A Project Report on

NEURAL STYLE TRANSFER

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

Bachelor of Technology

In

Computer Science and Engineering

Submitted by

M.PAVAN
(20H51A05H9)
P.NEETHIKA
(20H51A05J2)

Under the esteemed guidance of
Dr. Siva Skandha Sanagala
(Assosiate Professor & HOD)

Department of Computer Science and Engineering

CMR COLLEGE OF ENGINEERING & TECHNOLOGY
(UGC Autonomous)
*Approved by AICTE *Affiliated to JNTUH *NAAC Accredited with A^{+} Grade
KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

2020- 2024

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Major Project report entitled **"NEURAL STYLE TRANSFER"** being submitted by **M.Pavan (20H51A05H9), P.Neethika (20H51A05J2),** in partial fulfillment for the award of **Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING** is a record of bonafide work carried out under my guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

Dr. Siva Skandha Sanagala
Associate Professor and HOD
Dept. of CSE

Dr. Siva Skandha Sanagala
Associate Professor and HOD
Dept. of CSE

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express our heartfelt gratitude to all the people who helped in making this project a grand success.

We are grateful to **Siva Skandha Sanagala (Associate Professor and HOD)**, Department of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work. We would like to thank**, Siva Skandha Sanagala,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete our project work successfully.

We are very grateful to **Dr. Ghanta Devadasu**, Dean-Academics, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Dept Name for their co-operation

We express our sincere thanks to **Shri. Ch. Gopal Reddy**, Secretary& Correspondent, CMR Group of Institutions, and **Shri Ch Abhinav Reddy**, CEO, CMR Group of Institutions for their continuous care and support.

Finally, we extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly or indirectly in completion of this project work.

M.Pavan 20H51A05H9
P.Neethika 20H51A05J2

TABLE OF CONTENTS

List of Figures

ABSTRACT

In the dominance of digital media and creative content generation, there is a growing need for automated tools that can seamlessly blend the artistic style of one image with the content of another. We came with a solution to design and implement an efficient deep learning model for style transfer.

This model takes a content image ‘C’ and a style image ‘S’ and merge both images to produce a new image that contains the content of image ‘C’ and the style of image ‘S’. This can be implemented by neural style transferring using deep learning which involves transferring the artistic style of one image with the content of another image, resulting in creative and visually appealing images as outputs. It will be implemented through Convolutional Neural Networks (CNNs), utilizing pre-trained models. The process involves extracting features from both content and style images through pre-trained network, pre-processing of images, optimizing a generated image to minimize content and style loss and iteratively refining the results.

CHAPTER 1
INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1. Problem Statement

In the dominance of digital media and creative content generation, there is a growing need for automated tools that can seamlessly blend the artistic style of one image with the content of another. The problem at hand is to develop a robust and efficient style transfer algorithm capable of preserving the content of an input image while applying the desired artistic style from a reference image. This project aims to address the challenges of creating a user-friendly, real-time style transfer solution that produces high-quality results across various domains, including art, photography, and graphic design.

In the world of digital artistry, our Neural Style Transfer project pushes the boundaries of creativity and innovation. Neural Style Transfer is the combination of art and technology, which provides a way to transform ordinary images into visually stunning artworks. This technique uses the power of deep learning, specifically convolutional neural networks (CNNs), to seamlessly blend the content of one image with the artistic style of another[1]. The project aims to explore and implement neural style transfer algorithms, to create a fusion of content and style. This project not only explores the intersection of technology and art but also opens new way of personalized digital content creation. This project's main goals are to have a thorough understanding of NST, from theory to practical. The process involves employing pretrained models, defining loss functions, and optimizing the generated image to achieve a neat mixture of content and style. Additionally, it must focus is on user, with the development of this interface that makes users to upload images, select artistic styles, and observe real-time transformations. This project aims to make the artistic potential of NST accessible to a broader audience [2]. The Stochastic Gradient Descent (SGD) algorithm is the central component of the project. SGD is a machine learning model optimization variant of the Gradient Descent technique. It solves the classic Gradient Descent methods computational inefficiencies while working with big datasets in machine learning applications

1.2. Research Objective

Our goal is to develop a wen app that can take any photograph and apply the artistic style of a another image to it, creating a new image that blends the content of the photo with the aesthetic characteristics of the painting. This process, known as neural style transfer, involves training a deep learning model to understand the content of an image (such as objects, people, and scenery) and the artistic style of a painting (such as brushstrokes, colors, and textures). By achieving this objective, we aim to provide a tool that artists, photographers, and designers can use to quickly and easily transform their images into unique and visually captivating artworks, enhancing their creative capabilities and enabling them to explore new artistic possibilities.

Imagine if you could take any ordinary photo you've captured on your phone—a picture of your pet, a scenic landscape, or even a selfie—and turn it into a beautiful artwork inspired by famous paintings like Picasso. That's what we're aiming to achieve with our project. We want to teach computers to understand not only what's in the photo but also the artistic style of paintings. Then, using this understanding, the computer can apply the style of a painting onto your photo, creating a stunning blend of reality and artistry. This could be incredibly useful for artists who want to experiment with different styles or for photographers who want to give their images a unique touch. Ultimately, we hope that our work will empower people to unleash their creativity in new and exciting ways, all with the help of technology.

This tool could be incredibly useful for artists, photographers, and anyone who loves being creative. Artists could experiment with different styles without having to paint everything from scratch. Photographers could give their pictures a unique and artistic touch, making them stand out even more. Ultimately, we hope that by making this technology accessible, more people will be inspired to explore their creativity and create amazing art with just the click of a button.

1.3. Project Scope and Limitations

The scope of a neural style transfer project lies in its ability to blend the content of one image with the artistic style of another, creating visually unique results. This technology has a wide range of potential applications and directions for development.

One aspect of the project's scope involves improving the algorithms used for style transfer. Researchers are constantly working on refining these algorithms to make them faster, more efficient, and capable of producing higher-quality stylized images. By enhancing the underlying techniques, we can unlock new possibilities and ensure better results for users. Another area of exploration is the development of real-time applications. Currently, style transfer can be computationally intensive and time-consuming, but future advancements could enable it to be performed in real-time or near real-time. This would open up opportunities for applications like live video processing, interactive experiences, and augmented reality, where users can see stylized content in the moment.

Additionally, the project scope extends to creating interactive interfaces for style transfer.. Interactive interfaces would empower users to explore their creativity and tailor the style transfer process to their preferences, leading to more engaging and personalized experiences. Overall, the scope of a neural style transfer project brings the underlying algorithms, exploring real-time applications and interactive interfaces, and extending the technology to new domains.

LIMITATIONS:

Neural Style Transfer has a few limitations that are listed below:

Computational Resources: NST algorithms may sometimes requires significant processing power and memory to perform style transfer, especially for high-resolution images. This limits its practicality.

Artistic Style Representation: NST relies on pre-defined artistic styles extracted from reference images. Capturing the complexities of artistic styles accurately can be challenging, leading to limitations in the variety of styles.

Content-Style Mismatch: In some cases, the content and style of input images may not align well, leading to undesirable results.

Loss of Content Details: During the style transfer process, there is a trade-off between preserving content details and emphasizing style characteristics. In some cases, important content features may be lost, particularly in cases where content and style elements may overlap.

Ethical Concerns: There are ethical considerations related to the use of NST, including copyright infringement when using copyrighted artworks as style references and potential misuse for generating misleading or harmful content. Ensuring ethical use and responsible deployment of NST technology is essential to mitigate these concerns.

CHAPTER 2
BACKGROUND WORK

CHAPTER 2

BACKGROUND WORK

The primary part of background for any project includes reviewing of academic papers, articles, and books related to neural style transfer. This includes seminal papers like "A Neural Algorithm of Artistic Style" by Gatys et al. (2015) [3] and other included and related research that builds upon this work.

The next step includes understanding the following:

Deep Image Representations**:** The comes about displayed underneath were created on the premise of the VGG arrange, which was prepared to perform protest acknowledgment and localisation and is portrayed extensively within the unique work. We utilized the function area given by using a standardized version of the sixteen convolutional and 5 pooling layers of the nineteen-layer VGG community[4]. We normalized the arrange by using scaling the weights such that the merciless enactment of every convolutional channel over pix and positions is break even with to one. Such re-scaling may be done for the VGG arrange without converting its yield, as it carries because it had been correcting direct actuation capacities and no normalization or pooling over function maps. We don't use any of the completely related layers. The display is publicly handy and may be investigated within the caffe-framework. For picture combo we discovered that supplanting the maximum pooling operation via regular pooling yields particularly extra enticing comes approximately, that is why the pics appeared were produced with normal pooling.

CNN in Deep Learning: A well-known neural network may additionally have an input layer, output layer, and an hidden layers. CNNs are stimulated by using the structure of the mind. just like a neuron within the mind approaches and transmits records in some unspecified time in the future of the body, synthetic neurons or nodes in convolutional neural networks take inputs, strategize them and also send the final result of the output as output[5-6]. Photos are provided as an introduction. The input method takes image pixels as input for the image of the array. CNN may have many hidden layers that calculate the extracted features of the image[7]. This can include convolutions, pooling, rectified linear devices, and entire layers. Convolution is an

important layer for removing the input image. The link layer completely separates and defines the product within the output layer. A convolutional neural network is a feeder network in which information flows along a path from input to output. Just as artificial neural networks (ANN) are inspired by biology[8], this is also the inspiration for neural networks (CNN). The cortex, located in the brain, has alternating layers of simple and complex cells that inspire their structure. There are many versions of CNN architectures; However, popularly there are convolution and pooling (or subsampling) layers that can be divided into groups.

Style Transfer: Gatys et al. By combining reduction and reconstruction, can be an additional tool for the ability to extract image content from previous communication to distinguish between other images. Stylish together; A similar method is To deal with the deficiencies of in step with-pixel losses and empower our loss features to superior diploma perceptual and semantic contrasts used for texture. Their strategy produces excellent outcomes, but is computationally expensive because each back and forth through a preliminary discussion. To overcome this computational burden, we train feed forward networks to quickly solve the optimization problem

2.1 Neural Algorithm of Artistic Style

2.1.1 Introduction

In fine art, especially painting, humans have mastered the skill to create unique visual experiences through composing a complex interplay between the content and style of an image. Thus far the algorithmic basis of this process is unknown and there exists no artificial system with similar capabilities. However, in other key areas of visual perception such as object and face recognition near-human performance were recently demonstrated by a class of biologically inspired vision models called Deep Neural Networks. Here we introduce an artificial system based on a Deep Neural Network that creates artistic images of high perceptual quality. The system uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images. Moreover, in light of the striking similarities between performance-optimized artificial neural networks and biologic

vision, our work offers a path forward to an algorithmic understanding of how humans create and perceive artistic imagery[3].

2.1.2 Merits, Demerits and Challenges

The neural algorithm of artistic style, as proposed by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge in their 2015 paper titled "A Neural Algorithm of Artistic Style" (published in IEEE), has certain merits and demerits.

Merits:

- High-Quality Artistic Transformations: This approach is capable of producing high quality artistic transformations that closely resemble the style of famous artworks. It allows for the generation of visually appealing and creative images.
- Content Preservation: The method effectively preserves the content of the input image while applying the style of a reference image, ensuring that the final output remains recognizable and meaningful.
- Flexibility: It is highly versatile and can be applied to a wide range of content and styles, making it a valuable tool for artists and designers. You can apply various artistic styles to any image.
- End-to-End Deep Learning: Leveraging deep learning, this approach can learn and adapt to different styles and content automatically, without the need for manual feature engineering.

Demerits:

- Computational Complexity: The neural algorithm of artistic style can be computationally expensive and may require powerful hardware, such as GPUs, for practical usage. This limits its real-time or large-scale applicability.
- Training: The method involves training a convolutional neural network (CNN) on a particular style image, which can be time-consuming and may not work well for less common or less well-defined artistic styles.
- Parameter Tuning: Achieving the desired results often involves fine-tuning hyperparameters, which can be a time-consuming process. Adjusting these parameters for optimal output can be challenging for non-experts.
- Limited Realism: While the method produces visually appealing images, it doesn't inherently enhance the realism of the content. The output can sometimes look more like a

- painting or artwork than a realistic photograph.
- Legal and Ethical Concerns: Applying the style of famous artworks to images can raise legal and ethical concerns, particularly related to copyright and intellectual property rights.

