



## **Project Proposal DS 3002 Data Mining**

### **Text-to-Image Generation for E-Commerce Products (Apparel and Footwear Products)**

---

#### **Group Members (BDS-6A)**

Amna Habib 22L-7515

Habiba Tariq 22L-7501

Asma Fatima 22L-7540

Alina Baber 22L-7492

## I. Introduction

E-commerce platforms rely heavily on high-quality product images to attract customers and boost sales. However, manually capturing and uploading thousands of product images is a very time-consuming and costly process that requires significant human input and Labor. To resolve and overcome these challenges, our project is on text-to image generation for e-commerce fashion products where we will leverage advanced deep learning models, to automatically generate realistic product images from textual descriptions, which will reduce the dependency on traditional photography and enhance the efficiency of online retail platforms.

### A. Motivation

Since e-commerce platforms require vast amounts of high-quality product images to enhance user experience and drive sales, manually generating and managing images is costly and time intensive as stated above. To create product images directly from textual descriptions, this automation will significantly enhance product catalogues, reduce dependency on photography, and improve accessibility for retailers looking to scale and expand operations to an even bigger and better level.

### B. Dataset Description

Link of dataset: Kaggle, "Fashion Images Dataset."

Available: <https://www.kaggle.com/datasets/vikashrajluhaniwal/fashion-images>

The dataset utilized in this project comprises fashion product images along with their corresponding textual descriptions. It consists of **2,906** entries and **11** attributes, providing essential details to enhance the accuracy of generating product images from text descriptions. It includes the Product Title, which provides a textual description of each fashion item, and the Category and Subcategory, which classifies products into different types such as footwear, top wear, and bottom wear. Additionally, the dataset specifies the Colour of each item, ensuring that the generated images align with the described attributes. It also includes Image URLs, which link to real product images used for training and evaluation as seen in Fig1.

	ProductId	Gender	Category	SubCategory	ProductType	Colour	Usage	ProductTitle	Image	ImageURL	ImagePath
0	42419	Girls	Apparel	Topwear	Tops	White	Casual	Gini and Jony Girls Knit White Top	42419.jpg	http://assets.myntassets.com/v1/images/style/p...	fashion_images/0.jpg
1	34009	Girls	Apparel	Topwear	Tops	Black	Casual	Gini and Jony Girls Black Top	34009.jpg	http://assets.myntassets.com/v1/images/style/p...	fashion_images/1.jpg
2	40143	Girls	Apparel	Topwear	Tops	Blue	Casual	Gini and Jony Girls Pretty Blossom Blue Top	40143.jpg	http://assets.myntassets.com/v1/images/style/p...	fashion_images/2.jpg
3	23623	Girls	Apparel	Topwear	Tops	Pink	Casual	Doodle Kids Girls Pink I love Shopping Top	23623.jpg	http://assets.myntassets.com/v1/images/style/p...	fashion_images/3.jpg
4	47154	Girls	Apparel	Bottomwear	Capris	Black	Casual	Gini and Jony Girls Black Capris	47154.jpg	http://assets.myntassets.com/v1/images/style/p...	fashion_images/4.jpg

Fig. 1. First 5 rows of our dataset

## C. Text to Image Generation

The aim of this project to build **Text-to-Image Generation Model** for eCommerce fashion products. It involves data preprocessing, handling class imbalance, and implementing a deep learning model that maps text descriptions to realistic fashion images. It aims to generate high quality, visually accurate fashion images from textual inputs, enhancing eCommerce product visualization.

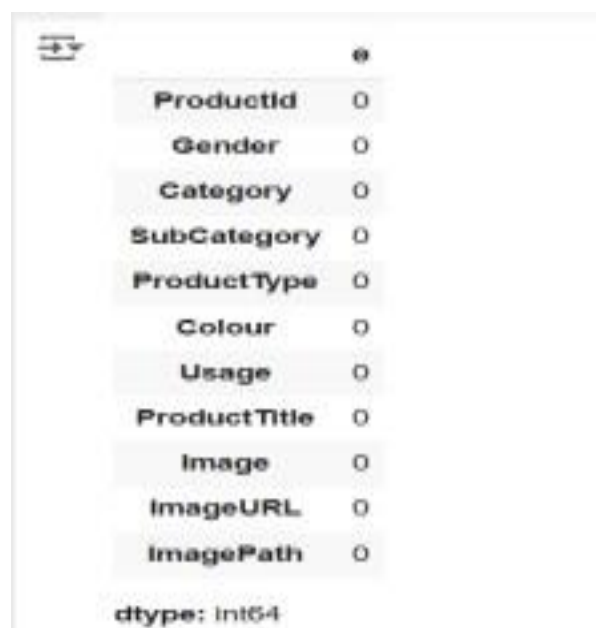
## II. Text to Image Generation

### A. Initialization And Preprocessing

The initialization phase of the project begins by importing essential libraries, including TensorFlow for deep learning, Pandas and NumPy for data manipulation, and OpenCV for image processing. Additionally, it includes libraries for text preprocessing, model building, and visualization, such as Matplotlib and Seaborn.

Next, the dataset is loaded from a CSV file (**fashion.csv**), which contains information about fashion products, including image URLs and textual descriptions. To manage image data efficiently, a directory named "fashion images" is created to store the downloaded images.

Before training, to enhance consistency, improve data quality, and ensure optimal model performance, preprocessing was done. The first step involved **data cleaning**, where **missing entries** were removed to avoid inconsistencies. The dataset has no missing values in any columns as shown in below fig.



The screenshot shows a Jupyter Notebook interface with a Pandas Series of zeros for various product attributes. The attributes listed are ProductId, Gender, Category, SubCategory, ProductType, Colour, Usage, ProductTitle, Image, ImageURL, and ImagePath. All values are 0, indicating no missing data. The dtype is Int64.

