

**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

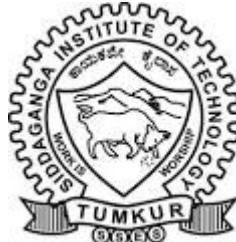
**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**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

Assistant Professor

Dept. of CSE

SIT,Tumakuru-03

Dr. N R Sunitha

Head of the Department

Dept. of CSE

SIT,Tumakuru-03

Dr. S V Dinesh

Principal

SIT,Tumakuru-03

### **External viva:**

#### **Names of the Examiners**

1.

2.

#### **Signature with date**

## ACKNOWLEDGEMENT

We offer our humble pranams at the lotus feet of **His Holiness, Dr. Sree Sivakumar 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 Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
CO-1											3			
CO-2		3												
CO-3											3			
CO-4						3								
CO-5	3	3												
CO-6					3							3		
CO-7			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:

- 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.

# Abstract

This project presents a Neural Style Transfer system using CycleGAN (Cycle-Consistent Generative Adversarial Networks) to transform photographs into four artistic styles: One Piece anime, Disney animation, Studio Ghibli animation, and Van Gogh painting. The system transforms human faces into animated characters and landscapes into post-impressionist paintings using unpaired datasets with bidirectional mappings through cycle-consistency constraints.

The implementation includes dual generators and discriminators, adversarial loss for realistic generation, cycle-consistency loss for content preservation, identity loss for color preservation, and perceptual loss using VGG-19. A custom frame extraction algorithm using OpenCV creates animation datasets from video sources, while Van Gogh data combines landscapes and paintings from Kaggle.

Built with PyTorch 2.0 and CUDA acceleration, the system achieves inference times under 3 seconds per image on NVIDIA GPUs. The modular architecture enables extension to additional styles, and results demonstrate successful style capture while preserving semantic content.

**Keywords:** Neural Style Transfer, CycleGAN, Generative Adversarial Networks, Deep Learning, Image-to-Image Translation, Computer Vision, Animation Style Transfer, Van Gogh Style

# Contents

<b>Abstract</b>	i
<b>List of Figures</b>	ii
<b>List of Tables</b>	iii
<b>1 Introduction</b>	1
1.1 Background . . . . .	1
1.1.1 Evolution of Computer Vision . . . . .	1
1.1.2 Convolutional Neural Networks . . . . .	2
1.1.3 Generative Adversarial Networks . . . . .	2
1.1.4 Style Transfer Approaches . . . . .	2
1.2 Motivation . . . . .	2
1.3 Problem Statement . . . . .	3
1.4 Objective of the Project . . . . .	4
1.5 Scope of the Project . . . . .	5
1.6 Methodology Overview . . . . .	6
1.7 Organisation of the Report . . . . .	7
<b>2 Literature Survey</b>	9
<b>3 System Overview</b>	13
3.1 System Architecture Overview . . . . .	13
3.2 Style Transfer Modes . . . . .	13
3.2.1 One Piece Style Transfer . . . . .	13
3.2.2 Disney Style Transfer . . . . .	14
3.2.3 Studio Ghibli Style Transfer . . . . .	14
3.2.4 Van Gogh Style Transfer . . . . .	14
3.3 Dataset Creation Methodology . . . . .	15
3.3.1 Animation Frame Extraction Pipeline . . . . .	15
3.3.2 Dataset Details . . . . .	16

3.3.3	Dataset Summary . . . . .	16
3.4	CycleGAN Architecture Details . . . . .	16
3.4.1	Overall Architecture . . . . .	16
3.4.2	Generator Network Architecture . . . . .	17
3.4.3	Residual Block Architecture . . . . .	17
3.4.4	Discriminator Network Architecture (PatchGAN) . . . . .	17
3.5	Loss Functions . . . . .	18
3.5.1	Adversarial Loss (LSGAN) . . . . .	18
3.5.2	Cycle-Consistency Loss . . . . .	19
3.5.3	Identity Loss . . . . .	19
3.5.4	Perceptual Loss . . . . .	20
3.5.5	Total Generator Loss . . . . .	20
3.5.6	Discriminator Loss . . . . .	21
3.6	Training Configuration . . . . .	21
3.6.1	Hyperparameter Selection . . . . .	21
3.6.2	Training Procedure . . . . .	22
3.6.3	Training Stabilization Techniques . . . . .	22
3.6.4	Training Resources . . . . .	23
<b>4</b>	<b>System Architecture and High Level Design</b>	<b>24</b>
4.1	Terminology . . . . .	24
4.2	System Components . . . . .	25
4.2.1	Component Descriptions . . . . .	25
4.3	Class Structure . . . . .	26
4.4	Inference Sequence . . . . .	26
4.5	Software Requirements . . . . .	26
4.6	Use Cases . . . . .	27
4.7	Deployment Architecture . . . . .	27
<b>5</b>	<b>Software Architecture and Low Level Design</b>	<b>28</b>
5.1	Detailed Generator Architecture . . . . .	28
5.2	Residual Block Internal Structure . . . . .	29
5.3	PatchGAN Discriminator Architecture . . . . .	29

5.4	Training Process Flowchart . . . . .	30
5.5	Inference Pipeline Flowchart . . . . .	31
5.6	Loss Computation Diagram . . . . .	31
5.7	Training Algorithm . . . . .	32
5.8	Inference Algorithm . . . . .	33
5.9	Generator Forward Pass Algorithm . . . . .	33
5.10	Residual Block Algorithm . . . . .	34
5.11	Discriminator Forward Pass Algorithm . . . . .	35
5.12	Image History Buffer Algorithm . . . . .	35
5.13	Data Preprocessing Algorithm . . . . .	36
5.14	Frame Extraction Algorithm . . . . .	37
5.15	Data Preprocessing . . . . .	38
5.15.1	Preprocessing Pipeline . . . . .	38
5.15.2	Postprocessing Pipeline . . . . .	38
5.16	System Flow . . . . .	38
5.17	Data Flow . . . . .	38
5.18	Hardware and Software Specifications . . . . .	38
<b>6</b>	<b>Results</b>	<b>40</b>
6.1	Test Setup Environment . . . . .	40
6.1.1	Hardware Configuration . . . . .	40
6.1.2	Software Environment . . . . .	40
6.2	Test Procedures and Test Cases . . . . .	41
6.2.1	Test Case Design . . . . .	41
6.2.2	Test Procedure . . . . .	41
6.3	Style Transfer Results . . . . .	42
6.3.1	Disney Style Results . . . . .	42
6.3.2	Studio Ghibli Style Results . . . . .	43
6.3.3	Van Gogh Style Results . . . . .	44
6.3.4	One Piece Style Results . . . . .	45
6.4	Website User Interface . . . . .	46
6.4.1	Home Page . . . . .	46

---

6.4.2	Style Selection . . . . .	46
6.4.3	Results Display . . . . .	47
6.5	Inference Performance . . . . .	47
6.5.1	Speed Benchmarks . . . . .	47
6.5.2	Memory Usage . . . . .	47
6.6	Analysis and Discussion . . . . .	48
6.6.1	Training Observations . . . . .	48
6.6.2	Qualitative Analysis . . . . .	48
6.6.3	Key Findings . . . . .	49
6.6.4	Error Analysis . . . . .	49
6.7	Summary of Results . . . . .	49
<b>7</b>	<b>Conclusion</b>	<b>50</b>
7.1	Summary of the Project Work . . . . .	50
7.1.1	Achievements . . . . .	50
7.1.2	Technical Contributions . . . . .	51
7.1.3	Meeting Project Objectives . . . . .	52
7.2	Limitations . . . . .	52
7.3	Scope for Future Work . . . . .	53
7.4	Conclusion . . . . .	53
<b>Bibliography</b>		<b>53</b>
<b>Appendices</b>		<b>56</b>
<b>A</b>	<b>Project Planning</b>	<b>57</b>
A.1	Project Timeline . . . . .	57
A.2	Budget Estimation . . . . .	58
<b>B</b>	<b>Sustainable Development Goals (SDGs) Addressed</b>	<b>59</b>
<b>C</b>	<b>Self-Assessment of the Project</b>	<b>60</b>
<b>D</b>	<b>Dataset Details</b>	<b>62</b>

<b>E Configuration and Usage</b>	<b>63</b>
E.1 Repository Structure . . . . .	63
E.2 Installation Guide . . . . .	63
E.2.1 Installation Steps . . . . .	63

# List of Figures

1.1	Project Methodology Overview	6
3.1	High-Level System Architecture	13
3.2	Frame Extraction Pipeline	15
3.3	Complete CycleGAN Architecture for Style Transfer	16
3.4	Generator Network Architecture	17
3.5	Residual Block Structure	17
3.6	PatchGAN Discriminator Architecture	17
4.1	System Component Diagram	25
5.1	Detailed Generator Architecture with Layer Specifications	28
5.2	Internal Structure of Residual Block	29
5.3	PatchGAN Discriminator with Spatial Dimensions	29
5.4	Training Process Flowchart	30
5.5	Inference Pipeline Flowchart	31
5.6	Loss Computation Flow in CycleGAN	31
6.1	Disney Style Transfer Results - Sample 1	42
6.2	Disney Style Transfer Results - Sample 2	42
6.3	Studio Ghibli Style Transfer Results - Sample 1	43
6.4	Studio Ghibli Style Transfer Results - Sample 2	43
6.5	Van Gogh Style Transfer Results - Sample 1	44
6.6	Van Gogh Style Transfer Results - Sample 2	44
6.7	One Piece Style Transfer Results - Sample	45
6.8	Website Home Page - Style Selection Interface	46
6.9	Style Selection Menu - Choose from One Piece, Disney, Ghibli, or Van Gogh	46
6.10	Results Page - Side-by-Side Comparison of Original and Stylized Image	47
A.1	The Project Timeline.	57

# List of Tables

3.1	Complete Dataset Summary . . . . .	16
3.2	Generator Network Layers . . . . .	17
3.3	Discriminator Network Layers . . . . .	18
3.4	Loss Function Weights . . . . .	20
3.5	Training Hyperparameters . . . . .	21
3.6	Training Resource Requirements . . . . .	23
4.1	Technical Terminology . . . . .	24
6.1	Test Environment Hardware . . . . .	40
6.2	Software Configuration . . . . .	40
6.3	Test Cases for Style Transfer Evaluation . . . . .	41
6.4	Inference Speed Benchmarks . . . . .	47
6.5	GPU Memory Usage During Inference . . . . .	47
6.6	Qualitative Evaluation Criteria and Scores (1-5 scale) . . . . .	48
7.1	Objective Completion Status . . . . .	52
1.1	Project Milestone Schedule . . . . .	58
1.2	Project Budget Estimation . . . . .	58
4.1	Complete Dataset Statistics . . . . .	62
5.1	Project Directory Structure . . . . .	63

# Chapter 1

## Introduction

The intersection of artificial intelligence and creative arts has opened unprecedented possibilities for digital content creation. Neural Style Transfer (NST) represents one of the most fascinating applications of deep learning, enabling the transformation of ordinary photographs into stunning artistic renditions by learning and applying the visual characteristics of various art styles. This project focuses on developing a comprehensive style transfer system using CycleGAN (Cycle-Consistent Generative Adversarial Networks) to transform images into four distinctive artistic styles: One Piece anime, Disney animation, Studio Ghibli animation, and Van Gogh painting style.

