TEXT TO IMAGE GENERATION USING BERT AND GAN

Mini Project Report

Submitted in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology (B.Tech)

In

Department of CSE (Artificial Intelligence & Machine Learning)

By

Vangapally Bhanu	**21AG1A6659**
Syed Sameer Sohail	**21AG1A6656**
Dubba Harish	**22AG5A6605**

Under the Esteemed Guidance of
Dr. Kavitha Soppari
Hod & Assoc Prof

Department of CSE (Artificial Intelligence & Machine Learning)
ACE ENGINEERING COLLEGE

An Autonomous Institution

(NBA ACCREDITED B.TECH COURSES: EEE, ECE & CSE, ACCORDED NAAC 'A' GRADE)

Affiliated to Jawaharlal Nehru Technological University, Hyderabad, Telangana,

Ghatkesar, Hyderabad – 501 301

DECEMBER 2024

ACE
Engineering College
An Autonomous Institution

(NBA ACCREDITED B.TECH COURSES: EEE, ECE & CSE, ACCORDED NAAC 'A' GRADE)

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad, Telangana)

Ghatkesar, Hyderabad – 501 301

Website : www.aceec.ac.in E-mail: info@aceec.ac.in

CERTIFICATE

This is to certify that the Mini Project work entitled **"TEXT TO IMAGE GENERATION USING BERT AND GAN"** is being submitted by **Vangapally Bhanu (21AG1A6659), Syed Sameer Sohail (21AG1A6656), Dubba Harish (22AG5A6605)** in partial fulfillment for the award of Degree of **BACHELOR OF TECHNOLOGY** in **DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)** to the Jawaharlal Nehru Technological University, Hyderabad during the academic year 2024-25 is a record of bonafide work carried out by him/her under our guidance and supervision.

The results embodied in this report have not been submitted by the student to any other University or Institution for the award of any degree or diploma.

Internal Guide
Dr. KAVITHA SOPPARI
Professor and Head
Dept. of CSE (AI & ML)

Head of the Department
Dr. KAVITHA SOPPARI
Professor and Head
Dept. of CSE (AI & ML)

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our gratitude to all the people behind the screen who have helped us transform an idea into a real time application.

We would like to express our heart-felt gratitude to our parents without whom. We would not have been privileged to achieve and fulfill our dreams.

A special thanks to our Secretary, **Prof. Y. V. GOPALA KRISHNA MURTHY,** for having founded such an esteemed institution. We are also grateful to our beloved principal, **Dr. B. L. RAJU** for permitting us to carry out this project.

We profoundly thank **Dr. Kavitha Soppari,** Assoc. Professor and Head of the Department of CSE (Artificial Intelligence & Machine Learning), who has been an excellent guide and also a great source of inspiration to my work.

We extremely thank, **Mrs. Sri Sudha Garugu,** Assistant Professor , Mini Project coordinator, who helped us in all the way in fulfilling of all aspects in completion of our Mini Project.

We am very thankful to my **Dr. Kavitha Soppari,** Hod & Assoc Prof, who has been an excellent and also given continuous support for the Completion of my Mini Project work.

The satisfaction and euphoria that accompany the successful completion of the task would be great, but incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success. In this context, I would like to thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased my task.

Vangapally Bhanu (21AG1A6659),

Syed Sameer Sohail (21AG1A6656),

Dubba Harish (22AG5A6605)

DECLARATION

This is to certify that the work reported in the present project titled "**TEXT TO IMAGE GENERATION USING BERT AND GAN**" is a record work done by us in the Department of CSE (Artificial Intelligence & Machine Learning), ACE Engineering College.

No part of the thesis is copied from books/journals/internet and whenever the portion is taken, the same has been duly referred in the text; the reported are based on the project work done entirely by us not copied from any other source.

Vangapally Bhanu (21AG1A6659),
Syed Sameer Sohail (21AG1A6656),
Dubba Harish (22AG5A6605)

ABSTRACT

Text-to-image generation is a challenging task that bridges computer vision and natural language processing. Existing models based on generative adversarial networks (GANs) typically use text encoders pre-trained on image-text pairs. However, these encoders often fail to capture the semantic depth of unseen text during pre-training, making it difficult to generate images that align well with provided textual descriptions.to overcome this issue, we introduce a new text-to-image generation model that incorporates BERT, a highly effective pre-trained language model in natural language processing. By fine-tuning BERT on a large text corpus, we enable it to encode rich textual information, enhancing its suitability for image generation tasks. Experiments conducted on a CUB_200_2011 dataset reveal that our method outperforms baseline models in both quantitative metrics and qualitative assessments.

INDEX

CONTENTS PAGE NO

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Text-to-image generation represents a fascinating intersection of natural language processing (NLP) and computer vision, aiming to create realistic visual content from descriptive text inputs. This technology is powered by advanced machine learning models such as BERT (Bidirectional Encoder Representations from Transformers) and Generative Adversarial Networks (GANs), both of which bring unique capabilities to the task. By integrating the semantic understanding of text with the visual creativity of GANs, text-to-image generation is emerging as a transformative tool in areas ranging from creative design to automated content creation.

BERT, developed by Google, is a groundbreaking transformer-based model that has redefined how machines understand language. Unlike traditional NLP models, BERT processes text bidirectionally, meaning it considers the context of words from both preceding and succeeding tokens simultaneously. This bidirectional approach enables BERT to grasp subtle nuances and relationships in text, such as syntactic dependencies and semantic meanings, with remarkable precision. For instance, in the sentence "The bank was crowded with people waiting for a financial consultant," BERT can discern that 'bank' refers to a financial institution rather than a riverbank, thanks to its context-aware representation learning. Such a robust understanding of textual context forms the backbone of the text-to-image generation pipeline, ensuring that the system can accurately interpret even the most intricate or abstract textual descriptions.

On the other side of this innovative pipeline lies the Generative Adversarial Network (GAN), a class of machine learning frameworks that excels in image synthesis. A GAN consists of two neural networks—a generator and a discriminator—that engage in an adversarial training process. The generator's task is to produce images, starting from a noise vector, which are then refined based on textual embeddings supplied by the BERT model. The discriminator, in turn, evaluates the realism of the generated images, distinguishing between authentic images and those created by the generator. Over successive iterations, this adversarial process leads to a gradual improvement

in the quality and fidelity of the generated images. The generator learns to produce images that are not only visually realistic but also semantically aligned with the input textual descriptions.

The synergy between BERT and GANs lies in their complementary strengths. While BERT captures the semantic depth and contextual nuances of textual data, GANs translate these embeddings into coherent visual representations. The process begins with BERT encoding the input text into a dense vector representation that encapsulates its semantic content. These text embeddings are then fed into the GAN's generator as conditional inputs, guiding the image synthesis process. The GAN generator maps the text embeddings into the latent space, ensuring that the resulting images correspond closely to the described content. For example, if the input text describes "a serene sunset over a mountain range," the system generates an image that not only features mountains and a sunset but also captures the serene atmosphere implied by the description.

The iterative refinement process is a critical aspect of this architecture. During training, the generator and discriminator continuously improve through feedback loops, with the discriminator pushing the generator to create more realistic images. The final output is not merely a visual representation of the text but an artistic rendering that aligns with the semantics, mood, and tone of the input description. Furthermore, advanced GAN variants such as StackGAN and StyleGAN enhance the system's ability to produce high-resolution images with fine-grained details, taking text-to-image generation to unprecedented levels of quality.

Applications of text-to-image generation are vast and diverse. In the creative industries, artists and designers can use these systems to visualize concepts based on textual prompts, accelerating the ideation process. In automated content creation, businesses can generate product visuals, advertisements, and storyboards directly from textual inputs, reducing production time and costs. Additionally, this technology has potential applications in accessibility, enabling visually impaired individuals to gain insights into textual content through images. The ability to generate realistic and contextually relevant images from text also opens up new possibilities in education, gaming, and virtual reality, making it a cornerstone of next-generation human-computer interaction.

1.1 BACKGROUND AND CONTEXT OF THE PROJECT

The rapid advancements in artificial intelligence have fueled the convergence of natural language processing (NLP) and computer vision, enabling machines to interpret and generate content across modalities. Text-to-image generation, a prime example of this convergence, leverages the semantic understanding of textual descriptions to create corresponding visual representations. The project builds on state-of-the-art models like BERT (for deep contextual text understanding) and GANs (for realistic image synthesis), aiming to transform the way humans interact with AI for creative and practical applications. This innovative approach bridges the gap between textual input and visual creativity, offering immense potential in fields such as design, media, and education.

1.1.1 What is BERT?

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained NLP model introduced by researchers at Google in October 2018. Its primary innovation lies in its ability to deeply understand the context of words within sentences by analyzing them bidirectionally, which sets it apart from earlier unidirectional models like GPT (Generative Pre-trained Transformer). BERT revolutionized the field of NLP by enabling machines to achieve human-like understanding of language for tasks such as question answering, sentiment analysis, and text classification.

History and First Usage

BERT was first introduced in a research paper titled “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” published by Jacob Devlin and his team at Google. Upon release, BERT significantly outperformed existing models on a variety of NLP benchmarks, including the Stanford Question Answering Dataset (SQuAD) and the General Language Understanding Evaluation (GLUE) tasks. It marked a paradigm shift in NLP by setting new standards for pre-training and fine-tuning language models.

BERT was immediately adopted by Google Search to improve the understanding of user queries. By 2019, it became a core component in Google’s search algorithm, helping the search engine comprehend natural language queries with higher accuracy.

How BERT Works

BERT is based on the Transformer architecture, which uses self-attention mechanisms to process input sequences. Unlike traditional models that process text sequentially from left-to-right (or right-to-left), BERT processes the entire input sentence at once. This bidirectional approach enables BERT to consider the context of a word from both its preceding and succeeding tokens simultaneously, providing a deeper understanding of language nuances.

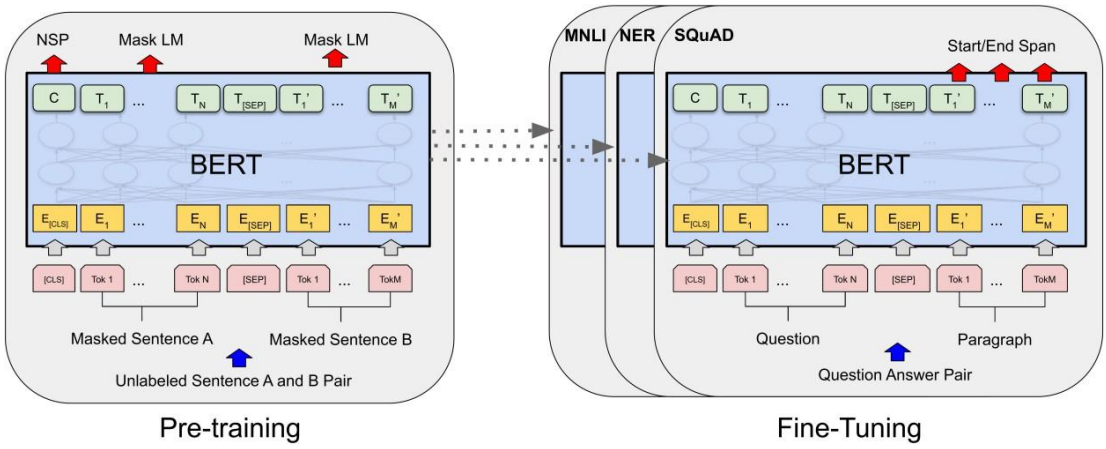


Fig 1.1.1-BERT

1. **Pre-training Phase**:

- BERT is pre-trained on a massive corpus of text using two primary tasks:
  - **Masked Language Modeling (MLM)**: During pre-training, 15% of the input tokens are randomly masked, and the model learns to predict these masked tokens based on their surrounding context.
  - **Next Sentence Prediction (NSP)**: BERT is trained to predict whether a given sentence logically follows another sentence, enhancing its understanding of sentence relationships

2. **Fine-tuning Phase**:

- After pre-training, BERT can be fine-tuned on specific downstream tasks such as sentiment analysis, question answering, or named entity recognition. Fine-tuning involves adding a task-specific layer to the pre-trained model and training it on labeled data for the target task.

Types of BERT

Several variants of BERT have been developed to cater to different applications and computational constraints:

1. **Base BERT**:

- Consists of 12 transformer layers with 110 million parameters.

  Suitable for most standard NLP tasks.

2. **Large BERT**:

- Consists of 24 transformer layers with 340 million parameters.

Offers better performance for complex tasks but requires more computational resources.

3. **DistilBERT**:

- A smaller, faster version of BERT with 40% fewer parameters and 60% faster inference times.

- Designed for scenarios with limited computational resources.

4. **ALBERT (A Lite BERT)**:

- A lightweight version of BERT that reduces memory usage and training time by sharing parameters across layers and decomposing embedding matrices.

5. **RoBERTa (Robustly Optimized BERT)**:

- A variant of BERT that removes the NSP task and is pre-trained on a larger corpus, resulting in improved performance on many NLP benchmarks.

How to Train BERT

Training BERT involves two key phases: pre-training and fine-tuning. Each phase has specific requirements and methodologies to achieve optimal results.

1. Pre-training BERT

- Pre-training is the phase where BERT learns contextual word representations from large amounts of unlabeled text data. The process requires significant computational resources and follows these steps:

- **Dataset Preparation**: Use a diverse and large corpus, such as Wikipedia or BooksCorpus, to ensure broad language coverage. The dataset should be tokenized using WordPiece or a similar subword tokenization method.
- **Masked Language Modeling (MLM)**: Randomly mask 15% of the tokens in the input and train the model to predict the masked tokens based on their context. This encourages the model to learn bidirectional representations.
- **Next Sentence Prediction (NSP)**: Create pairs of sentences, some of which are consecutive and some unrelated. Train the model to predict whether the second sentence follows the first.
- **Training Configuration**:
    - ✧ Optimizer: Use Adam with weight decay.
    - ✧ Learning Rate: Employ a warm-up strategy with an initial low rate that gradually increases.
    - ✧ Hardware: Utilize GPUs or TPUs for parallel processing due to the high computational cost.

