

PRINCIPLES OF DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS (DC-GAN) AND ITS APPLICATION IN TEXT-IMAGE SYNTHESIS

Submitted as a partial fulfillment of Bachelor of Technology in Computer Science & Engineering
of

Maulana Abul Kalam Azad University of Technology
(Formerly known as West Bengal University of Technology)



Project Report

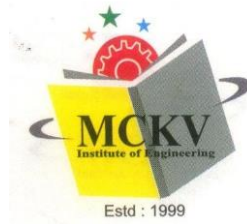
Submitted by

Kaushani Mukhopadhyay
Nirvik Ranjan Das
Sachin Kumar Roy
Riddhinath Ganguly
Sayak Das

11600117038
11600117034
11600117025
11600117026
11600117019

Under the supervision of

Ms. Rachita Ghoshhajra
Assistant Professor, Computer Science & Engineering



Department of Computer Science & Engineering,
MCKV Institute of Engineering
243, G.T. Road(N)
Liluah, Howrah – 711204

**Department of Computer Science & Engineering
MCKV Institute of Engineering
243, G. T. Road (N),
Liluah, Howrah-711204**

CERTIFICATE OF RECOMMENDATION

I hereby recommend that the thesis prepared under my supervision by Kaushani Mukhopadhyay, Nirvik Ranjan Das, Sachin Kumar Roy, Riddhinath Ganguly and Sayak Das entitled Principles of Deep Convolutional Generative Adversarial Networks (DC-GAN) and its application in Text-Image Synthesis be accepted in partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science & Engineering Department.

Mr. Avijit Bose

Assistant Professor & Head of the Department,
Computer Science & Engineering Department.
MCKV Institute of Engineering, Howrah

Project guide

Ms. Rachita Ghoshhajra, Assistant Professor,
Computer Science & Engineering Department.
MCKV Institute of Engineering, Howrah

MCKV Institute of Engineering
243, G. T. Road (N), Liluah
Howrah-711204

Affiliated to
Maulana Abul Kalam Azad University of Technology
(Formerly known as West Bengal University of Technology)

CERTIFICATE

This is to certify that the project entitled Principles of Deep Convolutional Generative Adversarial Networks (DC-GAN) and its application in Text-Image Synthesis submitted by

<u>Name of students</u>	<u>University Roll No.</u>
Kaushani Mukhopadhyay	11600117038
Nirvik Ranjan Das	11600117034
Sachin Kumar Roy	11600117025
Riddhinath Ganguly	11600117026
Sayak Das	11600117019

has been carried out under the guidance of myself following the rules and regulations of the degree of Bachelor of Technology in Computer Science & Engineering of **Maulana Abul Kalam Azad University of Technology** (Formerly West Bengal University of Technology).

(Signature of the project guide)

Ms. Rachita Ghoshhajra,
Assistant Professor,
Computer Science and Engineering

1. _____
2. _____
3. _____
4. _____
5. _____

MCKV Institute of Engineering
243, G. T. Road (N), Liluah
Howrah-711204

Affiliated to
Maulana Abul Kalam Azad University of Technology
(Formerly known as West Bengal University of Technology)

CERTIFICATE OF APPROVAL
(B.Tech Degree in Computer Science & Engineering)

This project report is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is to be understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed and conclusion drawn therein but approve the project report only for the purpose for which it has been submitted

COMMITTEE ON FINAL
EXAMINATION FOR
EVALUATION OF
PROJECT REPORT

1. _____
2. _____
3. _____
4. _____
5. _____

Acknowledgement

We would take the opportunity to express our deep sense of gratitude to **Mr. Avijit Bose, Assistant Professor and Head of the Department, Computer Science and Engineering, MCKV Institute of Engineering** for allowing us to form a group of five people and for supporting us to make our project worth.

We would like to express our special thanks of gratitude to **Ms. Rachita Ghoshhajra (Assistant Professor, CSE, MCKVIE) our Project Guide** who constantly encouraged, supported and guided us all along by providing necessary information required for this project. We are also thankful to **Mr. Abhisek Saha (Assistant Professor, CSE, MCKVIE), the Project Coordinator** for providing and clarifying the administrative formalities related to project proceedings. We would also like to thank the **Computer Science and Engineering Department of MCKV Institute of Engineering** for their immense support, contributions and valuable instructions in conducting this project report successfully.

We thank all our other faculty members and technical assistants at MCKV Institute of Engineering for playing a significant role during the development of the project. Last but not the least; we thank all our friends for their cooperation and encouragement.

Signature (**Kaushani Mukhopadhyay**)

Signature (**Nirvik Ranjan Das**)

Signature (**Sachin Kumar Roy**)

Signature (**Riddhinath Ganguly**)

Signature (**Sayak Das**)

Contents

Abstract	01
1. Introduction	02
2. GAN (Generative Adversarial Network)	03
2.1. Introduction	03
2.2. GAN Architecture	04
2.3. GAN Working Principle	05
2.4. GAN Algorithm	06
3. DCGAN (Deep Convolutional GAN)	07
3.1. Introduction	07
3.2. DCGAN Architecture	07
3.3. DCGAN Algorithm	08
3.4. DCGAN Flowchart	10
4. DCGAN Implementation on Text-Image Synthesis	11
4.1. Datasets	11
4.2. Implementation	11
4.3. Algorithm and parameters	11
4.4. Configurations	13
• Hardware Specifications	
• Software Specifications	
4.5. Pseudo Code	14
4.6. Experimental Results	15
• Experiment 1 (100 Epochs)	15
○ Captions	
○ Result	
○ Graphical Representation of Generator Loss	
○ Graphical Representation of Discriminator Loss	
○ Graphical Representation of RNN Loss	
• Experiment 2 (200 Epochs)	18
○ Captions	
○ Result	
○ Graphical Representation of Generator Loss	
○ Graphical Representation of Discriminator Loss	
○ Graphical Representation of RNN Loss	
5. Comparison between Standard GPU Implementation and CPU implementation	21
6. Conclusion	22
7. Future Scope	23
8. References	24

