Cardiff University

School of Computer Science and Informatics, Cardiff University

MSc Advanced Computer Science

CMT400 Dissertation

Virtual Try-On with Deep Generative Networks

Author: Yu Liu

Supervisor: Yukun Lai

Moderator: Xianfang Sun

Content

Abstract

With the rapid growth of e-commerce, online shopping has become a primary method for consumers. However, the lack of a physical try-on experience often leads to issues with size and style mismatches, increasing return rates and waste. Virtual Try-On (VTON) technology aims to simulate real-life try-ons, providing a more intuitive shopping experience. This study focuses on developing a deep learning-based VTON model that uses a generator from existing paper combined with its own multiscale discriminator and other techniques for adversarial training to achieve realistic virtual try-on effects, while solving the challenges of clothing deformation, occlusion processing, and human pose matching. Using Generative Adversarial Networks (GAN) and image translation techniques, we integrate clothing and human images to improve training accuracy. Results show that although the proposed VTON model was trained under limited resources, improvements are still needed in garment fitting, occlusion handling, and image realism. The model successfully addressed basic pose and clothing diversity requirements but lacked detail in areas such as wrinkles and texture. The ALIAS (ALIgnment-Aware Segment) Generator was reproduced, and a multiscale discriminator was added to complete adversarial training. While there is room for improvement in image detail and realism, this study demonstrates the feasibility of training GAN models for virtual try-on systems. Future work will enhance stability and garment detail while improving resource efficiency.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor, Professor Yukun Lai, for your guidance and assistance throughout this dissertation. Your advice and feedback have been crucial to the completion of this work. I also want to thank my family, especially my parents, for their unconditional support and encouragement during my research, which provided me with immense emotional strength. Finally, I would like to thank my friends, who have been invaluable companions. Together, we motivated each other and worked hard throughout the dissertation process, making this journey much more meaningful.

1. Introduction

1.1 Project Motivation

With the rapid growth of online clothing shopping in recent years, many people still prefer to purchase clothing in physical stores. If e-commerce can overcome the limitations of the lack of physical try-on experiences and address the persistent issue of high return rates, it would help consumers better predict the fit of garments during the shopping process and understand how the purchased items would actually appear on their bodies. This would reduce uncertainty in shopping decisions, thereby lowering return rates and accelerating the development of online clothing shopping. Image-based virtual try-on (VTON) technology allows consumers to virtually replace the clothing on a person's body with the items they wish to try on, helping them overcome the drawbacks of not being able to physically try on clothes. When consumers' sensory experiences are satisfied, they are more likely to be motivated to make purchases.

The earliest virtual try-on models largely relied on image processing techniques (Ma and Deng, 2004), using static or simple 2D images to overlay clothing images onto photos of users. Although this approach could initially simulate a try-on experience, it lacked realism and dynamic adaptability, making it difficult to handle different angles or movements. While virtual try-on technology shares similarities with image synthesis, it faces more unique and complex challenges. Later, Huang et al. introduced convolutional neural networks (CNN) for human parsing and pose estimation (Huang et al., 2017), and Zhu et al. proposed a pose-guided clothing transfer model, which utilized deep learning methods to simulate clothing deformation and fitting (Zhu et al., 2017). While these techniques improved certain aspects, they still fell short in generating highly realistic images like GANs. The challenges in virtual try-on models include the following: (1) the accuracy of multi-pose human models and whether or not body parts are retained, (2) the clothing must naturally deform to fit the person while maintaining fabric details, and (3) there are still many challenges in occlusion handling. Moreover, the resolution of images generated by existing technologies is often too low to meet consumers' expectations for high-definition try-on experiences.

The convenience of online shopping has led to increasing consumer reliance, and virtual try-on technology holds immense potential for future applications. Many e-commerce platforms have already begun exploring how to leverage this technology to

enhance user experience, creating a vast market demand for its development. In a customer-centric business environment, large e-commerce platforms are focusing on meeting consumers' desire for personalized experiences. Virtual try-on technology can fulfill the need for an intuitive shopping experience, making it a key area of competition as businesses strive to improve user engagement. This, in turn, opens up significant market opportunities for the advancement of the technology.

1.2 Aims and Objectives

The earliest use of Generative Adversarial Networks (GAN) for virtual try-on technology was introduced by Han et al. in 2018 (Han et al., 2018). Following this, several studies utilizing GANs to achieve similar goals were proposed, such as CP-VTON (Wang et al., 2018), ClothFlow (Han et al., 2019), and VTN (Dong et al., 2019). These early researchers laid the foundation for applying GANs to virtual try-on technology and demonstrated a technical process for image generation through the following steps: (1) clothing images are warped according to the person's pose and body posture, (2) the warped clothing images are fused with the person's image to ensure the clothing naturally fits the person's pose and body shape, and (3) during the process of warping and fusion, details such as wrinkles, shadows, and fabric textures are preserved. However, while these methods were able to produce basic virtual try-on effects, several significant challenges remain, including (1) resolution issues, (2) misalignment errors, and (3) loss of detail.

In 2021, Choi et al. proposed a solution to the misalignment problem between clothing and the person by introducing "Misalignment-Aware Normalization." This technique not only addressed the alignment errors but also enabled the generation of higher-resolution virtual try-on images (1024x768), significantly enhancing the realism of details. This approach provided an effective solution to the challenges mentioned earlier, particularly in improving resolution and preserving fine details (Choi et al., 2021).

The overall objective of this thesis is to develop a high-precision, high-resolution virtual try-on model that can realistically simulate the effect of different garments on users, thereby enhancing the online shopping experience and simultaneously reducing the return rate in e-commerce.

- **Retraining the GAN model:** Instead of fully relying on the original pre-trained model, the VITON-HD framework's ALIAS Generator (code available in the original literature) and a custom-built Multiscale Discriminator (no code in the

original literature) were used to perform complete adversarial training, aiming to achieve similar or better try-on performance.

- **Enhancing model stability and generalization:** Dropout techniques (Srivastava et al., 2014) were applied to reduce overfitting and improve the model's generalization capabilities, enabling better handling of various poses and clothing types.
- **Increasing training stability:** Gradient Penalty (Gulrajani et al., 2017) was employed to prevent instability issues such as vanishing or exploding gradients, thus improving the overall stability of the training process.
- **Overcoming resource limitations in training:** Due to limited GPU RAM, Gradient Accumulation (Andersson et al., 2022) was used in place of a larger batch size. By accumulating gradients over multiple smaller batches, the effect of training with larger batches was achieved without overburdening the GPU RAM.

This study will use a dataset consisting of high-resolution (1024x768) clothing and human images for experimentation. The model's performance will be validated through quantitative evaluation of the generated image quality (e.g., clarity, retention of clothing details) and qualitative analysis of user satisfaction.

1.3 Scope of the projects

This study focuses on developing a high-resolution virtual try-on model using Generative Adversarial Networks (GAN). Specifically, we will employ multi-pose human models and clothing image deformation, utilizing the VITON-HD framework. The ALIAS (ALIgnment-Aware Segment) Generator and Multiscale Discriminator models (Karnewar and Wang, 2020) will be introduced for adversarial training to achieve effective alignment between clothing and human poses. Gradient Penalty and Dropout techniques will be applied to improve the model's stability and generalization. Additionally, Gradient Accumulation will be used to reduce the burden on GPU RAM, allowing the model to simulate large-batch training effects even with limited GPU resources.

This study will utilize a dataset containing high-resolution (1024x768) images of clothing and humans. The dataset includes various human poses and a wide range of clothing styles, aiming to overcome the resolution limitations of traditional virtual try-on (VTON) technologies.

The scope of this study is limited to virtual try-on of upper-body clothing, excluding footwear, accessories, or other types of garments. It will not involve virtual try-on training in different scenarios. From a technical perspective, this study will not

include the training or optimization of the Segmentation Generator (Athar et al., 2021) or the Geometric Matching Module (GMM) Generator (Wang et al., 2018), as these modules will rely on existing pre-trained models. The focus of this research is primarily on the alignment between clothing and poses, as well as high-resolution image generation. Furthermore, this study does not address physical try-on technologies, instead concentrating solely on image-based virtual try-on models.

The emphasis on excluding accessories and other items beyond upper-body clothing highlights that these areas require different technologies and model architectures, which will serve as directions for future research expansion. Developing a versatile virtual try-on model that supports a variety of outfit combinations will enhance the consumer experience and offer more diverse options. Additionally, applying virtual try-on technology in different scenarios, such as simulating the try-on effect under varying lighting conditions, is also a key goal for future development.

2. Literature Review

Since the introduction of VITON technology, there has been a significant leap in the field of virtual try-on, compared to earlier methods that simply used static images or basic 2D pictures to overlay clothing onto user photos. This technology provides a more intuitive presentation of how garments appear on consumers. Following this, numerous models relying on Generative Adversarial Networks (GAN) were proposed (Dong et al., 2019; Lewis et al., 2021). Although these models have addressed some fundamental issues regarding the correspondence between clothing and human images, many challenges still remain.

First, the images generated by these models often have low resolution, and when the resolution is increased, they fail to deliver the necessary level of detail, falling short of consumer expectations in real-world applications. Second, when handling body pose deformations or complex movements, misalignment between clothing and the body or loss of image details frequently occurs, especially at the junctions between the clothing and the body.

To overcome these limitations, VITON-HD was developed. VITON-HD introduced the ALIAS Generator, which significantly improved the alignment accuracy between clothing and the human body, reducing misalignment issues. Additionally, it optimized the resolution problem by raising the image generation quality to 1024x768, making the virtual try-on images clearer and more realistic, better

showcasing clothing details and textures. These advancements have not only solved many of the bottlenecks in traditional VITON technology but also further propelled the development and adoption of virtual try-on technology in practical applications.

2.1 Early Image Processing Techniques

Early image segmentation techniques mainly included thresholding (Otsu, 1979) and edge detection (Canny, 1986). Thresholding segments the foreground and background areas based on pixel intensity values, often used in simple binarization tasks. However, thresholding is highly sensitive to noise and changes in brightness, especially when the image contains areas with significant variations in lighting, leading to either over-segmentation or under-segmentation. Edge detection, on the other hand, is used to distinguish object boundaries by identifying regions with sharp changes in brightness. Common methods include the Sobel operator and Canny edge detection. However, both techniques have several limitations when dealing with more complex image scenes.

Thresholding often fails to correctly distinguish regions when image contrast is low or when there are irregular changes in brightness, resulting in incorrect segmentation. Edge detection struggles to accurately locate object boundaries when the edges are blurry or when the background is complex. Additionally, these methods typically require manual parameter tuning, making them unable to adapt to a wide range of images and scenes.

