Project:Artistic Image Transformation in Ghibli Aesthetic

Team: FUSION EYES
Members: Kanan Pandit(B2430051), Partha Mete(B2430052)

April 28, 2025

1 Introduction

1.1 Background

Artistic style transfer has become an important application of deep learning, enabling the transformation of real-world images into different artistic representations. One of the major challenges in this field is learning mappings between two visual domains without needing paired datasets, particularly when transferring complex artistic styles like those seen in Studio Ghibli films.

1.2 Motivation

The unique, hand-drawn aesthetic of Studio Ghibli's works presents a fascinating challenge for image translation models. Capturing the subtle color palettes, smooth textures, and emotive expressiveness of Ghibli-style animations offers an opportunity to push the boundaries of AI-driven artistic generation.

1.3 Objectives

This project aims to:

- Transform real-world photographs into Studio Ghibli-style images.
- Utilize CycleGAN architecture to achieve unpaired image-to-image translation.
- Preserve important content features (e.g., facial structures) while adapting the artistic style.
- Provide qualitative evaluations of model outputs during and after training.

2 Literature Review

Recent research has demonstrated the power of Generative Adversarial Networks (GANs) for artistic applications:

- **CycleGAN** (Zhu et al., 2017) introduced unpaired image-to-image translation using cycle consistency loss to overcome the lack of paired training data.
- **pix2pix** (Isola et al., 2017) focused on paired translation but was less applicable in domains lacking paired examples.
- **UGATIT** (Kim et al., 2020) enhanced attention-based transformations between two domains with adaptive instance normalization.
- **StyleGAN** (Karras et al., 2019) and its successors improved high-resolution synthesis but were not primarily designed for domain translation tasks.

Our project adopts CycleGAN as it is well-suited for unpaired real-to-Ghibli translation, offering a lightweight, reliable, and proven framework.

3 Dataset Description

The Real to Ghibli Image Dataset contains 5,000 high-quality images. It consists of two folders: trainA with 2,500 real-world images, and trainB ghibli with 2,500 Ghibli-style artistic images.

3.1 Source

- **Dataset Source**: Sourced from Kaggle, link: Kaggle - Real to Ghibli Image Dataset

3.2 Features

3.2.1 Image Content (trainA):

- Human portraits
- Natural landscapes
- Rivers
- Mountains
- Forests
- City architecture
- Vehicles

3.2.2 Image Content (trainB_ghibli):

- Hand-drawn style backgrounds
- Fantasy landscapes
- Animated character scenes

3.3 Target Variable

- The target is not a specific label or value, but rather the generation of Ghibli-style images from real-world photographs through domain translation.

3.4 Dataset Characteristics

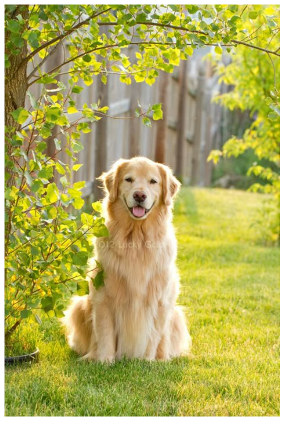
- **The images in trainA and trainB ghibli are independent collections, meaning that they do not represent paired images (i.e., there is no one-to-one correspondence between images in trainA and those in trainB ghibli). Instead, each set represents a distinct domain: real-world imagery in trainA and artistic, animated scenes in trainB ghibli. This unpaired nature offers flexibility for tasks like style transfer and image translation.All images are processed and maintained in high resolution.**

3.5 Dataset Sample

3.5.1 Sample Images from TrainA and TrainB

(a) TrainA Sample 1

(b) TrainA Sample 2


(c) TrainA Sample 3

(d) TrainB Sample 1

(e) TrainB Sample 2

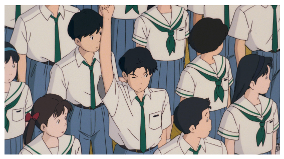

(f) TrainB Sample 3

Figure 1: Sample Images from Training Sets: TrainA (Real Images) and TrainB (Ghibli Images)

4 Data Preprocessing

4.1 Datasets Used

The project uses a custom dataset created for real-to-Ghibli style image translation, structured into two separate folders:

- **trainA**:
  - Contains 2,500 real-world photographs.
  - Includes various subjects such as portraits, landscapes, nature scenes, urban environments, etc.
- **trainB_ghibli**:
  - Contains 2,500 Ghibli-style artistic images.
  - Stylized animations inspired by the aesthetic of Studio Ghibli films, emphasizing soft colors, detailed backgrounds, and hand-drawn textures.