2. Fine-tuning BERT

Fine-tuning involves adapting the pre-trained model to a specific downstream task. The steps include:

- **Dataset Preparation**: Prepare labeled data for the target task, such as question-answer pairs for a question-answering system or sentiment-labeled text for sentiment analysis.
- **Task-Specific Architecture**: Add a task-specific layer on top of BERT's output, such as a classification head for classification tasks or a start-end token predictor for question answering.

➢ **Training Configuration**:

- ✧ Smaller Learning Rate: Fine-tune with a smaller learning rate (e.g., 2e-5) to avoid catastrophic forgetting.
- ✧ Batch Size: Use smaller batch sizes due to memory constraints.
- ✧ Epochs: Train for fewer epochs (e.g., 3-5) since the model is already pre-trained.

➢ **Evaluation**: Regularly evaluate the model on validation data to prevent overfitting and adjust hyperparameters as needed.

Tools and Frameworks for Training

Several open-source frameworks make it easier to train and fine-tune BERT:

- ➢ **Hugging Face Transformers**: Provides pre-trained BERT models and tools for fine-tuning.
- ➢ **TensorFlow**: Offers implementations of BERT with flexibility for custom modifications.
- ➢ **PyTorch**: Popular for research and experimentation, with robust support for BERT training.

Challenges in Training

- ➢ **Computational Cost**: Pre-training requires extensive computational resources, often necessitating cloud-based solutions or TPUs.
- ➢ **Overfitting**: Fine-tuning on small datasets can lead to overfitting. Techniques such as data augmentation and regularization can help mitigate this.

Applications of BERT

BERT's ability to deeply understand context has made it a cornerstone in various NLP applications:

- ➢ **Search Engines**: Improves query understanding in search results.
- ➢ **Chatbots**: Enhances conversational AI by enabling more accurate responses.
- ➢ **Question Answering**: Powers systems like virtual assistants to comprehend and answer user queries.
- ➢ **Text Summarization**: Generates concise summaries from lengthy documents.

1.1.2 WHAT IS GAN?

Generative Adversarial Networks is one of the pioneering frameworks of machine learning, initiated first by Ian Goodfellow in 2014. Fundamentally, GANs are developed to create new data samples, which are most like a dataset that has already been seen, ranging from creating photorealistic images or synthetic videos up to texts and music generation. The innovation happens in the adversarial structure. It engages two neural networks, called the generator and the discriminator, in a zero-sum game that supports iterative learning and progressively makes the output closer to the real-world data.

There are two components in GANs:

a) Generator

b) Discriminator

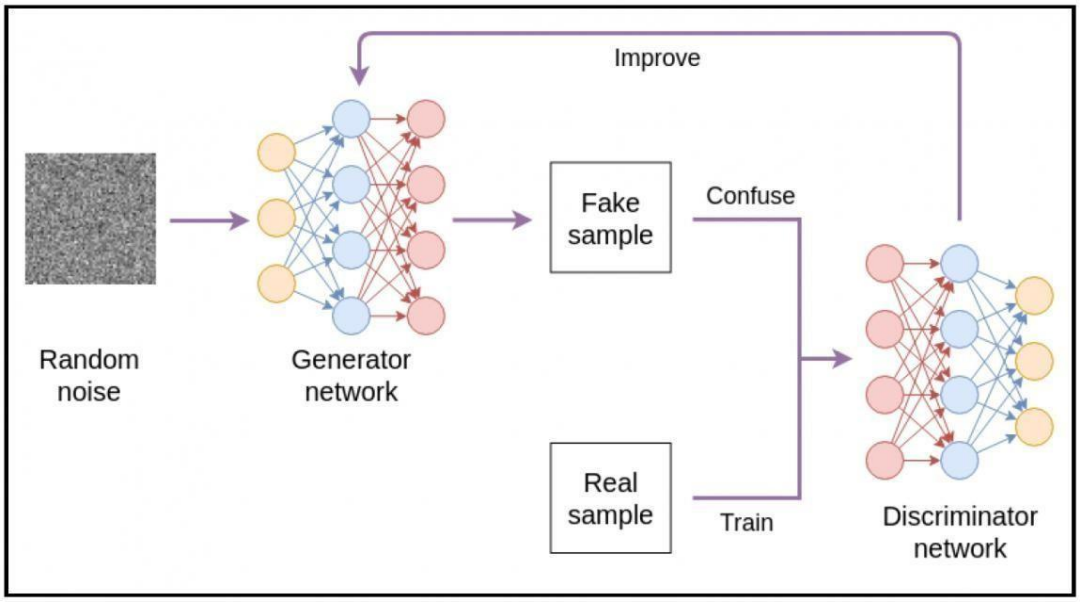


Fig 1.1.2: GAN

GENERATOR:

The generator is often considered the creative drive behind the Generative Adversarial Networks (GANs). It basically tries to generate new data inputs that would resemble the form of the actual data around which it has been exposed. The process starts at a stochastic input, usually given as uniformly or Gaussianly distributed, acting as a sort of latent space, which is

then a compact representation of probable outputs. This noise is translated by the generator into a data space that corresponds with the actual distribution of the data, thereby converting unstructured noise into organized and significant outputs.

These preliminary samples produced at the initiation of the training procedure are crude and ungrounded. Such simple early samples are obvious, riddled with artifacts or anomalies, so that a human observer, or even the discriminator network, can easily detect them as being artificial. However, as the training progresses, the generator continues to improve gradually. It learns to refine its outputs through iterative optimization and feedback from the discriminator, capturing complex patterns, textures, and details that exist in the real dataset. As such, over time, the generator becomes proficient at generating samples that are nearly indistinguishable from real data, pushing the boundaries of realism and diversity. This is the power of evolving from generating simple unrealistic data to producing complex high-quality samples, and that is the power of adversarial training and the fundamental principle of GANs.

DISCRIMINATOR:

The true discriminator in a Generative Adversarial Network (GAN) is quite the calculation model-his primary function is to determine whether a presentation is of real or fabricated nature stemmed from the generator. In essence, the generator acts as the critic in this adversarial establishment where the discriminator scrutinizes every piece of data it receives and assigns probabilities to how likely the sample is real. This constant evaluation ensures that the generator is being pushed to produce outputs that closely resemble real data.

The discriminator's goal becomes one of almost perfect classification: differentiating between real and generator-synthesized samples. In essence, a mixed batch of training samples comprises the generator's original data and those generated. Initially, when the generator output is still very crude, the discriminator had little or no trouble detecting the fakes, and so its classification capacity was enhanced. However, as time passes, the fakes get more accurate, and the discriminator has a harder time separating the two types of data. This increases the onus on the discriminator to hone the decision-making process and to learn the more subtle aspects that make real and fake samples different from each other.

The generator and discriminative models balance each other in the networking of generators and discriminators by existing in a state of constant and competing alternation. The generator tries to produce data that seems to be truly from the specific class of examples, and thus fool the pair, while the objective of the discriminator is to find the fakes and throw them out. This gives rise to a sublime balance: for the generator to improve, it must take note of the weaknesses of what is currently understood by the discriminator of that class of data. On the other hand, the discriminator must work acclimatizing with the ongoing discharge strategies of the generator gaining ever subtle information in discerning fakes from the real.

HOW TO TRAIN GANs:

The Generative Adversarial Network (GAN) is a family of models based on a mathematical rationale called the minimax game, which envisages a self-contained adversarial relationship between the generator and the discriminator. In that game-like situation, the generator is trained to fool the discriminator in such events, while the prior aims to maximize its ability to confidently separate the real data from the original dataset and the fakes generated by the generator. This interplay of opposing goals sets the stage for just such a dynamic: the generator tries to adapt enough to render anything he or she generates convincing. At the same time, the discriminator becomes ever more adept at detecting the slightest imperfections in the fake data. The process over time thus elevates both players into high perfection, such that the generator becomes good at generating the most realistic data and the discriminator very sensitive in distinguishing real data from the generated.

However, employing such a technique, despite its high efficiency, is somewhat challenging. A pivotal issue is mode collapse, where the generator learns to output a limited number of optimized versions, thus failing to capture diversity in the input training data. For example, with a visual-generation application producing images of human faces, one such generator could be detected to allow generating various non-mournful expressions of a single face, rather than resolving itself to develop variants for a substantial set of distinguished types of face appearances. Another challenge is non-convergence, where the generator and discriminators simply will not stabilize in its learning over time as training proceeds, instead oscillating while being far removed from just a little improvement. In this manner, the quest to

attain equilibrium is decisive for optimal performance.

Recognizing these challenges, researchers have proposed several advancements to improve the stability and performance of GANs. WGANs use a new loss function based on the Wasserstein distance (also known as Earth Mover's Distance) between the real data distribution and the generated samples. The objective will help in avoiding problems such as vanishing gradients, occurring when the discriminator becomes too confident in distinguishing real from fake samples. Additionally, stability in the training and good results are facilitated by improved gradients to the generator. Progressive Gan is another important contribution: the framework where substantial GANs are being learned in stages, starting from a low-resolution dataset while gradually increasing the resolution output.

Step-by-step improvements let the GAN first learn simple features, such as shapes and structures, then proceed to learn more complicated details such as texture and pattern. Progressive growing of GANs has shown much success in generating high-resolution images, especially of remarkably detailed and realistic-looking human faces. Even with such challenges, GANs continue to remain one of the most powerful tools in generative modeling. The adversarial training paradigm will always serve as a means regulating the generator and discriminator for these two to incessantly compete, consequently fostering development and creativity specifically in the mitigation of the setbacks that are accompanied by learning in the process, very often complicated. With the continuance of research and the introduction of new techniques, GANs continue to evolve, extending newer opportunities for both synthesis of data and creative utilization.

The general mathematical formula used to train GANs:

“GminDmaxEx~pdata(x)[logD(x)]+Ez~pz(z)[log(1−D(G(z)))] 1”

Types of GANs:

a) **TGANs**
b) **CGANs**

TEMPORAL GENERATIVE ADVERSARIAL NETWORKS (TGANs):

Temporal GANs find broad applications in time-series data generation and modeling because they implicitly capture temporal dependencies and generate sequential outputs. Time-series is available in abundance in segments such as finance, healthcare, energy, and IoT,

wherein the observations are recorded over time. The subsequent section presents a summary of the utilization and contribution of TGANs in time-series data generation:

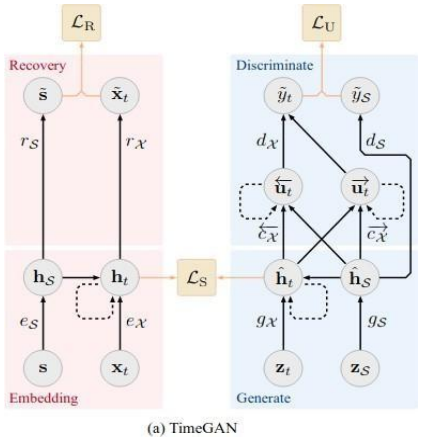


Fig 1.1.2: Time GAN

1. Synthetic Time-Series Data Generation

TGANs produce realistic synthetic time series that essentially conform to the statistical properties and dynamics of real data. This eases the problems posed by data privacy in such sensitive fields as finance and medicine, or in situations when the data is small enough that training good machine learning models has become hopeless. Training a TGAN on a given dataset allows synthetic sequences to be generated, which preserve the time structure inherent in the real data, thus adding more variety and quantity to analogies or model training.

2. Data Augmentation

In the case of a rather limited training set, primarily in the small datasets acquired in practice, TGAN can enhance the existing dataset with synthetic samples. For instance, weather datasets can be amended with the aid of TGANs for the training of advanced forecasting models, whereas IoT devices can use TGANs to create more user activity logs for their anomaly detection systems, which means adding more variety and volume to the available training data translates into better working models.

3. Missing Data Imputation

Due to malfunctioning sensors or insufficient recording, it may not be unusual to encounter missing values in time series data. The solution arises when TGANs learn the temporal patterns of the data and fill in the missing values with synthetic sequences conforming with the observed data.

CHALLENGES IN USING TGANs:

1. Capturing Long-Term Dependencies

The time-series data often involves long temporal relationships such as yearly seasonal patterns in climate data. TGAN may have difficulty maintaining those dependencies over long sequences. Hence, the generated sequences lack long-term coherence since their modelling requires more advanced techniques to maintain such patterns for longer periods.

2. Training Instability

Training instabilities inherited from the GANs, like mode collapse (where the generator produces limited diversity) and non-convergence, are exacerbated in the case of time-series data because of added complexities of temporal dynamics that complicate training.

3. High Computational Complexity

Temporal data modelling takes a lot more computation than static data modelling does. The addition of recurrent structures such as LSTMs and GRUs in both the generator and the discriminator only adds up to increased training times and resource races, which render TGANs computationally expensive and hardware hungry.

