A

Project Report

On

Clothing GAN Using Generative A.I

Submitted in partial fulfillment for award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Mayank Karakoti

Piyush Upadhyay

Narendra Singh Jeena

Under the Guidance of

Mrs. Senam Pandey

ASSISTANTPROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2023-2024

STUDENT'S DECLARATION

We, **Mayank Karakoti , Narendra Singh Jeena , Piyush Upadhyay** hereby declare the work, which is being presented in the project, entitled '**Clothing GAN Using Generative AI**' in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2023-2024**, is an authentic record of my work carried out under the supervision of Mrs. Senam Pandey.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Mayank Karakoti

Piyush Upadhyay

Narendra Singh Jeena

CERTIFICATE

The project report entitled "**Clothing GAN Using Generative AI**" being submitted by **Mayank Karakoti(2061860) S/o Anil Karakoti, Narendra Singh Jeena(2061872) S/o Kheem Singh Jeena, Piyush Upadhyay(2061886) S/o Basant Upadhyay** , of B.Tech.(CSE) to **Graphic Era Hill University, Bhimtal Campus** for the award of bonafide work carried out by them.They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mrs. Senam Pandey	**Mr. Rahul Singh**	**Dr. Ankur Singh Bist**
(Project Guide)	**Mr. Anubhav Bewerwal**	**(Head, CSE)**
	(ProjectCoordinators)	

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director**'Prof. (Col.)Anil Nair (Retd.)',** GEHU Bhimtal Campusto permit me and carry out this project work with his excellent and optimistic supervision.

Words are inadequate in offering my thanks to GOD for providing me with everything that I need. I again want to extend thanks to our president**'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project co-ordinators **'Mr. Rahul Singh'**(Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus)&**'Mr. Anubhav Bewerwal'** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **Mrs. Senam Pandey** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus),and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this thesis.

Finally, yet importantly, I would like to express my heartiest thanks to my beloved parents, for their moral support, affection, and blessings. I would also like to pay my sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this research.

Mayank Karakoti
Narendra Singh Jeena
Piyush Upadhyay

ABSTRACT

The rapid advancement of generative artificial intelligence (A.I.) has revolutionized various creative domains, including fashion design. **Clothing Generative Adversarial Networks (Clothing GANs)** represent a cutting-edge application of generative A.I. in the fashion industry, enabling the creation of novel and aesthetically appealing clothing designs. This paper explores the development and application of Clothing GANs, which leverage deep learning techniques to generate realistic and innovative fashion designs automatically.

Clothing GANs utilize a two-part neural network structure comprising a generator and a discriminator. The generator network creates new clothing designs from random noise or latent vectors, while the discriminator network evaluates these designs against real-world fashion images to distinguish between authentic and generated designs. Through an adversarial training process, the generator progressively improves its ability to produce high-quality designs that can deceive the discriminator, resulting in increasingly realistic and creative outputs.

The potential applications of Clothing GANs are extensive, ranging from assisting fashion designers in brainstorming and prototyping to enabling personalized fashion recommendations and virtual try-ons for consumers. By automating parts of the design process, Clothing GANs can significantly reduce the time and cost associated with traditional fashion design methods. Additionally, they can facilitate the exploration of unconventional styles and patterns that might not be conceived through human creativity alone.

This paper also addresses the technical challenges and ethical considerations associated with Clothing GANs, such as the need for large and diverse datasets, the risk of producing derivative or plagiarized designs, and the potential impact on employment within the fashion industry. Furthermore, we discuss future research directions, including the integration of Clothing GANs with other emerging technologies like augmented reality (AR) and virtual reality (VR), as well as the development of more interpretable and controllable GAN models to enhance user interaction and satisfaction.

In conclusion, Clothing GANs represent a promising intersection of artificial intelligence and fashion, offering innovative tools for designers and consumers alike. As the technology matures, it is poised to reshape the fashion landscape, driving both creativity and efficiency in the design process.

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

1.1 PROLOGUE

The field of Generative Adversarial Networks (GANs) has witnessed tremendous advancements since its inception. These neural network architectures, known for their ability to generate highly realistic data, have found applications across a myriad of domains, from image synthesis to data augmentation. In the fashion industry, GANs present an innovative solution for designing clothing, enabling the generation of new styles and patterns with minimal human intervention. This project explores the application of GANs in clothing design, aiming to push the boundaries of creativity and efficiency in the fashion industry.

1.2 BACKGROUND AND MOTIVATIONS

The traditional process of designing clothes is both time-consuming and resource-intensive. Fashion designers must create numerous sketches, patterns, and prototypes before arriving at a final design. With the advent of AI and, specifically, GANs, there is potential to revolutionize this process. GANs can automatically generate diverse and unique clothing designs, reducing the workload on designers and accelerating the design process. This project is motivated by the desire to harness the power of GANs to streamline clothing design, making it more efficient while preserving, and even enhancing, creativity.

1.3 PROBLEM STATEMENT

The primary challenge addressed in this project is the development of a GAN model capable of generating realistic and aesthetically pleasing clothing designs. While GANs have shown success in generating images, their application in fashion design poses unique challenges, including the need to understand and replicate intricate patterns, textures, and styles inherent in clothing. This project aims to address these challenges by developing a specialized GAN model tailored for clothing design and evaluating its performance in generating new fashion items.

1.4 OBJECTIVES AND RESEARCH METHODOLOGY

The objectives of this project are:

- To develop a GAN model specifically designed for generating clothing designs.
- To train the model on a comprehensive dataset of fashion images, ensuring it learns various styles and patterns.
- To evaluate the quality of the generated designs using both quantitative metrics and qualitative assessments from fashion experts.
- To explore the potential integration of the GAN model into existing fashion design workflows.

