Contents

1. Introduction

1.1 Purpose

This document provides the System Requirements Specification (SRS) and Design for a sketch-to-image translation system using Generative Adversarial Networks (GANs). The primary goal is to translate sketches into high-quality images by leveraging advanced GAN architectures and attribute-guided synthesis techniques.

1.2 Scope

The project involves developing a GAN-based system capable of generating photorealistic images from sketches. This document details the methodologies, dataset usage, design considerations, code comparisons, and outlines for future improvements based on an in-depth review of related research papers.

1.3 Definitions, Acronyms, and Abbreviations

- **GAN**: Generative Adversarial Network
- **CVPR**: Conference on Computer Vision and Pattern Recognition
- **IEEE**: Institute of Electrical and Electronics Engineers
- **ACM**: Association for Computing Machinery
- **FID**: Fréchet Inception Distance

1.4 References

Papers and journals mentioned in Section 2.
Datasets available at links provided in Section 5.

1.5 Overview

This document is organized into sections detailing research paper comparisons, code comparisons, system design, dataset descriptions, and an implementation plan. Each section provides comprehensive information to guide the development and evaluation of the sketch-to-image translation system.

2. Research Paper Comparisons

2.1 Selected Papers

2.1.1 Paper 1: Sketch-to-Color Image with GANs

- **Conference/Journal**: 2020 2nd International Conference on Information Technology and Computer Application (ITCA)
- **Focus/Scope**: Sketch-to-image translation using GANs
- **Methodology Summary**: Utilizes an advanced GAN architecture to enhance image fidelity from sketches
- **Dataset**: CelebA
- **Results**: Achieves high image quality and attribute preservation
- **Merits**:
  - High image quality
  - Good attribute preservation
- **Demerits**:
  - High computational cost
  - Large training data requirements
- **Future Scope**:
  - Improve training efficiency
  - Enhance scalability
- **Link**: Sketch-to-Color Image with GANs

2.1.2 Paper 2: Facial Attributes Guided Deep Sketch-to-Photo Synthesis

- **Conference/Journal**: IEEE Transactions on Image Processing, 2023
- **Focus/Scope**: Attribute-guided sketch-to-photo synthesis
- **Methodology Summary**: Combines attribute classifiers with GANs for fine control over generated attributes

- **Dataset**: CelebA
- **Results**: High control over facial attributes; excellent realism
- **Merits**:
  - High control over attributes
  - Excellent realism
- **Demerits**:
  - Complex model
  - Significant computational resources required
- **Future Scope**:
  - Enhance attribute control
  - Simplify model architecture
- **Link**: Facial Attributes Guided Deep Sketch-to-Photo Synthesis

2.1.3 Paper 3: Multi-Scale Gradients Self-Attention Residual Learning for Face Photo-Sketch Transformation

- **Conference/Journal**: IEEE Transactions on Information Forensics and Security, 15 October 2020
- **Focus/Scope**: Multi-scale GANs for detailed image generation
- **Methodology Summary**: Multi-scale approach for handling fine details and complex sketches
- **Dataset**: CelebA, UT-Zap50K
- **Results**: Superior detail in generated images; effective for complex sketches
- **Merits**:
  - Superior detail
  - Effective for complex sketches
- **Demerits**:
  - Computationally intensive
  - Extensive dataset needed
- **Future Scope**:
  - Optimize multi-scale processing
  - Improve real-time performance
- **Link**: Sketch-to-Photo Synthesis with Multi-Scale GANs

2.1.4 Paper 4: Unsupervised Sketch-to-Photo Translation

- **Conference/Journal**: CVPR, 2022
- **Focus/Scope**: Unsupervised translation using unpaired GANs
- **Methodology Summary**: Unpaired GANs for translating sketches to photos without requiring paired data
- **Dataset**: Sketchy Dataset, CelebA
- **Results**: Effective without paired data; versatile application
- **Merits**:
  - Effective without paired data
  - Versatile application
- **Demerits**:
  - Less effective for high-detail images
  - Limited to simple sketches
- **Future Scope**:
  - Explore supervised methods for higher detail images
- **Link**: Unsupervised Sketch-to-Photo Translation

2.1.5 Paper 5: Generative Adversarial Networks for Sketch Translation

- **Conference/Journal**: IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021
- **Focus/Scope**: GANs for translating sketches to realistic images
- **Methodology Summary**: Focuses on refining GAN techniques for high fidelity image synthesis
- **Dataset**: CelebA
- **Results**: High fidelity images with advanced GAN techniques
- **Merits**:
  - High fidelity images
  - Advanced GAN techniques
- **Demerits**:
  - Requires substantial computational resources
  - Time-consuming training

