Chapter 1 : Introduction

1.1 Motivation

Traditional machine learning approaches when introduced had given promising result in field of Artificial Intelligence. But deep learning a part of broader family of machine learning has changed the face of traditional machine learning. In deep learning, methods are based on data representation as opposed to task specific algorithm (or input output based learning).

One game changing model was introduced in field of deep learning in 2014 known as Generative Adversarial Network(GAN) [3] by Ian J. Goodfellow. This technique generates photo realistic images that look authentic to human observer.

GAN is implemented by a system of two neural network fighting against each other. One network generates candidates and other discriminate/evaluate them. Generative network learns to map from a latent space to a particular data distribution of interest while discriminative network discriminates between instance from true data distribution and candidate produced by generator.

Since 2014, a tremendous work had been done to further explore and exploit this technique. A lot of good result had been obtained till yet starting from generation of photorealistic image to generation of any multimedia like voice, video etc.

One such problem that can be beautifully solved by GAN is “Text to Image Synthesis”.

1.2 Objectives and Scope

The objective of the project is to synthesize images from text description. Since many powerful recurrent architectures have been proposed to learn the generality and discriminative power of text description and concurrently deep convolutional generative adversarial networks have started generating images of specific categories. We are trying to combine the advances in both the fields by making a novel model through the formulation of GAN.

Our ambition here is to learn mapping from words and characters to image pixels such that generated images can be mistaken as real by human, which can be used as a base to generate videos from text descriptions

Chapter 2 : Literature Review

2.1 Generative Adversarial Networks

These networks are used for generative modelling in which we wish to fit a function on a probability distribution and the most effective way to fit a function to a distribution is through neural networks.

Fig 2.1 Data Distribution

This network while training takes a random noise as input which passes through the generator (G)which performs deconvolution to generate the a image(Gi) , Gi combined with real images (Ri) are passed through the discriminator(D) which performs convolution on the feature maps and checks whether the image generated by G is real or not , if not real the generator and discriminator is trained through backpropagation .The process is continued till D is not fooled by the images generated by G to be real.

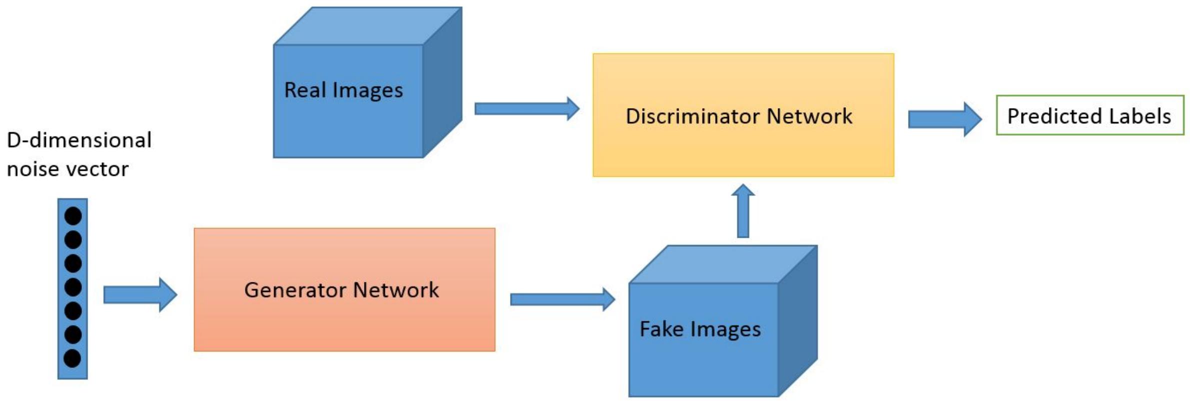


Fig 2.2 GAN -[4]

Here the whole model is fit into a equation by :-

$$min_G\ max_D\ V(D,G) = E_{x-p_{data}(x)}[log\ D(x)] + E_{z-p_{data}(z)}[log(1 - D(G(z)))] \qquad \text{-(Equation 1)}$$

Here,

Pdata(x) -> the distribution of real data

X -> sample from pdata(x)

P(z) -> distribution of generator

Z -> sample from p(z)

G(z) -> Generator Network

D(x) -> Discriminator Network

2.2 Parts of training GAN:

So broadly a training phase has two main subparts and they are done sequentially

- Pass 1: Train discriminator and freeze generator (freezing means setting training as false. The network does only forward pass and no backpropagation is applied)