4. Evaluation of the Generated Data

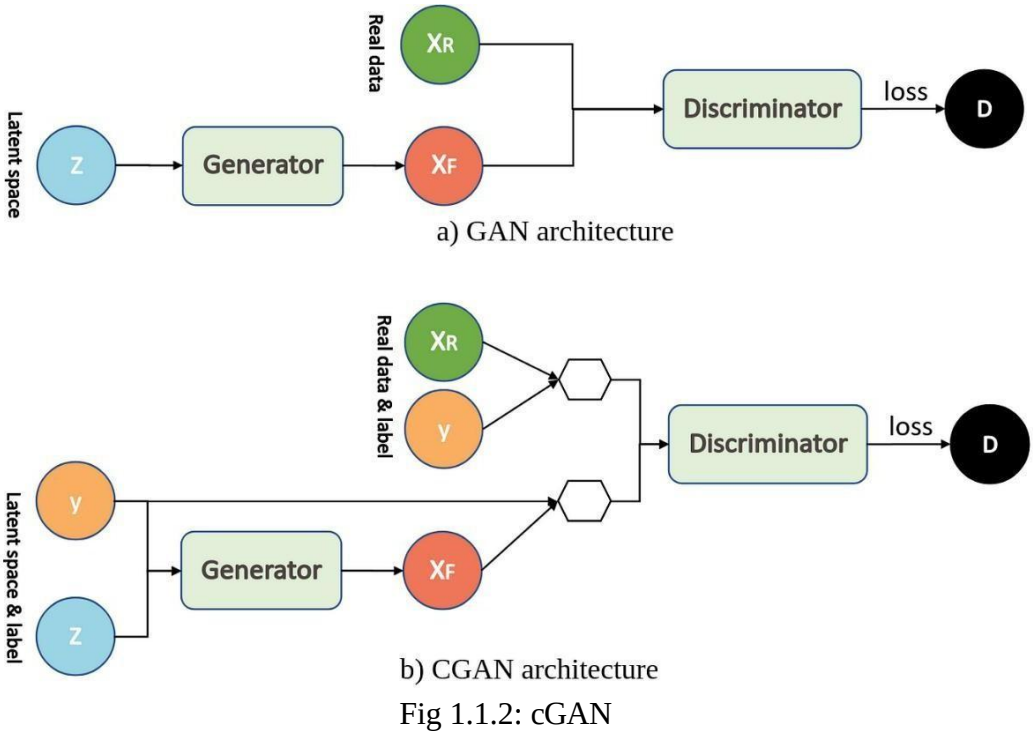
The evaluation of synthetic time-series data is less accessible than the evaluation of images. It needs metrics which consider temporal dependencies (for example, autocorrelation), distributional similarity with real data, and domain specificity. Existing evaluation metrics may fail to capture some or all these dimensions and would make it impossible to be sure of the fidelity of generated sequences.

5. Diversity vs. Realism Trade-Off

Another everlasting challenge for TGANs is to balance between producing diverse outputs and keeping them realistic. Focusing too much on realism may create a mode collapse, where the model outputs repetitive and low-variance sequences. Too much attention on diversity, however, can lead to unrealistic sequences that wander too far from the desired temporal patterns.

CONDITIONAL GENERATIVE ADVERSARAIL NETWORKS (cGANs):

Conditional Generative Adversarial Networks (cGANs) represent a sophisticated evolution of traditional Generative Adversarial Networks (GANs). In a standard GAN, the model comprises two components: a generator and a discriminator that are trained together to enhance each other's functions. In contrast, cGANs incorporate an additional element known as a conditioning variable, which supplies extra information to both the generator and the discriminator. This feature facilitates more precise data generation capabilities, allowing the model to create outputs that align with predefined attributes or conditions.



a) GAN architecture

b) CGAN architecture

Fig 1.1.2: cGAN

Structure of cGANs

The framework of cGANs maintains similarities with conventional GAN structures by including both a generator and a discriminator; however, it integrates an extra component—the conditioning variable. Typically represented as "y," this input can take various forms such as class labels, random noise, or other relevant domain-specific information. By incorporating this conditioning variable into both the generator and the discriminator's processes, cGANs ensure that output generation is directed or influenced by specified parameters.

Generator: Within the context of cGANs, the generator operates using inputs from both randomly generated noise vector (z) and the conditioning variable (y). As a result, it can produce outputs that not only reflect genuine data but also adhere to particular distributions or characteristics determined by y.

Discriminator: The role of the discriminator involves analyzing real data alongside either authentic samples or synthetic data produced by the generator while considering the conditioning variable (y). Its function expands considerably since it must evaluate whether data authenticity correlates accurately with designated conditions. Thus, assessing reality becomes more intricate for the discriminator due to its need for contextual evaluation against specific criteria.

Applications of Conditional Generative Adversarial Networks (cGANs)

Image-to-Image Translation: One of the primary applications of cGANs lies in image-to-image translation. This involves generating a modified version of an input image. For instance, a model can transform sketches into photographs or convert black-and-white images into colored ones. In this context, the conditioning could be the original image or specific attributes associated with it.

Super-Resolution: cGANs also play a pivotal role in super-resolution tasks where they create high-resolution images from their low-resolution counterparts. Here, the condition is typically represented by the initial low-resolution image, guiding the generator to enhance its resolution effectively.

Data Augmentation: Another valuable application is in data augmentation for machine learning models. When there is a scarcity of real-world data, cGANs can generate supplementary training samples that are conditioned on particular labels, thereby enriching datasets for specific categories.

Text-to-Image Generation: Additionally, cGANs facilitate text-to-image generation by producing visual content based on textual descriptions provided as conditions. The model generates imagery that corresponds closely to descriptive prompts like “a cat sitting on a mat.”

Challenges in cGANs

Mode Collapse: Applicable to traditional GANs, cGANs are susceptible to mode collapse, whereby a generator may provide limited variations of the output and skip some valid outputs conditioned by different inputs.

Training Instability: cGANs can also be unstable during training and may often lead to a scenario where the generator provides very poor quality outputs or unrealistic outputs. Some papers have suggested improved loss functions, normalization, & smarter training strategies to address these issues.

High Computational Costs: Training cGANs can be heavy on computation, especially with more complex data and larger datasets. The conditioning input requires extra processing and memory.

APPLICATIONS OF GANs

Generative Adversarial Networks (GANs), being extremely diverse and revolutionary, apply to many arenas and produce extraordinary advancements. The image generation of the most realistic images within the field of computer vision has made GANs a new era in the creating of photorealistic images so perfect as to be indistinguishable from real photographs. They are applied in upscaling an image by increasing the resolution, thus taking image quality from low or blurry to crisp and bright images. Another strong application is in completion or inference, whereby rough images-in the form of hand-sketched images or outlines-are transformed into high-quality and hyper-realistic images, as most creative and industrial design workflows realize their practical usefulness.

Apart from graphics, GANs are being massively applied for data augmentation: the generation of synthetic datasets for the training of ML models. This is important for cases where data acquisition is either expensive, scarce, or prone to privacy concerns. GANs allow the simulation of various datasets that augment higher model performance as generalization on tasks.

Furthermore, the creative industries have begun to adopt GANs to compose music, paint pictures, or design with a wealth of diversity there arises collaboration between man and machine. Music GANs can create melodies, harmonies, or whole new compositions; visual arts GANs generate paintings or sculptures modeled on specified styles or themes.

1.2 PROBLEM STATEMENT AND OBJECTIVE

We have proposed the combination of the two models i.e the BERT is an NLP model and the StackGAN is an Deep-learning model offers several significant advantages for image generation system.Mainly we are working on the Bird's Dataset to generate the Bird image based on the given Text by the user. BERT's exceptional ability to understand and represent the semantic meaning of text provides a strong foundation for image generation. By incorporating BERT-generated sentence embeddings, the StackGAN architecture can leverage deep semantic understanding of the text description.In StackGan basically the Architecture of this model is a hierarchical manner so we can able to generate the image for the Larger text Description.

So we combined this BERT and StackGan to generate the images aligned with the give text description Perfectly and to generate the High-quality realistic images.

1.3 SPECIFIC OBJECTIVES

The primary objective of this system is to develop a robust text-to-image generation system by integrating BERT and StackGAN models, enabling the creation of high-quality images that are semantically aligned with the input text. Specifically, the system aims to: (1) harness BERT's powerful language representation capabilities to extract meaningful and context-aware embeddings from textual descriptions, ensuring accurate interpretation of the text; (2) utilize StackGAN's hierarchical architecture to generate highly detailed and realistic images, even for complex and lengthy text inputs; (3) establish a seamless pipeline that combines these models to deliver efficient and scalable text-to-image synthesis; (4) achieve high performance in terms of both semantic consistency and visual quality, setting a benchmark for future research in multimodal AI systems. Through these objectives, the project seeks to expand the practical applications of text-to-image generation across diverse fields, such as creative industries, education, and personalized content creation.

1.4 SIGNIFICANCE AND MOTIVATION FOR OUR PROJECT

The significance of this System lies in its potential to revolutionize the way humans interact with AI by enabling the seamless transformation of textual descriptions into vivid, high-quality images. Text-to-image generation has broad implications across various domains, including creative design, media production, e-commerce, and education. For example, it can empower designers to rapidly prototype ideas, assist educators in visualizing abstract concepts, and enhance user experiences in online platforms by generating personalized content.

The motivation for this System stems from the growing demand for AI systems that can bridge the gap between language and vision, two of the most essential modalities of human communication. While existing text-to-image systems often struggle with capturing the nuances of complex textual descriptions, the integration of BERT and StackGAN offers a novel solution. BERT's deep semantic understanding ensures precise interpretation of text, while StackGAN's hierarchical design facilitates the creation of detailed and realistic visuals. By combining these advanced models, this project aspires to push the boundaries of multimodal AI, fostering innovations that make artificial intelligence more accessible, versatile, and impactful in everyday applications.

1.5 EXISTING SYSTEM

In the existing text-to-image generation systems, most approaches rely on either standalone generative models like GANs (Generative Adversarial Networks) or simplified NLP embeddings to convert textual descriptions into images. While these systems have made significant progress, they often face limitations in effectively capturing the nuanced semantics of complex or lengthy textual descriptions. Traditional GAN-based architectures typically focus on generating images based on shallow text embeddings, which lack a deep understanding of the text's contextual meaning. This often leads to misaligned or oversimplified visual representations that fail to reflect the richness of the input text.

Moreover, existing systems frequently struggle with scalability and generalization. For instance, models trained on specific datasets may perform poorly when exposed to new, unseen text descriptions or when tasked with generating images based on large or abstract textual inputs.

Furthermore, the generated images often lack fine-grained details and realism, particularly for intricate or descriptive text inputs, as these systems do not fully utilize hierarchical or context-aware frameworks.

1.6 PROPOSED SYSTEM

We have proposed the combination of the two models i.e the BERT is an NLP model and the StackGAN is an Deep-learning model offers several significant advantages for image generation system. BERT's exceptional ability to understand and represent the semantic meaning of text provides a strong foundation for image generation. By incorporating BERT-generated sentence embeddings, the StackGAN architecture can leverage deep semantic understanding of the text description. In StackGan basically the Architecture of this model is a Hierarichal manner so we can able to generate the image for the Larger text Description. So we combined this BERT and StackGan to generate the images aligned with the give Text description Perfectly and to generate the High-quality realistic images.

CHAPTER 2

LITERATURE SURVEY

2.1 ABOUT THE PROJECT

In this project, you will explore the integration of BERT, an NLP model, with StackGAN, a deep learning architecture, to generate high-quality, realistic images from text descriptions. The goal is to leverage BERT's powerful ability to understand and represent the semantic meaning of text, producing sentence embeddings that guide StackGAN in creating images. StackGAN's hierarchical structure allows it to generate images from complex text descriptions, first by creating low-resolution images and then refining them for high-resolution outputs. By combining these models, this project aims to generate images that are perfectly aligned with textual descriptions, offering a significant advancement in text-to-image synthesis. By the end of this project, you will learn how to combine natural language processing and deep learning techniques to generate realistic images from text, thus expanding the capabilities of generative models in creative and practical applications.

2.2 LITERATURE REVIEW

[1] Sutskever et al., 2014: This foundational work introduced the sequence-to-sequence (Seq2Seq) model, utilizing Long Short-Term Memory (LSTM) networks to map input sequences to output sequences effectively. It addressed challenges in machine translation, achieving state-of-the-art results on English-to-French translations in the WMT'14 dataset. The model demonstrated a BLEU score of 34.8, outperforming phrase-based systems. Key innovations included reversing input sequences to improve learning and utilizing beam search for decoding. This methodology laid the groundwork for handling sequential data in various domains.

[2] Reed et al., 2016: In this pioneering paper, Reed et al. introduce a novel method for generating images from textual descriptions using Generative Adversarial Networks (GANs). Their work involves the use of deep convolutional networks to process textual descriptions and translate them into visual representations. The paper demonstrates the use of a deep convolutional

GAN (DC-GAN), where both the generator and discriminator are conditioned on text encodings derived from recurrent neural networks (RNNs). The core contribution of the paper is the integration of text embeddings—which capture the semantic content of the description—into a GAN framework, leading to the generation of realistic images that match the provided text. For example, they show that their method can generate plausible images of birds and flowers from detailed textual descriptions, illustrating the potential for text-to-image synthesis.

[3] Mansimov et al., 2016: In this paper, Mansimov et al. introduce a novel approach for generating images from text descriptions using attention mechanisms**.** The key idea behind the method is the use of an iterative drawing process, where the model generates images by drawing patches on a canvas while simultaneously attending to relevant words in the input caption. This attention mechanism helps the model focus on specific parts of the caption that are critical for generating different parts of the image. The model, called alignDRAW, utilizes a deep recurrent attention-based generative model. The process involves using a recurrent neural network (RNN) to iteratively refine the image by drawing patches and attending to different parts of the caption at each step. The system is trained on the Microsoft COCO dataset, which contains a large number of images annotated with descriptive captions. The model is capable of generating images that align well with textual descriptions, producing results that can adapt to previously unseen captions, demonstrating a high degree of generalization.

[4] Odena et al., 2017: This paper introduces a variation of Generative Adversarial Networks (GANs) known as Auxiliary Classifier GANs (AC-GANs), which employs label conditioning to generate high-resolution, photorealistic images. AC-GANs improve image synthesis by incorporating an auxiliary classifier, which helps the generator produce images that are both class conditional and diverse. This model generates 128x128 resolution images with global coherence, significantly outperforming models that simply use resized low-resolution images..A key contribution is the introduction of two new image quality assessments: discriminability and diversity. The study shows that higher resolution images are more discriminable and provide better class information, as opposed to simply enlarging lower-resolution images. AC-GANs also exhibit a high degree of diversity across 1000 ImageNet classes, with results comparable to real ImageNet data.