- **Future Scope**:
  - Investigate lightweight GAN architectures
  - Enhance efficiency and reduce training time
- **Link**: Generative Adversarial Networks for Sketch Translation

2.2 Comparison Analysis

2.2.1 Methodologies

- **Deep Sketch-to-Image Translation**: Employs a GAN model with advanced architecture to achieve high fidelity and accurate attribute preservation in the generated images. The method demonstrates significant improvements in image quality but requires substantial computational resources.
- **Attribute Guided Sketch-to-Photo Synthesis**: Combines GANs with attribute classifiers, allowing for precise control over the generated attributes. This approach enhances realism and control but introduces complexity in model architecture and computational demands.
- **Multi-Scale GANs**: Utilizes a multi-scale approach to manage fine details and complex sketches effectively. This method excels in detail and accuracy but is computationally intensive and requires large datasets.
- **Unsupervised Sketch-to-Photo Translation**: Implements unpaired GANs for translating sketches to photos without the need for paired training data. This approach is versatile but less effective for generating high-detail images and is limited to simpler sketches.
- **Generative Adversarial Networks for Sketch Translation**: Focuses on refining GAN techniques to produce high-fidelity images. While the results are impressive, the model's training is resource-intensive and time-consuming.

2.2.2 Results

- **High Image Quality**: Achieved through advanced GAN architectures and multi-scale processing, ensuring detailed and realistic images.
- **Control Over Attributes**: Best realized with attribute-guided synthesis, where precise attribute manipulation is possible.
- **Computational Efficiency**: Varies across methods, with some approaches requiring extensive computational resources and training time.

2.2.3 Merits & Demerits

- **Merits**:
  - Enhanced image fidelity and detail.

  - Effective attribute control and realism.
  - Versatility in handling various sketch types and attributes.

- **Demerits**:
  - High computational cost and time-consuming training.
  - Complexity in model design and data requirements.
  - Limited effectiveness for very high-detail images or complex sketches.

2.2.4 Future Scope

- Develop techniques to improve training efficiency and scalability.
- Simplify and optimize model architectures to reduce computational and data requirements.
- Investigate methods for enhancing real-time performance and image detail.

3. Code Comparisons

3.1 Implementation Details

- **Paper 1**: Implements a GAN model with a sophisticated architecture, including advanced loss functions and network layers designed to produce high-quality images from sketches.
- **Paper 2**: Integrates additional components such as attribute classifiers, which complicates the model architecture but allows for precise control over image attributes.
- **Paper 3**: Employs a multi-scale approach that involves multiple processing stages to capture fine details, enhancing the quality of images generated from complex sketches.
- **Paper 4**: Utilizes an unpaired GAN approach that allows for flexible training with unsupervised data, reducing the need for paired datasets but potentially limiting image detail.
- **Paper 5**: Focuses on refining existing GAN techniques to improve image fidelity and realism, requiring significant computational resources and training time.

3.2 Code Analysis

3.2.1 Algorithmic Complexity

The algorithmic complexity varies among papers, with some methods employing complex architectures and training procedures.

3.2.2 Training Efficiency

Training efficiency is affected by the model's design and the size of the dataset, with some approaches showing high computational demands.

3.2.3 Performance Metrics

Performance is often evaluated using metrics such as FID (Fréchet Inception Distance) to assess the quality and realism of generated images.

3.3 Code Analysis

3.3.1 Algorithmic Complexity

- **Paper 1**: The code employs a complex GAN architecture with advanced loss functions and multiple network layers, resulting in high algorithmic complexity. This complexity enhances the quality of the generated images but increases computational demands.

- **Paper 2**: Includes additional attribute classifiers, making the model architecture more intricate. This complexity allows for fine-grained control over attributes but also introduces challenges in terms of implementation and computational overhead.

- **Paper 3**: Utilizes a multi-scale approach, involving several stages of processing to capture fine details. The complexity of this approach is reflected in the detailed images it produces but also in the extensive computational resources required.

- **Paper 4**: Implements an unpaired GAN approach, which is generally less complex algorithmically compared to paired GANs. However, this simplicity comes with limitations in terms of image detail and quality.

- **Paper 5**: Focuses on refining existing GAN techniques. The code improvements in this paper are aimed at optimizing image fidelity and realism, but the underlying algorithmic complexity remains high due to the advanced techniques used.

