ART GENERATION WITH NEURAL STYLE TRANSFER

A dissertation submitted to the Jawaharlal Nehru Technological University, Hyderabad in partial fulfilment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

B. ASHISH PREETHAM (20B81A0566)

SREENIDHI PAKA (20B81A05A7)

Under the Guidance of

Mr. N.N.S.S.S. Adithya
Assistant Professor

Department of Computer Science and Engineering

CVR COLLEGE OF ENGINEERING

(An UGC Autonomous Institution, Affiliated to JNTUH, Accredited by NBA, and NAAC)
Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),
Ranga Reddy (Dist.) - 501510, Telangana State

2023-24

CVR COLLEGE OF ENGINEERING

(An UGC Autonomous Institution, Affiliated to JNTUH,
Accredited by NBA, and NAAC)
Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),
Ranga Reddy (Dist.) - 501510, Telangana State.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **"ART GENERATION WITH NEURAL STYLE TRANSFER"** is being submitted by **B ASHISH PREETHAM (20B81A0566), SREENIDHI PAKA (20B81A05A7),** in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering,** during the academic year 2023-2024.

Project Guide	**Professor-in-charge projects**
N.N.S.S.S. Adithya	**Dr. G. Balakrishna**
Assistant Professor	

External Examiner	**Professor and Head, CSE**
	Dr. A. Vani Vathsala

DECLARATION

We hereby declare that the project entitled **"ART GENERATION WITH NEURAL STYLE TRANSFER"** submitted by us to CVR College of Engineering in partial fulfilment of the requirement for the award of degree of B. Tech in COMPUTER SCIENCE AND ENGINEERING is a record of major project work carried out by us under the esteemed guidance of **N.N.S.S.S. Adithya (Assistant Professor).** We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or in any other institute or university.

BIRUDHURAJU ASHISH PREETHAM (20B81A0566)

PAKA SREENIDHI (20B81A05A7)

Date:

Place:

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We avail this opportunity to express our deep sense of gratitude and hearty thanks to management of CVR College of Engineering, for providing congenial atmosphere and encouragement.

We would like to thank to our principal **Dr K. Rammohan Reddy,** Vice Principal Prof. **L. C. Siva Reddy, Mrs. A. Vani Vathsala, Head of the Department, Computer Science and Engineering** for her expert guidance and encouragement at various levels of our Project.

We are thankful to our guide **N.N.S.S.S. Adithya** sir, for his sustained inspiring Guidance and cooperation throughout the process of this project. His wise counsel and suggestions were valuable. Our project coordinator, **Mr. B. Srikanth**, Senior Assistant Professor, CSE, provided us with all the necessary information, timelines, and inputs when and where required for the project's successful completion.

We express our deep sense of gratitude and thanks to all the **Teaching** and **Non-Teaching Staff** of our college who stood with us during the project and helped us to make it a successful venture.

We convey our heartfelt thanks to management for providing excellent lab facilities and tools. Finally, we thank all those guidance helpful to us in this regard.

ABSTRACT

Neural Style Transfer is the technique of blending style from one image into another image keeping its content intact. The only change is the style configurations of the image to give an artistic touch to your image. It is an application of Computer Vision related to image processing techniques and Deep Convolutional Neural Networks. Neural Style Transfer deals with two sets of images: Content image and Style image. Style transfer works by activating the neurons in a particular way, such that the output image and the content image should match particularly in the content, whereas the style image and the desired output image should match in texture and capture the same style characteristics in the activation maps. Convolutional Neural Networks (CNNs) form the backbone of this process, with architectures like VGG19, a deep convolutional neural network architecture, comprises 19 layers, including 16 convolutional layers and three fully connected layers. It excels at image classification tasks by employing small 3x3 convolutional filters throughout its deep structure, capturing intricate features and patterns in the input images. Algorithms, such as the Gram matrix computation for style representation and Total Variation minimization for smoothing, contribute to refining the generated artwork. Additionally, it discusses challenges, advancements, and future directions in the dynamic field of Art Generation through Neural Style Transfer.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS:

NST-Neural Style Transfer.

CNN-Convolutional Neural Network.

UI-User Interface.

UAT-User Acceptance Testing.

VGG19- Visual Geometry Group (Variant of vgg model with 19 layers).

CV-Computer Vision.

PIL-Python Imaging Library.

AI-Artificial Intelligence.

DNN-Deep Neural Network.

GUI-Graphical User Interface.

CPU-Central Processing Unit.

RGB-Red, Green, Blue.

GUI-Graphical User Interface

1.INTRODUCTION

Art Generation with Neural Style Transfer is a fascinating technique that lets us blend the style of one image with the content of another while keeping the content intact. It's like taking a regular photo and giving it an artistic makeover inspired by famous artworks or styles. This process uses advanced computer vision and deep learning techniques, particularly Convolutional Neural Networks (CNNs), to achieve stunning results. By activating specific neurons in the network, we can ensure that the final image maintains the content of the original while adopting the texture and style characteristics of the chosen style image. Architectures like VGG19, with its intricate layers, play a crucial role in this process by capturing detailed features and patterns. Various algorithms, such as Gram matrix computation and Total Variation minimization, further refine the generated artwork, ensuring a visually appealing result. Through this method, we can explore endless creative possibilities and pave the way for exciting advancements in the field of digital art.

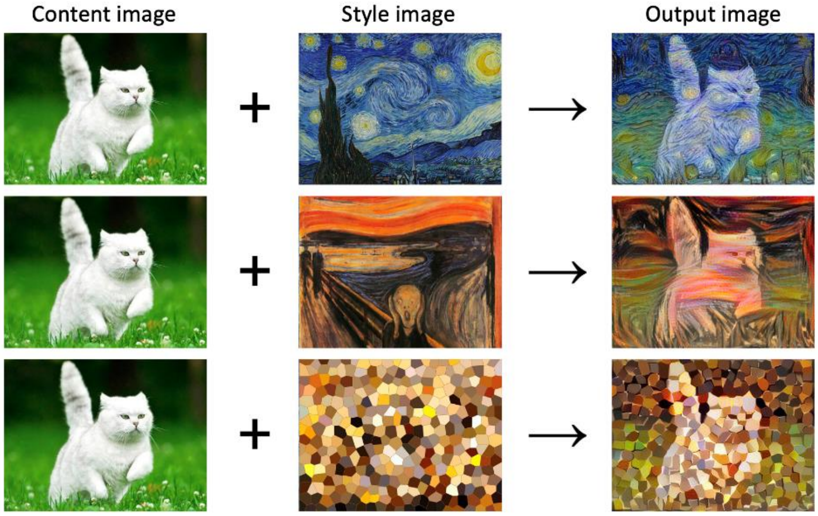


Figure 1. Introduction

In Figure 1, we present a captivating illustration demonstrating the application of Neural Style Transfer to images of a cat. The figure showcases three distinct styles being applied to the original content image of the cat. Each style transformation offers a unique artistic interpretation, showcasing the versatility of the technique. Finally, the figure reveals the stunning result: the cat image transformed into the target style, representing the culmination of the Neural Style Transfer process.

INRODUCTION TO CNN AND VGG19

Convolutional Neural Networks (CNNs) play a pivotal role in the art generation process using Neural Style Transfer (NST) projects. CNNs are a class of deep learning models designed specifically for processing and analysing visual data, such as images. In the context of NST, CNNs serve as the backbone of the style transfer algorithm, enabling the generation of artistic images that blend the content of one image with the style of another. The CNN architecture, such as VGG19, is utilized to extract and represent the content and style features of input images. Through a series of convolutional and pooling layers, CNNs capture hierarchical representations of features, ranging from low-level details like edges and textures to high-level concepts like object shapes and patterns. These learned feature representations are then leveraged in the style transfer process to reconstruct an output image that preserves the content of the content image while incorporating the artistic style of the style image. By harnessing the power of CNNs, NST projects are able to create visually compelling and stylistically coherent artworks, offering a unique intersection of art and artificial intelligence.

VGG19 Architecture

In the art generation project using Neural Style Transfer (NST), the VGG19 architecture plays a pivotal role in feature extraction. VGG19 stands for "Visual Geometry Group 19," a deep convolutional neural network (CNN) architecture. It comprises 19 layers, including 16 convolutional layers and 3 fully connected layers. VGG19 excels at image classification tasks by employing small 3x3 convolutional filters throughout its deep structure, capturing intricate features and patterns in the input images. These layers are used to extract content and style features from the input images, facilitating the generation of artistically stylized outputs through NST.

The VGG19 architecture consists of 19 layers, including 16 convolutional layers and 3 fully connected layers. Here's a breakdown of the layers:

Convolutional Layers (Conv): The convolutional layers perform feature extraction by applying a set of learnable filters (kernels) to the input images. These filters slide across the input image, capturing different features such as edges, textures, and patterns at various spatial scales.

Max Pooling Layers (MaxPooling): Max pooling layers down sample the feature maps obtained from the convolutional layers by selecting the maximum value within each local region. This helps in reducing the spatial dimensions of the feature maps while retaining the most important information.