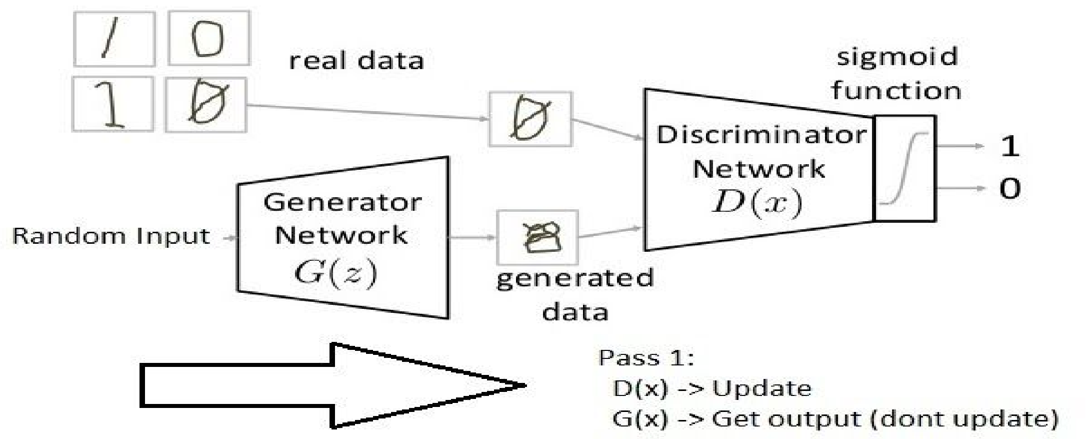


Fig 2.3 Training Discriminator

- Pass 2: Train generator and freeze discriminator

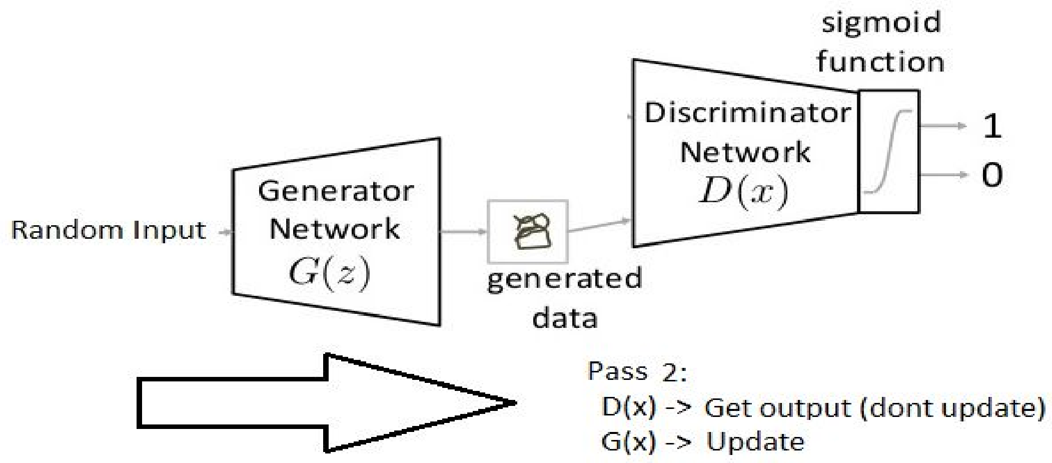


Fig 2.4 Training Generator

2.3 Generator Network:

The Generator is a network made from the combination of several deconvolution layers and fully connected layer. The deconvolution layers densify the sparse activations obtained by unpooling through convolution-like operations with multiple learned filters. However, contrary to convolutional layers, which connect multiple input activations within a filter window to a single activation, deconvolutional layers associate a single input activation with multiple outputs, as illustrated in Figure . The output of the deconvolutional layer is an enlarged and dense activation map..

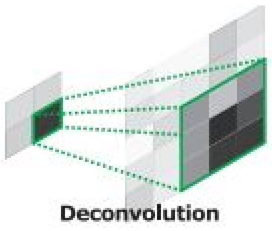
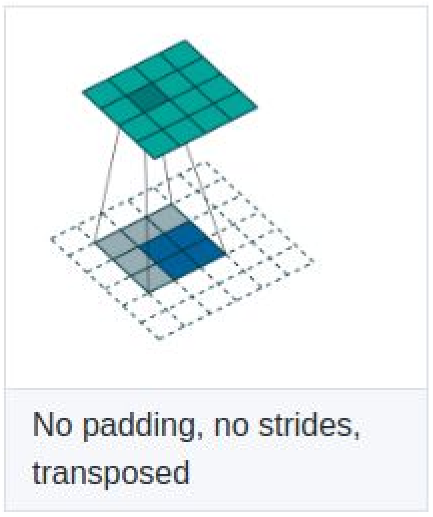
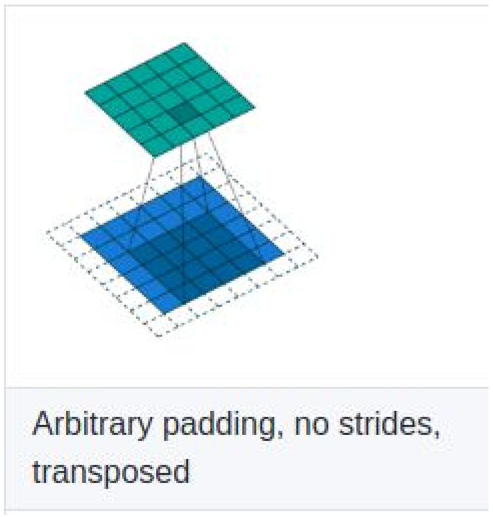


Fig 2.5: Illustration of Deconvolution.

The learned filters in deconvolutional layers correspond to bases to reconstruct shape of an input object. Therefore, similar to the convolution network, a hierarchical structure of deconvolutional layers are used to capture different level of shape details. The filters in lower layers tend to capture overall shape of an object while the class-specific fine details are encoded in the filters in higher layers. In this way, the network directly takes class-specific shape information into account for semantic segmentation**[8].**

Initially the 100x1 noise is combined with encoded skip thought vectors to form a vector of 356x1 which is passed on to several deconvolution layers to form final image as shown in figure .





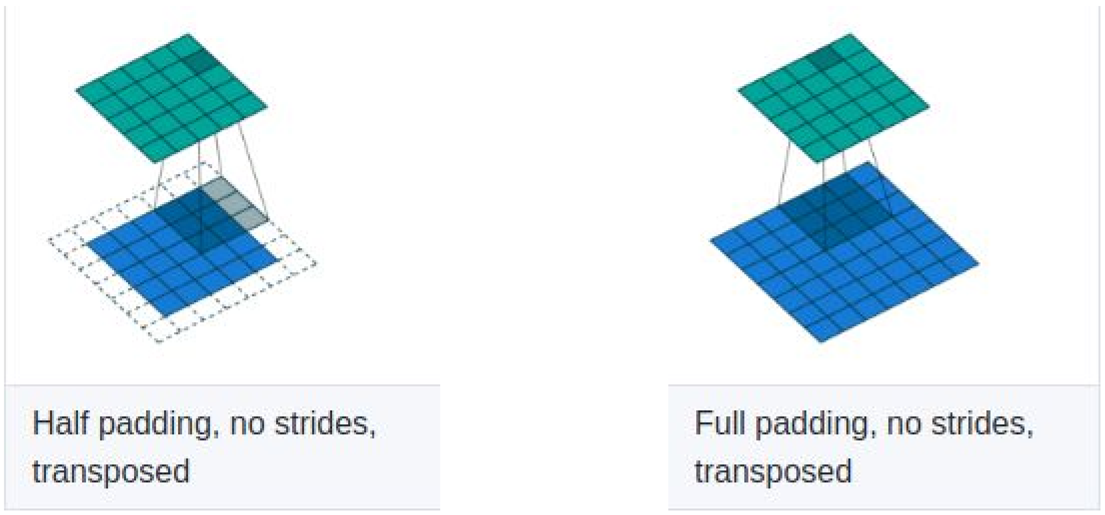

Half padding, no strides, transposed
Full padding, no strides, transposed

Fig 2.6 : Deconvolution formats .

Deconvolution can be performed in different ways as using variation in padding, strides length and space as shown in figure we chose full padding, no strides format since it gives best results**[9].**

