A Project report on

Text to Image Synthesis Using Deep Fusion Generative Adversarial Networks

Submitted in partial fulfillment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

in the faculty of

COMPUTER SCIENCE AND ENGINEERING

Submitted By

PAVANI. K	[17021A0541]
JAGADEESH. N	[17021A0535]
SAI LAKSHMI RAJESWARI. K	[18025A0568]
SUNITHA. P	[18025A0572]

Under the supervision of

Dr. K. V. RAMANA

Professor

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KAKINADA – 533003, A.P, INDIA

2020 – 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KAKINADA – 533003, A. P., INDIA

DECLARATION FROM THE STUDENTS

We hereby declare that the project work described in this thesis, entitled **"Text to Image Synthesis Using Deep fusion Generative Adversarial Networks"** *is being submitted by our team in partial fulfillment of the requirements for the award of the degree of* ***Bachelor of Technology (B. Tech)*** *in the faculty of Computer Science & Engineering to the University College of Engineering (Autonomous), Jawaharlal Nehru Technological University Kakinada – 533003 A. P., is the result of investigations carried out by us under the guidance of* **DR.K.V.RAMANA,** ***Professor* ,** *Department of Computer Science and Engineering, University College of Engineering Kakinada (A), Jawaharlal Nehru Technological University Kakinada – 533003.*

The work is original and has not been submitted to any other University or Institute for the award of any degree or diploma.

Place : Kakinada

Date :

Signature:

Pavani. K	[17021A0541]
Jagadeesh. N	[17021A0535]
S. L. Rajeswari. K	[18025A0568]
Sunitha. P	[18025A0572]

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA(A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KAKINADA-533003, A.P, INDIA

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the project report titled ***"Text to Image Synthesis Using Deep fusion Generative Adversarial Networks"*** *being submitted by* ***PAVANI. K [17021A0541], JAGADEESH. N [17021A0535], SAI LAKSHMI RAJESWARI. K [18025A0568], SUNITHA. P [18025A0572]*** *in partial fulfillment of requirements for the award of the degree of* ***Bachelor of Technology (B. Tech)*** *in Computer Science and Engineering to Jawaharlal Nehru Technological University, Kakinada is a record of bonafide project work carried out by them under my guidance in the department of computer science and engineering. The results of the project have been verified and found satisfactory. The results embodied in this project have not been submitted to any other university or institute for the award of any other degree or diploma.*

Signature of the Supervisor

Dr. K. V. Ramana

Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KAKINADA – 533003, A. P., INDIA

CERTIFICATE FROM THE HEAD OF DEPARTMENT

*This is to certify that the project work entitled **"Text to Image Synthesis Using Deep Fusion Generative Adversarial Networks"** is being submitted by Pavani. K [17021A0541], Jagadeesh. N [17021A0535], Sai Lakshmi Rajeswari. K [18025A0568], Sunitha. P [18025A0572] in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology (B. Tech)** in the faculty of Computer Science and Engineering to the University College of Engineering Kakinada (Autonomous), Jawaharlal Nehru Technological University Kakinada – 533003, A. P., is a record of bonafide project work carried by them at our department.*

Signature of Head of the Department

Dr. D. Haritha

Head & professor of CSE

ACKNOWLEDGEMENT

*We express my deep sense of gratitude to my project guide **Dr. K. V. Ramana,** Professor, Department of Computer Science and Engineering, University College of Engineering Kakinada, (Autonomous), Jawaharlal Nehru Technological University, Kakinada, for his able guidance and constant encouragement throughout the project. He has been a constant source of inspiration and helped me in each step. I express my deep gratitude to him for being extremely considerate to us.*

*We are extremely thankful to **Dr. D. Haritha, Head of the Department, Computer Science and Engineering** for providing a good environment and better infrastructure, facilities, and encouragement to complete the project in time.*

We all express sincere thanks to all professors and Faculty Members in the department for their teaching and academic support and thanks to the technical staff and non-teaching staff of the Department for their support.

Pavani. K	[17021A0541]
Jagadeesh. N	[17021A0535]
S. L. Rajeswari. K	[18025A0568]
Sunitha. P	[18025A0572]

ABSTRACT

Synthesizing high-quality realistic images from text descriptions is a challenging problem in computer vision and has many practical applications. Almost all existing text-to-image Generative Adversarial Networks employ stacked architecture as the backbone. They utilize cross-modal attention mechanisms to fuse text and image features and introduce extra networks to ensure text-image semantic consistency. In this work, we propose a much simpler, but more effective text-to-image model than previous works. Corresponding to the above three limitations, we propose: 1. a novel one-stage text-to-image backbone which is able to synthesize high-quality images directly by one pair of generator and discriminator, 2. a novel fusion module called deep text-image fusion block which deepens the text-image fusion process in generator, 3. a novel target-aware discriminator composed of matching-aware gradient penalty and one-way discriminator which promotes the generator to synthesize more realistic and text-image semantic consistent images without introducing extra networks. Compared with existing text-to-image models, our proposed method (i.e., DF-GAN) is simpler but more efficient to synthesize realistic and text-matching images and achieves better performance.

Index Terms: Text-to-Image Synthesis, Generative Adversarial Networks, Deep fusion Block, Generator, Discriminator, Matching-Aware Gradient Penalty.

TABLE OF CONTENTS

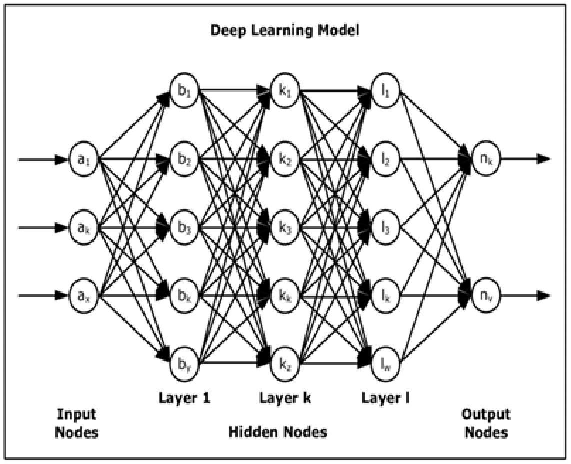
LIST OF FIGURES

CHAPTER-1

INTRODUCTION

1.1 Deep Learning

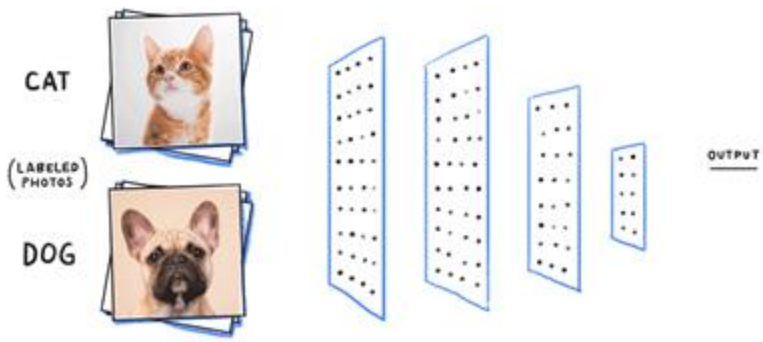
Deep Learning is a subfield of Machine Learning concerned with algorithms inspired by the structure and functions of the brain called Artificial Neural Networks.



Deep learning models:

1. Discriminative models
2. Generator models

Discriminative model



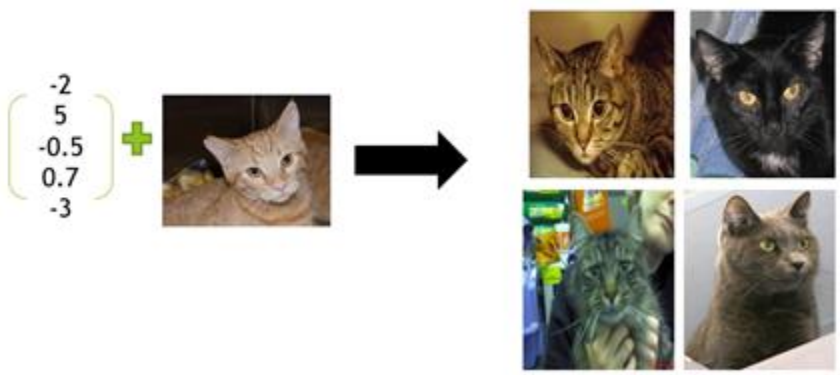
In supervised learning, we may be interested in developing a model to predict a class label given an example of input variables. This predictive modeling task is called classification. Classification is also traditionally referred to as discriminative modeling.

We use the training data to find a discriminant function f(x) that maps each x directly onto a class label, thereby combining the inference and decision stages into a single learning problem.

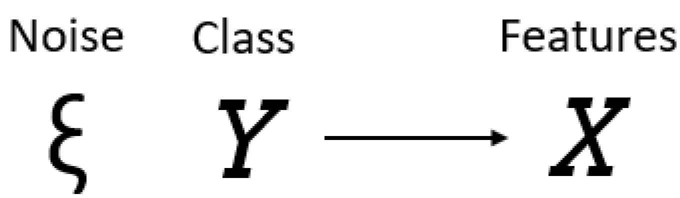
This is because a model must discriminate between examples of input variables across classes, it must choose or make a decision as to what class a given example belongs to.

Discriminator model captures the conditional probability of P(Y|X).

Generative model



Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

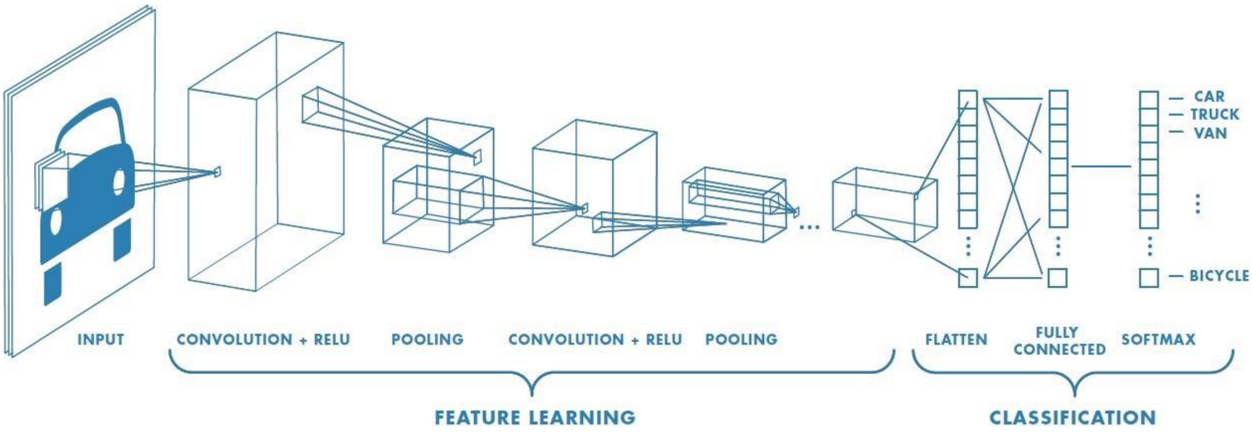


Generative models capture the joint probability P(X|Y) or P(X) if there are no labels.
Unsupervised models that summarize the distribution of input variables may be able to be used to create or generate new examples in the input distribution.
For example, a single variable may have a known data distribution, such as a Gaussian distribution or bell shape. A generative model may be able to sufficiently summarize this data distribution and then be used to generate new variables that plausibly fit into the distribution of the input variable.

There are many types of generative models; the most popular ones are

1. Variational Autoencoders
2. Generative adversarial networks

1.2 Convolution Neural Networks (CNN)



Convolution Neural Networks (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other.

A convert is a sequence of layers, and every layer transforms on volume to another through a differentiable function.

Types of layers:

- Input Layer: This layer holds the raw input of the image with width, height, and depth.
- Convolution Layer: This layer computes the output volume by computing the dot product between all filters and the image patch.
- Activation Function Layer: This layer will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are ReLU, Sigmoid, Tanh, Leaky ReLU, etc.,
- Pool Layer: This layer is periodically inserted in the converts and its main function is to reduce the size of volume which makes the computation fast, reduces memory, and also prevents overfitting. Two types of pooling layers are max pooling and average pooling.
- Fully connected Layer: This layer is a regular neural network layer that takes input from the previous layer and computes the class scores and outputs the 1-D array of size equal to the number of classes.

