A Project Report on

TEXT TO IMAGE SYNTHESIS USING STACKGANs

submitted in partial fulfilment for the award of

Bachelor of Technology

in

Computer Science and Engineering

By

B. Ajay Kumar Reddy(Y21ACS422) **G. Narendra (L22ACS598)**

G. Dhyanendra (Y21ACS455) **G. George (Y21ACS449)**

Under the guidance of

Dr. K. ManiDeep, M.Tech., Ph.D.

Associate Professor

Department of Computer Science and Engineering

Bapatla Engineering College

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

BAPATLA – 522 102, Andhra Pradesh, INDIA

2024-2025

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the project report entitled **Text To Image Synthesis Using StackGans** that is being submitted by B. Ajay Kumar Reddy (Y21ACS422), G. Narendra (L22ACS598), G. Dhyanendra (Y21ACS455), G. George sterret (Y21ACS449) in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

Date:

Signature of the Guide	**Signature of the HOD**
Dr. K. Manideep	**Dr. M. Rajesh Babu**
Assoc. Professor	**Assoc. Prof. & Head**

DECLARATION

We declare that this project work is composed by ourselves, that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

B. Ajay Kumar Reddy (Y21ACS422)

G. Narendra (L22ACS598)

G. Dhyanendra (Y21ACS455)

G. George Sterret (Y21ACS449)

Acknowledgement

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide **Dr. K. Manideep, Associate Professor,** Department of CSE, for his valuable and inspiring guidance, comments, suggestions and encouragement.

We extend our sincere thanks to **Dr. M. Rajesh Babu**, Prof. & Head of the Dept. for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal **Dr. N. Rama Devi** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to our project coordinator **Dr. P. PardhaSaradhi,** Prof. Dept. of CSE for his helpful suggestions in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

B. Ajay Kumar Reddy (Y21ACS422)

G. Narendra (L22ACS598)

G. Dhyanendra (Y21ACS455)

G. George Sterret (Y21ACS449)

Table of Contents

List Of Figures

Abstract

The transformation of textual descriptions into visually meaningful images remains a complex challenge in Generative AI. Although deep learning techniques have advanced feature extraction and image generation, the introduction of Generative Adversarial Networks (GANs), particularly Stacked Generative Adversarial Networks (StackGANs), has significantly improved the resolution and authenticity of generated images. Traditional approaches, such as Variational Autoencoders and Autoregressive models, often struggle to maintain consistency and fine-grained details, whereas StackGANs employ a two-stage adversarial training process to enhance both image clarity and semantic alignment.

This paper introduces a novel architecture that utilizes StackGANs to effectively convert descriptive text inputs into high-quality images. By improving text embedding techniques and optimizing image generation at multiple scales, the proposed model captures structural relationships and intricate visual features, leading to more realistic and coherent outputs. Unlike conventional methods, the two-stage refinement in StackGANs progressively enhances image resolution and fidelity. The model's effectiveness is demonstrated through the generation of detailed images that closely match their textual descriptions. Experimental evaluations confirm that the approach significantly improves both visual accuracy and semantic consistency in text-to-image synthesis.

1 Introduction

Effective Text-to-image synthesis is a challenging task that involves generating realistic images from natural language descriptions. It bridges natural language processing and computer vision domains. Traditional GANs struggle to generate high-resolution, detailed images directly from text. StackGAN addresses this by using a two-stage architecture: Stage-I creates a coarse image, and Stage-II refines it with more details. This stacked approach improves both image quality and semantic relevance. In this project, we implement StackGAN on datasets like Oxford-102 Flowers. Our goal is to explore how well StackGAN captures and visualizes textual descriptions.

1.1 Problem Statement

Generating high-quality and semantically accurate images from textual descriptions is a complex task that involves understanding natural language and translating it into visual content. Traditional image generation methods struggle to capture fine-grained details and often fail to produce images that match the input text precisely. The challenge lies in bridging the gap between the discrete nature of text and the continuous nature of images.

1.2 Objective

The primary objective of this project is to generate realistic and high-resolution images from natural language text descriptions using the StackGAN (Stacked Generative Adversarial Network) architecture. The project aims to implement both stages of StackGAN, where the first stage generates a coarse image from the text, and the second stage refines it to add more realistic details. This involves working with datasets such Oxford-102 Flowers, which contain annotated image-text pairs. The project also focuses on extracting meaningful text embeddings to guide the image generation process effectively. By training and evaluating the model, the goal is to produce images that accurately reflect the semantics of the given descriptions and demonstrate the potential of StackGAN in bridging the gap between textual and visual data.

1.3 Overview Of Related Work

This project explores the task of text-to-image synthesis, where the goal is to generate photo-realistic images based on natural language descriptions. To achieve this, we implement the StackGAN architecture, which uses a two-stage approach. In the first stage, a low-resolution image is generated from the text embedding that captures the basic shape and color of the object. In the second stage, the model refines this image by adding finer details and improving the overall resolution, resulting in a more realistic and semantically meaningful image. This stacked structure overcomes the limitations of traditional GANs that often struggle with generating high-resolution images directly from text.

The project uses publicly available datasets Oxford-102 Flowers, which provide a collection of images along with textual annotations. These datasets help in training the model to learn the relationship between descriptive text and corresponding visual features. The implementation includes steps such as text preprocessing, embedding generation, model training, and performance evaluation using visual results and quality metrics. Through this project, we demonstrate how combining deep learning, computer vision, and natural language processing can enable machines to understand and visualize human language in the form of images.

1.4 Deep Learning

Deep Learning is a specialized subset of machine learning that focuses on neural networks with multiple layers, allowing models to learn complex and abstract patterns from large amounts of data. Unlike traditional machine learning, where feature extraction is often manually defined, deep learning models automatically extract high-level features from raw input, such as images, text, or audio. These models are particularly powerful in fields like computer vision, natural language processing, speech recognition, and generative modeling. Deep learning architectures such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Generative Adversarial Networks (GANs) have revolutionized how machines perceive and generate

1.4.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing data with a grid-like topology, such as images. CNNs have become the foundation of most computer vision applications due to their ability to automatically and efficiently extract spatial features from images. Unlike traditional neural networks, CNNs use convolutional layers that apply filters (kernels) over the input image to detect low-level features like edges, corners, and textures, and progressively learn higher-level features such as shapes, objects, or patterns.

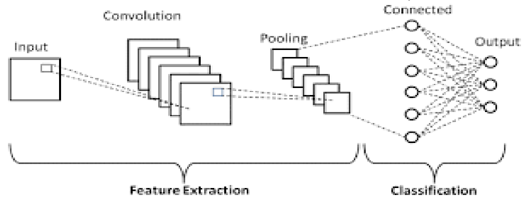


Figure 1.1 Convolution Neural Networks

1.4.2 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a specialized class of neural networks designed to handle sequential data, such as time series, text, or audio. Unlike traditional feedforward networks, RNNs maintain a hidden state that is passed from one time step to the next, enabling the model to retain information about previous inputs. This memory-like behavior allows RNNs to capture temporal dependencies in sequences, making them especially effective for tasks involving natural language processing and sequence prediction.

However, traditional RNNs often face challenges such as vanishing or exploding gradients, which limit their ability to model long-term dependencies. To overcome this, more advanced variants like Long Short-Term Memory (LSTM) networks or Gated Recurrent Units (GRUs) are often used, as they are capable of learning long-term dependencies more effectively.

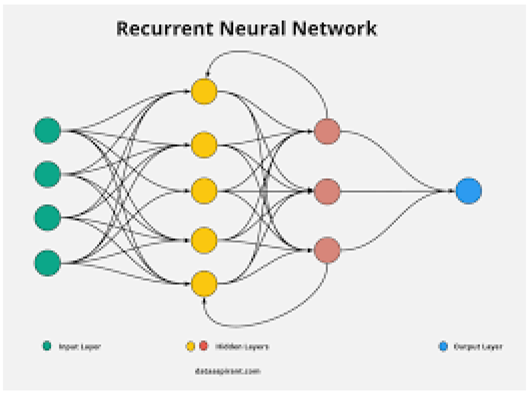


Figure 1.2 Recurrent Neural Networks

1.4.3 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a powerful class of deep learning models introduced by Ian Goodfellow in 2014. GANs are designed to generate new, synthetic data samples that resemble a given training dataset. What makes GANs unique is their architecture, which consists of two neural networks: a Generator and a

Discriminator, that are trained simultaneously in a competitive setting. The generator aims to create realistic data (such as images), while the discriminator attempts to distinguish between real and generated data.

The training process of a GAN is like a game between two players. The generator tries to fool the discriminator by producing images that are as realistic as possible, whereas the discriminator tries to correctly classify whether the input image is real (from the dataset) or fake (generated). Over time, both networks improve through this adversarial process. The generator learns to produce increasingly realistic images, and the discriminator becomes better at detecting fakes — until ideally, the generator produces outputs that the discriminator cannot distinguish from real data.

One of the major strengths of GANs is their ability to capture complex data distributions without needing explicit probability models. However, training GANs is also known to be challenging due to issues such as mode collapse, vanishing gradients, and instability in convergence. Despite these challenges, with advanced versions like StackGAN, StyleGAN, and CycleGAN, researchers have made significant progress in improving the training process and the quality of generated outputs.

