

# Multi-Style Image Generation using StyleGANs

Aarush Singh Kushwaha - 230019  
Anirudh Singh - 230142  
Fariz Ali - 230402  
Kshitij Bhardwaj - 230580

## Abstract

*This project introduces a simple and effective framework for multi-style image generation using a CycleGAN-based architecture with multiple types of attention mechanisms, namely Squeeze and Excitation, Convolutional Block Attention Modules(CBAM), Coordiante Attention and Multi Head Attention. The system is trained on balanced, pre-processed datasets of real faces (UTKFace, ages 18–120), each resized and normalized for stable GAN training. The model uses adversarial, cycle-consistency, and identity losses to enable unpaired translation between realistic and Ghibli-style faces, while also supporting plausible age manipulation. The codebase demonstrates robust training, clear visual improvements, and identity-preserving results, providing a practical solution for creative style and attribute transfer tasks in digital arts and entertainment.*

## 1. Introduction

Image-to-image translation stands as a cornerstone of contemporary computer vision, enabling the transformation of images across domains while preserving core semantic content. Recent advances in generative adversarial networks (GANs) and diffusion models have dramatically expanded the creative and practical horizons of this technology, making possible applications that were once the domain of science fiction. In this work, we present two independent, state-of-the-art systems that exemplify the research and engineering progress in this field: a robust age manipulation framework and a Studio Ghibli-inspired style conversion pipeline.

The first application, **age manipulation**, leverages a StyleGAN-based architecture [4]to perform realistic, identity-preserving age progression and regression on facial images. Users can upload a centered facial photograph and select either "old to young" or "young to old" transformation, receiving high-fidelity results in real time. This system is grounded in the latest research on disentangled latent space manipulation and age-conditioned generative model-

ing, reflecting the growing demand for controllable facial attribute editing in entertainment, forensics, and personalized digital experiences. Our approach is informed by recent literature that emphasizes fine-grained, continuous age transformation while rigorously maintaining subject identity, addressing key challenges highlighted in works such as SAM and AgeTransGAN.

The second application focuses on **Ghibli-style image generation**, utilizing an enhanced CycleGAN architecture with Coord(Coordinate Attention) blocks [2] to translate real facial images into the distinct, expressive style of Studio Ghibli animation. The model is trained on a **custom, Stable Diffusion-generated dataset** of Ghibli character faces, paired with carefully curated real face images. This system supports bidirectional translation, including the tentative capability to revert Ghibli-style images back to photorealistic face,a frontier rarely explored in prior work. The integration of Coord blocks is a deliberate architectural choice, enabling the model to recalibrate feature responses and better capture the nuanced textures and palettes characteristic of Ghibli art, as evidenced by recent advances in unpaired image translation.

Both systems are built on meticulously prepared datasets and rigorous data augmentation strategies, ensuring diversity and robustness in training. Preprocessing pipelines incorporate **Deepface** face detection and alignment, with all images standardized to **128x128 resolution** and normalized for stable GAN optimization. The training regimen, employs adversarial, cycle-consistency, and identity losses, but evaluation is purposefully centered on visual fidelity and user perception—a reflection of the ongoing discourse in generative modeling regarding the limitations of traditional quantitative metrics for subjective, creative outputs.

Our contributions are **twofold**: first, we provide deployable, user-facing applications that demonstrate the practical impact of generative modeling; second, we offer a research-centric analysis of architectural innovations and dataset curation, that can inform future work in multi-domain, multi-attribute image synthesis. These systems not only illustrate the current state of the art but also chart a path forward for

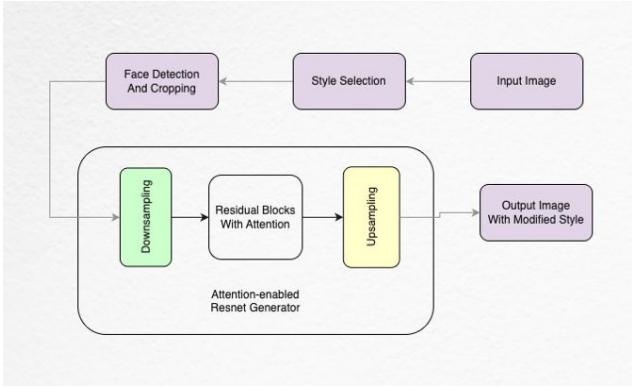


Figure 1. Proposed Pipeline

research at the intersection of AI, digital art, and human-centered design.

## 2. Background and Related Work

### 2.1. Evolution of Generative Adversarial Networks

Generative Adversarial Networks (GANs) have emerged as one of the most powerful tools in the field of image synthesis. GANs consist of two neural networks, **a generator and a discriminator**, that compete in a zero-sum game. The generator attempts to create images that look as realistic as possible, while the discriminator tries to distinguish between real and generated images. Through this adversarial process, the generator learns to produce increasingly convincing outputs.

Early GAN architectures, such as DCGAN (Deep Convolutional GAN) [1], introduced convolutional layers and batch normalization to improve training stability and image quality. However, GANs still struggled with issues like mode collapse, training instability, and low-resolution outputs.

Significant progress was made with the introduction of StyleGAN [4], which transformed the field of high-resolution image generation. StyleGAN introduced a style-based architecture that uses adaptive instance normalization (AdaIN) to inject latent style vectors into various layers of the generator. This allows for fine-grained control over visual attributes such as pose, texture, and facial features, and enables smooth interpolation in the disentangled latent space. As a result, users can manipulate specific aspects of an image (e.g., changing hair color or expression) without affecting other unrelated features.

In parallel, other GAN variants tackled different challenges. CycleGAN [7], for instance, addressed the problem of unpaired image-to-image translation enabling transformations between two domains (e.g., horses - zebras, summer - winter) without needing corresponding image pairs. This is achieved by enforcing a cycle-consistency

loss, ensuring that translating from domain A to B and back to A yields the original input. Another notable approach, Pix2Pix, focuses on paired datasets and uses a conditional GAN framework for tasks like image colorization and sketch-to-photo synthesis.

Recent advancements also include the integration of attention mechanisms and transformer-based architectures into GAN pipelines, further enhancing their ability to model global relationships and fine details. These innovations are paving the way for more robust, flexible, and interpretable generative models.

Together, these developments have made GANs central to many computer vision tasks, ranging from artistic style transfer and photo editing, to medical imaging and data augmentation.

### 2.2. Attention Mechanism

Attention mechanisms have become a fundamental component in modern neural network architectures. The core idea behind attention is to enable a model to focus on relevant parts of the input data while suppressing irrelevant information. In its basic form, an attention function maps a query and a set of key-value pairs to an output, where the output is computed as a weighted sum of the values, with the weight assigned to each value determined by a compatibility function between the query and the corresponding key.

The Transformer architecture, introduced in the paper "Attention Is All You Need" [5], popularized attention mechanisms by showing that complex sequence processing tasks could be handled using attention alone, without relying on recurrent or convolutional operations. The effectiveness of this approach stems from attention's ability to capture long-range dependencies regardless of their distance in the input sequence.

#### 2.2.1 Squeeze-and-Excitation (SE) Blocks

Building on the concept of attention, Squeeze-and-Excitation (SE) blocks [3] were introduced as a channel attention mechanism specifically for convolutional neural networks. SE blocks train upon inter dependencies between channels by using global information to selectively emphasize informative features.

The SE module operates in two steps:

1. **Squeeze:** Global average pooling is applied to compress spatial information into channel descriptors
2. **Excitation:** A simple gating mechanism with two fully-connected layers and a sigmoid activation function adaptively recalibrates channel-wise feature responses

By explicitly modeling channel interdependencies, SE blocks allow the network to boost useful features and suppress less useful ones, improving representation power with minimal computational overhead. This approach demonstrated significant improvements across various computer vision tasks including image classification.

### 2.2.2 Convolutional Block Attention Module (CBAM)

CBAM [6] extends the concept of attention further by incorporating both channel and spatial attention. While SE blocks focus solely on channel relationships, CBAM argues that both "what" (channel) and "where" (spatial) are important for enhancing feature representation.

CBAM sequentially applies two submodules:

- Channel attention module:** Unlike SE which uses only average-pooled features, CBAM leverages both average-pooled and max-pooled features to capture different aspects of channel information. These aggregated features are fed into a shared network to produce the channel attention map.
- Spatial attention module:** This module generates a spatial attention map by utilizing channel-pooled features (both average and max pooling along the channel axis) followed by a convolution operation.

The two attention maps are applied sequentially to refine the intermediate feature map. Extensive experiments on ImageNet classification, MS COCO object detection, and VOC 2007 detection demonstrate that CBAM consistently improves performance across various network architectures with minimal parameter increase.

### 2.2.3 Coordinate Attention

Coordinate Attention [2] represents a further evolution in attention mechanisms, addressing limitations of both channel and spatial attention. While channel attention (like SE) loses spatial information and standard spatial attention can be computationally expensive, Coordinate Attention efficiently encodes both channel and positional information.

Coordinate Attention decomposes the 2D spatial attention into two 1D feature encoding processes that capture the interdependencies between channels and spatial coordinates. By leveraging positional information along horizontal and vertical directions separately, it creates attention maps that maintain precise positional information while being computationally efficient.

This approach bridges the gap between channel and spatial attention mechanisms, providing a balanced solution that enhances representation power while preserving

position-sensitive information. The coordinate-wise attention design enables networks to selectively emphasize informative features with specific spatial locations, which is particularly beneficial for tasks requiring precise spatial understanding such as object detection and semantic segmentation.

Through this progression from basic attention to more sophisticated mechanisms like Coordinate Attention, we can observe a continuous refinement of how neural networks attend to and emphasize important information, leading to increasingly powerful visual representations with relatively small computational overhead.