Abstract

Artificial intelligence has been an important sector for generation and prediction based on human inputs and synthesizing real images from text captions has been an important implementation in this regard. **Deep Convolutional Generative Adversarial Networks(DC-GANs)** have been a breakthrough in such synthesis and is getting popular day by day. In this work, a thorough study and analysis of Deep **Convolutional Generative Adversarial Networks (DC-GAN)** will be done where two models are trained simultaneously by an adversarial process. A *generator* ("the artist") learns to create images that look real, while a *discriminator* ("the art critic") learns to tell real images apart from fakes. Using Tensor Layers to create convolutions and de-convolutions an implementation of text-image synthesis can be achieved.

1. Introduction

Artificial intelligence has been an important sector for generation and prediction based on human inputs and synthesizing real images from text captions has been an important implementation in this regard.

Deep learning techniques need a huge volume of data to train effective models for tasks such as image recognition/ classification. Data augmentation is a technique commonly used in deep learning to expand data and prevent over-fitting in such data-limited situations. In this work, we investigate the use of Deep Convolutional Generative Adversarial Networks for generating images from proposed texts . Generative Adversarial Networks (GAN's) were introduced by Ian Goodfellow and his colleagues in 2014 [Ref. 1.]. GAN's utilize two neural networks, a generator which takes random noise as input to create samples (data) as realistic as possible to the original dataset and a discriminator to distinguish between data that is real (original data) vs fake (generated data).

The DCGAN is a direct extension of the original GAN, except that the discriminator and generator explicitly use convolutional and convolutional-transpose layers, respectively.

One of the most interesting parts of Generative Adversarial Networks is the design of the Generator network. The Generator network is able to take random noise and map it into images such that the discriminator cannot tell which images came from the dataset and which images came from the generator.

This DCGAN architecture is especially interesting the way the first layer expands the random noise. This network takes in a 100x1 noise vector, denoted z , and maps it into the $G(z)$ output which is 64x64x3. The network goes from 100x1 to 1024x4x4! This layer is denoted 'project and reshape'.

This is a very interesting application of neural networks. Typically neural nets map input into a binary output, (1 or 0), maybe a regression output, (some real-valued number), or even multiple categorical outputs, (such as MNIST or CIFAR-10/100). Classical convolutional layers are applied which reshape the network with the $(N+P - F)/S + 1$ equation classically taught with convolutional layers.

2. GAN(Generative adversarial network)

2.1. Introduction

Generative adversarial network (GAN) is proposed by Goodfellow in 2014 [Ref. 1.], which is a kind of generative model. It consists of a discriminator network D and a generator network G . The input of the generator is a random vector z from a fixed distribution such as normal distribution and the output of it is an image. The input of discriminator is an image, the output is a value in $(0; 1)$. The two networks compete during training, the objective function of GAN is:

$$\min_G \max_D V(D, G) = \min_G \max_D \mathbb{E}_{x \sim p_d(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Where,

G = Generator

D = Discriminator

$P_d(x)$ = distribution of real data

$P(z)$ = distribution of generator

x = sample from $P_d(x)$

z = sample from $P(z)$

$D(x)$ = Discriminator network

$G(z)$ = Generator network

2.2. GAN Architecture

A generative adversarial network (GAN) has two parts:

- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake.

As training progresses, the generator gets closer to producing output that can fool the discriminator.

Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

The whole process is depicted in Figure 1.

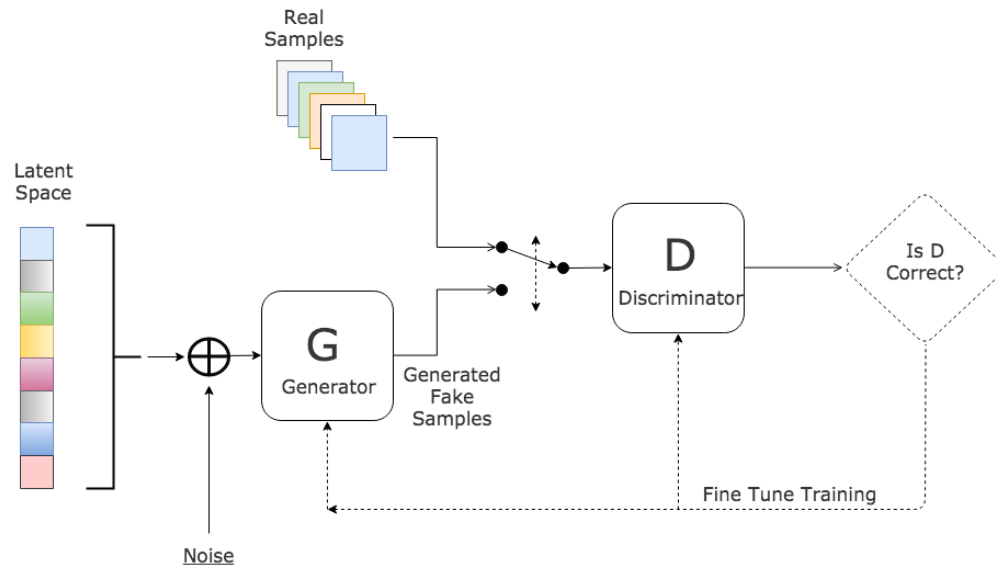


Figure 1. GAN Architecture

Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

2.3. Working Principle

Generator goal: Maximize the likelihood that the discriminator misclassifies its output as real.