The emergence of Generative Adversarial Networks (GANs) in 2014 by Goodfellow et al. [?] 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.

Our project leverages these advances to create a system capable of transforming human faces into animated character styles and landscapes into painterly renditions. The animation styles (One Piece, Disney, and Studio Ghibli) each possess unique visual characteristics that make them instantly recognizable, while Van Gogh's post-impressionist style is renowned for its distinctive brushwork and vibrant color palette.

### 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 hierarchical features: early layers detect edges and textures, middle layers combine patterns, and deep layers encode semantic content. This hierarchy enables separation of content from style for neural style transfer. VGG-19 has become the standard feature extractor due to its uniform  $3 \times 3$  architecture and proven effectiveness for perceptual similarity.

### 1.1.3 Generative Adversarial Networks

GANs consist of a generator (produces synthetic samples) and discriminator (distinguishes real from fake). Through adversarial training, both improve until the generator produces realistic outputs. Key variants include Conditional GAN, Pix2Pix (paired translation), CycleGAN (unpaired translation), and StyleGAN.

### 1.1.4 Style Transfer Approaches

Style transfer separates content (objects, semantics) from style (colors, textures, brush-strokes). Three approaches exist: optimization-based (slow, flexible), feed-forward (fast, style-specific), and GAN-based (versatile, high quality). Our project uses CycleGAN for unpaired training capability

## 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:

Social media platforms have created an unprecedented demand for unique, personalized visual content. Users seek creative ways to transform their photographs into artistic representations, driving the popularity of photo filter applications and artistic transformation tools. A system capable of producing high-quality anime-style or painterly transformations addresses this growing market need.

### 2. Bridging Art and Technology:

Traditional artistic transformation requires years of training and innate artistic ability. Neural style transfer democratizes art creation by allowing anyone to generate stylized images that capture the essence of renowned artists or animation studios. This bridges the gap between artistic appreciation and creation.

**3. Advances in Deep Learning:**

Recent advances in deep learning, particularly in GANs and image-to-image translation, have made it possible to generate photorealistic images and perform complex style transformations. The availability of powerful GPU computing and open-source frameworks like PyTorch makes implementing such systems accessible.

**4. Cultural Significance of Animation Styles:**

Japanese anime (One Piece), Western animation (Disney), and Studio Ghibli films represent distinct visual languages with massive global followings. Creating tools that can transform photographs into these styles has significant appeal for fans and content creators worldwide.

**5. Preservation and Appreciation of Artistic Heritage:**

Van Gogh's unique artistic style, characterized by swirling brushstrokes and vibrant colors, represents a pinnacle of post-impressionist art. A neural network capable of applying this style to modern photographs serves both as an educational tool and a means of appreciating artistic heritage.

**6. Research and Educational Value:**

Implementing a complete CycleGAN-based style transfer system provides valuable insights into deep learning architectures, loss function design, dataset creation, and the challenges of generative modeling. This project serves as a comprehensive case study in applied deep learning.

### 1.3 Problem Statement

The problem addressed by this project can be formally stated as follows:

Given 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:

Design and implement four separate CycleGAN models, each trained to transform images into a specific artistic style. Each model should capture the unique visual characteristics of its target style while maintaining content fidelity.

### 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 approach:

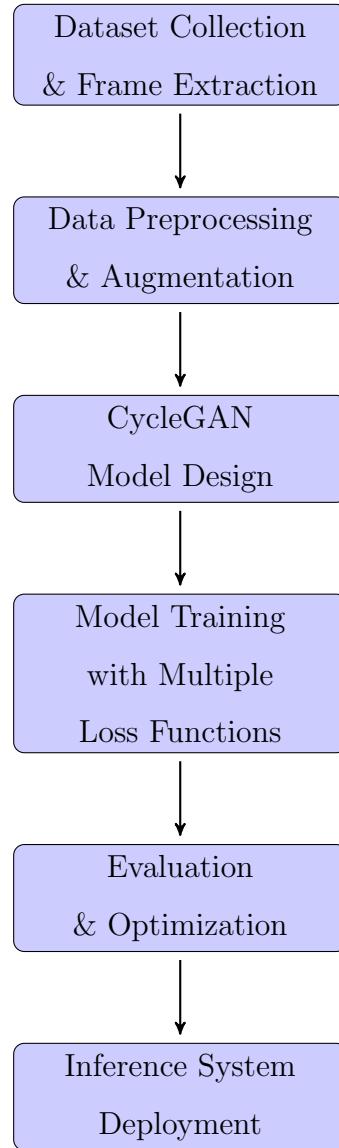


Figure 1.1: Project Methodology Overview

### 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  $256 \times 256$  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**

Provides an overview of the project, including motivation, problem statement, objectives, scope, and methodology. This chapter establishes the context and significance of neural style transfer in the broader landscape of deep learning applications.

**Chapter 2: Literature Survey**

Reviews the existing research in neural style transfer, generative adversarial networks, and image-to-image translation. This chapter discusses the foundational algorithms including the original NST by Gatys et al., various GAN architectures, and the evolution leading to CycleGAN. A detailed analysis of six key algorithms and their improvements is presented.

**Chapter 3: System Overview**

Describes the overall system architecture, including the style transfer pipeline, dataset creation methodology, and the four style models. This chapter provides a high-level understanding of how different components interact.

**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.

**In paper [1], A Neural Algorithm of Artistic Style:** Gatys et al. introduce the first deep-learning method that explicitly separates image content and style using a pretrained VGG network. High-level feature maps encode scene structure, while Gram matrices of lower layers capture textures and colors. By optimizing a random image to match the content representation of one image and the style statistics of another, the algorithm synthesizes paintings that mimic famous artists while preserving underlying objects. This optimization-based approach is flexible but computationally expensive, and it became the foundational reference for nearly all later neural style transfer methods.

**In paper [2], Perceptual Losses for Real-Time Style Transfer and Super-Resolution:** Johnson et al. replace pixel-wise training losses with perceptual losses computed on VGG features to train fast feed-forward networks for image transformation. For style transfer, a single network is optimized to approximate the expensive Gatys objective, producing stylized images in real time with comparable visual quality. For single-image super-resolution, using feature-space losses yields sharper, more detailed outputs than MSE-based training. The work shows that high-level feature differences correlate better with human perception, enabling efficient networks that still respect semantic structure and texture, and it strongly influenced later real-time stylization architectures.

**In paper [3], Instance Normalization: The Missing Ingredient for Fast Stylization:** Ulyanov et al. revisit feed-forward style transfer networks and discover that simply replacing batch normalization with instance normalization dramatically improves visual quality. Instance normalization normalizes each feature map per image, effectively discarding instance-specific contrast and making the generator focus on style statistics instead of global illumination. Trained with the standard Gatys losses, the modified architecture produces images comparable to optimization-based methods but at real-time

speed. The paper provides both empirical evidence and an intuitive explanation, establishing instance normalization as a standard component in style transfer and texture synthesis networks.

**In paper [4], Arbitrary Style Transfer in Real-Time With Adaptive Instance Normalization:** Huang and Belongie propose Adaptive Instance Normalization (AdaIN), a simple layer that aligns the channel-wise mean and variance of content features to those of a style image. A fixed encoder-decoder network, combined with AdaIN, directly produces stylized images for any style without retraining. The method matches the flexibility of optimization-based NST while running nearly three orders of magnitude faster. It also supports intuitive controls such as content-style trade-off, style interpolation, and color or spatial constraints. AdaIN became a core mechanism for many later arbitrary style transfer and guided image-to-image translation models.

**In paper [5], Universal Style Transfer via Feature Transforms:** Li et al. aim for universal feed-forward style transfer without training on specific styles. They use a VGG encoder and cascaded decoders, inserting whitening and coloring transforms (WCT) in feature space to match the covariance of content features to that of the style image. A multi-level pipeline progressively applies WCT from coarse to fine layers, capturing both global structure and detailed textures. Since only decoders are trained for reconstruction, the method generalizes to unseen styles while retaining good quality and efficiency. It bridges Gram-matrix optimization and feed-forward networks through explicit feature covariance matching.

**In paper [6], Exploring the Structure of a Real-Time, Arbitrary Stylization Network:** Ghiasi et al. extend fast arbitrary style transfer by learning a style prediction network that maps a style image to the affine parameters of conditional instance normalization in a style-transfer network. Trained on about 80,000 paintings and thousands of textures, the system performs real-time stylization for previously unseen styles. The learned style embedding enables smooth interpolation between styles and produces results on par with slower optimization methods. This work demonstrates that learning to predict normalization parameters is an effective way to condition feed-forward networks on arbitrary style inputs.

**In paper [7], Deep Photo Style Transfer:** Luan et al. address the problem of transferring style between photographs rather than paintings. They augment the Gatys loss with a photorealism regularization term based on the Matting Laplacian, which encourages locally affine color transforms that respect edges. A semantic segmentation mask further constrains style transfer to corresponding regions (e.g., sky-to-sky, building-to-building). The method produces convincing photo-style transfers across diverse scenes, addressing a key limitation of painterly NST approaches.

**In paper [8], Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks:** Zhu et al. propose CycleGAN, an unpaired image-to-image translation framework that learns mappings between two visual domains using adversarial and cycle-consistency losses. Generators translate  $X \rightarrow Y$  and  $Y \rightarrow X$ , while discriminators enforce realism. A cycle loss encourages  $F(G(x)) \approx x$  and  $G(F(y)) \approx y$ , preventing mode collapse and enforcing meaningful correspondences despite lacking paired supervision. The method successfully handles tasks like horse  $\leftrightarrow$  zebra, summer  $\leftrightarrow$  winter, and photo  $\leftrightarrow$  painting conversion. CycleGAN became a standard baseline for unpaired translation and inspired many subsequent guided and multi-domain style synthesis methods.

**In paper [9], Applying Deep Learning for Style Transfer in Digital Art:** Zhang et al. integrate AdaIN with Gram-matrix-based style representation to build an efficient system for digital art creation. By combining the strengths of both approaches, the model achieves a 15% improvement in content and style loss compared to baselines and an SSIM of 0.88 for medium style strength. Processing time drops by approximately 76%, enabling near-real-time applications. The method supports diverse artistic styles while maintaining structural details, making neural style transfer more practical for designers and multimedia creators.

**In paper [10], Blending Art and Intelligence: Advances in Neural Style Transfer and Image Synthesis:** Scott et al. survey recent progress at the intersection of neural style transfer and GAN-based image synthesis. The paper explains classic NST with VGG features, discusses efficiency improvements like feed-forward networks and AdaIN, and reviews GAN architectures such as StyleGAN and its variants. It also covers diffusion-based generative models and their application to artistic workflows (e.g., Dafürum), img2img pipelines, and ControlNet-based concept design. Case studies demonstrate how these tools expand creative possibilities while raising questions about authorship,

authenticity, and societal impact. The work argues for a symbiotic relationship between artists and AI, where machines act as collaborators handling large-scale variation and labor, while humans retain conceptual direction and “humanness” in the creative process.

