JIMMA UNIVERSITY

JIMMA INSTITUTE OF TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATICS

MSC IN DATA SCIENCE

AMHARIC TEXT TO IMAGE GENERATION MODEL USING CONDITIONAL GENERATIVE ADVERSARIAL NETWORK

BY: GETNET ZERU GEDIF

Principal Advisor: AMANUEL AYDE (PhD)

Co-Advisor: ELSABET WEDAJO (PhD candidate)

A Thesis Submitted to Data science chair of Jimma University for the Fulfillment of the Requirements for the Masters in Data Science.

Jimma, Ethiopia

June, 2024

Approval Sheet

I affirm that the title of the dissertation thesis "AMHARIC TEXT TO IMAGE GENERATION MODEL USING CONDITIONAL GENERATIVE ADVERSARIAL NETWORK", is my original work, with all references to the materials used in it being appropriately cited.

_____ _____

Signature Date

(GETNET ZERU)

This research thesis has been submitted for examination with our approval as University advisors

Signature of Principal Advisor _____Date _____

AMANUEL AYDE (PhD)

Signature of Co-Advisor _____Date _____

ELSABET WEDAJO (PhD candidate.)

Muktar Bedaso (MSc)

Signature of Internal Examiner _____ Date_____

Tibebe Beshah (PhD)

Signature of External Examiner_____Date_____

ACKNOWLEDGEMENT

First and foremost, I want to thank God Almighty his graces and for giving me the courage, knowledge, and opportunity to begin this thesis and maintain, finish it satisfactorily.

Next, I want to sincerely thank my principal advisor Amanuel Ayde (PHD), co-advisor Elsabet Wodajo (PhD candidate) for their assistance and leadership while I worked on my thesis. They provided great support, courage, telling me all comments lovely and brotherly without which this thesis could not have been completed.

I want to thank Bonga University for its sponsorship and I would like to thank Jimma University for teaching me by fulfilling all the materials. Beside this, I want to thank all my friends helping me in data collection, image capturing and Amharic text writing.

ABSTRACT

Amharic text-to-image generation model using a conditional generative adversarial network (CGAN) is a novel concept that can be made possible by advances in deep learning. The aim of this study is to develop a model for Amharic text-to-image generation using CGAN algorithm. This study employed Experimental research design as study method. For this research, 2575 images of clothes and shoes were acquired, and the corresponding Amharic texts were written manually. For Amharic text preprocessing, stop word removal, punctuation mark removal, tokenizing the text, and creating word embedding using Word2Vec have been done. For image data preprocessing, noise removal, image segmentation, image resizing, normalizing, and converting to numpy arrays have been done. 80% of the paired Amharic text with the corresponding images was used to train the generator and discriminator networks for 1000 epochs and 32 batch sizes of data. In training, the generator network achieved 100% accuracy, and the discriminator achieved 40–50% accuracy, but the discriminator was unable to distinguish the generated images. Finally, the generator network trained on the training data has to be tested with the testing data to produce fake images to be compared with the tested real images. The generator achieved a Fréchet inception distance score of 4.99e+108 and an inception score of 417.2, which indicates the quantitative measure of the generated image quality. These numbers indicate that the generated images by the trained generator are not comparable with the real images. Training both the generator and discriminator at the updated values of parameters is much better than the default values of parameters as it is seen in the testing results. It is possible to develop a perfect model for Amharic text image generation with enough dataset, enough computational resources, and by using other variants of CGAN.

Keywords: *Deep learning, neural networks, Amharic text dataset, Natural language processing (NLP)*

Table of Contents

LIST OF ABBREVIATIONS

CGAN = Conditional Generative Adversarial Network

GAN = Generative Adversarial Network

G = Generator

D = Discriminator

AttnGAN= Attention Generative Adversarial Network

NLTK =Natural Language Tool Kit

NLP = Natural Language Processing

Glove=Global Vectors for Word Representation

CBOW = Continuous Bag-of-Words

Word2Vec = Word to Vector

MS COCO = Micro soft Common Objects in Context

FID = Fréchet Inception Distance

IS =inception score

OpenCV= Open Source Computer Vision Library

CSV = Comma Separated Values

GPU= Graphics Processing Unit

ReLU = Rectified Linear Unit

Tanh= hyperbolic tangent function

SD-GAN=Semantics Disentangling Generative Adversarial Network

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

Introduction

1.1 Background of the study

As many as 100 million people speak Amharic, and it is the official working language of Ethiopia [1]. Amharic language is written using an adapted derivation of the Ge'ez script called Fidel[2]. Amharic is readily written using Fidel, a Ge'ez script and alphabet in which each character represents a consonant vowel order since the consonant defines the form of a letter [3]. Amharic has 34 basic letters, each organized in seven orders, in the phonetic language and it is considered a highly inflected language, with grammatical markers denoting tense, person, and gender [4].

Image data refers to the digital illustration of visual information[5]. This data can be stored, processed, and transmitted using various formats and technologies, enabling a wide range of applications from photography to medical imaging and computer vision.

A vital part of contemporary digital life is image data, which includes everything from simple snapshots to intricate scientific visualizations[6]. To fully utilize picture data across a range of industries and technology, one must have a thorough understanding of its types, applications, and processing techniques.

In 2014, Ian Good fellow and associates created a class of machine learning frameworks called Generative Adversarial Networks, or GANs[7]. GANs consist of two neural networks: the generator and the discriminator. These two networks are trained simultaneously with competing objectives: Discriminator determines whether the created data is authentic and can distinguish between actual and false instances, while Generator creates new, synthetic data instances that mimic the training data.

By conditioning the generating process on additional input, Conditional Generative Adversarial Networks (CGANs) expand on the basic GAN structure[7]. Class labels, data from other modalities, or any other kind of auxiliary input could be this information. Generator Generates data conditioned on both a random noise vector and the additional information. Discriminator evaluates the authenticity of the generated data while considering the additional information.

The process of creating an image from a text description is known as text-to-image generation[8]. Text-to-image generation has a variety of applications, such as producing images from documents, providing visual assistance for visually impaired individuals, creating graphically pleasing editing options for web and graphic design projects, creating assisted learning materials, and enhancing marketing campaigns [9] .

A paper published in 2016 [10] was the first original effort in text-to-image generation and one of the pioneering studies in text-to-image creation that created visuals from textual descriptions. This system includes a real-versus-generated image discriminator network and an image-synthesizing generator network.

Developing a model for Amharic Text-to-Image Generation using a Conditional Generative Adversarial Network is a novel concept in computer vision that has been made possible by advances in deep learning[11].

One of the GAN versions that can produce accurate images from text descriptions is the conditional GAN, which enables the creation of intricate and visually appealing images using only a small number of descriptive words[12]

Some researchers are conducted research on Amharic text-image recognition, which means the process of recognizing characters in an image that have been written in the Amharic language. For example, [13] presented a mixed Attention-Connectionist Temporal Classification (CTC) network design. The model consists of an encoder module, attention module, and transcription module in a unified framework. The attention mechanism allows for learning powerful representations by integrating information from different time steps. The model surpasses modern techniques and achieves 1.04% and 0.93% of the character error rate on ADOCR test datasets.

The authors of the paper [14] proposed a method that can distinguish between handwritten and machine-printed text lines in an image of an Amharic document. Additionally, they show what happens when a binary support vector machine, which minimizes a margin-based loss rather than the cross-entropy loss, is used to replace the final completely linked layer. Based on the outcomes of the experiments, binary SVM provides a considerable improvement in discrimination performance over fully linked layers.

The authors of the paper [15] tackled the difficulties with Amharic OCR in two key ways. In their first algorithmic contribution, they look at deep learning strategies that can meet the need for Amharic OCR. The second is a technical contribution that includes various efforts to develop the

OCR model; as a result, it introduced a new Amharic database made up of collections of photos with character- and text-level annotations. Additionally, a unique CNN-based framework was developed, which achieves an overall character recognition accuracy of 94.97%. Using the Fidel-Gebeta character encoding, Text-line-level approaches are explored and developed based on sequence-to-sequence learning in addition to character-level methods. By doing away with the requirement to segment individual characters, these models bypass several of the pre-processing steps employed in earlier research. Before the Bi-LSTM layers in this approach, they employed a stack of CNNs and train the model from beginning to end. With the ADOCR test set, this model outperforms the LSTM-CTC based network with an average CER of 3.75%. They presented a unique attention-based methodology by fusing the attention mechanism into CTC objective function in response to the success of attention in tackling the issues of long sequences in Neural Machine Translation (NMT). With a CER of 1.04 % and 0.93 % on printed materials, this approach outperforms the current methods by a significant margin.

The authors of the paper [16] attempted to create an OCR system for Amharic, one of these rare languages with a distinctive script. They enhanced the attention mechanism of the Amharic text-image recognition system. The efficacy of the proposed attention-based model is evaluated using the test dataset from the ADOCR database, which includes printed and artificially created Amharic text-line images and achieved a promising results with a CER of 1.54% and 1.17% respectively.

1.2 Motivation of the Study

Text-to-image generation is an active area of research and innovation. Through the visual representation of traditional stories and practices, research on Amharic text-to-image generation using CGANs contributes to the preservation and promotion of Ethiopia's rich cultural heritage. By ensuring the inclusion of Amharic-speaking communities in the digital transition, this research promotes linguistic diversity in technology. This work advances computer vision and natural language processing, especially for underrepresented languages. The production of useful datasets and resources for Amharic as a result of this research can be applied to the advancement of AI and language technologies. By focusing on Amharic text, we can contribute to the advancement of the field and explore new techniques or adaptations that are specifically tailored to the characteristics of the Amharic language. Despite being a language with various applications and widely used in our country, there are no studies on Amharic text-to-image generation. Text-to-image generation models have the potential to bridge the gap between textual descriptions and visual representations.

By developing a model specifically for Amharic text, we can explore the unique challenges and opportunities associated with generating images from a language with its own linguistic and cultural nuances. It can enhance communication and understanding by providing visual interpretations of Amharic text.

1.3 Problem Statement

Digital content is currently growing exponentially both globally and in the context of Ethiopia because of technological advancements. A deep learning model that uses generative modeling technology to extract pertinent information in the form of images would be necessary given the exponential growth of locally accessible textual data[9]. Text comprehension is getting harder to read and visualize[17]. Although Amharic text-to-image generation has many applications, there is a shortage of research in this field. The following issues are brought on by the lack of studies on Amharic text to image generation.

Making visual content in Amharic is difficult for those without experience in graphic design or image creation. Screen readers used by people with visual impairments cannot be used to view Amharic text if there are no associated visuals[18].

It is difficult to hold users' attention and effectively communicate a message without the use of visual aids. As a result, the absence of such a model leads to low user engagement or comprehension of Amharic content. Many cultures place a high value on visual art, and without a model for generating Amharic text into images, there is a loss of cultural expression through visual means[19].

There is a unique writing system in Amharic, including native characters for the language[13]. The performance and accuracy of text-to-image generating models are impacted by differences in syntax, vocabulary, and textual nuances because these variations should be taken into account [12] Several studies are conducted on text-to-image synthesis, particularly for the English language. For example, the quality of the generated images and the inception score on the MS COCO dataset are both greatly enhanced by the inclusion of a dialogue [20]. The study by [21] built a deep learning-based recurrent convolutional generative adversarial network that successfully bridges the advancements in text and picture modeling, converting visual notions from words to pixels.

The text adaptive generative adversarial network was proposed in the study by [22] to produce semantically altered images while maintaining text-irrelevant contents. In order to provide high-quality image generation, the study by [23] constructed ARTIST, a transformer-based Chinese

text-to-image synthesizer and Rich linguistic and relational knowledge facts are fed into the model to assure better model performance.

Despite the growing advancements in text-to-image generation using Conditional Generative Adversarial Networks (CGANs), there is a notable research gap in applying these technologies to underrepresented languages such as Amharic. This gap includes the lack of language-specific models, scarcity of annotated datasets, and Amharic-speaking communities have limited access to advanced technologies, resulting in reduced educational tools and business support. Addressing this research gap not only promotes technological inclusivity but also contributes to the preservation and promotion of Ethiopian culture.

Determining how to organize and preprocess the data into a suitable dataset for the Amharic text-to-image generation model is essential due to the scarcity of annotated datasets.
A methodological difficulty needs to be addressed when applying Conditional Generative Adversarial Networks (CGAN) to the task of Amharic text-to-image generation.
Understanding the extent to which different configurations of CGAN parameters affect the accuracy and performance of Amharic text-to-image generation is crucial for optimizing the model.

This study tackled the challenge of generating images from Amharic text descriptions, specifically for shoes and clothes. It highlighted the issue of limited annotated datasets available for this task. To address this, the researcher employed a deep learning method called Conditional Generative Adversarial Network (CGAN). CGAN involves two neural networks that work together, one generating images and the other evaluating them. The generator creates images based on the given Amharic text descriptions. The discriminator assesses the realism of these images. Through this adversarial process, both networks improve over time. Despite the constraints of the dataset, the goal of this technique was to improve the quality and relevancy of the generated images.

1.4. Research Questions

- How to organize and preprocess the data into suitable dataset for Amharic text to image generation model?
- How can CGAN be applied to Amharic text-to-image generation?
- To what extent different configurations of CGAN parameters affect the accuracy and performance of Amharic text-to-image generation?

1.5. Objective of the study

1.5.1 General objective

The main objective of this study is to develop Amharic text-to-image generation model using CGAN.

1.5.2 Specific objective

The following specific objectives are formulated to address the main objective of the study:

- To organize and preprocess a data into a suitable dataset for Amharic text to image generation model.
- To develop an image generation model for Amharic text to image generation by employing CGAN
- To analyze the impact of different configurations of CGAN parameters on the accuracy and performance of Amharic text-to-image generation.
- To evaluate the performance of the model for Amharic text-to-image generation.
- To provide future lines of works for further improvements in the field of Amharic text to image generation.

1.6 Scope and limitations of the study

This study uses conditional generative adversarial neural networks to develop a model for Amharic text-to-image generation. This study attempted to show that given Amharic text describing images of shoes and clothes and the Conditional Generative Adversarial Network model can generate an image. Additionally, it focuses on the availability and significance of training data for the study as well as the creation of a model and efficient methods for assessing the produced images. The

Amharic text dataset for this study consists only of simple Amharic sentences and phrases to describe the visual contents of the image. The image dataset includes images of shoes and clothes.

The model lacks the ability to generate real-looking images as the test result shows because of the limited amount of data. The model is not comparable with state-of-the-art models for converting text into images in other languages.

1.7 Significance of the study

This study is beneficial for Digital marketing and e-commerce. In order to better market to Amharic-speaking customers, businesses can be able to use this technology to automatically create images for their goods or services based on Amharic text descriptions. Additionally, it is useful resource for linguists and researchers who study the Amharic language and need to visualize particular theories or concepts.

Because images are straightforward to understand, Amharic text-to-image generation makes the language simpler to learn. This study is significant because it investigates the possibility of generating images from Amharic text using a sophisticated machine-learning method called a conditional adversarial network. Applications for this technology can be found in natural language processing and computer vision. This study makes these technologies accessible to Amharic readers by adapting them for the language. This is crucial because millions of Ethiopians speak the Amharic language.

1.8 Definition of terms

Amharic text: refers to written content in Amharic language and specifically denotes textual descriptions of fashion images. These textual descriptions include Amharic phrases and sentences describing the images.

Image generation: The process of creating visual content based on textual descriptions in the Amharic language**.**

CGAN: a type of generative model used for image generation conditioned on text in this case.

Generator: A neural network responsible for creating images from random noise and Amharic text embedding.

Discriminator: a neural network that evaluates whether an input image is real (from the actual image) or fake (generated by the Generator).

Generative Adversarial Network: is a type of AI algorithms that is applied to machine learning without supervision.

Model development: refers to the process of creating and training the CGAN to generate images based on Amharic text descriptions.

Text-to-image generation: the process of creating images based on textual inputs or based on Amharic texts

1.9 Organization of the Thesis

Five chapters make up this thesis. What was done for this thesis work is presented in the thesis' remaining section. The literature review and relevant works are covered in Chapter 2, the research Method is covered in Chapter 3, the results and discussion are covered in Chapter 4, and the conclusion and recommendations are covered in Chapter 5.

CHAPTER TWO

Literature Review

2.1 Introduction

This chapter covers an overview of GAN, CGAN, text-to-image generation, the unique characteristics of Amharic language, an overview of deep learning, an overview of natural language processing (NLP), text embedding models, and a summary of related works in English and other languages.

