Ain Shams University
Faculty of Computer & Information Sciences
Scientific Computing Department

Image Style Transfer

July 2022

Ain Shams University
Faculty of Computer & Information Sciences
Scientific Computing Department

Image Style Transfer

Thisdocumentationsubmittedasrequiredforthedegreeofbachelor’s in computer and Information Sciences.

By

<u>Basma Salah El-Din Sayed</u>
(Scientific Computing Department)
<u>Omnia Mohamed Sayed</u>
(Scientific Computing Department)
<u>Ahmed Elsayed Ahmed</u>
(Scientific Computing Department)
<u>Basel Amr Mohamed</u>
(Scientific Computing Department)
<u>Abdelrahman Ashraf</u>
(Scientific Computing Department)

Under Supervision of

[Dr/ Maryam Nabil Al-Berry]
[Assistant Professor at Scientific Computing Department],
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

[TA/ Menna Allah EL-Kholy]
[Teaching Assistant at Scientific Computing Department],
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

Acknowledgement

All praise and thanks to ALLAH, who provided us the ability to complete this work. We hope to accept this work from us. We are grateful of our parents and family who are always providing help and support throughout the whole years of study. We hope we can give that back to them.

We also offer our sincerest gratitude to our supervisors, Associate **Prof. Dr.** ***Maryam Nabil Al-Berry***, and **T.A.** ***Menna Allah EL-Kholy*** who have supported us throughout our thesis with their patience, knowledge and experience.

Finally, We would thank our friends and all people who gave us support and encouragement.

Abstract

This paper introduces a deep-learning approach to photographic style transfer that handles a large variety of image content while faithfully transferring the reference style Rendering the semantic content of an image in different styles is a difficult image processing task. Arguably, a major limiting factor for previous approaches has been the lack of image representations that explicitly represent semantic information and, thus, allow to separate image content from style.

Here we use image representations derived from Convolution Neural Networks optimized for object recognition, which make high level image information explicit. We introduce A Neural Algorithm of Artistic Style that can separate and recombine the image content and style of natural images. The algorithm allows us to produce new images of high perceptual quality that combine the content of an arbitrary photograph with the appearance of numerous well-known artworks. Our results provide new insights into the deep image representations learned by Convolution Neural Networks and demonstrate their potential for high level image synthesis and manipulation.

Extensive research in neural style transfer methods has shown that the correlation between features extracted by a pre-trained VGG network has remarkable ability to capture the visual style of an image.

Table of Contents

List of Figures

List of Abbreviations:

- **Avgpool:** Average pool layer
- **CNN**:ConvolutionNeuralNetwork
- **FCL:** Fully connected layer
- **MRF:** Markov Random Field
- **ReLU**: Rectified Linear Unit
- **VGG**:VisualGeometryGroup

Chapter1: Introduction

In this chapter, we will present an overview on the topic and explain the problem and the motives that made us apply to do this project that provide services that benefit many people and also we clarify the goal that we sought to achieve, and in the end we will display the document outlines to explain briefly the content of rest of the chapters.

1.1 Problem definition

Image style transfer is an artificial system based on the Deep Neural Network to generate artistic images. This system aims to map a content image into the style of a different reference image (taking a content image and a style image as input, and outputting an image that has the content of the content image and the style of the style image). It has received substantial attention, particularly with the introduction of neural style transfer algorithms based on deep networks.

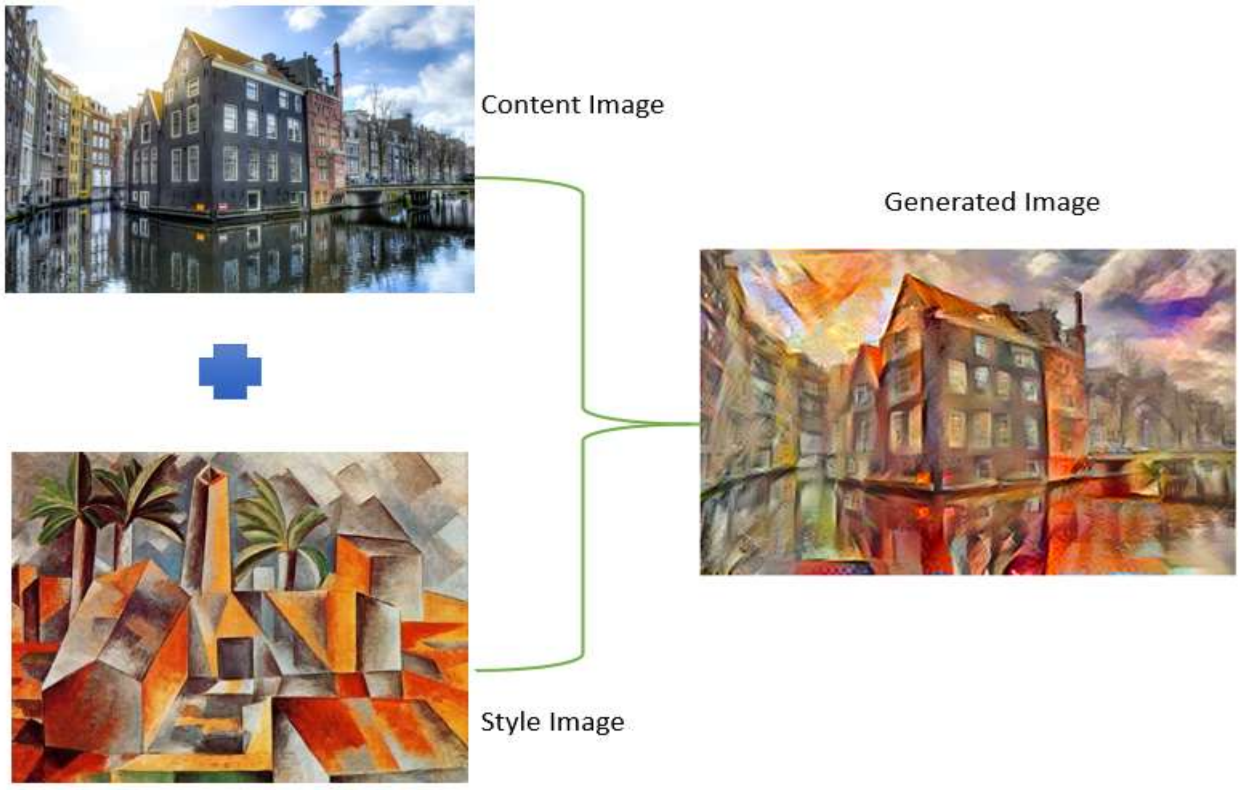


Figure 1.1: styling image

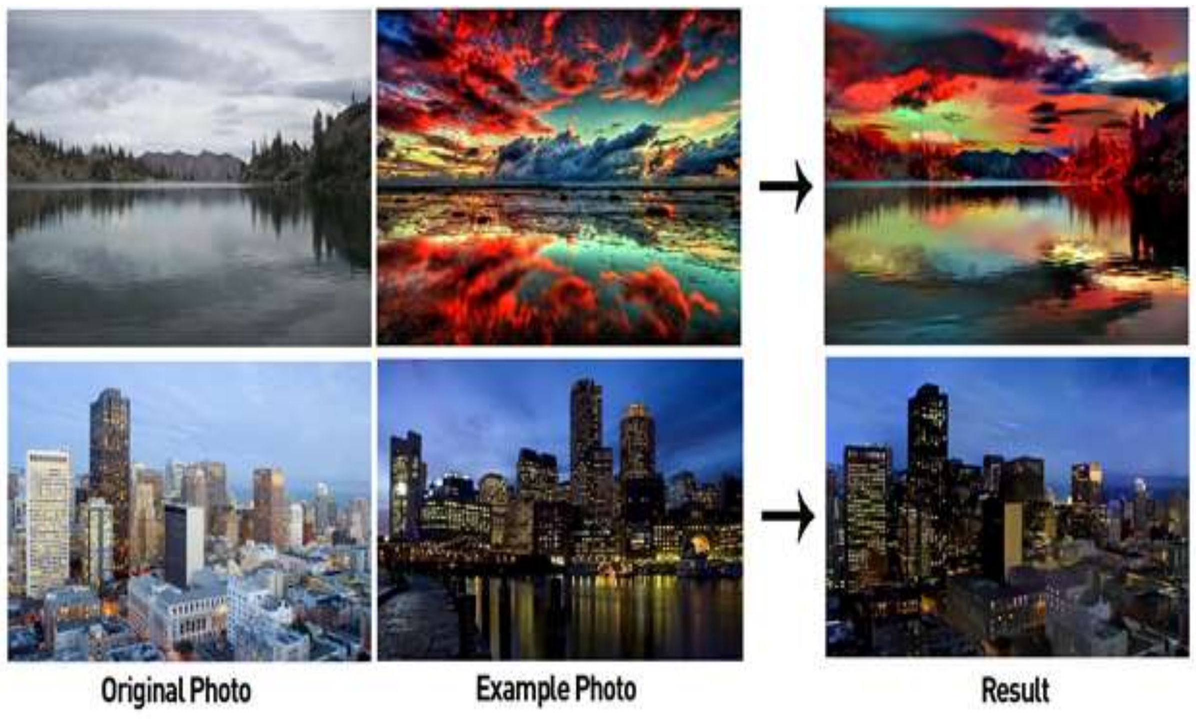


Figure 1.2: styling image

1.2Applications and prospects

Neural Style Transfer plays an important role not only in the field of art, but also in other areas. Popular photo-editing software for young people, allows you to change the style of your photos in one step.

Through the integration of computer science and animation art, it can make up for the shortcomings of the original hand-painting, and at the same time can stimulate the new creativity of animation special effects design and obtain new results. The style transfer algorithm can quickly and effectively generate a variety of image styles, greatly improving efficiency, saving manpower and material resources, and creating unique and novel animation special effects designs.

Provide strong technical support for animation production from an artistic perspective, and further expand the scope of animation special effects design.

And a new idea for image generation is proposed. The image style transfer algorithm can quickly generate a variety of images through a variety of effects, which can be applied to film post-processing, poster design, artistic creation, conceptual design, and so on. Transfer images with different artistic style attributes, and perform color correction on the transfer results. Experiments show that the implemented algorithm can obtain a more ideal animation pattern special effect design according to actual needs.

Video post processing. In future studies, researchers can extend the style model to make the extraction of arbitrary styles more accurate, and put more research into video style transfer. We should gradually extend the style transfer to more engineering fields for the benefit of mankind.

So, some applications:

- Photo and video editing.
- Commercial art.
- Gaming: video game is made to look like another with style transfer.
- Virtual reality: applied in VR

1.3 Motivation

We want to use machine learning programs to transfer daily pictures to artistic style images by using a style image for obtaining the transferred style and a content image which we want to transfer the style onto. This project can demonstrate how well machine learning techniques perform on image processing and how artificial intelligence can, to some extent, achieve considerable success in the field of art and creativity.

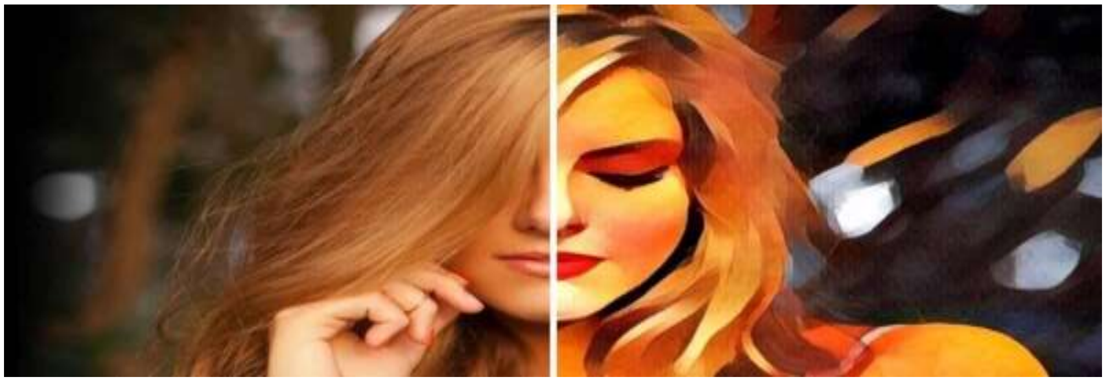


Figure 1.3: style transfer image

1.4 Objective

The goal of style transfer is to isolate the style of an artistic image (the style image) and blend or adapt it to the content image to produce a new image called Target image.

1.5 Time plan

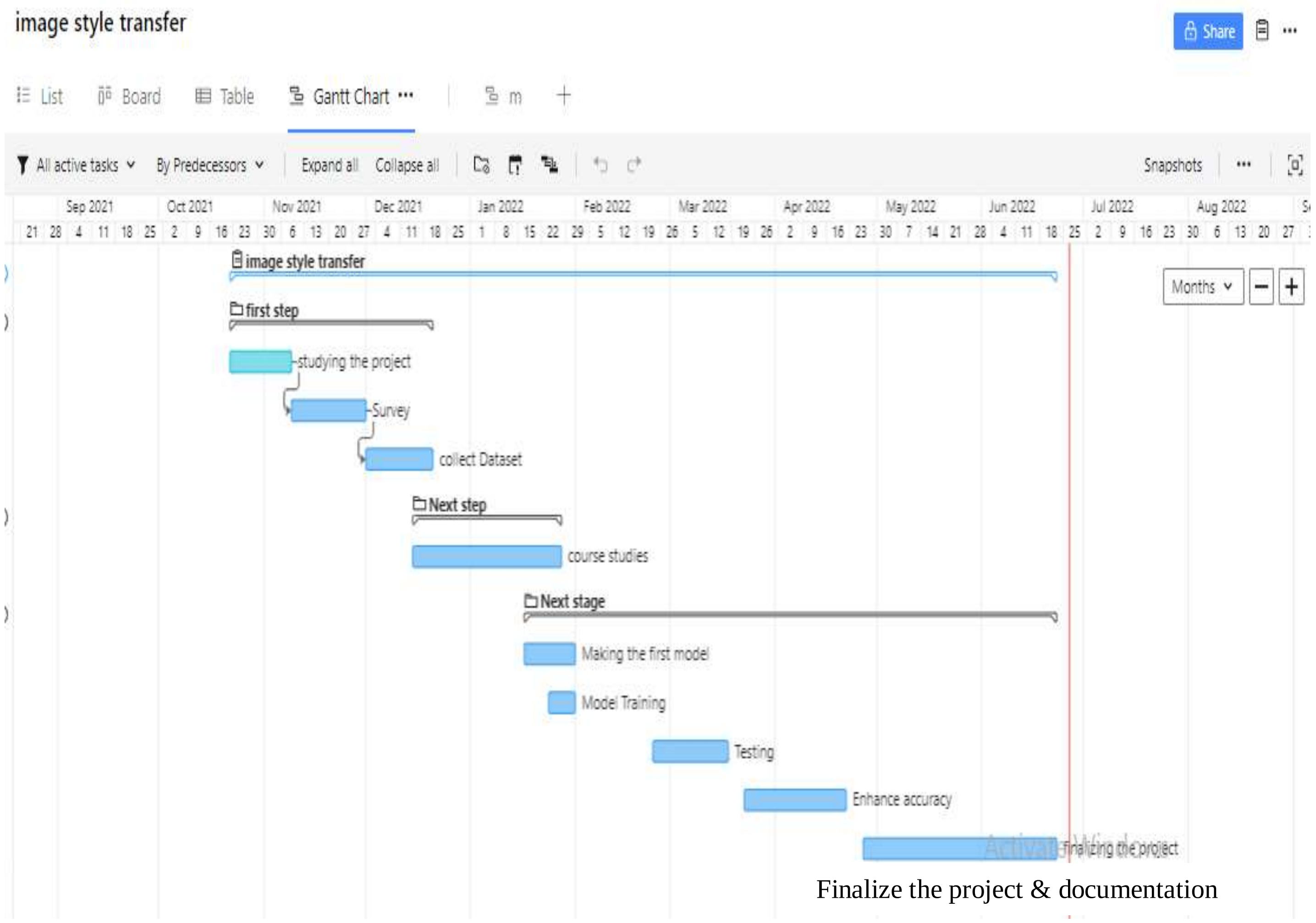


Figure 1.4: Time plan

1.6 Documentation Outline

The rest of the document is divided into six main Chapters and a separate Chapter for conclusions and future work at the end of the document.

Chapter 2: presents a detailed overview on the main concepts and theory. It gives summary for the history developed applications which use the technology.

Chapter 3: reviews the system architecture and description of each part and most of the recent and famous algorithms that use to apply each one.

Chapters 4: The proposed work of the document is presented in.

Chapter 5: present the process to test our system and result of it.

Chapter6: Give a user manual to handle the system.

Chapter2: Background

Before starting in any project or in any topic, we must see what researchers have done. Then it’s important to make survey on papers which deal with our interests to see what techniques and Methods that the researchers used to deal with our project to our topic, so we have read many papers that deal with style transfer.

2.1 related work

In the August of 2015, a paper came out titled ‘A Neural Algorithm of Artistic Style’. Back then, and a few months later, an app called Prisma was released and, people just went crazy about it. In case you don’t know what Prisma is, it is basically an app that allows you to apply painting styles of famous painters to your own photos, and the results are quite visually pleasing. It isn’t like Instagram filters, where some sort of transformation is applied to the picture only in the color space. It is much more elaborate, and the results are even more interesting.
Style transfer can be traced back to the research of image texture synthesis. All previous papers on image texture were manually modeled, and the most important idea used was that texture can be described by the statistical model of image local features. At the same time, the researchers also tried the transfer of oil painting style and head portrait style transfer.
With the development of computer graphics, deep learning has been greatly developed because it can be used to train object recognition models. CNN can extract features. In addition to image recognition and image classification, CNN is also used for style transfer.

2.1.1 Style transfer without neural network

Before the rise of CNN, researchers tried to transfer styles without relying on neural network algorithms. Transfer the paintbrush of a famous artist to the photo to be rendered to get a good portrait. Portrait drawing is an extension of image analogy, but its data is difficult to obtain in our practice, which has caused certain limitations to the extension research. Many objects in nature have similar textures. We can extract the texture from the sample and regenerate a large amount of image data. The texture synthesis algorithm can be extended to transfer the texture obtained from different targets and re-render the image. Or combine existing textures to create new ones.

2.1.2 Neural Style Transfer

In 2012, deep learning gradually developed and rapidly gained wide attention from researchers. Neural networks can automatically extract useful features after a certain amount of training, rather than simply dividing objects into small pieces.

According to the way the image is optimized, there are two main types of approaches in style transfer based on deep neural networks model, global approaches based on the Gram matrix, and local approaches based on patch matching. Li and Wand [4] presented a combination of generative Markov Random Field (MRF) models for image style transfer, which can gives effective transfer results and keep the information of the content image

2.2 Observations

We will abstract comparison of all previous papers in the following table:

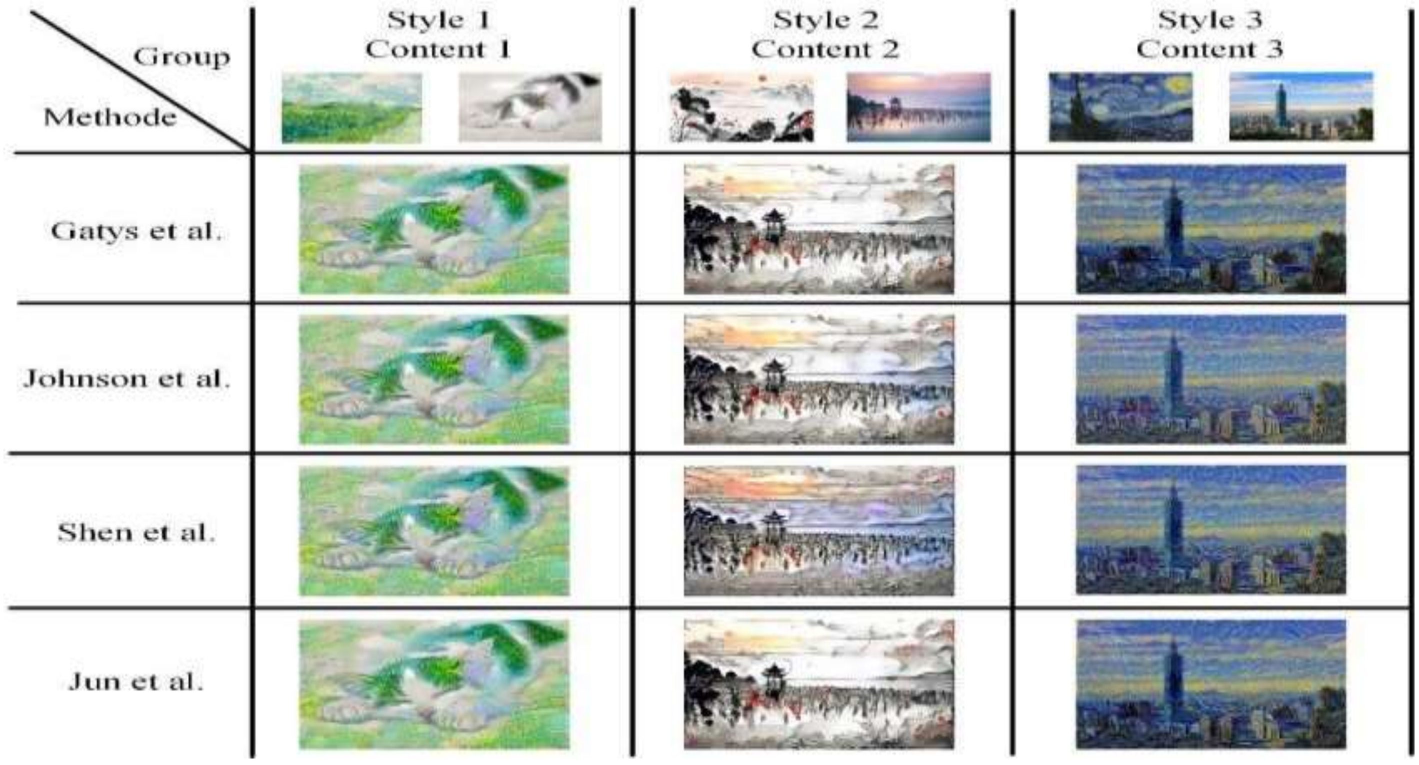
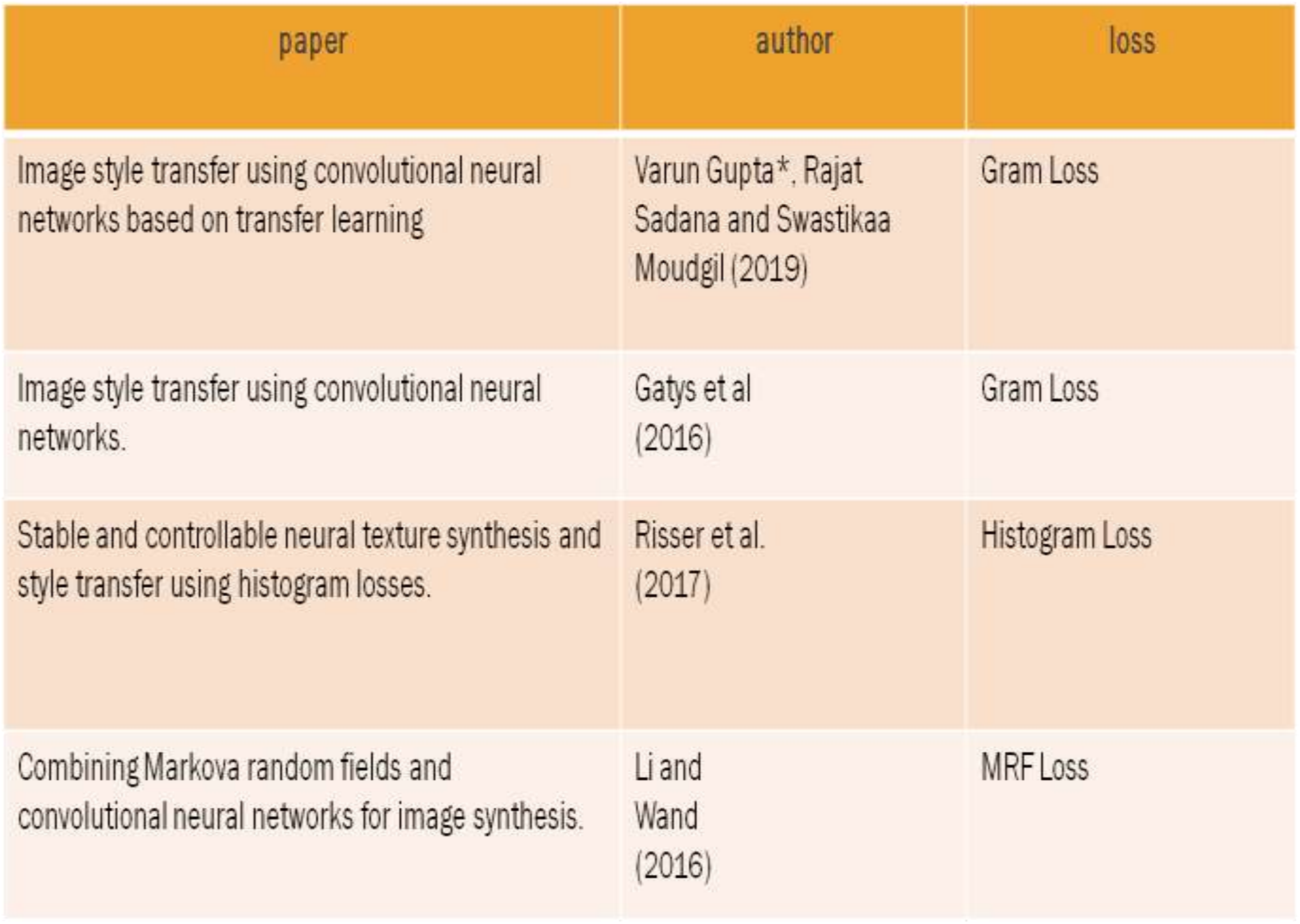


Figure 2.1:comparison



paper	author	loss
Image style transfer using convolutional neural networks based on transfer learning	Varun Gupta*, Rajat Sadana and Swastikaa Moudgil (2019)	Gram Loss
Image style transfer using convolutional neural networks.	Gatys et al (2016)	Gram Loss
Stable and controllable neural texture synthesis and style transfer using histogram losses.	Risser et al. (2017)	Histogram Loss
Combining Markova random fields and convolutional neural networks for image synthesis.	Li and Wand (2016)	MRF Loss

Figure 2.2:comparison

Chapter3: System Architecture

In this chapter we will explain the system architecture for image style transfer system, first of all the system contain many stages, each one has many technique and algorithm that can been applied, so we will describe each stage and it's benefit and the most used and famous algorithms can have been applied.

3.1 General System Architecture

Image representations in a Convolutional Neural Network (CNN). A given input image is represented as a set of filtered images at each processing stage in the CNN. While the number of different filters increases along the processing hierarchy, the size of the filtered images is reduced by some down sampling mechanism (e.g., max pooling) leading to a decrease in the total number of units per layer of the network.

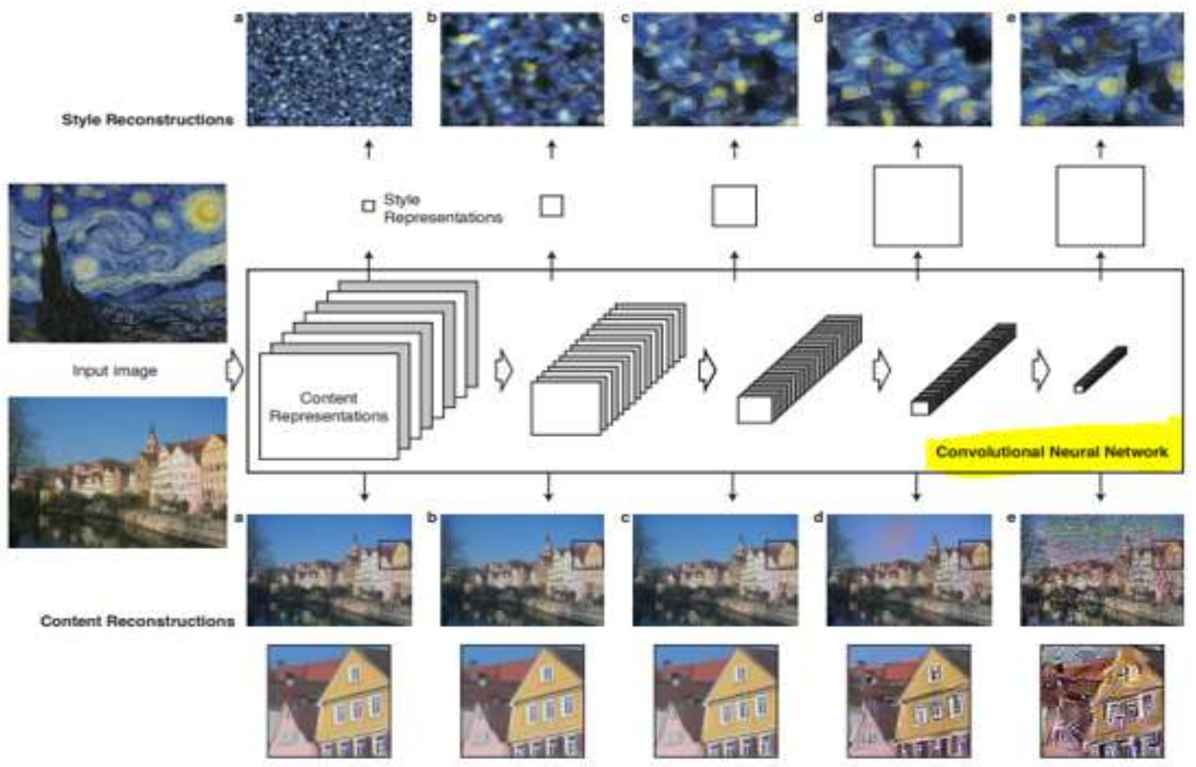
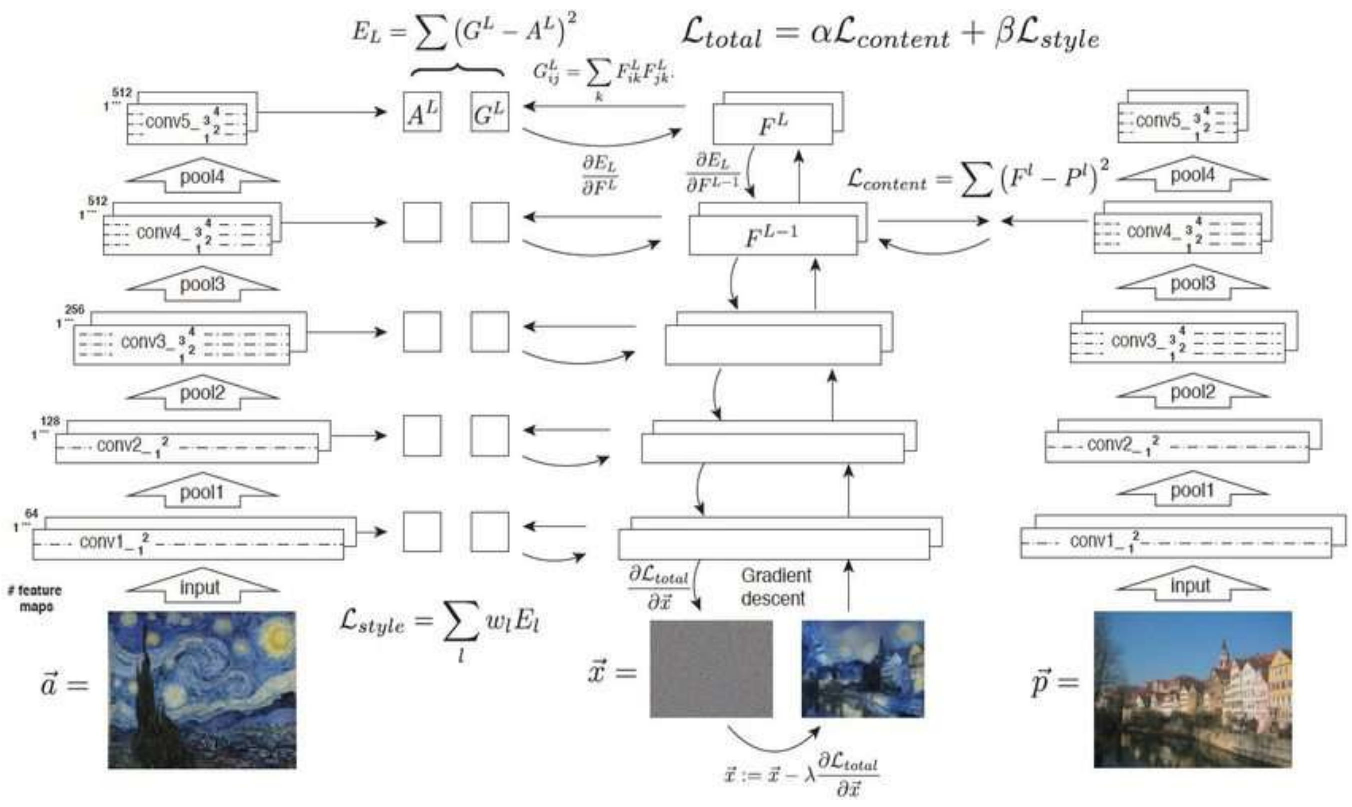


Image representations in a Convolutional Neural Network (CNN)

Fig.3.1: systemarchitecture



$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik}F^L_{jk}$
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
conv5_3
conv4_3
conv3_3
conv2_1
conv1_1
pool4
pool3
pool2
pool1
input
# feature maps
512
512
256
128
64
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

Fig.3.2: system architecture

Inputs:

- First content and style features are extracted and stored.
- The style image ~a is passed through the network and its style representation *Al* on all layers included are computed and stored.
- The content image ~p is passed through the network and the content representation *Pl* in one layer is stored.
- Then a random white noise image ~x is passed through the network and its style features *Gl* and content features *Fl* are computed.

3.2 Data Acquisition

We used two data sets:

1- Best Artworks of All Time:
Collection of artworks of the 50 most influential artists of all time.

2- Google Scraped Image Dataset:

The images belong typically to 4 classes:

- Art & Culture
- Architecture
- Food and Drinks
- Travel and Adventure

3.3 Pre-processing

It refers to the transformations applied to our data before feeding it into our models, because it directly affects the ability of our model to learn. The pre-processing techniques we had applied in two steps filtering and resampling.

To make an image compatible with the model, the image has to be preprocessed. Using the torch.transforms() we will perform some of the basic preprocessing that will include the following steps:

Resize: resize all the image to 300 x 300
To Tensor: to convert the image into a tensor

And also Normalize the tensor with mean (0.5, 0.5, 0.5) and standard deviation of (0.5, 0.5, 0.5) for the pretrained vgg16 model. But don't forget to convert it back to its original scale. So, define a function that loads the image using the PIL library and preprocess it. Add an extra dimension at 0th index, using unsqueeze () for batch size and then load it to the device and return it.

3.4 Classification

Convolution Neural Network (CNN)

The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolutional networks are a specialized type of neural network that uses convolution in place of general matrix multiplication in at least one of their layers.

A convolutional neural network consists of an input layer, hidden layers, and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically, this includes a layer that performs a dot product of the convolution kernel with the layer's input matrix. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, dropout layers, and activation functions.

We used five different models of CNN will be discussed in the following chapter.

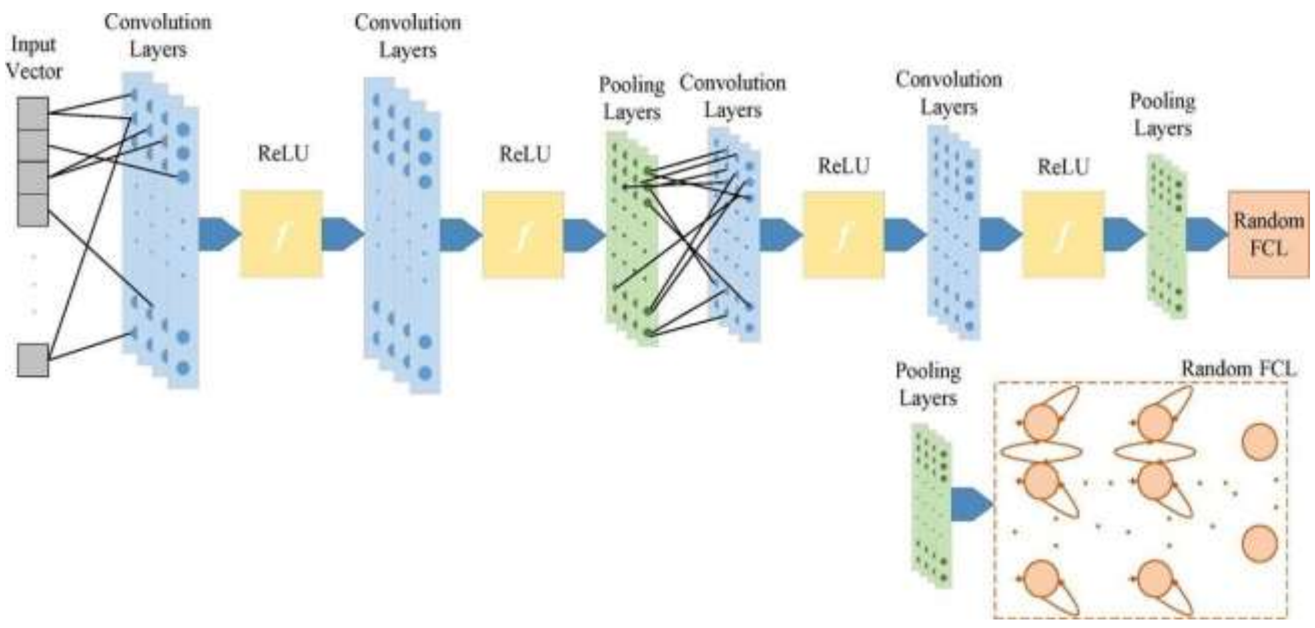


Figure 3.3: CNN architecture

Convolution layer:

The convolution layer is sometimes called the feature extractor layer because features of the image are get extracted within this layer. First of all, a part of the image is connected to the Convo layer to perform the convolution operation as we saw earlier and calculate the dot product between the receptive field (it is a local region of the input image that has the same size as that of filter) and the filter. The result of the operation is a single integer of the output volume. Then we slide the filter over the next receptive field of the same input image by a Stride and do the same operation again. We will repeat the same.
Process again and again until we go through the whole image. The output will be the input for the next layer.

SoftMax activation function:

SoftMax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

The most common use of the SoftMax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the SoftMax function is interpreted as the probability of membership for each class.

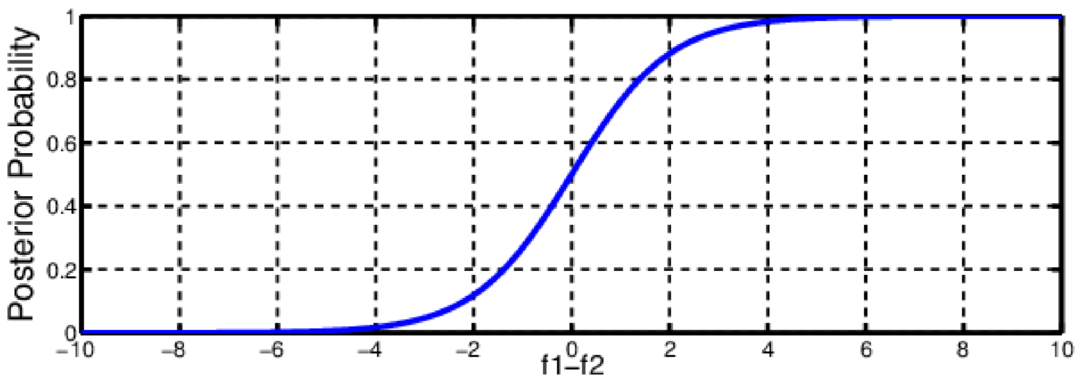


Figure 3.4:SoftMax activation function

Fully Connected Layer (FC):

It involves weights, biases, and neurons. It connects neurons in one layer to neurons in another layer. It is used to classify images between different categories by training.

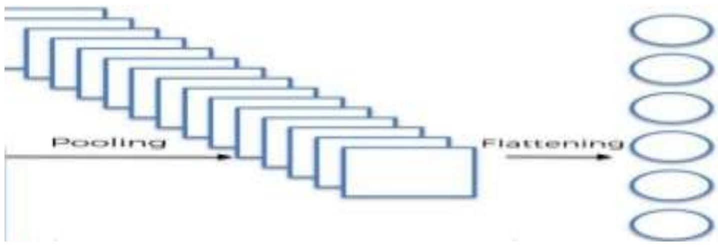


Figure 3.5: flatten layer

Dropout layer:

Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase.

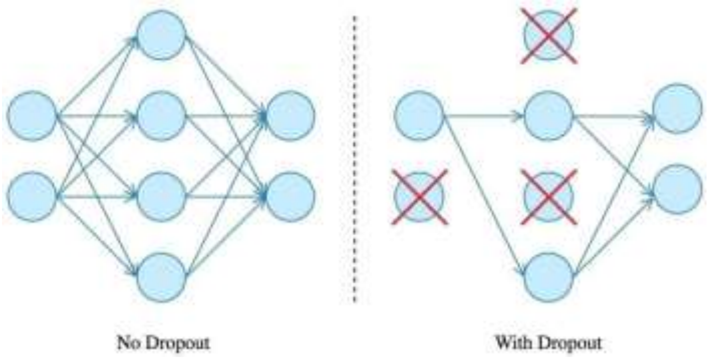


Figure 3.6: dropout layer

Pooling layer:

The pooling layer is used to reduce the spatial volume of the input image after convolution. It is used between two convolution layers. If we apply FC after the Convo layer without applying pooling or max pooling, then it will be computationally expensive, and we don't want it. So, max-pooling is the only way to reduce the spatial volume of the input image. In the above example, we have applied max-pooling in a single depth slice with a Stride of 2. You can observe the 4x4 dimension input is reduced to a 2x2 dimension.

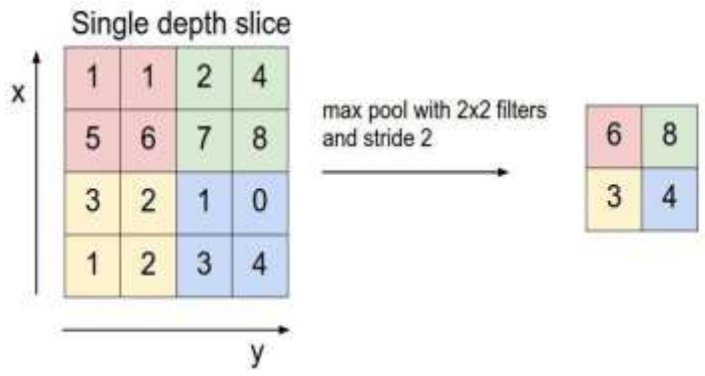


Figure 3.7: Max pooling filter

Activation functions:

The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. We have different types of activation functions.

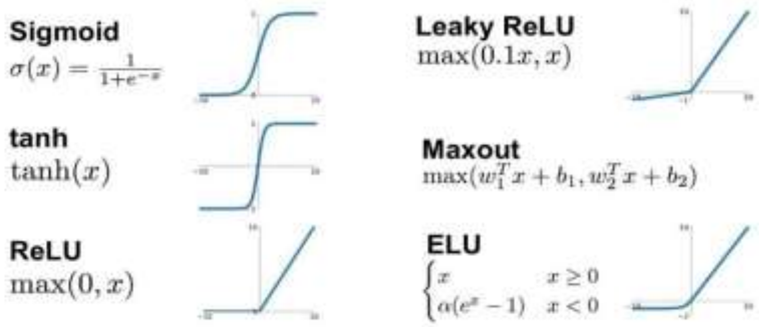


Figure 3.8: activation function

Batch Normalization:

Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.

Generally, when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale. The reason we normalize is partly to ensure that our model can generalize appropriately.

Batch normalization is a process to make neural networks faster and more stable by adding extra layers to a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.

