Major Project

(ETIT - 450)

CREATING ART USING NEURAL STYLE TRANSFER

Submitted to the partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Information Technology

(2019 - 2023)

Submitted by

Amit Yadav (00220703119)

Pushkar Raja (01020703119)

Sumit Jha (02820703119)

Shivam Maurya (03520703119)

Supervised by

Dr. Sanjeev Kumar

Ch. Brahm Prakash Government Engineering College

Jaffarpur, Najafgarh, New Delhi – 110073

CH. BRAHM PRAKASH
GOVERNMENT ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

We hereby certify that the work which is being presented in the project, entitled "ART CREATION USING NEURAL STYLE TRANSFER" is in partial fulfillment of the requirements for the award of the Bachelor of Technology (Information Technology) submitted in Ch. Brahm Prakash Govt. Engineering College, is an original work, under the supervision of Dr. Sanjeev Kumar, Ch. Brahm Prakash Govt. Engineering College.

The matter presented in the project has not been submitted by us for the award of any other degree of this or any other place.

AMIT YADAV (00220703119)
PUSHKAR RAJA (01020703119)
SUMIT JHA (02820703119)
SHIVAM MAURYA (03520703119)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

SIGNATURE
Dr. Sanjeev Kumar
Dept. of Information Technology

Signature of the External Examiner

CERTIFICATE

The Final Project Viva-Voce examination of **AMIT YADAV (00220703119), PUSHKAR RAJA (01020703119), SUMIT JHA (02820703119), SHIVAM MAURYA (03520703119)** has been held on **1 July 2023**, and their work is recommended for the award of **Bachelors of Technology in Information Technology**.

Signature of Examiner(s)　　　　　　　　Signature of Supervisor(s)

Date: 1 July 2023
Place: Jaffarpur, Delhi

ABSTRACT

Neural style transfer is a fascinating computational technique that merges the style of one image with the content of another, resulting in visually compelling and artistically inspired compositions.

This project explores the application of neural networks to facilitate the seamless transfer of artistic styles onto arbitrary images.

By leveraging deep learning algorithms and convolutional neural networks, the project aims to bridge the gap between the content and style domains, enabling users to effortlessly infuse their images with the aesthetic qualities of famous artworks or unique artistic styles.

Through an iterative optimization process, the neural style transfer algorithm analyzes and manipulates the image representations to align the content while adopting the desired style characteristics.

The project delves into the implementation of various neural architecture designs, loss functions, and optimization strategies to enhance the quality and efficiency of style transfer. Additionally, user-friendly interfaces and web-based applications are developed to democratize the accessibility and usability of neural style transfer, empowering artists, photographers, and enthusiasts to unlock a new dimension of creative expression.

The results demonstrate the versatility and potential of neural style transfer in producing captivating and personalized visual outputs that merge the essence of multiple artistic worlds.

TABLE OF CONTENTS

Table of Contents

CHAPTER - 1

1.1 INTRODUCTION

Neural style transfer has emerged as a captivating and innovative technique that combines the power of artificial intelligence and artistic expression.
It enables the seamless merging of artistic styles onto arbitrary images, resulting in visually stunning compositions that blend the content of one image with the aesthetic qualities of another. This project explores the fascinating realm of neural style transfer and aims to leverage deep learning algorithms to facilitate the transformation of images into captivating pieces of art.

The concept of neural style transfer is rooted in the idea of extracting the content and style information from different images and merging them harmoniously.
By utilizing convolutional neural networks (CNNs) and advanced optimization techniques, this project seeks to bridge the gap between the domains of content and style, allowing users to effortlessly apply various artistic styles to their own images.

The potential applications of neural style transfer are vast and diverse. Artists and designers can leverage this technology to create unique and visually striking artworks that fuse different artistic influences. Photographers can explore new dimensions of creativity by blending photographic styles with famous painting techniques.
Additionally, neural style transfer can be utilized in areas such as visual effects, digital marketing, and user interface design to create captivating visuals that engage and captivate audiences.

In this project, we delve into the implementation and optimization of neural style transfer algorithms. We explore different neural network architectures, loss functions, and optimization strategies to enhance the quality and efficiency of style transfer. Additionally, we aim to develop user-friendly interfaces and web-based applications that make neural style transfer accessible and easy to use for artists, photographers, and enthusiasts.

Through this project, we aim to showcase the artistic potential of neural style transfer and its ability to transform ordinary images into extraordinary pieces of art. By harnessing the power of artificial intelligence and deep learning, we open up a new realm of creative expression, inviting users to explore, experiment, and redefine visual aesthetics in previously unimaginable ways.

1.2 PROBLEM FORMULATION

1.2.1 PROBLEM STATEMENT

The neural style transfer project aims to address the challenge of seamlessly merging artistic styles onto arbitrary images in a user-friendly and efficient manner. Traditional methods of manually recreating artistic styles or blending images require extensive manual effort and artistic expertise, limiting accessibility and hindering creative exploration. The project seeks to overcome these limitations by leveraging neural networks and deep learning techniques to automate the style transfer process, democratizing artistic expression and enabling users to effortlessly apply various artistic styles to their own images. The goal is to provide a solution that empowers artists, photographers, and enthusiasts to explore and experiment with different visual aesthetics, unleashing their creativity and producing visually captivating compositions without the need for specialized technical knowledge or time-consuming manual techniques.