Discriminator goal: Optimize toward a goal of 0.5, where the discriminator can't distinguish between real and generated images.

The Minimax Problem (sometimes called MinMax) is a theory that focuses on maximizing a function at the greatest loss (or vice versa). In the case of GANs, this is represented by the two models training in an adversarial way. The training step will focus on minimizing the error on the training loss for the generator while getting as close to 0.5 as possible on the discriminator (where the discriminator can't tell the difference between real and fake).

In the GAN framework, the generator will start to train alongside the discriminator; the discriminator needs to train for a few epochs prior to starting the adversarial training as the discriminator will need to be able to actually classify images. There's one final piece to this structure, called the loss function. The loss function provides the stopping criteria for the **Generator** and **Discriminator** training processes.

Generator Steps:

1. First, the generator samples from a latent space and creates a relationship between the latent space and the output.
2. We then create a neural network that goes from an input (latent space) to output (image for most examples).
3. We'll train the generator in an adversarial mode where we connect the generator and discriminator together in a model (every generator and GAN recipe in this book will show these steps).
4. The generator can then be used for inference after training.

Discriminator Steps:

1. First, it'll create a convolutional neural network to classify real or fake (binary classification).
2. It'll create a dataset of real data and use our generator to create fake dataset.
3. We train the discriminator model on the real and fake data.
4. It'll learn to balance training of the discriminator with the generator training — if the discriminator is too good, the generator will diverge.

2.4. Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator k , is a hyperparameter and $k = 1$ is used.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$

end for

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)})))$$

end for

3. DCGAN (Deep Convolutional GAN)

3.1. Introduction

The idea is to use convolutional and transpose layers instead of dense layers, along with introducing certain constraints such as batch normalization, use of proper activation functions for different layers, constraining the optimizer used, and designing the upscaling and downscaling layers in a specified manner. All these prove to be invaluable to the overall learning process as well as contribute to different convolutional layers capturing various important features as well.

It uses a couple of guidelines, in particular:

- Replacing any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Using batchnorm in both the generator and the discriminator.
- Removing fully connected hidden layers for deeper architectures.
- Using ReLU activation in generator for all layers except for the output, which uses tanh.
- Using LeakyReLU activation in the discriminator for all layer.

3.2. Architecture

(a)