2.2 The Evolution of Image Segmentation with Deep Learning

With the rapid development of deep learning, convolutional neural network (CNN) models, such as U-Net (Ronneberger et al., 2015) and Mask R-CNN (He et al., 2017), have brought revolutionary changes to image segmentation. U-Net employs a symmetrical encoder-decoder structure, where the encoding stage captures multi-level features of the image and the decoding stage gradually restores details. Skip connections further enhance the restoration of image details, making U-Net highly effective in scenarios that require fine segmentation. The SegGenerator model structure in this thesis is a typical CNN-based image segmentation architecture, and it shares characteristics similar to those of U-Net.

2.3 K-Means Clustering vs. Geometric Matching Modules (GMM)

K-Means (Lloyd, 1982) is one of the earliest and most widely used techniques for image segmentation and clustering tasks. The concept of K-Means involves dividing data into K clusters by minimizing the distance between each point and its corresponding cluster center, generating a center for each cluster. Its simple calculation and fast processing speed made it popular in early applications. However, it is not well-suited for non-linear data distributions. Additionally, K-Means is highly sensitive to initial values, often leading to local minima, which makes it inadequate for capturing the complex correspondence between clothing structures and human body shapes in virtual try-on scenarios. The deformations between clothing and the human body are typically highly non-linear and complex, causing K-Means to struggle with achieving accurate matching in such applications.

To address this issue, the Geometric Matching Module (GMM) emerged as a more suitable solution. GMM is a probabilistic clustering method that can handle non-linear data distributions and capture more complex shapes and structures, which is particularly important in virtual try-on models. It enables more precise matching between clothing and the diverse shapes of the human body. The advantage of GMM lies in its use of probabilistic models to describe the data, allowing for more flexible segmentation and matching based on the probability distribution of each data point. This solves the problem of handling highly variable data, which K-Means fails to address. By introducing probabilistic modeling, GMM enhances the flexibility and accuracy of data segmentation and matching, leading to significant improvements in virtual try-on models.

2.4 Image Deformation Technology

Early approaches to image deformation relied on affine transformation (Klein, 1872), a linear transformation that can perform operations such as translation, rotation, scaling, and shearing on an image. It uses a linear matrix to consistently transform all points in the image, but it is typically limited to simple changes and not well-suited for handling complex deformations. In the context of virtual try-on models, it is quite challenging for affine transformation to accommodate more complicated non-linear

deformations, such as those required for changes in human posture or clothing deformation.

To address these issues, Thin Plate Spline (TPS) transformation (Bookstein, 1989) was later introduced into virtual try-on models. TPS is a non-linear transformation technique based on control points, which optimizes the positions of these points to create smooth and flexible warping effects. Therefore, TPS is well-suited for applications involving clothing deformation, as it can flexibly adapt to different human poses, providing more precise control over deformations than affine transformation (Wang et al., 2018). Although TPS significantly improved the accuracy of deformations, it may still struggle to capture fine deformations between clothing and the human body in high-resolution images and scenes with intricate details.

To further enhance accuracy, VITON-HD introduced the ALIAS (ALIgnment-Aware Segment) normalization technique (Choi et al., 2021). Through adversarial training, the ALIAS generator not only generates realistic clothing images but also interacts with the discriminator in a learning process, progressively improving image realism. This generator specifically optimizes clothing details and human pose variations, addressing the limitations of TPS in terms of precision and enabling virtual try-on models to generate higher-quality, more detailed composite images.

2.5 Normalization Techniques in Deep Learning

Batch Normalization (Ioffe and Szegedy, 2015) was one of the most widely applied normalization techniques in the early days. Its core concept is to stabilize the output distribution of neural networks by normalizing the outputs of each layer. Batch Normalization standardizes the mean and variance of a mini-batch's outputs, reducing distribution differences across input data. This effectively mitigates problems like vanishing and exploding gradients, thus speeding up network convergence. However, its performance heavily relies on large batch sizes, making it less effective in small-batch training scenarios.

To address these limitations, Conditional Normalization techniques (Dumoulin and de Vries, 2017) were introduced and have been widely adopted in Generative Adversarial Networks (GAN) and style transfer tasks. These techniques adjust the affine parameters based on external data, making the normalization process more flexible. For example, Conditional Batch Normalization and Adaptive Instance Normalization have demonstrated good results in style transfer. Additionally,

techniques like SPADE and SEAN apply spatially varying affine transformations using segmentation maps, further enhancing the realism and detail retention in generated images. In virtual try-on models, these methods help achieve more realistic alignment between clothing and human body deformations.

Another significant advancement aimed at addressing instability in GAN training is Spectral Normalization (Miyato et al., 2018). By regularizing the weights of both the generator and discriminator, Spectral Normalization controls the gradient range, effectively reducing gradient explosions, thus producing more stable and realistic generated images. This technique plays a crucial role in improving the quality of high-resolution image generation in virtual try-on models.

In addition to Spectral Normalization, Layer Normalization and Instance Normalization are also common alternatives, particularly effective in small-batch training and style transfer applications. These methods normalize at different levels or across samples, avoiding the batch size dependency seen in Batch Normalization. They have shown greater adaptability in certain image generation and style transfer tasks. Altogether, these advancements have significantly contributed to the development of virtual try-on technology.

Before the advent of Generative Adversarial Networks (GAN), the primary techniques for evaluating image realism relied on traditional classifiers such as Convolutional Neural Networks (CNN). These classifiers were designed to distinguish between different types of images and learned to recognize differences through pre-trained models. However, their performance in handling forged images was quite limited, as they were not designed to precisely detect subtle differences between generated and real images. A major drawback of traditional discriminators is that they cannot dynamically improve and optimize their ability to distinguish fake images alongside the generator's improvements, often leading to misjudgments. This makes them unsuitable for virtual try-on models.

With the development of GANs, the GAN discriminator was introduced as a tool specifically designed to counteract the generator. Unlike traditional classifiers, the GAN discriminator evolves and improves as the generator gets better, enabling interactive learning between the two. This allows the discriminator to detect the fine details that differentiate generated images from real ones.

2.6 Multiscale Discriminator and Adversarial Training

The **Multiscale Discriminator** is an advanced technique used in adversarial training within virtual try-on models (Karnewar and Wang, 2020). Unlike traditional single-scale discriminators, the Multiscale Discriminator evaluates image realism across multiple scales. It is more effective at capturing image details at different resolutions, improving accuracy. Additionally, it can identify subtle compositional issues between the clothing and the human body across various image layers, helping the generator produce more realistic and natural clothing composites.

This study selects VITON-HD as the foundation for improvements because, as mentioned earlier, it addresses many of the bottlenecks present in traditional VITON technology. The model's pre-trained ALIAS Generator has demonstrated high-quality image generation in virtual try-on models. However, the original paper only provides a pre-trained model and lacks a complete adversarial training process, particularly in the discriminator component. This omission leads to certain limitations in updating and optimizing the model, as it cannot dynamically learn based on the quality of the generated images, potentially restricting the realism and detail representation of the output.

To address this issue, this study designs a Multiscale Discriminator, which undergoes adversarial training with the ALIAS Generator. This allows the generator to learn and enhance image quality at different scales. Not only does this resolve the limitations of VITON-HD’s sole reliance on the generator, but it also improves the model's adaptability, making it more effective in handling more complex human poses and clothing details. Additionally, this enhancement aids in further optimization and generalization capabilities.

In the following sections, I will delve into the adversarial training process between the Multiscale Discriminator and the ALIAS Generator, along with the technical details. This will further illustrate how these improvements enhance the image generation performance of the virtual try-on model.

3. Methodology

3.1 Technology Selection and Detailed Explanation

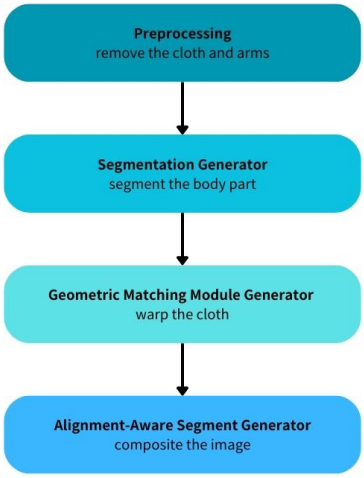

Figure 1: Process of VITON

3.1.1 Preprocessing Techniques

Human Parsing is a technique used to segment a human image into different regions, distinguishing specific body parts such as the upper body, lower body, arms, and neck (Liu et al., 2015). In virtual try-on technology, it enables precise identification of the areas where clothing needs to be synthesized, ensuring that the garments are accurately fitted onto the model. This technique typically relies on deep learning models for pixel-level segmentation.

In the model used in this thesis, the **get_parse_agnostic** function applies human parsing data to mask specific body parts. This prevents these regions from interfering with the clothing generation process, ensuring that the garments are naturally placed on the model. By accurately segmenting each body part, this method is better suited to handle various poses and clothing types, offering improved accuracy and flexibility compared to traditional pose estimation methods. Other techniques, such as simple bounding box detection, only provide rough regional information and struggle to deal with the overlapping of different body parts.

OpenPose is a pose estimation framework capable of detecting keypoints on the human body and providing 2D pose data (Cao et al., 2017). These coordinates help the system understand the body's posture in space. In the model, OpenPose is used to

extract human pose data and generate **pose_keypoints_2d**, marking various joint points on the body. This data is crucial for generating "agnostic" images, ensuring that the clothing accurately fits the upper body in the generated images, and preventing distortions or misalignments.

OpenPose's accuracy in pose estimation is very high, as it can quickly detect up to 135 body keypoints, allowing the system to handle various complex poses. OpenPose has already been widely applied in the field of pose estimation, making it easily adaptable to virtual try-on models. Compared to traditional keypoint estimation techniques, such as template matching, HOG (Histogram of Oriented Gradients) + SVM, or Pictorial Structures, OpenPose introduces deep learning through multi-stage Convolutional Neural Networks (CNN). This allows it to simultaneously handle multi-person keypoint estimation, detect basic limb positions, and accurately locate every joint point, including finer details like finger and facial keypoints—something that traditional methods struggle to achieve.

The pre-processing steps used in this model, including human parsing and estimation of pose using OpenPose, follow the same methodology outlined in the original paper. These steps are widely employed in virtual fitting systems utilising GAN models to ensure consistency in accurately matching garments to body shapes and handling pose changes.