A typical neural network is trained using a collected set of input data called batch. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

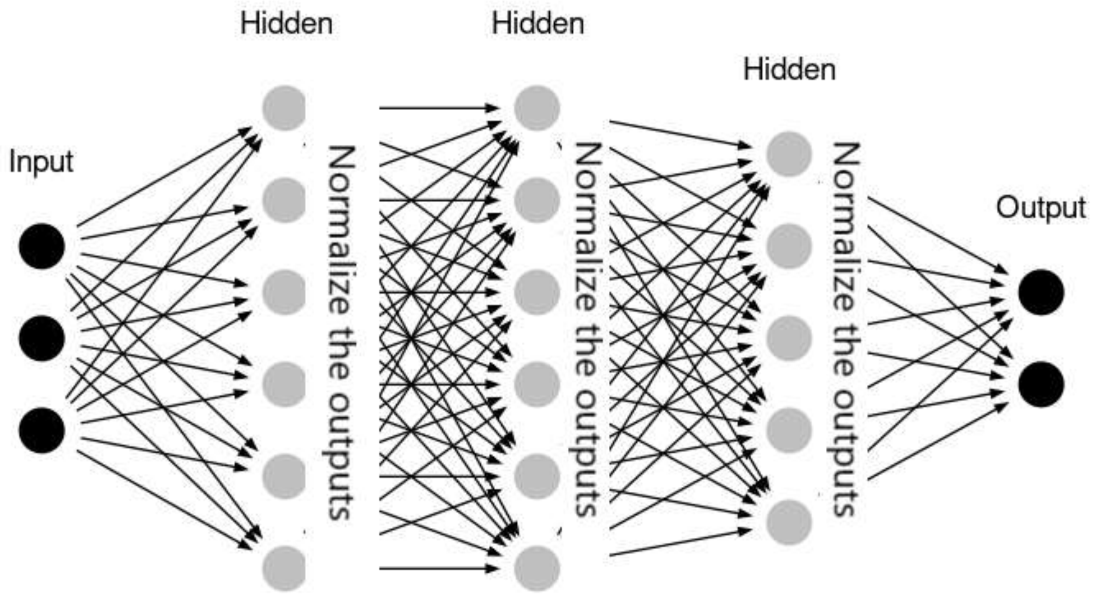


Figure 3.9: Batch Normalization

Chapter 4: System Implementation

In this chapter, we will present the proposed system that we have implemented to develop image style transfer systems that we can combine two images, we did that by applying training and testing using many deep learning classification models to determine which is the best one.

<u>4.1 Data Acquisition</u>

We used two data sets:

1- Best Artworks of All Time
Collection of artworks of the 50 most influential artists of all time.

2- Google Scraped Image Dataset

The images belong typically to 4 classes

- Art & Culture
- Architecture
- Food and Drinks
- Travel and Adventure

<u>4.2 Pre-processing</u>

It refers to the transformations applied to our data before feeding it into our models, because it directly affects the ability of our model to learn. The pre-processing techniques we had applied in two steps filtering and resampling.

To make an image compatible with the model, the image has to be preprocessed. Using the torch.transforms() we will perform some of the basic preprocessing that will include the following steps:

Resize: resize all the image to 300 x 300
To Tensor: to convert the image into a tensor

And also Normalize the tensor with mean (0.5, 0.5, 0.5) and standard deviation of (0.5, 0.5, 0.5) for the pretrained vgg16 model. But don’t forget to convert it back to its original scale. So, define a function that loads the image using the PIL library and preprocess it. Add an extra dimension at 0th index, using unsqueeze() for batch size and then load it to the device and return it.

4.3 image representation

4.3.1Content Representation

The convolutional neural network develops representations of the image along the processing hierarchy. As we move deeper into the network, the representations will care more about the structural features or the actual content than the detailed pixel data. To obtain these representations, we can reconstruct the images using the feature maps of that layer. Reconstruction from the lower layer will reproduce the exact image. In contrast, the higher layer's reconstruction will capture the high-level content and hence we refer to the feature responses from the higher layer as the content representation.

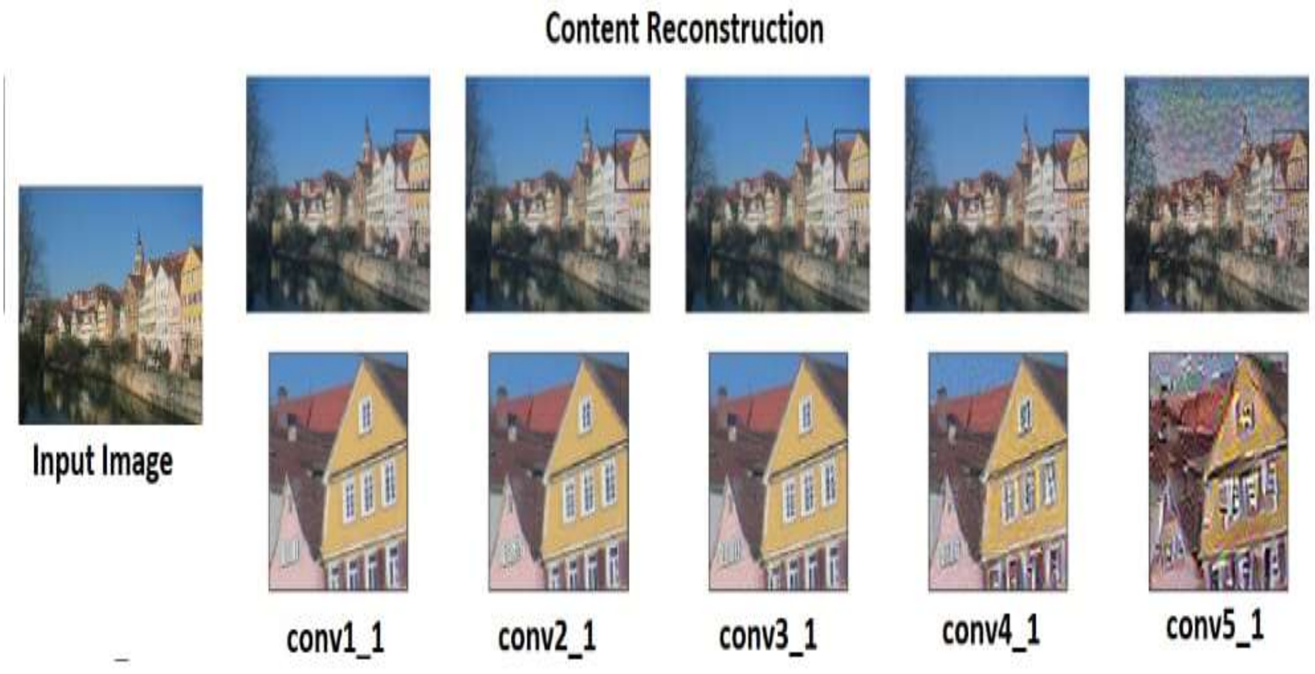


Figure 4.1: Content Representation

The above figure *4.1* shows the reconstruction of the input image from the layers 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1'. We find that the reconstruction from the lower layers is almost the same as the input image, but as we move deeper into the network, detailed pixel information is lost. In contrast, the high-level content of the image is preserved.

4.3.2 Style Representation

To extract the representation of the style content, we build a feature space on the top of the filter responses in each network layer. It consists of the correlations between the different filter responses over the spatial extent of the feature maps. The filter correlation of different layers captures the texture information of the input image. This creates images that match a given image's style on an increasing scale while discarding information of the global arrangement. This multi-scale representation is called style representation.

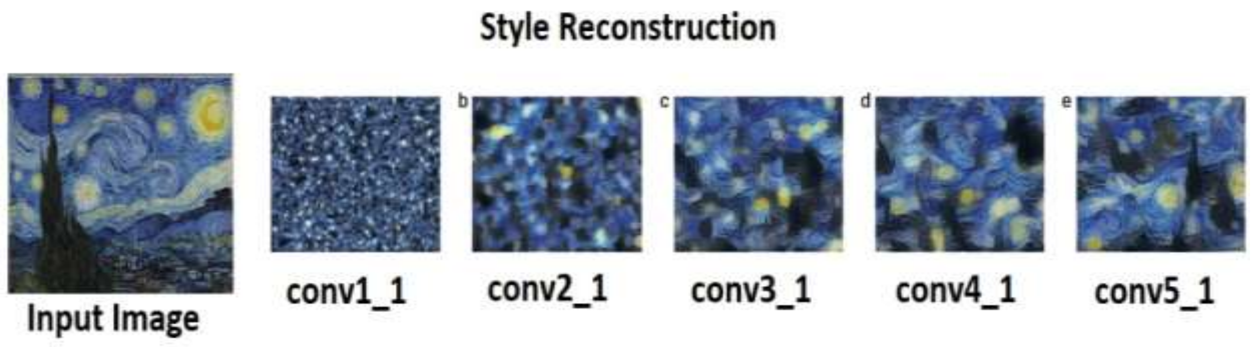


Figure 4.2: style Representation

The above figure *4.2* shows the feature space above each convolutional layer representing the correlation between different features in different layers of CNN. As we move deeper into the network, we can see that the global arrangement or the structural features are discarded.

Model Architecture:

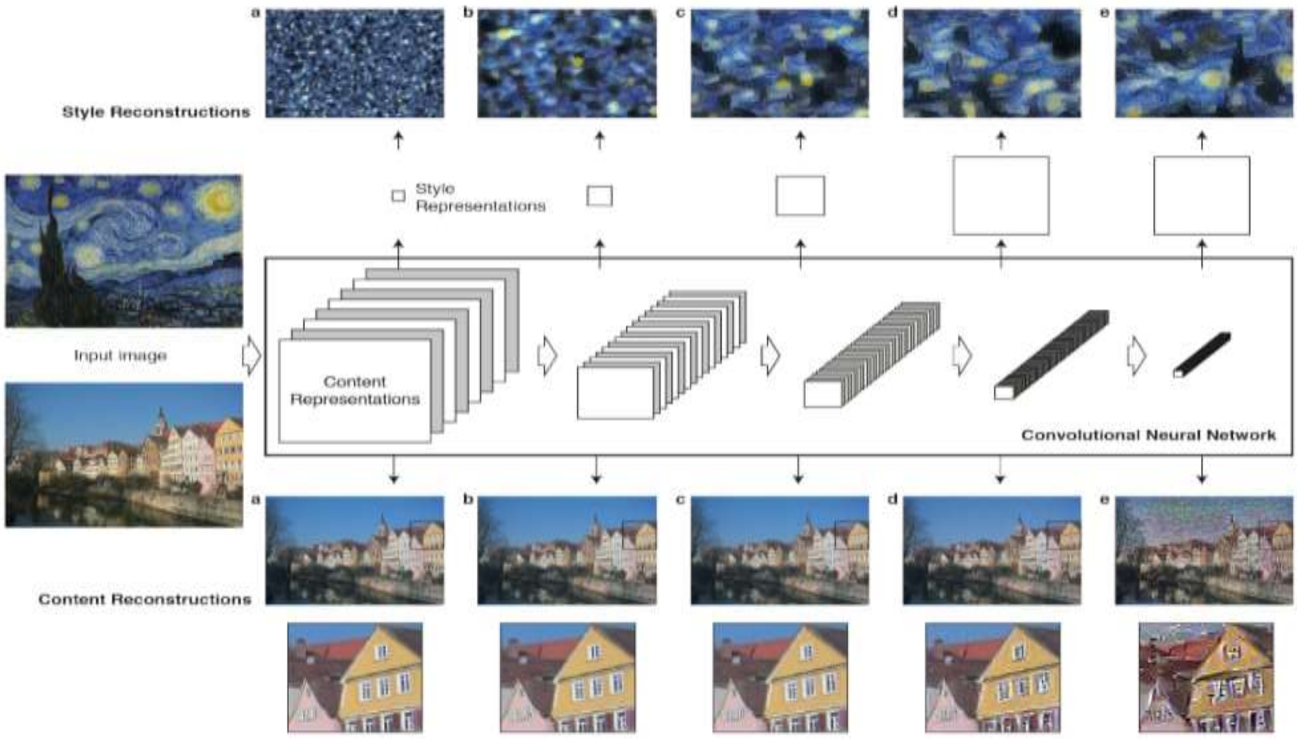


Figure 4.3: system architecture

Figure *4.3* Image representations in a Convolutional Neural Network (CNN). A given input image is represented as a set of filtered images at each processing stage in the CNN. While the number of different filters increases along the processing hierarchy, the size of the filtered images is reduced by some down sampling mechanism (e.g. max-pooling) leading to a decrease in the total number of units per layer of the network. Content Reconstructions. We can visualize the information at different processing stages in the CNN by reconstructing the input image from only knowing the network's responses in a particular layer. We reconstruct the input image from layers 'conv1 2' (a), 'conv2 2' (b), 'conv3 2' (c), 'conv4 2' (d) and 'conv5 2' (e) of the original VGG-Network. We find that reconstruction from lower layers is almost perfect (a–c). In higher layers of the network, detailed pixel information is lost while the high-level content of the image is preserved (d,e). Style Reconstructions. On top of the original CNN activations we use a feature space that captures the texture information of an input image. The style representation computes correlations between the different features in different layers of the CNN. We reconstruct the style of the input image from a style representation built on different subsets of CNN layers ('conv1 1' (a), 'conv1 1' and 'conv2 1' (b), 'conv1 1', 'conv2 1' and 'conv3 1' (c), 'conv1 1', 'conv2 1', 'conv3 1' and 'conv4 1' (d), 'conv1 1', 'conv2 1', 'conv3 1', 'conv4 1' and 'conv5 1' (e). This creates images that match the style of a given image on an increasing scale while discarding information of the global arrangement of the scene.

4.4 CNN model

4.4.1 VGG 19 model

VGG-19 is a trained Convolutional Neural Network, from Visual Geometry Group, Department of Engineering Science, University of Oxford. The number 19 stands for the number of layers with trainable weights. 16 Convolutional layers and 3 Fully Connected layers.

Our VGG 19 model architecture:

It consists of the input layer and 2 convolution layers of size (1x3) then 5 blocks of VGG19 that conation sequential layers of convolution and pooling (conv1D, conv1D, maxpooling, conv1D, conv1D, maxpooling, conv1D, conv1D, conv1D, conv1D, maxpooling, conv1D, conv1D, conv1D, conv1D, maxpooling, conv1D, conv1D, conv1D, conv1D, maxpooling.

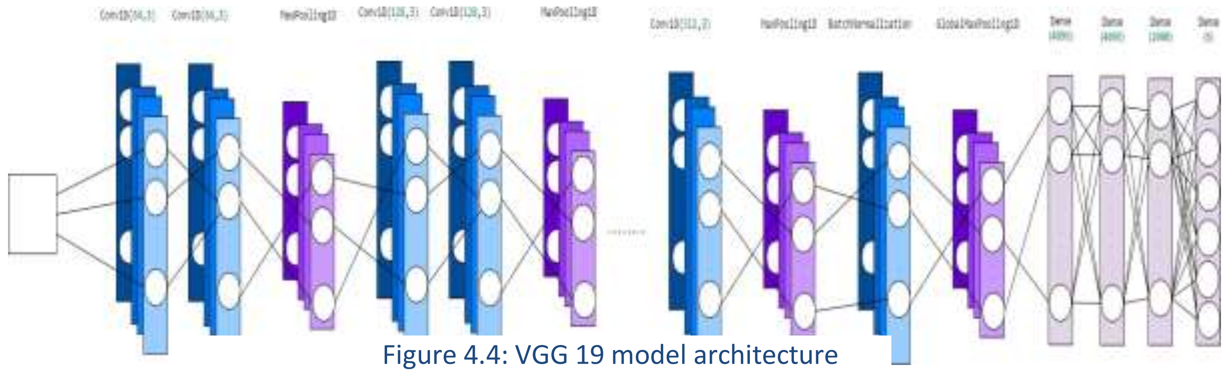


Figure 4.4: VGG 19 model architecture

The results presented below were generated on the basis of the VGG network [17], which was trained to perform object recognition and localization [16] and is described extensively in the original work [17]. We used the feature space provided by a normalized version of the 16 convolutional and 5 pooling layers of the 19-layer VGG network. We normalized the network by scaling the weights such that the mean activation of each convolutional filter over images and positions is equal to one. Such re-scaling can be done for the VGG network without changing its output, because it contains only rectifying linear activation functions and no normalization or pooling over feature maps. We do not use any of the fully connected layers. The model is publicly available and can be explored in the caffe-framework [14]. For image synthesis we found that replacing the maximum pooling operation by average pooling yields slightly more appealing results, which is why the images shown were generated with average pooling.

VGG 16 model:

VGG-16 is a trained Convolutional Neural Network, from Visual Geometry Group, Department of Engineering Science, University of Oxford. The number 16 stands for the number of layers with trainable weights. 13 Convolutional layers and 3 Fully Connected layers.

Our VGG 16 model architecture:

It consists of the input layer and 2 convolution layers of size (1x3) then 5 blocks of VGG16 that conation sequential layers of convolution and pooling (conv1D, conv1D, maxpooling, conv1D, conv1D, maxpooling, conv1D, conv1D, conv1D, maxpooling, conv1D, conv1D, conv1D, maxpooling,conv1D,conv1D,conv1D,maxpooling)

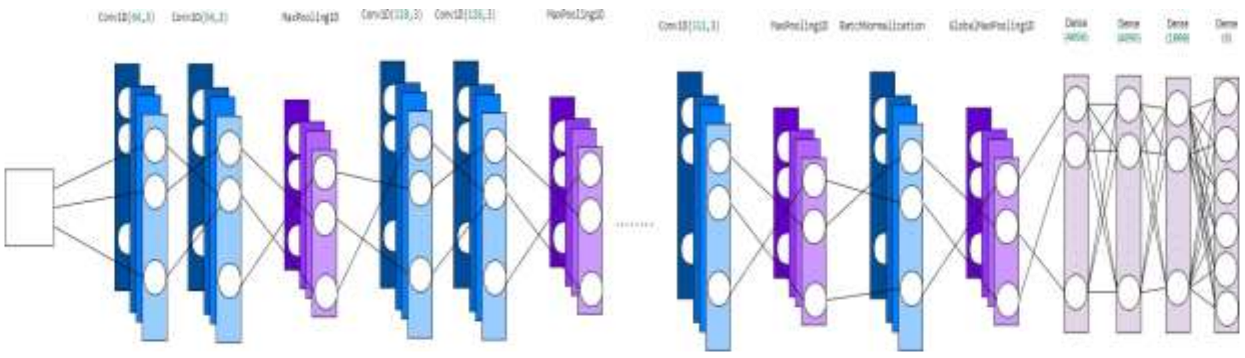


Figure 4.5: VGG 16 model architecture

The results presented below were generated on the basis of the VGG network [17], which was trained to perform object recognition and localization [16] and is described extensively in the original work [17]. We used the feature space provided by a normalized version of the 13 convolutional and 5 pooling layers of the 16-layer VGG network. We normalized the network by scaling the weights such that the mean activation of each convolutional filter over images and positions is equal to one. Such re-scaling can be done for the VGG network without changing its output, because it contains only rectifying linear activation functions and no normalization or pooling over feature maps. We do not use any of the fully connected layers. The model is publicly available and can be explored in the caffe-framework [14]. For image synthesis we found that replacing the maximum pooling operation by average pooling yields slightly more appealing results, which is why the images shown were generated with average pooling.

How we use the model

In this case, we will load the pre-trained VGG16 model from the torchvision.models(). The vgg16 model has three components features, avgpool and classifier.

- The feature holds all the convolutional, max pool and ReLu layers
- avgpool holds the average pool layer.
- Classifier holds the dense layers.

Let's define a class that will provide the feature representations of the intermediate layers. Intermediate layers are used because these layers serve as a complex feature extractor. Hence these can describe the style and content of the input image. In this class, we will initialize a model by eliminating the unused layers(layers beyond conv5_1) of the vgg19 model and extract the activations or the feature representations of the 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv4_2' and 'conv5_1' layers (index values [0, 5, 10, 19, 21, 28]). Store these activations of 5 convolutional layers in an array and return the array.

4.5 Define the Loss

4.5.1 Content Loss

Let p and x be the original image and the image that is generated, and P l and F l their

respective feature representation in layer l. We then define the squared-error loss between the two feature representations

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2}\sum_{i,j}\left(F_{ij}^{l} - P_{ij}^{l}\right)^2$$

In the equation, we calculate the squared error between the feature representation of the content image (p) and the input base image (x) of layer (l).

The derivative of this loss with respect to the activations in layer l equals

$$\frac{\partial \mathcal{L}_{\text{content}}}{\partial F^l_{ij}} = \begin{cases} \left(F^l - P^l\right)_{ij} & \text{if } F^l_{ij} > 0 \\ 0 & \text{if } F^l_{ij} < 0\,, \end{cases}$$

from which the gradient with respect to the image ~x can be computed using standard error back-propagation. Thus we can change the initially random image ~x until it generates the same response in a certain layer of the Convolutional Neural Network as the original image ~p. When Convolutional Neural Networks are trained on object recognition, they develop a representation of the image that makes object information increasingly explicit along the processing hierarchy [10]. Therefore, along the processing hierarchy of the network, the input image is transformed into representations that are increasingly sensitive to the actual content of the image, but become relatively invariant to its precise appearance. Thus, higher layers in the network capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction very much (Fig 4.3, content reconstructions d, e). In contrast, reconstructions from the lower layers simply reproduce the exact pixel values of the original image (Fig 4.3, content reconstructions a–c). We therefore refer to the feature responses in higher layers of the network as the content representation.

4.5.2 Style Loss

We build a feature space over each layer of the network representing the correlation between the different filter responses. The Gram matrix calculates these feature correlations.

The gram matrix represents the correlation between each filter in an intermediate representation. Gram matrix is calculated by taking the dot product of the unrolled intermediate representation and its transpose. The gram matrix G dimension is $n^l c$ x $n^l c$, where $n^l c$ is the number of channels in the intermediate representation of layer l.

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

In the equation, G^l_{ij} is the inner product between the vectorized feature map i and j in layer l. The vectorized equation of a gram matrix is shown in the following figure, where G is the gram matrix of intermediate representation A.