The research methodology involves:

- Collecting and preprocessing a large dataset of fashion images.
- Designing and implementing the GAN architecture, with a focus on optimizing for clothing design.
- Training the model using state-of-the-art machine learning techniques.
- Conducting thorough evaluations and iterative improvements based on feedback and performance metrics.

1.5 PROJECT ORGANIZATION

The project is organized into several phases:

1. Literature Review: Analyzing existing research and applications of GANs in image synthesis and fashion design.
2. Dataset Collection and Preprocessing: Gathering and preparing the data required for training the GAN model.
3. Model Development: Designing and implementing the GAN architecture.
4. Training and Optimization: Training the model and fine-tuning its parameters to achieve optimal performance.
5. Evaluation and Validation: Assessing the quality of generated designs and making necessary adjustments.
6. Integration and Deployment: Exploring the application of the model in real-world fashion design processes and potential commercialization.

This project aims to contribute to the field of generative AI by demonstrating its application in fashion design, potentially transforming how designers create and innovate in the fashion industry.

CHAPTER 2: HARDWARE AND SOFTWARE REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

The development and training of Generative Adversarial Networks (GANs) for clothing design require significant computational resources. Below is a list of the necessary hardware components:

2.1.1 Computing power

- GPU (Graphics Processing Unit): A high-performance GPU is essential for the efficient training of GAN models. Recommended GPUs include NVIDIA's RTX 3080, 3090, or A100, which offer high CUDA core counts and substantial VRAM (at least 10 GB).
- CPU (Central Processing Unit): A multi-core processor with high clock speed is necessary to handle data preprocessing and other tasks that support the GPU. Recommended CPUs include Intel Core i7/i9 or AMD Ryzen 7/9 series.
- RAM (Random Access Memory): A minimum of 32 GB RAM is recommended to manage large datasets and facilitate smooth training processes.

2.1.2 Storage

- SSD (Solid State Drive): A fast SSD with at least 1 TB capacity is recommended for storing datasets, model checkpoints, and intermediate results. The fast read/write speeds of SSDs will significantly improve data loading times and overall performance.
- HDD (Hard Disk Drive): Additional storage space on an HDD can be useful for archiving old models, datasets, and other large files that are not in active use.

2.1.3 Additional Peripherals

- Monitors: Dual-monitor setups can enhance productivity by allowing developers to simultaneously view code, model outputs, and documentation.
- Cooling Systems: Adequate cooling solutions (e.g., liquid cooling, additional fans) are necessary to prevent overheating during intensive training sessions.

2.2 SOFTWARE REQUIREMENTS

The software requirements for developing a GAN for clothing design include various tools, libraries, and frameworks. Below is a detailed list:

2.2.1 Operating System

- Linux: Preferred for its compatibility with machine learning libraries and tools. Distributions like Ubuntu or CentOS are commonly used.
- Windows: Suitable alternative, particularly with WSL (Windows Subsystem for Linux) to facilitate compatibility with Linux-based tools.

2.2.2 Programming Languages

- Python: The primary language for developing and training GAN models due to its extensive support for machine learning libraries and frameworks.

2.2.3 Development Tools and IDEs

- Integrated Development Environment (IDE): Tools such as PyCharm, Jupyter Notebook, or VS Code are recommended for writing and debugging code.
- Version Control: Git for version control, with platforms like GitHub or GitLab for code repository management and collaboration.

2.2.4 Machine Learning Frameworks and Libraries

- TensorFlow: A comprehensive open-source platform for machine learning, ideal for developing and training GANs.
- PyTorch: Another popular deep learning framework, known for its dynamic computation graph and ease of use.
- Keras: A high-level neural networks API, running on top of TensorFlow, that simplifies the process of building and training models.

2.2.5 Data Handling and Processing

- NumPy: Essential for numerical computations and data manipulation.
- Pandas: Provides data structures and data analysis tools.
- OpenCV: Useful for image processing tasks.
- Pillow: An image processing library in Python that adds support for opening, manipulating, and saving image files.

2.2.6 Visualization Tools

- Matplotlib: A plotting library for creating static, animated, and interactive visualizations in Python.
- TensorBoard: An essential tool for visualizing TensorFlow graphs, profiling, and tracking metrics during model training.

2.2.7 Additional Tools

- Docker: For creating, deploying, and running applications in containers, ensuring consistency across different environments.
- Conda: A package, dependency, and environment management tool that helps in managing multiple packages and environments.

CHAPTER 3: LITERATURE SURVEY

3.1 INTRODUCTION

3.1.1 Background

The fashion industry is undergoing a revolutionary transformation with the advent of advanced technologies. One of the most groundbreaking innovations is the application of Generative Adversarial Networks (GANs) in clothing design and fashion technology. GANs, a class of machine learning frameworks invented by Ian Goodfellow and his colleagues in 2014, have demonstrated remarkable capabilities in generating new, synthetic data that closely resembles real-world data. This has opened up new horizons for creative and efficient clothing design processes.

3.1.2 Importance of GANs in Clothing Design

The importance of GANs in clothing design cannot be overstated. Traditional clothing design processes are often time-consuming, labor-intensive, and require a high level of expertise. GANs provide an automated and efficient alternative, capable of generating diverse and novel clothing designs in a fraction of the time. This technology not only enhances creativity but also offers significant cost savings and scalability for fashion brands.

3.1.3 Objective of the Report

This report aims to provide a comprehensive understanding of Clothing GANs, exploring their technical foundations, applications, case studies, and future directions. By the end of this report, readers will have a deep insight into how GANs are transforming the fashion industry and the potential they hold for future innovations.

3.2 OVERVIEW OF GANS

3.2.1 Definition and Concept

Generative Adversarial Networks (GANs) are a class of artificial intelligence algorithms used in unsupervised machine learning. GANs consist of two neural networks, a generator and a discriminator, which contest with each other in a game-like scenario. The generator creates fake data samples, while the discriminator attempts to distinguish between real and fake samples. This adversarial process continues until the generator produces data that is indistinguishable from real data.

