SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMAKURU-572103

(An Autonomous Institute under Visvesvaraya Technological University, Belagavi)

Project Report on

"Adaptive and Attentive Neural Style Transfer for Digital Art"

submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE & ENGINEERING

Submitted by

Batch ID 5

Aman Kumar	(1SI22CS014)
Ashay Amal	(1SI22CS027)
Avinash Sarraf	(1SI22CS030)
Chandragupta Kumar	(1SI22CS046)

under the guidance of

Rajeshwari KR

Assistant Professor

Department of CSE

SIT, Tumakuru-03

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2025-26

SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMAKURU-572103

(An Autonomous Institute under Visvesvaraya Technological University, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project work entitled "ADAPTIVE AND ATTENTIVE NEURAL STYLE TRANSFER FOR DIGITAL ART" is a bonafide work carried out by Aman Kumar (1SI22CS014), Ashay Amal (1SI22CS027), Avinash Sarraf (1SI22CS030) and Chandragupta Kumar (1SI22CS046) in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science & Engineering from Siddaganga Institute of Technology, an autonomous institute under Visvesvaraya Technological University, Belagavi during the academic year 2025-26. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The Project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the Bachelor of Engineering degree.

Rajeshwari KR	Dr. N R Sunitha	Dr. S V Dinesh
Assistant Professor	Head of the Department	Principal
Dept. of CSE	Dept. of CSE	SIT,Tumakuru-03
SIT,Tumakuru-03	SIT,Tumakuru-03	

External viva:

Names of the Examiners **Signature with date**

1.

2.

ACKNOWLEDGEMENT

We offer our humble pranams at the lotus feet of **His** Holiness, **Dr. Sree Sree Sivakumara Swamigalu**, Founder President and **His** Holiness, **Sree Sree Siddalinga Swamigalu**, President, Sree Siddaganga Education Society, Sree Siddaganga Math for bestowing upon their blessings.

We deem it as a privilege to thank **Dr. Shivakumaraiah**, CEO, SIT, Tumakuru, and **Dr. S V Dinesh**, Principal, SIT, Tumakuru for fostering an excellent academic environment in this institution, which made this endeavor fruitful.

We would like to express our sincere gratitude to **Dr. N R Sunitha**, Professor and Head, Department of CS&E, SIT, Tumakuru for her encouragement and valuable suggestions.

We thank our guide **Rajeshwari KR**, Assistant Professor, Department of Computer Science & Engineering, SIT, Tumakuru for the valuable guidance, advice and encouragement.

Aman Kumar	(1SI22CS014)
Ashay Amal	(1SI22CS027)
Avinash Sarraf	(1SI22CS030)
Chandragupta Kumar	(1SI22CS046)

Course Outcomes

After successful completion of major project, graduates will be able:

CO1: To identify a problem through literature survey and knowledge of contemporary engineering technology.

CO2: To consolidate the literature search to identify issues/gaps and formulate the engineering problem

CO3: To prepare project schedule for the identified design methodology and engage in budget analysis, and share responsibility for every member in the team

CO4: To provide sustainable engineering solution considering health, safety, legal, cultural issues and also demonstrate concern for environment

CO5: To identify and apply the mathematical concepts, science concepts, engineering and management concepts necessary to implement the identified engineering problem

CO6: To select the engineering tools/components required to implement the proposed solution for the identified engineering problem

CO7: To analyze, design, and implement optimal design solution, interpret results of experiments and draw valid conclusion

CO8: To demonstrate effective written communication through the project report, the one-page poster presentation, and preparation of the video about the project and the four page IEEE/Springer/ paper format of the work

CO9: To engage in effective oral communication through power point presentation and demonstration of the project work

CO10:To demonstrate compliance to the prescribed standards/ safety norms and abide by the norms of professional ethics

CO11: To perform in the team, contribute to the team and mentor/lead the team

CO-PO and PSO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
CO-1											3			
CO-2		3												
CO-3										3			3	
CO-4						3							3	
CO-5	3	3										3	3	
CO-6					3						3		3	
CO-7			3	3								3	3	3
CO-8									3					
CO-9									3					
CO-10							3							
CO-11								3						
Average	3	3	3	3	3	3	3	3	3	3	3			

Attainment level:

1: Slight (low)

2: Moderate (medium)

3: Substantial (high)

POs(Program Outcomes):
PO1: Engineering knowledge,

PO2: Problem analysis,

PO3: Design of solutions,

PO4: Conduct investigations of complex problems,

PO5: Engineering tool usage,

PO6: Engineer and the world,

PO7: Ethics,

PO8: Individual and collaborative work,

PO9: Communication,

PO10: Project management and finance,

PO11: Life-long learning.

PSOs(Program Specific Outcomes):
PSO1: Computer-based systems development,

PSO2: Software development,

PSO3: Computer communications and Internet applications.

Abstract

The project proposes a CycleGAN based Neural Style Transfer (NST) system for converting photographs into four distinct artistic styles of One Piece anime, Disney, Studio Ghibli, and paintings by Van Gogh. The system will utilize unpaired data sets with cycle-consistency constraints, which allows for the bidirectional mapping of human faces to animated characters and scenery to post-impressionist style works of art.

Utilization of the generator-discriminator (GAN-D) model involves utilizing both a generator and discriminator to create realistic artwork through adversarial loss, preserve content using Cycle-consistency loss, use identity loss for colour preservation, and incorporate VGG-19 perceptual loss. OpenCV's custom frame extraction method retrieves animated datasets from video sources, while Van Gogh datasets are constructed using landscapes and paintings retrieved from Kaggle.

Utilizing PyTorch 2.0 and accelerated by CUDA, the system provides an estimated 3-second or less delivery timeframe for each image processed by NVIDIA Graphics Processing Unit (GPU). This modular framework permits the extension of additional styles as confirmed through successful styling capture with semantic content maintained.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The convergence of artificial intelligence and creative arts has created infinite opportunities for the creation of digital based artwork. One of the most exciting current uses of deep learning is in Neural Style Transfer (NST), which allows you to take a normal photo and create an amazing piece of art from it, by learning what makes a particular style "art" and then applying that same "art" style to create something new with your own image. This project's goal is to design a seamless style transfer system using CycleGAN (Cycle-Consistent Generative Adversarial Networks) to enable the ability to convert images into 4 very unique artistic styles, which include: One Piece anime, Disney animated characters, Studio Ghibli animated characters, and Van Gogh style painting.

The emergence of Generative Adversarial Networks (GANs) in 2014 marked a paradigm shift in generative modeling. Subsequently, the introduction of CycleGAN by Zhu et al. [8] revolutionized image-to-image translation by eliminating the requirement for paired training data. This advancement is particularly significant for artistic style transfer, where obtaining perfectly aligned pairs of content and stylized images is impractical.

This project utilises the characterisation of the above examples to produce an animated human face and to create painting-like landscapes based on this characterisation. Each of these three animation styles (One Piece, Disney, Studio Ghibli) has a unique visual appearance, making it immediately identifiable. The post-impressionist style designed by Vincent Van Gogh is accepted as having a very particular brushstroke and a colour palette that is especially vibrant.

1.1 Background

1.1.1 Evolution of Computer Vision

Computer vision has transformed dramatically with deep learning. Key milestones include: traditional hand-crafted features (pre-2012), the AlexNet breakthrough (2012) demonstrating CNN superiority, deeper architectures like VGGNet and ResNet (2014-2016), and generative models including GANs (2014-present) enabling image synthesis.

1.1.2 Convolutional Neural Networks

CNNs learn features in a hierarchical manner. The early layers typically capture low-level information such as edges and textures. The intermediate layers build on these to represent more complex patterns, while the deepest layers encode high-level semantic concepts. This hierarchical organisation allows neural style transfer to separate an image's content from its style, enabling generation of new images that preserve the content but adopt an alternative style.VGG-19 is widely used for feature extraction because of its consistent 3×3 convolutional architecture and its proven reliability in modelling perceptual similarity.

1.1.3 Generative Adversarial Networks

GANs consist of two components: the generator, which is responsible for generating synthetic samples, and the discriminator, which determines whether or not a given sample is considered real or fake. Using adversarial training techniques, both components are trained until the generator produces samples created in a way that they appear realistic to real humans. Some popular variants of GAN include Conditional GAN, Pix2Pix, Cycle GAN, and Style GAN.

1.1.4 Style Transfer Approaches

GANs comprise two essential elements: a Generator responsible for the creation of synthetic samples, and a Discriminator which judges if a particular sample is classified as real or fake. By using the principles of adversarial training, both components were trained until the generator produces synthetically produced samples in such a manner that they look real to the human eye. Examples of several types of GANs include cGANs (Conditional GANs), Pix2Pix, CycleGANs and StyleGANs.

1.2 Motivation

The motivation for this project stems from several converging factors in technology, art, and social media culture:

1. Growing Demand for Personalized Digital Art:

The popularity of photo filter apps and personalized artistic transformation utilities are driven by the need for users to create artistically altered versions of their own images in creative ways. There is a large market demand for systems that can provide high-quality anime or painterly transformations of photographs to fulfill this need.

2. Bridging Art and Technology:

It normally takes a lot of time and training to develop your artistic skills through traditional means, but thanks to neural style transfer, you do not need either of those to create art. With neural style transfer, anyone can create photo-based artwork that captures the essence of famous artists or animation studios. As a result, neural style transfer has made it easier than ever for people to appreciate and create art.

3. Advances in Deep Learning:

Deep learning has made tremendous strides in just a few years. Developments in image generation and transformation based on GANs (Generative Adversarial Networks) enable the creation of complex and highly realistic photographic imagery via computational techniques. Combining the development of very capable GPU computing systems with the growing number of open-source frameworks such as PyTorch means that building these capabilities is now easier than ever before.

4. Cultural Significance of Animation Styles:

Japanese Animation (One Piece) is an entirely different visual vocabulary than Western Animation (Disney) and Studio Ghibli. Fans of both styles have a huge following around the world and this is why there is such a high demand to develop software that converts photographs into these two styles for fans and content developers around the globe.

5. Preservation and Appreciation of Artistic Heritage:

Considered one of the greatest achievements in post-impressionistic art, Van Gogh's distinctive style features swirling brush strokes and bright colours. It can be used as an educational tool and a way to appreciate the artistic heritage of Van Gogh's work and the influence it has had on contemporary visual art through the application of this style by a neural network trained to apply Van Gogh's style to current day photographs.

6. Research and Educational Value:

Completing the process of building a full CycleGAN for style transfer was an instructive way to learn about the many areas of designing a Deep Learning Architecture, developing a Loss Function, creating a Dataset, and exploring the difficulties associated with generative models. Thus, this project serves as a thorough example of how to use Applied Deep Learning.

1.3 Problem Statement

The problem addressed by this project can be formally stated as follows:
Taking an input image x from domain X (real photographs), the goal is to learn a mapping function $G : X \rightarrow Y$ that transforms x into an output image $G(x)$ in domain Y (stylized images), such that:

1. The output $G(x)$ exhibits the visual characteristics of the target artistic style
2. The semantic content and structure of the input image are preserved
3. The transformation is realistic and free from artifacts
4. The system operates efficiently for real-time applications

The specific style domains addressed are:

- **One Piece Style:** Bold outlines, exaggerated expressions, vibrant colors characteristic of Eiichiro Oda's artwork
- **Disney Style:** Smooth gradients, large expressive eyes, soft color palettes of modern Disney animation
- **Studio Ghibli Style:** Watercolor textures, naturalistic expressions, warm earthy tones of Hayao Miyazaki's films
- **Van Gogh Style:** Swirling brushstrokes, impasto technique, vibrant post-impressionist colors

1.4 Objective of the Project

The primary objectives of this project are:

1. Develop Style-Specific Neural Style Transfer Models:

You will create four different models for transforming images through CycleGAN methods, each trained on one specific art stimulus. Each model must be capable of reproducing the specific characteristics associated with the targeted artistic styles while preserving the content of the original photograph.

2. Create Custom Datasets through Frame Extraction:

Develop an automated pipeline for extracting character frames from animation sources (One Piece episodes, Disney movies, Studio Ghibli films) using computer vision techniques including face detection and quality filtering.

3. Implement and Optimize Core Algorithms:

Implement the following key algorithms with improvements tailored to our application:

- CycleGAN architecture with ResNet-based generators
- PatchGAN discriminators for local style assessment
- Cycle-consistency loss for bidirectional mapping
- Identity loss for color preservation
- Perceptual loss using VGG-19 features
- Custom frame extraction algorithm

4. Optimize for Real-Time Performance:

Achieve inference times of under 3 seconds per image through model optimization, efficient data pipelines, and GPU acceleration.

5. Build a Modular and Extensible System:

Design the system architecture to allow easy addition of new styles and modification of existing models.

1.5 Scope of the Project

The scope of this project encompasses the following:

Included in Scope:

- Implementation of CycleGAN-based style transfer for four artistic styles
- Custom dataset creation through frame extraction from video sources
- Training pipeline with configurable hyperparameters
- Inference system for single image transformation
- Documentation and analysis of results
- Comparison with existing methods

1.6 Methodology Overview

The methodology employed in this project follows a systematic six-phase approach, as illustrated in Figure 1.1. The pipeline begins with dataset collection and frame extraction from animation sources and public datasets, followed by data preprocessing and augmentation to prepare images for training. The CycleGAN model design phase implements the generator and discriminator architectures, which are then trained using multiple loss functions. Finally, evaluation and optimization assess model performance before deploying the inference system for real-world use.