## 3. Experimentation and Discussion

### 3.1. Ablation and Architectural Experiments

We systematically experimented with key architectural and training parameters:

- SE Block Ablation:** Removing Squeeze-and-Excitation blocks led to further degradation in identity preservation and increased noise, confirming their positive contribution to feature recalibration.
- Normalization Strategies:** Switching from instance normalization to batch or layer normalization resulted in either mode collapse or ghosting effects, reaffirming the choice of instance normalization for style transfer tasks.
- Loss Function Variations:** Increasing the cycle consistency loss weight improved structure but blurred stylistic features; perceptual loss based on VGG16 did not transfer well due to domain gap between natural and anime images.
- Progressive Growing:** Training with progressive image resolutions (from  $64 \times 64$  to  $128 \times 128$ ) improved texture synthesis at lower resolutions but introduced phase artifacts at higher scales.

### 3.2. Custom Dataset and Data Augmentation

To address the shortcomings of the large-scale anime dataset, we curated a custom dataset of 4,000 Ghibli-style faces using Stable Diffusion, paired with a filtered subset of real faces from UTKFace. Data augmentation (random flips, rotations, color jitter) was employed to increase diversity and mitigate overfitting.

**Results:** Training on the custom dataset yielded more visually consistent and stylistically faithful outputs, with improved color harmony and better preservation of facial structure. However, some artifacts persisted, particularly when translating challenging poses or low-light images.

### 3.3. Qualitative Evaluation and Failure Analysis

Throughout all experiments, evaluation relied primarily on visual inspection of input-output pairs, as GAN loss metrics did not correlate well with perceptual quality. Successes were most prominent on frontal, well-lit faces with neutral expressions. Failure cases included:

- Profile views or occluded faces, which led to disfigurements and loss of identity.
- Reverse translation (anime to real) occasionally resulted in uncanny, overly smooth faces.
- Low-light or noisy images, which the model misinterpreted as stylistic shading.

### 3.4. Training on Large-Scale AnimeFace Dataset

Our initial experiments utilized the publicly available AnimeFace dataset, comprising over 63,000 images, in conjunction with the UTKFace dataset for the real face domain. The expectation was that the model, with its CycleGAN backbone and SE-enhanced residual blocks, would learn a generalizable mapping between photorealistic and anime-style faces, producing visually coherent cartoonish outputs.

**Observations:** Despite extensive training and regular visual validation on custom and held-out images, the outputs were frequently unstable and unpredictable. Generated images often exhibited:

- Severe artifacts such as checkerboard patterns and color blotches.
- Anatomically implausible features, including distorted facial proportions and unnatural blending of human and anime characteristics.
- Over-saturation, loss of facial identity, and inconsistent stylistic transfer.

**Analysis:** These issues can be attributed to several factors:

- *Domain Mismatch:* The AnimeFace dataset features a wide variety of stylizations, facial poses, and artistic conventions, making it difficult for the model to learn a consistent style mapping.
- *Preprocessing Limitations:* Face detection and cropping algorithms, optimized for real faces, often failed on stylized anime faces, leading to poorly aligned training data.
- *Resolution Constraints:* Downsampling to  $128 \times 128$  pixels, while necessary for computational feasibility, led to loss of fine details crucial for both photorealism and anime aesthetics.

**Summary:** Our experimentation underscores the importance of domain-specific data curation, robust preprocessing, and architectural choices such as SE blocks and instance normalization. While the custom Ghibli-style dataset improved results, further work is needed to address failure cases and enhance generalization across diverse facial inputs.

## 4. Methodology and Implementation

### 4.1. System Architecture

Our framework is built on a dual-generator, dual-discriminator CycleGAN architecture tailored for robust, multi-style facial synthesis and age manipulation. This configuration enables unpaired image-to-image translation between photorealistic and Ghibli-style facial domains, as well as effective aging and de-aging transformations.

**Generator G\_A2B:** Transforms real (photographic) faces into anime/Ghibli-style faces.

**Generator G\_B2A:** Converts anime/Ghibli-style faces back to realistic faces.

**Discriminator D\_A:** Judges the realism of generated or real faces in the photographic domain.

**Discriminator D\_B:** Assesses the stylistic authenticity of faces in the anime/Ghibli domain.

Each generator is based on a ResNet-inspired architecture featuring:

1. An initial convolutional block (reflection padding, convolution, instance normalization, ReLU).
2. Two downsampling layers to increase feature depth.
3. Nine residual blocks with integrated Squeeze-and-Excitation (SE) modules for enhanced channel-wise attention.
4. Two upsampling layers for spatial reconstruction.
5. A final convolutional block with Tanh activation for output normalization.

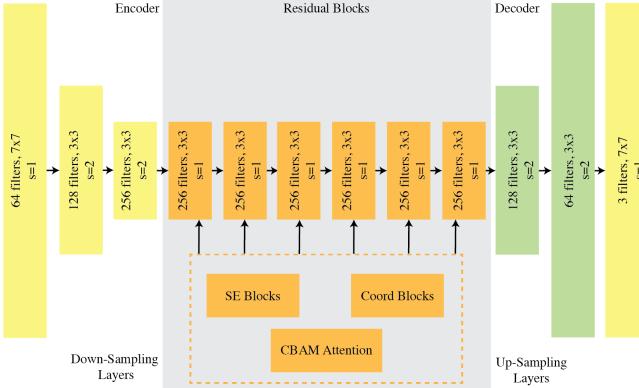


Figure 2. Attention Enabled ResNet generator

The discriminators adopt a PatchGAN-style CNN design, evaluating local image patches for domain authenticity, which is well established for stable and effective adversarial training in unpaired image translation tasks.

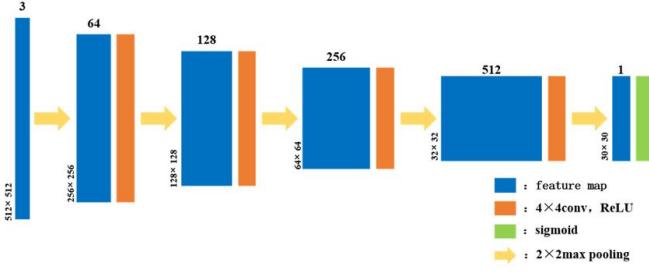


Figure 3. PatchGAN Discriminator Architecture

## 4.2. Loss Functions

We learn mappings between domains  $X$  and  $Y$  via generators  $G : X \rightarrow Y$  and  $F : Y \rightarrow X$ , and discriminators  $D_X, D_Y$ . These discriminators enforce that generated samples match the distribution of real images in the target domain.

### 4.2.1 Adversarial Loss

The adversarial loss for generator  $G$  and discriminator  $D_Y$  is defined as:

$$\begin{aligned} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = & \mathbb{E}_{y \sim p_{\text{data}}(y)} [\log D_Y(y)] \\ & + \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log(1 - D_Y(G(x)))] \end{aligned} \quad (1)$$

A similar loss is defined for  $F$  and  $D_X$ .

### 4.2.2 Cycle Consistency Loss

To ensure meaningful mappings, we enforce that translations are cycle-consistent:

$$\begin{aligned} \mathcal{L}_{\text{cyc}}(G, F) = & \mathbb{E}_{x \sim p_{\text{data}}(x)} [\|F(G(x)) - x\|_1] \\ & + \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|G(F(y)) - y\|_1] \end{aligned} \quad (2)$$

### 4.2.3 Full Objective

The overall objective combines both adversarial and cycle-consistency losses:

$$\begin{aligned} \mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{\text{cyc}}(G, F) \end{aligned} \quad (3)$$

Here,  $\lambda$  controls the relative importance of the cycle consistency loss.

## 4.3. Novel Approaches

### 4.3.1 Squeeze-and-Excitation (SE) Enhanced Residual Blocks

A key innovation in our architecture is the integration of Squeeze-and-Excitation (SE) blocks within each residual block of the generators. SE blocks perform dynamic, channel-wise feature recalibration, allowing the network to adaptively emphasize the most informative features for both style and age transformations. This is especially important in facial synthesis, where subtle cues (e.g., wrinkles, color gradients) are critical for photorealism and stylistic fidelity.

#### Mathematical Formulation:

$$\text{SE}(x) = x \cdot \sigma(W_2 \delta(W_1 \text{GAP}(x)))$$

where  $W_1, W_2$  are learned transformations,  $\text{GAP}$  denotes Global Average Pooling, (reduction ratio = 16),  $\delta = \text{ReLU}$ , and  $\sigma = \text{Sigmoid}$ .

```

1 class SEBlock(nn.Module):
2     def __init__(self, channel, reduction=16):
3         super(SEBlock, self).__init__()
4         self.fc = nn.Sequential(
5             nn.AdaptiveAvgPool2d(1), # Squeeze:
6             nn.Conv2d(channel, channel // reduction, 1),
7             nn.ReLU(inplace=True),
8             nn.Conv2d(channel // reduction, channel, 1),
9             nn.Sigmoid() # Excitation: channel
10            weights between 0 and 1
11        )
12
13    def forward(self, x):
14        weights = self.fc(x)
15        return x * weights # channel-wise
16        multiplication

```

#### **Justification:**

SE blocks have proven effective in both classification and generative tasks, improving the model's ability to focus on domain-specific details.

In our context, they allow the generator to better distinguish and preserve identity-relevant features during age transformation, and to capture stylistic nuances in Ghibli-style synthesis.

This approach is supported by recent literature demonstrating the benefits of attention and adaptive residual mechanisms in facial image generation and translation tasks.

#### **4.3.2 Instance Normalization**

We employ instance normalization throughout the architecture instead of batch normalization. Instance normalization is empirically superior for style transfer and unpaired image translation, as it normalizes each sample independently, preserving instance-specific style cues and supporting more stable GAN training.

#### **4.3.3 PatchGAN Discriminators**

Our discriminators are designed as PatchGANs, which classify overlapping **70×70** patches as real or fake. This local focus encourages the generator to produce realistic fine-grained details, a crucial property for both facial realism and stylization.

### **4.4. Dataset Engineering**

#### **4.4.1 Age Manipulation Domain**

Parameter	Value
Source	UTKFace
Total Images	4000
Age range	Young: 18 - 30 Old: 50-120
Resolution	128×128

#### **Preprocessing:**

Images are filtered by age, face-cropped using RetinaFace/Haar cascades, resized to **128×128**, and normalized to **[1,1]**.