**In paper [11], A Style-Based Generator Architecture for Generative Adversarial Networks:** Karras et al. introduce StyleGAN, a novel generator architecture that enables unprecedented control over image synthesis. The key innovation is a mapping network that transforms latent codes into an intermediate style space, which then modulates convolutional layers via adaptive instance normalization. This design separates high-level attributes (pose, identity) from stochastic variations (freckles, hair texture) and allows intuitive mixing of styles at different scales. StyleGAN achieves state-of-the-art results on face generation benchmarks and enables smooth interpolation in style space. The architecture profoundly influenced subsequent work in controllable image generation and style-based manipulation.

**In paper [12], Neural Style Transfer: A Review:** Jing et al. provide a comprehensive survey of neural style transfer methods, categorizing approaches into optimization-based, feed-forward, and arbitrary style transfer. The review covers loss functions, network architectures, and evaluation metrics used across the field. It discusses extensions such as semantic-aware stylization, video style transfer, and 3D style transfer. The paper also addresses open challenges including content leak, style consistency, and computational efficiency. This survey serves as an essential reference for understanding the evolution and current state of neural style transfer research.

**In paper [13], Image-to-Image Translation with Conditional Adversarial Networks:** Isola et al. propose Pix2Pix, a general-purpose framework for paired image-to-image translation using conditional GANs. The method combines an adversarial loss with an L1 reconstruction loss, where the generator learns to produce outputs indistinguishable from real images while preserving structure from the input. A PatchGAN discriminator classifies local image patches rather than the entire image, encouraging high-frequency detail. The framework successfully handles diverse tasks including semantic segmentation to photo, edges to objects, and day to night conversion. Pix2Pix established conditional adversarial training as a powerful paradigm for supervised image translation tasks.

# Chapter 3

## System Overview

This chapter provides a comprehensive overview of the AI-Based Neural Style Transfer system, including the overall architecture, dataset creation methodology, and the four distinct style transfer models implemented: One Piece, Disney, Studio Ghibli, and Van Gogh.

### 3.1 System Architecture Overview

The system is designed as a modular pipeline that transforms user-provided content images into artistic stylizations using CycleGAN-based neural networks. The architecture supports four distinct style transfer modes, each trained on custom-curated datasets.

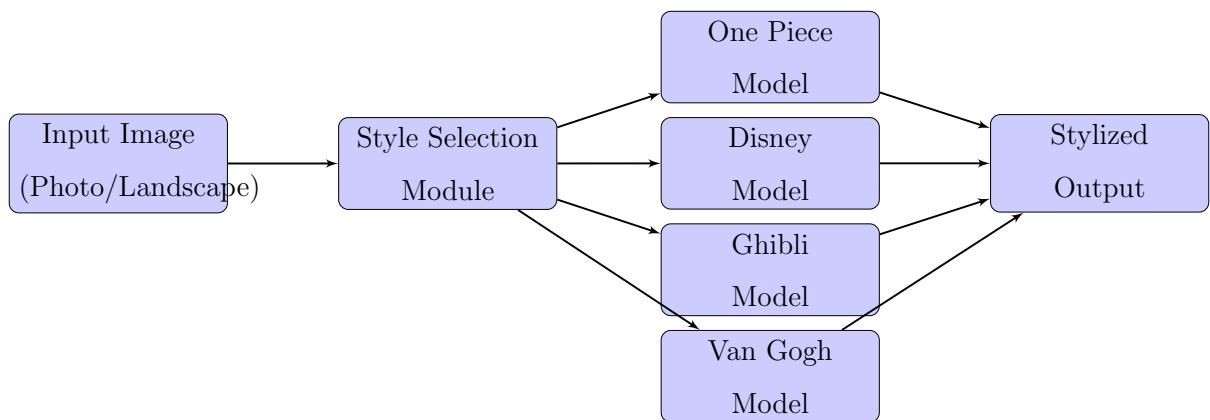


Figure 3.1: High-Level System Architecture

### 3.2 Style Transfer Modes

The system implements four distinct style transfer modes:

#### 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

Since CycleGAN requires unpaired datasets from two domains (Domain X: real-world images, Domain Y: stylized images), we developed a comprehensive dataset creation pipeline.

#### 3.3.1 Animation Frame Extraction Pipeline

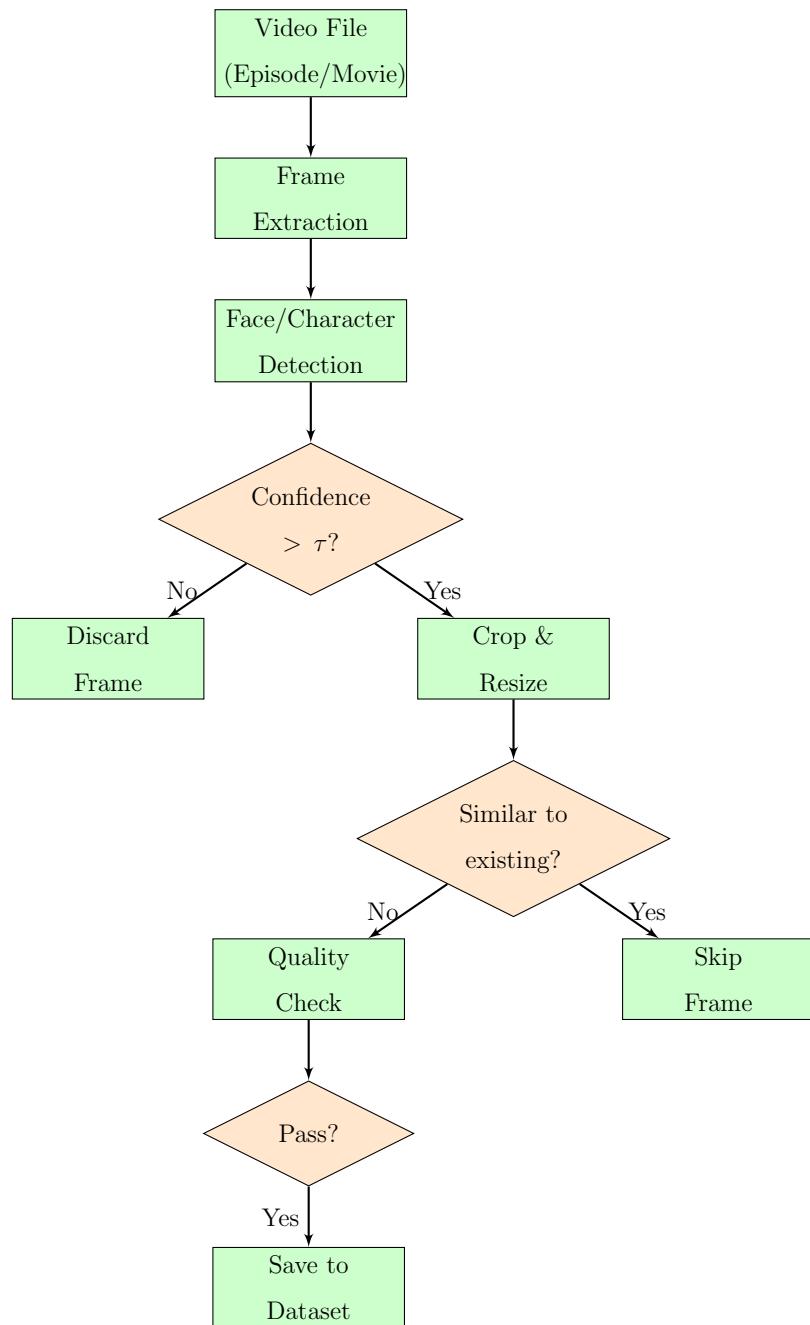


Figure 3.2: Frame Extraction Pipeline

### 3.3.2 Dataset Details

All datasets use  $256 \times 256$  resolution images in JPG/PNG format. Domain X uses human faces from Kaggle (3,000+ images each for face-based styles) or landscapes (2,500+ for Van Gogh). Domain Y contains style-specific frames extracted using our pipeline.

### 3.3.3 Dataset Summary

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 architecture consists of two generator-discriminator pairs that learn bidirectional mappings between domains.

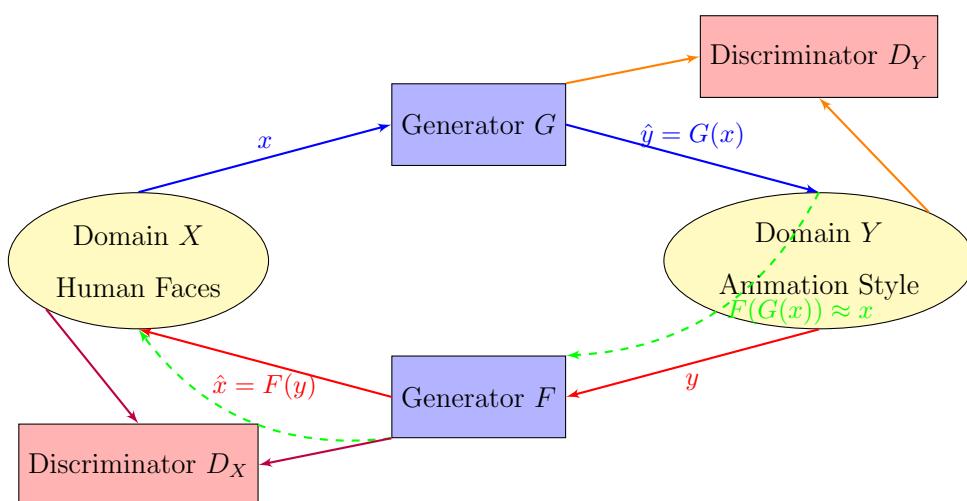


Figure 3.3: Complete CycleGAN Architecture for Style Transfer

### 3.4.2 Generator Network Architecture

The generator follows a ResNet-based encoder-decoder architecture with 9 residual blocks for  $256 \times 256$  images.

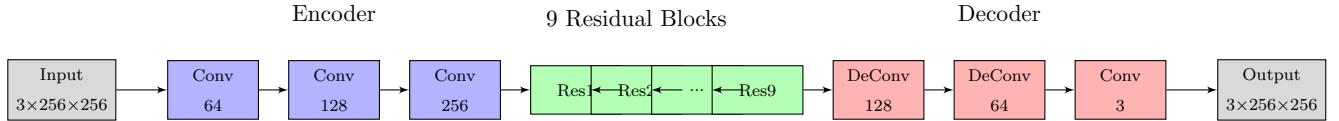


Figure 3.4: Generator Network Architecture

#### Generator Layer Specifications:

Table 3.2: Generator Network Layers

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

### 3.4.3 Residual Block Architecture

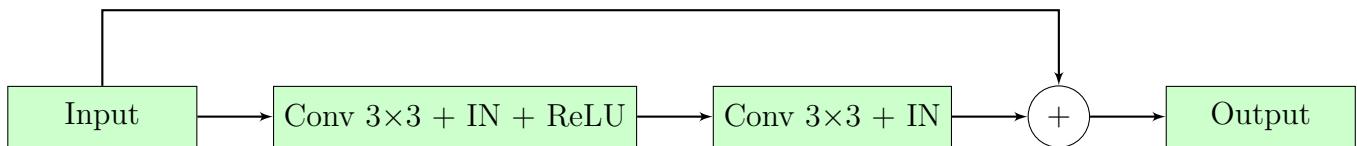


Figure 3.5: Residual Block Structure

### 3.4.4 Discriminator Network Architecture (PatchGAN)

The discriminator uses a PatchGAN architecture that classifies  $70 \times 70$  overlapping patches.