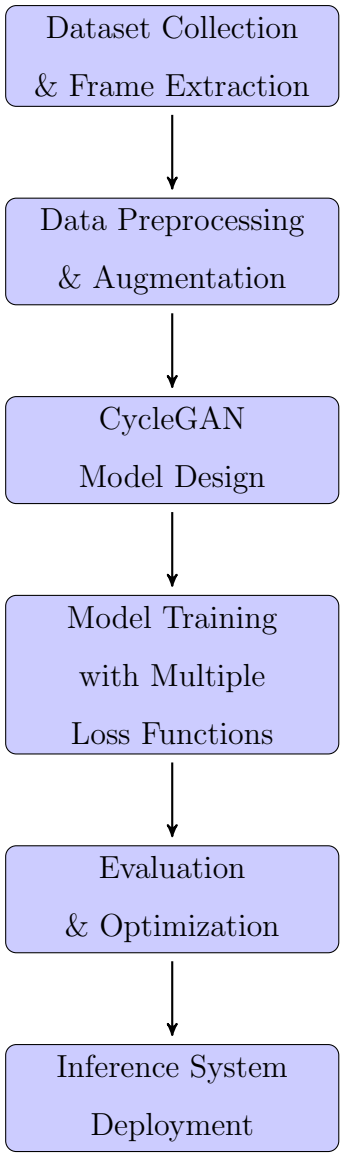

Figure 1.1: Project Methodology Overview

Phase 1: Dataset Collection and Preparation

- Extract character frames from One Piece episodes, Disney animated movies, and Studio Ghibli films
- Collect Van Gogh paintings and landscape photographs from Kaggle
- Collect human face photographs from Kaggle datasets

Phase 2: Data Preprocessing

- Resize all images to 256×256 pixels
- Apply data augmentation (horizontal flip, rotation)
- Normalize pixel values to [-1, 1] range
- Create training and validation splits

Phase 3: Model Implementation

- Implement generator networks with ResNet architecture
- Implement PatchGAN discriminator networks
- Define loss functions (adversarial, cycle-consistency, identity, perceptual)
- Set up training loops with Adam optimizer

Phase 4: Training and Optimization

- Train separate models for each style
- Monitor training progress with validation metrics
- Apply learning rate scheduling
- Save best model checkpoints

Phase 5: Evaluation and Deployment

- Evaluate models using FID scores and visual inspection
- Optimize inference pipeline for speed
- Document results and create demonstration system

1.7 Organisation of the Report

This project report is organized into the following chapters:

Chapter 1: Introduction

An overview of the project is provided, including motivation, problem statement, project objectives, scope and methodology. This chapter outlines the general context and importance of neural style transfer as part of the larger field of deep learning.

Chapter 2: Literature Survey

An overview of the project is provided, including motivation, problem statement, project objectives, scope and methodology. This chapter outlines the general context and importance of neural style transfer as part of the larger field of deep learning.

Chapter 3: System Overview

The descriptions of the overall system architecture will include the styles transfer pipeline, methodology for creating the data set and the four styles used. This chapter presents an overview of the relationships between the various components of the system at a higher level.

Chapter 4: System Architecture and High-Level Design

Presents the detailed architecture of the CycleGAN system, including generator and discriminator networks, loss function formulations, and training procedures. UML diagrams, flowcharts, and architectural diagrams illustrate the design.

Chapter 5: Software Architecture and Low-Level Design

Provides detailed algorithm descriptions, pseudocode, and implementation specifics. This chapter covers the frame extraction algorithm, network architectures, and training procedures at a granular level.

Chapter 6: Results

Presents the experimental setup, test procedures, and results obtained from training and evaluating the four style transfer models. Includes sample outputs, quantitative metrics, and comparative analysis.

Chapter 7: Conclusion

Summarizes the project achievements, discusses limitations, and suggests directions for future work.

Appendices

Contains supplementary material including project timeline, budget estimation, dataset details, SDG alignment, and configuration information.

Chapter 2

Literature Survey

This chapter presents a comprehensive review of the existing research in neural style transfer, generative adversarial networks, and image-to-image translation that forms the foundation of our implementation.

Gatys et al., [1] developed the first deep learning method capable of separating image style from content using a pre-trained VGG network. They demonstrated that higher-level feature maps capture the scene's structure, while the Gram matrices of lower layers encode texture and colour information. Their optimization-based approach generates a new image that matches the content representation of one image while adopting the style statistics of another. This enables the creation of artwork that mimics famous artists while preserving the original scene's objects. Although flexible, the method requires significant computation time, establishing it as a foundational reference for subsequent neural style transfer research.

Johnson et al., [2] replaced pixel-wise loss functions with perceptual loss functions based on VGG features to train fast feedforward networks for style transfer. A single network can learn to approximate the expensive optimization enabling real-time stylization at comparable quality. Their experiments show that feature-space losses produce sharper super-resolution results than MSE-based approaches, as human perception correlates better with high-level feature differences. This work significantly influenced architectural design choices for real-time style transfer networks.

Ulyanov et al., [3] demonstrated that replacing batch normalization with instance normalization substantially improves the quality of feed-forward style transfer networks. Instance normalization normalizes each feature map independently, removing instance-specific contrast and allowing the generator to focus on style statistics rather than global illumination. Networks trained with this modification achieve results comparable to optimization-based methods while operating in real-time. This work established instance normalization as a standard component in style transfer and texture synthesis networks.

Huang and Belongie, [4] proposed Adaptive Instance Normalization (AdaIN), a technique that aligns the channel-wise mean and variance of content features to match those of the style image. This approach enables arbitrary style transfer without retraining, allowing any style to be applied to any content image while being over 100x faster than optimization-based methods. AdaIN also provides intuitive controls for adjusting the content-style trade-off, interpolating between multiple styles, and applying spatial or colour constraints. This technique has become a foundational component in arbitrary style transfer models and image-to-image translation applications.

Li et al., [5] proposed a universal style transfer method using VGG as an encoder with cascaded decoders and whitening and colouring transforms (WCT) in feature space. The WCT matches the covariance of content features to that of the style image through a multi-level pipeline, progressing from coarse to fine layers. Since the decoders are trained only for reconstruction, the method generalizes to unseen styles without retraining. This approach bridges Gram-matrix optimization with feed-forward networks through explicit feature covariance matching.

Ghiasi et al., [6] extended fast arbitrary style transfer by developing a network that predicts affine parameters for conditional instance normalization layers. Trained on approximately 80,000 paintings and thousands of textures, the model performs real-time style transfer for previously unseen styles. The learned style embeddings enable smooth interpolation between different styles. This work demonstrated that predicting normalization parameters allows feed-forward networks to function as efficient conditional networks for arbitrary styles.

Luan et al., [7] focused on transferring visual styles between photographic images rather than paintings. They introduced an additional photorealism loss based on the Matting Laplacian to constrain local colour adjustments and preserve photorealistic appearance. Semantic segmentation masks guide the transfer to ensure corresponding regions are matched appropriately. Their method produces high-quality results across diverse photographic scenes.

Zhu et al., [8] proposed CycleGAN, an unpaired image-to-image translation framework using adversarial and cycle-consistency losses. Two generators learn mappings $X \to Y$ and $Y \to X$, while discriminators enforce realism. The cycle-consistency loss ensures $F(G(x)) \approx x$ and $G(F(y)) \approx y$, preventing mode collapse without paired supervision.

The method successfully handles diverse tasks including horse-zebra, summer-winter, and photo-painting conversions, establishing a standard baseline for unpaired translation research.

Zhang et al., [9] combined AdaIN with Gram-matrix style features to create a faster and more flexible digital art generation system. Their model improves content and style accuracy by approximately 15% over existing approaches, achieving an SSIM of 0.88 at medium style intensity while reducing processing time by nearly 76%. The method handles diverse artistic styles while preserving content structure, making neural style transfer more practical for designers and multimedia creators.

Scott et al., [10] reviewed developments combining GAN image synthesis with neural style transfer. Their survey covers traditional VGG-based style transfer, efficient feed-forward networks with AdaIN, and various GAN architectures including StyleGAN. They also examine diffusion-based models and their applications in img2img pipelines and ControlNet workflows. The review discusses how recent models have expanded creative potential while raising concerns about authorship and authenticity. The authors advocate for a symbiotic relationship between humans and AI, where humans provide creative vision while AI assists with variation and execution.

Karras et al., [11] proposed StyleGAN, a generator architecture providing unprecedented control over image synthesis. The architecture features a mapping network that transforms latent codes into an intermediate style space, with adaptive instance normalization modulating convolutional layers. This design separates high-level attributes (pose, identity) from stochastic variations (freckles, hair texture), enabling intuitive style mixing at multiple scales. StyleGAN achieves state-of-the-art face generation quality with smooth latent space transitions, significantly influencing subsequent work on controllable image generation and style manipulation.

Jing et al., [12] conducted a comprehensive survey of neural style transfer methods, covering loss functions, network architectures, and evaluation metrics. They examined emerging directions including semantic-aware, video, and 3D stylization techniques. The survey identifies key challenges such as content leakage, style consistency, and computational efficiency. This work serves as an essential reference for understanding the current state and future directions of neural style transfer research.

Isola et al., [13] proposed pix2pix, a versatile image-to-image translation framework using conditional GANs for paired training data. The framework combines adversarial loss with L1 reconstruction loss to preserve spatial information. A PatchGAN discriminator classifies local image patches rather than entire images, improving high-frequency detail. The method successfully addresses diverse tasks including semantic segmentation to photo synthesis, edge-to-photo generation, and day-to-night conversion, establishing a powerful paradigm for supervised image translation.

Chapter 3

System Overview

The overall architecture, dataset creation process, and four different style transfer models: One Piece, Disney, Studio Ghibli, and Van Gogh of the AI-Based Neural Style Transfer system are all covered in detail in this chapter.

3.1 System Architecture Overview

The system employs a modular pipeline design using CycleGAN-based neural networks to create artistic stylizations from user-provided content images, as shown in Figure 3.1. The architecture supports four different style transfer modes, each trained on specially chosen datasets.

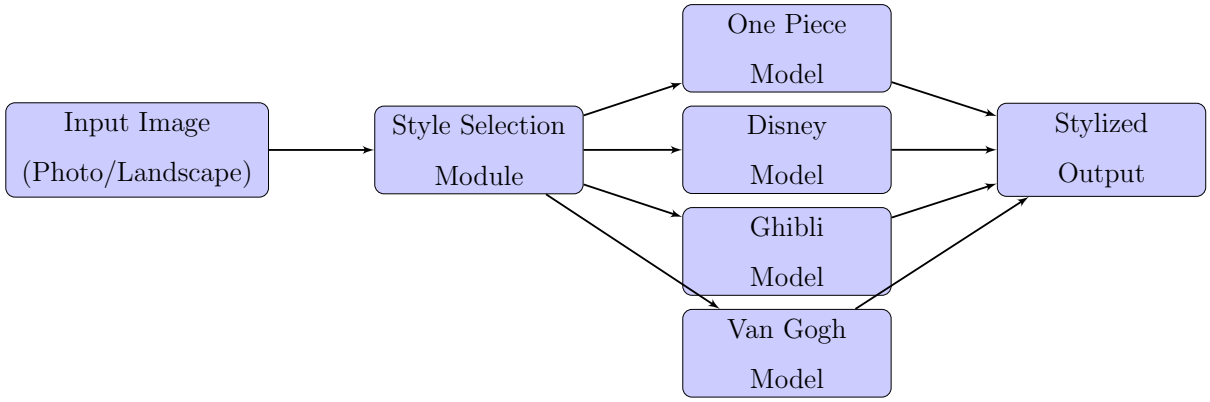

Figure 3.1: High-Level System Architecture

3.2 Style Transfer Modes

Four different style transfer modes are used by the system:

3.2.1 One Piece Style Transfer

- **Input:** Human face photographs
- **Output:** One Piece anime-style character faces
- **Characteristics:** Bold black outlines, exaggerated expressions, vibrant saturated colors, distinctive large eyes
- **Dataset Source:** Frames extracted from One Piece anime episodes and movies

3.2.2 Disney Style Transfer

- **Input:** Human face photographs
- **Output:** Disney animation-style character faces
- **Characteristics:** Smooth color gradients, large expressive eyes with reflections, soft pastel color palettes, polished clean lines
- **Dataset Source:** Frames extracted from Disney animated feature films

3.2.3 Studio Ghibli Style Transfer

- **Input:** Human face photographs
- **Output:** Studio Ghibli animation-style character faces
- **Characteristics:** Soft watercolor-like textures, naturalistic expressions, warm earthy tones, hand-drawn aesthetic
- **Dataset Source:** Frames extracted from Studio Ghibli films (Spirited Away, Howl's Moving Castle, etc.)

3.2.4 Van Gogh Style Transfer

- **Input:** Landscape photographs
- **Output:** Van Gogh painting-style landscapes
- **Characteristics:** Swirling brushstrokes, vibrant colors (yellows, blues), impasto technique, post-impressionist aesthetic
- **Dataset Source:** Van Gogh paintings from Kaggle dataset

3.3 Dataset Creation Methodology

We created a thorough dataset creation pipeline because CycleGAN requires unpaired datasets from two domains (Domain X: real-world images, Domain Y: stylized images).

3.3.1 Animation Frame Extraction Pipeline

The frame extraction pipeline, as shown in Figure 3.2, automates the process of extracting high-quality character frames from video sources. The pipeline starts by reading video files and extracting individual frames, followed by face or character detection. Frames with detection confidence below 0.9 are discarded. Accepted frames are cropped and resized, then checked for similarity against existing images to avoid duplicates. Finally, a quality check filters out blurry or artifact-containing frames before saving to the dataset.