**Augmentation:** Horizontal flip, rotation, and color jitter are applied to increase diversity, improve generalization, and mitigate overfitting.

#### **4.4.2 Ghibli-Style Domain**

#### **Preprocessing:**

Synthetic Ghibli-style faces are generated via Stable Diffusion, face-cropped, resized, and normalized identically to the real-face pipeline.

Parameter	Value
Source	Custom Ghibli Dataset (StableDiffusion)
Total Images	4000
Resolution	128×128

**Augmentation:** Identical to the real domain to ensure balanced training and robust translation.

#### **4.4.3 Data Loading**

Custom PyTorch Dataset classes and DataLoaders are used for efficient batch processing and shuffling, ensuring each mini-batch contains diverse samples from both domains

### **4.5. Training Strategy and Stability**

#### **4.5.1 Loss Functions**

- Adversarial Loss: Least-squares GAN loss for stable convergence.
- Cycle Consistency Loss ( $\lambda_{cycle}=10.0$ ): Ensures that translating an image to the other domain and back reconstructs the original.
- Identity Loss ( $\lambda_{identity}=5.0$ ): Encourages generators to preserve color composition and facial structure when the input is already in the target domain.

#### **4.5.2 Optimization**

- Adam optimizer, learning rate 0.0002,  $\beta_1 = 0.5$ ,  $\beta_2 = 0.999$
- Batch size: 6, Epochs: 500 (selected based on GPU memory and empirical stability).
- Weight Initialization: Normal initialization for convolutional layers to promote stable early training.

#### **4.5.3 Stability Enhancements**

- Instance normalization for all convolutional layers.
- Regular checkpointing and visual inspection every 10 epochs.
- Training on diverse, augmented data to prevent mode collapse.
- Visual validation on held-out, user-supplied images.

#### **4.5.4 Hardware**

Training distributed across Kaggle (P100 GPUs), Colab (T4 GPUs), and local machines, with a 30-hour compute cap per session.

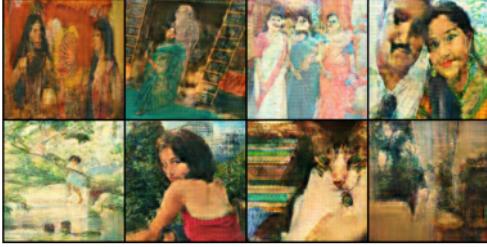


Figure 4. Ghibli Style transfer Results using SE-Block

## 4.6. Evaluation and Validation

### 4.6.1 Visual Evaluation

- Outputs are assessed by direct visual inspection, as GAN loss metrics do not reliably reflect perceptual quality in creative tasks.
- Identity preservation is verified using DeepFace for select transformations.
- Side-by-side input/output galleries and qualitative user feedback are used for model selection.

## 5. Results

### 5.1. Results for Ghibli using SE and coord blocks

From the results obtained in Figure 4 and Figure 5, we can visually observe that the outputs generated using Coordinate Attention are superior to those produced with Squeeze-and-Excitation (SE) blocks. Coordinate Attention effectively encodes both channel relationships and precise positional information, allowing the model to more accurately focus on important spatial regions and enhance feature localization. This leads to improved visual quality and more faithful style transfer in generated images.

In summary, adopting a more advanced attention mechanism such as Coordinate Attention yields clear improvements in image generation quality over both SE-based and attention-free architectures, demonstrating the value of integrating direction-aware and position-sensitive attention modules in generative models.

Attention Mechanism	Avg. Generator Loss	Epochs
SE	1.99	110
Coordinate Attention	1.817	134

Table 1. Comparison of average generator loss and epochs to convergence for models with SE and with coordinate attention.

### 5.2. Results of Young-Old model

Figure 6 and Figure 7 are the outputs on our own images generated from the demo application that we have deployed

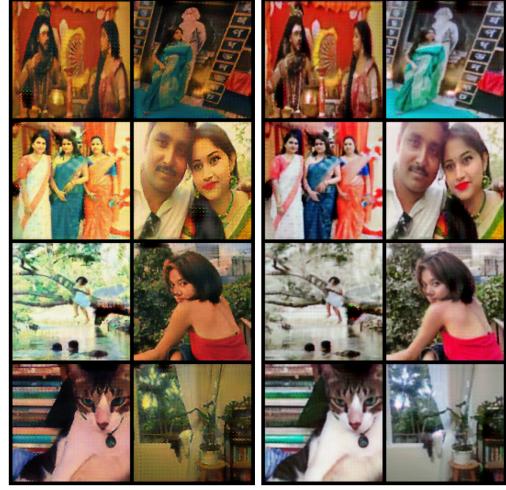


Figure 5. Ghibli style transfer Results using Coordinate Attention  
LEFT: Generated Ghibli images — Right: Reconstructed Original Images

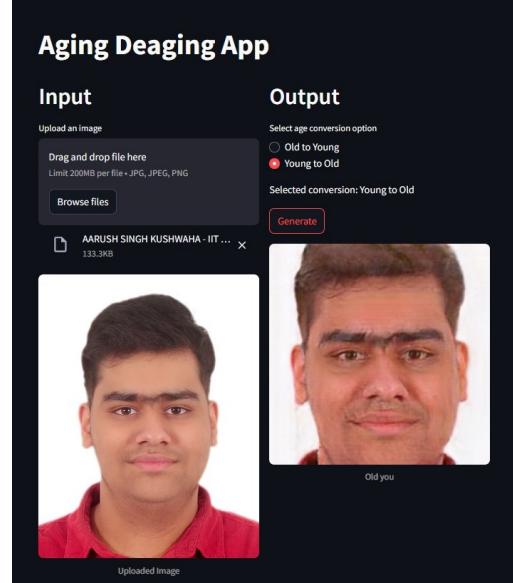


Figure 6. Sample image 1 generated using our pipeline

Attention Mechanism	Avg. Generator Loss	Epochs
None (Without Attention)	1.75	130
Coordinate Attention	1.05	66

Table 2. Comparison of average generator loss and epochs to convergence for models without attention and with coordinate attention.

## 6. Conclusion

In this project, we developed a comprehensive framework for multi-style image generation using advanced GAN

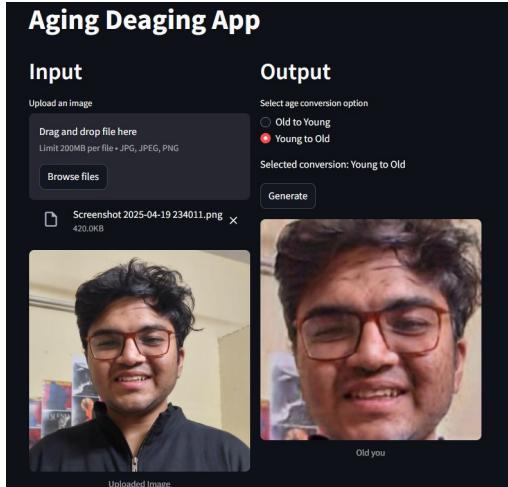


Figure 7. Sample Image 2 generated using our pipeline

architectures enhanced with multiple attention mechanisms, including Squeeze-and-Excitation, CBAM, and Coordinate Attention blocks.. Our system preserves important identity features while demonstrating strong unpaired image-to-image translation between real and Ghibli-style faces and efficient age manipulation. We addressed issues with domain mismatch, preprocessing constraints, and style fidelity by means of meticulous dataset curation, focused data augmentation, and innovative architectural designs. Although some artefacts persist in difficult situations like occluded or low-light images, qualitative evaluation demonstrated that our method yields visually convincing and identity-preserving results. Overall, our work highlights the potential of attention-augmented GANs for creative style and attribute transfer, and provides practical insights for future research at the intersection of generative modeling, digital art, and human-centered design

## 7. Appendix

### 7.1. Links to datasets

[UtkFace Dataset](#) [Ghibli Dataset](#)

### 7.2. Source Code

#### 7.2.1 Utk-Face Dataloader (Young-Old)

```

1
2
3 import os
4 import shutil
5 from PIL import Image
6
7 # Define paths
8 input_dir = '/kaggle/input/utkface-new/UTKFace'
# Replace with the path to your UTKFace
# dataset

```

```

9 young_dir = '/kaggle/working/Input/young' # Replace with the path where you want to save
# the young images
10 old_dir = '/kaggle/working/Input/old' # Replace
# with the path where you want to save the old
# images
11
12 # Create directories if they don't exist
13 os.makedirs(young_dir, exist_ok=True)
14 os.makedirs(old_dir, exist_ok=True)
15
16 # Age brackets
17 young_min_age, young_max_age = 18, 30
18 old_min_age, old_max_age = 50, 120
19
20 # Function to get age from filename
21 def get_age_from_filename(filename):
22     return int(filename.split('_')[0])
23
24 # Process images
25 for filename in os.listdir(input_dir):
26     if filename.endswith('.jpg'):
27         age = get_age_from_filename(filename)
28
29         if young_min_age <= age <= young_max_age:
30             if random.choice([0, 1]):
31                 shutil.copy(os.path.join(
32                     input_dir, filename), os.path.join(young_dir,
33                     filename))
34             elif old_min_age <= age <= old_max_age:
35                 if random.choice([0, 1]):
36                     shutil.copy(os.path.join(
37                         input_dir, filename), os.path.join(old_dir,
38                         filename))
39
40 print("Images have been successfully divided and
41 saved.")
42
43
44 import os
45 from PIL import Image
46 from torchvision import transforms, datasets
47 from torch.utils.data import Dataset, DataLoader
48
49
50 class CustomImageFolderDataset(Dataset):
51     def __init__(self, root_dir, transform=None):
52         self.root_dir = root_dir
53         self.transform = transform
54         self.images = os.listdir(root_dir)
55
56     def __len__(self):
57         return len(self.images)
58
59     def __getitem__(self, idx):
60         img_name = self.images[idx]
61         img_path = os.path.join(self.root_dir,
62                               img_name)
63         image = Image.open(img_path).convert('RGB')
64
65         if self.transform:
66             image = self.transform(image)
67
68         label = 0 if 'young' in self.root_dir
69         else 1 # Assign labels based on directory
# name