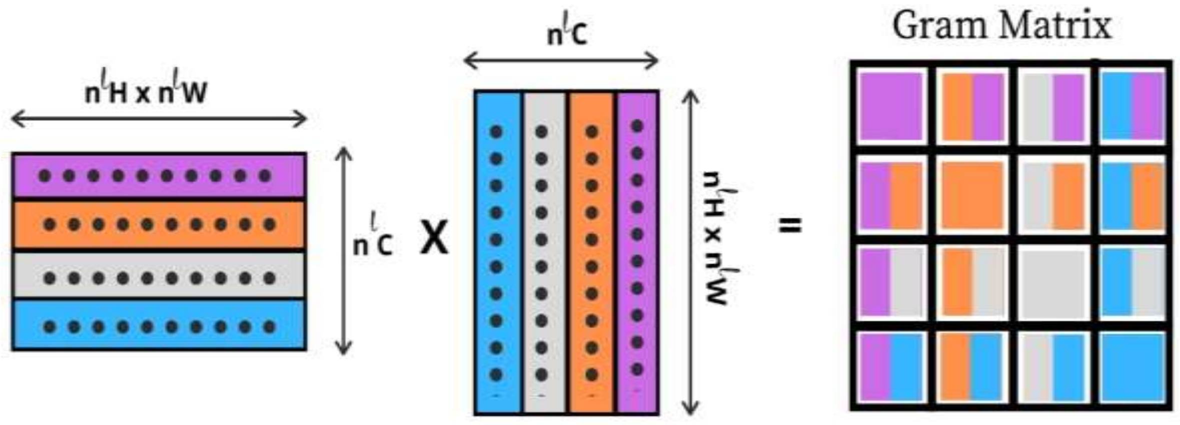


Figure4.6: Gram Matrix Calculation

$$G_{gram} = A_{unrolled} A^T_{unrolled}$$

In the equation, Gram matrix vectorized Equation the style loss of layer l is the squared error between the gram matrices of the intermediate representation of the style image and the generated image.

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - A^l_{ij}\right)^2$$

In the equation, Where E_l is the style loss for layer l, N_l and M_l are the numbers of channels and height times width in the feature representation of layer l respectively. G^l_{ij} and A^l_{ij} are the gram matrices of the intermediate representation of the style image (a) and the input base image (x) respectively.

The total style loss is:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

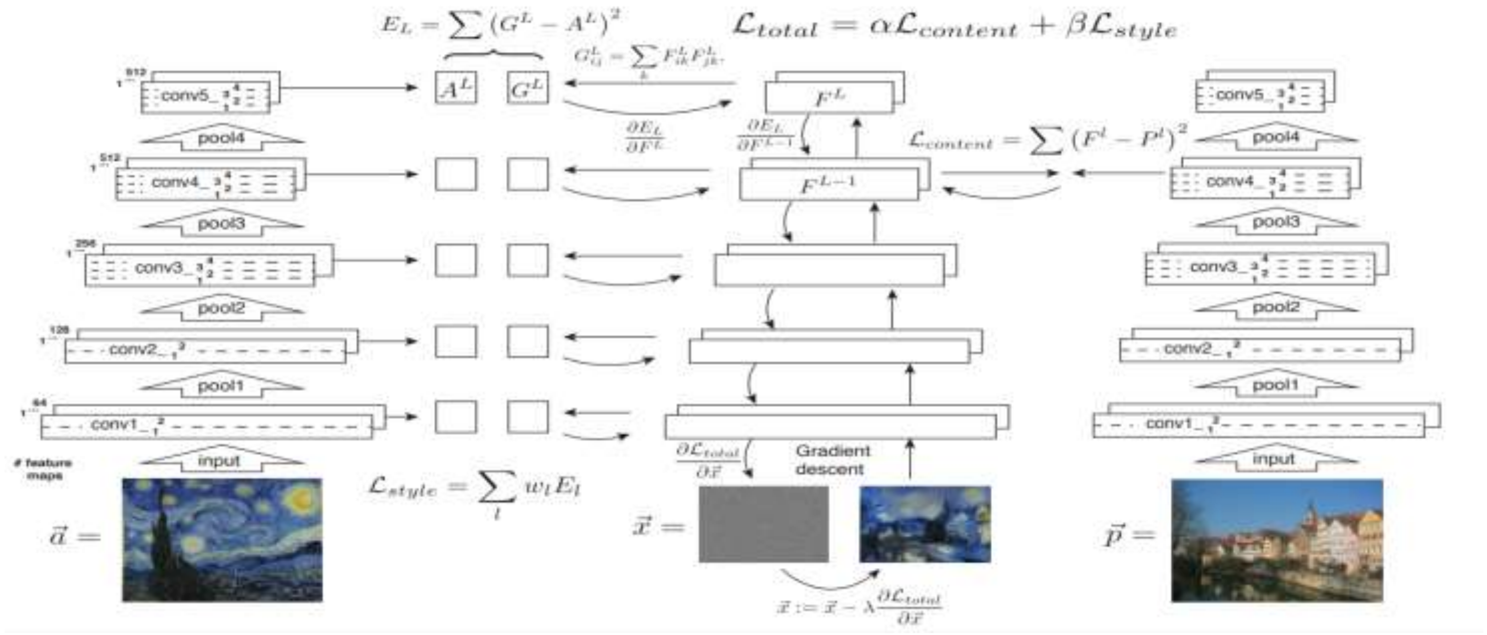
Where w^l is the weight factor of the contribution of each layer to the total style loss.

The derivative of El with respect to the activations in layer l can be computed analytically:

$$\frac{\partial E_l}{\partial F^l_{ij}} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left((F^l)^{\mathrm{T}} \left(G^l - A^l \right) \right)_{ji} & \text{if } F^l_{ij} > 0 \\ 0 & \text{if } F^l_{ij} < 0\,. \end{cases}$$

The gradients of El with respect to the pixel values ~x can be readily computed using standard error back-propagation

Figure. Style transfer algorithm. First content and style features are extracted and stored. The style image ~a is passed through the network and its style



$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
conv5_
pool4
conv4_
pool3
conv3_
pool2
conv2_
pool1
conv1_
input
# feature maps
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

Figure 4.7: General architecture

representation A l on all layers included are computed and stored (left).

The content image ~p is passed through the network and the content representation P l in one layer is stored (right).

Then a random white noise image ~x is passed through the network and its style features G l and content features F l are computed.

On each layer included in the style representation, the element-wise mean squared difference between G l and A l is computed to give the style loss L style (left). Also the mean squared difference between F l and P l is computed to give the content loss L content (right).

The total loss L total is then a linear combination between the content and the style loss. Its derivative with respect to the pixel values can be computed using error back-propagation (middle).

This gradient is used to iteratively update the image ~x until it simultaneously matches the style features of the style image ~a and the content features of the content image ~p (middle, bottom).

4.5.3 Total Loss

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$

In the above equation, L_{total} is the total loss, L c_{ontent} is the content loss of all the intermediate layers and $L_{st\gamma le}$ is the style loss of all the intermediate layers. Here, α and β are the weighting coefficients of the content and the style loss, respectively.

p, a and x are the content image, style image and the generated image or the base input image. We perform gradient descent on the loss function and instead of the model parameters we update the pixel values of the input image x to minimize the loss.

This will make the input image similar to the content and the style image. We can emphasize either the style or the content loss by altering the values of α and β. A strong emphasis on style will result in images that match the artwork's appearance, effectively giving a texturized version of it, but hardly show any of the photograph’s content. When placing a strong emphasis on content, one can identify the photograph, but the painting style is not as well-matched.

4.6 Style transfer:

To transfer the style of an artwork ~a onto a photograph ~p we synthesise a new image that simultaneously matches the content representation of ~p and the style representation of ~a (Fig 4.7).

Thus we jointly minimize the distance of the feature representations of a white noise image from the content representation of the photograph in one layer and the style representation of the painting defined on a number of layers of the Convolutional Neural Network. The loss function we minimize is:

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$

where α and β are the weighting factors for content and style reconstruction, respectively.

The gradient with respect to the pixel values ∂L total ∂~x can be used as input for some numerical optimization strategy. Here we use Adam, which we found to work best for image synthesis.

To extract image information on comparable scales, we always resized the style image to the same size as the content image before computing its feature representations.

Finally, note that in difference to [15] we do not regularize our synthesis results with image priors. It could be argued, though, that the texture features from lower layers in the network act as a specific image prior for the style image.

Additionally some differences in the image synthesis are expected due to the different network architecture and optimization algorithm we use.

4.7 Training

- Initialize the variables that we will need to train our model. So, before moving on to train, we need to set the hyper parameters.

- create an object for the class VGG. Initializing the object will call the constructor and it will return a model with the first 29 layers and load it to the device. Epoch is 4000, the learning rate is 0.004, alpha (weighting coefficient of content loss) is 8 and beta (weighting coefficient of style loss) is 70.

- Adam is used as an optimizer. The pixel data of the generated image will be optimized to pass the generated image as the optimizer parameter.

- Use a for loop to iterate over the number of epochs. Extract the feature representation of the intermediate layers of the content, style and the generated image using the model. On passing an image to the model, it will return an array of length 6. Each element corresponds to the feature representation of each intermediate layer.

- Calculate the total loss by using the above-defined functions. Set the gradients to zero with optimizer.zero_grads(), backpropagate the total loss with total_loss.backward() and then update the pixel values of the generated image using optimizer.step(). We will save the generated image after every 500 epochs and print the total loss.

After training, you can find the artistic image in your current working directory.

Feel free to play with the hyper parameters.

Sometimes you may get the expected results but sometimes you may struggle to achieve the desired result.

Chapter 5: System Results

5.1Result

The key finding of this paper is that the representations of content and style in the Convolutional Neural Network are well separable. That is, we can manipulate both representations independently to produce new, perceptually meaningful images. To demonstrate this finding, we generate images that mix the content and style representation from two different source images. The images shown in results were synthesized by matching the content representation on layer 'conv4_2' and the style representation on layers 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' and epochs :4000.

Results of different models

1- VGG19Model:

We try to changing the hyper parameters many times.
*First time we use content weight α=100,style weight β=1e8 and learning rate = 0.01

Style Image Content Image

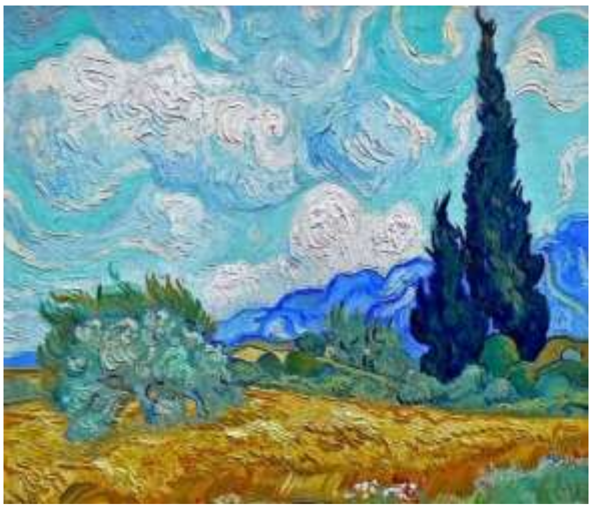


Figure 5.1: Style image

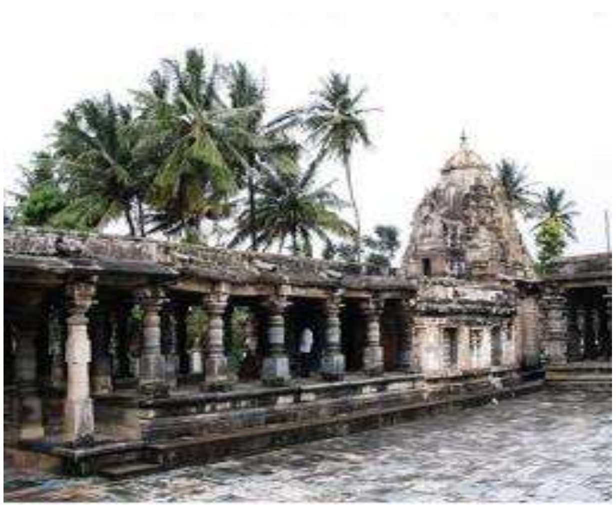


Figure5.2: Content image

Output image

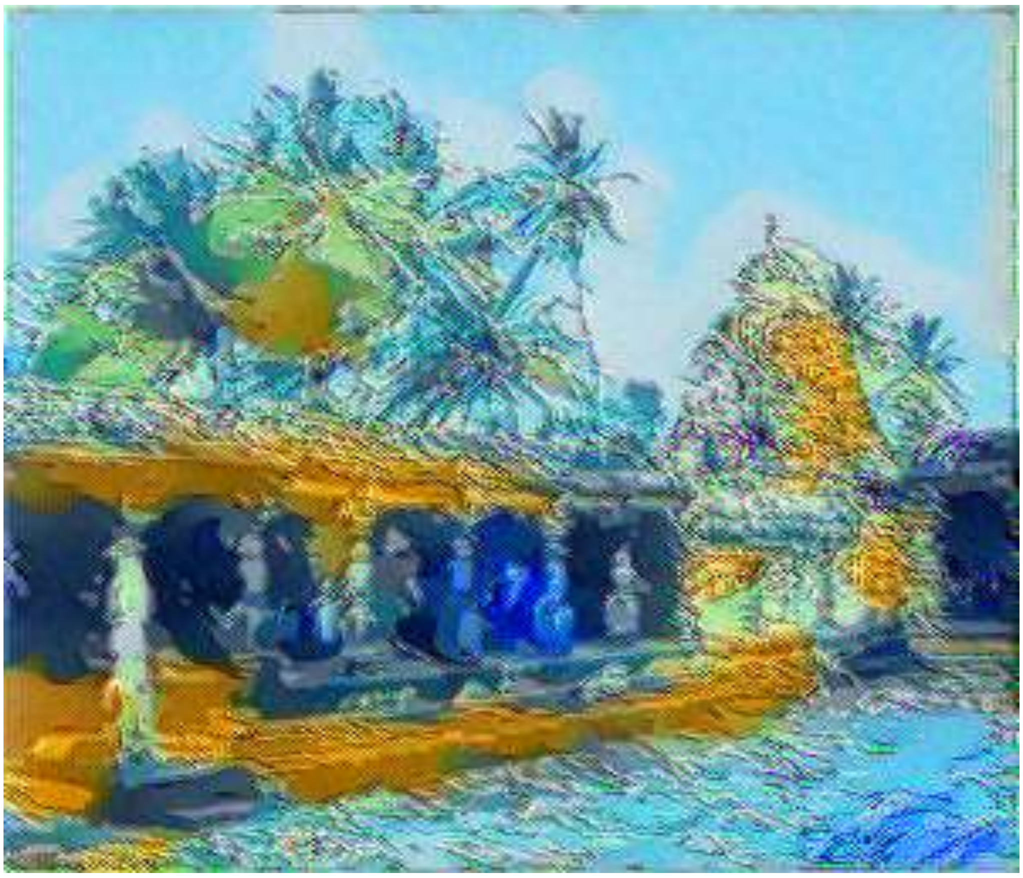


Figure 5.3: Output image

<u>The loss was :2.2055e+16</u>

Style Image   Content Image

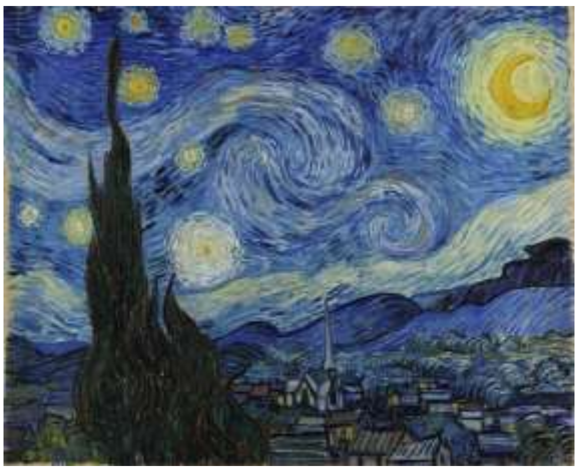


Figure 5.4: Style image

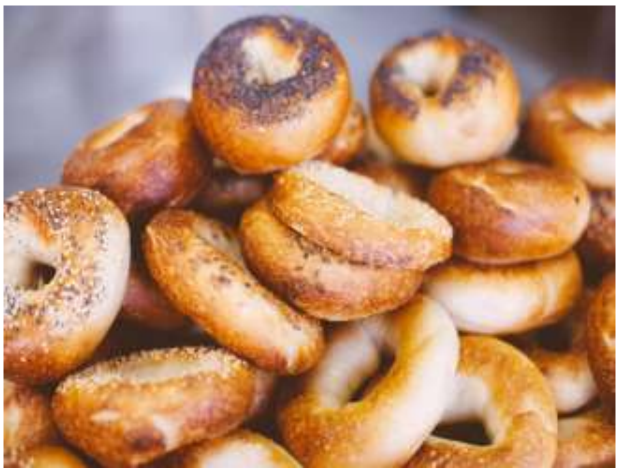


Figure 5.5: Content image

Output image

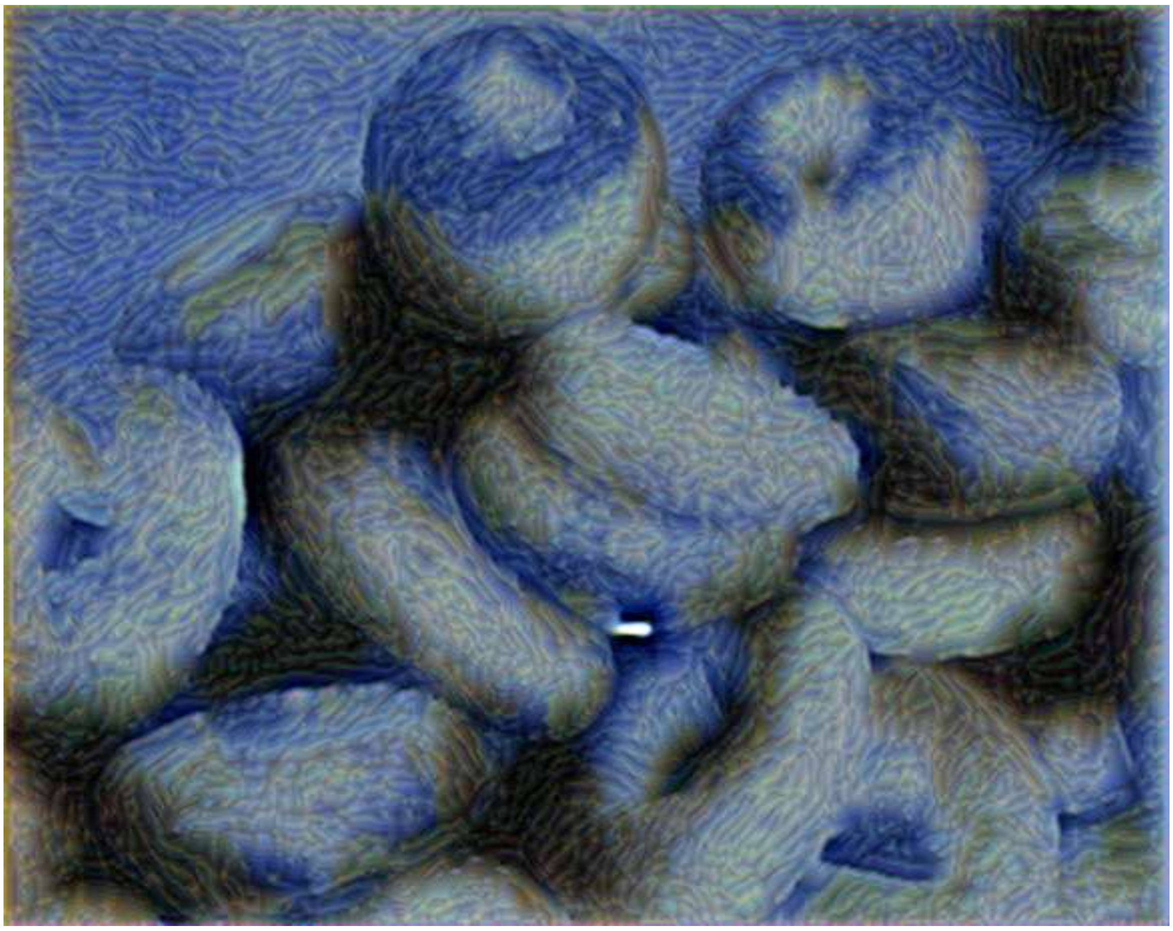


Figure 5.6: Output image

<u>The loss was :2.1380e+17</u>

*Second time we use content weight α=200, style weight β =1e6 and learning rate = 0.007.