In summary, the neural algorithm of artistic style proposed by Gatys, Ecker, and Bethge is a powerful technique for creating visually appealing images by combining content and style. It shares similar merits and demerits with the original approach, including computational challenges, the need for parameter tuning, and the potential for loss of realism and semantic information. Despite these limitations, it has had a significant impact on the field of computer vision and art generation.

2.1.3. Implementation of A Neural Algorithm of Artistic Style

Implementing the Neural Algorithm of Artistic Style, as proposed by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, typically involves using deep learning frameworks like TensorFlow or PyTorch. Below, I'll provide a high-level outline of the steps involved in implementing this algorithm. Please note that this is a simplified version for educational purposes, and a complete implementation may require additional details and optimizations.

- The initial step is preprocessing where; the content image and style image are loaded. Resize both images to the same dimensions. The pixel values of both images are normalized. Images are converted into tensors that can be processed by the deep learning framework.
- A pre-trained model is choosing convolutional neural network. You can load these models from TensorFlow Hub or torch vision models in PyTorch.
- Define Content and Style Layers: Identify the layers in the chosen model that will be used to extract content and style information. Typically, earlier layers are used for style, and later layers are used for content.
- Loss Functions: Define the content loss, which is the Mean Squared Error (MSE) between the feature maps of the content image and the generated image at the selected content layers.
- Define the style loss, which is computed by comparing the Gram matrices of the feature maps at the selected style layers.

- Generate the Target Image: Initialize a target image (usually as a copy of the content image).
- Create a TensorFlow Variable or PyTorch Tensor to represent the target image, which allows you to optimize its pixel values.
- Optimization: Use an optimization algorithm to minimize a combination of content loss and style loss. You can also add a total variation loss to reduce noise in the generated image.
- Perform several iterations of the optimization process, updating the target image to minimize the combined loss.
- Postprocessing: Once you've obtained the optimized target image, deformalize it by reversing the normalization process.

2.2 Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

2.2.1 Introduction

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. We present an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. Our goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from G(X) is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping $F : Y \rightarrow X$ and introduce a cycle consistency loss to enforce $F(G(X)) \approx X$ (and vice versa). Qualitative results are presented on several tasks where paired training data does not exist, including collection style transfer, object transfiguration, season transfer, photo enhancement, etc. Quantitative comparisons against several prior methods demonstrate the superiority of our approach[9].

2.2.2 Merits, Demerits and Challenges

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks by JunYan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros, introduces a method for translating images from one domain to another without the need for paired training data. The primary technique used in this paper is CycleGAN, which leverages cycle-consistency to enforce a bijective mapping between two domains. Here are the merits, demerits, and challenges associated with unpaired image-to-image translation using Cycle-Consistent Adversarial Networks (CycleGAN):

Merits:

- No Paired Data Required: The most significant advantage of CycleGAN is that it can perform image-to-image translation without the need for paired data (i.e., images from the source domain and their corresponding images in the target domain). This makes it a more versatile and practical solution since collecting paired data can be expensive and time-consuming.
- Applicability: CycleGAN is versatile and can be applied to various image translation tasks, including style transfer, season transfer, and more. It can also be adapted for other domains beyond images.
- Generative Adversarial Network (GAN) Framework: CycleGAN is based on GANs, which have shown their effectiveness in generating high-quality images. This ensures that the generated images have a high degree of realism.

Demerits:

- Limited Control: CycleGAN may not provide fine-grained control over the translation process. While it can learn to perform image translation, it may not allow users to specify precise attributes or features to be translated, limiting its use in certain applications.
- Quality and Consistency: The quality and consistency of the generated images can vary. Some images may look realistic, while others may not be as convincing, depending on the complexity of the translation task.
- Computational Resources: Training a Cycle GAN model can be computationally intensive and time-consuming, especially for high-resolution images and complex

translation tasks. This may require powerful hardware and substantial training time.

- The challenges to this method include: Mode Collapse where GAN"s including those which are used in Cycle GAN can suffer from mode collapse where the generator produces a limited set of similar images. Another challenge is that Data Augmentation where processing data set and augmenting it to for better results can be tough.

2.2.3 Implementation

Network Architecture : They have adopted an architecture for generative networks. which have shown impressive results for neural style transfer and superresolution. This network contains three convolutions, several residual blocks, two fractionally-strided convolutions with stride 1 2 , and one convolution that maps features to RGB. They used 6 blocks for 128 × 128 images and 9 blocks for 256×256 and higher-resolution training images. They have used an instance normalization. For the discriminator networks they used 70 × 70 PatchGANs, which aim to classify whether overlapping image patches are real or fake.

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

2.3.1 Introduction

They have considered image transformation problems, where an input image is transformed into an output image. Recent methods for such problems typically train feed-forward convolutional neural networks using a per-pixel loss between the output and ground-truth images. Parallel work has shown that high-quality images can be generated by defining and optimizing perceptual loss functions based on high-level features extracted from pretrained networks. They have combined the benefits of both approaches, and propose the use of perceptual loss functions for training feed-forward networks for image transformation tasks. It shows results on image style transfer, where a feed-forward network is trained to solve the optimization problem proposed by Gatys et al in real-time. Compared to the optimization-based method, our network gives similar qualitative results but is three orders of magnitude faster. They also experiment with single-image super-resolution, where replacing a per-pixel loss with a perceptual loss gives visually pleasing results[10].

2.3.2 Merits, Demerits and Challenges

Merits:

- Real-Time Style Transfer: The method allows for real-time style transfer, which means that it can be applied to images or videos in a live or interactive setting. This is a significant advantage for applications like video games, augmented reality, and live video processing.
- Style Preservation: The approach is effective at preserving the artistic style of the reference image during style transfer. This is important for applications where the style is a key component, such as art generation and branding.
- Generalization: The technique can be applied to a wide range of style transfer and superresolution tasks without needing specific models or training for each task. This makes it more versatile and adaptable to various applications.
- Low-Level Details: In the case of super-resolution, the approach can effectively recover fine details and textures in images, making it useful for enhancing image quality.

Demerits:

- Resource Intensive: Implementing perceptual loss methods can be computationally intensive, particularly during training. Pre-trained deep neural networks like VGG are required, and training may demand powerful hardware.
- Complex Implementation: Implementing perceptual loss-based style transfer and super resolution can be more complex than traditional techniques. It requires a good understanding of deep learning frameworks and often involves custom code.
- Overfitting Risk: Like other deep learning methods, there's a risk of overfitting, especially if the dataset used for training is limited or not diverse. Careful regularization and data augmentation are required.
- Model Dependency: The technique relies on pre-trained deep neural networks for feature extraction. Changes or advancements in these networks may affect the performance and stability of the method.

- style transfer or super-resolution process. Users have limited direct control over specific attributes or elements of the output.
- Hyperparameter Tuning: Tuning hyperparameters in the loss function and the optimization process can be challenging and time-consuming.

2.3.3. Implementation

The implementation of "Perceptual Losses for Real-Time Style Transfer and Super-Resolution" proposed by Justin Johnson, Alexandre Alahi, and Li Fei-Fei involves several steps

- Feature Extraction:With the selected neural network in place, the next step involves feature extraction. This is done by feeding both the reference image (for style transfer) or the low-resolution image (for super-resolution) and the target image (generated) through the network. Intermediate feature maps are extracted from specific layers of the network to capture information about texture, style, and content.
- Perceptual Loss Calculation:The perceptual loss is then computed using the extracted features. This loss typically comprises three components: content loss, style loss, and total variation loss. The content loss measures the similarity between the feature maps of the generated image and the reference image. The style loss quantifies the difference in texture and style between the generated image and the reference image. The total variation loss helps smooth out the final output.
- Optimization: Once the perceptual loss is computed, it is used as the objective function for optimization. The goal is to minimize this loss by adjusting the parameters of the generated image. This is often achieved using gradient-based optimization techniques like stochastic gradient descent (SGD) or its variants. The process involves iteratively updating the generated image to make it more similar to the reference image in terms of content and style.
- Post-Processing: Depending on the application and the specific neural network architecture used, some post-processing steps may be needed. For example, in the case of super-resolution, additional techniques like bicubic interpolation may be used to upscale the generated image to the desired resolution.

- Hyperparameter Tuning: Fine-tuning the hyperparameters is a crucial aspect of the implementation. These hyperparameters include the learning rate, the weighting of the content and style losses, the number of iterations, and regularization terms. Careful tuning is necessary to achieve the desired balance between content preservation and style
  Transfer.
- Deployment and Real-Time Processing: The final step involves deploying the trained model for real-time processing. This may require optimizing the implementation for speed and efficiency to achieve real-time or interactive performance, which could involve optimizations like parallelization or using hardware accelerators.
- Neural Network Architecture: The first step in implementing perceptual losses for style transfer and super-resolution is to define the neural network architecture. In the original paper, a pre-trained VGG network is commonly used. VGG is well-suited for its ability to capture high-level features from images, which will be used for perceptual loss computation. Depending on the task and the specific architecture used, the layers from which features are extracted can vary.

CHAPTER 3
PROPOSED SYSTEM

CHAPTER 3
PROPOSED SYSTEM

3.1 Objective of Proposed System

The objective of the proposed neural style transfer model is to create a new image that combines the content o one image (content image) with the artistic style of another image (style image). This model separates content and style by analyzing both images to differentiate between the image's content (objects, shapes) and the style (brushstrokes, color palettes). Then the model preserves the content by ensuring the generated image retains the key elements and structure from the content image. Then model incorporates the style elements from the style image into the generated image. This include transferring brushstroke patterns, color distributions, or other stylistic features. Finally this model strives to produce an image that looks like the content image, but rendered in the artistic style of the reference image. This proposed model works iteratively as we used stochastic gradient descent algorithm to minimize the loss functions.

3.2 Implementation of Proposed Model

Our framework comprises of two components: an picture transformation network fW and a loss network ϕ, which is utilized to characterize different loss functions l1, . . , lk. The picture transformation network could be a leftover convolutional neural network parameterized by weights W; changes over the input images x into output images y by mapping ŷ = fW (x). Each loss function calculates a scalar esteem li(ŷ, yi) that measures the distinction between the output picture ŷ and a target picture yi. To minimize the weighted combination of loss functions, the image transformation network is trained using stochastic gradient descent algorithm :

$$W^* = \arg\min_W E_{x,\{yi\}} [\sum_{i=10} \lambda_i l_i(f_W(x),y_i)]$$

To deal with the deficiencies of in step with-pixel losses and empower our loss features to superior diploma perceptual and semantic contrasts between images, we draw motivation from recent work on developing snap shots via optimization. The center idea of these techniques

is that convolutional neural networks already educated for photo category have already discovered to encode the perceptual and semantic records that we need to degree in our loss features. on this manner, we utilize a network ϕ already skilled for image category as a settled loss network to construct our loss functions. At that point our deep convolutional neural network is trained utilizing loss capabilities, which might be furthermore deep convolutional neural networks. The loss network ϕ is utilized to construct a feature loss l_{feat}^\\phi and a style loss, l_{style}^\\phi which measures contrasts in content and style between pictures. For each input picture x, we have a content target yc and a style target ys. For style transfer, the target content yc is the input picture x, and the output picture ŷ must combine the content of x = yc with the style of ys; we train a network consistent with target fashion. In single-image super-resolution, the enter image x is a input, the goal content yc is the real excessive-resolution image, and the style reconstruction loss isn't utilized. We train a network with the awesome resolution aspect.

$$L_{CONTENT}(\vec{p},\vec{x},l) = \frac{1}{2}\sum_{i,j}(F^{l}_{ij} - P^{l}_{ij})^2 \qquad \ldots\ldots (1)$$

$$L_{STYLE}(\vec{a},\vec{x}) = \sum_{l=0}^{L} w_l E_l \qquad \ldots\ldots (2)$$

$$L_{TOTAL} = \alpha L_{CONTENT} + \beta L_{STYLE} \qquad \ldots\ldots (3)$$

ALGORITHM

Stochastic gradient descent (SGD) is a type of gradient descent, a technique that functions well with optimization (e.g. variance or variance). It can be considered an approximation of gradient descent, as it replaces the actual gradient (e.g. from all statistical data) with its estimate (e.g. from a random subset of data). Particularly in hyperdimensional optimization problems, this reduces the computational burden and increases the speed of iterations in exchange for reduced convergence costs [11]. Error functions are not often as easy as an ordinary parabola. most customarily they have plenty of hills and valleys, just like the characteristic pictured right here. on this graph, if real gradient descent started on the left facet of the graph, it might stop on the left valley because no matter which route you travel from this factor, you need to travel upwards. This factor is known as a nearby minimal. but, there exists another factor inside the graph that is decrease. the lowest point in the complete graph is the

global minimal, that's what stochastic gradient descent tries to discover. The following is the process of Stochastic Gradient Descent algorithm:

First shuffle the dataset randomly and choose a data point randomly and cycle through all the elements. Now cycle on all weights or parameters and adjust the current weight or parameter under the derivative of the cost or loss function. Calculate the new Gradient with the new value of the parameter or weights. Repeat all the above steps until convergence or global minima is achieved.