3.1.2 Training Model Techniques

The **Segmentation Generator** is able to produce high-quality and stable segmentation images for virtual try-on tasks due to three key components: (1) U-Net architecture, (2) Dropout, and (3) Batch Normalization.

The U-Net structure is a classic convolutional neural network architecture designed for image segmentation (Ronneberger et al., 2015). Its critical feature is the symmetrical convolution and upsampling design, which allows the network to capture image features at different scales, enabling the generation of fine-grained segmentation maps. The U-Net structure is used to predict semantic segmentation maps of clothing worn on the human body from the given input data, which includes clothing, human pose, and human parsing images. By extracting multi-level features through convolutional layers and restoring high-resolution segmentation through upsampling layers, the model can accurately predict the positions of different body parts and generate high-quality segmentation images.

While this thesis does not focus primarily on this component, the Segmentation Generator follows VITON-HD's training process. According to the literature, the final total loss L_S used in the training comprises two parts:

$$L_S = L_{cGAN} + \lambda_{CE} L_{CE}$$

In this context, L_{cGAN} represents the conditional generative adversarial loss, ensuring that the generated segmentation images appear realistic. L_{CE} is the pixel-level cross-entropy loss, and the hyperparameter L_{CE} is used to balance the relative importance of these two loss terms. Compared to other techniques, U-Net's advantage lies in its symmetrical upsampling and downsampling structure, which enables it to combine both local and global image features, making it suitable for processing high-resolution images. In comparison to architectures like Fully Convolutional Networks (FCN), U-Net excels at preserving image details, particularly in the segmentation of small objects or boundaries. While not as deep as networks like ResNet (He et al., 2016), U-Net's skip connections ensure that important image features are not lost during the upsampling process.

Dropout is a regularization technique used to prevent overfitting in neural networks. By randomly dropping the outputs of certain neurons, the model is forced to avoid relying on specific neurons during the training process, thereby enhancing its generalization ability. This allows the model to better adapt to new data, rather than overfitting to specific features of the training data, effectively preventing overfitting, especially in high-level feature extraction. The expression for applying Dropout is as follows:

$$Dropout(h) = \begin{cases} \dfrac{h}{1-p}, & if\ active \\ 0, & if\ dropped \end{cases}$$

Where **p** represents the dropout rate.

Batch Normalization is a technique used in deep neural networks to accelerate training and enhance stability. It is applied to each convolutional layer (such as in the layers from conv1 to conv9). This approach effectively stabilizes the input distribution of each layer, preventing issues such as vanishing or exploding gradients in the network, thereby improving the network's stability and convergence speed.

The **GMM (Geometric Matching Module)** is primarily used to geometrically align the target clothing with the human pose, ensuring that the clothing naturally fits the person. GMM mainly utilizes TPS (Thin-Plate Spline) transformation techniques to

achieve natural clothing deformation that corresponds to the human pose (Wang et al., 2018).

Thin-Plate Spline (TPS) is a commonly used deformation model, originally applied in image registration and deformation tasks. Its goal is to smoothly and naturally deform an image or object based on a set of control points. The deformation process can be represented by the following formula:

$$W(x) = A + \sum_{i=1}^{N} w_i U(||x - p_i||)$$

Where $W(x)$ represents the deformed point, A is the affine transformation parameter, w_i denotes the weight of the control points, p_i is the position of the control points, and $U(r)$ is the radial basis function, defined as:

$$U(r) = r^2 \log r$$

TPS utilizes these radial basis functions to ensure smooth deformations around the control points while maintaining natural deformation in areas farther away. The advantage of this model is its ability to adaptively adjust the deformation of clothing according to the human pose, achieving a natural fit between the clothing and the body.

In the VITON-HD literature, the GMM optimizes the deformation parameters θ through a loss function, which consists of two parts:

$$L_{warp} = ||I_c - W(c, \theta)||_{L1} + \lambda_{const} L_{const}$$

The first term, $||I_c - W(c, \theta)||_{L1}$, represents the $L1\ loss$, which measures the pixel-wise difference between the deformed clothing $W(c, \theta)$ and the original clothing I_c. This ensures that the deformed clothing reasonably matches the human form.
The second term, L_{const}, is a second-order difference constraint, used to reduce noticeable distortions during the clothing deformation process. It calculates the difference in distances between the deformation control points to ensure smoothness in the transformation.

Based on the above, TPS enables smooth clothing deformations and can handle local deformation issues, which is crucial for ensuring that the clothing naturally fits the

human body. Additionally, this technique can independently adapt to different human poses, improving the flexibility and applicability of virtual try-on models.

The **ALIAS (Alignment-Aware Segment) Generator** (Choi et al., 2021) is capable of generating high-quality and precise virtual try-on images, primarily due to three key components: (1) **ALIASNorm**, (2) **ALIASResBlock**, and (3) **Spectral Normalization**.

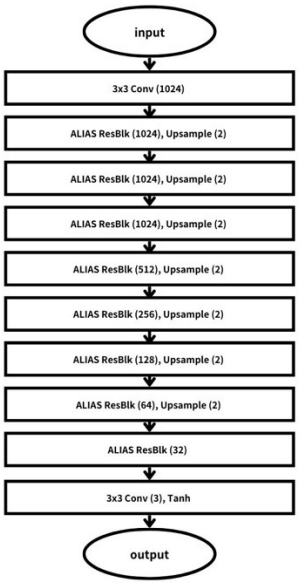

Figure 2: Process of ALIAS Generator

ALIASNorm (Alignment-Aware Segment Normalization) is a custom conditional normalization technique designed specifically for virtual try-on image synthesis. It adjusts the activations in each layer of the generator using semantic segmentation maps and human pose information, thereby enhancing the realism of the composite images. While traditional normalization techniques such as Batch Normalization or Instance Normalization can standardize activations in neural networks, they fall short when handling the complexity of human poses and clothing deformations, lacking the dynamic adjustment required. ALIASNorm improves upon traditional normalization through the following two steps:

(1) The activations are normalized, but unlike traditional methods, they are dynamically adjusted based on the segmentation map and misalignment masks. This allows differentiation between the clothing and various human body regions, ensuring precision in localized adjustments.

(2) Convolutional layers extract features from the segmentation map, generating two sets of affine parameters, γ and β. These parameters are adjusted according to the human pose and misalignment areas, applying an affine transformation to the activations. This transformation is passed to the next layer, enabling fine-tuned control of the generation process.

ALIASResBlock (ALIAS Residual Block) is responsible for handling multi-scale features during the image generation process. It inherits the design principles of Residual Networks, using skip connections to retain the original features, which reduces the risk of vanishing gradients and improves network stability. Combined with the previously mentioned ALIASNorm, ALIASResBlock allows dynamic adjustment during convolution operations based on semantic segmentation maps and human pose information. The main functions of ALIASResBlock include:

(1) **Feature Extraction**: Extracts both local and global features through two layers of convolution, applying conditional normalization based on semantic segmentation maps and misalignment masks.
(2) **Skip Connections**: Utilizes skip connections to retain features from the previous layer, reducing feature loss and enhancing the detail representation in the synthesized images.

Compared to traditional residual blocks, the ALIASResBlock performs better in handling the semantic relationships between clothing and the human body by incorporating semantic information and misalignment masks. It enables localized adjustments in areas with misalignment, making its application in virtual try-on models more flexible. This effectively addresses the semantic misalignment issues that can occur during image generation.

Spectral Normalization is a regularization technique used in Generative Adversarial Networks (GANs) to constrain the spectral norm of the weight matrices in each layer of the neural network, preventing numerical instability during the training process (Miyato et al., 2018). In the ALIAS Generator, Spectral Normalization is applied to every convolutional operation in both the generator and discriminator, limiting the weight norms and preventing issues such as numerical overflow, gradient explosion, and vanishing gradients. Specifically, in convolutional layers, the weight matrix W_{conv} is constrained by Spectral Normalization during each update:

$$W'_{\text{conv}} = \frac{W_{\text{conv}}}{\sigma_{\max}(W_{\text{conv}})}$$

This ensures that during GANs training, the generator's output does not experience overflow or vanishing gradients due to excessively large or small weights, thereby maintaining stability in adversarial training and improving the quality of generated images.

The three techniques mentioned above enable the ALIAS Generator to produce high-quality virtual try-on images while effectively handling the complex details between clothing and the human body with greater flexibility.

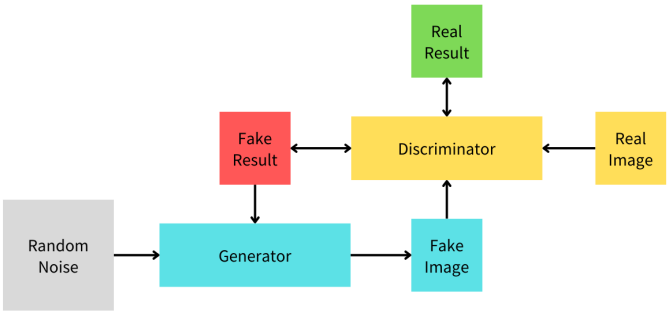

Figure 3: Process of GAN

The primary focus of this thesis is on adversarial training with the ALIAS Generator, ultimately generating virtual try-on images. The discriminator adopts a **Multi-Scale Discriminator**, which, compared to a single-scale discriminator that can only evaluate image realism at a fixed resolution, struggles with capturing details in high-resolution images (Karnewar and Wang, 2020). The design concept of the multi-scale discriminator is to comprehensively evaluate the realism of generated images by extracting features at different scales. This structure typically employs multiple discriminators, each taking images at different resolutions as input. Such a design allows the discriminator to handle global features in the image while also finely assessing details.

In this study, two discriminators are used (num_D = 2), with each one processing images at different resolutions. To capture various levels of detail using different convolutional layers and normalization techniques, the goal is to improve the quality of the generated images. In the multi-scale processing, Average Pooling is applied, which scales down the input images, ensuring that images at different scales can be processed by the same discriminator. After each pooling operation, the image size is halved, allowing the discriminator to capture image details at multiple resolutions.

In addition to this technique, the Multi-Scale Discriminator incorporates several methods to stabilize the training process, including: (1) Spectral Normalization (2) Instance Normalization (3) Dropout。

Instance Normalization (IN) is a normalization technique applied to each individual sample, which helps improve the stability and quality of the generated results, particularly when there are significant differences in data distribution across each image (Ulyanov et al., 2016). In virtual try-on models, where the images generated by the generator come from various samples with potentially diverse feature distributions, Instance Normalization is especially suitable. Compared to Batch Normalization, IN is better suited for single-image processing tasks, as it does not rely on the data distribution of a mini-batch but normalizes each channel of every sample individually.