Style Image Content Image

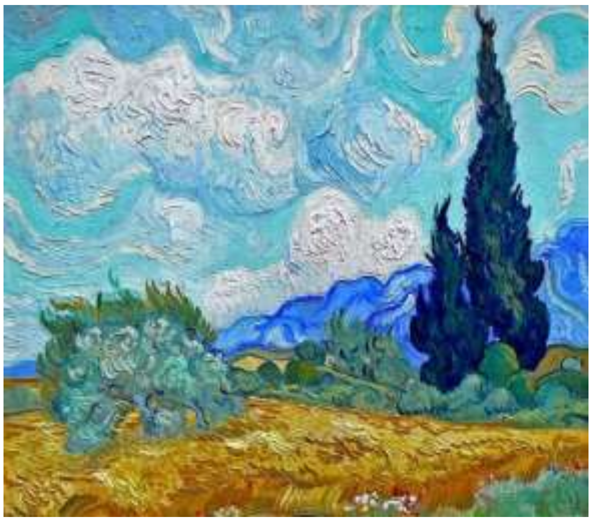


Figure 5.7: Style image

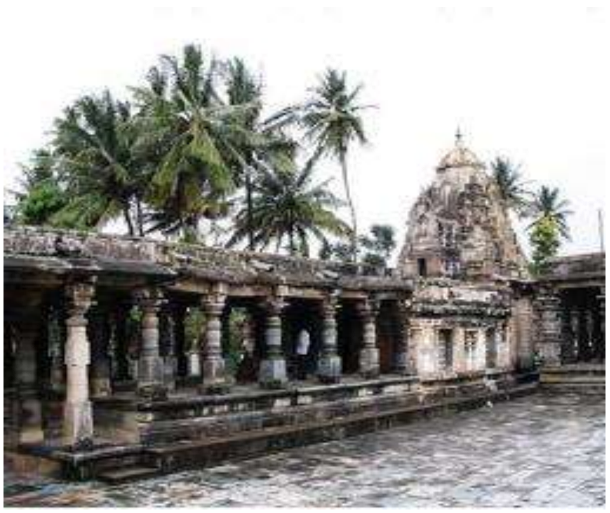


Figure 5.8: Content image

Output image

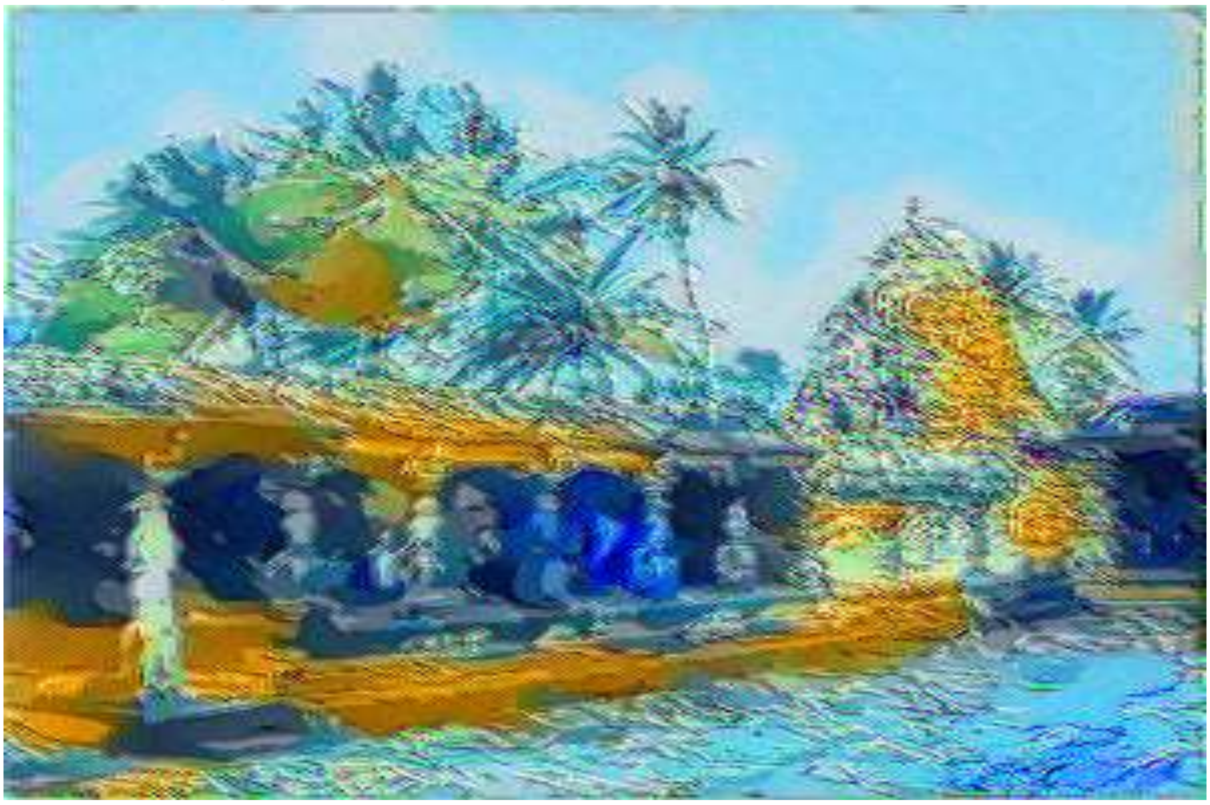


Figure 5.9: Output image

<u>The loss was :2.4188e+14</u>

Style Image Content Image

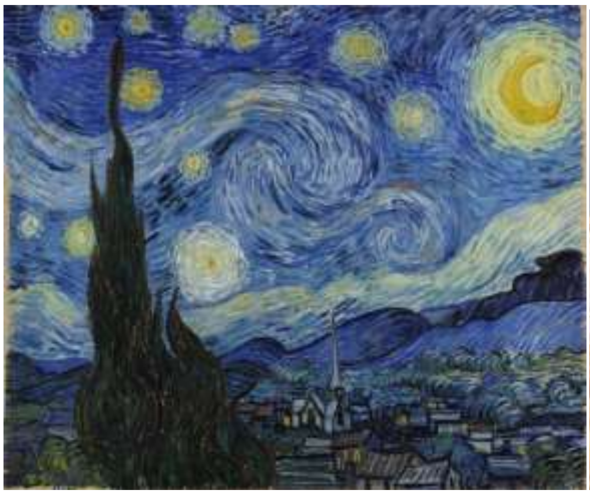


Figure 5.10: Style image

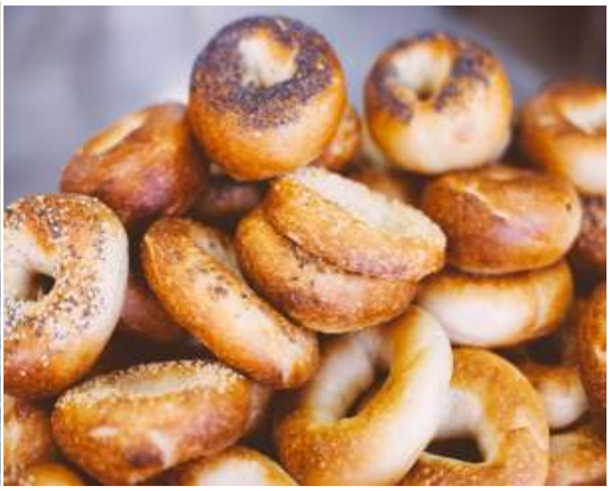


Figure 5.11: Content image

Output image

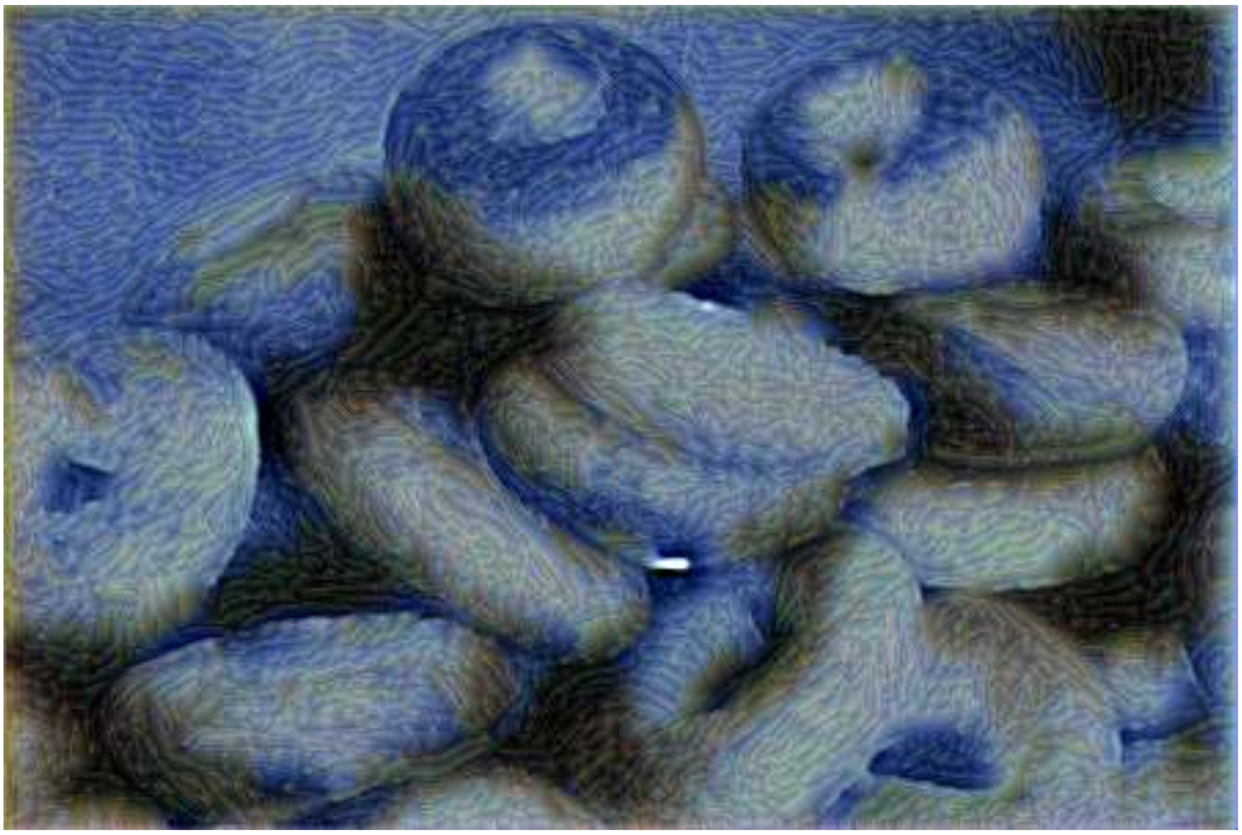


Figure 5.12: Output image

<u>The loss was :2.1507e+15</u>

* Third time we use content weight α=1000, style weight β =1e6 and learning rate = 0.04.

Style Image

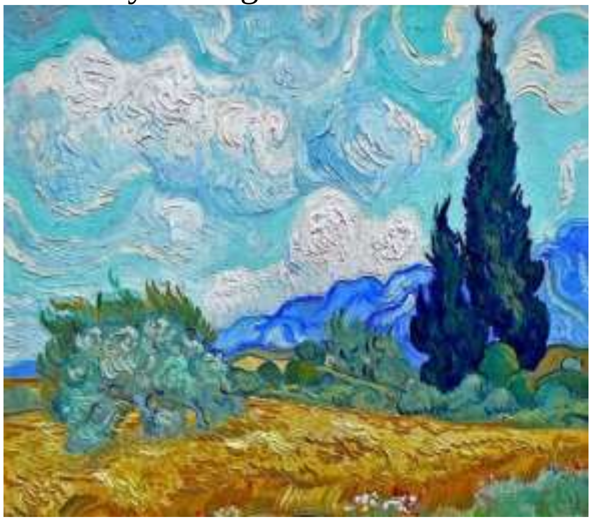


Figure 5.13: Style image

Content Image

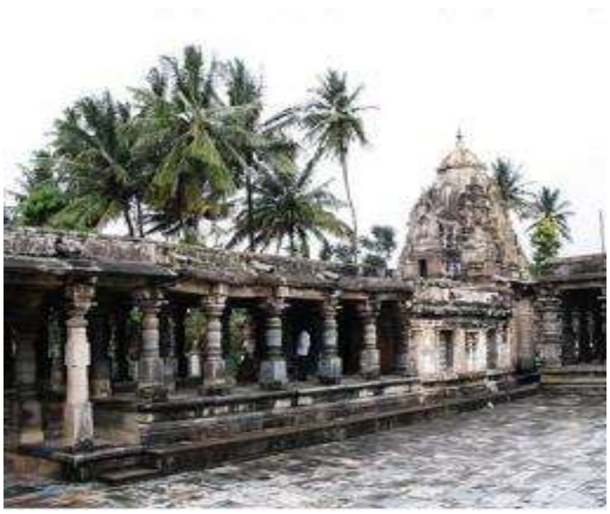


Figure 5.14: Content image

Output image

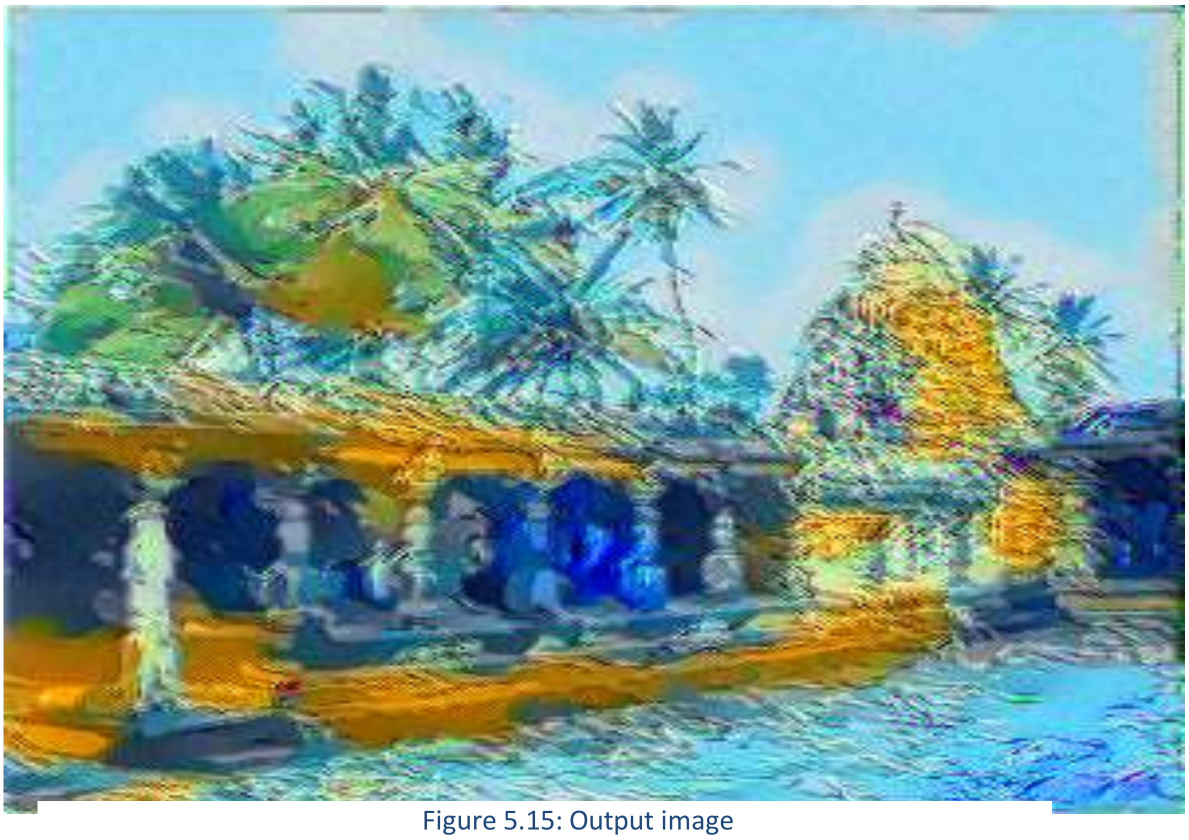


Figure 5.15: Output image

<u>**The loss was :2.8882e+14**</u>

Style Image

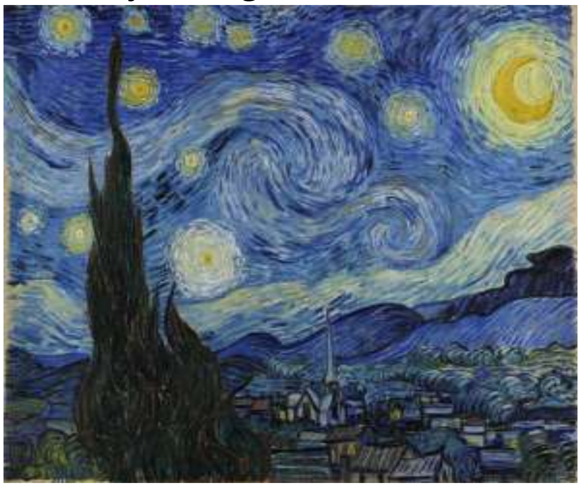


Figure 5.16: Style image

Content Image

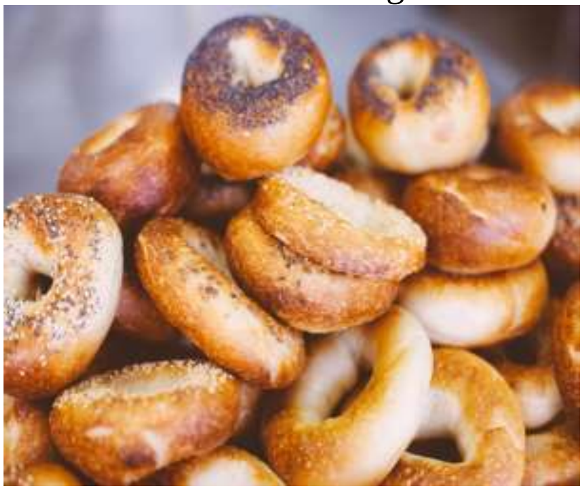


Figure 5.17: Content image

Output image

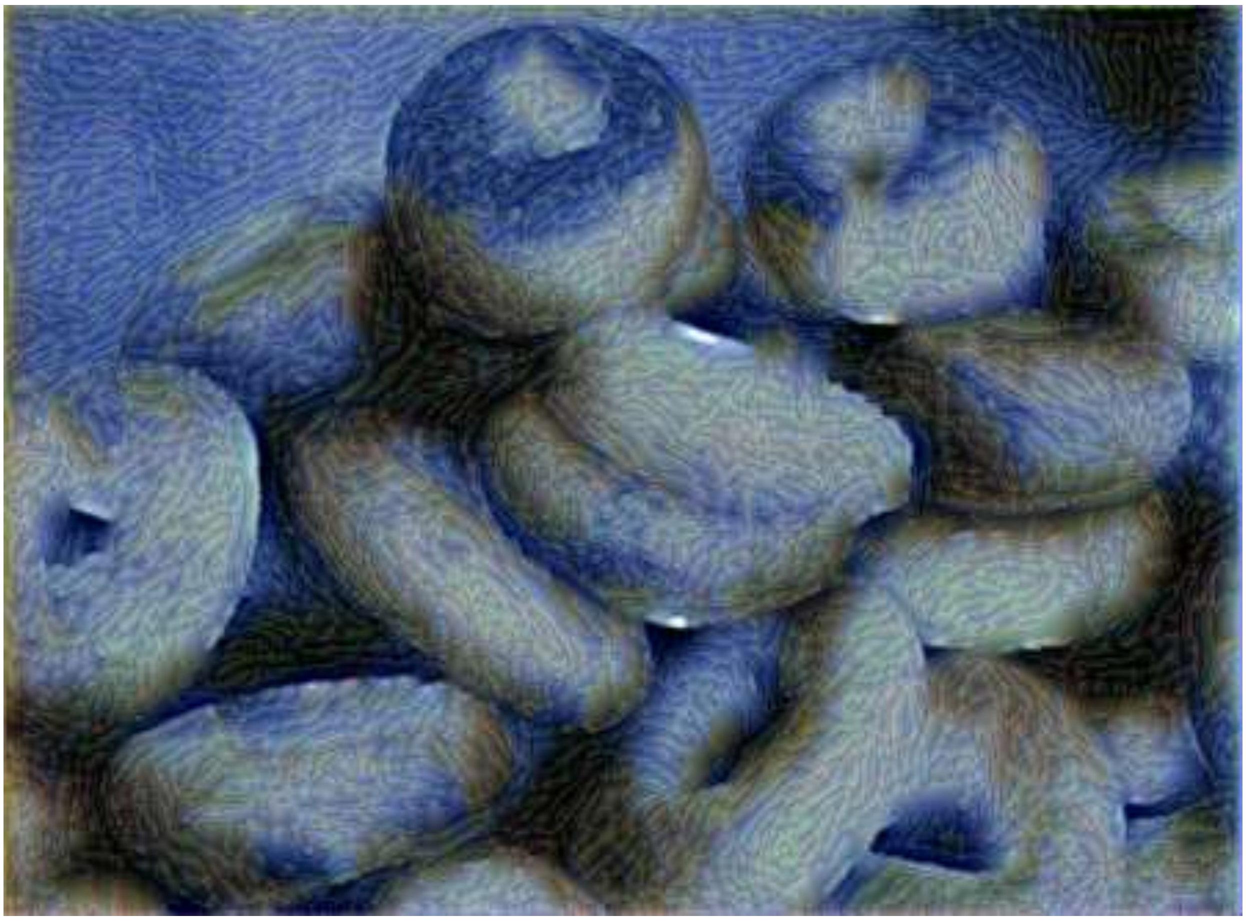


Figure 5.18: Output image

<u>The loss was :2.2014e+15</u>

*Fourth time we use content weight α=8, style weight β =70 and learning rate = 0.004.