2.2. Overview of text-to-image generation

Text-to-image generation, a promising branch of deep learning, aims to create precise images from textual descriptions [19]. This can be compared to image captioning, which generates natural language descriptions for photographs. The use of GANs for text-to-image generation was first proposed in a seminal paper called "Generative Adversarial Text-to-Image Synthesis"[10]. Figure 1 shows the general architecture of GAN.

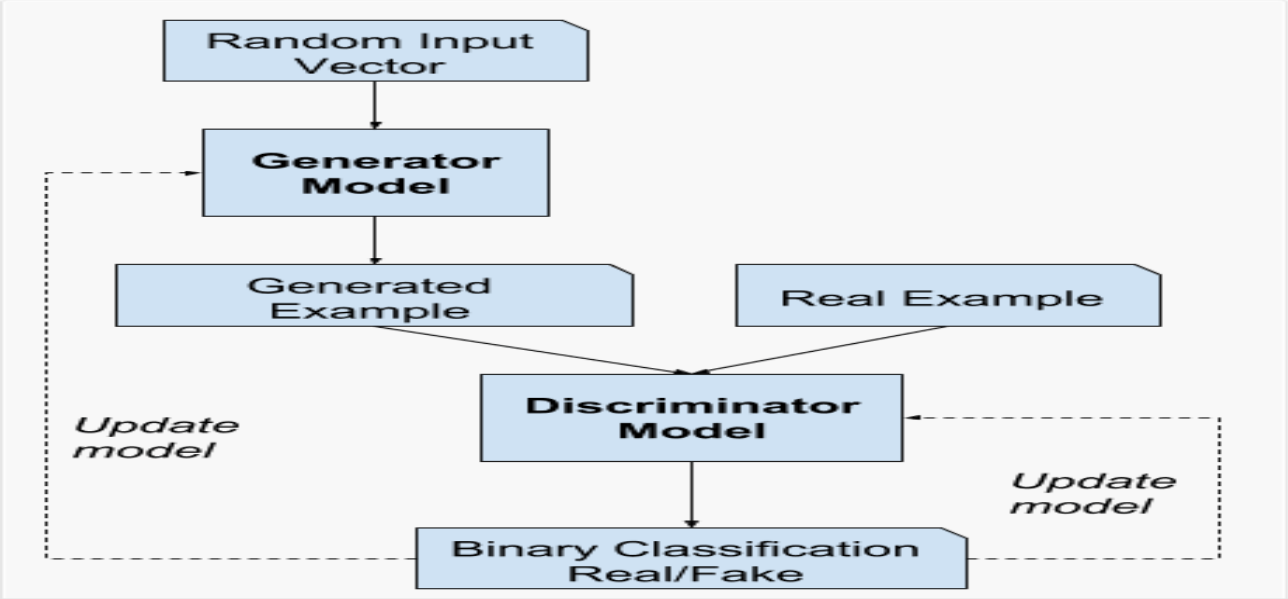

Figure 2. 1 General architecture of GAN

Image source[8]

The conditional type of GAN is called conditional generative Adversarial network [9]. Adding the additional auxiliary information (text or labels) turns the GAN into CGAN [9]. Generator of CGAN takes the extra auxiliary information c (text) and a latent vector z so to generates conditional real-looking data given by (G(z|c)), and discriminator of CGAN uses real data x and additional auxiliary information c to discriminate between real data x and generator generated samples and given by D(G(z|c)). CGAN can control the generation of data, which is impossible with the vanilla GAN. CGAN updated loss function is given as follows:

$$L_{CGAN} = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x|c)] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D\left(G\left(z|c\right)\right)\right)\right] \quad [9]$$

Where c is the conditional variable

z is input noise variable

x is real data

D is Discriminator

G is Generator and the discriminator's objective was trained on fake (p_g (G)) and real data (P_{data} (x)) to increase its cost value , that is, log (D(x|c) as well as the generator's random noise vector training (p_z(z)) is to minimize its cost value, i.e., log (1−D (G(z|c))). Figure 2 shows the general Architecture of CGAN.

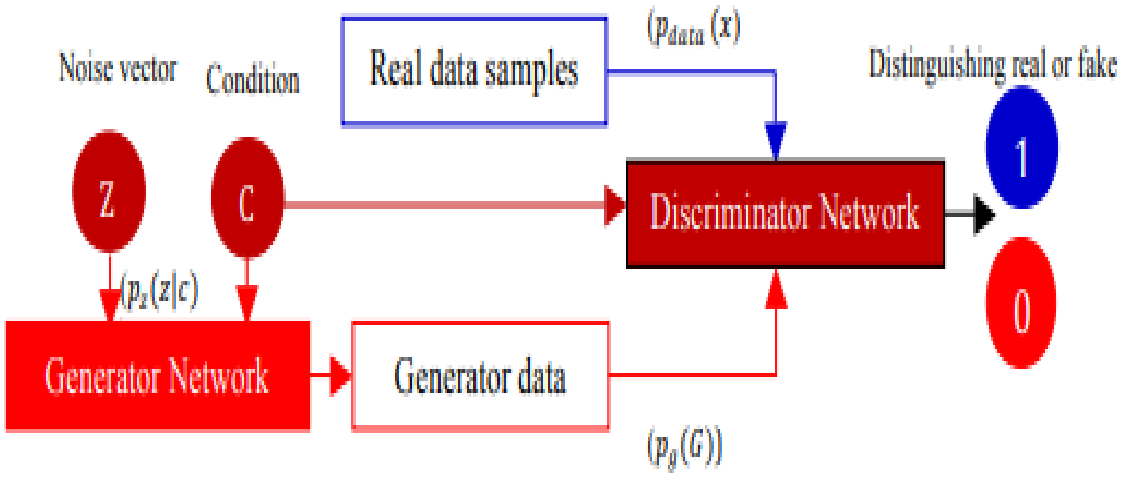

Figure 2. 2 Genre architecture of CGAN

(source [9]**)**

2.3 Steps of text-to-image generation using CGAN

The following are the basic steps for text-to-image generation using a conditional generative adversarial network (CGAN).

2.3.1 Data Preparation

An essential first step in text-to-image creation with a CGAN is data preparation [24]. It entails getting the dataset ready, which is made up of matching text descriptions and images. Ensuring that the images and text descriptions are properly positioned such that every text description has an image that correlates with it.

Cleaning the text descriptions by removing any unwanted characters, punctuation, or special symbols. Depending on our preferences, We also need to handle any specific preprocessing steps required for our specific text data, such as tokenization and stop-word removal[25].

Preprocessing the image data to ensure that it is in a suitable format for training. This entails normalizing the pixel values to a common range, such as [0, 1], scaling the images to a consistent size, and converting to an image format to standardize [17].

Establishing a clear correspondence between the text descriptions and their corresponding images. This alignment is crucial for training the CGAN effectively [10]. Ensuring that the order of the text descriptions and images matches, so that each text description aligns with the correct image. It's common to use a naming convention or indexing system to link the text and image files together.

Separating the dataset into sets for testing and training: The testing set is used to assess the trained model after it has been trained using the training set to train the CGAN. The typical split ratio is around 70–80% for training and 10–20% for testing, but this can vary depending on the size of the dataset and specific requirements [26] .

2.3.2 Text embedding

Text embedding is a natural language processing approach that turns words, sentences, or paragraphs of text into vectors of numbers that machine learning algorithms can quickly analyze [27]. According to its context and meaning, each word or phrase is given a numerical value, which is then used to assign similar or related words numerical values. In natural language processing applications like text-to-image creation, common word embedding models are Word2Vec, GloVe,

and FastText[28]. It's vital to keep in mind that these models could not already include embeddings that are tailored to the Amharic language because they are often trained on extensive English corpora. They can nevertheless be utilized as a starting point for learning distributed representations of Amharic words, which can then be incorporated into a CGAN-based text-to-image creation system. Depending on the particular work at hand, it needs to be started from scratch. By capturing the semantic meaning of words and phrases rather than just considering them as discrete units, text embeddings enable machine learning models to produce better predictions.

Continual vector representations of words are created using the word embedding technique Word2Vec, which takes into account how frequently words occur together in a corpus [27]. The continuous bag-of-words (CBOW) variation and the skip-gram variant make up the model. While Skip-gram predicts context words based on the target word, CBOW predicts the target word based on its context words. To quantify word similarity and carry out tasks requiring analogical reasoning, one can use the learned word vectors, which capture semantic associations.

Another well-liked word embedding model is GloVe (Global Vectors for Word Representation), which blends local context window-based approaches with global matrix factorization techniques [29]. The word vectors that the model learns to represent both local and global semantic links are based on word co-occurrence statistics. Glove embeddings are commonly employed in many NLP tasks, including language creation and comprehension.

With its representation of words as collections of character n-grams, the FastText word embedding model expands on the concepts of Word2Vec [28]. FastText can better handle out-of-vocabulary words and capture morphological variants by taking into account sub-word information.

2.3.3 Training Setup

Defining the hyperparameters for training the CGAN, such as the learning rate, batch size, and number of epochs, is the fifth step in text-to-image generation [30]. Choosing an appropriate loss function, such as binary cross-entropy, for both the generator and discriminator is also included in this step.

2.3.4 Training Loop

Training the CGAN by alternating between training the discriminator and the generator is the sixth step. In each iteration, randomly selecting a batch of real image-text pairs and a batch of random noise-text pairs is important, as is using these batches to update the discriminator and generator weights through backpropagation [26].

2.3.5 Evaluation and Fine-tuning

Evaluate the trained CGAN by generating images for new text descriptions and assessing their quality and Fine-tune the CGAN if necessary by adjusting the hyperparameters or modifying the network architecture [29].

2.4 Overview of deep learning

Deep learning uses artificial neural networks, which were developed to imitate how the human brain works, to help models learn from vast and complex data [26]. The system can identify intricate patterns and traits with the aid of these neural networks, which have numerous layers of interconnected nodes and can learn hierarchical representations of the data [31].

In more technical terms, deep learning algorithms adjust the model's parameters in reaction to incorrect predictions using backpropagation techniques. Deep learning systems learn hierarchical data representations by utilizing many layers of nonlinear processing units[31].Through this procedure, the model can automatically fine-tune the internal representations of the data that it uses to make predictions that get better and better over time in a variety of applications, including computer vision, natural language processing, speech recognition, and Amharic text-to-image generation, deep learning is utilized to produce state-of-the-art outcomes [32].

2.5 Natural language processing (NLP)

NLP is the study of how human language and computers interact and Computers can now comprehend, decipher, and produce human language because of NLP[12]. The goal of NLP is to make interactions between people and computers more effective, natural, and nuanced [33]. To evaluate text data and derive meaning from it, this field of study employs a variety of techniques, including machine learning, deep learning, and computational linguistics. The NLP branch of artificial intelligence is concerned with the use of natural language in both machine and human

communication. The technique of creating an image from a given textual description is referred to as text-to-image generation[19]. This can be accomplished by employing NLP techniques to convert the text into a visual representation.

2.6 Over view of Amharic language

Amharic is the official language of Ethiopia and an Afro-Asiatic Semitic language [34] . A Geez script variant, which dates back to ancient Ethiopia, is used to write it. It is written using a variation of the Ge'ez script, an ancient script from Ethiopia. It is known for its unique script and grammatical structure, which makes it distinct from other languages in the region.

Amharic uses a syllabic writing system called the Fidel, which consists of 33 basic characters and various combinations of these characters to form complex words [15]. In contrast to other writing systems, the Fidel is written from left to right, and each letter denotes both a consonant and a vowel sound. Therefore, Amharic text is a combination of consonants and vowels arranged in a specific order to form words and sentences [34].Amharic texts can range from old religious manuscripts and poetry to modern literature and scientific documents [34].

Amharic has consonant-vowel pairings that are represented by a variety of symbols in its particular writing system.

Amharic is an agglutinative language, which means that new words are formed by affixing prefixes, suffixes, and infixes to the root[35].

Amharic verbs are conjugated based on the subject, object, tense, aspect, and mood. Amharic has a subject-object-verb (SOV) basic word order, but word order can be flexible because case marking is so common[2] .

Amharic strictly follows the verb-subject-object (VSO) word order when constructing questions. Adjectives are often positioned following the noun they describe. Prepositions are used to indicate the connections between words in sentences. Relative clauses are widely used in Amharic to provide information about a noun [34]].

2.7 Text to image generation in English and other languages

Most text-to-image synthesis research is conducted in the English language, and a few is in the Chinese language.

The study by [36] proposed a controllable text-to-image generative adversarial network that is capable of both efficiently creating high-quality synthetic images and controlling specific aspects of image synthesis in accordance with descriptions in plain language.

The study by [37] proposed GAN-INT-CLS (GAN with Integrated Classification) for the enhancement of text-to-image generation in the English language. The enhancements include text conditioning augmentation, feature matching, and distance loss functions.

The paper by[23] proposed transformer-based TIS (DALL-E) to enhance text-to-image generation for Chinese.

In order to produce high-quality, photo-realistic images, the research by [25] proposes stacked generative adversarial networks (StackGANs), a two-stage generative adversarial network design. Using a text description as input, Stage-I GAN creates low-resolution images that depict a scene's basic shape and colors. Stage-II GAN uses the text description and Stage-I results to create high-resolution images with photo-realistic details.

The study [38] suggested an Attentional Generative Adversarial Network that allows attention-driven, multi-stage refinement for fine-grained text-to-image synthesis. Through a unique attentional generating network, the AttnGAN, which focuses on significant words in the natural language description, is able to generate detailed information in several sub-regions of the image.

CogView, a 4-billion-parameter transformer with a VQ-VAE tokenizer, was proposed in the study [39] to improve text-to-image generation for the Chinese language. Additionally, the authors present the fine-tuning techniques for a number of downstream tasks, including fashion design, text-image ranking, super-resolution, and style learning.

The study by [40] addressed the problem of semantic coherence between the text description and visual content by introducing a novel global-local attentive and semantic-preserving text-to-image-to-text framework called MirrorGAN. MirrorGAN, which comprises of three modules, exploits the concept of learning text-to-image production by redescription. Semantic text alignment and regeneration, global-local collaborative attentive, and semantic text embedding are the three modules that are accessible for making cascading images.

The study [41] investigated a unique photo-realistic text-to-image generation model that satisfies both low-level semantic diversity and high-level semantic consistency by implicitly disentangling semantics.

2.8 Related works

The following table shows the summary of the recent studies that are conducted on text to image generation using.

Table 2.1 Related works

Ref.No	Authors	datasets	Methods	performance	limitation/ drawback	Language
[25]	Sharma et al. (2018)	MS COCO	StackGAN	9.74 IS	Lack of Fine-Grained Control.	English
[23]	Liu et al. (2022)	COCO-CN& Flickr8k-CN	transformer-based TIS(DALL-E)	66.66(FID)& 14.71(IS) in COCO-CN, 49.42(FID) &15.01(IS) in Flickr8k-CN	dataset dependency	Chinse
[37]	Tan et al. (2022)	CUB-200-2011,11788 images	GAN-INT-CLS	5.26 IS on CUB-200-2011 dataset	The image lacks some visual detail.	English
[36]	Li et al. (2019)	CUB-bird, 11,788 datasets &MS COCO,123,287 datasets	Control GAN	4.58 IS on CUB data &, 24.06 IS on MS-COCO	lack of semantic consistency	English
[38]	Xu et al. n.d.	CUB dataset & COCO dataset	Attention GAN	14.14% IS on CUB & 170.25% IS on MS-COCO data	lacks more visual detail of the image	English
[39]	Ding et al. (2021)	blurred MS COCO dataset	DALE-E	85.0 (FID)score& 17.9(IS) score	Image blurriness	Chinse
[40]	Qiao et al. (2019)	CUB-bird dataset & MS COCO dataset	MirrorGAN	4.56 (IS) on CUB dataset, 26.47 (IS) on MS- COCO dataset	Lacks complete end-to-end training due to limited computational resources.	English
[41]	Yin et al. (2019)	CUB-bird dataset & MS COCO dataset	SD-GAN	4.67 IS on CUB dataset & 35.69 on MS-COCO dataset	dependency on Semantics disentangling	English

The above-mentioned works were conducted in English and Chinese by using the variants of GAN using three publically available datasets to train a model for text-to-image generation. This research is able to address the following notable gaps:

The lack of publicly accessible datasets created specifically for text-to-image generation in the Amharic language. Amharic is a unique language with its own linguistic and cultural characteristics. Examining the specific challenges and nuances of translating Amharic text into visually coherent and meaningful images is interesting. Understanding how linguistic features, cultural context, and semiotic aspects of Amharic impact the generation process can provide insights into improving the quality and relevance of generated images, so this study aims to build a model for Amharic text-to-image generation using CGAN.