	0
ProductId	0
Gender	0
Category	0
SubCategory	0
ProductType	0
Colour	0
Usage	0
ProductTitle	0
Image	0
ImageURL	0
ImagePath	0

dtype: Int64

Secondly, for **Text preprocessing**, we start by converting all text to lowercase to maintain consistency. Next, we remove URLs and special characters using regular expressions to clean the text. The **Combined Title** column is created by merging key product attributes like Usage, Gender, Colour, and Product Type, ensuring a meaningful representation of the product description. After concatenation, extra spaces are removed to keep the text well-formatted as shown in Fig 2. This cleaned and structured text is essential for training the model effectively.

```
import pandas as pd
import re

def strip_before_gender(name):
    if pd.isna(name):
        return name # skip NaNs
    match = re.search(r'\b(Girls|Boys|Men|Women)\b', name, flags=re.IGNORECASE)
    if match:
        return name[match.start():]
    return name

df['CleanedProductTitle'] = df['ProductTitle'].apply(strip_before_gender)
print(df[['ProductTitle', 'CleanedProductTitle']].head())
```

	ProductTitle	CleanedProductTitle
0	Gini and Jony Girls Knit White Top	Girls Knit White Top
1	Gini and Jony Girls Black Top	Girls Black Top
2	Gini and Jony Girls Pretty Blossom Blue Top	Girls Pretty Blossom Blue Top
3	Doodle Kids Girls Pink I love Shopping Top	Girls Pink I love Shopping Top
4	Gini and Jony Girls Black Capris	Girls Black Capris

Furthermore, Combined title is converted to **numerical sequences** for deep learning. A Tokenizer is initialized with a vocabulary limit of 5,000 words and an out-of-vocabulary (<OOV>) token. It then fits on the Combined Title column to create a word index. The text is converted into sequences and padded to ensure uniform input length. Finally, the vocabulary size and maximum sequence length are computed. This step prepares the text data for models like CNN-LSTM, ensuring compatibility with neural network architectures .

#### Padding and Tokenizing

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
tokenizer.fit_on_texts(df['CombinedTitle'])
sequences = tokenizer.texts_to_sequences(df['CombinedTitle'])

max_length = max(len(seq) for seq in sequences)

padded_captions = pad_sequences(sequences, maxlen=max_length, padding='post')