Style Image | Content Image

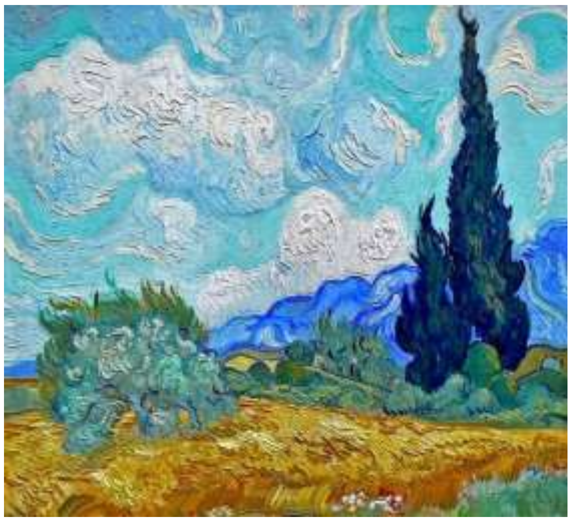


Figure 5.19: Style image

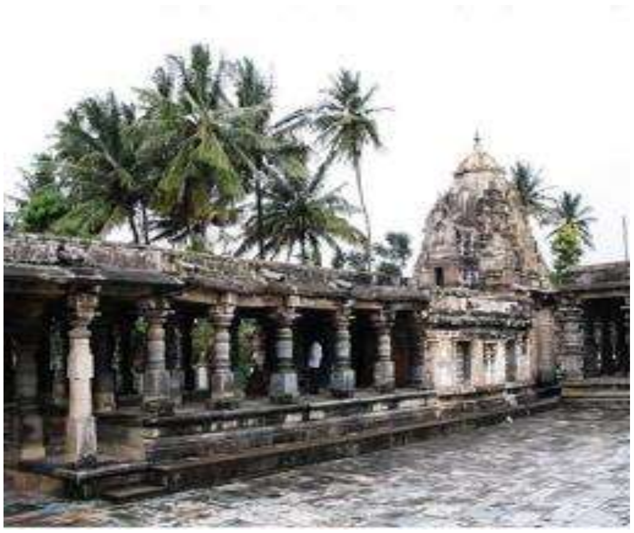


Figure 5.20: Content image

Output image

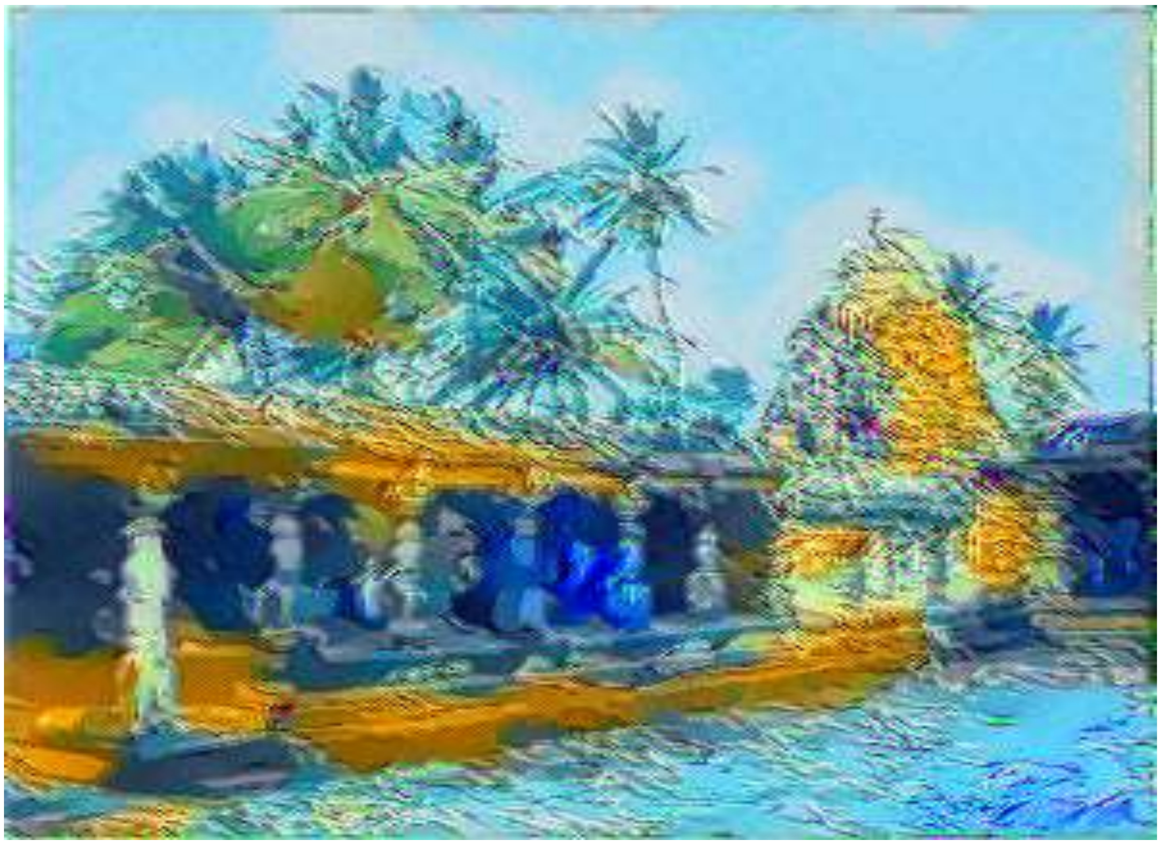


Figure 5.21: Output image

<u>The loss was :2.0260e+10</u>

Style Image Content Image

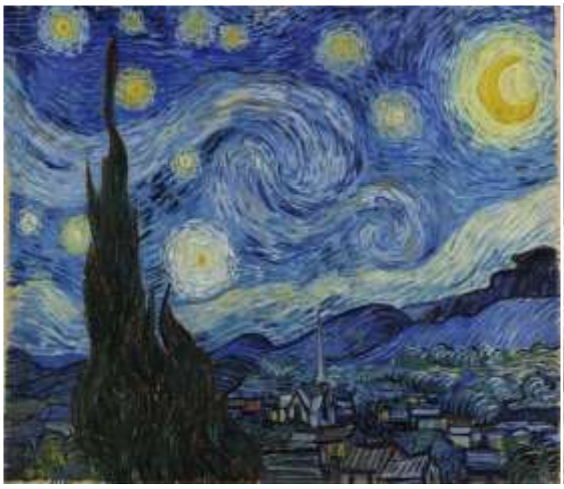


Figure 5.22: Style image

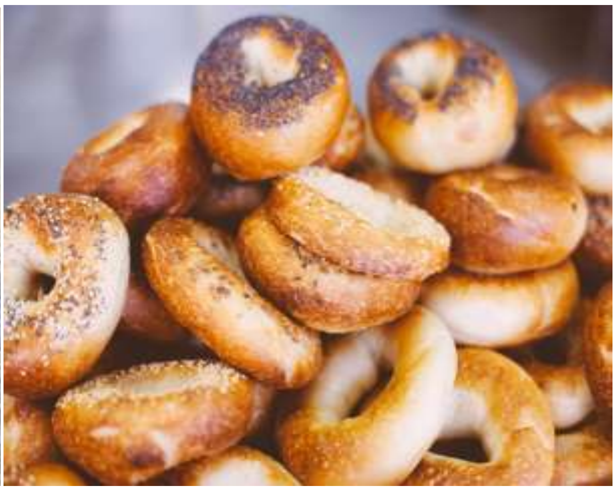


Figure 5.23: Content image

Output image

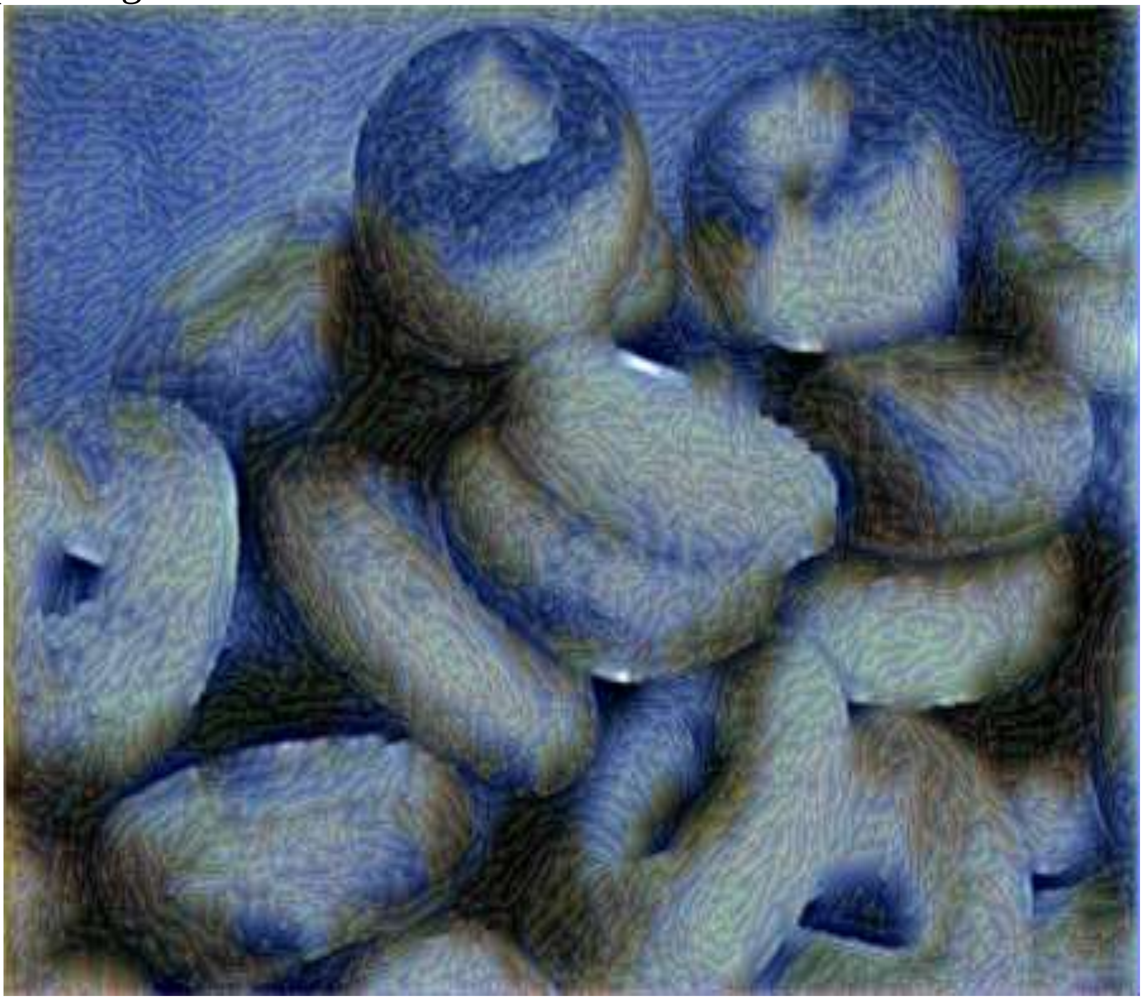


Figure 5.24: Output image

<u>The loss was :1.5419e+11</u>

2- VGG16 Model:

We try to changing the hyper parameters many times.

*First time we use content weight α=100, style weight β=1e8 and learning rate = 0.01.

Style Image

Content Image

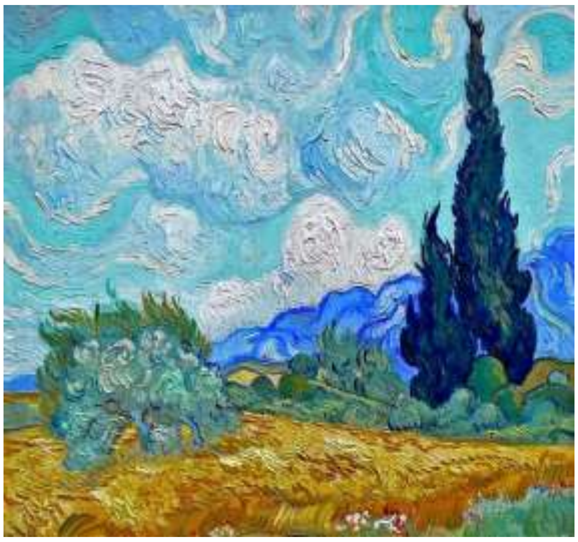


Figure 5.25: Style image

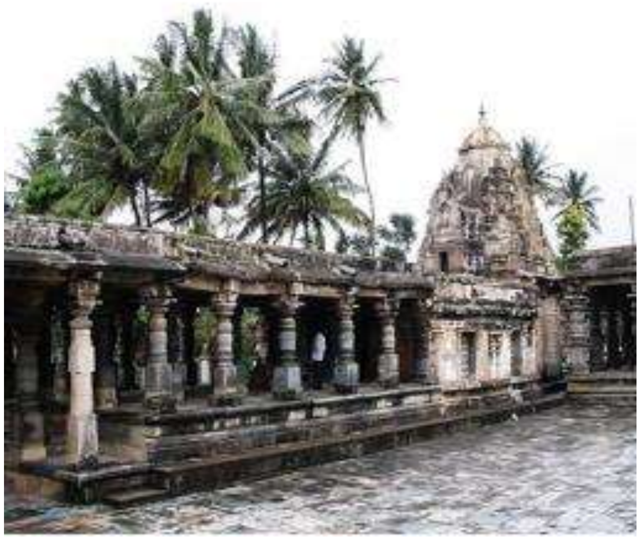


Figure 5.26: Content image

Output image

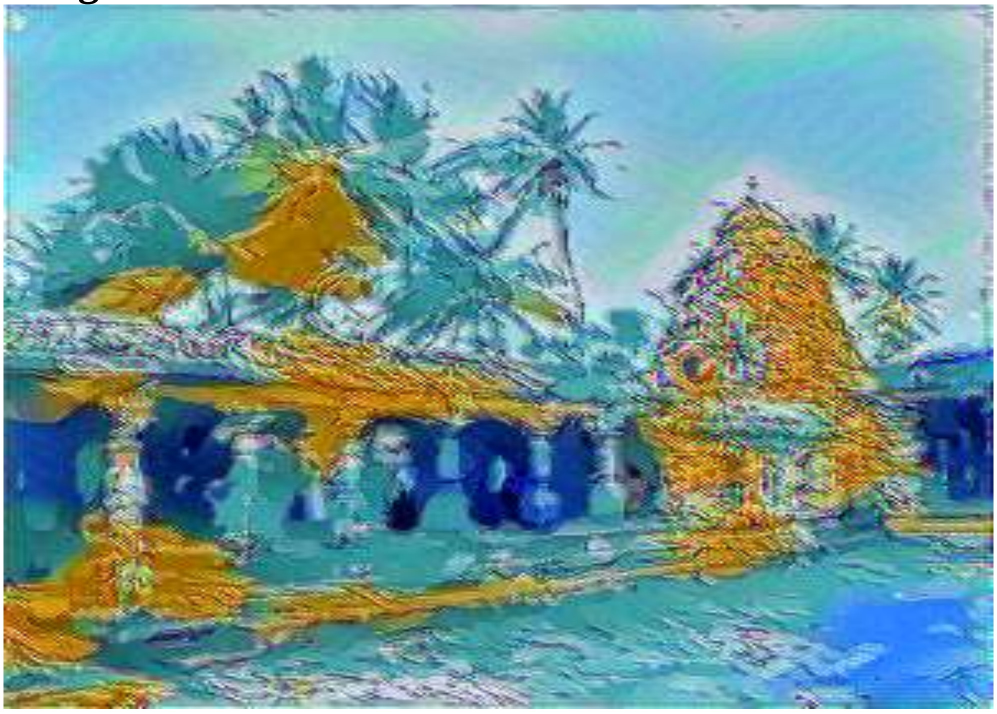


Figure 5.27: Output image

The loss was :1.9218e+16

Style Image

Content Image

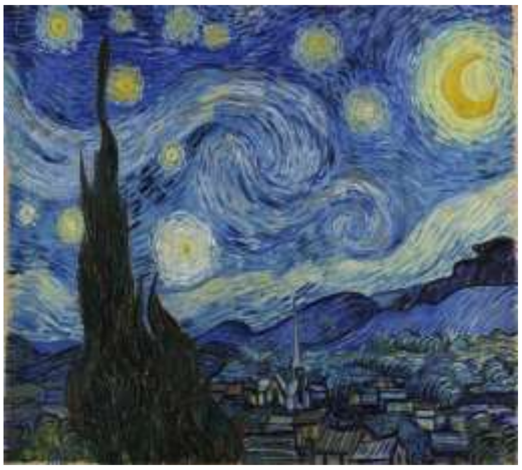


Figure 5.28: Style image

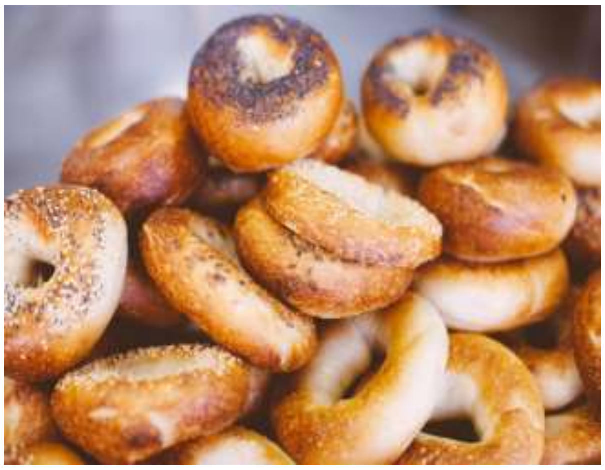


Figure 5.29: Content image

Output image

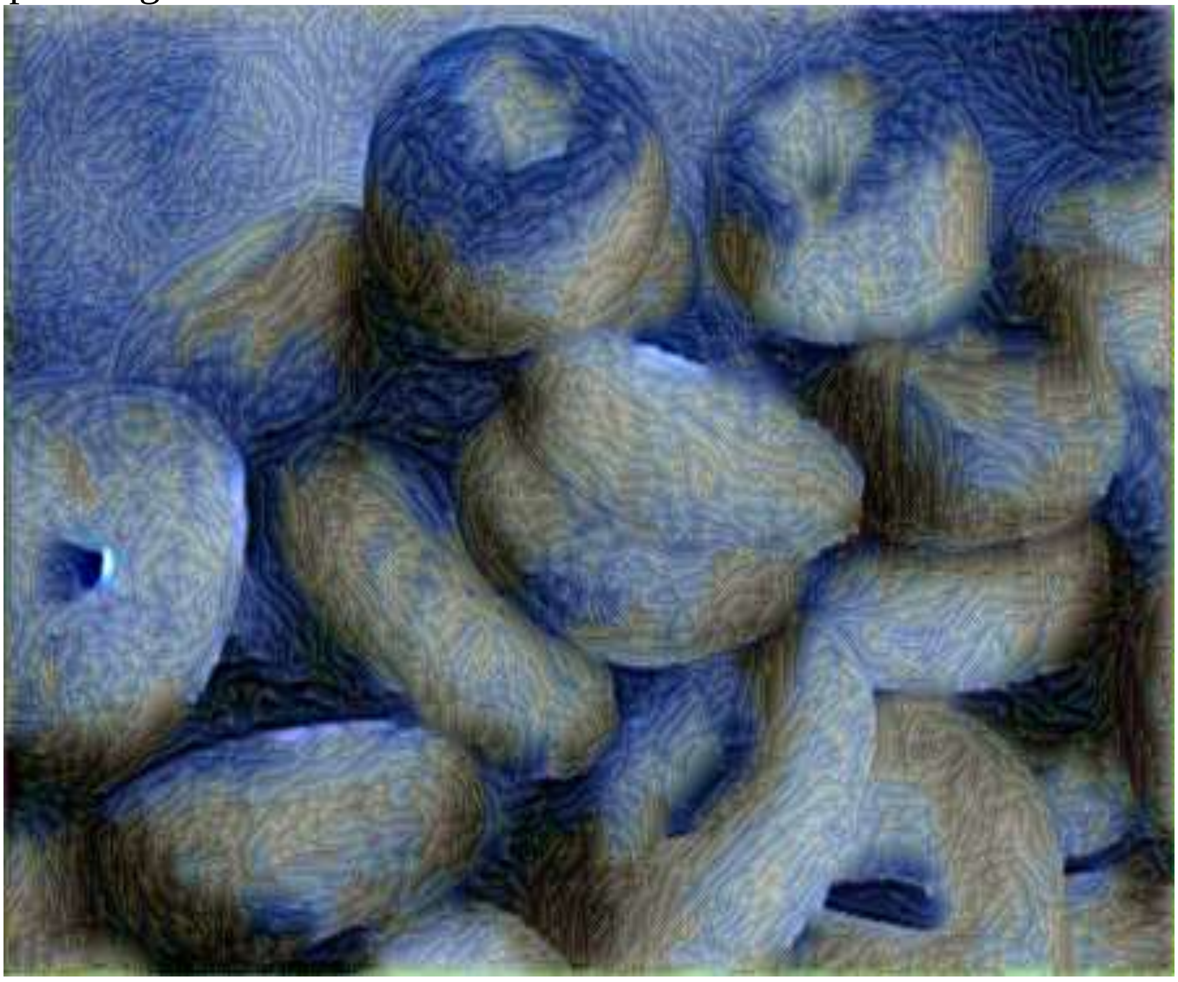


Figure 5.30: Output image

The loss was :1.4730e+17

*Second time we use content weight α=200, style weight β =1e6 and learning rate = 0.007.