```

```

63     return image, label
64
65 transform = transforms.Compose([
66     transforms.Resize((128,128)),
67     transforms.ToTensor(),
68     transforms.Normalize((0.5, 0.5, 0.5), (0.5,
69                         0.5, 0.5)),
70 ])
71
72 young_dataset = CustomImageFolderDataset('/kaggle/
73     /working/Input/young', transform=transform)
74 old_dataset = CustomImageFolderDataset('/kaggle/
75     /working/Input/old', transform=transform)
76
77 # Create DataLoaders
78 batch_size = 6 # Adjust batch size as needed
79 young_loader = DataLoader(young_dataset,
80     batch_size=batch_size, shuffle=True)
81 old_loader = DataLoader(old_dataset, batch_size=
82     batch_size, shuffle=True)
83
84 # Example usage of the loaders
85 for inputs, labels in young_loader:
86     # Process inputs and labels as needed
87     print(f"Young batch shape: {inputs.shape},
88         Labels: {labels}")
89     break
90
91 for inputs, labels in old_loader:
92     # Process inputs and labels as needed
93     print(f"Old batch shape: {inputs.shape},
94         Labels: {labels}")
95     break

```

```

24             split_path = os.path.join(base_dir,
25                 split)
26             if not os.path.exists(split_path):
27                 continue
28
29             for folder in os.listdir(split_path):
30                 folder_path = os.path.join(
31                     split_path, folder)
32                 if os.path.isdir(folder_path):
33                     img_path = os.path.join(
34                         folder_path, f'{image_type}.png')
35                     if os.path.exists(img_path):
36                         self.samples.append(
37                             img_path)
38
39             def __len__(self):
40                 return len(self.samples)
41
42             def __getitem__(self, idx):
43                 img_path = self.samples[idx]
44                 image = Image.open(img_path).convert('RGB')
45
46                 if self.transform:
47                     image = self.transform(image)
48
49                 return image, self.label
50
51             transform = transforms.Compose([
52                 transforms.Resize((140, 140)),
53                     # Slightly upscale for cropping
54                 transforms.RandomCrop((128, 128)),
55                     # Random crop to desired size
56                 transforms.RandomHorizontalFlip(),
57                     # Flip images with 0.5 prob
58                 transforms.RandomRotation(15),
59                     # Rotate images randomly within 15 degrees
60                 transforms.ColorJitter(
61                     # Randomly change brightness/contrast/etc.
62                     brightness=0.2, contrast=0.2, saturation
63                     =0.2, hue=0.1
64                 ),
65                 transforms.ToTensor(),
66                 transforms.Normalize((0.5,), (0.5,))
67             ])
68
69             transform_2 = transforms.Compose([
70                 transforms.Resize((128,128)),
71                 transforms.ToTensor(),
72                 transforms.Normalize((0.5, 0.5, 0.5), (0.5,
73                         0.5, 0.5)),
74             ])
75
76             # Datasets
77             g_dataset = GhibliImageTypeDataset('/kaggle/input/
78                 /ghibli-dataset/Ghibli', image_type='g',
79                 transform=transform)
80             o_dataset = GhibliImageTypeDataset('/kaggle/input/
81                 /ghibli-dataset/Ghibli', image_type='o',
82                 transform=transform)
83
84             # DataLoaders
85             batch_size = 6
86             g_loader = DataLoader(g_dataset, batch_size=
87                 batch_size, shuffle=True)
88             o_loader = DataLoader(o_dataset, batch_size=
89                 batch_size, shuffle=True)

