A Major Project Report On

Artistic Image and Video Style Transformation using Deep Neural Networks

Submitted in fulfillment of the requirements for the award of the

Bachelor of Technology

In

Department of Computer Science and Engineering

By

Student Names	Roll Numbers
Sunkari Sriya	20241A05T2
M. Sri Sathvika	20241A05R3
L. Aishwarya	20241A05R0
Kavali Sruthi	20241A05Q6

Under the Esteemed guidance of

Dr. V. Sri Lakshmi

Associate Professor

Department of Computer Science and Engineering

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

Bachelupalli, Kukatpally, Hyderabad, Telangana, India,500090

2023-2024

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

CERTIFICATE

This is to certify that the major project entitled "**Artistic Image and Video Style Transformation using Deep Neural Networks**" is submitted by **S. Sriya (20241A05T2), M. Sri Sathvika (20241A05R3), L. Aishwarya (20241A05R0), K. Sruthi(20241A05Q6)** in fulfillment of the award of a degree in BACHELOR OF TECHNOLOGY in Computer Science and Engineering during the academic year **2023-2024.**

INTERNAL GUIDE	HEAD OF THE DEPARTMENT
Dr. V. Sri Lakshmi	**Dr. B. SANKARA BABU**
Associate Professor	**Professor**

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Many people helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all. First, we wish to express our deep gratitude to our internal guide **Dr. V. Sri Lakshmi**, **Associate Professor**, Department of CSE for his support in the completion of our project report. We wish to express our honest and sincere thanks to **Dr. B. Sankara Babu, HOD,** Department of CSE, and to our principal **Dr. J. Praveen** for providing the facilities to complete our major project. We would like to thank all our faculty and friends for their help and constructive criticism during the project completion phase. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

Sunkari Sriya	**20241A05T2**
M. Sri Sathvika	**20241A05R3**
L.Aishwarya	**20241A05R0**
K. Sruthi	**20241A05Q6**

DECLARATION

We hereby declare that the major project entitled "**Artistic Image and Video Style Transformation using Deep Neural Networks**" is the work done during the period from **2023-2024** and is submitted in the fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering from **Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad).** The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

Sunkari Sriya	**20241A05T2**
M. Sri Sathvika	**20241A05R3**
L.Aishwarya	**20241A05R0**
K. Sruthi	**20241A05Q6**

	LIST OF FIGURES	
Fig No	**Fig Title**	**Page No**

	LIST OF TABLES	
Table No	**Table Name**	**Page No**

ABSTRACT

Neural Style Transfer (NST) has emerged as a powerful and innovative technique in the realm of computer vision and artistic expression. This project explores the fusion of neural networks and artistic aesthetics to transform ordinary images and videos into captivating visual experiences. NST leverages deep convolutional neural networks, such as VGG19, and a combination of content and style loss functions to separate and recombine content from one source image with the artistic style from another. By doing so, it creates novel and visually striking compositions that merge the content of one image or video frame with the style of a famous artwork or a unique aesthetic, thereby transcending the boundaries of traditional image processing and artistic creation.

In this project, we implement NST for both images and videos, demonstrating its ability to enrich the visual storytelling capabilities in various domains, from filmmaking to content creation. The integration of deep learning and artistic expression opens new avenues for creative professionals and enthusiasts, allowing them to imbue their work with unique and captivating styles. We delve into the technical intricacies of the NST process, highlighting the role of convolutional neural networks in extracting content and style features, and showcase how this method transforms the mundane into the extraordinary. The results of this project reveal the potential of neural networks in solving the problem of artistic style transfer and offer a glimpse into the exciting possibilities this technology holds for the future of visual content creation.

Keywords: Neural Style Transfer (NST), deep convolutional neural networks, VGG19(Visual Geometry Group), artistic aesthetics.

1. Introduction

Style transfer is an advanced technique in artificial intelligence that involves blending the visual style of one image (the style image) with the content of another image (the content image). This process leverages deep neural networks, specifically convolutional neural networks (CNNs), to extract features from both the content and style images and create a new image that combines the artistic style of one with the subject and structure of the other. By optimizing a loss function that measures the similarity of the generated image to the content and style images, the technique produces a unique visual output.

Through this project, there are numerous applications across various fields. In the realm of artistic creation, the project allows for the transformation of photographs using the style of renowned paintings, leading to the production of unique and visually stunning works of art. In video production, the project applies style transfer to each frame to generate striking effects in movies, music videos, and other media forms. Additionally, this project enhances virtual and augmented reality experiences by altering the visual style of environments, making them more immersive and engaging.

This project also addresses certain challenges associated with style transfer, such as ensuring consistent style across video frames and achieving high performance, especially for high-resolution images and videos. Another key focus is the mitigation of unwanted artifacts that may appear in generated images, which requires careful tuning of the algorithm. By working on these challenges, the project aims to push the boundaries of style transfer research and development, exploring innovative techniques such as neural style transfer based on CNNs and real-time style transfer using lightweight networks.

1.1 Problem Definition including the significance and objective

The project aims to address the challenge of transforming images or videos by blending the visual style of one image (the style image) with the content of another image (the content image) in a seamless and coherent manner. The primary objective is to develop a method that can effectively combine the artistic elements of the style image with the detailed subject and structure of the content image, producing a new image or video that retains the context of the content image while adopting the aesthetic style of the style image. This process requires maintaining the integrity of both the content and the style during the transformation, ensuring that the final output is visually striking and adheres to the original vision of the content image. Achieving this delicate balance poses several technical challenges, including preserving fine details, maintaining consistency across video frames, and avoiding artifacts in the generated images or videos. The project seeks to develop a robust solution that addresses these issues and delivers high-quality style transfer results.

This project has the potential to revolutionize various industries by offering the powerful tool for visual transformation and artistic creation. Style transfer enables unique and captivating combinations of different visual elements, which can be ap- plied in artistic and creative fields, media and entertainment, virtual and augmented reality, and design and marketing. It enhances visual storytelling, improves im- mersive experiences, and contributes to the advancement of AI and deep learning research. By refining neural networks and optimization techniques, the project opens up new possibilities for innovation and creative expression across diverse domains.

1.2 Methodologies

To perform style transfer, the VGG19 model—a pre-trained convolutional neural network (CNN) with strong feature extraction capabilities—was employed. The model extracted features from both the content and style images at various layers of the network. These layers captured different aspects of the images, such as textures, colors, patterns, and shapes. Lower layers focused on capturing fine details like edges and textures, while higher layers emphasized more complex features such as patterns and overall style.

After extracting the features, two types of loss functions were calculated to measure the similarity between the content and style images with the generated image. Content loss measured how closely the structure and subject of the generated image matched the content image, while style loss evaluated how well the visual style of the generated image aligned with the style image. Balancing these two losses ensured that the final image maintained the integrity of the content image while adopting the artistic style of the style image.

Optimization of the generated image involved adjusting its pixels to minimize the combined loss function, which included both the content and style losses. An iterative optimization process gradually refined the generated image, bringing it closer to the desired balance of content and style. By fine-tuning the VGG19 model and the optimization process, high-quality, visually appealing images and videos were produced that effectively blended content and style.

1.3 Outline of the results

The use of a user interface (UI) streamlined the style transfer process by allowing users to easily upload input images or videos for transformation. The UI provided a user-friendly platform where users could select content and style images or videos, customize the degree of style transfer, and preview the results in real-time. This seamless interaction with the model contributed to an efficient workflow and an enhanced user experience. The model, based on the VGG19 architecture, produced high-quality outputs that successfully blended the visual style of the style image with the content of the content image. The results showcased the model's ability to maintain the integrity of the content image while incorporating artistic elements from the style image. Users observed visually striking and coherent outputs, including smooth transitions and well-preserved details. In addition to generating appealing still images, the model demonstrated strong performance in video style transfer. Despite the challenges of ensuring consistency across frames, the model effectively maintained coherence and style continuity throughout the videos. Overall, the results of the model were impressive, highlighting its potential for artistic and practical applications across various domains.

1.4 Scope of the project

The scope of the style transfer project encompasses the development and optimization of a deep learning model based on the VGG19 architecture for blending the content of one image with the visual style of another. The project addresses both still images and videos, allowing for the transformation of various types of visual media while maintaining consistency and coherence across video frames. It integrates a user-friendly interface that enables easy uploading of content and style images or videos, customization of style transfer parameters, and real-time previews of the results. Users can personalize the process according to their preferences, adjusting the degree of style influence and other relevant settings. Ongoing research and development efforts focus on refining neural networks and optimization techniques to improve quality and efficiency. The project's scope extends to a variety of fields, including art and design, media and entertainment, virtual and augmented reality, and marketing, offering innovative visual transformation tools for creative expression and practical applications.

2. System Requirements

2.1 Software Requirements

- Pytorch : It provides tensor computation (similar to NumPy) with strong GPU acceleration and deep neural networks built on a tape-based autograd system.
- numpy : NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
- scipy : SciPy is an open-source library for mathematics, science, and engineering. It builds on top of NumPy and provides additional functionality for optimization, integration, interpolation, linear algebra, statistics, and more.
- clip: CLIP (Contrastive Language-Image Pretraining) is a deep learning model developed by OpenAI that understands natural language in the context of images.
- scikit-image: scikit-image is a collection of algorithms for image processing in Python. It provides a simple and efficient way to manipulate and analyze images, including tasks such as filtering, segmentation, feature extraction, and transformation.

2.2 Hardware Requirements

- Minimum 12GB RAM : RAM (Random Access Memory) is a type of computer memory that is used to temporarily store data that the CPU needs to access quickly.
- Intel Core i5 or higher: Intel Core i5 is a line of mid-range processors manufactured by Intel. Processors in this series typically offer a balance of performance and power efficiency suitable for a wide range of computing tasks
- T4 GPU having 2560 CUDA cores: The T4 GPU is a graphics processing unit developed by NVIDIA, primarily designed for data center and cloud computing applications.

2.3 Data Set

We used a pre-trained VGG19 model, the dataset used for training the VGG19 network is the ImageNet dataset, renowned for its vast scale and diversity. Comprising over 14 million labeled images distributed across 20,000 categories, ImageNet covers a wide spectrum of objects, scenes, and patterns. Its purpose primarily revolves around training deep learning models like VGG19 for tasks such as object recognition, classification, and localization. Each image in ImageNet is meticulously annotated with labels corresponding to its contents, facilitating the learning process for models like VGG19 to recognize and differentiate between various objects and visual cues. During training, a subset of ImageNet serves as the training set, aiding VGG19 in learning meaningful features and patterns. Another subset, the validation set, helps fine-tune the model's parameters and prevent overfitting, while a separate testing set evaluates the model's generalization and accuracy on unseen data. Overall, ImageNet's comprehensive nature and extensive usage make it a foundational dataset for training and evaluating image recognition models like VGG19.