This pairing of datasets enables the CycleGAN model to learn mappings between real-world imagery and Ghibli-style artwork.

4.2 Custom Dataset Class

A custom `CustomImageDataset` class is implemented to handle data loading efficiently:

Folder Structure: Automatically loads all images ending with `.jpg`, `.jpeg`, or `.png` from the specified directory.

Data Transformations Applied: Each image undergoes the following preprocessing steps to enhance model generalization:

Step	**Transformation**	**Purpose**
1	Resize (to 286×286)	Ensures uniform size across all input images
2	Random Crop (to 256×256)	Introduces randomness and improves robustness
3	Random Horizontal Flip	Augments dataset and prevents overfitting
4	ToTensor	Converts images to PyTorch tensors
5	Normalization (to $[-1, 1]$)	Stabilizes GAN training

4.3 DataLoader Settings

- **Batch Size:** Set to 1, following CycleGAN best practices to ensure diverse and uncorrelated updates per training step.
- **Shuffling:** Enabled to randomize the order of data at each epoch and improve learning dynamics.

5 Methodology

5.1 Algorithms Used

- **Generators (G_AB and G_BA)**: Transform images between real and Ghibli domains.
- **Discriminators (D_A and D_B)**: Classify whether images are real or generated.

5.1.1 Losss Function

- **Adversarial Loss**: Mean Squared Error (MSE) Loss.
- **Cycle Consistency Loss**: L1 Loss.
- **Identity Loss**: L1 Loss.

5.2 Justification

5.2.1 Why CycleGAN?

Traditional image-to-image translation methods like Pix2Pix require paired datasets — meaning each input image must have an exact corresponding target image. However, in many real-world applications (such as artistic style transfer), it is extremely difficult or impossible to collect such paired data.

CycleGAN is designed specifically for unpaired image-to-image translation. Instead of requiring matching image pairs, CycleGAN uses a **cycle consistency loss** to ensure that translating an image from one domain to another and back results in the original image. This innovative idea allows CycleGAN to learn meaningful mappings between two domains without paired supervision.

Given that our dataset (Real $\rightarrow$ Ghibli images) consists of independent sets of real-world photographs and Ghibli-style artworks without any explicit correspondence, CycleGAN is the perfect choice for this task.

5.2.2 Why Implement from Scratch?

Instead of using pretrained models or existing CycleGAN libraries, we chose to implement CycleGAN from scratch to achieve the following educational and technical goals:

1.Deep Understanding of Adversarial Training GANs are notoriously difficult to train due to the dynamic between the generator and discriminator.

Implementing from scratch allowed us to experience firsthand the challenges in balancing generator and discriminator loss, avoiding mode collapse, and ensuring stable training.

2.Insight into Stability Challenges Training GANs is inherently unstable.

Issues such as vanishing gradients, oscillating loss curves, and generator overfitting were directly observed and tackled.

We experimented with various techniques such as using instance normalization, adjusting learning rates, and tuning Adam optimizer hyperparameters to stabilize training.

3.Hyperparameter Effects We manually set and experimented with hyperparameters such as:

- Learning rates for generators and discriminators.
- Weightings for cycle consistency and identity losses.
- Optimizer momentum terms (β_1, β_2 values in Adam).

This direct exposure helped us understand how small changes in hyperparameters can dramatically affect GAN training outcomes.

5.3 Model Architecture

5.3.1 Generator Architecture (G_{AB} and G_{BA})

The Generator network follows an encoder–ResNet–decoder structure. It uses reflection padding to reduce edge artifacts, instance normalization for style flexibility, and ReLU non-linearities.

