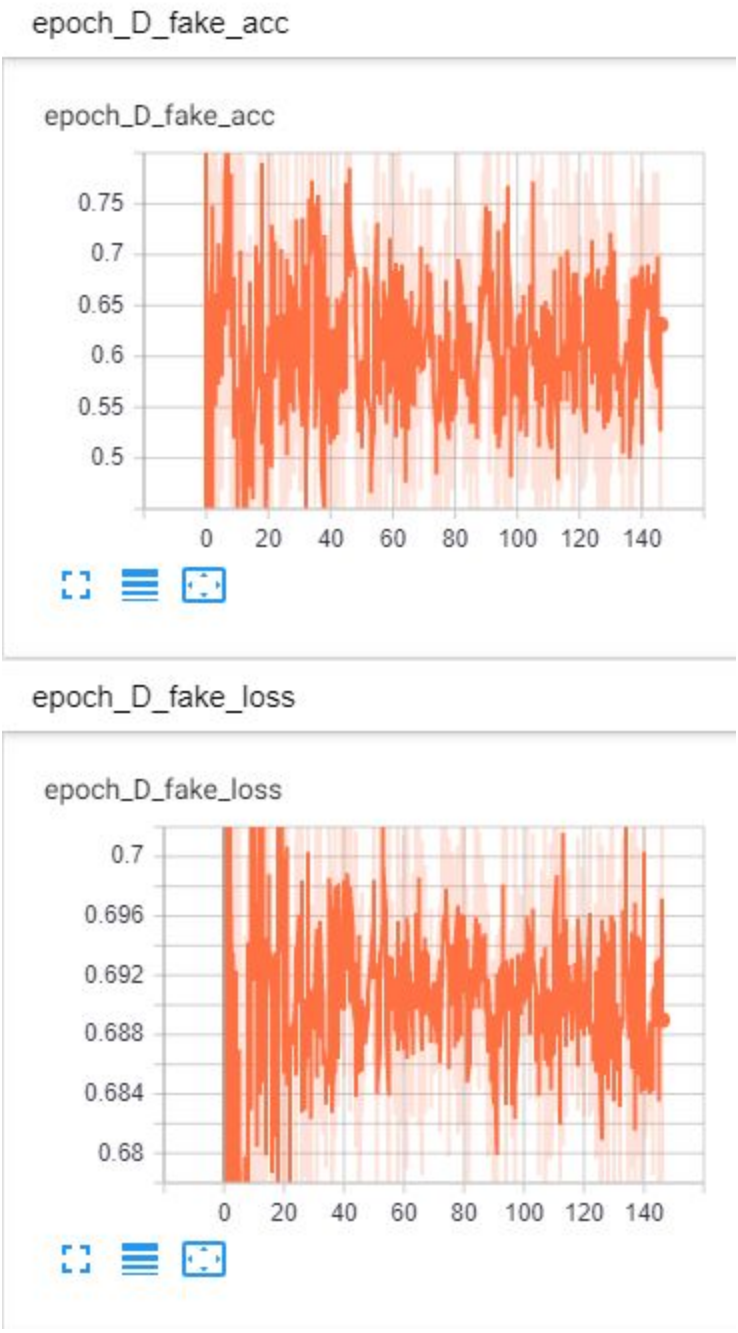
$$V(D,G) = E_{p_{data}}[\log(D(x))] + E_{p_z}[\log(1 - D(G(z)))]$$

- **LOSS and Accuracy PLOTS:**
    - **X-axis: No Of Epoch.**
    - **Y-axis: The loss values between 0 and 1.**
- **Discriminator model's loss and accuracy on Real images :**