vocab_size = len(tokenizer.word_index) + 1

print(f"Vocabulary Size: {vocab_size}")
print(f"Max Sequence Length: {max_length}")
```

Vocabulary Size: 1155  
Max Sequence Length: 12

For **image preprocessing**, we have defined a fixed target size of (64X64) pixels to ensure uniformity across all input images. Each image is loaded from its file path using `load_img()`, resized accordingly, and then converted into a numerical array using `img_to_array()`. To enhance model performance, the pixel values are normalized between 0 and 1, which helps in faster and more stable training. After preprocessing, all valid images are stored in a NumPy array, making them ready for further processing in the deep learning model.



Fig. 3. Example of preprocessing applied to image

B. Dataset Summary



C. Visualizations

To better understand the dataset, various visualization techniques were applied. The distribution of different product categories, colour variations, and gender-based classifications were analysed. These insights help refine the model and ensure balanced representation across different fashion types.

• Most Common Fashion Items

**Figure 4** illustrates the distribution of fashion products across two main categories: Footwear and Apparel. The **bar chart** shows that both categories have a similar count, with Footwear slightly outnumbering Apparel. This distribution helps in understanding the dataset's balance and identifying potential class imbalances for model training.



Fig. 4. Distribution of Fashion Categories

**Fig.5.** shows a **bar plot** which displays the **top 10 most common subcategories**. A bar plot is used on to visualize the frequency of each category, helping to understand which types of fashion items are most prevalent. The main categories are sorted in descending order to show the most accessed ones to help us identify a sort of bias present in the dataset. And by further ranking the subcategories based, we can observe which fashion items dominate the dataset.

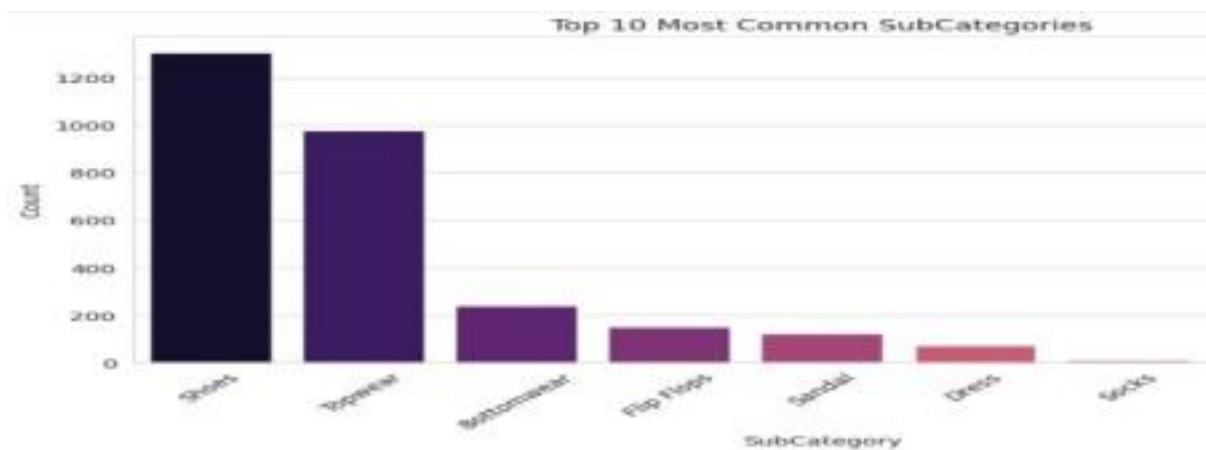
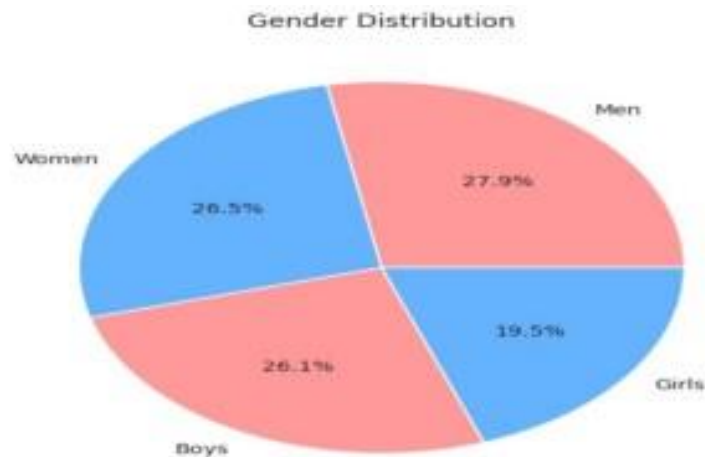


Fig. 5 Most Common Subcategories

- **Gender-Based Representation**