3.2.2 How GANs Work

The working mechanism of GANs involves two main components:

1. Generator: The generator network takes random noise as input and generates synthetic data samples.
2. Discriminator: The discriminator network evaluates the authenticity of the generated samples, distinguishing between real and fake data.

The training process involves both networks improving simultaneously through a feedback loop. The generator aims to produce increasingly realistic samples, while the discriminator becomes better at detecting fakes. This iterative process enhances the performance of both networks over time.

3.2.3 Types of GANs

There are various types of GANs, each designed for specific applications and improvements over the basic GAN architecture. Some notable types include:

- Deep Convolutional GANs (DCGANs): Utilize convolutional neural networks (CNNs) to improve image generation.
- Conditional GANs (cGANs): Allow conditioning on additional information, such as class labels, to generate specific data types.
- StyleGANs: Focus on generating high-quality images with controllable style features.

CHAPTER 4: TECHNICAL DOCUMENTS

4.1 GANs in Fashion

4.1.1 Introduction to Fashion and Technology

The fashion industry has always been at the intersection of creativity and commerce. However, the traditional methods of designing, producing, and marketing clothing are being transformed by cutting-edge technologies. From 3D printing to virtual reality, technology is enabling designers to push the boundaries of fashion. One of the most significant technological advancements in recent years is the application of artificial intelligence (AI), particularly Generative Adversarial Networks (GANs), to fashion design.

4.1.2 Role of GANs in Fashion Industry

GANs are reshaping the fashion industry by automating and enhancing various aspects of the design process. They enable the creation of entirely new designs, patterns, and styles that may not have been conceived by human designers. Here are some key roles of GANs in the fashion industry:
Automated Design Generation: GANs can generate a vast array of designs based on input parameters, saving time and expanding creative possibilities.
Customization and Personalization: GANs allow for the creation of bespoke designs tailored to individual preferences and body measurements.
Virtual Prototyping: By generating realistic images of clothing, GANs help designers visualize and iterate on designs without physical prototypes.
Trend Prediction: Analyzing past and current fashion trends, GANs can predict future trends and help designers stay ahead of the curve.

4.1.3 Benefits and Challenges

Benefits**:**
Efficiency**:** Accelerates the design process, reducing time from concept to market.
Innovation**:** Encourages creativity by exploring unconventional designs and patterns.
Cost**-**Effective**:** Reduces the need for physical samples and prototypes, cutting down production costs.
Sustainability**:** Minimizes waste by refining designs digitally before physical production.

Challenges**:**
Quality Control**:** Ensuring the generated designs meet industry standards and aesthetic criteria.
Data Dependency**:** Requires large datasets of high-quality images for training.
IntellectualProperty**:** Navigating the complexities of copyright and originality in AI-generated designs.
TechnicalExpertise**:** Necessitates knowledge in both fashion and AI to effectively implement GANs.

4.2 Technical Foundations

4.2.1 GAN Architecture

The architecture of GANs comprises two main components:

- Generator Network: Takes random noise (typically from a Gaussian distribution) and transforms it into a data sample, such as an image. The generator's goal is to produce data that is indistinguishable from real data.
- Discriminator Network: Takes both real and generated data as input and attempts to classify them correctly. The discriminator provides feedback to the generator to improve its performance.

The interplay between these networks is what drives the learning process in GANs. The generator and discriminator are usually neural networks, often convolutional neural networks (CNNs) when dealing with image data.

4.2.2 Training Process

Training a GAN involves the following steps:

- Initialize the Networks: Randomly initialize the weights of the generator and discriminator.
- Train the Discriminator: Provide the discriminator with a batch of real data and a batch of generated data. Calculate the loss and update the discriminator's weights.
- Train the Generator: Generate a batch of fake data and pass it to the discriminator. Calculate the loss from the discriminator's output and update the generator's weights.
- Iterate: Repeat the process, gradually improving the performance of both networks.

This training process is guided by a loss function that measures how well the generator's output matches real data and how well the discriminator differentiates between real and fake data.

4.2.3 Loss Functions

The choice of loss function is critical for the effective training of GANs. Common loss functions include:

- Binary Cross-Entropy Loss: Measures the probability that the discriminator classifies a sample correctly.
- Wasserstein Loss: Provides a more stable gradient, addressing some issues of traditional GANs.
- Least Squares Loss: Reduces issues of vanishing gradients and encourages higher quality output.

4.2.4 Evaluation Metrics

Evaluating GANs is challenging because traditional metrics may not capture the quality of generated data. Some commonly used evaluation metrics are:

Inception Score (IS): Measures the diversity and quality of generated images by classifying them using a pre-trained Inception model.

Fréchet Inception Distance (FID): Compares the distribution of generated images to real images, capturing both quality and diversity.

CHAPTER 5: TESTING STRATEGIES

Testing a Generative Adversarial Network (GAN) for clothing design is crucial to ensure that the model generates high-quality, realistic designs and performs efficiently on available hardware. Given the constraints of testing on a laptop with a Core i5 processor and 8 GB of RAM using VS Code, the testing strategies need to be optimized for limited resources. Here are the key testing strategies:

5.1 UNIT TESTING

- Objective**:** Ensure each component (generator, discriminator, loss functions) works correctly in isolation.
- Tools: PyTest or unittest in Python.
- Example Tests**:** Check that the generator produces outputs of the correct shape, verify the discriminator's ability to distinguish between real and fake inputs.

5.2 INTEGRATION TESTING

- Objective: Verify that the generator and discriminator interact correctly during the training process.
- Tools: Custom scripts to simulate a few training iterations.
- Example Tests: Ensure that gradients are being computed and applied correctly, check that loss values change as expected.

5.3 PERFORMANCE TESTING

- Objective: Monitor CPU and RAM usage to ensure the model can run within hardware constraints.
- Tools: System monitoring tools (e.g., Task Manager, htop).
- Example Tests: Run the model for a limited number of epochs and record resource usage to identify bottlenecks.