3.Literature Survey

Through this literature review, we aim to gain insights into the current landscape of neural style transfer for images and videos. By analyzing methodologies, appli- cations, and existing systems, we can identify trends, challenges, and opportunities for further research and development in this rapidly evolving field.related works.

3.1 Introduction to the problem domain terminology

Computer vision using deep learning is a rapidly advancing field that enables machines to interpret and understand visual information. Deep learning algorithms, particularly Convolutional Neural Networks (CNNs), have revolutionized computer vision by enabling machines to perform tasks such as image classification, object detection, segmentation, and image generation with unprecedented accuracy.

3.2 Existing solutions

Several researchers have studied the individual metrics generated in this system. Some of the prominent research papers are provided below

3.2.1 Neural Algorithm of Artistic Style

The paper[1] introduced an innovative approach to style transfer by formulating it as an optimization problem within the framework of deep neural networks. Gatys and his collaborators proposed a method that separated the content and style of images, allowing for the generation of visually striking and artistically stylized outputs.

The key contribution of the paper lies in its methodology, which leverages deep neural networks to represent the content and style of images separately. This separation enables the extraction of high-level features that capture the essence of both the content and the style. By defining a loss function that combines content and style losses, the approach effectively blends the content of one image with the artistic style of another.

One of the notable aspects of Gatys et al.'s approach is its ability to generate outputs that not only mimic the style of the reference image but also preserve the essential structure and content of the content image. This ensures that the generated outputs are visually appealing while retaining the semantic meaning of the original content.

3.2.2 Perceptual Losses for Real-Time Style Transfer and Super-Resolution

”Perceptual Losses for Real-Time Style Transfer and Super-Resolution” [2]is a notable paper by Johnson et al., published in 2016, which introduced an innovative approach to neural style transfer and super-resolution tasks. Unlike traditional meth- ods that relied on computationally expensive optimization processes, Johnson and his collaborators proposed a feed-forward neural network architecture that enabled real-time style transfer and super-resolution.

The key contribution of the paper lies in its use of perceptual losses derived from pre-trained deep neural networks. Instead of directly optimizing pixel values, the approach utilizes high-level feature representations to measure the difference between generated and target images. This perceptual loss function captures perceptual similarity, allowing for the generation of visually appealing outputs that closely resemble the desired style or resolution.

The paper demonstrates the effectiveness of the approach in achieving real-time style transfer and super-resolution across a range of images and artistic styles. By training the neural network on a dataset of paired input and target images, the model learns to generate outputs that preserve the structure and content of the input while incorporating the desired style or resolution enhancements.

The Figure 2.1 illustrates the System overview.The system trains an image trans- formation network to transform input images into output images. It uses a loss network pretrained for image classification to define perceptual loss functions that measure perceptual differences in content and style between images. The loss network remains fixed during the training process.

3.2.3 Instance Normalization : The Missing Ingredient for Fast Stylization

”Instance Normalization: The Missing Ingredient for Fast Stylization” (Ulyanov et al., 2016)[3] focused on optimizing the efficiency of neural style transfer.The key contribution of the paper lies in its exploration of instance normalization as a means of stabilizing the stylization process and reducing computational overhead. Unlike batch normalization, which operates on entire batches of data, instance normalization normalizes the activations of each individual sample independently. This allows for faster convergence during training and more efficient processing during inference, leading to significant improvements in speed without sacrificing quality.

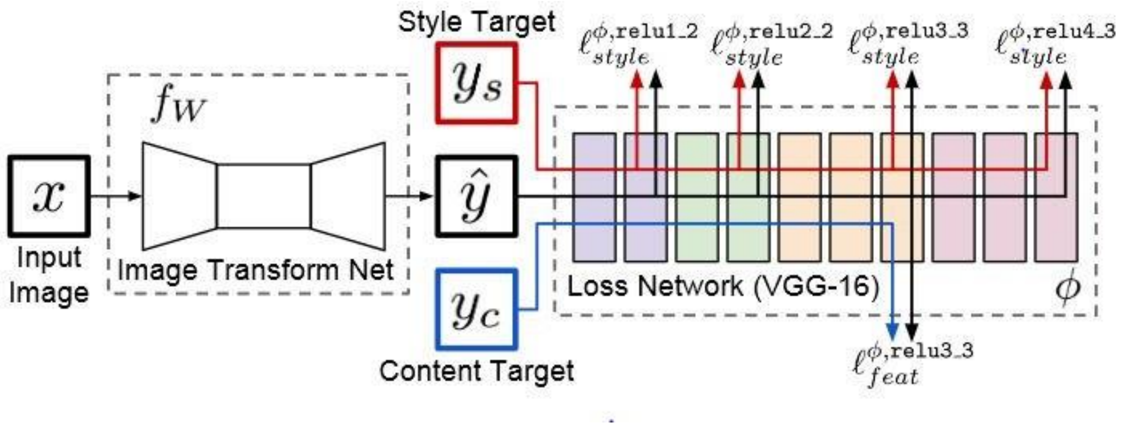

Style Target
y_s
$\ell_{style}^{\phi,relu1\_2}$
$\ell_{style}^{\phi,relu2\_2}$
$\ell_{style}^{\phi,relu3\_3}$
$\ell_{style}^{\phi,relu4\_3}$
f_W
x
Input Image
Image Transform Net
$\hat{y}$
y_c
Content Target
Loss Network (VGG-16)
ϕ
$\ell_{feat}^{\phi,relu3\_3}$

Figure 3.1: System overview.

The paper demonstrates the effectiveness of instance normalization in various style transfer applications, including image stylization and style transfer in videos. By incorporating instance normalization into the stylization network architecture, the authors achieve faster processing speeds while maintaining high-quality stylized outputs. Additionally, the paper explores the impact of different normalization techniques on the stylization process and provides insights into the underlying mechanisms of instance normalization.

3.2.4 Image and Video Style Transfer based on Transformers

”Image and Video Style Transfer based on Transformers” (Sun Yanguo and Lan Zhenping, 2018)[4] introduced the STLTSF framework, a significant departure from traditional style transfer methods. This paper leveraged transformer architectures, commonly associated with natural language processing, to tackle image style transfer. The STLTSF framework utilized two transformer-based encoders to capture domain-specific information and a transformer decoder to progressively generate output sequences of image patches. This novel application of transformers showcased their adaptability beyond their initial linguistic context.

The Figure 2.2 depicts the Structure of the Transformer network. The model consists of three parts: the content Transformer encoder, the style Transformer encoder, and the Transformer decoder. The content Transformer encoder encodes the images’ long-range information in the content domain. The style Transformer encoder, on the other hand,encodes information in the style domain. The Transformer decoder is used to style the content features, transforming them into stylized visuals.

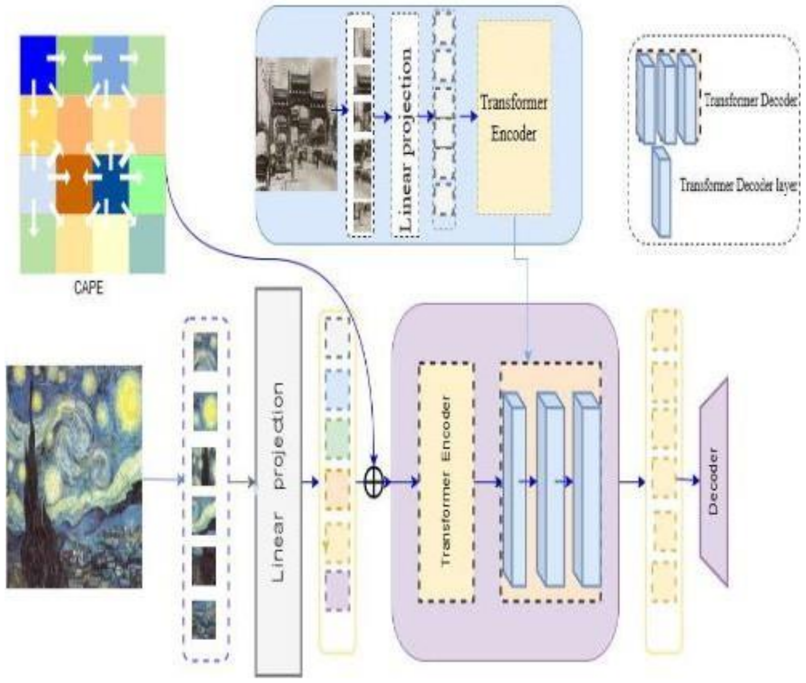


Figure 3.2 : Network Structure.

3.2.5 Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization

”Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization” (Huang and Belongie, 2017)[5] further advanced the field by introducing adaptive instance normalization. This technique dynamically adjusted normalization param- eters,allowing the model to accommodate a broad spectrum of artistic styles in real-time.By providing users with the capability to apply arbitrary styles seamlessly, this paper significantly enhanced the versatility and user-friendliness of neural style transfer applications

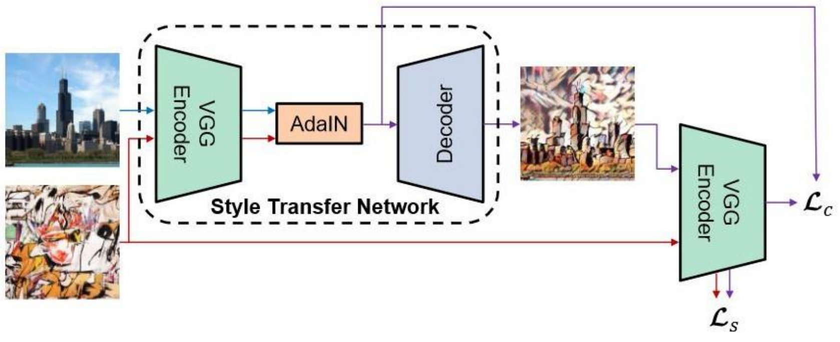


Figure 3.3 : An overview of Style Transfer algorithm

Fig. 3.3 shows an overview of their style transfer network based on the proposed AdaIN layer. AdaIN is an extension of the traditional instance normalization (IN) technique commonly used in deep learning architectures, particularly in style transfer tasks. While IN normalizes the activations of each individual sample independently, AdaIN dynamically adjusts the normalization parameters based on the style input.