Pie charts are used to see proportions, and such is the case with Fig. 6. where the visualizations provide a breakdown of the dataset by gender (e.g., Men's, Women's, and Kids' fashion). If one category significantly outweighs the others, it means that there is an imbalance in the dataset. For example, if women's fashion dominates, it may lead to a bias in model training, affecting the ability to generate images for men's or unisex products accurately. If the distribution is relatively even, the dataset is well-balanced, ensuring fair representation across different gender categories which is the case for our dataset as seen in fig. 6.



- **Colour Trends in Fashion Products**

Similarly **Figure 7** represents the **top 10 most common colours** in the dataset, with Black and White being the most dominant. Blue, Brown, Red, and Pink also appear frequently, indicating a preference for neutral and classic shades in fashion products. The lower-ranked colours, such as Green, Grey, Yellow, and Navy Blue, suggest relatively lower occurrences in the dataset. This analysis helps in understanding **colour trends** and their impact on product categorization.

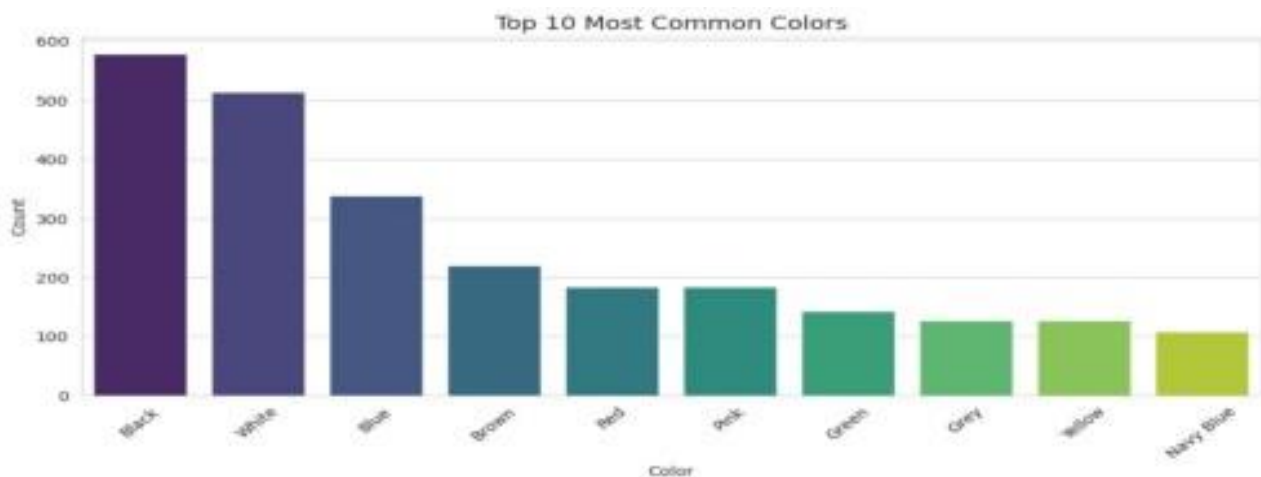


Fig.7.

- **Brand Representation**

This below visualization highlights the **top 10 most common fashion brands** in the dataset. Peter England dominates with the highest count, followed by U.S. Polo, Catwalk, Adidas, and Nike. Other brands like Puma, Red Tape, Fila, Reebok, and Lee also have a notable presence.

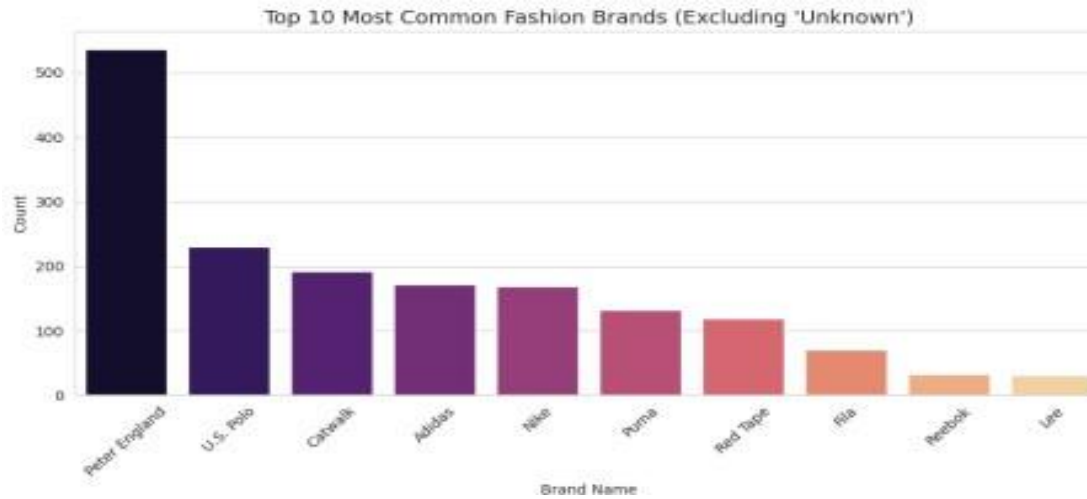


Fig. 8.

#### D. Class Imbalance

Class imbalance is a significant challenge in training text-to-image models. The class imbalance analysis checks whether one category significantly outnumbers another in the dataset. Here, the Footwear category has 1,580 samples, while Apparel has 1,326 samples. Since the smallest class (1,326) is more than 50% of the largest class (1,580), the dataset is considered **balanced**, and SMOTE (Synthetic Minority Over-sampling Technique) is not applied. This ensures that the model is trained on a fair distribution of categories without introducing artificial samples.