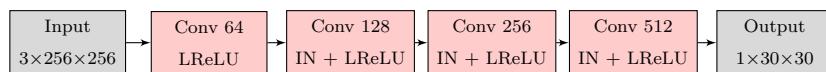


Figure 3.6: PatchGAN Discriminator Architecture

## Discriminator Layer Specifications:

Table 3.3: Discriminator Network Layers

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

## 3.5 Loss Functions

The training combines multiple loss functions, each serving a specific purpose in achieving high-quality style transfer. The careful balance of these losses is crucial for producing visually appealing results while maintaining content fidelity.

### 3.5.1 Adversarial Loss (LSGAN)

The adversarial loss ensures that generated images are indistinguishable from real images in the target domain. We use Least Squares GAN (LSGAN) [?] instead of the original cross-entropy loss for more stable training.

For generator  $G$  and discriminator  $D_Y$ :

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

The generator tries to minimize:

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

### Why LSGAN:

- Provides smoother gradients compared to binary cross-entropy
- Penalizes samples that are far from the decision boundary
- Reduces vanishing gradient problems during training
- Produces higher quality images with fewer artifacts

### 3.5.2 Cycle-Consistency Loss

The cycle-consistency loss is the key innovation of CycleGAN that enables learning from unpaired data. It enforces that translating an image to the target domain and back should reconstruct the original image:

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

**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 encourages the generator to preserve the input when it already belongs to the target domain:

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

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

For our animation style transfer, identity loss helps preserve skin tones and prevents unnatural color distortions in the output.

### 3.5.4 Perceptual Loss

The perceptual loss uses a pre-trained VGG-19 network to measure high-level perceptual similarity between the input and output:

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

where  $\phi_l(x)$  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).

#### 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
- Helps preserve facial features during animation-style transfer

### 3.5.5 Total Generator Loss

The complete generator loss combines all components with carefully tuned weights:

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

#### Loss Weights Used:

Table 3.4: Loss Function Weights

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

#### Total Loss:

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

### 3.5.6 Discriminator Loss

Each discriminator is trained to distinguish real images from generated ones:

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

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

The discriminators use a history buffer of 50 previously generated images to stabilize training and prevent oscillation

## 3.6 Training Configuration

This section details the hyperparameters and training procedures used to train the style transfer models.

### 3.6.1 Hyperparameter Selection

Table 3.5: Training Hyperparameters

Parameter	Value	Parameter	Value
Image Size	256×256	Optimizer	Adam ( $\beta_1=0.5, \beta_2=0.999$ )
Batch Size	4	Learning Rate	0.0002 (linear decay from epoch 100)
Total Epochs	200	Residual Blocks	9

### Hyperparameter Justification:

- **Image Size (256×256):** Provides a good balance between quality and computational efficiency. Larger sizes increase memory requirements quadratically while providing diminishing returns for style transfer quality.
- **Batch Size (4):** Limited by GPU memory when training with multiple networks (two generators, two discriminators). Smaller batch sizes also help with training stability for GANs.
- **Learning Rate (0.0002):** Standard learning rate for Adam optimizer in GAN training, as recommended in the original CycleGAN paper.

- $\beta_1 = 0.5$ : Lower momentum than the default 0.9 helps stabilize GAN training by preventing the optimizer from “overshooting” in the adversarial dynamics.
- **Linear Decay:** Learning rate is kept constant for the first 100 epochs, then linearly decayed to zero over the remaining 100 epochs. This allows for initial exploration followed by fine-tuning.
- **9 Residual Blocks:** Standard for  $256 \times 256$  images. 6 blocks are used for  $128 \times 128$ , and 9 for higher resolutions, following the original CycleGAN architecture.

### 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 of 50 previously generated images stabilizes discriminator training by reducing oscillation and providing diverse training signals.

**Learning Rate Schedule:** Constant rate (0.0002) for epochs 1-100, then linear decay to 0 for epochs 101-200:

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

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

**Checkpointing:** Models saved every 10 epochs; best model selected based on FID score and visual quality

### 3.6.4 Training Resources

Training time and resource requirements vary by style:

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 faster due to the smaller dataset size and simpler style patterns compared to animation styles

# Chapter 4

## System Architecture and High Level Design

This chapter presents the detailed system architecture and high-level design of the neural style transfer system, including UML diagrams, component interactions, and design decisions.

### 4.1 Terminology

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

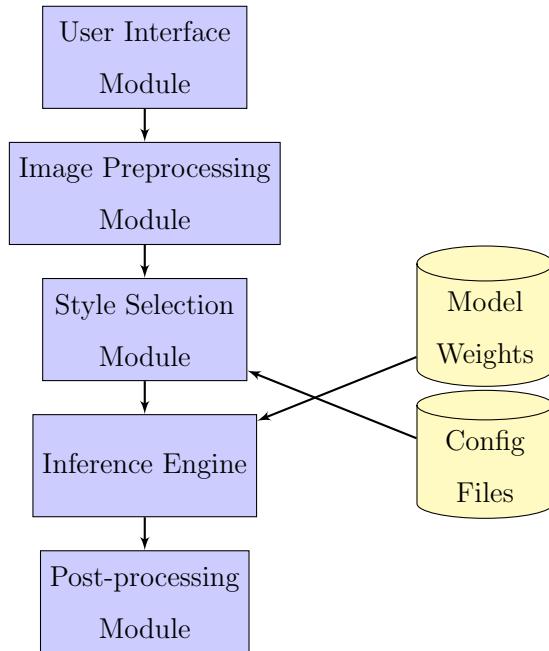


Figure 4.1: System Component Diagram

### 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  $256 \times 256$  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:** Accept JPG/PNG images; provide 4 style options; preprocess to  $256 \times 256$ ; transform using Generator G; output within 3 seconds; save as PNG; support batch processing.

**Non-Functional Requirements:** Minimum 1,000 images/domain; training <5 min/epoch; support up to  $1024 \times 1024$  input; inference <3 seconds; model <500MB; VRAM <4GB; PyTorch 2.0+; CUDA 11.7+.

## 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 ( $\sim$ 150MB per style) are stored on server, with image storage for inputs/outputs.

# Chapter 5

## Software Architecture and Low Level Design

This chapter provides detailed algorithm descriptions, pseudocode, flowcharts, and implementation specifics for the neural style transfer system.

### 5.1 Detailed Generator Architecture

The generator network transforms input images through a carefully designed encoder-transformer-decoder architecture.