3.3 Related works

Gatys et al.[6] in his paper ”Preserving Color in Neural Artistic Style Transfer”addresses the issue of color distortion in neural style transfer and proposes techniques to preserve the color of the content image while transferring the style.The authors propose a novel approach to address this issue by separating the style and color information of the input images. They leverage the features extracted by a pre-trained convolutional neural network (CNN) to capture the style of the style image and the content of the content image. However, instead of directly applying the style features to the content image, they propose to transfer only the texture and structure of the style image while preserving the color information of the content image.

Ruder et al.[7] in his paper ”Coherent Online Video Style Transfer” introduces a method for achieving coherent style transfer in online videos, addressing challenges such as temporal consistency and efficient processing.The key contribution of the paper lies in the formulation of coherence constraints, which penalize deviations in style and content between adjacent frames. By incorporating these constraints into the optimization process, the method encourages the generated frames to maintain consistency with both the preceding and succeeding frames, resulting in visually coherent stylized videos.

Ruder et al.[8] in his paper ”Deep Video Style Transfer” extends neural style transfer techniques to video processing, enabling the transfer of artistic styles to videos with temporal coherence and smooth transitions.To achieve temporal coher- ence, the authors propose incorporating coherence constraints into the optimization process. These constraints penalize deviations in style and content between consec- utive frames, encouraging the model to produce visually coherent sequences with consistent artistic style throughout.

Johnson et al.[9] in ”Fast Photo Style Transfer Using Convolutional Neural Networks” introduces a fast style transfer method based on convolutional neural networks, allowing for real-time application of artistic styles to images with high-quality results.The key innovation of the paper lies in the introduction of feed-forward neu- ral networks trained to perform style transfer in a single forward pass. Unlike traditional optimization-based methods, which require iterative processing, the proposed approach applies a pre-trained CNN to directly generate stylized images from content images and style references. This feed-forward architecture enables real-time or near-real-time style transfer, making it suitable for a wide range of applications where speed is critical.

Zhu et al.[10] in his paper ”Learning to See Creatively: Designing Artistic Visual Patterns for Neural Style Transfer” presents a pioneering approach to enhance the creative potential of Neural Style Transfer (NST). The paper addresses the need for more nuanced control over the artistic output generated by NST algorithms. By introducing the concept of artistic visual patterns, the authors aim to enable users to design and customize stylized images according to specific artistic preferences and creative goals.Central to the methodology proposed in the paper is the idea of learning artistic patterns from a diverse set of reference images. Unlike traditional NST approaches that rely solely on predefined artistic styles, Zhu et al. advocate for a more flexible and adaptive framework. By leveraging a dataset of artistic visual patterns, the proposed model learns to capture the intricate details and nuances characteristic of various artistic styles, allowing for more expressive and personalized stylization.

4. Propose Approach , Modules Description, and UML Diagrams

4.1 Modules

The proposed integrated system for Neural Style Transfer for Images and Videos encompasses several key modules. Module 1 Pretrained VGG-19 network module is used to extract the features from the content image or video and also Style image provided. Module 2 is used for Style and Content loss calculation. Module 3 Detection utilizes LBFGS optimization algorithm to make generate the output fastly and accurately. Module 4 is used for designing a UI using streamlit.

Module 1: Pretrained model for Feature representation

We chose VGG19 as the feature extraction module in our Neural Style Trans- fer (NST) implementation due to its robustness and effectiveness in capturing both low-level and high-level image representations. VGG19 is a well-established convo- luminal neural network architecture originally designed for image classification tasks, renowned for its ability to discern intricate patterns and features within images.

By leveraging VGG19 as our feature extractor, we tap its hierarchical structure which spans 19 layers and includes convolutional, max-pooling, and fully connected layers. This architecture allows us to extract increasingly abstract features from the input images, capturing both fine-grained details and high-level semantic information.

Through the utilization of VGG19, we aim to efficiently extract main features from both the content and style images. These features serve as the basis for computing the content loss, which measures the divergence in content features between the generated and content images, and the style loss, which quantifies the deviation in style features between the generated and style images. By iteratively optimizing these losses, our NST algorithm strives to produce a stylized image that seamlessly combines the content of the content image with the style of the style image, resulting in visually compelling outputs.

Module 2: Content loss and Style loss generation

In our NST implementation, we utilize the feature maps generated by specific layers of VGG19 to compute the style and content losses. The content loss is calculated by comparing the feature maps of the generated image and the content image at a chosen convolutional layer. This loss measures the discrepancy in content representation between the two images, encouraging the generated image to mimic the content of the content image.

Similarly, the style loss is computed by comparing the Gram matrices of the feature maps at multiple convolutional layers between the generated image and the style image. The Gram matrix captures the correlations between different feature maps, representing the style information of the image. By minimizing the difference in Gram matrices across multiple layers, the style loss ensures that the generated image inherits the style characteristics of the style image.

The decision to use the Gram matrix is rooted in its ability to capture the style of an image by representing the relationships between features at different spatial locations within the image. Each element of the Gram matrix corresponds to the dot product of the feature vectors of different channels in a given layer of the CNN. This dot product measures the similarity or correlation between the features, indicating how frequently they occur together across the spatial dimensions of the image.

Module 3: Optimization

We chose the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm as the optimization method for our Neural Style Transfer (NST) implementation due to its efficiency and effectiveness in minimizing the loss function associated with style transfer.

L-BFGS is well-suited for optimizing the complex objective function involved in NST, which comprises both content and style losses. This algorithm iteratively adjusts the pixel values of the generated image to minimize the total loss, resulting in a stylized image that combines the content of the content image with the style of the style image.

One of the key advantages of L-BFGS is its memory efficiency. Unlike gradient descent-based optimization methods, L-BFGS maintains a limited-memory approximation of the inverse Hessian matrix, making it suitable for optimizing objective functions with a large number of parameters, such as the pixel values of high- resolution images in NST.

Additionally, L-BFGS is known for its fast convergence rate, especially for smooth and convex optimization problems. It typically requires fewer iterations to reach convergence compared to first-order optimization methods like gradient descent, which is beneficial for efficiency in NST.

Module 4: Streamlit for User Interface

For our user interface, we opted for Streamlit, a powerful tool for creating interactive web applications with Python. Users are presented with two options: to style either an image or a video. Upon selection, users can upload their desired style image and content image or video using the designated upload buttons. Once the user uploads the images or video and clicks the submit button, the style transfer process begins. For image styling, the styled image is displayed directly on the screen once the process is complete. However, for video styling, users are provided with a button to download the styled video after the process finishes.

The user interface is designed to be intuitive and straightforward, allowing users to easily navigate and interact with the application. Streamlit's user-friendly interface facilitates a seamless experience, enabling users to upload their content and view the styled results with just a few clicks.

4.2 Algorithms

The VGG network will be initially trained for object recognition and localization, is employed as the basis for image generation. The network comprises 19 layers, including 16 convolutional and 5 pooling layers, and serves as a foundation for the subsequent analysis.The feature space utilized is derived from a normalized version of the 16 convolutional and 5 pooling layers of the 19-layer VGG network. The normalization involves scaling the weights to ensure that the mean activation of each convolutional filter across images and positions is equal to one. This re-scaling is feasible for the VGG network due to its reliance on rectifying linear activation functions and the absence of normalization or pooling over feature maps. The fully connected layers are excluded from the analysis.The standard maximum pooling operation is replaced by average pooling for enhanced visual appeal during image synthesis.

4.2.1 Neural Style Transfer for Images

The detail procedure for NST for Images is as follows and the same is represented visually in the figure (*Figure 3.1, Proposed Work Flow for Neural Style Transfer for Images*).

1. **Input Images: The** project starts with input images, consisting of a content image and a style image. These images serve as the basis for creating stylized images.

2. **Preprocessing Images:** This preprocessing ensures that the images are in the appropriate tensor format, and they possess uniform dimensions, enabling efficient calculation of the loss function during the subsequent training phase.

3. **Building the model:** We build the model over a VGG19 network. It is used to Extract features both the content and style images. The standardized version of the 19-layer VGG network includes 16 convolutional and five pooling layers. By scaling weights, the network was normalized such that the mean activation of each convolutional filter over images and positions was equivalent to one.

4. **Content Representation:** Each layer in the Convolutional Neural Network (CNN) defines a non-linear filter bank, with complexity increasing with the layer's position in the network. Given an input image x, each layer encodes the image through filter responses, resulting in N 1 feature maps of size Ml. To visualize the image information encoded at different layers, gradient descent is performed on a white noise image to find another image matching the feature responses of the original image. The content reconstructions are achieved by minimizing the squared-error loss between the feature representations of the original and generated images.

5. **Style Representation:** To obtain a representation of the style of an input image, a feature space designed to capture texture information is utilized. This feature space is built on top of filter responses in any layer of the network and consists of correlations

between different filter responses, yielding the Gram matrix G_l:

$$G_{l_{ij}} = \sum F_{l_{ik}} F_{l_{jk}}$$

Visualizing the style representation involves constructing an image matching the style representation of a given input image. This is achieved by using gradient descent from a white noise image to minimize the mean-squared distance between the entries of the Gram matrices from the original and generated images.

6. Style Transfer: The style transfer process involves synthesizing a new image that simultaneously matches the content representation of a given photograph and the style representation of an artwork. The joint minimization of content and style representations is achieved through a total loss function. The total loss includes both content and style reconstructions, each weighted by factors α and β, respectively:

$$L_{\text{total}}(\tilde{p}, \tilde{a}, \tilde{x}) = \alpha L_{\text{content}}(\tilde{p}, \tilde{x}) + \beta L_{\text{style}}(\tilde{a}, \tilde{x})$$

The resulting synthesis does not incorporate image priors, but texture features from lower layers in the network may serve as a specific image prior for the style image. The absence of regularization with image priors distinguishes this approach from previous methods.

4.2.2 Neural Style Transfer for Videos

Similar to images, videos consist of several frames which can be perceived as different images and NST can be applied to each frame separately this process is visually represented in the (*Figure 3.2, Proposed Work Flow for Neural Style Transfer for Videos*).

1. Input Video: Similar to images, videos consist of a sequence of frames. Each frame serves as the basis for creating stylized video frames.