```
Class Distribution Before Balancing:
Category
Footwear    1580
Apparel     1326
Name: count, dtype: int64

Dataset is already balanced. No SMOTE applied.
```

### III. Modelling Phase

During the modelling phase of our Image-to-Text Generator project, we explored and implemented five deep learning architectures: Simple GAN, Attention GAN, a custom version of Stable Diffusion, Stack GAN, and Style GAN. All models take structured product metadata (such as gender, color, product type, and usage) as input, which we merged into a combined text string (e.g., “men black running shoes”). This text input is tokenized and padded before being processed by a text encoder, typically an LSTM. A random noise vector of shape (100,) is also used to introduce variability in the generated outputs.

The **Simple GAN** model uses a basic LSTM to encode text and combines it with the noise vector to generate images via transposed convolution layers. The **Attention GAN** enhances this setup by applying an attention mechanism over the LSTM outputs, allowing the model to focus on the most relevant words



in the input for better text–image alignment. The **Stack GAN** generates low-resolution images in Stage-I and progressively refines them in Stage-II, conditioned on text features, improving sharpness and detail. The **Style GAN** leverages style-based modulation and adaptive instance normalization to inject text conditioning at various layers of the generator, enabling better control over generated image attributes. Our custom implementation of **Stable Diffusion** follows a denoising-based approach, gradually refining noise into coherent images based on encoded text features, providing higher realism and fidelity.

## A. Image Generation using Simple GAN Model

### Model Overview

This project implements a basic text-to-image generation model using a Generative Adversarial Network (GAN) architecture. The system integrates a text encoder and an image generator to synthesize images that match structured product descriptions (e.g., "girls red frock"). The architecture consists of three primary components: a **Text Encoder**, a **Generator**, and a **Discriminator**.

- **Text Encoder:** Uses an Embedding layer followed by an LSTM to convert a tokenized and padded text sequence into a fixed-length vector representation.
- **Generator:** Combines this encoded text vector with a random noise vector of shape (100,) using a concatenation layer. It then passes through a series of Dense, Batch Normalization, LeakyReLU, and Conv2DTranspose layers to generate an image output of size **64x64x3** (RGB).
- **Discriminator:** Accepts both the generated (or real) image and the corresponding text input. It processes the image and reshaped text features through Conv2D and Dense layers to predict whether the input image-text pair is real or fake.

The model takes:

- **Text input** (tokenized and padded to length 12)
- **Random noise vector** (size: 100)
- **Generated image** of size **64x64x3**

### Limitations of the Model

1. **Cannot Handle Unprocessed Images**  
The model expects input images (used for training the discriminator) to be resized to **64x64** and normalized. If images are raw, inconsistent, or of different resolutions, the training pipeline fails due to tensor shape mismatches.
2. **Low-Resolution Output**  
The current generator outputs low-resolution images (64x64), which often lack fine-grained details or texture consistency. For real-world usage, a higher-resolution generator is required.
3. **Quality Limitations with Few Epochs**  
The generator requires extensive training to produce coherent results. In early epochs, the images are noisy and lack visual structure.
4. **Text-Image Misalignment**

With no attention mechanism, the generator may ignore some important descriptive words (e.g., color or gender), leading to mismatches between the input description and the output image.

### Epoch wise Output Comparison

**INPUT 1:** *ethnic Boys Printed Maroon Kurta Sports Shoes*

**INPUT 2:** *sports Men Gel Running White*

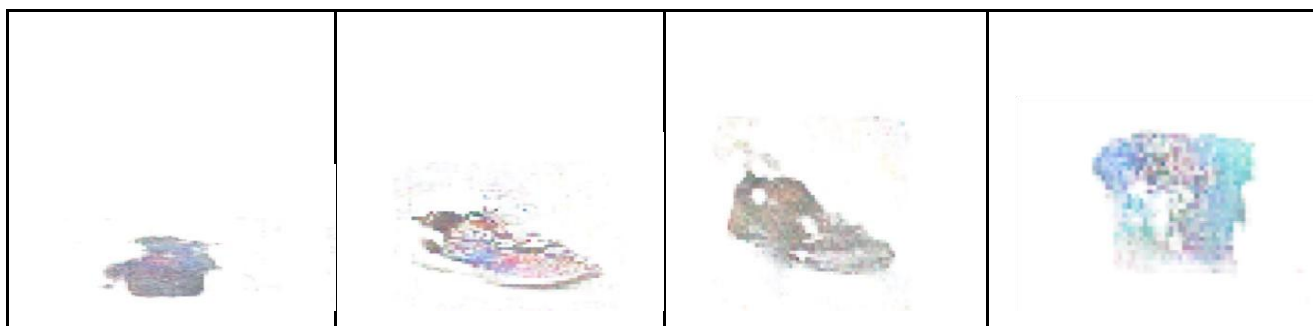
**INPUT 3:** *casual Men Aventura Sandal*

**INPUT 4 :** *casual Boys Expressions Yellow T-shirt*

#### 1. 50 EPOCHS



#### 2. 200 EPOCHS:



#### 3. 600 EPOCHS:



## Evaluation

The **Simple GAN** model achieved an **FID score of 12.01**, indicating **moderate realism** and **reasonable alignment** with real images. While not on par with high-resolution models, the score is acceptable for a basic 64×64 GAN with an LSTM text encoder.

**As we can see, the basic structure of the generated image is present, but it lacks fine details and high-resolution features.**

## B. Image Generation using Style GAN Model

### Model Overview

This model implements a StyleGAN-inspired text-to-image generator that integrates a **mapping network** and **style modulation mechanism** for improved control and diversity in image synthesis. It combines encoded textual descriptions with a style vector derived from a random noise input, allowing the generator to learn more expressive representations. The text is first tokenized, padded, and encoded using an LSTM. The noise vector (size 100) is passed through a multi-layer mapping network to produce a **style vector**, which is then fused with the text features.