[5] Sadia Ramzan: This paper explores the use of deep learning techniques, particularly GANs, for generating images from textual descriptions. The authors provide an overview of different architectures employed in text-to-image synthesis, such as StackGAN, AttnGAN, and other deep learning approaches. Their work emphasizes the use of deep convolutional networks and RNNs for encoding the text and generating corresponding visual outputs. The paper compares various model performances and highlights the challenges of maintaining semantic coherence and image quality.

[6] Wentong Liao et al. In this paper, the authors propose a semantic-spatial aware GAN (SSA-GAN) that improves the spatial accuracy and semantic coherence in text-to-image generation. The SSA-GAN model enhances the quality of generated images by incorporating semantic and spatial information to better align the text descriptions withthe generated visual output. This method significantly improves the model's ability to generate complex scenes and nuanced details in the images.the generated visual output. This method significantly improves the model's ability to generate complex scenes and nuanced details in the images.

[7] Han Zhang: This foundational work introduced the StackGAN model, a two-stage Generative Adversarial Network designed for text-to-image synthesis. Stage-I GAN generates low-resolution images (64x64) capturing basic shapes and colors from textual descriptions, while Stage-II GAN refines these into high-resolution, photo-realistic images (256x256). It outperformed existing single-GAN models by adopting a hierarchical structure, enabling better text-image alignment and stability during training. Key innovations include leveraging text-conditioned refinement and progressive generation to achieve superior visual fidelity. This methodology set a new benchmark for synthesizing high-quality images from textual descriptions.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS

- GPU (NVIDIA RTX 3070 or RTX 3090)
- Processor greater than intel core i5.
- RAM Greater than 8GB.
- Hard Disk more than or equal to 1TB

3.2 SOFTWARE REQUIREMENTS

- Windows.
- Python Libraries.
- Deep Learning Frameworks : Tensorflow, Keras, Pytorch
- Cloud Platform : Google Colab
- Natural Language Processing Library : NLTK, BERT
- serialization module in Python : Pickle
- GAN Specific Libraries : Tensorflow_GPU, Keras-GAN

CHAPTER 4
SYSTEM ARCHITECTURE

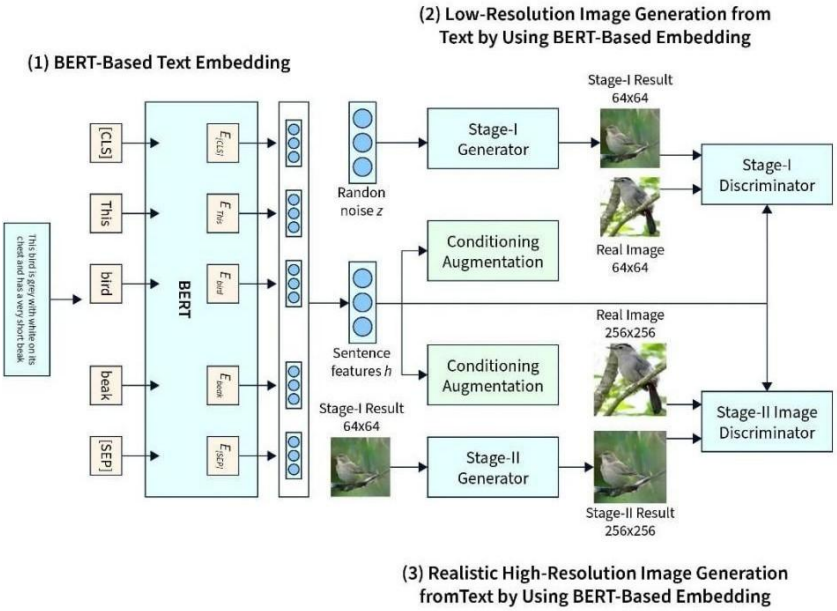


Fig 4.1 : System Architecture

1. BERT-Based Text Embedding

- **Input:** A text description, e.g., "This bird is grey with white on its chest and has a very short beak."
- **Process:**
    - ✧ **[CLS] Token:** The [CLS] token is added at the beginning of the input text. It serves as a summary representation of the entire sentence.
    - ✧ **Word Tokens:** The input sentence is tokenized into individual words, and each word is represented by an embedding (e.g., E_This, E_bird, E_beak).
    - ✧ **BERT Model:** BERT processes the sequence of token embeddings, outputting context-aware embeddings for each token.
    - ✧ **Sentence Feature Extraction:** The embedding corresponding to the [CLS] token is extracted as the **sentence feature vector (**h**)**, which encodes the semantic information of the input text.
- **Output:** The sentence feature vector (h) is passed to the next stage.

2. Low-Resolution Image Generation (Stage-I)

- **Input:**
  - ✧ Sentence feature vector (h) from BERT.
  - ✧ Random noise vector (z) to introduce diversity in image generation.
- **Conditioning Augmentation:**
  - ✧ Applies transformations to the sentence features to generate **conditioned embeddings** that are more robust and suitable for the GAN training process.
- **Stage-I Generator:**
  - ✧ Combines the conditioned embeddings with the random noise (z) to generate low-resolution images (64x64 pixels).
  - ✧ Focuses on generating basic object shapes and colors based on the text description.
- **Stage-I Discriminator:**
  - ✧ Evaluates the authenticity of the generated 64x64 images by distinguishing them from real images of the same resolution.
  - ✧ Also verifies whether the generated images align with the input text description.

3. Realistic High-Resolution Image Generation (Stage-II)

- **Input:**
  - ✧ Sentence feature vector (h).
  - ✧ Low-resolution image (64x64) generated by Stage-I.
- **Conditioning Augmentation (Stage-II):**
  - ✧ Similar to Stage-I, it refines the sentence feature vector to ensure better alignment with the high-resolution image generation task.

- **Stage-II Generator:**

  - ✧ Takes the low-resolution image and the refined text features to produce high-resolution images (256x256 pixels).
  - ✧ Refines details such as texture, patterns, and small structures to create photorealistic images.

- **Stage-II Discriminator:**

  - ✧ Evaluates the 256x256 generated images for realism and checks their semantic alignment with the text description.

Key Advantages of the Architecture

1. **Semantic Understanding:**

   - ✧ BERT provides high-quality sentence embeddings that capture nuanced meanings and contextual relationships in the text, ensuring the generated images align semantically with the descriptions.

2. **Hierarchical Generation (StackGAN):**

   - ✧ Stage-I provides a rough image structure, while Stage-II refines it to achieve photorealism.
   - ✧ This hierarchical approach improves training stability and overall image quality.

3. **Conditioning Augmentation:**

   - ✧ Introduces variability in the generated images while maintaining alignment with the input text, preventing overfitting and improving diversity.

CHAPTER 5

UML DIAGRAMS

5.1 INTRODUCTION TO UML

The Unified Modelling Language (UML) offers software engineers a standardized approach to visually represent an analysis model, governed by a set of rules that ensure proper syntax, semantics, and practicality. A UML system is visualized through unique views, each highlighting a different aspect of the system. These views are outlined as follows:

User Model View

- Focuses on the system from the user's standpoint.
- Describes usage scenarios to showcase how end-users interact with the system.

Structural Model View

- Highlights the static framework of the system.
- Represents internal data and functionality, emphasizing relationships and dependencies among components such as classes and objects.

Behavioral Model View

- Depicts the dynamic nature of the system.
- Illustrates interactions between structural elements, combining insights from the User and Structural Model Views to showcase workflows, object communications, and state changes.

Implementation Model View

- Represents the system's structural and behavioral aspects as they are meant to be implemented.
- Serves as a guide for developers, outlining how the system's components will be constructed.

5.2 UML DIAGRAMS

5.2.1 USE CASE DIAGRAM:

When modeling a system, it is crucial to focus on capturing its dynamic behavior, which refers to how the system behaves when it is running or actively functioning. To elaborate, dynamic behavior refers to the system's activities, interactions, and state changes during operation. Use case diagrams play a key role in eliciting requirements for a system, including both internal and external factors that influence the system. These requirements are typically related to the design phase of the system. When analyzing a system to determine its functionalities, use cases are created, and actors—both internal and external—are identified.

To summarize, the main purposes of use case diagrams can be described as:

a. To gather the functional requirements of a system.

b. To obtain an external perspective of how the system interacts with its environment.

c. To identify the external and internal elements that impact the system's behavior and performance.

In essence, use case diagrams help in understanding the interactions, behaviors, and requirements of the system from a high-level viewpoint, providing valuable insights for the design and development phases.

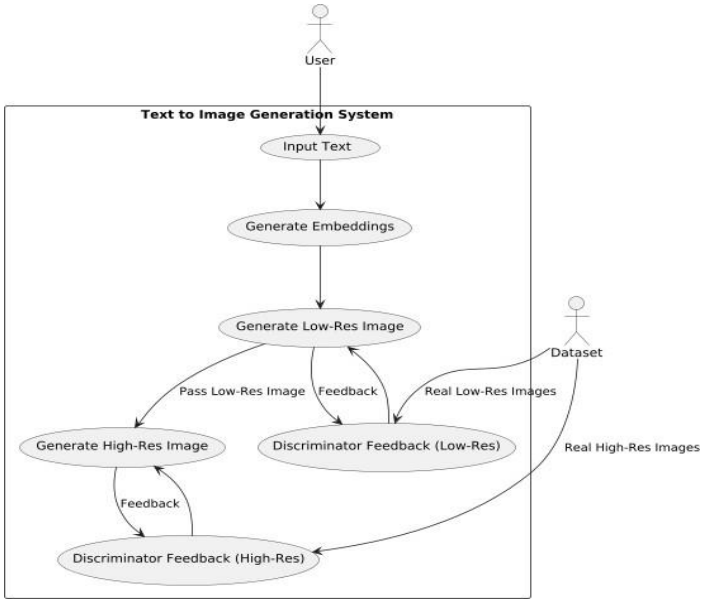


Fig 5.1: Use Case Diagram

5.2.2 ACTIVITY DIAGRAM

The activity diagram is another essential diagram in UML used to represent the dynamic behavior of a system. Essentially, the activity diagram functions as a flowchart, illustrating how the system transitions from one activity to another. An activity in this context refers to an operation or task that the system performs.

The key purposes of an activity diagram can be outlined as:

- Represent the flow of activities within the system.
- Depict the sequence in which activities occur, from one step to the next.
- Illustrate parallel, conditional, and concurrent flows, showing how different activities can happen simultaneously or branch off based on conditions.

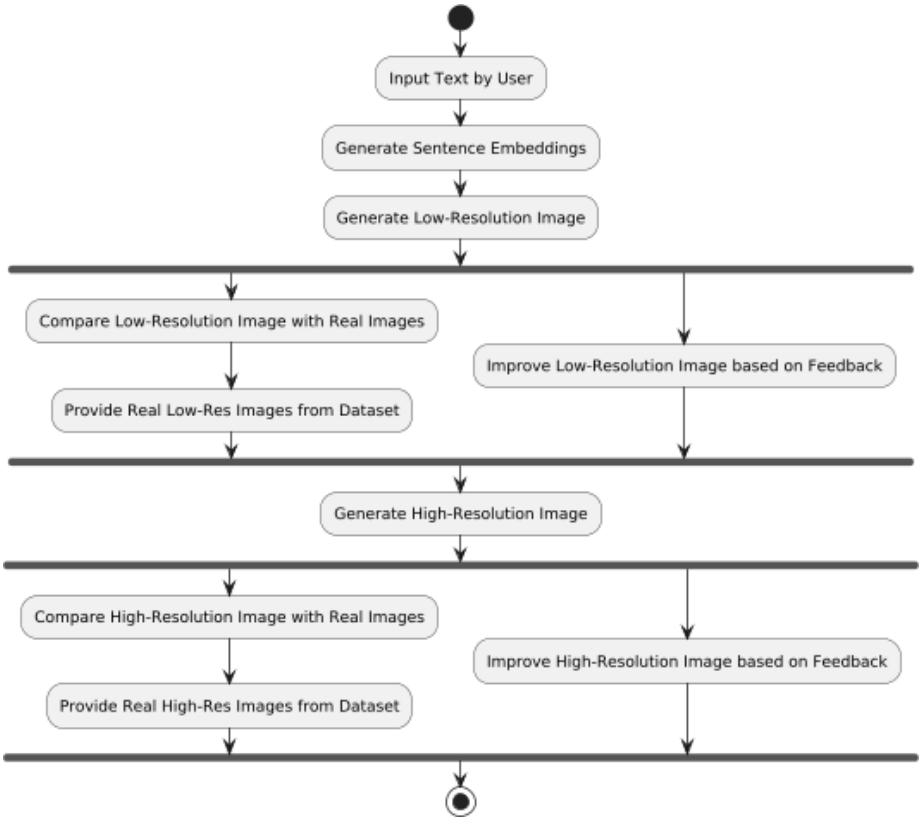


Fig 5.2: Activity Diagram

5.2.3 SEQUENCE DIAGRAM

Sequence diagrams are used to represent interactions between classes or objects in terms of message exchanges over time. Often referred to as event diagrams, sequence diagrams are highly effective in visualizing and validating various runtime scenarios. They help in understanding how the system will behave during execution and can be instrumental in uncovering the responsibilities that a class should assume when designing a new system.

The primary purposes of sequence diagrams can be summarized as:

- Visualize the interaction flow between objects or classes over time.
- Validate different runtime scenarios, illustrating how the system components collaborate in real-time.
- Identify the responsibilities of each class or object involved in the interaction during system design.