2. Efficient Frame Handling: TensorFlow and OpenCV are used to efficiently process and manipulate video frames. These libraries help ensure proper synchronization between video frames and stylization.

3. L-BFGS Algorithm for Video: Similar to image style transfer, the L-BFGS optimization algorithm is used to minimize the total loss for each frame in the video sequence. This process is applied iteratively for each frame to generate the stylized video.

4.3 UML Diagrams

4.3.1 Class Diagram

A Class Diagram represents the static structure of a system, focusing on classes, their attributes, methods, relationships, and associations. It shows the blueprint of the system's classes and their relationships. As Figure 3.4 shows class diagram for our proposed work.

This class diagram as shown in the figure 3.4 meticulously details the design of a web application that empowers users to upload images or videos and unleash their creativity through neural style transfer. Users commence their artistic journey by selecting an upload option class, either Image Option or Video Option, depending on their desired medium. Each upload option class fosters the inclusion of a style image through the includes association, allowing users to define the artistic essence they wish to infuse into their content. Once the upload option and style image are chosen, the user's selection triggers the Neural Transfer class via the trigger's association. This class serves as the artistic engine, wielding the power of neural networks to craft a captivating new image. The performTransfer() method within this class takes center stage, meticulously combining the user's uploaded content (image or video) with the artistic flair of the style image. The result of this artistic alchemy is embodied in the ResultImage class. This class proudly stores the newly generated image, a testament to the user's creative vision. Finally, the user can choose to download this masterpiece using the download() method provided by the ResultImage class, allowing them to share their artistic creation with the world. In essence, this class diagram serves as a blueprint, meticulously outlining the user journey and the intricate interplay between the application's components, fostering an engaging and artistically empowering user experience.

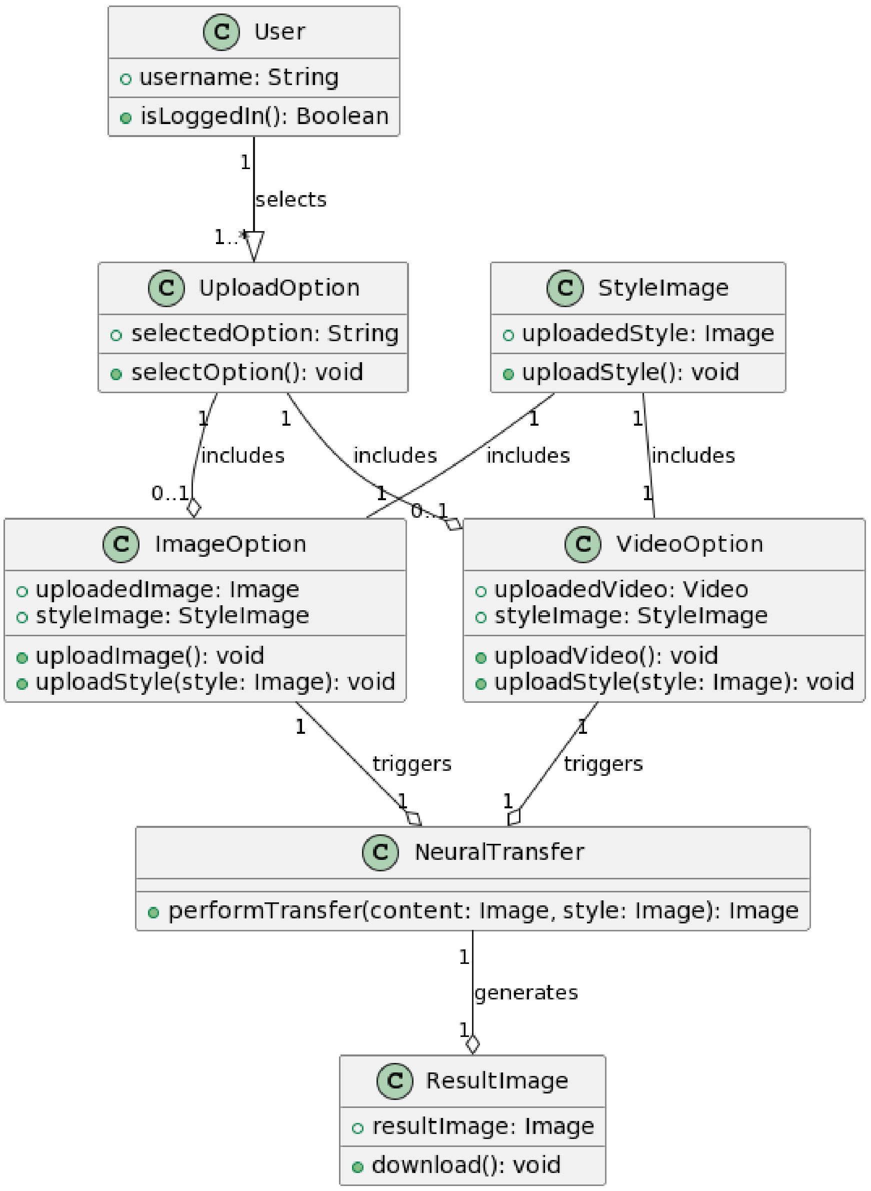


Figure 4.3.1: Class Diagram for Neural Style Transfer for Images and Video

4.3.2 Sequence Diagram

This detailed sequence diagram in Figure 3.5 outlines the steps involved in uploading media and creating stylized images on a website. It depicts the interaction between the user and the system in chronological order. The diagram starts with the user initiating the process by selecting an upload option. The system then presents choices for uploading either an image (zdjecie in Polish) or a video (film in Polish). Depending on the user's selection, the system calls specific functions. Uploading an image triggers the uploadimage() function, followed by a confirmation message ”Image uploaded.” For stylized creations, the user can upload a separate style image using the uploadStyle(style) function. A confirmation message “StyleImage uploaded” appears upon successful upload. The system then merges the content of the uploaded image with the artistic style of the chosen style image through a function

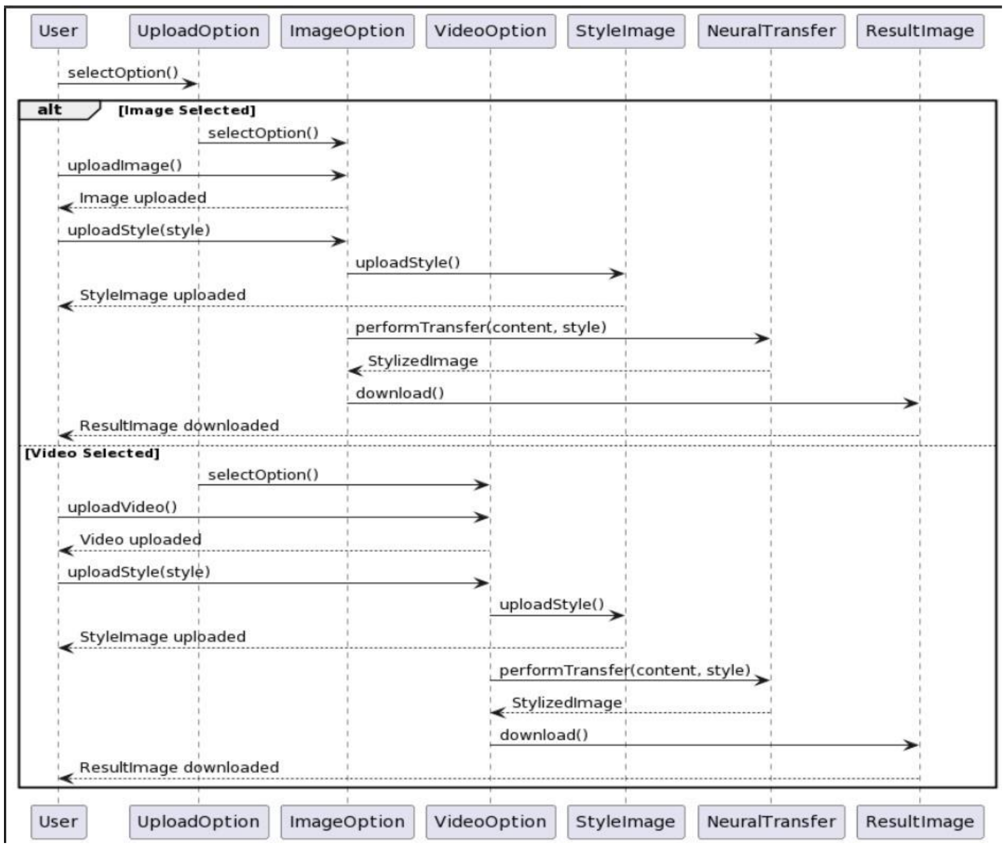


Figure 4.3.2: Sequence Diagram for Neural Style Transfer for Images and Video

4.3.3 Use Case Diagram

A Use Case Diagram as shown in 3.6 illustrates the interactions between actors (users or systems) and a system to achieve specific goals. It shows the functionality of the system from the user's perspective, focusing on what users can do and how the system responds.

The use case diagram illustrates an image and video style transfer application, focusing on user interactions and system functionalities. Users start by logging in to access the system's features, then upload their photos or videos to apply artistic styles. The system provides an interface for style selection and preview before initiating the transformation process.

Behind the scenes, the system facilitates content and style upload, employing Deep learning algorithms for style transfer. It calculates content and style losses to ensure the final styled output reflects both the original content's essence and the chosen artistic style. The system iteratively refines the styled output to balance content preservation and artistic influence.