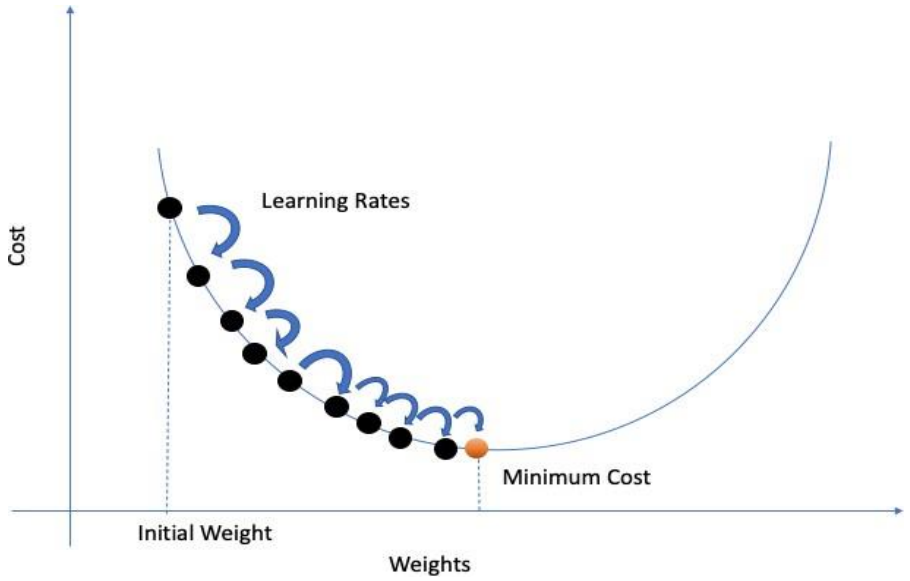


Figure 3.2.1 SGD Graph.

3.3 Designing ARCHITECTURE

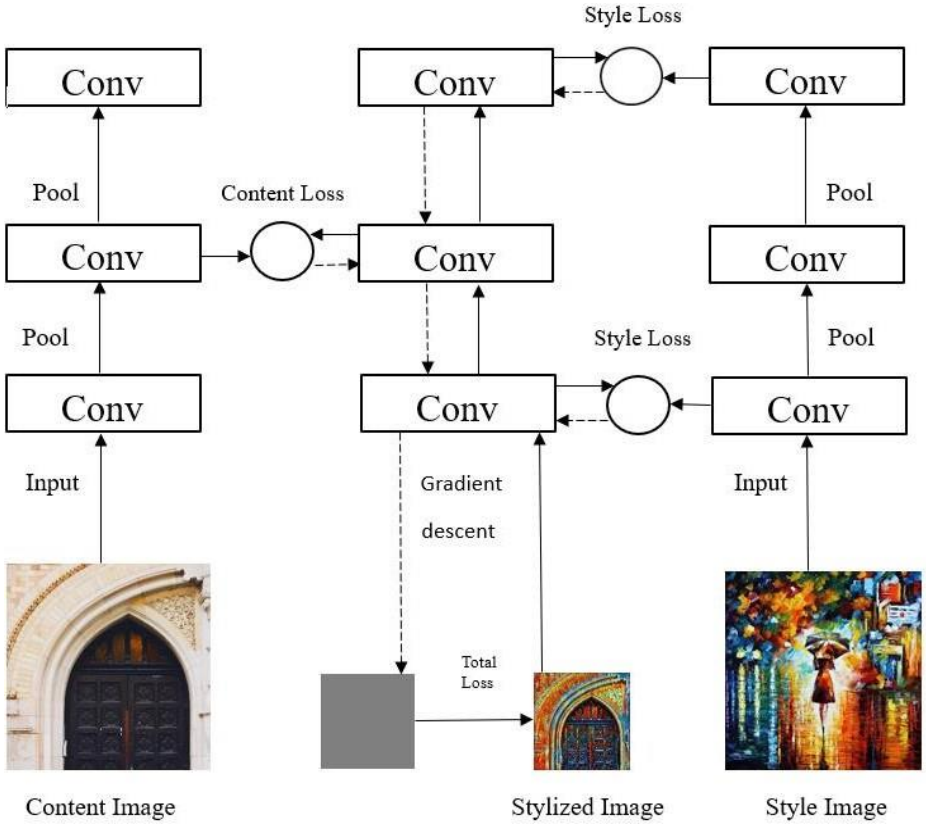


Figure 3.3.1 System Architecture.

3.4 Stepwise Implementation and Code

IMPLEMENTATION

- Clone this repository : https://github.com/PavanM07/Neural_Style_Transfer.git using git bash.
- Open “neural_style” folder in VS code.
- Open new terminal in the VS code and type “cd neural_style” and enter.

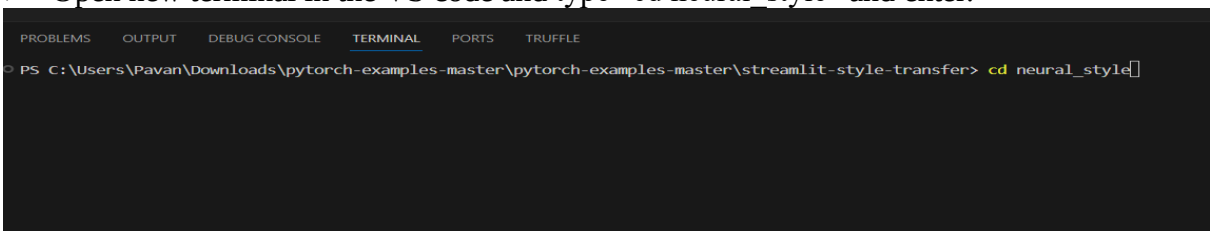


Figure 3.4.1 VS Code Terminal.

- Then type “streamlit run main.py” and enter.

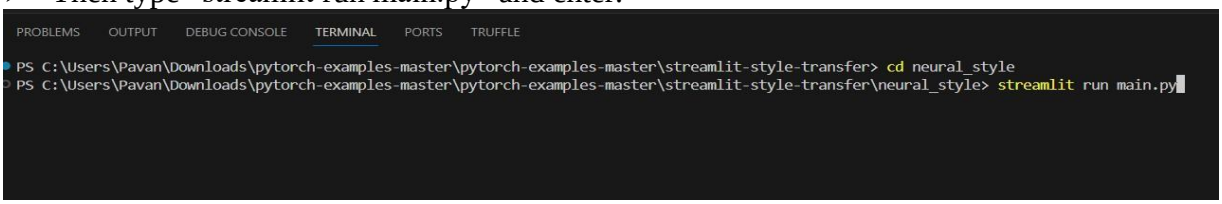


Figure 3.4.2 To activate Streamlit.

- Select the content image from the field ‘Select Image’.

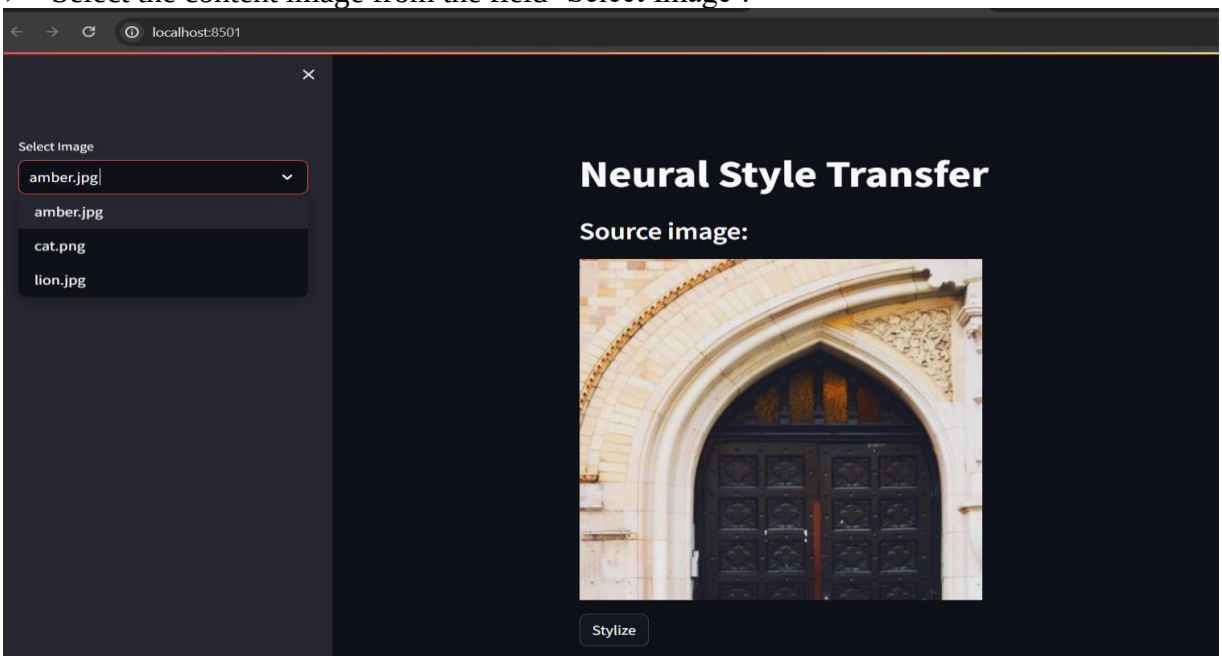


Figure 3.4.3 Selecting Content Image.

- Then select the style image from the field ‘Select Style’.

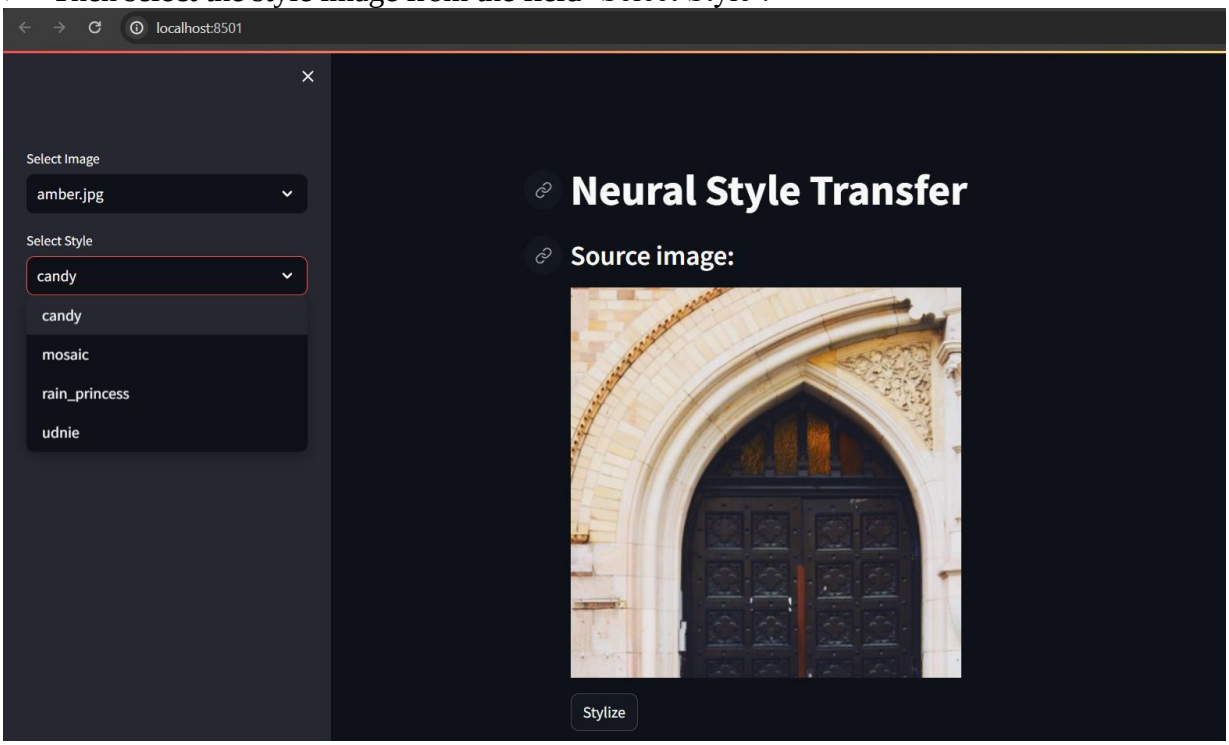


Figure 3.4.4 Selecting the Stylized model.