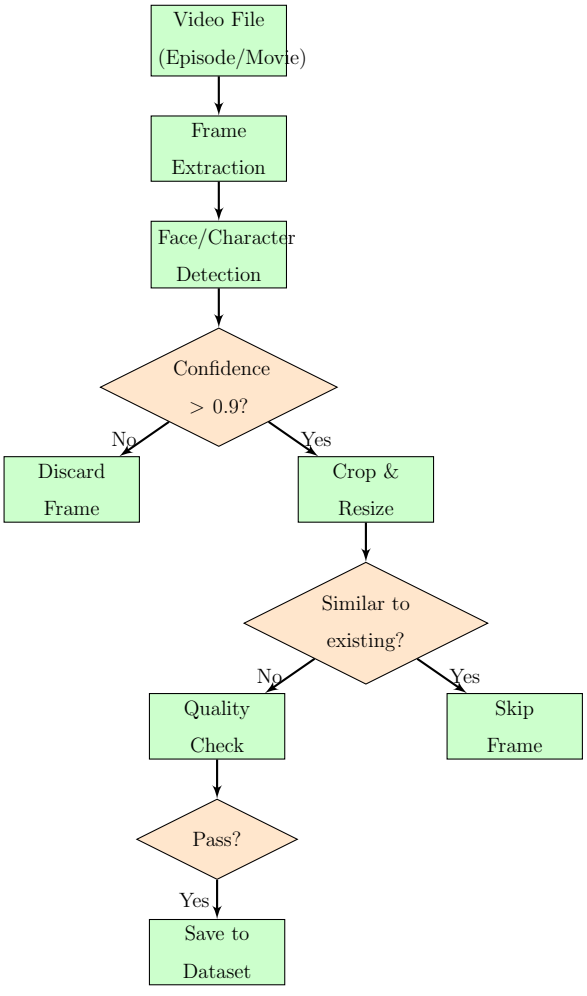

Figure 3.2: Frame Extraction Pipeline

3.3.2 Dataset Details

Every dataset uses JPG/PNG images with a resolution of 256 ×256. Domain X utilizes either landscapes or human faces from Kaggle (3,000+ images each for face-based styles).

3.3.3 Dataset Summary

Table 3.1 provides a comprehensive overview of the datasets used for training each style transfer model, showing the domain sources and the number of images collected for both real and stylized domains.

Table 3.1: Complete Dataset Summary

Style	Domain X	Domain X Size	Domain Y	Domain Y Size
One Piece	Human Faces	3,000+	Anime Frames	2,500+
Disney	Human Faces	3,000+	Disney Frames	2,000+
Ghibli	Human Faces	3,000+	Ghibli Frames	2,000+
Van Gogh	Landscapes	2,500+	Paintings	400+

3.4 CycleGAN Architecture Details

3.4.1 Overall Architecture

The CycleGAN model uses two generator–discriminator pairs that work together to learn mappings in both directions between the two domains, as illustrated in Figure 3.3.

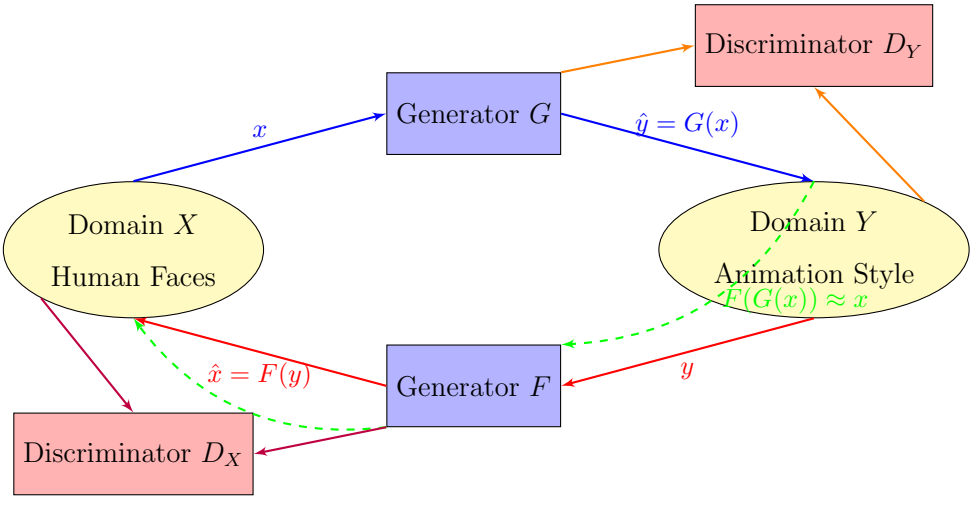

Figure 3.3: Complete CycleGAN Architecture for Style Transfer

3.4.2 Generator Network Architecture

The generator is built using a ResNet-style encoder–decoder structure and includes nine residual blocks, which is suitable for 256×256 images. Figure 3.4 shows the detailed architecture of the generator network.

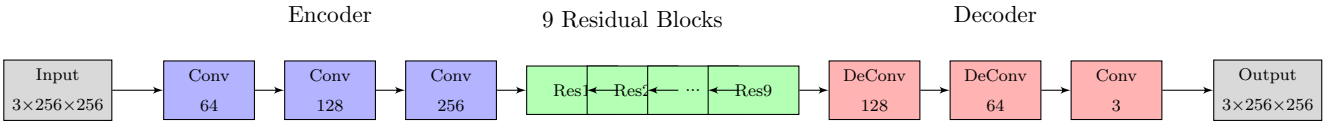

Figure 3.4: Generator Network Architecture

Generator Layer Specifications:

The detailed layer-by-layer specifications of the generator network are presented in Table 3.2, including filter sizes, kernel dimensions, and stride values for each convolutional layer.

Table 3.2: Generator Network Layers

Layer	**Type**	**Filters**	**Kernel**	**Stride**
c7s1-64	Conv + IN + ReLU	64	7×7	1
d128	Conv + IN + ReLU	128	3×3	2
d256	Conv + IN + ReLU	256	3×3	2
R256 ×9	Residual Block	256	3×3	1
u128	DeConv + IN + ReLU	128	3×3	2
u64	DeConv + IN + ReLU	64	3×3	2
c7s1-3	Conv + Tanh	3	7×7	1

3.4.3 Residual Block Architecture

Each residual block preserves the input information through a skip connection while learning additional transformations, as depicted in Figure 3.5.

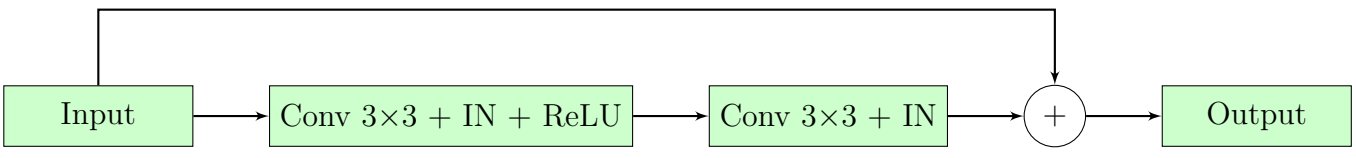

Figure 3.5: Residual Block Structure

3.4.4 Discriminator Network Architecture (PatchGAN)

The discriminator adopts a PatchGAN design, where it evaluates and classifies overlapping 70×70 image patches, as shown in Figure 3.6.

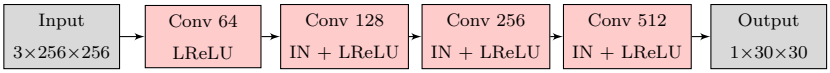

Figure 3.6: PatchGAN Discriminator Architecture

Discriminator Layer Specifications:

Table 3.3 details the PatchGAN discriminator architecture, showing how spatial dimensions are progressively reduced while feature depth increases through successive convolutional layers.

Table 3.3: Discriminator Network Layers

Layer	**Type**	**Filters**	**Kernel**	**Stride**
C64	Conv + LeakyReLU	64	4×4	2
C128	Conv + IN + LeakyReLU	128	4×4	2
C256	Conv + IN + LeakyReLU	256	4×4	2
C512	Conv + IN + LeakyReLU	512	4×4	1
Output	Conv	1	4×4	1

3.5 Loss Functions

Training uses a combination of different loss functions, each contributing to a specific aspect of achieving high-quality style transfer. Balancing these losses properly is essential to generate visually appealing outputs while still preserving the original content.

3.5.1 Adversarial Loss (LSGAN)

The adversarial loss helps the model generate images that look as realistic as those in the target domain. For improved training stability, we use the Least Squares GAN (LSGAN) approach instead of the standard cross-entropy loss.

For generator G and discriminator D_Y:

$$\mathcal{L}_{LSGAN}(G, D_Y) = \mathbb{E}_{y \sim p_{data}(y)}[(D_Y(y) - 1)^2] + \mathbb{E}_{x \sim p_{data}(x)}[D_Y(G(x))^2] \tag{3.1}$$

The generator tries to minimize:

$$\mathcal{L}_G^{adv} = \mathbb{E}_{x \sim p_{data}(x)}[(D_Y(G(x)) - 1)^2] \tag{3.2}$$

Why LSGAN:

- Offers smoother gradients than traditional binary cross-entropy
- Applies stronger penalties to samples that lie far from the decision boundary
- Helps prevent vanishing gradient issues during training
- Generates higher-quality images with fewer visible artifacts

3.5.2 Cycle-Consistency Loss

The cycle-consistency loss is the core idea behind CycleGAN that makes training on unpaired data possible. It ensures that when an image is translated to the target domain and then converted back, the result remains close to the original image:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1] \tag{3.3}$$

Forward Cycle: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$
Backward Cycle: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$
The L1 norm (absolute difference) is used rather than L2 to produce sharper reconstructions. The cycle-consistency loss:

- Prevents mode collapse (generator producing identical outputs for all inputs)
- Ensures the mapping is meaningful and preserves content
- Regularizes the generators to learn bijective mappings
- Enables learning without paired training data

3.5.3 Identity Loss

The identity loss helps the generator keep the input unchanged when it is already from the target domain:

$$\mathcal{L}_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[||G(y) - y||_1] + \mathbb{E}_{x \sim p_{data}(x)}[||F(x) - x||_1] \tag{3.4}$$

This loss is particularly important for:

- **Color Preservation:** Prevents drastic, unnecessary color shifts

- **Style Consistency:** Ensures that images already in the target style are not over-transformed
- **Regularization:** Provides additional constraint on the mapping function

In our animation style transfer models, the identity loss plays an important role in maintaining natural skin tones and avoiding unwanted color distortions in the final output.

3.5.4 Perceptual Loss

The perceptual loss relies on a pre-trained VGG-19 network to compare high-level perceptual features between the input and the generated output:

$$\mathcal{L}_{perceptual} = \sum_{l \in L} \lambda_l \frac{1}{C_l H_l W_l} ||\phi_l(G(x)) - \phi_l(x)||_2^2 \tag{3.5}$$

Here, $\phi_l(x)$ denotes the feature map at layer l of VGG-19, and L is the set of layers used (typically relu1_1, relu2_1, relu3_1, relu4_1). The term $C_l H_l W_l$ normalizes by feature map dimensions, ensuring equal contribution from all layers. The weight λ_l controls each layer's importance in the final loss.

Benefits of Perceptual Loss:

- Preserves semantic content better than pixel-wise losses
- Maintains structural similarity during style transformation
- Correlates better with human perception of image quality

3.5.5 Total Generator Loss

The final generator loss is obtained by combining all the loss components, each weighted appropriately to achieve the best overall performance:

$$\mathcal{L}_{G,F} = \lambda_{adv}(\mathcal{L}_G^{adv} + \mathcal{L}_F^{adv}) + \lambda_{cyc}\mathcal{L}_{cyc} + \lambda_{id}\mathcal{L}_{identity} + \lambda_{perc}\mathcal{L}_{perceptual} \tag{3.6}$$

Here, $\mathcal{L}_G^{adv}$ and $\mathcal{L}_F^{adv}$ are the adversarial losses for generators G and F, $\mathcal{L}_{cyc}$ ensures reversible mapping, $\mathcal{L}_{identity}$ preserves colors, and $\mathcal{L}_{perceptual}$ maintains semantic features. The λ parameters control the relative importance of each component.

Loss Weights Used:

The carefully tuned weights for each loss component are shown in Table 3.4. These hyperparameters balance the competing objectives of generating realistic stylizations while preserving content and color fidelity.

Table 3.4: Loss Function Weights

Loss Component	**Weight**	**Purpose**
Adversarial (λ_{adv})	1.0	Realism of generated images
Cycle-Consistency (λ_{cyc})	10.0	Content preservation
Identity (λ_{id})	5.0	Color preservation
Perceptual (λ_{perc})	1.0	Semantic content

Total Loss:

$$\mathcal{L}_{total} = \mathcal{L}_{GAN} + 10\mathcal{L}_{cyc} + 5\mathcal{L}_{identity} + \mathcal{L}_{perceptual} \tag{3.7}$$

3.5.6 Discriminator Loss

Each discriminator is trained to tell apart real images from those produced by the generator:

$$\mathcal{L}_{D_Y} = \mathbb{E}_y[(D_Y(y) - 1)^2] + \mathbb{E}_x[D_Y(G(x))^2] \tag{3.8}$$

$$\mathcal{L}_{D_X} = \mathbb{E}_x[(D_X(x) - 1)^2] + \mathbb{E}_y[D_X(F(y))^2] \tag{3.9}$$

The discriminators use a history buffer of 50 previously generated images to help stabilize training and reduce oscillations.

3.6 Training Configuration

This section explains the hyperparameters and the training processes used to develop the style transfer models.

3.6.1 Hyperparameter Selection

Table 3.5 summarizes the key training hyperparameters used across all style transfer models, including image resolution, batch size, optimizer settings, and learning rate schedules.