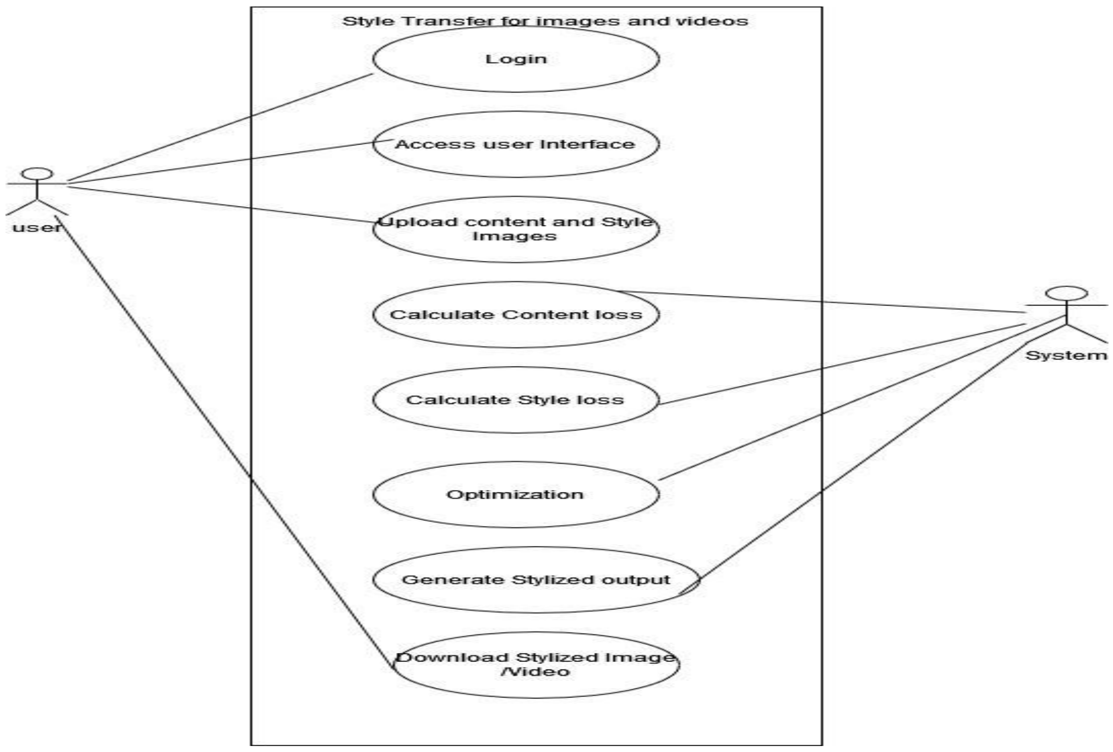


Figure 4.3.3 : Use Case diagram for Neural Style Transfer for Images and Video

5. Implementation & Results

5.1. Neural style transfer for Images code

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

from keras.utils import load_img
from keras.utils import img_to_array
import numpy as np
from numpy import uint8
from keras.applications import vgg19
from keras import backend as K

from scipy.optimize import fmin_l_bfgs_b
from keras.utils import save_img
import time

from matplotlib import pyplot as plt
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import numpy as np
import cv2
import PIL
import streamlit as st
from PIL import Image

from skimage.metrics import structural_similarity as ssim
from skimage.metrics import peak_signal_noise_ratio as
    psnr

def perceptual_loss(content_features, stylized_features):
    # Compute the mean squared error between content and
        stylized features
    return tf.reduce_mean(tf.square(content_features - st
        ylized_features))

def style_loss(style_features, stylized_features):
    # Compute the mean squared error between style and
        stylized features
    return tf.reduce_mean(tf.square(style_features - st
        ylized_features))

def compute_ssim(image1, image2):
    # Compute Structural Similarity Index (SSIM)
    return ssim(image1, image2, multichannel=True)
def style_transfer_image(style_image_path,ta
rget_image_path):

    import torch
    import torch.nn as nn
    import torch.nn.functional as F
    import torch.optim as optim

    from PIL import Image
    import matplotlib.pyplot as plt

    import torchvision.transforms as transforms
    from torchvision.models import vgg19, VGG19_Weights

    import copy

    device = torch.device("cuda" if torch.cuda.
        is_available() else "cpu")
    torch.set_default_device(device)
    print(device)

    imsize = 512 if torch.cuda.is_available() else 128

    loader = transforms.Compose([transforms.Resize(imsize
        ), transforms.ToTensor()])


    def image_loader(image_name):
        image = Image.open(image_name)
        image_resized = image.resize((512, 512))
        image1 = loader(image_resized).unsqueeze(0)
        return image1.to(device, torch.float)


    style_img = image_loader(style_image_path)
    content_img = image_loader(target_image_path)

    assert (
        style_img.size() == content_img.size()
    ), "You have to to import style and content images of
        the same size"

    unloader = transforms.ToPILImage() # reconvert into
        PIL image

    plt.ion()
```

```


def imshow(tensor, title=None):
    image = tensor.cpu().clone()  # we clone the
        tensor to not do changes on it
    image = image.squeeze(0)  # remove the fake batch
         dimension
    image = unloader(image)
    plt.imshow(image)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are
         updated


plt.figure()
imshow(style_img, title="Style Image")

plt.figure()
imshow(content_img, title="Content Image")


class ContentLoss(nn.Module):

    def __init__(
        self,
        target,
    ):
        super(ContentLoss, self).__init__()
        # we 'detach' the target contentfrom the
            tree used
        # to dynamically compute the gradient: this
            is a stated value,
        # not a variable. Otherwise the forward
            method of the criterion
        # will throw an error.
        self.target = target.detach()

    def forward(self, input):
        self.loss = F.mse_loss(input, self.target)
        return input


def gram_matrix(input):
    a, b, c, d = input.size()  # a=batch size(=1)
    # b=number of feature maps
    # (c,d)=dimensions of a f. map (N=c*d)

    features = input.view(a * b, c * d)  # resize F_XL into \hat F_XL

    G = torch.mm(features, features.t())  # compute the gram product

    # we 'normalize' the values of the gram matrix
    # by dividing by the number of element in each feature maps.
    return G.div(a * b * c * d)


class StyleLoss(nn.Module):

    def __init__(self, target_feature):
        super(StyleLoss, self).__init__()
        self.target = gram_matrix(target_feature).detach()

    def forward(self, input):
        G = gram_matrix(input)
        self.loss = F.mse_loss(G, self.target)
        return input


cnn = vgg19(weights=VGG19_Weights.DEFAULT).features.eval()

cnn_normalization_mean = torch.tensor([0.485, 0.456, 0.406])
cnn_normalization_std = torch.tensor([0.229, 0.224, 0.225])


# create a module to normalize input image so we can easily put it in a
# ``nn.Sequential``
class Normalization(nn.Module):
    def __init__(self, mean, std):
        super(Normalization, self).__init__()
        # .view the mean and std to make them [C x 1 x 1] so that they can
        # directly work with image Tensor of shape [B x C x H x W].
        # B is batch size. C is number of channels. H is height and W is width.
        self.mean = torch.tensor(mean).view(-1, 1, 1)
        self.std = torch.tensor(std).view(-1, 1, 1)

    def forward(self, img):
        # normalize ``img``
        return (img - self.mean) / self.std


# desired depth layers to compute style/content losses :
content_layers_default = ["conv_4"]
style_layers_default = ["conv_1", "conv_2", "conv_3", "conv_4", "conv_5"]


def get_style_model_and_losses(
    cnn,
    normalization_mean,
    normalization_std,
    style_img,
    content_img,
    content_layers=content_layers_default,
    style_layers=style_layers_default,
):
    # normalization module
    normalization = Normalization(normalization_mean, normalization_std)

    # just in order to have an iterable access to or list of content/style
    # losses
    content_losses = []
    style_losses = []

    # assuming that ``cnn`` is a ``nn.Sequential``, so we make a new ``nn.Sequential``
    # to put in modules that are supposed to be activated sequentially
    model = nn.Sequential(normalization)

    i = 0  # increment every time we see a conv
    for layer in cnn.children():
        if isinstance(layer, nn.Conv2d):
            i += 1
            name = "conv_{}".format(i)
        elif isinstance(layer, nn.ReLU):
            name = "relu_{}".format(i)
            # The in-place version doesn't play very nicely with the ``ContentLoss``
            # and ``StyleLoss`` we insert below. So
```

```
                we replace  with out-of-place
                # ones here.
                layer = nn.ReLU(inplace=False)
            elif isinstance(layer, nn.MaxPool2d):
                name = "pool_{}".format(i)
            elif isinstance(layer, nn.BatchNorm2d):
                name = "bn_{}".format(i)
            else:
                raise RuntimeError(
                    "Unrecognized layer: {}".format(layer
                        .__class__.__name__)
                )

            model.add_module(name, layer)

            if name in content_layers:
                # add content loss:
                target = model(content_img).detach()
                content_loss = ContentLoss(target)
                model.add_module("content_loss_{}".format
                    (i), content_loss)
                content_losses.append(content_loss)

            if name in style_layers:
                # add style loss:
                target_feature = model(style_img).detach()
                style_loss = StyleLoss(target_feature)
                model.add_module("style_loss_{}".format(i
                    ), style_loss)
                style_losses.append(style_loss)

        # now we trim off the layers after the last
            content and style losses
        for i in range(len(model) - 1, -1, -1):
            if isinstance(model[i], ContentLoss) or
                isinstance(model[i], StyleLoss):
                break

        model = model[:(i + 1)]

        return model, style_losses, content_losses


    input_img = content_img.clone()
    # if you want to use white noise by using the
        following code:
    #
```

```
# .. code-block:: python
#
#     input_img = torch.randn(content_img.data.size())

# add the original input image to the figure:
plt.figure()
imshow(input_img, title="Input Image")


def get_input_optimizer(input_img):
    # this line to show that input is a parameter
        that requires a gradient
    optimizer = optim.LBFGS([input_img])
    return optimizer


def run_style_transfer(
    cnn,
    normalization_mean,
    normalization_std,
    content_img,
    style_img,
    input_img,
    num_steps=300,
    style_weight=1000000,
    content_weight=1,
):

    print("Building the style transfer model..")
    model, style_losses, content_losses =
        get_style_model_and_losses(
        cnn, normalization_mean, normalization_std,s
            tyle_img, content_img
    )

    # We want to optimize the input and not the model
         parameters so we
    # update all the requires_grad fields accordingly
    input_img.requires_grad_(True)
    # We also put the model in evaluation mode, so
        that specific layers
    # such as dropout or batch normalization layers
        behave correctly.
    model.eval()
    model.requires_grad_(False)

    optimizer = get_input_optimizer(input_img)

    print("Optimizing..")
    run = [0]
    while run[0] <= num_steps:

        def closure():
            # correct the values of updated input
               image
            with torch.no_grad():
                input_img.clamp_(0, 1)

            optimizer.zero_grad()
            model(input_img)
            style_score = 0
            content_score = 0

            for sl in style_losses:
                style_score += sl.loss
            for cl in content_losses:
                content_score += cl.loss

            style_score *= style_weight
            content_score *= content_weight

            loss = style_score + content_score
            loss.backward()

            run[0] += 1
            if run[0] % 50 == 0:
                print("run {}:".format(run))
                print(
                    "Style Loss : {:4f} Content Loss:
                        {:4f}".format(
                        style_score.item(),
                           content_score.item()
                    )
                )
                print()

            return style_score + content_score

        optimizer.step(closure)