1.3 Recurrent Neural Networks (RNN)

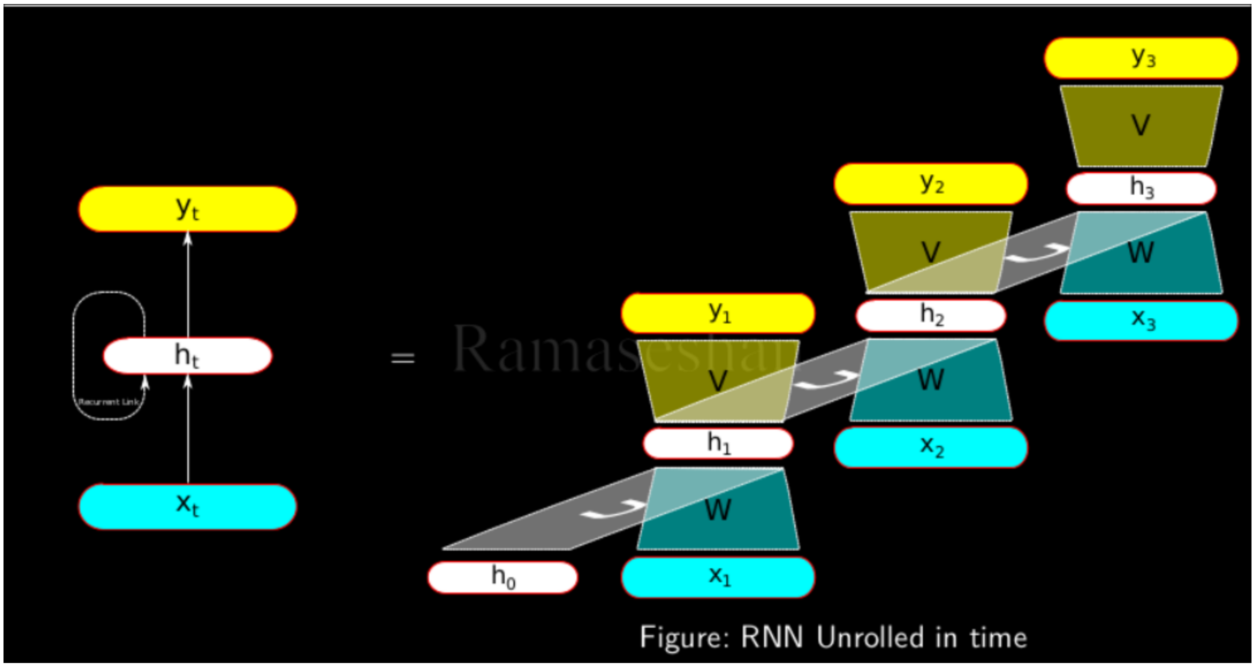


Figure: RNN Unrolled in time

- Recurrent Neural Networks (RNN) are a type of Neural network where the output from the previous step is fed as input to the current step.
- In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.
- Thus, RNN came into existence, which solves this issue with the help of a hidden state, which remembers some information about a sequence.
- RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous output by giving each output as input to the next hidden layer.

Word Embedding

It is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that words that are closer in the vector space are expected to be similar in meaning. It can be obtained using a set of language modeling and feature learning techniques where words or phrases from the vocabulary are mapped to vectors of real numbers.

1.4 Generative Adversarial Networks

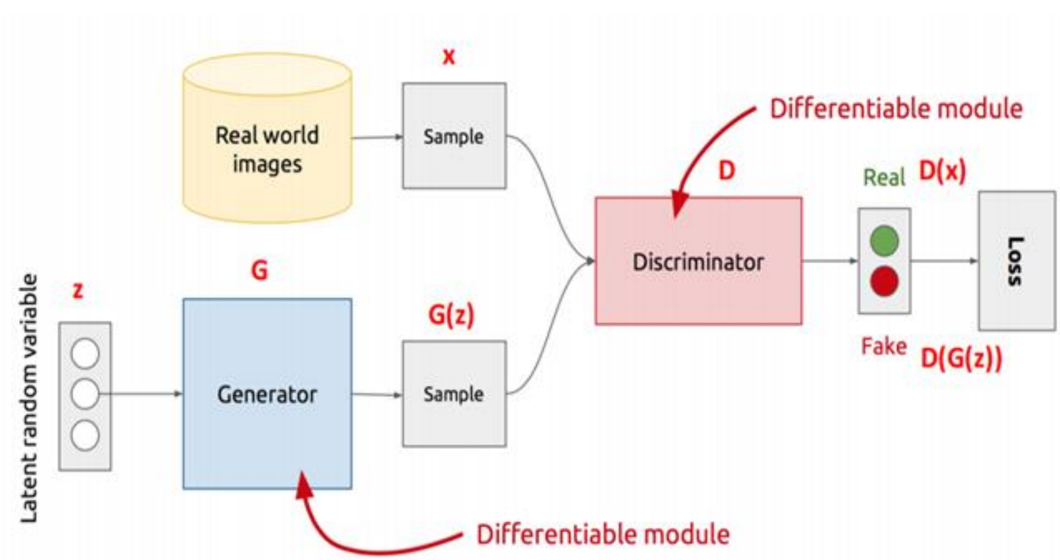
Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset.

Generative Adversarial Networks (GANs) can be broken down into three parts:

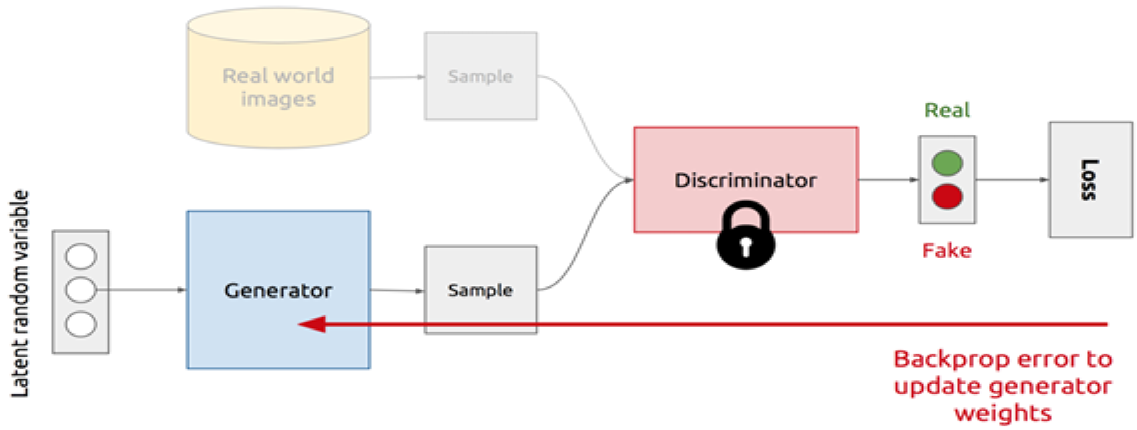
- Generative: To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- Adversarial: the training of a model is done in an adversarial setting.
- Networks: Use deep neural networks as the artificial intelligence (AI) algorithms for training purposes.

Generative adversarial networks are based on a game-theoretic scenario in which the generator network must compete against an adversary. The generator network directly produces samples. Its adversary, the discriminator network, attempts to distinguish between samples drawn from the training data and samples drawn from the generator.

GAN's Architecture



Generator



This model is used to generate new plausible examples from the problem domain.

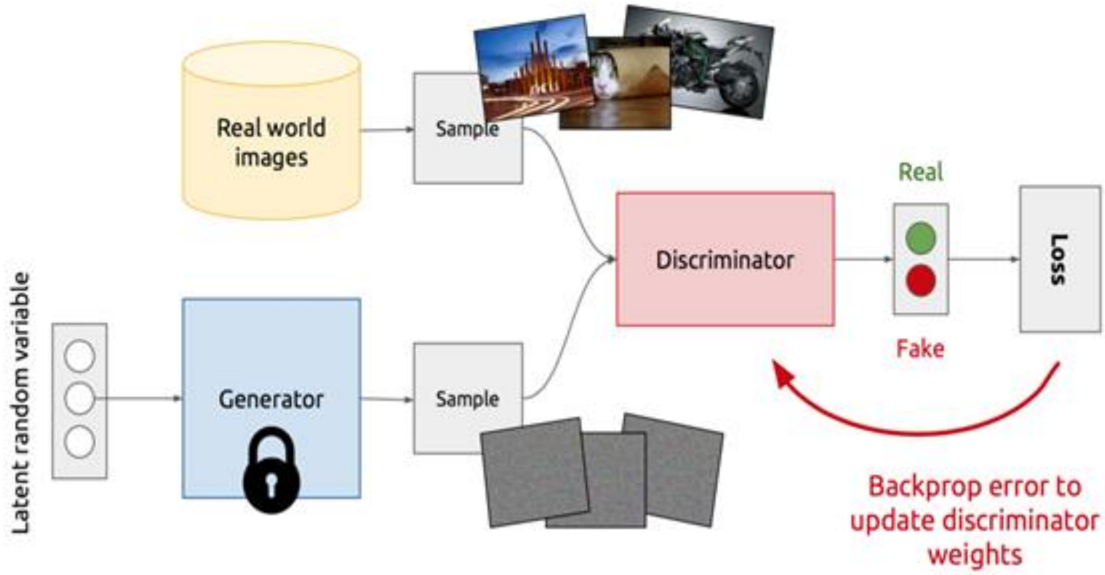
The Generator generates fake samples of data (be it an image, audio, etc.) and tries to fool the discriminator.

The generator model takes a fixed-length random vector as input and generates a sample in the domain. The Vector is drawn randomly from a Gaussian distribution, and the vector is used to seed the generative process.

Train the generator with the following procedure:

- Sample random noise
- Produce generator output from sampled random noise
- Get discriminator "Real" or "Fake" classification and generator output.
- Calculate loss from discriminator classification
- Backpropagate through both the discriminator and generator to obtain gradients.
- Use gradients to change only the generator weights.

Discriminator



This model is used to classify examples as real (from the domain) or fake (generated).

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data creed by the generator. It could use any network architecture appropriate to the type of data it's classifying.

During the discriminator training, the generator does not train. Its weights remain constant while it produces examples for the discriminator to train on.

The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss.

During the discriminator training:

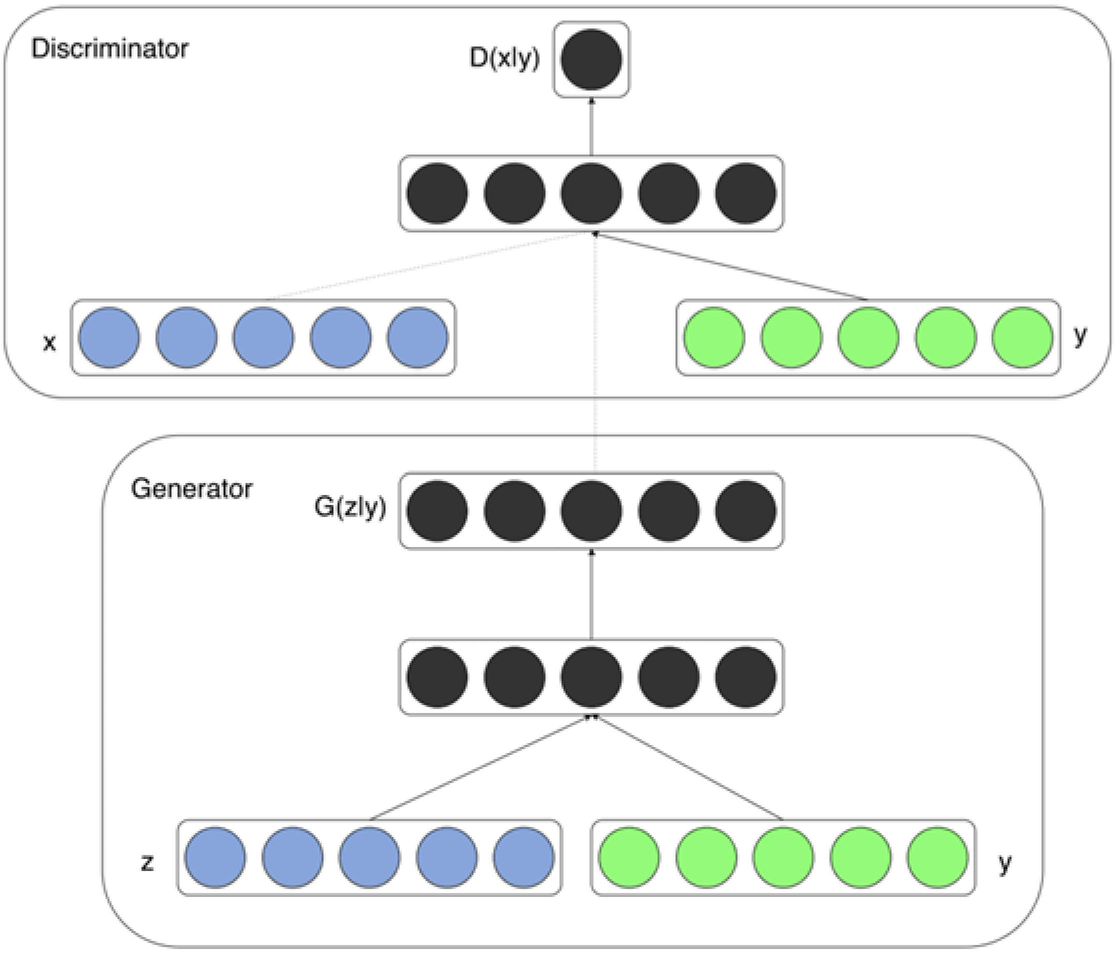
- The discriminator classifies both real data and fake data from the generator.
- The discriminator loss penalized the discriminator for misclassifying a real instance as fake or a fake instance as real.
- The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator work.