.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

This chapter covers the research design, data collection methods, procedures for preparing the Amharic text and the corresponding image dataset, the architecture of the model and its components, and the implementation tools that were employed in this study.

3.2 Research Design

This study employed an experimental research design because it involves manipulating variables, establish a cause-and-effect relationship between the independent (input text in this case) and dependent (output images) variables[42]. The study aims to determine the effectiveness of the CGAN model in generating images from Amharic text inputs. This requires the collection of data, preprocessing data, pairing text with the corresponding images, training of CGAN, testing the trained model with testing Amharic text, and evaluating the quality of the images generated by the trained model.

Generally, the study focusing on two main aspects.

The development of an Amharic text to image generation model: can be achieved through data collection both text descriptions and corresponding images, data preprocessing and training the neural networks and updating the parameters based on training lose and training accuracy of neural networks. Finally, a CGAN model should be tested with testing Amharic texts and the generated images are evaluated by Inception score and Frechet inception score. The following figure shows flow chart of experimental research design.

a

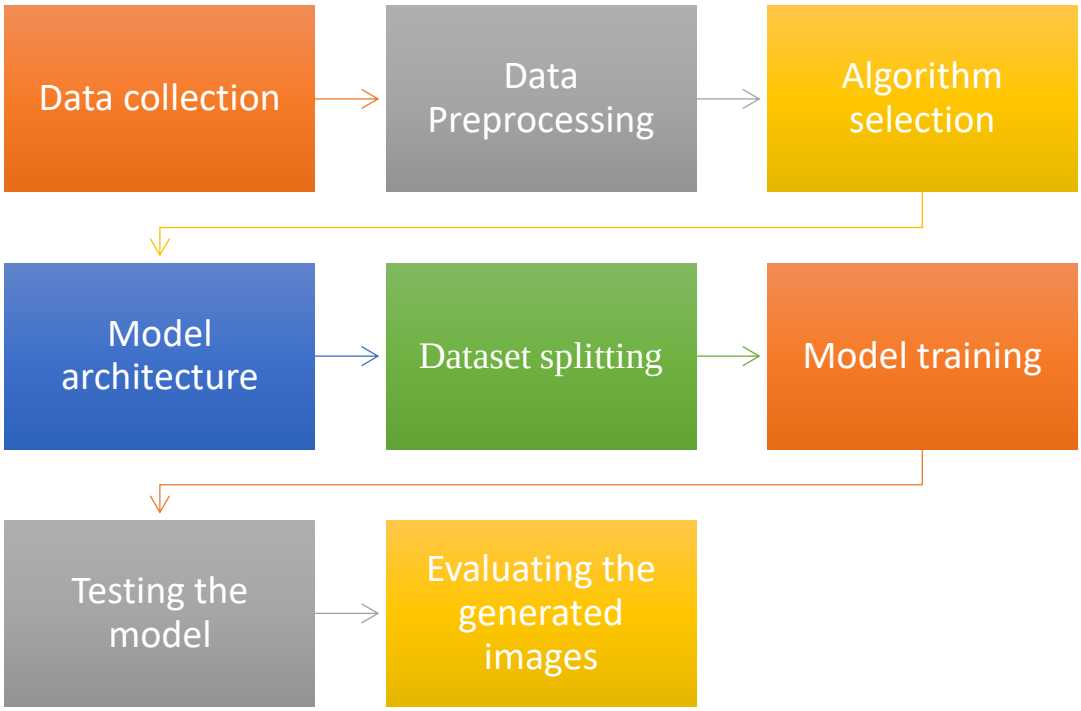

Figure 3. 1 Flow chart of Experimental research design

3.3 Data Collection

Two types of datasets were collected: An image dataset and Amharic descriptive texts that describe the corresponding images.

Since there is no publicly available dataset of Amharic descriptive texts and the corresponding image pairs, Using web scraping techniques to gather images of shoes and clothes from online marketplaces, fashion websites, and e-commerce platforms is considered as a method[43].

Capturing images consistently involves using the same angles, distances, and settings for each photo[44]. This uniformity helps maintain a standard appearance across the dataset, making it easier for the model to learn from the images[45]. Consistent settings also reduce variability caused by lighting or focus differences, leading to more reliable results. By standardizing the image capture process, we enhance the quality and effectiveness of the dataset for training the Amharic text-to-image generation model, so the following methods are used for collecting the dataset:

Collecting images: 1075 images were collected from social media sites. All the images have the same data format that is jpg file format which more suitable for model training. JPG is the most widely used one of the most common image formats and is well-supported[42].

Image Capturing: 1500 images of shoes and clothes are captured by smart phones, and the corresponding Amharic texts are written manually by describing the visual detail of each image.

3.4 Data Preprocessing

After the required amount of data is collected, data preparation is the next and most important step. There are two types of data preprocessing for this study. Amharic text preprocessing, which involves Amharic text preprocessing such as stop word removal, punctuation mark removal, Amharic text tokenization, and converting Amharic text to word embeddings. The second one is image data preprocessing, which involves noise removal, image segmentation, image resizing, and image normalizing, and converting the resized and normalized image to Numpy arrays.

3.4.1 Amharic text preprocessing

After images and Amharic texts were collected and properly arranged, preprocessing Amharic texts like removing Amharic stop words, removing Amharic punctuation marks, removing unwanted characters, text tokenization, and converting the cleaned Amharic texts into word embeddings were done.

Stop word removal

Stop word removal is a frequent preprocessing step in natural language processing activities, including text-to-image generation utilizing CGAN. Stop words are often used words that lack major semantic value[12]. They are frequently eliminated to cut down on background noise and draw attention to the text's more crucial terms. To remove stop words from Amharic text, we need to compile a list of Amharic stop words specific to the Amharic language. This list can be created by identifying frequently occurring words in a representative Amharic language.

Punctuation marks removal

Amharic has its own set of punctuation marks, like any other language. Punctuation mark removal is a standard preprocessing step in the process of creating a model for text-to-image generation using CGAN. In writing, punctuation marks are symbols that indicate pauses, emphasis, and other

linguistic elements[46]. To remove punctuation marks from Amharic text, it is important to compile a list of some Amharic punctuation marks specific to Amharic language.

Amharic text Tokenization

Tokenization is the process of dividing text into discrete pieces known as tokens [33]. In order to create a model for text to image generation using Conditional Generative Adversarial Networks, tokenization is a crucial step. Tokens are often words or sub word units in the context of natural language processing (NLP). Tokenization is crucial because it helps the model to successfully comprehend and handle the text data. Express the text in a structured fashion that is simple to feed into the Word2Vec model by dividing it up into tokens. Tokenization also aids in controlling the vocabulary size and lowering the model's processing needs[29]. Before training Word2Vec to create Amharic text embeddings, the cleaned Amharic text should be converted to a list of words called tokens.

Amharic text embeddings

Creating Amharic text embeddings using Word2Vec is crucial for Amharic text-to-image generation using CGAN. Word2Vec-based text embeddings capture the semantic meaning of words in a dense vector space[27] .Word2Vec embeddings encode important semantic information by modeling words as continuous vectors, which enables the CGAN model to comprehend and make use of the relationships between words in the Amharic language. This semantic representation helps the model generate visually coherent images that align with the corresponding Amharic text descriptions.

Amharic word embeddings refer to vector representations of Amharic words in a continuous vector space. These word embeddings capture semantic relationships between words by placing them in a multi-dimensional space where similar words are positioned close to each other. These embeddings are converted by training Word2Vec after the preparation of Amharic texts. After the generator of CGAN receives conditioning input that guides the image generation process, the generator learns to create images based on the semantic content of the Amharic text descriptions by using Amharic text embeddings as the conditioning input. This makes sure that the generated visuals match the text's intended meaning and are contextually relevant. Amharic text exhibits variations due to factors like word order, synonyms, morphology, or different ways of expressing the same concept [3]. Word2Vec embeddings reduce these variations by translating various word

surface shapes to related vector representations. This makes it possible for text descriptions to express similar semantic content.

Word2Vec generates a single vector for each word, making it easier to interpret and visualize[27]. This can be particularly useful in the initial stages of research where understanding the behavior of embeddings is crucial. It is generally faster and less resource-intensive to train compared to FastText, which needs to account for sub-word information, and GloVe, which requires computing and factorizing a large co-occurrence matrix[47]. FastText represents words as a sum of their character n-grams, leading to potentially more complex representations[47]. GloVe is based on a global word-word co-occurrence matrix, which can be less intuitive compared to the direct prediction-based approach of Word2Vec[47].

3.4.2 Image data preprocessing

Image Segmentation

Partitioning an image into numerous segments or areas to simplify or alter its representation, making it more meaningful and easier to analyze, is a critical approach in image processing and computer vision[6]. Image segmentation's primary objective is to identify objects and boundaries (such as lines, curves, etc.) inside images. More precisely, image segmentation gives each pixel in an image a label so that those pixels that have the same label have similar properties. Each pixel in a region is similar with respect to some characteristic or computed property, such as color, intensity, or texture. Segmenting an image helps in identifying and isolating different objects or regions within an image, leading to better quality and more detailed image generation. Segmenting the image can ensure that the generated images are more accurate and meaningful by focusing on the relevant parts of the image.

Noise removal

Removing noise from photos is essential to improving the clarity and quality of images that are produced. By minimizing undesirable artifacts and flaws in photos, noise reduction techniques seek to enhance the overall visual quality of the image[48]. Noise can be introduced via a number of sources, such as sensor errors, transmission errors, or environmental conditions, and it can affect the quality and accuracy of image analysis. To assist the model in concentrating on picking up important aspects and characteristics, resulting in the creation of more accurate and realistic

images. Removing noise from images ensures that the training data is cleaner, leading to better learning outcomes and Noise can confuse the model and degrade its performance[49].

Image resizing

The image data should be resized according to the architecture of the model and the number of layers making up the model. For training purposes in this study, the image is resized to 128x128 pixels in size. The image data should also be normalized to standardize the range and converted to Numpy arrays for compatibility and efficiency. Resizing images to a standard scale that matches the computational capabilities and architecture of the model is very important to guarantee that all images have the same dimensions and aspect ratio[17].

Image Normalizing

Normalizing the pixel values of the images to a standardized range, typically between 0 and 1, helps stabilize the training process and improve convergence [41]]. Converting the resized and normalized images to NumPy arrays in the context of text-to-image generation offers standardization, efficiency, compatibility, integration with data processing pipelines, and flexibility in data manipulation, which are all advantageous for training and generating images with the CGAN model.

3.5 Conditional Generative Adversarial Networks (CGAN)

Conditional Generative Adversarial Network (CGAN) is a deep learning approach that involves two deep neural networks competing against each other in a game-like setting, called the “Generative Adversarial Network," and uses supervised learning to generate new synthetic data, which can then be used to enhance existing data sets [50]. In text-to-image generation, a conditional generative adversarial network (CGAN) is used to generate images from natural language descriptions. A generator (G) in CGAN creates an image based on random noise and text embeddings, while a discriminator (D) in CGAN tries to determine if the generated image looks real or like it was generated by the generator [50]. The generator ‘wins’ if it produces an image that is good enough for the discriminator to not be able to tell the difference between generated images and real images. The discriminator ‘wins’ by correctly identifying the generated images as fake. By training both networks to compete against each other, each can make improvements towards its goal until both reach a point where neither can tell which images are fake or real.

By adding conditional inputs, CGAN expands the capabilities of conventional GANs and allows them to produce high-quality images that precisely represent specified conditions, such textual descriptions. This makes them especially well-suited for intricate tasks like text-to-image synthesis[51].

2.5.1 Generator Network

The generator network takes two inputs: a random noise vector (z) and conditional information (c).The generator generates images from text embeddings and noise vectors. Typically, the generator network is composed of layers like convolutional, upsampling, and dense layers. The goal of the generator is to learn to generate realistic images that match the given text descriptions [7]. It aims to generate synthetic images G(z, c) that resemble the real data distribution. This is given by equation 1, where G stands for Generator and D stands for Discriminator.

G: z, c---------->G(z, c) ---------->(1)

G_L=-log (D(G(z, c), c)) ---------->(2)

Equation 2 measures how well the generator is fooling the discriminator into classifying the generated image as real when conditioned on the given textual description. During training, the generator aims to minimize this loss, which encourages it to generate images that align with the provided textual information [7].

G(z, c): represents the output of the generator, where z is a random noise vector and c is the conditional information (textual description).

D(G(z, c), c): symbolizes the discriminator's output given the image produced by the generator and the corresponding conditional information c (text).

log(D(G(z, c), c)): This calculates the logarithm of the output of the discriminator. Taking the logarithm helps to rescale the discriminator's output probabilities.

-log(D(G(z, c), c)): The negative sign is applied to the logarithm result. This is carried out in order to maximize the likelihood that the discriminator would identify the created image as real. By minimizing the negative logarithm, the generator is encouraged to produce images that have a high probability of being classified as real.

2.5.2 Discriminator Network

The discriminator network accepts generated images generated by Generator, word embeddings, and the corresponding real image as input [7]. The goal of the discriminator network is to classify whether the image is real (matches the text description) or fake (generated by the generator). The discriminator network helps train the generator by providing feedback on the generated images. The discriminator receives as inputs conditional information (text) and the actual image (x). It aims to classify whether the input is real or generated and represented by equation 3.

D: x, c-------->D(x, c) ----------> (3)

The discriminator aims to correctly classify real text-image pairs from generated ones. The discriminator loss penalizes the discriminator for misclassifying real and generated pairs. It can be formulated as a binary cross-entropy loss:

D_L = -log(D(x, c)) – log(1-D(G(z, c), c)) ----------> (4)

Equation 4 measures how well the discriminator is distinguishing between real images and generated images when conditioned on the corresponding textual information. During training, the discriminator aims to minimize this loss, which encourages it to be more accurate in its classification [7].

D(x, c): represents the output of the discriminator when given a real image x and its corresponding conditional information c (text). The discriminator aims to classify whether this input pair is real.

Log (D(x, c)): calculates the logarithm of the output of the discriminator for the real image. Taking the logarithm helps to rescale the discriminator's output probabilities.

1 - D(G(z, c), c): denotes the output of the discriminator for the generated image G(z, c) and the corresponding conditional information c. It calculates the probability that the discriminator correctly classifies the generated image as fake. The likelihood that the discriminator will classify the generated image as real can be obtained by subtracting this number from 1.

Log (1 - D(G(z, c), c)): This calculates the logarithm of the probability that the discriminator classifies the generated image as fake.

-Log (D(x, c)) - log(1 - D(G(z, c), c)): The negative sign is applied to the sum of the logarithms. This is done because the goal of the discriminator is to maximize the probability of correctly classifying real and generated images. By minimizing the negative logarithm, the discriminator is encouraged to improve its classification accuracy.