    # a last correction...
    with torch.no_grad():
        input_img.clamp_(0, 1)

    return input_img


    output = run_style_transfer(
        cnn,
        cnn_normalization_mean,
        cnn_normalization_std,
        content_img,
        style_img,
        input_img,
    )


    output_img = unloader(output.squeeze(0).cpu())   #
       Convert tensor to PIL image
    output_img.save("output_image.jpg")
    image = Image.open(target_image_path)
    width, height = image.size

    image_path = '/content/output_image.jpg'   # Update
       this with the actual path to your image file

    # Open the image using PIL
    image = Image.open(image_path)
    resized_image = image.resize((width, height))

    # Save the resized image back to the disk
    output_path ='/content/output_image1.jpg'    # Update
       this with the desired output path
    resized_image.save(output_path)


# Save the      image to a file
```

5.2. Neural Style Transfer for Videos Code

```
def style_transfer_video(content_video,   style_image):

    #read image, convert to tensor, normalize and resize
    def image_read(image):
        max_dim = 512
        image = tf.convert_to_tensor(image,   dtype=tf.
           float32)
        image = image / 255.0
        shape = tf.shape(image)
        shape = shape[:-1]   # Exclude the last dimension
           (channels)
        long_dim = tf.cast(tf.reduce_max(shape),   tf.
           float32)
    scale = max_dim / long_dim
    new_shape = tf.cast(shape * scale, tf.int32)
    new_image = tf.image.resize(image, new_shape)
    new_image = new_image[tf.newaxis, :]

    return new_image

#convert tensor to numpy array
def tensor_toimage(tensor):
    tensor =tensor*255
    tensor = np.array(tensor, dtype=np.uint8)
    if np.ndim(tensor)>3:
        assert tensor.shape[0]==1
        tensor=tensor[0]

    return tensor

    # Read style image
style_img = Image.open(style_image)
style_img = np.array(style_img)  # Convert PIL image to NumPy array

# Convert style image to tensor
style_im = tf.convert_to_tensor(style_img, dtype=tf.float32)
style_im = style_im / 255.0

# Save content video to disk
video_path = ”content_video”
with open(video_path, ”wb”) as f:
    f.write(content_video.read())



cap = cv2.VideoCapture(video_path)
#in order to get the size of width and shape of video, we used first frame of video
ret, frame = cap.read()
frame_width = image_read(frame)[0].shape[1]
frame_height= image_read(frame)[0].shape[0]

out = cv2.VideoWriter(’output.mp4’, cv2.VideoWriter_fourcc(*’XVID’), 10,
                          (frame_width,frame_height))



while True:
        ret, frame = cap.read()
        if ret == True:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB
                )
            frame = image_read(frame)
            stylized_frame = style_transfer_images(tf.
                constant(frame), tf.constant(style_im))[0]
            image = tensor_toimage(stylized_frame)
            out.write(image)
        else:
            break

    cap.release()
    out.release()
return    "output.mp4"
```

5.3. Streamlit code

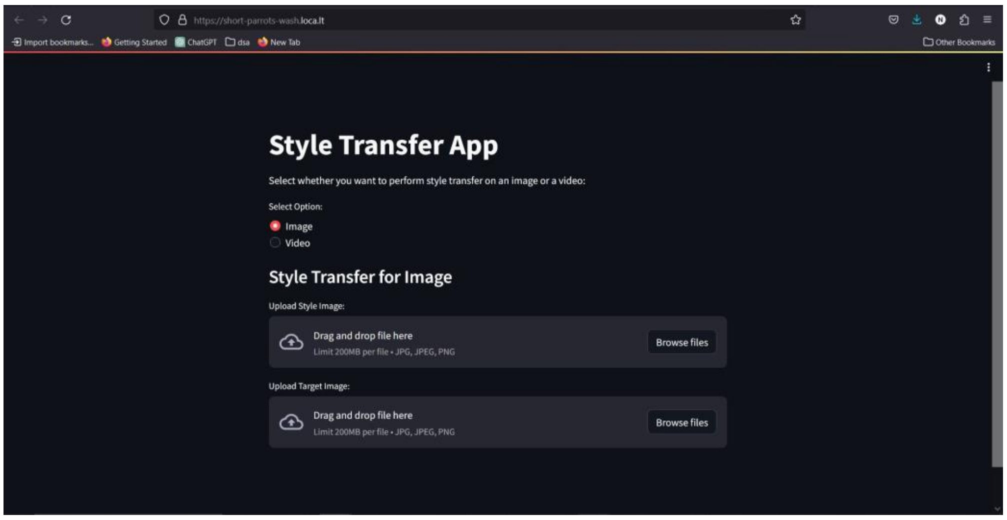
```

def main():
    st.title("Style Transfer App")
    st.write("Select whether you want to perform style
        transfer on an image or a video:")

    # Selection for image or video
    option = st.radio("Select Option:", ("Image", "Video
        "))

    if option == "Image":
        st.subheader("Style Transfer for Image")

        # Upload style image
        style_image = st.file_uploader("Upload Style
            Image:", type=["jpg", "jpeg", "png"])

        # Upload target image
        target_image = st.file_uploader("Upload Target
            Image:", type=["jpg", "jpeg", "png"])

        # Perform style transfer when both images are
            uploaded and style transfer button is clicked
        if style_image is not None and target_image is
            not None:
            style_img = Image.open(style_image)
            target_img = Image.open(target_image)

            # Display style image
            st.image(style_img, caption="Style Image", use_column_width=True)

            # Display target image
            st.image(target_img, caption="Target Image",use_column_width=True)

            # Perform style transfer when button is clicked
            if st.button("Perform Style Transfer"):
                style_transfer_image(style_image,target_image)
                new_size=(256,256)
                # Display styled image
                styled_img=Image.open("output_image1.jpg")
                st.image("output_image1.jpg", caption="Styled Image", use_column_width=True)
                img1_resized = cv2.resize(img_to_array(styled_img), new_size)
                img2_resized = cv2.resize(img_to_array(target_img), new_size)
                st.write('ssim:',compute_ssim(img1_resized,img2_resized))

    elif option == "Video":
        st.subheader("Style Transfer for Video")

        # Upload style image
        style_image = st.file_uploader("Upload Style Image:", type=["jpg", "jpeg", "png"])

        # Upload content video
        content_video = st.file_uploader("Upload Content Video:", type=["mp4"])

        # Perform style transfer when both image and video are uploaded and style transfer button is clicked
        if style_image is not None and content_video is not None:
            style_img = Image.open(style_image)

            # Display style image
            st.image(style_img, caption="Style Image", use_column_width=True)

            # Perform style transfer when button
            if st.button("Perform Style Transfer"):
                styled_video = style_transfer_video(c
                    ontent_video, style_image)