Overall, the training procedure is a min-max two-player game with the following objective function.

$$\min_G \max_D V(D, G)$$

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

1.5 Conditional GAN's



The conditional generative adversarial network, or cGAN for short, is a type of GAN that involves the conditional generation of images by a generator model. Image generation can be conditional on a class label, if available, allowing the targeted generation of images of a given type.

CGANs are mainly employed in image labeling, where both the generator and the discriminator are fed with some extra information **y** which works as auxiliary information, such as class labels from or data associated with different modalities.

The conditioning is usually done by feeding the information **y** into both the discriminator and the generator as an additional input layer to it.

CHAPTER-2

LITERATURE SURVEY

Generative Adversarial Networks (GANs) are an attractive framework that can be used to mimic complex real-world distributions by solving a min-max optimization problem between generator and discriminator. The generator intends to synthesize visually plausible images to fool the discriminator, while the discriminator attempts to distinguish the synthetic images from real images. With proper adversarial training, the generator can synthesize high-quality images.

Scott et al. (Scott Reed, "Generative Adversarial Text to Image Synthesis," 2016.) within the paper titled "**Generative Adversarial Text to Image Synthesis**" inferred that Deep convolutional generative adversarial networks (GANs) have begun to manufacture exceptionally convincing photos of specific classifications. For instance, faces, collection covers, and room insides. Right now, novel profound engineering and GAN setup are formed to successfully connect these advances in content and image demonstrating, deciphering visual concepts from characters to pixels. The model was equipped for making conceivable photos of winged creatures and blossoms from point-by-point content depictions. The generalizability of their way to manage to take photos with various articles and variable foundations was shown with their outcomes on the MS-COCO dataset.

Tao et al. (X. H. Tao Xu, "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks," 2017) within the paper titled **AttnGAN: Fine-Grained Text to Image Generation** with basic cognitive process Generative Adversarial Networks details the utilization of an attentional generative network, the AttnGAN model that is employed to synthesize fine-grained details at completely different regions of the image. It will do so by taking note of the necessary words within the given text description. This AttnGAN outperforms the previous models by a big margin, beating the most effective reported inception score by a rise of fourteen when tried on the CUB dataset and by 25% on the far more difficult coco dataset. For the first time, it had been seen that for generating completely different elements of the image the superimposed attentional GAN is in a position to mechanically choose the condition at the word level.

Han et al. (M. Han Zhang, "**StackGAN++: realistic image synthesis with stacked generative adversarial networks**," 2018.) proposed to utilize StackGAN++. They proposed to utilize stacked generative adversarial networks for practical picture union. Even though generative adversarial systems (gans) have had momentous triumphs in different errands, they face difficulties in creating top-notch pictures. Right now, stackgan was proposed for producing high-goals photograph practical pictures. The stackganv1 with molding enlargement is first proposed for text-to-picture combination through a novel sketch-refinement process. It prevails with regards to producing pictures of 256Ã—256 goals with photograph sensible subtleties from content portrayals. To additionally improve the nature of created tests and settle gans' preparation, the stackgan-v2 is actualized.

With conditional GANs, Reed et al. (S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text-to-image synthesis. In ICML, 2016) successfully generated plausible 64×64 images for birds and flowers based on text descriptions. Their follow-up work was able to generate 128×128 images by utilizing additional annotations on object part locations. Given the difficulties in modeling details of natural images, many works have been proposed to use multiple GANs to improve sample quality.

Wang et al. (X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. In ECCV, 2016) utilized a structure GAN and a style GAN to synthesize images of indoor scenes. Yang et al. (J. Yang, A. Kannan, D. Batra, and D. Parikh. LR-GAN: layered recursive generative adversarial networks for image generation. In ICLR, 2017) factorized image generation into foreground and background generation with layered recursive GANs.

Huang et al. (X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie. Stacked generative adversarial networks. In CVPR, 2017) added several GANs to reconstruct the multilevel representations of a pre-trained discriminative model. But they were unable to generate high-resolution images with photo-realistic details.

Durugkar et al. (I. P. Durugkar, I. Gemp, and S. Mahadevan. Generative multi-adversarial networks. In ICLR, 2017) used multiple discriminators along with one generator to increase the chance of the generator receiving effective feedback. However, all discriminators in their framework are trained to approximate the image distribution at a single scale.

CHAPTER-3

SYSTEM ANALYSIS

3.1 Existing Model

StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks.

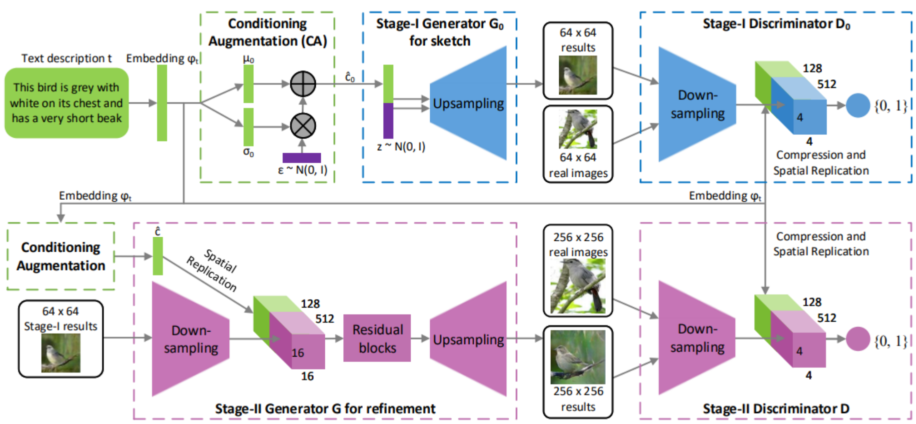
Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris N. Metaxas

The existing text-to-image synthesis model, Stacked Generative Adversarial Networks, StackGAN-v1, and StackGAN-v2, is proposed to decompose the difficult problem of generating realistic high-resolution images into more manageable sub-problems.

StackGAN-V1

The StackGAN-v1, to generate high-resolution images with photo-realistic details, has a simple yet effective two-stage generative adversarial network. As shown in fig, it decomposes the text-to-image generative process into two stages.

- Stage-I GAN sketches the primitive shape and basic colors of the object conditioned on the given text description and draws the background layout from a random noise vector, yielding a low-resolution image.
- Stage-II GAN corrects defects in the low-resolution image from Stage-I and completes details of the object by reading the text description again, producing a high-resolution photo-realistic image.



The StackGANv1 with Conditioning Augmentation is first proposed for text-to-image synthesis through a novel sketch-refinement process. It succeeds in generating images of 256×256 resolution with photo-realistic details from text descriptions.

StackGAN-V2

To further improve the quality of generated samples and stabilize GANs' training, StackGAN-v2, to model a series of multi-scale image distributions. As shown in Fig

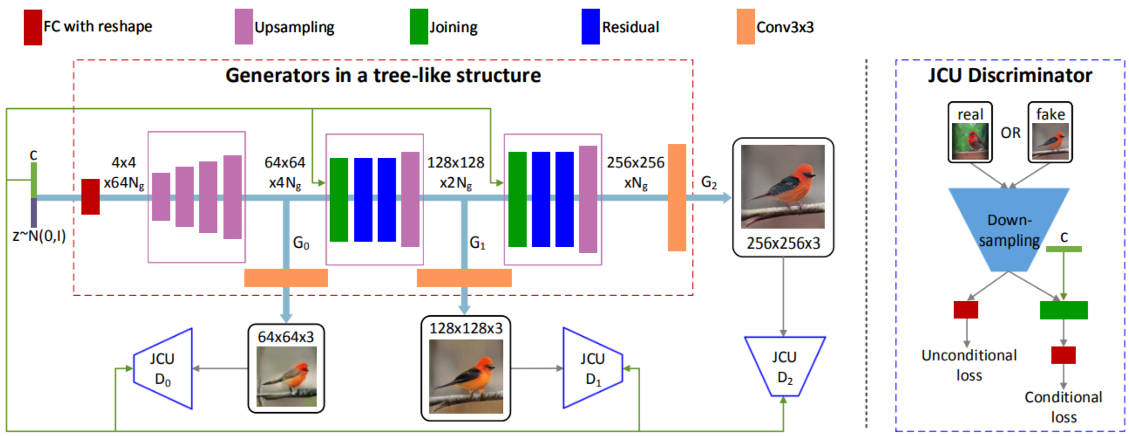


Fig. 2: The overall framework of our proposed StackGAN-v2 for the conditional image synthesis task. c is the vector of conditioning variables which can be computed from the class label, the text description, *etc.*. N_g and N_d are the numbers of channels of a tensor.

It consists of multiple generators (Gs) and discriminators (Ds) in a tree-like structure. Images from low-resolution to high-resolution are generated from different branches of the tree. At each branch, the generator captures the image distribution at that scale and the discriminator estimates the probability that a sample came from training images of that scale rather than the generator. The generators are jointly trained to approximate the multiple distributions, and the generators and discriminators are trained in an alternating fashion. It jointly approximates multiple related distributions, including (1) multi-scale image distributions and (2) jointly conditional and unconditional image distributions. In addition, a color-consistency regularization is proposed to facilitate multi-distribution approximation.

Datasets

We evaluate our conditional StackGAN for text-to-image synthesis on the CUB, Oxford-102, and COCO datasets. CUB contains 200 bird species with 11,788 images. Oxford-102 contains 8,189 images of flowers from 102 different categories. To show the generalization capability of our approach, a more challenging dataset, COCO is also utilized for evaluation. Different from CUB and Oxford-102, the COCO dataset contains images with multiple objects and various backgrounds. Each image in COCO has 5 descriptions, while 10 descriptions are provided for every image in CUB and Oxford-102 datasets. We directly use the training and validation sets provided by COCO, meanwhile, we split CUB and Oxford-102 into class-disjoint training and test sets.

Evaluation Metric

Inception Score

$$\mathrm{IS} = \exp(\mathbb{E}_{x \sim p_\varepsilon} \underbrace{D_{KL}(p(y \mid x) \| p(y))}_{\text{KL Divergence}})$$

- X denotes one generated sample.
- Y is the label predicted by the inception model.
- KL divergence between the marginal distribution p(y) and the conditional distribution p(y|x) should be large.
- Higher IS means higher quality of the generated images, and each image clearly belongs to a specific class.

Frechet Inception Distance

$$\|\mu_X - \mu_Y\|^2 + \mathrm{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

- It directly measures the distance between the synthetic data distribution and the real data distribution.
- Lower FID values mean closer distances between synthetic and real data distributions.