```

## 7.2.2 Ghibli Dataloader

```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torch.autograd import Variable
5 import itertools
6 import random
7
8 device = torch.device("cuda" if torch.cuda.
9     is_available() else "cpu")
10 import os
11 from PIL import Image
12 from torchvision import transforms, datasets
13 from torch.utils.data import Dataset, DataLoader
14
15 class GhibliImageTypeDataset(Dataset):
16     def __init__(self, base_dir, image_type='g',
17         transform=None):
18
19         """"
20             image_type: either 'g' or 'o' to pick
21             specific images.
22             """
23             self.transform = transform
24             self.samples = []
25             self.label = 1 if image_type == 'g' else
26
27             0
28
29             for split in ['training', 'validation']:
30
31                 if split == 'training':
32                     g_dataset = GhibliImageTypeDataset('/kaggle/input/
33                         /ghibli-dataset/Ghibli', image_type='g',
34                         transform=transform)
35                     o_dataset = GhibliImageTypeDataset('/kaggle/input/
36                         /ghibli-dataset/Ghibli', image_type='o',
37                         transform=transform)
38
39                     # DataLoaders
40                     batch_size = 6
41                     g_loader = DataLoader(g_dataset, batch_size=
42                         batch_size, shuffle=True)
43                     o_loader = DataLoader(o_dataset, batch_size=
44                         batch_size, shuffle=True)
45
46                     self.samples += list(itertools.zip_longest(
47                         g_loader, o_loader))
48
49             else:
50
51                 g_dataset = GhibliImageTypeDataset('/kaggle/input/
52                     /ghibli-dataset/Ghibli', image_type='g',
53                     transform=transform)
54
55                 o_dataset = GhibliImageTypeDataset('/kaggle/input/
56                     /ghibli-dataset/Ghibli', image_type='o',
57                     transform=transform)
58
59                 # DataLoaders
60                 batch_size = 6
61                 g_loader = DataLoader(g_dataset, batch_size=
62                     batch_size, shuffle=True)
63                 o_loader = DataLoader(o_dataset, batch_size=
64                     batch_size, shuffle=True)
65
66                 self.samples += list(itertools.zip_longest(
67                     g_loader, o_loader))
68
69             self.label = 0
70
71             self.transform = transform
72             self.image_type = image_type
73
74             self.samples = list(itertools.zip_longest(
75                 g_loader, o_loader))
76
77             self.label = 1 if image_type == 'g' else
78
79             0
80
81             for sample in self.samples:
82
83                 image, label = sample
84
85                 if self.image_type == 'g':
86                     if label == 1:
87                         self.samples.append((image, label))
88
89                 else:
90                     if label == 0:
91                         self.samples.append((image, label))
92
93             self.length = len(self.samples)
94
95             self.g_dataset = g_dataset
96             self.o_dataset = o_dataset
97
98             self.g_loader = g_loader
99             self.o_loader = o_loader
100
101             self.batch_size = batch_size
102
103             self.shuffle = shuffle
104
105             self.transform = transform
106
107             self.image_type = image_type
108
109             self.label = label
110
111             self.samples = samples
112
113             self.length = length
114
115             self.g_dataset = g_dataset
116             self.o_dataset = o_dataset
117
118             self.g_loader = g_loader
119             self.o_loader = o_loader
120
121             self.batch_size = batch_size
122
123             self.shuffle = shuffle
124
125             self.transform = transform
126
127             self.image_type = image_type
128
129             self.label = label
130
131             self.samples = samples
132
133             self.length = length
134
135             self.g_dataset = g_dataset
136             self.o_dataset = o_dataset
137
138             self.g_loader = g_loader
139             self.o_loader = o_loader
140
141             self.batch_size = batch_size
142
143             self.shuffle = shuffle
144
145             self.transform = transform
146
147             self.image_type = image_type
148
149             self.label = label
150
151             self.samples = samples
152
153             self.length = length
154
155             self.g_dataset = g_dataset
156             self.o_dataset = o_dataset
157
158             self.g_loader = g_loader
159             self.o_loader = o_loader
160
161             self.batch_size = batch_size
162
163             self.shuffle = shuffle
164
165             self.transform = transform
166
167             self.image_type = image_type
168
169             self.label = label
170
171             self.samples = samples
172
173             self.length = length
174
175             self.g_dataset = g_dataset
176             self.o_dataset = o_dataset
177
178             self.g_loader = g_loader
179             self.o_loader = o_loader
180
181             self.batch_size = batch_size
182
183             self.shuffle = shuffle
184
185             self.transform = transform
186
187             self.image_type = image_type
188
189             self.label = label
190
191             self.samples = samples
192
193             self.length = length
194
195             self.g_dataset = g_dataset
196             self.o_dataset = o_dataset
197
198             self.g_loader = g_loader
199             self.o_loader = o_loader
200
201             self.batch_size = batch_size
202
203             self.shuffle = shuffle
204
205             self.transform = transform
206
207             self.image_type = image_type
208
209             self.label = label
210
211             self.samples = samples
212
213             self.length = length
214
215             self.g_dataset = g_dataset
216             self.o_dataset = o_dataset
217
218             self.g_loader = g_loader
219             self.o_loader = o_loader
220
221             self.batch_size = batch_size
222
223             self.shuffle = shuffle
224
225             self.transform = transform
226
227             self.image_type = image_type
228
229             self.label = label
230
231             self.samples = samples
232
233             self.length = length
234
235             self.g_dataset = g_dataset
236             self.o_dataset = o_dataset
237
238             self.g_loader = g_loader
239             self.o_loader = o_loader
240
241             self.batch_size = batch_size
242
243             self.shuffle = shuffle
244
245             self.transform = transform
246
247             self.image_type = image_type
248
249             self.label = label
250
251             self.samples = samples
252
253             self.length = length
254
255             self.g_dataset = g_dataset
256             self.o_dataset = o_dataset
257
258             self.g_loader = g_loader
259             self.o_loader = o_loader
260
261             self.batch_size = batch_size
262
263             self.shuffle = shuffle
264
265             self.transform = transform
266
267             self.image_type = image_type
268
269             self.label = label
270
271             self.samples = samples
272
273             self.length = length
274
275             self.g_dataset = g_dataset
276             self.o_dataset = o_dataset
277
278             self.g_loader = g_loader
279             self.o_loader = o_loader
280
281             self.batch_size = batch_size
282
283             self.shuffle = shuffle
284
285             self.transform = transform
286
287             self.image_type = image_type
288
289             self.label = label
290
291             self.samples = samples
292
293             self.length = length
294
295             self.g_dataset = g_dataset
296             self.o_dataset = o_dataset
297
298             self.g_loader = g_loader
299             self.o_loader = o_loader
300
301             self.batch_size = batch_size
302
303             self.shuffle = shuffle
304
305             self.transform = transform
306
307             self.image_type = image_type
308
309             self.label = label
310
311             self.samples = samples
312
313             self.length = length
314
315             self.g_dataset = g_dataset
316             self.o_dataset = o_dataset
317
318             self.g_loader = g_loader
319             self.o_loader = o_loader
320
321             self.batch_size = batch_size
322
323             self.shuffle = shuffle
324
325             self.transform = transform
326
327             self.image_type = image_type
328
329             self.label = label
330
331             self.samples = samples
332
333             self.length = length
334
335             self.g_dataset = g_dataset
336             self.o_dataset = o_dataset
337
338             self.g_loader = g_loader
339             self.o_loader = o_loader
340
341             self.batch_size = batch_size
342
343             self.shuffle = shuffle
344
345             self.transform = transform
346
347             self.image_type = image_type
348
349             self.label = label
350
351             self.samples = samples
352
353             self.length = length
354
355             self.g_dataset = g_dataset
356             self.o_dataset = o_dataset
357
358             self.g_loader = g_loader
359             self.o_loader = o_loader
360
361             self.batch_size = batch_size
362
363             self.shuffle = shuffle
364
365             self.transform = transform
366
367             self.image_type = image_type
368
369             self.label = label
370
371             self.samples = samples
372
373             self.length = length
374
375             self.g_dataset = g_dataset
376             self.o_dataset = o_dataset
377
378             self.g_loader = g_loader
379             self.o_loader = o_loader
380
381             self.batch_size = batch_size
382
383             self.shuffle = shuffle
384
385             self.transform = transform
386
387             self.image_type = image_type
388
389             self.label = label
390
391             self.samples = samples
392
393             self.length = length
394
395             self.g_dataset = g_dataset
396             self.o_dataset = o_dataset
397
398             self.g_loader = g_loader
399             self.o_loader = o_loader
400
401             self.batch_size = batch_size
402
403             self.shuffle = shuffle
404
405             self.transform = transform
406
407             self.image_type = image_type
408
409             self.label = label
410
411             self.samples = samples
412
413             self.length = length
414
415             self.g_dataset = g_dataset
416             self.o_dataset = o_dataset
417
418             self.g_loader = g_loader
419             self.o_loader = o_loader
420
421             self.batch_size = batch_size
422
423             self.shuffle = shuffle
424
425             self.transform = transform
426
427             self.image_type = image_type
428
429             self.label = label
430
431             self.samples = samples
432
433             self.length = length
434
435             self.g_dataset = g_dataset
436             self.o_dataset = o_dataset
437
438             self.g_loader = g_loader
439             self.o_loader = o_loader
440
441             self.batch_size = batch_size
442
443             self.shuffle = shuffle
444
445             self.transform = transform
446
447             self.image_type = image_type
448
449             self.label = label
450
451             self.samples = samples
452
453             self.length = length
454
455             self.g_dataset = g_dataset
456             self.o_dataset = o_dataset
457
458             self.g_loader = g_loader
459             self.o_loader = o_loader
460
461             self.batch_size = batch_size
462
463             self.shuffle = shuffle
464
465             self.transform = transform
466
467             self.image_type = image_type
468
469             self.label = label
470
471             self.samples = samples
472
473             self.length = length
474
475             self.g_dataset = g_dataset
476             self.o_dataset = o_dataset
477
478             self.g_loader = g_loader
479             self.o_loader = o_loader
480
481             self.batch_size = batch_size
482
483             self.shuffle = shuffle
484
485             self.transform = transform
486
487             self.image_type = image_type
488
489             self.label = label
490
491             self.samples = samples
492
493             self.length = length
494
495             self.g_dataset = g_dataset
496             self.o_dataset = o_dataset
497
498             self.g_loader = g_loader
499             self.o_loader = o_loader
500
501             self.batch_size = batch_size
502
503             self.shuffle = shuffle
504
505             self.transform = transform
506
507             self.image_type = image_type
508
509             self.label = label
510
511             self.samples = samples
512
513             self.length = length
514
515             self.g_dataset = g_dataset
516             self.o_dataset = o_dataset
517
518             self.g_loader = g_loader
519             self.o_loader = o_loader
520
521             self.batch_size = batch_size
522
523             self.shuffle = shuffle
524
525             self.transform = transform
526
527             self.image_type = image_type
528
529             self.label = label
530
531             self.samples = samples
532
533             self.length = length
534
535             self.g_dataset = g_dataset
536             self.o_dataset = o_dataset
537
538             self.g_loader = g_loader
539             self.o_loader = o_loader
540
541             self.batch_size = batch_size
542
543             self.shuffle = shuffle
544
545             self.transform = transform
546
547             self.image_type = image_type
548
549             self.label = label
550
551             self.samples = samples
552
553             self.length = length
554
555             self.g_dataset = g_dataset
556             self.o_dataset = o_dataset
557
558             self.g_loader = g_loader
559             self.o_loader = o_loader
560
561             self.batch_size = batch_size
562
563             self.shuffle = shuffle
564
565             self.transform = transform
566
567             self.image_type = image_type
568
569             self.label = label
570
571             self.samples = samples
572
573             self.length = length
574
575             self.g_dataset = g_dataset
576             self.o_dataset = o_dataset
577
578             self.g_loader = g_loader
579             self.o_loader = o_loader
580
581             self.batch_size = batch_size
582
583             self.shuffle = shuffle
584
585             self.transform = transform
586
587             self.image_type = image_type
588
589             self.label = label
590
591             self.samples = samples
592
593             self.length = length
594
595             self.g_dataset = g_dataset
596             self.o_dataset = o_dataset
597
598             self.g_loader = g_loader
599             self.o_loader = o_loader
600
601             self.batch_size = batch_size
602
603             self.shuffle = shuffle
604
605             self.transform = transform
606
607             self.image_type = image_type
608
609             self.label = label
610
611             self.samples = samples
612
613             self.length = length
614
615             self.g_dataset = g_dataset
616             self.o_dataset = o_dataset
617
618             self.g_loader = g_loader
619             self.o_loader = o_loader
620
621             self.batch_size = batch_size
622
623             self.shuffle = shuffle
624
625             self.transform = transform
626
627             self.image_type = image_type
628
629             self.label = label
630
631             self.samples = samples
632
633             self.length = length
634
635             self.g_dataset = g_dataset
636             self.o_dataset = o_dataset
637
638             self.g_loader = g_loader
639             self.o_loader = o_loader
640
641             self.batch_size = batch_size
642
643             self.shuffle = shuffle
644
645             self.transform = transform
646
647             self.image_type = image_type
648
649             self.label = label
650
651             self.samples = samples
652
653             self.length = length
654
655             self.g_dataset = g_dataset
656             self.o_dataset = o_dataset
657
658             self.g_loader = g_loader
659             self.o_loader = o_loader
660
661             self.batch_size = batch_size
662
663             self.shuffle = shuffle
664
665             self.transform = transform
666
667             self.image_type = image_type