The model takes:

- Text input (converted into padded sequences via tokenizer)
- Random noise vector (size: 100) and outputs:
- Generated images of size 64×64×3

### Key Characteristics

- Progressive growing of image features using **Conv2DTranspose** layers
- Style injection via a dense **mapping network**
- Joint conditioning on textual and latent features

### Limitations of the Model

#### 1. Cannot Handle Unprocessed Images

The model expects all input images (used during discriminator training) to be resized to **64×64** and normalized. If the images are raw, inconsistently sized, or unprocessed, the training pipeline fails due to **tensor shape mismatches**.

#### 2. Lower Resolution Output

Despite using a style-aware generation process, the output resolution is limited to **64×64 pixels**, which restricts the generation of **fine-grained details** and high visual fidelity.

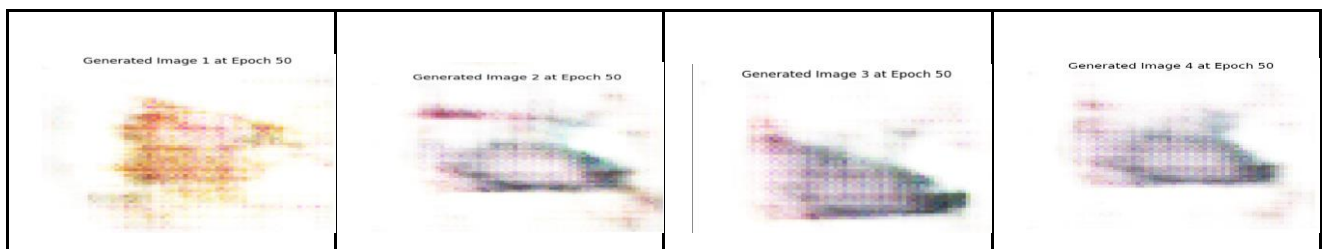
## Epoch-wise Output Comparison

**INPUT 1:** *ethnic Boys Printed Maroon Kurta*     **INPUT 2:** *sports Men Gel Running White Sports Shoes*

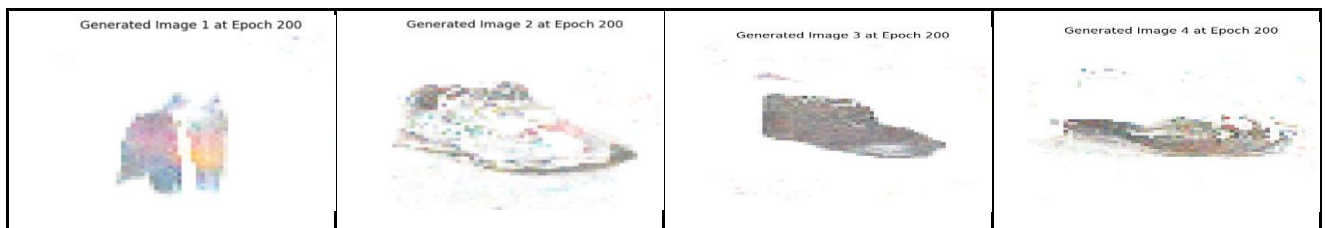
**INPUT 3:** *casual Men Aventura Sandal*

**INPUT 4 :** *Ipanema Women Black Flip Flop*

### 1. 50 EPOCHS



### 200 EPOCHS:



### 600 EPOCHS:



## Evaluation

To assess the performance of the StyleGAN-based model, we used the **Fréchet Inception Distance (FID)**, which measures the similarity between the feature distributions of real and generated images.

A lower FID score indicates higher visual fidelity and better alignment with real image statistics.

For the current model, the FID score on the validation set was:

## FID Score: 8.76

This score reflects a **noticeable improvement** over the basic GAN model (FID 12.01), suggesting that the addition of a style mapping network and a more structured fusion of noise and text features leads to **more realistic and coherent image generation**.

As we can see, the basic structure of the image is better preserved, and the visual artifacts are reduced compared to earlier results.

## C. Image Generation using Attention GAN Model

### Model Overview

The Attention GAN model leverages a powerful **attention mechanism** integrated into the text encoder to enhance the relevance and accuracy of generated images from textual descriptions. The model uses a **LSTM-based text encoder** to process the text input, followed by an **attention layer** that helps the model focus on relevant words and their relationships, creating a more meaningful connection between the input text and generated image. A random noise vector, passed through a mapping network, is combined with the encoded text features to generate the image.

The model takes

- Text input (encoded via LSTM with attention mechanism)
- Random noise vector (size: 100) and outputs:
- Generated images of size 64×64×3

### Key Characteristics

- Attention mechanism in the text encoder for improved semantic understanding
- Joint conditioning on text and latent noise vectors
- Progressive refinement of image features using **Conv2DTranspose** layers

### Limitations of the Model

#### 1. Attention Mechanism Limitation

While the attention mechanism allows the model to focus on relevant text elements, it is still limited by the LSTM's ability to capture long-range dependencies effectively.

#### 2. Cannot Handle Unprocessed Images

The model expects all input images (used during discriminator training) to be resized to **64×64** and normalized. If the images are raw, inconsistently sized, or unprocessed, the training pipeline fails due to **tensor shape mismatches**.

## Epoch-wise Output Comparison:

**INPUT 1:** *ethnic Boys Printed Maroon Kurta*    **INPUT 2:** *sports Men Gel Running White Sports Shoes*

**INPUT 3:** *casual Men Aventura Sandal*

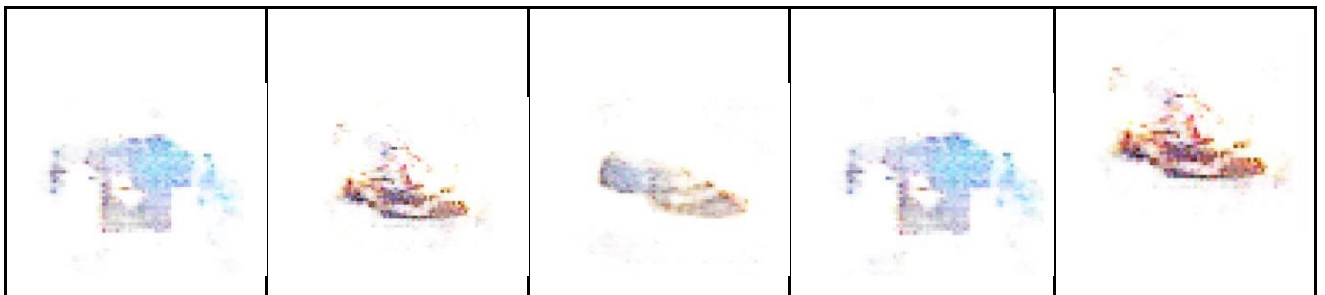
**INPUT 4:** *casual Boys Be Born Green Sweatshirt*

**INPUT 5:** *casual Women Beige Flats*

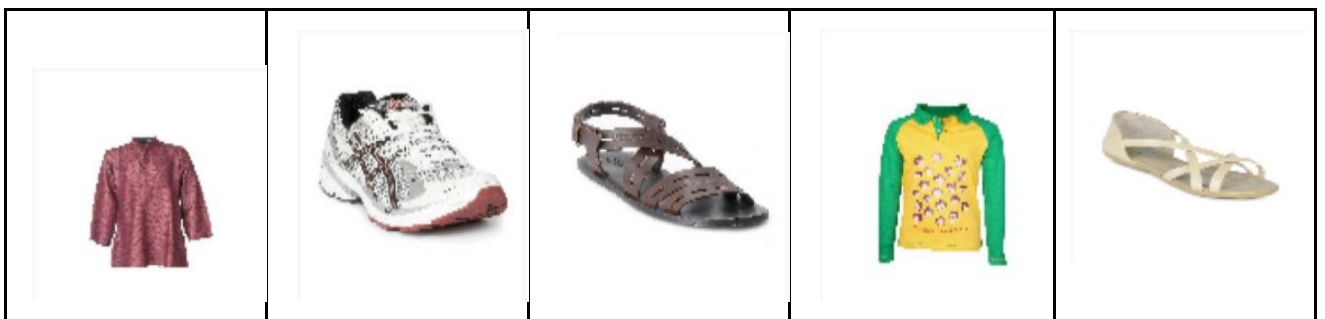
### 1.50 EPOCHS:



### 2. 200 EPOCHS:



### 3. 600 EPOCHS:



## Evaluation

To evaluate performance, we computed the **Frechet Inception Distance (FID)**, which compares the feature distributions of real and generated images. A lower FID score indicates that the generated images are perceptually closer to real images.

### FID Score: 5.986

This score represents a **significant improvement** over the Simple GAN model, indicating that the **attention mechanism** has led to better alignment between text and image features. The generated images show improved relevance to the text input, with finer details and better structure, although higher resolution would further enhance texture details.

As we can see, the inclusion of the attention layer results in more accurate feature alignment and semantic relevance in the generated images. However, some intricate visual details are still underdeveloped, suggesting that higher-resolution outputs would further improve the model.