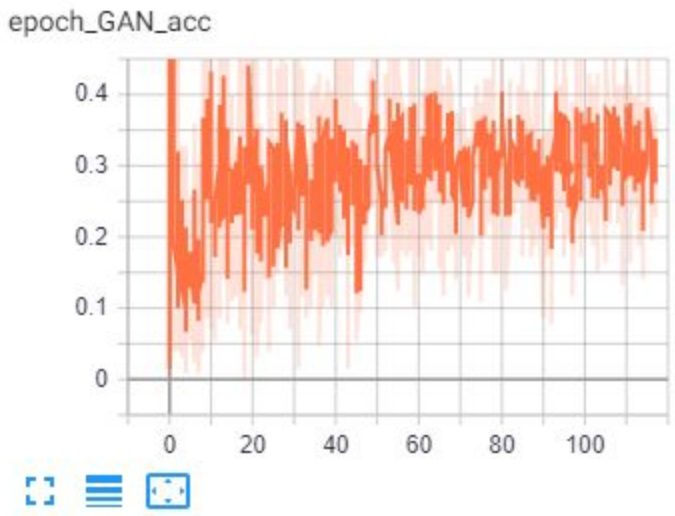


- **Discriminator model's loss and accuracy on fake images:**

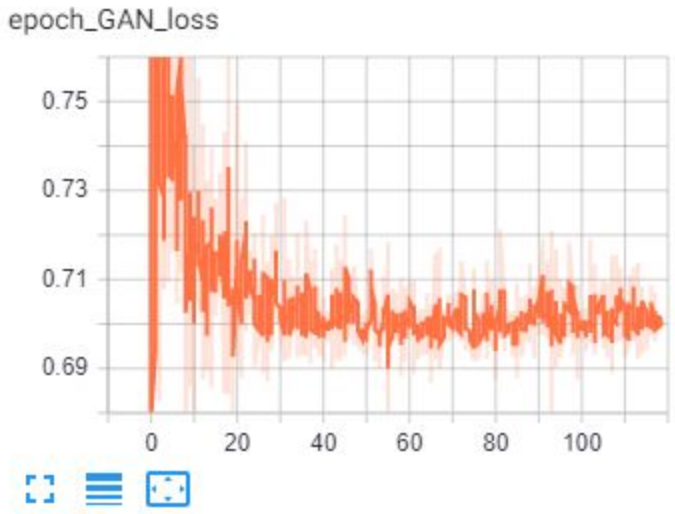


- We can observe from all the above plots that when discriminator performs better, generator is trying to fool it and when the generator is performing better, the discriminator is trying to classify it properly as real or fake.

- So, both the models fighting to perform better compared to one another. That is the reason for the graphs being very noisy, the fluctuations are rapid as each epoch discriminator is trained and then the generator is trained.
- **Generator model's accuracy and loss:**



epoch_GAN_loss

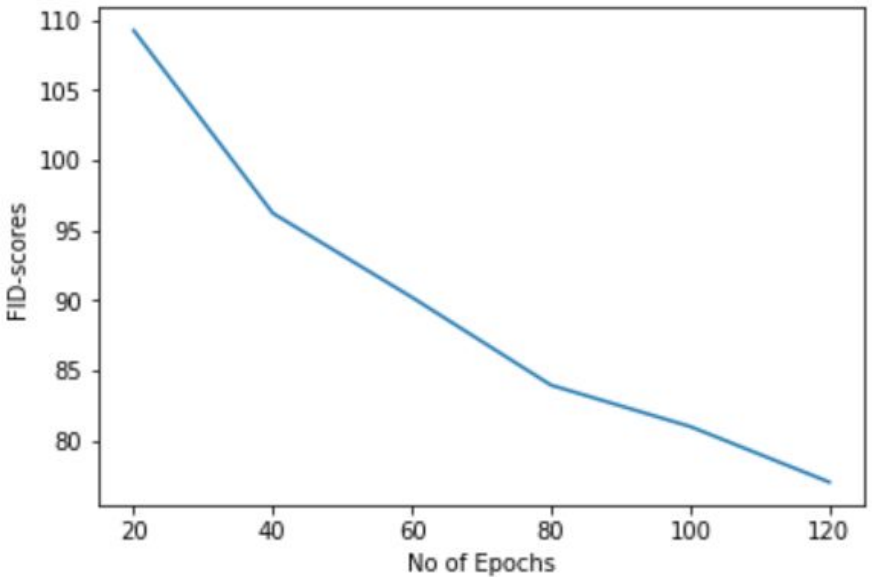


- **Evaluation Metric: [Frechet Inception Distance]:**
    - It is a metric used to calculate distance between feature vectors calculated for real images and generated images.
    - The score summarizes how similar the two groups are in terms of statistics on computer vision features of the raw images calculated using the inception v3 model used for image classification.
    - Lower scores indicate the two groups of images are more similar, or have more similar statistics, with a perfect score being 0.0 indicating that the two groups of images are identical.
    - Lower the FID score implies higher the quality of the image generated by generator in our scenario.

      Formula for FID calculation:

      $$d^2 = ||mu\_1 - mu\_2||^2 + Tr(C\_1 + C\_2 - 2*sqrt(C\_1*C\_2))$$

      FID Plot during the training process:

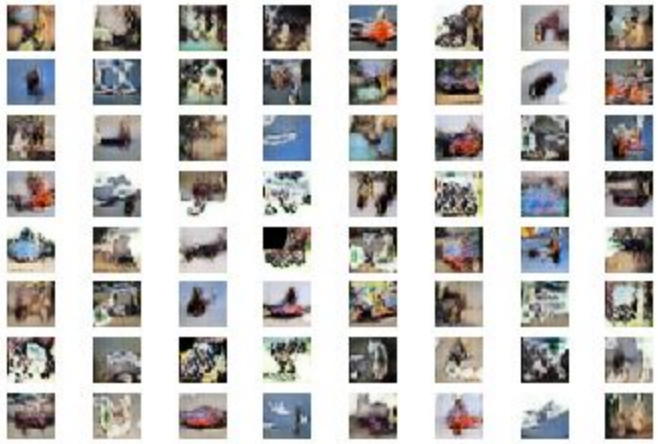


- The above graph shows us that the quality of images is improving as the number of epochs are increasing.
- The best **FID-score** achieved was **77.0583.**