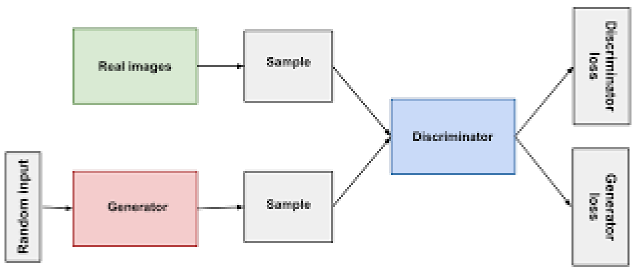


Figure 1.3 Generative Adversarial Networks

2 Literature Survey

Text-to-image synthesis is a significant research area in artificial intelligence that combines natural language processing and computer vision. The core idea is to enable machines to visualize descriptive text by generating corresponding images. Early work in this field relied on template-based or rule-based methods, which lacked flexibility and struggled to handle complex descriptions. With the advent of deep learning, particularly Generative Adversarial Networks (GANs), researchers began exploring data-driven approaches for more realistic and scalable image generation.

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow et al. in 2014, marked a turning point in generative modeling. GANs consist of two neural networks: a generator that creates images and a discriminator that evaluates them. They compete in a minimax game, pushing the generator to produce increasingly realistic images. While standard GANs were effective at generating images from random noise, they were not directly suitable for text-to-image tasks due to the lack of a mechanism to incorporate textual data.

To overcome this, Conditional GANs (cGANs) were introduced, which allowed the inclusion of auxiliary information—like class labels or text descriptions—during the generation process. In 2016, Reed et al. proposed one of the first methods for text-to-image generation using cGANs, where text features extracted from recurrent neural networks (RNNs) guided the image synthesis process. However, the generated images lacked fine details and were often blurry, especially when dealing with complex scenes.

To address these issues, StackGAN (Stacked Generative Adversarial Networks) was proposed by Zhang et al. in 2017. StackGAN introduced a two-stage architecture. Stage-I generates a low-resolution image that roughly reflects the input text, and Stage-II refines this output by adding more detail, producing high-resolution images. This decomposition helps the model focus on the global structure first and then enhance local details. The introduction of StackGAN significantly improved the visual quality and realism of generated images compared to earlier models.

The choice of text embeddings also plays a vital role in the quality of generated images. While early models used character-level or word-level embeddings generated by RNNs or CNN-RNN hybrids, later approaches adopted more powerful language models like BERT and CLIP for richer and more contextual embeddings. These embeddings help the generator better understand and reflect the fine-grained semantics of the textual description in the visual output.

Datasets used for benchmarking text-to-image synthesis models include Oxford-102 Flowers, and MS-COCO. These datasets contain images paired with descriptive captions, allowing models to learn text-image mappings. Among them, CUB and Oxford-102 are relatively simpler with fewer object categories and structured descriptions, while MS-COCO presents more challenging, complex scenes and diverse language, making it suitable for advanced models.

In summary, the development of StackGAN and its successors has significantly advanced the field of text-to-image synthesis. These models demonstrate that breaking down the image generation task into stages and incorporating attention or advanced embeddings can greatly improve both image quality and text-image alignment. The current project builds on these foundations, implementing StackGAN to explore and evaluate its effectiveness on dataset Oxford-102 Flowers.

3 Existing System

In traditional image generation systems, the goal of producing images from textual descriptions has been a significant challenge due to the complexity of mapping between language and pixel-level representations. Early methods were often limited in their ability to generate realistic, high-resolution images. With the evolution of deep learning, particularly Generative Adversarial Networks (GANs) and Recurrent Neural Networks (RNNs), significant progress has been made in text-to-image synthesis. Existing systems rely on models that encode textual information into meaningful embeddings and generate images conditioned on these embeddings.

These models have made progress in generating coarse visual features but often suffer from lack of detail, resolution, and semantic alignment. Limitations such as blurry outputs, poor color representation, and lack of fine-grained control over image details make these models less optimal for real-world applications. As a result, more robust models like StackGAN have been introduced to overcome these issues.

3.1 Algorithms in existing system

These various deep learning-based generative models have been developed for text-to-image synthesis. Below are some commonly used models in earlier systems.

i. Recurrent Neural Networks (RNN) with Fully Connected Decoders

ii. Conditional Generative Adversarial Networks (cGAN)

iii. Deep Convolutional GANs (DCGAN)

iv. AttnGAN (Attention Generative Adversarial Network)

3.1.1 RNN with Fully Connected Decoders

Early approaches used RNNs or LSTMs to process text input and generate features, which were then passed into fully connected neural networks to generate image pixels. These systems could only generate low-resolution images and often struggled with preserving semantic details from complex sentences. Additionally, training such systems was computationally expensive and unstable.

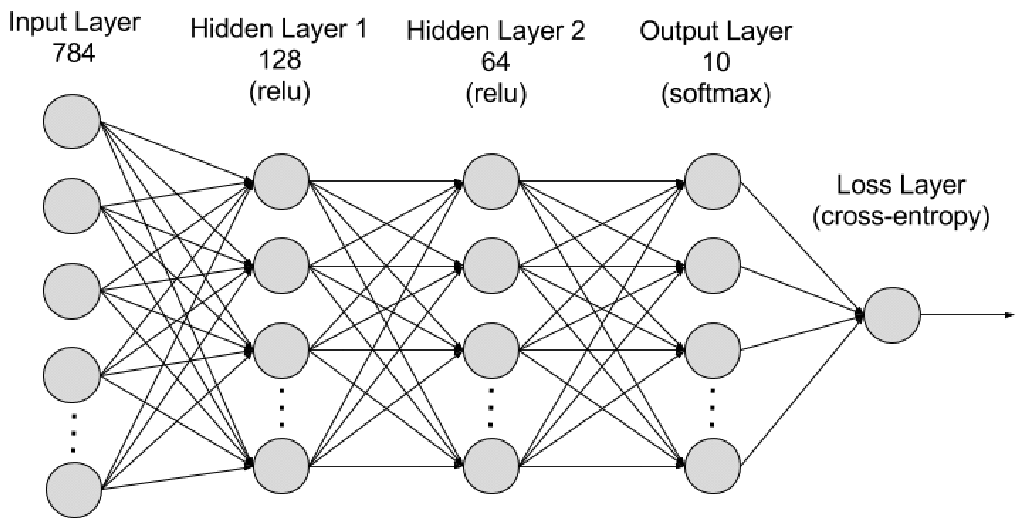


Figure 3.1 Rnn

3.1.2 Conditional GAN (cGAN)

Conditional GANs were a step forward, allowing both the generator and discriminator to be conditioned on text embeddings. These embeddings were obtained using pre-trained models like Skip-Thoughts or word2vec. While cGANs improved the conditioning aspect, they still produced images with artifacts and often failed to capture the fine-grained details described in the text.

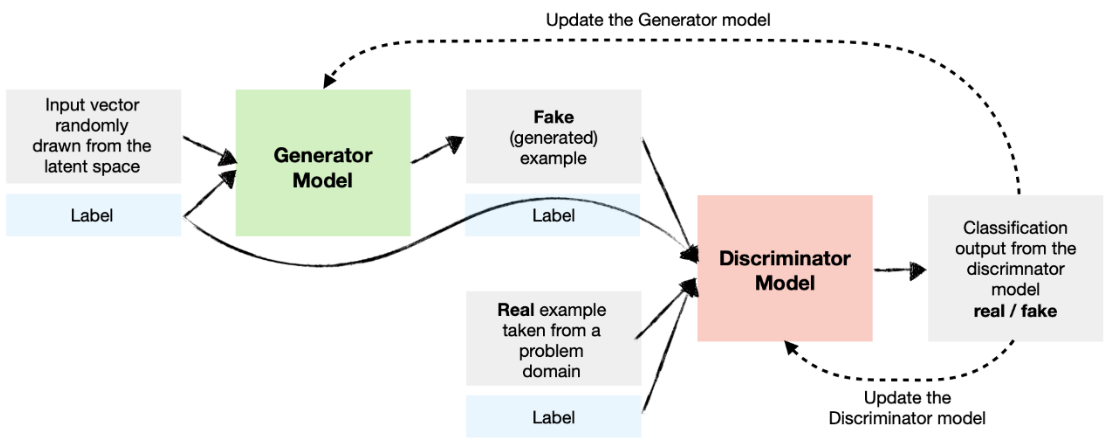

Update the Generator model
Input vector randomly drawn from the latent space
Label
Generator Model
Fake (generated) example
Label
Real example taken from a problem domain
Label
Discriminator Model
Classification output from the discrimnator model real / fake
Update the Discriminator model

Figure 3.2 Conditional GAN

3.1.3 Deep Convolutional GAN (DCGAN)

DCGANs introduced convolutional layers into the GAN architecture, allowing better image structure and sharper textures. They worked better than basic cGANs and vanilla GANs for visual quality. However, DCGANs alone still couldn't handle complex sentence-to-image mappings, especially for higher resolution generation. They also lacked mechanisms to maintain semantic consistency across different regions of the generated image.