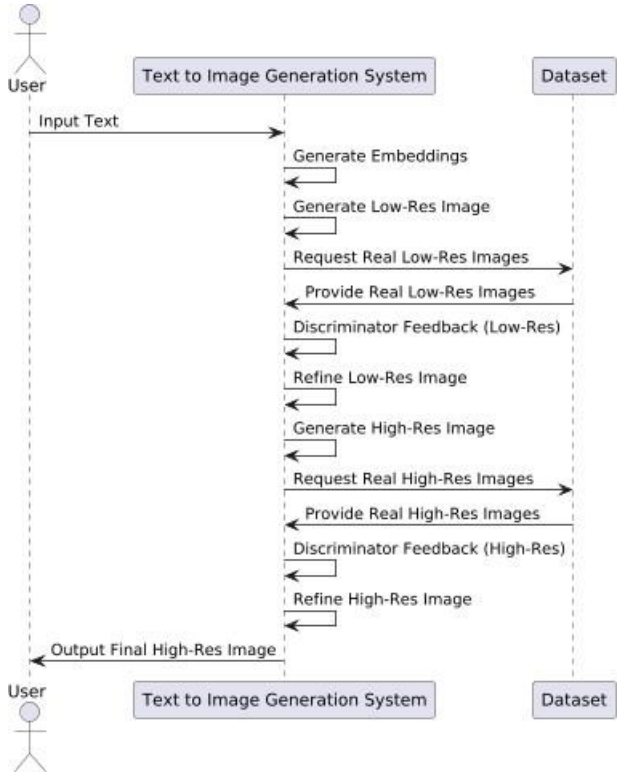


Fig 5.3: Sequence Diagram

5.2.4 STATE CHART DIAGRAM

Statechart diagrams, also known as state machine diagrams, are used to describe the dynamic behavior of an object in response to different events or stimuli. They show how an object transitions from one state to another based on internal or external events. These diagrams are particularly useful for modeling the lifecycle of an object or an entity, tracking its state changes and how it reacts to different conditions.

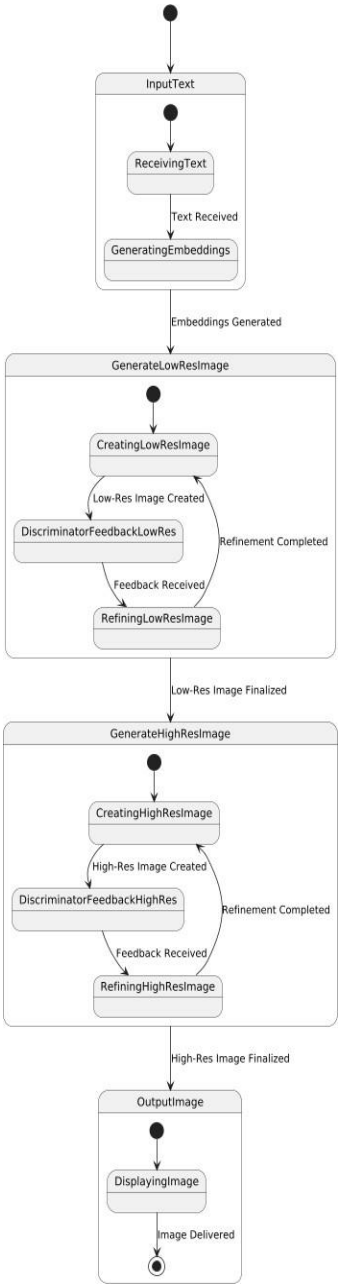


Fig 5.4: State Chart Diagram

5.2.5 OBJECT DIAGRAM

An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them.Object diagrams are a crucial aspect of UML, providing a snapshot of the instances in a system at a particular moment. They help in visualizing the relationships between objects. An object diagram in UML is useful because it provides a clear and visual representation of specific instances of classes and their relationships at a particular point in time, aiding in understanding and communicating the structure and interactions within a system. In other words, "An object diagram in the Unified Modeling Language (UML), is a diagram that shows a complete or partial view of the structure of a modeled system at a specific time.

The use of object diagrams is limited, mainly to show examples of data structures. During the analysis phase of a project, you might create a class diagram to describe the structure of a system and then create a set of object diagrams as test cases to verify the accuracy and completeness of the class diagram.

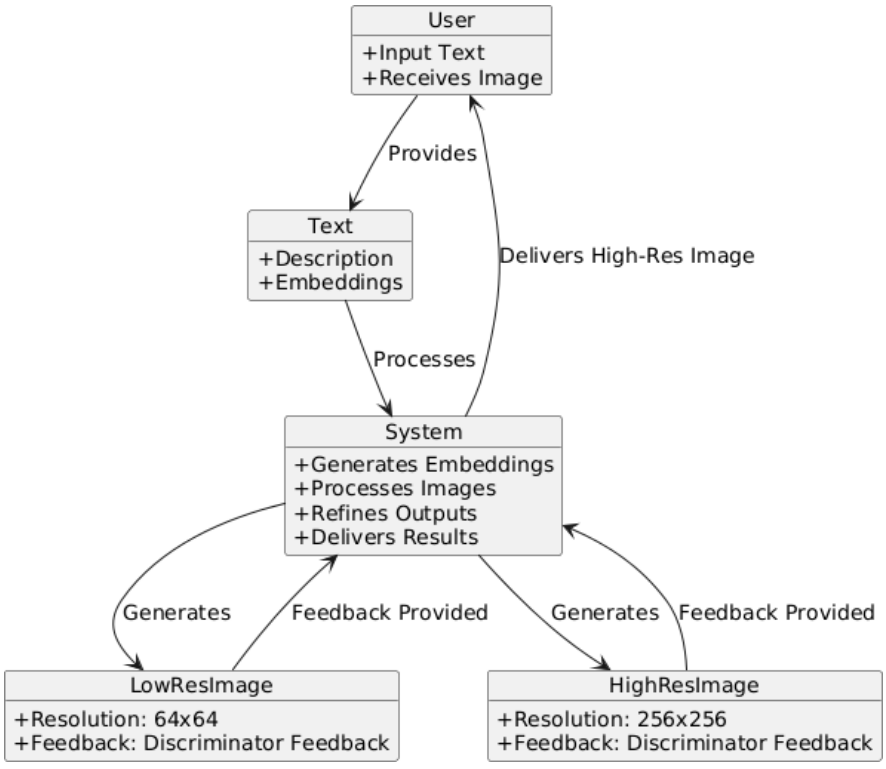


Fig 5.5: Object diagram

5.2.6 DEPLOYMENT DIAGRAM

A UML deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams is a kind of structure diagram used in modeling the physical aspects of an object-oriented system. They are often be used to model the static deployment view of a system (topology of the hardware).

- They show the structure of the run-time system
- They capture the hardware that will be used to implement the system and the links between different items of hardware.
- They model physical hardware elements and the communication paths between them
- They can be used to plan the architecture of a system.
- They are also useful for Document the deployment of software components or nodes

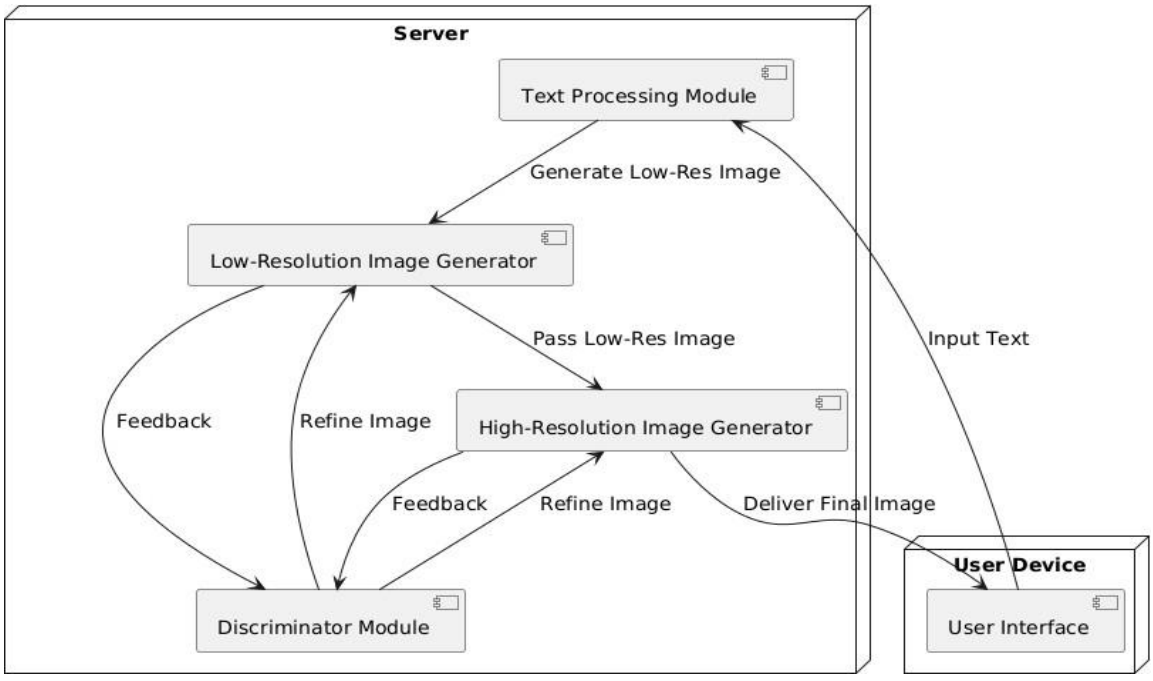


Fig 5.6: Deployment diagram

5.2.7 COMPONENT DIAGRAM

UML Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.

UML Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering.

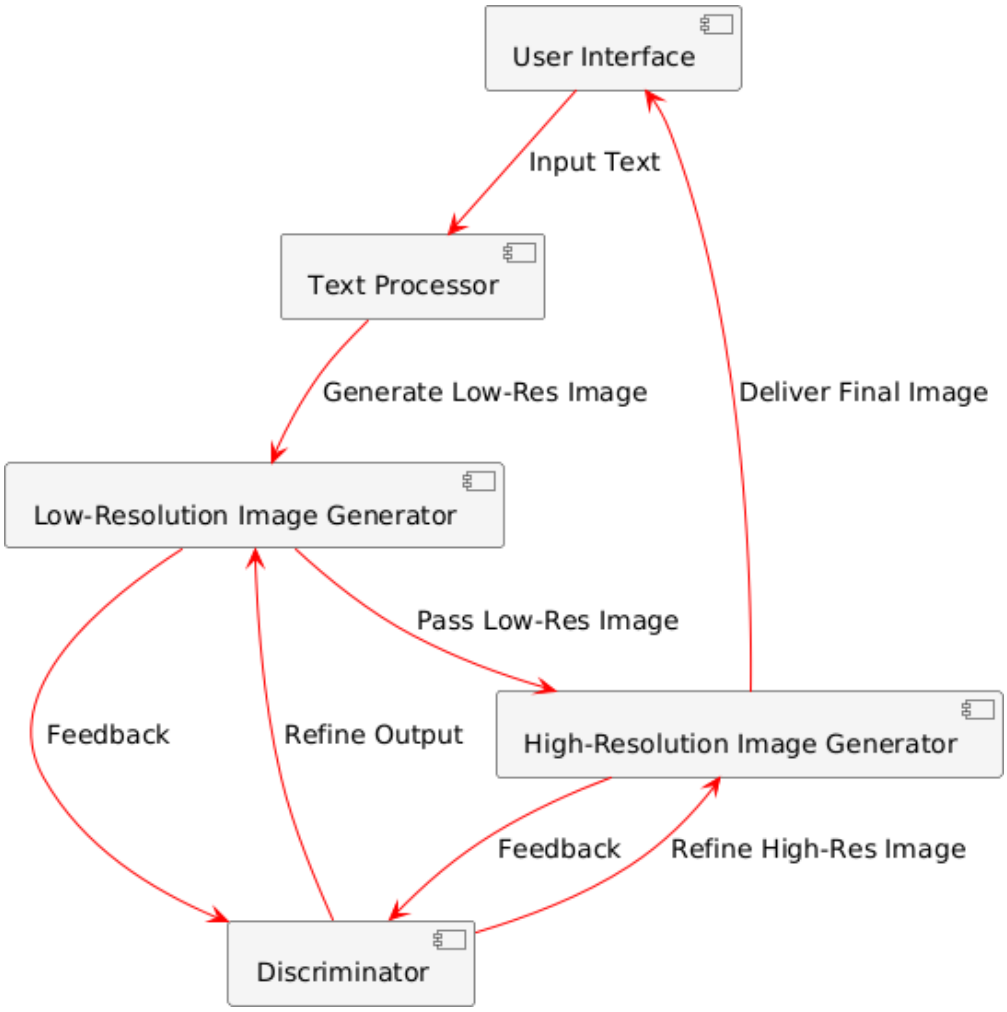


Fig 5.7: Component diagram

5.2.8 COLLABORATION DIAGRAM

Collaboration diagrams (known as Communication Diagram in UML 2.x) are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case. Along with sequence diagrams, collaboration are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. They are the primary source of information used to determining class responsibilities and interfaces.

It mainly puts emphasis on the structural aspect of an interaction diagram, i.e., how lifelines are connected.

- The syntax of a collaboration diagram is similar to the sequence diagram; just the difference is that the lifeline does not consist of tails.
- The messages transmitted over sequencing is represented by numbering each individual message.
- The collaboration diagram is semantically weak in comparison to the sequence diagram.