- Then click on Stylize button. On clicking this button, we get the output image as shown below.

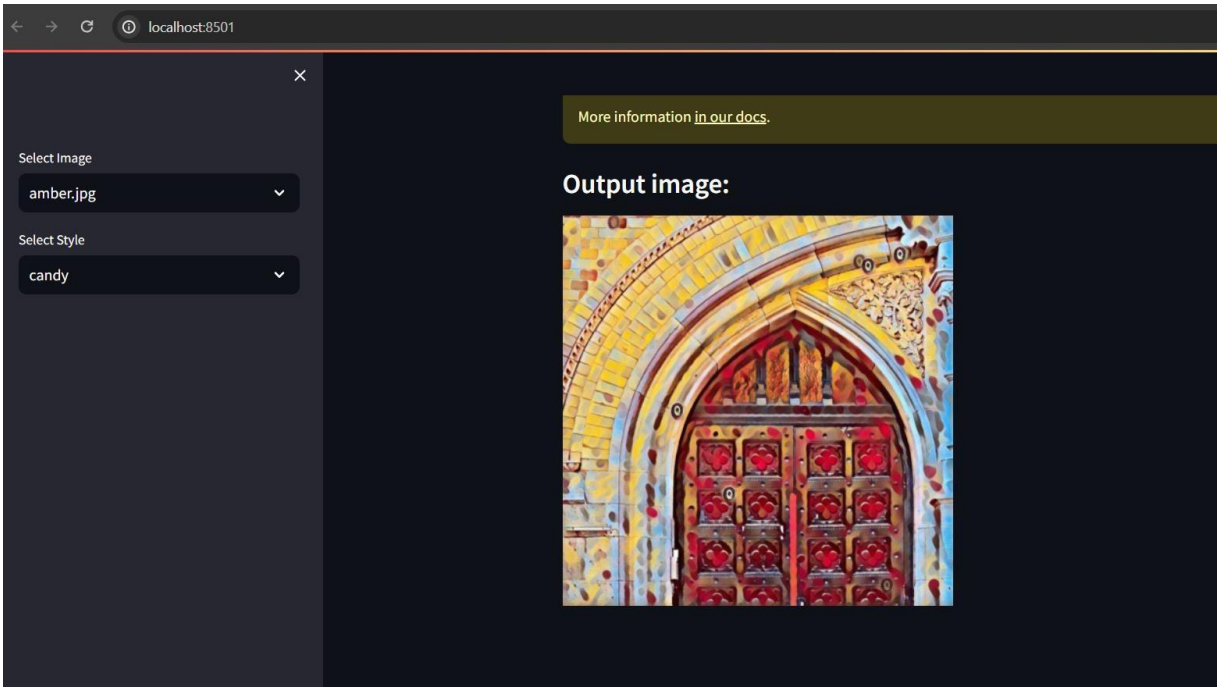


Figure 3.4.5 Output Image.

SOURCE CODE:

transformer_net.py:

```
import torch


class TransformerNet(torch.nn.Module):
    def __init__(self):
        super(TransformerNet, self).__init__()
        # Initial convolution layers
        self.conv1 = ConvLayer(3, 32, kernel_size=9, stride=1)
        self.in1 = torch.nn.InstanceNorm2d(32, affine=True)
        self.conv2 = ConvLayer(32, 64, kernel_size=3, stride=2)
        self.in2 = torch.nn.InstanceNorm2d(64, affine=True)
        self.conv3 = ConvLayer(64, 128, kernel_size=3, stride=2)
        self.in3 = torch.nn.InstanceNorm2d(128, affine=True)
        # Residual layers
        self.res1 = ResidualBlock(128)
        self.res2 = ResidualBlock(128)
        self.res3 = ResidualBlock(128)
        self.res4 = ResidualBlock(128)
        self.res5 = ResidualBlock(128)
        # Upsampling Layers
        self.deconv1 = UpsampleConvLayer(128, 64, kernel_size=3, stride=1, upsample=2)
        self.in4 = torch.nn.InstanceNorm2d(64, affine=True)
        self.deconv2 = UpsampleConvLayer(64, 32, kernel_size=3, stride=1, upsample=2)
        self.in5 = torch.nn.InstanceNorm2d(32, affine=True)
        self.deconv3 = ConvLayer(32, 3, kernel_size=9, stride=1)
        # Non-linearities
        self.relu = torch.nn.ReLU()

    def forward(self, X):
        y = self.relu(self.in1(self.conv1(X)))
        y = self.relu(self.in2(self.conv2(y)))
        y = self.relu(self.in3(self.conv3(y)))
        y = self.res1(y)
        y = self.res2(y)
        y = self.res3(y)
        y = self.res4(y)
        y = self.res5(y)
        y = self.relu(self.in4(self.deconv1(y)))
        y = self.relu(self.in5(self.deconv2(y)))
        y = self.deconv3(y)
        return y
```

```
class ConvLayer(torch.nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride):
        super(ConvLayer, self).__init__()
        reflection_padding = kernel_size // 2
        self.reflection_pad = torch.nn.ReflectionPad2d(reflection_padding)
        self.conv2d = torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride)

    def forward(self, x):
        out = self.reflection_pad(x)
        out = self.conv2d(out)
        return out


class ResidualBlock(torch.nn.Module):
    """ResidualBlock
    introduced in: https://arxiv.org/abs/1512.03385
    recommended architecture: http://torch.ch/blog/2016/02/04/resnets.html
    """

    def __init__(self, channels):
        super(ResidualBlock, self).__init__()
        self.conv1 = ConvLayer(channels, channels, kernel_size=3, stride=1)
        self.in1 = torch.nn.InstanceNorm2d(channels, affine=True)
        self.conv2 = ConvLayer(channels, channels, kernel_size=3, stride=1)
        self.in2 = torch.nn.InstanceNorm2d(channels, affine=True)
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        residual = x
        out = self.relu(self.in1(self.conv1(x)))
        out = self.in2(self.conv2(out))
        out = out + residual
        return out


class UpsampleConvLayer(torch.nn.Module):
    """UpsampleConvLayer
    Upsamples the input and then does a convolution. This method gives better results
    compared to ConvTranspose2d.
    ref: http://distill.pub/2016/deconv-checkerboard/
    """

    def __init__(self, in_channels, out_channels, kernel_size, stride, upsample=None):
        super(UpsampleConvLayer, self).__init__()
        self.upsample = upsample
```

```
reflection_padding = kernel_size // 2
self.reflection_pad = torch.nn.ReflectionPad2d(reflection_padding)
        self.conv2d = torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride)

    def forward(self, x):
        x_in = x
        if self.upsample:
            x_in = torch.nn.functional.interpolate(x_in, mode='nearest',
scale_factor=self.upsample)
        out = self.reflection_pad(x_in)
        out = self.conv2d(out)
        return out
```

vgg.py :

```
from collections import namedtuple

import torch
from torchvision import models


class Vgg16(torch.nn.Module):
    def __init__(self,  requires_grad=False):
        super(Vgg16, self).__init__()
        vgg_pretrained_features = models.vgg16(pretrained=True).features
        self.slice1 = torch.nn.Sequential()
        self.slice2 = torch.nn.Sequential()
        self.slice3 = torch.nn.Sequential()
        self.slice4 = torch.nn.Sequential()
        for x in range(4):
            self.slice1.add_module(str(x), vgg_pretrained_features[x])
        for x in range(4, 9):
            self.slice2.add_module(str(x), vgg_pretrained_features[x])
        for x in range(9, 16):
            self.slice3.add_module(str(x), vgg_pretrained_features[x])
        for x in range(16, 23):
            self.slice4.add_module(str(x), vgg_pretrained_features[x])
        if not requires_grad:
            for param in self.parameters():
                param.requires_grad = False

    def forward(self, X):
        h = self.slice1(X)
        h_relu1_2 = h
        h = self.slice2(h)
```

```
        h_relu2_2 = h
        h = self.slice3(h)
        h_relu3_3 = h
        h = self.slice4(h)
        h_relu4_3 = h
        vgg_outputs = namedtuple("VggOutputs", ['relu1_2', 'relu2_2', 'relu3_3', 'relu4_3'])
        out = vgg_outputs(h_relu1_2, h_relu2_2, h_relu3_3, h_relu4_3)
        return out
```

utils.py :

```
import torch
from PIL import Image


def load_image(filename, size=None, scale=None):
    img = Image.open(filename)
    if size is not None:
        img = img.resize((size, size), Image.LANCZOS)
    elif scale is not None:
        img = img.resize((int(img.size[0] / scale), int(img.size[1] / scale)), Image.LANCZOS)
    return img


def save_image(filename, data):
    img = data.clone().clamp(0, 255).numpy()
    img = img.transpose(1, 2, 0).astype("uint8")
    img = Image.fromarray(img)
    img.save(filename)


def gram_matrix(y):
    (b, ch, h, w) = y.size()
    features = y.view(b, ch, w * h)
    features_t = features.transpose(1, 2)
    gram = features.bmm(features_t) / (ch * h * w)
    return gram


def normalize_batch(batch):
    # normalize using imagenet mean and std
    mean = batch.new_tensor([0.485, 0.456, 0.406]).view(-1, 1, 1)
    std = batch.new_tensor([0.229, 0.224, 0.225]).view(-1, 1, 1)
    batch = batch.div_(255.0)
    return (batch - mean) / std
```

style.py :

```
import argparse
import os
import sys
import time
import re

import numpy as np
import torch
from torch.optim import Adam
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision import transforms
import torch.onnx

import utils
from transformer_net import TransformerNet
from vgg import Vgg16
import streamlit as st

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

@st.cache
def load_model(model_path):
    print('load model')
    with torch.no_grad():
        style_model = TransformerNet()
        state_dict = torch.load(model_path)
        # remove saved deprecated running_* keys in InstanceNorm from the checkpoint
        for k in list(state_dict.keys()):
            if re.search(r'in\d+\.running_(mean|var)$', k):
                del state_dict[k]
        style_model.load_state_dict(state_dict)
        style_model.to(device)
        style_model.eval()
        return style_model

@st.cache
def stylize(style_model, content_image, output_image):
    content_image = utils.load_image(content_image)
    content_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Lambda(lambda x: x.mul(255))
    ])
```

```
content_image = content_transform(content_image)
content_image = content_image.unsqueeze(0).to(device)

    with torch.no_grad():
        output = style_model(content_image).cpu()

    utils.save_image(output_image, output[0])
```