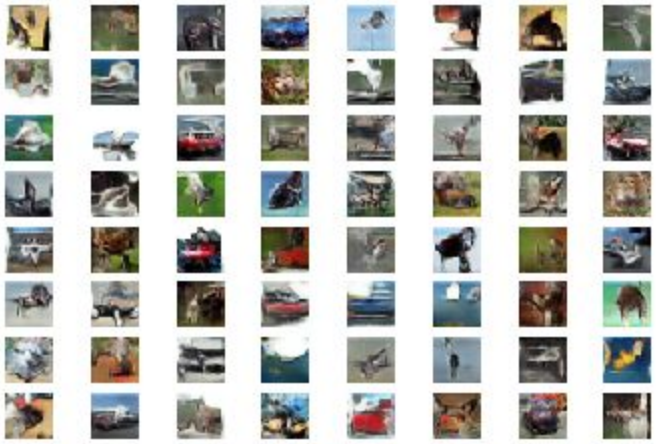
- **Trend of FID scores:**

No of Epochs	**FID scores**
20	109.25747
40	96.22475
60	90.230095
80	83.966255
100	81.01923
120	**77.05386**

- **Images generated by the generator:**
    - **Epoch 20**



- **Epoch 40**



- **Epoch 80**

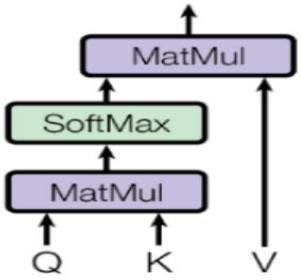
- **Epoch 120**

- From the above images, we can observe that the quality of generated images is improving over time and the number of classes its able to learn to generate is gradually increasing with the training time.

Self-Attention:

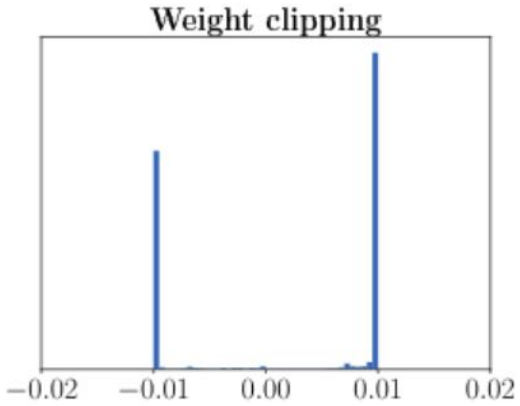
ATTENTION LAYER (custom implementation):

- A mechanism which captures the long-range and multi-level dependencies in the image.
- self-attention simply calculates the response at a single position as a weighted sum of the features at all positions.
- This mechanism allows the network to focus on areas of images that are widely separated yet have structural relevancy.
- In SAGAN, the self-attention module works in conjunction with the convolution network and uses the key-value-query model.
- This module takes the feature map, created by the CNN, and transforms it into three feature spaces. These feature spaces, called key f(x), value h(x), and query g(x), are created by passing the original feature map through three different 1x1 convolution maps. Key f(x) and query g(x) matrices are then multiplied. Next, the softmax operation is applied on each row of the multiplication result. The attention map generated from softmax identifies which areas of the image the network should attend to, the attention map is then multiplied with value h(x) to generate the self-attention feature map.
- Finally, the output is calculated by adding the original input feature map to the scaled self-attention map. The scaling parameter gamma is initialized to 0 at the beginning to cause the network to first focus on the local information. As the scaling parameter gamma gets updated during training, the network gradually learns to attend to the non-local areas of an image.
- This layer helps the network capture fine details form even distant parts of the image.



Weight Clipping:

- The goal of weight clipping is to satisfy the constraint of Lipschitz continuity.
- Before passing the weights for further computation the weight clipping ensures the model weights are in the specified range, to avoid the problem of vanishing gradients in the process of training.



- From the above image, we can see that weights are being clipped between -0.01 and 0.01
- As we replaced the Adam optimizer, to ensure the gradients don't vanish we make use of weight clipping.

Wasserstein Loss: (Earth Mover Distance(EMD)**)**

- Each distribution is viewed as a unit amount of "dirt" piled on M, the metric is the minimum "cost" of turning one pile into the other, which is assumed to

be the amount of dirt that needs to be moved times the mean distance it has to be moved.