Table 3.5: Training Hyperparameters

Parameter	**Value**	**Parameter**	**Value**
Image Size	256×256	Optimizer	Adam (β_1=0.5, β_2=0.999)
Batch Size	4	Learning Rate	0.0002 (linear decay from epoch 100)
Total Epochs	200	Residual Blocks	9

Hyperparameter Justification:

- **Image Size (256×256):** This size offers a good balance between output quality and computational efficiency. Using larger resolutions would significantly increase memory usage while giving only minimal improvements in style transfer quality.
- **Batch Size (4):** GPU memory limits become a concern when training multiple networks simultaneously (two generators and two discriminators). Using smaller batch sizes also improves the stability of GAN training.
- **Learning Rate (0.0002):** This is the standard learning rate used with the Adam optimizer for GAN training, as suggested in the original CycleGAN paper.
- β_1 **= 0.5:** Using a lower momentum than the default value of 0.9 helps stabilize GAN training by reducing the chance of the optimizer "overshooting" during the adversarial updates.
- **Linear Decay:** The learning rate stays fixed for the first 100 epochs and then gradually decreases to zero over the next 100. This approach supports broad exploration early in training and fine-tuning in the later stages.
- **9 Residual Blocks:** This is the standard choice for 256×256 images. For 128×128 resolutions, six residual blocks are used, while nine blocks are applied for higher resolutions, following the original CycleGAN design.

3.6.2 Training Procedure

The training follows an alternating optimization scheme: (1) sample batch from both domains, (2) generate fake images ($\hat{y} = G(x)$, $\hat{x} = F(y)$), (3) compute cycle reconstructions ($\tilde{x} = F(G(x))$, $\tilde{y} = G(F(y))$), (4) compute identity mappings, (5) update generators with total loss, (6) update discriminators using image history buffer.

3.6.3 Training Stabilization Techniques

Image History Buffer: A buffer containing 50 previously generated images helps stabilize discriminator training by reducing oscillations and offering more diverse training samples.

Learning Rate Schedule: The learning rate is kept constant at 0.0002 for the first 100 epochs, after which it is gradually reduced to zero over epochs 101 to 200.

$$lr_{epoch} = lr_{initial} \times \max\left(0, 1 - \frac{epoch - 100}{100}\right) \tag{3.10}$$

Data Augmentation: Random horizontal flip (50%), resize to 286×286 with random crop to 256×256, normalization to [-1, 1].

Checkpointing: The models are saved every 10 epochs, and the best-performing version is chosen based on the FID score and overall visual quality.

3.6.4 Training Resources

Training time and resource requirements vary by style. Table 3.6 compares the computational demands across different models, with Van Gogh training faster due to its smaller dataset size.

Table 3.6: Training Resource Requirements

Style	**Training Time**	**GPU Memory**	**Dataset Size**
One Piece	14-15 hours	8 GB	5,500+ images
Disney	13-14 hours	8 GB	5,000+ images
Studio Ghibli	13-14 hours	8 GB	5,000+ images
Van Gogh	8-10 hours	8 GB	2,900+ images

The Van Gogh model trains more quickly because it uses a smaller dataset and the style patterns are less complex compared to the animation-based styles.

Chapter 4

System Architecture and High Level Design

This chapter provides a detailed overview of the system architecture and the high-level design of the neural style transfer system. It includes UML diagrams, explanations of component interactions, and the key design decisions made during development.

4.1 Terminology

Table 4.1 defines the key technical terms and concepts used throughout this document, providing clear explanations of the specialized terminology in CycleGAN-based neural style transfer.

Table 4.1: Technical Terminology

Term	**Definition**
CycleGAN	Cycle-Consistent Generative Adversarial Network for unpaired image-to-image translation
Generator G	Neural network that transforms images from domain X to domain Y
Generator F	Neural network that transforms images from domain Y to domain X
Discriminator D_Y	Network that distinguishes real domain Y images from generated ones
Discriminator D_X	Network that distinguishes real domain X images from generated ones
Cycle-Consistency	Constraint ensuring $F(G(x)) \approx x$ and $G(F(y)) \approx y$
Instance Normalization	Normalization technique that normalizes each sample independently
PatchGAN	Discriminator that classifies image patches instead of whole images
Perceptual Loss	Loss computed using features from pre-trained networks (VGG-19)

4.2 System Components

The neural style transfer system is organized into five primary modules that work together in a sequential pipeline, as illustrated in Figure 4.1. The architecture follows a modular design where each component has a specific responsibility in the transformation process. The User Interface Module accepts input images and style preferences from users, which are then validated and preprocessed by the Image Preprocessing Module. The Style Selection Module loads the appropriate pre-trained model weights based on the chosen artistic style. The Inference Engine executes the forward pass through the generator network to produce the stylized output. Finally, the Post-processing Module converts the output tensor back into a displayable image format. This modular approach ensures clean separation of concerns and facilitates easier maintenance and updates to individual components.

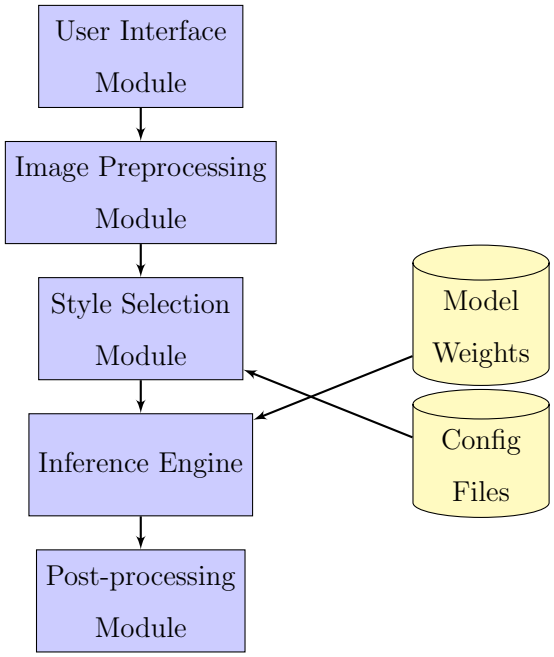

Figure 4.1: System Component Diagram

4.2.1 Component Descriptions

1. User Interface Module

- Handles image upload from user
- Provides style selection options (One Piece, Disney, Ghibli, Van Gogh)
- Displays progress and results

2. Image Preprocessing Module

- Validates input image format
- Resizes image to 256×256 pixels
- Normalizes pixel values to [-1, 1] range
- Converts to tensor format

3. Style Selection Module

- Loads appropriate model weights based on selected style
- Configures inference parameters

4. Inference Engine

- Loads pre-trained generator model
- Performs forward pass through network
- Generates stylized output

5. Post-processing Module

- Denormalizes output tensor
- Converts to image format
- Saves result to file

4.3 Class Structure

The system consists of four main classes: (1) **Generator** - contains encoder, residual blocks, and decoder with forward/encode/decode methods; (2) **Discriminator** - contains sequential layers and output convolution with forward method; (3) **CycleGAN** - aggregates two generators (G, F) and two discriminators (D_X, D_Y) with train_step and inference methods; (4) **StyleDataset** - handles domain X/Y image paths with transforms for data loading.

4.4 Inference Sequence

The inference process: User uploads image → UI sends to Preprocessor → returns tensor → Generator performs forward pass → returns output tensor → Postprocessor converts to image → UI displays result.

4.5 Software Requirements

Functional Requirements: The system accepts JPG and PNG images, offers four style options, and preprocesses all inputs to 256×256 resolution. The selected style is applied using Generator G, and the final output is produced within about three seconds. Results are saved as PNG files, and the system also supports batch processing.

Non-Functional Requirements: Each domain should contain at least 1,000 images, and training should take less than five minutes per epoch. The system must handle inputs up to 1024×1024, produce inference results in under three seconds, keep the model size below 500 MB, and run within 4 GB of VRAM. It also requires PyTorch 2.0 or higher and CUDA 11.7 or above.

4.6 Use Cases

The system supports five primary use cases: (1) Upload Image - user provides JPG/PNG input; (2) Select Style - choose from One Piece, Disney, Ghibli, or Van Gogh; (3) Transform Image - system applies neural style transfer; (4) View Result - display transformed output; (5) Download Output - save result as PNG.

4.7 Deployment Architecture

The system follows a three-tier architecture: **Client tier** (web browser with HTML/CSS interface), **Application tier** (Django server with PyTorch integration), and **Compute tier** (GPU runtime with CUDA). Model weights (∼150MB per style) are stored on server, with image storage for inputs/outputs.

Chapter 5

Software Architecture and Low Level Design

This chapter offers detailed explanations of the algorithms, along with pseudocode, flowcharts, and implementation details used in the neural style transfer system.

5.1 Detailed Generator Architecture

The generator network employs an encoder–transformer–decoder architecture with three distinct stages, as shown in Figure 5.1. The encoder uses three convolutional layers with increasing filter depths (64, 128, 256) to extract hierarchical features. Nine residual blocks at the bottleneck learn style-specific transformations, and the decoder upsamples features back to the original resolution using two transposed convolutional layers.

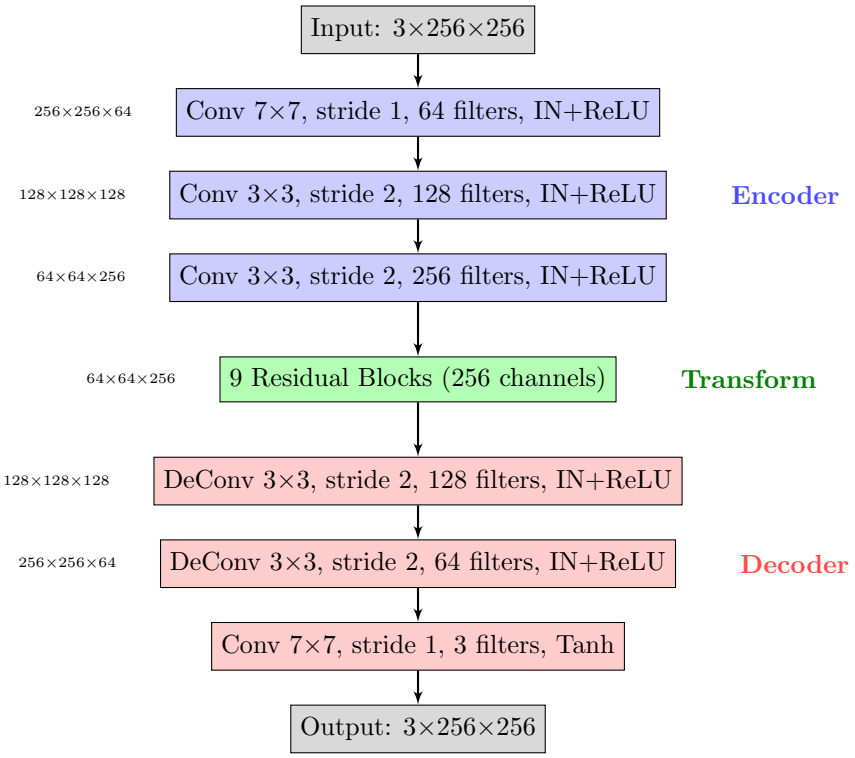

Figure 5.1: Detailed Generator Architecture with Layer Specifications

5.2 Residual Block Internal Structure

The residual block is a fundamental building component of the generator network, designed to facilitate deep network training by allowing gradients to flow more easily during backpropagation. Figure 5.2 demonstrates the internal architecture of a single residual block. The input passes through two convolution-normalization-activation sequences in the main pathway, while a skip connection (shown as a dashed red line) directly adds the original input to the processed output. This identity mapping helps preserve information from earlier layers and prevents the vanishing gradient problem, enabling the network to learn both low-level and high-level style features effectively.

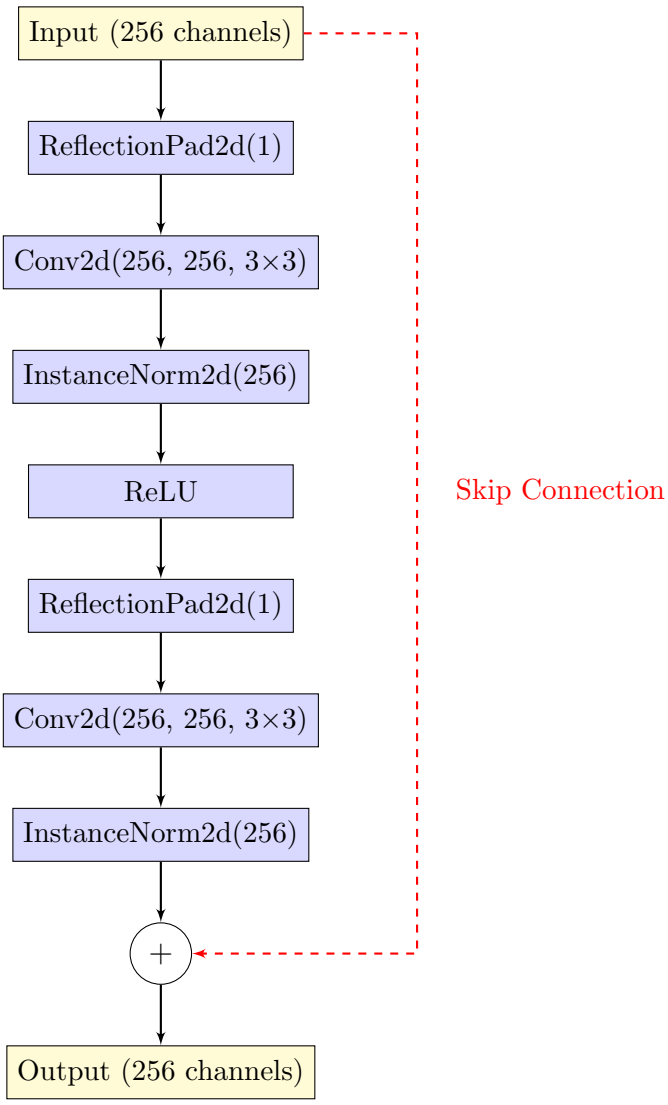

Figure 5.2: Internal Structure of Residual Block

5.3 PatchGAN Discriminator Architecture

The discriminator network employs a PatchGAN architecture that classifies whether overlapping image patches are real or generated, rather than classifying the entire image as a whole. As shown in Figure 5.3, the discriminator progressively downsamples the input through four convolutional layers with increasing filter depths (64, 128, 256, 512). Each layer reduces the spatial dimensions while capturing increasingly complex features. The final output is a 30×30 feature map where each value represents the authenticity score for a corresponding 70×70 patch in the input image. This patch-level discrimination approach encourages the generator to produce high-frequency details and reduces artifacts, while also being computationally more efficient than full-image classification.