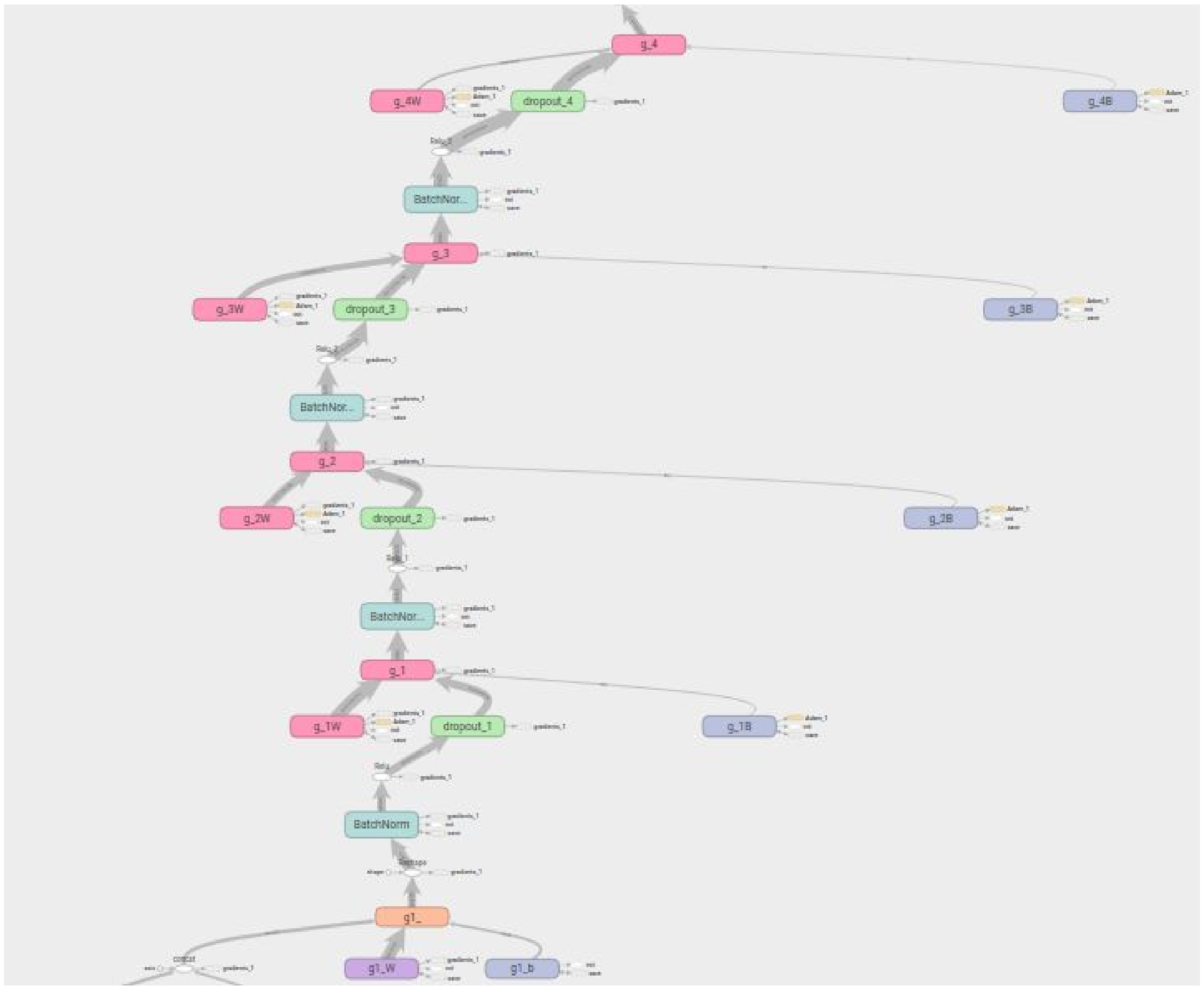


Fig 2.7 : Generator network in main architecture.

2.4 Discriminator Network:

Task of discriminator network is to guide generator network in its journey towards real data distribution by providing gradients to update its hyperparameters. Discriminator network is basically an convolution neural network(CNN) which take high dimensional image as input and output a single bit i.e either real or fake.

CNN's are deep, feedforward artificial neural network. Remember that feed-forward neural networks are also called multi-layer perceptrons(MLPs), which are the quintessential deep learning models. The models are called "feed-forward" because information flows right through the model. There are no feedback connections in which outputs of the model are fed back into itself.

CNNs specifically are inspired by the biological visual cortex. The cortex has small regions of cells that are sensitive to the specific areas of the visual field. This idea was expanded by a captivating experiment done by Hubel and Wiesel in 1962. In this experiment, the researchers showed that some individual neurons in the brain activated or fired only in the presence of edges of a particular orientation like vertical or horizontal edges. For example, some neurons fired when exposed to vertical sides and some when shown a horizontal edge. Hubel and Wiesel found that all of these neurons were well ordered in a columnar fashion and that together they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks is one that machines use as well and one that you can also find back in CNNs.

Convolutional neural networks have been one of the most influential innovations in the field of computer vision. They have performed a lot better than traditional computer vision and have produced state-of-the-art results. These neural networks have proven to be successful in many different real-life case studies and applications, like:

1. Image classification, object detection, segmentation, face recognition;
2. Self driving cars that leverage CNN based vision systems;
3. Classification of crystal structure using a convolutional neural network;
4. And many more, of course!

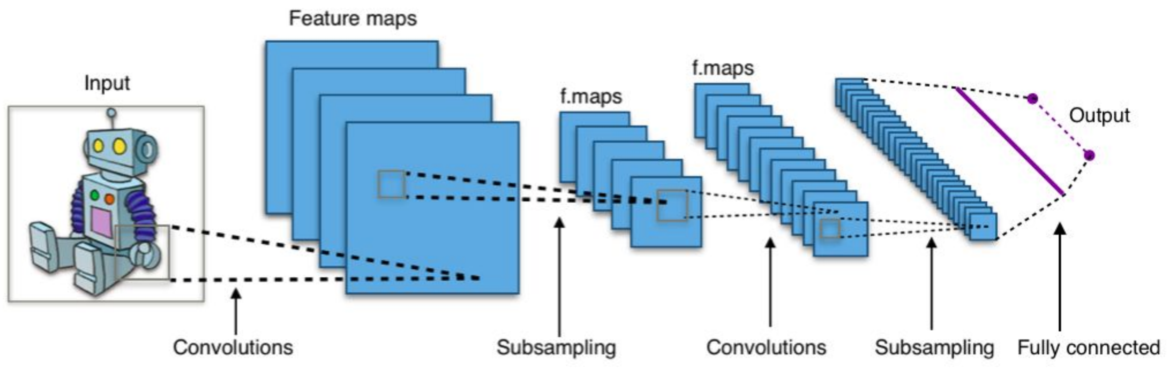


Fig 2.8: Convolutional Neural Network from Wikimedia.

The image shows you that you feed an image as an input to the network, which goes through multiple convolutions, subsampling, a fully connected layer and finally outputs something.

But what are all these concepts?

The convolution layer computes the output of neurons that are connected to local regions or receptive fields in the input, each computing a dot product between their weights and a small receptive field to which they are connected to in the input volume. Each computation leads to extraction of a feature map from the input image. In other words, imagine you have an image represented as a 5x5 matrix of values, and you take a 3x3 matrix and slide that 3x3 window or kernel around the image. At each position of that matrix, you multiply the values of your 3x3 window by the values in the image that are currently being covered by the window. As a result, you'll get a single number that represents all the values in that window of the images. You use this layer to filtering: as the window moves over the image, you check for patterns in that section of the image. This works because of filters, which are multiplied by the values outputted by the convolution.