668
669             self.label = label
670
671             self.samples = samples
672
673             self.length = length
674
675             self.g_dataset = g_dataset
676             self.o_dataset = o_dataset
677
678             self.g_loader = g_loader
679             self.o_loader = o_loader
680
681             self.batch_size = batch_size
682
683             self.shuffle = shuffle
684
685             self.transform = transform
686
687             self.image_type = image_type
688
689             self.label = label
690
691             self.samples = samples
692
693             self.length = length
694
695             self.g_dataset = g_dataset
696             self.o_dataset = o_dataset
697
698             self.g_loader = g_loader
699             self.o_loader = o_loader
700
701             self.batch_size = batch_size
702
703             self.shuffle = shuffle
704
705             self.transform = transform
706
707             self.image_type = image_type
708
709             self.label = label
710
711             self.samples = samples
712
713             self.length = length
714
715             self.g_dataset = g_dataset
716             self.o_dataset = o_dataset
717
718             self.g_loader = g_loader
719             self.o_loader = o_loader
720
721             self.batch_size = batch_size
722
723             self.shuffle = shuffle
724
725             self.transform = transform
726
727             self.image_type = image_type
728
729             self.label = label
730
731             self.samples = samples
732
733             self.length = length
734
735             self.g_dataset = g_dataset
736             self.o_dataset = o_dataset
737
738             self.g_loader = g_loader
739             self.o_loader = o_loader
740
741             self.batch_size = batch_size
742
743             self.shuffle = shuffle
744
745             self.transform = transform
746
747             self.image_type = image_type
748
749             self.label = label
750
751             self.samples = samples
752
753             self.length = length
754
755             self.g_dataset = g_dataset
756             self.o_dataset = o_dataset
757
758             self.g_loader = g_loader
759             self.o_loader = o_loader
760
761             self.batch_size = batch_size
762
763             self.shuffle = shuffle
764
765             self.transform = transform
766
767             self.image_type = image_type
768
769             self.label = label
770
771             self.samples = samples
772
773             self.length = length
774
775             self.g_dataset = g_dataset
776             self.o_dataset = o_dataset
777
778             self.g_loader = g_loader
779             self.o_loader = o_loader
780
781             self.batch_size = batch_size
782
783             self.shuffle = shuffle
784
785             self.transform = transform
786
787             self.image_type = image_type
788
789             self.label = label
790
791             self.samples = samples
792
793             self.length = length
794
795             self.g_dataset = g_dataset
796             self.o_dataset = o_dataset
797
798             self.g_loader = g_loader
799             self.o_loader = o_loader
800
801             self.batch_size = batch_size
802
803             self.shuffle = shuffle
804
805             self.transform = transform
806
807             self.image_type = image_type
808
809             self.label = label
810
811             self.samples = samples
812
813             self.length = length
814
815             self.g_dataset = g_dataset
816             self.o_dataset = o_dataset
817
818             self.g_loader = g_loader
819             self.o_loader = o_loader
820
821             self.batch_size = batch_size
822
823             self.shuffle = shuffle
824
825             self.transform = transform
826
827             self.image_type = image_type
828
829             self.label = label
830
831             self.samples = samples
832
833             self.length = length
834
835             self.g_dataset = g_dataset
836             self.o_dataset = o_dataset
837
838             self.g_loader = g_loader
839             self.o_loader = o_loader
840
841             self.batch_size = batch_size
842
843             self.shuffle = shuffle
844
845             self.transform = transform
846
847             self.image_type = image_type
848
849             self.label = label
850
851             self.samples = samples
852
853             self.length = length
854
855             self.g_dataset = g_dataset
856             self.o_dataset = o_dataset
857
858             self.g_loader = g_loader
859             self.o_loader = o_loader
860
861             self.batch_size = batch_size
862
863             self.shuffle = shuffle
864
865             self.transform = transform
866
867             self.image_type = image_type
868
869             self.label = label
870
871             self.samples = samples
872
873             self.length = length
874
875             self.g_dataset = g_dataset
876             self.o_dataset = o_dataset
877
878             self.g_loader = g_loader
879             self.o_loader = o_loader
880
881             self.batch_size = batch_size
882
883             self.shuffle = shuffle
884
885             self.transform = transform
886
887             self.image_type = image_type
888
889             self.label = label
890
891             self.samples = samples
892
893             self.length = length
894
895             self.g_dataset = g_dataset
896             self.o_dataset = o_dataset
897
898             self.g_loader = g_loader
899             self.o_loader = o_loader
900
901             self.batch_size = batch_size
902
903             self.shuffle = shuffle
904
905             self.transform = transform
906
907             self.image_type = image_type
908
909             self.label = label
910
911             self.samples = samples
912
913             self.length = length
914
915             self.g_dataset = g_dataset
916             self.o_dataset = o_dataset
917
918             self.g_loader = g_loader
919             self.o_loader = o_loader
920
921             self.batch_size = batch_size
922
923             self.shuffle = shuffle
924
925             self.transform = transform
926
927             self.image_type = image_type
928
929             self.label = label
930
931             self.samples = samples
932
933             self.length = length
934
935             self.g_dataset = g_dataset
936             self.o_dataset = o_dataset
937
938             self.g_loader = g_loader
939             self.o_loader = o_loader
940
941             self.batch_size = batch_size
942
943             self.shuffle = shuffle
944
945             self.transform = transform
946
947             self.image_type = image_type
948
949             self.label = label
950
951             self.samples = samples
952
953             self.length = length
954
955             self.g_dataset = g_dataset
956             self.o_dataset = o_dataset
957
958             self.g_loader = g_loader
959             self.o_loader = o_loader
960
961             self.batch_size = batch_size
962
963             self.shuffle = shuffle
964
965             self.transform = transform
966
967             self.image_type = image_type
968
969             self.label = label
970
971             self.samples = samples
972
973             self.length = length
974
975             self.g_dataset = g_dataset
976             self.o_dataset = o_dataset
977
978             self.g_loader = g_loader
979             self.o_loader = o_loader
980
981             self.batch_size = batch_size
982
983             self.shuffle = shuffle
984
985             self.transform = transform
986
987             self.image_type = image_type
988
989             self.label = label
990
991             self.samples = samples
992
993             self.length = length
994
995             self.g_dataset = g_dataset
996             self.o_dataset = o_dataset
997
998             self.g_loader = g_loader
999             self.o_loader = o_loader
1000
1001             self.batch_size = batch_size
1002
1003             self.shuffle = shuffle
1004
1005             self.transform = transform
1006
1007             self.image_type = image_type
1008
1009             self.label = label
1010
1011             self.samples = samples
1012
1013             self.length = length
1014
1015             self.g_dataset = g_dataset
1016             self.o_dataset = o_dataset
1017
1018             self.g_loader = g_loader
1019             self.o_loader = o_loader
1020
1021             self.batch_size = batch_size
1022
1023             self.shuffle = shuffle
1024
1025             self.transform = transform
1026
1027             self.image_type = image_type
1028
1029             self.label = label
1030
1031             self.samples = samples
1032
1033             self.length = length
1034
1035             self.g_dataset = g_dataset
1036             self.o_dataset = o_dataset
1037
1038             self.g_loader = g_loader
1039             self.o_loader = o_loader
1040
1041             self.batch_size = batch_size
1042
1043             self.shuffle = shuffle
1044
1045             self.transform = transform
1046
1047             self.image_type = image_type
1048
1049             self.label = label
1050
1051             self.samples = samples
1052
1053             self.length = length
1054
1055             self.g_dataset = g_dataset
1056             self.o_dataset = o_dataset
1057
1058             self.g_loader = g_loader
1059             self.o_loader = o_loader
1060
1061             self.batch_size = batch_size
1062
1063             self.shuffle = shuffle
1064
1065             self.transform = transform
1066
1067             self.image_type = image_type
1068
1069             self.label = label
1070
1071             self.samples = samples
1072
1073             self.length = length
1074
1075             self.g_dataset = g_dataset
1076             self.o_dataset = o_dataset
1077
1078             self.g_loader = g_loader
1079             self.o_loader = o_loader
1080
1081             self.batch_size = batch_size
1082
1083             self.shuffle = shuffle
1084
1085             self.transform = transform
1086
1087             self.image_type = image_type
1088
1089             self.label = label
1090
1091             self.samples = samples
1092
1093             self.length = length
1094
1095             self.g_dataset = g_dataset
1096             self.o_dataset = o_dataset
1097
1098             self.g_loader = g_loader
1099             self.o_loader = o_loader
1100
1101             self.batch_size = batch_size
1102
1103             self.shuffle = shuffle
1104
1105             self.transform = transform
1106
1107             self.image_type = image_type
1108
1109             self.label = label
1110
1111             self.samples = samples
1112
1113             self.length = length
1114
1115             self.g_dataset = g_dataset
1116             self.o_dataset = o_dataset
1117
1118             self.g_loader = g_loader
1119             self.o_loader = o_loader
1120
1121             self.batch_size = batch_size
1122
1123             self.shuffle = shuffle
1124
1125             self.transform = transform
1126
1127             self.image_type = image_type
1128
1129             self.label = label
1130
1131             self.samples = samples
1132
1133             self.length = length
1134
1135             self.g_dataset = g_dataset
1136             self.o_dataset = o_dataset
1137
1138             self.g_loader = g_loader
1139             self.o_loader = o_loader
1140
1141             self.batch_size = batch_size
1142
1143             self.shuffle = shuffle
1144
1145             self.transform = transform
1146
1147             self.image_type = image_type
1148
1149             self.label = label
1150
1151             self.samples = samples
1152
1153             self.length = length
1154
1155             self.g_dataset = g_dataset
1156             self.o_dataset = o_dataset
1157
1158             self.g_loader = g_loader
1159             self.o_loader = o_loader
1160
1161             self.batch_size = batch_size
1162
1163             self.shuffle = shuffle
1164
1165             self.transform = transform
1166
1167             self.image_type = image_type
1168
1169             self.label = label
1170
1171             self.samples = samples
1172
1173             self.length = length
1174
1175             self.g_dataset = g_dataset
1176             self.o_dataset = o_dataset
1177
1178             self.g_loader = g_loader
1179             self.o_loader = o_loader
1180
1181             self.batch_size = batch_size
1182
1183             self.shuffle = shuffle
1184
1185             self.transform = transform
1186
1187             self.image_type = image_type
1188
1189             self.label = label
1190
1191             self.samples = samples
1192
1193             self.length = length
1194
1195             self.g_dataset = g_dataset
1196             self.o_dataset = o_dataset
1197
1198             self.g_loader = g_loader
1199             self.o_loader = o_loader
1200
1201             self.batch_size = batch_size
1202
1203             self.shuffle = shuffle
1204
1205             self.transform = transform
1206
1207             self.image_type = image_type
1208
1209             self.label = label
1210
1211             self.samples = samples
1212
1213             self.length = length
1214
1215             self.g_dataset = g_dataset
1216             self.o_dataset = o_dataset
1217
1218             self.g_loader = g_loader
1219             self.o_loader = o_loader
1220
1221             self.batch_size = batch_size
1222
1223             self.shuffle = shuffle
1224
1225             self.transform = transform
1226
1227             self.image_type = image_type
1228
1229             self.label = label
1230
1231             self.samples = samples
1232
1233             self.length = length
1234
1235             self.g_dataset = g_dataset
1236             self.o_dataset = o_dataset
1237
1238             self.g_loader = g_loader
1239             self.o_loader = o_loader
1240
1241             self.batch_size = batch_size
1242
1243             self.shuffle = shuffle
1244
1245             self.transform = transform
1246
1247             self.image_type = image_type
1248
1249             self.label = label
1250
1251             self.samples = samples
1252
1253             self.length = length
1254
1255             self.g_dataset = g_dataset
1256             self.o_dataset = o_dataset
1257
1258             self.g_loader = g_loader
1259             self.o_loader = o_loader
1260
1261             self.batch_size = batch_size
1262
1263             self.shuffle = shuffle
1264
1265             self.transform = transform
1266
1267             self.image_type = image_type
1268
1269             self.label = label
1270
1271             self.samples = samples
1272
1273             self.length = length
1274
1275             self.g_dataset = g_dataset
1276             self.o_dataset = o_dataset
1277
1278             self.g_loader = g_loader
1279             self.o_loader = o_loader
1280
1281             self.batch_size = batch_size
1282
1283             self.shuffle = shuffle
1284
1285             self.transform = transform
1286
1287             self.image_type = image_type
1288
1289             self.label = label
1290
1291             self.samples = samples
1292
1293             self.length = length
1294
1295             self.g_dataset = g_dataset
1296             self.o_dataset = o_dataset
1297
1298             self.g_loader = g_loader
1299             self.o_loader = o_loader
1300
1301             self.batch_size = batch_size
1302
1303             self.shuffle = shuffle
1304
1305             self.transform = transform
1306
1307             self.image_type = image_type
1308
1309             self.label = label
1310
1311             self.samples = samples
1312
1313             self.length = length
1314
1315             self.g_dataset = g_dataset
1316             self.o_dataset = o_dataset
1317
1318             self.g_loader = g_loader
1319             self.o_loader = o_loader
1320
1321             self.batch_size = batch_size
1322
1323             self.shuffle = shuffle
1324
1325             self.transform = transform
1326
1327             self.image_type = image_type
1328
1329             self.label = label
1330
1331             self.samples = samples
1332
1333             self.length = length
1334
1335             self.g_dataset = g_dataset
1336             self.o_dataset = o_dataset
1337
1338             self.g_loader = g_loader
1339             self.o_loader = o_loader
1340
1341             self.batch_size = batch_size
1342
1343             self.shuffle = shuffle
1344
1345             self.transform = transform
1346
1347             self
```

```

73 # Sample usage
74 for inputs, labels in g_loader:
75     print(f"G image batch shape: {inputs.shape},
76           Labels: {labels}")
77     break
78
79 for inputs, labels in o_loader:
80     print(f'O image batch shape: {inputs.shape},
81           Labels: {labels}")
82     break