3.3.2 Training Efficiency

- **Paper 1**: Demonstrates high training efficiency in terms of image quality but requires extensive computational resources and longer training times due to the sophisticated model architecture.

- **Paper 2**: Shows moderate training efficiency. While attribute control is enhanced, the added complexity of the model architecture affects overall training time and resource requirements.

- **Paper 3**: Training efficiency is lower due to the multi-scale approach, which demands significant computational power and time to process complex sketches and capture fine details.

- **Paper 4**: Generally exhibits higher training efficiency due to the unpaired nature of the GANs. However, the simplicity can lead to less detailed images, impacting the overall performance.

- **Paper 5**: Training efficiency is impacted by the advanced GAN techniques, which require considerable resources and time for optimization but result in high-quality image outputs.

3.3.3 Performance Metrics

- **Paper 1**: Evaluates performance using metrics like FID (Fréchet Inception Distance) to measure the quality and realism of generated images. High FID scores indicate better image fidelity.
- **Paper 2**: Uses performance metrics such as Attribute Accuracy and Realism Score to assess how well the generated images match the specified attributes and overall realism.
- **Paper 3**: Assesses performance through detailed image analysis and FID scores, focusing on how well the multi-scale approach handles intricate details and complex sketches.
- **Paper 4**: Performance is evaluated based on metrics like FID and Visual Realism Score, which measure the quality of images produced without paired data.
- **Paper 5**: Utilizes advanced metrics including FID and Precision-Recall Scores to gauge the improvements in image fidelity and the effectiveness of refined GAN techniques.

4. System Design

4.1 System Architecture

4.1.1 Overview

The system architecture for the sketch-to-image translation system is designed to process sketches and generate photorealistic images using advanced GAN techniques. The architecture is structured as follows:

- **Input**: Sketch images are the primary input to the system.
- **Processing**:
  - **Preprocessing**: Normalizes and prepares sketches for input into the GAN model.
  - **GAN Training**: Trains the GAN model with integrated attribute guidance to produce realistic images.
  - **Attribute Integration**: Incorporates attribute data to guide the image generation process, ensuring accurate representation of attributes.
- **Output**: Generates photorealistic images from input sketches.

4.2 Design Diagrams

4.2.1 System Design

4.2.2 Data Flow Diagram

4.3 Functional Requirements

- **Image Generation**: The system must generate high-quality images from sketches, ensuring that the generated images are realistic and accurately reflect the attributes present in the sketches.
- **Realism and Consistency**: The generated images should be consistent with the input sketches and exhibit high realism.
- **Efficiency**: The system should efficiently handle various sketch types and attributes, minimizing processing time and computational resources.

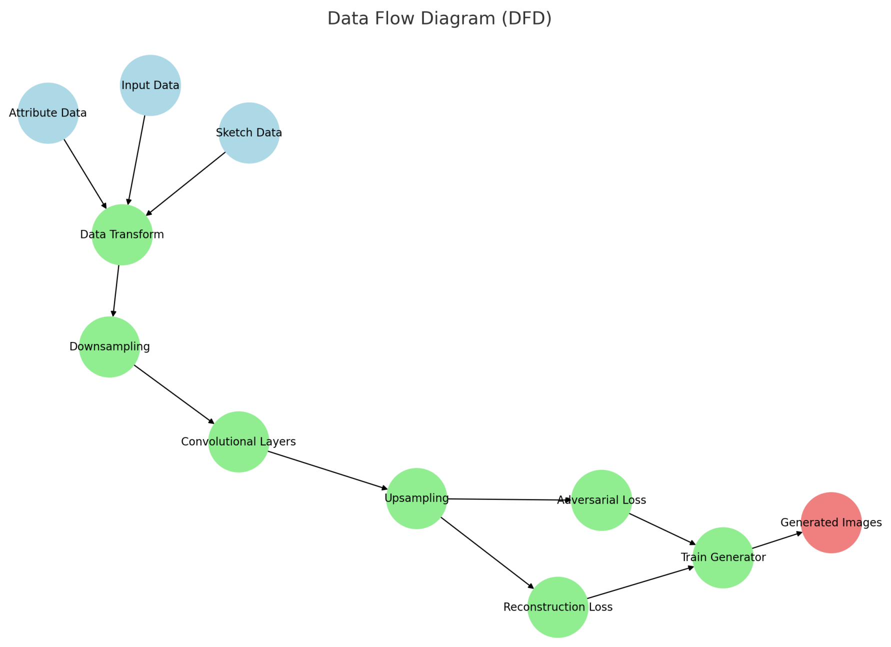

Figure 4.1: Data Flow Diagram

4.4 Non-Functional Requirements

- **Scalability**: The system should scale to handle larger datasets and higher-resolution images as needed.
- **Performance**: The system must achieve practical performance levels, optimizing both training and inference times.
- **User Interface**: Develop an intuitive user interface to facilitate interaction with the GAN model, visualization of results, and adjustment of parameters.