The objective of subsampling is to get an input representation by reducing its dimensions, which helps in reducing overfitting. One of the techniques of subsampling is max pooling. With this technique, you select the highest pixel value from a region depending on its size. In other words, max pooling takes the largest value from the window of the image currently covered by the kernel. For example, you can have a max-pooling layer of size 2 x 2 will select the maximum pixel intensity value from 2 x 2 region. You're right to think that the pooling layer then works a lot like the convolution layer! You also take a kernel or a window and move it over the image; The only difference is the function that is applied to the kernel and the image window isn't linear.

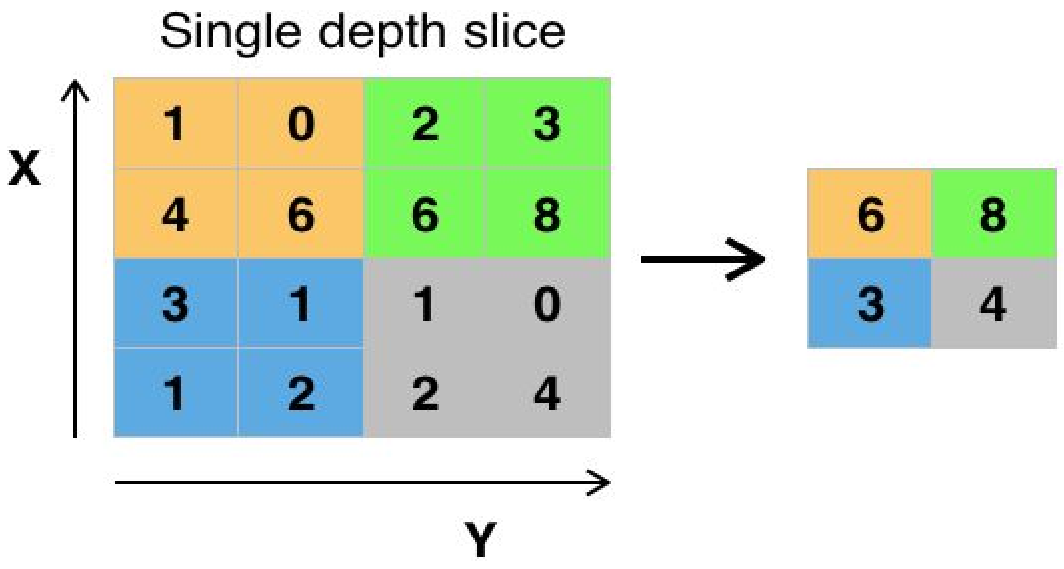


Fig 2.9: Max-Pooling from Wikipedia.

The objective of the fully connected layer is to flatten the high-level features that are learned by convolutional layers and combining all the features. It passes the flattened output to the output layer where you use a softmax classifier or a sigmoid to predict the input class label.

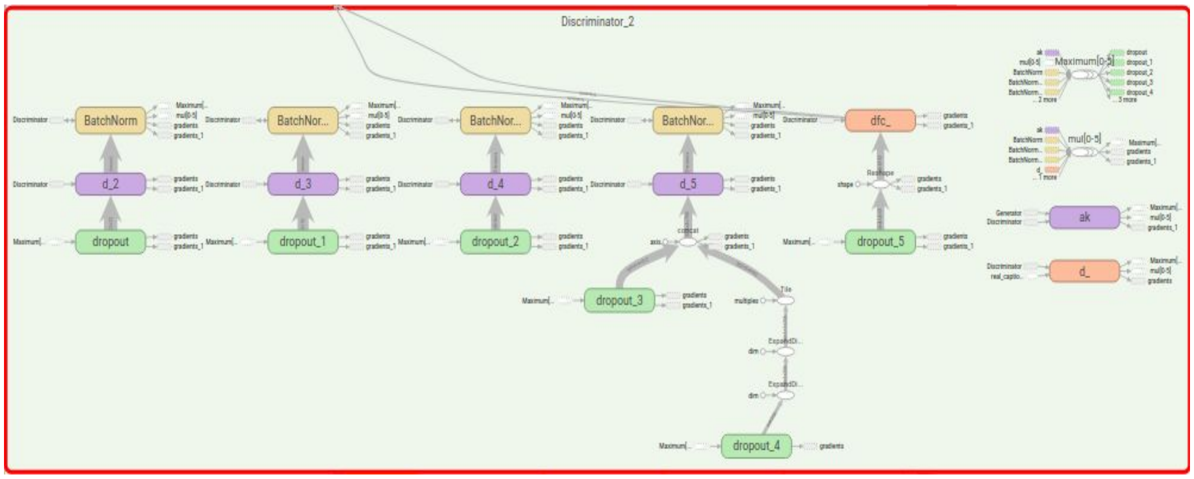


Fig 2.10 : Discriminator network in main architecture..

2.5 Generative Adversarial Text to Image Synthesis

As suggested in the [3] (Good Fellow paper) that the probability function fitted on the distribution can be conditional [1] (reed paper) used the text encoding as condition which will be on both discriminator and generator.
[1](reed paper) suggests that provided the text encoding of text with the noise as the combined input we can train the generator to generate images from the text given that text encoding is provided to both G and D .

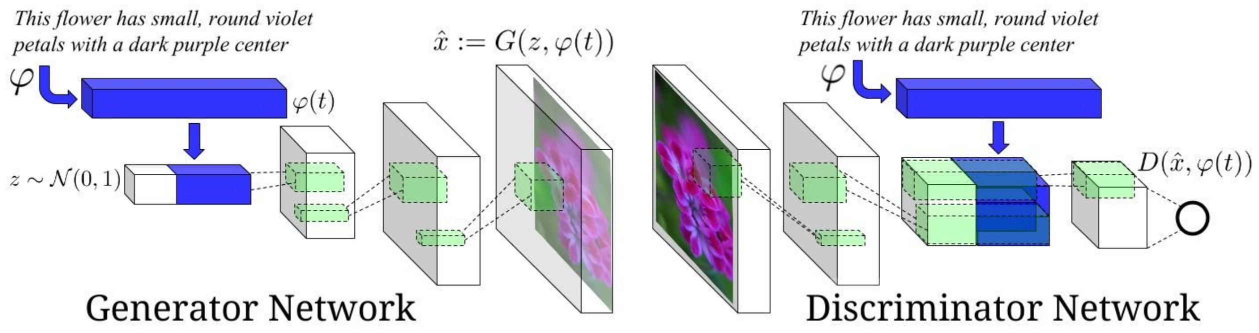

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Generator Network
Discriminator Network

Fig 2.11 Conditional GAN for text to image synthesis

Chapter 3 : Work Done

3.1 :- Part 1:

Implementation of such system involves combination two technique first is to convert text description into latent space/embeddings this could be done by using an autoencoder. Second is to train a GAN using hidden layer of autoencoder as input to GAN.