3.6 Model Architecture

The architecture of Amharic text-to-image generation using CGAN has the following components: noise victor, Amharic text embedding the generator network having input layers for the text input and noise input, convolution layers with up-sampling layers and a final convolutional layer to generate the output image, the discriminator network having input layers for the text input and image input, convolution layers with down-sampling layers, a final fully connected layer to predict the probability, the loss function of the generator to deceive the discriminator and the discriminator to distinguish between a real and generated image, and evaluating the CGAN model by generating images from randomly selected noise vectors and text conditions. The following figure 4 shows the model architecture and where:

Fake image = generated image by the Generator

Real image = the preprocessed image

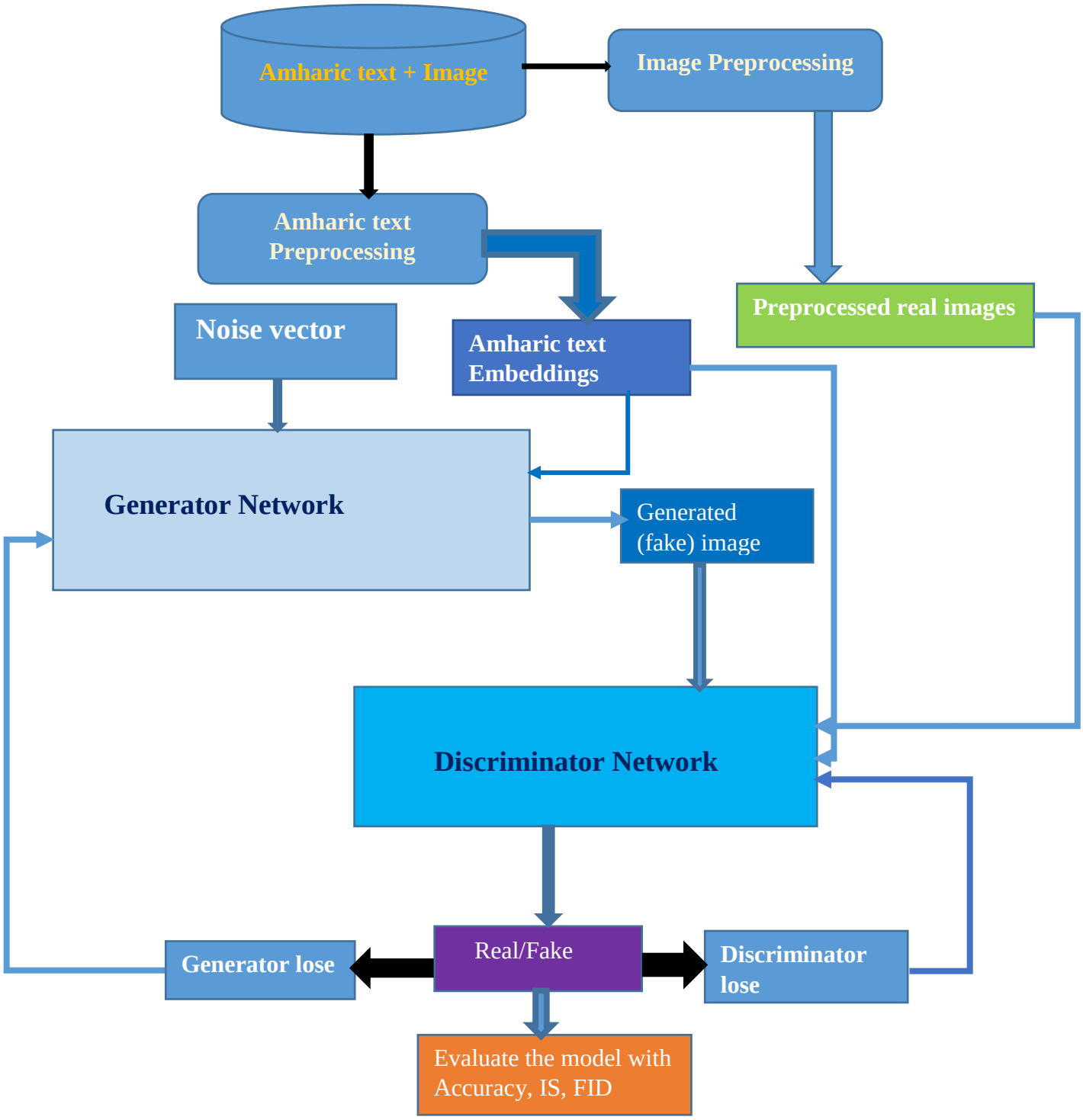

Figure 3. 2 Model Architecture

3.6.1 Noise vector

The noise vector is an important component that plays a crucial role in the image generation process[52]. The noise vector is a random vector of numbers sampled from a probability distribution, such as a uniform or Gaussian distribution [53]. It functions as an additional input to the generator in a CGAN and a source of randomness. The purpose of the noise vector is to introduce stochasticity and generate diverse images from the same Amharic text description. By

combining the Amharic text embedding with different noise vectors, the generator can produce multiple plausible interpretations of the same text description. This allows for the generation of diverse images that capture different aspects or styles associated with the given Amharic text.
The noise vector can be manipulated to control specific characteristics of the generated images. By modifying the values in the noise vector, we can influence attributes such as image style, color, texture, or other visual features. Experimenting with different noise vectors enables the generation of images with desired visual characteristics while keeping the underlying text description constant. The noise vector provides the generator with the freedom to explore different variations, while the Amharic text embedding guides the generation process based on the semantic content of the text. Finding the right balance between noise and text information is crucial to ensure that the generated images are both diverse and relevant to the given text [30].

3.6.2 Amharic Word Embeddings

Amharic word embeddings refer to vector representations of Amharic words in a continuous vector space[28]. These word embeddings capture semantic relationships between words by placing them in a multi-dimensional space where similar words are positioned close to each other. These embeddings are converted by training Word2Vec after the preparation of Amharic texts. The generator learns to create images based on the semantic content of the Amharic text descriptions by using Amharic text embeddings as the conditioning input. This makes sure that the generated visuals match the text's intended meaning and are contextually relevant.Word2Vec embeddings reduce these variations by translating various word surface shapes to related vector representations.

3.6.3 The generator network

The generator network has two inputs. These inputs are the noise vector and Amharic word embeddings. Based on these inputs, the generator generates fake images (generated images) to be inputted to the discriminator for determination. In its architecture, the generator network has the following typical components:

Two input layers, one for noise_input having the shape of the noise dimension and the other for text_input (text embedding), have the shape of the text embedding dimension.

The generator frequently consists of multiple convolutional layers after the input layers. These layers are in charge of taking the input text's hierarchical representations and turning them into a

feature space that can be utilized to create images. Typically, the convolutional layers use filters to extract and learn different kinds of visual characteristics. Furthermore, the feature maps' spatial dimensions are progressively increased by the use of upsampling layers. By expanding the width and height of the feature maps, upsampling aids in the creation of images with a greater resolution.

The output image is produced by the generator network's last convolutional layer. For upsampling the feature maps to the required image dimensions, it employs a transposed convolution or deconvolution procedure. The created image is represented by this layer's output, which is subsequently transferred to the discriminator for analysis.

3.6.4 Discriminator network

The discriminator network is responsible for classifying an image as real or fake in training after the generator generates a fake image, and this fake image is fed to the discriminator with the real image and the corresponding Amharic text embeddings. In the architecture, the discriminator network has the following typical components:
The discriminator has two input layers, one for image input and the other for text input or text embedding. The shape of the image is similar to the shape of the image generated by the generator.

Following the input layers, the discriminator consists of several convolutional layers. These layers are responsible for learning hierarchical representations from the input image and extracting discriminative features. The convolutional layers utilize filters to capture different visual patterns and features present in the images. Additionally, down-sampling layers such as stride convolutions are used to minimize the spatial dimensions of the feature maps. Down-sampling reduces processing complexity while assisting in the capture of high-level information and global context.

The last layer of the discriminator is a fully connected layer. This layer determines whether the input image is created or real by processing the flattened feature maps from the preceding convolutional layers. The fully connected layer applies transformations and activations to the features and produces a single scalar output, indicating the discriminator's certainty that the input image is real or fake. Accurate image classification is taught to the discriminator network, which can differentiate between created images from the generator and actual images from the real images. By optimizing the discriminator's parameters, the network learns to identify the features and characteristics that distinguish real images from generated ones.

3.6.5 Generator lose

The generator in a CGAN is responsible for generating fake images based on the input Amharic text descriptions and the noise vector. The goal of the generator is to produce images that are indistinguishable from real images. The generator loss quantifies the difference between the generated images and real images. The adverse loss function computed using binary cross-entropy loss is used to quantify the loss of the generator, and loss information is important to adjust the training parameters, such as the number of epochs, batch size, and an optimizer.

3.6 .6 Discriminator loss

The discriminator is responsible for distinguishing between real and generated (fake) images. It learns to classify real images as real and generated images as fake. The discriminator loss quantifies the difference between the discriminator's predictions and the ground truth labels. Adversarial loss computed using binary cross-entropy loss measures the ability of the discriminator to correctly classify real and generated images. It quantifies the loss of the discriminator, whether it classifies correctly or not, by minimizing the loss. This loss information is also important to adjust the training parameters, such as the number of epochs, batch size, and an optimizer.

3.7 Implementation tools

For text and image processing, preprocessing, and model development, the following implementation tools have been used:

Python: serves as the programming language for implementing the development of the model for Amharic text-to-image generation to handle data preprocessing, training the model, and evaluating the results

Google Colab: Python programming environment with cloud computing resources. It allows us to write and execute Python code in a web browser, eliminating the need for local setup and providing access to hardware resources like disk space and GPUs. The following Python libraries were used in the development of the model:

CSV (Comma-Separated Values): a Python library that provides functionality for reading and writing files. It is used to handle Amharic text descriptions and corresponding image filenames.

Pandas: a Python library for data manipulation and analysis and provides data structures to create data frames that allow us to efficiently manipulate, process, preprocess, and analyze Amharic text and image data before feeding it into the CGAN model.

Os: a Python library that provides a way to interact with the operating system and allows us to perform various operations such as accessing directories, manipulating file paths, and checking file existence.

Nltk (Natural Language Toolkit): a Python natural language processing package that offers preprocessing functions for Amharic text, such as stop word removal and tokenization.

re (regular expression): a Python library that provides tools for text manipulation, allowing us to define patterns and search for specific text patterns within strings. It is used to perform text preprocessing tasks, such as removing special characters or cleaning up the Amharic text data.

PIL (Python Imaging Library): a Python library that provides functionality for image processing and manipulation. It allows to open, resize, and save images. It is used to preprocess and transform image data before feeding it into the CGAN model.

matplotlib: a Python library used for data visualization, providing a wide range of plotting functions and tools to create visualizations for images and to visualize the generated images or plot training and testing metrics during the model training process.

Tensorflow: is a widely-used deep learning framework used for defining, building, training CGAN model and performing operations on tensors.

Genism: a Python library for creating Amharic word vector representations by training Word2Vec on Amharic text data to capture semantic relationships between Amharic words, which can be helpful for generating meaningful image representations from text.

Numpy: a Python library we used for data manipulation tasks, such as handling image data, reshaping tensors, and performing mathematical operations during the model training.

cv2 (OpenCV (Open Source Computer Vision Library)): a Python library that provides functions to load and save image files in various formats and to perform image preprocessing such as resizing.

Sklearn (scikit-learn): a Python library we used to perform tasks such as data splitting and evaluation of the CGAN model's performance.

3.8 Evaluating the Model

In training and after training is completed, evaluating how well each network is performing during the training process and evaluating the trained model by generating images from new Amharic text descriptions and assessing their quality and coherence is important. Evaluation metrics like accuracy quantify the training process, and Inception Score (IS) and Fréchet Inception Distance (FID) are used to quantitatively measure the generated image quality.

The training accuracy of the generator indicates how well it is learning to generate images that are convincing and resemble the desired output. Generator's high training accuracy means that, given Amharic text input, the Generator is successful in producing images that match the target distribution.

The training accuracy of the discriminator represents its ability to correctly classify real and generated/fake images. Discriminator's high training accuracy means that it is effectively learning to differentiate between real and generated images.

The effectiveness and variety of images created are evaluated by the Inception Score. It assesses the degree to which the generated images are diverse and how well they adhere to the target distribution [30]. The KL (Kullback-Leibler) divergence between the conditional class distribution and the marginal class distribution provided by the Inception network is calculated as part of the IS metric. Better image quality and diversity are indicated by higher IS ratings.

Using feature representations retrieved from an Inception network, the Fréchet Inception Distance evaluates the similarity between the generated images and the original images[54]. FID evaluates the distribution of generated and real images in feature space to assess both image quality and diversity. Better image quality and resemblance to real images are indicated by a lower FID score.

CHAPTER FOUR

Model Development and Discussion

4.1 Introduction

In this chapter, data acquired, data preprocessing, dataset description, model development, detailed implementation procedures, parameters for training, and experimental results are presented.

4.2 Data acquired

For this study, a total of 2575 images of shoes and clothes with the corresponding Amharic descriptive texts were acquired and Amharic texts are written manually by describing the visual detail of each image.

The Amharic text is prepared as an Excel file with three columns: “Text” containing the Amharic descriptive text, “Image” containing the images, the file path or image path, and “category” containing the category of the image, whether it is apparel or footwear. The corresponding image data are arranged according to their filenames or image paths put in the "Image" column of the Amharic text dataset. The text dataset is formatted as a.csv file after being loaded into Google Collab in order to be easily parsed by a deep learning algorithm.

4.4 Data preprocessing

Following the proper organization of the acquired data, data preparation is the next and most important step. There are two types of data preparation for this study. Amharic text preprocessing, which involves Amharic text preprocessing such as stop word removal, punctuation mark removal, Amharic text tokenization, and converting Amharic text to word embeddings. The second one is image data preprocessing, which involves noise removal, image segmentation, image resizing, and image normalizing, and converting the resized and normalized image to Numpy arrays.

4.4.1. Amharic stop words removal

Stop words are often used words that lack major semantic value [12]. They are frequently eliminated to cut down on background noise and draw attention to the text's more crucial terms.

To remove stop words from Amharic text, we need to compile a list of Amharic stop words specific to the Amharic language. This list can be created by identifying frequently occurring words in a representative Amharic language. Figure 4.1 shows a list of Amharic stop words that are probably found in the collected Amharic text for this study.

Figure 4. 1 Snipped code to declare Amharic stop words

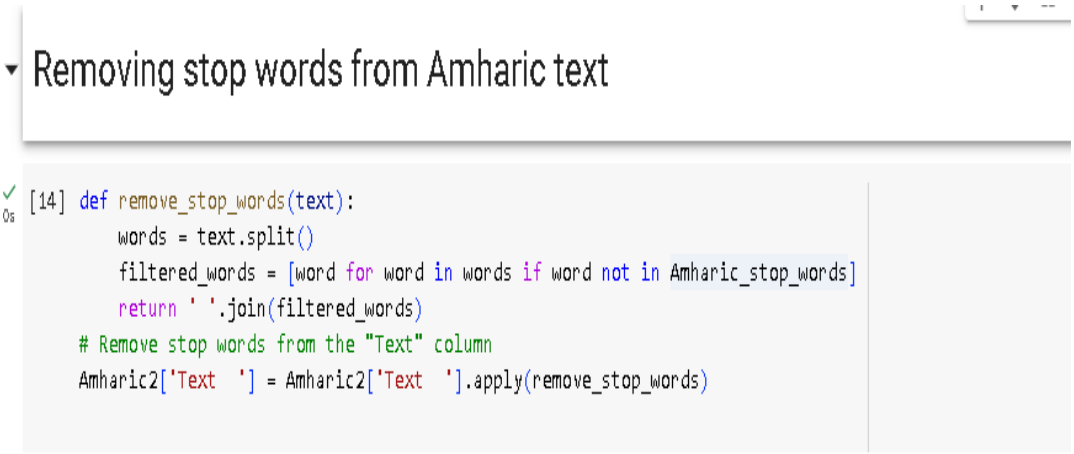

Figure 4. 2 Snipped code to remove Amharic stop words

4.4.2 Amharic punctuation marks removal

In writing, punctuation marks are symbols that indicate pauses, emphasis, and other linguistic elements [33]. To remove punctuation marks from Amharic text, it is important to compile a list

of some Amharic punctuation marks specific to Amharic language. Figure 4.3 shows snipped code to list Amharic punctuation marks and remove them from the Amharic text dataset.

Removing Amharic punctuation marks

```
[33] import re
     # Define a function to remove Amharic punctuation marks
     def remove_amharic_punctuation(text):
         # Define the Amharic punctuation marks to be removed
         Amharic_punctuation_marks={ '።', '፤', '?','፣','፦ ','»','!','፧'}
         # Remove the Amharic punctuation marks from the text
         Cleaned_Text = ''.join([c for c in text if c not in Amharic_punctuation_marks])
         return Cleaned_Text
```

```
[34] # Remove Amharic punctuation marks from the 'Text' column and adding column "Cleaned_Text" in the dataframe
     Amharic2['Cleaned_Text']= Amharic2['Text  '].apply(remove_amharic_punctuation)
```

Figure 4. 3 Snipped code to remove Amharic punctuation marks

4.4.3 Amharic text Tokenization

Tokenization is the process of dividing text into discrete pieces known as tokens [33]. In order to create a model for text-to-image generation using conditional generative adversarial networks, tokenization is a crucial step. Tokens are often words or sub-word units in the context of NLP and feed into the Word2Vec model by dividing it up into tokens. Tokenization also aids in controlling the vocabulary size and lowering the model's processing needs. Before training Word2Vec to create Amharic text embeddings, the cleaned Amharic text should be converted to a list of words called tokens. Figure 4.4 shows snipped code to convert cleaned text to tokens.