Fully Connected Layers (Dense): The fully connected layers process the high-level features extracted by the convolutional layers and perform classification or regression tasks. These layers are composed of neurons that are densely connected to the neurons in the preceding layer.

VGG19 collects features based on hierarchical abstraction. As the input image passes through the network, each layer extracts increasingly abstract and complex features. The initial layers capture low-level features such as edges and textures, while deeper layers capture higher-level features such as object parts and entire objects.

The feature extraction process is driven by the learned parameters (weights and biases) of the convolutional filters. During training, these parameters are optimized using backpropagation and gradient descent to minimize a predefined loss function, allowing the network to learn to extract discriminative features relevant to the task at hand, such as image classification or style transfer.

1.1 Motivation:

The main motivation for our project, "Art Generation with Neural Style Transfer," is to explore how technology can help people create beautiful and unique artworks easily. By using neural networks, we aim to enable anyone, regardless of their artistic skill, to transform ordinary photos into stunning pieces of art inspired by famous artists or artistic styles. Our goal is to make the process of artistic expression more accessible and enjoyable, allowing individuals to unleash their creativity and produce visually striking compositions with just a few clicks. Through this project, we hope to showcase the exciting possibilities that emerge when art and technology come together, inspiring people to explore their creative potential and share their unique visions with the world.

Furthermore, the technology's impact extends into the digital landscape, where it enhances the visual appeal of content, contributing to the evolution of generative art and establishing itself as a transformative force in the dynamic synergy between technology and artistic exploration.

Some key motivations for undertaking such projects:

Blend of Styles

Experimentation with Styles

Personalization and Customization

Understanding Neural Networks

Visual Effects in media and entertainment

Inspiration from Art History

Showcasing AI Capabilities

Advancements in Image Processing

1.2 Project Objectives:

1. Understand NST: To understand Neural Style Transfer (NST), grasp its fundamental principles and algorithms, which involve blending the content of one image with the style of another. Selecting a suitable NST model is crucial, as it determines the effectiveness and quality of the style transfer process, ensuring optimal artistic results in the final output.
2. Data Preprocessing: To prepare data for Neural Style Transfer (NST), collect a diverse set of content and style images. Preprocess the data by resizing images to a consistent size and normalizing pixel values to facilitate efficient model training and style transfer.
3. Model Training: Train the Neural Style Transfer (NST) model using the collected dataset, adjusting hyperparameters for optimal performance and efficiency. Fine-tune the model to achieve the desired style transfer results.
4. Evaluate Performance: Evaluate the performance of the Neural Style Transfer model by defining metrics for art quality and verifying its ability to generalize across different styles and content.
5. UI Design: Design a user-friendly interface enabling users to upload images, adjust parameters, and preview/download generated artwork.
6. Real-Time Transfer: Optimize Neural Style Transfer for real-time application by exploring lightweight architectures.
7. Customization: Enable customization of the Neural Style Transfer process by providing user control over parameters, facilitating experimentation, and allowing fine-tuning for optimal results.
8. Compatibility: Ensure compatibility by optimizing the Neural Style Transfer application for cross-device and cross-platform accessibility, including both desktop and mobile environments.
9. Documentation: Develop comprehensive user guides as part of the documentation process.
10. Community Engagement: Foster user feedback and community engagement to facilitate sharing experiences and enhance the user community.
11. Performance Optimization: Optimize the performance of the model to enhance efficiency and handle larger images or batches while maintaining quality.

12. Ethical Considerations: Ethical considerations in Neural Style Transfer involve addressing content usage and copyright concerns while implementing measures to prevent misuse.

1.3 Problem statement:

The field of art generation using Neural Style Transfer (NST) presents both opportunities and challenges. While NST offers a novel approach to creating artistic images by combining the content of one image with the style of another, several issues persist that hinder its widespread adoption and effectiveness.

One of the primary challenges in NST is achieving a balance between preserving the content of the input image and faithfully transferring the style characteristics from the reference image. Often, NST algorithms struggle to retain the essential features of the content image while applying the desired artistic style, leading to distortions or loss of important details. Moreover, the quality of the generated artwork heavily depends on the choice of NST model, hyperparameters, and the diversity of the training dataset. Therefore, there is a pressing need to develop more robust NST models that can accurately capture and transfer style while preserving content fidelity.

Another significant concern in art generation with NST is the computational complexity and resource requirements associated with model training and inference. Training deep neural networks for NST typically involves processing large datasets of content and style images, which demands substantial computational power and memory resources. Additionally, the inference process, where the model generates stylized images in real-time, can be computationally intensive, especially for high-resolution images. As a result, there is a need for optimization techniques and hardware accelerators to improve the efficiency and scalability of NST algorithms.

Furthermore, ethical considerations surrounding the use of NST for art generation cannot be overlooked. Issues such as copyright infringement, intellectual property rights, and cultural appropriation may arise when using NST to create and distribute artistic content. It is crucial to establish guidelines and best practices for responsibly using NST models, addressing concerns related to content usage, ownership, and attribution.

In summary, the problem statement for art generation using Neural Style Transfer encompasses the need for improving the fidelity and robustness of NST models, optimizing computational efficiency, and addressing ethical considerations. By tackling these challenges, we aim to advance the field of computer-generated art and enable more accessible and responsible use of NST technology for creative expression.

1.4 Project report Organization:

1. Title Page
2. Abstract
3. Table of Contents
4. Introduction
5. Literature Review
6. Methodology
7. Implementation
8. Results
9. Discussion
10. UI Design
11. Conclusion
12. Future Work
13. References

2.LITERATURE SURVEY

source	Author	Year	Methodology	Key Findings	Limitations
1	Gatys et al.	2015	Neural Style Transfer (NST)	Introduces NST using deep neural networks	Limited customization, lack of spatial coherence
2	Ulyanov et al.	2016	Instance Normalization for NST	Improves NST with instance normalization	Limited exploration of diverse artistic styles
3	Liao et al.	2017	Adaptive Instance Normalization	Proposes adaptive instance normalization in NST	Overfitting to training data, lack of user control
4	Jing et al.	2018	Arbitrary Style Transfer with CNN	Applies arbitrary style transfer with CNN	Limited exploration of temporal aspects

2.1 Existing Works

1. "A Neural Algorithm of Artistic Style" (2015) by Gatys et al.:

- Introduction to Neural Style Transfer using Convolutional Neural Networks (CNNs).

2."Instance Normalization: The Missing Ingredient for Fast Stylization" (2017) by Ulyanov et al.:

- Explores instance normalization for accelerated and improved style transfer.

3."Perceptual Losses for Real-Time Style Transfer and Super-Resolution" (2016) by Johnson et al.:

- Proposes perceptual loss functions for real-time style transfer.

4. "Fast Neural Style Transfer via Instance Normalization" (2016) by Huang and Belongie:

- Examines instance normalization's role in speeding up style transfer.

5. "Preserving Colour in Neural Artistic Style Transfer" (2016) by Gatys et al.:

- Focuses on preserving colour information during style transfer.

6. "Artistic Style Transfer for Videos with Convolutional Neural Networks" (2017) by Ruder et al.:

- Extends neural style transfer to video content.

7. "Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization" (2017) by Huang et al.:

- Presents adaptive instance normalization for arbitrary style transfer.

8. "Exploring the Limits of Weakly Supervised Pretraining" (2021) by Mahajan et al.:

- Investigates weakly supervised pretraining for image generation.

9. "Image Style Transfer Using Convolutional Neural Networks" (2016) by Li et al.:

- Discusses CNNs for image style transfer.

10. "Artistic style transfer for videos: A comprehensive review" (2020) by Zhang et al.:

- Provides a comprehensive review of video style transfer advancements.

2.2 Limitations of Existing Work

1. Computational Intensity: Existing methods often demand significant computational resources, limiting real-time applications.

2. Memory Requirements: High memory usage during training and inference hampers deployment on resource-limited devices.

3. Loss of Content Details: Some models struggle to preserve intricate content details during style transfer.

4. Overemphasis on Style Features: Certain algorithms may overly emphasize style, potentially distorting the original content.

