A Detailed Study
of
Deep Convolutional Generative Adversarial Networks

Brief Introduction and Problem Statement

GANs(Generative Adversarial Networks), they've been hype during the last few years. These algorithms are unsupervised algorithms, which aims to learn the underlying structure of the given data, without specifying a target value. Generative models learn the intrinsic distribution function of the input data, allowing them to generate both synthetic inputs and outputs/targets.

This problem description is about training two generative models to generate new images from samples.I will be implementing two types of Generative Adversarial Networks (GANs): One known as the Deep Convolution GAN (DCGAN) and Self-Attention GAN (SA-GAN) using CIFAR-10 dataset.

The basic DCGAN consists of two neural networks: a discriminator (D) and a generator (G) which will compete with each other, making each other stronger at the same time. One of the simple variants of GAN in Deep Convolutional GANs (DCGANs) in which G and D are both based on deep convolutional neural networks.

There are a few setbacks in the DCGANs. Since DCGANs use only convolutional layers which has a local receptive field, long range dependencies can be only processed after passing through several layers. In order to learn the long term dependencies, we might need to increase the number of convolutional layers, which might increase the representational capacity of the model thus making the training slower and mode collapses may occur.

In order to overcome the shortcomings of the DCGANs we will implement a SAGAN using self-attention layers and WGAN.

We will be using the very simple CIFAR10(Canadian Institute for Advanced Research) dataset for training the GAN models.

Dataset : CIFAR10

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes
- There are 6000 images per class.
- There are 50000 training images and 10000 test images.
- The classes of Images are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck

DCGAN(Deep Convolutional GAN)

Architecture

Like all other GANs, the DCGAN has 2 neural networks stacked one after the other, a Discriminator(D) and the Generator(G) network.

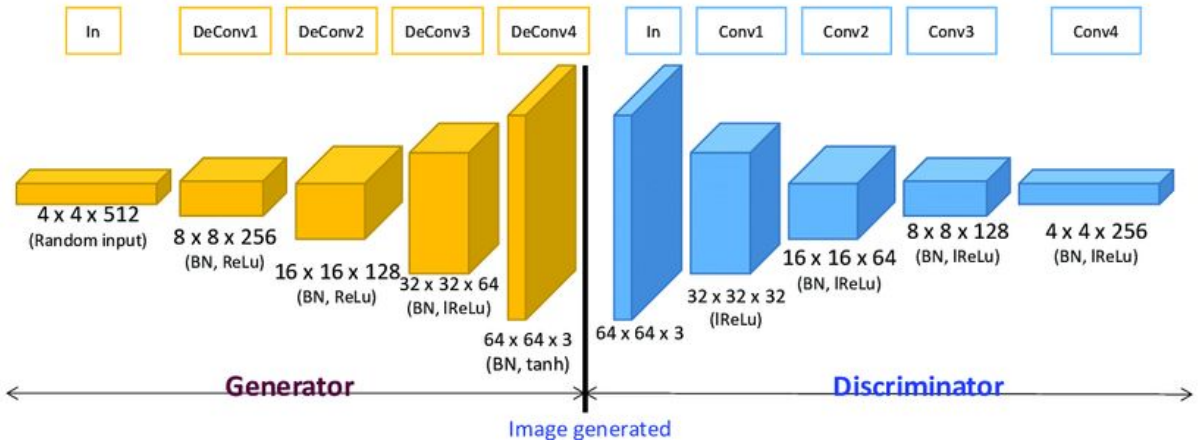

Fig: General structure of a DCGAN with generator and discriminator networks

Generator Model

- The input to this network is a random noise vector of the size of the latent vector dimension. The generator has a series of transpose convolution layers.

- There are 4 convolution2DTranspose layers(upsampling) which upscale or downscale the images extracting the feature maps. Batch normalization and ReLu is applied with the convolution2D layers.
- The last layer has tanh activation function which enables the output values in the image to be between -1 to 1 which aids in fast computation.
- The result is a 32x32x3 image in our case.