We mainly used the Oxford-102 Flowers dataset along with five text descriptions per image we collected as our evaluation setting.

We had tried to generate text embedding using pre-trained autoencoder of skip thought network [2] on an i5 processor having 4GB of RAM. But the system was not able to load the dictionary of words used for tokenization, every time system failure occurred. Problem was limited space of RAM. Then we moved to a system having 32GB of RAM with Xeon processor here the process was able to start successfully but the task of generation of text embedding from text description took around 4 days even though we were not training the model instead we were exploiting the pre-trained network to generate text embeddings. Here we figured out that the problem was that huge amount of computation are involved (i.e. Multiplication/Addition of large matrices) to generate text embedding.

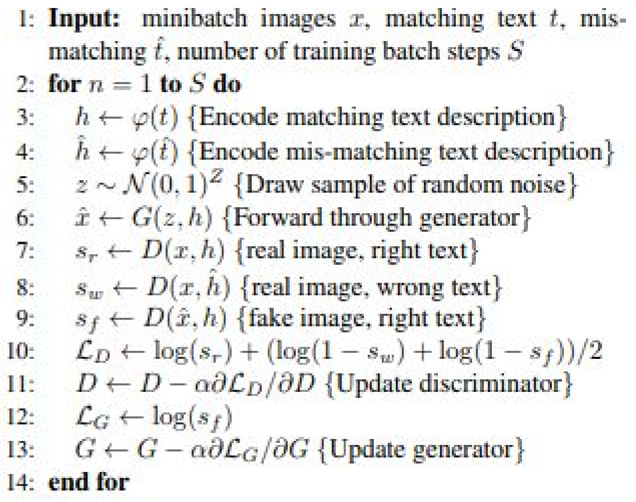
Working of deep neural network requires high computation, but CPU works sequentially at max we can have some limited number of thread running but our requirement was large amount of parallel computation. For this reason, Graphics Processing Unit(GPU) have helped. GPU have a parallel architecture with thousands of computation cores that work all together at same time. This is our basic requirement to run a deep network.

So, we finally moved to a system having Nvidia GeForce GTX 1080 GPU, with i7 7th generation processor with 32GB of RAM. Here we have started everything from scratch to setup the environment and install the tool's required to use GPU for arithmetic computation. Our first task was to install Ubuntu (a Linux based OS) and to configure and install Cuda drivers for GPU utilization. After this task of configuration and installation had accomplished (took a lot of time due to various kind of dependency issue) then we finally made use of docker container for utilizing deep learning tools.

At last we were finally able to generate text embedding and the overall process of encoding took less than 40 minutes (the same process took 4 days on Xeon based processor).

3.2 :- Part 2:

Algorithm:-



```
1: Input: minibatch images $x$, matching text $t$, mis-
   matching $\hat{t}$, number of training batch steps $S$
2: for $n = 1$ to $S$ do
3:     $h \leftarrow \varphi(t)$ {Encode matching text description}
4:     $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
5:     $z \sim \mathcal{N}(0,1)^Z$ {Draw sample of random noise}
6:     $\hat{x} \leftarrow G(z,h)$ {Forward through generator}
7:     $s_r \leftarrow D(x,h)$ {real image, right text}
8:     $s_w \leftarrow D(x,\hat{h})$ {real image, wrong text}
9:     $s_f \leftarrow D(\hat{x},h)$ {fake image, right text}
10:    $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1-s_w) + \log(1-s_f))/2$
11:    $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
12:    $\mathcal{L}_G \leftarrow \log(s_f)$
13:    $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
14: end for
```

Fig 3.1 Algorithm for training the Conditional GAN

The Generative adversarial networks takes three inputs {real image,right text}, {real image,wrong text},{fake image,right text} which give s_r, s_w, s_f losses respectively as stated in line 7, 8, 9. L_D is the total loss of discriminator, L_G is the loss of generator using which the weights of the models will be updated.

Result of the first training on flower dataset-

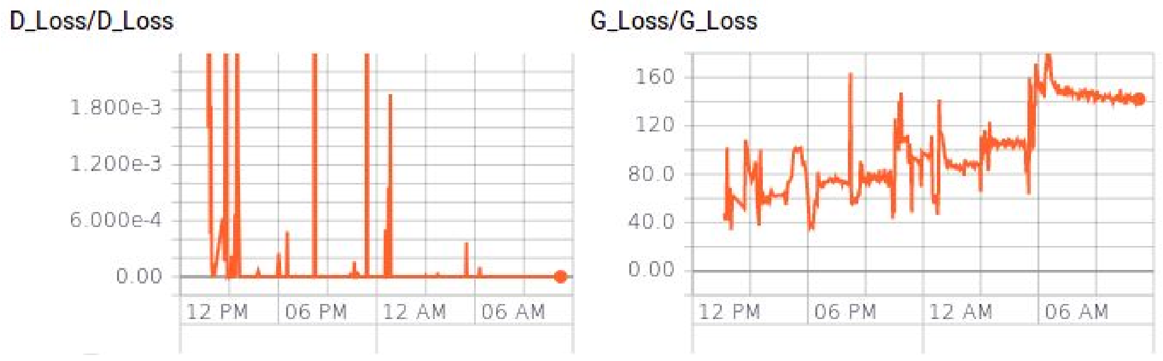


Fig 3.2 Graph of first training.

Here clearly discriminator and generator loss are far away from each other but for properly training a GAN network difference between the loss of two network should be small. The problem is vanishing gradients. Discriminator can easily identify real and fake images hence its loss is close to zero which when back propagates result in very small gradient hence weights of generator can't be trained there by gradually increasing loss of generator network as shown in the plot, according to (equation 1).

Q. So what is the problem ?

A. There are many reason, some we figured out are as follows.

We figure out that our network was having some flaws. Firstly, we figured out that it is important that how we are going to initialize our weights. We found that weights of convolution and deconvolution layers should have mean zero and standard deviation 0.02 mathematically explained in **[5]**. Figure 3.5 shows that the mean of the weights after training comes near to zero and this specifies that the training of the model has been done properly.

One of the intuition behind batch normalization is output of activation function in hidden layer is input to next layer and if we scale the that output using normalization than we will achieve convergence a little faster, it also helps in reducing internal covariate shift both the properties mentioned in **[7]**.

In our initial architecture we place bach normalization at output of last deconvolution layer but output is supposed to be a image(i.e non scaled) so batch normalization is not required.

During training of GAN it has been seen that if loss of generator network remains below discriminator loss for achieving convergence. So, for a single batch we update weight of discriminator once and that of generator thrice.

After modifying the architecture we finally get convergence as shown in figure 3.3.