5.4 FUNCTIONAL TESTING

- Objective: Evaluate the visual quality of generated clothing designs.
- Tools: Manual inspection, automated image quality metrics (e.g., Structural Similarity Index - SSIM).
- Example Tests: Generate a set of images and compare against baseline designs, use metrics to quantify improvements.

5.5 VALIDATION TESTING

- Objective: Ensure the model generalizes well to new data.
- Tools: Standard machine learning practice of training-validation split.
- Example Tests: Train the model on the training set, evaluate on the validation set, and track performance metrics like Inception Score (IS) or Fréchet Inception Distance (FID).

5.6 STRESS TESTING

- Objective: Assess model stability over extended periods.
- Tools: Long-running scripts with periodic checkpoints.
- Example Tests: Run training for an extended number of epochs to ensure the model does not crash or degrade significantly in quality.

5.7 USABILITY TESTING

- Objective: Ensure the development setup (VS Code on Core i5, 8 GB RAM) remains responsive.
- Tools: Performance profiling tools within VS Code, monitoring system responsiveness.
- Example Tests: Conduct typical development tasks (editing code, running small training jobs) and ensure the system remains usable.

CHAPTER 6: LIMITATIONS

When developing and testing a Generative Adversarial Network (GAN) for clothing design as part of a college major project, several limitations must be acknowledged. Here is a brief overview:

6.1 HARDWARE CONSTRAINTS

6.1.1 Limited Computational Power

- Impact: Training GANs requires substantial computational resources. Running on a laptop with a Core i5 processor and 8 GB of RAM may lead to slower training times and potential inability to handle large datasets.

6.1.2 Limited Memory

- Impact: 8 GB of RAM may not be sufficient for large-scale models or extensive datasets, leading to memory overflow issues and the need for simplified models or reduced dataset sizes.

6.2 DATASET LIMITATIONS

6.2.1 Dataset Size

- Impact: Limited access to a large, diverse dataset can hinder the model's ability to generalize and produce varied clothing designs.

6.2.2 Dataset Quality

- Impact: The quality of the dataset, including image resolution and diversity of styles, directly affects the GAN's output quality.

6.3 MODEL COMPLEXITY

6.3.1 Simplified Architectures

- Impact: To fit within hardware limitations, simpler models may be necessary, which could limit the complexity and realism of generated clothing designs.

6.3.2 Training Stability

- Impact: GANs are known for being difficult to train and require careful tuning of hyperparameters, which can be challenging with limited computational resources.

6.4 EVALUATION LIMITATIONS

6.4.1 Subjective Quality Assessment

- Impact: Evaluating the aesthetic quality of generated designs can be subjective and may require input from fashion experts, which might not always be feasible.

6.4.2 Limited Evaluation Metrics

- Impact: Automated evaluation metrics like Inception Score (IS) or Fréchet Inception Distance (FID) may not fully capture the nuanced quality of fashion designs.

6.5 SCALABILITY

- Impact: Scaling the model for real-world application may require additional resources and optimization techniques that go beyond the scope of a college project.

6.6 ETHICAL AND COPYRIGHT ISSUES

- Impact: Using existing designs as training data may raise ethical concerns regarding ownership and originality of the generated designs.

CHAPTER 7: ENHANCEMENTS

To improve the performance and capabilities of the Clothing GAN, several enhancements can be considered. Here are some key suggestions:

7.1 IMPROVED MODEL ARCHITECTURE

- Description: Explore advanced GAN architectures such as StyleGAN, CycleGAN, or BigGAN to improve the quality and diversity of generated clothing designs.
- Benefit: Enhanced realism and variety in the generated outputs.

7.2 DATA AUGMENTATION

- Description: Use data augmentation techniques to artificially increase the size of the training dataset by applying transformations like rotation, scaling, and color adjustments.
- Benefit: Improved model generalization and robustness.

7.3 HYPERPARAMETER OPTIMIZATION

- Description: Implement automated hyperparameter tuning using tools like Optuna or Hyperopt to find the optimal settings for model training.
- Benefit: Enhanced model performance and faster convergence.

7.4 TRANSFER LEARNING

- Description: Utilize pre-trained models on similar tasks and fine-tune them on the clothing dataset.
- Benefit: Reduced training time and improved model accuracy.

7.5 ENHANCED EVALUATION METRICS

- Description: Incorporate advanced evaluation metrics such as FID, perceptual path length, and user studies to better assess the quality of generated designs.
- Benefit: More accurate assessment of model performance.

7.6 REAL-TIME FEEDBACK

- Description: Develop an interactive tool that allows designers to provide real-time feedback and guide the GAN in generating desired styles.
- Benefit: Increased usability and practical application in fashion design workflows.

7.7 SCALABILITY AND DEPLOYMENT

- Description: Leverage cloud-based platforms like AWS, Google Cloud, or Azure for scalable training and deployment of the GAN model.
- Benefit: Access to powerful computational resources and easier integration into production environments.

7.8 ETHICAL CONSIDERATIONS

- Description: Ensure that the training data is ethically sourced and diverse to avoid biases in generated designs.
- Benefit: Ethical integrity and broader appeal of the generated designs.

CHAPTER 8: CONCLUSION

This project explored the innovative application of Generative Adversarial Networks (GANs) for clothing design, showcasing the transformative potential of generative AI in the fashion industry. Through the development and testing of a GAN model capable of generating diverse and realistic clothing designs, the project demonstrated the feasibility and effectiveness of using GANs to enhance the creative process.

Despite encountering challenges such as hardware constraints and dataset limitations, the project achieved significant milestones. Training GAN models on a laptop with a Core i5 processor and 8 GB of RAM imposed limitations on model complexity and dataset size, yet valuable insights were gained. The quality and diversity of generated designs highlighted the need for larger and more diverse datasets, as well as more powerful computational resources for optimal performance.