```

### 7.2.3 Helper Functions

```

1 import numpy as np
2 # Adversarial ground truths
3 def get_adversarial_targets(batch_size, device,
4     output_shape):
5     valid = torch.ones((batch_size, *output_shape),
6         device=device, requires_grad=False)
6     fake = torch.zeros((batch_size, *output_shape),
7         device=device, requires_grad=False)
8     return valid, fake
9
10 def weights_init_normal(m):
11     if isinstance(m, nn.Conv2d):
12         if m.weight is not None:
13             nn.init.normal_(m.weight.data, 0.0,
14                             0.02)
15         if m.bias is not None:
16             nn.init.constant_(m.bias.data, 0.0)
17
18     elif isinstance(m, nn.BatchNorm2d):
19         if m.weight is not None:
20             nn.init.normal_(m.weight.data, 1.0,
21                             0.02)
22         if m.bias is not None:
23             nn.init.constant_(m.bias.data, 0.0)

```

### 7.2.4 SE-Block

```

1 class SEBlock(nn.Module):
2     def __init__(self, channel, reduction=16):
3         super(SEBlock, self).__init__()
4         self.fc = nn.Sequential(
5             nn.AdaptiveAvgPool2d(1), # Squeeze:
6             nn.Conv2d(channel, channel // reduction, 1),
7             nn.ReLU(inplace=True),
8             nn.Conv2d(channel // reduction, channel, 1),
9             nn.Sigmoid() # Excitation: channel
10            weights between 0 and 1
11        )
12
13    def forward(self, x):
14        weights = self.fc(x)
15        return x * weights

```

### 7.2.5 CBAM-Blocks(along with its helper functions)

```

1 import torch
2 import math
3 import torch.nn as nn
4 import torch.nn.functional as F
5
6 class BasicConv(nn.Module):
7     def __init__(self, in_planes, out_planes,
8         kernel_size, stride=1, padding=0, dilation=1,
9         groups=1, relu=True, bn=True, bias=False):
10        super(BasicConv, self).__init__()
11        self.out_channels = out_planes
12        self.conv = nn.Conv2d(in_planes,
13            out_planes, kernel_size=kernel_size, stride=
14            stride, padding=padding, dilation=dilation,
15            groups=groups, bias=bias)
16        self.bn = nn.BatchNorm2d(out_planes, eps=1
17            e-5, momentum=0.01, affine=True) if bn else
18            None
19        self.relu = nn.ReLU() if relu else None
20
21    def forward(self, x):
22        x = self.conv(x)
23        if self.bn is not None:
24            x = self.bn(x)
25        if self.relu is not None:
26            x = self.relu(x)
27        return x
28
29 class Flatten(nn.Module):
30     def forward(self, x):
31         return x.view(x.size(0), -1)
32
33 class ChannelGate(nn.Module):
34     def __init__(self, gate_channels,
35         reduction_ratio=16, pool_types=['avg', 'max'
36         ]):
37         super(ChannelGate, self).__init__()
38         self.gate_channels = gate_channels
39         self.mlp = nn.Sequential(
40             Flatten(),
41             nn.Linear(gate_channels,
42             gate_channels // reduction_ratio),
43             nn.ReLU(),
44             nn.Linear(gate_channels //
45             reduction_ratio, gate_channels)
46         )
47         self.pool_types = pool_types
48
49     def forward(self, x):
50         channel_att_sum = None
51         for pool_type in self.pool_types:
52             if pool_type=='avg':
53                 avg_pool = F.avg_pool2d( x, (x.
54                 size(2), x.size(3)), stride=(x.size(2), x.
55                 size(3)))
56                 channel_att_raw = self.mlp(
57                 avg_pool )
58             elif pool_type=='max':
59                 max_pool = F.max_pool2d( x, (x.
60                 size(2), x.size(3)), stride=(x.size(2), x.
61                 size(3)))
62                 channel_att_raw = self.mlp(
63                 max_pool )
64             elif pool_type=='lp':
65                 lp_pool = F.lp_pool2d( x, 2, (x.
66                 size(2), x.size(3)), stride=(x.size(2), x.
67                 size(3)))

```

```

    size(3)))
        channel_att_raw = self.mlp(
lp_pool )
    elif pool_type=='lse':
        # LSE pool only
        lse_pool = logsumexp_2d(x)
        channel_att_raw = self.mlp(
lse_pool )

    if channel_att_sum is None:
        channel_att_sum = channel_att_raw
    else:
        channel_att_sum = channel_att_sum
+ channel_att_raw

    scale = F.sigmoid( channel_att_sum ).unsqueeze(2).unsqueeze(3).expand_as(x)
    return x * scale

def logsumexp_2d(tensor):
    tensor_flatten = tensor.view(tensor.size(0),
tensor.size(1), -1)
    s, _ = torch.max(tensor_flatten, dim=2,
keepdim=True)
    outputs = s + (tensor_flatten - s).exp().sum(
dim=2, keepdim=True).log()
    return outputs

class ChannelPool(nn.Module):
    def forward(self, x):
        return torch.cat( (torch.max(x,1)[0].unsqueeze(1), torch.mean(x,1).unsqueeze(1)), dim=1 )

class SpatialGate(nn.Module):
    def __init__(self):
        super(SpatialGate, self).__init__()
        kernel_size = 7
        self.compress = ChannelPool()
        self.spatial = BasicConv(2, 1,
kernel_size, stride=1, padding=(kernel_size
-1) // 2, relu=False)
    def forward(self, x):
        x_compress = self.compress(x)
        x_out = self.spatial(x_compress)
        scale = F.sigmoid(x_out) # broadcasting
        return x * scale

class CBAM(nn.Module):
    def __init__(self, gate_channels,
reduction_ratio=16, pool_types=['avg', 'max'],
no_spatial=False):
        super(CBAM, self).__init__()
        self.ChannelGate = ChannelGate(
gate_channels, reduction_ratio, pool_types)
        self.no_spatial=no_spatial
        if not no_spatial:
            self.SpatialGate = SpatialGate()
    def forward(self, x):
        x_out = self.ChannelGate(x)
        if not self.no_spatial:
            x_out = self.SpatialGate(x_out)
        return x_out

```

## 7.2.6 Coordinate Attention Blocks(along with helper functions)

```

1 import torch
2 import torch.nn as nn
3 import math
4 import torch.nn.functional as F
5
6 class h_sigmoid(nn.Module):
7     def __init__(self, inplace=True):
8         super(h_sigmoid, self).__init__()
9         self.relu = nn.ReLU6(inplace=inplace)
10
11    def forward(self, x):
12        return self.relu(x + 3) / 6
13
14 class h_swish(nn.Module):
15    def __init__(self, inplace=True):
16        super(h_swish, self).__init__()
17        self.sigmoid = h_sigmoid(inplace=inplace)
18
19    def forward(self, x):
20        return x * self.sigmoid(x)
21
22 class CoordAtt(nn.Module):
23    def __init__(self, inp, oup, reduction=32):
24        super(CoordAtt, self).__init__()
25        self.pool_h = nn.AdaptiveAvgPool2d((None,
1))
        self.pool_w = nn.AdaptiveAvgPool2d((1,
None))
26
27        mip = max(8, inp // reduction)
28
29        self.conv1 = nn.Conv2d(inp, mip,
kernel_size=1, stride=1, padding=0)
        self.bn1 = nn.BatchNorm2d(mip)
        self.act = h_swish()
30
31        self.conv_h = nn.Conv2d(mip, oup,
kernel_size=1, stride=1, padding=0)
        self.conv_w = nn.Conv2d(mip, oup,
kernel_size=1, stride=1, padding=0)
32
33
34    def forward(self, x):
35        identity = x
36
37        n, c, h, w = x.size()
        x_h = self.pool_h(x)
        x_w = self.pool_w(x).permute(0, 1, 3, 2)
38
39        y = torch.cat([x_h, x_w], dim=2)
        y = self.conv1(y)
        y = self.bn1(y)
        y = self.act(y)
40
41        x_h, x_w = torch.split(y, [h, w], dim=2)
        x_w = x_w.permute(0, 1, 3, 2)
42
43        a_h = self.conv_h(x_h).sigmoid()
        a_w = self.conv_w(x_w).sigmoid()
44
45        out = identity * a_w * a_h
46
47        return out
48
49
50
51
52
53
54
55
56
57
58

```

## 7.2.7 Common Resnet Blocks

```

1 class ResnetBlock(nn.Module):
2     def __init__(self, dim, reduction=16):
3         super(ResnetBlock, self).__init__()
4         self.conv_block = self.build_conv_block(dim)
5
6         self.attn = CoordAtt(dim, dim, reduction)
7         # or SEBlock(dim, reduction) or CBAM(dim,
8         reduction) accordingly
9
10    def build_conv_block(self, dim):
11        conv_block = [
12            nn.ReflectionPad2d(1),
13            nn.Conv2d(dim, dim, kernel_size=3,
14                      padding=0),
15            nn.InstanceNorm2d(dim),
16            nn.ReLU(True),
17            nn.ReflectionPad2d(1),
18            nn.Conv2d(dim, dim, kernel_size=3,
19                      padding=0),
20            nn.InstanceNorm2d(dim)
21        ]
22        return nn.Sequential(*conv_block)
23
24    def forward(self, x):
25        out = self.conv_block(x)
26        out = self.attn(out)
27        return x + out