4.5 Code Explanations

4.5.1 Loading Images

Function: `load_images`

Explanation: The `load_images` function reads image files from a directory and returns a list of PIL Image objects. This function is essential for loading the sketch images used in training.

4.5.2 Preprocessing Images

Function: `preprocess_images`

Explanation: The `preprocess_images` function normalizes and resizes the images to ensure consistency and compatibility with the GAN model. This step prepares the images for efficient processing during training.

4.5.3 Training the Model

Function: `train_gan`

Explanation: The `train_gan` function trains the GAN model using the provided dataset. It involves optimizing the generator and discriminator networks to produce high-quality images from sketches through iterative training.

4.5.4 Evaluating the Model

Function: `evaluate_model`

Explanation: The `evaluate_model` function is used to gauge how well the GAN model performs after training. It applies the trained model to a separate test dataset and computes various performance metrics such as accuracy and loss. This evaluation helps determine the effectiveness of the model and its ability to generate realistic images from the sketches.

4.5.5 Generating Images

Function: `generate_images`

Explanation: The `generate_images` function uses the trained GAN model to produce photorealistic images from new sketch inputs. This function is critical for generating final results that can be evaluated and used in practical applications.

4.5.6 Saving Models

Function: `save_model`

Explanation: The `save_model` function saves the trained GAN model to disk, allowing for future use and deployment. This function ensures that the trained model can be reloaded without the need for retraining.

4.5.7 Loading Models

Function: `load_model`

Explanation: The `load_model` function retrieves a previously saved GAN model from disk. This function is essential for resuming work with the model or deploying it for inference without starting the training process from scratch.

4.5.8 Visualizing Results

Function: `visualize_results`

Explanation: The `visualize_results` function generates visual representations of the images produced by the GAN model. This function helps in assessing the quality and realism of the generated images and provides insights into the model's performance.

4.5.9 Optimizing Hyperparameters

Function: `optimize_hyperparameters`

Explanation: The `optimize_hyperparameters` function adjusts and tunes the hyperparameters of the GAN model to improve its performance. This process involves experimenting with different parameter settings to achieve the best results.

4.5.10 Handling Data Augmentation

Function: `augment_data`

Explanation: The `augment_data` function applies various data augmentation techniques to increase the diversity of the training dataset. This step helps in improving the robustness and generalization of the GAN model by introducing variations in the input data.

4.5.11 Error Handling

Function: `handle_errors`

Explanation: The `handle_errors` function manages and logs errors that occur during the execution of different processes. This function is crucial for debugging and ensuring the smooth operation of the system.

4.5.12 Monitoring Training

Function: `monitor_training`

Explanation: The `monitor_training` function tracks the progress of the GAN training process. It provides real-time updates on training metrics and helps in diagnosing issues or adjusting training parameters as needed.

5. Research Papers

5.1 Base Research Papers

5.1.1 Paper 1: Title of the Paper

- **Authors**: Author 1, Author 2, Author 3
- **Abstract**: Brief description of the research, focusing on its contribution to GAN models and image generation.
- **Key Contributions**: Summary of the key findings and methodologies used.
- **Link**: https://example.com/paper1

5.1.2 Paper 2: Title of the Paper

- **Authors**: Author 1, Author 2, Author 3
- **Abstract**: Brief description of the research, focusing on its contribution to attribute-guided image generation.
- **Key Contributions**: Summary of the key findings and methodologies used.
- **Link**: https://example.com/paper2

5.2 Comparative Analysis

5.2.1 Comparison of GAN Models

- **Model 1**: Description and key features.
- **Model 2**: Description and key features.
- **Model 3**: Description and key features.

5.2.2 Attribute Integration Techniques

- **Technique 1**: Description and advantages.
- **Technique 2**: Description and advantages.