Style Image

Content Image

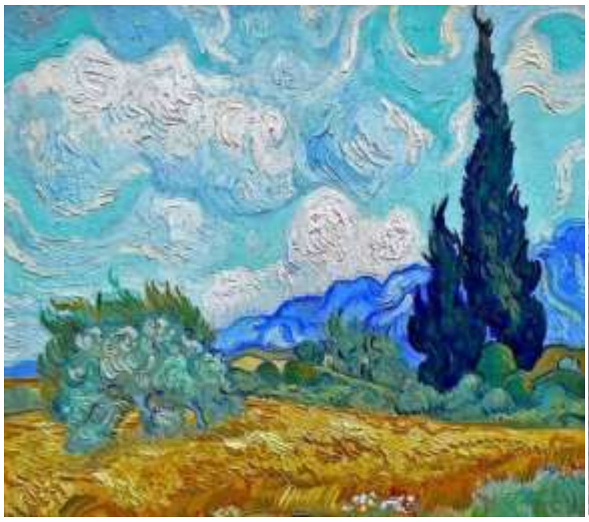


Figure 5.31: Style image

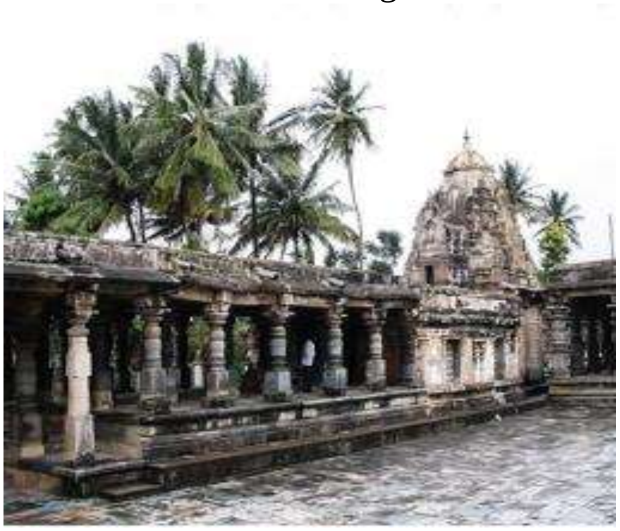


Figure 5.32: Content image

Output image

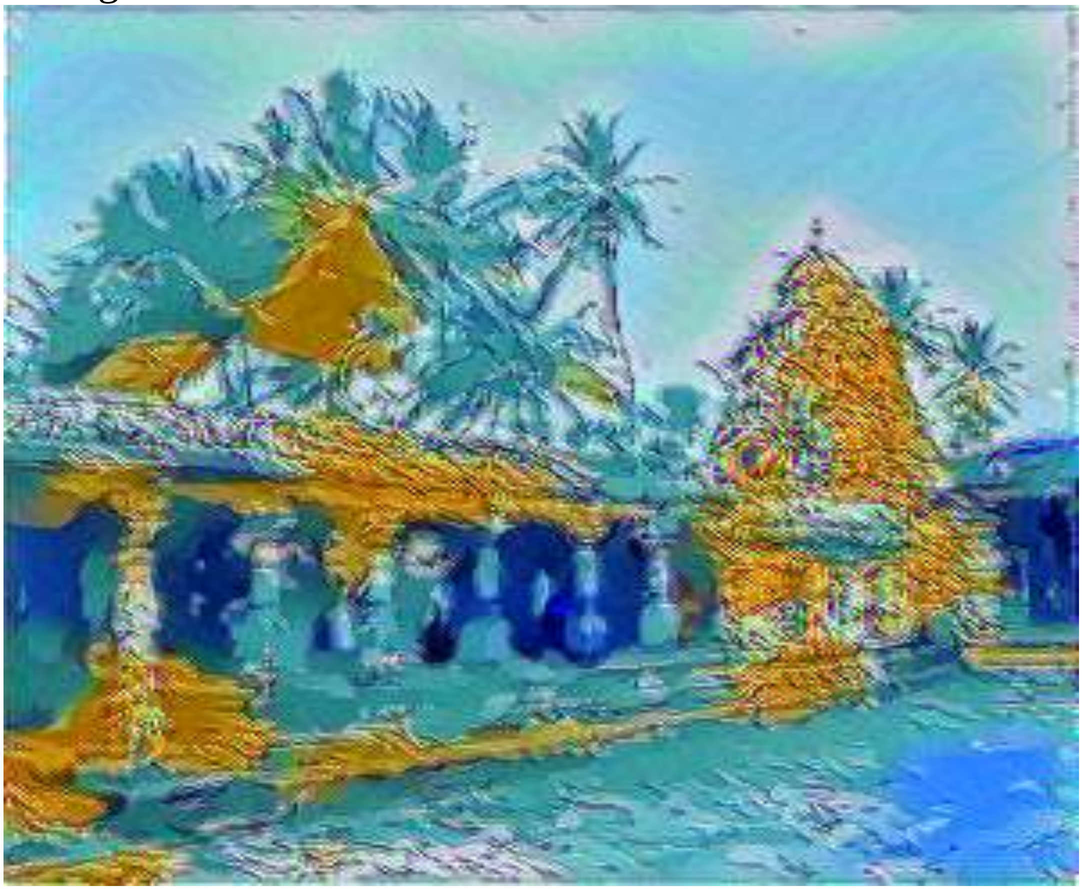


Figure 5.33: Output image

<u>The loss was :2.1151e+14</u>

Style Image

Content Image

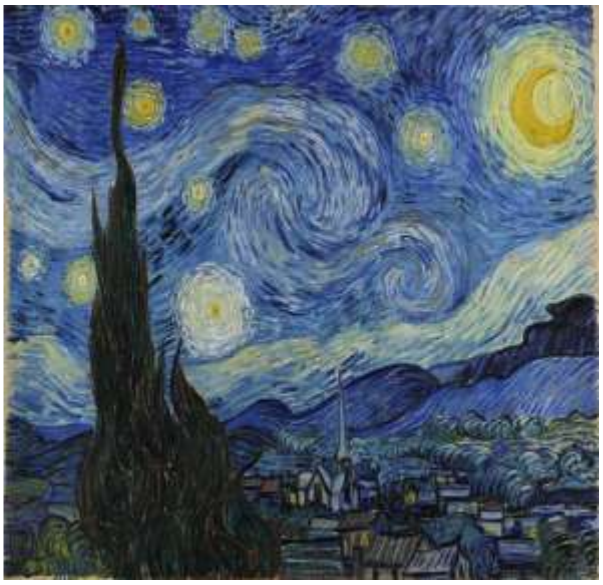


Figure 5.34: Style image

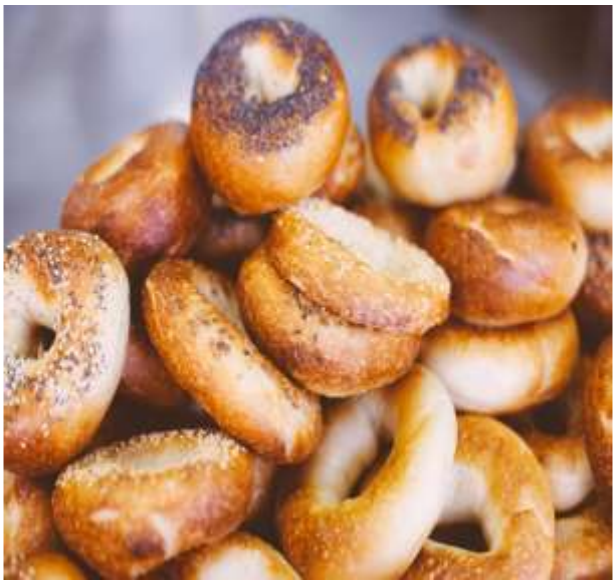


Figure 5.35: Content image

Output image

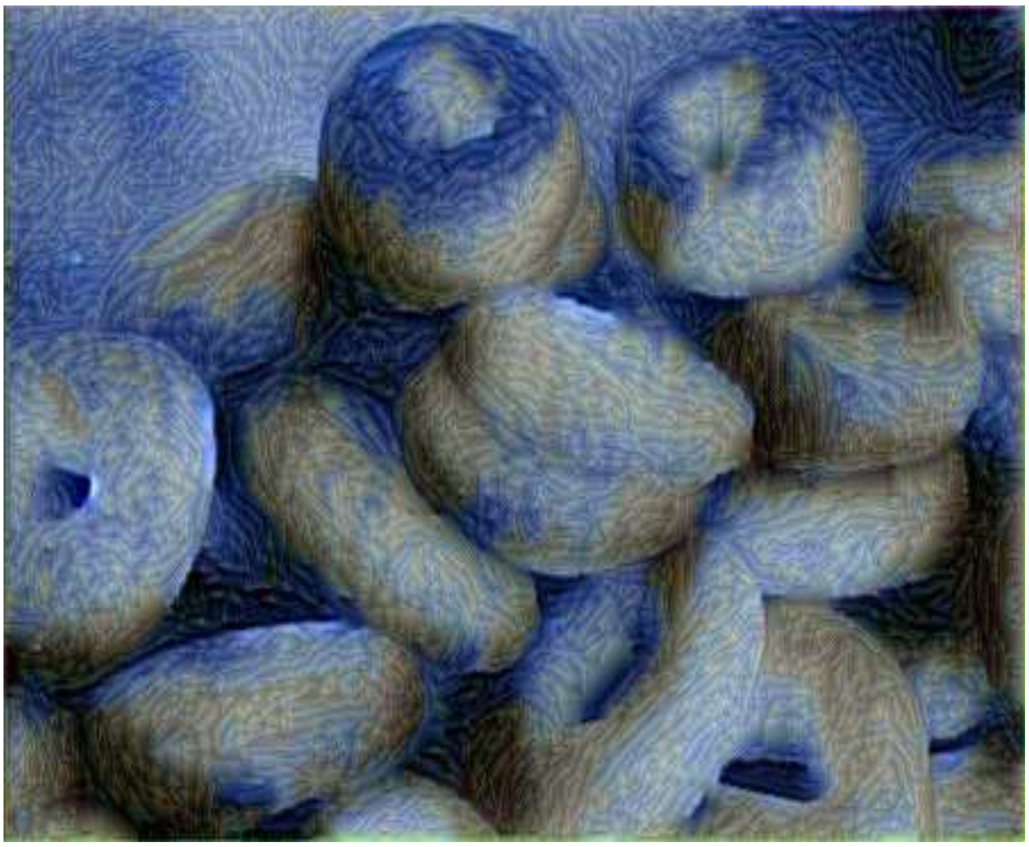


Figure 5.36: Output image

<u>The loss was :1.4845e+15</u>

*Third time we use content weight α= 1000, style weight β =1e6 and learning rate = 0.04.

Style Image

Content Image

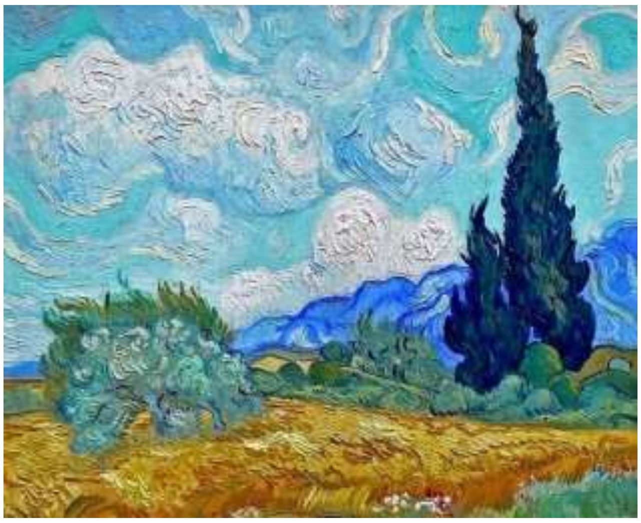


Figure 5.37: Style image

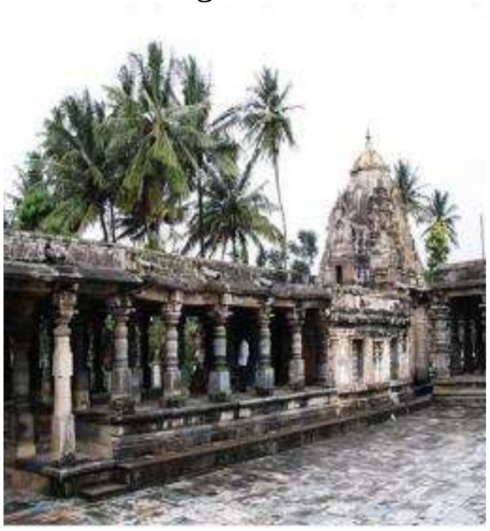


Figure 5.38: Content image

Output image

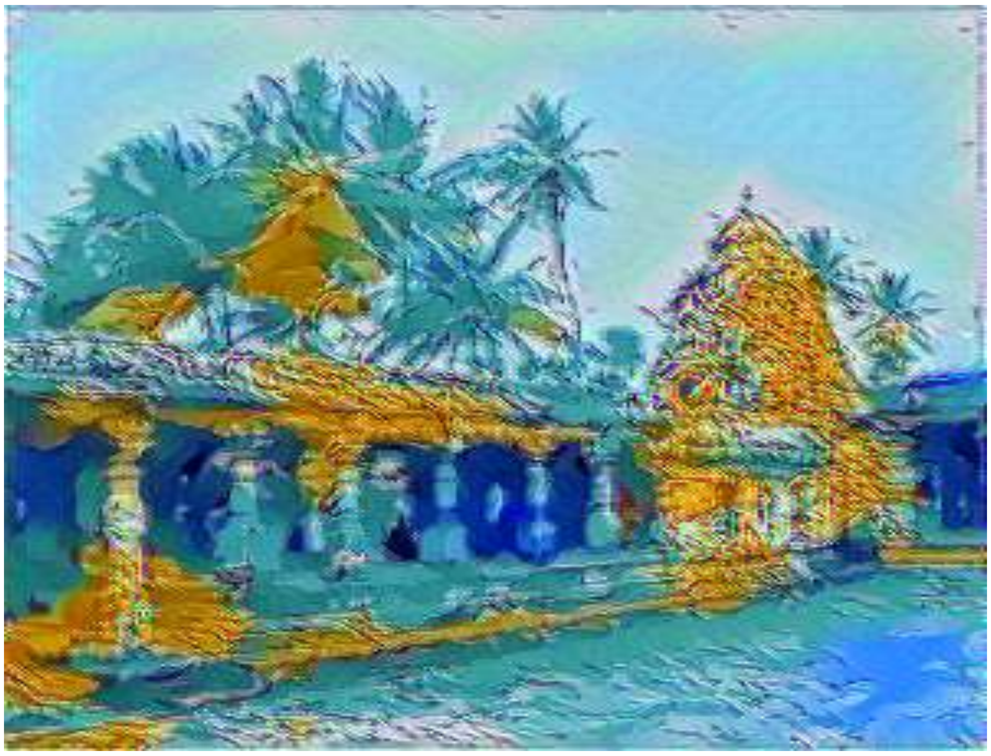


Figure 5.39: Output image

The loss was :2.5419e+14

Style Image

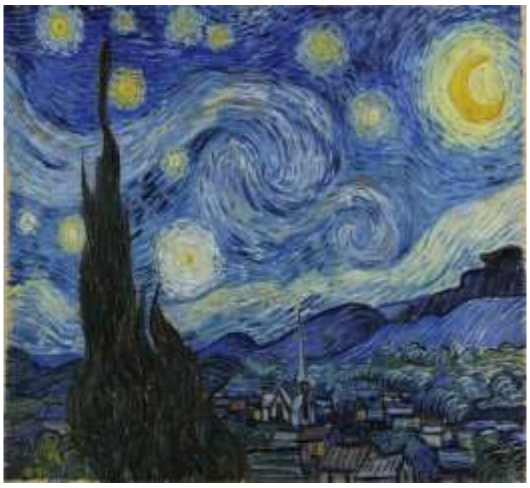


Figure 5.40: Style image

Content Image

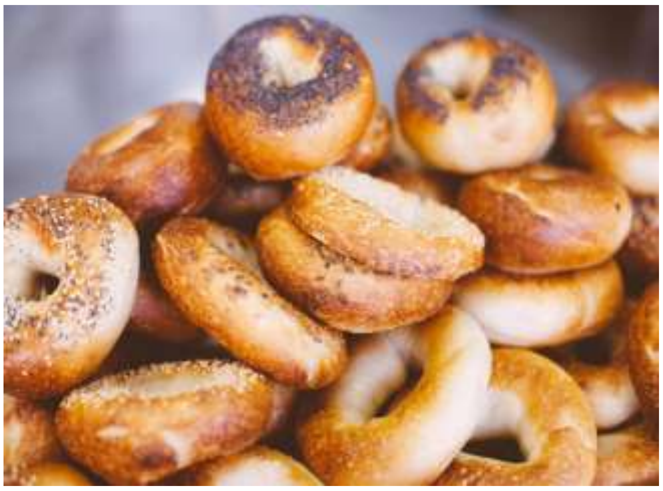


Figure 5.41: Content image

Output image

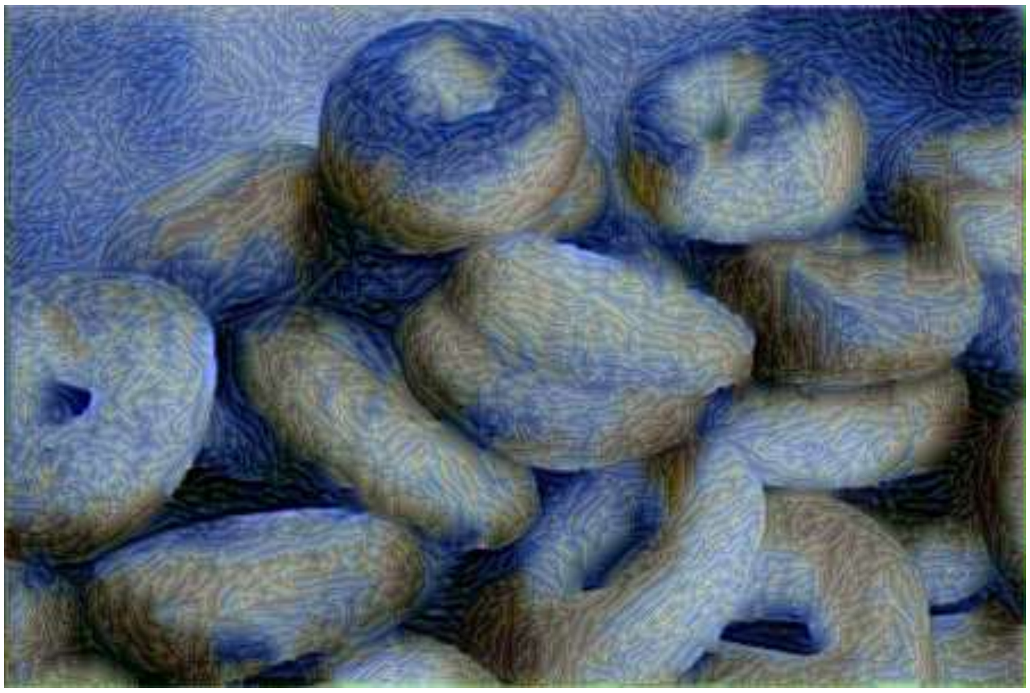


Figure 5.42: Output image

<u>**The loss was :1.5152e+15**</u>

*Fourth time we use content weight α=8, style weight β =70 and learning rate = 0.004.

Style Image

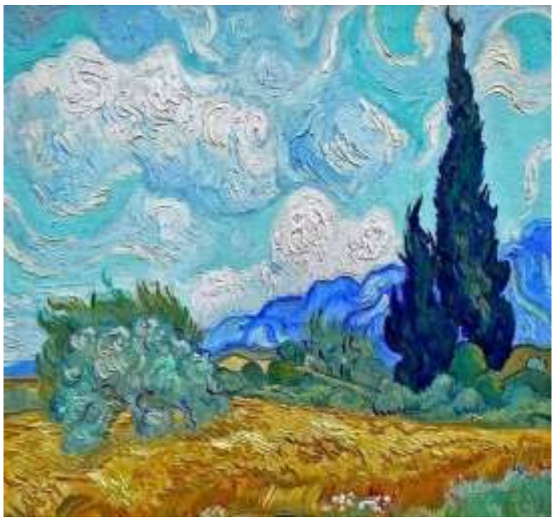


Figure 5.43: Style image

Content Image

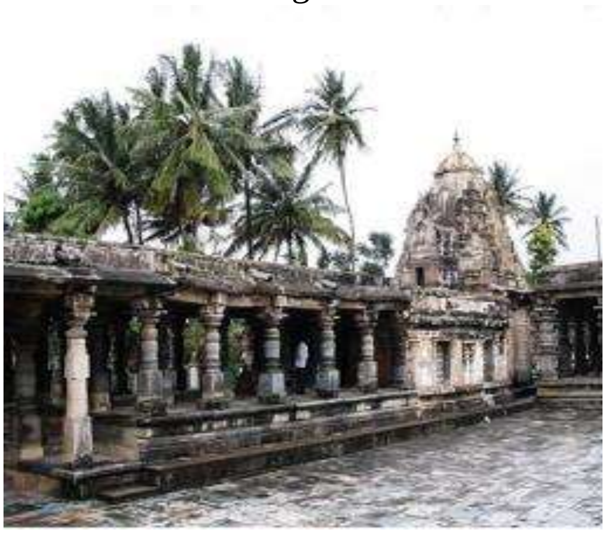


Figure 5.44: Content image

Output image

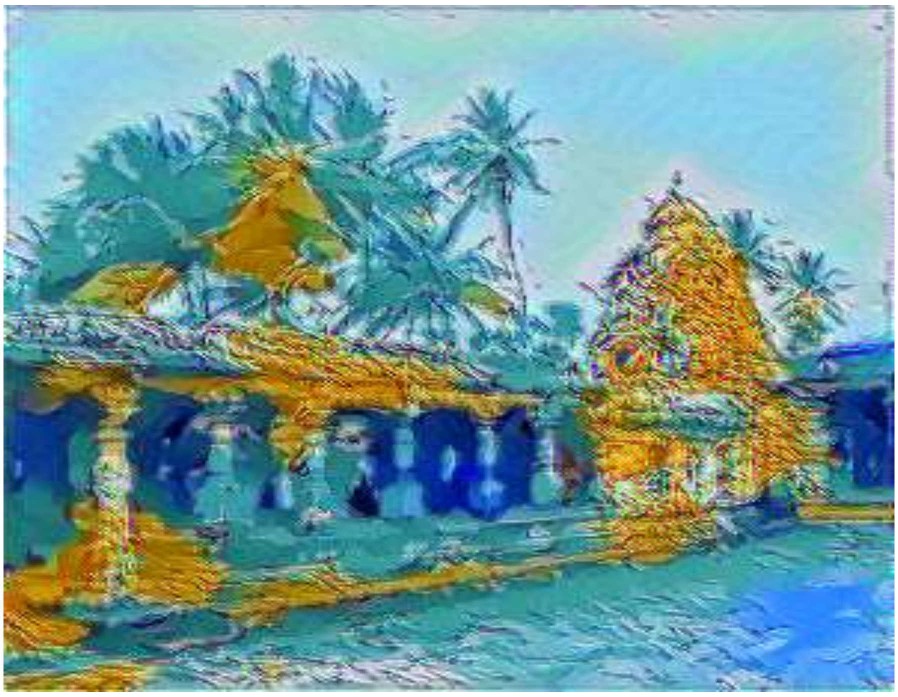


Figure 5.45: Output image

<u>The loss was :1.7799e+10</u>

Style Image Content Image

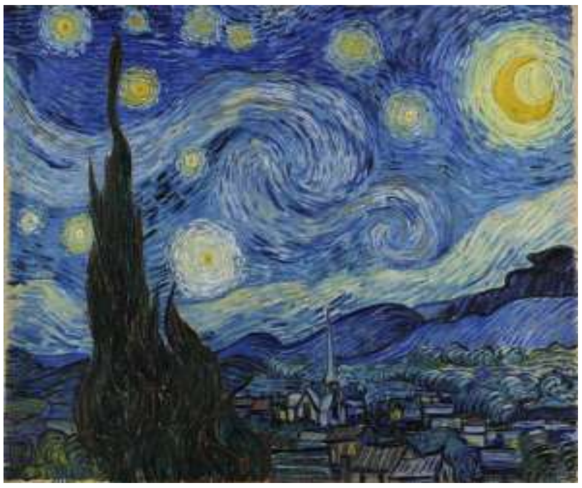


Figure 5.46: Style image

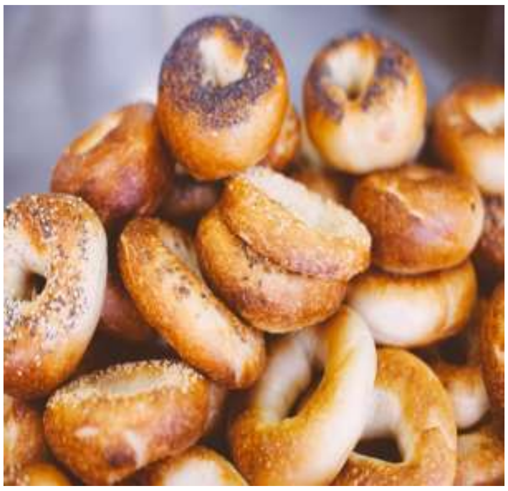


Figure 5.47: Content image

Output image

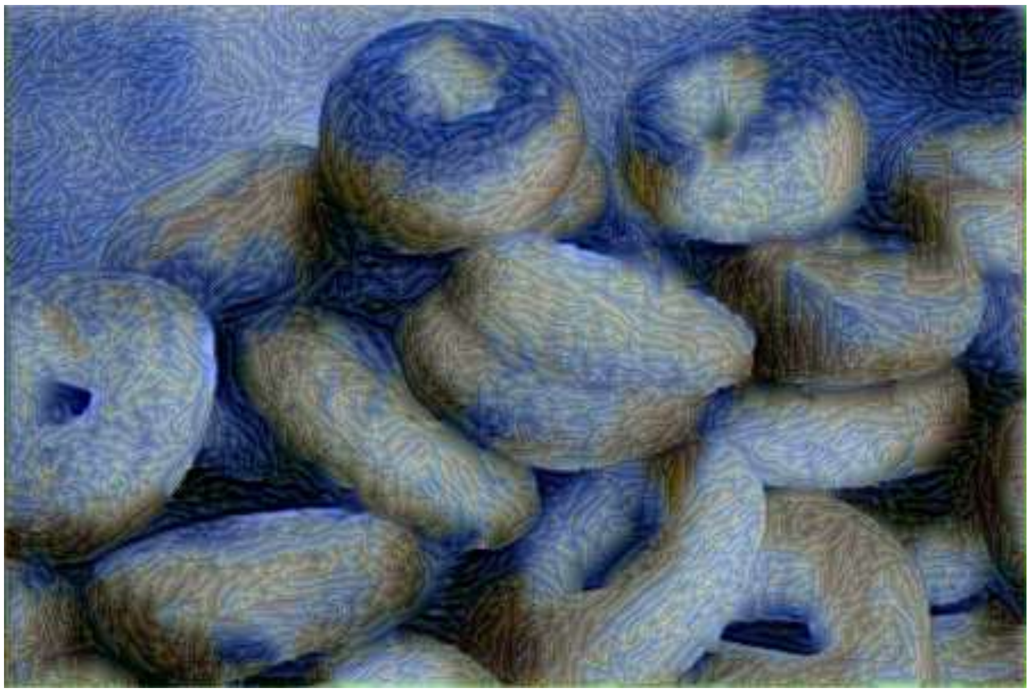


Figure 5.48: Output image

The loss was :1.0604e+11

Best model and best hyper parameters:

VGG16 Model: content weight α =8, style weight β =70 and learning rate = 0.004.