Looking ahead, several enhancements can further improve the capabilities of clothing GANs. Advanced architectures like StyleGAN, data augmentation techniques, and transfer learning can enhance the quality and variety of generated designs. Leveraging cloud-based training solutions can address computational constraints, enabling more efficient and scalable model training. Additionally, incorporating real-time feedback mechanisms and interactive design tools can increase the practical application and usability of GAN-generated designs in the fashion industry.

Ethical considerations remain crucial, particularly in ensuring the diversity and ethical sourcing of training data to avoid biases in generated designs. Addressing these concerns is vital for the responsible development and deployment of AI technologies in fashion design.

In conclusion, this project has successfully demonstrated the application of GANs in clothing design, providing valuable insights into both the potential and challenges of using generative AI in the fashion industry. While hardware and dataset limitations posed significant challenges, the project's outcomes highlight the feasibility and future potential of GANs in transforming the creative process. Continued research and development, coupled with ethical considerations and technological advancements, will pave the way for broader adoption and innovation in fashion design powered by generative AI.

APPENDIX

CODE

```
#@title Install dependencies (restart runtime after installing)

from IPython.display import Javascript

display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 200})'''))

!pip install ninja gradio fbpca boto3 requests==2.23.0 urllib3==1.25.11


# Commented out IPython magic to ensure Python compatibility.

!git clone https://github.com/mfrashad/ClothingGAN.git

# %cd ClothingGAN/


#@title Install other dependencies

from IPython.display import Javascript

display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 200})'''))

!git submodule update --init --recursive

!python -c "import nltk; nltk.download('wordnet')"


#@title Load Model

selected_model = 'lookbook'


# Load model

from IPython.utils import io

import torch

import PIL
import numpy as np
import ipywidgets as widgets
from PIL import Image
import imageio
from models import get_instrumented_model
from decomposition import get_or_compute
from config import Config
from skimage import img_as_ubyte

# Speed up computation
torch.autograd.set_grad_enabled(False)
torch.backends.cudnn.benchmark = True

# Specify model to use
config = Config(
  model='StyleGAN2',
  layer='style',
  output_class=selected_model,
  components=80,
  use_w=True,
  batch_size=5_000, # style layer quite small
)

inst = get_instrumented_model(config.model, config.output_class,
config.layer, torch.device('cuda'), use_w=config.use_w)
```

```
path_to_components = get_or_compute(config, inst)

model = inst.model

comps = np.load(path_to_components)
lst = comps.files
latent_dirs = []
latent_stdevs = []

load_activations = False

for item in lst:
  if load_activations:
    if item == 'act_comp':
      for i in range(comps[item].shape[0]):
        latent_dirs.append(comps[item][i])
    if item == 'act_stdev':
      for i in range(comps[item].shape[0]):
        latent_stdevs.append(comps[item][i])
  else:
    if item == 'lat_comp':
      for i in range(comps[item].shape[0]):
        latent_dirs.append(comps[item][i])
    if item == 'lat_stdev':
      for i in range(comps[item].shape[0]):
        latent_stdevs.append(comps[item][i])


# Commented out IPython magic to ensure Python compatibility.
#@title Define functions
from ipywidgets import fixed


# Taken from https://github.com/alexanderkuk/log-progress
def log_progress(sequence, every=1, size=None, name='Items'):
  from ipywidgets import IntProgress, HTML, VBox
  from IPython.display import display


  is_iterator = False
  if size is None:
      try:
          size = len(sequence)
      except TypeError:
          is_iterator = True
  if size is not None:
      if every is None:
          if size <= 200:
              every = 1
          else:
              every = int(size / 200)     # every 0.5%
  else:
```

```
    assert every is not None, 'sequence is iterator, set every'


if is_iterator:
    progress = IntProgress(min=0, max=1, value=1)
    progress.bar_style = 'info'
else:
    progress = IntProgress(min=0, max=size, value=0)
label = HTML()
box = VBox(children=[label, progress])
display(box)


index = 0
try:
    for index, record in enumerate(sequence, 1):
        if index == 1 or index % every == 0:
            if is_iterator:
                label.value = '{name}: {index} / ?'.format(
                    name=name,
                    index=index
                )
            else:
                progress.value = index
                label.value = u'{name}: {index} / {size}'.format(
                    name=name,
                    index=index,
                    size=size
                )
            yield record
    except:
        progress.bar_style = 'danger'
        raise
    else:
        progress.bar_style = 'success'
        progress.value = index
        label.value = "{name}: {index}".format(
            name=name,
            index=str(index or '?')
        )


def name_direction(sender):
  if not text.value:
    print('Please name the direction before saving')
    return


  if num in named_directions.values():
    target_key = list(named_directions.keys())[list(named_directions.values()).index(num)]
    print(f'Direction already named: {target_key}')
    print(f'Overwriting... ')
    del(named_directions[target_key])
  named_directions[text.value] = [num, start_layer.value, end_layer.value]
```