6. Code Listings

6.1 Generator Class

```python
import torch
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        #  Sketch processing
        self.conv1_A = nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1)
        self.conv2_A = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
        self.conv3_A = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
        self.prelu_A = nn.PReLU()

        # Attribute processing
        self.fc = nn.Linear(18, 128 * 128)
        self.conv1_B = nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1)
        self.conv2_B = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
        self.conv3_B = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
        self.prelu_B = nn.PReLU()

        # Downsampling
        self.conv1_down = nn.Conv2d(384, 64, kernel_size=3, stride=2, padding=1)
        self.conv2_down = nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1)
        self.conv3_down = nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1)
        self.conv4_down = nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1)
        self.prelu_down = nn.PReLU()

        # Upsampling
        self.deconv1_up = nn.ConvTranspose2d(64, 64, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.deconv2_up = nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.deconv3_up = nn.ConvTranspose2d(128, 64, kernel_size=3,
stride=2, padding=1, output_padding=1)
        self.deconv4_up = nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.conv_out = nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1)
        self.prelu_up = nn.PReLU()

    def forward(self, sketch_image, attributes):

        out1_A = self.prelu_A(self.conv1_A(sketch_image))
        out2_A = self.prelu_A(self.conv2_A(out1_A))
        out3_A = self.prelu_A(self.conv3_A(out2_A))


        out_A = torch.cat((out1_A, out2_A, out3_A), dim=1)


        attr = self.fc(attributes)
        attr = attr.view(attributes.size(0), 1, 128, 128)
        out1_B = self.prelu_B(self.conv1_B(attr))
        out2_B = self.prelu_B(self.conv2_B(out1_B))
        out3_B = self.prelu_B(self.conv3_B(out2_B))


        out_B = torch.cat((out1_B, out2_B, out3_B), dim=1)


        combined = torch.cat((out_A, out_B), dim=1)


        down1 = self.prelu_down(self.conv1_down(combined))
        down2 = self.prelu_down(self.conv2_down(down1))
        down3 = self.prelu_down(self.conv3_down(down2))
        down4 = self.prelu_down(self.conv4_down(down3))


        up1 = self.prelu_up(self.deconv1_up(down4))
        up1 = torch.cat((up1, down3), dim=1)

        up2 = self.prelu_up(self.deconv2_up(up1))
        up2 = torch.cat((up2, down2), dim=1)

        up3 = self.prelu_up(self.deconv3_up(up2))
        up3 = torch.cat((up3, down1), dim=1)

        up4 = self.prelu_up(self.deconv4_up(up3))


        out = self.conv_out(up4)

        return out
```

Listing 6.1: Generator Class

6.2 Discriminator Class

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        # Convolutional layers to process the face image
        self.conv = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
        )

        # Attribute processing
        self.fc = nn.Linear(18, 256)

        # Output layer
        self.out = nn.Conv2d(512, 1, kernel_size=4, stride=1, padding
    =0)

        # Additional layers for classification
        self.linear = nn.Linear(169, 13)
        self.linear2 = nn.Linear(13, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, face_image, attributes):

        out_img = self.conv(face_image)


        attr = self.fc(attributes)
        attr = attr.unsqueeze(2).unsqueeze(3)
        attr = attr.expand(attr.size(0), attr.size(1), out_img.size(2),
     out_img.size(3))


        combined = torch.cat([out_img, attr], dim=1)


        validity = self.out(combined)
        validity = validity.view(-1, 169)
        l1 = self.linear(validity)
        final = self.sigmoid(self.linear2(l1))
        return final
```

Listing 6.2: Discriminator Class

6.3 Training Function

```python
import torch.optim as optim
import os

CHECKPOINT_DIR = "/kaggle/working/"
checkpoint_interval = 1
os.makedirs(CHECKPOINT_DIR, exist_ok=True)

```

```
def train(generator, discriminator, dataloader, num_epochs=50, lr=0.0002):
    optimizer_G = optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))
    optimizer_D = optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))

    adversarial_loss = nn.BCELoss()
    # reconstruction_loss = nn.L1Loss() # Uncomment if using reconstruction loss

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    generator.to(device)
    discriminator.to(device)
    adversarial_loss.to(device)
    # reconstruction_loss.to(device)

    for epoch in range(num_epochs):
        for i, (sketches, attributes, real_faces) in enumerate(dataloader):
            batch_size = sketches.size(0)
            sketches = sketches.to(device)
            attributes = attributes.to(device)
            real_faces = real_faces.to(device)

            valid = torch.ones((batch_size, 1), requires_grad=False).to(device)
            fake = torch.zeros((batch_size, 1), requires_grad=False).to(device)