Chapter 6: User Interface

6.1 GUI

We integrate our system using Web development.

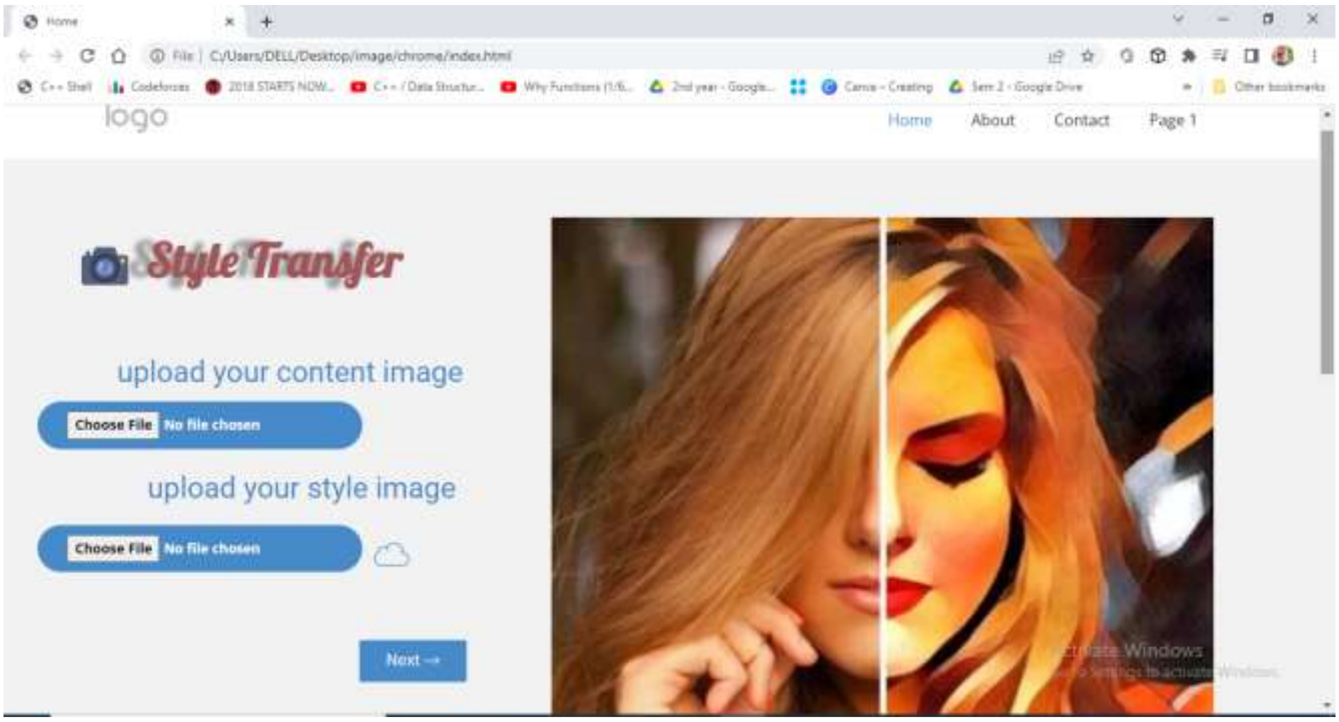

logo
Home
About
Contact
Page 1
Style Transfer
upload your content image
Choose File No file chosen
upload your style image
Choose File No file chosen
Next →

Figure 6.1: GUI 1

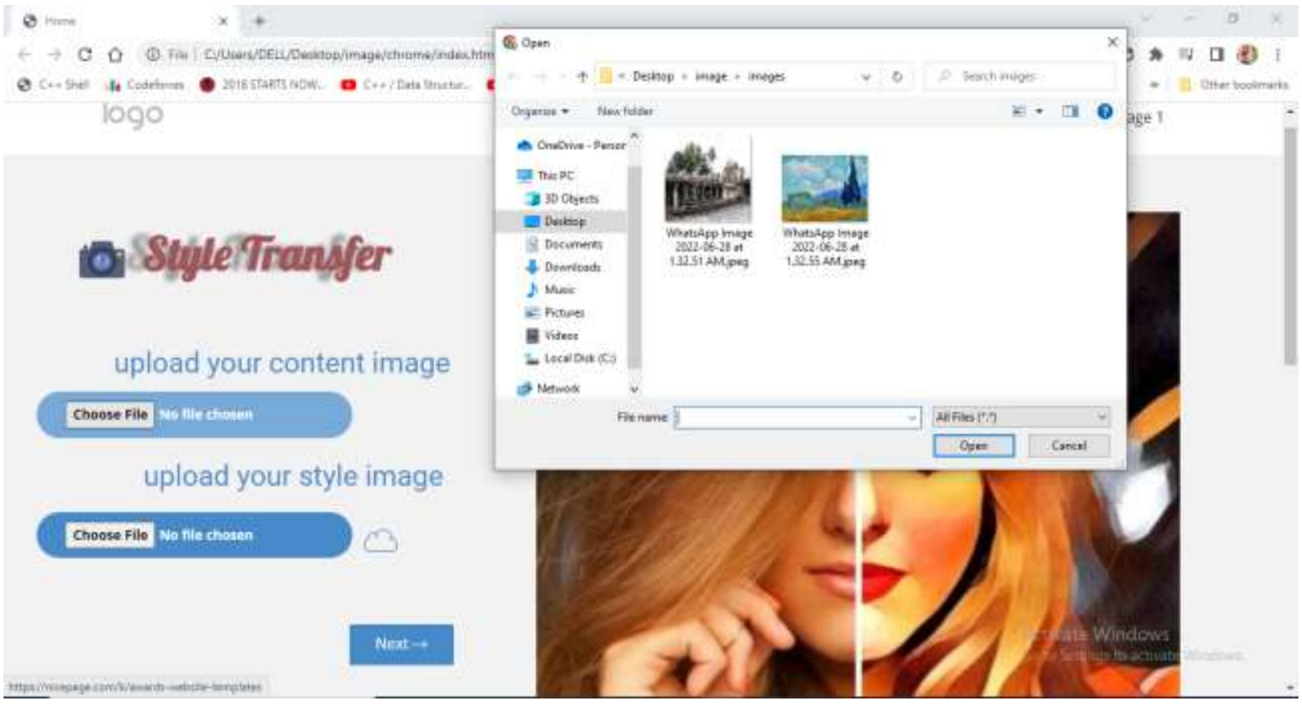


Figure 6.2: GUI 2

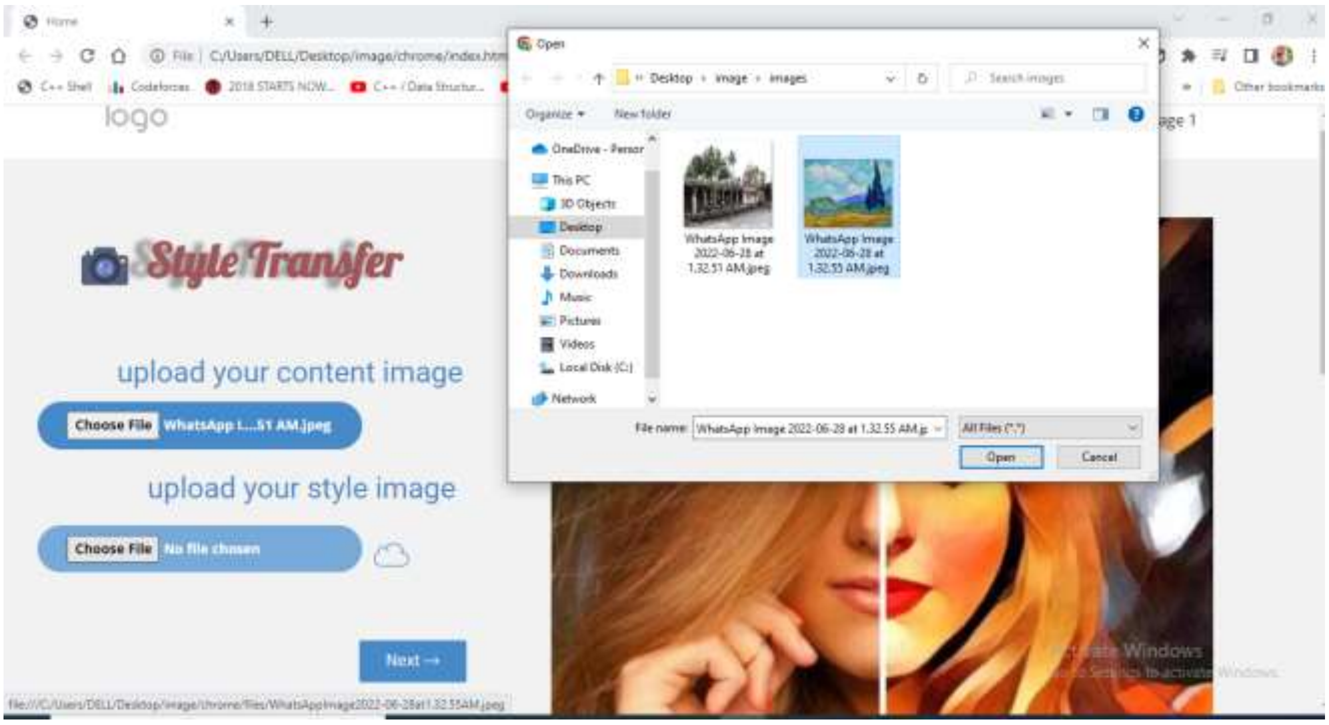

logo
Style Transfer
upload your content image
Choose File WhatsApp I....51 AM.jpeg
upload your style image
Choose File No file chosen
Next →
Open
Desktop › image › images
Organize ▾
New folder
WhatsApp Image 2022-06-28 at 1.32.51 AM.jpeg
WhatsApp Image 2022-06-28 at 1.32.55 AM.jpeg
File name: WhatsApp Image 2022-06-28 at 1.32.55 AM.jp
All Files (*.*)
Open
Cancel

Figure 6.3: GUI 3

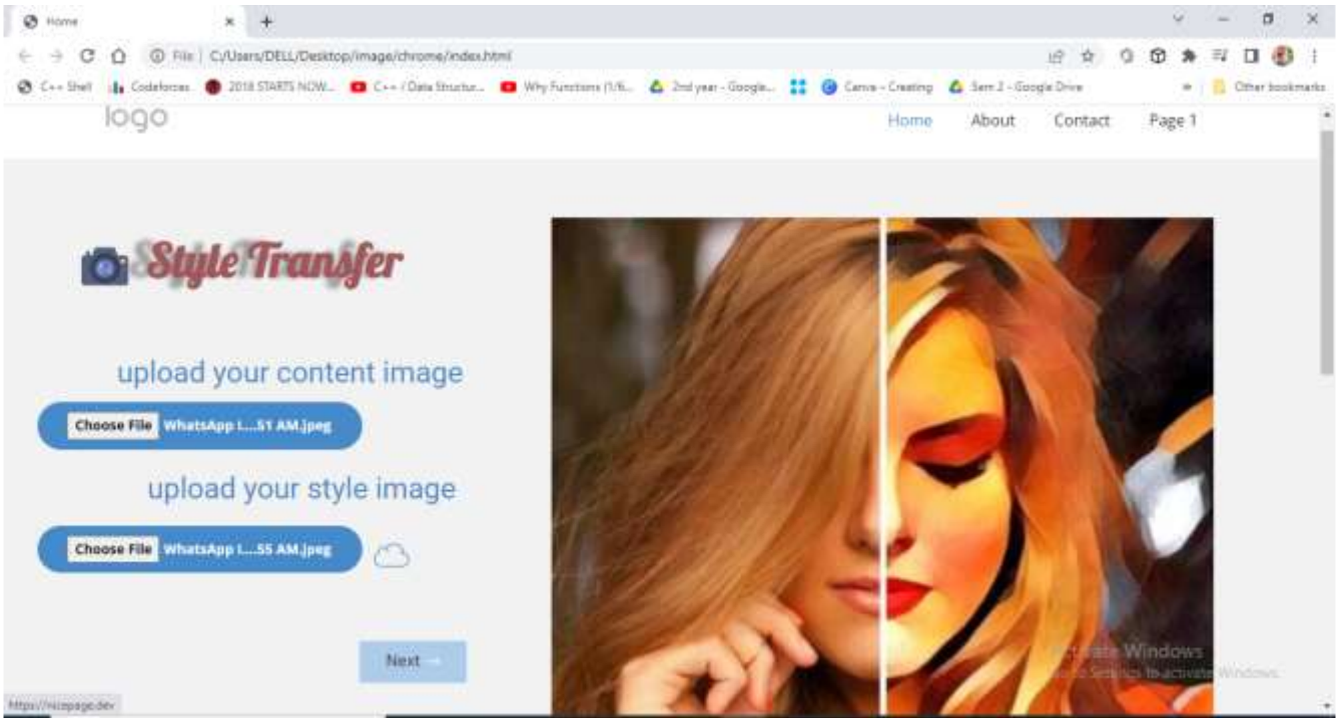

logo
Home
About
Contact
Page 1
Style Transfer
upload your content image
Choose File WhatsApp I....51 AM.jpeg
upload your style image
Choose File WhatsApp I....55 AM.jpeg
Next →

Figure 6.4: GUI 4

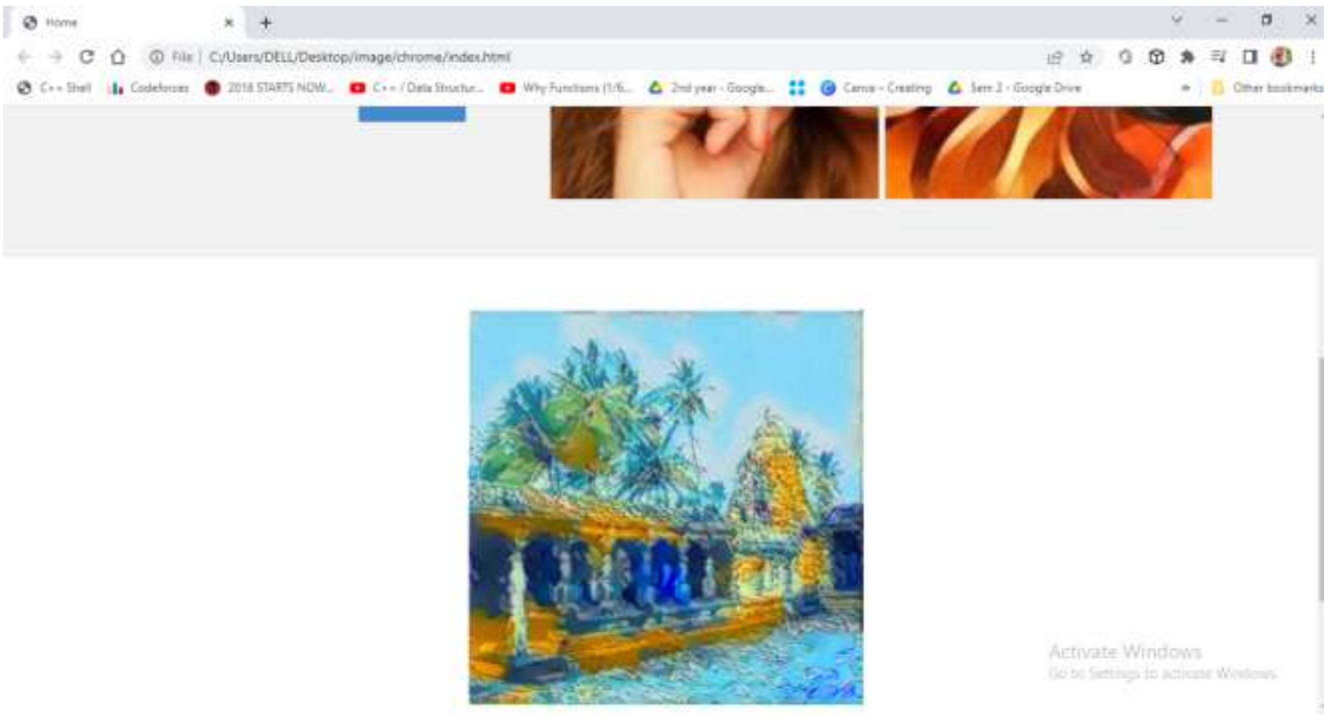

Home
C:/Users/DELL/Desktop/image/chrome/index.html
Activate Windows
Go to Settings to activate Windows

Figure 6.5: GUI 5

Chapter7:

Conclusion&Future Work

Conclusion:

we have learned the neural style transfer and built some intuitions. we have loaded a pretrained vgg16 model, froze its weight and customized the model layers. Loaded the images and performed some basic preprocessing.

Then defined the content and the style loss, which combined to calculate the total loss.

Finally, we ran the model and made an artistic image, which is the blend of the content and the style image.

<u>Future Work</u>

After what was mentioned above about our development, our Future works can focus on video styling transfer,

And, focus on object segmentations for better performance, especially for portraits. According to our observations, abstract paintings with clear outlines and impressionism art can often perform well as style images for portraits. Further testing and research should be done to testify this observation point.

References:

[1] Bruce, G., Amy, G. (2001) Non-Photorealistic rendering. AK Peters, Ltd., USA.

[2] Gatys, L. A., Ecker, A. S., Bethge, M. (2015) Texture synthesis using convolutional neural networks.

[3] Gatys, L. A., Ecker, A. S., Bethge, M. (2015) A neural algorithm of artistic style. Journal of Vision.

[4] Gatys, L. A., Ecker, A. S., Bethge, M. (2016) Image style transfer using convolutional neural networks. In: Computer Vision & Pattern Recognition.

[5] Lecun, Y., Bottou, and L., et al. (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE.

[6] Risser, E., Wilmot, P., Barnes, C. (2017) Stable and controllable neural texture synthesis and style transfer using histogram losses.

[7] Li, C., Wand, M. (2016) Combining markov random fields and convolutional neural networks for image synthesis. In: 29th IEEE Conference on Computer Vision and Pattern Recognition.

[8] Varun Gupta*, Rajat Sadana and Swastikaa Moudgil (2019) Image style transfer using convolutional neural networks based on transfer learning

[9] N. Ashikhmin. Fast texture transfer. IEEE Computer Graphics and Applications, 23(4):38–43, July 2003.

[10]L. A. Gatys, A. S. Ecker, and M. Bethge. Texture Synthesis Using Convolutional Neural Networks. In Advances in Neural Information Processing Systems 28, 2015.

[11] M. Berning, K. M. Boergens, and M. Helmstaedter. SegEM: Efficient Image Analysis for High-Resolution Connectomics. Neuron, 87(6):1193–1206, Sept. 2015.

[12] C. F. Cadieu, H. Hong, D. L. K. Yamins, N. Pinto, D. Ardila, E. A. Solomon, N. J. Majaj, and J. J. DiCarlo. Deep Neural Networks Rival the Representation of Primate IT Cor2421 tex for Core Visual Object Recognition. PLoSComput Biol, 10(12):e1003963, Dec. 2014.

[13] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. arXiv:1412.7062 [cs], Dec. 2014. arXiv: 1412.7062.

[14] 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.

[15] A. Mahendran and A. Vedaldi. Understanding Deep Image Representations by Inverting Them. arXiv:1412.0035 [cs], Nov. 2014. arXiv: 1412.0035.

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

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

[18] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua, "Stylebank: An explicit representation for neural image style transfer," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1897–1906.

[19] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for realtime style transfer and super-resolution," in European Conference on Computer Vision, 2016, pp. 694–711.

[20] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. pages 3431–3440, 2015.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.