The FID and IS score for StackGAN-V1 is 51.89 and 3.70 $\pm$ .04

The FID and IS score for StackGAN-V2 is 15.30 and 4.04 $\pm$ .05

The samples generated by StackGAN-v2 are consistently better than those by StackGAN-v1. The end-to-end training scheme together with the color-consistency regularization enables StackGAN-v2 to produce more feedback and regularization for each branch so that consistency is better maintained during the multi-step generation process.

Drawbacks

- For the backbone, there are multiple generators and discriminators stacked for generating different scales of images making the training process slow and inefficient.
- For semantic consistency, the existing models employ extra networks to ensure semantic consistency, increasing the training complexity and bringing an additional computational cost.

Although impressive results have been presented by stacked text-to-image GANs, there remain two problems for stacked generators and discriminators:

- The initial images have a great influence on the final refined results, and if the initial image is not well synthesized, the following generators can hardly refine it to a satisfactory quality.
- Different discriminators correspond to different image scales, and each discriminator predicts a discriminator loss to evaluate the input image. The Stacked GANs family makes a summation of all discriminator losses as the whole discriminator loss for training end to end. But it is hard to balance all discriminator losses while training. This uncertainty makes the final refined images look like a simple combination of fuzzy shapes and some details.

3.2 Proposed Model

DF-GAN: Deep Fusion Generative Adversarial networks for Text-to-Image Synthesis
Ming Tao, Hao Tang, Songsong Wu, Nicu Sebe, Fei Wu, and Xiao-Yuan Jing

The goal of our proposed Model is to generate realistic and text-image semantic consistent images from given natural language descriptions.
To overcome the existing model drawbacks or limitations, our proposed model is much different from previous models. DF-GAN uses 1. A novel simplified text-to-image backbone which can directly generate high-resolution images from text descriptions by one pair of generator and discriminator. 2. A novel DF-Block (Deep Text-Image Fusion Block) is proposed which fuses text and image features more effectively and deeply. Armed with Df-Block, the generator archives higher performance in text-to-image generation. 3. A novel regularization method matching-aware zero-centered Gradient penalty (MA-GP) which promotes the generator to synthesize more realistic and text-image semantic consistent images without introducing extra networks. Finally, the one-way discriminator is employed to promote the effectiveness of MA-GP.
Compared with the previous text-to-image models, our proposed model is simpler and more efficient and achieves better performance.

Advantages

- Compared with previous models, the proposed model is able to directly synthesize more realistic and text-image semantic consistent images without stacking architecture and extra networks.

- The experimental results on two challenging datasets prove that the proposed DF-GAN outperforms previous state-of-the-art text-to-image models.

3.3 Hardware Requirements

1. RAM : 8GB
2. Hard Disk : 1TB
3. Processor : Corei5 or Higher
4. Speed : 2.6Ghz
5. GPU : Nvidia-P100GPU

3.4 Software Requirements

Operating System

In Python, file names, command-line arguments, and environment variables are represented using the string type. On some systems, decoding these strings to and from bytes is necessary before passing them to the operating system. Python uses the file system encoding to perform this conversion (see sys.getfilesystemencoding()). This module provides a portable way of using operating system-dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling sees the shutil module.

The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about the path in the same format (which happens to have originated with the POSIX interface). Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability. All functions accepting path or file names accept both bytes and string objects and result in an object of the same type if a path or file name is returned. On VxWorks, os.fork, os.execv, and os.spawnp are not supported. All functions in this module raise OSError (or subclasses thereof) in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type but are not accepted by the operating system.

About Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python’s simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

Python 3.6 picks up where many of those improvements left off and nudges them into new realms. Python 3.5 added syntax used by static type checking tools to ensure software quality; Python 3.6 expands on that idea, which could eventually lead to high-speed statically compiled Python programs. Python3.5 gave us options to write asynchronous functions; Python 3.6 bolsters them. But the biggest changes in Python 3.6 lie under the hood, and they open up possibilities that didn't exist before.

Google Colaboratory

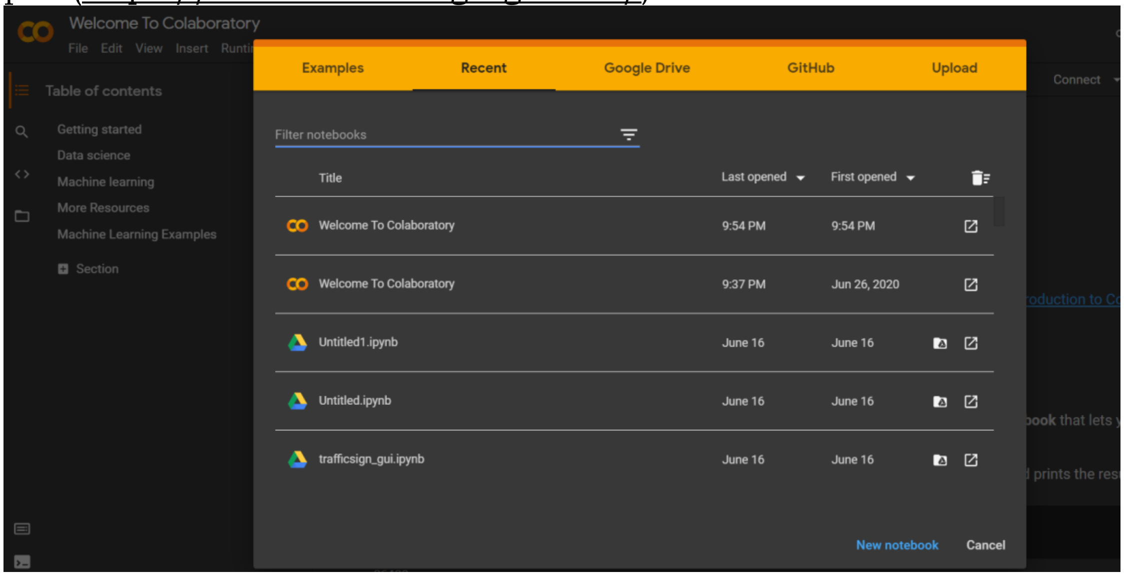
- Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members just the way you edit documents in Google Docs.

- Colab supports many popular machine learning libraries which can be easily loaded in your notebook. If you have a GPU or a good computer, creating a local environment with anaconda and installing packages and resolving installation issues are a hassle.

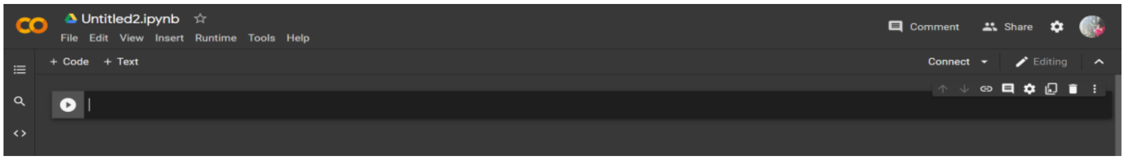
- Colaboratory is a free Jupyter notebook environment provided by Google where you can use free GPUs and TPUs which can solve all these issues.

How to run a Python Code in Colab

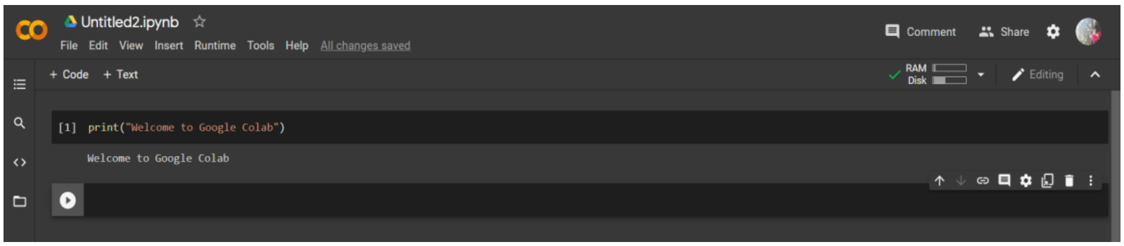
1. open (https://colab.research.google.com/)



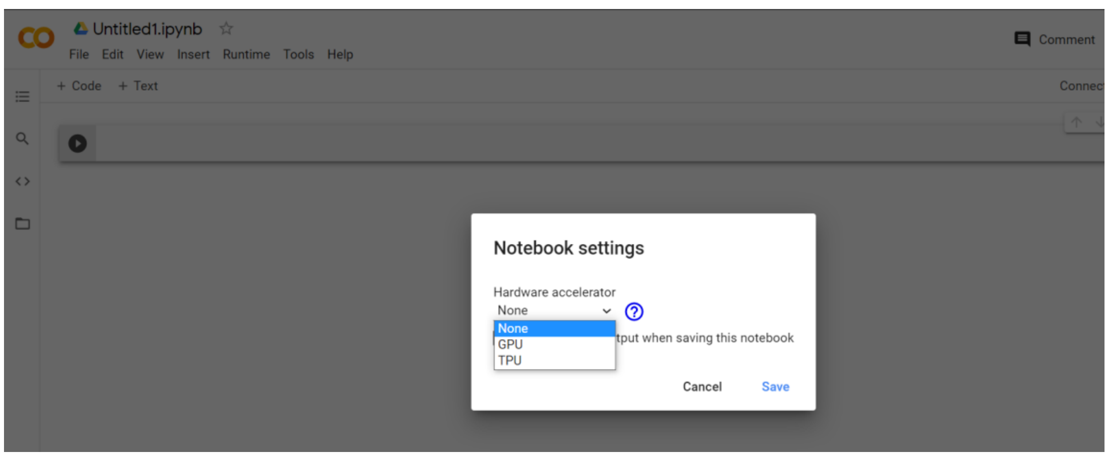
2. Click on the NEW NOTEBOOK link at the bottom of the screen. A new notebook would open up as shown on the screen below.



3. Enter a trivial Python code in the code cell and execute it, to execute the code, click on the left side of the code cell.



4. To change the runtime environment, click the “Runtime” dropdown menu. Select “Change runtime type”. Now select anything (GPU, CPU, TPU) we want in the “Hardware accelerator” dropdown menu.



5. To mount our drive, execute the following lines

```
from google.colab import drive
drive.mount('/content/drive')
```

Then you’ll see a link, click on the link, then allow access, copy the code that pops up, paste it at “Enter your authorization code:”.

About PyTorch

- PyTorch is a Python machine learning package based on Torch, which is an open-source machine learning package based on the programming language Lua.
- PyTorch is also great for deep learning research and provides maximum flexibility and speed.

PyTorch has two main features:

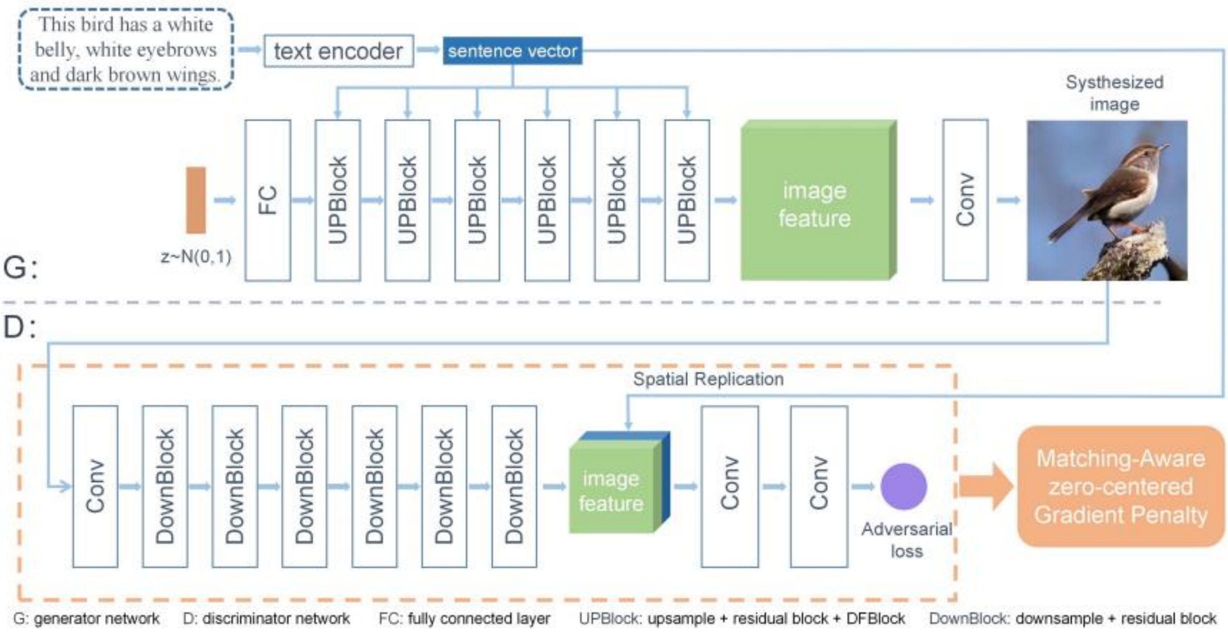
1. Tensor computation (like NumPy) with strong GPU acceleration.
2. Automatic differentiation for building and training neural networks

CHAPTER-4

SYSTEM IMPLEMENTATION

4.1 Deep Fusion Generative Adversarial Networks (DF-GAN)

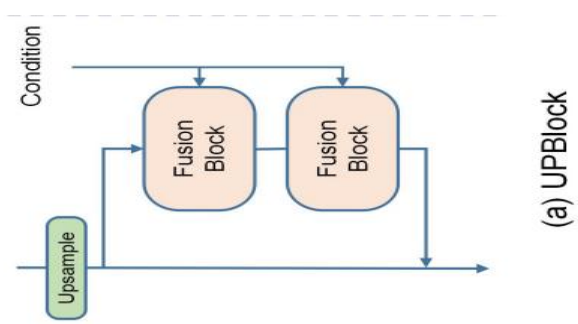
Architecture



The entire network is composed of a generator, a discriminate, and a pre-trained text encoder.