Layer-wise Flow:

- **ReflectionPad2d:** Padding of 3 pixels on all sides.
- **Conv2d:** (in_channels=3, out_channels=64, kernel_size=7, stride=1, padding=0)
- **InstanceNorm2d:** 64 features.
- **ReLU:** In-place activation.
- **Conv2d:** (in_channels=64, out_channels=128, kernel_size=3, stride=2, padding=1)
- **InstanceNorm2d:** 128 features.
- **ReLU:** In-place activation.
- **Conv2d:** (in_channels=128, out_channels=256, kernel_size=3, stride=2, padding=1)
- **InstanceNorm2d:** 256 features.
- **ReLU:** In-place activation.
- **6 $\times$ ResidualBlock:** Each block contains:
  - **ReflectionPad2d:** Padding of 1.
  - **Conv2d:** (in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=0)

  - **InstanceNorm2d:** 256 features.
  - **ReLU:** In-place activation.
  - **ReflectionPad2d:** Padding of 1.
  - **Conv2d:** (in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=0)
  - **InstanceNorm2d:** 256 features.

- **ConvTranspose2d:** (in_channels=256, out_channels=128, kernel_size=3, stride=2, padding=1, output_padding=1)
- **InstanceNorm2d:** 128 features.
- **ReLU:** In-place activation.
- **ConvTranspose2d:** (in_channels=128, out_channels=64, kernel_size=3, stride=2, padding=1, output_padding=1)
- **InstanceNorm2d:** 64 features.
- **ReLU:** In-place activation.
- **ReflectionPad2d:** Padding of 3.
- **Conv2d:** (in_channels=64, out_channels=3, kernel_size=7, stride=1, padding=0)
- **Tanh:** Activation to output pixel values in $[-1, 1]$.

5.3.2 Discriminator Architecture (D_A and D_B)

The Discriminator network is a PatchGAN classifier (70×70 receptive field) that classifies local patches of an image as real or fake, promoting high-frequency detail preservation.

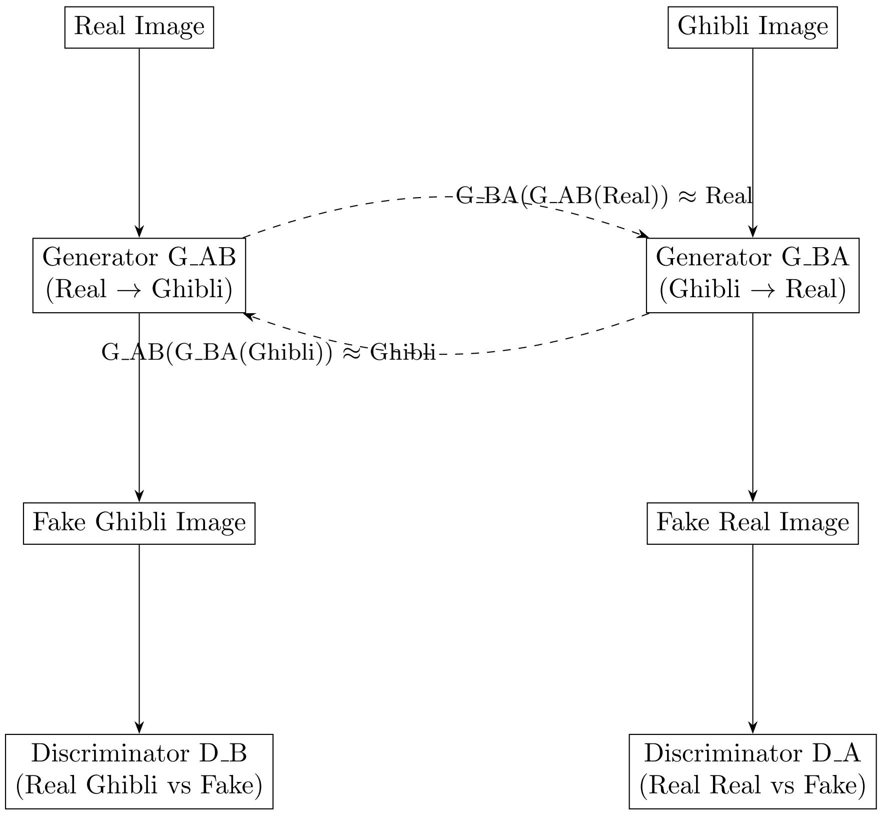
Layer-wise Flow:

- **Conv2d:** (in_channels=3, out_channels=64, kernel_size=4, stride=2, padding=1)
- **LeakyReLU:** Negative slope=0.2
- **Conv2d:** (in_channels=64, out_channels=128, kernel_size=4, stride=2, padding=1)
- **InstanceNorm2d:** 128 features.
- **LeakyReLU:** Negative slope=0.2
- **Conv2d:** (in_channels=128, out_channels=256, kernel_size=4, stride=2, padding=1)
- **InstanceNorm2d:** 256 features.
- **LeakyReLU:** Negative slope=0.2
- **Conv2d:** (in_channels=256, out_channels=512, kernel_size=4, stride=1, padding=1)
- **InstanceNorm2d:** 512 features.
- **LeakyReLU:** Negative slope=0.2
- **Conv2d:** (in_channels=512, out_channels=1, kernel_size=4, stride=1, padding=1)
- **Output:** Patch-wise prediction map (not a single scalar).

5.4 Key Design Choices

- **Reflection Padding:** Used in the generator to avoid border artifacts typically caused by zero-padding.
- **Instance Normalization:** Replaces batch normalization for better results on artistic images and stylization tasks.
- **Residual Connections:** Help the generator maintain identity features across domains without losing critical structural information.
- **PatchGAN Discriminator:** Instead of evaluating the entire image, PatchGAN classifies small patches, reducing the number of parameters and focusing learning on texture and style rather than global structure.

5.5 Architechtural Flow



Real Image
Ghibli Image
G_BA(G_AB(Real)) ≈ Real
Generator G_AB (Real → Ghibli)
Generator G_BA (Ghibli → Real)
G_AB(G_BA(Ghibli)) ≈ Ghibli
Fake Ghibli Image
Fake Real Image
Discriminator D_B (Real Ghibli vs Fake)
Discriminator D_A (Real Real vs Fake)

5.6 Loss Functions

In CycleGAN, multiple losses are combined to guide training. Each plays a specific role in ensuring quality and consistency during image translation.

5.6.1 Adversarial Loss (GAN Loss)

Objective: Encourage generators (G_{AB} and G_{BA}) to produce outputs indistinguishable from real images in the target domain.

Implementation: Mean Squared Error (MSELoss) is used instead of Binary Cross-Entropy Loss for better stability.

Mathematical Formulation:

$$\mathcal{L}_{GAN}(G, D, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{data}(x)}[(D(G(x)))^2]$$

where:

- G is the generator,
- D is the discriminator,
- X and Y are source and target domain samples respectively.

Explanation:

- Discriminator D learns to output 1 for real images and 0 for fake ones.
- Generator G learns to fool D into predicting 1 for generated images.

5.6.2 Cycle Consistency Loss

Objective: Ensure that translating an image to another domain and back returns the original image.

Implementation: L1 loss (Mean Absolute Error) between the input image and the reconstructed image.

Mathematical Formulation:

$$\mathcal{L}_{cycle}(G_{AB}, G_{BA}) = \mathbb{E}_{x \sim p_{data}(x)}[\|G_{BA}(G_{AB}(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G_{AB}(G_{BA}(y)) - y\|_1]$$

Explanation:

- If x from domain A is translated to domain B and back, it should recover x.
- Likewise for y from domain B.

5.6.3 Identity Loss

Objective: Regularize the generator so that when real samples of the target domain are given, the output remains similar.

Implementation: L1 loss between the generator output and the input when the input is already from the target domain.

Mathematical Formulation:

$$\mathcal{L}_{identity}(G_{AB}, G_{BA}) = \mathbb{E}_{y \sim p_{data}(y)}[\|G_{AB}(y) - y\|_1] + \mathbb{E}_{x \sim p_{data}(x)}[\|G_{BA}(x) - x\|_1]$$

Explanation:

- Feeding target domain images into the generator should ideally result in no changes.
- Helps preserve color and content characteristics.

5.6.4 Full Generator Loss

Objective: The full generator objective combines all three components:

$$\mathcal{L}_G = \mathcal{L}_{GAN}(G_{AB}, D_B, A, B) + \mathcal{L}_{GAN}(G_{BA}, D_A, B, A) + \lambda_{cycle} \cdot \mathcal{L}_{cycle}(G_{AB}, G_{BA}) + \lambda_{identity} \cdot \mathcal{L}_{identity}(G_{AB}, G_{BA})$$

where:

- $\lambda_{cycle} = 10$
- $\lambda_{identity} = 5$

Meaning of Weights:

- λ_{cycle} heavily prioritizes cycle consistency.
- $\lambda_{identity}$ regularizes identity preservation, but with a smaller weight.