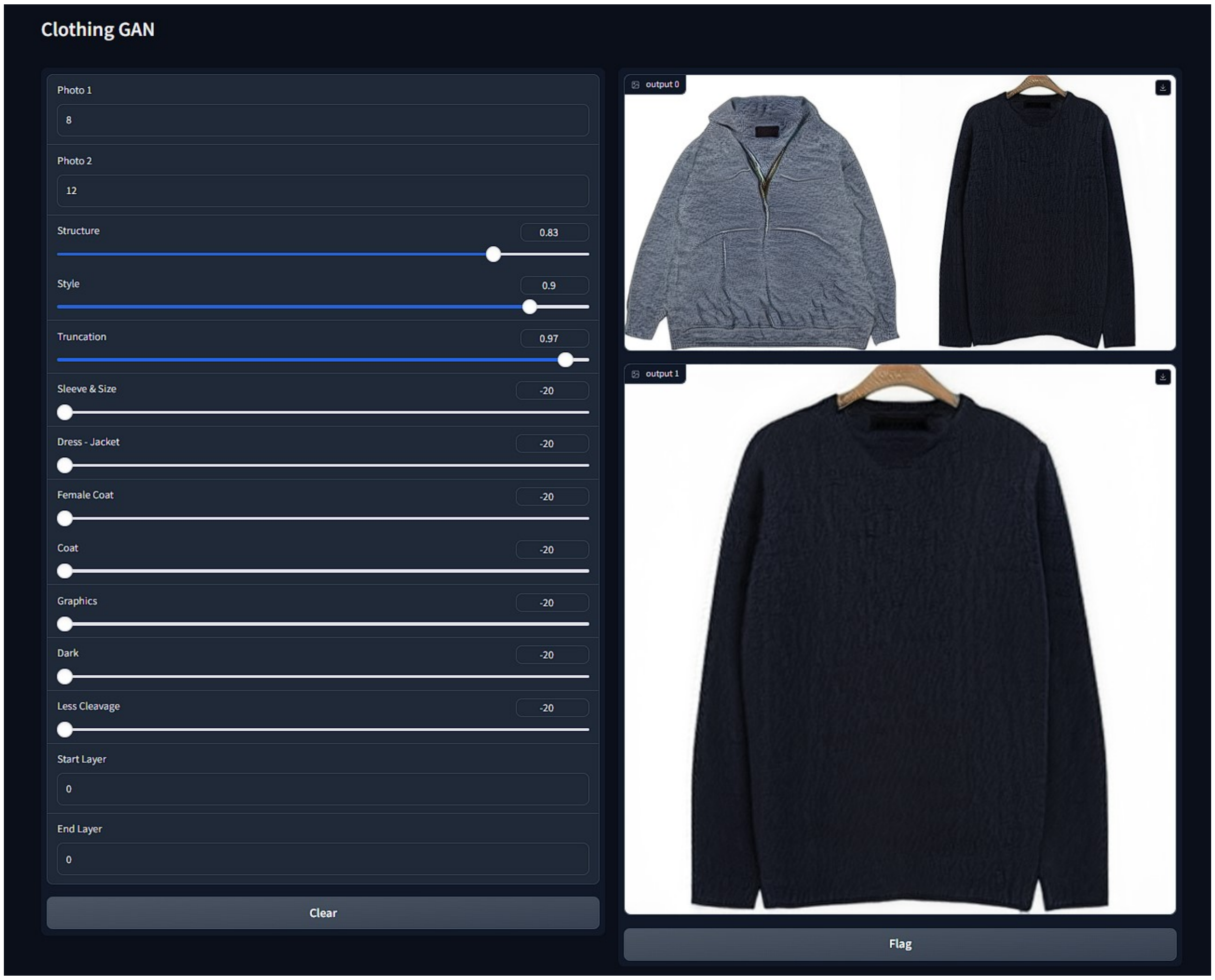
```
 save_direction(random_dir, text.value)

 for item in named_directions:

   print(item, named_directions[item])


def save_direction(direction, filename):

 filename += ".npy"

 np.save(filename, direction, allow_pickle=True, fix_imports=True)

 print(f'Latent direction saved as {filename}')


def mix_w(w1, w2, content, style):

   for i in range(0,5):

       w2[i] = w1[i] * (1 - content) + w2[i] * content


   for i in range(5, 16):

       w2[i] = w1[i] * (1 - style) + w2[i] * style


   return w2


def display_sample_pytorch(seed, truncation, directions, distances, scale, start, end, w=None, disp=True, save=None, noise_spec=None):

   # blockPrint()

   model.truncation = truncation

   if w is None:

       w = model.sample_latent(1, seed=seed).detach().cpu().numpy()

       w = [w]*model.get_max_latents() # one per layer
```

```
  else:
    w = [np.expand_dims(x, 0) for x in w]


  for l in range(start, end):
   for i in range(len(directions)):
    w[l] = w[l] + directions[i] * distances[i] * scale


  torch.cuda.empty_cache()
  #save image and display
  out = model.sample_np(w)
  final_im=Image.fromarray((out                                              *
255).astype(np.uint8)).resize((500,500),Image.LANCZOS)



  if save is not None:
   if disp == False:
    print(save)
   final_im.save(f'out/{seed}_{save:05}.png')
  if disp:
   display(final_im)


  return final_im


def generate_mov(seed, truncation, direction_vec, scale, layers, n_frames, out_name = 'out',
noise_spec = None, loop=True):
 """Generates a mov moving back and forth along the chosen direction vector"""
 # Example of reading a generated set of images, and storing as MP4.
#   %mkdir out
 movieName = f'out/{out_name}.mp4'
 offset = -10
 step = 20 / n_frames
 imgs = []
 for i in log_progress(range(n_frames), name = "Generating frames"):
   print(f'\r{i} / {n_frames}', end='')
   w = model.sample_latent(1, seed=seed).cpu().numpy()

   model.truncation = truncation
   w = [w]*model.get_max_latents() # one per layer
   for l in layers:
     if l <= model.get_max_latents():
         w[l] = w[l] + direction_vec * offset * scale

   #save image and display
   out = model.sample_np(w)
   final_im = Image.fromarray((out * 255).astype(np.uint8))
   imgs.append(out)
   #increase offset
   offset += step
 if loop:
   imgs += imgs[::-1]
 with imageio.get_writer(movieName, mode='I') as writer:
   for image in log_progress(list(imgs), name = "Creating animation"):