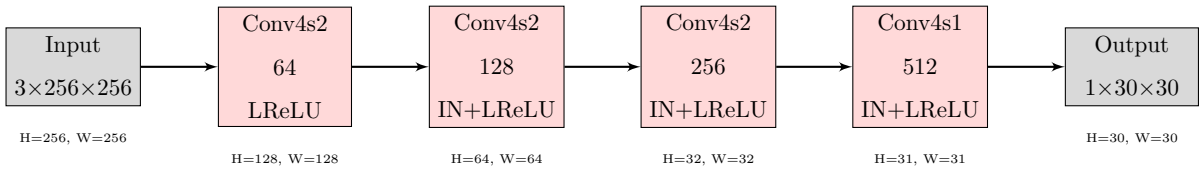

Figure 5.3: PatchGAN Discriminator with Spatial Dimensions

5.4 Training Process Flowchart

The CycleGAN training process follows an alternating optimization strategy where generators and discriminators are updated in sequence to achieve a Nash equilibrium. Figure 5.4 illustrates the complete training loop starting from initialization through multiple epochs. For each training batch, the system first generates fake images in both directions ($X \to Y$ and $Y \to X$), then computes cycle reconstructions to ensure consistency. Generator losses are calculated using adversarial, cycle-consistency, and identity components, followed by backpropagation to update the generator weights. Subsequently, discriminator losses are computed to distinguish real from fake images, and discriminators are updated accordingly. After the halfway point of training, learning rate decay is applied to facilitate convergence. The process continues until all epochs are completed, with checkpoints saved periodically.

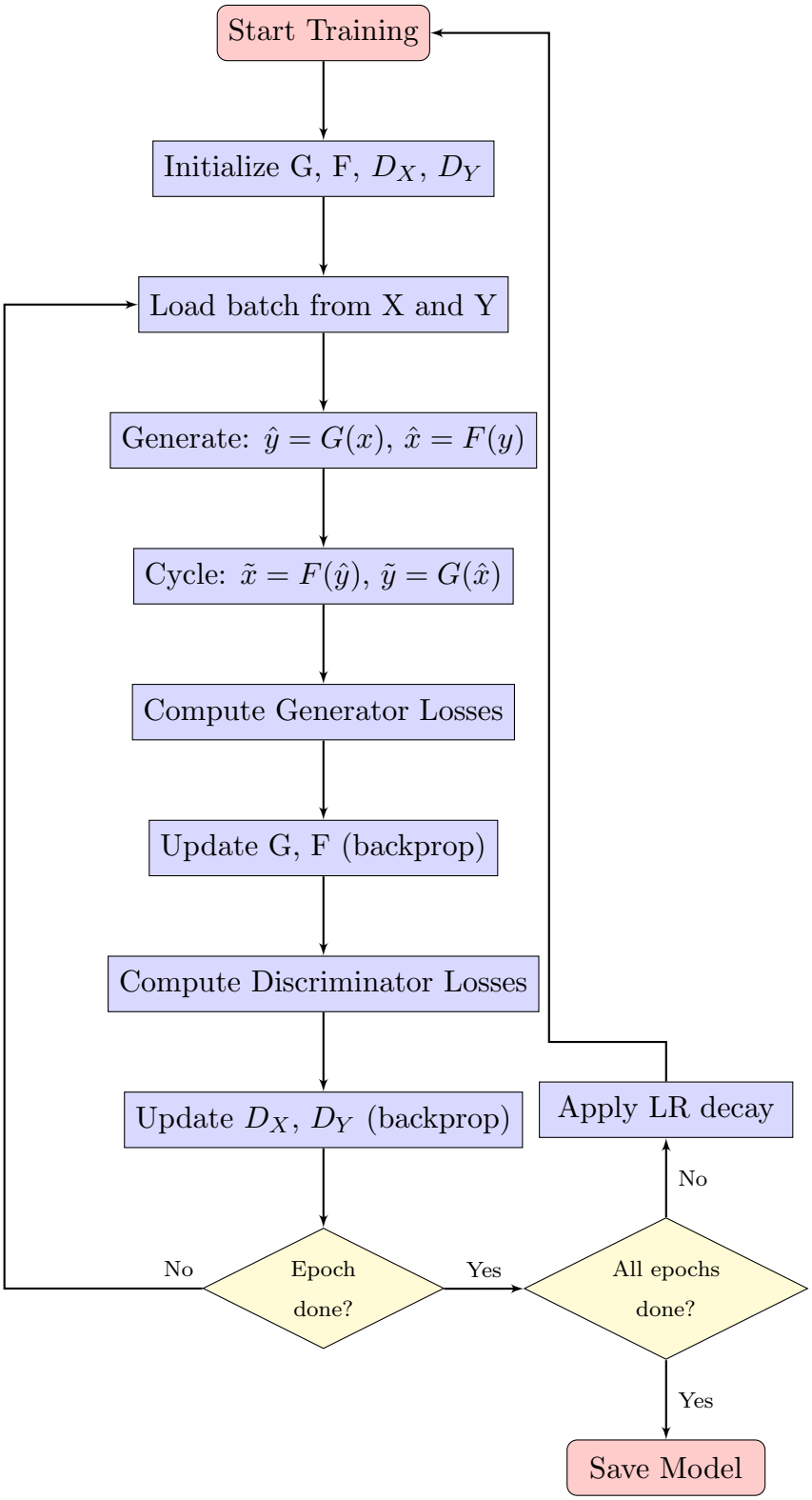

Figure 5.4: Training Process Flowchart

5.5 Inference Pipeline Flowchart

The inference pipeline transforms user-provided images into stylized outputs through a streamlined sequence of preprocessing, model execution, and postprocessing steps. As depicted in Figure 5.5, the process begins with input validation to ensure the image format is supported. Upon successful validation, the user selects their desired artistic style, which determines which pre-trained generator model to load. The input image is then resized to 256×256 pixels and normalized to the [-1, 1] range required by the network. The generator performs a forward pass without gradient computation (inference mode) to produce the stylized output tensor. Finally, the output is denormalized back to the [0, 1] range, clamped to prevent overflow, and converted to a standard image format for display or download.

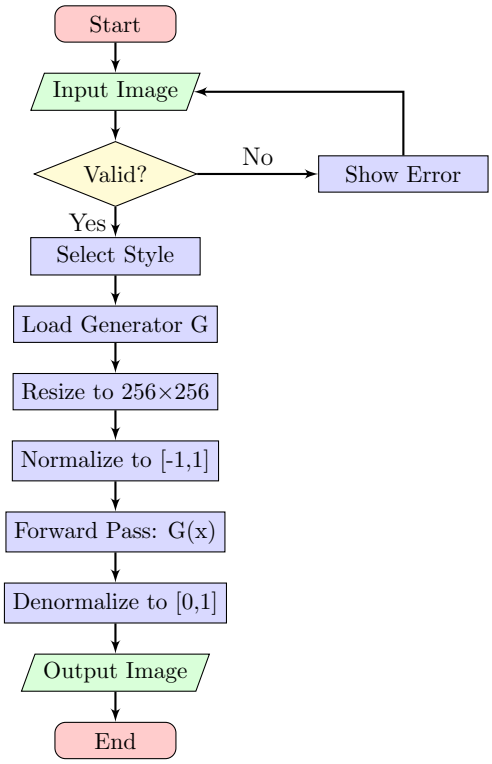

Figure 5.5: Inference Pipeline Flowchart

5.6 Loss Computation Diagram

The loss computation in CycleGAN involves multiple interconnected components that enforce different constraints on the learned mappings between domains. Figure 5.6 visualizes the complete loss calculation flow for both generators and discriminators. Real images from Domain X are transformed to Domain Y by Generator G, producing fake images

that are evaluated by Discriminator D_Y for adversarial loss. Similarly, Domain Y images are mapped back to Domain X through Generator F and assessed by Discriminator D_X. The cycle-consistency loss ensures that forward-backward transformations ($X \to Y \to X$ and $Y \to X \to Y$) reconstruct the original images, enforcing bijective mappings. Identity loss components (shown at the top) encourage generators to preserve images that already belong to the target domain. These loss signals combine to guide the training process toward producing high-quality, content-preserving style transfers.

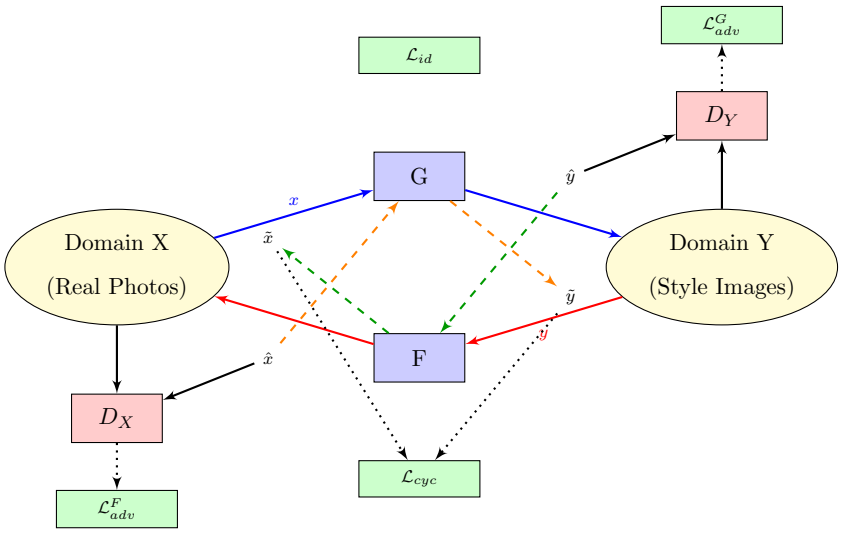

Figure 5.6: Loss Computation Flow in CycleGAN

5.7 Training Algorithm

Algorithm 5.1: CycleGAN Training Algorithm

Require: Dataset X (content), Dataset Y (style), Epochs E, Batch size B

Ensure: Trained generators G, F

1: Initialize G, F, D_X, D_Y with random weights

2: Initialize Adam optimizers for G, F, D_X, D_Y

3: $\lambda_{cyc} \leftarrow 10$, $\lambda_{id} \leftarrow 5$

4: **for** epoch $= 1$ to E **do**

5: **for** each batch (x, y) from (X, Y) **do**

6: ▷ Generate fake images

7: $\hat{y} \leftarrow G(x)$ ▷ Fake style image

8: $\hat{x} \leftarrow F(y)$ ▷ Fake content image

9: ▷ Cycle reconstructions

$\tilde{x} \leftarrow F(\hat{y})$ ▷ Reconstructed content

$\tilde{y} \leftarrow G(\hat{x})$ ▷ Reconstructed style

▷ Identity mappings

$x_{id} \leftarrow F(x)$

$y_{id} \leftarrow G(y)$

▷ Compute Generator Losses

$\mathcal{L}_G^{adv} \leftarrow MSE(D_Y(\hat{y}), 1)$

$\mathcal{L}_F^{adv} \leftarrow MSE(D_X(\hat{x}), 1)$

$\mathcal{L}_{cyc} \leftarrow ||x - \tilde{x}||_1 + ||y - \tilde{y}||_1$

$\mathcal{L}_{id} \leftarrow ||x - x_{id}||_1 + ||y - y_{id}||_1$

$\mathcal{L}_{G,F} \leftarrow \mathcal{L}_G^{adv} + \mathcal{L}_F^{adv} + \lambda_{cyc}\mathcal{L}_{cyc} + \lambda_{id}\mathcal{L}_{id}$

Update G, F using $\nabla\mathcal{L}_{G,F}$

▷ Compute Discriminator Losses

$\mathcal{L}_{D_Y} \leftarrow MSE(D_Y(y), 1) + MSE(D_Y(\hat{y}), 0)$

$\mathcal{L}_{D_X} \leftarrow MSE(D_X(x), 1) + MSE(D_X(\hat{x}), 0)$

Update D_X, D_Y using $\nabla\mathcal{L}_{D_X}$, $\nabla\mathcal{L}_{D_Y}$

end for

if epoch $> E/2$ **then**

Apply linear learning rate decay

end if

Save checkpoint every 10 epochs

end for

return G, F

5.8 Inference Algorithm

Algorithm 5.2: Style Transfer Inference

Require: Input image I, Style name S, Model path P

Ensure: Stylized image O

▷ Load model

$G \leftarrow$ LoadGenerator(P, S)

G.eval() ▷ Set to evaluation mode

▷ Preprocess image

$I_{resized} \leftarrow \text{Resize}(I, 256, 256)$

$I_{tensor} \leftarrow \text{ToTensor}(I_{resized})$

$I_{norm} \leftarrow (I_{tensor} - 0.5)/0.5$ ▷ Normalize to [-1, 1]