neural_style.py :

```
import argparse
import os
import sys
import time
import re

import numpy as np
import torch
from torch.optim import Adam
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision import transforms
import torch.onnx

import utils
from transformer_net import TransformerNet
from vgg import Vgg16


def check_paths(args):
    try:
        if not os.path.exists(args.save_model_dir):
            os.makedirs(args.save_model_dir)
        if args.checkpoint_model_dir is not None and not
(os.path.exists(args.checkpoint_model_dir)):
            os.makedirs(args.checkpoint_model_dir)
    except OSError as e:
        print(e)
        sys.exit(1)


def train(args):
    device = torch.device("cuda" if args.cuda else "cpu")

    np.random.seed(args.seed)
```

```
torch.manual_seed(args.seed)
transform = transforms.Compose([
        transforms.Resize(args.image_size),
        transforms.CenterCrop(args.image_size),
        transforms.ToTensor(),
        transforms.Lambda(lambda x: x.mul(255))
    ])
    train_dataset = datasets.ImageFolder(args.dataset, transform)
    train_loader = DataLoader(train_dataset, batch_size=args.batch_size)

    transformer = TransformerNet().to(device)
    optimizer = Adam(transformer.parameters(), args.lr)
    mse_loss = torch.nn.MSELoss()

    vgg = Vgg16(requires_grad=False).to(device)
    style_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Lambda(lambda x: x.mul(255))
    ])
    style = utils.load_image(args.style_image, size=args.style_size)
    style = style_transform(style)
    style = style.repeat(args.batch_size, 1, 1, 1).to(device)

    features_style = vgg(utils.normalize_batch(style))
    gram_style = [utils.gram_matrix(y) for y in features_style]

    for e in range(args.epochs):
        transformer.train()
        agg_content_loss = 0.
        agg_style_loss = 0.
        count = 0
        for batch_id, (x, _) in enumerate(train_loader):
            n_batch = len(x)
            count += n_batch
            optimizer.zero_grad()

            x = x.to(device)
            y = transformer(x)

            y = utils.normalize_batch(y)
            x = utils.normalize_batch(x)

            features_y = vgg(y)
            features_x = vgg(x)
```

```
content_loss = args.content_weight * mse_loss(features_y.relu2_2, features_x.relu2_2)

        style_loss = 0.
        for ft_y, gm_s in zip(features_y, gram_style):
            gm_y = utils.gram_matrix(ft_y)
            style_loss += mse_loss(gm_y, gm_s[:n_batch, :, :])
        style_loss *= args.style_weight

        total_loss = content_loss + style_loss
        total_loss.backward()
        optimizer.step()

        agg_content_loss += content_loss.item()
        agg_style_loss += style_loss.item()

        if (batch_id + 1) % args.log_interval == 0:
            mesg = "{}\tEpoch {}:\t[{}/{}]\tcontent: {:.6f}\tstyle: {:.6f}\ttotal: {:.6f}".format(
                time.ctime(), e + 1, count, len(train_dataset),
                            agg_content_loss / (batch_id + 1),
                            agg_style_loss / (batch_id + 1),
                            (agg_content_loss + agg_style_loss) / (batch_id + 1)
            )
            print(mesg)

        if args.checkpoint_model_dir is not None and (batch_id + 1) %
args.checkpoint_interval == 0:
            transformer.eval().cpu()
            ckpt_model_filename = "ckpt_epoch_" + str(e) + "_batch_id_" + str(batch_id + 1) +
".pth"
            ckpt_model_path = os.path.join(args.checkpoint_model_dir, ckpt_model_filename)
            torch.save(transformer.state_dict(), ckpt_model_path)
            transformer.to(device).train()

    # save model
    transformer.eval().cpu()
    save_model_filename = "epoch_" + str(args.epochs) + "_" + str(time.ctime()).replace(' ', '_')
+ "_" + str(
        args.content_weight) + "_" + str(args.style_weight) + ".model"
    save_model_path = os.path.join(args.save_model_dir, save_model_filename)
    torch.save(transformer.state_dict(), save_model_path)

    print("\nDone, trained model saved at", save_model_path)


def stylize(args):
```

```
    device = torch.device("cuda" if args.cuda else "cpu")
    content_image = utils.load_image(args.content_image, scale=args.content_scale)
    content_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Lambda(lambda x: x.mul(255))
    ])
    content_image = content_transform(content_image)
    content_image = content_image.unsqueeze(0).to(device)

    if args.model.endswith(".onnx"):
        output = stylize_onnx_caffe2(content_image, args)
    else:
        with torch.no_grad():
            style_model = TransformerNet()
            state_dict = torch.load(args.model)
            # remove saved deprecated running_* keys in InstanceNorm from the checkpoint
            for k in list(state_dict.keys()):
                if re.search(r'in\d+\.running_(mean|var)$', k):
                    del state_dict[k]
            style_model.load_state_dict(state_dict)
            style_model.to(device)
            if args.export_onnx:
                assert args.export_onnx.endswith(".onnx"), "Export model file should end with
.onnx"
                output = torch.onnx._export(style_model, content_image, args.export_onnx).cpu()
            else:
                output = style_model(content_image).cpu()
    utils.save_image(args.output_image, output[0])


def stylize_onnx_caffe2(content_image, args):
    """
    Read ONNX model and run it using Caffe2
    """

    assert not args.export_onnx

    import onnx
    import onnx_caffe2.backend

    model = onnx.load(args.model)
    prepared_backend = onnx_caffe2.backend.prepare(model, device='CUDA' if args.cuda else
'CPU')
    inp = {model.graph.input[0].name: content_image.numpy()}
```

```
  c2_out = prepared_backend.run(inp)[0]
  return torch.from_numpy(c2_out)
def main():
    main_arg_parser = argparse.ArgumentParser(description="parser for fast-neural-style")
    subparsers = main_arg_parser.add_subparsers(title="subcommands", dest="subcommand")

    train_arg_parser = subparsers.add_parser("train", help="parser for training arguments")
    train_arg_parser.add_argument("--epochs", type=int, default=2,
                                  help="number of training epochs, default is 2")
    train_arg_parser.add_argument("--batch-size", type=int, default=4,
                                  help="batch size for training, default is 4")
    train_arg_parser.add_argument("--dataset", type=str, required=True,
                                  help="path to training dataset, the path should point to a folder "
                                       "containing another folder with all the training images")
    train_arg_parser.add_argument("--style-image", type=str, default="images/style-
images/mosaic.jpg",
                                  help="path to style-image")
    train_arg_parser.add_argument("--save-model-dir", type=str, required=True,
                                  help="path to folder where trained model will be saved.")
    train_arg_parser.add_argument("--checkpoint-model-dir", type=str, default=None,
                                  help="path to folder where checkpoints of trained models will be saved")
    train_arg_parser.add_argument("--image-size", type=int, default=256,
                                  help="size of training images, default is 256 X 256")
    train_arg_parser.add_argument("--style-size", type=int, default=None,
                                  help="size of style-image, default is the original size of style image")
    train_arg_parser.add_argument("--cuda", type=int, required=True,
                                  help="set it to 1 for running on GPU, 0 for CPU")
    train_arg_parser.add_argument("--seed", type=int, default=42,
                                  help="random seed for training")
    train_arg_parser.add_argument("--content-weight", type=float, default=1e5,
                                  help="weight for content-loss, default is 1e5")
    train_arg_parser.add_argument("--style-weight", type=float, default=1e10,
                                  help="weight for style-loss, default is 1e10")
    train_arg_parser.add_argument("--lr", type=float, default=1e-3,
                                  help="learning rate, default is 1e-3")
    train_arg_parser.add_argument("--log-interval", type=int, default=500,
                                  help="number of images after which the training loss is logged, default is
500")
    train_arg_parser.add_argument("--checkpoint-interval", type=int, default=2000,
                                  help="number of batches after which a checkpoint of the trained model
will be created")

    eval_arg_parser = subparsers.add_parser("eval", help="parser for evaluation/stylizing
```

```
arguments")
    eval_arg_parser.add_argument("--content-image", type=str, required=True,
                                 help="path to content image you want to stylize")
    eval_arg_parser.add_argument("--content-scale", type=float, default=None,
                                 help="factor for scaling down the content image")
    eval_arg_parser.add_argument("--output-image", type=str, required=True,
                                 help="path for saving the output image")
    eval_arg_parser.add_argument("--model", type=str, required=True,
                                 help="saved model to be used for stylizing the image. If file ends in .pth -
PyTorch path is used, if in .onnx - Caffe2 path")
    eval_arg_parser.add_argument("--cuda", type=int, required=True,
                                 help="set it to 1 for running on GPU, 0 for CPU")
    eval_arg_parser.add_argument("--export_onnx", type=str,
                                 help="export ONNX model to a given file")

    args = main_arg_parser.parse_args()

    if args.subcommand is None:
        print("ERROR: specify either train or eval")
        sys.exit(1)
    if args.cuda and not torch.cuda.is_available():
        print("ERROR: cuda is not available, try running on CPU")
        sys.exit(1)

    #print(type(args))
    #print(args)
    #return
    if args.subcommand == "train":
        check_paths(args)
        train(args)
    else:
        stylize(args)


if __name___== "__main__":
    main()

# python neural_style.py eval --content-image "images/content-images/amber.jpg" --model
"saved_models/candy.pth" --output-image "images/output-images/amber-test.jpg"  --cuda 0

# python neural_style/neural_style.py train --dataset </path/to/train-dataset> --style-image
</path/to/style/image> --save-model-dir </path/to/save-model/folder> --epochs 2 --cuda 1
```

main.py :

```
# python3 -m venv venv
# . venv/bin/activate
# pip install streamlit
# pip install torch torchvision
# streamlit run main.py
import streamlit as st
from PIL import Image

import style

st.title('Neural Style Transfer')

img = st.sidebar.selectbox(
    'Select Image',
    ('amber.jpg', 'cat.png', 'lion.jpg')
)

style_name = st.sidebar.selectbox(
    'Select Style',
    ('candy', 'mosaic', 'rain_princess', 'udnie')
)


model= "saved_models/" + style_name + ".pth"
input_image = "images/content-images/" + img
output_image = "images/output-images/" + style_name + "-" + img

st.write('### Source image:')
image = Image.open(input_image)
st.image(image, width=400) # image: numpy array

clicked = st.button('Stylize')

if clicked:
    model = style.load_model(model)
    style.stylize(model, input_image, output_image)

    st.write('### Output image:')
    image = Image.open(output_image)
    st.image(image, width=400)
```

CHAPTER 4
RESULTS AND DISCUSSION

CHAPTER 4

RESULTS AND DISCUSSION

Neural style transfer is a technique in deep learning that combines the content of one image with the style of another image. It has gained significant attention due to its ability to generate visually appealing images that blend content and style in unique ways. The key aspects and results of neural style transfer include: algorithm overview, content representation, style representation, optimization, and the final results. Overall, neural style transfer represents a powerful tool for creative expression and image manipulation, with a wide range of applications and ongoing research efforts to improve its capabilities and efficiency.

4.1 Performance Metrics

Evaluating the performance of neural style transfer can be somewhat subjective since it primarily deals with the aesthetics of generated images. However, there are a few metrics and methods use to assess the quality of neural style transfer results:

Content Loss: These measures how closely the content of the generated image matches the content image. It's often calculated as the mean squared error (MSE) or the feature wise cosine distance between feature maps from a pre-trained convolutional neural network.

Perceptual Loss: Similar to content loss, it considers feature differences in deeper layers of a network to capture higher-level content. It's often used in combination with style loss.

Style Loss: Measures how well the style of the generated image matches the style image. It is computed by comparing the Gram matrices of feature maps from pre-trained networks.

Total Variation (TV) Loss: It's a regularization term that encourages spatial smoothness in the generated image. It can help reduce artifacts in the output.

Human Evaluation: Often, the best way to assess the quality of style transfer is through human judgment. Conduct user studies or surveys to get feedback from people on the generated images.

Comparative Evaluation: Compare the style transfer results with alternative methods or baseline techniques to demonstrate the superiority of your approach.

Execution Time: Measure the time it takes to perform the neural style transfer, as this can be an important factor in practical applications.

Style Diversity: If your neural style transfer method allows for multiple styles to be applied, you can measure the diversity of styles in the generated images.

Consistency in Video or Sequences: If you are applying style transfer to videos or image sequences, you can measure the temporal consistency to ensure smooth transitions between frames.

Depending on the specific use case, you might need to design custom metrics that capture certain desired features, textures, or objects in the generated images.