6 Implementation

6.1 Tools and Libraries

- **Python**
- **PyTorch**
- **Torchvision**
- **Google Colab** (for GPU support)
- **Matplotlib**, **PIL**, **TQDM**

6.2 Hyperparameters

Parameter	**Value**
Learning Rate	0.0002
Optimizer	Adam ($\beta_1 = 0.5$, $\beta_2 = 0.999$)
Batch Size	1
Epochs	50

6.3 Training Process

6.3.1 Optimizer Settings

Optimizer Type:
The Adam optimizer is used for both generators and discriminators.

Hyperparameters:

- Learning Rate (LR): 2×10^{-4}
- β_1 (first moment decay): 0.5
- β_2 (second moment decay): 0.999

Purpose:
Using Adam with these β parameters stabilizes GAN training by reducing oscillations and providing smooth gradients.

6.3.2 Device Setup

Hardware:
The model automatically selects GPU (if available) or falls back to CPU.

Purpose:
Utilizing GPU accelerates model training significantly, especially for deep networks like CycleGAN.

6.4 Checkpointing Strategy

To ensure training continuity and recoverability:

- **Latest Checkpoint Save:**
  Every 100 iterations, the latest model weights and optimizer states are saved to: `/content/drive/MyDrive/cyclegan_c`
- **Epoch Checkpoint Save:**
  After each epoch, a new checkpoint is saved with the epoch number (e.g., `cyclegan_epoch_5.pth`).

Purpose:

- Enables recovery from interruptions.
- Allows model selection based on best visual results.

Sample Saving:
Generates a sample output at the end of each epoch. Saves the real image, generated Ghibli-style image, and the reconstructed real image for visual tracking of progress.

6.4.1 Loss Functions

The model optimizes multiple objectives:

Loss Type	Description
Adversarial Loss (GAN Loss)	Encourages generators to produce outputs indistinguishable from real images.
Cycle Consistency Loss	Ensures that translating to another domain and back recovers the original image.
Identity Loss	Regularizes the generator by minimizing unnecessary changes when input is already in target domain.

Loss Functions Used:

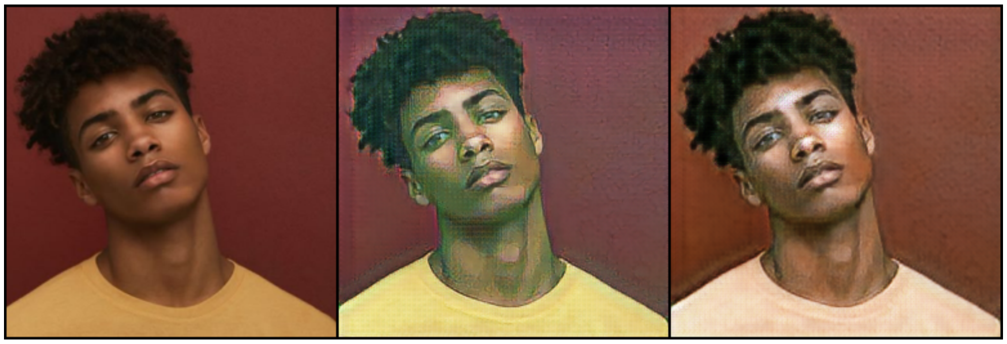
- **MSELoss** for adversarial losses.
- **L1Loss** for cycle and identity losses.