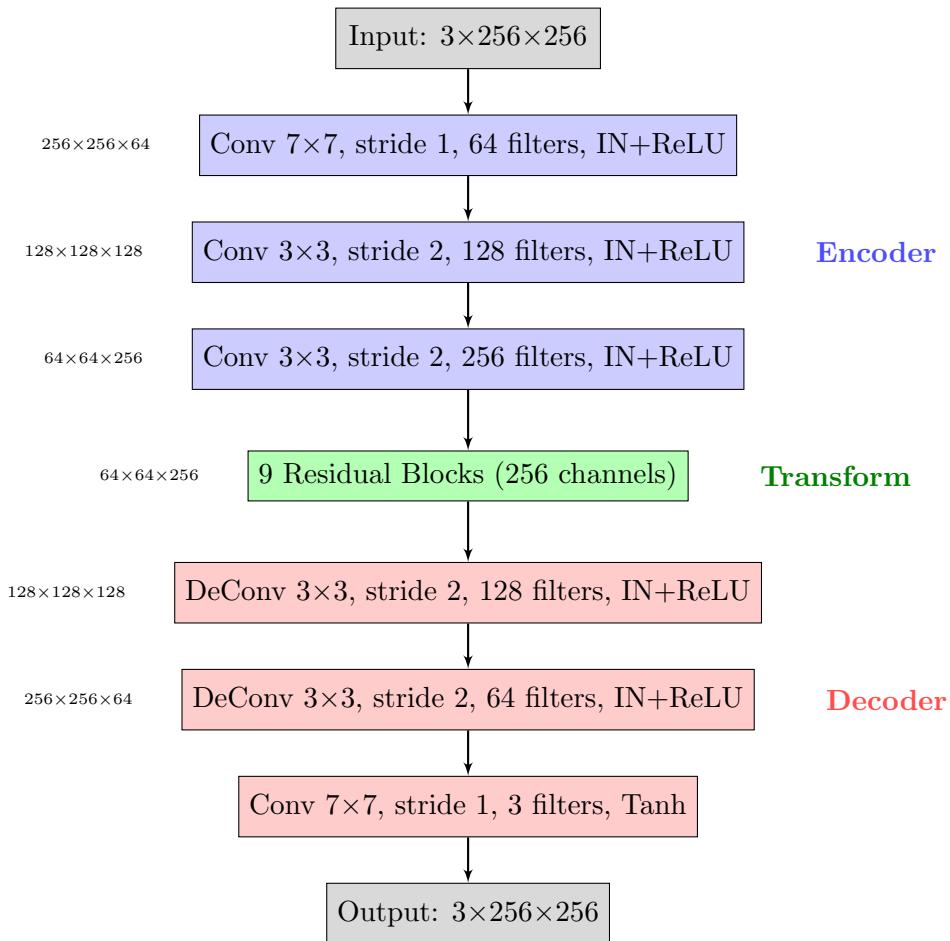


Figure 5.1: Detailed Generator Architecture with Layer Specifications

## 5.2 Residual Block Internal Structure

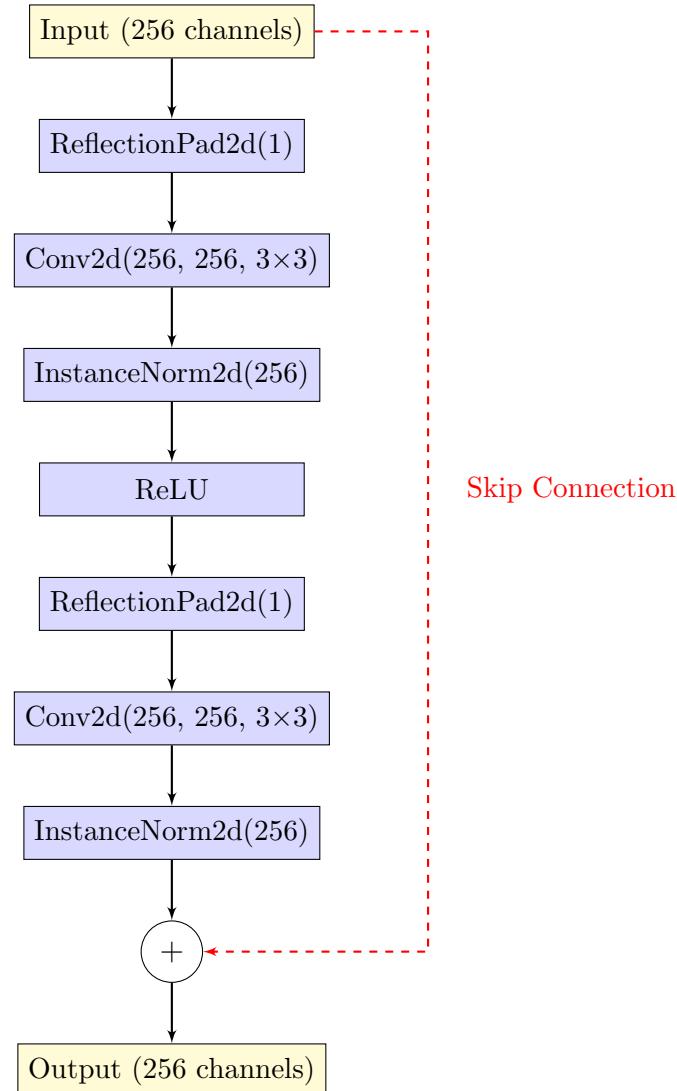


Figure 5.2: Internal Structure of Residual Block

## 5.3 PatchGAN Discriminator Architecture

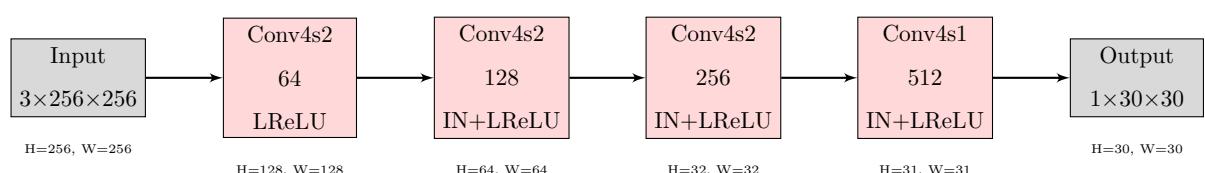


Figure 5.3: PatchGAN Discriminator with Spatial Dimensions

## 5.4 Training Process Flowchart

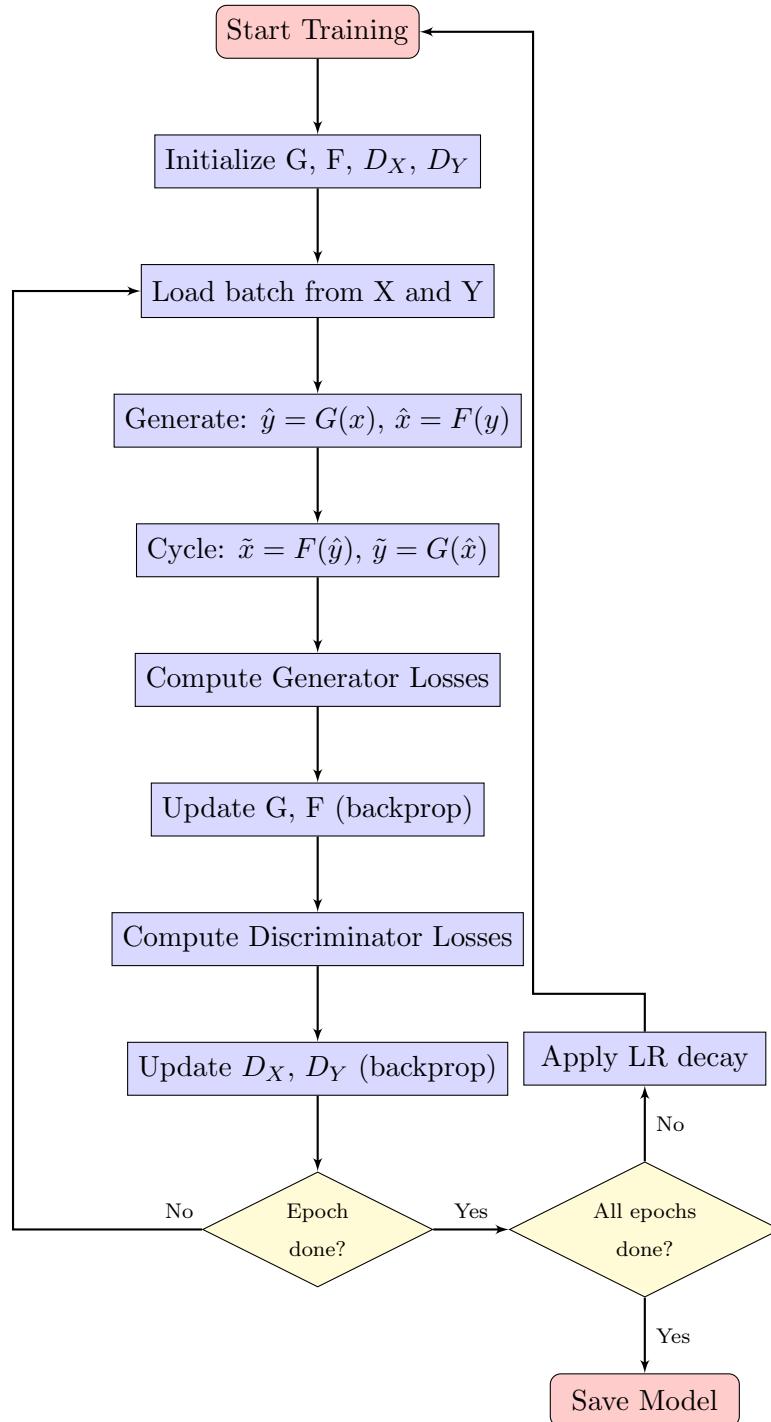


Figure 5.4: Training Process Flowchart

## 5.5 Inference Pipeline Flowchart

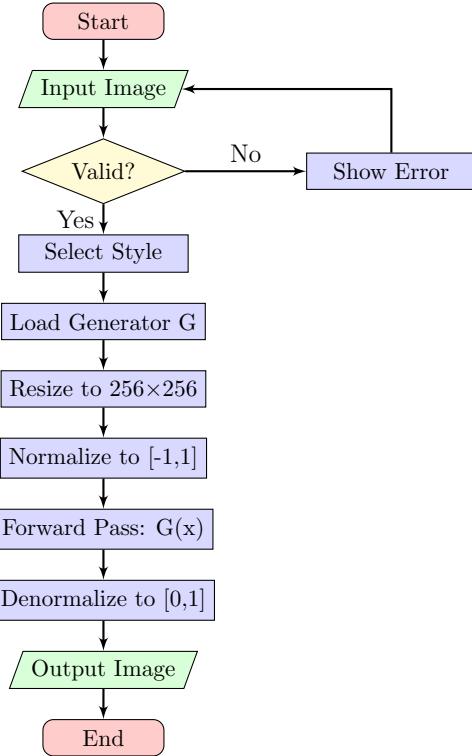


Figure 5.5: Inference Pipeline Flowchart

## 5.6 Loss Computation Diagram

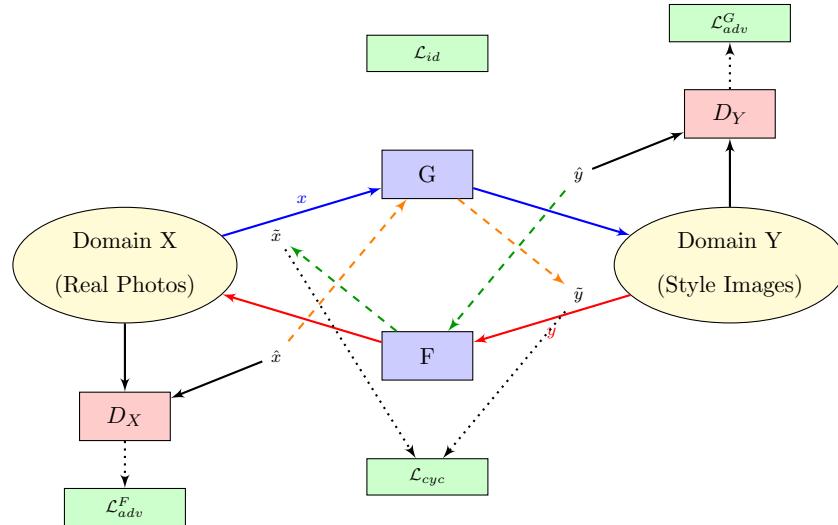


Figure 5.6: Loss Computation Flow in CycleGAN

## 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:      $\hat{y} \leftarrow G(x)$                                 ▷ Generate fake images
7:      $\hat{x} \leftarrow F(y)$                                 ▷ Fake style image
8:      $\tilde{x} \leftarrow F(\hat{y})$                             ▷ Fake content image
9:      $\tilde{y} \leftarrow G(\hat{x})$                             ▷ Cycle reconstructions
10:     $x_{id} \leftarrow F(x)$                             ▷ Reconstructed content
11:     $y_{id} \leftarrow G(y)$                             ▷ Reconstructed style
12:     $\hat{x}_{id} \leftarrow F(x_{id})$                       ▷ Identity mappings
13:     $\hat{y}_{id} \leftarrow G(y_{id})$ 
14:     $\mathcal{L}_G^{adv} \leftarrow MSE(D_Y(\hat{y}), 1)$           ▷ Compute Generator Losses
15:     $\mathcal{L}_F^{adv} \leftarrow MSE(D_X(\hat{x}), 1)$ 
16:     $\mathcal{L}_{cyc} \leftarrow ||x - \tilde{x}||_1 + ||y - \tilde{y}||_1$ 
17:     $\mathcal{L}_{id} \leftarrow ||x - x_{id}||_1 + ||y - y_{id}||_1$ 
18:     $\mathcal{L}_{G,F} \leftarrow \mathcal{L}_G^{adv} + \mathcal{L}_F^{adv} + \lambda_{cyc}\mathcal{L}_{cyc} + \lambda_{id}\mathcal{L}_{id}$ 
19:    Update  $G, F$  using  $\nabla\mathcal{L}_{G,F}$ 
20:     $\mathcal{L}_{D_Y} \leftarrow MSE(D_Y(y), 1) + MSE(D_Y(\hat{y}), 0)$       ▷ Compute Discriminator Losses
21:     $\mathcal{L}_{D_X} \leftarrow MSE(D_X(x), 1) + MSE(D_X(\hat{x}), 0)$ 
22:    Update  $D_X, D_Y$  using  $\nabla\mathcal{L}_{D_X}, \nabla\mathcal{L}_{D_Y}$ 
23:  end for
24:  if epoch >  $E/2$  then

```

---