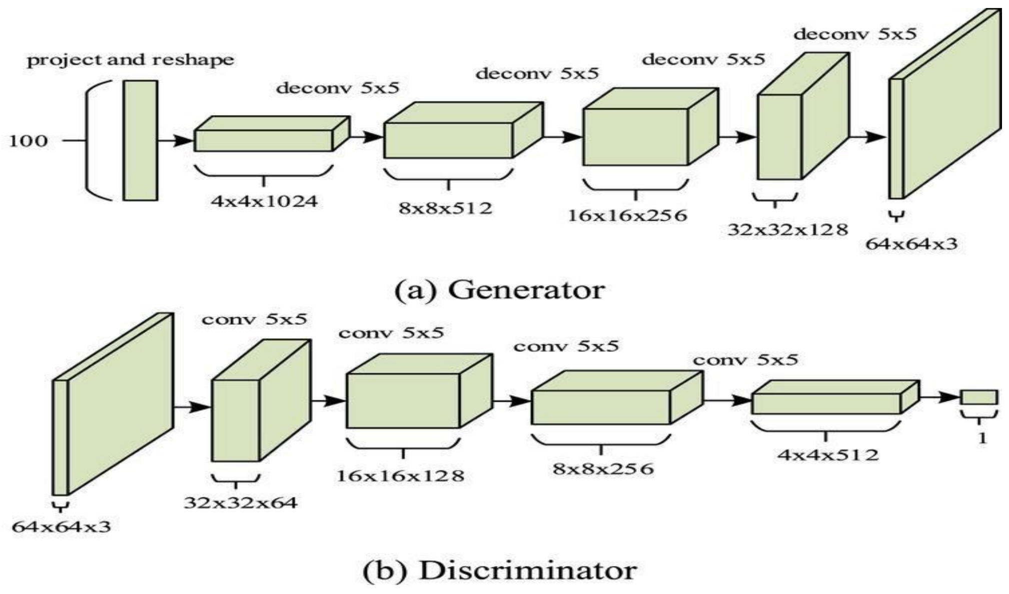


Figure 3.3 DC GAN

3.1.4 AttnGAN (Attention GAN)

AttnGAN introduced an attention mechanism that helped the generator focus on specific words in the text when generating corresponding image regions. This led to much better alignment between visual content and the descriptive text. However, the model was very complex, required a large amount of training data, and was computationally intensive. Despite generating high-quality results, training and interpretability were often challenging.

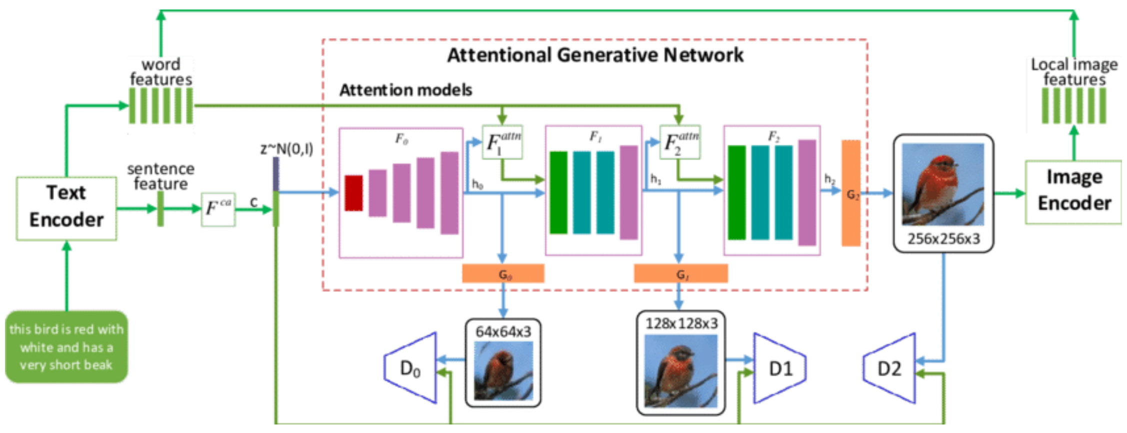


Figure 3.4 Attention GAN

The architecture of AttnGAN consists of a text encoder, multi-stage generators, and discriminators. The text encoder, typically a bidirectional RNN like a GRU, processes the sentence to generate both word-level and sentence-level embeddings. The Stage-I generator creates a basic layout of the image, while Stage-II (and any additional stages) refine it using the attention-driven conditioning. Discriminators at each stage evaluate both the realism of the image and its relevance to the text.

4 Proposed System

This chapter provides an overview of our model's operations, covering input details, data sources, and deep learning methodologies. Our objective is to present a concise understanding of our proposed deep learning approach for text-to-image synthesis, utilizing descriptive textual inputs to generate realistic images through the StackGAN architecture.

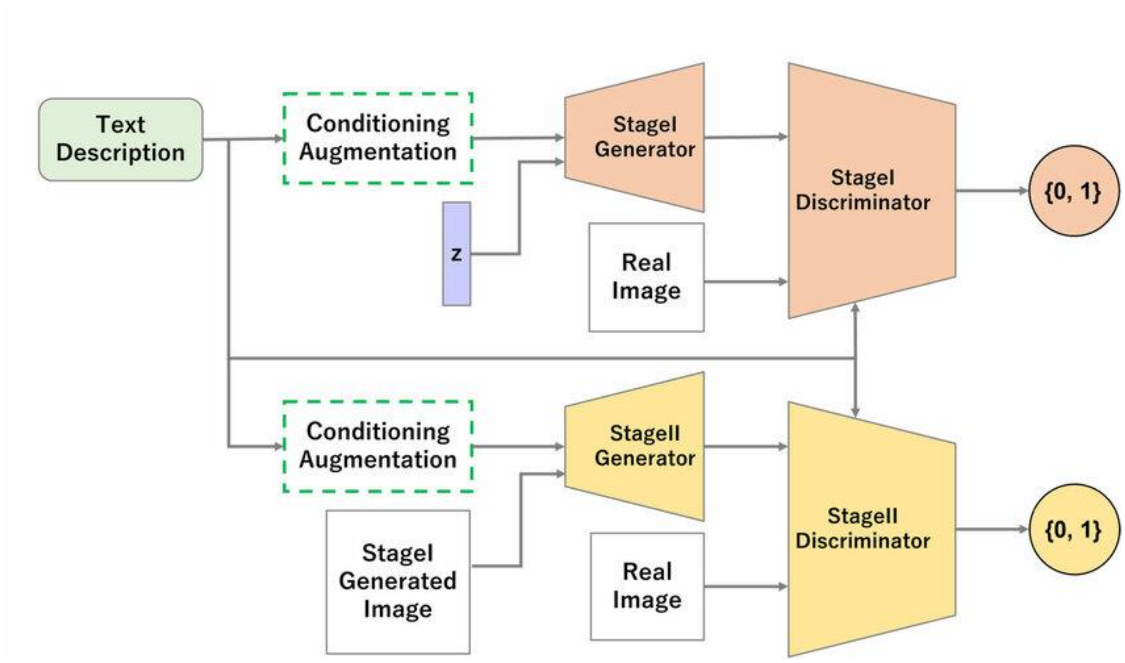


Figure 4.1 Flow Chart of Proposed System

4.1 Data Source

The Oxford 102 Category Flower Dataset is a high-quality image dataset commonly used in computer vision and deep learning research, particularly in image classification and generation tasks. It consists of 8,189 images of flowers from 102 different categories. Each category represents a specific flower species, making it suitable for fine-grained image generation and recognition tasks. The categories are diverse and carefully labelled to ensure detailed representation of real-world flower varieties.

This dataset is particularly suitable for text-to-image synthesis because it is also accompanied by textual descriptions for each image, provided by the Oxford VGG team. These descriptions can be used as input for training models like StackGAN, enabling the generation of flower images directly from text.