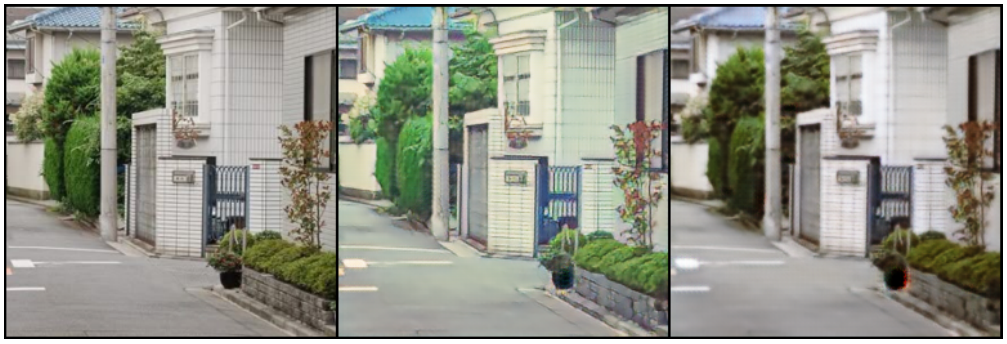
6.4.2 Training Configuration

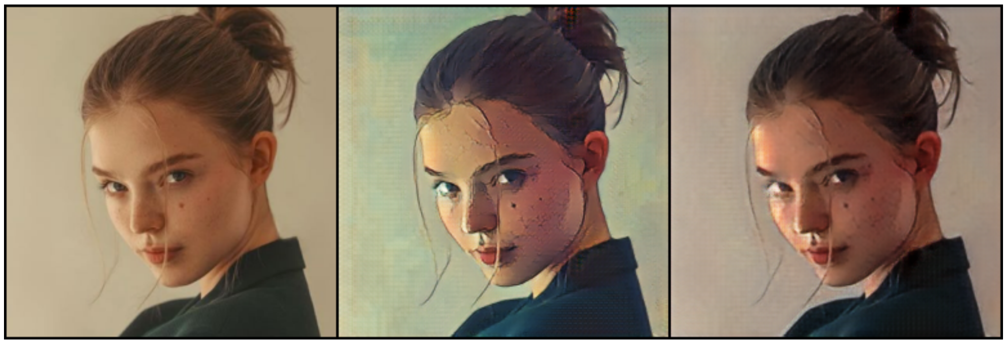
Aspect	Value
Number of Epochs	50
Batch Size	1
Optimizer	Adam
Learning Rate	2×10^{-4}
β Parameters	(0.5, 0.999)
Device	GPU (if available), else CPU
Checkpoints	Every 100 iterations + every epoch
Sample Generation	Every epoch

7 Results

7.1 Training Progression Results


(a) Epoch 10: Real Image → Generated Image → Reconstructed Image


(b) Epoch 20: Real Image → Generated Image → Reconstructed Image


(c) Epoch 30: Real Image → Generated Image → Reconstructed Image

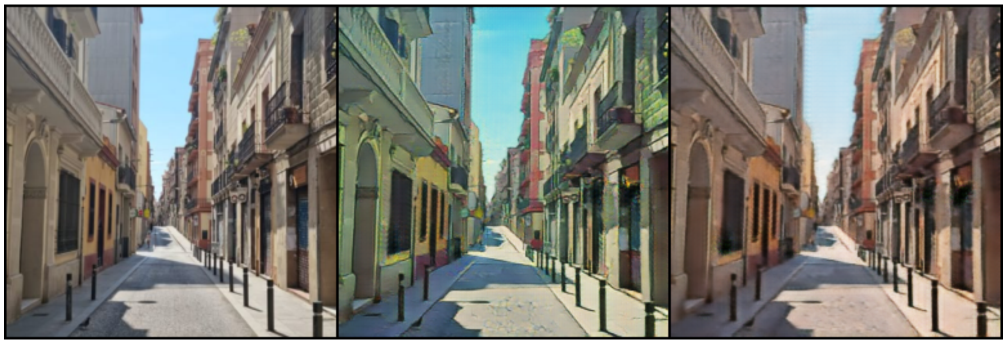

(d) Epoch 50: Real Image → Generated Image → Reconstructed Image

Figure 2: Training Progression: Visualization of image translation results respect to epochs.