                # Display styled video
                st.video(styled_video)



if __name__ == "__main__":
main()
```

5.4. Frontend Display Commands

```
!wget -q -O - ipv4.icanhazip.com
!streamlit run app.py & npx localtunnel –port 8501
```

5.5. User Interface screenshots

The below Figures are the screenshots of Neural Style Transfer for Imaages and Videos User Interface. The Figure 5.1 is the homepage our application using Streamlit. It consists of 2 options, whether we want to do a style transfer for Image or Video. After selecting one, we have to upload the input files from the system



Style Transfer App

Select whether you want to perform style transfer on an image or a video:

Select Option:
- [✓ Image]
- Video

Style Transfer for Image

Upload Style Image:

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Upload Target Image:

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Figure 5.1: Screenshot of Home page

The Figure 5.2 shows the Perform Style transfer Button after uploading content image and Style and Image

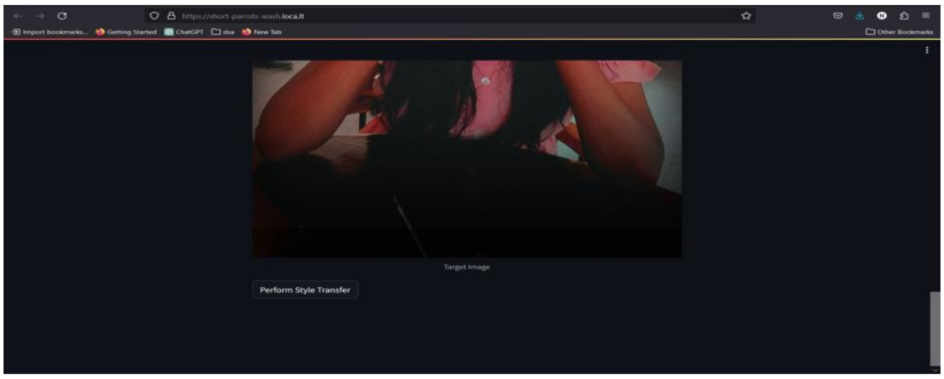


Figure 5.2: Screenshot ”Apply Style Transfer” Button

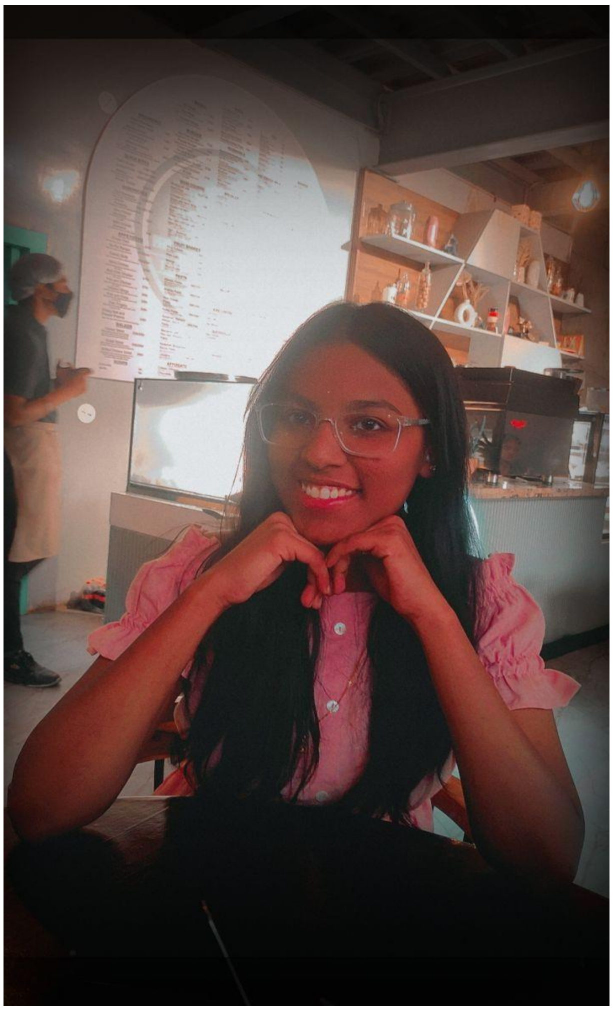


Figure 5.3: Input Content Image

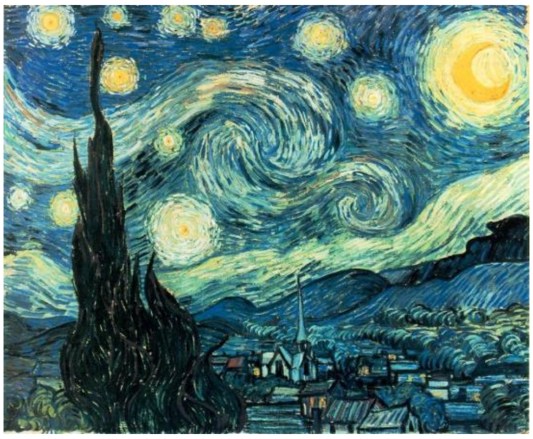


Figure 5.4: Input Style Image

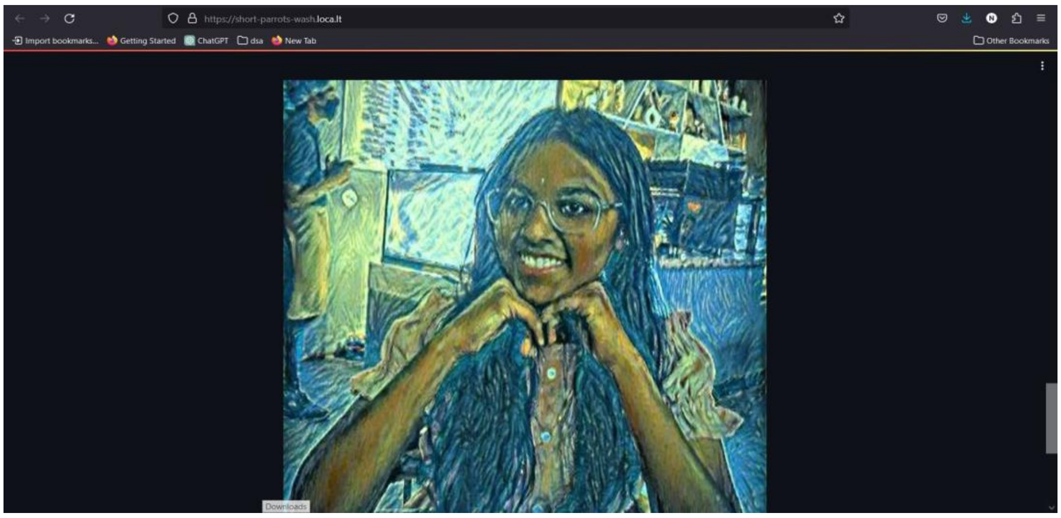


Figure 5.5: Screenshot of Stylized output Image

Similarly for Videos, we have took same style Image Figure 5.4 as style input and took Figure 5.6 as the video input and got Fig 5.7 as Video output

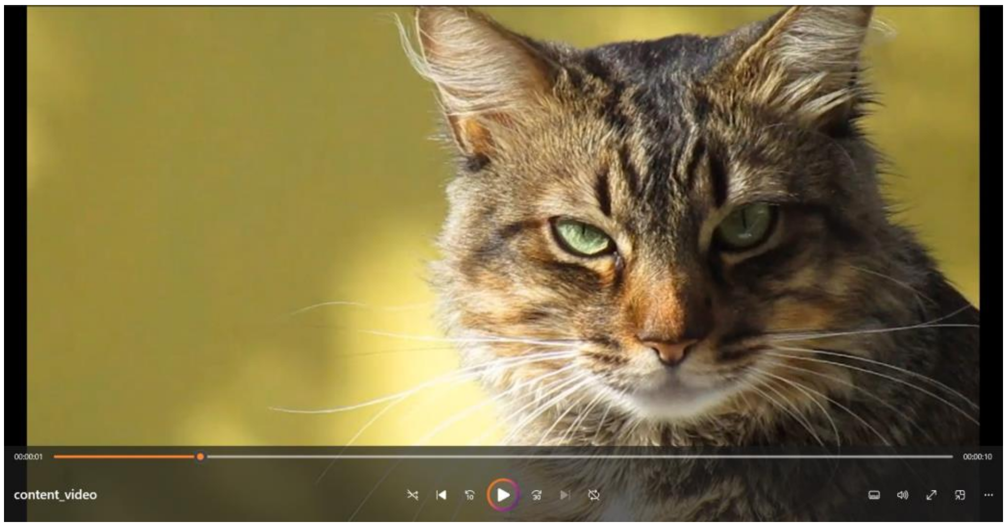


Figure 5.6: Video Input

After generating Video output the user can be able to download the output stylized video using the download button shown in Figure 5.8

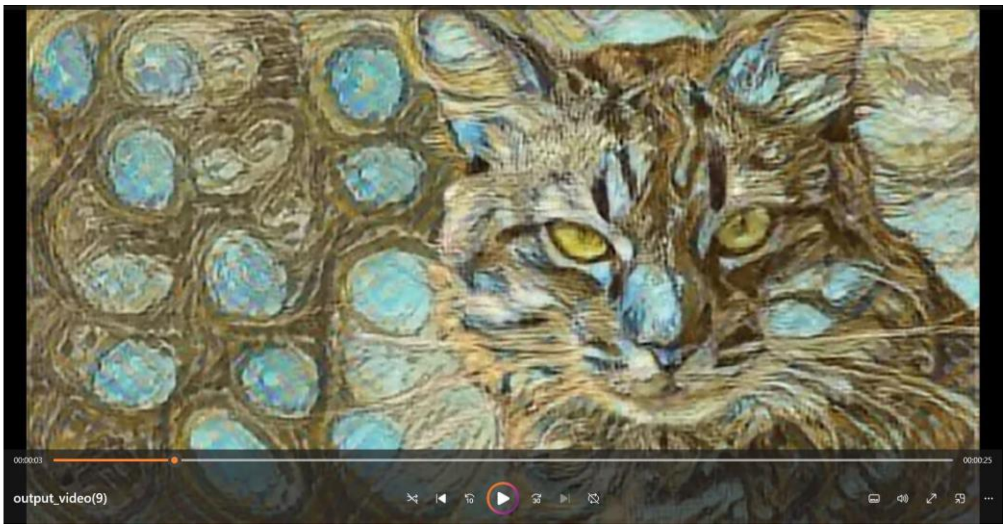
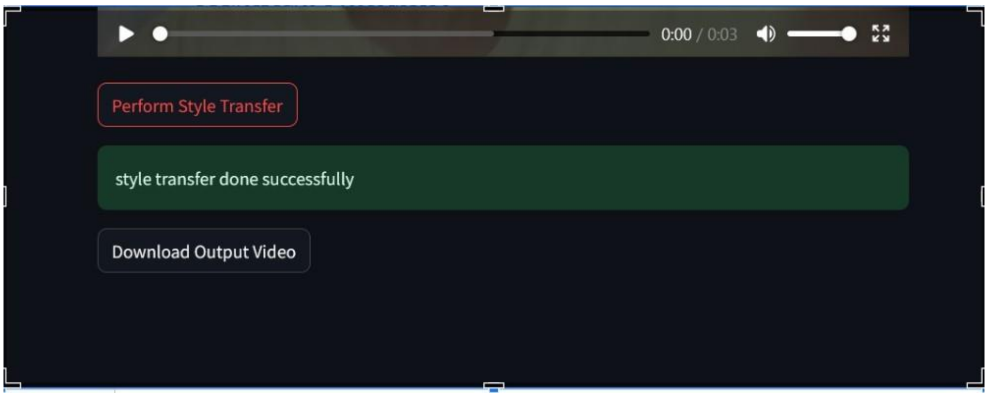


Figure 5.7: Stylized Video Output



style transfer done successfully

Figure 5.8: Screenshot ”Download Output Video” button

5.6. Comparative Analysis

A comparative study was conducted to evaluate the performance of different neural networks, namely ResNet, InceptionV3, and VGG19, in the context of neural style transfer. The study focused on measuring the Structural Similarity Index (SSIM) as a metric to assess the quality of style transfer results. The SSIM scores obtained for each network were as follows:

- ResNet: SSIM = 0.12 - InceptionV3: SSIM = 0.20 - VGG19: SSIM = 0.45

The results indicate significant variations in SSIM scores across the three networks, suggesting differing capabilities in preserving structural similarity and perceptual quality during style transfer.

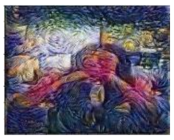
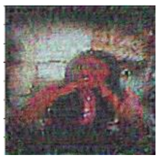
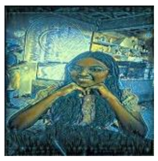
1. **ResNet (SSIM = 0.12): **- ResNet, although known for its performance in image classification tasks, exhibited lower SSIM scores in neural style transfer. This could be attributed to its architecture, which may not capture fine- grained style details as effectively as other networks.

2. **InceptionV3 (SSIM = 0.20): ** - InceptionV3 demonstrated improved SSIM compared to ResNet but still fell short of achieving higher fidelity in style trans-

fer. The network's design, which includes multiple parallel pathways for feature extraction, may contribute to its ability to capture some stylistic nuances.

3. **VGG19 (SSIM = 0.45): ** - VGG19 emerged as the top performer in terms of SSIM scores, indicating its superior capability in preserving both content and style features during transfer. The deep architecture of VGG19, which captures hierarchical features at various scales, likely contributes to its success in style preservation.

Overall, these results suggest that VGG19 outperforms ResNet and InceptionV3 in neural style transfer tasks based on SSIM scores. However, it's essential to consider other factors such as computational efficiency, model complexity, and specific style/content combinations when choosing a neural network for style transfer applications. Further research and experimentation may help in understanding the nuanced performance differences and optimizing neural networks for specific style transfer requirements.

Neural Network	**SSIM Score**	**Result Image**
ResNet	0.12	
InceptionV3	0.20	
VGG19	0.45	

6. Conclusion and Future Scope

6.1. Conclusions

This project represents a significant advancement in the realm of neural style transfer (NST) by extending its application to both images and videos, maintain- ing spatial and temporal coherence throughout the process. The core technique employed in this project involves harnessing the power of Convolutional Neural Networks (CNNs) for feature extraction, a crucial step in achieving high-quality style transfer results.

With the utilization of CNNs, we are able to extract rich and meaningful fea- tures from input images and videos. This feature extraction process allows us to capture the content and style information separately, enabling us to blend them seamlessly in the style transfer process. By dissecting the input content and style into their respective components, we ensure that the final output retains the essence of the original content while adopting the desired artistic style.