5. Limited Style Diversity: Models may not generalize well across diverse artistic styles.

6. Colour Distortion: Challenges exist in preserving accurate colour representation, leading to distortions.

7. Lack of User Control: Limited customization options and user control over artistic output.

8. Difficulty in Video Styling: Adapting style transfer to videos is computationally demanding and challenging.

9. Style Transfer Consistency: Achieving consistent style transfer across resolutions and aspect ratios is challenging.

10. Robustness to Noisy Inputs: Sensitivity to input noise can result in undesired distortions in generated art.

3.REQUIREMENT ANALYSIS

3.1 Functional and Non-Functional requirements

Functional Requirements:

1. Style Transfer
2. Real-Time Processing
3. Customization
4. User Interface (UI)
5. Compatibility
6. Batch Processing
7. Video Style Transfer
8. Performance Metrics

Non-Functional Requirements:

1. Efficiency
2. Memory Efficiency
3. Scalability
4. Robustness
5. Security Measures
6. Ethical Considerations
7. Accessibility
8. Documentation
9. Community Engagement
10. Update and Maintenance

3.2 Software requirements

Python 3.11.4 (bother for backend and front end)

Libraries:

TensorFlow and Pytorch (DNN Libraries)

OpenCV (Image Processing)

Pillow (Image Handling)

Pandas (Data Frame)

NumPy (Numeric, Array and Matrix Operations)

PyWebIo (User Web Interface)

3.3 Hardware requirements

Development

NVIDIA P100 8 GB GPU

I5 8^{th} gen CPU

16 GB RAM

256 SSD

Ubuntu 20.04 LTS OS

With CUDA, CUDNN Setup

(Which we will be compensating with Kaggle)

Deployment/Inference

I3 5^{th} gen > CPU

4 GB RAM

Windows OS

256 > HDD/SSD

4.SYSTEM DESIGN

4.1 Proposed Methodology

1. Problem Definition: Clearly define the problem and objectives of the art generation project. Determine the target audience and the intended purpose of the generated artwork.

2. Data Collection: Collect a diverse dataset of content and style images that represent the desired artistic styles. Ensure a balance between content-rich images and images with distinct styles.

3. Preprocessing: Resize and normalize the collected images to a consistent format suitable for the NST model. Extract features from the images that will be used for content and style representations.

4. Model Selection: Choose a suitable pre-trained neural network for NST, such as VGG19 or another architecture. Fine-tune the model if necessary to align with the project's artistic goals.

5. Feature Extraction: Implement functions to extract content and style features from the chosen layers of the neural network. Verify that the extracted features capture relevant content and style information.

6. Loss Function Design: Develop loss functions to measure the difference between the generated image and the content image (content loss) and the style image (style loss). Fine-tune the weighting of content and style losses to achieve desired trade-offs.

7. Optimization Process: Utilize an optimization algorithm (e.g., Adam optimizer) to iteratively update the generated image to minimize the total loss. Set hyperparameters like learning rate, number of iterations, and convergence criteria.

8. User Interface Design: Create an interactive and user-friendly interface for users to upload content and style images, adjust parameters, and preview generated artwork. Include options for users to download the final stylized artwork.

9. Testing and Evaluation: Conduct thorough testing with different combinations of content and style images. Collect user feedback to assess the quality and user-friendliness of the generated artwork.

10. Fine-tuning and Optimization: Based on user feedback and evaluation results, refine the model, loss functions, and parameters to enhance the quality of generated art.

11. Deployment: Deploy the NST model and user interface, making it accessible to the target audience. Ensure scalability and responsiveness of the deployed system.

12. Monitoring and Maintenance: Implement monitoring mechanisms to track system performance and user interactions.

This proposed methodology provides a structured approach to developing an art generation project with Neural Style Transfer.

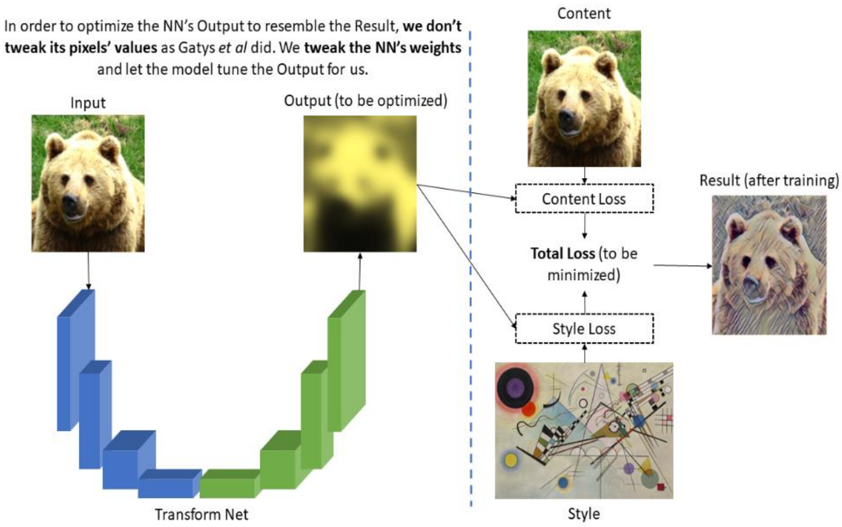


Figure 2.Proposed approach of work.

Figure 2 showcases the transformation process inspired by Gatys et al.'s research, where content and style images are combined to generate stylized outputs while preserving the content structure. Through this process, the images undergo controlled alterations, maintaining the essence of the content while incorporating stylistic elements from the reference image.

4.2 Architecture

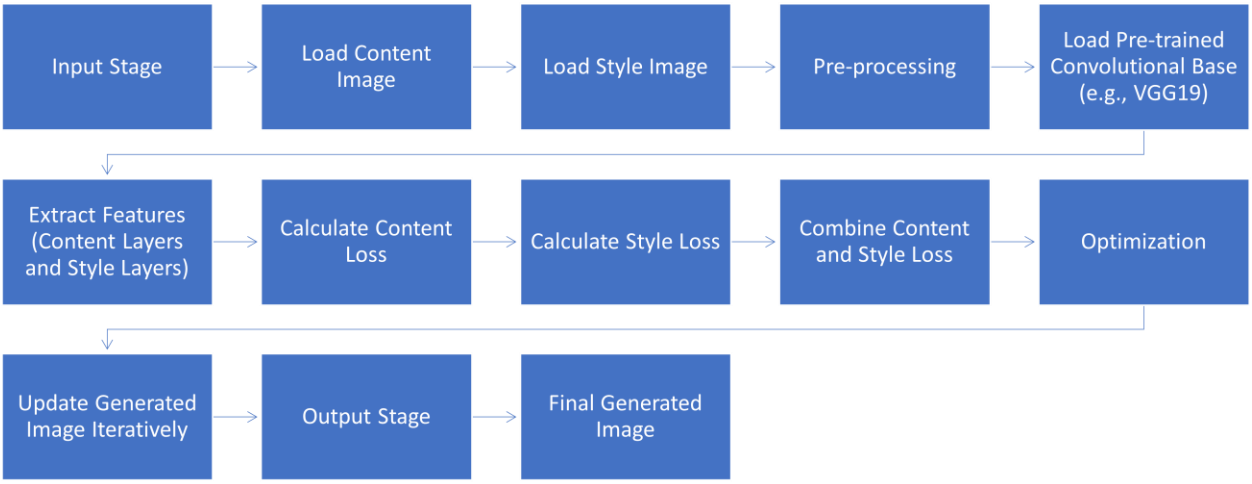


Figure 3 .Architecture

The architecture as shown in figure 2 in our project documentation provides a concise overview of the neural style transfer process, showcasing the deep neural network framework for separating and recombining content and style features from input images to generate artistically stylized outputs.

1. User Interaction: Users provide content and style images through the interface.

2. Preprocessing: Input images are pre-processed to ensure compatibility with the chosen neural network.

3. Feature Extraction: The pre-trained convolutional base extracts content and style features from the input images.

4. Loss Calculation: Content loss and style loss are calculated based on the extracted features.

5. Loss Combination: Content loss and style loss are combined with user-defined weights.

6. Optimization: The optimization algorithm minimizes the total loss to update the generated image.

7. Iteration Loop: The generated image is iteratively updated until convergence.

8. Output Generation: The final stylized artwork is presented to the user.

This architectural overview outlines the flow of the art generation process with Neural Style Transfer, from user input to the generation of the final stylized image.

4.3 UML Diagrams

The system design for the art generation project with Neural Style Transfer comprises a user-friendly interface allowing users to upload content and style images. The architecture includes a pre-processing module for image normalization, a pre-trained convolutional base for feature extraction, and an optimization algorithm to iteratively update the generated image based on calculated content and style losses. The system aims for seamless user interaction, efficient feature extraction, and effective loss optimization, resulting in a visually appealing final stylized artwork.

4.3.1 Use Case Diagram

A use case diagram visually represents the interactions between actors and the system to illustrate the functionality of the system. In the context of our project on art generation with neural style transfer, the use case diagram showcases actors (users) interacting with the system to generate artistically stylized images through neural style transfer.

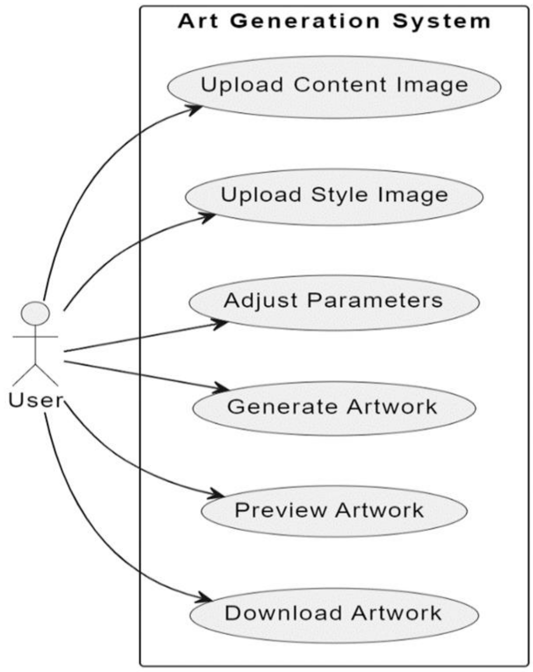


Figure 4. Use Case Diagram

4.3.2 Class Diagram

The class diagram for our "Art Generation with Neural Style Transfer" project illustrates the key classes and their relationships, providing a structural overview of the system's components and interactions.