Tokenizing the column text

```
# Tokenize the 'Tokenized_Text' column
Amharic2['Tokenized_Text'] = Amharic2['Cleaned_Text'].apply(word_tokenize)
```

Figure 4. 4 Snipped code to tokenize Amharic text

index	Tokenized_Text	Image_Path	categry
0	ነጭ,ጥቁር,ቀለም,ያለው,የአዋቂ,ወንዶች,የስፖርት,ናይክ,ጫማ	/content/drive/MyDrive/Dataset/images/ 1636.jpg	1
1	ነጭ,ቀይ,ቀለም,ያለው,የአዋቂ,ወንዶች,የስፖርት,ናይክ,ጫማ	/content/drive/MyDrive/Dataset/images/ 1637.jpg	1
2	ቡናማ,ቀለም,ያለው,የአዋቂ,ወንዶች,የስፖርት,ሪቡክ,ጫማ	/content/drive/MyDrive/Dataset/images/ 1653.jpg	1
3	ድፍን,ነጭ,ቀለም,ያለው,የአዋቂ,ወንዶች,የስፖርት,ሪቡክ,ጫማ	/content/drive/MyDrive/Dataset/images/ 1654.jpg	1
4	ቡናማ,የአዋቂ,ወንዶች,የተለመደ,ከፍት,ጫማ	/content/drive/MyDrive/Dataset/images/ 1806.jpg	1
5	ጥቁር,ነጭ,ቀለም,ያለው,የአዋቂ,ወንዶች,የስፖርት,ናይክ,ጫማ	/content/drive/MyDrive/Dataset/images/ 1831.jpg	1
6	ጥቁር,ማሰሪያ,ያለው,ብር,የአዋቂ,ወንዶች,የስፖርት,ናይክ,ጫማ	/content/drive/MyDrive/Dataset/images/ 1836.jpg	1
7	ጥቁር,ደማቅ,ሰማያዊ,ቀለም,ያለው,የአዋቂ,ወንዶች,የተለመደ,ከፍት,ጁማ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2211.jpg	1
8	ጥቁር,ቢጫ,ቀለም,ያለው,የአዋቂ,ወንዶች,የተለመደ,ከፍት,ጁማ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2218.jpg	1
9	ጥቁር,ነጭ,ቀለም,ያለው,የአዋቂ,ወንዶች,የተለመደ,አዲዳስ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2219.jpg	1
10	ጥቁር,ቀለም,ያለው,የአዋቂ,ወንዶች,የተለመደ,ከፍት,ጁማ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2220.jpg	1
11	ግራጫ,ጥቁር,ቀለም,ያለው,የአዋቂ,ወንዶች,የተለመደ,ከፍት,ጫማ	/content/drive/MyDrive/Dataset/images/ 2227.jpg	1
12	ሰማያዊ,ቀለም,ያለው,የአዋቂ,ወንዶች,የስፖርት,ናይክ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2477.jpg	1
13	ነጭ,ቀለም,ያለው,የአዋቂ,ወንዶች,የተለመደ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2504.jpg	1
14	ወርቃማ,ቀለም,ያለው,የአዋቂ,ሴቶች,የተለመደ,ተረከዝ,ነጠላ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2616.jpg	1
15	ቀይ,የአዋቂ,ሴቶች,የተለመደ,ተረከዝ,ከፍት,ጫማ	/content/drive/MyDrive/Dataset/images/ 2626.jpg	1
16	ወርቃማ,ብር,ቀለም,ያለው,የአዋቂ,ሴቶች,የተለመደ,ተረከዝ,ጫማ	/content/drive/MyDrive/Dataset/images/ 2628.jpg	1
17	ሰማያዊ,ቀለም,ያለው,የወንዶች,ኦጭር,ቁምጣ	/content/drive/MyDrive/Dataset/images/ 2691.jpg	0
18	ሰማያዊ,ኪስ,ያለው,ቢጫ,የወንድ,ህጻናት,ሸሚዝ	/content/drive/MyDrive/Dataset/images/ 2693.jpg	0
19	ቀይ,ቀለም,ያለው,የወንዶች,ሸሚዝ	/content/drive/MyDrive/Dataset/images/ 2694.jpg	0

Figure 4. 5 Snipped sample of tokenized text with image file name

4.4.4 Amharic text embedding using Word2Vec

There are parameters to train Word2Vec with Amharic text, and each parameter is assigned according to the dataset [27] . In the context of training word embeddings, training parameters refer to the configuration settings or hyperparameters that can be specified before beginning the training process [28]]. These parameters determine the characteristics of the word embeddings that will be learned during training. Each parameter serves a specific purpose in shaping the behavior of the training algorithm. Skip-gram is used to perform better with rare words because it predicts the context words given a target word[49]. This approach provides a more nuanced understanding of infrequent words. It focuses on predicting context words for a given target word, capturing a richer representation of the word's context .Generally, it is more effective with smaller datasets The following shows the parameter values for training Word2Vec with Amharic texts.

Table 4. 2 Training parameters for Amharic word embedding using Word2Vec [28]

Parameter	Value
Vector Size (Embedding dimension)	100
Window Size	5
Min Count	1
workers	4
Skip-gram(sg)	1

Vector Size: refers to the dimensionality of the word vectors. Each word in the vocabulary will be represented as a vector in a space of this dimension.

Window Size: refers to the maximum distance between the current and predicted words within a sentence.

Minimum Count: ignores all words with a total frequency lower than this and helps filter out rare words that cannot contribute much to the embeddings.

Number of Workers (workers): The number of CPU cores to use for training.

Algorithm sg (Skip-gram): if sg equals 1, the skip-gram model is used; if sg equals 0, the continuous bag of words (CBOW) is used. Figure 4.6 shows a snipped code to train Word2Vec using skip gram with Amharic text.

Training Word2Vec using Skip-gram

```
from gensim.models import Word2Vec
# Replace 'amharic_sentences' with your list of sentences
# Train Word2Vec model
model = Word2Vec(sentences=Amharic2['Tokenized_Text'], vector_size=100, window=5, min_count=1, workers=4)
# Save the trained model
model.save("Amharic_word2vec.model")
```

Figure 4. 6 Snipped code to convert Amharic text to word embeddings

	Tokenized_Text	Word_Embeddings
0	[ኦም, ጥቁር, ቀነም, ያነው, የአዋቂ, ወንዶች, የስፖርት, ናይክ, ጫማ]	[[-0.07420161, 0.12236383, -0.0019241202, 0.05...
1	[ኦም, ቀይ, ቀነም, ያነው, የአዋቂ, ወንዶች, የስፖርት, ናይክ, ጫማ]	[[-0.07420161, 0.12236383, -0.0019241202, 0.05...
2	[ቡናማ, ቀነም, ያነው, የአዋቂ, ወንዶች, የስፖርት, ፊቡክ, ጫማ]	[[-0.046815448, 0.090976425, -0.03406281, 0.05...
3	[ደፍን, ኦም, ቀነም, ያነው, የአዋቂ, ወንዶች, የስፖርት, ፊቡክ, ጫማ]	[[-0.031975858, 0.0673523, -0.0049534324, 0.03...
4	[ቡናማ, የአዋቂ, ወንዶች, የተሰመደ, ክፍት, ጫማ]	[[-0.046815448, 0.090976425, -0.03406281, 0.05...
5	[ጥቁር, ኦም, ቀነም, ያነው, የአዋቂ, ወንዶች, የስፖርት, ናይክ, ጫማ]	[[-0.05177219, 0.08864595, -0.018674314, 0.065...
6	[ጥቁር, ማሰሪያ, ያነው, ብር, የአዋቂ, ወንዶች, የስፖርት, ናይክ, ጫማ]	[[-0.05177219, 0.08864595, -0.018674314, 0.065...
7	[ጥቁር, ደማቅ, ሰማያዊ, ቀነም, ያነው, የአዋቂ, ወንዶች, የተሰመደ, ...	[[-0.05177219, 0.08864595, -0.018674314, 0.065...
8	[ጥቁር, ቢጫ, ቀነም, ያነው, የአዋቂ, ወንዶች, የተሰመደ, ክፍት, ጭማ...	[[-0.05177219, 0.08864595, -0.018674314, 0.065...
9	[ጥቁር, ኦም, ቀነም, ያነው, የአዋቂ, ወንዶች, የተሰመደ, አዲዳስ, ጫማ]	[[-0.05177219, 0.08864595, -0.018674314, 0.065...
10	[ጥቁር, ቀነም, ያነው, የአዋቂ, ወንዶች, የተሰመደ, ክፍት, ጭማ, ጫማ]	[[-0.05177219, 0.08864595, -0.018674314, 0.065...
11	[ሻራጫ, ጥቁር, ቀነም, ያነው, የአዋቂ, ወንዶች, የተሰመደ, ክፍት, ጫማ]	[[-0.052886907, 0.07256186, -0.021537706, 0.05...
12	[ሰማያዊ, ቀነም, ያነው, የአዋቂ, ወንዶች, የስፖርት, ናይክ, ጫማ]	[[-0.070293725, 0.10531454, -0.01427987, 0.071...
13	[ኦም, ቀነም, ያነው, የአዋቂ, ወንዶች, የተሰመደ, ጫማ]	[[-0.07420161, 0.12236383, -0.0019241202, 0.05...
14	[ወርቃማ, ቀነም, ያነው, የአዋቂ, ሴቶች, የተሰመደ, ተረከዝ, አጠላ, ጫማ]	[[-0.043764982, 0.060139485, -0.016293349, 0.0...
15	[ቀይ, የአዋቂ, ሴቶች, የተሰመደ, ተረከዝ, ክፍት, ጫማ]	[[-0.0512754, 0.118251525, 0.0004362301, 0.047...
16	[ወርቃማ, ብር, ቀነም, ያነው, የአዋቂ, ሴቶች, የተሰመደ, ተረከዝ, ጫማ]	[[-0.043764982, 0.060139485, -0.016293349, 0.0...
17	[ሰማያዊ, ቀነም, ያነው, የወንዶች, አጭር, ቁምጣ]	[[-0.070293725, 0.10531454, -0.01427987, 0.071...
18	[ሰማያዊ, ኪስ, ያነው, ቢጫ, የወንድ, ህፃናት, ሸሚዝ]	[[-0.070293725, 0.10531454, -0.01427987, 0.071...
19	[ቀይ, ቀነም, ያነው, የወንዶች, ሸሚዝ]	[[-0.0512754, 0.118251525, 0.0004362301, 0.047...

Figure 4. 7 Sample snipped of tokenized text with the corresponding word embeddings

4.5 Image data preprocessing

The image data should be noise removed, segmented and resized according to the architecture of the model and the number of layers making up the model. For training purposes in this study, the image is resized to 128x128 pixels in size. The image data should also be normalized to standardize the range and converted to Numpy arrays for compatibility and efficiency. Resizing images to a standard scale that matches the computational capabilities and architecture of the model is very important to guarantee that all images have the same dimensions and aspect ratio [32] .

Normalizing the pixel values of the images to a standardized range, typically between 0 and 1 helps in stabilizing the training process and improves convergence[50].

Noise removal and Image segmentation

```
[ ]  import matplotlib.pyplot as plt
     import os
     # Directory containing the images
     image_directory = '/content/drive/MyDrive/Dataset/images'
     # Function to remove noise using Gaussian Blur
     def remove_noise(image):
         return cv2.GaussianBlur(image, (5, 5), 0)
     # Function to segment the image using Otsu's thresholding
     def segment_image(image):
         gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
         ret, segmented = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
         return segmented
     # Function to display images without axis
     def display_images_without_axis(original, denoised, segmented):
         fig, axes = plt.subplots(1, 3, figsize=(15, 5))
         axes[0].imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
         axes[0].set_title('Original Image')
         axes[0].axis('off')  # Turn off axis
         axes[1].imshow(cv2.cvtColor(denoised, cv2.COLOR_BGR2RGB))
         axes[1].set_title('Denoised Image')
         axes[1].axis('off')  # Turn off axis
         axes[2].imshow(segmented, cmap='gray')
         axes[2].set_title('Segmented Image')
         axes[2].axis('off')  # Turn off axis
         plt.tight_layout()
         plt.show()
     # Loop through the images in the directory
     for filename in os.listdir(image_directory):
         if filename.endswith(".jpg") or filename.endswith(".png"):  # Adjust the file extensions as needed
             image_path = os.path.join(image_directory, filename)
             image = cv2.imread(image_path)
             denoised_image = remove_noise(image)
             segmented_image = segment_image(denoised_image)
             display_images_without_axis(image, denoised_image, segmented_image)
```

Figure 4. 8 Snipped code to remove noise and image segmentation

Figure 4. 9 Snipped sample of denoised and segmented image

Converting the resized and normalized images to NumPy arrays in the context of text-to-image generation offers standardization, efficiency, compatibility, integration, and flexibility in data manipulation, which are all advantageous for training and generating images with the CGAN model. Figure 4.10 shows snipped code to resize 2575 images as (128, 128, 3) indicating height, width, and RGB color arrangement, respectively, to normalize and convert to numpy arrays.

Laoding and preprocessing Imaga data

```
import os
import numpy as np
import cv2
# Specify the directory where the image files are stored
image_directory = '/content/drive/MyDrive/Dataset/images'
# Specify the target image size for resizing
target_size = (128, 128)
# Creating an empty list to store the resized and normalized image data
real_image= []
# Iterate through the image files in the directory
for filename in os.listdir(image_directory):
 # Reading  the image file
    image_path = os.path.join(image_directory, filename)
    image = cv2.imread(image_path)
    # Resizing the image to the target size
    image = cv2.resize(image, target_size)
    # Normalizing the image by dividing 255.0
    image = image.astype(np.float32) / 255.0
    # Appending the preprocessed image to the list
    real_image.append(image)
# Converting the image data list to a numpy array
real_image = np.array(real_image)
# Printing the shape of the image data array
print("Image Data Shape:", real_image.shape)
```

```
Image Data Shape: (2575, 128, 128, 3)
```

Figure 4. 10 Snipped code to resize and normalize image

4.6 Model development

In the development of the model, the following actions are taken: Dataset splitting as training and testing, defining both the generator and discriminator architecture, compiling the generator and discriminator, and training the generator and discriminator with the training data set

4.6.1 Training and Testing data

According to [45] , the choice of the train-test split ratio depends on various factors, such as the size of the dataset, the complexity of the model, and the specific requirements of the task. However, a common split ratio is 80-20 and 70-30, which means using 80% or 70% of the data for training and the remaining 20% or 30% for testing. For this study, the preprocessed dataset was spliced for 70-30, 90-10, and 80-20 ratios, and each train-test ratio had its impact on training and testing the model. Finally, an 80-20 train-test ratio was used for this study because the model has better

performance than 70-30 and 90-10 train-test split ratios. In this case, 80% of the data was used for training, and 20% of the data was used for testing.

4.6.2 Defining generator and discriminator

The generator takes the noise vector having 100 dimensions and the text embeddings having 100 dimensions as input to generate a fake image, which can be an input to the discriminator to determine whether it is a fake or real image. The generator has the following layers in its definition:

Input Layers: Two input layers, one for noise input having the shape of noise dimension and the other for text input (text embedding) having the shape text_embedding_dim.

Concatenate layer: This layer concatenates the noise and the text embedding.

Dense Layer: The concatenated input is passed through a dense layer with ReLU activation to project it to a shape suitable for upsampling, and batch normalization is applied to normalize the activations here.

Reshape Layer: It converts the combined input to a 4D tensor.

Upsampling Blocks: The reshaped output in the reshape layer is then passed through a series of upsampling blocks, each having an UpSampling2D layer, followed by a 2D convolutional layer with ReLU activation and batch normalization.

Output layer: The final layer is a 2D convolutional layer with a tanh activation function, generating the output image having the shape (128, 128, 3) as specified.