Generator Structure

- The generator has two inputs, a sentence vector that is encoded by text encoder and a noise vector sampled from the gaussian distribution to ensure the diversity of generated images.
- The noise vector is first fed into a fully connected layer and the output is reshaped to (-1, 4, 4).
- Then apply a series of UPBlocks to upsample the image features.
- The UPBlock is composed of upsample layers, a residual block, and DFBlocks to fuse the text and image features during the image generation process.



- Finally, a convolution layer converts image features into images.

Discriminator Structure

- The discriminator is composed of some DownBlocks and convolution layers.
- This converts images into feature maps and the output is downsampled by a series of DownBlocks.
- Then the sentence vector will be replicated and concatenated on the image feature.
- An adversarial loss will be predicated to evaluate the visual realism and semantic consistency of inputs.
- By distinguishing generated images from real samples, the discriminator promotes the generator to synthesize images with higher quality and text-image semantic consistency.

Simplified Text-to-Image Backbone

A simplified text to image backbone which can synthesize high-resolution images directly by one pair of generator and discriminator. Our proposed simplified text-to-image backbone can even achieve better performance than most well-designed stacked GANs.
We employ hinge loss to stabilize the training process.

$$\begin{aligned} L_D = &- \mathbb{E}_{x\sim\mathbb{P}_r}[min(0, -1 + D(x, e))] \\ &- (1/2)\mathbb{E}_{G(z)\sim\mathbb{P}_g}[min(0, -1 - D(G(z), e))] \\ &- (1/2)\mathbb{E}_{x\sim\mathbb{P}_{mis}}[min(0, -1 - D(x, e))] \\ L_G = &- \mathbb{E}_{G(z)\sim\mathbb{P}_g} D(G(z), e) \end{aligned}$$

When Z is the noise vector sampled from Gaussian distribution. E is the sentence vector. P_g, P_r, P_{mis} denotes the synthetic data distribution, real data distribution, and mismatching data distribution, respectively.

Matching-Aware Zero-Centered Gradient Penalty

- A novel conditional Matching Aware Zero-Centered Gradient Penalty (MA-GP) to enable the generator to synthesize more realistic and text-image semantic consistent images.
- The discriminator should do two things to ensure the quality of synthetic data.
  - First, it should put the real data at the minimum point and put synthetic data at a high point.

- Second, it should ensure that the loss surface of the real data point and its vicinity are smooth to help the generator converge.
- The discriminator observes four kinds of inputs:
    - Synthetic images with matching text, Synthetic images with mismatched text.
    - Real images with matching text, Real images with mismatched text.
- To generate text-matching and realistic images from given text descriptions, we should put real and matching data points to the minimum point and put other inputs at high points, and ensure a smooth vicinity of real and matching data points to help the generator converge to the minimum point.

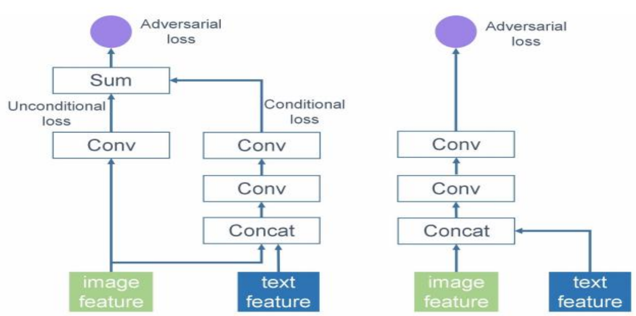
- The whole formulation of our model is:

$$\begin{aligned} L_D = &- \mathbb{E}_{x\sim\mathbb{P}_r}[min(0, -1 + D(x, e))] \\ &- (1/2)\mathbb{E}_{G(z)\sim\mathbb{P}_g}[min(0, -1 - D(G(z), e))] \\ &- (1/2)\mathbb{E}_{x\sim\mathbb{P}_{mis}}[min(0, -1 - D(x, e))] \\ &+ k\mathbb{E}_{x\sim\mathbb{P}_r}[(\|\nabla_x D(x, e)\| + \|\nabla_e D(x, e)\|)^p] \\ L_G = &- \mathbb{E}_{G(z)\sim\mathbb{P}_g} D(G(z), e) \end{aligned}$$

- Where k and p are two hyperparameters. We set the k=2 and p=6 in our network.
- MA-GP does not employ extra networks to compute text-image semantic similarity.

One Way Discriminator

- The two-way discriminator slows the convergence of the generator network and weakens the effectiveness of MA-GP.



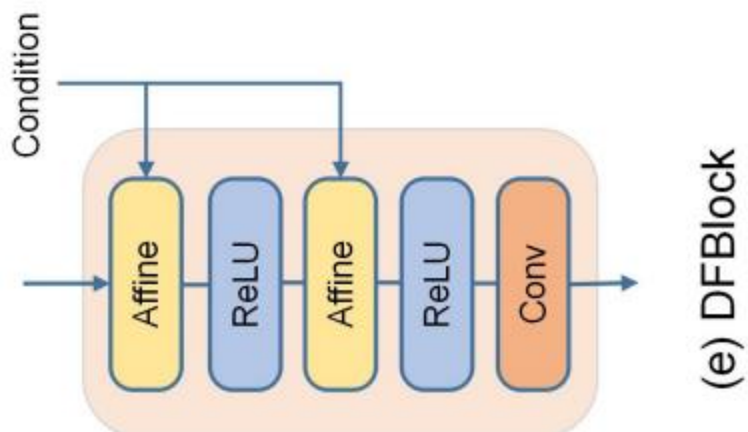
(a) two-ways discriminator (b) one-way discriminator

- The one-way discriminator concatenates the image feature and sentence vector, then outputs one adversarial loss through two convolution layers.

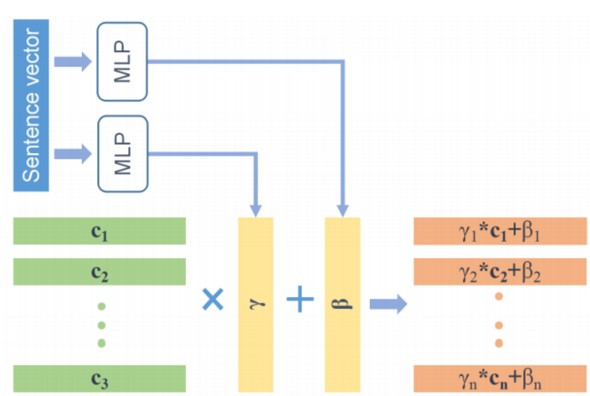
- The one-way discriminator only gives one gradient pointing to the real and match data points, it gives a more clear convergence goal to the generator.
- Armed with MA-GP, the one-way discrimination will guide the generator to synthesize more realistic images with better text-image semantic consistency.

Deep Text-Image Fusion Block

- A novel Deep text-image Fusion Block (DFBlock) fuses text and visual information more effectively and deeply during the generation process.
- The DFBlock is composed of a series of Affine Transformations, ReLU layers, and convolution layers.



Affine Transformation



- The normalization slightly reduces the efficiency of the text-image fusion process.

- Affine Transformation manipulates the output by scaling and shifting parameters predicted from conditions.
- We only employ Affine Transformation to manipulates visual feature maps conditioned on natural language descriptions.
- We adopt two one-hidden-layer MLPs to predict the language-conditioned channel-wise scaling parameters $\boldsymbol{\gamma}$ and shifting parameters $\boldsymbol{\beta}$ from the sentence vector *e*, respectively.

$$\boldsymbol{\gamma} = MLP_1(\boldsymbol{e}), \qquad \boldsymbol{\beta} = MLP_2(\boldsymbol{e}).$$

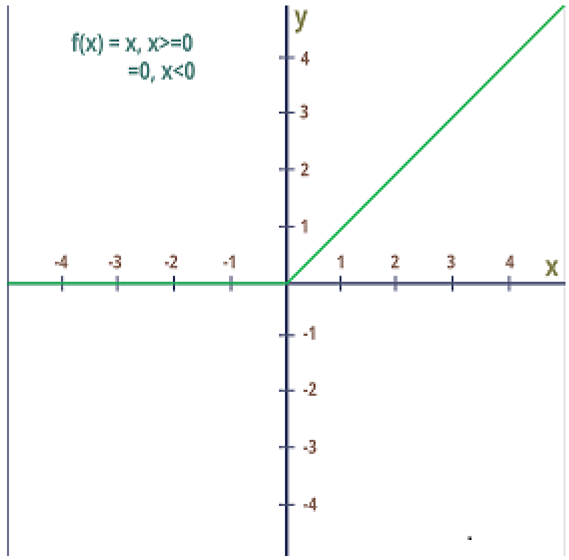
- If the input is a feature map X $R^{B \times C \times H \times W}$, the output of MLP will be a vector of size C. We first conduct the channel-wise scaling operation on X with the scaling parameter $\boldsymbol{\gamma}$, then we apply the channel-wise shifting operation on X with the shifting parameter $\boldsymbol{\beta}$. This process can be formally expressed as follows:

$$AFF(\boldsymbol{x_i}|\boldsymbol{e}) = \gamma_i \cdot \boldsymbol{x_i} + \beta_i,$$

- Where AFF is affine Transformation, x_i is the i^{th} channel of visual feature maps, e is the sentence vector, γ_i and $\boldsymbol{\beta}_i$ is the scaling parameter and shifting parameter for the i^{th} channel f visual feature maps.
- Through channel-wise scaling and shifting, the generator can capture the semantic information in the text description and synthesize realistic images matching with given text descriptions.
- The DF-Block deepens the depth of the text-image fusion process. For neural networks, a deeper network always means a stronger ability.
- The deeping the fusion process brings three main benefits for text-to-image generation:
  - First, it gives the generator more chances to fuse text and image features, so that the text information can be fully exploited.
  - Second, deepening the fusion process makes the fusion network have more nonlinearities, which is beneficial to generate semantic consistent images from different text descriptions.
  - Third, stacking multiple Affine Transformations can achieve a more complex and effective fusion process.
- DFBlock does not normalize the feature map before the scale and shift operation.

ReLU Activation Function

Stands for a Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural Network.



- **Equation**: A(X) = max (0, X). It gives an output X if X is positive and 0 otherwise.
- **Value Range**: [0, inf)
- Nature: Non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- Uses: ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network spare making it efficient and easy for computation.

4.2 Datasets

We evaluate the proposed model on two challenging datasets. i.e., CUB bird and COCO. The CUB bird dataset is commonly used in text-to-image generation, which contains 11788 images belonging to 200 bird species. Each bird image has 10 language descriptions. We split 150 bird species with 8855 images as the training set and 50 bird species with 2933 images as the test set.
The COCO dataset contains 80K images for training and 40K images for testing. Compared with the CUB dataset, the images in COCO are more complex, which makes it more challenging for text-to-image generation tasks.