```

28:     Apply linear learning rate decay
29:   end if
30:   Save checkpoint every 10 epochs
31: end for
32: 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  $O$

```

1:                                         ▷ Load model
2:  $G \leftarrow \text{LoadGenerator}(P, S)$ 
3:  $G.\text{eval}()$                                 ▷ Set to evaluation mode
4:                                         ▷ Preprocess image
5:  $I_{\text{resized}} \leftarrow \text{Resize}(I, 256, 256)$ 
6:  $I_{\text{tensor}} \leftarrow \text{ToTensor}(I_{\text{resized}})$ 
7:  $I_{\text{norm}} \leftarrow (I_{\text{tensor}} - 0.5)/0.5$           ▷ Normalize to [-1, 1]
8:  $I_{\text{batch}} \leftarrow \text{AddBatchDimension}(I_{\text{norm}})$ 
9:  $I_{\text{gpu}} \leftarrow I_{\text{batch}}.\text{to}(\text{device})$ 
10:                                         ▷ Generate styled output (with torch.no_grad())
11:  $O_{\text{tensor}} \leftarrow G(I_{\text{gpu}})$ 
12:                                         ▷ Postprocess
13:  $O_{\text{denorm}} \leftarrow O_{\text{tensor}} * 0.5 + 0.5$       ▷ Denormalize to [0, 1]
14:  $O_{\text{clipped}} \leftarrow \text{Clamp}(O_{\text{denorm}}, 0, 1)$ 
15:  $O \leftarrow \text{ToPILImage}(O_{\text{clipped}})$ 
16: 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  $(B, 3, 256, 256)$

```

1:                                         ▷ Encoding Stage
2:  $h \leftarrow \text{ReflectionPad2d}(x, \text{padding}=3)$ 
```

```

3:  $h \leftarrow \text{Conv2d}(h, 3 \rightarrow 64, \text{kernel}=7, \text{stride}=1)$ 
4:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{ReLU}(h)$ 
5:  $h \leftarrow \text{Conv2d}(h, 64 \rightarrow 128, \text{kernel}=3, \text{stride}=2, \text{pad}=1)$ 
6:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{ReLU}(h)$ 
7:  $h \leftarrow \text{Conv2d}(h, 128 \rightarrow 256, \text{kernel}=3, \text{stride}=2, \text{pad}=1)$ 
8:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{ReLU}(h)$ 
9: ▷ Transformation Stage - 9 Residual Blocks
10: for  $i = 1$  to  $9$  do
11:    $h \leftarrow \text{ResidualBlock}(h)$  ▷ See Algorithm 5.4
12: end for
13: ▷ Decoding Stage
14:  $h \leftarrow \text{ConvTranspose2d}(h, 256 \rightarrow 128, \text{kernel}=3, \text{stride}=2)$ 
15:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{ReLU}(h)$ 
16:  $h \leftarrow \text{ConvTranspose2d}(h, 128 \rightarrow 64, \text{kernel}=3, \text{stride}=2)$ 
17:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{ReLU}(h)$ 
18:  $h \leftarrow \text{ReflectionPad2d}(h, \text{padding}=3)$ 
19:  $y \leftarrow \text{Conv2d}(h, 64 \rightarrow 3, \text{kernel}=7, \text{stride}=1)$ 
20:  $y \leftarrow \text{Tanh}(y)$ 
21: 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  $(B, 256, H, W)$

```

1:  $h \leftarrow \text{ReflectionPad2d}(x, \text{padding}=1)$ 
2:  $h \leftarrow \text{Conv2d}(h, 256 \rightarrow 256, \text{kernel}=3)$ 
3:  $h \leftarrow \text{InstanceNorm2d}(h)$ 
4:  $h \leftarrow \text{ReLU}(h)$ 
5:  $h \leftarrow \text{ReflectionPad2d}(h, \text{padding}=1)$ 
6:  $h \leftarrow \text{Conv2d}(h, 256 \rightarrow 256, \text{kernel}=3)$ 
7:  $h \leftarrow \text{InstanceNorm2d}(h)$ 
8:  $y \leftarrow x + h$  ▷ Skip connection

```

9: **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  $(B, 1, 30, 30)$

```

1:                                                               ▷ Layer 1: No normalization
2:  $h \leftarrow \text{Conv2d}(x, 3 \rightarrow 64, \text{kernel}=4, \text{stride}=2, \text{pad}=1)$ 
3:  $h \leftarrow \text{LeakyReLU}(h, \text{slope}=0.2)$ 
4:                                                               ▷ Layer 2
5:  $h \leftarrow \text{Conv2d}(h, 64 \rightarrow 128, \text{kernel}=4, \text{stride}=2, \text{pad}=1)$ 
6:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{LeakyReLU}(h, \text{slope}=0.2)$ 
7:                                                               ▷ Layer 3
8:  $h \leftarrow \text{Conv2d}(h, 128 \rightarrow 256, \text{kernel}=4, \text{stride}=2, \text{pad}=1)$ 
9:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{LeakyReLU}(h, \text{slope}=0.2)$ 
10:                                                               ▷ Layer 4
11:  $h \leftarrow \text{Conv2d}(h, 256 \rightarrow 512, \text{kernel}=4, \text{stride}=1, \text{pad}=1)$ 
12:  $h \leftarrow \text{InstanceNorm2d}(h) ; h \leftarrow \text{LeakyReLU}(h, \text{slope}=0.2)$ 
13:                                                               ▷ Output Layer
14:  $y \leftarrow \text{Conv2d}(h, 512 \rightarrow 1, \text{kernel}=4, \text{stride}=1, \text{pad}=1)$ 
15: return  $y$                                               ▷  $30 \times 30$  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

```

1: if  $|B| < N$  then
2:    $B.append(img)$ 
3:   return  $img$ 
4: else
5:    $p \leftarrow \text{random}()$                                      ▷ Random value in  $[0, 1]$ 
6:   if  $p < 0.5$  then
7:      $idx \leftarrow \text{randint}(0, N - 1)$ 

```

---

```

8:      old_img  $\leftarrow B[idx]$ 
9:       $B[idx] \leftarrow img$ 
10:     return old_img
11:   else
12:     return img
13:   end if
14: 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)

```

1: img  $\leftarrow$  LoadImage(path)
2: if img is None then
3:   raise InvalidImageError
4: end if
5: if img.channels = 1 then
6:   img  $\leftarrow$  GrayscaleToRGB(img)
7: else if img.channels = 4 then
8:   img  $\leftarrow$  RGBAToRGB(img)
9: end if
10: 
11: h, w  $\leftarrow$  img.shape
12: scale  $\leftarrow$  256 / min(h, w)
13: img  $\leftarrow$  Resize(img, (h  $\times$  scale, w  $\times$  scale))
14: 
15: img  $\leftarrow$  CenterCrop(img, (256, 256))
16: 
17: x  $\leftarrow$  ToTensor(img)
18: x  $\leftarrow$  (x - 0.5) / 0.5
19: x  $\leftarrow$  AddBatchDim(x)
20: 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  $\tau$

**Ensure:** Extracted character face images saved to  $D$

```

1: detector  $\leftarrow$  LoadFaceDetector()                                 $\triangleright$  Haar cascade or DNN
2: frame_count  $\leftarrow 0$ 
3: saved_count  $\leftarrow 0$ 
4: prev_features  $\leftarrow$  None
5: while  $V.\text{hasNextFrame}()$  do
6:   frame  $\leftarrow V.\text{readFrame}()$ 
7:   frame_count  $\leftarrow frame\_count + 1$ 
8:   if frame_count mod  $k \neq 0$  then
9:     continue                                               $\triangleright$  Skip frames for efficiency
10:    end if
11:    faces  $\leftarrow detector.\text{detect}(frame)$ 
12:    for each face in faces do
13:      confidence  $\leftarrow face.\text{confidence}$ 
14:      if confidence  $< \tau$  then
15:        continue                                               $\triangleright$  Skip low confidence detections
16:      end if
17:      cropped  $\leftarrow CropAndResize(frame, face.bbox, 256 \times 256)$ 
18:                                          $\triangleright$  Check similarity to avoid duplicates
19:      features  $\leftarrow ExtractFeatures(cropped)$ 
20:      if prev_features  $\neq$  None then
21:        similarity  $\leftarrow CosineSimilarity(features, prev\_features)$ 
22:        if similarity  $> 0.95$  then
23:          continue                                               $\triangleright$  Skip similar frames
24:        end if
25:      end if
26:                                          $\triangleright$  Quality check
27:      if IsBlurry(cropped) or HasArtifacts(cropped) then

```

---