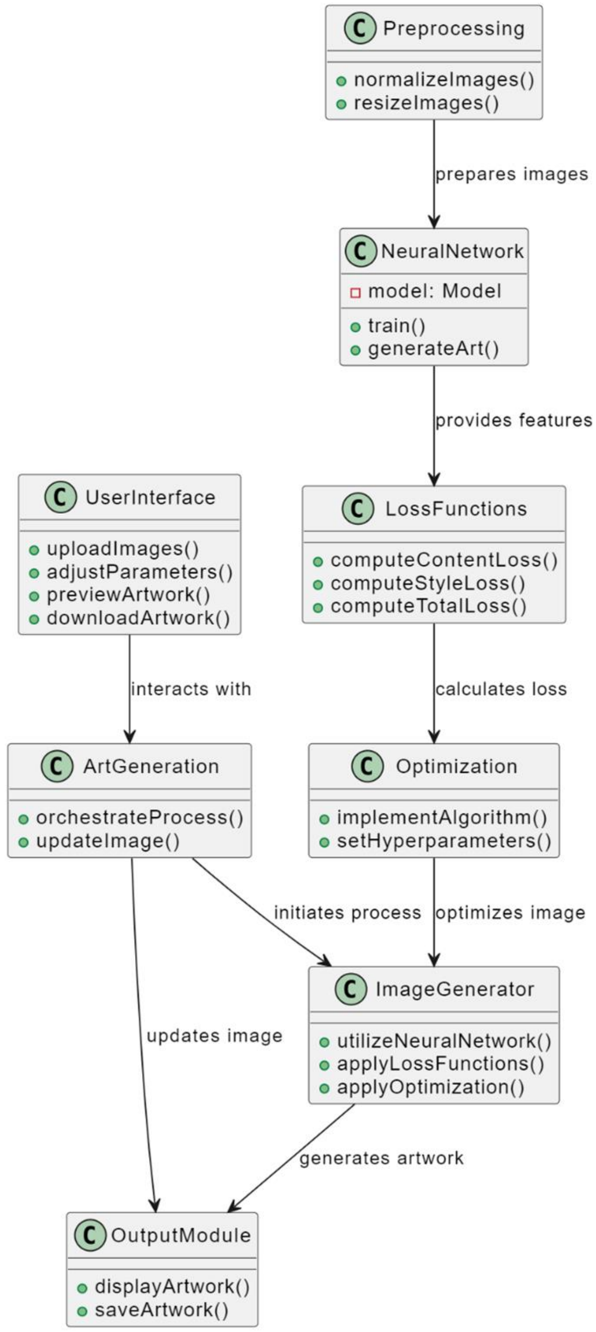


Figure 5. Class Diagram

4.3.3 Sequence Diagram

A sequence diagram illustrates the interactions between different components or objects in a system over time, depicting the flow of messages or method calls between them. In our project, a sequence diagram can visually depict the flow of operations between components such as data preprocessing, neural network inference, and image rendering, elucidating the sequential steps involved in generating stylized artwork.

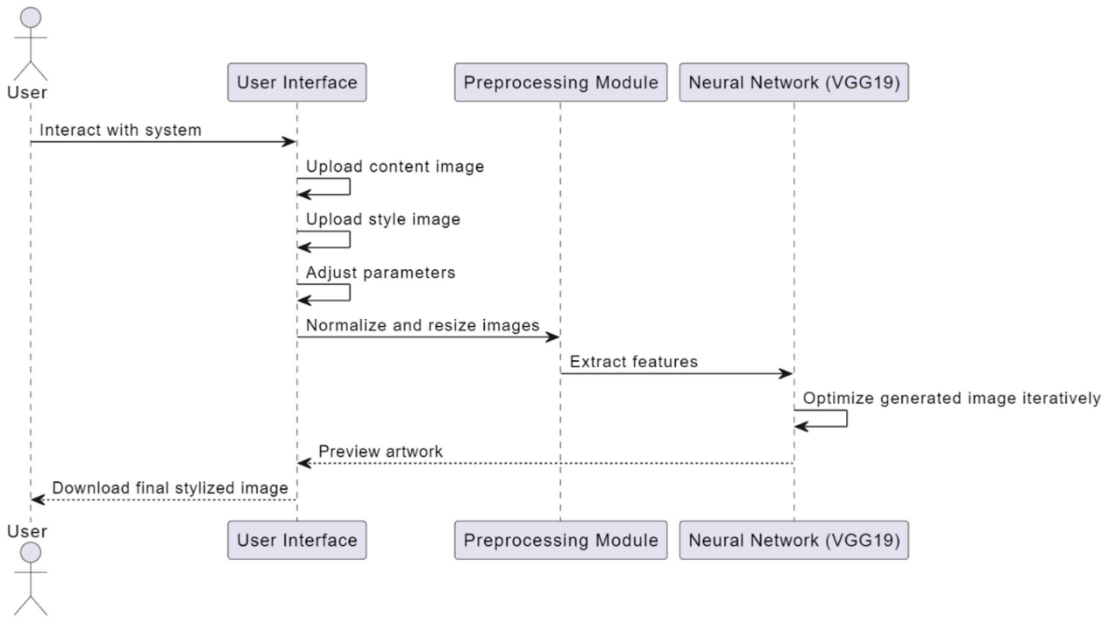


Figure 6. Sequence Diagram

4.3.4 Collaboration Diagram

A collaboration diagram visually represents the relationships and interactions between objects or components within a system, depicting how they work together to achieve a common goal. In our project, a collaboration diagram showcases the cooperative efforts between modules such as data preprocessing, neural network inference, and image rendering to produce stylized artwork.

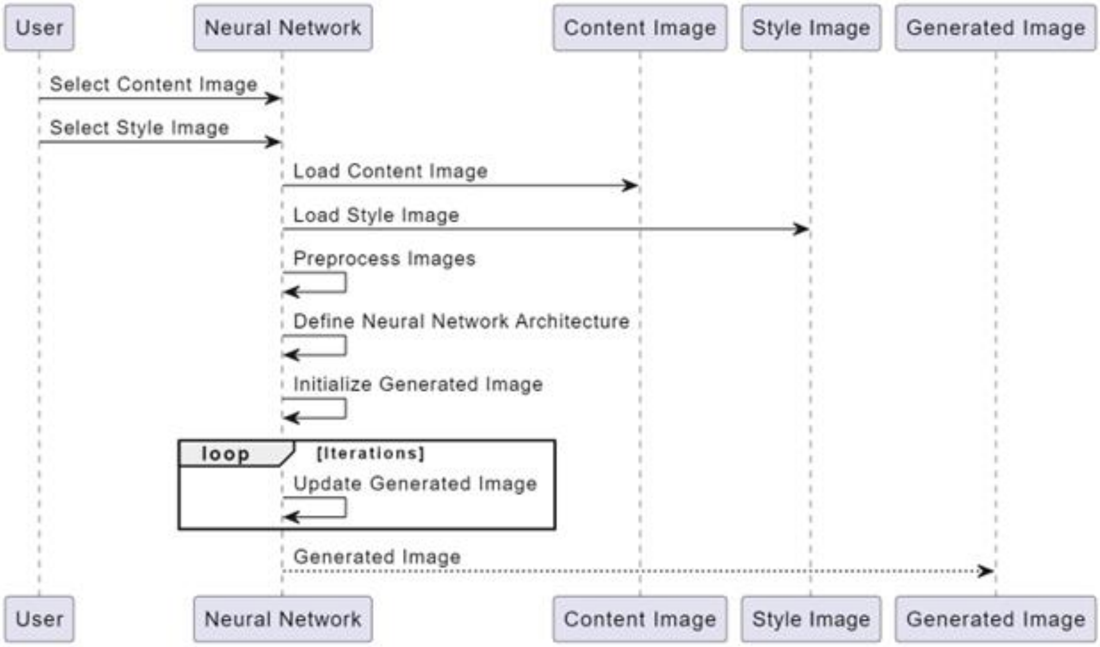


Figure 7. Collaboration Diagram

4.3.5 Activity Diagram

An activity diagram visually represents the flow of activities or processes within a system, showing the sequence of actions and decisions. It provides a clear overview of the steps involved in our project, such as data preprocessing, neural style transfer, and output generation, facilitating understanding of the workflow in art generation with neural style transfer.