## D. Image Generation using Stack GAN Model

### First Code Explanation

The first code provided is an implementation of a Stack GAN-like architecture, where the generator and discriminator are divided into two stages. Here's a breakdown of the components:

#### Text Encoder

- A simple **LSTM-based text encoder** is used, which takes the input caption and embeds it using an embedding layer. The sequence is then processed by an LSTM layer, followed by a **Dense layer**. This produces a fixed-size vector representing the textual description of the image.

#### Stage I Generator

- The **Stage 1 Generator** takes both a random noise vector and the encoded textual features. These are concatenated and passed through a **dense layer** that reshapes it into an initial 8x8x256 tensor.
- After reshaping, the network up samples this feature map through **Conv2DTranspose** layers, gradually increasing the spatial dimensions and applying activation functions such as **LeakyReLU**.
- The output is a 64x64x3 image, which is the initial generated image.

## Stage I Discriminator

- The **Stage 1 Discriminator** receives the generated image and the caption as input. The caption is passed through the same text encoder, which produces features that are then reshaped and concatenated with the image.
- The combined image-text features are passed through several **Conv2D** layers, and the result is flattened and passed through a **Dense layer** to output a prediction (real or fake).

## Stage II Generator

- The **Stage II Generator** takes the 64x64 image from Stage I and performs further up sampling to produce a larger image, specifically 128x128, through additional **Conv2DTranspose** layers.

## Stage II Discriminator

- Similar to the Stage I Discriminator, the **Stage II Discriminator** takes the generated 128x128 image along with the caption and processes them through several convolutional layers. The output is a binary prediction of whether the image is real or fake.

## FOR BOTH 10 AND 200 EPOCHS

Despite having a solid architectural foundation, the results from this StackGAN model were not satisfactory. The generated images did not show the desired quality, and the model failed to produce meaningful outputs

**INPUT 1:** *ethnic Boys Printed Maroon Kurta*

**INPUT 2:** *sports Men Gel Running White Sports Shoes*

**INPUT 3:** *casual Men Aventura Sandal*



## Modifications for Improvement

To improve the results, several parameters were tuned and adjustments were made in the second code. Below are the key changes:

### Changes in Text Encoder



- In the first model, the **LSTM layer** in the text encoder had 128 units. In the second code, the number of units was increased to **512**, and a **Dropout layer** with a rate of 0.5 was added to prevent overfitting and improve generalization.

### Generator Adjustments

- The **Stage I Generator** architecture remained the same in terms of the initial image generation process. However, the output image size of the **Stage II Generator** was increased from 128x128 to **256x256**, resulting in better image resolution. The up sampling layers were also adjusted for higher output resolution.

### Discriminator Adjustments

The **Stage I Discriminator** now has an additional convolutional layer, increasing the network's ability to differentiate between real and generated images. It now goes up to **256 filters**, enhancing its capacity to extract features from the image.

Similarly, the **Stage II Discriminator** was improved to handle the larger **256x256** image size.

The reshaped text features were adjusted to match the new image size.

### Learning Rate Schedule

A learning rate schedule was introduced in the second code. This employs **Exponential Decay**, starting with an initial learning rate of **2e-4** and gradually decreasing it. This helps stabilize training and prevents the learning rate from being too large, which might have caused instability in the first code.

### Optimizers

The **Adam optimizer** was used with the new learning rate schedule for both the generator and discriminator, ensuring a more controlled and adaptive learning process during training.

### Expected Impact of Changes

These modifications were expected to:

- **Improve text feature representation** by increasing the complexity of the text encoder, leading to better image-caption alignment.
- **Enhance image resolution** by increasing the output size of the second stage generator, resulting in more detailed and visually appealing images.
- **Refine discriminator's ability** to distinguish real and fake images by adding more convolutional layers and adjusting for the new image size.
- **Ensure stable training** with the introduction of a learning rate decay schedule and a larger dropout in the text encoder to prevent overfitting.

## Generated Image After Parameters Changes

*INPUT 1: ethnic Boys Printed Maroon Kurta*

*INPUT 2: sports Men Gel Running White Sports Shoes*

*INPUT 3: casual Men Aventura Sandal*