4.3 Training Details

- We optimize our network using Adam with $\beta_1 = 0.0$ and $\beta_2 = 0.9$.
- The learning rate is set to 0.0001 for the generator and 0.0004 for the discriminator according to the Two Timescale Update Rule (TTUR).
- The training is performed for 600 epochs for the CUB birds dataset and 120 epochs for the COCO dataset.

- We use the pre-trained sentence encoder and fix its parameters during training.
- During the discriminator training:
  - The generator does not train. Its weights remain constant while it produces examples for the discriminator to train on.
  - The discriminator ignores the generator loss and just uses the discriminator loss.
  - The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.
- Generator training requires more integration between the generator and the discriminator than discriminator training requires.
- During generator training, the discriminator does not train. Its weights remain constant.

4.4 Evaluation Details

Inception Score

$$\mathrm{IS} = \exp\left(\mathbb{E}_{x\sim p_\varepsilon} \underbrace{D_{KL}(p(y \mid x) \| p(y))}_{\text{KL Divergence}}\right)$$

- X denotes one generated sample.
- Y is the label predicted by the inception model.
- KL divergence between the marginal distribution p(y) and the conditional distribution p(y|x) should be large.
- Higher IS means higher quality of the generated images and each image clearly belongs to a specific class.

Frechet Inception Distance

$$\|\mu_X - \mu_Y\|^2 + \mathrm{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

- It directly measures the distance between the synthetic data distribution and the real data distribution.
- Lower FID values mean closer distances between synthetic and real data distributions.

CHAPTER-5
SOURCE CODE

GENERATOR

The generator has two inputs, a sentence vector that is encoded by text encoder and a noise vector.Then apply a series of UPBlocks to upsample the image features.The UPBlock is composed upsample layers, a residual block, and DFBlocks

```
#Generator Architecture
class Generator(nn.Module):
  def __init__(self,ngf=64,nz=100):
    super(Generator,self).__init__()
    self.ngf=ngf
    self.fc=nn.Linear(nz,ngf*8*4*4)
    self.Upblock0=UpBlock(ngf*8, ngf*8)
    self.Upblock1=UpBlock(ngf*8, ngf*8)
    self.Upblock2=UpBlock(ngf*8, ngf*8)
    self.Upblock3=UpBlock(ngf*8, ngf*4)
    self.Upblock4=UpBlock(ngf*4, ngf*2)
    self.Upblock5=UpBlock(ngf*2, ngf*1)
    self.convimg=nn.Sequential(nn.LeakyReLU(0.2,inplace=True),
                               nn.Conv2d(ngf,3,3,1,1),
                               nn.Tanh())#convolution 2d
  def forward(self,x,c):
    out=self.fc(x)
    out=out.view(x.size(0),8*self.ngf,4,4)
    out=self.Upblock0(out,c)

    out=F.interpolate(out,scale_factor=2)#upsampling
    out=self.Upblock1(out,c)

    out=F.interpolate(out,scale_factor=2)
    out=self.Upblock2(out,c)

    out=F.interpolate(out,scale_factor=2)
    out=self.Upblock3(out,c)

    out=F.interpolate(out,scale_factor=2)
    out=self.Upblock4(out,c)

    out=F.interpolate(out,scale_factor=2)
    out=self.Upblock5(out,c)

    out=F.interpolate(out,scale_factor=2)
    out=self.Upblock6(out,c)

    out=self.convimg(out)
    return out#output is synthesized image
```

UPBLOCK

The upblock is composed of upsample layers,a residual block, and DFBlocks to fuse the text and image features during the image generation process

```
class UpBlock(nn.Module):
  def __init__(self,in_ch,out_ch):
    super(UpBlock,self).__init__()
    self.learnable_sc=in_ch!=out_ch
    self.c1=nn.Conv2d(in_ch,out_ch,3,1,1)
    self.c2=nn.Conv2d(out_ch,out_ch,3,1,1)
    self.affine0=Affine(in_ch)
    self.affine1=Affine(in_ch)
    self.affine2=Affine(out_ch)
    self.affine3=Affine(out_ch)
    self.gamma=nn.Paramatere(torch.zeros(1))
    if(self.learnable_sc):
      self.c_sc=nn.Conv2d(in_out,out_ch,1,stride=1,padding=0)
  def forward(self,x,y=None):
    k=self.shortcut(x)+self.gamma*self.residual(x,y)
    return k
  def shortcut(self,x):
    if(self.learnable_sc):
      x=self.c_sc(x)
  #DF-Block
  def residual(self,x,y=None):
    h=self.affine0(x,y)
    h=nn.LeakyReLU(0.2,inplace=True)(h)
    h=self.affine1(h,y)
    h=nn.LeakyReLU(0.2,inplace=True)(h)
    h=self.c1(h)

    h=self.affine2(h,y)
    h=nn.LeakyReLU(0.2,inplace=True)(h)
    h=self.affine3(h,y)
    h=nn.LeakyReLU(0.2,inplace=True)(h)
    h=self.c2(h)
    return h;
```

4s completed at 8:41 AM

DISCRIMINATOR

The discriminator is composed of some DownBlocks and convolution layers. This converts images into feature maps and the output is downsampled by a series of DownBlocks.Then the sentence vector will be replicated and concatenated on the image feature.

```
#Discriminator architecture
class Discriminator(nn.Module):
  def __init__(self,ndf):
    super(Discriminator,self).__init__()
    self.covimg=nn.Conv2d(3,ndf,3,1,1)
    self.Downblock0=DownBlock(ndf*1,ndf*2)
    self.Downblock1=DownBlock(ndf*2,ndf*4)
    self.Downblock2=DownBlock(ndf*4,ndf*8)
    self.Downblock3=DownBlock(ndf*8,ndf*16)
    self.Downblock4=DownBlock(ndf*16,ndf*16)
    self.Downblock5=DownBlock(ndf*16,ndf*16)

    self.Cond_D=D_GET_LOGITS(ndf)

  def forward(self,x):#input is image
    out=self.convimg(x)
    out=self.Downblock0(out)
    out=self.Downblock1(out)
    out=self.Downblock2(out)
    out=self.Downblock3(out)
    out=self.Downblock4(out)
    out=self.Downblock5(out)
    return out#output is features of images
```

DOWN BLOCKS

```
class DownBlock(nn.Module):
  def __init__(self,fin,fout,dowmsample=True):
    super().__init__()
    self.downsample=downsample
    self.learned_shortcut=(fin!=fout)
    self.conv_r=nn.Sequential(
        nn.Conv2d(fin,fout,4,2,1,bias=False),
        nn.LeakyReLU(0.2,inplace=True),

        nn.Conv2d(fout,fout,3,1,1,bias=True),
        nn.LeakyReLU(0.2,inplace=True),
        )
    self.conv_s=nn.Conv2d(fin,fout,1,1stride=1,padding=0)
    self.gamma=nn.Parameter(torch.zeros(1))

  def forward(self,x,c=None):
    k=self.shortcut(x)+self.gamma * self.residual(x)
    return k

  def shortcut(self,x):
    if(self.learned_shortcut):
      x=self.conv_s(x)
    if(self.downsample):
      return F.avg_pool2d(x,2)
    return x

  def residual(self,x):
    return self.conv_r(x)
```

AFFINE TRANSFORMATION

```
[ ]  class Affine(nn.Module):
       def __init__(self,num_features):
         super(Affine,self).__init__()
         self.fc_gamma = nn.Sequential(OrderedDict([('linear1',nn.Linear(256,256)),
                                                    ('relu1',nn.ReLU(inplace=True)),
                                                    ('linear2',nn.Linear(256,num_features)),
                                                    ]))
         self.fc_beta=nn.Sequential(OrderedDict([('linear1',nn.Linear(256,256)),
                                               ('relu1',nn.ReLU(inplace=True)),
                                               ('linear2',nn.Linear(256,num_features)),
                                               ]))
         self._initialize()

       def _initalize(self):#initalize weights
         nn.init.zeros_(self.fc_gamma.linear2.weight.data)
         nn.init.ones_(self.fc_gamma.linear2.bias.data)
         nn.init.zeros_(self.fc_beta.linear2.weight.data)
         nn.init.zeros_(self.fc_beta.linear2.bias.data)

       def forward(self,x,y=None):#calculate aff(x)=gamma * weight +beta
         weight=self.fc_gamma(y)
         bias=self.fc_beta(y)
         if(weight.dim()==1):
           weight=weight.unsqueeze(0)
         if(bias.dim()==1):
           bias=bias.unsqueeze(0)
         size=x.size()
         weight=weight.unsqueeze(-1).unsqueeze(-1).expand(size)
         bias=bias.unsqueeze(-1).unsqueeze(-1).expand(size)
         k=weight*x+bias
         return k
```

CONCAT IMAGES FEATURES AND TEXT

```python
class D_GET_LOGITS(nn.Module):
  def __init__(self,ndf):
    super(D_GETS_LOGITS,self).__init__()
    self.df_dim=ndf
    self.joint_conv=nn.Sequential(
        nn.Conv2d(ndf*16+256, ndf*2,3,1,1,bias=False),
        nn.LeakyReLU(0.2,inplace=True),
        nn.Conv2d(ndf*2,1,4,1,0,bias=False),
    )#convluation

    def forward(self,out,y):
      y=y.view(-1,256,1,1)
      y=y.repeat(1,1,4,4)
      h_c_code=torch.cat((out,y),1)#concatination features and sentence embedding
      out=self.joint_conv(h_c_code)
      return out
```

RNN ENCODER

```python
class RNN_ENCODER(nn.Module):
    def __init__(self, ntoken, ninput=300, drop_prob=0.5,
                 nhidden=128, nlayers=1, bidirectional=True):
        super(RNN_ENCODER, self).__init__()
        self.n_steps = cfg.TEXT.WORDS_NUM
        self.ntoken = ntoken  # size of the dictionary
        self.ninput = ninput  # size of each embedding vector
        self.drop_prob = drop_prob  # probability of an element to be zeroed
        self.nlayers = nlayers  # Number of recurrent layers
        self.bidirectional = bidirectional
        self.rnn_type = cfg.RNN_TYPE
        if bidirectional:
            self.num_directions = 2
        else:
            self.num_directions = 1
        # number of features in the hidden state
        self.nhidden = nhidden // self.num_directions

        self.define_module()
        self.init_weights()

    def define_module(self):
        self.encoder = nn.Embedding(self.ntoken, self.ninput)
        self.drop = nn.Dropout(self.drop_prob)
        if self.rnn_type == 'LSTM':
            # dropout: If non-zero, introduces a dropout layer on
            # the outputs of each RNN layer except the last layer
            self.rnn = nn.LSTM(self.ninput, self.nhidden,self.nlayers, batch_first=True, dropout=self.drop_prob,bidirectional=self.bidirectional)
        elif self.rnn_type == 'GRU':
            self.rnn = nn.GRU(self.ninput, self.nhidden,self.nlayers, batch_first=True,dropout=self.drop_prob,bidirectional=self.bidirectional)
        else:
            raise NotImplementedError

    def init_weights(self):
        initrange = 0.1
        self.encoder.weight.data.uniform_(-initrange, initrange)
```

```python
def init_hidden(self, bsz):
    weight = next(self.parameters()).data
    if self.rnn_type == 'LSTM':
        return (Variable(weight.new(self.nlayers * self.num_directions,bsz,self.nhidden).zero_()),
                Variable(weight.new(self.nlayers * self.num_directions,bsz,self.nhidden).zero_()))
    else:
        return Variable(weight.new(self.nlayers * self.num_directions,bsz, self.nhidden).zero_())

def forward(self, captions, cap_lens, hidden, mask=None):
    # input: torch.LongTensor of size batch x n_steps
    # --> emb: batch x n_steps x ninput
    emb = self.drop(self.encoder(captions))
    #
    # Returns: a PackedSequence object
    cap_lens = cap_lens.data.tolist()
    emb = pack_padded_sequence(emb, cap_lens, batch_first=True)
    # #hidden and memory (num_layers * num_directions, batch, hidden_size):
    # tensor containing the initial hidden state for each element in batch.
    # #output (batch, seq_len, hidden_size * num_directions)
    # #or a PackedSequence object:
    # tensor containing output features (h_t) from the last layer of RNN
    output, hidden = self.rnn(emb, hidden)
    # PackedSequence object
    # --> (batch, seq_len, hidden_size * num_directions)
    output = pad_packed_sequence(output, batch_first=True)[0]#unpack
    # output = self.drop(output)
    # --> batch x hidden_size*num_directions x seq_len
    words_emb = output.transpose(1, 2)
    # --> batch x num_directions*hidden_size
    if self.rnn_type == 'LSTM':
        sent_emb = hidden[0].transpose(0, 1).contiguous()
    else:
        sent_emb = hidden.transpose(0, 1).contiguous()
    sent_emb = sent_emb.view(-1, self.nhidden * self.num_directions)
    return words_emb, sent_emb
```