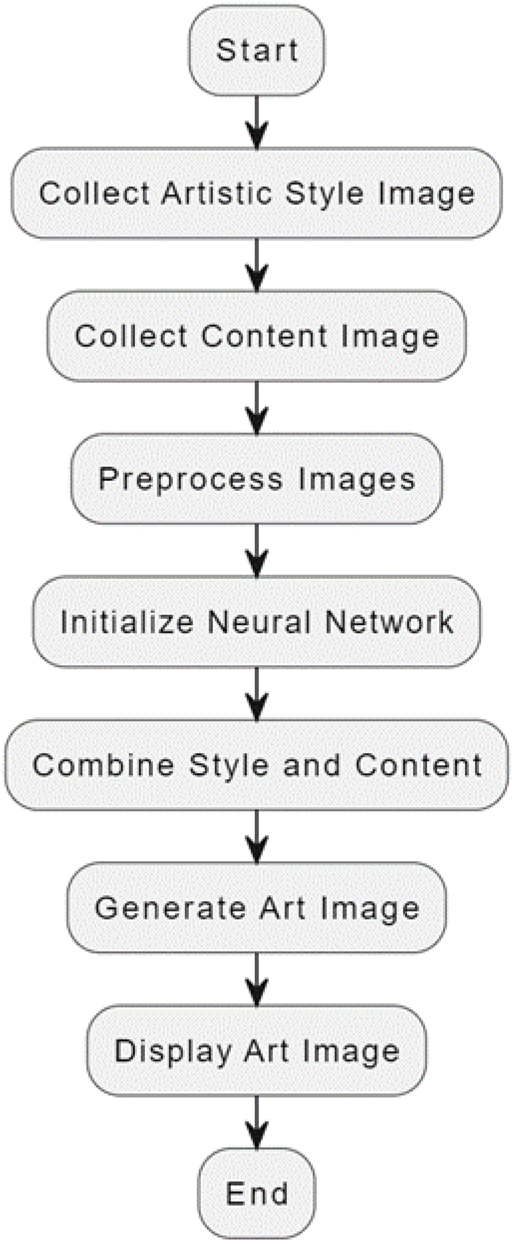


Figure 8. Activity Diagram

4.4. Data Set Requirements

We will collect our own dataset with two types of images.

1.Hand drawn Art Styles like Acrylic, Dotted, photorealistic and Retro – Style Image

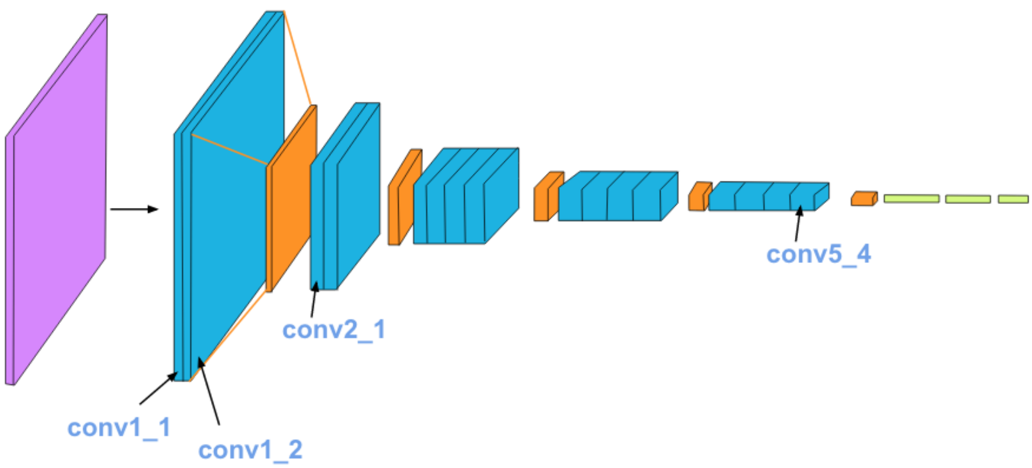
2.Target images which is captured with cell phones and computer web cam – Content Image

5. IMPLEMENTATION

Artistic Neural Style Transfer using Pytorch.

In this kernel, we'll implement the style transfer method that is outlined in the paper, Image Style Transfer Using Convolutional Neural Networks, by Gatys in Pytorch.

In this paper, style transfer uses the features found in the 19-layer VGG Network, which is comprised of a series of convolutional and pooling layers, and a few fully connected layers. In the image below, the convolutional layers are named by stack and their order in the stack. Conv_1_1 is the first convolutional layer that an image is passed through, in the first stack. Conv_2_1 is the first convolutional layer in the *second* stack. The deepest convolutional layer in the network is conv_5_4.



Separating Style and Content.

Style transfer relies on separating the content and style of an image. Given one content image and one style image, we aim to create a new, *target* image which should contain our desired content and style components:

1.objects and their arrangement are similar to that of the **content image.**

2.style, colours, and textures are similar to that of the **style image.**

An example is shown below, where the content image is of a cat, and the style image is of Hokusai's Great Wave. The generated target image still contains the cat but is stylized with the waves, blue and beige colours, and block print textures of the style image!


```
# import resources
%matplotlib inline

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

import torch
import torch.optim as optim
from torchvision import transforms, models
```

Load in VGG19 (features)

```
# get the "features" portion of VGG19 (we will not need the "classifier" portion)
vgg = models.vgg19(pretrained=True).features

# freeze all VGG parameters since we're only optimizing the target image
for param in vgg.parameters():
```

```
    param.requires_grad_(False)
```

```
# move the model to GPU, if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

vgg.to(device)
```

Load in Content and Style Images

```
def load_image(img_path, max_size=400, shape=None):
    ''' Load in and transform an image, making sure the image
       is <= 400 pixels in the x-y dims.'''

    image = Image.open(img_path).convert('RGB')

    # large images will slow down processing
    if max(image.size) > max_size:
        size = max_size
    else:
        size = max(image.size)

    if shape is not None:
        size = shape

    in_transform = transforms.Compose([
                        transforms.Resize(size),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),
                                             (0.229, 0.224, 0.225))])

    # discard the transparent, alpha channel (that's the :3) and add the batch dimensio
n
```

```
    image = in_transform(image)[:3,:,:].unsqueeze(0)

    return image
```

Next, I'm loading in images by file name and forcing the style image to be the same size as the content image.

```
# load in content and style image
content = load_image('/kaggle/input/art-style-transfer/input_image/RKS.jpg').to(device)
# Resize style to match content, makes code easier
style = load_image('/kaggle/input/art-style-transfer/target_style/mona.jpg', shape=content.shape[-2:]).to(device)
```

```
# helper function for un-normalizing an image
# and converting it from a Tensor image to a NumPy image for display
def im_convert(tensor):
    """ Display a tensor as an image. """

    image = tensor.to("cpu").clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1,2,0)
    image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456, 0.406))
    image = image.clip(0, 1)

    return image
```

```
# display the images
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
# content and style ims side-by-side
ax1.imshow(im_convert(content))
ax1.set_title("Content Image",fontsize = 20)
ax2.imshow(im_convert(style))
```

```
ax2.set_title("Style Image", fontsize = 20)
plt.show()
```

VGG19 Layers

To get the content and style representations of an image, we have to pass an image forward through the VGG19 network until we get to the desired layer(s) and then get the output from that layer.

```
# print out VGG19 structure so you can see the names of various layers
print(vgg)
```

Content and Style Features

```
def get_features(image, model, layers=None):
    """ Run an image forward through a model and get the features for
      a set of layers. Default layers are for VGGNet matching Gatys et al (2016)
```

```
    """

    ## TODO: Complete mapping layer names of PyTorch's VGGNet to names from the paper
    ## Need the layers for the content and style representations of an image
    if layers is None:
        layers = {'0': 'conv1_1',
                  '5': 'conv2_1',
                  '10': 'conv3_1',
                  '19': 'conv4_1',
                  '21': 'conv4_2',  ## content representation
                  '28': 'conv5_1'}

    features = {}
    x = image
    # model._modules is a dictionary holding each module in the model
    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x

    return features
```

Gram Matrix

The output of every convolutional layer is a Tensor with dimensions associated with the batch size, a depth, d and some height and width (h, w). The Gram matrix of a convolutional layer can be calculated as follows:

- Get the depth, height, and width of a tensor using batch size, d, h, w = tensor. Size
- Reshape that tensor so that the spatial dimensions are flattened.
- Calculate the gram matrix by multiplying the reshaped tensor by its transpose.