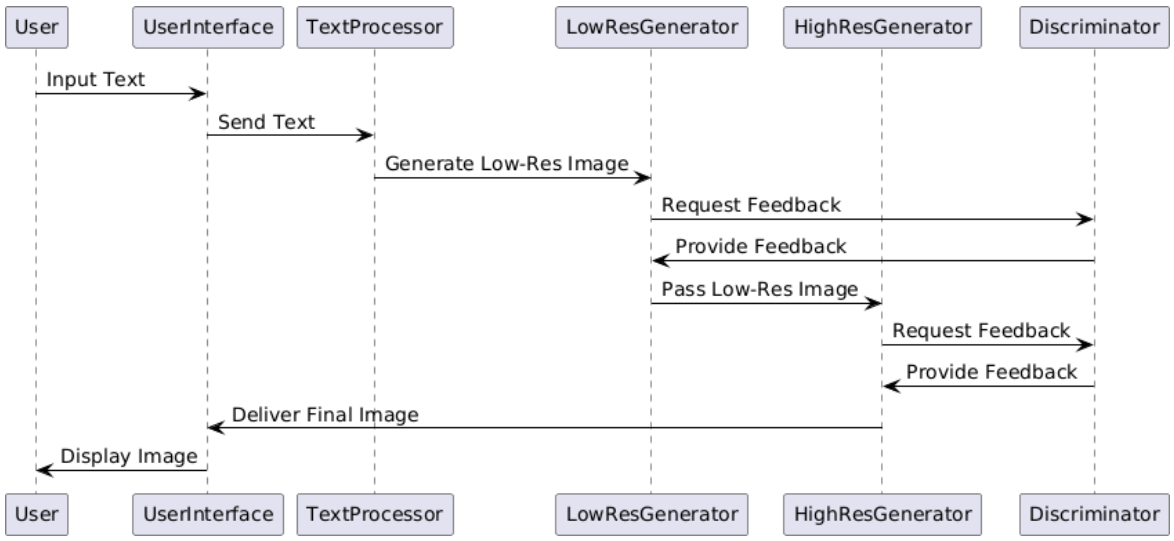


Fig 5.8: Collaboration diagram

5.3 CLASS DIAGRAM

Class diagrams are used to represent the static structure of a system by showing its classes, attributes, operations, and the relationships between objects. These diagrams are essential for understanding how a system is organized and how the different components or classes interact with each other. Class diagrams are foundational in object-oriented modeling and help in both the design and implementation phases.

The key purposes of class diagrams can be outlined as:

- Represent the static structure of the system, detailing the classes and their relationships.
- Illustrate the attributes and operations of each class, helping to define the behavior and data associated with them.
- Show the relationships between classes, such as associations, inheritance, and dependencies, enabling a clear view of how the system's components are connected.

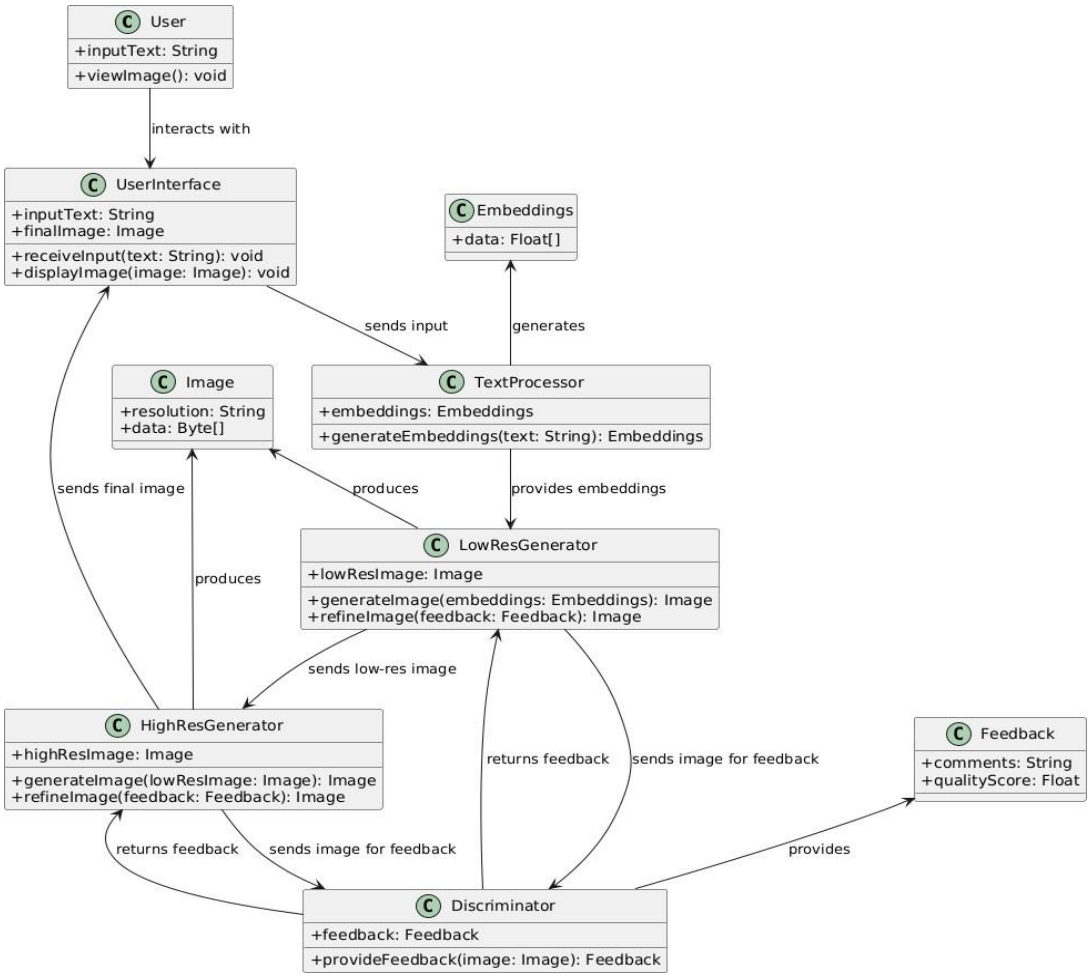


Fig 5.10: Class Diagram

5.4 ALGORITHM

Bert-Embedding Algorithm:

1: Initialize parameters pre-trained BERT θ_{BERT}

2: Set number of fine-tuning step Sf , descriptions of datasets $T = \{t_1 , t_2, \ldots , t_n\}$

3: **For** $f = 1, \ldots , S_f$: **do**

4: Perform *step of adam update on* θ_{BERT} with T

5: **End for**

StackGan-Algorithm:

Stage 1

Let x_{row} be the low-resolution image, z be the random vector, D_1 be the discriminator of stage 1, G_1 be the generator, h be the text embedding of the provided text description via fine-tuned BERT, hca be the text embedding taken from the Gaussian conditioning variable, with λ serving as the regularization parameter. The stage 1 generator is taught to reduce the loss function of Equation (2), whereas the stage 1 discriminator is trained to maximize Equation (1). Equation (2)'s regularization term, the KL divergence, $D_{KL}(N(\mu_0(h), \sum_0(h))||(0, X_{row}))$, recovers latent text representation from an independent Gaussian distribution. Randomness is introduced by sampling the Gaussian conditioning variable h_{ca} from $N(\mu_0(h), \sum_0(h))$.

$$LD_1 = E(x_{row},h)\sim P_{data}\,[\log D_1(x_{row}, h)] + E_{z\sim pz,h\sim pdata}\,[\log(1 - D_1(G_1(z, h_{ca}), h))] \quad \text{------(1)}$$

$$LG_1 = E_{z\sim pz,h\sim pdata}\,[\log(1 - D_1(G_1(z, h_{ca}), h))] + \lambda D_{KL}(N(\mu_0(h), \sum 0(h))||N(0, X_{row})) \quad \text{-----(2)}$$

Algorithm 2 Low-Resolution Image Generation from Text by Using BERT-Based Embedding

Initialize parameters of stage1 generator and discriminator G_1, D_1
Set fine-tuned BERT $\theta_{BERT}^{fine-tuned}$, number of stage1 step S_1, and G_1, D_1 learning rates α, β
text descriptions and images of dataset $T = \{t_1, t_2, \ldots, t_n\}$, $X = \{x_1, x_2, \ldots, x_n\}$
For $step = 1, \ldots, S_1$ **do**
    For $i = 1, \ldots, n$ **do**
        $z \sim \mathcal{N}(0, 1)$ # *sample of random noise*
        $h \leftarrow \theta_{BERT}^{fine-tuned}(t_i)$ # *sample of matching text description*
        $h' \leftarrow \theta_{BERT}^{fine-tuned}(t_i')$ # *sample of miss − matching text description*
        $h_{ca} \leftarrow ca(h)$ # *conditioning augmentation of matching text description*
        $h'_{ca} \leftarrow ca(h')$ # *conditioning augmentation of miss − matching text description*
        $\hat{x}_{row} \leftarrow G_1(z, h_{ca})$ # *generated image from text description*
        $s^r_{row} \leftarrow D_1(x_{row}, h_{ca})$ # *real image, right text description*
        $s^w_{row} \leftarrow D_1(x_{row}, h'_{ca})$ # *real image, wrong text description*
        $s^w_{row} \leftarrow D_1(\hat{x}_{row}, h_{ca})$ # *fake image, right text description*
        $D_1 \leftarrow D_1 - \alpha\partial\mathcal{L}_{D_1}/\partial D_1$ # *update stage*1 *discriminator* $\mathcal{L}_{D_1} = equation\ (1)$.
        $G_1 \leftarrow G_1 - \beta\partial\mathcal{L}_{G_1}/\partial G_1$ # *update stage*1 *Generator* $\mathcal{L}_{G_1} = equation\ (2)$.
    End for
End for

Fig 5.11: Algorithm-1

Stage 2

Assume G_2 is the stage 2 generator and D_2 is the stage 2 discriminator. Equation (3) is maximized by training the stage 2 discriminator, and Equation (4) is minimized by training the stage 2 generator. Only in stage 1 is random noise utilized, not at this point. Rather, stage 1 X_{row} uses a low-resolution image produced by the generator. By entering the low resolution image produced by the stage1 generator and the text embedding vector taken from the text encoder, a high resolution image, X_{high}, is produced. By entering an image and text embedding vector, the discriminator ascertains whether the image and text match.

$$LD_2 = \mathrm{E}(X_{row}, h)\sim P_{data}\,[\log D_2(x, h)] + \mathrm{E}x_{row}\sim p_{G1},\, h\sim p_{data}\,[\log(1 - D_2(G_2(x_{row}, h_{ca}),h))] \quad --(3)$$

$$LG_2 = \mathrm{E}x_{row}\sim p_{G1}\,,\, h\sim p_{data}\,[\log(1 - D_2(G_2(x_{row}, h_{ca}), h))] + \lambda D_{KL}(N\,(\mu_0(h),\textstyle\sum(h))\|N\,(0, X)) \quad --(4)$$

Algorithm 3 Realistic High-Resolution Image Generation from Text by Using BERT-Based Embedding

Initialize parameters of stage2 generator and discriminator $G_2,\ D_2$
Set fine-tuned BERT $\theta_{BERT}^{fine-tuned}$, number of stage2 step S_2, and $G_2,\ D_2$ learning rates γ, ω text descriptions and images of dataset $T = \{t_1,\ t_2,\ \ldots,\ t_n\},\ X = \{x_1,\ x_2,\ \ldots,\ x_n\}$
For $step = 1,\ \ldots,\ S_2$ **do**
    For $i = 1,\ \ldots,\ n$ **do**
        $z \sim \mathcal{N}(0,\ 1)$ # *sample of random noise*
        $h \leftarrow \theta_{BERT}^{fine-tuned}(t_i)$ # *sample of matching text description*
        $h' \leftarrow \theta_{BERT}^{fine-tuned}(t_i')$ # *sample of miss − matching text description*
        $h_{ca} \leftarrow ca(h)$ # *conditioning augmentation of matching text description*
        $h'_{ca} \leftarrow ca(h')$ # *conditioning augmentation of miss − matching text description*
        $\hat{x}_{row} \leftarrow G_1(z, h_{ca})$ *generated row resolution images from stage*1 *generator*
        $\hat{x}_{high} \leftarrow G_2(\hat{x}_{row},\ h_{ca})$ # *generated image from text description*
        $s^r_{high} \leftarrow D_2\left(x_{high},\ h_{ca}\right)$ # *real image, right text description*
        $s^w_{high} \leftarrow D_2\left(x_{high},\ h'_{ca}\right)$ # *real image, wrong text description*
        $s^w_{high} \leftarrow D_2\left(\hat{x}_{high},\ h_{ca}\right)$ # *fake image, right text description*
        $D_2 \leftarrow D_2 - \gamma \partial \mathcal{L}_{D_2} / \partial D_2$ # *update stage*1 *discriminator* $\mathcal{L}_{D_2} = equation\ (3)$.
        $G_2 \leftarrow G_2 - \omega \partial \mathcal{L}_{G_2} / \partial G_2$ # *update stage*1 *Generator* $\mathcal{L}_{G_2} = equation\ (4)$.
    End for
End for

Fig 5.12: Algorithm-2

CHAPTER 6
CODE IMPLEMENTATION

6.1 CODE SAMPLE

Imports

```
import os
import pickle
import random
import time

import PIL
import numpy as np
import pandas as pd
import tensorflow as tf
from PIL import Image
from keras import Input, Model
from keras import backend as K
from keras.callbacks import TensorBoard
from keras.layers import Dense, LeakyReLU, BatchNormalization, ReLU, Reshape, UpSampling2D, Conv2D,
Activation, \
    concatenate, Flatten, Lambda, Concatenate
from tensorflow.keras.optimizers import Adam
from matplotlib import pyplot as plt
```