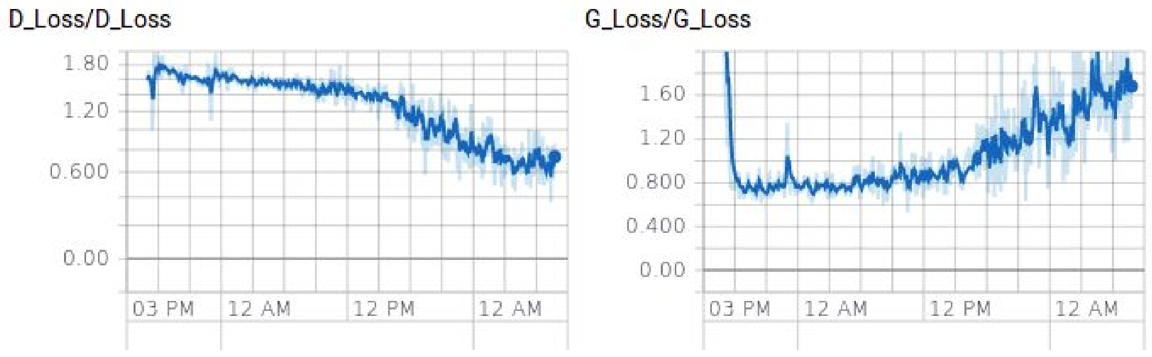
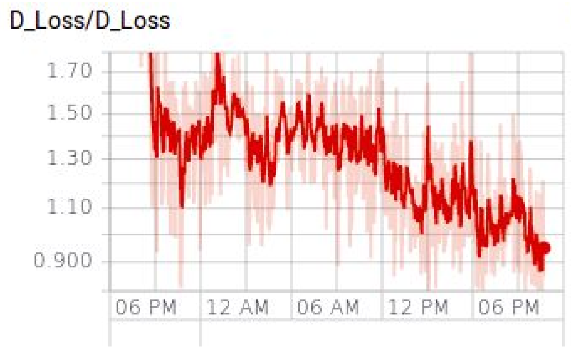


Fig 3.3 Graph of second training.

Finally, we applied Dropout**[6]** on the layers in the discriminator and the models converged in better fashion with lesser gap between the discriminator and generator as shown in figure 3.4. We know that Dropout with some probability ‘p’ removes some neurons from the layer while training this helps the models to not overfit on the dataset.



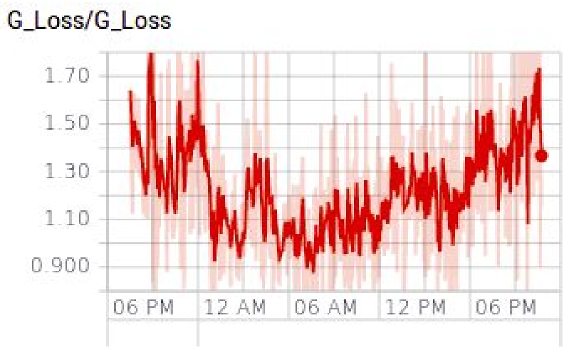


Fig 3.4 Graph of third training.

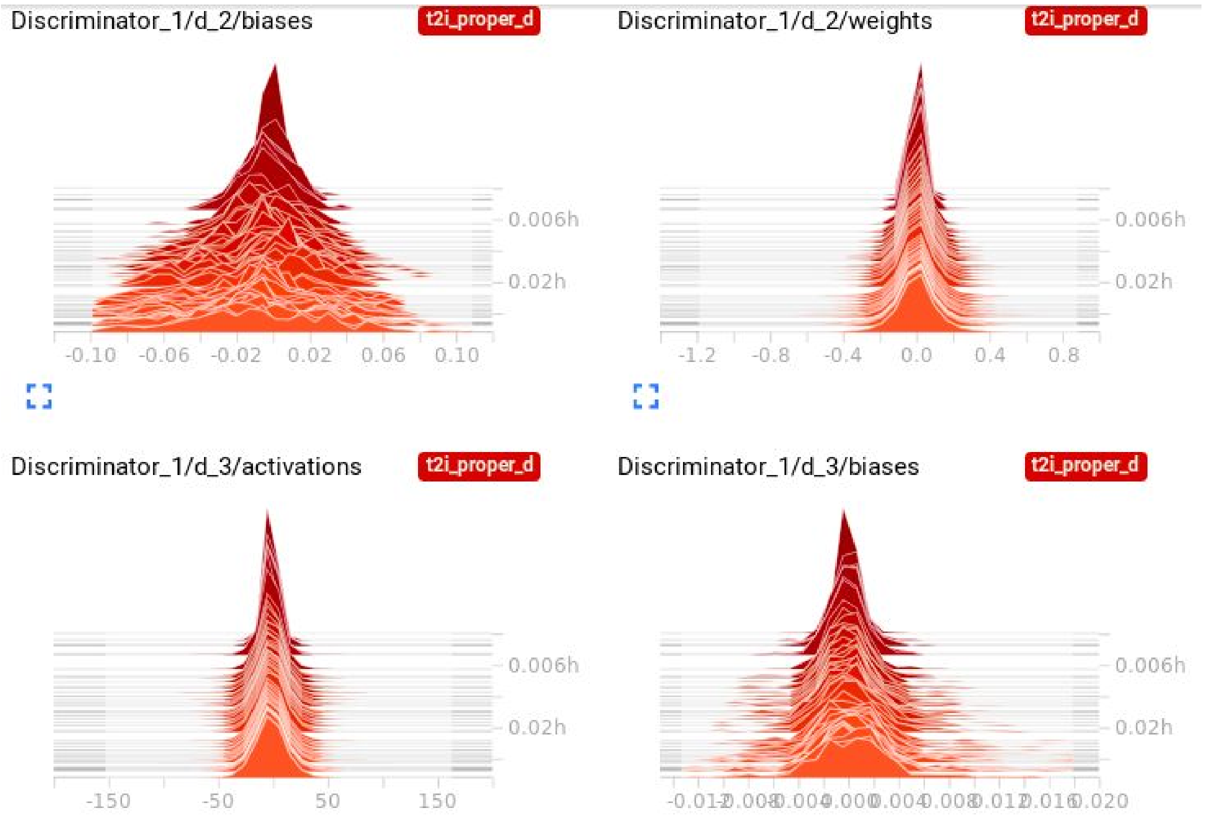


Fig 3.5 weights and activations after third training.