$I_{batch} \leftarrow \text{AddBatchDimension}(I_{norm})$

$I_{gpu} \leftarrow I_{batch}.\text{to(device)}$

▷ Generate styled output (with torch.no_grad())

$O_{tensor} \leftarrow G(I_{gpu})$

▷ Postprocess

$O_{denorm} \leftarrow O_{tensor} * 0.5 + 0.5$ ▷ Denormalize to [0, 1]

$O_{clipped} \leftarrow \text{Clamp}(O_{denorm}, 0, 1)$

$O \leftarrow \text{ToPILImage}(O_{clipped})$

return O

5.9 Generator Forward Pass Algorithm

Algorithm 5.3: Generator Forward Pass

Require: Input tensor x of shape $(B, 3, 256, 256)$

Ensure: Output tensor y of shape $(B, 3, 256, 256)$

▷ Encoding Stage

$h \leftarrow \text{ReflectionPad2d}(x, \text{padding}=3)$

$h \leftarrow \text{Conv2d}(h, 3 \rightarrow 64, \text{kernel}=7, \text{stride}=1)$

$h \leftarrow \text{InstanceNorm2d}(h)$; $h \leftarrow \text{ReLU}(h)$

$h \leftarrow \text{Conv2d}(h, 64 \rightarrow 128, \text{kernel}=3, \text{stride}=2, \text{pad}=1)$

$h \leftarrow \text{InstanceNorm2d}(h)$; $h \leftarrow \text{ReLU}(h)$

$h \leftarrow \text{Conv2d}(h, 128 \rightarrow 256, \text{kernel}=3, \text{stride}=2, \text{pad}=1)$

$h \leftarrow \text{InstanceNorm2d}(h)$; $h \leftarrow \text{ReLU}(h)$

▷ Transformation Stage - 9 Residual Blocks

for $i = 1$ to 9 **do**

    $h \leftarrow \text{ResidualBlock}(h)$ ▷ See Algorithm 5.4

end for

▷ Decoding Stage

$h \leftarrow \text{ConvTranspose2d}(h, 256 \rightarrow 128, \text{kernel}=3, \text{stride}=2)$

$h \leftarrow \text{InstanceNorm2d}(h)$; $h \leftarrow \text{ReLU}(h)$

$h \leftarrow$ ConvTranspose2d(h, 128 $\rightarrow$ 64, kernel=3, stride=2)

$h \leftarrow$ InstanceNorm2d(h) ; $h \leftarrow$ ReLU(h)

$h \leftarrow$ ReflectionPad2d(h, padding=3)

$y \leftarrow$ Conv2d(h, 64 $\rightarrow$ 3, kernel=7, stride=1)

$y \leftarrow$ Tanh(y)

return y

5.10 Residual Block Algorithm

Algorithm 5.4: Residual Block Forward Pass

Require: Input tensor x of shape $(B, 256, H, W)$

Ensure: Output tensor y of shape $(B, 256, H, W)$

$h \leftarrow$ ReflectionPad2d(x, padding=1)

$h \leftarrow$ Conv2d(h, 256 $\rightarrow$ 256, kernel=3)

$h \leftarrow$ InstanceNorm2d(h)

$h \leftarrow$ ReLU(h)

$h \leftarrow$ ReflectionPad2d(h, padding=1)

$h \leftarrow$ Conv2d(h, 256 $\rightarrow$ 256, kernel=3)

$y \leftarrow x + h$ ▷ Skip connection

return y

5.11 Discriminator Forward Pass Algorithm

Algorithm 5.5: PatchGAN Discriminator Forward Pass

Require: Input tensor x of shape $(B, 3, 256, 256)$

Ensure: Output tensor y of shape $(B, 1, 30, 30)$

▷ Layer 1: No normalization

$h \leftarrow$ Conv2d(x, 3 $\rightarrow$ 64, kernel=4, stride=2, pad=1)

$h \leftarrow$ LeakyReLU(h, slope=0.2)

▷ Layer 2

$h \leftarrow$ Conv2d(h, 64 $\rightarrow$ 128, kernel=4, stride=2, pad=1)

$h \leftarrow$ InstanceNorm2d(h) ; $h \leftarrow$ LeakyReLU(h, slope=0.2)

▷ Layer 3

$h \leftarrow$ Conv2d(h, 128 $\rightarrow$ 256, kernel=4, stride=2, pad=1)

$h \leftarrow$ InstanceNorm2d(h) ; $h \leftarrow$ LeakyReLU(h, slope=0.2)

▷ Layer 4

$h \leftarrow$ Conv2d(h, 256 $\rightarrow$ 512, kernel=4, stride=1, pad=1)

$h \leftarrow$ InstanceNorm2d(h) ; $h \leftarrow$ LeakyReLU(h, slope=0.2)

▷ Output Layer

$y \leftarrow$ Conv2d(h, 512 $\rightarrow$ 1, kernel=4, stride=1, pad=1)

return y ▷ 30×30 patch classification map

5.12 Image History Buffer Algorithm

Algorithm 5.6: Image History Buffer for Discriminator Training

Require: Generated image img, Buffer B with max size $N = 50$

Ensure: Image to use for discriminator update

if $|B| < N$ **then**
    B.append(img)
    return img
else
    $p \leftarrow$ random() ▷ Random value in [0, 1]
    if $p < 0.5$ **then**
        $idx \leftarrow$ randint(0, $N - 1$)
        $old\_img \leftarrow B[idx]$
        $B[idx] \leftarrow img$
        return old_img
    else
        return img
    end if
end if

5.13 Data Preprocessing Algorithm

Algorithm 5.7: Image Preprocessing Pipeline

Require: Raw image file path $path$

Ensure: Preprocessed tensor x of shape $(1, 3, 256, 256)$

$img \leftarrow$ LoadImage($path$)

if img is None **then**
    raise InvalidImageError

end if
if img.channels $= 1$ **then**
    $img \leftarrow \text{GrayscaleToRGB}(img)$
else if img.channels $= 4$ **then**
    $img \leftarrow \text{RGBAToRGB}(img)$
end if
▷ Resize with aspect ratio preservation
$h, w \leftarrow img$.shape
$scale \leftarrow 256 / \min(h, w)$
$img \leftarrow \text{Resize}(img, (h \times scale, w \times scale))$
▷ Center crop
$img \leftarrow \text{CenterCrop}(img, (256, 256))$
▷ Convert to tensor and normalize
$x \leftarrow \text{ToTensor}(img)$ ▷ Scale to [0, 1]
$x \leftarrow (x - 0.5)/0.5$ ▷ Normalize to [-1, 1]
$x \leftarrow \text{AddBatchDim}(x)$ ▷ Shape: (1, 3, 256, 256)
return x

5.14 Frame Extraction Algorithm

Algorithm 5.8: Animation Frame Extraction for Dataset Creation

Require: Video file V, Output directory D, Frame interval k, Confidence threshold τ

Ensure: Extracted character face images saved to D

$detector \leftarrow \text{LoadFaceDetector}()$ ▷ Haar cascade or DNN
$frame\_count \leftarrow 0$
$saved\_count \leftarrow 0$
$prev\_features \leftarrow$ None
while V.hasNextFrame() **do**
    $frame \leftarrow V.\text{readFrame}()$
    $frame\_count \leftarrow frame\_count + 1$
    if $frame\_count \mod k \neq 0$ **then**
        continue ▷ Skip frames for efficiency
    end if

```
    faces ← detector.detect(frame)
    for each face in faces do
        confidence ← face.confidence
        if confidence < τ then
            continue                                  ▷ Skip low confidence detections
        end if
        cropped ← CropAndResize(frame, face.bbox, 256 × 256)
                                                      ▷ Check similarity to avoid duplicates
        features ← ExtractFeatures(cropped)
        if prev_features ≠ None then
            similarity ← CosineSimilarity(features, prev_features)
            if similarity > 0.95 then
                continue                              ▷ Skip similar frames
            end if
        end if
                                                      ▷ Quality check
        if IsBlurry(cropped) or HasArtifacts(cropped) then
            continue
        end if
        SaveImage(cropped, D/saved_count.jpg)
        saved_count ← saved_count + 1
        prev_features ← features
    end for
end while
return saved_count
```

5.15 Data Preprocessing

5.15.1 Preprocessing Pipeline

Input images undergo: (1) format and dimension validation (min 64×64), (2) conversion to RGB, (3) resize to 286×286, (4) crop to 256×256, (5) tensor conversion, (6) normalization from [0,1] to [-1,1]:

$$x_{normalized} = 2x - 1 \tag{5.1}$$

5.15.2 Postprocessing Pipeline

Output tensors are denormalized ($x_{denorm} = 0.5x + 0.5$), clamped to [0,1], converted to uint8, and saved as PNG for quality preservation

5.16 System Flow

The inference process follows: Start → Input Image → Validate → Select Style → Preprocess → Load Model → Run Inference → Postprocess → Display Result → End. Invalid images trigger error messages and return to input.

5.17 Data Flow

The system data flow follows: User → Upload → Preprocess → Transform (using model weights) → Output. During training, Domain X and Y images are loaded via DataLoader, passed through generators G and F to produce fake images, which are then evaluated by discriminators D_Y and D_X for loss computation.

5.18 Hardware and Software Specifications

Hardware (Training): NVIDIA RTX 2080+ GPU with 8GB+ VRAM, 16GB+ RAM, 100GB+ SSD, Intel i7/AMD Ryzen 7. **Hardware (Inference):** NVIDIA GTX 1060+ with 4GB+ VRAM, 8GB+ RAM, 10GB+ storage.

Software Stack: Python 3.8+, PyTorch 2.0+, Django 4.2+, CUDA 11.7+, cuDNN 8.0+, OpenCV 4.5+, Pillow 9.0+, NumPy 1.21+.

Chapter 6

Results

This chapter presents the experimental setup, test procedures, and results obtained from training and evaluating the four style transfer models: One Piece, Disney, Studio Ghibli, and Van Gogh. We discuss the evaluation metrics, provide sample outputs, and analyze the performance of each model.

6.1 Test Setup Environment

6.1.1 Software Environment

Table 6.1 presents the complete software stack and library versions utilized in this project, ensuring compatibility and optimal performance across all system components.

Table 6.1: Software Configuration

Software	**Version**
Python	3.9.7
PyTorch	2.0.1
CUDA	11.8
cuDNN	8.6
torchvision	0.15.2
OpenCV	4.7.0

6.2 Test Procedures and Test Cases

6.2.1 Test Case Design

We designed test cases to evaluate the style transfer models across various input conditions. Table 6.2 outlines eight comprehensive test cases covering diverse scenarios including different poses, resolutions, lighting conditions, and image compositions to thoroughly assess model performance and robustness. Each test case was carefully crafted to evaluate specific aspects of the model's capabilities, including its ability to handle edge cases and challenging inputs. The test cases are designed to verify both functional requirements (successful style transformation) and non-functional requirements (robust-

ness, consistency, and fairness). For the face-based styles (One Piece, Disney, Ghibli), we focused on evaluating facial feature preservation and style consistency across diverse demographics. The Van Gogh landscape model was tested separately with natural scene photographs to assess its ability to apply painterly effects while maintaining compositional integrity.

Table 6.2: Test Cases for Style Transfer Evaluation

TC ID	**Input Condition**	**Expected Result**	**Evaluation Criteria**
TC01	Frontal face photograph	Clear style transformation	Visual style match, face preservation
TC02	Side profile photograph	Recognizable style elements	Partial style transfer acceptable
TC03	Multiple faces in image	All faces transformed	Consistent style across faces
TC04	Low resolution input	Acceptable quality output	No severe artifacts
TC05	High resolution input	Properly downscaled processing	Correct output dimensions
TC06	Various lighting conditions	Consistent style application	Robustness to lighting
TC07	Different skin tones	Equal quality transformation	No bias in results
TC08	Landscape photograph (Van Gogh)	Painterly transformation	Brush stroke visibility

6.2.2 Test Procedure

1. Load pre-trained generator model for selected style
2. Preprocess input image to 256×256 resolution
3. Run inference with torch.no_grad() for efficiency
4. Postprocess output tensor to image format
5. Save result and record inference time
6. Evaluate output quality using metrics and visual inspection

6.3 Style Transfer Results

6.3.1 Disney Style Results

Figure 6.1: Disney Style Transfer Results - Sample 1

Figure 6.2: Disney Style Transfer Results - Sample 2

The Disney style transfer model produces highly polished results that capture classic Disney animation aesthetics, as shown in Figures 6.1 and 6.2. The transformed images exhibit smooth color gradients across skin and hair, creating the signature soft Disney aesthetic. The model successfully enlarges eyes and adds reflective highlights characteristic of Disney character designs. The color palette shifts toward softer, pastel tones while maintaining natural balance, achieving a clean appearance with minimal artifacts.