7.2 Visual Results on Test

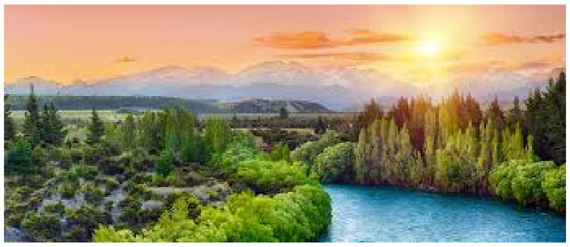


Figure 3: Real Image - 1

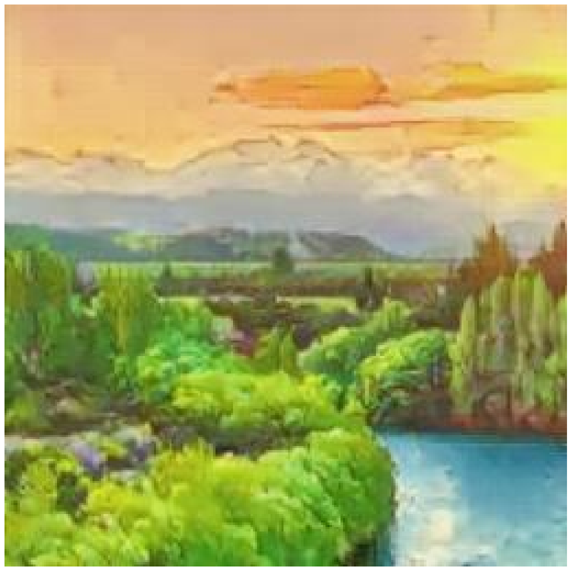


Figure 4: Translated Ghibli Image - 1

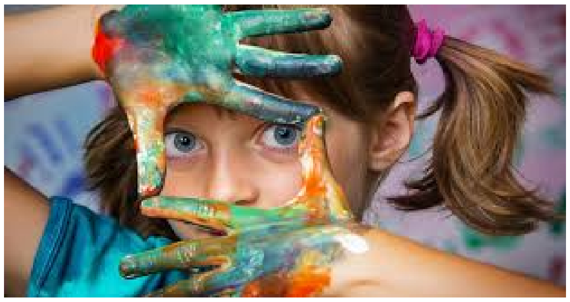


Figure 5: Real Image - 2

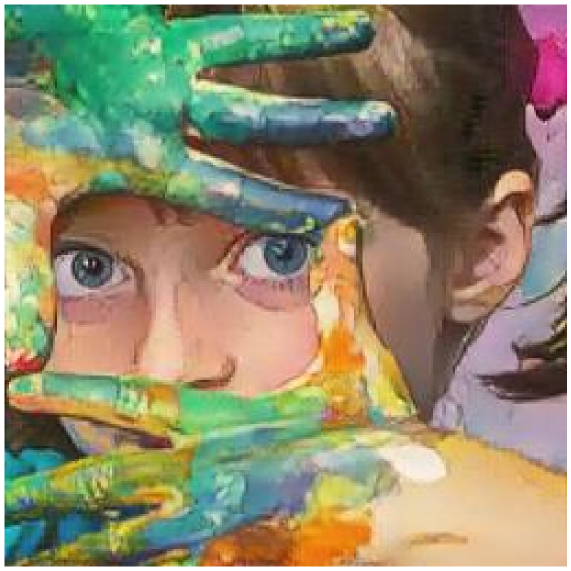


Figure 6: Translated Ghibli Image - 2

Qualitative Results: Real Images translated to Ghibli Style using CycleGAN

8 Discussion

8.1 Interpretation of Results

The CycleGAN model successfully learned to translate real-world images into Ghibli-style outputs.

- Identity loss helped in maintaining color consistency.
- Cycle consistency loss ensured content structures were preserved.

8.2 Challenges

- Training GANs from scratch is unstable; careful hyperparameter tuning was required.
- Mode collapse was occasionally observed.

8.3 Limitations

- Some outputs lacked fine texture details.
- Training with a larger batch size or longer epochs could further improve results.

9 Conclusion

This project successfully demonstrates unpaired image-to-image translation using a CycleGAN architecture implemented entirely from scratch. The model effectively translates real-world images into stylized Ghibli illustrations without using any pretrained components.

10 Future Work

- Incorporate attention mechanisms to enhance detail preservation.
- Introduce perceptual loss for better texture learning.
- Extend training duration and explore different GAN stabilization techniques.

11 References

1. Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* (CycleGAN). In *Proceedings of the IEEE International Conference on Computer Vision (ICCV).*
2. Huang, G. B., Ramesh, M., Berg, T., & Learned-Miller, E. (2008). *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.* University of Massachusetts, Amherst.
3. Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2016). *Image-to-Image Translation with Conditional Adversarial Networks.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
4. Wang, Z., & Bovik, A. C. (2009). *Mean Squared Error: Love it or Leave it? A New Look at Signal Fidelity Measures. IEEE Signal Processing Magazine*, 26(1), 98–117.
5. Official CycleGAN Code Repository: `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`.