One of the key highlights of this project is its ability to maintain spatial coher- ence during image style transfer. Spatial coherence refers to the preservation of structural details and relationships within an image. By leveraging CNNs, we can ensure that the style transfer process does not distort or misalign important elements in the image, thus preserving its overall spatial coherence.

Moreover, this project also focuses on maintaining temporal coherence in video style transfer. Temporal coherence is crucial in video processing as it ensures smooth transitions and consistency between frames. By extending our approach to video style transfer, we ensure that the artistic style remains consistent and visually appealing across consecutive frames, creating a seamless and immersive viewing experience.

Overall, this project represents a sophisticated fusion of cutting-edge technologies, leveraging CNNs for feature extraction to achieve compelling image style transfer and extending this capability to videos while preserving spatial and temporal co- herence. This expansion of NST into the realm of videos opens up new avenues for creative expression and artistic exploration, offering users a powerful tool to transform their visual content in captivating and innovative ways.

6.2. Future Work

6.2.1. **Optimization for Efficiency**: Implementing optimization techniques to enhance computational efficiency, enabling real-time or near-real-time performance, particularly for video style transfer applications.

6.2.2. **Style Variation Enhancement**: Exploring methods to handle a broader range of artistic styles, including abstract, surreal, or highly detailed styles, while maintaining coherence and fidelity in the output.

6.2.3. **User Interaction:** Integrating user-controlled parameters or interactive interfaces to provide users with more fine-grained control over style transfer, allowing them to customize and tweak the output according to their preferences.

6.2.4. **Adversarial Robustness:** Investigating techniques to improve the model's robustness against adversarial attacks or input variations, ensuring consistent and reliable performance across diverse scenarios and datasets.

6.2.5. **Multi-Modal Style Transfer:** Extending the project to support multi-modal style transfer, where users can combine multiple artistic styles or seamlessly transition between different styles within the same image or video sequence.

6.2.6. **Semantic Understanding:** Incorporating semantic understanding and context-aware techniques to enhance the model's ability to preserve important semantic content while applying stylistic transformations, leading to more meaningful and contextually relevant style transfers.

7.References

[1] L. A. Gatys and et al. “A Neural Algorithm of Artistic Style”. In: (2015).

[2] J. Johnson and et al. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: (2016).

[3] D. Ulyanov and et al. “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: (2016).

[4] Yanguo Sun and Lan Zhenping. “Image and Video Style Transfer based on Transformers”. In: (2018).

[5] X. Huang and S. Belongie. “Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization”. In: (2017).

[6] Leon A Gatys, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. “Pre- serving color in neural artistic style transfer”. In: *arXiv preprint arXiv:1606.05897* (2016).

[7] Dongdong Chen, Jing Liao, Lu Yuan, Nenghai Yu, and Gang Hua. “Coher- ent online video style transfer”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1105–1114.

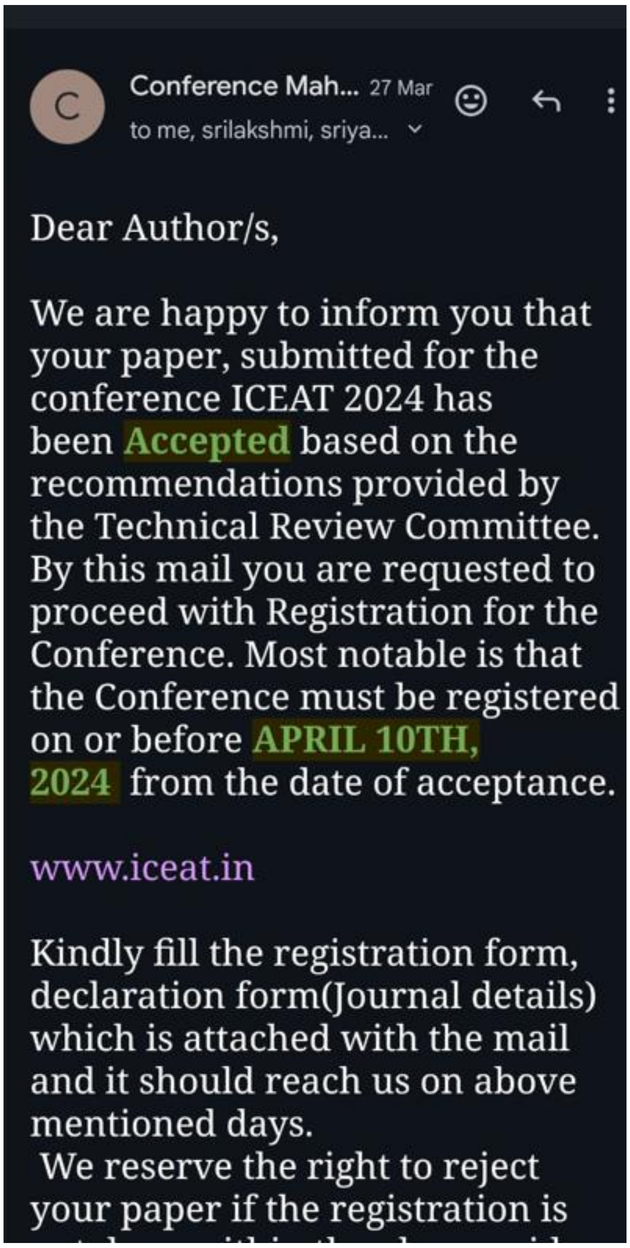
[8] Ni Liu. “Dynamic Style Adaptation Network: A Comprehensive Approach for Video Style Transfer”. In: *2023 International Conference on Image, Algo- rithms and Artificial Intelligence (ICIAAI 2023)*. Atlantis Press. 2023, pp. 776– 787.

[9] Aishwarya Deshmane. “Image Style Transfer using Neural Network”. In: (Dec. 2023).

[10] Xi-Ping Zhu, Wei Chen, Xian-Ming Guo, and De-Yan Liu. “Learning to See Creatively: Designing Artistic Visual Patterns for Neural Style Transfer”. In: *arXiv preprint arXiv:1707.01476* (2017).

APPENDIX

I). Proof of Publication



Conference Mah... 27 Mar
to me, srilakshmi, sriya...

Dear Author/s,

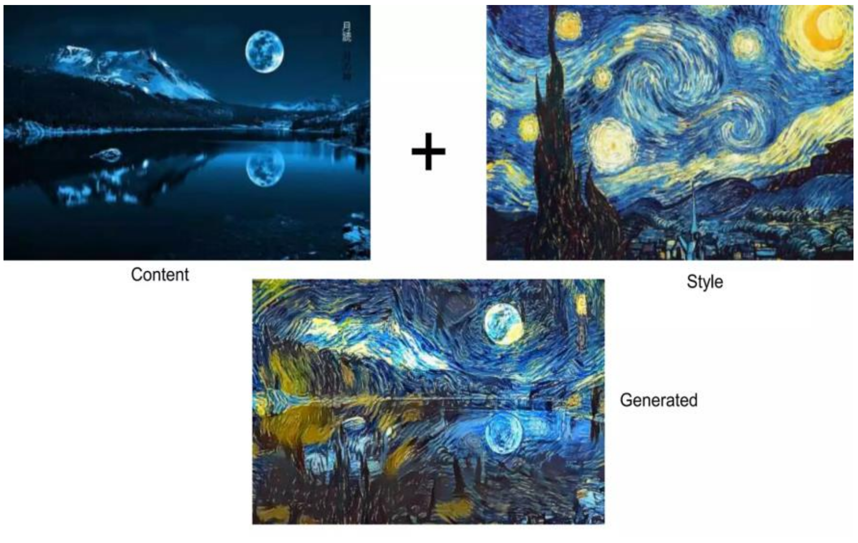
We are happy to inform you that your paper, submitted for the conference ICEAT 2024 has been Accepted based on the recommendations provided by the Technical Review Committee. By this mail you are requested to proceed with Registration for the Conference. Most notable is that the Conference must be registered on or before APRIL 10TH, 2024 from the date of acceptance.

www.iceat.in

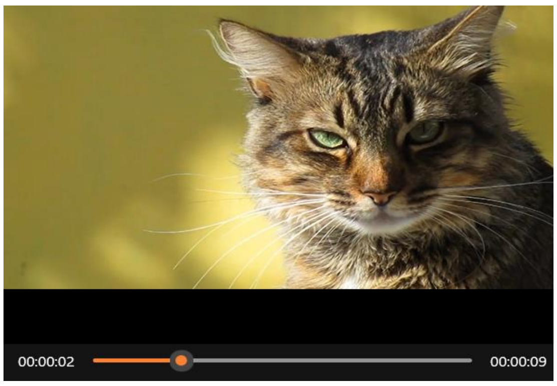
Kindly fill the registration form, declaration form(Journal details) which is attached with the mail and it should reach us on above mentioned days.
We reserve the right to reject your paper if the registration is

II) Snap Shot of Results

Style Transfer of Images

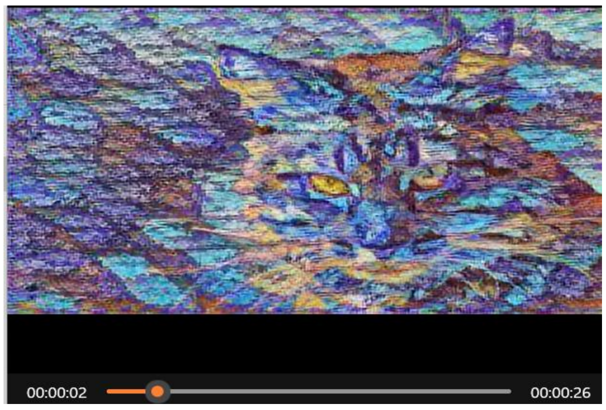


Style Transfer of Videos



Content

style image



output