## Final Statement

Although we made several improvements to the architecture, parameters, and training procedure in the second code, these changes did not lead to the expected enhancement in image generation quality. This outcome could be due to a variety of factors, including model complexity, training instability, and the difficulty of mapping textual descriptions to high-quality images using a StackGAN framework.

## E. Image Generation using Stable Diffusion Fine-Tuned Model

### Model Overview

The Stable Diffusion Fine-Tuned Model is a state-of-the-art **text-to-image generation** framework that utilizes a **denoising diffusion process** to generate high-quality images from text descriptions. This implementation integrates a fine-tuned pipeline on fashion-related data, employing several critical components:

- **CLIP Text Encoder:** Converts the textual input into embeddings, aligning them with visual features.
- **Variational Autoencoder (VAE):** Encodes and decodes images into latent space, enhancing image clarity.
- **UNet2DConditionModel:** The core generator model that creates images from the conditioned text embeddings.
- **DDIM Scheduler:** Controls the number of inference steps and the guidance scale to refine the generated output.

The model architecture enables it to generate visually coherent images by conditioning both the **text** and the **latent noise vector**, guiding the diffusion process toward realistic outputs.

## Key Characteristics

- **Text-to-Image Conditioning:** The text is tokenized and encoded into embeddings, which serve as the condition for image generation.
- **Diffusion Process:** A denoising process iterates over the latent space to generate images progressively.
- **Guided Generation:** The model uses **guidance scales** to direct the generation process more closely aligned with the text prompt.
- **Fine-Tuned on Fashion Data:** The model is specifically fine-tuned for fashion-related products, allowing it to generate fashion-forward designs and styles.

## Model Comparison

### Prompts

1. casual men white T-shirt
2. casual men black running shoes

### Fined Tuned Results



## Pre-Trained Results

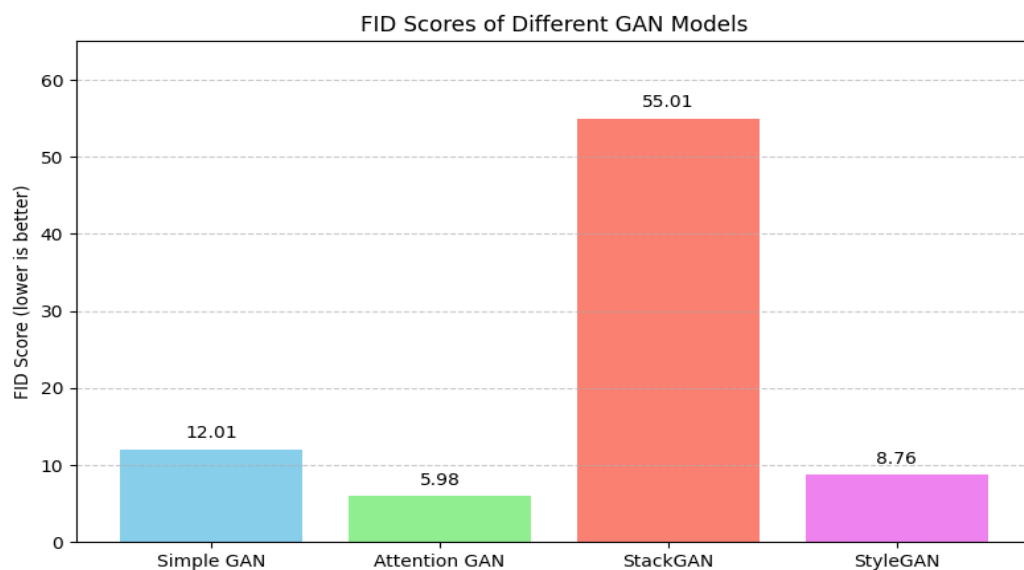


## IV. Best Performed Models

To evaluate the effectiveness of each text-to-image generation model, we compared their performance using the **Fréchet Inception Distance (FID)** score, a standard metric that assesses the realism and diversity of generated images by comparing them to real images in feature space. Lower FID scores indicate better performance.

As observed, the **Simple GAN** struggled the most, producing low-resolution images with weak correlation to the text prompts. The **StyleGAN-inspired model** further enhanced image quality and diversity by incorporating style modulation and a mapping network, showing a considerable drop in FID. The **Attention GAN** showed significant improvement by incorporating attention mechanisms, resulting in more focused image content.

The following bar graph summarizes the comparison:



## V. Best Performed Models

Attention GAN outperformed the Simple GAN and showed a notable reduction in FID score (from 12.01 to 5.98) due to its attention mechanism, which allows the generator to focus on the most relevant parts of the text description when synthesizing different regions of the image.

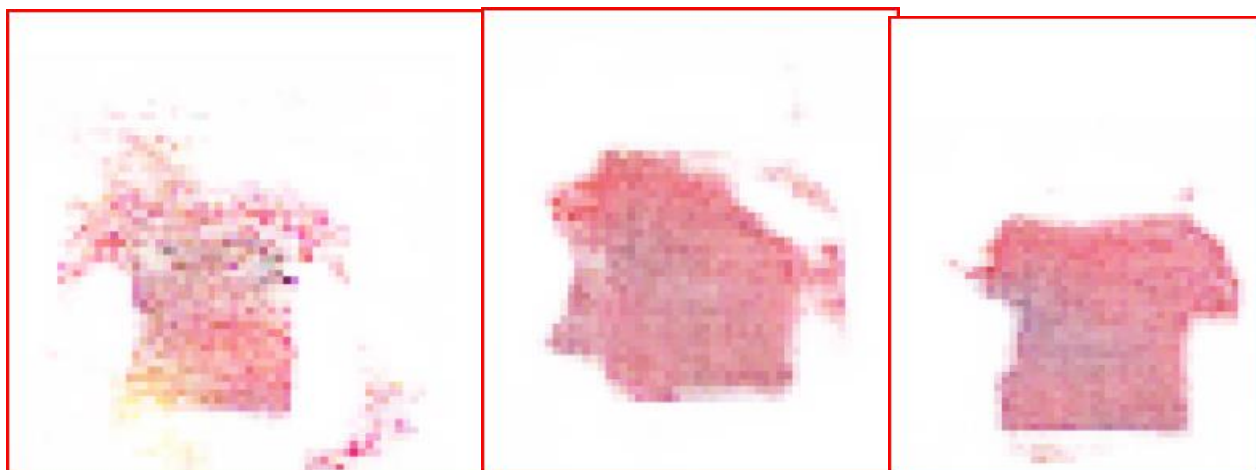
### Key reasons for its improved performance include

Attention GANs achieve improved performance due to their ability to establish fine-grained alignment between text and image features. The attention mechanism allows the generator to focus on specific words in the text—such as associating "black" with color and "shoes" with structure—leading to more accurate and meaningful visual representations. This targeted attention reduces semantic drift, meaning the generated images more faithfully reflect the attributes described in the textual prompt, including details like color, style, and object type.

## VI. Experiments On Random Captions

### Prompts

1. a red balloon in the sky
2. a glass of orange juice
3. a fire truck on the road



## VII. Conclusion

In this project, we explored and implemented a range of text-to-image generation models—Simple GAN, Attention GAN, Stack GAN, StyleGAN-inspired architecture, and a fine-tuned Stable Diffusion model—each progressively improving in complexity and performance. While the earlier GAN-based approaches demonstrated the foundational ability to synthesize images from textual descriptions, they were limited in resolution, coherence, and realism. By incorporating attention mechanisms and style modulation, we achieved moderate improvements in visual fidelity. However, it was the fine-tuned

Stable Diffusion model that delivered the most promising results, both in terms of visual quality and semantic alignment with the input prompts, as reflected by the lowest FID score. Despite certain limitations such as resolution constraints and fine detail preservation, our findings affirm the strong potential of diffusion-based models in fashion-focused text-to-image synthesis. Future work can involve higher-resolution fine-tuning, integration with multimodal transformers, and real-time product customization applications in the fashion industry.