LOAD TEXT DATA

```
class TextDataset(data.Dataset):
    def __init__(self, data_dir, split='train',
                 base_size=64,
                 transform=None, target_transform=None):
        self.transform = transform
        self.norm = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
        self.target_transform = target_transform
        self.embeddings_num = cfg.TEXT.CAPTIONS_PER_IMAGE

        self.imsize = []
        for i in range(cfg.TREE.BRANCH_NUM):
            self.imsize.append(base_size)
            base_size = base_size * 2

        self.data = []
        self.data_dir = data_dir
        if data_dir.find('birds') != -1:
            self.bbox = self.load_bbox()
        else:
            self.bbox = None
        split_dir = os.path.join(data_dir, split)

        self.filenames, self.captions, self.ixtoword, \
            self.wordtoix, self.n_words = self.load_text_data(data_dir, split)

        self.class_id = self.load_class_id(split_dir, len(self.filenames))
        self.number_example = len(self.filenames)

    def load_bbox(self):#building boxes for image
        data_dir = self.data_dir
        bbox_path = os.path.join(data_dir, 'CUB_200_2011/bounding_boxes.txt')
        df_bounding_boxes = pd.read_csv(bbox_path,
                                        delim_whitespace=True
```

```
        filenames = df_filenames[1].tolist()
        print('Total filenames: ', len(filenames), filenames[0])
        #
        filename_bbox = {img_file[:-4]: [] for img_file in filenames}
        numImgs = len(filenames)
        for i in range(0, numImgs):
            # bbox = [x-left, y-top, width, height]
            bbox = df_bounding_boxes.iloc[i][1:].tolist()

            key = filenames[i][:-4]
            filename_bbox[key] = bbox
        #
        return filename_bbox

    def load_captions(self, data_dir, filenames):#text
        all_captions = []
        for i in range(len(filenames)):
            cap_path = '%s/text/%s.txt' % (data_dir, filenames[i])
            with open(cap_path, "r") as f:
                captions = f.read().decode('utf8').split('\n')
                cnt = 0
                for cap in captions:
                    if len(cap) == 0:
                        continue
                    cap = cap.replace("\ufffd\ufffd", " ")
                    # picks out sequences of alphanumeric characters as tokens
                    # and drops everything else
                    tokenizer = RegexpTokenizer(r'\w+')
                    tokens = tokenizer.tokenize(cap.lower())
                    # print('tokens', tokens)
                    if len(tokens) == 0:
                        print('cap', cap)
                        continue

                    tokens_new = []
                    for t in tokens:
                        t = t.encode('ascii', 'ignore').decode('ascii')
                        if len(t) > 0:
                            tokens_new.append(t)
```

+ Code + Text

```
                    all_captions.append(tokens_new)
                    cnt += 1
                    if cnt == self.embeddings_num:
                        break
                if cnt < self.embeddings_num:
                    print('ERROR: the captions for %s less than %d'
                          % (filenames[i], cnt))
        return all_captions

    def build_dictionary(self, train_captions, test_captions):#dictionary for given text
        word_counts = defaultdict(float)
        captions = train_captions + test_captions
        for sent in captions:
            for word in sent:
                word_counts[word] += 1

        vocab = [w for w in word_counts if word_counts[w] >= 0]

        ixtoword = {}
        ixtoword[0] = '<end>'
        wordtoix = {}
        wordtoix['<end>'] = 0
        ix = 1
        for w in vocab:
            wordtoix[w] = ix
            ixtoword[ix] = w
            ix += 1

        train_captions_new = []
        for t in train_captions:
            rev = []
            for w in t:
                if w in wordtoix:
                    rev.append(wordtoix[w])
            # rev.append(0)  # do not need '<end>' token
            train_captions_new.append(rev)

        test_captions_new = []
        for t in test_captions:
            rev = []
            for w in t:
```

```
            for w in t:
                if w in wordtoix:
                    rev.append(wordtoix[w])
            # rev.append(0)  # do not need '<end>' token
            test_captions_new.append(rev)

        return [train_captions_new, test_captions_new,
                ixtoword, wordtoix, len(ixtoword)]

    def load_text_data(self, data_dir, split):#loading text data from given train data and test data
        filepath = os.path.join(data_dir, 'captions.pickle')
        train_names = self.load_filenames(data_dir, 'train')
        test_names = self.load_filenames(data_dir, 'test')
        if not os.path.isfile(filepath):
            train_captions = self.load_captions(data_dir, train_names)
            test_captions = self.load_captions(data_dir, test_names)

            train_captions, test_captions, ixtoword, wordtoix, n_words = \
                self.build_dictionary(train_captions, test_captions)
            with open(filepath, 'wb') as f:
                pickle.dump([train_captions, test_captions,
                             ixtoword, wordtoix], f, protocol=2)
                print('Save to: ', filepath)
        else:
            with open(filepath, 'rb') as f:
                x = pickle.load(f)
                train_captions, test_captions = x[0], x[1]
                ixtoword, wordtoix = x[2], x[3]
                del x
                n_words = len(ixtoword)
                print('Load from: ', filepath)
        if split == 'train':
            # a list of list: each list contains
            # the indices of words in a sentence
            captions = train_captions
            filenames = train_names
        else:  # split=='test'
            captions = test_captions
            filenames = test_names
        return filenames, captions, ixtoword, wordtoix, n_words
```

```
        return filenames, captions, ixtoword, wordtoix, n_words

    def load_class_id(self, data_dir, total_num):
        if os.path.isfile(data_dir + '/class_info.pickle'):
            with open(data_dir + '/class_info.pickle', 'rb') as f:
                class_id = pickle.load(f, encoding="bytes")
        else:
            class_id = np.arange(total_num)
        return class_id

    def load_filenames(self, data_dir, split):
        filepath = '%s/%s/filenames.pickle' % (data_dir, split)
        if os.path.isfile(filepath):
            with open(filepath, 'rb') as f:
                filenames = pickle.load(f)
            print('Load filenames from: %s (%d)' % (filepath, len(filenames)))
        else:
            filenames = []
        return filenames

    def get_caption(self, sent_ix):
        # a list of indices for a sentence
        sent_caption = np.asarray(self.captions[sent_ix]).astype('int64')
        if (sent_caption == 0).sum() > 0:
            print('ERROR: do not need END (0) token', sent_caption)
        num_words = len(sent_caption)
        # pad with 0s (i.e., '<end>')
        x = np.zeros((cfg.TEXT.WORDS_NUM, 1), dtype='int64')
        x_len = num_words
        if num_words <= cfg.TEXT.WORDS_NUM:
            x[:num_words, 0] = sent_caption
        else:
            ix = list(np.arange(num_words))  # 1, 2, 3,..., maxNum
            np.random.shuffle(ix)
            ix = ix[:cfg.TEXT.WORDS_NUM]
            ix = np.sort(ix)
            x[:, 0] = sent_caption[ix]
            x_len = cfg.TEXT.WORDS_NUM
        return x, x_len
```

```
        return x, x_len

    def __getitem__(self, index):
        #
        key = self.filenames[index]
        cls_id = self.class_id[index]
        #
        if self.bbox is not None:
            bbox = self.bbox[key]
            data_dir = '%s/CUB_200_2011' % self.data_dir
        else:
            bbox = None
            data_dir = self.data_dir
        #
        img_name = '%s/images/%s.jpg' % (data_dir, key)
        imgs = get_imgs(img_name, self.imsize,
                        bbox, self.transform, normalize=self.norm)
        # random select a sentence
        sent_ix = random.randint(0, self.embeddings_num)
        new_sent_ix = index * self.embeddings_num + sent_ix
        caps, cap_len = self.get_caption(new_sent_ix)
        return imgs, caps, cap_len, cls_id, key


    def __len__(self):
        return len(self.filenames)
```

TRAINING

```
def train(dataloader,netG,netD,text_encoder,optimizerG,optimizerD,state_epoch,batch_size,device):
    f=open("/content/drive/MyDrive/DF-GAN/code/epoch.txt",'r')
    epochcount=0
    for i in f:
        epochcount=i
    f.close()
    netgpath="/content/drive/MyDrive/DF-GAN/models/bird/netG.pth"
    netdpath="/content/drive/MyDrive/DF-GAN/models/bird/netD.pth"
    netG.load_state_dict(torch.load(netgpath))
    netD.load_state_dict(torch.load(netdpath))
    for epoch in range(int(epochcount)+1, cfg.TRAIN.MAX_EPOCH+1):
        for step, data in enumerate(dataloader, 0):

            imags, captions, cap_lens, class_ids, keys = prepare_data(data)
            hidden = text_encoder.init_hidden(batch_size)
            # words_embs: batch_size x nef x seq_len
            # sent_emb: batch_size x nef
            words_embs, sent_emb = text_encoder(captions, cap_lens, hidden)
            words_embs, sent_emb = words_embs.detach(), sent_emb.detach()

            imgs=imags[0].to(device)
            real_features = netD(imgs)
            output = netD.COND_DNET(real_features,sent_emb)
            errD_real = torch.nn.ReLU()(1.0 - output).mean()

            output = netD.COND_DNET(real_features[:(batch_size - 1)], sent_emb[1:batch_size])
            errD_mismatch = torch.nn.ReLU()(1.0 + output).mean()

            # synthesize fake images
            noise = torch.randn(batch_size, 100)
            noise=noise.to(device)
            fake = netG(noise,sent_emb)

            # G does not need update with D
            fake_features = netD(fake.detach())

            errD_fake = netD.COND_DNET(fake_features,sent_emb)
            errD_fake = torch.nn.ReLU()(1.0 + errD_fake).mean()

            errD = errD_real + (errD_fake + errD_mismatch)/2.0
            optimizerD.zero_grad()
            optimizerG.zero_grad()
            errD.backward()
            optimizerD.step()
```

```
            #MA-GP
            interpolated = (imgs.data).requires_grad_()
            sent_inter = (sent_emb.data).requires_grad_()
            features = netD(interpolated)
            out = netD.COND_DNET(features,sent_inter)
            grads = torch.autograd.grad(outputs=out,inputs=(interpolated,sent_inter),
                                        grad_outputs=torch.ones(out.size()).cuda(),
                                      retain_graph=True,
                                    create_graph=True,
                                    only_inputs=True)
            grad0 = grads[0].view(grads[0].size(0), -1)
            grad1 = grads[1].view(grads[1].size(0), -1)
            grad = torch.cat((grad0,grad1),dim=1)
            grad_l2norm = torch.sqrt(torch.sum(grad ** 2, dim=1))
            d_loss_gp = torch.mean((grad_l2norm) ** 6)
            d_loss = 2.0 * d_loss_gp
            optimizerD.zero_grad()
            optimizerG.zero_grad()
            d_loss.backward()
            optimizerD.step()

            # update G
            features = netD(fake)
            output = netD.COND_DNET(features,sent_emb)
            errG = - output.mean()
            optimizerG.zero_grad()
            optimizerD.zero_grad()
            errG.backward()
            optimizerG.step()

            print('[%d/%d][%d/%d] Loss_D: %.3f Loss_G %.3f'
                % (epoch, cfg.TRAIN.MAX_EPOCH, step, len(dataloader), errD.item(), errG.item()))
        if(epoch%10==0):
            vutils.save_image(fake.data,
                        '%s/fake_samples_epoch_%03d.png' % ('/content/drive/MyDrive/DF-GAN/code/miscc/imgs/', epoch),
                        normalize=True)

        if epoch%1==0:
            os.remove('/content/drive/MyDrive/DF-GAN/models/bird/netG.pth')
            os.remove('/content/drive/MyDrive/DF-GAN/models/bird/netD.pth')
            torch.save(netG.state_dict(), '/content/drive/MyDrive/DF-GAN/models/%s/netG.pth' % (cfg.CONFIG_NAME))
            torch.save(netD.state_dict(), '/content/drive/MyDrive/DF-GAN/models/%s/netD.pth' % (cfg.CONFIG_NAME))
            f=open("/content/drive/MyDrive/DF-GAN/code/epoch.txt",'w')
            f.write(str(epoch))
            f.close()
```