```

```

28             model += [nn.ConvTranspose2d(
29                 in_features, out_features, 3, stride=2,
30                 padding=1, output_padding=1),
31                         nn.InstanceNorm2d(
32                             out_features),
33                         nn.ReLU(inplace=True)]
34             in_features = out_features
35             out_features = in_features // 2
36
37             # Output layer
38             model += [nn.ReflectionPad2d(3),
39                         nn.Conv2d(64, output_nc, 7),
40                         nn.Tanh()]
41
42             self.model = nn.Sequential(*model)
43
44     def forward(self, x):
45         return self.model(x)
46
47     # Discriminator model
48     class Discriminator(nn.Module):
49         def __init__(self, input_nc):
50             super(Discriminator, self).__init__()
51
52             model = [nn.Conv2d(input_nc, 64, 4,
53                               stride=2, padding=1),
54                     nn.LeakyReLU(0.2, inplace=True)]
55
56             model += [nn.Conv2d(64, 128, 4, stride=2,
57                               padding=1),
58                     nn.InstanceNorm2d(128),
59                     nn.LeakyReLU(0.2, inplace=True)
60             ]
61
62             model += [nn.Conv2d(128, 256, 4, stride
63                               =2, padding=1),
64                     nn.InstanceNorm2d(256),
65                     nn.LeakyReLU(0.2, inplace=True)
66             ]
67
68             model += [nn.Conv2d(256, 512, 4, stride
69                               =1, padding=1),
70                     nn.InstanceNorm2d(512),
71                     nn.LeakyReLU(0.2, inplace=True)
72             ]
73
74             model += [nn.Conv2d(512, 1, 4, stride=1,
75                               padding=1)]
76
77             self.model = nn.Sequential(*model)
78
79     def forward(self, x):
80         return self.model(x)

```

## 7.2.8 Generator and Discriminator Architecture

```

1 class GeneratorResNet(nn.Module):
2     def __init__(self, input_nc, output_nc,
3                  n_residual_blocks=14):
4         super(GeneratorResNet, self).__init__()
5
6         # Initial convolution block
7         model = [nn.ReflectionPad2d(3),
8                  nn.Conv2d(input_nc, 64, 7),
9                  nn.InstanceNorm2d(64),
10                 nn.ReLU(inplace=True)]
11
12         # Downsampling
13         in_features = 64
14         out_features = in_features * 2
15         for _ in range(2):
16             model += [nn.Conv2d(in_features,
17                               out_features, 3, stride=2, padding=1),
18                         nn.InstanceNorm2d(
19                             out_features),
20                         nn.ReLU(inplace=True)]
21             in_features = out_features
22             out_features = in_features * 2
23
24         # Residual blocks
25         for _ in range(n_residual_blocks):
26             model += [ResnetBlock(in_features)]
27
28         # Upsampling
29         out_features = in_features // 2
30         for _ in range(2):
31

```

## 7.2.9 Training Loop

```

1 import torchvision
2 num_epochs = 500
3 def imshow(img):
4     npimg = img.numpy()
5     plt.imshow(np.transpose(npimg, (1, 2, 0)))
6     plt.axis('off')
7     plt.show()

```

```

9 # Training Loop
10 for epoch in range(num_epochs):
11     shown = 0
12     if (epoch%2):
13         torch.save(netG_A2B.state_dict(), '/kaggle/working/netG_A2B.pth')
14         torch.save(netG_B2A.state_dict(), '/kaggle/working/netG_B2A.pth')
15         torch.save(netD_A.state_dict(), '/kaggle/working/netD_A.pth')
16         torch.save(netD_B.state_dict(), '/kaggle/working/netD_B.pth')
17     for i, (young_img, old_img) in enumerate(zip(o_loader, g_loader)):
18         # Set model input
19         real_A = Variable(young_img[0].to(device))
20
21         real_B = Variable(old_img[0].to(device))
22         # real_A = Variable(batch[0][batch[1]==0].to(device))
23         # real_B = Variable(batch[0][batch[1]==1].to(device))
24
25         # Adversarial ground truths
26         # valid = Variable(torch.Tensor(np.ones((real_A.size(0), *D_A.output_shape[1:])).to(device), requires_grad=False)
27         # fake = Variable(torch.Tensor(np.zeros((real_A.size(0), *D_A.output_shape[1:])).to(device), requires_grad=False)
28         output_shape = netD_A(real_A).shape[1:]
29         valid_A, invalid_A =
30         get_adversarial_targets(real_A.size(0),
31         device, output_shape)
32         valid_B, invalid_B =
33         get_adversarial_targets(real_B.size(0),
34         device, output_shape)
35         # -----
36         # Train Generators
37         # -----
38         # fake_A = fake_A.expand(-1, 3, -1, -1)
39         # print(invalid_A.shape)
40         # print(valid_A.shape)
41         # print(netD_A(fake_A.detach()).shape)
42         optimizer_G.zero_grad()
43
44         # Identity loss
45         loss_id_A = criterion_identity(netG_B2A(
46             real_A), real_A)
47         loss_id_B = criterion_identity(netG_A2B(
48             real_B), real_B)
49
50         loss_identity = (loss_id_A + loss_id_B) /
51             2
52
53         # GAN loss
54         fake_B = netG_A2B(real_A)
55         loss_GAN_A2B = criterion_GAN(netD_B(
56             fake_B), valid_B)
57         fake_A = netG_B2A(real_B)
58         loss_GAN_B2A = criterion_GAN(netD_A(
59             fake_A), valid_A)
60
61         loss_GAN = (loss_GAN_A2B + loss_GAN_B2A) /
62             2
63
64         # Cycle loss
65
66         recovered_A = netG_B2A(fake_B)
67         loss_cycle_A = criterion_cycle(
68             recovered_A, real_A)
69         recovered_B = netG_A2B(fake_A)
70         loss_cycle_B = criterion_cycle(
71             recovered_B, real_B)
72
73         loss_cycle = (loss_cycle_A + loss_cycle_B) / 2
74
75         # Total loss
76         loss_G = loss_GAN + 10.0 * loss_cycle +
77             5.0 * loss_identity
78
79         loss_G.backward()
80         optimizer_G.step()
81
82         # -----
83         # Train Discriminator A
84         # -----
85
86         optimizer_D_A.zero_grad()
87
88         # Real loss
89         loss_real = criterion_GAN(netD_A(real_A),
90             valid_A)
91         # Fake loss (on batch of previously
92         # generated samples)
93         loss_fake = criterion_GAN(netD_A(fake_A),
94             detach(), invalid_A)
95         # Total loss
96         loss_D_A = (loss_real + loss_fake) / 2
97
98         loss_D_A.backward()
99         optimizer_D_A.step()
100
101         # -----
102         # Train Discriminator B
103         # -----
104
105         optimizer_D_B.zero_grad()
106
107         # Real loss
108         loss_real = criterion_GAN(netD_B(real_B),
109             valid_B)
110         # Fake loss (on batch of previously
111         # generated samples)
112         loss_fake = criterion_GAN(netD_B(fake_B),
113             detach(), invalid_B)
114         # Total loss
115         loss_D_B = (loss_real + loss_fake) / 2
116
117         loss_D_B.backward()
118         optimizer_D_B.step()
119
120         # -----
121         # Log Progress
122         # -----
123         if epoch % 10 == 0 and epoch != 0 and
124             shown == 0:
125             # Save model checkpoints
126             torch.save(netG_A2B.state_dict(), f'/kaggle/working/netG_A2B_epoch{epoch}.pth')
127             torch.save(netG_B2A.state_dict(), f'/kaggle/working/netG_B2A_epoch{epoch}.pth')
128             torch.save(netD_A.state_dict(), f'/kaggle/working/netD_A_epoch{epoch}.pth')

```

```

107         torch.save(netD_B.state_dict(), f'/kaggle/working/netD_B_epoch{epoch}.pth')
108         if (i%100==0):
109             print(f"[Epoch {epoch+1}/{num_epochs}] [Batch {i+1}/{len(g_loader)}] "
110                   f"[D loss: {loss_D_A.item() + loss_D_B.item()}] "
111                   f"[G loss: {loss_G.item()}, adv: {loss_GAN.item()}, cycle: {loss_cycle.item()}], identity: {loss_identity.item()}]")

```

## References

- [1] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: [1406.2661 \[stat.ML\]](https://arxiv.org/abs/1406.2661). URL: <https://arxiv.org/abs/1406.2661>.
- [2] Qibin Hou, Daquan Zhou, and Jiashi Feng. *Coordinate Attention for Efficient Mobile Network Design*. 2021. arXiv: [2103.02907 \[cs.CV\]](https://arxiv.org/abs/2103.02907). URL: <https://arxiv.org/abs/2103.02907>.
- [3] Jie Hu et al. *Squeeze-and-Excitation Networks*. 2019. arXiv: [1709.01507 \[cs.CV\]](https://arxiv.org/abs/1709.01507). URL: <https://arxiv.org/abs/1709.01507>.
- [4] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: [1812.04948 \[cs.NE\]](https://arxiv.org/abs/1812.04948). URL: <https://arxiv.org/abs/1812.04948>.
- [5] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: [1706.03762 \[cs.CL\]](https://arxiv.org/abs/1706.03762). URL: <https://arxiv.org/abs/1706.03762>.
- [6] Sanghyun Woo et al. *CBAM: Convolutional Block Attention Module*. 2018. arXiv: [1807.06521 \[cs.CV\]](https://arxiv.org/abs/1807.06521). URL: <https://arxiv.org/abs/1807.06521>.
- [7] Jun-Yan Zhu et al. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. arXiv: [1703.10593 \[cs.CV\]](https://arxiv.org/abs/1703.10593). URL: <https://arxiv.org/abs/1703.10593>.