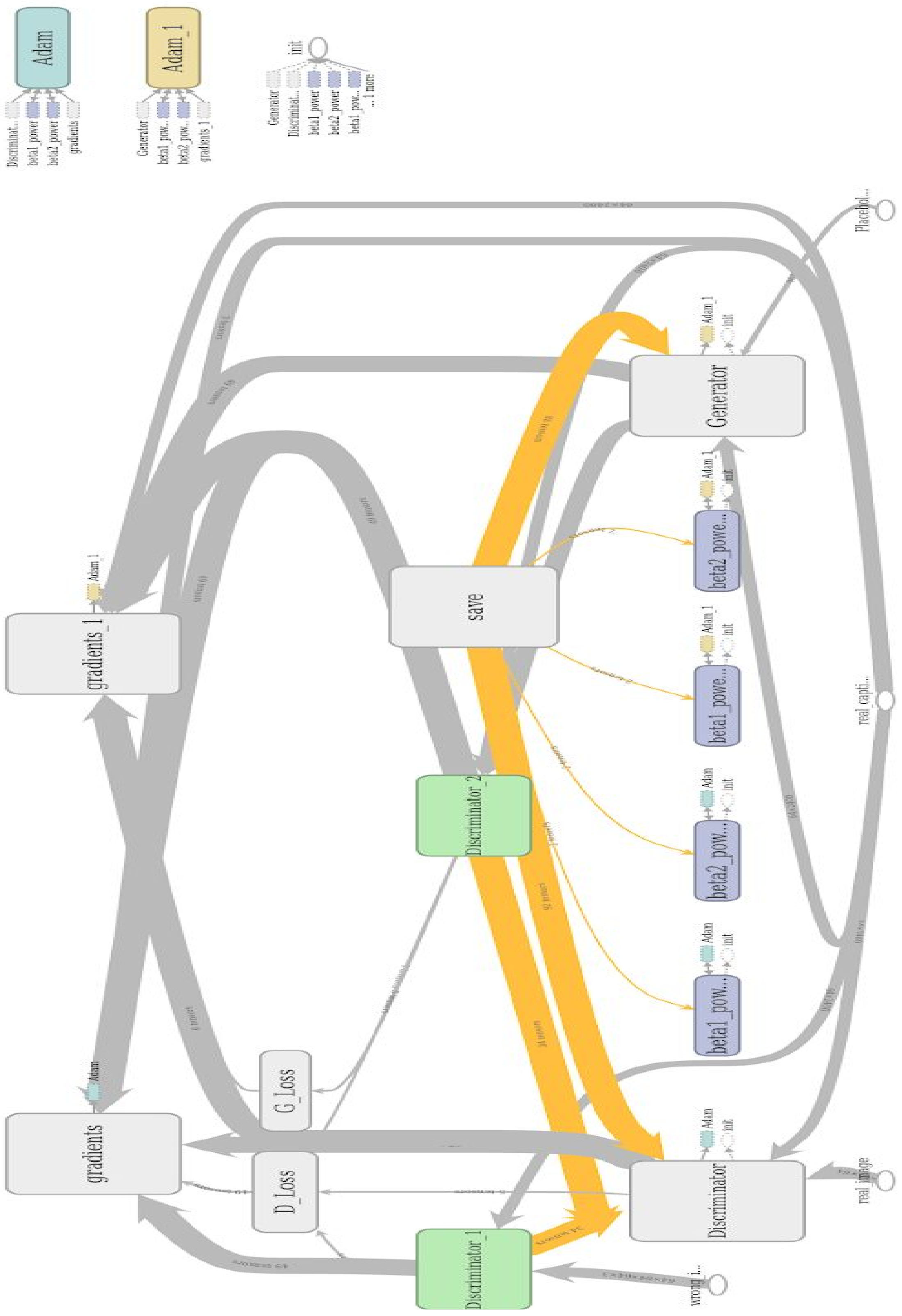

Adam
Discriminat...
beta1_power
beta2_power
gradients
Adam_1
Generator
beta1_pow...
beta2_pow...
gradients_1
init
Generator
Discriminat...
beta1_power
beta2_power
beta1_pow...
... 1 more
Placehol...
Generator
gradients_1
save
beta2_powe...
beta1_powe...
beta2_pow...
beta1_pow...
Discriminator_2
real capti...
G_Loss
D_Loss
gradients
Discriminator
Discriminator_1
real image
wrong i...

Fig 3.6 Graph of Generative Adversarial Network.

Chapter 4 : Conclusions and Future Work

4.1 Conclusions

The intermediate images generated using first training method is as shown in figure .The network is going in away from the optimal equilibrium hence the images are distorted and only dots can be seen.

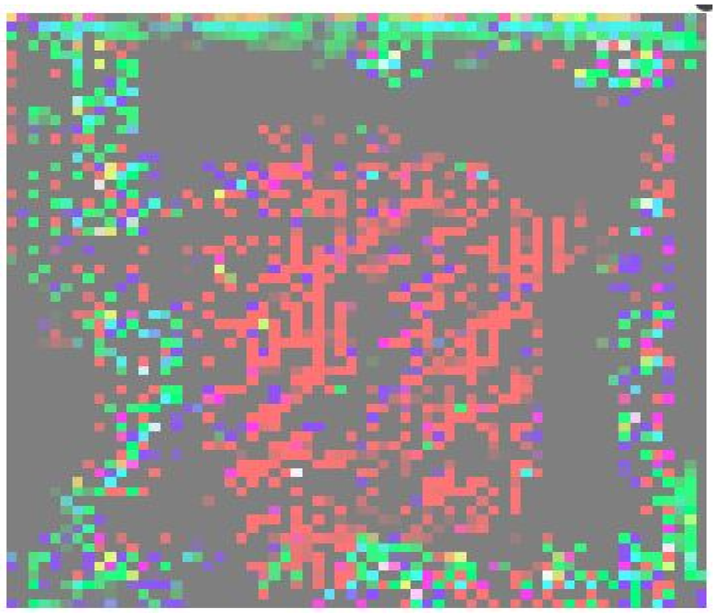


Fig 4.1: intermediate image of training 1.

The intermediate images generated using second training method is as shown in figure .The usage of batch normalisation, correct normalization of weights and biases makes the learning simple and near optimal direction as shown in figure.

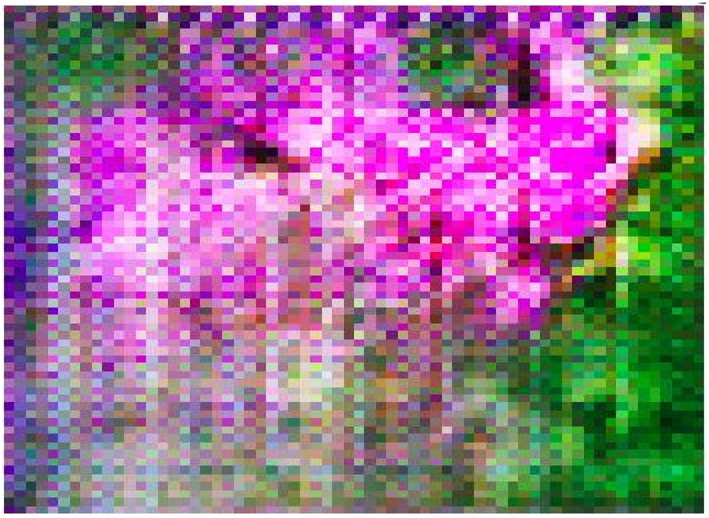


Fig 4.2: intermediate image of training 2.