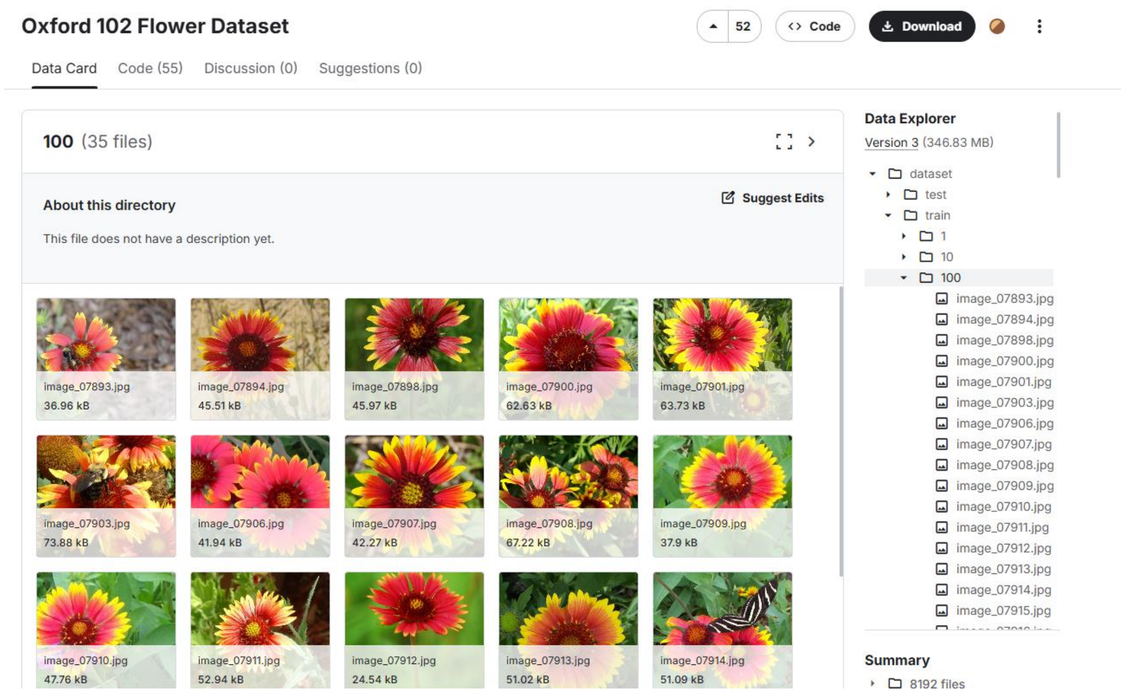


Figure 4.2 Data Set

4.2 Deep Learning Life Phases

In the context of text-to-image synthesis, deep learning models like StackGAN offer an innovative approach for generating realistic images based solely on textual descriptions. These models have wide-ranging applications in creative content generation, education, and accessibility tools. However, the effectiveness of such models relies heavily on the quality of both the image and textual data, careful model architecture design, and well-structured training-validation processes to ensure high-fidelity image generation aligned with the input text.

In our proposed system, the dataset is sourced from the Oxford 102 Flower Categories dataset, which contains paired flower images and their corresponding textual descriptions. The text descriptions are preprocessed and converted into numerical embeddings using a text encoder (e.g., RNN or transformer-based encoder). Simultaneously, the images undergo preprocessing such as resizing and normalization. These processed inputs are fed into the two-stage StackGAN model.

Stage-I generates a low-resolution image conditioned on the text embedding, and Stage-II refines this output to generate a high-resolution and more detailed version.

The key lifecycle phases of this model—data acquisition, preprocessing, model training and image generation—are represented in Figure 4.3 below.

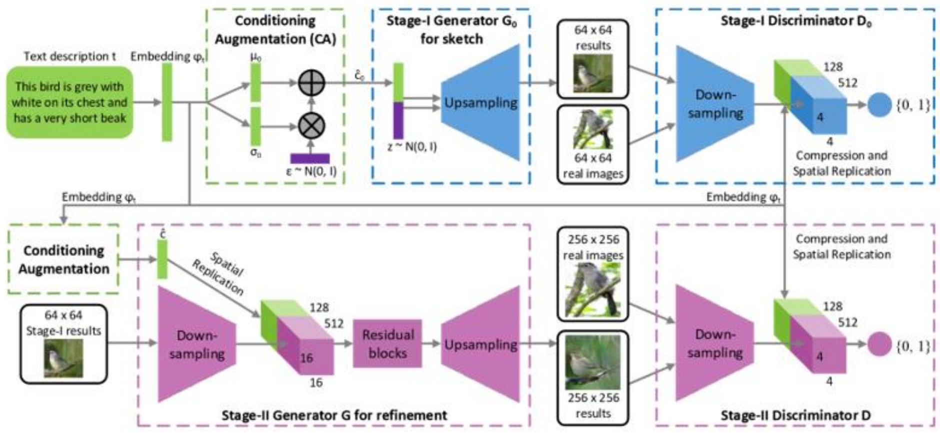


Figure 4.3 Life Phases Of Model

5 System Design

System design encompasses the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. This chapter delves into the intricacies of designing robust, scalable, and efficient systems.

5.1 Architecture

In the realm of system design, the architecture plays a pivotal role in shaping the effectiveness, scalability, and efficiency of a system. Within the context of machine learning applications, particularly those leveraging datasets sourced from platforms like Kaggle.

5.1.1 Component Diagram

The component diagram depicts essential system components and their interactions. It showcases a Data Preprocessing Component for cleaning and normalizing Kaggle data, a Data Splitting Component for partitioning data into training and validation sets, and a Machine Learning Model Component for training models. An Integration Component enables interaction with the Kaggle data source. Data flows between these components, guided by control flow sequences, while interfaces ensure seamless communication and feedback loops enable iterative refinement.

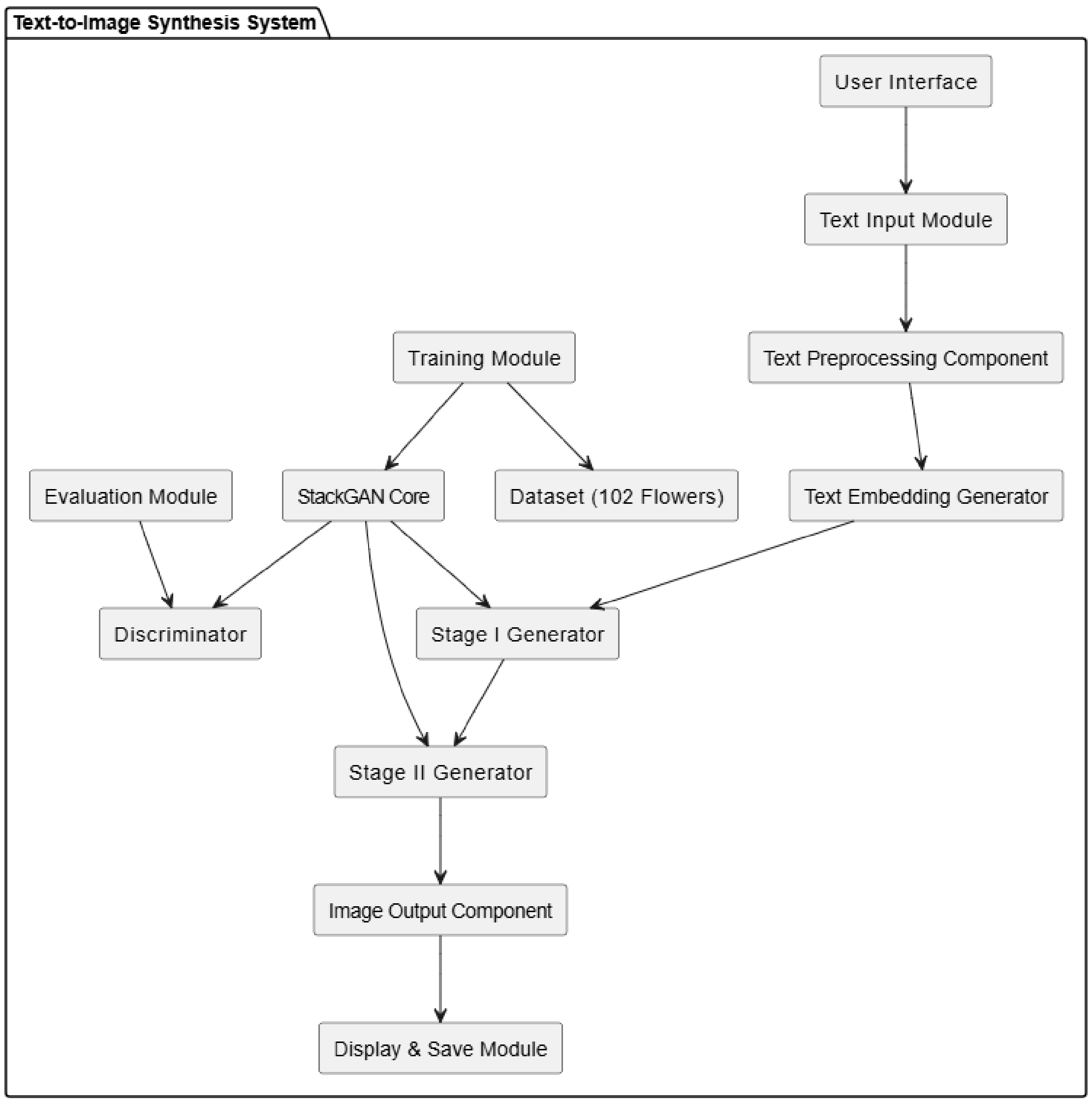


Figure 5.1 Components of Model

5.2 Input

The input to the system is a textual description of a flower. These descriptions are simple yet detailed sentences that capture the visual features of the flower such as colour, shape, petal structure, and size. Examples of input text might include

1. “This flower has bright yellow petals and a dark brown centre.”
2. “A red flower with rounded petals and green leaves around the stem.”

These descriptions are first tokenized and then transformed into vector embeddings using a text encoder. These embeddings serve as the conditional input for the StackGAN model.