Discriminator Model

- The discriminator model has 4 convolutional 2D layers(downsampling).
- The first three layers are accompanied by batch normalization and leaky ReLu layers. While the last layer is a dense layer with sigmoid function.

The Generator and Discriminator nets were put together to form a GAN network.

Batch Normalization

Normalization of any data involves finding the mean and variance of the data and normalizing the data so that the data has 0 mean and unit variance. In batch normalization we will normalize each hidden unit activation. Consider we have d number of hidden units in a hidden layer of any Deep neural network. If the activation values of the layers is given as x=[x1,x2,……xd].
The formula for calculating hidden unit activation for Kth hidden unit is

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Where,
E(x^k) is the expectation of the kth hidden units values/mean value
Var(x^k) is the variance of the kth hidden unit

Training of the DCGAN

- Initially the generator is not trained and is kept idle. While the discriminator is trained by passing the samples from training along with their ground truth labels.

- The discriminator model weights are updated and it learns from the training and provides a probability of the image given being a real image.
- Then the generator is given a random noise vector and is allowed to train based on the discriminators loss while keeping the discriminator idle.
- The generator is trained to cheat the discriminator. The discriminator is not updated during this operation but it provides the gradient information required to update the weights of the generator model. While the discriminator tries to outsmart the generator during the next iteration of the run. This happens like a cat and mouse chase.

  The loss function used for training is called the binary cross-entropy(BCE) which is given by the formula

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

Evaluation Metrics for DCGAN: FID Score

- Most of the machine learning models use accuracy as the evaluation metrics. But for GANs we use the Fréchet Inception Distance or FID Score.
- FID Score in literal terms measures how close the distribution of the data in the generated sample is to the real images. We calculate the distance between the feature vectors of the real and fake images.
- For the calculations, we use the Inception V3 model which is used for image classification with a simple tweak.
- The Inception V3 model is fed with an image and the activation input of the layer before the last is collected and used for distance calculations.

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

- Lower FID score indicates poor performance of the generator. I.e. the generator has not learnt the data distribution good enough to replicate the training data accurately.

- A FID score around 300 -150 is considered as okay, while a score of 30 or less is considered state of the art for GANs which is good enough to cheat human eyes.