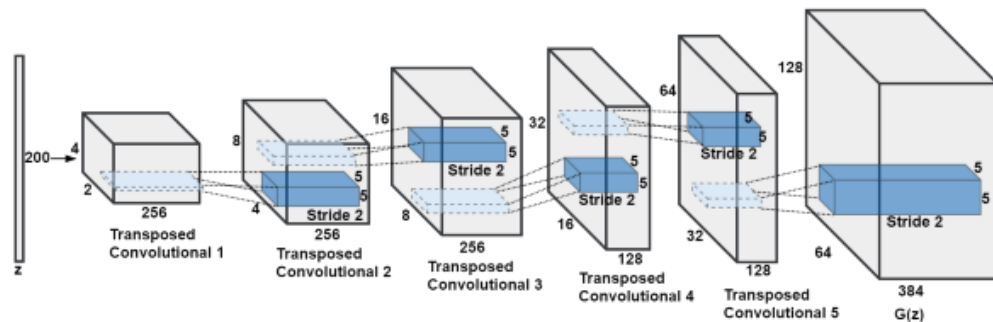


Figure 2. Transposed Convolution

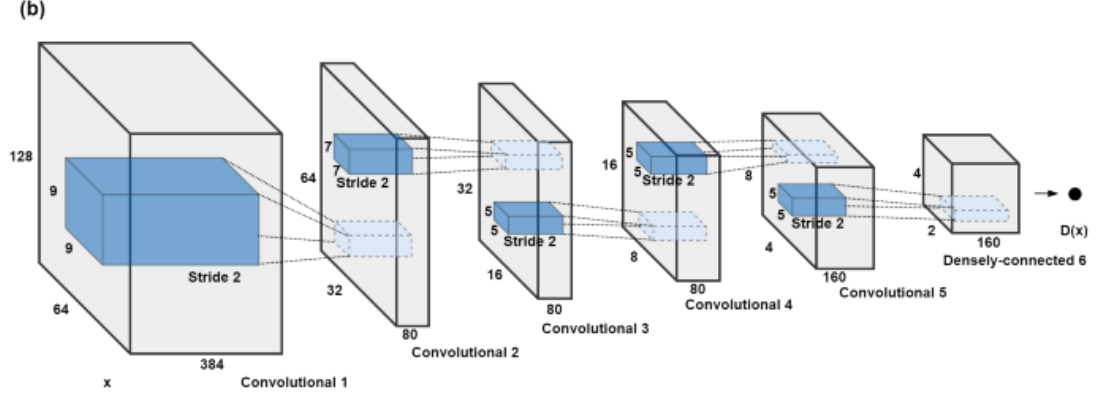


Figure 3. Convolution

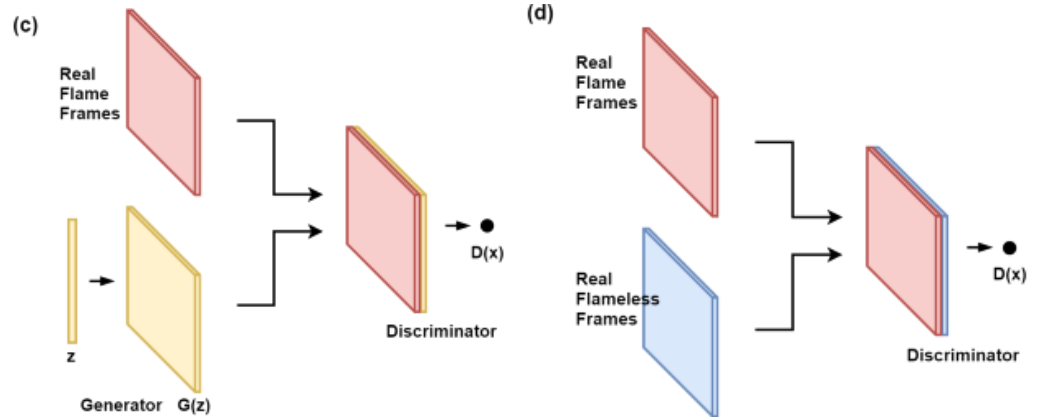


Figure 4. Generator and Discriminator

3.3. Algorithm

Algorithm 2. Deep Convolutional Generative Adversarial Network

1. Load and Prepare the dataset
2. Create the models (Both the generator and discriminator models are defined using Keras Sequential API)
 - a. **The Generator:** The generator uses `tf.keras.layers.Conv2DTranspose` (upsampling) layers to produce an image from a seed (random noise). Start with a dense layer that takes this seed as input, then upsample several times until you reach the desired image size of

3.4. Flowchart

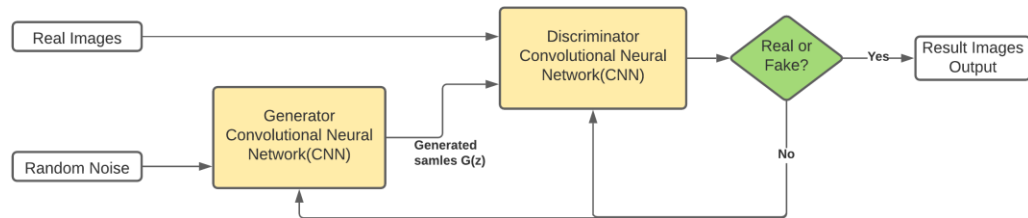


Figure 5. DCGAN Flowchart

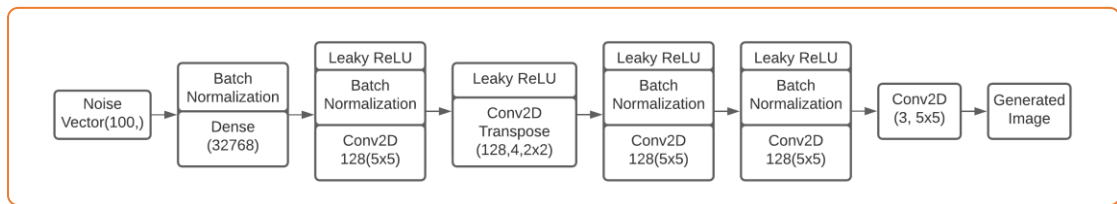


Figure 6. Generator Flowchart

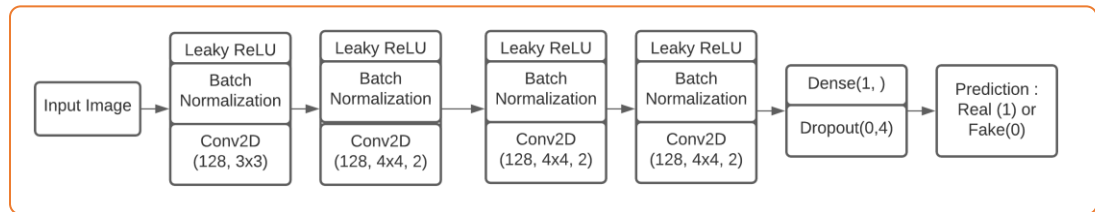


Figure 7. Discriminator Flowchart

Here Random Noise is basically a 100x1 noise vector. This network takes in a 100x1 noise vector, denoted z , and maps it into the $G(Z)$ output which is 64x64x3. The network goes from 100x1 to 1024x4x4! This layer is denoted ‘project and reshape’.

4. DCGAN IMPLEMENTATION ON TEXT-IMAGE SYNTHESIS

4.1. Datasets

We executed our algorithm on the Oxford-102 flower dataset [Ref. 6]. For the Oxford-102 dataset, it has 102 classes, which contains 82 training classes and 20 test classes. Each of the images in the dataset has 10 corresponding text descriptions. We have used the DC-GAN architecture for training the model on this dataset and we have done twice, once for 100 epochs with one set of captions and another for 200 epochs with a different set of captions.

The URL for the same is: <https://www.robots.ox.ac.uk/~vgg/data/flowers/102/>

4.2. Implementation:

This is a Tensorflow implementation of synthesizing images. The images are synthesized using the GAN-CLS Algorithm explained in [Algorithm 1]. This implementation is built on top of the excellent DCGAN in Tensorflow whose algorithm is explained in [Algorithm 2].

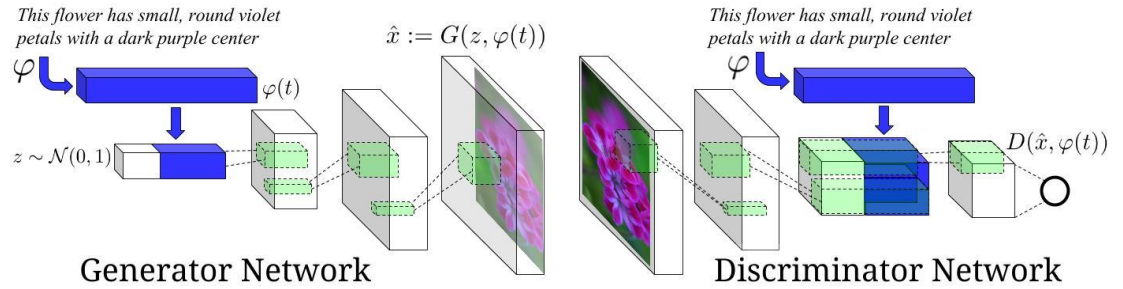


Figure 8. Implemented Architecture of DC-GAN

4.3. Algorithms and Parameters

- **Algorithms**

The text-image synthesis experiment is carried out using two algorithms:

1. GAN-CLS algorithm explained in [Algorithm 1] which is the basic GAN model used for image synthesis.

The objective function is:

$$\min_G \max_D V(D, G) = \min_G \max_D \mathbb{E}_{x \sim p_d(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))].$$

2. Deep Convolutional GAN algorithm explained in [Algorithm 2] which is a modified version of GAN CLS algorithm with convolutions and neural network for image synthesis. The objective function for Loss and optimizers are:

$$\text{Loss_D: } \log(D(x)) + \log(D(G(z))) \log(D(x)) + \log(D(G(z)))$$

$$\text{Loss_G: } \log(D(G(z)))$$

Binary Cross Entropy Loss:

$$\ell(x,y) = L = \{l1, ..., lN\}^T, \quad l_n = -\sum y_n \cdot \log x_n + (1-y_n) \cdot \log(1-x_n)$$

- **The Parameters**

Parameter	Description	Value
z_dim	Noise Dimension	100
t_dim	Text Feature Dimension	256
batch_size	Batch Size during training	64
image_size	Image Dimension	64
gf_dim	Number of convolutions in the first layer of generator	64
df_dim	Number of convolutions in the first layer of discriminator	64
gfc_dim	Dimension of gen units for fully connected layer	1024
caption_vector_length	Length of the caption vector	1024
learning_rate	Learning Rate for optimizers	.0002
beta1	Beta1 hyper PARAM for Adam optimizers	0.5
epochs	Number of epochs	100/200
data_set	Data set to train on	Oxford-102 flower dataset
ngpu	Number of GPUs available	0

Table 1. List of Parameters

4.4. Configurations

- **Hardware Specifications**

Details	PC-1	PC-2
Processor	Intel Core i5(7 th Gen) Processor	AMD Ryzen 5 2400G
Clock Speed	2.5 GHZ	3.6 GHz
Graphic Processor	Intel HD 620	RX Vega 11 Graphics
Ram	8GB DDR4	8GB DDR4
Ram Speed	2133 MHz	2933 MHz
Hard Drive	1 TB SATA	1 TB SATA
OS Name	Windows 10 Home	Windows 10 pro
OS Type	64 bit	64bit

Table 2. Hardware Specifications

- **Software Specifications**

1. Python 3 or above
2. Anaconda/ Jupyter/ Google Colab
3. TensorFlow
4. Tensor Layer
5. scikit-learn : for skip thought vectors
6. NLTK : for skip thought vectors
7. Numpy

4.5. Pseudo-Code

1. Import Dataset
2. Save the following as pickles:
 - a. Caption Pickle – Merge captions with image in a vector format
 - b. Image Test, Train Pickle – Train-Test Split of the Pickle
 - c. vocab Pickle – Synthesizing of the words in the captions
3. Load data from pickle
4. Define **main_train** :
 - a. CNN encoding
 - b. Defining Loss functions
 - c. **Loop until number of epochs:**
 - i. **Noise to Generator:**
 1. Batch Normalization
 2. Leaky ReLU
 3. Convolution
 4. Transpose Convolution
 - ii. **Discriminator:**
 1. Inputs the real image pickle data
 2. Inputs the generated samples from generator $G(z)$
 3. Leaky ReLU
 4. Batch Normalization
 5. Convolutions(4x)
 - iii. **Decision Making Real or Fake:**
 1. If fake the discriminator output serves as a feedback input for the generator in the next epoch
 - iv. Print the generated image for n^{th} epoch, along with the Generator losses, discriminator losses and RNN-losses.
 - d. **End loop**

4.6. Experimental Results

Experiment 1(100 epochs)

In this experiment, the same dataset has been trained for 100 epochs using a set of captions (texts) as noted below and accordingly the following images are generated from them. Here each epoch took roughly 40 minutes execution time and total 100 epochs took 70 hours that is almost 3 days.

Captions:

1. *the flower shown has yellow anther red pistil and bright red petals.*
2. *this flower has petals that are yellow, white and purple and has dark lines*
3. *the petals on this flower are white with a yellow center*
4. *this flower has a lot of small round pink petals.*
5. *this flower is orange in color, and has petals that are ruffled and rounded.*
6. *the flower has yellow petals and the center of it is brown.*
7. *this flower has petals that are blue and white.*
8. *these white flowers have petals that start off white in color and end in a white towards the tips.*

Results



Figure 9. Expt. 1 - Epoch 01



Figure 10. Expt. 1 - Epoch 17



Figure 11. Expt. 1 - Epoch 33



Figure 12. Expt. 1 - Epoch 50



Figure 13. Expt. 1 - Epoch 68



Figure 14. Expt. 1 - Epoch 100

The loss functions constantly calculate the loss in the generator, discriminator and the RNN-loss for every epoch. The loss for every epoch has been checked while generating the images in every epoch and they are plotted in a graph where the X-axis is the Epoch number x10 i.e., 1 means 10th epoch and so on and the Y axis represents the loss for every epoch. The losses of 10 epochs are averaged and depicted for each of the epochs that are multiples of 10.