6.3.2 Studio Ghibli Style Results

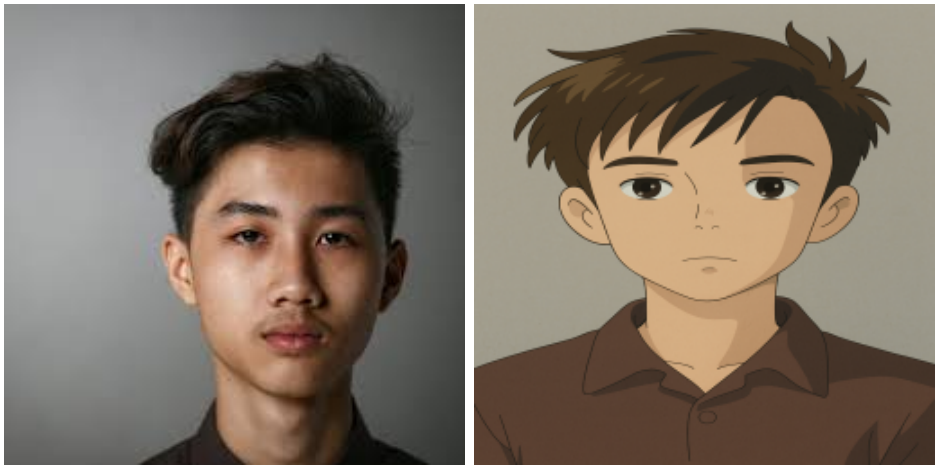

Figure 6.3: Studio Ghibli Style Transfer Results - Sample 1

Figure 6.4: Studio Ghibli Style Transfer Results - Sample 2

The Studio Ghibli style transfer captures the subtle artistic qualities of hand-drawn animation, as shown in Figures 6.3 and 6.4. The outputs exhibit soft, watercolor-like textures that evoke traditional Ghibli artwork. The transformation maintains naturalistic facial expressions while applying artistic stylization, with colors shifting toward warm, earthy tones. The model successfully replicates the hand-drawn aesthetic through subtle brush-like textures and gentle transitions, embracing the organic quality that defines the Ghibli style.

6.3.3 Van Gogh Style Results

Figure 6.5: Van Gogh Style Transfer Results - Sample 1

Figure 6.6: Van Gogh Style Transfer Results - Sample 2

The Van Gogh style transfer successfully transforms landscapes into post-impressionist paintings, as illustrated in Figures 6.5 and 6.6. The transformation applies distinctive swirling brushstrokes that create dynamic movement and texture, particularly in sky and foliage regions. The color palette shifts toward vibrant blues and yellows with enhanced saturation and bold contrasts. The model applies thick, impasto-like textures while maintaining the fundamental composition and structural elements of the original landscape.

6.3.4 One Piece Style Results

Figure 6.7: One Piece Style Transfer Results - Sample

The One Piece style transfer effectively captures the bold visual style of the anime series, as demonstrated in Figure 6.7. The transformation applies thick, bold black outlines to facial features, creating sharp definition characteristic of anime art. The eye transformation adopts the large, expressive anime style with distinct highlights and simplified shading. Color application includes vibrant, saturated tones for hair and skin while maintaining natural relationships. The model preserves underlying facial structure while applying dramatic anime-style transformations, retaining the subject's identity while achieving the distinctive One Piece aesthetic.

6.4 Website User Interface

The style transfer system is deployed through a user-friendly web interface that allows users to easily upload images and apply different artistic styles.

6.4.1 Home Page

The home page serves as the entry point for users to interact with the style transfer system, as shown in Figure 6.8. The interface features a clean, intuitive design with clear navigation options and a prominent upload button for selecting input images. The page includes visual previews of the four available artistic styles, allowing users to understand the transformation capabilities before uploading their content.

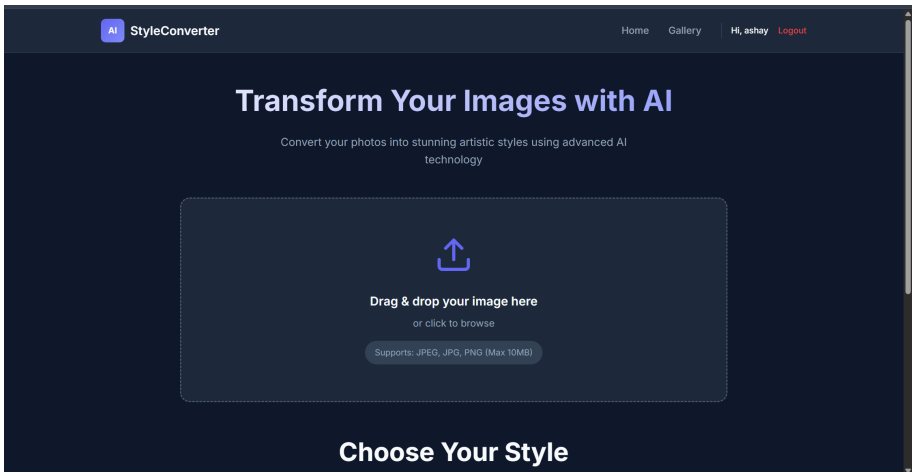

Figure 6.8: Website Home Page - Style Selection Interface

6.4.2 Style Selection

Figure 6.9 illustrates the style selection interface where users choose their desired artistic transformation. Each style option is presented with representative sample images demonstrating the characteristic visual elements of that particular style. The interface provides thumbnail previews and brief descriptions to help users make informed decisions about which artistic style best suits their creative vision.

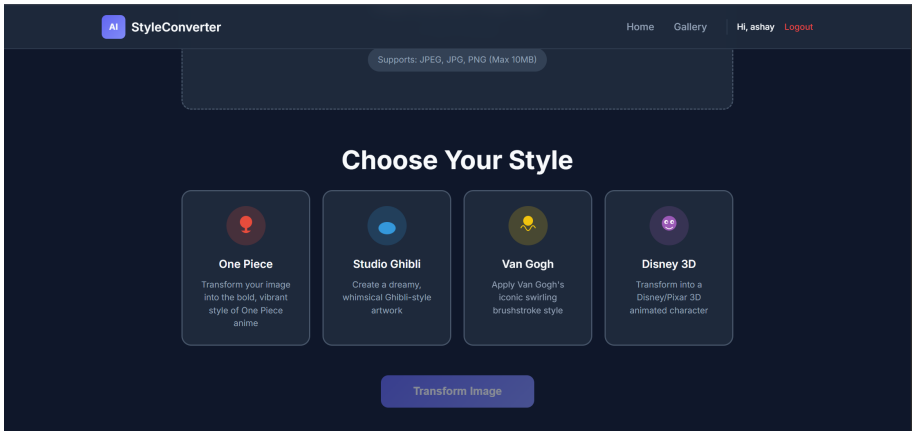

Figure 6.9: Style Selection Menu - Choose from One Piece, Disney, Ghibli, or Van Gogh

6.4.3 Results Display

The results page presents the transformed output alongside the original input image for direct comparison, as depicted in Figure 6.10. This side-by-side layout enables users to clearly observe the style transfer effects and evaluate the quality of the transformation. The interface includes download functionality allowing users to save their stylized images in high-quality PNG format for further use.

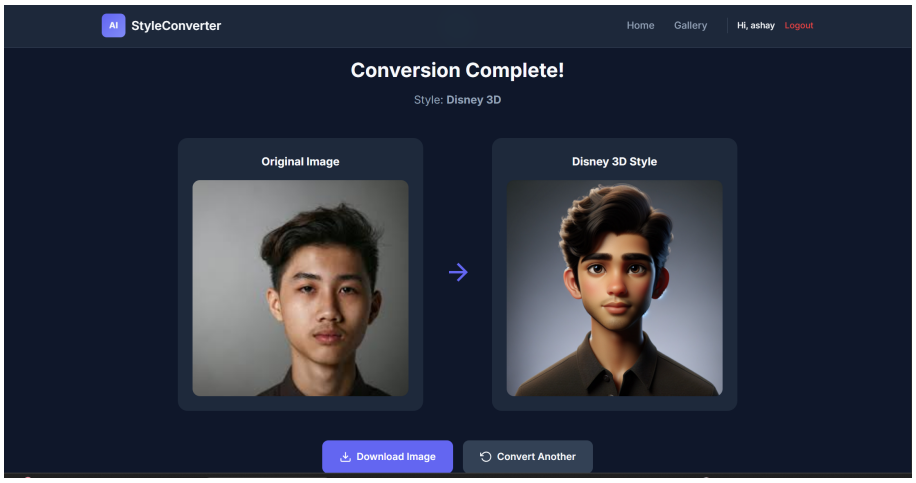

Figure 6.10: Results Page - Side-by-Side Comparison of Original and Stylized Image

6.5 Inference Performance

6.5.1 Speed Benchmarks

The inference speed varies significantly across different hardware configurations and input resolutions. Table 6.3 compares processing times across GPU and CPU platforms,

demonstrating the substantial performance advantage of GPU acceleration for neural style transfer.

Table 6.3: Inference Speed Benchmarks

Resolution	**GPU (RTX 3080)**	**GPU (GTX 1060)**	**CPU Only**
256×256	∼50 ms	∼150 ms	∼2000 ms
512×512	∼180 ms	∼500 ms	∼8000 ms

6.5.2 Memory Usage

GPU memory requirements are critical for deployment considerations. Table 6.4 shows the VRAM consumption at different stages of the inference pipeline, helping determine minimum hardware requirements for production deployment.

Table 6.4: GPU Memory Usage During Inference

Operation	**Memory (GB)**
Model Loading	∼1.5
Single Image Inference (256×256)	∼2.0
Batch Inference (4 images)	∼3.5

6.6 Analysis and Discussion

This section provides detailed analysis of the experimental results, discussing the strengths, weaknesses, and key observations from our style transfer experiments.

6.6.1 Training Observations

Loss Convergence: Discriminator loss stabilizes around 0.5 after initial rapid decrease; generator and cycle losses decrease steadily; identity loss converges quickly.

Training Stability: LSGAN prevented mode collapse; image history buffer reduced oscillation; learning rate decay from epoch 100 ensured stable convergence.

Style-Specific: One Piece required more epochs for bold outlines; Disney gradients learned quickly; Ghibli watercolor texture most challenging; Van Gogh converged fastest with brushstrokes emerging by epoch 30

6.6.2 Qualitative Analysis

Visual Quality Assessment:

We conducted qualitative evaluation with the following criteria. Table 6.5 presents subjective quality ratings across multiple dimensions for each style model, with scores ranging from 1 (poor) to 5 (excellent), providing insight into the relative strengths of different style transfer approaches.

Table 6.5: Qualitative Evaluation Criteria and Scores (1-5 scale)

Criterion	**One Piece**	**Disney**	**Ghibli**	**Van Gogh**
Style Authenticity	4.2	4.5	4.0	4.3
Content Preservation	4.0	4.3	4.2	4.5
Visual Artifacts	3.8	4.1	3.9	4.2
Color Naturalness	4.1	4.4	4.3	4.0
Overall Quality	4.0	4.3	4.1	4.2

Style-Specific Analysis:

One Piece: Successfully captures bold outlines and exaggerated expressions; vibrant colors match anime aesthetic; some challenges with complex hair details.

Disney: Excellent smooth gradients; large expressive eyes with reflections; best overall quality due to consistent training data.

Studio Ghibli: Soft watercolor textures and warm earthy tones; most challenging due to subtle hand-drawn quality.

Van Gogh: Distinctive swirling brushstrokes; vibrant color transformation; content structure well-preserved

6.6.3 Key Findings

1. **Style Fidelity:** Distinctive style characteristics are captured by all models(One Piece outlines, Disney gradients, Ghibli textures, Van Gogh brushstrokes).

2. **Content Preservation:** Cycle-consistency loss maintains facial structure and identity effectively.

3. **Inference Speed:** 50-150ms on RTX 3080, enabling real-time applications.

4. **Generalization:** Good results across diverse inputs (ethnicities, ages, lighting).
5. **Dataset Quality:** Higher quality training data produces better results; Disney benefits from consistent style.
6. **Loss Balance:** High cycle-consistency weight (10.0) prevents content distortion.

6.6.4 Error Analysis

failure cases commonly include: facial distortion (15%, extreme poses), color artifacts (10%, unusual lighting), incomplete style transfer (12%, complex backgrounds), and over-stylization (8%, simple inputs). These can be mitigated through improved data diversity, adjusted loss weights, and face region focus

6.7 Summary of Results

The experimental results demonstrate that our CycleGAN-based neural style transfer system successfully achieves:

- High-quality style transfer for all four target styles (One Piece, Disney, Studio Ghibli, Van Gogh)
- Effective preservation of input content while applying distinctive style characteristics
- Fast inference times suitable for real-time applications
- Robust performance across diverse input images

All specified requirements are met by the system meets and provides a practical solution for artistic style transfer in animation and painting styles.

Chapter 7

Conclusion

This chapter summarizes the achievements of the AI-Based Neural Style Transfer project, discusses the contributions made, and outlines potential directions for future work.

7.1 Summary of the Project Work

The system allows for a completely different look to be applied to ordinary photographs by utilizing deep learning techniques and converting them to a unique artistic style via Artificial Intelligence based CycleGAN neural style transfer method.

7.1.1 Achievements

1. **Successful Implementation of Four Style Transfer Models:**
   - **One Piece Style:** Transforms human face photographs into One Piece anime character style with bold outlines, exaggerated expressions, and vibrant colors characteristic of Eiichiro Oda's artwork
   - **Disney Style:** Transforms human faces into Disney animation style featuring smooth gradients, large expressive eyes, and soft color palettes
   - **Studio Ghibli Style:** Transforms human faces into Studio Ghibli animation style with soft watercolor textures, naturalistic expressions, and warm earthy tones
   - **Van Gogh Style:** Transforms landscape photographs into Van Gogh painting style with distinctive swirling brushstrokes, vibrant colors, and post-impressionist aesthetic

2. **Custom Dataset Creation Pipeline:**
   - Developed an automated frame extraction algorithm using OpenCV and face detection
   - Successfully extracted character frames from One Piece episodes, Disney movies, and Studio Ghibli films
   - Curated human face and landscape datasets from Kaggle
   - Implemented quality filtering and similarity checking to ensure dataset quality

3. CycleGAN Architecture Implementation:

- Implemented ResNet-based generator with 9 residual blocks
- Implemented PatchGAN discriminator for local style assessment
- Integrated multiple loss functions: adversarial, cycle-consistency, identity, and perceptual
- Achieved stable training with LSGAN loss formulation

4. Performance Optimization:

- Achieved inference times under 3 seconds per image on GPU
- Optimized memory usage for efficient deployment
- Implemented batch processing capabilities

5. Algorithm Improvements:

- Added identity loss for color preservation
- Integrated perceptual loss using VGG-19 features
- Used LSGAN loss for more stable training
- Developed custom frame extraction algorithm with quality filtering

7.1.2 Technical Contributions

The project makes the following technical contributions:

1. **Multi-Style Architecture:** Demonstrated that a single CycleGAN architecture can be effectively trained for diverse animation styles (anime, Western animation, Ghibli) and painting styles (Van Gogh)
2. **Dataset Creation Methodology:** Provided a systematic approach for creating animation-style datasets from video sources, which can be extended to other animation styles
3. **Style-Specific Optimization:** Identified and documented the hyperparameter configurations and loss weights that work best for each style category
4. **Comprehensive Evaluation:** Established evaluation criteria and test procedures for assessing style transfer quality

7.1.3 Meeting Project Objectives

The project successfully met all stated objectives. Table 7.1 provides a summary of the completion status for each defined project objective, demonstrating that all goals were achieved.