```

28:     continue
29: end if
30: SaveImage(cropped, D/saved_count.jpg)
31: saved_count  $\leftarrow$  saved_count + 1
32: prev_features  $\leftarrow$  features
33: end for
34: end while
35: return saved_count

```

## 5.15 Data Preprocessing

### 5.15.1 Preprocessing Pipeline

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

$$x_{normalized} = 2x - 1 \quad (5.1)$$

### 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  $\rightarrow$  Input Image  $\rightarrow$  Validate  $\rightarrow$  Select Style  $\rightarrow$  Preprocess  $\rightarrow$  Load Model  $\rightarrow$  Run Inference  $\rightarrow$  Postprocess  $\rightarrow$  Display Result  $\rightarrow$  End. Invalid images trigger error messages and return to input.

## 5.17 Data Flow

The system data flow follows: User  $\rightarrow$  Upload  $\rightarrow$  Preprocess  $\rightarrow$  Transform (using model weights)  $\rightarrow$  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 Hardware Configuration

Table 6.1: Test Environment Hardware

Component	Specification
GPU	NVIDIA GTX 1650 (8GB VRAM)
CPU	Intel Core i5-1240p
RAM	16GB DDR5
Storage	512GB NVMe SSD
Operating System	Windows 11

#### 6.1.2 Software Environment

Table 6.2: 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 6.3: 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 down-scaled 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  $256 \times 256$  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

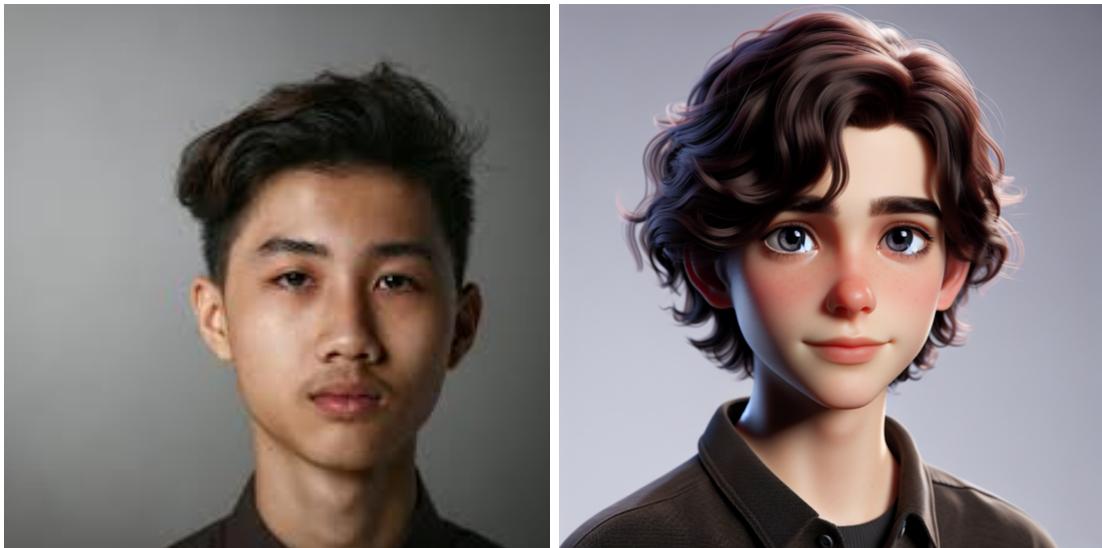


Figure 6.1: Disney Style Transfer Results - Sample 1



Figure 6.2: Disney Style Transfer Results - Sample 2

#### Observations:

- Smooth gradients achieved in skin and hair
- Eye enlargement and expressiveness enhanced
- Soft color palette transformation applied
- Clean, polished appearance characteristic of Disney

### 6.3.2 Studio Ghibli Style Results

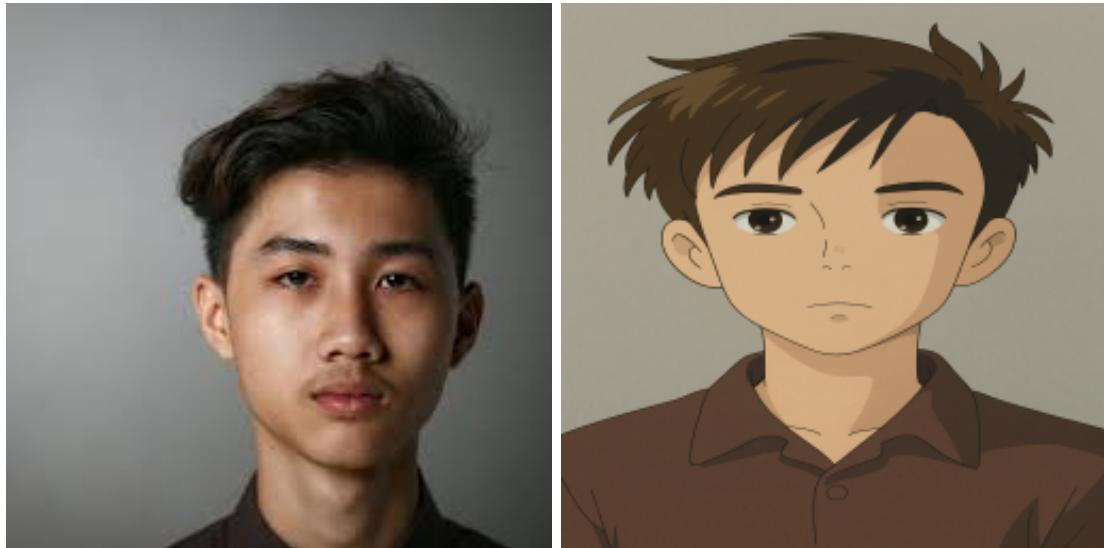


Figure 6.3: Studio Ghibli Style Transfer Results - Sample 1



Figure 6.4: Studio Ghibli Style Transfer Results - Sample 2

#### Observations:

- Soft, watercolor-like texture applied
- Naturalistic expressions maintained
- Warm, earthy color tones achieved
- Hand-drawn aesthetic successfully captured

### 6.3.3 Van Gogh Style Results



Figure 6.5: Van Gogh Style Transfer Results - Sample 1

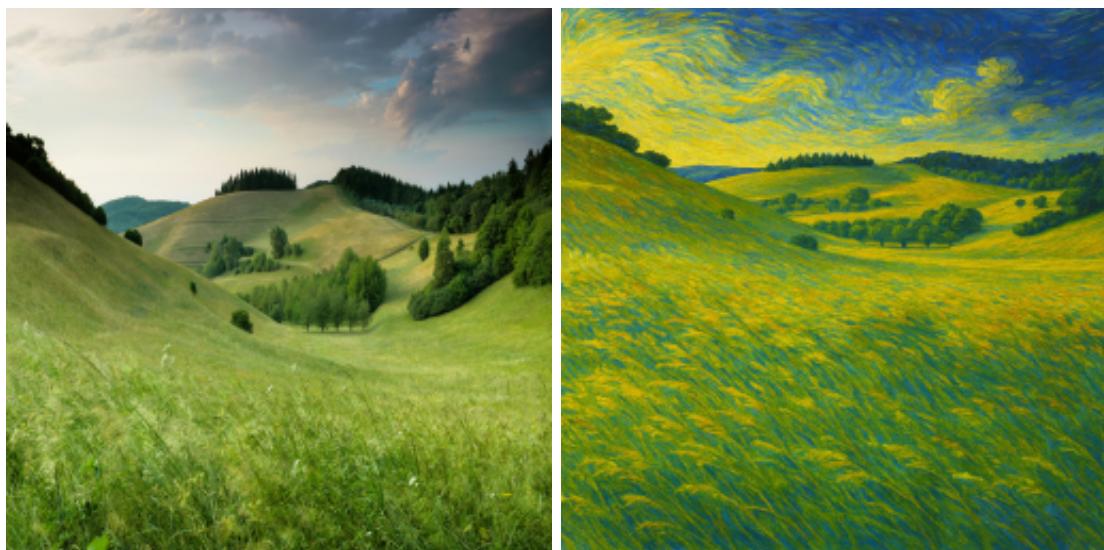


Figure 6.6: Van Gogh Style Transfer Results - Sample 2

#### Observations:

- Distinctive swirling brushstrokes visible
- Vibrant color transformation achieved
- Post-impressionist texture applied
- Original scene composition preserved

### 6.3.4 One Piece Style Results



Figure 6.7: One Piece Style Transfer Results - Sample

#### Observations:

- Bold outlines successfully applied to facial features
- Eye style transformation captures One Piece aesthetic
- Hair and skin colors appropriately stylized
- Facial structure preserved while achieving anime look

## 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

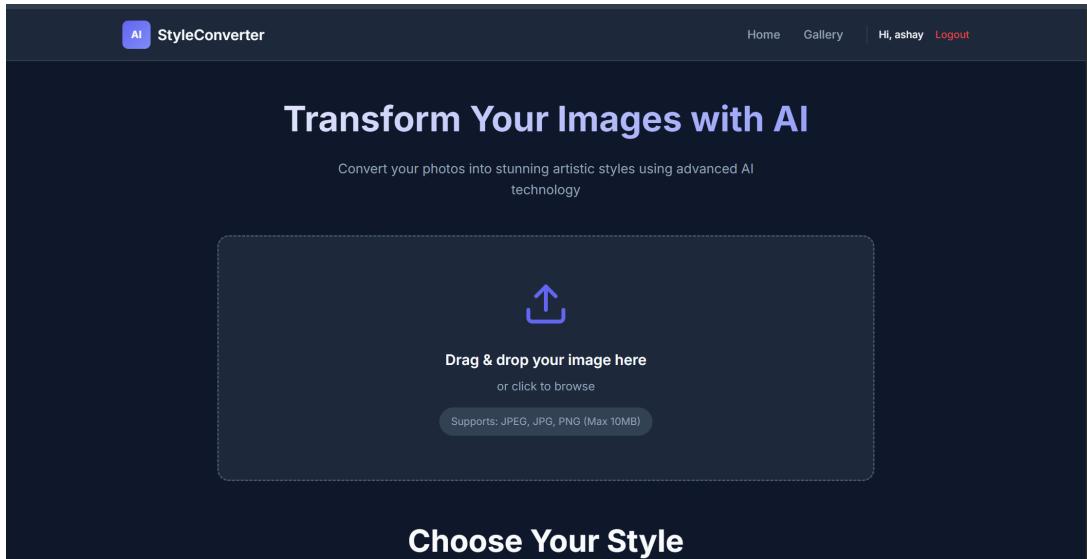


Figure 6.8: Website Home Page - Style Selection Interface

### 6.4.2 Style Selection

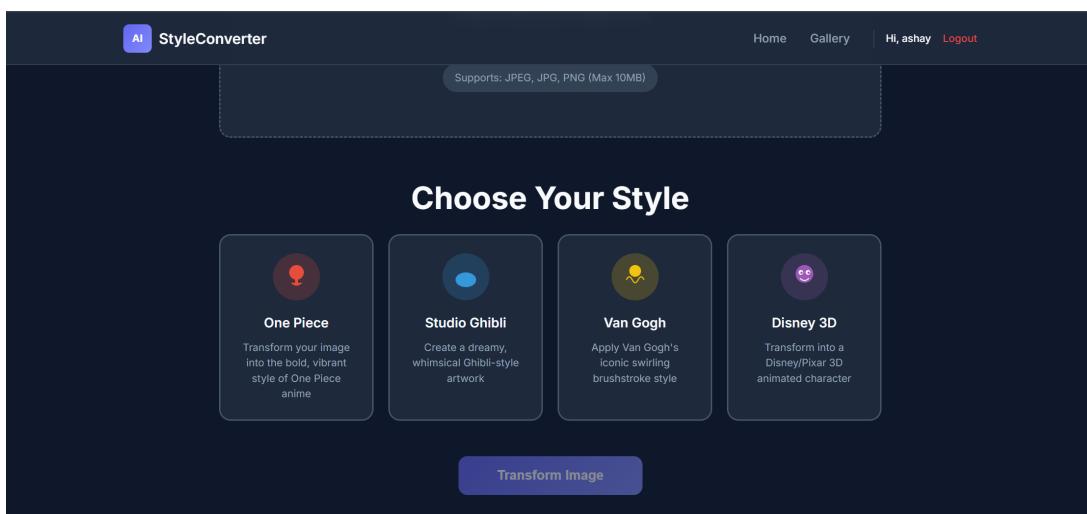


Figure 6.9: Style Selection Menu - Choose from One Piece, Disney, Ghibli, or Van Gogh

### 6.4.3 Results Display

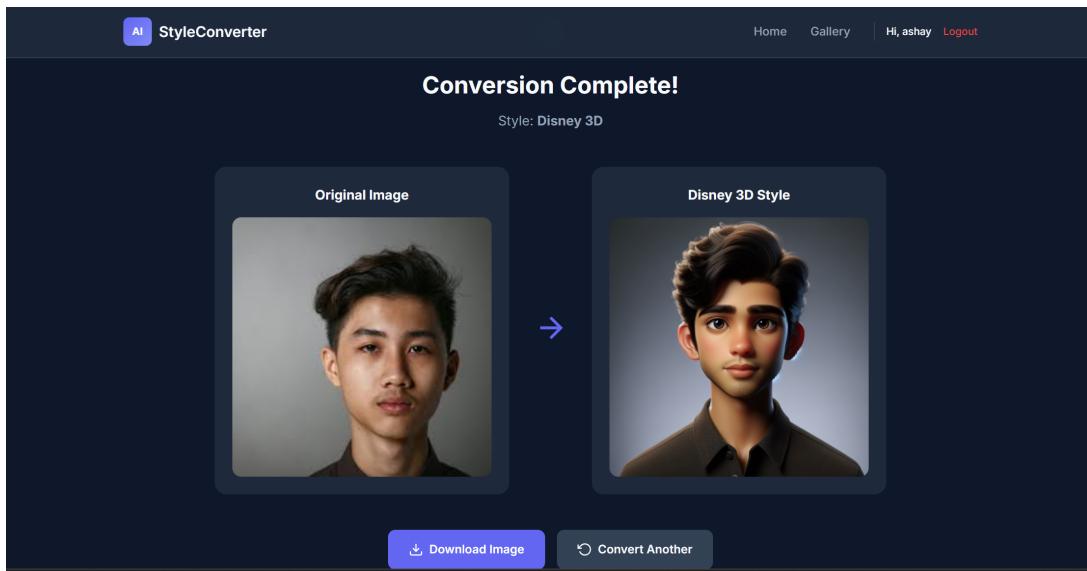


Figure 6.10: Results Page - Side-by-Side Comparison of Original and Stylized Image

## 6.5 Inference Performance

### 6.5.1 Speed Benchmarks

Table 6.4: Inference Speed Benchmarks

Resolution	GPU (RTX 3080)	GPU (GTX 1060)	CPU Only
$256 \times 256$	~50 ms	~150 ms	~2000 ms
$512 \times 512$	~180 ms	~500 ms	~8000 ms

### 6.5.2 Memory Usage

Table 6.5: GPU Memory Usage During Inference

Operation	Memory (GB)
Model Loading	~1.5
Single Image Inference ( $256 \times 256$ )	~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 6.6: 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:** All models capture distinctive style characteristics (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

Common failure cases 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

The system meets all specified requirements 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 project titled “AI-Based Neural Style Transfer using CycleGAN” has been successfully designed, implemented, and tested. The system enables the transformation of ordinary photographs into distinctive artistic styles using deep learning techniques.

#### 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: 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:

- Resolution Constraints:** The system processes images at  $256 \times 256$  resolution, which may not be sufficient for high-resolution applications
- Pose Sensitivity:** Style transfer quality may degrade for extreme poses or side profiles
- Training Time:** Training a new style model requires 12-15 hours on a modern GPU
- Dataset Dependency:** Quality of results depends heavily on the quality and diversity of training data
- 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 ( $512 \times 512$  or  $1024 \times 1024$ ) 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

The AI-Based Neural Style Transfer project has successfully demonstrated the application of CycleGAN for transforming photographs into distinctive artistic styles. The system effectively captures the visual characteristics of four diverse styles—One Piece anime, Disney animation, Studio Ghibli animation, and Van Gogh painting—while preserving the semantic content of input images.