Note: You can multiply two matrices using torch.mm (matrix1, matrix2)

```
def gram_matrix(tensor):
    """ Calculate the Gram Matrix of a given tensor
        Gram Matrix: https://en.wikipedia.org/wiki/Gramian_matrix
    """

    # get the batch_size, depth, height, and width of the Tensor
    _, d, h, w = tensor.size()

    # reshape so we're multiplying the features for each channel
    tensor = tensor.view(d, h * w)

    # calculate the gram matrix
    gram = torch.mm(tensor, tensor.t())

    return gram
```

```
# get content and style features only once before training
content_features = get_features(content, vgg)
style_features = get_features(style, vgg)

# calculate the gram matrices for each layer of our style representation
style_grams = {layer: gram_matrix(style_features[layer]) for layer in style_features}

# create a third "target" image and prep it for change
# it is a good idea to start of with the target as a copy of our *content* image
# then iteratively change its style
target = content.clone().requires_grad_(True).to(device)
```

Loss and Weights

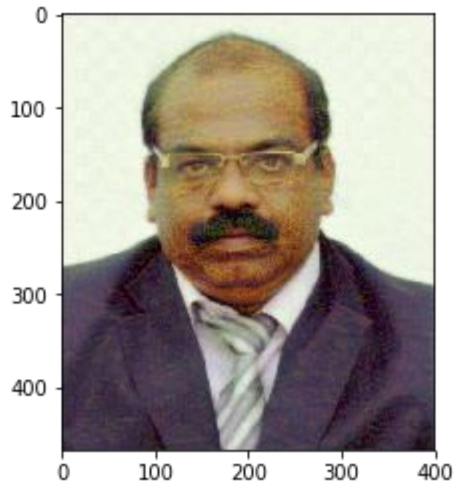
```
# weights for each style layer
# weighting earlier layers more will result in *larger* style artifacts
# notice we are excluding `conv4_2` our content representation
style_weights = {'conv1_1': 1.,
                 'conv2_1': 0.75,
                 'conv3_1': 0.2,
                 'conv4_1': 0.2,
                 'conv5_1': 0.2}

content_weight = 1  # alpha
style_weight = 1e9  # beta
```

Updating the Target & Calculating Losses

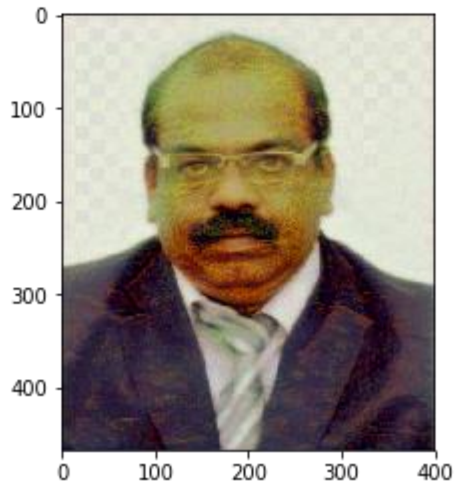
```
content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2'])*
*2)

# for displaying the target image, intermittently
show_every = 400

# iteration hyperparameters
optimizer = optim.Adam([target], lr=0.003)
steps = 2000  # decide how many iterations to update your image (5000)

for ii in range(1, steps+1):

    # get the features from your target image
    target_features = get_features(target, vgg)

    # the content loss
```

```
    content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2']
)**2)

    # the style loss
    # initialize the style loss to 0
    style_loss = 0
    # then add to it for each layer's gram matrix loss
    for layer in style_weights:
        # get the "target" style representation for the layer
        target_feature = target_features[layer]
        target_gram = gram_matrix(target_feature)
        _, d, h, w = target_feature.shape
        # get the "style" style representation
        style_gram = style_grams[layer]
        # the style loss for one layer, weighted appropriately
        layer_style_loss = style_weights[layer] * torch.mean((target_gram - style_gram)
**2)
        # add to the style loss
        style_loss += layer_style_loss / (d * h * w)

    # calculate the *total* loss
    total_loss = content_weight * content_loss + style_weight * style_loss

    # update your target image
    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()

    # display intermediate images and print the loss
    if  ii % show_every == 0:
        print('Total loss: ', total_loss.item())
        plt.imshow(im_convert(target))
        plt.show()
```

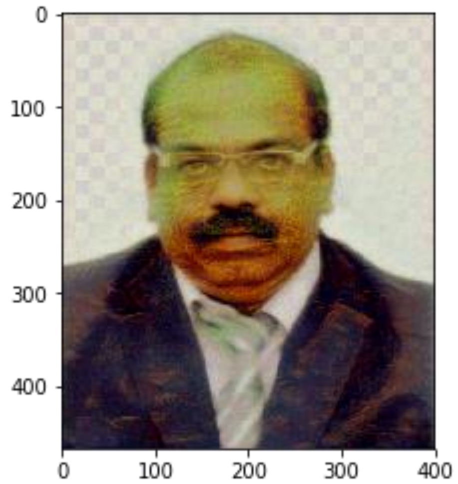
Total loss: 15016456192.0



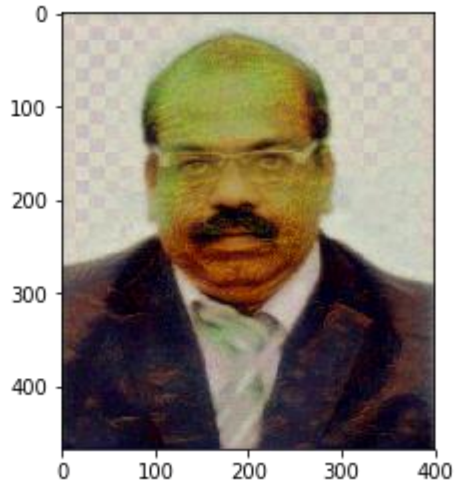
Total loss: 6737242112.0



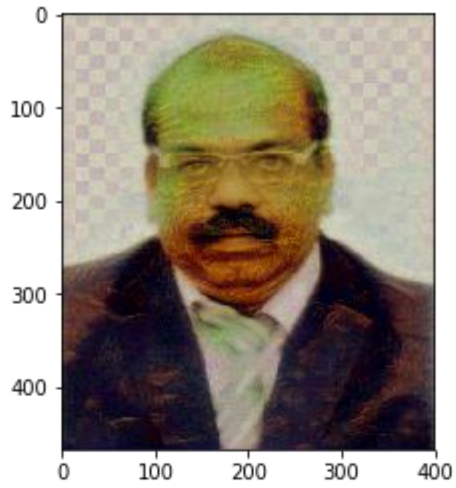
Total loss: 3311703040.0



Total loss: 1866015872.0



Total loss: 1183230464.0



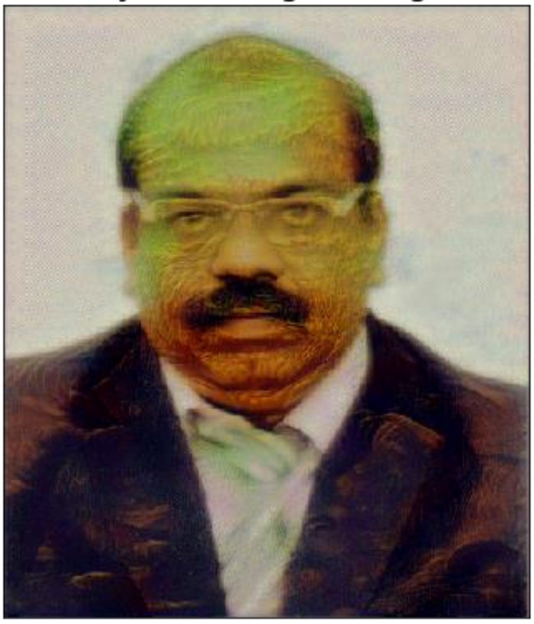
Display the Target Image

```
# display content and final, target image
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 15))
ax1.imshow(im_convert(content))
ax1.set_title("Content Image", fontsize = 20)
ax2.imshow(im_convert(target))
ax2.set_title("Stylized Target Image", fontsize = 20)
ax1.grid(False)
```

```
ax2.grid(False)
# Hide axes ticks
ax1.set_xticks([])
ax1.set_yticks([])
ax2.set_xticks([])
ax2.set_yticks([])
plt.show()
```

Content Image

Stylized Target Image



Front end Implementation Using PyWebIo Interface

PyWebIo is a Python library that simplifies the creation of web-based interfaces for applications like Art Generation with Neural Style Transfer, enhancing accessibility and user interaction.

```
import pywebio as peb
from pywebio import start_server
from inference_tens import Master
import time as t
m = Master()

def app():
    peb.output.put_markdown(r""" # Neural Style Transfer
    Style Base
    """)
    imgs = peb.input.file_upload("Select Content Image:", accept="data/*", multiple=True)
    addr1,addr2 = "",""
    for img in imgs:
        #peb.output.put_image(img['content'])
        addr1 = "data/input_image/" + img['filename']
    imgs = peb.input.file_upload("Select Style Image:", accept="data/*", multiple=True)
    for img in imgs:
        #peb.output.put_image(img['content'])
        addr2 = "data/target_style/" + img['filename']

    org = open(addr1, 'rb').read()
    peb.output.put_image(org, width='300px')
    peb.output.put_text("Content Image")
    style = open(addr2, 'rb').read()
    peb.output.put_image(style, width='300px')
    peb.output.put_text("Style Image")

    with peb.output.put_loading():
            peb.output.put_text("Please Wait...")
            m.put_to(addr1,addr2)
            t.sleep(4)


    out = open('style.jpg', 'rb').read()
    peb.output.put_image(out, width='300px')
    peb.output.put_text("Style Transfer Image")

if __name__ == '__main__':
    start_server(app, port=8080, debug=True)
```