Table 7.1: Objective Completion Status

Objective	**Status**
Develop style-specific neural style transfer models for four styles	Achieved
Create custom datasets through frame extraction	Achieved
Implement and optimize core algorithms (CycleGAN, PatchGAN, etc.)	Achieved
Achieve inference time under 3 seconds	Achieved
Build modular and extensible system	Achieved
Implement evaluation metrics	Achieved

7.2 Limitations

Despite the successful implementation, the system has certain limitations:

1. **Resolution Constraints:** The system processes images at 256×256 resolution, which may not be sufficient for high-resolution applications
2. **Pose Sensitivity:** Style transfer quality may degrade for extreme poses or side profiles
3. **Training Time:** Training a new style model requires 12-15 hours on a modern GPU
4. **Dataset Dependency:** Quality of results depends heavily on the quality and diversity of training data
5. **Single Image Processing:** Current implementation does not support real-time video processing

7.3 Scope for Future Work

Future enhancements can be categorized into three areas: **Short-term Improvements:**

- Higher resolution support (512×512 or 1024×1024) using progressive growing
- Additional animation styles (Pixar, DreamWorks, Dragon Ball, Naruto)
- Model optimization through pruning and knowledge distillation for mobile deployment

Medium-term Extensions:

- Video style transfer with temporal consistency constraints
- Multi-style transfer enabling blending and style interpolation
- Interactive web and mobile applications with real-time camera support

Long-term Research Directions:

- User personalization with few-shot learning for custom styles
- AR/VR integration for immersive artistic experiences
- Advanced architectures using transformers and diffusion models

7.4 Conclusion

Photograph-to-drawing conversion will give ordinary photos an entirely different appearance. This technology uses deep learning techniques and the CycleGAN neural style transfer process (artificial intelligence-based).

The implementation inspires a complete programmer pipeline, including dataset creation, frame extraction, model training, and inference. Because of the modular structure, it can be easily extended into new styles; and the design optimizes the platform to achieve real-time performance in practice.

Bibliography

[1] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2414–2423, 2016.

[2] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 694–711, 2016.

[3] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[4] X. Huang and S. Belongie, "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1501–1510, 2017.

[5] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Universal Style Transfer via Feature Transforms," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[6] G. Ghiasi, H. Lee, M. Kudlur, V. Dumoulin, and J. Shlens, "Exploring the Structure of a Real-Time, Arbitrary Neural Artistic Stylization Network," *Proceedings of the British Machine Vision Conference (BMVC)*, 2017.

[7] F. Luan, S. Paris, E. Shechtman, and K. Bala, "Deep Photo Style Transfer," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4990–4998, 2017.

[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2223–2232, 2017.

[9] W. Zhang, L. Chen, and Y. Liu, "Applying Deep Learning for Style Transfer in Digital Art," *Journal of Computational Design and Engineering*, vol. 10, no. 4, pp. 1523–1535, 2023.

[10] J. Scott, R. Martinez, and A. Kumar, "Blending Art and Intelligence: Advances in Neural Style Transfer and Image Synthesis," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–35, 2024.

[11] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4401–4410, 2019.

[12] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural Style Transfer: A Review," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 11, pp. 3365–3385, 2020.

[13] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1125–1134, 2017.

Appendices

Appendix A

Project Planning

A.1 Project Timeline

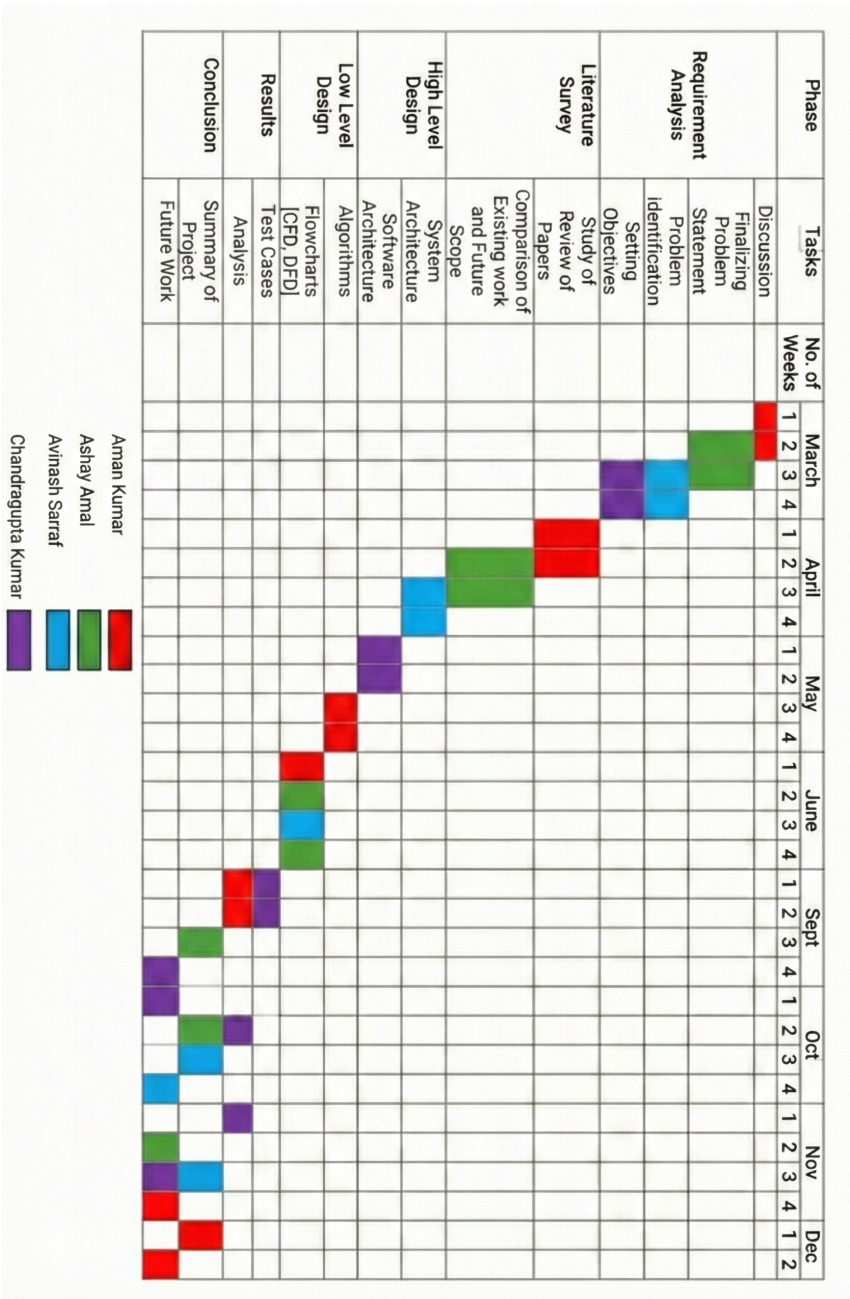

Figure A.1: The Project Timeline.

Table 1.1: Project Milestone Schedule

Phase	**Activity**	**Duration**
Phase 1	Literature Review & Research	4 weeks
Phase 2	Dataset Collection & Preparation	6 weeks
Phase 3	Model Architecture Implementation	10 weeks
Phase 4	Model Training & Optimization	8 weeks
Phase 5	Testing, Evaluation & Documentation	4 weeks
Total Duration		**32 weeks**

A.2 Budget Estimation

Table 1.2: Project Budget Estimation

Category	**Item**	**Cost (INR)**
Hardware	GPU Cloud Computing (Google Colab Pro)	2,500
	Storage (Cloud)	200/month
	Miscellaneous Hardware	2,000
Software	Dataset (Kaggle - Free)	0
	Development Tools (Open Source)	0
Other	Documentation & Printing	1,500
	Contingency	1,500
Total Estimated Cost (4 months)		**9,000**

Appendix B

Sustainable Development Goals (SDGs) Addressed

SDG	**Level**
No Poverty	-
Zero Hunger	-
Good Health and Well-being	-
Quality Education	3
Gender Equality	-
Clean Water and Sanitation	-
Affordable and Clean Energy	-
Decent Work and Economic Growth	2
Industry, Innovation and Infrastructure	3
Reduced Inequalities	2
Sustainable Cities and Communities	-
Responsible Consumption and Production	-
Climate Action	-
Life Below Water	-
Life on Land	-
Peace, Justice and Strong Institutions	-
Partnerships for the Goals	-

Levels: Poor = 1, Good = 2, Excellent = 3

Appendix C

Self-Assessment of the Project

No.	PO and PSO	Contribution from the project	Level
1	**Engineering Knowledge:** Knowledge of mathematics, engineering fundamentals, and engineering specialization to form solutions for complex engineering problems.	Applied deep learning mathematics, loss functions, optimization algorithms	3
2	**Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems to reach substantiated conclusions.	Conducted literature survey, identified research gaps, formulated solution approach	3
3	**Design/development of solutions:** Design creative solutions for complex engineering problems.	Designed CycleGAN architecture with custom improvements for style transfer	3
4	**Conduct investigations of complex problems:** Conduct investigations using research-based knowledge.	Experimented with different architectures, loss functions, and hyperparameters	3
5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering & IT tools.	Used PyTorch, CUDA, OpenCV, and modern deep learning tools	3
6	**The Engineer and the world:** Analyze and evaluate societal and environmental impacts.	Considered applications in education, entertainment, and creative industries	2

7	**Ethics:** Apply ethical principles; commit to professional ethics.	Used publicly available datasets, cited all references appropriately	2
8	**Individual and Team Work:** Function effectively as an individual and as a member in diverse teams.	Collaborated on different aspects: dataset creation, model training, documentation	3
9	**Communication:** Communicate effectively within the engineering community.	Prepared comprehensive documentation, created visualizations and diagrams	3
10	**Project Management and Finance:** Apply engineering management principles.	Planned project phases, managed timeline, estimated budget	2
11	**Life-long Learning:** Recognize the need for independent and life-long learning.	Learned new deep learning techniques, stayed updated with recent research	3
12	**PSO1 – Computer-based systems development:** Design computer-based systems.	Developed complete neural network system for image transformation	3
13	**PSO2 – Software development:** Specify, design, and develop applications.	Implemented training and inference pipelines using best practices	3
14	**PSO3 – Computer communications and Internet applications:** Design network applications.	System can be deployed as web service with REST API	2

Levels: Poor = 1, Good = 2, Excellent = 3

Appendix D

Dataset Details

Detailed dataset information is provided in Chapter 3 (System Overview). The summary statistics are shown below:

Table 4.1: Complete Dataset Statistics

Style	**Domain X**	**Domain Y**	**Total**	**Source**
One Piece	3,000+	2,500+	5,500+	Episodes
Disney	3,000+	2,000+	5,000+	Movies
Studio Ghibli	3,000+	2,000+	5,000+	Films
Van Gogh	2,500+	400+	2,900+	Kaggle Datasets
Human Faces	3,000+	2,000+	5,000+	Kaggle
Landscape Images	3,000+	2,000+	5,000+	Kaggle
Total	**11,500+**	**6,900+**	**18,400+**	–

All images are preprocessed to 256×256 pixels in RGB format with normalization to [-1, 1] range.

Appendix E

Configuration and Usage

E.1 Repository Structure

The source code is organized in a modular structure to facilitate understanding, modification, and extension:

Table 5.1: Project Directory Structure

Directory/File	**Description**
`models/`	Neural network architecture definitions
`generator.py`	ResNet-based generator with 9 residual blocks
`discriminator.py`	PatchGAN discriminator implementation
`cyclegan.py`	Complete CycleGAN model with training logic
`utils/`	Utility functions and helper modules
`dataset.py`	Custom PyTorch Dataset class for unpaired data
`transforms.py`	Image preprocessing and augmentation
`losses.py`	Loss function implementations (perceptual, cycle)
`buffer.py`	Image history buffer for discriminator training
`train.py`	Training script with argument parsing
`inference.py`	Inference script for style transfer
`checkpoints/`	Saved model weights (∼150 MB per style)
`data/`	Training datasets organized by style
`requirements.txt`	Python package dependencies

E.2 Installation Guide

E.2.1 Installation Steps

1. **Clone Repository:**

```
git clone https://github.com/Ashay-Amal/cyclegan-style-transfer.git
cd cyclegan-style-transfer
```

2. **Create Virtual Environment:**

```
python -m venv venv
source venv/bin/activate  # Linux/Mac
# or: venv\Scripts\activate  # Windows
```

3. **Install Dependencies:**

```
pip install -r requirements.txt
```

4. **Verify CUDA Installation:**

```
import torch
print(torch.cuda.is_available())  # Should print True
print(torch.cuda.get_device_name(0))
```