The summary of the generator shows that there are 13,823,939 parameters in the model. Among these, 13,692,419 are trainable parameters that will be updated and optimized during the training process, and the remaining 131,520 are non-trainable parameters that remain constant during training and are not updated. Finally, the generator model takes both noise and text embedding as input and outputs a generated image. Figure 4.11 shows the summary of the generator model.

```
Generator summary
Model: "model_13"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_27 (InputLayer)       [(None, 100)]             0         []

 input_28 (InputLayer)       [(None, 100)]             0         []

 concatenate_4 (Concatenate  (None, 200)               0         ['input_27[0][0]',
 )                                                                 'input_28[0][0]']

 dense_6 (Dense)             (None, 65536)             13172736  ['concatenate_4[0][0]']

 batch_normaliration_8 (Bat  (None, 65536)             262144    ['dense_6[0][0]']
 chNormaliration)

 reshape_2 (Reshape)         (None, 16, 16, 256)       0         ['batch_normaliration_8[0][0]'
                                                                 ]

 up_sampling2d_6 (UpSamplin  (None, 32, 32, 256)       0         ['reshape_2[0][0]']
 g2D)

 conv2d_14 (Conv2D)          (None, 32, 32, 128)       295040    ['up_sampling2d_6[0][0]']

 batch_normaliration_9 (Bat  (None, 32, 32, 128)       512       ['conv2d_14[0][0]']
 chNormaliration)

 up_sampling2d_7 (UpSamplin  (None, 64, 64, 128)       0         ['batch_normaliration_9[0][0]'
 g2D)                                                            ]

 conv2d_15 (Conv2D)          (None, 64, 64, 64)        73792     ['up_sampling2d_7[0][0]']

 batch_normaliration_10 (Ba  (None, 64, 64, 64)        256       ['conv2d_15[0][0]']
 tchNormaliration)

 up_sampling2d_8 (UpSamplin  (None, 128, 128, 64)      0         ['batch_normaliration_10[0][0]
 g2D)                                                            ']

 conv2d_16 (Conv2D)          (None, 128, 128, 32)      18464     ['up_sampling2d_8[0][0]']

 batch_normaliration_11 (Ba  (None, 128, 128, 32)      128       ['conv2d_16[0][0]']
 tchNormaliration)

 conv2d_17 (Conv2D)          (None, 128, 128, 3)       867       ['batch_normaliration_11[0][0]
                                                                 ']

=================================================================================================
Total params: 13823939 (52.73 MB)
Trainable params: 13692419 (52.23 MB)
Non-trainable params: 131520 (513.75 KB)
```

Figure 4. 11 Generator model summary

The discriminator evaluates whether an input image is real or generated. The discriminator has the following layers, according to the definition:

Input Layers: There are two input layers, one for the image input and the other for text input or text embedding.

Downsampling layer: The image input goes through a series of convolutional layers with ReLU activation to down-sample the image spatially.

Flatten Layer: flattens the features obtained from the convolutional layers.

Concatenate layer: Concatenates the flattened features with the text embedding.

Dense Layers (Fully connected layers): The concatenated features are passed through dense layers with ReLU activation for classification.

Output layer: The final dense layer has a sigmoid activation to produce a single output representing the probability that the input is a real image or was generated.