5.3 Output

The output of the system is a **realistic image** generated based on the given textual description. The output goes through two stages:

1. **Stage-I Output:** A low-resolution (e.g., 64×64) image that gives a rough idea of the flower’s shape and colours.
2. **Stage-II Output:** A high-resolution (e.g., 256×256) image that includes finer details and better texture quality, making it look more realistic and aligned with the input description.

6 Libraries

This chapter serves as a comprehensive exploration of the foundational libraries essential for system development within the Python programming landscape. It delves into the critical role these libraries play in streamlining development workflows, enhancing code efficiency, and enabling the creation of robust and scalable software systems.

6.1 Libraries used in system

i. NumPy

ii. Pandas

iii. TensorFlow

iv. Seaborn

v. Pickle

vi. Matplotlib

vii. Flask

6.1.1 Numpy

NumPy is a Python library for numerical computing, offering support for large multidimensional arrays and matrices. It provides efficient array operations, mathematical functions, and integration with other scientific computing libraries, making it essential for data analysis, machine learning, and scientific research.

6.1.2 Pandas

Pandas is a python's data manipulation and analysis library, featuring the DataFrame structure for structured data handling, alongside tools for reading, writing, cleaning, transforming, and analyzing datasets, pivotal in data science, finance, and research.

6.1.3 TensorFlow

TensorFlow is an open-source deep learning framework developed by Google that is widely used for building and deploying machine learning and deep learning models. In our project, TensorFlow serves as the backbone for implementing and training the StackGAN architecture. It provides powerful tools for defining neural networks, handling data pipelines, and performing tensor computations efficiently.

Using TensorFlow, we construct both Stage-I and Stage-II GAN models, define loss functions, optimize model parameters, and manage GPU acceleration for faster training. Its flexibility and scalability make it ideal for generating high-quality images from textual descriptions in our text-to-image synthesis task.

6.1.4 Seaborn

Seaborn is a python's statistical visualization library, built on Matplotlib, simplifying the creation of attractive graphics for exploring relationships in datasets through functions like scatter plots and bar plots, with options for styling and statistical estimation.

6.1.5 Pickle

Pickle is a python's serialization library, allowing objects to be serialized into byte streams and saved to disk, and deserialized back into objects, facilitating data storage and transfer between Python programs.

6.1.6 Matplotlib

Matplotlib is a python's versatile plotting library, offering comprehensive tools for creating static, interactive, and publication-quality visualizations, widely used in scientific computing, data analysis, and visualization tasks.

6.1.7 Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

7 Software And Hardware Requirements

In this chapter, we will explore the software and hardware requirements necessary to support the system design outlined in earlier chapters. Effective management of software and hardware resources is essential for ensuring optimal performance, scalability, and reliability of the system.

7.1 Software Requirements

i.	Coding language	:	Python
ii.	Platform	:	Google Colab and Kaggle
iii.	GUI Interface	:	flask

7.2 Hardware Requirements

i.	Processor	:	Intel i7 + GPU support
ii.	RAM	:	16GB
iii.	Hard Disk	:	256GB
iv.	Operating System	:	Windows 11 required

Effective management of resources is crucial for ensuring optimal performance, scalability, and reliability. For software, Python is chosen as the coding language, Visual Studios as the platform, and Flask for the GUI interface. Hardware requirements include an Intel i7 processor with GPU support, 16GB of RAM, a 256GB hard disk.

8 Algorithms

This chapter provides a deep dive into the core deep learning algorithms and techniques employed in the development of our text-to-image synthesis model using StackGAN. These algorithms work collaboratively to transform textual descriptions into visually realistic images through a two-stage generative process.

8.1 Algorithms used in the system

i. Generative Adversarial Networks (GANs)

ii. StackGAN – Stage-I

iii. StackGAN – Stage-II

8.1.1 Generative Adversarial Networks (GANs)

GANs are a class of machine learning frameworks where two neural networks—the generator and the discriminator—compete against each other. The generator tries to produce realistic images from noise and text embeddings, while the discriminator attempts to distinguish between real and generated images. This adversarial process helps improve image quality over time.

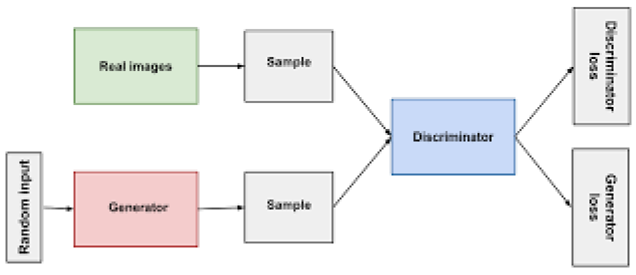


Figure 8.1 Working of GAN

8.1.1.1 Working of StackGANs

Step-1: A random noise vector is given as input to the generator network. This noise acts as a starting point for generating a fake image

Step-2: The generator tries to convert the random noise into a realistic-looking image. Its goal is to "fool" the discriminator into thinking the image is real.

Step-3: Both real images from the training dataset and fake images from the generator are passed to the discriminator. The discriminator's job is to classify images as real or fake.

Step-4: The discriminator provides feedback by calculating the error in prediction. It tells how well the generator was able to fool it and how accurately it classified real and fake images.

Step-5: Both generator and discriminator are updated using backpropagation. The generator improves its ability to create realistic images, and the discriminator gets better at spotting fake ones. This process continues until the generated images are nearly indistinguishable from real ones**.**

8.1.2 StackGAN Stage- I

Stage-I of the StackGAN architecture generates a basic low-resolution image (e.g., 64×64) from the input text embedding. It focuses on getting the overall shape and dominant colours correct, laying the foundation for further refinement.

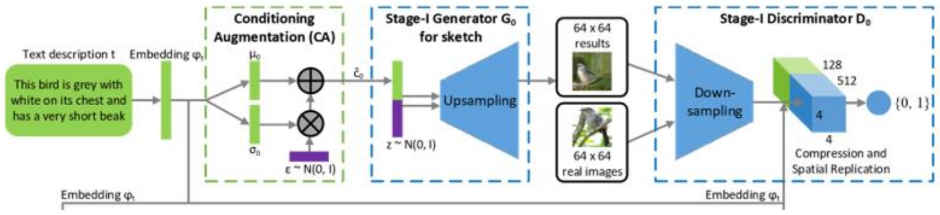

Text description t
This bird is grey with white on its chest and has a very short beak
Embedding φ_t
Conditioning Augmentation (CA)
μ_0
σ_0
ε ~ N(0, I)
ĉ_0
Stage-I Generator G_0 for sketch
z ~ N(0, I)
Upsampling
64 x 64 results
64 x 64 real images
Stage-I Discriminator D_0
Down-sampling
128
512
4
4
Compression and Spatial Replication
{0, 1}
Embedding φ_t
Embedding φ_t

Figure 8.2 Working of StackGAN

8.1.2.1 Working Of StackGAN Stage-I

Step-1: Stage-I begins by taking a text embedding generated from a sentence description (using an RNN or pre-trained model like char-CNN-RNN) as input, which encodes the semantic meaning of the text

Step-2: A random noise vector is concatenated with the text embedding to introduce variation and creativity in the image generation process

Step-3: The Stage-I generator outputs a low-resolution (e.g., 64×64) image that captures the basic shape, color, and layout corresponding to the input text.

Step-4: A discriminator evaluates whether the generated image is real or fake based on both the image and the corresponding text. It ensures the image aligns well with the description.

Step-5: Although coarse and less detailed, the Stage-I output provides a foundation for Stage-II, which will refine the image further by adding finer details and improving resolution.

8.1.3 StackGAN Stage-II

Stage-II takes the output from Stage-I and further enhances it to produce a high-resolution image (e.g., 256×256). It adds finer details, improves texture quality, and ensures that the generated image more accurately reflects the input description.

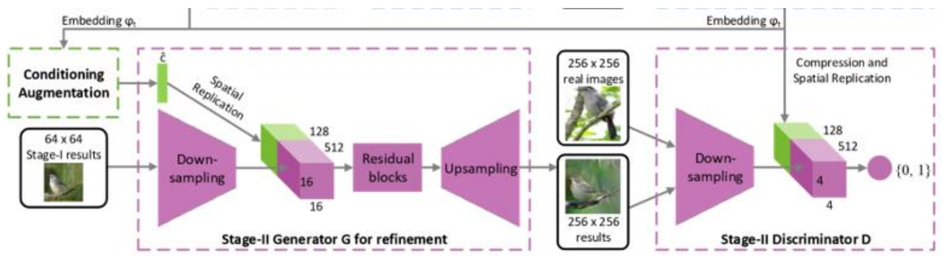


Figure 8.3 Working of StackGAN-II

8.1.3.1 Working StackGAN Stage-II

Step-1: The low-resolution image generated in Stage-I (e.g., 64×64) along with the text embedding is taken as input to Stage-II.

Step-2: Stage-II learns to refine the coarse image by adding more realistic textures, edges, and shapes that match the input text description closely.

Step-3: A more detailed and high-resolution image (e.g., 256×256) is generated using deeper convolutional layers and attention mechanisms.

Step-4: Another discriminator is used in Stage-II to check if the high-res image is realistic and semantically aligned with the given text input and this process continues to until it produces realistic image.