Loading of Dataset

```
def load_dataset(filenames_file_path, class_info_file_path, cub_dataset_dir, embeddings_file_path,
image_size):
    """
    Load dataset
    """
    filenames = load_filenames(filenames_file_path)
    class_ids = load_class_ids(class_info_file_path)
    bounding_boxes = load_bounding_boxes(cub_dataset_dir)
    all_embeddings = load_embeddings(embeddings_file_path)

    X, y, embeddings = [], [], []
```

```
    print("Embeddings shape:", all_embeddings.shape)
```

```
    for index, filename in enumerate(filenames):
        bounding_box = bounding_boxes[filename]
```

```
        try:
            # Load images
            img_name = '{}/images/{}.jpg'.format(cub_dataset_dir, filename)
            img = get_img(img_name, bounding_box, image_size)

            all_embeddings1 = all_embeddings[index, :, :]

            embedding_ix = random.randint(0, all_embeddings1.shape[0] - 1)
            embedding = all_embeddings1[embedding_ix, :]

            X.append(np.array(img))
            y.append(class_ids[index])
            embeddings.append(embedding)
        except Exception as e:
            print(e)

    X = np.array(X)
    y = np.array(y)
    embeddings = np.array(embeddings)
    return X, y, embeddings
```

Drive Link:
https://drive.google.com/drive/folders/1ypkZfJrjaDbh7PyKnZJLM8qC1AFbF_ql?usp=drive_link

Main File:

```
if __name__== '__main__':
    data_dir = "/content/drive/MyDrive/MiniPrjt-GAN/birds/birds"
    train_dir = data_dir + "/train"
    test_dir = data_dir + "/test"
    image_size = 64
    batch_size = 64
    z_dim = 100
    stage1_generator_lr = 0.0002
    stage1_discriminator_lr = 0.0002
    stage1_lr_decay_step = 600
    epochs = 10      #changed
    condition_dim = 128

    embeddings_file_path_train = train_dir + "/char-CNN-RNN-embeddings.pickle"
    embeddings_file_path_test = test_dir + "/char-CNN-RNN-embeddings.pickle"
```

```
    filenames_file_path_train = train_dir + "/filenames.pickle"
    filenames_file_path_test = test_dir + "/filenames.pickle"

    class_info_file_path_train = train_dir + "/class_info.pickle"
    class_info_file_path_test = test_dir + "/class_info.pickle"

    cub_dataset_dir = "/content/drive/MyDrive/MiniPrjt-GAN/CUB_200_2011"

    # Define optimizers
    dis_optimizer = Adam(learning_rate=stage1_discriminator_lr, beta_1=0.5, beta_2=0.999)
    gen_optimizer = Adam(learning_rate=stage1_generator_lr, beta_1=0.5, beta_2=0.999)

    """
    Load datasets
    """
    X_train, y_train, embeddings_train = load_dataset(filenames_file_path=filenames_file_path_train,
                                                      class_info_file_path=class_info_file_path_train,
                                                      cub_dataset_dir=cub_dataset_dir,
                                                      embeddings_file_path=embeddings_file_path_train,
                                                      image_size=(64, 64))

    X_test, y_test, embeddings_test = load_dataset(filenames_file_path=filenames_file_path_test,
                                                   class_info_file_path=class_info_file_path_test,
                                                   cub_dataset_dir=cub_dataset_dir,
                                                   embeddings_file_path=embeddings_file_path_test,
                                                   image_size=(64, 64))

    """
    Build and compile networks
    """
    ca_model = build_ca_model()
    ca_model.compile(loss="binary_crossentropy", optimizer="adam")

    stage1_dis = build_stage1_discriminator()
    stage1_dis.compile(loss='binary_crossentropy', optimizer=dis_optimizer)

    stage1_gen = build_stage1_generator()
    stage1_gen.compile(loss="mse", optimizer=gen_optimizer)

    embedding_compressor_model = build_embedding_compressor_model()
    embedding_compressor_model.compile(loss="binary_crossentropy", optimizer="adam")

    adversarial_model = build_adversarial_model(gen_model=stage1_gen, dis_model=stage1_dis)
```

```
    adversarial_model.compile(loss=['binary_crossentropy', KL_loss], loss_weights=[1, 2.0],
                              optimizer=gen_optimizer, metrics=None)

    tensorboard = TensorBoard(log_dir="logs/".format(time.time()))
    tensorboard.set_model(stage1_gen)
    tensorboard.set_model(stage1_dis)
    tensorboard.set_model(ca_model)
    tensorboard.set_model(embedding_compressor_model)

    # Generate an array containing real and fake values
    # Apply label smoothing as well
    real_labels = np.ones((batch_size, 1), dtype=float) * 0.9
    fake_labels = np.zeros((batch_size, 1), dtype=float) * 0.1

    for epoch in range(epochs):
        print("==========================================")
        print("Epoch is:", epoch)
        print("Number of batches", int(X_train.shape[0] / batch_size))

        gen_losses = []
        dis_losses = []

        # Load data and train model
        number_of_batches = int(X_train.shape[0] / batch_size)
        for index in range(number_of_batches):
            print("Batch:{}".format(index+1))

            """
            Train the discriminator network
            """
            # Sample a batch of data
            z_noise = np.random.normal(0, 1, size=(batch_size, z_dim))
            image_batch = X_train[index * batch_size:(index + 1) * batch_size]
            embedding_batch = embeddings_train[index * batch_size:(index + 1) * batch_size]
            image_batch = (image_batch - 127.5) / 127.5

            # Generate fake images
            fake_images, _ = stage1_gen.predict([embedding_batch, z_noise], verbose=3)

            # Generate compressed embeddings
            compressed_embedding = embedding_compressor_model.predict_on_batch(embedding_batch)
            compressed_embedding = np.reshape(compressed_embedding, (-1, 1, 1, condition_dim))
            compressed_embedding = np.tile(compressed_embedding, (1, 4, 4, 1))
```

```
            dis_loss_real = stage1_dis.train_on_batch([image_batch, compressed_embedding],
                                                      np.reshape(real_labels, (batch_size, 1)))
            dis_loss_fake = stage1_dis.train_on_batch([fake_images, compressed_embedding],
                                                      np.reshape(fake_labels, (batch_size, 1)))
            dis_loss_wrong = stage1_dis.train_on_batch([image_batch[:(batch_size - 1)],
compressed_embedding[1:]],
                                                       np.reshape(fake_labels[1:], (batch_size-1, 1)))

             d_loss = 0.5 * np.add(dis_loss_real, 0.5 * np.add(dis_loss_wrong, dis_loss_fake))

            print("d_loss_real:{}".format(dis_loss_real))
            print("d_loss_fake:{}".format(dis_loss_fake))
            print("d_loss_wrong:{}".format(dis_loss_wrong))
            print("d_loss:{}".format(d_loss))
            """
            Train the generator network
            """
            g_loss = adversarial_model.train_on_batch([embedding_batch, z_noise,
compressed_embedding],[K.ones((batch_size, 1)) * 0.9, K.ones((batch_size, 256)) * 0.9])
            print("g_loss:{}".format(g_loss))

            dis_losses.append(d_loss)
            gen_losses.append(g_loss)

        """
        Save losses to Tensorboard after each epoch
        """
        write_log(tensorboard, 'discriminator_loss', np.mean(dis_losses), epoch)
        write_log(tensorboard, 'generator_loss', np.mean(gen_losses[0]), epoch)

        # Generate and save images after every 2nd epoch
        if epoch % 2 == 0:
            # z_noise2 = np.random.uniform(-1, 1, size=(batch_size, z_dim))
            z_noise2 = np.random.normal(0, 1, size=(batch_size, z_dim))
            embedding_batch = embeddings_test[0:batch_size]
            fake_images, _ = stage1_gen.predict_on_batch([embedding_batch, z_noise2])

            # Save images
            for i, img in enumerate(fake_images[:10]):
                save_rgb_img(img, "results/gen_{}_{}.png".format(epoch, i))

    # Save models
    stage1_gen.save_weights("stage1_gen.h5")
    stage1_dis.save_weights("stage1_dis.h5")
```

6.2 DATA SET SAMPLE

We have used the data-set from the kaggle website.

The link is below: https://www.kaggle.com/datasets/xiaojiu1414/cub-200-2011/data

CUB_200_2011.The CUB data-set consists of 200 classes. It consists of 11,788 images of birds, each with 10 textual descriptions. In about 80% of the CUB dataset, the proportion of the object is less than half the image size. In the experiment of this paper, preprocessing was performed so that the ratio of the object was greater than 0.75 by using a bounding box for the object.

6.3 FINAL OUTPUT

Provided Text: "A bird with a medium orange bill white body gray wings and webbed feet"

Fig-17 Generated Image-1

Provided Text: "This small bird has a white breast, light grey head, and black wings and tail"

Fig-18 Generated Image-2

CHAPTER 7

TESTING

7.1 Test Plan Overview

Testing in this project ensures that the generated images align with textual descriptions both semantically and visually. The primary focus is on evaluating the model's accuracy, image quality, coherence, and performance under different scenarios.

Objectives of Testing:

1. **Correctness** – Ensure generated images correctly reflect input text descriptions.
2. **Quality** – Assess image resolution, clarity, and realism.
3. **Performance** – Measure inference time, system resource utilization, and efficiency.
4. **Robustness** – Verify the system's handling of ambiguous, long, or out-of-distribution inputs.
5. **Comparative Evaluation** – Compare model performance with baseline methods.

Functional Testing

Objective: Evaluate the system's ability to generate accurate bird images from textual descriptions.

- **Basic Bird Descriptions:** Test with simple inputs (e.g., "A red parrot with a yellow beak").
- **Complex Bird Features:** Check for accuracy in descriptions with multiple attributes (e.g., "A green hummingbird with iridescent feathers and a long beak").
- **Rare & Hybrid Birds:** Assess performance on unusual inputs (e.g., "A blue owl with golden wings").

Performance Testing

Objective: Measure efficiency, speed, and resource utilization.

- **Inference Time:** Ensure images are generated within <5 seconds.
- **Memory Consumption:** Monitor GPU usage, preventing excessive overhead.
- **Batch Processing:** Test system performance when generating multiple bird images simultaneously.

User Evaluation

Objective: Collect human feedback to assess realism and semantic accuracy.

- **Image Realism:** Evaluate if generated birds appear lifelike and natural.
- **Feature Accuracy:** Compare images to descriptions to ensure correct attributes (e.g., feather color, beak shape).

- **Comparative Study:** Compare generated images with real bird photos to measure similarity.

7.2 Test Case

Test Case-1

A small bird with varying shades of brown with white under the eyes

Fig-19 Generated Image-3

Test Case-2

The bird is short and stubby with yellow on its body

Fig-20 Generated Image-4

CHAPTER 8
FUTURE ENHANCEMENT AND CONCLUSION

8.1 FUTURE ENHANCEMENT

This project can be enhanced in several ways to improve its capabilities and broaden its applications. Future work could focus on incorporating more advanced language models, such as GPT-based embeddings or multimodal frameworks, to better understand complex and abstract text descriptions. The system could also be extended to accept additional modalities, like audio or sketches, for richer inputs. Producing ultra-high-resolution images (e.g., 1024x1024 or beyond) using newer architectures like StyleGAN or diffusion models could further refine image quality. Additionally, enabling real-time image generation would make the system suitable for interactive applications, such as gaming and virtual reality. Domain-specific customization for industries like e-commerce, healthcare, or education could expand its practical use cases, while features to handle complex scenes and object interactions described in text would add versatility. The integration of explainable AI techniques to visualize how text influences the generated images, along with reinforcement learning mechanisms for incorporating user feedback, would enhance interpretability and user experience. Finally, deploying the model on cloud platforms would ensure scalability and accessibility for a broader user base.

8.2 CONCLUSION

The proposed methodology is used to create StackGAN model which is a deep learning model that generates photo-realistic images from text descriptions using a two-stage process. First, it creates a low-resolution sketch based on the text. Then, it refines the sketch with intricate details and colors to produce a high-resolution image. By incorporating "BERT Text Embeddings ," StackGAN ensures the generated images accurately reflect the input text, leading to significant improvements in image quality, realism, and diversity.

CHAPTER 9
REFERENCES

[1] Sequence to Sequence Learning with Neural Networks,(Sutskever et al., 2014).

[2] Generative Adversarial Text-to- Image Synthesis,(Reed et al., 2016).

[3] Generating Images from Captions with Attention,(Mansimov et al., 2016).

[4] Conditional Image Synthesis with Auxiliary Classifier GANs.

[5] Text-to-Image Generation Using Deep Learning,(Sadia Ramzan 1,*)

[6] Text-to-Image Generation Using Gan,(Mrs. L. Indira, M. Sunil et al.).

[7] Text to Image Generation with Semantic-Spatial Aware GAN,(Wentong Liao, Kai Hu).

[8] StackGAN: Text to Photorealistic Image Synthesiswith StackedGenerative Adversarial Networks

Annexure

International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Text to image generation using BERT and GAN

Kavitha Soppari, Bhanu Vangapally *, Syed Sameer Sohail and Harish Dubba

Department of CSE (Artificial Intelligence and Machine Learning), ACE Engineering College, India.

International Journal of Science and Research Archive, 2025, 14(01), 720-725
Publication history: Received on 04 December 2024; revised on 13 January 2025; accepted on 15 January 2025
Article DOI: https://doi.org/10.30574/ijsra.2025.14.1.0137

Abstract

Generating text to images is a difficult task that combines natural language processing and computer vision. Currently available generative adversarial network (GAN)-based models usually employ text encoders that have already been trained on image-text pairs. Nevertheless, these encoders frequently fall short in capturing the semantic complexity of unread text during pre-training, which makes it challenging to produce images that accurately correspond with the written descriptions supplied.Using BERT, a very successful pre-trained language model in natural language processing, we present a novel text-to-image generating model in order to address this problem. BERT's aptitude for picture generating tasks is improved by allowing it to encode rich textual information through fine-tuning on a large text corpus. Results from experiments on a CUB_200_2011 dataset show that our approach performs better than baseline models in both qualitative and quantitative measures.