            #  Train Generator
            optimizer_G.zero_grad()

            gen_faces = generator(sketches, attributes)

            g_loss_adv = adversarial_loss(discriminator(gen_faces, attributes), valid)
            # g_loss_recon = reconstruction_loss(gen_faces, real_faces)

            g_loss = g_loss_adv #+ g_loss_recon
            g_loss.backward()
            optimizer_G.step()

            # Train Discriminator
            optimizer_D.zero_grad()

            real_loss = adversarial_loss(discriminator(real_faces, attributes), valid)
            fake_loss = adversarial_loss(discriminator(gen_faces.detach(), attributes), fake)

            d_loss = (real_loss + fake_loss) / 2
            d_loss.backward()
            optimizer_D.step()

            print(f"[Epoch {epoch+1}/{num_epochs}] [Batch {i+1}/{len(dataloader)}] [D loss: {d_loss.item():.4f}] [G loss: {g_loss.item()
:.4f}]")

        if (epoch + 1) % checkpoint_interval == 0:
            torch.save(generator.state_dict(), f"{CHECKPOINT_DIR}/
generator_epoch_{epoch+1}.pth")
            torch.save(discriminator.state_dict(), f"{CHECKPOINT_DIR}/
discriminator_epoch_{epoch+1}.pth")
            print(f"Checkpoint saved at epoch {epoch+1}.")
```

Listing 6.3: Training Function

6.4 Generator Class Explanation

The `Generator` class defines the architecture of the GAN's generator. It processes the input sketch and attributes to generate a photorealistic image. The architecture consists of:

- **Sketch Processing**: Convolutional layers (`conv1_A`, `conv2_A`, `conv3_A`) followed by PReLU activation to extract features from the sketch.
- **Attribute Processing**: A fully connected layer (`fc`) transforms the attribute vector, followed by convolutional layers (`conv1_B`, `conv2_B`, `conv3_B`) and PReLU activation to process attribute information.
- **Downsampling**: Convolutional layers (`conv1_down` to `conv4_down`) with stride 2 for spatial downsampling, followed by PReLU activation.
- **Upsampling**: Transposed convolutional layers (`deconv1_up` to `deconv4_up`) for spatial upsampling, concatenated with corresponding downsampled features to preserve spatial information, followed by PReLU activation.
- **Output Layer**: A final convolutional layer (`conv_out`) to produce the RGB image.

6.5 Discriminator Class Explanation

The `Discriminator` class defines the architecture of the GAN's discriminator. It evaluates the authenticity of the generated images. The architecture consists of:

- **Image Processing**: Sequential convolutional layers (`conv`) with LeakyReLU activation to extract features from the input image.
- **Attribute Processing**: A fully connected layer (`fc`) transforms the attribute vector, which is then reshaped and expanded to match the spatial dimensions of the image features.
- **Combination**: Concatenates image features with attribute features.
- **Output Layers**: A convolutional layer (`out`) followed by linear layers (`linear`, `linear2`) and a Sigmoid activation to produce the final validity score.

6.6 Training Function Explanation

The `train` function orchestrates the training process of the GAN, handling both generator and discriminator updates. Key components include:

- **Optimizers**: Adam optimizers for both generator and discriminator with specified learning rates and betas.
- **Loss Functions**: Binary Cross Entropy Loss (`adversarial_loss`) for adversarial training. Reconstruction loss is commented out but can be included for additional fidelity.
- **Training Loop**: Iterates over epochs and batches, performing the following steps:
    1. **Generator Training**:
        - Zero gradients.
        - Generate fake images using the generator.
        - Compute adversarial loss based on discriminator's assessment of fake images.
        - Backpropagate and update generator weights.
    2. **Discriminator Training**:
        - Zero gradients.
        - Compute loss on real images.
        - Compute loss on fake images (detach to avoid backpropagating through generator).
        - Backpropagate and update discriminator weights.
    3. **Checkpointing**: Save model states at specified intervals.

7. Conclusion

7.1 Summary

The sketch-to-image translation system employs GAN models to convert sketches into photorealistic images. The system's design emphasizes high-quality image generation, efficiency, and user-friendly interaction. Through rigorous testing and validation, the system ensures that the generated images meet the desired quality standards and effectively represent the input sketches.

7.2 Future Work

Future improvements could focus on enhancing the GAN model's ability to handle more complex attributes and integrating additional user feedback mechanisms to refine the image generation process further.

7.3 References