9 Modules

In this chapter, we explore the modular structure of our system designed for text-to-image synthesis using StackGANs. The project is organized into well-defined modules that collectively perform text embedding, image preprocessing, GAN modeling, and evaluation. This modular approach ensures scalability, easier debugging, and clean abstraction between functionalities.

9.1 Modules Used

The different modules in application are:

i. Data collection

ii. Data cleaning and preprocessing

iii. Text and Image Embedding

iv. Model Development using StackGAN

v. Image Generation and Evaluation

9.1.1 Data Collection

The data used in this project is sourced from the **Oxford-102 Flowers Dataset**, which contains:

i. 8,189 flower images across 102 different categories.

ii. 5 unique natural language captions per image, describing the visual appearance of the flower.

The dataset is downloaded and stored in Google Drive. This ensures fast read/write access during training on Google Colab.

Additional steps:

i. Organized images into class folders.

ii. Verified that all images had corresponding captions.

iii. Maintained a consistent mapping between image filenames and caption files.

9.1.2 Data Cleaning and Preprocessing

This step is essential for feeding quality input into the GAN architecture. It includes:

9.1.2.1 Handling Missing Values

Missing or corrupted image files were filtered out. Captions with encoding issues or missing fields were regenerated using caption augmentation strategies (e.g., paraphrasing with NLP tools) to maintain consistency in the number of training samples.

9.1.2.2 Data Formatting

All images were resized to a consistent dimension of **64x64 (Stage-I)** and **128x128 or 256x256 (Stage-II)** using TensorFlow image utilities. Captions were stripped of punctuation, lowercased, and tokenized for embedding generation.

9.1.2.3 Feature Engineering

Since training a language model from scratch was beyond the project scope, we used **pretrained GloVe embeddings**. Captions were converted to 300-dimensional vector representations, and their mean was taken for all 5 captions of an image. This helped represent the semantics of the entire caption group.

9.1.2.4 Normalization

All image pixel values were normalized to a range of **[-1, 1]**, which aligns with the output range of the Tanh activation function used in the generator. Text embeddings were scaled using L2 normalization.

9.1.3 Text and Image Embedding

This module prepares the input for the GAN models. It handles:

i. **Text-to-vector transformation** using GloVe vectors.

ii. **Image-to-tensor conversion** using TensorFlow/Keras preprocessing layers.

iii. **Concatenation of text embeddings with latent noise vectors**, which are then used as input to the generator.

To speed up training, all embeddings were cached into binary files using NumPy for reuse during every training epoch.

9.1.4 StackGAN Modeling

This is the heart of the project. We implemented a **two-stage GAN architecture**:

Stage-I Generator

i. Accepts a noise vector concatenated with text embedding.

ii. Outputs a **64x64 low-resolution image** that reflects rough shapes and colors.

iii. Uses **Residual Blocks, Conv2DTranspose, and Batch Normalization**.

Stage-I Discriminator

i. Takes in a pair (image + text embedding).

ii. Classifies the pair as **real or fake** using sigmoid activation.

Stage-II Generator

i. Takes Stage-I's output and refines it to a higher resolution.

ii. Adds fine-grained details like petal texture, stamen structure, and background clarity.

iii. Uses advanced architectural layers like **up-sampling residual blocks**.

Stage-II Discriminator

i. Performs the same real/fake classification task, but on higher-resolution images.
ii. Enforces stronger alignment with caption semantics.

We used **Wasserstein GAN loss** with gradient penalty for improved training stability.

9.1.5 Evaluation Metrics

Once training is complete, this module handles:

i. **User Input Handling:** Takes a custom text prompt from the user.
ii. **Embedding Generation:** Uses GloVe to embed the input description.
iii. **Image Generation:** Feeds embedding into Stage-I and Stage-II generators.
iv. **Image Output:** Saves or displays the generated image.

valuation Metrics Used:

Manual Assessment:

Visual inspection was also used to assess how closely generated images match the input text

10 Implementation

This chapter provides an in-depth overview of the implementation process of the various modules in our project. It outlines the steps involved in preparing the dataset, designing the model, training and evaluating the StackGAN architecture, and deploying the final system. We also discuss the challenges encountered during development and the tools and frameworks used throughout the implementation.

10.1 Preprocessing

Preprocessing is a critical step in preparing both textual and visual data for GAN training. The following operations were conducted:

i. **Text Cleaning**: Each image has five natural language captions. These were lowercased, tokenized, and stripped of punctuation using Python’s re and nltk libraries.

ii. **Text Embedding**: Cleaned text was converted into 300-dimensional embeddings using **GloVe vectors**. All five caption embeddings per image were averaged to get a single semantic vector.

iii. **Image Resizing**: All flower images were resized to consistent dimensions:

   a. 64×64 for Stage-I

   b. 128×128 or 256×256 for Stage-II

iv. **Normalization**: Images were normalized to [-1, 1] to match the output range of the Tanh activation function in the generator. This was done using transforms.Normalize in PyTorch.

v. **Data Loader**: Custom data loaders were implemented using PyTorch's Dataset and DataLoader classes to fetch (image, caption_embedding) pairs during training.

10.2 Implementation Of Algorithm and Evaluation Metrics :

Algorithm Used: StackGAN

StackGAN consists of two networks:

i. **Stage-I Generator & Discriminator**: Produces low-resolution images (64×64) from noise and text embeddings.

ii. **Stage-II Generator & Discriminator**: Refines Stage-I images to 128×128 or 256×256 resolution with more realistic details.

Both networks were implemented using **PyTorch**, employing:

i. Convolutional layers

ii. Residual connections

iii. Batch normalization

iv. Tanh and LeakyReLU activations

Loss Functions Used:

i. **Generator Loss**: Binary cross-entropy + KL-divergence (for conditioning augmentation).

ii. **Discriminator Loss**: Binary cross-entropy.

iii. **Overall Objective**: Minimize Generator loss while maximizing Discriminator accuracy.

Evaluation Metrics:

i. **Inception Score (IS)**: Used to assess the diversity and clarity of generated images.

ii. **Fréchet Inception Distance (FID)**: Measures distance between real and generated image features.

iii. **Manual Visual Inspection**: Used for checking semantic alignment with captions.

10.3 Process To Implement

i. Begin by collecting the Oxford-102 Flower Dataset, including the corresponding text descriptions. These image-caption pairs form the foundation of the model's training data and are essential for teaching the GAN to relate visual and textual information.

ii. Prepare the data by cleaning the captions, tokenizing them, and generating fixed-length semantic embeddings using pre-trained GloVe vectors. Concurrently, resize and normalize the images to meet the input requirements of the GAN architecture.

iii. Construct the StackGAN model by implementing Stage-I and Stage-II architectures in Tensorflow. Stage-I is responsible for generating low-resolution images from text embeddings and noise, while Stage-II enhances these outputs to higher resolution with additional visual details.

iv. Train the model using adversarial learning, where the generator and discriminator are alternately optimized. The generator learns to produce realistic images that align with the input captions, while the discriminator distinguishes between real and fake image-text pairs.

v. Evaluate the model's output using quantitative metrics like Inception Score (IS) and Fréchet Inception Distance (FID), alongside qualitative visual inspection to ensure that the generated images are both realistic and semantically aligned with their textual descriptions.

vi. Deploy the final trained model using a Flask-based web application (app.py). This interface allows users to input custom text, generate images based on it, and visualize the output, enabling interactive exploration and validation of the model's capabilities.

10.4 Github Repository Having Project Code

All source code and dataset files are maintained in a public GitHub repository:

📎 GitHub Repository – sem8_project

The repository contains:

i. Jupyter Notebook files for training, testing, and evaluation

ii. Dataset in folders and image format

iii. Web app backend (app.py)

iv. HTML templates and image output folders

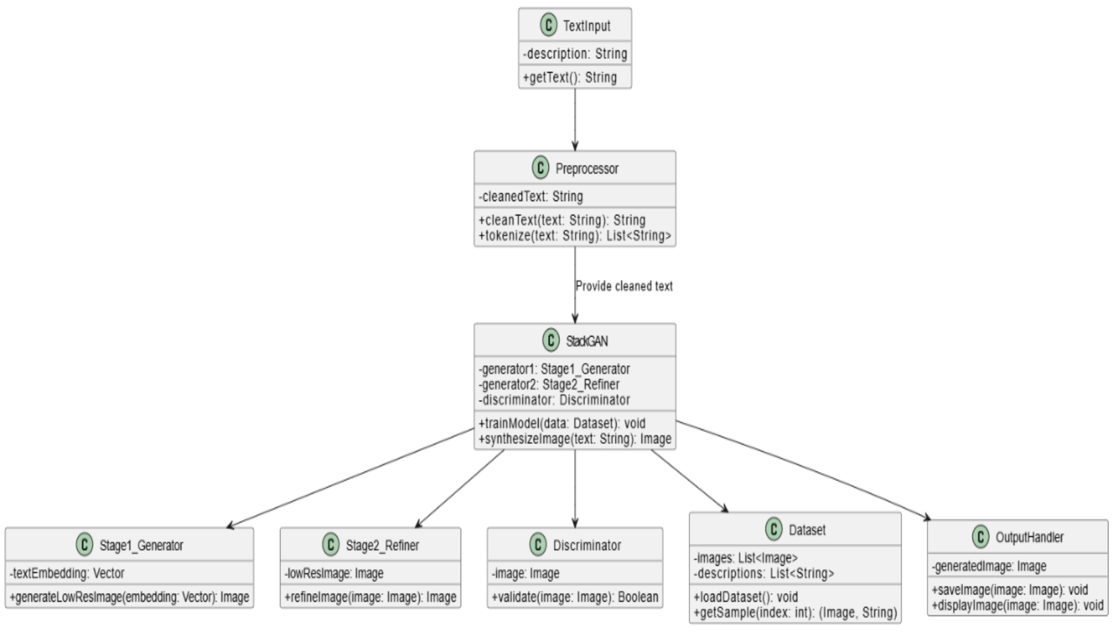
This allows users to run, modify, or deploy the entire pipeline in their own environment

11 Uml Diagrams

Unified Modeling Language (UML) is a standardized modeling language used in software engineering for visualizing, specifying, constructing, and documenting the artifacts of software systems. UML diagrams serve as a common language for stakeholders to communicate and understand the various aspects of a system. These diagrams provide different perspectives on the system, capturing its structure, behavior, and interactions.