- It encourages the discriminator to predict a score of how real or fake a given input looks. This transforms the role of the discriminator from a classifier into a critic for scoring the realness or fakeness of images, where the difference between the scores is as large as possible

Equation for calculating W Loss:

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

Conditional SAGAN:

- The conditional generative adversarial network, is a type of GAN that involves the conditional generation of images by the generator(G) model.
- This improvement in CGAN may come in the form of more stable training, faster training, or generated images that have a better quality.
- We provide a class of images to the generator along with the noise vector to make sure the generated image belong to the specified classes.
- A CGAN can be trained in such a way that both the generator and the discriminator models are conditioned on the class label.
- This means that when the trained generator model is used as a standalone model to generate images in the domain, images of a given type, or class label, can be generated.

Architecture:

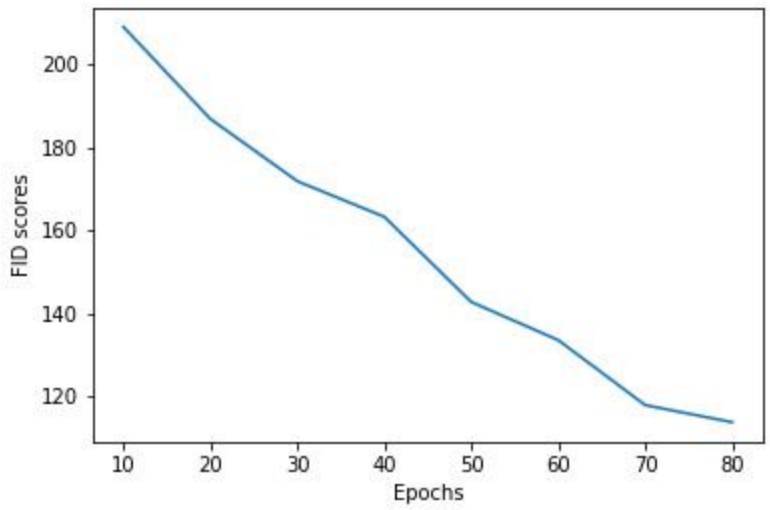
- The Discriminator(D) model has 4 convolution layers, LeakyReLU activations ,a Dense layer and an Attention layer at the layer which has image size (16x16).

- The Generator(G) model has 4 transposed convolution layers, LeakyReLU activations, and an Attention layer at the layer which has image size(16x16).

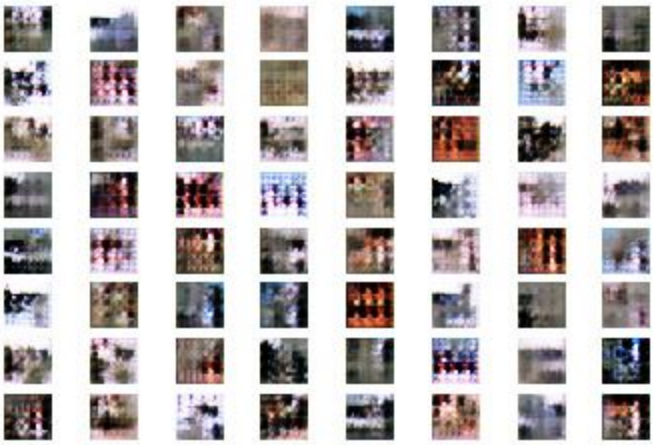
Implementation:

- The concatenation of the one-hot-vectors is done after the attention layer to ensure the attention map is not affected by the inclusion of the class labels in the layer preceding the attention layer.
- The input to generator is noise with latent dimensions is concatenated with the one-hot-vector representation of the class labels to be generated.
- The one-hot-vector concatenated with the random noise vector increases its shape by 10, which is passed to the fully connected layer of the generator.
- The discriminator is also passed with the same one-hot representation of the class labels used for generator and it is accordingly trained.

Evaluation Metric: [Frechet Inception Distance]



- **Images generated by the generator:**
    - **Epoch 20**



    - **Epoch 40**

Difference between SAGAN and DCGAN:

- DCGAN have difficulty in learning the image distributions of diverse multi-class datasets like Imagenet. Researchers observed that these DCGANs have difficulty in modeling some image classes than others when trained on multi-class datasets.
- The quality of the generated images are far superior in case of SAGAN, whereas the DCGAN generated images are comparable.
- From the FID scores we can tell that SAGAN have less score compared to the DCGAN FID scores, because the image produced are of better quality in SAGAN.
- The DCGAN was able to generate the texture of furs of dog but was unable to generate distinct legs, which is addressed in the SAGAN implementation.
- SAGAN training is much better compared to the DCGAN. Due to attention mechanism we implemented, the feature maps extract relevant features useful for further computations.
- This leads to better results compared to DCGAN as the local receptive features are not the only features learnt by the model, but it also learns long-range dependencies in the image.
- SAGAN takes more time to train compared to the DCGAN.
- The quality of images for each class can be validated, which is not the case with the DCGAN(random images).