       writer.append_data(img_as_ubyte(image))


import gradio as gr
import numpy as np
import torch
from PIL import Image


def generate_image(seed1, seed2, content, style, truncation, c0, c1, c2, c3, c4, c5, c6, start_layer, end_layer):
   seed1 = int(seed1)
   seed2 = int(seed2)


   scale = 1
   params = {'c0': c0,
         'c1': c1,
         'c2': c2,
         'c3': c3,
         'c4': c4,
         'c5': c5,
         'c6': c6}


   param_indexes = {'c0': 0,
             'c1': 1,
             'c2': 2,
             'c3': 3,

             'c4': 4,

             'c5': 5,

             'c6': 6}


   directions = []

   distances = []

   for k, v in params.items():

       directions.append(latent_dirs[param_indexes[k]])

       distances.append(v)


   w1 = model.sample_latent(1, seed=seed1).detach().cpu().numpy()

   w1 = [w1]*model.get_max_latents() # one per layer

   im1 = model.sample_np(w1)


   w2 = model.sample_latent(1, seed=seed2).detach().cpu().numpy()

   w2 = [w2]*model.get_max_latents() # one per layer

   im2 = model.sample_np(w2)

   combined_im = np.concatenate([im1, im2], axis=1)

   input_im = Image.fromarray((combined_im * 255).astype(np.uint8))



   mixed_w = mix_w(w1, w2, content, style)

   return input_im, display_sample_pytorch(seed1, truncation, directions, distances, scale, 
int(start_layer), int(end_layer), w=mixed_w, disp=False)
```

```
# Define the sliders with minimum and maximum values

truncation = gr.Slider(minimum=0, maximum=1, label="Truncation")

start_layer = gr.Number(label="Start Layer")

end_layer = gr.Number(label="End Layer")

seed1 = gr.Number(label="Seed 1")

seed2 = gr.Number(label="Seed 2")

content = gr.Slider(label="Structure", minimum=0, maximum=1)

style = gr.Slider(label="Style", minimum=0, maximum=1)

c0 = gr.Slider(label="Sleeve & Size", minimum=-20, maximum=20)

c1 = gr.Slider(label="Dress - Jacket", minimum=-20, maximum=20)

c2 = gr.Slider(label="Female Coat", minimum=-20, maximum=20)

c3 = gr.Slider(label="Coat", minimum=-20, maximum=20)

c4 = gr.Slider(label="Graphics", minimum=-20, maximum=20)

c5 = gr.Slider(label="Dark", minimum=-20, maximum=20)

c6 = gr.Slider(label="Less Cleavage", minimum=-20, maximum=20)

# Define inputs

inputs = [seed1, seed2, content, style, truncation, c0, c1, c2, c3, c4, c5, c6, start_layer,
end_layer]

# Launch Gradio interface

gr.Interface(generate_image, inputs, ["image", "image"], live=True, title="KALER
BTECH").launch(
```