11.1 Class Diagram

Class diagrams are essential components in software engineering, providing a graphical depiction of a system's static structure. They offer insights into classes, their attributes, operations, and relationships, aiding in system comprehension and design. Classes serve as blueprints for objects, encapsulating both data and behaviors within the system architecture.



TextInput
-description: String
+getText(): String
Preprocessor
-cleanedText: String
+cleanText(text: String): String
+tokenize(text: String): List<String>
Provide cleaned text
StackGAN
-generator1: Stage1_Generator
-generator2: Stage2_Refiner
-discriminator: Discriminator
+trainModel(data: Dataset): void
+synthesizeImage(text: String): Image
Stage1_Generator
-textEmbedding: Vector
+generateLowResImage(embedding: Vector): Image
Stage2_Refiner
-lowResImage: Image
+refineImage(image: Image): Image
Discriminator
-image: Image
+validate(image: Image): Boolean
Dataset
-images: List<Image>
-descriptions: List<String>
+loadDataset(): void
+getSample(index: int): (Image, String)
OutputHandler
-generatedImage: Image
+saveImage(image: Image): void
+displayImage(image: Image): void

Figure 11.1 Class Diagram

11.2 Use Case Diagram

Use case diagrams serve as a vital tool in capturing and illustrating the functional requirements of a system from the perspective of its users or external entities. These diagrams provide a visual representation of the interactions between actors, such as users or external systems, and the system itself. Actors are depicted outside the system boundary, representing entities interacting with the system, while use cases, depicted within the system boundary, represent specific functionalities or tasks offered by the system.

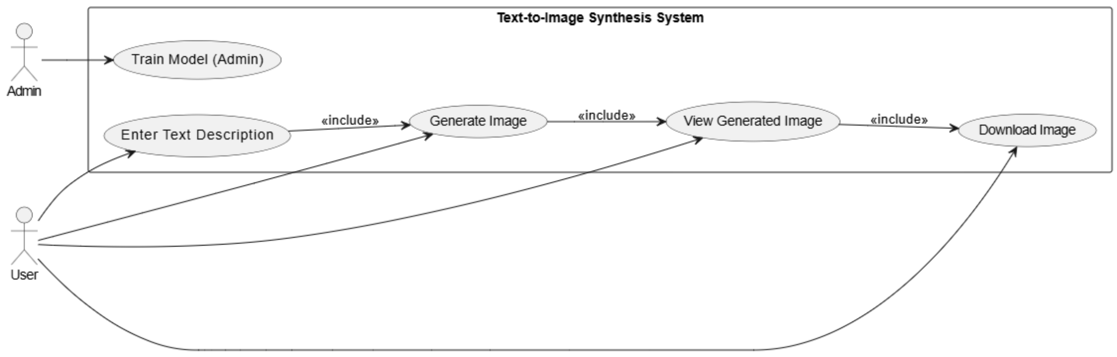


Figure 11.2 Use Case Diagram

11.3 Sequence Diagram

Sequence diagrams are instrumental in visualizing the dynamic behavior of a system by depicting the sequence of interactions between objects over time. These diagrams offer a graphical representation of how objects communicate with each other in terms of messages exchanged and the order of execution. Each object is represented as a vertical lifeline, with messages passed between them depicted as arrows along the timeline

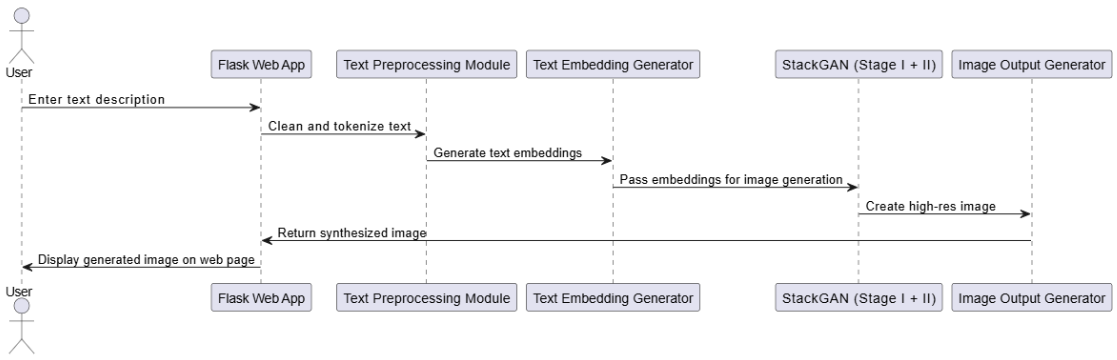


Figure 11.3 Sequence Diagram

11.4 Activity Diagram

Activity diagrams offer a comprehensive visualization of the flow of activities within a system, showcasing the sequence of actions and decision points. These diagrams provide a high-level overview of the workflow or business process, capturing both the control flow and the object flow. Actions are represented as nodes, connected by arrows to illustrate the flow of control, while decision points are depicted using diamondshaped nodes to indicate branching paths based on conditions

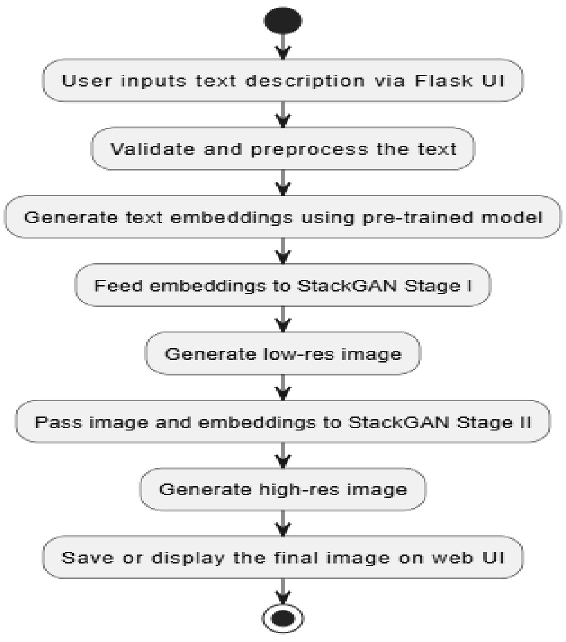

User inputs text description via Flask UI
Validate and preprocess the text
Generate text embeddings using pre-trained model
Feed embeddings to StackGAN Stage I
Generate low-res image
Pass image and embeddings to StackGAN Stage II
Generate high-res image
Save or display the final image on web UI

Figure 11.4 Activity Diagram

12 Results

The journey from conceptualization to implementation culminates in the generation of visual outputs that validate the effectiveness of the proposed StackGAN-based text-to-image synthesis system. In this section, we present a comprehensive overview of the generated images and assess their quality, relevance, and alignment with the input textual descriptions. These results serve as clear indicators of the model's ability to understand and translate semantic content into realistic visual representations.

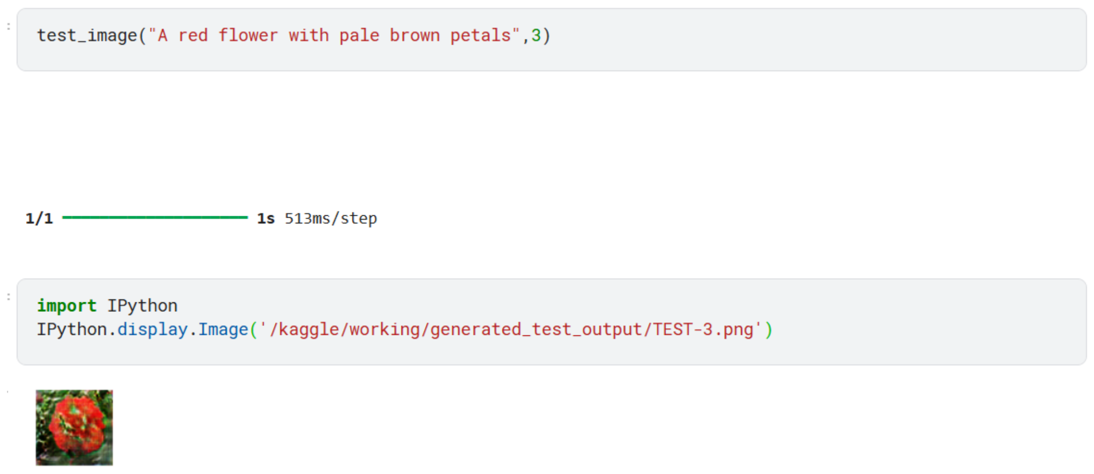
12.1 Flask Web Application

We have deployed our text-to-image synthesis model through a web application built using the Flask framework. Flask, being a lightweight and versatile Python-based web framework, enables the seamless integration of our trained StackGAN model with an interactive user interface. This setup allows users to input custom textual descriptions and receive corresponding synthesized images in real-time.