TESTING

```
[ ] def sampling(text_encoder, netG, dataloader,device):
        print("..............starting..................")
        model_dir = cfg.TRAIN.NET_G
        split_dir = 'valid'
        # Build and load the generator
        netG.load_state_dict(torch.load('/content/drive/MyDrive/DF-GAN/models/%s/netG_600.pth'%(cfg.CONFIG_NAME)))
        netG.eval()

        batch_size = cfg.TRAIN.BATCH_SIZE
        s_tmp = model_dir
        save_dir = '%s/%s' % (s_tmp, split_dir)
        mkdir_p(save_dir)
        cnt = 0
        for i in range(1):  # (cfg.TEXT.CAPTIONS_PER_IMAGE):
            for step, data in enumerate(dataloader, 0):
                imags, captions, cap_lens, class_ids, keys = prepare_data(data)
                cnt += batch_size
                if step % 100 == 0:
                    print('step: ', step)
                # if step > 50:
                #     break
                hidden = text_encoder.init_hidden(batch_size)
                # words_embs: batch_size x nef x seq_len
                # sent_emb: batch_size x nef
                words_embs, sent_emb = text_encoder(captions, cap_lens, hidden)
                words_embs, sent_emb = words_embs.detach(), sent_emb.detach()
                #######################################################
                # (2) Generate fake images
                ######################################################
                with torch.no_grad():
                    noise = torch.randn(batch_size, 100)
                    noise=noise.to(device)
                    fake_imgs = netG(noise,sent_emb)
                for j in range(batch_size):
                    s_tmp = '%s/single/%s' % (save_dir, keys[j])
                    folder = s_tmp[:s_tmp.rfind('/')]
                    if not os.path.isdir(folder):
                        print('Make a new folder: ', folder)
                        mkdir_p(folder)
                    im = fake_imgs[j].data.cpu().numpy()
                    # [-1, 1] --> [0, 255]
                    im = (im + 1.0) * 127.5
                    im = im.astype(np.uint8)
                    im = np.transpose(im, (1, 2, 0))
```

MAIN SECTION

```
imsize = cfg.TREE.BASE_SIZE
    batch_size = cfg.TRAIN.BATCH_SIZE
    image_transform = transforms.Compose([
        transforms.Resize(int(imsize * 76 / 64)),
        transforms.RandomCrop(imsize),
        transforms.RandomHorizontalFlip()])
    if cfg.B_VALIDATION:
        dataset = TextDataset(cfg.DATA_DIR, 'test',
                                base_size=cfg.TREE.BASE_SIZE,
                                transform=image_transform)
        print(dataset.n_words, dataset.embeddings_num)
        assert dataset
        dataloader = torch.utils.data.DataLoader(
            dataset, batch_size=batch_size, drop_last=True,
            shuffle=True, num_workers=int(cfg.WORKERS))
    else:
        dataset = TextDataset(cfg.DATA_DIR, 'train',
                            base_size=cfg.TREE.BASE_SIZE,
                            transform=image_transform)
        print(dataset.n_words, dataset.embeddings_num)
        assert dataset
        dataloader = torch.utils.data.DataLoader(
            dataset, batch_size=batch_size, drop_last=True,
            shuffle=True, num_workers=int(cfg.WORKERS))

    # # validation data #

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    netG = NetG(cfg.TRAIN.NF, 100).to(device)
    netD = NetD(cfg.TRAIN.NF).to(device)

    text_encoder = RNN_ENCODER(dataset.n_words, nhidden=cfg.TEXT.EMBEDDING_DIM)
    state_dict = torch.load(cfg.TEXT.DAMSM_NAME, map_location=lambda storage, loc: storage)
    text_encoder.load_state_dict(state_dict)
    text_encoder.cuda()
```

```python
# # validation data #

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

netG = NetG(cfg.TRAIN.NF, 100).to(device)
netD = NetD(cfg.TRAIN.NF).to(device)

text_encoder = RNN_ENCODER(dataset.n_words, nhidden=cfg.TEXT.EMBEDDING_DIM)
state_dict = torch.load(cfg.TEXT.DAMSM_NAME, map_location=lambda storage, loc: storage)
text_encoder.load_state_dict(state_dict)
text_encoder.cuda()

for p in text_encoder.parameters():
    p.requires_grad = False
text_encoder.eval()

state_epoch=0

optimizerG = torch.optim.Adam(netG.parameters(), lr=0.0001, betas=(0.0, 0.9))
optimizerD = torch.optim.Adam(netD.parameters(), lr=0.0004, betas=(0.0, 0.9))


if cfg.B_VALIDATION:
    count = sampling(text_encoder, netG, dataloader,device)  # generate images for the whole valid dataset
    print('state_epoch:  %d'%(state_epoch))
else:
    count = train(dataloader,netG,netD,text_encoder,optimizerG,optimizerD, state_epoch,batch_size,device)
```

CHAPTER-6

RESULTS

INPUT TEXT

the large brown bird has a big bill and white throat
this medium sized bird is primarily black and has a large wingspan and a long black bill with a strip of white at the beginning of it.
this bird has crown, a black bill, and a large wingspan
this bird features a broad wingspan and a slightly curved, dark bill.
this larger bird is black and has a large black beak
this bird is mostly black with white around the base of the large curved bill.
this is a mostly black and grey bird with a spectrum of white and grey secondaries and wing bars.
this bird is all black and has a long, pointy beak.
a medium sized bird with a long bill and brown wings
this bird is black with white and has a long, pointy beak.

GROUND TRUTH

OUTPUT(GENERATED IMAGE)

DF-GAN Text to Image Synthesis Result-1

INPUT TEXT

the bird has a blue crown and a small blue bill.
this is a blue bird with black wings and a white beak.
a medium sized bird with the same color blue covering entire body
this beautiful blue bird has a narrow beak and a long tail.
a larger blue bird with a proportionate black and white beak.
this bird has wings that are blue and has a small bill
this small bird is covered in all blue feathers, the top part of its bill and feet are dark blue colored, and the bottom part of its bill a light blue.
this particular bird has a belly that is blue with black streaks on the secondaries
this bird is blue all over, with black legs, and a black pointed bill.
this is a small bright blue bird with a large breast and belly, with a bill that is black on top and white on the bottom.

GROUND TRUTH

OUTPUT

DF-GAN Text to Image Synthesis Result-2

INPUT TEXT

a bird with a white breast, short yellow bill and gray feet.
this bird has a white belly and breast with a gray crown and short pointy bill.
this bird is white and black in color with a orange curved beak and black eye rings.
a bird with a white colored stomach, chest, and chin, and a blackish brown upper body and wings
this is a white bird with a grey crown and orange bill.
this bird has wings that are black and has a yellow bill
this bird has an off white belly and breast, a gray crown, and an orange bill.
this medium-sized bird has light colored belly and dark grey colored wings.
this bird has wings that are black and has a white belly
this bird has a white breast, with a long orange bill.

GROUND TRUTH

OUTPUT

DF-GAN Text to Image Synthesis Result-3

CHAPTER-7

CONCLUSION

We propose a novel Deep Fusion Generative Adversarial Networks (DF-GAN) for text-to-image generation tasks. The proposed model is able to directly synthesize more realistic and text-image semantic consistent images without stacking architecture and extra networks. Moreover, we propose a novel Matching-Aware zero-centered Gradient Penalty to ensure the text-image semantic consistency and promote the generator convergence to real data distribution. Besides, we decompose the effectiveness of Affine Transformation from CBN and present a Deep text-image Fusion Block to fuse text and image features more effectively. In addition, we propose the one-way discriminator which stabilizes the training process and accelerates the convergence of the generator network. Extensive experiment results show that our proposed DF-GAN significantly outperforms state-of-the-art models on the CUB dataset.

CHAPTER-8

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in Neural Information Processing Systems, 2014, pp. 2672– 2680. 1, 3, 4

[2] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” in Proceedings of the International Conference on Machine Learning, 2016, pp. 1060–1069. 1, 2, 3

[3] S. E. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee, “Learning what and where to draw,” in Advances in neural information processing systems, 2016, pp. 217–225. 1

[4] M. Yuan and Y. Peng, “Ckd: Cross-task knowledge distillation for text-to-image synthesis,” IEEE Transactions on Multimedia, 2019. 1

[5] S. Hong, D. Yang, J. Choi, and H. Lee, “Inferring semantic layout for hierarchical text-to-image synthesis,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7986–7994. 1

[6] R. Li, N. Wang, F. Feng, G. Zhang, and X. Wang, “Exploring global and local linguistic representation for text-to-image synthesis,” IEEE Transactions on Multimedia, 2020. 1

[7] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, “Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks,” in Proceedings of the IEEE international conference on computer vision, 2017, pp. 5907–5915. 1, 2, 3, 5

[8] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, “Stackgan++: Realistic image synthesis with stacked generative adversarial networks,” IEEE TPAMI, vol. 41, no. 8, pp. 1947–1962, 2018. 1, 2, 3, 5, 8, 9

[9] Z. Zhang, Y. Xie, and L. Yang, “Photographic text-to-image synthesis with a hierarchically-nested adversarial network,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6199–6208. 1

[10] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, "Attngan: Fine-grained text to image generation with attentional generative adversarial networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 1316–1324. 1, 2, 3, 4, 5, 7, 8, 9, 10

[11] B. Li, X. Qi, T. Lukasiewicz, and P. Torr, "Controllable text-to-image generation," in Advances in Neural Information Processing Systems, 2019, pp. 2065–2075. 1

[12] T. Qiao, J. Zhang, D. Xu, and D. Tao, "Learn, imagine and create: Text-to-image generation from prior knowledge," in Advances in Neural Information Processing Systems, 2019, pp. 887–897. 1

[13] T. Qiao, J. Zhang, D. Xu, and D. Tao, "Mirrorgan: Learning text to-image generation by redescription," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 1505–1514. 1, 2, 3, 8

[14] M. Zhu, P. Pan, W. Chen, and Y. Yang, "Dm-gan: Dynamic memory generative adversarial networks for text-to-image synthesis," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 5802–5810. 1, 2, 3, 7, 8, 9, 10

[15] G. Yin, B. Liu, L. Sheng, N. Yu, X. Wang, and J. Shao, "Semantics disentangling for text-to-image generation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2327–2336. 1, 2, 3, 6, 8

[16] W. Li, P. Zhang, L. Zhang, Q. Huang, X. He, S. Lyu, and J. Gao, "Object Driven text-to-image synthesis via adversarial training," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 12 174–12 182. 1, 3, 8

[17] Y. Gou, Q. Wu, M. Li, B. Gong, and M. Han, "Segattngan: Text to image generation with segmentation attention," arXiv preprint arXiv:2005.12444, 2020. 1

[18] J. Cheng, F. Wu, Y. Tian, L. Wang, and D. Tao, "Rifegan: Rich feature generation for text-to-image synthesis from prior knowledge," in Proceedings of

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 10 911–10 920. 1

[19] L. Chen, H. Zhang, J. Xiao, L. Nie, J. Shao, W. Liu, and T.-S. Chua, "Sca-CNN: Spatial and channel-wise attention in convolutional networks for image captioning," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5659–5667. 2

[20] D. Yu, J. Fu, T. Mei, and Y. Rui, "Multi-level attention networks for visual question answering," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4709–4717. 2

[21] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville, "Modulating early visual processing by language," in Advances in Neural Information Processing Systems, 2017, pp. 6594– 6604. 2, 6, 7

[22] T. Miyato and M. Koyama, "cgans with projection discriminator," arXiv preprint arXiv:1802.05637, 2018. 2, 6, 7

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778. 2

[24] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in International Conference on Machine Learning, 2019, pp. 7354–7363. 2, 3, 4

[25] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011. 2, 7