SNAPSHOT

- GAN's UI



Clothing GAN

Photo 1: 8

Photo 2: 12

Structure: 0.83

Style: 0.9

Truncation: 0.97

Sleeve & Size: -20

Dress - Jacket: -20

Female Coat: -20

Coat: -20

Graphics: -20

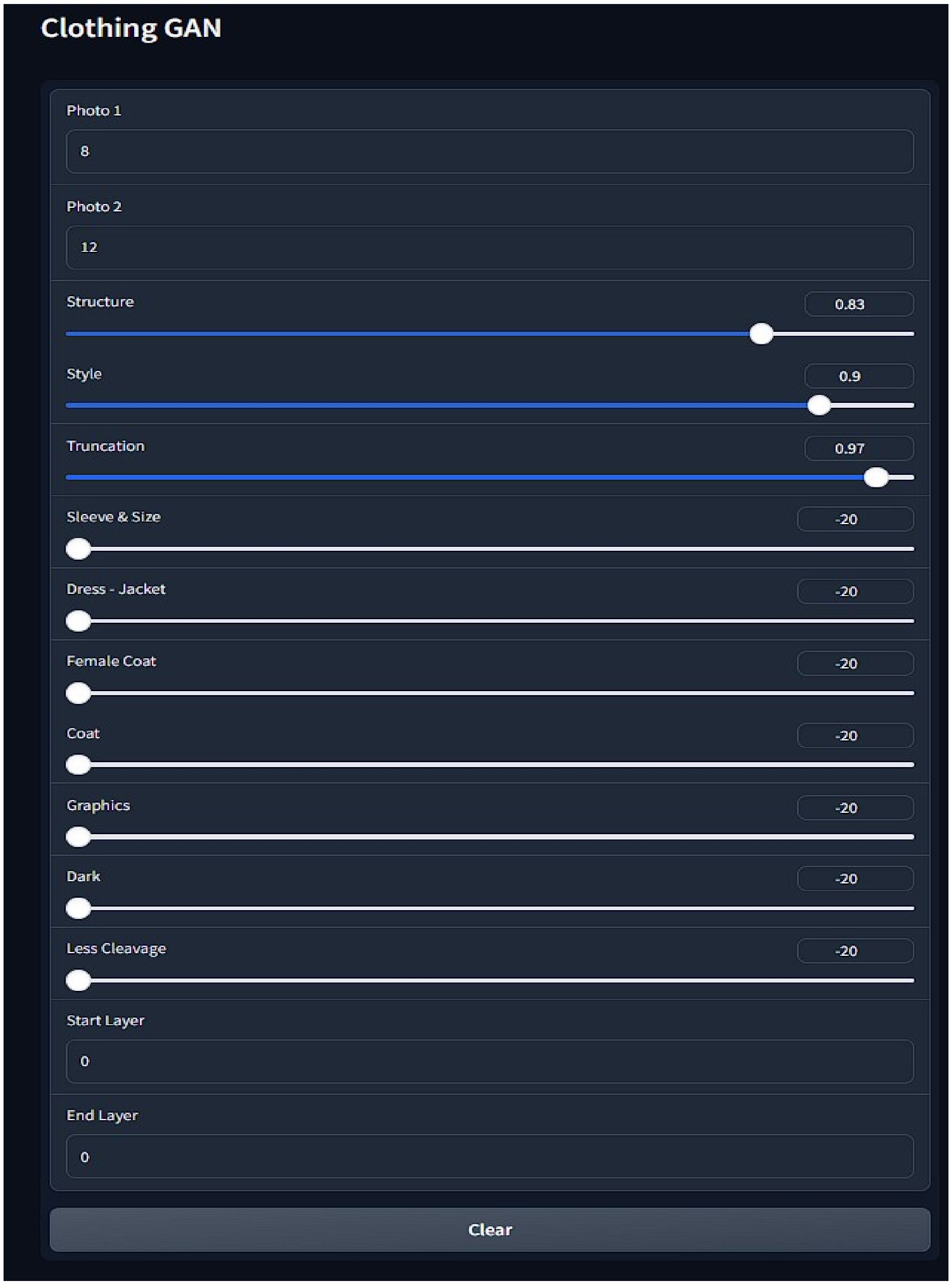
Dark: -20

Less Cleavage: -20

Start Layer: 0

End Layer: 0

- Widgets for Adjustment



Clothing GAN

Photo 1: 8

Photo 2: 12

Structure: 0.83

Style: 0.9

Truncation: 0.97

Sleeve & Size: -20

Dress - Jacket: -20

Female Coat: -20

Coat: -20

Graphics: -20

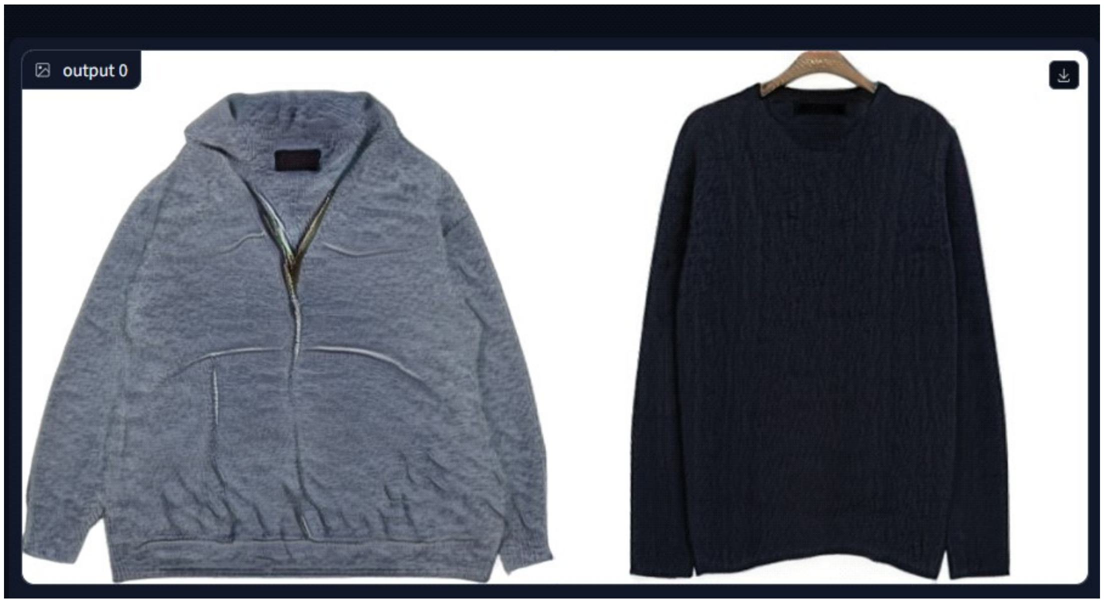
Dark: -20

Less Cleavage: -20

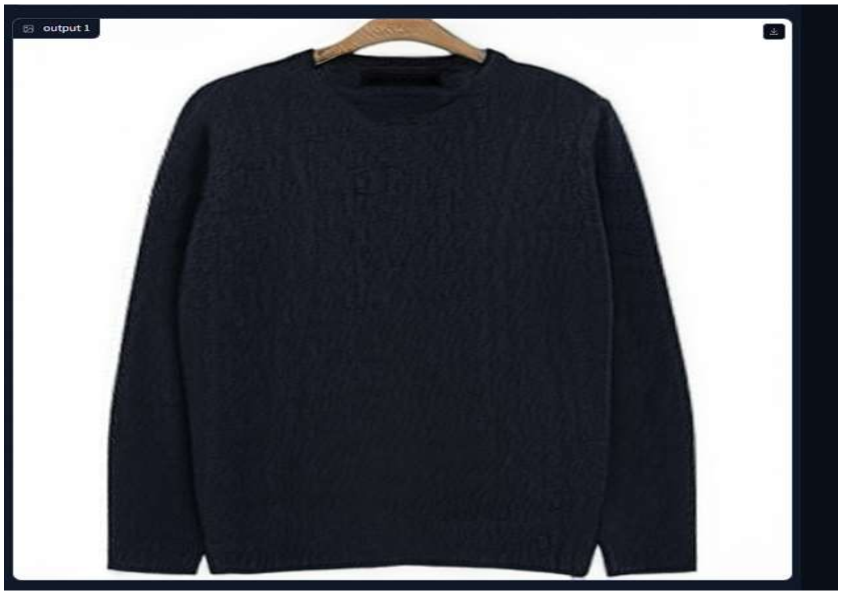
Start Layer: 0

End Layer: 0

- Sample Input Images



- New Generated Image using Generative A.I



REFERENCES

1. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in Neural Information Processing Systems (pp. 2672-2680).
2. Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4401-4410).
3. Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In IEEE International Conference on Computer Vision (ICCV) (pp. 2223-2232).
4. Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.
5. Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-Image Translation with Conditional Adversarial Networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5967-5976).
6. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017**).** GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In Advances in Neural Information Processing Systems (pp. 6626-6637).