3.1.3 Stability Training Techniques

Gradient Penalty is introduced in Generative Adversarial Network (GAN) training to address the instability that often arises from the adversarial relationship between the discriminator and the generator (Gulrajani et al., 2017). Specifically, when the discriminator becomes too powerful, it can lead to issues like gradient explosion or mode collapse. To mitigate these issues, Gradient Penalty is applied as a regularization technique to the discriminator, ensuring that its gradients remain within a reasonable range during adversarial training. It enforces a constraint on the gradient of the discriminator with respect to interpolated samples, aiming to keep the gradients close to 1, thereby stabilizing the adversarial training process. This method is particularly effective for Wasserstein GANs (WGAN-GP). In this study, the gradient_penalty function implements this technique. For each discriminator, it computes the gradient for interpolated samples between real and generated images, ensuring that the gradient is constrained to prevent the discriminator from becoming overly dominant or causing gradient explosion. The steps involved are as follows:

(1) Generating Interpolated Samples:

$$interpolates = \alpha \cdot real_{data} + (1 - \alpha) \cdot fake_{data}$$

Where **α** is a random weight, and the real data and generated data are linearly interpolated.

(2) Calculating the Gradient:

$$\nabla_{x'} D(x')$$

(3) Computing the Gradient Penalty Term:

$$gradient\_penalty = (\left|\left|\nabla_{x'} D(x')\right|\right|_2 - 1)^2$$

This gradient penalty ensures that the learning process of the discriminator remains stable, preventing issues like mode collapse from occurring.

Mixed Precision Training was initially introduced by hardware manufacturers and research institutions, and it is now widely supported by modern deep learning frameworks such as TensorFlow and PyTorch (Micikevicius et al., 2017). For example, PyTorch's torch.cuda.amp provides automatic support for mixed precision, allowing the selection of FP16 or FP32 computations based on the specific situation. NVIDIA's Automatic Mixed Precision (AMP) framework was one of the earliest mainstream tools, combining GPU hardware optimizations to achieve more efficient training.

In this thesis, mixed precision training is applied to efficiently manage limited GPU RAM consumption while maintaining training efficiency. It achieves this by converting part of the computation from the traditional 32-bit floating point (float32) precision to 16-bit floating point (float16), significantly reducing memory usage per training iteration. The precision reduction is handled by PyTorch's torch.cuda.amp automatic mixed precision technology, which selectively converts some computations to float16 based on computation requirements, while keeping other critical operations in float32 to ensure numerical accuracy and stability.

However, using lower precision can lead to floating point overflow or computation errors. To counter this, GradScaler is utilized, which dynamically adjusts the gradient scaling during backpropagation, avoiding precision loss or overflow due to excessively large or small numerical ranges. GradScaler automatically adjusts the scaling factor based on the current training state, allowing mixed precision training to balance speed and stability, enhancing overall training efficiency while ensuring the numerical stability of gradient backpropagation.

This technique not only significantly reduces GPU memory usage but also improves computational performance, making it highly valuable, especially in scenarios involving large model parameters and large-scale data training.

In mathematical terms, reducing the floating-point precision from 32-bit (float32) to 16-bit (float16) can be represented:

$$x_{float32} = m_{float32} \times 2^{e_{float32}} \rightarrow x_{float16} = m_{float16} \times 2^{e_{float16}}$$

The reduced precision of the mantissa and exponent decreases the range and precision of numerical representation, thereby reducing GPU memory usage.

Gradient Accumulation in deep learning is used when the model is too large or GPU memory resources are limited, allowing only small batch sizes during each training step (Andersson et al., 2022). However, very small batch sizes can increase the variance in gradient estimates, negatively affecting the model's stability and convergence speed. Gradient Accumulation works by accumulating gradients over several forward and backward passes across smaller batches, and once enough gradients have been accumulated, the model weights are updated as if a larger batch size had been used. By accumulating gradients, it is possible to simulate larger batch training even with limited hardware resources, improving both the quality and stability of the generative model.

3.2 Evaluation Methods

SSIM (Structural Similarity Index) is a metric used to measure the similarity between two images, specifically designed to simulate how the human visual system perceives images. SSIM evaluates the quality of images not only through pixel differences but also by considering similarities in aspects such as brightness, contrast, and structure. Compared to simple pixel difference metrics like MSE (Mean Squared Error), structural information in SSIM better aligns with human visual perception (Wang et al., 2004).

The mathematical formula for SSIM is as follows:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Where:

- μ_x and μ_y are the mean values of images x and y.
- σ_x^2 and σ_y^2 are the variances of x and y, respectively.
- σ_{xy} is the covariance between x and y.
- C_1 and C_2 are constants used to avoid division by zero.

In this study, SSIM is used to measure the similarity between generated images (such as images of a person wearing virtual clothing) and real images. The SSIM value ranges [-1, 1], where 1 indicates that the two images are identical. When evaluating the generated images, a high SSIM value signifies a higher structural similarity between the generated and real images, indicating better image quality.

LPIPS (Learned Perceptual Image Patch Similarity) evaluates the perceptual similarity between two images using a deep neural network. LPIPS relies on a pre-trained Convolutional Neural Network (CNN) to capture high-level feature representations of the images, rather than merely focusing on pixel differences (Zhang et al., 2018).

The mathematical formula for LPIPS is based on the Euclidean distance between activation features from the convolutional neural network. Assuming the feature maps from the CNN are $\phi(x)$ and $\phi(y)$, LPIPS can be expressed as:

$$LPIPS(x, y) = \sum_{l} \frac{1}{H_l W_l} \sum_{h,w} \left|\left| w_1 \cdot (\phi_l(x)_{h,w} - \phi_l(y)_{h,w}) \right|\right|_2^2$$

Where:

- l represents a specific layer in the network.
- $\phi(x)$ and $\phi(y)$ are the activation features of images x and y at layer l.
- w_1 are learned weights used to weight the features from different layers.
- H_l and W_l are the height and width of the feature maps at layer l.

In this study, LPIPS is used to assess the perceptual quality of images by comparing the perceptual features of generated images with real images to measure their similarity. A lower LPIPS value indicates that the perceptual quality of the generated image is closer to that of the real image. In virtual try-on models, LPIPS helps evaluate whether the generated try-on images possess sufficient realism and retain detailed features.

FID (Fréchet Inception Distance) is a commonly used metric for evaluating the quality of images generated by GAN models. It works by inputting both real and generated images into a pre-trained Inception model, then comparing the high-level feature distributions of these two sets of images to assess the realism of the generated images (Heusel et al., 2017).

The mathematical formula for calculating **FID** is based on the Fréchet distance between the high-level feature distributions of real and generated images (extracted by the Inception model). The FID formula is:

$$FID = \left|\left| \mu_r - \mu_g \right|\right|_2^2 + \mathrm{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

Where:

- μ_r and μ_g are the means of the features for real and generated images, respectively.
- Σ_r and Σ_g are the covariance matrices of the features for real and generated images.

In this study, **FID** evaluates the overall distribution difference between real and generated images, assessing not only image quality but also the generator's performance in terms of sample diversity. A lower FID value indicates that the quality and distribution of the generated images are closer to those of the real images. In virtual try-on models, a low FID value signifies that the generated try-on images not only possess high quality but also match the diversity of real images.

The combination of these three metrics—SSIM, LPIPS, and FID—provides a comprehensive quantitative analysis for this study, allowing for a full evaluation of the generated images' quality, from structure and perceptual quality to overall distribution diversity.

4. Experiment

4.1 Environment Configuration

The experiment was conducted on the Google Colab platform, primarily using the T4 GPU, which has 15GB of GPU RAM. Due to the relatively small GPU memory provided by the Colab platform, careful management of memory usage was necessary during the model training process. Additionally, Google Drive was mounted as storage to access model training data and results, ensuring the availability and efficiency of resources.

In terms of software, this study used PyTorch as the primary deep learning framework, running on the default Ubuntu platform provided by Colab, with Python 3.8 as the programming language. The specific software environment configuration is as follows:

- Operating System: Ubuntu 18.04 LTS (Colab default)
- Python Version: 3.8
- Deep Learning Framework: PyTorch 1.12.1
- Other dependencies: OpenCV (image processing), NumPy, Pillow (for handling image formats), torchvision (providing image transformations and pre-trained models), matplotlib (data visualization)

For the dataset, this study used the dataset provided by VITON-HD, which includes 13,679 sets of front-facing female images, upper-body clothing images, and auxiliary images (such as human parsing maps and pose estimation data). These paired data were divided into training and testing sets, with 11,647 and 2,032 sets, respectively. All the images are of 1024×768 resolution, with a primary focus on overcoming the challenges of high-resolution virtual try-on. The dataset also includes train_pairs.txt and test_pairs.txt files for pairing person images with clothing images, with the clothing images randomly shuffled for synthesized image generation.

4.2 Model Training

In this experiment, we first used VITONDataset to process and load the training and testing data. All input images were scaled to a resolution of 1024×768 pixels to ensure consistent image dimensions, which facilitates model training. Then, we used transforms.Normalize to normalize the image data to a range of **[-1, 1]**, which not only accelerates model convergence but also improves stability during the training process.

The main function of the SegGenerator module is to generate a detailed human parsing map, which accurately segments different body parts such as the upper body, neck, and arms based on the input portrait and pose data. The preprocessed portrait and pose data are used as inputs, and the generated parsing map is utilized in the subsequent clothing try-on process, helping the model better identify the position of the clothing. After the SegGenerator generates the human parsing map, the GMM (Geometric Matching Module) is responsible for geometric matching, deforming the clothing image to fit the body structure of the target portrait. Specifically, the GMM module takes the clothing image and the target portrait's pose data as inputs and uses Thin Plate Spline (TPS) transformation to generate the deformed clothing image, aligning it with the pose of the portrait.

In the final generation process, we used the ALIAS Generator to produce the final portrait image, ensuring that the synthesized clothing appears naturally fitted on the person. This process is based on the GAN (Generative Adversarial Network) architecture, and this study adopts a Multiscale Discriminator to evaluate the realism of the generated images at different scales. During training, the ALIAS Generator receives preprocessed pose data, the clothing image deformed by the GMM module, and the human parsing map as inputs, ultimately generating an image of the person wearing the new clothing.