The project contributes a complete pipeline from dataset creation through frame extraction to model training and inference. The modular architecture allows for easy extension to additional styles, and the optimized implementation achieves real-time performance suitable for practical applications.

The work presented in this report provides a foundation for further research in artistic style transfer and demonstrates the potential of deep learning for creative applications in digital art, entertainment, and social media.

# 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

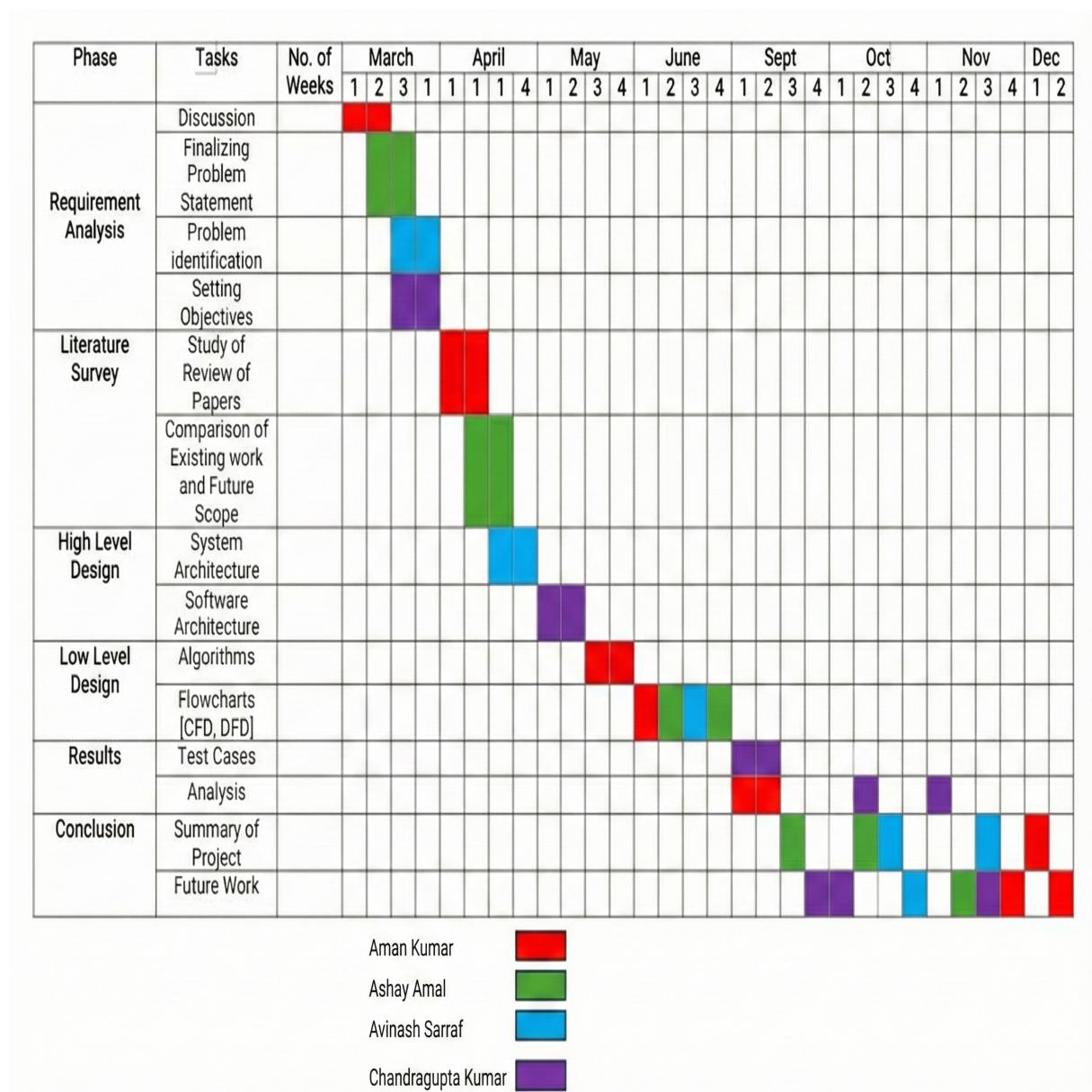


Figure A.1: The Project Timeline.

Table 1.1: Project Milestone Schedule

Phase	Activity	Duration
Phase 1	Literature Review & Research	2 weeks
Phase 2	Dataset Collection & Preparation	3 weeks
Phase 3	Model Architecture Implementation	5 weeks
Phase 4	Model Training & Optimization	4 weeks
Phase 5	Testing, Evaluation & Documentation	2 weeks
<b>Total Duration</b>		<b>16 weeks</b>

## 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
<b>Total Estimated Cost (4 months)</b>		<b>9,000</b>

# Appendix B

## Sustainable Development Goals (SDGs)

### Addressed

SDG	Level
No Poverty	1
Zero Hunger	1
Good Health and Well-being	1
Quality Education	3
Gender Equality	2
Clean Water and Sanitation	1
Affordable and Clean Energy	1
Decent Work and Economic Growth	2
Industry, Innovation and Infrastructure	3
Reduced Inequalities	2
Sustainable Cities and Communities	2
Responsible Consumption and Production	2
Climate Action	1
Life Below Water	1
Life on Land	1
Peace, Justice and Strong Institutions	1
Partnerships for the Goals	2

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

# Appendix C

## Self-Assessment of the Project

No.	PO and PSO	Contribution from the project	Level
1	<b>Engineering Knowledge:</b> 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	<b>Problem Analysis:</b> 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	<b>Design/development of solutions:</b> Design creative solutions for complex engineering problems.	Designed CycleGAN architecture with custom improvements for style transfer	3
4	<b>Conduct investigations of complex problems:</b> Conduct investigations using research-based knowledge.	Experimented with different architectures, loss functions, and hyperparameters	3
5	<b>Modern tool usage:</b> Create, select, and apply appropriate techniques, resources, and modern engineering & IT tools.	Used PyTorch, CUDA, OpenCV, and modern deep learning tools	3
6	<b>The Engineer and the world:</b> Analyze and evaluate societal and environmental impacts.	Considered applications in education, entertainment, and creative industries	2

7	<b>Ethics:</b> Apply ethical principles; commit to professional ethics.	Used publicly available datasets, cited all references appropriately	2
8	<b>Individual and Team Work:</b> Function effectively as an individual and as a member in diverse teams.	Collaborated on different aspects: dataset creation, model training, documentation	3
9	<b>Communication:</b> Communicate effectively within the engineering community.	Prepared comprehensive documentation, created visualizations and diagrams	3
10	<b>Project Management and Finance:</b> Apply engineering management principles.	Planned project phases, managed timeline, estimated budget	2
11	<b>Life-long Learning:</b> Recognize the need for independent and life-long learning.	Learned new deep learning techniques, stayed updated with recent research	3
12	<b>PSO1 – Computer-based systems development:</b> Design computer-based systems.	Developed complete neural network system for image transformation	3
13	<b>PSO2 – Software development:</b> Specify, design, and develop applications.	Implemented training and inference pipelines using best practices	3
14	<b>PSO3 – Computer communications and Internet applications:</b> 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
<b>Total</b>	<b>11,500+</b>	<b>6,900+</b>	<b>18,400+</b>	–

All images are preprocessed to  $256 \times 256$  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
<code>models/</code>	Neural network architecture definitions
<code>generator.py</code>	ResNet-based generator with 9 residual blocks
<code>discriminator.py</code>	PatchGAN discriminator implementation
<code>cyclegan.py</code>	Complete CycleGAN model with training logic
<code>utils/</code>	Utility functions and helper modules
<code>dataset.py</code>	Custom PyTorch Dataset class for unpaired data
<code>transforms.py</code>	Image preprocessing and augmentation
<code>losses.py</code>	Loss function implementations (perceptual, cycle)
<code>buffer.py</code>	Image history buffer for discriminator training
<code>train.py</code>	Training script with argument parsing
<code>inference.py</code>	Inference script for style transfer
<code>checkpoints/</code>	Saved model weights (~150 MB per style)
<code>data/</code>	Training datasets organized by style
<code>requirements.txt</code>	Python package dependencies

### E.2 Installation Guide

#### E.2.1 Installation Steps

1. Clone Repository:

```
1 | git clone https://github.com/user/cyclegan-style-transfer.git
```

```
2 cd cyclegan-style-transfer
```

## 2. Create Virtual Environment:

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

## 3. Install Dependencies:

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

## 4. Verify CUDA Installation:

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