Inference code using pytorch library

from PIL import Image

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.image as mpimg

import torch

```
import torch.optim as optim
from torchvision import transforms, models

class Master:

    def __init__(self):
        # get the "features" portion of VGG19 (we will not need the "classifier" portion)
        self.vgg = models.vgg19(torch.load('vgg19-dcbb9e9d.pth')).features
        # freeze all VGG parameters since we're only optimizing the target image
        for param in self.vgg.parameters():
            param.requires_grad_(False)
        # move the model to GPU, if available
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.vgg.to(self.device)


    def load_image(self,img_path, max_size=400, shape=None):
        ''' Load in and transform an image, making sure the image
       is <= 400 pixels in the x-y dims.'''
        image = Image.open(img_path).convert('RGB')

        # large images will slow down processing
        if max(image.size) > max_size:
            size = max_size
        else:
            size = max(image.size)

        if shape is not None:
            size = shape
```

```
        in_transform = transforms.Compose([
                    transforms.Resize(size),
                    transforms.ToTensor(),
                    transforms.Normalize((0.485, 0.456, 0.406),
                                         (0.229, 0.224, 0.225))])

        # discard the transparent, alpha channel (that's the :3) and add the batch
dimension
        image = in_transform(image)[:3,:,:].unsqueeze(0)

        return image

# content and style image sequezz
    def process_input(self,main_addr,style_addr):

        content = self.load_image(main_addr).to(self.device)
        # Resize style to match content, makes code easier
        style = self.load_image(style_addr, shape=content.shape[-2:]).to(self.device)

        return content,style


    def im_convert(self,tensor):
        """ Display a tensor as an image. """

        image = tensor.to("cpu").clone().detach()
        image = image.numpy().squeeze()
        image = image.transpose(1,2,0)
        image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456,
0.406))
        image = image.clip(0, 1)
```

```
    return image


def get_features(self,image, model, layers=None):
    if layers is None:
        layers = {'0': 'conv1_1',
                  '5': 'conv2_1',
                  '10': 'conv3_1',
                  '19': 'conv4_1',
                  '21': 'conv4_2',  ## content representation
                  '28': 'conv5_1'}

    features = {}
    x = image

    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x

    return features

def gram_matrix(self,tensor):
    # get the batch_size, depth, height, and width of the Tensor
    _, d, h, w = tensor.size()

    # reshape so we're multiplying the features for each channel
    tensor = tensor.view(d, h * w)
```

```
        # calculate the gram matrix
        gram = torch.mm(tensor, tensor.t())

        return gram

    def put_to(self,addr1,addr2):# get content and style features only once before
training

        content,style = self.process_input(addr1,addr2)
        content_features = self.get_features(content, self.vgg)
        style_features = self.get_features(style, self.vgg)

        style_grams = {layer: self.gram_matrix(style_features[layer]) for layer in
style_features}

        target = content.clone().requires_grad_(True).to(self.device)

        style_weights = {'conv1_1': 1.,
                 'conv2_1': 0.75,
                 'conv3_1': 0.2,
                 'conv4_1': 0.2,
                 'conv5_1': 0.2}

        content_weight = 1  # alpha
        style_weight = 1e9  # beta

        show_every = 400

        optimizer = optim.Adam([target], lr=0.003)
        steps = 2000  # decide how many iterations to update your image (5000)
```

```
        for ii in range(1, steps+1):

            target_features = self.get_features(target, self.vgg)

            content_loss = torch.mean((target_features['conv4_2'] -
content_features['conv4_2'])**2)

            style_loss = 0

            for layer in style_weights:
                target_feature = target_features[layer]
                target_gram = self.gram_matrix(target_feature)
                _, d, h, w = target_feature.shape

                style_gram = style_grams[layer]

                layer_style_loss = style_weights[layer] * torch.mean((target_gram -
style_gram)**2)

                style_loss += layer_style_loss / (d * h * w)

            total_loss = content_weight * content_loss + style_weight * style_loss
            optimizer.zero_grad()
            total_loss.backward()
            optimizer.step()

        # display intermediate images and print the loss
            """ if  ii % show_every == 0:
                print('Total loss: ', total_loss.item())
                plt.imshow(self.im_convert(target))
                plt.show()
```

```
        """
    mpimg.imsave("Content.png", self.im_convert(content))
    mpimg.imsave("style.png", self.im_convert(target))

m = Master ()
m.put_to("data/input_image/RKS.jpg","data/target_style/acrylic.jpg")
```

Inference Code Using TensorFlow Library

```
import os
import tensorflow as tf

import matplotlib.image as mpimg
import numpy as np
import PIL.Image

import tensorflow_hub as hub
# Load compressed models from tensorflow_hub
os.environ['TFHUB_MODEL_LOAD_FORMAT'] = 'COMPRESSED'

class Master:

  def __init__(self) -> None:
    self.hub_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-
stylization-v1-256/2')



  def tensor_to_image(self,tensor):
   tensor = tensor*255
   tensor = np.array(tensor, dtype=np.uint8)
   if np.ndim(tensor)>3:
```

```
            assert tensor.shape[0] == 1
            tensor = tensor[0]
        PIL.Image.fromarray(tensor).save('style.jpg')

    def load_img(self,path_to_img):
        max_dim = 512
        img = tf.io.read_file(path_to_img)
        img = tf.image.decode_image(img, channels=3)
        img = tf.image.convert_image_dtype(img, tf.float32)

        shape = tf.cast(tf.shape(img)[:-1], tf.float32)
        long_dim = max(shape)
        scale = max_dim / long_dim

        new_shape = tf.cast(shape * scale, tf.int32)

        img = tf.image.resize(img, new_shape)
        img = img[tf.newaxis, :]
        return img


    def put_to(self,main_addr,style_addr):

        content = self.load_img(main_addr)
        style = self.load_img(style_addr)
        stylized_image = self.hub_model(tf.constant(content), tf.constant(style))[0]
        self.tensor_to_image(stylized_image)

#m = Master()
#m.put_to("data/input_image/RKS.jpg","data/target_style/dotted_style.jpg")
```

5.1. FRONT PAGE SCREENSHOT

Art Generation using Style Transfer is facilitated through a user-friendly PyWebIo interface developed within the 'app ()' function. This interface enables users to select both content and style images seamlessly. Upon selection, the chosen images are displayed, allowing users to visualize the input images for style transfer. The system then processes the selected images, generating a stylized output by applying the chosen style to the content image. Once the transformation is complete, the resulting stylized image is presented to the user, providing a convenient and interactive platform for exploring different artistic styles through neural style transfer.

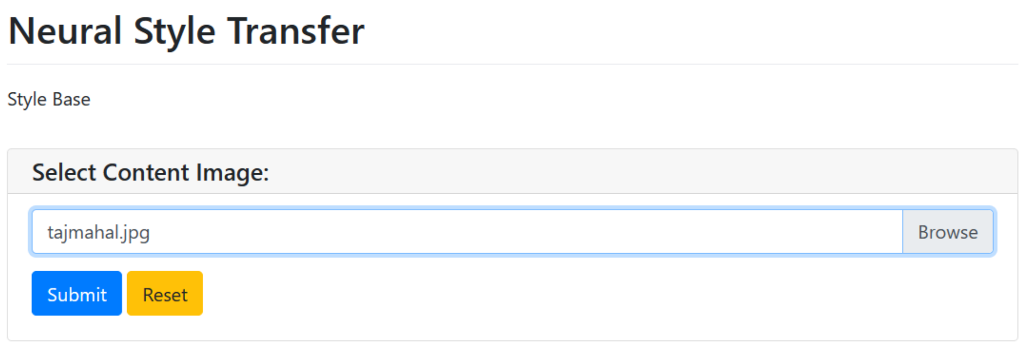


Figure 9.Content Image Selection Page

The Content Image Selection page, as depicted in Figure 9, allows users to upload and choose their desired content image for neural style transfer. Through a user-friendly PyWebIo interface, users can easily navigate and select the content image of their choice, facilitating seamless integration into the art generation process.

Figure 10 illustrates the Style Image Selection page of the PyWebIo interface, allowing users to upload and choose style images seamlessly. Users can visually inspect and select the desired style image for the neural style transfer process.

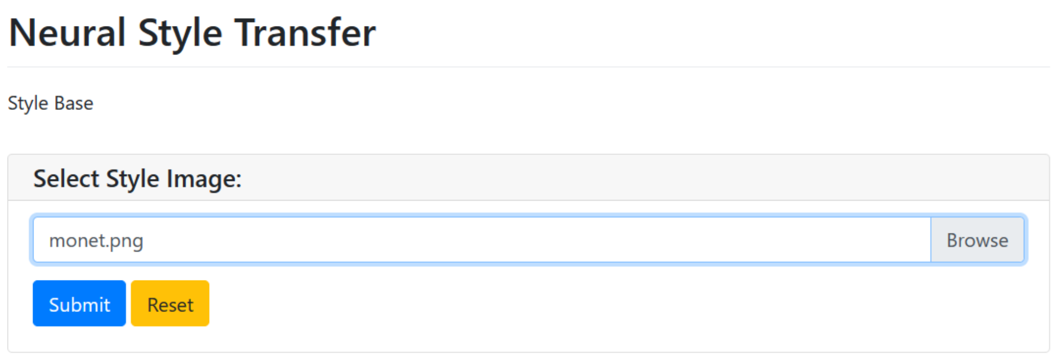


Figure 10.style image selection page.