In the experiment, the learning rate for the ALIAS Generator was set to 0.0001, and the training was conducted with a batch size of 1. Due to GPU resource limitations, Gradient Accumulation was used to simulate a batch size of 4, which reduces memory load while achieving the stability of training with a batch size of 4. The loss function for the ALIAS Generator includes L1 Loss, Perceptual Loss, and Adversarial Loss. The L1 Loss measures the pixel-level difference between the generated image and the real image, while the Perceptual Loss leverages the VGG feature network to extract high-level features for evaluating the visual realism of the image, aiming to generate more natural composite images.

The generator loss function formula is as follows:：

$$Loss_g = Loss_{adversarial} + 10 \cdot Loss_{L1} + 10 \cdot Loss_{perceptual}$$

The Multiscale Discriminator is responsible for evaluating the generated images at multiple scales and comparing them with the real images. To further enhance the stability of the training process and avoid issues such as vanishing or exploding gradients, we introduced Gradient Penalty into the adversarial loss. Gradient Penalty is a gradient-based regularization technique that applies a penalty to the gradients of the input image, ensuring that the magnitude of the discriminator's gradients remains within a reasonable range.

The learning rate for this module was set to 0.0004, primarily using the Adversarial Loss to guide the generator in producing more realistic images, ensuring that image quality remains consistent across different scales.

The formula for the discriminator's loss function is as follows:

$$Loss_d = Loss_{adversarial} + GP$$

$$GP = \lambda \cdot \mathrm{E}_{x'}((\left|\left|\nabla_{x'} D(x')\right|\right|_2 - 1)^2)$$

In the formula, λ is a hyperparameter used to control the weight of the gradient penalty (typically set to 10), and x' represents the interpolated samples from real and generated data, with $D(x')$ denoting the discriminator's output.

Due to limited GPU RAM, we adopted Gradient Accumulation. However, to achieve more effective training results, we provided additional features as input to the discriminator, which increased the likelihood of the training process halting due to

exceeding memory limits. To further reduce memory usage and improve training efficiency, Mixed Precision training was implemented in both the generator and discriminator training processes. This was done using torch.cuda.amp.autocast() to handle precision conversion during the forward and backward propagation steps, reducing resource consumption for floating-point computations. Additionally, GradScaler was used to ensure numerical stability during backpropagation. This approach allowed us to perform more efficient training with less GPU memory without sacrificing model performance.

5. Results

In this section, we present the generated results from the ALIAS Generator and Multiscale Discriminator during the adversarial training process and discuss the challenges encountered during training, as well as their impact on the outcomes. The main challenges include GPU resource limitations, instability in adversarial training, pairing issues in data preprocessing, and balancing problems between the generator and discriminator. These issues ultimately had a negative impact on the quality of the generated images, specifically manifesting as blurriness and insufficient detail retention.

After 26,000 iterations, significant improvements were observed compared to 15,000 iterations. The overall structure of the generated images remained relatively consistent, but blurriness persisted in the details of the clothing. There is still ample room for improvement, but due to time constraints, it was not possible to perform a sufficient number of training iterations. When comparing the generated images to real images, the lack of clarity in clothing details is evident, especially in wrinkles and textures. There are also instances where parts of the text on the clothing either disappear or are not fully generated, indicating areas for further improvement and development.

Here are the quantitative and Qualitative analysis results:

- Image Quality: When comparing generated images with real images, we found that the clothing areas in the generated images were blurry, especially in terms of detail handling (e.g., wrinkles and edges of the clothing). Some poses still failed to adequately present fine details, or certain body parts were omitted.
- Quantitative Metrics: In this study, FID, SSIM and LPIPS were used to evaluate the quality of the generated images. The results show that FID and

LPIPS decrease as the number of iterations increases, which indicates an improvement in the perceived quality of the images; however, there is still a gap in the image generation performance of the model compared to the original VITON-HD model. This is especially evident on Image_15000 and MoreFeatures_15000.

- Qualitative Observations: The image quality improves after 26,000 iterations, showing better edge processing and colour consistency, but still suffers from blurred texture details.The introduction of features in the MoreFeatures model improves some image details, but the progress is limited compared to the original model.
- Loss Value Changes: During the training process, we recorded the loss values of the model at different steps. The results show that during the early iterations, the generator's L1 loss and the discriminator's adversarial loss dropped rapidly, but later on, they gradually stabilized, with little subsequent progress, indicating that the balance between the generator and discriminator was not fully achieved.

During the experiment, we encountered several challenges that affected the quality of the results. These challenges primarily arose from the following aspects:

- Resource Limitations: Due to the limited GPU RAM provided by Google Colab, we were unable to perform large-batch training. To address this issue, we utilized Gradient Accumulation, simulating a batch size of 4 from 1, allowing for larger gradient updates without increasing memory usage. Additionally, we employed Mixed Precision training by reducing the precision of floating-point calculations (from float32 to float16), further reducing GPU RAM usage, and used GradScaler to maintain numerical stability during training. While the combination of these techniques enabled more efficient and stable training in a resource-constrained environment, the results still did not achieve the desired generation quality.
- Training Instability: We attempted to improve the stability of adversarial training by adjusting the learning rates of the generator and discriminator and modifying the weights of the loss function. Specifically, we increased the weights of feature matching loss and perceptual loss. We also incorporated additional features for comparison instead of merely computing the loss between the synthetic and original images. Although these adjustments improved the detail in the generated images to some extent, the results were still unsatisfactory.

- Balance Between Discriminator and Generator: We discovered that the original discriminator lacked complexity and was unable to effectively distinguish between real and generated images. In response, we increased the number of layers and complexity of the discriminator, but after 30,000 iterations, the improvements were limited. This indicates that the balance between the generator and discriminator had not been fully achieved.
- Incorrect Data Preprocessing: During the training process, we realized that errors in paired data led to blurry clothing regions in the generated images. Generally, early training is conducted using the original clothing worn by the person, which helps the generator learn basic synthesis techniques. After the model reaches a certain level of proficiency, the ultimate goal is to synthesize different clothing onto the person's image, which trains the generator to produce high-quality try-on effects in various scenarios. If different clothes are directly synthesized onto the person's image, it would require a significant amount of time and resources for comparison. Therefore, the typical approach is to train with the original clothing first and then switch to paired data for generalization training and testing after reaching a certain level. After correcting this error, the quality of the generated images improved, but further optimization requires longer training time.

Despite taking various measures to address the challenges during the training process, the final results were still affected by certain limitations. Specifically:

- Suboptimal ALIAS Generator results: Due to the limited ability of the discriminator to differentiate real and generated images across multiple scales, the generated images lacked detailed refinement.
- Training instability: Although we increased the discriminator's complexity and used Gradient Accumulation to stabilize the training process, insufficient resources and adversarial training instability prevented the results from reaching the desired state.
- GPU resource limitations: Constrained by GPU RAM, we were unable to conduct long-duration and large-batch training, which restricted the performance potential of both the generator and the discriminator.

Future directions for this research to further improve model stability and image generation quality include:

- Introducing Diffusion-StyleGAN2: Exploring the concept of Diffusion-StyleGAN2 in the training phase of the ALIAS Generator to improve training stability and image generation quality (Wang et al., 2023).
- Optimizing data augmentation techniques: Implementing data augmentation techniques (such as rotation, scaling, and color shifting) to increase data diversity, improving the model's generalization ability and helping the generator adapt to clothing detail variations.
- Adjusting the discriminator architecture: Experimenting with discriminators like PatchGAN (Isola et al., 2017) to enhance the discriminator's ability to distinguish fine details, thereby improving the quality of generated image details.
- Using more efficient loss functions: Implementing more efficient loss functions like Style Loss (Gatys et al., 2016) and Hinge Loss (Vapnik and Guyon, 1992) to help the model learn details and style features more effectively during the generation process, improving the stability of adversarial training.
- Gradually fine-tuning the model architecture: Using a Progressive GAN (Karras et al., 2017) approach, starting with generating images at a low resolution and gradually increasing to higher resolutions. This can help the model learn both global features and details more stably, improving the final image quality.
- Enhancing resource utilization: Exploring more efficient model architectures to reduce GPU resource consumption and conducting longer training sessions to optimize adversarial training between the generator and the discriminator.
- Upgrading hardware resources: Increasing GPU memory or using multi-GPU training and considering cloud-based high-performance servers for distributed training to reduce resource limitations and improve training efficiency and result quality.

6. Analysis

6.1 Quantitative Analysis

In this section, the paper provides a detailed analysis of the FID (Fréchet Inception Distance), LPIPS (Learned Perceptual Image Patch Similarity), and SSIM (Structural Similarity Index Measure) metrics, which are used to evaluate the quality and visual similarity of the generated images. The performance of different models is compared

based on different training iterations (15,000 and 26,000 iterations) and training configurations (Image and MoreFeatures). The table below presents the specific results of the experiments.

	FID	LPIPS	SSIM
Original	10.3087	0.1164	0.8579
Image_15000	33.1703	0.1774	0.8565
Image_26000	27.3046	0.1630	0.8601
MoreFeatueres_15000	30.1401	0.1794	0.8532
MoreFeatueres _26000	26.9683	0.1783	0.8557

Table 1: Results of Quantitative Analysis

Table Explanation

- Original: The test results of the original VITON-HD model, which was trained for 200,000 iterations, serve as the baseline for comparison.
- Image_15000 and MoreFeatueres_15000: These models were tested after 15,000 iterations. Image_15000 was trained using only the generated and real images in the multiscale discriminator for adversarial training, while MoreFeatueres_15000 added more features from the ALIAS generator as input for the multiscale discriminator.
- Image_26000 and MoreFeatueres_26000: These models were tested after 26,000 iterations, aiming to further assess the model's performance after more iterations.

6.1.1 FID Analysis

The **FID** metric measures the distance between the generated images and real images, with lower values indicating better image quality. From the table, we can see that the **VITON-HD** model has an FID of 10.3087, which is expected due to the original VITON-HD model being well-trained after 200,000 iterations.

In comparison, the FID values of the **Image** and **MoreFeatures** models are significantly higher, indicating that with only 15,000 and 26,000 iterations, these models have not yet generated images that are sufficiently close to the real ones. **Image_15000** has an FID of 33.1703, showing a large gap between the generated and real images. As the iterations increased to 26,000, the FID dropped to 27.3046, indicating that the model's image quality improves as training progresses.

Similarly, the FID values for the **MoreFeatures** series also decreased as the number of iterations increased, from 30.1401 to 26.9683. This suggests that introducing more features into the discriminator during adversarial training can improve image quality to some extent, though there remains a large gap compared to the original model.

6.1.2 LPIPS Analysis

The **LPIPS** metric measures the perceptual difference between the generated and real images, with lower values indicating higher perceptual similarity. From the table, the **VITON-HD** model has an LPIPS value of 0.1164, reflecting a high perceptual similarity between the generated and real images.