It's important to note that neural style transfer is often more of an art than a science, and the choice of metrics can depend on the specific goals of the project. In many cases, a combination of different metrics and human judgment is used to evaluate the performance comprehensively. Additionally, practical applications and user preferences often play a significant role in determining the success of a neural style transfer method. In conclusion, neural style transfer has proven to be a powerful and creative application of deep learning. It has the potential to revolutionize art and design while finding applications in various other fields. However, it is not without its challenges, including computational complexity and ethical concerns.

As research in this area continues, we can expect further advancements that will make neural style transfer more accessible and impactful in the creative and commercial realms.

Below is the graph showcasing the **Performance Measure** of the neural style transfer:

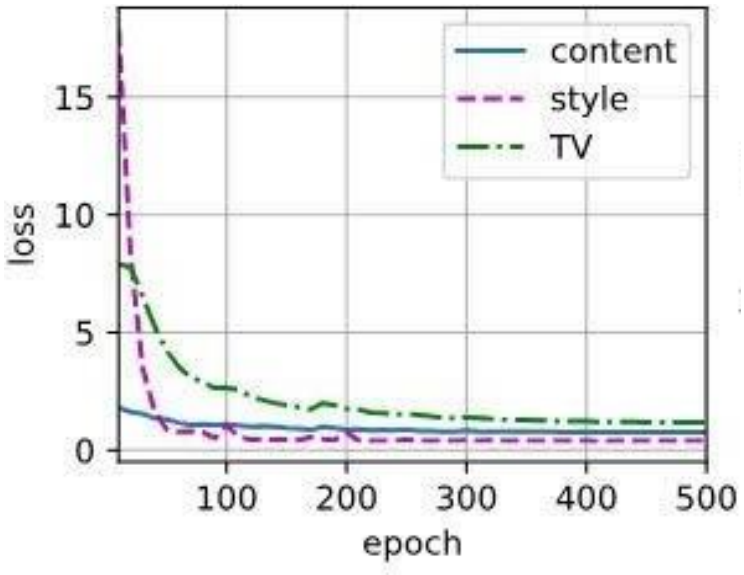


Figure 4.1.1 Performance Measure Graph

From the above shown graph we can conclude that:

The x-axis in the above graph is used to represent the epoch which means one complete pass of the training dataset through the algorithm. The y-axis represents the content loss that happens during the style transfer process. The two lines on the graph represent the content loss(blue) and the style loss(purple). As the number of epochs increases, the content loss and style loss both decrease. By this we can say that our model is getting better at capturing the content from the content image and the style from the style image.

Below are two result images of Neural Style Transfer:

Result 1:

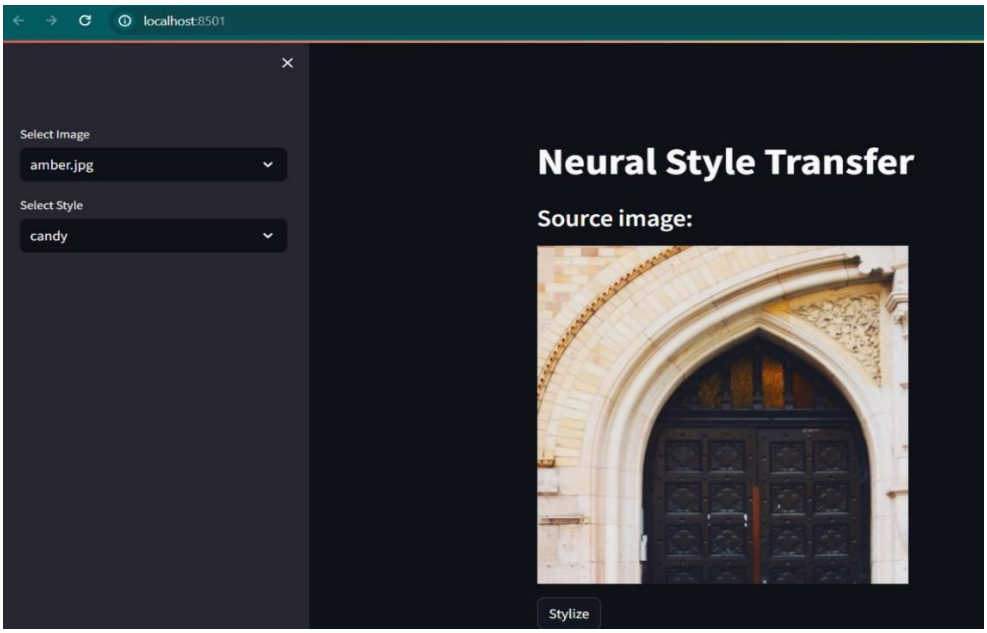


Figure 4.1.2 Example Source Image 1.

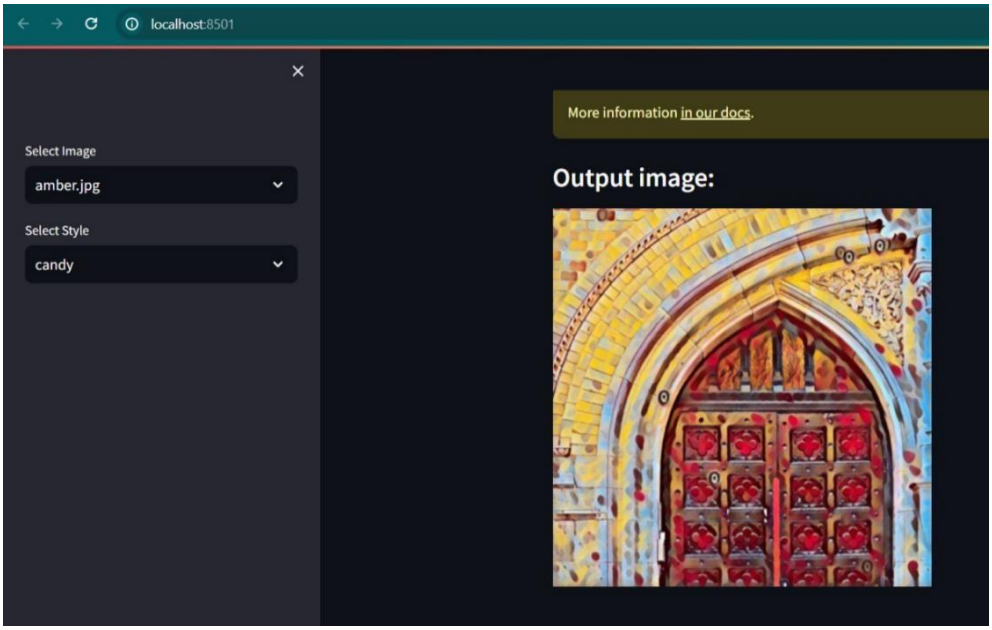


Figure 4.1.3 Example Output Image 1.

Result 2 :

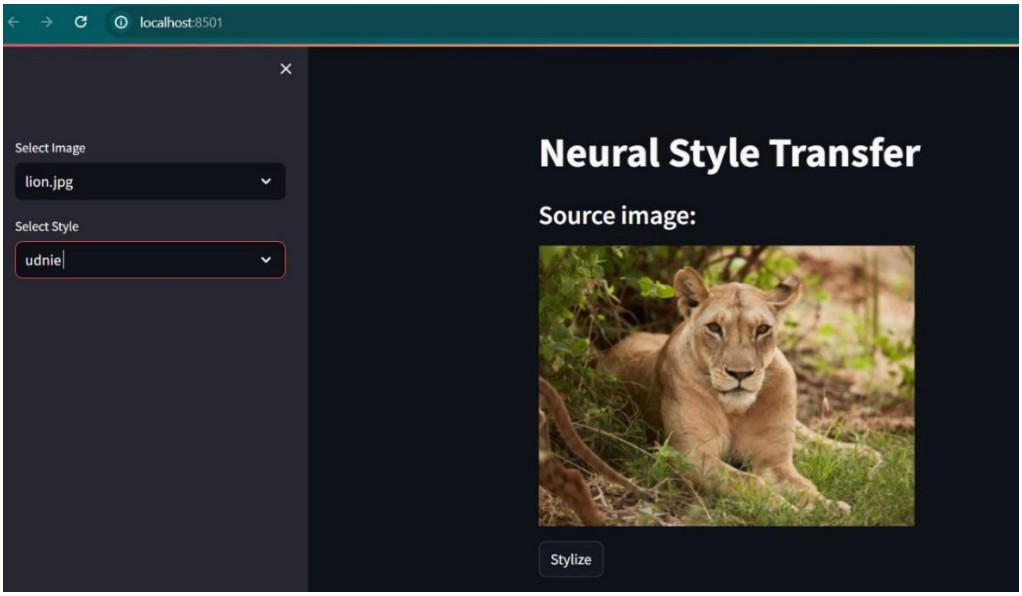


Figure 4.1.4 Example Source Image 2.

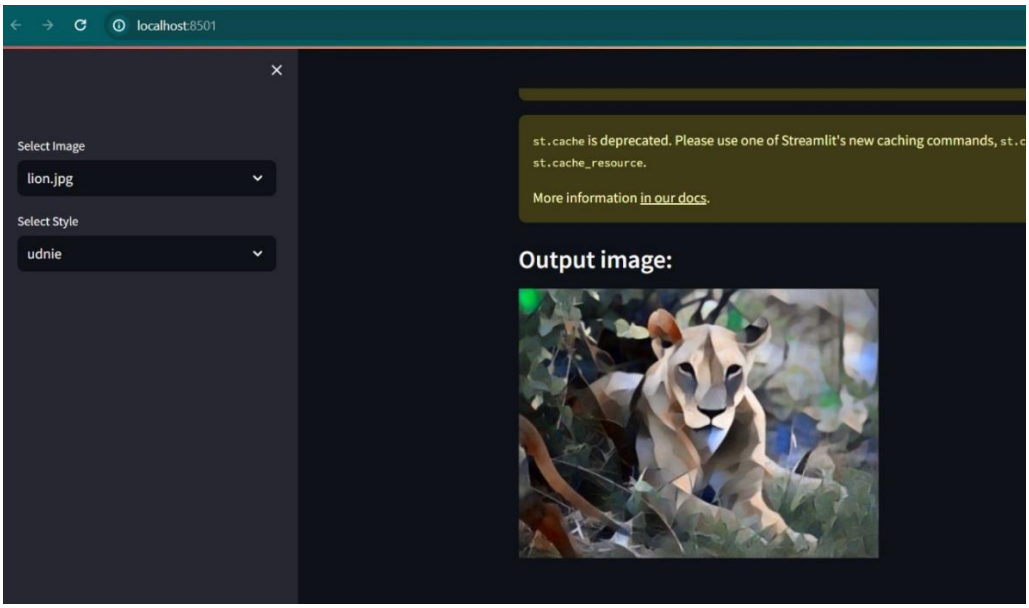


Figure 4.1.5 Example Output Image 3.

CHAPTER 5
CONCLUSION

CHAPTER 5
CONCLUSION

In conclusion, the Neural Style Transfer (NST) project has successfully achieved the complex and tricky task of merging artistic expression with technology. Through the implementation of NST algorithms and the better understandings of the concept, this project has showcased the transformative power of deep learning in reshaping visual content. By seamlessly blending the content of one image with the artistic style of another, the project has not only produced visually stunning results but also represented the artistic creation. The application of NST, originally introduced by Gatys et al., has been extended and optimized, ending with a user-friendly interface that empowers individuals to effortlessly engage in the artistic process. Looking ahead, the project's significance lies not only in its technical achievements but also in its broader implications for the intersection of technology and creative expression. As artificial intelligence continues to fill various aspects of our lives, the NST project stands as a proof to the coexistence of algorithms and artistry. The insights gained from this problem contribute to the evolving landscape of computer vision applications, offering a glimpse into the potential of AI to serve as a tool for unleashing human creativity. The NST project serves as a stepping stone for future innovations, inspiring further exploration in the realms of image synthesis, artistic collaboration, and the dynamic interplay between machine learning and human ingenuity. In essence, the Neural Style Transfer project not only refines and advances the state-of-the-art in image manipulation but also underscores the profound possibilities that arise when technology becomes a medium for creative expression, promoting a balance collaboration between the field of artificial intelligence and artistic imagination.