12.2 Output

The performance of the model was assessed based on both qualitative and observational criteria, with a focus on visual realism, semantic accuracy, and the ability of the generated image to reflect the input description. The two-stage generation process demonstrated a notable improvement in visual quality at the second stage, where the generated images exhibited more refined details, better color consistency, and clearer structural patterns of flowers.

Notepad Output Example:



```
test_image("A red flower with pale brown petals",3)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 513ms/step
```

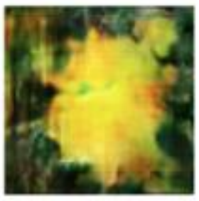
```
import IPython
IPython.display.Image('/kaggle/working/generated_test_output/TEST-3.png')
```

Figure 12.1 Colab Output

Example 1:

Text Description: "a flower that has vivid yellow petals and dark purple stamen."

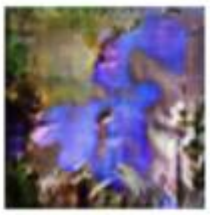
Output:



Example 2:

Text Description: "this is a blue flower with round petals."

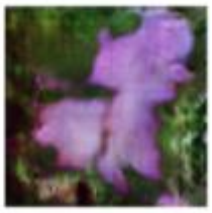
Output:



Example 3:

Text Description: "the pistil is white and is very noticeable, and the petals are purple."

Output:



Flask Output Example:

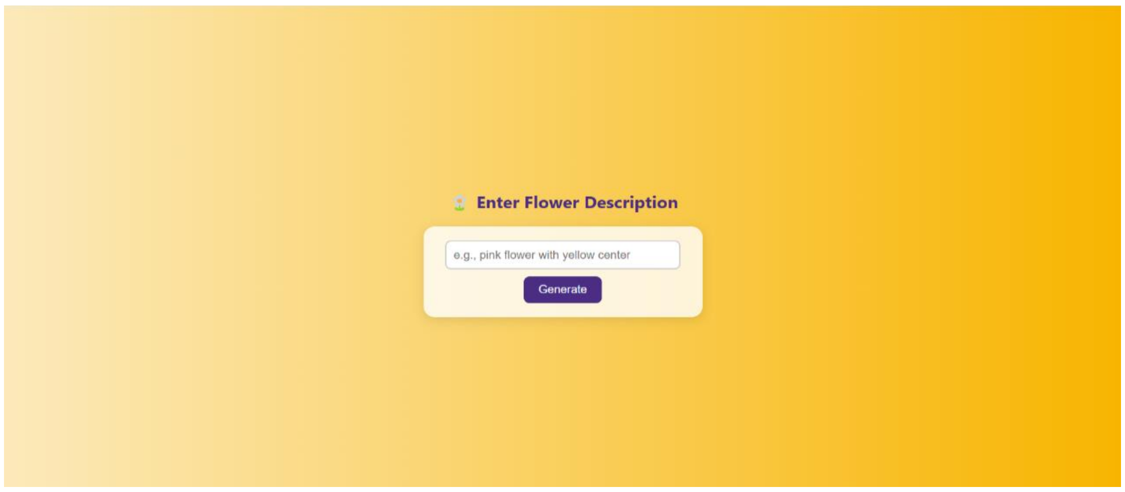
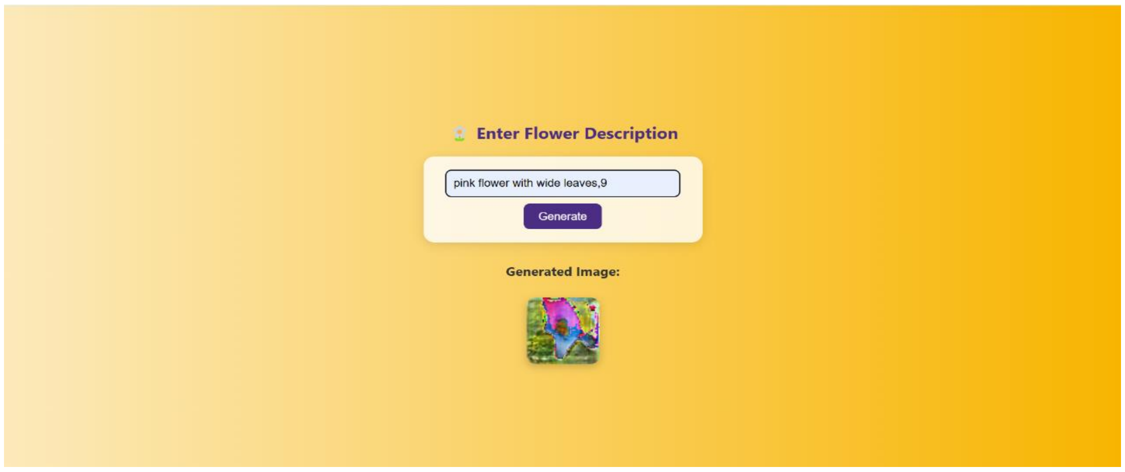


Figure 12.2 Flask Output-1



pink flower with wide leaves,9

Figure 12.3 Flask Output-2

13 Conclusion And Future Work

This concluding chapter encapsulates the outcomes, significance, and potential advancements of the text-to-image synthesis project. The journey of building a generative system that transforms natural language descriptions into coherent and visually accurate images using StackGAN has been both technically challenging and rewarding. The project highlights the fusion of natural language processing and computer vision, showcasing how generative adversarial networks (GANs) can be used creatively for cross-modal learning tasks.

13.1 Conclusion

The goal of this project was to implement a text-to-image synthesis system using StackGAN architecture that can generate high-resolution, semantically relevant images from textual descriptions. Throughout the development cycle, the project involved preprocessing textual data, extracting embeddings using a pretrained text encoder, training the Stage-I and Stage-II GAN models, and evaluating the quality of the generated images.

By successfully training StackGAN on relevant datasets, the system demonstrated the capability to synthesize realistic images that correspond closely to the descriptive input. This project reflects the growing potential of deep generative models in bridging the gap between language and vision.

The results indicate that conditional GANs, especially StackGAN, are effective in generating detailed and visually appealing images when trained with sufficient and well-structured data. The project thus provides a strong foundation for future research and practical applications in the domain of generative AI.

13.2 Future Work

While the current implementation achieves promising results, several areas offer scope for further improvement and research:

i. Replacing basic text encoders with transformer-based models like BERT or CLIP can enhance semantic understanding and lead to better image alignment with textual input.

ii. Introducing attention mechanisms or integrating modules like AttnGAN can help the model focus on specific words or phrases, improving the consistency and detail of generated images.

iii. Expanding the dataset with more diverse and complex captions and images can help improve generalization across various object categories, styles, and scenes.

iv. Optimizing the model for inference speed could enable real-time generation, which is useful for interactive applications like games, virtual storytelling, and design tools.

v. Future work should also address the ethical use of generative models, ensuring that generated content is used responsibly and aligns with societal norms and legal frameworks

In summary, text-to-image synthesis remains an exciting and rapidly evolving field in AI. With advancements in deep learning and transformer architectures, the project sets the stage for continued innovation in generating visually accurate and contextually relevant images from natural language descriptions.

14 Bibliography

[1] Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). **Generative Adversarial Text to Image Synthesis**. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (pp. 1060–1069).

[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). **Generative Adversarial Nets**. In *Advances in Neural Information Processing Systems* (pp. 2672–2680).

[3] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021). **Zero-shot Text-to-Image Generation**. *arXiv preprint arXiv:2102.12092*.

[4] Li, B., Qi, X., Lukasiewicz, T., & Torr, P. H. (2019). **Controllable Text-to-Image Generation**. In *Advances in Neural Information Processing Systems*, 32.

[5] Brock, A., Donahue, J., & Simonyan, K. (2019). **Large Scale GAN Training for High Fidelity Natural Image Synthesis**. In *International Conference on Learning Representations (ICLR)*.

[6] Zhang, Y., Jiang, H., & Yang, C. (2021). **Cross-Modal Contrastive Learning for Text-to-Image Generation**. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 833-841).

[7] Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D. N. (2017). **StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks**. In *2017 IEEE International Conference on Computer Vision (ICCV)* (pp. 5907-5915). IEEE