However, the LPIPS values for the **Image** and **MoreFeatures** models are relatively higher, indicating greater perceptual differences. For example, **Image_15000** has an LPIPS value of 0.1774, which decreases to 0.1630 as training progresses to 26,000 iterations, showing an improvement in perceptual similarity. Similarly, **MoreFeatures_15000** and **MoreFeatures_26000** have LPIPS values of 0.1794 and 0.1783, respectively, indicating that adding more features only slightly improves perceptual quality.

6.1.3 SSIM Analysis

The **SSIM** metric measures the structural similarity between generated and real images, with higher values indicating greater structural similarity. From the data, the **VITON-HD** model has an SSIM value of 0.8579, consistent with the high-quality generated images resulting from prolonged training.

In the case of short-term training, the SSIM values for the **Image** and **MoreFeatures** models show minimal differences. **Image_15000** has an SSIM of 0.8565, which is close to that of the **VITON-HD** model. As the training iterations increase to 26,000, the value slightly rises to 0.8601, indicating a small improvement in structural similarity. For the **MoreFeatures** series, the SSIM values after 15,000 and 26,000 iterations are 0.8532 and 0.8557, respectively, indicating that adding more features did not significantly enhance the structural similarity.

6.2 Qualitative Analysis

Figure 4: Comparison of generated images

6.2.1 Visual Quality and Detail Analysis

As the number of iterations increases, the quality and detail of the image changes significantly:

- **Original:** The VITON-HD image has the best clarity and texture detail, with accurate alignment of clothing to the body and natural colours.
- **Image_15000:** At 15,000 iterations, the image quality is poorer, showing significant blurring and edge distortion. The colour performance is also unstable and there are serious problems with the alignment of the clothes to the body.
- **Image_26000:** As the number of iterations increases to 26,000, the image sharpness improves, the alignment problem improves slightly, and the colour stability improves, but the detailed texture performance is still not precise enough.
- **MoreFeatures_15000:** After 15,000 iterations of adding features, the image quality is relatively better, but the edges of the clothing are still blurred, the texture is not fine enough, and the colour is slightly improved.
- **MoreFeatures_26000:** After 26,000 iterations, the image quality has been further improved, with more natural alignment of the clothing to the human body, stable colours, and more details, but still not as good as the original image.

6.2.2 Effect of Feature Introduction

Comparison of the images with the added features with the image-only generation model resulted in the following observations:

- **Image model:** The image-only models (Image_15000 and Image_26000) show significant blurring and colour distortion in the early iterations. As the iterations increase, the quality of the images improves slightly, but the texture details and edge definition are still far from the original.
- **MoreFeatures model:** The models with added features (MoreFeatures_15000 and MoreFeatures_26000) show better texture and edge processing. The introduction of features results in richer details in the generated images, and the alignment problem is reduced, especially after 26,000 iterations. However, there are still minor blurring and colour deviations, indicating that the model performance has improved but there is still room for improvement.

6.3 Combined Analysis and Trends

Based on the quantitative and qualitative analyses described above, the original VITON-HD model trained over 200,000 iterations demonstrated superior image generation quality in terms of FID, LPIPS and SSIM metrics, as well as in terms of image clarity, colour stability and texture detail. With longer training, the original model is more accurate in terms of garment-to-human alignment and texture presentation.

In contrast, the Image and MoreFeatures models trained with only 15,000 and 26,000 iterations show a significant performance gap. In particular, the FID and LPIPS metrics show significant perceptual and structural differences between the generated images and the real images. For example, in the qualitative analysis, Image_15000 and Image_26000 exhibit fuzzy textures and unstable colours, which are not up to the performance level of the original VITON-HD model despite the improvement with increasing iterations.

The MoreFeatures model shows some degree of improvement, especially in MoreFeatures_26000, showing better suit alignment and improved detail handling, as observed in the qualitative analysis. However, these improvements, although reflected in the FID and LPIPS metrics, are still not enough to match the original model, suggesting that simply adding more feature inputs does not significantly improve image quality. Furthermore, even after 26,000 iterations, the MoreFeatures model still suffers from problems such as slight blurring and colour inconsistencies.

Due to time and resource constraints, our models were trained for only 15,000 and 26,000 iterations, which is much lower than the 200,000 iterations of the original VITON-HD model. Although quantitative metrics and qualitative observations suggest that more iterations can further improve image quality, especially in terms of texture clarity and alignment, these improvements are still incremental. We expect that as the number of training sessions increases, the accuracy of the model, especially in terms of its ability to produce high quality images that are perceptually closer to the real thing, will improve significantly, closing the gap seen in the FID, LPIPS, and SSIM metrics.

7. Conclusions

This research successfully reproduced the ALIAS Generator technology and supplemented the missing discriminator part in the original model by adopting the Multiscale Discriminator, fully realizing the adversarial training process. During this work, we faced significant resource limitations, but by introducing Mixed Precision Training and Gradient Accumulation, we effectively overcame the issue of insufficient GPU memory, allowing for large-scale training. Specifically, Gradient Accumulation allowed us to simulate larger batch sizes, helping stabilize the training process while avoiding excessive GPU RAM usage. This is especially important in GAN training since small batch sizes often lead to unstable gradient estimates, negatively affecting the model's overall performance.

By using Mixed Precision Training, we significantly reduced GPU memory consumption per training step, allowing us to further enhance training efficiency in memory-constrained environments. Additionally, thanks to the use of GradScaler, numerical stability during backpropagation was ensured, and no issues with numerical overflow or gradient explosions were observed.

However, despite completing full adversarial training, the final generated images still exhibited limitations in terms of detail processing and matching accuracy. This was particularly evident in the details of the clothing, such as wrinkles and textures, especially at the edges where the clothing meets the body, where the generated images failed to preserve all the details, resulting in some blurriness. This indicates that there is still considerable room for optimization in balancing the generator and discriminator during adversarial training. The issues we encountered, including resource limitations, instability in adversarial training, and pairing errors during data preprocessing, all contributed to the quality limitations of the final results.

In addition, although we attempted to increase the complexity of the discriminator to better distinguish between real and generated images, the model did not achieve the expected results due to insufficient training iterations. The original VITON-HD model lacked a complete discriminator, so we designed a Multiscale Discriminator to evaluate the realism of images at different resolutions. However, due to limitations in GPU resources and time, we were unable to conduct enough iterations, leading to suboptimal final model performance.

The choice of loss function also played a crucial role in our adversarial training process. In addition to the standard adversarial loss, we introduced L1 Loss and Perceptual Loss to enhance the visual realism of the images. Perceptual Loss, in particular, measures image quality using high-level features from the VGG feature network, which helps generate more natural-looking images. Nonetheless, progress in image quality was still limited due to the balance issues between the ALIAS Generator and Multiscale Discriminator at different scales.

During the training process, we experimented with various techniques to improve the stability of adversarial training and the quality of the generated images. However, the final images still had some limitations, especially in terms of detail retention and fitting accuracy. The main challenges arose from GPU resource constraints, which not only affected our training efficiency but also limited the model's overall performance. Additionally, the instability of adversarial training partially stemmed from the collaborative training issues between the generator and discriminator. As the number of training iterations increased, the model's loss values tended to stabilize, but the desired image generation results were still not fully achieved.

Another challenge arose from pairing errors in the early data preprocessing stages. Due to issues with data pairing, our generator was unable to accurately learn the detail matching between clothing and the human body during the initial training phase, which negatively impacted subsequent training. Although we made adjustments to this issue later, additional training time is still needed to further optimize the generated results.

Additionally, we attempted to improve results by increasing the complexity of the discriminator, but we found that due to the limited training iterations, the improvements were marginal. We believe that with more resources and time, the balance between the generator and discriminator will reach a more optimal state, and the quality of the generated images will improve further.

To address the issues encountered in this study, we plan to explore the following directions in the future to further improve model stability and generation quality:

- Introducing Diffusion-StyleGAN2
- Data augmentation techniques
- Optimizing the discriminator structure
- Further refining the loss function
- Enhancing resource utilization

In conclusion, this research demonstrated how adversarial training between the ALIAS Generator and Multiscale Discriminator can improve virtual try-on models in a resource-constrained environment. Although we overcame several technical challenges during the experiments, the final image quality did not reach the desired level due to limitations in time and resources. However, this study provides valuable insights for future optimization and highlights specific areas for improvement, such as the introduction of Diffusion-StyleGAN2, and optimizing both the discriminator and loss functions. We believe that these further studies and technical improvements will significantly enhance the application of virtual try-on models. ◦

8. Reflection and Learning

During this research, I encountered numerous technical challenges, particularly related to hardware resource limitations and the instability of adversarial training. This experience provided me with a deeper and more comprehensive understanding of the processes, details, and potential issues involved in training Generative Adversarial Networks (GANs), especially how to improve training efficiency in a resource-constrained environment. By continuously exploring new techniques, I applied Gradient Accumulation, which allowed me to simulate larger batch sizes by accumulating gradients over multiple smaller batches, effectively solving the memory insufficiency issue. Additionally, I employed Mixed Precision Training, which further reduced GPU RAM usage, and used GradScaler to ensure numerical stability. These strategies significantly enhanced both the efficiency and stability of the training process.

Throughout the research process, my mindset improved significantly. Faced with frequent technical issues as well as time and resource limitations, I initially felt considerable pressure, especially when multiple attempts failed to yield ideal results. However, as I continued to overcome these challenges, I learned to remain calm in the face of difficulties, systematically analyze problems, and find solutions. I gradually

adapted to this high-pressure research environment and developed a more positive and composed attitude when confronting issues. This experience greatly enhanced my ability to handle stress, enabling me to focus on problem-solving.

In addition, this research has made me more familiar with the detailed requirements of virtual try-on technology, which will be greatly beneficial for future studies and developments in related fields. Apart from resource challenges, the pairing issues during data preprocessing also hindered the training results. After repeated adjustments, the accuracy of data processing was ensured, which improved the quality of the model. Through these challenges, I learned the importance of paying close attention to details during experiments, as early mistakes can lead to significant time loss.

In the future, I will focus more on how to manage time effectively to avoid being constrained by time limitations, as happened in this project, where I couldn’t achieve the optimal results. Additionally, I plan to deepen my understanding of virtual try-on technology, particularly in how to achieve more detailed garment deformation and wrinkle handling in different scenarios. This experience has also taught me the importance of staying calm and finding appropriate solutions when facing resource constraints and technical challenges. I believe that these experiences and skills will help me tackle various challenges more effectively in both my future research and professional work.