5. 1 Future Enhancement

Neural style transfer in future may be seen focusing on developing more advanced algorithms that can be capable of producing higher-quality stylized images with improved accuracy of image in content and style. And there might be a discovery that enables users to interactively control and manipulate the style transfer process in real-time. Below are few neural style transfer techniques may be able to apply to specific domains and industries, such as fashion, interior design, gaming, or virtual reality.

Improved Algorithms: Neural Style Transfer algorithms continue to evolve, aiming for better performance, efficiency, and quality of generated images. Future research may focus on developing more advanced architectures, or reinforcement learning techniques to enhance the quality and realism of stylized images.

Real-Time Applications: Current NST methods often require significant computational resources and time to process images. Future research may focus on developing real-time or near-real-time style transfer algorithms suitable for applications like video processing, live streaming, and augmented reality experiences.

Multimodal Style Transfer: While traditional NST focuses on transferring the style of a single image onto content, future research could explore multimodal style transfer, where multiple style references are combined or dynamically applied. This could enable more diverse and expressive artistic outputs by blending different artistic styles or creating entirely novel ones.

Cross-Domain Applications: NST may find applications beyond image processing, such as transferring artistic styles to other domains like text, audio, or 3D models. For instance, generating stylized text or applying music styles to audio signals could open up new paths for multimedia content generation.

Commercial Applications: NST techniques have commercial potential in various industries, including advertising, fashion, gaming, and interior design. Future applications may involve personalized style transfer services, product customization tools, or immersive virtual experiences that leverage NST for creative storytelling and branding.

Overall, the future scope of neural style transfer projects is vast and multidimensional, bringing advancements in algorithms, applications across different domains, interactive interfaces, and ethical considerations. With ongoing research and innovation, NST has the potential to revolutionize creative expression, design, and multimedia content creation in the years to come.

REFERENCES

REFERENCES

[1] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the ACM International Conference on Multimedia, pages 675–678. ACM, 2014.

[2] Nuthanakanti Bhaskar, K Bharath Kumar, Vootla Srisuma, A Vivekananda, B Archana, R Suhasini: A Quantitative Assessment of the Implementation of Neural Network Methods for Identifying and Recognizing Signs of Diabetic Retinopathy.

[3] Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576 (2015).

[4] A. Zisserman and K. Simonyan Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs], Sept. 2014. arXiv: 1409.1556.

[5] J. Deng, O. Russakovsky, J. Krause, H. Su, S. Ma, Z. Huang, S. Satheesh, A. Khosla, A. Karpathy, A. C. Berg, M. Bernstein, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575 [cs], Sept. 2014. arXiv: 1409.0575.

[6] Zheng, S., Romera-Paredes , V., Su, B., Vineet, S., Jayasumana, Z., Du, D., Huang, C.,Torr, P.H.: Conditional random fields as recurrent neural networks. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 1529–1537.

[7] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. CVPR (2015) .

[8] A Bathula, S Muhuri, S Merugu, SK Gupta: Designing Framework for Intrusion Detection in IoT Based on Spotted Hyena-Based ANN.

[9] Jun-Yan Zhu; Taesung Park; Phillip Isola; Alexei A. Efros : Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks, arXiv:1703.10593v7 [cs.CV] 24 Aug 2020.

[10] Justin Johnson, Alexandre Alahi, Li Fei-Fei : Perceptual Losses for Real-Time Style Transfer and Super-Resolution, arXiv:1603.08155v1 [cs.CV] 27 Mar 2016.

[11] Sanagala S Skandha, Ashish Kumar, Sampath Kumar K: Deep Learning Paradigm and Its Bias for Coronary Artery Wall Segmentation in Intravascular Ultrasound Scans: A Closer Look.

GITHUB LINK

https://github.com/PavanM07/Neural_Style_Transfer

Submission of Paper on ICTCI:

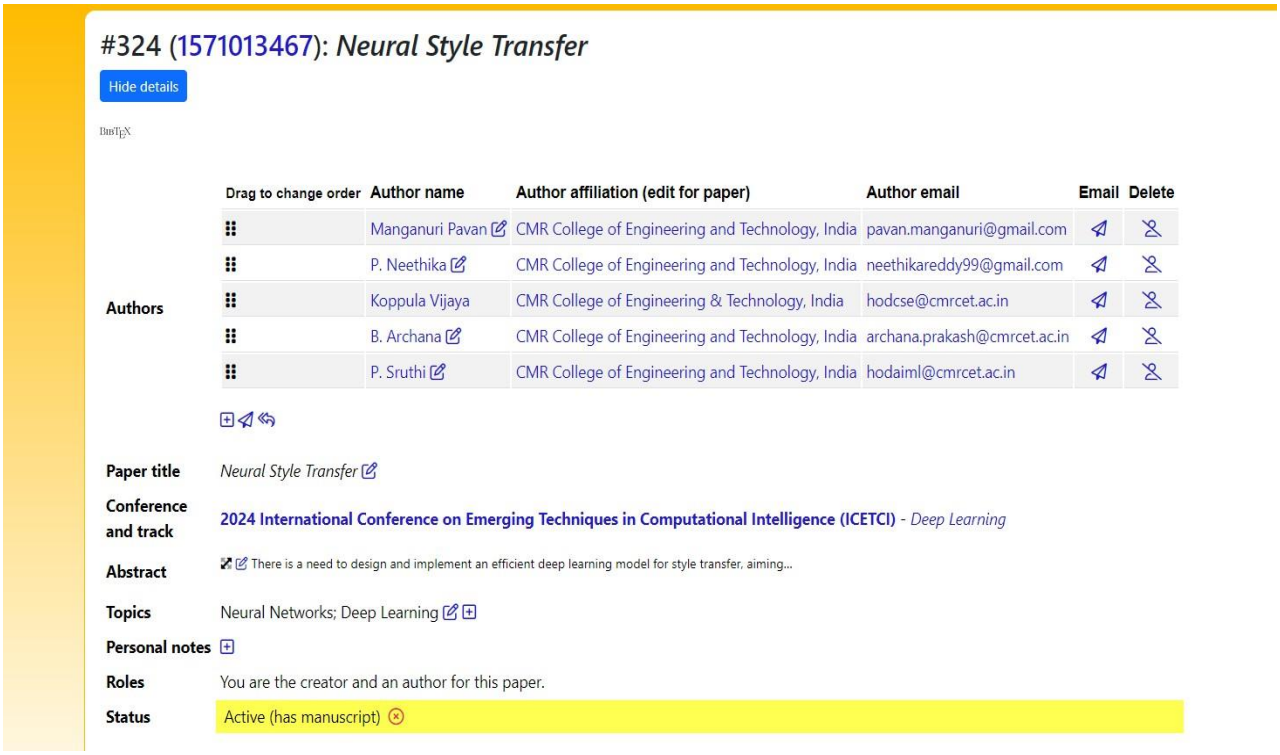


Fig : 6 Submission of Paper

Link of Submission of the Paper:

https://edas.info/showPaper.php?m=1571013467

Submitted Paper:

NEURAL STYLE TRANSFER

M. Pavan
U G Student, Department of Computer Science and Engineering, CMR College of Engineering & Technology Hyderabad, India
pavan.manganuri@gmail.com

P. Neethika
U G Student, Department of Computer Science and Engineering, CMR College of Engineering & Technology Hyderabad, India
neethikareddy99@gmail.com

S. Siva Skandha
Head of Department, Department of Computer Science and Engineering CMR College of Engineering & Technology Hyderabad, India
hodcse@cmrcet.ac.in

Abstract: ***The problem statement for the proposed project, Neural Style Transfer, is to design and implement an efficient deep learning model for style transfer, aiming to combine the artistic style of one image with the content of another image (content image). We came with a solution to design and implement an efficient deep learning model for style transfer. This model takes a content image 'C' and a style image 'S' and merge both images to produce a new image that contains the content of image 'C' and the style of image 'S'. This can be implemented by neural style transferring using deep learning which involves transferring the artistic style of one image with the content of another image, resulting in creative and visually appealing images as outputs. It will be implemented through Convolutional Neural Networks (CNNs), utilizing pre-trained models. The procedure entails obtaining characteristics from both content and style images through pre-trained network, preprocessing of images, optimizing a generated image to minimize content and style loss and iteratively refining the results.***

Keywords— Deep learning model for Style transfer, Feed forward Image Transformation, CNN's, pre-trained networks.

I. INTRODUCTION

In the world of digital artistry, our Neural Style Transfer project pushes the boundaries of creativity and innovation. Neural Style Transfer is the combination of art and technology, which provides a way to transform ordinary images into visually stunning artworks. This technique uses the power of deep learning, specifically convolutional neural networks (CNNs), to seamlessly blend the content of one image with the artistic style of another. The project aims to explore and implement neural style transfer algorithms, to create a fusion of content and style. This project not only explores the intersection of technology and art but also opens new way of personalized digital content creation. This project's main goals are to have a thorough understanding of NST, from theory to practical. The process involves employing pretrained models, defining loss functions, and optimizing the generated image to achieve a neat mixture of content and style. Additionally, it must focus is on user, with the development of this interface that makes users to upload images, select artistic styles, and observe real-time transformations. This project aims to make the artistic potential of NST accessible to a broader audience[10]. The Stochastic Gradient Descent (SGD) algorithm is the central component of the project. SGD is a machine learning model optimization variant of the Gradient Descent technique. It solves the classic Gradient Descent methods computational inefficiencies while working with big datasets in machine learning applications.

II. RELATEDWORKS

A. Deep Image Representations

The comes about displayed underneath were created on the premise of the VGG arrange [1], which was prepared to perform protest acknowledgment and localisation [2] and is portrayed extensively within the unique work [1]. We utilized the function area given by using a standardized version of the sixteen convolutional and 5 pooling layers of the nineteen-layer VGG community. We normalized the arrange by using scaling the weights such that the merciless enactment of every convolutional channel over pix and positions is break even with to one. Such re-scaling may be done for the VGG arrange without converting its yield, as it carries because it had been correcting direct actuation capacities and no normalization or pooling over function maps. We don't use any of the completely related layers. The display is publicly handy and may be investigated within the caffe-framework [7]. For picture combo we discovered that supplanting the maximum pooling operation via regular pooling yields particularly extra enticing comes approximately, that is why the pics appeared were produced with normal pooling.

B. CNN in Deep Learning

A well-known neural network may additionally have an input layer, output layer, and an hidden layers. CNNs are stimulated by using the structure of the mind. just like a neuron within the mind approaches and transmits records in some unspecified time in the future of the body, synthetic neurons or nodes in convolutional neural networks take inputs, strategize them and also send the final result of the output as output. Photos are provided as an introduction. The input method takes image pixels as input for the image of the array. CNN may have many hidden layers that calculate the extracted features of the image. This can include convolutions, pooling, rectified linear devices, and entire layers. Convolution is an important layer for removing the input image. The link layer completely separates and defines the product within the output layer. A convolutional neural network is a feeder network in which information flows along a path from input to output. Just as artificial neural networks (ANN) are inspired by biology, this is also the inspiration for neural networks (CNN)[9]. The cortex, located in the brain, has alternating layers of simple and complex cells that inspire their structure. There are many versions of CNN architectures; However, popularly there are convolution and pooling (or subsampling) layers that can be divided into groups[10].

C. Feed Forward Image Transformation

In recent times, a wide assortment of feed-forward image transformations have been illuminated by preparing profound convolutional neural systems with per-pixel misfortune(loss) functions. Semantic division strategies [4] create thick scene labels by running a organize in a fullyconvolutional way over an input picture, training with a per-pixel classification loss. [3] moves past per-pixel losses by surrounding CRF deduction as a repetitive layer prepared together with the rest of the arrange. The design of our change systems are propelled by [4] , which utilize in-network downsampling to diminish the spatial degree of feature maps taken after by in-network upsampling to create the ultimate yield image. Later strategies for profundity and surface typical estimation are comparative in that they change a color input picture into a geometrically meaningful yield picture employing a feedforward convolutional arrange prepared with perpixel relapse or classification [5] losses. A few strategies move past per-pixel losses by penalizing picture slopes or employing a CRF loss layer to implement neighborhood consistency within the yield picture. In a feed-forward demonstrate is prepared employing a per-pixel misfortune to convert grayscale pictures to color.