The discriminator model takes both an image and a text embedding as input and outputs the probability of the input being a real image or generated. The discriminator has 33,977,473 parameters, all of which are trainable. Figure 4.11 shows the discriminator summary.

```
discriminator summary
Model: "model_14"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_29 (InputLayer)       [(None, 128, 128, 3)]     0         []

 conv2d_18 (Conv2D)          (None, 64, 64, 64)        1792      ['input_29[0][0]']

 conv2d_19 (Conv2D)          (None, 32, 32, 128)       73856     ['conv2d_18[0][0]']

 conv2d_20 (Conv2D)          (None, 16, 16, 256)       295168    ['conv2d_19[0][0]']

 flatten_2 (Flatten)         (None, 65536)             0         ['conv2d_20[0][0]']

 input_30 (InputLayer)       [(None, 100)]             0         []

 concatenate_5 (Concatenate  (None, 65636)             0         ['flatten_2[0][0]',
 )                                                                 'input_30[0][0]']

 dense_7 (Dense)             (None, 512)               3360614   ['concatenate_5[0][0]']
                                                       4

 dense_8 (Dense)             (None, 1)                 513       ['dense_7[0][0]']

=================================================================================================
Total params: 33977473 (129.61 MB)
Trainable params: 33977473 (129.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 4. 12 Discriminator model summary

4.6.3 Compiling generator and discriminator

Compiling generators and discriminators involves configuring parameters such as the optimization process, defining the loss functions, and defining metrics. Table 4.3 shows the parameters for compiling the generator and the discriminator in training.

Table 4.3 Parameters for Compiling Generator and Discriminator[55]

parameter	**Description**	**default value**
optimizer	Optimizer algorithm for both generator and discriminator.	Adam Optimizer (learning_rate=0.0002, beta_1=1.0)
loss	Loss function for both generator and discriminator	Binary Crossetropy
metrics	Evaluation metrics to be monitored during training.	['accuracy']

4.6.4 Training generator and discriminator

Here are two scenarios to train both the generator and the discriminator. Training with default values of parameters and updating the values of the parameters.

Training both the generator and discriminator networks involves feeding the noise vector and the text embeddings to the generator and feeding the generated images, the real images, and the text embeddings to the discriminator according to the architecture and definition of the models. Here are the definition of training parameters.

Epochs: The number of times the entire training dataset is passed forward and backward through the generator and discriminator.

Batch Size: The number of training samples used in one iteration.

Learning Rate: The step size at each iteration while moving toward a minimum of the loss function.

Loss Weight: The weight assigned to the generator loss and the discriminator loss in the combined loss function to balance the training process.

Table 4.4 Training parameters with default values for Generator and Discriminator [10]

Parameters	**values**
number of Epochs	100
Batch Size	64
Learning Rate	0.0002
Loss Weight	1.0

Table 4.5 Training parameters of Generator and Discriminator with updated values

Parameters	**values**
number of Epochs	1000
Batch Size	32
Learning Rate	0.0005
Loss Weight	0.5

4.7 Training result

4.7.1 Generator and Discriminator losses

The generator's loss is a measure of how well it has succeeded in fooling the discriminator. The discriminator loss is a measure of how well the discriminator is able to distinguish between real and generated images. Figure 4.13 shows the loss of the generator and the discriminator. The blue line shows the training loss of the discriminator throughout 1000 epochs, and the orange line shows the training loss of the generator throughout 1000 epochs. Generator loss shows a zero (0) value, meaning the generator is fooling the discriminator or the discriminator could not distinguish the generated images.

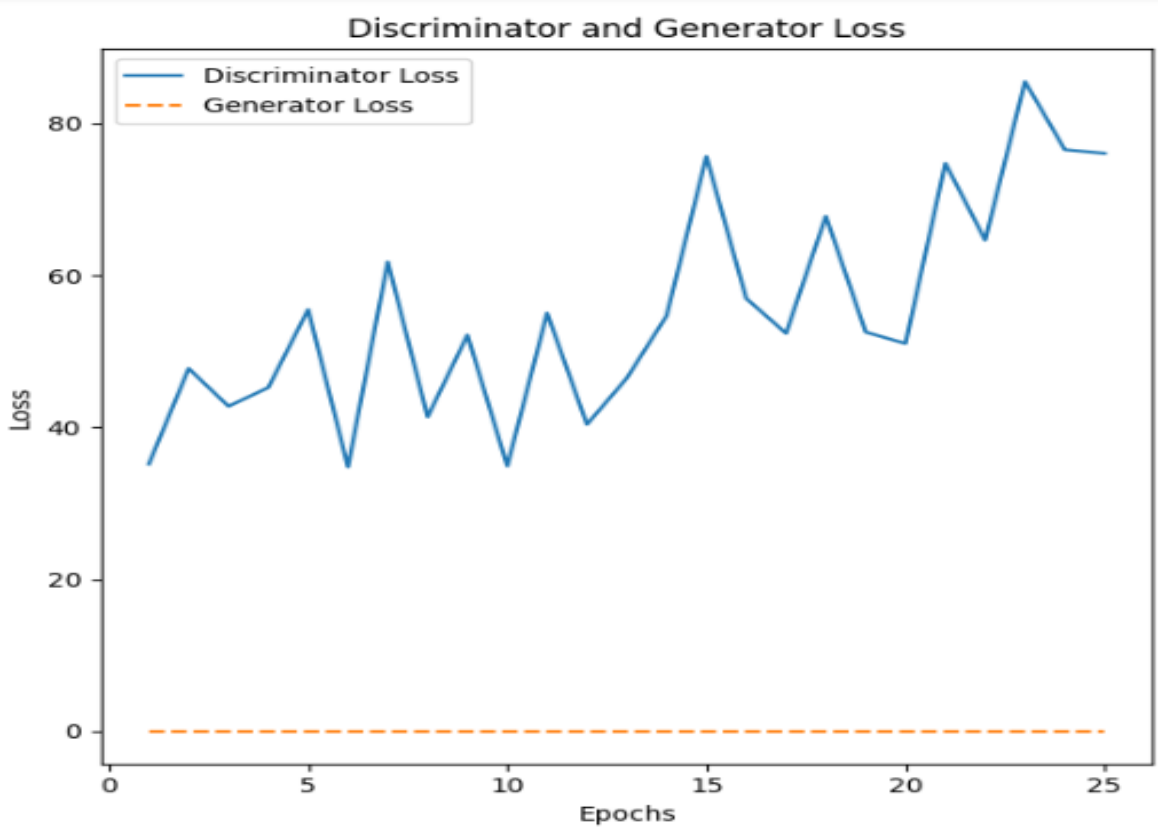

Figure 4. 13 Generator and Discriminator loss

4.7. 2 Generator and Discriminator accuracies

Generator accuracy and discriminator Accuracy refers to the accuracy of the generator and discriminator networks during the training process. Generator accuracy indicates how well the generated images match the desired distribution or characteristics specified by the conditional information. Discriminator accuracy measures how well the discriminator can correctly classify samples as real or fake. Figure 4.14 shows generator and discriminator accuracies. The generator achieved100% accuracy, meaning that it is successful in generating samples that are indistinguishable by the discriminator, and the discriminator achieved 40–50% accuracy, meaning that it can distinguish between real and fake images only at 40–50 percent.

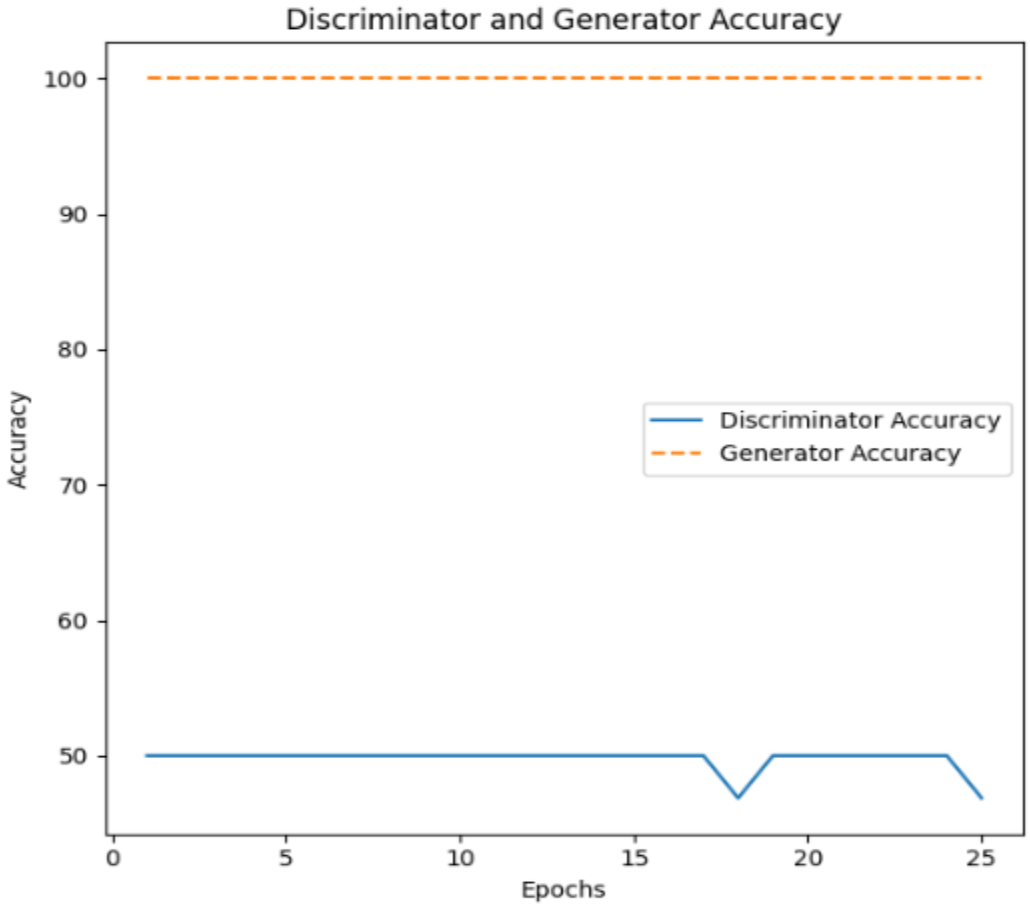

Figure 4. 14 Generator and Discrimination accuracy

4.8 Test results

After training the generator and the discriminator with the training set of the dataset and the specified parameters above, the generator should be tested with the testing set. The Frechet Inception Distance (FID) and Inception Score (IS) are used to evaluate the quality of images generated by a generator model. FID measures the similarity between the distributions of real and generated images. The IS score evaluates the quality and diversity of generated images based on their predicted class labels.

The FID score for the generator model is 4.99e+108, suggesting that there is a dissimilarity between the distributions of real and generated images since a lower FID score indicates better similarity between the distributions, with 0 being a perfect match.

The IS score of the generator model is 417.2, which is relatively high and suggests that the generated images are diverse according to the Inception Score metric.

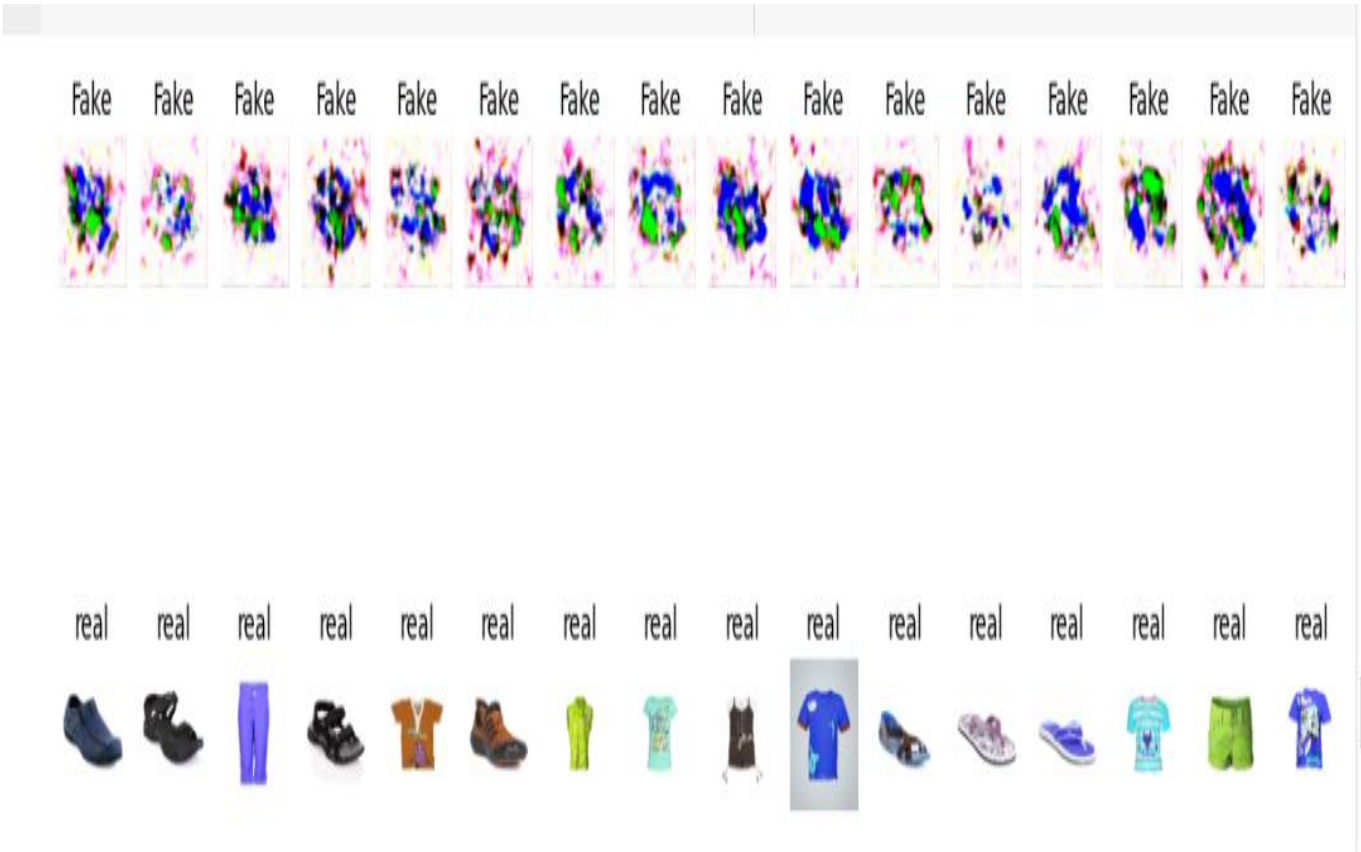

Figure 4. 15 Fake image generated by generator vs real image

Figure 4.16 shows the comparison of generated (fake) images generated by Generator tested with the testing text set and the testing image data (the real testing images). As the figure shows, there is a difference between the fake images and the real testing images.

4.9 Discussion

For the development of a model for Amharic text-to-image generation using CGAN, Amharic texts and the corresponding images were collected specifically for this purpose. Some of the fashion images were collected from different sites, as indicated above, and some of them were captured by smartphones. These fashion images include a variety of styles of shoes and apparel. The Amharic descriptive texts were written manually, describing the visual detail of each image. The Amharic text was organized with three columns: “text” containing the Amharic text, the “Image” column containing the path or file name of each image, and the “category“ column containing the category of the image, whether it is apparel or footwear. The image was also arranged in the order of their filenames in the image column of the Amharic text dataset. Following the proper organization of both the text dataset and image dataset, text cleaning, text tokenization, and converting to word embeddings were important for the Amharic text dataset, and image resizing, normalization, and converting the resized and normalized image to a numpy array were done for the image dataset.

Writing Amharic descriptive text for each fashion image is a time-consuming task and a challenging process of this study. Amharic text-to-image generation is a conditional task where the generated image is highly dependent on the input text. CGAN is well-suited for conditional tasks as it allows the incorporation of additional information (text) during the generation process. In the case of Amharic text, the conditional input (Amharic text) guides the model to produce images relevant to the provided text. The generator and discriminator architectures can be chosen with flexibility using CGAN.

The proposed model is compared with other models developed for other languages to show the originality of this study as follows.

The work by [23] used COCO-CN data and 8,000 Flickr8k-CN image data and proposed a transformer-based TIS (DALL-E) to enhance text-to-image generation for Chinese language and achieved an inception score of 14.71, a Frechet inception distance score of 66.66 in COCO-CN

data, and an inception score of 15.01, a Frechet inception distance score of 49.42 in Flickr8k-CN in this work the authors addressed the problem of language-specific text-to-image synthesizer benchmarks for Chinese, together with high-performing models with moderate sizes.

In order to generate high-resolution, photo-realistic images, the other study by [25] proposed stacked generative adversarial networks (StackGANs) using MS COCO data and attained an inception score of 9.74. In this work, the problem of creating 256x256 photo-realistic images conditioned on text descriptions was addressed.

Similarly, the study by [36] used 11,788 CUB-bird images and 123,287 MS COCO images and proposed a novel controllable text-to-image generative adversarial network (ControlGAN) and achieved an inception score of 4.58 on CUB-bird image data and an inception score of 24.06 on MS-COCO image data, which can effectively synthesize high-quality images and also control parts of the image generation according to natural language descriptions.

The study by [37] used 11,788 images from 200 bird species and proposed GAN-INT-CLS (Generative Adversarial Network with Integrated Classification) for the enhancement of text-to-image generation in the English language with an inception score of 5.26. The LI distance loss function, feature matching, and text conditioning augmentation are among the improvements. The model is encouraged to create more photo-realistic images and enhance the variations in image content by the feature matching. The text conditioning augmentation expands the text embedding feature space to improve the semantic consistency of the model.

Another by [38] used the CUB-birds dataset and the MS-COCO dataset and proposed an Attentional Generative Adversarial Network (AttnGAN) that allows attention-driven, multi-stage refinement for fine-grained text-to-image generation and achieved an Inception score of 4.29 on the CUB-birds image data and 25.89 on the MS-COCO. The model can synthesize fine-grained details in different sub-regions of the image by paying attention to the relevant words in the natural language description.

The study by [39] used MS COCO image data and proposed CogView, a 4-billion-parameter transformer with a VQ-VAE tokenizer, to advance text-to-image generation in Chinese and achieved an inception score of 17.9 and a Fréchet inception distance score of 85.0 and advance the problem of text-to-image generation in the general domain.

The work by [40] also used CUB-bird and MS COCO to address the problem of semantic consistency between the text description and visual content by proposing a novel global-local

attentive and semantic-preserving text-to-image-to-text framework known as MirrorGAN and achieved inception scores of 4.56 on the CUB-bird image and 26.47 on the MS COCO image.

The paper by [41] used CUB-bird image data and MS COCO image data and proposed a novel photo-realistic text-to-image generation model using SD-GAN (Semantics Disentangling GAN) that implicitly disentangles semantics to both achieve high-level semantic consistency and low-level semantic diversity and achieved an Inception score of 4.67 on CU-birds and 35.69 on MS-COCO images.

The proposed model used 2575 fashion images for Amharic text-to-image generation using CGAN and achieved an inception score of 417.2 and a Frechet inception distance of 4.99e+108. These numbers show that there is a great difference between the images generated by the trained model and the real images as shown in figure 4.16 above. Table 4.6 shows summary of the discussions.

Table 4.6 Summary of the Discussions

Ref No	**dataset(s)**	**Method**	**performance**
[23]	COCO-CN and 8,000 Flickr8k-CN image	TIS (DALL-E)	14.71 IS 66.66 FID in COCO-CN and 15.01 IS and 49.42 in Flickr8k-CN.
[25]	MS COCO	StackGANs	9.74 IS
[36]	11,788CUB-bird images and 123,287 MS COCO	ControlGAN	4.58 IS on CUB and 24.06 IS on MS COCO
[37]	11,788 CUB-bird images	GAN-INT-CLS	5.26 IS
[38]	MS COCO and CUB-bird images	AttnGAN	4.29 on CUB-bird and 25.89 on MS COCO
[39]	MS COCO	DALE-E	17.9 IS and 85.0 FID
[40]	MS COCO and CUB-bird images	MirrorGAN	4.56 IS on the CUB-bird and 26.47 on MS COCO
[41]	MS COCO and CUB-bird images	SD-GAN	4.67 IS on CU-birds and 35.69 on MS-COCO images.
Proposed model	2575 images of shoes and clothes	CGAN	417.2 IS and 4.99e+108 FID

In general, this study is critical to the area of NLP and some of the contributions were presented as follows:

The study shows how to develop a model for Amharic text-to-image generation using CGAN learning algorithms by preparing a dataset of Amharic texts with the corresponding images.

This research showed the possibility of image generation from Amharic text and that all researchers are busy studying Amharic text-image recognition. One of the major problems in machine learning, particularly in deep learning, is data, especially for Amharic language. Some datasets of Amharic texts with the corresponding fashion images were prepared to Amharic text-to-image generation because creating a dataset that contains Amharic text paired with corresponding images is a valuable contribution to the research community.

CHAPTER FIVE

CONCLUSION, FUTURE WORK AND RECOMMENDATION

5.1 Conclusion

In this study, a model for Amharic text-to-image generation using CGAN was proposed. This algorithm was trained with a pair of 2575 Amharic descriptive texts with corresponding images. The algorithm has two parts: the generator trained on the Amharic text embedding, which has the shape of the vector size in which its value is specified or assigned during the training of word2vec to convert preprocessed Amharic texts to word embeddings, and the noise vector to generate fake images. The discriminator network trained on the Amharic text embeddings, with the real images having the same shape as the generated images, classifies them as real and fake. These two networks trained each other so that the discriminator gives feedback to the generator and the generator generates an image that is indistinguishable from the discriminator. During training, the loss of the generator is 0, which means the generator generates images that are indistinguishable by the discriminator. The accuracy of the generator is much better than the accuracy of the discriminator, as figures 16 show. To say it another way, the generator is generating images, which makes the discriminator unable to distinguish between the real and the generated ones.

The training of the generator and the discriminator, as well as the testing of the generator with the testing data, have been done at all possible values of the parameters. The parameters for training Word2Vec (given in table 4. 2), the parameters for compiling the generator and the discriminator (given in table 4.3), and the parameters for training the generator and the discriminator (given in table 4.4 and 4.5) The loss and accuracy of both the generator and discriminator, the FID score and IS score of the generator, are relatively balanced at the given values of each parameter specified in each table stated above. In general, the training results and the testing results showed that the generator was better at training by achieving 0 loss and 100% accuracy to fool the discriminator and generate a fake image shown in figure 4.15 when tested with the testing text of the dataset against the real testing images of the dataset. Training both the generator and discriminator at the updated values of parameters is much better than the default values of parameters as it is seen in the testing results.

5.2 FUTURE WORK

Other researchers can use other deep learning algorithms, such as StackGAN, because it has two stages to increase the resolution of the image but requires more computational resources and a larger dataset.

Other researchers can use different types of word embeddings, such as Glove and FastText. Glove is basically used to construct word vectors that capture meaning in vector space and take advantage of global count statistics rather than just local information. Whereas FastText can better handle out-of-vocabulary words and capture morphological variants by taking into account sub-word information with large datasets

Other researchers can prepare enough datasets for this purpose on different domains. Focused on specific domains or visual attributes relevant to Amharic culture and context can lead to more targeted research.

5.3 Recommendation

Many tasks had been done to design a model for Amharic text-to-image generation using CGAN. It is common to prepare corpus for every NLP task for research purposes, like the English language has many prepared corpus for text-to-image generation. However, there was no corpus prepared for Amharic text-to-image generation. It is possible to develop a perfect model for Amharic text image generation with enough dataset, enough computational resources, and by using other variants of CGAN.

References

[1] W. Dinku, “Multi Label Amharic Text Classification Using Convolutional Neural Network Approaches,” 2020.

[2] R. S. Colavin, R. Levy, and S. Rose, “Modeling OCP-place in Amharic with the Maximum Entropy phonotactic learner,” *46th Meet. Chicago Linguist. Soc.*, 2010.

[3] “Amaric-English Dictionary.pdf.”

[4] D. Jurafsky and J. H. Martin, “Book Review Speech and Language Processing (second edition),” pp. 0–3, 2009.

[5] R. H. Wiggins, H. C. Davidson, H. R. Harnsberger, J. R. Lauman, and P. A. Goede, “Image file formats: past, present, and future.,” *Radiographics*, vol. 21, no. 3, pp. 789–798, 2001, doi: 10.1148/radiographics.21.3.g01ma25789.

[6] V. Tyagi, “Understanding Digital Image Processing,” *Underst. Digit. Image Process.*, no. November, 2018, doi: 10.1201/9781315123905.

[7] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” pp. 1–7, 2014, [Online]. Available: http://arxiv.org/abs/1411.1784

[8] “A Gentle Introduction to Generative Adversarial Networks... - Google Scholar.” https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=A+Gentle+Introduction+to+Generative+Adversarial+Networks+%28GANs%29&btnG= (accessed Apr. 24, 2023).

[9] M. Farajzadeh-Zanjani, R. Razavi-Far, M. Saif, and V. Palade, “Generative Adversarial Networks: A Survey on Training, Variants, and Applications,” *Intell. Syst. Ref. Libr.*, vol. 217, pp. 7–29, 2022, doi: 10.1007/978-3-030-91390-8_2.

[10] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 3, pp. 1681–1690, 2016.

[11] B. Belay, T. Habtegebrial, M. Meshesha, and M. Liwicki, “applied sciences Amharic OCR : An End-to-End Learning,” pp. 1–13, 2020, doi: 10.3390/app10031117.

[12] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, “Natural language processing: An introduction,” *J. Am. Med. Informatics Assoc.*, vol. 18, no. 5, pp. 544–551, 2011, doi: 10.1136/amiajnl-2011-000464.

[13] B. H. Belay, T. Habtegebrial, M. Liwicki, and G. Belay, “A Blended Attention-CTC Network Architecture for Amharic Text-image Recognition,” no. Icpram, pp. 435–441, 2021, doi: 10.5220/0010284204350441.

[14] B. Belay, T. Habtegebrial, G. Belay, and D. Stricker, “Using automatic features for text-image classification in Amharic documents,” *ICPRAM 2020 - Proc. 9th Int. Conf. Pattern Recognit. Appl. Methods*, no. Icpram 2020, pp. 440–445, 2020, doi: 10.5220/0008940704400445.

[15] B. H. Belay, “Deep Learning for Amharic Text-Image Recognition : Algorithm , Dataset and Application,” 2021.

[16] B. Belay, T. Habtegebrial, G. Belay, and M. Meshesha, “Learning by Injection : Attention Embedded Recurrent Neural Network for Amharic Text-image Recognition,” no. October, 2020.

[17] J. Zakraoui, M. Saleh, S. Al-Maadeed, and J. M. Jaam, “Improving text-to-image generation with object layout guidance,” *Multimed. Tools Appl.*, vol. 80, no. 18, pp. 27423–27443, 2021, doi: 10.1007/s11042-021-11038-0.

[18] M. R. Morris, J. Johnson, C. L. Bennett, and E. Cutrell, “Rich representations of visual content for Screen reader users,” *Conf. Hum. Factors Comput. Syst. - Proc.*, vol. 2018-April, 2018, doi: 10.1145/3173574.3173633.

[19] A. Singh and S. Anekar, “Text to Image using Deep Learning; Text to Image using Deep Learning,” *Artic. Int. J. Eng. Tech. Res.*, vol. 10, no. March, 2022, [Online]. Available: www.ijert.org

[20] S. Sharma, D. Suhubdy, V. Michalski, S. E. Kahou, and Y. Bengio, “ChatPainter: Improving text to image generation using dialogue,” *6th Int. Conf. Learn. Represent. ICLR 2018 - Work. Track Proc.*, 2018.

[21] S. Ramzan and M. M. Iqbal, “Text-to-Image Generation Using Deep Learning †,” pp. 1–6,

2022.

[22] S. Nam, Y. Kim, and S. J. Kim, “Text-adaptive generative adversarial networks: Manipulating images with natural language,” *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. NeurIPS, pp. 42–51, 2018.

[23] T. Liu *et al.*, “ARTIST: A Transformer-based Chinese Text-to-Image Synthesizer Digesting Linguistic and World Knowledge,” *Find. Assoc. Comput. Linguist. EMNLP 2022*, pp. 881–888, 2022.

[24] A. Dandekar, R. Malladi, P. Gore, and D. V. Dalal, “Text to Image Synthesis using Generative Adversarial Networks,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 11, no. 4, pp. 2723–2730, 2023, doi: 10.22214/ijraset.2023.50584.

[25] H. Zhang *et al.*, “StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 8, pp. 1947–1962, 2019, doi: 10.1109/TPAMI.2018.2856256.

[26] H. Wang, R. Czerminski, and A. C. Jamieson, “Neural Networks and Deep Learning,” *Mach. Age Cust. Insight*, pp. 91–101, 2021, doi: 10.1108/978-1-83909-694-520211010.

[27] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *1st Int. Conf. Learn. Represent. ICLR 2013 - Work. Track Proc.*, pp. 1–12, 2013.

[28] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information,” *Trans. Assoc. Comput. Linguist.*, vol. 5, pp. 135–146, 2017, doi: 10.1162/tacl_a_00051.

[29] H. Liu, Y. Wu, S. Zhai, B. Yuan, and N. Zhang, “RIATIG: Reliable and Imperceptible Adversarial Text-to-Image Generation with Natural Prompts,” pp. 20585–20594, 2023, doi: 10.1109/cvpr52729.2023.01972.

[30] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training GANs,” *Adv. Neural Inf. Process. Syst.*, pp. 2234–2242, 2016.

[31] J. Moolayil, *Learn Keras for Deep Neural Networks Learn Keras for Deep*.

[32] I. G. and Y. B. and A. Courville, “Deep learning 简介 一 、 什么是 Deep Learning ？,” *Nature*, vol. 29, no. 7553, pp. 1–73, 2016, [Online]. Available: http://deeplearning.net/

[33] S. J. Mielke *et al.*, “Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP,” 2021, [Online]. Available: http://arxiv.org/abs/2112.10508

[34] P. Cameron, “This is a reproduction of a library book that was digitized by Google as part of an ongoing effort to preserve the information in books and make it universally accessible .,” *Biol. Cent.*, vol. 2, pp. v–413, 2005.

[35] C. Darwin, “This is a reproduction of a library book that was digitized by Google as part of an ongoing effort to preserve the information in books and make it universally accessible. https://books.google.com,” *Oxford Univ.*, vol. XXX, p. 60, 1895.

[36] B. Li, X. Qi, T. Lukasiewicz, and P. H. S. Torr, “Controllable Text-to-Image Generation,” no. NeurIPS, 2019.

[37] Y. X. Tan, C. P. Lee, M. Neo, K. M. Lim, and J. Y. Lim, “Enhanced Text-to-Image Synthesis Conditional Generative Adversarial Networks,” *IAENG Int. J. Comput. Sci.*, vol. 49, no. 1, pp. 1–7, 2022.

[38] T. Xu *et al.*, “AttnGAN : Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks,” pp. 1316–1324.

[39] M. Ding *et al.*, “CogView: Mastering Text-to-Image Generation via Transformers,” *Adv. Neural Inf. Process. Syst.*, vol. 24, no. NeurIPS, pp. 19822–19835, 2021.

[40] T. Qiao, J. Zhang, D. Xu, and D. Tao, “Mirrorgan: Learning text-to-image generation by redescription,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 1505–1514, 2019, doi: 10.1109/CVPR.2019.00160.

[41] G. Yin, B. Liu, L. Sheng, N. Yu, X. Wang, and J. Shao, “Semantics disentangling for text-to-image generation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 2322–2331, 2019, doi: 10.1109/CVPR.2019.00243.

[42] S. Loewen and M. Sato, “The Routledge Handbook of Instructed Second Language Acquisition,” *Routledge Handb. Instr. Second Lang. Acquis.*, pp. 1–601, 2017, doi: 10.4324/9781315676968.

[43] M. Gheorghe, F.-C. Mihai, and M. Dârdală, “Modern techniques of web scraping for data scientists,” *Rev. Rom. Interactiune Om-Calculator*, vol. 11, no. 1, pp. 63–75, 2018.

[44] A. K. Morya *et al.*, “Evaluating the viability of a smartphone-based annotation tool for faster and accurate image labelling for artificial intelligence in diabetic retinopathy,” *Clin. Ophthalmol.*, vol. 15, pp. 1023–1039, 2021, doi: 10.2147/OPTH.S289425.

[45] K. G. Kim, “Deep learning book review,” *Nature*, vol. 29, no. 7553, pp. 1–73, 2019.

[46] P. Nema, S. Shetty, P. Jain, A. Laha, K. Sankaranarayanan, and M. M. Khapra, “Generating descriptions from structured data using a bifocal attention mechanism and gated orthogonalization,” *NAACL HLT 2018 - 2018 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, pp. 1539–1550, 2018, doi: 10.18653/v1/n18-1139.

[47] T. Demeester, T. Rocktäschel, and S. Riedel, “Lifted rule injection for relation embeddings,” *EMNLP 2016 - Conf. Empir. Methods Nat. Lang. Process. Proc.*, pp. 1389–1399, 2016, doi: 10.18653/v1/d16-1146.

[48] R. Szeliski, “Computer vision: algorithms and applications,” *Choice Rev. Online*, vol. 48, no. 09, pp. 48-5140-48–5140, 2011, doi: 10.5860/choice.48-5140.

[49] P. Doll, R. Girshick, and F. Ai, “Mask R-CNN ar”.

[50] P. Shamsolmoali *et al.*, “Image synthesis with adversarial networks: A comprehensive survey and case studies,” *Inf. Fusion*, vol. 72, pp. 126–146, 2021, doi: 10.1016/j.inffus.2021.02.014.

[51] X. Ding, Y. Wang, Z. Xu, W. J. Welch, and Z. J. Wang, “Image Generation Using Continuous Conditional Generative Adversarial Networks,” *Intell. Syst. Ref. Libr.*, vol. 217, pp. 87–113, 2022, doi: 10.1007/978-3-030-91390-8_5.

[52] S. Mahdizadehaghdam, A. Panahi, and H. Krim, “Sparse generative adversarial network,”

Proc. - 2019 Int. Conf. Comput. Vis. Work. ICCVW 2019, pp. 3063–3071, 2019, doi: 10.1109/ICCVW.2019.00369.

[53] M. Cha, Y. Gwon, and H. T. Kung, "Adversarial nets with perceptual losses for text-to-image synthesis," *IEEE Int. Work. Mach. Learn. Signal Process. MLSP*, vol. 2017-Septe, pp. 1–6, 2017, doi: 10.1109/MLSP.2017.8168140.

[54] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 6627–6638, 2017, doi: 10.18034/ajase.v8i1.9.

[55] L. Theis, A. Van Den Oord, and M. Bethge, "A note on the evaluation of generative models," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–10, 2016.

Appendix

Snipped python code to define, build the Generator and the Discriminator networks

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Reshape, Concatenate, Conv2D, Flatten, BatchNormalization, UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
# Hyperparameters
noise_dim =100
text_embedding_dim = 100
image_shape = (128, 128, 3)
# Generator architecture
def build_generator(noise_dim, text_embedding_dim, image_shape):
    # Input layers
    noise_input = Input(shape=(noise_dim,))
    text_input = Input(shape=(text_embedding_dim,))
    # Concatenate noise and text embedding
    combined_input = Concatenate()([noise_input, text_input])
    # Dense layer to project to initial shape
    x = Dense(256 * 16 * 16, activation='relu')(combined_input)
    x = BatchNormalization()(x)
    x = Reshape((16, 16, 256))(x)
    # Upsampling blocks
    x = UpSampling2D()(x)
    x = Conv2D(128, (3, 3), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    x = UpSampling2D()(x)
    x = Conv2D(64, (3, 3), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    x = UpSampling2D()(x)
    x = Conv2D(32, (3, 3), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
```