9. Reference

Athar, S., Koriakina, N., and Sladoje, N. (2021). 'InSeGAN: A Generative Approach to Segmenting Identical Instances in Depth Images', Proceedings of the IEEE International Conference on Computer Vision (ICCV).

Andersson, A., Koriakina, N., and Sladoje, N. (2022). 'End-to-End Multiple Instance Learning with Gradient Accumulation', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Bookstein, F.L. (1989). 'Principal Warps: Thin-Plate Splines and the Decomposition of Deformations', IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI).

Canny, J. (1986). 'A Computational Approach to Edge Detection', IEEE Transactions on Pattern Analysis and Machine Intelligence.

Cao, Z., Simon, T., Wei, S.E. and Sheikh, Y. (2017). 'Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Choi, M., Choi, S., Kim, M., Kim, T., Yu, H. and Kim, J. (2021). 'VITON-HD: High-Resolution Virtual Try-On via Misalignment-Aware Normalization', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Dong, H., Liang, X., Shen, X., Wang, B., Lai, H., Zhu, J., Hu, Z. and Yin, J. (2019). 'Towards Multi-Pose Guided Virtual Try-On Network', Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).

Dong, H., Liang, X., Wei, Y., Lai, H., Zhu, J. and Yin, J. (2019). 'VTN: Virtual Try-on Network', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Dumoulin, V. and de Vries, F. G. (2017). 'A Learned Representation For Artistic Style: Conditional Batch Normalization for Style Transfer', Proceedings of the International Conference on Learning Representations (ICLR).

Gatys, L.A., Ecker, A.S. and Bethge, M. (2016). 'Image Style Transfer Using Convolutional Neural Networks', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A. (2017). 'Improved Training of Wasserstein GANs', Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS).

He, K., Zhang, X., Ren, S. and Sun, J. (2016). 'Deep Residual Learning for Image Recognition', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). 'Mask R-CNN', Proceedings of the IEEE International Conference on Computer Vision (ICCV).

Huang, H., Li, Y., Zhang, J. and Zhang, X. (2017). 'Toward Virtual Try-On via Human Parsing and Pose Estimation', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. and Hochreiter, S. (2017). 'GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium', Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS).

Han, X., Wu, Z., Wu, Z., Yu, R. and Davis, L.S. (2018). 'VITON: An Image-based Virtual Try-On Network', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Han, X., Wu, Z., Lin, Z., Liu, J., Davis, L.S. and Zhou, J. (2019). 'ClothFlow: A Flow-based Model for Clothed Person Generation', Proceedings of the IEEE International Conference on Computer Vision (ICCV).

Ioffe, S. and Szegedy, C. (2015). 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', Proceedings of the 32nd International Conference on Machine Learning (ICML).

Isola, P., Zhu, J.Y., Zhou, T. Efros, A.A. (2017). 'Image-to-Image Translation with Conditional Adversarial Networks', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Klein, F. (1872). 'Erlangen Program: A Comparative Review of Recent Researches in Geometry'.

Karras, T., Aila, T., Laine, S. and Lehtinen, J. (2017). 'Progressive Growing of GANs for Improved Quality, Stability, and Variation', Proceedings of the International Conference on Learning Representations (ICLR).

Karnewar, A. and Wang, O. (2020). 'MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Lloyd, S. (1982). 'Least Squares Quantization in PCM', IEEE Transactions on Information Theory.

Liu, X., Yamaguchi, K., Wang, L. and Berg, T. (2015). 'Fashion Parsing with Weak Color-Category Labels', Proceedings of the European Conference on Computer Vision (ECCV).

Lewis, K.M., Varadharajan, S. and Kemelmacher-Shlizerman, I. (2021). 'TryOnGAN: Body-Aware Try-On via Layered Interpolation', ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2021), 40(4).

Ma, K. and Deng, J. (2004). 'Virtual Try-on Clothing using Image Processing', Journal of Virtual Clothing, 10(2).

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G. Wu, H. (2017). 'Mixed Precision

Training', Proceedings of the International Conference on Learning Representations (ICLR).

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). 'Spectral Normalization for Generative Adversarial Networks', Proceedings of the International Conference on Learning Representations (ICLR).

Otsu, N. (1979). 'A Threshold Selection Method from Gray-Level Histograms', IEEE Transactions on Systems, Man, and Cybernetics, 9(1)

Ronneberger, O., Fischer, P., and Brox, T. (2015). 'U-Net: Convolutional Networks for Biomedical Image Segmentation', Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', Journal of Machine Learning Research.

Ulyanov, D., Vedaldi, A. and Lempitsky, V. (2016). 'Instance Normalization: The Missing Ingredient for Fast Stylization', Proceedings of the International Conference on Learning Representations (ICLR).

Vapnik, V. and Guyon, I. (1992). 'A Training Algorithm for Optimal Margin Classifiers', Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory.

Wang, Z., Bovik, A.C., Sheikh, H.R. Simoncelli, E.P. (2004). 'Image Quality Assessment: From Error Visibility to Structural Similarity', IEEE Transactions on Image Processing.

Wang, B., Zheng, H., Liang, X., Chen, Y., Zhang, L. and Lin, L. (2018). 'CP-VTON: Clothing Shape and Texture Preserving Image-Based Virtual Try-On', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Wang, Z., Zheng, H., He, P., Chen, W. and Zhou, M. (2023). 'DIFFUSION-GAN: TRAINING GANS WITH DIFFUSION', Proceedings of the International Conference on Learning Representations (ICLR).

Zhu, Z., Luo, S., Wang, X., and Tang, X. (2017). 'Human Pose Guided Clothing Transfer', Proceedings of the IEEE International Conference on Computer Vision (ICCV).

Zhang, R., Isola, P., Efros, A.A., Shechtman, E. and Wang, O. (2018). 'The Unreasonable Effectiveness of Deep Features as a Perceptual Metric', Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

10. Appendices

```
        # Part 3. Try-on synthesis
        misalign_mask = parse[:, 2:3] - warped_cm
        misalign_mask[misalign_mask < 0.0] = 0.0
        parse_div = torch.cat((parse, misalign_mask), dim=1)
        parse_div[:, 2:3] -= misalign_mask

        with autocast():  # Forward propagation and loss calculation with autocast
            alias_input = torch.cat((img_agnostic, pose, warped_c), dim=1)
            fake_output = alias(alias_input, parse, parse_div, misalign_mask)

            # =========Discriminator Training=========

            # Discriminator input: alias_input, parse, parse_div, misalign_mask, add img
            real_input = torch.cat([alias_input, parse, parse_div, misalign_mask, img], dim=1)
            real_output = discriminator(real_input)

            # Discriminator input: alias_input, parse, parse_div, misalign_mask, add fake_output.detach()
            fake_input_detached = torch.cat([alias_input, parse, parse_div, misalign_mask, fake_output.detach()], dim=1)
            fake_output_detached = discriminator(fake_input_detached)

            # Convert real_labels and fake_labels to their corresponding tensors.
            real_labels = [torch.ones_like(real_out).cuda() for real_out in real_output]
            fake_labels = [torch.zeros_like(fake_out).cuda() for fake_out in fake_output_detached]

            # For multi-scale output, calculate and sum the loss on a scale-by-scale basis.
            d_loss_real = sum([adversarial_loss(real_out, real_label) for real_out, real_label in zip(real_output, real_labels)])
            d_loss_fake = sum([adversarial_loss(fake_out, fake_label) for fake_out, fake_label in zip(fake_output_detached, fake_labels)])


            # Calculating the Gradient Penalty of a Criterion
            gp = gradient_penalty(discriminator, real_input, fake_input_detached, img.device)
            d_loss = (d_loss_real + d_loss_fake) * 0.5 + lambda_gp * gp

            # ==========Generator Training=========

            # Generator Loss: Ensure that the output of the generator matches the true labels
            # Use the same input as the fake image fake_input (don't use detach(), let the generator get the gradient)
            fake_input_g = torch.cat([alias_input, parse, parse_div, misalign_mask, fake_output], dim=1)
            fake_output_g = discriminator(fake_input_g)

            # The generator expects the discriminator to determine that the generated image is real.
            real_labels_g = [torch.ones_like(fake_out_g).cuda() for fake_out_g in fake_output_g]

            # Calculate the generator loss for each scale individually using the zip function
            g_adversarial_loss = sum([adversarial_loss(fake_out_g, real_label_g) for fake_out_g, real_label_g in zip(fake_output_g, real_labels_g)])
            # Generator Feature Matching Loss
            g_feature_matching_loss = feature_matching_loss(fake_output, img)
            # Loss of perception
            perceptual_loss_value = g_perceptual_loss(fake_output, img)

            # total loss
            g_loss = g_adversarial_loss + 10 * g_feature_matching_loss + 10 * perceptual_loss_value

        # Using scaler for gradient backward
        # Backpropagation using gradient scaler for generator and discriminator respectively
        scaler.scale(d_loss).backward(retain_graph=True)  # cumulative gradient
        scaler.scale(g_loss).backward()  # cumulative gradient

        if (i + 1) % accumulation_steps == 0:
            scaler.step(optimizer_D)
            scaler.update()
            optimizer_D.zero_grad()

            scaler.step(optimizer_G)
            scaler.update()
            optimizer_G.zero_grad()

            torch.cuda.empty_cache()

        discriminator_losses.append(d_loss.item() * accumulation_steps)

        generator_losses.append(g_loss.item() * accumulation_steps)
```

Figure 5: Code of ALIAS Generator and Multiscale Discriminator training

```
def feature_matching_loss(fake_feats, real_feats):
    loss = 0
    for fake, real in zip(fake_feats, real_feats):
        K = len(fake)  # Assuming the number of features per layer
        for fake_feat, real_feat in zip(fake, real):
            loss += F.l1_loss(fake_feat, real_feat) / K  # Layer-by-layer normalisation
    return loss

class perceptual_loss(nn.Module):
    def __init__(self, layids = None):
        super(perceptual_loss, self).__init__()
        self.vgg = VGG19FeatureExtractor()
        self.vgg.cuda()
        self.criterion = nn.L1Loss()
        self.weights = [1.0/32, 1.0/16, 1.0/8, 1.0/4, 1.0]
        self.layids = layids

    def forward(self, x, y):
        x_vgg, y_vgg = self.vgg(x), self.vgg(y)
        loss = 0
        if self.layids is None:
            self.layids = list(range(len(x_vgg)))
        for i in self.layids:
            loss += self.weights[i] * self.criterion(x_vgg[i], y_vgg[i].detach())
        return loss
```