Figure 11 showcases the final stylized image resulting from the neural style transfer process, visualized on the PyWebIo page. Users can observe the transformed image, where the chosen artistic style has been applied to the original content, providing a captivating demonstration of the art generation capabilities enabled by the interface.

Neural Style Transfer

Style Base

Content Image

Style Image

Style Transfer Image

Figure 11. target image.

5.2. Testing And Validation

Testing an art generation project involving Neural Style Transfer (NST) typically involves verifying various aspects of the system, including functionality, performance, and usability. Here are some example test cases for an art generation project with NST.

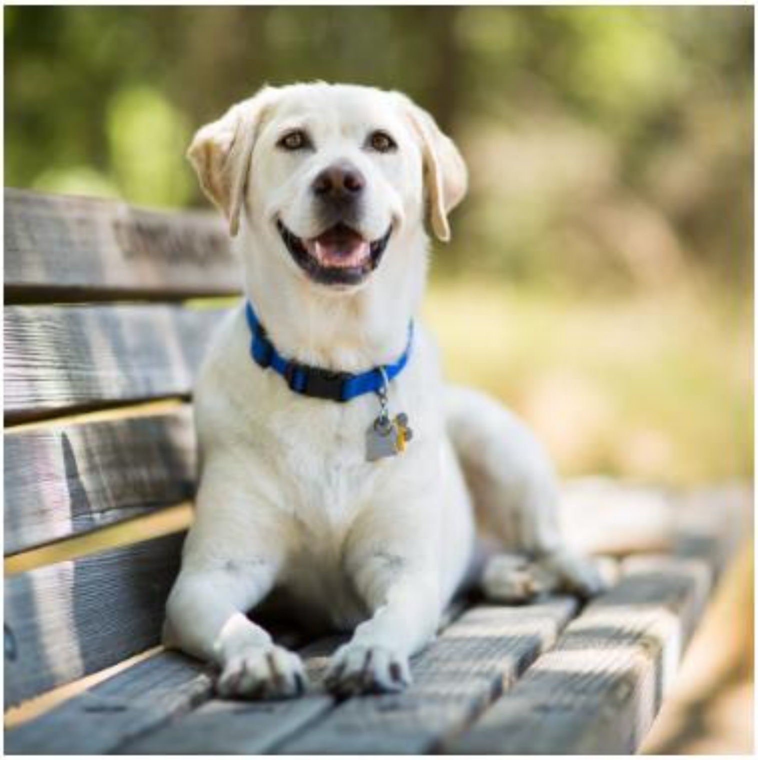
1.Input Validation: Verify that the system properly handles invalid or unsupported input formats for content and style images. Ensure the system provides appropriate error messages for invalid inputs.
2.Style Transfer Accuracy: Validate that the generated artwork reflects the desired style characteristics from the style image. Compare the output artwork against expectations and benchmarks for style transfer quality.
3.Performance Testing: Measure the time taken by the system to perform style transfer on different sizes and complexities of content and style images. Test the system's scalability by evaluating its performance with large datasets or concurrent user requests.
4.Robustness and Error Handling: Verify that the system gracefully handles unexpected errors or exceptions during style transfer. Assess the system's fault tolerance and resilience to adverse conditions.
5.User Interface Testing: Evaluate the usability and intuitiveness of the user interface for uploading content and style images, adjusting parameters, and viewing results. Verify that the user interface provides clear feedback and guidance to users throughout the art generation process.
6.Compatibility Testing: Test the system's compatibility with different operating systems, web browsers, and hardware configurations.
7.Security Testing: Assess the system's security measures for protecting user data, such as uploaded images and personal information.
8.Integration Testing: Verify the integration between different system components, such as the NST model, user interface, and backend services.
9.Regression Testing: Re-run previously executed test cases to ensure that recent changes or updates have not introduced new defects or regressions.
10.User Acceptance Testing (UAT): Solicit feedback from end-users or stakeholders to assess the overall satisfaction with the system and its ability to meet user requirements.

These test cases provide a comprehensive framework for validating the functionality, performance, and quality of an art generation project with Neural Style Transfer.

Testing and validation are crucial stages in ensuring the efficacy and reliability of the art generation project using Neural Style Transfer (NST). Through rigorous testing methodologies, including unit testing, integration testing, and end-to-end validation, the functionality and performance of the NST model are thoroughly assessed. Unit tests are conducted to verify the correctness of individual components, such as data preprocessing, model training, and style transfer algorithms. Integration testing ensures seamless interaction between different modules and components of the system, identifying any potential inconsistencies or errors in the process. End-to-end validation involves testing the entire system workflow, from input image selection to style transfer output, to validate the overall functionality and user experience. Additionally, validation techniques such as qualitative assessment by experts and quantitative evaluation using predefined metrics are employed to measure the quality and fidelity of the generated artworks. Through comprehensive testing and validation processes, the art generation project with NST can confidently deliver high-quality, visually appealing results while meeting user expectations and requirements.

5.3. Results

Style Base



Content Image

Style Image

Figure 12.dog content image with Krishna style.

In Figure 12, the results depict a compelling fusion of a dog content image with a styled image inspired by the iconic artwork of Lord Krishna. Leveraging the PyWebIo user interface selection, users witness the harmonious integration of the chosen content and style, yielding an artistic rendition that seamlessly blends the canine subject with the divine aesthetic of Krishna, exemplifying the diverse creative possibilities facilitated by neural style transfer techniques.

Target Style Image

Style Transfer Image

Figure 13. target image of dog with lord Krishna style

Figure 13 showcases the transformed image of a dog, where its style has been transferred into the artistic representation of Lord Krishna's style. This visualization is realized through a PyWebIo page, offering an interactive platform for exploring diverse style transformations in real-time.

6.Conclusion

6.1 Conclusion

In conclusion, the art generation project employing Neural Style Transfer (NST) has demonstrated remarkable potential in transforming ordinary images into captivating artworks infused with diverse styles. Through the integration of PyWebIo interface, users can effortlessly engage in the creative process, exploring various style transformations and witnessing the seamless amalgamation of content and style. Leveraging the principles of NST and advancements in deep learning, this project not only facilitates artistic expression but also underscores the power of technology in democratizing the realm of visual creativity. As the project continues to evolve, it holds promise for fostering innovation in digital artistry and expanding the horizons of artistic exploration for enthusiasts and professionals alike.

The success of the project highlights several key points:

Innovative Application of NST

User-Centric Design

Artistic Exploration

Educational Value

AI-Driven Artistic Suggestions

6.2. Future Scope

Looking ahead, there are several avenues for future development and enhancement of the art generation project with NST:

1.Advanced Style Transfer Techniques: Explore and implement advanced NST algorithms, such as adaptive style transfer or multi-style transfer, to further enhance the diversity and quality of generated artworks.

2.Real-Time Processing: Optimize the NST model and algorithms to support real-time style transfer, enabling users to see immediate results as they interact with the application.

3.Customization and Personalization: Introduce features for users to customize and personalize their art generation experience, such as fine-tuning style parameters, combining multiple styles, or incorporating user-generated content.

4.Collaborative Art Creation: Enable collaborative art creation by allowing multiple users to contribute to the same artwork simultaneously, fostering collaboration and creativity among users.

5.Integration with Blockchain: Explore the integration of blockchain technology to provide provenance and authentication for generated artworks, creating a decentralized marketplace for buying, selling, and trading digital art.

6.AI Assistance and Recommendations: Implement AI-driven assistance and recommendation systems to provide users with personalized suggestions for styles, compositions, and artistic techniques based on their preferences and past interactions.

In summary, the project on art generation with NST has laid the foundation for a dynamic and engaging platform for digital art creation. With ongoing innovation and refinement, the project has the potential to evolve into a powerful tool for artistic expression, exploration, and collaboration in the digital age.

References

1. Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2414-2423).

2. Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In European Conference on Computer Vision (pp. 694-711). Springer, Cham.

3. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., & Yang, M. H. (2017). Universal style transfer via feature transforms. In Advances in Neural Information Processing Systems (pp. 386-396).

4. Chen, Y., Wang, M., Kuo, C. C. J., & Liao, H. Y. M. (2020). Universal style transfer via feature space translation. IEEE Transactions on Multimedia, 22(1), 120-133.

5. Dumoulin, V., Shlens, J., & Kudlur, M. (2017). A learned representation for artistic style. In International Conference on Learning Representations.

6. Ruder, M., Dosovitskiy, A., & Brox, T. (2016). Artistic style transfer for videos. In German Conference on Pattern Recognition (pp. 26-36). Springer, Cham.

7. Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022.