Graphs for DCGAN

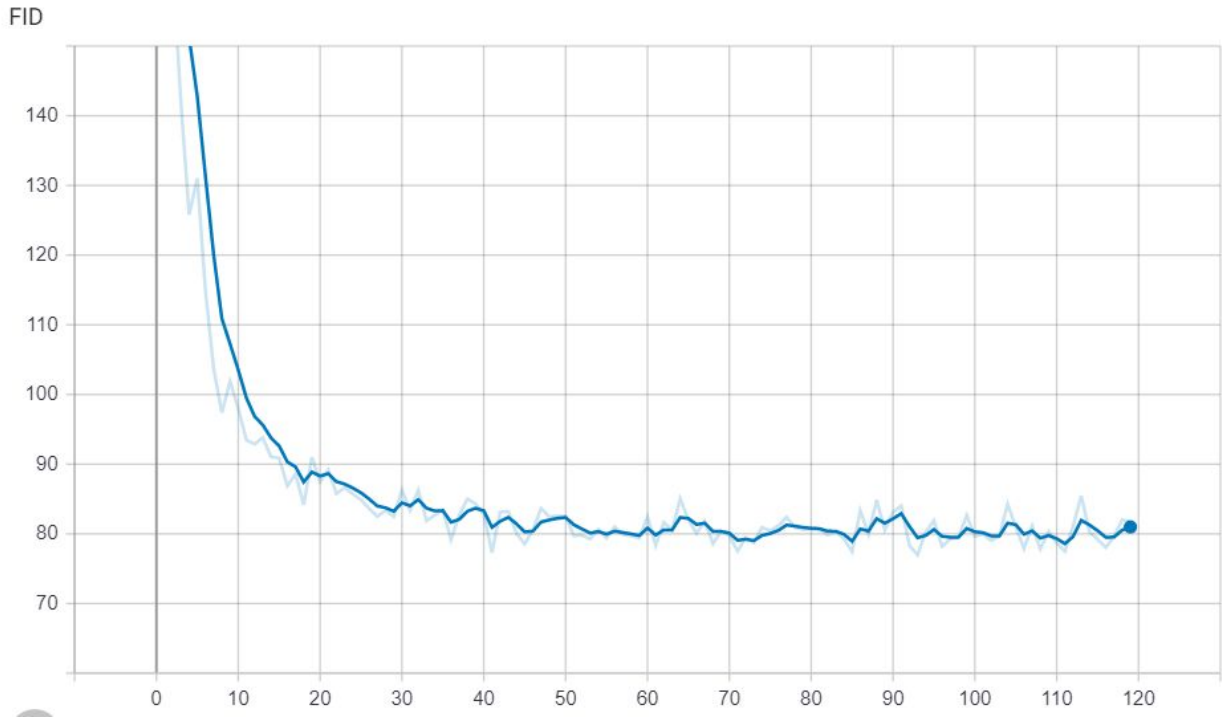

Fig: FID vs Epohs graphs for DCGANs

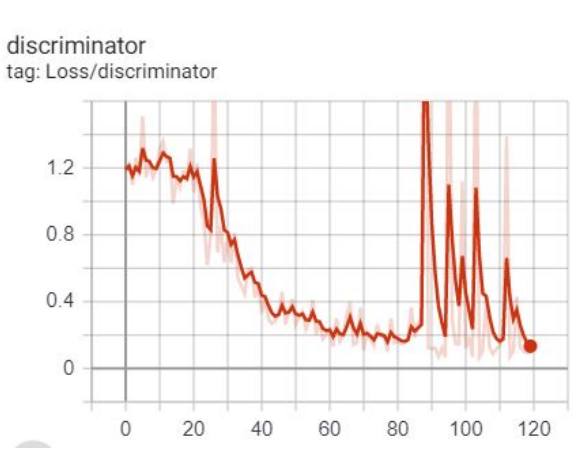

Fig: Discriminator Loss Vs Epochs

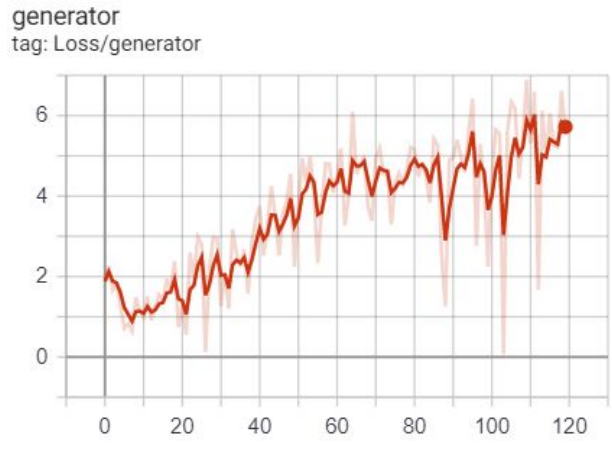

Fig: Generator Loss Vs Epochs

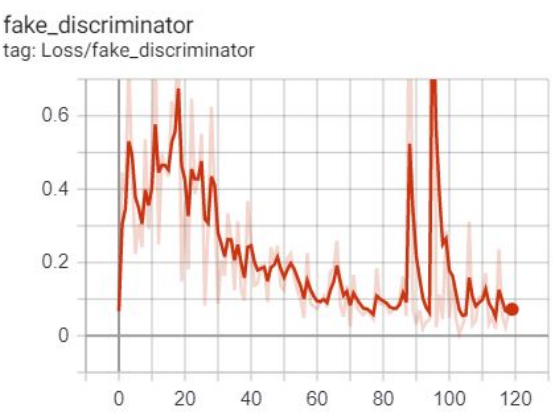

Fig: Fake Loss Vs Epochs

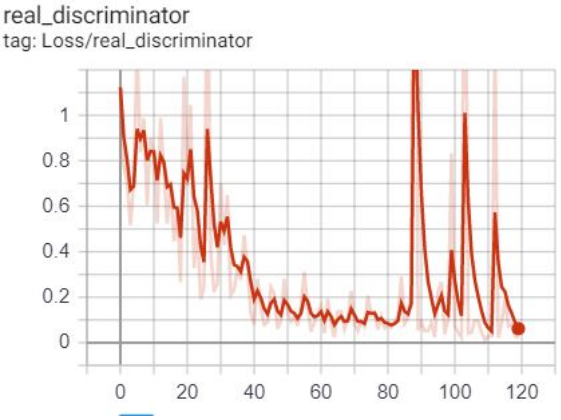

Fig: Real Loss Vs Epochs

Fig: Fake Images Generated during epoch 5

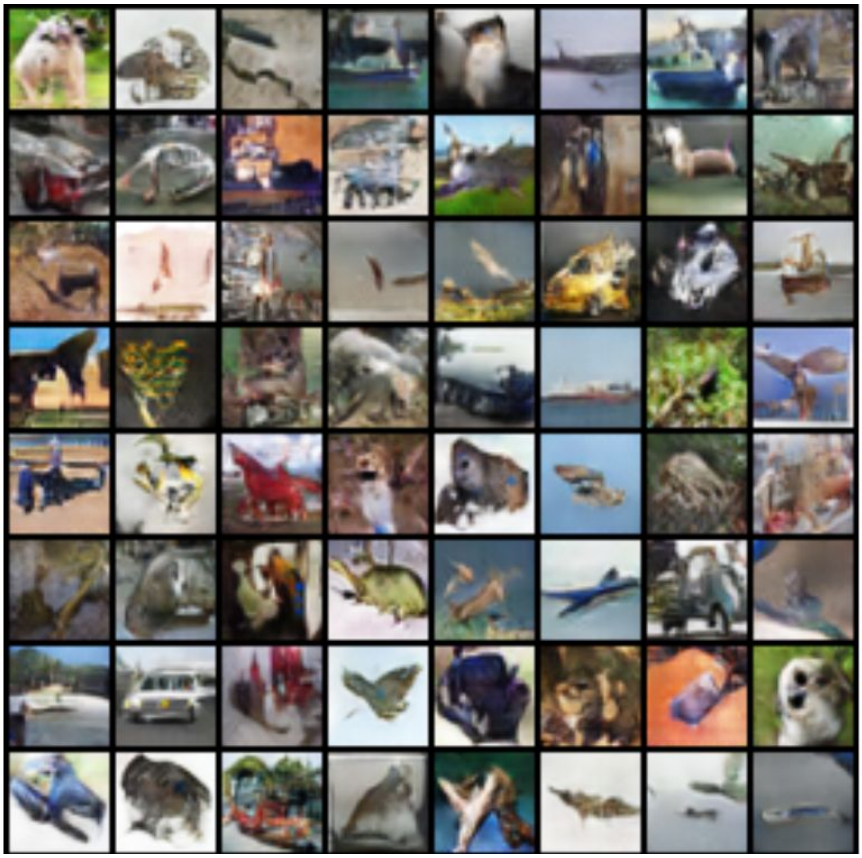

Fig: Fake Images generated after 120 epochs

FID Results

Epochs	FID Score
0	320
5	142
10	98
20	88
40	82
60	80
80	80
100	80
120	80

Table: Epochs Vs FID scores

References

1. https://github.com/pytorch/examples/tree/master/dcgan
2. https://christiancosgrove.com/blog/2018/01/04/spectral-normalization-explained.html
3. https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/
4. https://medium.com/@sunnerli/the-story-about-wgan-784be5acd84c
5. https://developers.google.com/machine-learning/gan/loss
6. https://www.cs.toronto.edu/~kriz/cifar.html
7. https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html