Keywords: Text to image generation; Multimodal data; BERT; GAN; High quality

Introduction

Since the actual world is intrinsically multimodal, research in multimodal deep learning is essential to the advancement of artificial intelligence, even if many deep learning techniques were created for single-modality problems. One important example of multimodal learning is text-to-picture generation, where combining text and image modalities presents special difficulties. Frequently, images come with tags or descriptions that explain their significance and offer context. However, because the input (text) and output (picture) have essentially distinct properties, learning to create images from text is intrinsically difficult. For text-to-image generation to be effective, three key issues must be resolved. Learning text representations that highlight visually meaningful characteristics is crucial first. Second, these language features need to be used to create realistic-looking, high-quality graphics.Third, for previously encountered text during training, significant feature extraction is required. In order to address these issues, text-to-image generation models usually consist of a GAN for picture production based on the embedded characteristics and a text encoder for embedding. Learning mappings that allow for the

creation of various images that are in line with the semantic information in the text is one of the main objectives. In this research, we propose a text-to-image generation model that combines high-quality image generation using StackGAN with BERTbased embeddings. Because they rely on pre-trained text encoders made for zero-shot visual identification tasks, existing models frequently include gaps in the text manifold. By optimizing the pre-trained BERT for the text-to-image creation task, our method overcomes this constraint.

Literature analysis

Sutskever and colleagues (2014) The sequence -to-sequence (Seq2Seq) model was first presented in this seminal paper, which efficiently maps input sequences to output sequences using Long Short-Term Memory (LSTM) networks. By

* Corresponding author: Bhanu Vangapally.

tackling machine translation issues, it produced cutting-edge outcomes for English-to-French translations in the WMT'14 dataset. The model outperformed phrase-based systems with a BLEU score of 34.8. Reversing input sequences to enhance learning and using beam search for decoding were two significant developments. The foundation for managing sequential data in a variety of fields was established by this concept.

In 2016, Reed et al. In this groundbreaking study, Reed et al. provide a brand-new technique for employing Generative Adversarial Networks (GANs) to create images from textual descriptions. They take textual descriptions and convert them into visual representations using deep convolutional networks. The deployment of a deep convolutional GAN (DCGAN), in which the discriminator and generator are both conditioned on text encodings obtained from recurrent neural networks (RNNs), is demonstrated in the paper.The paper's main contribution is the incorporation of text embeddings, which capture the description's semantic information, into a GAN framework. This results in the creation of realistic visuals that correspond to the text that is supplied. For instance, they demonstrate the potential of text-to-image synthesis by demonstrating how their approach can produce realistic images of flowers and birds from in-depth verbal descriptions.

Mansimov and colleagues, 2016: Mansimov et al. present a novel method in this research that uses attention mechanisms to generate visuals from text descriptions.The method's main concept is the employment of an iterative drawing process, in which the model simultaneously attends to pertinent words in the input caption and creates images by painting patches on a canvas. This attention mechanism aids the model in concentrating on particular caption elements that are essential for producing various image elements. A deep recurrent attention-based generative model is used by the alignDRAW model.In order to iteratively improve the image, a recurrent neural network (RNN) is used to construct patches and focus on certain caption sections at each stage. The Microsoft COCO dataset, which has many photos with informative captions, is used to train the system. A high degree of generalization is demonstrated by the model's ability to produce visuals that complement textual descriptions and outcomes that can adjust to previously unknown captions.

Odena and others, 2017) In order to produce high-resolution, lifelike photographs, this research presents Auxiliary Classifier GANs (AC-GANs), a variant of Generative Adversarial Networks (GANs) that include label conditioning. By adding an auxiliary classifier, AC-GANs enhance image synthesis by assisting the generator in creating diverse and classconditional images. Compared to models that only use scaled low-quality photos, this model produces images with global coherence and 128x128 resolution.The addition of two new measures of image quality—discriminability and diversity—is a significant contribution. Compared to merely expanding lower-quality photos, the study demonstrates that higher resolution images are more discriminable and offer greater class information. With outcomes similar to actual ImageNet data, AC-GANs also demonstrate a significant degree of variation across 1000 ImageNet classes.

Ramzan, Sadia.This study investigates the creation of visuals from textual descriptions using deep learning methods, specifically GANs. An overview of the various architectures used in text-to-image synthesis, including StackGAN, AttnGAN, and other deep learning techniques, is given by the authors.Their research focuses on using RNNs and deep convolutional networks to encode text and provide visual outputs that correspond to it. The study contrasts different

model performances and emphasizes the difficulties in preserving image quality and semantic coherence.

Table 1 Comparative Study of Existing Methodologies

Sequence to sequence learning with Neural Networks	Encoder-Decoder architecture	RNN	Moderate	-	-	ReLU, Tanh	Limited	Medium	Limited
Generative Adversarial Text-to-Image Synthesis	GAN	GAN	Moderate	-	-	ReLU LeakyReLU	Limited	High	Limited
Generating Images from Captions with Attention	CNN-RNN With Attention	CNNRNN	Moderate	-	-	ReLU, Tanh	Limited	Low	Limited
Conditional Image Synthesis with Auxiliary Classifier GANs	GAN	GAN	Moderate	-	-	ReLU LeakyReLU	Limited	Limited	Limited
Text-to-Image Generation Using Deep Learning	CNN-RNN	CNN-RNN	Moderate	-	-	ReLU, Tanh	Limited	Moderate	Limited
Text to Image Generation with Semantic-Spatial Aware GAN	GAN	GAN	High (8590%)	Moderate	Moderate	ReLU, LeakyReLU	Limited	Moderate	Complex training requirements
StackGAN	Progressive GAN, two-stage model	GAN	Moderate (7075%	Moderate	Moderate	ReLU, LeakyReLU	No	High	Struggles with long text inputs

Ours: BERT+Stack GAN	Bert,Stack Gan	GAN	Improved (8590%)	Moderate	Moderate	ReLU, LeakyReLU	Yes	High	computationally significant hardware resources required.

Architecture

Mainly here the Architecture is divided into two Parts.The part is consisting the BERT it is an Natural language Processing (NLP) which will take the input from the user and converts the given text description into Numerical vectors (embeddings).These sentence embeddings are then used as input for the subsequent image generation stages, providing crucial information about the text to guide the image generation process.Second Part consisting the Generative Adversarial Network(GAN) there are so many types of Gan's are availabe.the Gan is consisting a Generator and a discriminator.the role of the generator is to generate the image from the given text and then the role of the discriminator is to validate the generated image is perfectly aligned with the given text or not .Here in this System we are using mainly the Stacked Generative Adversarial Network it is a deep- learning algorithm Architecture which is used to convert the given numerical vectors(generated by the BERT)into image.It addresses the challenge of generating high-resolution images from textual descriptions by employing a two-stage approach.Stage-1 generates the image from the given feed by the BERT Transformer and it generates the Low-Quality Images. Stage-2 generates High-Quality images which will take the input as the image generated by the Stage-1 generator.the role of the Stage-1 and Stage-2 Discriminator is to validate the images generated by the generators if they are aligning with the given text discription or not

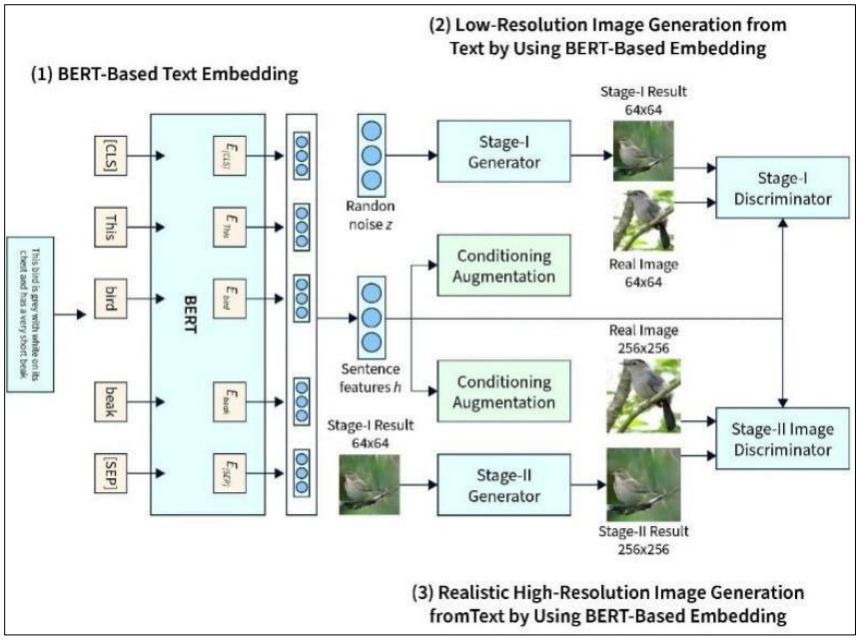


Figure 1 Generating realistic images from text using BERT-based embeddings

Proposed Methodology

BERT+StackGAN

We have proposed the combination of the two models i.e the BERT is an NLP model and the StackGAN is an Deeplearning model offers several significant advantages for image generation system. BERT's exceptional ability to understand and represent the semantic meaning of text provides a strong foundation for image generation. By incorporating BERT-generated sentence embeddings, the StackGAN architecture can leverage deep semantic understanding of the text description.In StackGan basically the Architecture of this model is a Hierarchical manner so we can able to generate the image for the Larger text Description.
So, we combined this BERT and StackGan to generate the images aligned with the give Text description Perfectly and to generate the High-quality realistic images.

Dataset

CUB_200_2011 is the dataset that we are utilizing.There are 200 classes in the CUB dataset. Each of the 11,788 bird photos has ten textual descriptions. Approximately 80% of the CUB dataset has an object proportion that is smaller than half the size of the image. Using a bounding box for the item, preprocessing was done in the experiment described in this study so that the object's ratio was greater than 0.75.

Algorithm

Bert-Embedding Algorithm

- Step 1: Set up the pre-trained parameters BERT θBERT
- Step 2: Define the number of fine-tuning steps (Sf) and dataset descriptions T is equal to {t1, t2,..., tn} • Step 3: For f = 1,..., Sf: perform
- Step 4: Execute the Adam update on θBERT using T.
- Step 5: Close for

StackGan-Algorithm

6.2.1. Stage 1

Let x_{row} be the low-resolution image, z be the random vector, D1 be the discriminator of stage 1, G1 be the generator, h be the text embedding of the provided text description via fine-tuned BERT, h_{ca} be the text embedding taken from the Gaussian conditioning variable, with λ serving as the regularization parameter. The stage 1 generator is taught to reduce the loss function of Equation (2), whereas the stage 1 discriminator is trained to maximize Equation (1). Equation (2)'s regularization term, the KL divergence, $D_{KL}(N(\mu_0(h), \sum_0(h))k(0, X_{row}))$, recovers latent text representation from an independent Gaussian distribution. Randomness is introduced by sampling the Gaussian conditioning variable h_{ca} from $N(\mu_0(h), \sum_0(h))$.

$$LD_1 = E(xrow, h) \sim Pdata\ [\log D1(xrow, h)] + Ez \sim pz, h \sim pdata\ [\log(1 - D1(G1(z, hca), h))] \quad \ldots (1)$$

$$LG_1 = Ez \sim pz, h \sim pdata[\log(1 - D1(G1(z, hca), h) + \lambda DKL(N(\mu 0(h), \sum 0(h))||N(0, Xrow) \ldots\ldots (2)$$

Quantitative Results

Stage 2

Assume G2 is the stage 2 generator and D2 is the stage 2 discriminator. Equation (3) is maximized by training the stage 2 discriminator, and Equation (4) is minimized by training the stage 2 generator. Only in stage 1 is random noise utilized, not at this point. Rather, stage 1 xrow uses a low-resolution image produced by the generator. By entering the lowresolution image produced by the stage1 generator and the text embedding vector taken from the text encoder, a highresolution image, xhigh, is produced. By entering an image and text embedding vector, the discriminator ascertains whether the image and text match.

Table 2 IS and FID of existing and our Proposed model on the CUB dataset**.**

Dataset	**Model**	**Inception Score**	**FréchetInception Distance**
CUB	Sequence to Sequence Learning with Neural Networks	2.88	68.79
CUB	Generative Adversarial Text-to- Image Synthesis	3.62	67.22
CUB	Generating Images from Captions with Attention	3.7	51.89
CUB	Conditional Image Synthesis with Auxiliary Classifier GANs	4.36	NaN
CUB	stackGAN	4.16	38.73
CUB	StackGAN-BERT	4.44	37.79

$$LD_2 = \mathrm{E}(X_{row}, h) \sim P_{data}\,[\log D_2(x, h)] + \mathrm{E}x_{row} \sim P_{G1}\,, h \sim P_{data}\,[\log(1 - D_2\,(G2(x_{row}, h_{ca}), h))] \text{ ..... (3)}$$

$$LG_2 = \mathrm{E}x_{row} \sim p_{G1} h \sim p_{data}[\log(1 - D_2\,(G_2(x_{row}, h_{ca}), h))] + \lambda D_{kl}(N\,(\mu 0(h), \sum(h))N\,(0, X) \quad \text{.....(4)}$$

Sample Generated Images

Text The bird is short and stubby with yellow on its body

Figure 2 Image generated by the proposed model

Conclusion

StackGAN is a deep learning model that generates photo-realistic images from text descriptions using a two-stage process. First, it creates a low-resolution sketch based on the text. Then, it refines the sketch with intricate details and colors to produce a high-resolution image. By incorporating "BERT Text Embeddings," StackGAN ensures the generated images accurately reflect the input text, leading to significant improvements in image quality, realism, and diversity.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Sequence to Sequence Learning with Neural Networks"(Sutskever et al., 2014).

[2] Generative Adversarial Text-to- Image Synthesis"(Reed et al., 2016)

[3] Generating Images from Captions with Attention"(Mansimov et al., 2016)

[4] Conditional Image Synthesis with Auxiliary Classifier GANs"

[5] Text-to-Image Generation Using Deep Learning"(Sadia Ramzan 1,* , Muhammad Munwar Iqbal 1 and Tehmina Kalsum 2)