Figure 6: Code of Feature Matching Loss & Perceptual Loss

```python
def gradient_penalty(discriminator, real_data, fake_data, device):
    """Calculating gradient penalties"""
    alpha = torch.rand(real_data.size(0), 1, 1, 1).to(device)
    interpolates = (alpha * real_data + ((1 - alpha) * fake_data)).requires_grad_(True)

    d_interpolates = discriminator(interpolates)
    fake = torch.ones(d_interpolates[0].size()).to(device)

    gradients = autograd.grad(
        outputs=d_interpolates[0], inputs=interpolates,
        grad_outputs=fake, create_graph=True, retain_graph=True, only_inputs=True
    )[0]

    gradients = gradients.view(gradients.size(0), -1)
    gradient_penalty = ((gradients.norm(2, dim=1) - 1) ** 2).mean()
    return gradient_penalty
```

Figure 7: Code of Gradient Penalty

```python
class MultiScaleDiscriminator(nn.Module):
    # Define a multi-scale discriminator combining instance normalisation and spectral normalisation.
    def __init__(self, input_nc, ndf=64, n_layers=6, num_D=2):
        super(MultiScaleDiscriminator, self).__init__()
        self.num_D = num_D
        self.n_layers = n_layers

        for i in range(num_D):
            netD = self.get_discriminator(input_nc, ndf, n_layers)
            self.add_module('discriminator_%d' % i, netD)

    def get_discriminator(self, input_nc, ndf, n_layers):
        # Constructing the layers of each multiscale discriminator
        layers = [nn.Conv2d(input_nc, ndf, kernel_size=4, stride=2, padding=1)]
        nf = ndf
        for n in range(1, n_layers):
            nf_prev = nf
            nf = min(ndf * (2 ** n), 512)  # Ensure that the maximum number of channels does not exceed 512
            layers += [spectral_norm(nn.Conv2d(nf_prev, nf, kernel_size=4, stride=2, padding=1)),
                       nn.InstanceNorm2d(nf),
                       nn.LeakyReLU(0.2, True),
                       nn.Dropout(0.2)  # add Dropout
                       ]
        # Last level of output
        layers += [spectral_norm(nn.Conv2d(nf, 1, kernel_size=4, padding=1))]
        return nn.Sequential(*layers)

    def forward(self, x):
        results = []
        for name, D in self.named_children():
            # Print input size
            # print(f"Input size before {name}: {x.size()}")

            # Ensure that the input feature map size is larger than the kernel size.
            if x.size(2) < 4 or x.size(3) < 4:
                print(f"Skipping {name} due to small input size: {x.size()}")
                break

            results.append(D(x))
            # Reduction in average pooling scale
            x = F.avg_pool2d(x, kernel_size=3, stride=2, padding=[1, 1], count_include_pad=False)
        return results
```

Figure 8: Code of Multi-Scale Discriminator

```python
class VGG19FeatureExtractor(nn.Module):
    def __init__(self, requires_grad=False):
        super(VGG19FeatureExtractor, self).__init__()
        vgg_pretrained_features = vgg19(pretrained=True).features
        self.slice1 = torch.nn.Sequential()
        self.slice2 = torch.nn.Sequential()
        self.slice3 = torch.nn.Sequential()
        self.slice4 = torch.nn.Sequential()
        self.slice5 = torch.nn.Sequential()
        for x in range(2):
            self.slice1.add_module(str(x), vgg_pretrained_features[x])
        for x in range(2, 7):
            self.slice2.add_module(str(x), vgg_pretrained_features[x])
        for x in range(7, 12):
            self.slice3.add_module(str(x), vgg_pretrained_features[x])
        for x in range(12, 21):
            self.slice4.add_module(str(x), vgg_pretrained_features[x])
        for x in range(21, 30):
            self.slice5.add_module(str(x), vgg_pretrained_features[x])
        if not requires_grad:
            for param in self.parameters():
                param.requires_grad = False

    def forward(self, X):
        h_relu1 = self.slice1(X)
        h_relu2 = self.slice2(h_relu1)
        h_relu3 = self.slice3(h_relu2)
        h_relu4 = self.slice4(h_relu3)
        h_relu5 = self.slice5(h_relu4)
        out = [h_relu1, h_relu2, h_relu3, h_relu4, h_relu5]
        return out
```

Figure 9: Code of VGG Feature Extractor

```python
def calculate_fid_with_progress(real_dir, fake_dir, batch_size=50):
    real_paths = [osp.join(real_dir, f) for f in os.listdir(real_dir) if osp.isfile(osp.join(real_dir, f))]
    fake_paths = [osp.join(fake_dir, f) for f in os.listdir(fake_dir) if osp.isfile(osp.join(fake_dir, f))]

    fid_value = fid_score.calculate_fid_given_paths([real_dir, fake_dir], batch_size=batch_size, device='cuda', dims=2048)

    print(f'FID Calculation Completed: {fid_value}')
    return fid_value

def setup_model_cache():
    os.environ['TORCH_HOME'] = '/content/drive/MyDrive/checkpoints'
    checkpoints_path = '/content/drive/MyDrive/checkpoints/pt_inception-2015-12-05-6726825d.pth'

    if not os.path.exists(checkpoints_path):
        print(f"Downloading Inception model weights to {checkpoints_path}")
        !wget -O {checkpoints_path} https://github.com/mseitzer/pytorch-fid/releases/download/fid_weights/pt_inception-2015-12-05-6726825d.pth
    else:
        print(f"Inception model weights already exist at {checkpoints_path}")

setup_model_cache()

fake_dir = '/content/drive/MyDrive/datasets/results/my_test_result_modified4_26000'
real_dir = '/content/drive/MyDrive/datasets/test2/image'

# FID Analysis
fid = calculate_fid_with_progress(real_dir, fake_dir)
print(f'FID : {fid}')
torch.cuda.empty_cache()
```

Figure 10: Code of FID Analysis

```python
# Initialise LPIPS model
lpips_model = lpips.LPIPS(net='alex').cuda()  # using one of 'alex', 'vgg', 'squeeze'

def check_image_range(images, label):
    for idx, img in enumerate(images):
        min_val, max_val = img.min().item(), img.max().item()
        if min_val < -1 or max_val > 1:
            print(f"Warning: {label} Image {idx} out of range. Min: {min_val}, Max: {max_val}")

# LPIPS batch size
def calculate_lpips_in_batches(lpips_model, real_images, fake_images, batch_size=16):
    lpips_scores = []
    total_batches = len(real_images) // batch_size + 1
    for i in tqdm(range(total_batches), desc="Calculating LPIPS in Batches"):
        real_batch = real_images[i*batch_size:(i+1)*batch_size]
        fake_batch = fake_images[i*batch_size:(i+1)*batch_size]

        check_image_range(real_batch, label=f"Real Batch {i}")
        check_image_range(fake_batch, label=f"Fake Batch {i}")

        with torch.no_grad():
            score = lpips_model(real_batch, fake_batch).mean().item()

        if not np.isnan(score):  # check whether there is NaN
            lpips_scores.append(score)
        else:
            print(f"Warning: NaN detected in batch {i}, skipping this batch.")

    if len(lpips_scores) == 0:
        print("Error: No valid LPIPS scores, all batches contain NaN.")
        return float('nan')
    return np.mean(lpips_scores)

lpips_transform = transforms.Compose([
    transforms.Resize((512, 384)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

fake_dir = '/content/drive/MyDrive/datasets/results/my_test_result_modified4_26000'
real_dir = '/content/drive/MyDrive/datasets/test2/image'

# sort
real_images = sorted([f for f in os.listdir(real_dir) if osp.isfile(osp.join(real_dir, f))])
fake_images = sorted([f for f in os.listdir(fake_dir) if osp.isfile(osp.join(fake_dir, f))])

fake_images_lpips = [lpips_transform(Image.open(osp.join(fake_dir, f))) for f in tqdm(fake_images, desc="Processing Fake Images")]
real_images_lpips = [lpips_transform(Image.open(osp.join(real_dir, f))) for f in tqdm(real_images, desc="Processing Real Images")]

fake_images_lpips = torch.stack(fake_images_lpips).cuda()
real_images_lpips = torch.stack(real_images_lpips).cuda()

# Calculate LPIPS
batch_size = 16
lpips_score = calculate_lpips_in_batches(lpips_model, real_images_lpips, fake_images_lpips, batch_size=batch_size)
print(f'LPIPS : {lpips_score}')

torch.cuda.empty_cache()
```

Figure 11: Code of LPIPS Analysis

```python
def calculate_ssim_in_batches(real_images, fake_images, batch_size=16):
    ssim_scores = []
    total_batches = len(real_images) // batch_size + 1
    for i in tqdm(range(total_batches), desc="Calculating SSIM in Batches"):
        real_batch = real_images[i*batch_size:(i+1)*batch_size].cpu().numpy()
        fake_batch = fake_images[i*batch_size:(i+1)*batch_size].cpu().numpy()
        for real_img, fake_img in zip(real_batch, fake_batch):
            real_img = np.clip(real_img.transpose(1, 2, 0), 0, 1)
            fake_img = np.clip(fake_img.transpose(1, 2, 0), 0, 1)
            score = ssim(real_img, fake_img, channel_axis=-1, data_range=real_img.max() - real_img.min())
            ssim_scores.append(score)
    return sum(ssim_scores) / len(ssim_scores)


ssim_transform = transforms.Compose([
    transforms.Resize((512, 384)),
    transforms.ToTensor()
])

fake_dir = '/content/drive/MyDrive/datasets/results/my_test_result_modified4_26000'
real_dir = '/content/drive/MyDrive/datasets/test2/image'  #

# sort
real_images = sorted([f for f in os.listdir(real_dir) if osp.isfile(osp.join(real_dir, f))])
fake_images = sorted([f for f in os.listdir(fake_dir) if osp.isfile(osp.join(fake_dir, f))])



# Calculate SSIM
batch_size = 16
fake_images_ssim = [ssim_transform(Image.open(osp.join(fake_dir, f))) for f in tqdm(fake_images, desc="Processing Fake Images")]
real_images_ssim = [ssim_transform(Image.open(osp.join(real_dir, f))) for f in tqdm(real_images, desc="Processing Real Images")]
fake_images_ssim = torch.stack(fake_images_ssim).cuda()
real_images_ssim = torch.stack(real_images_ssim).cuda()
ssim_score = calculate_ssim_in_batches(real_images_ssim, fake_images_ssim, batch_size=batch_size)
print()
print(f'SSIM : {ssim_score}')
```

Figure 12: Code of SSIM Analysis