The intermediate images generated using third method.Adding up the dropout layer helps the model to generate more precise, clear images in intermediate stage where the background can also be noticed.

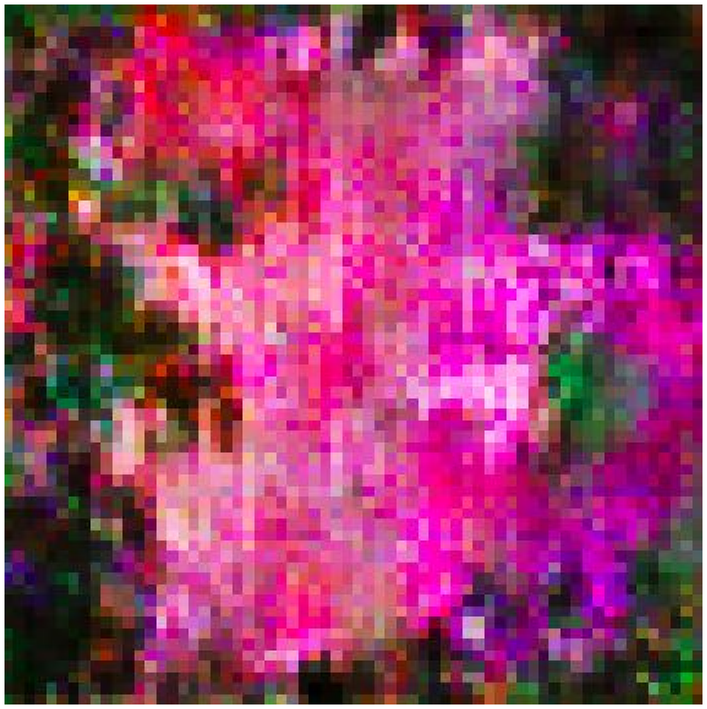


Fig 4.3: intermediate image of training 3.

The network was trained on Oxford-102 Flowers dataset and following are the results of the model.

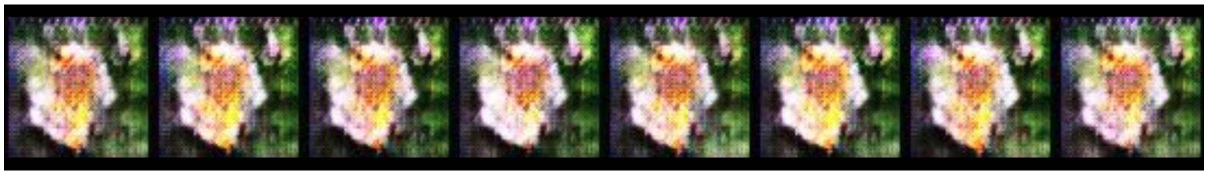


Fig 4.4: Output of GAN for "this flower has petals that are yellow, white and purple and has dark lines"

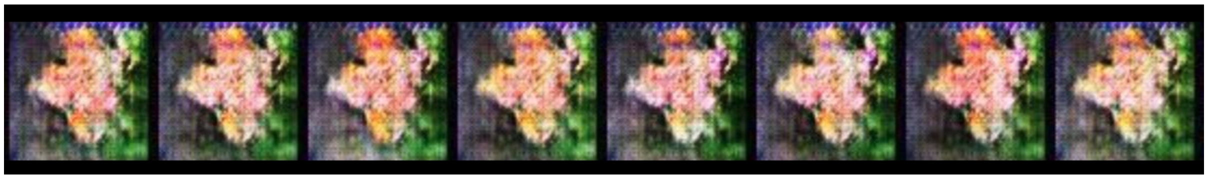


Fig 4.5: Output of GAN for "this flower has petals that are yellow, white and purple and has dark lines."

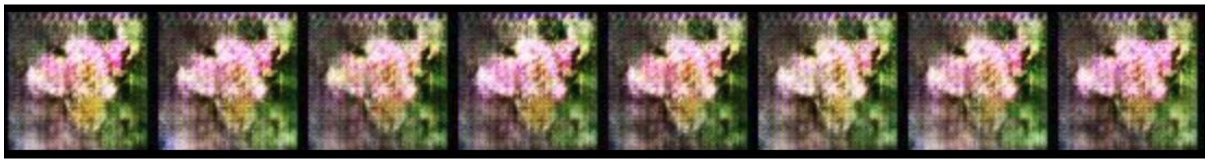


Fig 4.6: Output of GAN for "this flower has a lot of small round pink petals."

We observe that the model has started giving near to real images as shown in the outputs above.

4.2 Future Work:

We finally achieved convergence in training GAN and generated image that are closed to low dimensional photorealistic image.

Next improvement to this architecture is going more deeper there by increasing quality of image. Other idea is to train multiple generator model along with same discriminator.

We have trained on Oxford-102 Flowers dataset , in future the work will include understanding nature of GAN while training since we observe that the losses of the generator and discriminator become almost zero after 150^{th} epoch , training the model on different datasets like MS-COCO dataset , also we will work on to find the relation between the latent space of text and the images.

References

[1] Reed, Scott, et al. "Generative adversarial text to image synthesis." *arXiv preprint arXiv:1605.05396* (2016).

[2] Kiros, R., Zhu, Y., Salakhutdinov, R.R., Zemel, R., Urtasun, R., Torralba, A. and Fidler, S., 2015. Skip-thought vectors. In *Advances in neural information processing systems* (pp. 3294-3302).

[3] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NIPS, 2014.*

[4] Jon Bruner, A. (2017). *Generative Adversarial Networks for Beginners*. [online] O'Reilly Media. Available at: https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners [Accessed 3 Dec. 2017].

[5] Xavier glorot, Y.Bengio(2010). *Proceedings of the thirteenth international conference on artificial intelligence and statistics.*
Available at:http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf?hc_location=ufi
Simply explained on: https://intoli.com/blog/neural-network-initialization/

[6] N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov
Dropout: A simple way to prevent neural networks from overfitting
The Journal of Machine Learning Research 15 (1), 1929-1958

[7] Sergey Ioffe, Christian Szegedy.*arXiv preprint arXiv:1502.03167*

Batch normalization: Accelerating deep network training by reducing internal covariate shift

[8] Hyeonwoo Noh, Seunghoon Hong, Bohyung Han
Learning deconvolution network for semantic segmentation(2015)
Proceedings of the IEEE International Conference on Computer Vision

[9] https://github.com/vdumoulin/conv_arithmetic