D. Style Transfer

Gatys et al. By combining reduction and reconstruction, [6] can be an additional tool for the ability to

extract image content from previous communication to distinguish between other images. Stylish together; A similar method is To deal with the deficiencies of in step with-pixel losses and empower our loss features to superior diploma perceptual and semantic contrasts used for texture [6]. Their strategy produces excellent outcomes, but is computationally expensive because each back and forth through a preliminary discussion. To overcome this computational burden, we train feedforward networks to quickly solve the optimization problem.

III. METHODOLOGY

Our framework comprises of two components: an picture transformation network f_W and a loss network ϕ, which is utilized to characterize different loss functions $l_1, \ldots, l_k$. The picture transformation network could be a leftover convolutional neural network parameterized by weights W; changes over the input images x into output images y by mapping $\hat{y} = f_W(x)$. Each loss function calculates a scalar esteem $l_i(\hat{y}, y_i)$ that measures the distinction between the output picture $\hat{y}$ and a target picture y_i. To minimize the weighted combination of loss functions, the image transformation network is trained using stochastic gradient descent algorithm :

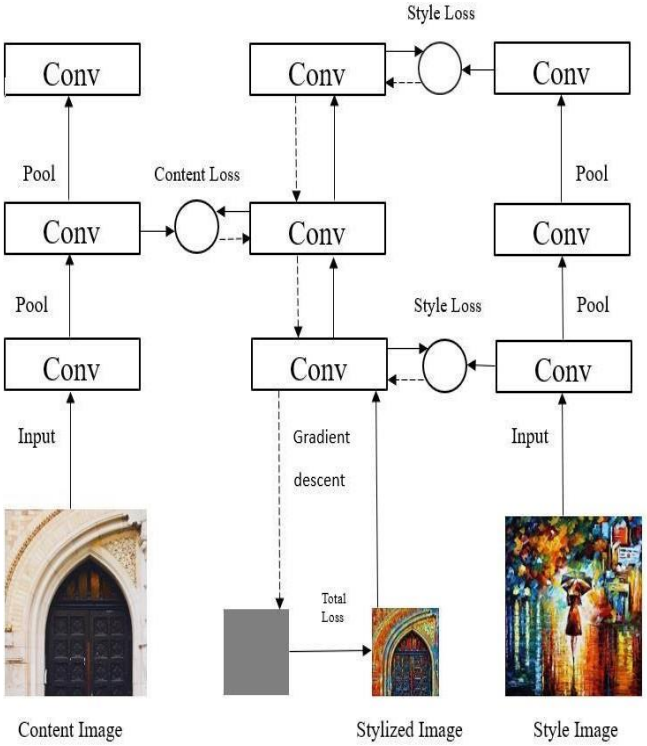
$$W^* = \arg\min E_{x,\{yi\}} [\sum_{i=10} \lambda_i l_i(f_W(x), y_i)]$$

between images, we draw motivation from recent work on developing snap shots via optimization. The center idea of these techniques is that convolutional neural networks a lready educated for photo category have already discovered to encode the perceptual and semantic records that we need to degree in our loss features. on this manner, we utilize a network ϕ already skilled for image category as a settled loss network to construct our loss functions. At that point our deep convolutional neural network is trained utilizing loss capabilities, which might be furthermore deep convolutional neural networks. The loss network ϕ is utilized to construct a feature loss l_{feat}^{ϕ} and a style loss, l_{style}^{ϕ} which measures contrasts in content and style between pictures. For each input picture x, we have a content target y_c and a style target y_s. For style transfer, the target content y_c is the input picture x, and the output picture $\hat{y}$ must combine the content of $x = y_c$ with the style of y_s; we train a network consistent with target fashion. In singleimage super-resolution, the enter image x is a input, the goal content y_c is the real excessive-resolution image, and the style reconstruction loss isn't utilized. We train a network with the awesome resolution aspect.

$$L_{CONTENT}(\vec{p},\vec{x},l) = \frac{1}{2}\sum_{i,j}(F_{ij}^{l} - P_{ij}^{l})^2 \quad \ldots\ldots (1)$$

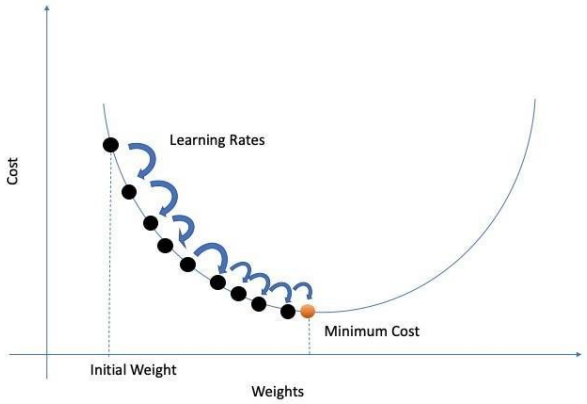
$$L_{STYLE}(\vec{a},\vec{x}) = \sum_{l=0}^{L} w_l E_l \quad \ldots.. (2)$$

$$L_{TOTAL} = \alpha L_{CONTENT} + \beta L_{STYLE} \quad \ldots\ldots(3)$$



III. ALGORITHM

Stochastic gradient descent (often abbreviated as SGD) is a type of gradient descent, a technique that functions well with optimization (e.g. variance or variance). It can be considered an approximation of gradient descent, as it replaces the actual gradient (e.g. from all statistical data) with its estimate (e.g. from a random subset of data). Particularly in hyperdimensional optimization problems, this reduces the computational burden and increases the speed of iterations in exchange for reduced convergence costs[8]. Error functions are not often as easy as an ordinary parabola. most customarily they have plenty of hills and valleys, just like the characteristic pictured right here. on this graph, if real gradient descent started on the left facet of the graph, it might stop on the left valley because no matter which route you travel from this factor, you need to travel upwards. This factor is known as a nearby minimal. but, there exists another factor inside the graph that is decrease. the lowest point in the complete graph is the global minimal, that's what stochastic gradient descent tries to discover. The following is the process of Stochastic Gradient Descent algorithm: First shuffle the dataset randomly and choose a data point randomly and cycle through all the elements. Now cycle on all weights or parameters and adjust the current weight or parameter under the derivative of the cost or loss function. Calculate the new Gradient with the new value of the parameter or weights. Repeat all the above steps until convergence or global minima is achieved.



IV. DATASET

Our adaptive network model was trained on the Microsoft COCO dataset. We update every 80,000 training images to 256×256 and train our network with a batch length of 4 to 40,000 iterations, achieving approximately twice the throughput of the training data. We use SGD with a gaining knowledge of rate of 1 × 10-3. The output photographs are regularized with a complete variational regularization with a strength among 1 × 10-6 and 1 × 10-4, decided on by using cross-validation by style aim. We do now not use weight loss or dropouts because the version does now not overfit in two epochs.

Candy

Mosaic

Rainy

Dye

V. RESULTS

Through our model we can generate images that blends the representations of content and style from two different images. Using Convolutional Neural Networks the representations of style and content are well separable. So, we can manipulate both the representations to generate new artistic and meaningful images.

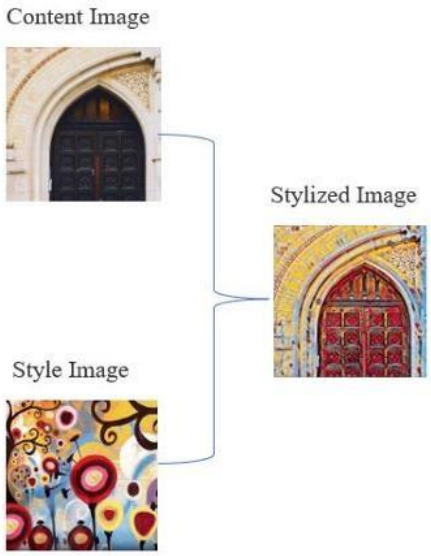


Figure 3. Example1

In the above example the model is applied in a fully - convolutional manner to generate stylized images. It is clearly visible that the network is aware of the content of the images. The monument in the content image is clearly recognizable in the transformed or stylized image but the style of the style image is applied.

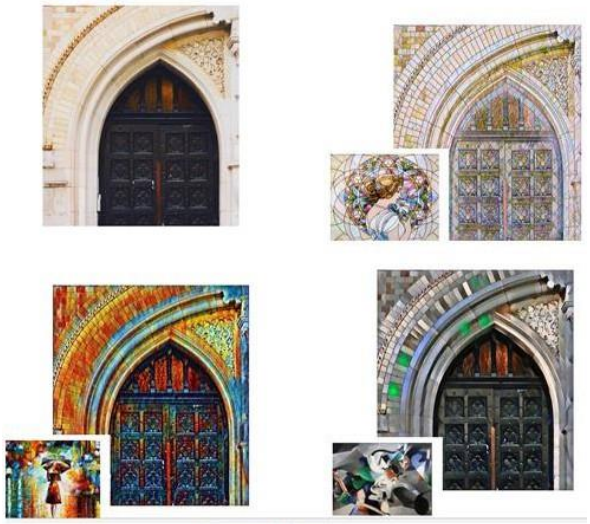


Figure 4. Results

VII. CONCLUSION

In conclusion, the Neural Style Transfer (NST) project has successfully achieved the complex and tricky task of merging artistic expression with technology. Through the implementation of NST algorithms and the better understandings of the concept, this project has showcased the transformative power of deep learning in reshaping visual content. By seamlessly blending the content of one image with the artistic style of another, the project has not only produced visually stunning results but also represented the artistic creation. The application of NST, originally introduced by Gatys et al., has been extended and optimized, ending with a user-friendly interface that empowers individuals to effortlessly engage in the artistic process. Looking ahead, the project's significance lies not only in its technical achievements but also in its broader implications for the intersection of technology and creative expression. As artificial intelligence continues to fill various aspects of our lives, the NST project stands as a proof to the coexistence of algorithms and artistry. The insights gained from this problem contribute to the evolving landscape of computer vision applications, offering a glimpse into the potential of AI to serve as a tool for unleashing human creativity. The NST project serves as a stepping stone for future innovations, inspiring further exploration in the realms of image synthesis, artistic collaboration, and the dynamic interplay between machine learning and human ingenuity. In essence, the Neural Style Transfer project not only refines and advances the state-of-the-art in image manipulation but also underscores the profound possibilities that arise when technology becomes a medium for creative expression, promoting a balance collaboration between the field of artificial intelligence and artistic imagination.

VIII. REFERENCES

[1] A. Zisserman and K. Simonyan Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs], Sept. 2014. arXiv: 1409.1556.

[2] J. Deng, O. Russakovsky, J. Krause, H. Su, S. Ma, Z. Huang, S. Satheesh, A. Khosla, A. Karpathy, A. C. Berg, M. Bernstein, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575 [cs], Sept. 2014. arXiv: 1409.0575.

[3] Zheng, S., Romera-Paredes , V., Su, B., Vineet, S., Jayasumana, Z., Du, D., Huang, C., Torr, P.H.: Conditional random fields as recurrent neural networks. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 1529–1537.

[4] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. CVPR (2015) .

[5] Gupta, A, Fouhey, D., Wang, X.,: Designing deep networks for surface normal estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 539–547.

[6] Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576 (2015).

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the ACM International Conference on Multimedia, pages 675–678. ACM, 2014.

[8] Sanagala S Skandha, Ashish Kumar, Sampath Kumar K: Deep Learning Paradigm and Its Bias for Coronary Artery Wall Segmentation in Intravascular Ultrasound Scans: A Closer Look [9] A Bathula, S Muhuri, S Merugu, SK Gupta: Designing Framework for Intrusion Detection in IoT Based on Spotted Hyena-Based ANN.

[10] Nuthanakanti Bhaskar, K Bharath Kumar, Vootla Srisuma, A Vivekananda, B Archana, R Suhasini: A Quantitative Assessment of the Implementation of Neural Network Methods for Identifying and Recognizing Signs of Diabetic Retinopathy.