Graphical Representation of Generator-LOSS for epochs:

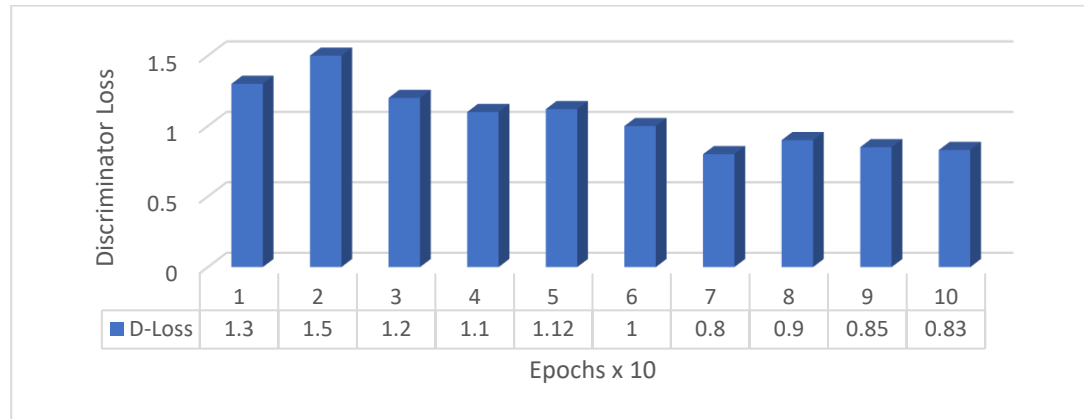


Figure 15. The Generator Loss graph – Expt. 1

Graphical Representation of Discriminator-LOSS for epochs:

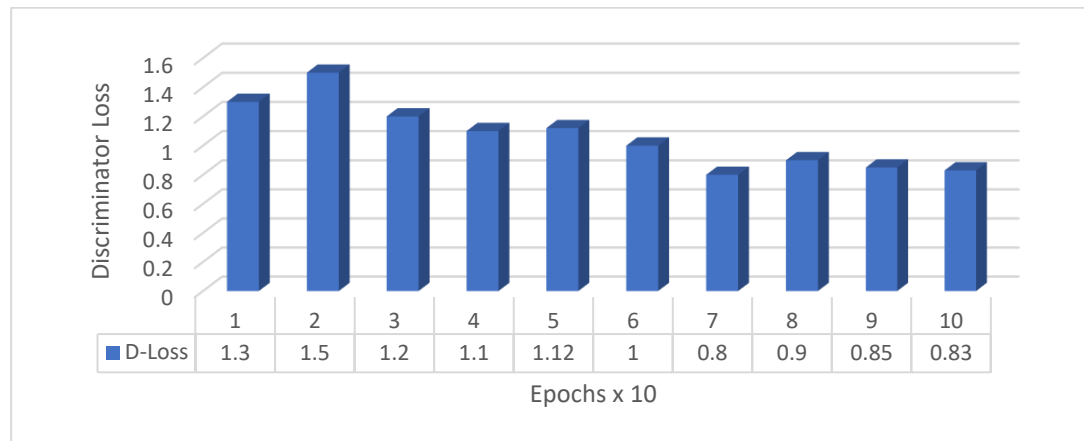


Figure 16. The Discriminator Loss graph– Expt. 1

Graphical Representation of RNN-LOSS for epochs:

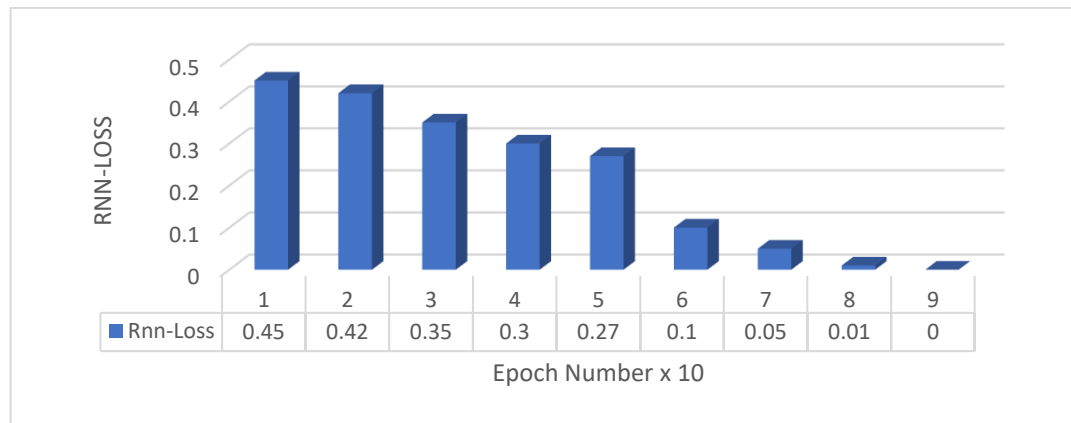


Figure 17. The RNN-Loss graph – Expt. 1

Experiment 2(200 epochs)

In this experiment the same dataset has been trained for 200 epochs using a set of captions(texts) as noted below and accordingly the following images are generated from them. Here each epoch took roughly 1hour of execution time and total 200 epochs took almost 10 days.

Captions:

1. *the flower shown has purple and white petals*
2. *this flower has petals that are yellow and purple and has dark lines*
3. *the petals on this flower are red with a yellow center*
4. *this flower has a lot of small round orange petals*
5. *this flower is dark orange in color, and has petals that are ruffled and rounded*
6. *the flower has yellow petals and the center of it is brown*
7. *this flower has petals that are yellow and white*
8. *these flowers have pink colored petals*

Results:

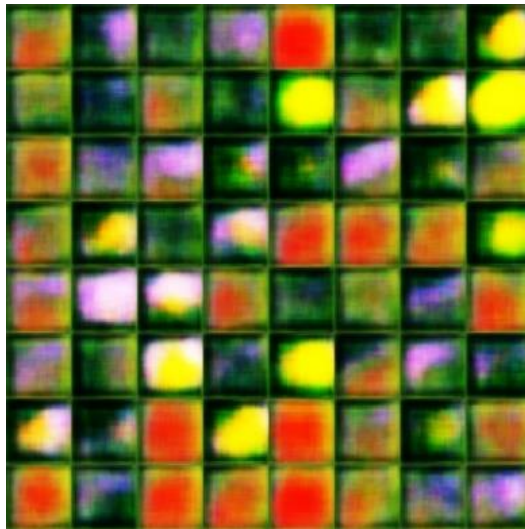


Figure 18. Expt. 2 - Epoch 01



Figure 19. Expt. 2 - Epoch 50



Figure 20. Expt. 2 - Epoch 100



Figure 21. Expt. 2 - Epoch 140



Figure 22. Expt. 2 - Epoch 175



Figure 23. Expt. 2 - Epoch 200

The loss functions constantly calculate the loss in the generator, discriminator and the RNN-loss for every epoch. The RNN-loss for every epoch has been checked while generating the images in every epoch and they are plotted in a graph where the X-axis is the Epoch number x10 i.e., 1 means 10th epoch and so on and the Y axis represents the RNN-loss for every epoch. The losses of 10 epochs are averaged and depicted for each of the epochs that are multiples of 10.

Graphical Representation of Generator-LOSS for epochs:

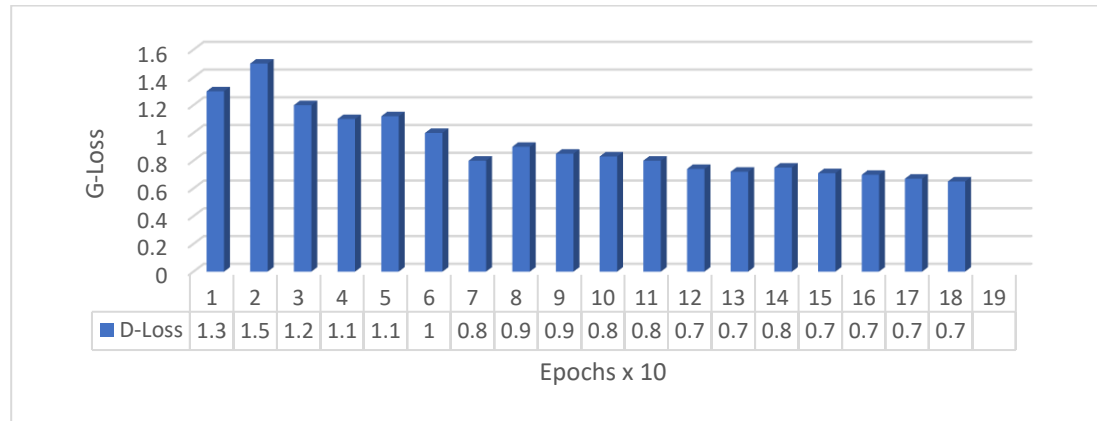


Figure 24. The Generator Loss Graph – Expt. 2

Graphical Representation of Discriminator-LOSS for epochs:

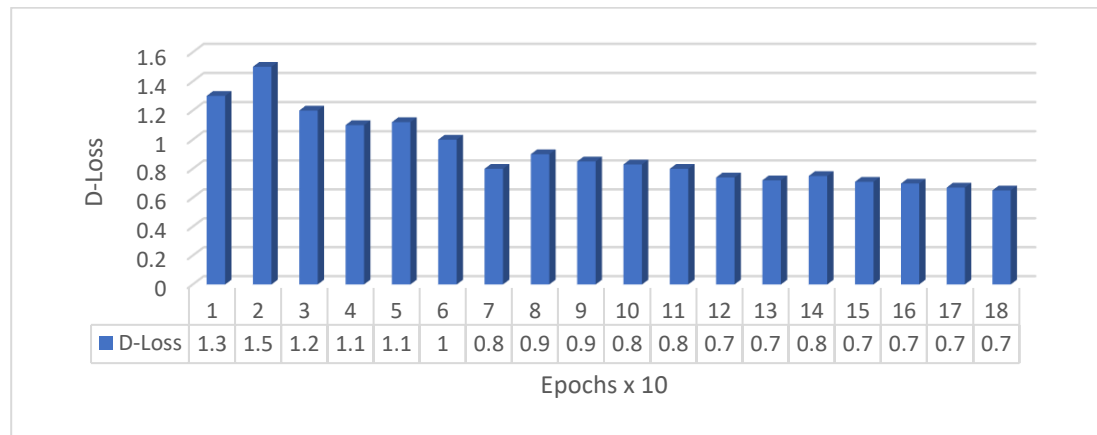


Figure 24. The Generator Loss Graph – Expt. 2

Graphical Representation of RNN-LOSS for epochs:

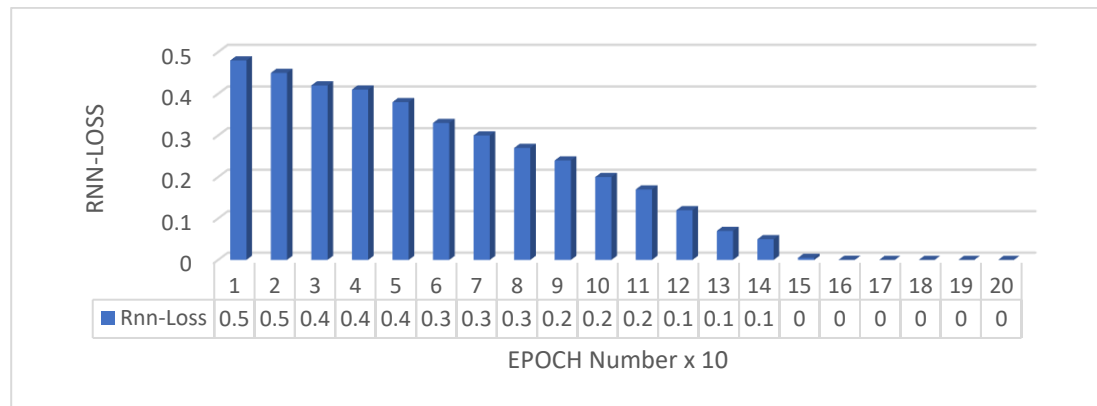


Figure 22. The RNN-Loss graph – Expt. 2

5. COMPARISON BETWEEN STANDARD GPU IMPLEMENTATION AND CPU IMPLEMENTATION

Standard GPU result: This is the result of a standard implementation where GPU tensor flow was implemented for running the experiment and the results after 600 epochs are as such:









Captions	Generated Images
the flower shown has yellow anther red pistil and bright red petals.	
this flower has petals that are yellow, white and purple and has dark lines	
the petals on this flower are white with a yellow center	
this flower has a lot of small round pink petals.	
this flower is orange in color, and has petals that are ruffled and rounded.	
the flower has yellow petals and the center of it is brown.	
this flower has petals that are blue and white.	
these white flowers have petals that start off white in color and end in a white tip.	

Figure 23. Standard GPU implementation Results

CPU Implementation result: This is the result of our implementation where CPU tensor flow used for running the experiment with same set of captions and the results after 100 epochs:









Captions	Generated Images
the flower shown has yellow anther red pistil and bright red petals.	
this flower has petals that are yellow, white and purple and has dark lines	
the petals on this flower are white with a yellow center	
this flower has a lot of small round pink petals.	
this flower is orange in color, and has petals that are ruffled and rounded.	
the flower has yellow petals and the center of it is brown	
this flower has petals that are blue and white.	
these white flowers have petals that start off white in color and end in a white tip.	

Figure 24. CPU implementation Results

The standard GPU result was implemented on top of dcgan.torch [Ref. 7.] and the code was run for around 600 epochs which took them roughly 2-3 days . The standard implementation was shown in the paper Generative Adversarial Text to Image Synthesis [Ref. 3.]

We ran our CPU implementation on top of tensor flow [Ref. 5.] on hardware specifications [Table 2.] without any GPU and it was run twice, once for 100 epochs and another time for 200 epochs and it took roughly 3 days for the first run and 10 days for the second and are able to achieve a result as shown in Figure 9-14 and Figure 18-23.

The comparison of both the standard implementation and our implementation has been depicted in Figure 23. and Figure 24. and the difference is clearly visible.

A better hardware with dedicated GPU would have decreased one of the main constraints of this experiment i.e. time and it would have been much more efficient. The images generated would have been more generic.

6. Conclusion

In this project a thorough study of Generative adversarial network(GAN) and its modified version i.e. DCGAN(Deep Convolutional GAN) has been done and an implementation has been performed using the DCGAN architecture where images are synthesized on the basis of a set of given captions. The images are not a selection of images from the dataset rather, a set of synthesized images which have undergone repetitive convolution and transposed convolutions. The more the number of epochs, the better will be the end result. Here the implementation has been done twice, once with 100 epochs and another with 200 epochs. The main constraint of this experiment is having a machine with top hardware specifications with dedicated GPU and the time factor . With better hardware the time constraint of this experiment could be reduced and more generic images would have been the output resulting in fulfilling the exact need of this experiment.

7. Future Scope

1. **GPU Tensor Flow Implementation:** The CPU tensor flow is more time consuming and less efficient in terms of minimizing the pixel loss and so the next aim is to train the model using a GPU tensor flow that is more efficient.
2. **Training on Birds dataset:** Training the model on a birds dataset to verify the model.
3. **Training on MS-COCO dataset:** Training the model on MS-COCO dataset to get more images that are generic.
4. **Different Embedding Options:** Trial of different embedding options for captions and to train the caption embedding RNN along with the GAN-CLS model.
5. **Web Application:** Preparing a trained model and uploading it to a server to create a web application that accepts a caption as input and instantly generates the required image based on that captions without the need of training the data every time and making the process time and resource efficient.
6. **Other types of GAN Implementation:** To get more photorealistic pictures from our dataset we can use other types of GANs like Stack GAN, Style GAN to generate a different set of results, which are more effective.
7. **Derivations of DCGAN:** To understand all the aspect of the DCGAN one example is not sufficient. Many derivations of DCGAN can help us understand how DCGAN is effective like Sin GAN, Cycle GAN, Conditional DCGAN can help bring out other aspects of this architecture.

8. References

1. **Generative Adversarial Nets** by **Ian Goodfellow**, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio; Part of Advances in Neural Information Processing Systems 27 (NIPS 2014).
2. **Generate the corresponding Image from Text Description using Modified GAN-CLS Algorithm** by Fuzhou Gong and Zigeng Xia ; University of Chinese Academy of Sciences; arXiv:1806.11302v1 [cs.LG] 29 Jun 2018.
3. **Generative Adversarial Text to Image Synthesis** by Scott Reed, Zeynep Akata, Xincheng Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee ; Neural and Evolutionary Computing (cs.NE); Computer Vision and Pattern Recognition ; ICML 2016.
4. **Skip-Thought Vectors** by Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, Sanja Fidler; Computation and Language (cs.CL); Machine Learning (cs.LG) ; arXiv:1506.06726.
5. **Tensor flow Implementation of Deep Convolutional Generative Adversarial Networks**
- <https://www.tensorflow.org/tutorials/generative/dcgan>
6. **Automated flower classification over a large number of classes**, Nilsback, M-E. And Zisserman, A.; Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing (2008).
7. **DCGAN Torch Code**: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html