Sample of real images

```
[ ]     # Output image
        generated_image = Conv2D(image_shape[2], (3, 3), padding='same', activation='tanh')(x)
        # Rescale generated image to [0, 1]
        generated_image = (generated_image + 1) / 2
        # Define the generator model
        generator = Model(inputs=[noise_input, text_input], outputs=generated_image)
        return generator
    # Discriminator architecture
    def build_discriminator(image_shape, text_embedding_dim):
        # Input layers
        image_input = Input(shape=image_shape)
        text_input = Input(shape=(text_embedding_dim,))
        # Downsample the image
        x = Conv2D(64, (3, 3), strides=(2, 2), padding='same', activation='relu')(image_input)
        x = Conv2D(128, (3, 3), strides=(2, 2), padding='same', activation='relu')(x)
        x = Conv2D(256, (3, 3), strides=(2, 2), padding='same', activation='relu')(x)
        # Flatten the features
        x = Flatten()(x)
        # Concatenate with text embedding
        x = Concatenate()([x, text_input])
        # Dense layers for classification
        x = Dense(512, activation='relu')(x)
        validity = Dense(1, activation='sigmoid')(x)
        # Define the discriminator model
        discriminator = Model(inputs=[image_input, text_input], outputs=validity)
        return discriminator
    # Example usage
    generator = build_generator(noise_dim, text_embedding_dim, image_shape)
    discriminator = build_discriminator(image_shape, text_embedding_dim)
    # Print model summaries for inspection
    print('Generator summary')
    generator.summary()
    print('discriminator summary')
    discriminator.summary()
```

Snipped python code to compile, train the Generator and the Discriminator networks

```
# Compiling the discriminator
discriminator.compile(
    optimizer=Adam(learning_rate=0.0005, beta_1=0.5),
    loss=BinaryCrossentropy(),
    metrics=['accuracy'])
# Compiling the generator
discriminator.trainable = False  # Freeze the discriminator during generator training
noise_input = Input(shape=(noise_dim,))
text_input = Input(shape=(text_embedding_dim,))
generated_image = generator([noise_input, text_input])
validity = discriminator([generated_image, text_input])
combined_model = Model(inputs=[noise_input, text_input], outputs=validity)
# Compile the combined model
combined_model.compile(
    optimizer=Adam(learning_rate=0.0005, beta_1=0.5),
    loss=BinaryCrossentropy(),
    metrics=['accuracy'])
# Training loop
epochs =100
batch_size =32
# X_train, Y_train are placeholders for real images and corresponding text embeddings
# Make sure to preprocess your data accordingly
for epoch in range(epochs):
    # Train the discriminator
    idx = np.random.randint(0, train_images.shape[0], batch_size)
    real_images = train_images[idx]
    real_texts = train_text[idx]
    noise = np.random.normal(0, 1, (batch_size, noise_dim))
    generated_images = generator.predict([noise, real_texts])
    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))
    d_loss_real = discriminator.train_on_batch([real_images, real_texts], valid)
    d_loss_fake = discriminator.train_on_batch([generated_images, real_texts], fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
# Train the generator
noise = np.random.normal(0, 1, (batch_size, noise_dim))
valid = np.ones((batch_size, 1))
g_loss = combined_model.train_on_batch([noise, real_texts], valid)
# Print progress
print(f"Epoch {epoch+1}/{epochs}, [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}] [G loss: {g_loss[0]} | G accuracy: {100 * g_loss[1]}]")
```

The following snipped is the output of the Generator loss, accuracy and the Discriminator loss, accuracy for 1000 epochs.

```
1/1 [==============================] - 0s 58ms/step
Epoch 1/100, [D loss: 35.15514604561031 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 2/100, [D loss: 47.72799403918907 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 24ms/step
Epoch 3/100, [D loss: 42.76079559353745 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 4/100, [D loss: 45.23009583561134 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 22ms/step
Epoch 5/100, [D loss: 55.50896156113595 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 21ms/step
Epoch 6/100, [D loss: 34.781797178748434 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 21ms/step
Epoch 7/100, [D loss: 61.780056001205736 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 21ms/step
Epoch 8/100, [D loss: 41.36792766631649 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 9/100, [D loss: 52.14534057426573 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 22ms/step
Epoch 10/100, [D loss: 34.88745880126953 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 11/100, [D loss: 55.027870178222656 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 12/100, [D loss: 40.36959088717046 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 22ms/step
Epoch 13/100, [D loss: 46.45863881777041 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 23ms/step
Epoch 14/100, [D loss: 54.67641403246671 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 15/100, [D loss: 75.63344588326287 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 21ms/step
Epoch 16/100, [D loss: 56.95899200439453 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 21ms/step
Epoch 17/100, [D loss: 52.36958383023739 | D accuracy: 46.875] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 18/100, [D loss: 67.7844711849466 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
```

```
1/1 [==============================] - 0s 18ms/step
Epoch 19/100, [D loss: 52.52555086145374 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 20/100, [D loss: 51.024017333984375 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 21/100, [D loss: 74.75876617431823 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 22/100, [D loss: 64.63667297839201 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 17ms/step
Epoch 23/100, [D loss: 85.51023864746283 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 24/100, [D loss: 76.53816223144531 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 25/100, [D loss: 76.05651303380728 | D accuracy: 46.875] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 26/100, [D loss: 68.59939939744072 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 22ms/step
Epoch 27/100, [D loss: 74.98644285005693 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 28/100, [D loss: 70.23561096191406 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 29/100, [D loss: 68.48184204588877 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 30/100, [D loss: 59.37149493023753 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 31/100, [D loss: 81.27520751953125 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 32/100, [D loss: 57.64580535888672 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 22ms/step
Epoch 33/100, [D loss: 65.24729919438958 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 22ms/step
Epoch 34/100, [D loss: 93.69262129068375 | D accuracy: 46.875] [G loss: 0.0 | G accuracy: 100.0]
```

```
1/1 [==============================] - 0s 18ms/step
Epoch 35/100, [D loss: 86.66142272959927 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 36/100, [D loss: 80.41880811396622 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 37/100, [D loss: 88.55228096107021 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 38/100, [D loss: 83.57435607910156 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 39/100, [D loss: 96.45969391075674 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 24ms/step
Epoch 40/100, [D loss: 96.61318969726919 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 22ms/step
Epoch 41/100, [D loss: 83.3092346382443 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 21ms/step
Epoch 42/100, [D loss: 80.70259094238281 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 20ms/step
Epoch 43/100, [D loss: 96.63031005859375 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 44/100, [D loss: 91.25276947021993 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 19ms/step
Epoch 45/100, [D loss: 87.86197627396177 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 18ms/step
Epoch 46/100, [D loss: 74.20851135253906 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 28ms/step
Epoch 47/100, [D loss: 104.18009521426575 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 21ms/step
Epoch 48/100, [D loss: 96.35236358642578 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 23ms/step
Epoch 49/100, [D loss: 82.86049024760723 | D accuracy: 46.875] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 25ms/step
Epoch 50/100, [D loss: 94.87630695477128 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
```

```
1/1 [==============================] - 0s 20ms/step
Epoch 51/100, [D loss: 97.21112060549822 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 26ms/step
Epoch 52/100, [D loss: 106.83892392495181 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 31ms/step
Epoch 53/100, [D loss: 111.21977233886719 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 28ms/step
Epoch 54/100, [D loss: 95.98162841798023 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 28ms/step
Epoch 55/100, [D loss: 105.52896118164114 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 28ms/step
Epoch 56/100, [D loss: 109.75060272224138 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 32ms/step
Epoch 57/100, [D loss: 114.94434692784853 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 30ms/step
Epoch 58/100, [D loss: 125.66526794433642 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 34ms/step
Epoch 59/100, [D loss: 120.47484056465328 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 29ms/step
Epoch 60/100, [D loss: 97.89286041261767 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 33ms/step
Epoch 61/100, [D loss: 139.91751126172335 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 31ms/step
Epoch 62/100, [D loss: 102.02210729965009 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 28ms/step
Epoch 63/100, [D loss: 128.0010377226824 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 28ms/step
Epoch 64/100, [D loss: 125.00720977787505 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 27ms/step
Epoch 65/100, [D loss: 102.73989868164307 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 29ms/step
Epoch 66/100, [D loss: 117.08275879919529 | D accuracy: 46.875] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 29ms/step
Epoch 67/100, [D loss: 121.16274099628208 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
1/1 [==============================] - 0s 27ms/step
Epoch 68/100, [D loss: 127.63000488282185 | D accuracy: 50.0] [G loss: 0.0 | G accuracy: 100.0]
```

Snipped code to calculate FID score

```
from scipy.linalg import sqrtm
from skimage.transform import resize
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
def calculate_fid(test_images, generated_images):
    # Ensure the number of samples matches
    num_samples = min(test_images.shape[0], generated_images.shape[0])
    test_images = test_images[np.random.choice(test_images.shape[0], num_samples, replace=False)]
    # Load InceptionV3 model pre-trained on ImageNet
    model = InceptionV3(include_top=False, pooling='avg', input_shape=(299, 299, 3))
    # Resize images to (299, 299) as required by InceptionV3
    real_resized = np.array([resize(image, (299, 299)) for image in test_images])
    gen_resized = np.array([resize(image, (299, 299)) for image in generated_images])
    # Preprocess images for InceptionV3
    real_preprocessed = preprocess_input(real_resized)
    gen_preprocessed = preprocess_input(gen_resized)
    # Get intermediate representations
    real_features = model.predict(real_preprocessed)
    gen_features = model.predict(gen_preprocessed)
    # Calculate mean and covariance statistics
    mu1, sigma1 = real_features.mean(axis=0), np.cov(real_features, rowvar=False)
    mu2, sigma2 = gen_features.mean(axis=0), np.cov(gen_features, rowvar=False)
    # Calculate FID score
    ssdiff = np.sum((mu1 - mu2) ** 2.0)
    covmean = sqrtm(sigma1.dot(sigma2))
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    fid = ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid
# Example usage:
# Assuming test_images and generated_images are numpy arrays of shape (num_images, height, width, channels)
fid_score = calculate_fid(test_images, generated_images)
print(f"FID Score: {fid_score}")
```

Snipped code to calculate the IS score

Evaluateing the model with IS (Inception Score)

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from scipy.stats import entropy
import numpy as np
from skimage.transform import resize
# Load pre-trained InceptionV3 model
pre_trained_inceptionv3_model = InceptionV3(include_top=False, pooling='avg', input_shape=(299, 299, 3))
def calculate_is(generated_images, model):
    # Ensure the number of samples matches
    num_samples = min(generated_images.shape[0], 10)  # Choose the desired number of samples, e.g., 10
    generated_images = generated_images[np.random.choice(generated_images.shape[0], num_samples, replace=False)]
    # Preprocess images for InceptionV3
    gen_resized = np.array([resize(image, (299, 299)) for image in generated_images])
    gen_preprocessed = preprocess_input(gen_resized)
    # Get predicted class probabilities
    preds = model.predict(gen_preprocessed)
    # Calculate Inception Score
    p_yx = preds / np.sum(preds, axis=1, keepdims=True)
    kl_divs = entropy(p_yx.T)
    is_score = np.exp(np.mean(kl_divs))
    return is_score
is_score = calculate_is(generated_images, pre_trained_inceptionv3_model)
print(f"IS score: {is_score}")
```