1.2.2 PROPOSED SOLUTION

The solution is to automate and simplify the process of merging artistic styles onto arbitrary images. The application will leverage the power of neural networks and deep learning algorithms to enable seamless and efficient style transfer.

The core of the solution lies in the implementation of a convolutional neural network architecture capable of extracting both content and style information from input images. By training the network on a large dataset of diverse artistic styles and content images, it learns to capture the unique characteristics and features of different styles.

To facilitate the style transfer process, the application will provide an intuitive and interactive user interface. Users will be able to upload their own images and select from a range of pre-defined artistic styles or even upload their own style references. The application will then analyze the content of the input image and adapt it to adopt the desired artistic style. Users will have control over the degree of style transfer, allowing them to achieve the desired balance between content and style.

To ensure the quality and accuracy of style transfer, the solution will employ advanced optimization techniques. It will utilize loss functions that capture both the content similarity between the input image and the content reference and the style similarity between the input image and the style reference. Through an iterative optimization process, the network will adjust the image representations to align the content while adopting the artistic style characteristics, resulting in visually appealing and harmonious compositions.

The proposed solution will prioritize efficiency and accessibility, aiming to deliver near real-time style transfer results. By leveraging the computational power of modern GPUs and optimizing the algorithms, the application will provide a seamless and responsive user experience, allowing users to experiment with different styles and iterate their creative ideas quickly.

Overall, the proposed solution offers a user-friendly and efficient platform for neural style transfer, empowering artists, photographers, and enthusiasts to effortlessly merge artistic styles onto their images, unlock new dimensions of creative expression, and produce visually captivating compositions with ease.

CHAPTER - 2

LITERATURE REVIEW

Neural style transfer (NST) is a technique that combines the content of one image with the style of another to create visually appealing and artistic images. It has gained significant attention in recent years due to its ability to generate unique and visually captivating artwork. This literature review aims to explore the existing research and literature on the topic of creating art using neural style transfer.

2.1 Neural Style Transfer Techniques

Numerous approaches have been proposed for neural style transfer, each with its unique characteristics and advantages. Gatys et al. (2015) introduced seminal work on neural style transfer using convolutional neural networks (CNNs). Their method employed deep feature representations to capture content and style information. Since then, several variations and improvements have been proposed, including Fast Neural Style Transfer (Johnson et al., 2016) and Instance Normalization Style Transfer (Ulyanov et al., 2016), which enhance the efficiency and quality of style transfer.

2.2 Understanding Style and Content Representations

To achieve successful style transfer, a fundamental understanding of content and style representations is crucial. Li and Wand (2016) explored the importance of content and style representations in NST and proposed adaptive instance normalization to improve the quality of stylized images. Similarly, Dumoulin et al. (2016) introduced the concept of "Gram matrix" to capture style information effectively. These studies highlight the significance of feature representations and their impact on the final stylized output.

2.3 Improving Style Transfer Quality

Various studies have focused on improving the quality and fidelity of neural style transfer. Luan et al. (2017) introduced "Deep Photo Style Transfer" to preserve semantic content while transferring the style, resulting in more realistic and visually pleasing images. Furthermore, Li et al. (2017) proposed "Combining Markov Random Fields and Convolutional Neural Networks" to enhance the texture and spatial coherence of stylized images. These advancements contribute to the refinement and realism of artistic outputs.

2.4 Applications and Extensions

Beyond its artistic implications, neural style transfer has found applications in various domains. For instance, Zhu et al. (2017) explored the application of NST in creating personalized artworks from user sketches. Additionally, Nguyen et al. (2017) extended NST to video sequences, enabling style transfer in dynamic visual content. These applications demonstrate the versatility and potential of NST beyond static image generation.

2.5 Challenges and Future Directions

While neural style transfer has achieved remarkable progress, several challenges remain. One significant challenge is the computational complexity and time required for high-resolution image generation. Researchers, such as Johnson et al. (2016), have explored optimization techniques and parallel computing to mitigate these challenges. Furthermore, exploring the combination of NST with other generative models, such as generative adversarial networks (GANs), holds promise for generating more diverse and novel artistic outputs.

CHAPTER - 3

PROJECT DESCRIPTION

3.1 Project Overview

A web application that performs neural style transfer using PyTorch and the Flask framework. Neural style transfer is an artificial system based on the Deep Neural Network to generate artistic images. This approach uses two random images, the content and the style image. It extracts the structural features from the content image, whereas the style features from the style image.

3.2 Scope

The application will be accessible via a web browser, and will allow users to upload an input images – a content image and will combine it with a pre-uploaded style image – and generate a new image that combines the content of the content image with the style of the style image. The application will be implemented in Python, using the PyTorch library for machine learning and the Flask framework for web development.

3.3 Definitions

Neural style transfer: A technique for generating an image that combines the content of one image with the style of another image.

PyTorch: An open-source machine learning library for Python, developed by Facebook's AI Research lab.

Flask: A micro web framework for Python that is easy to use and lightweight.

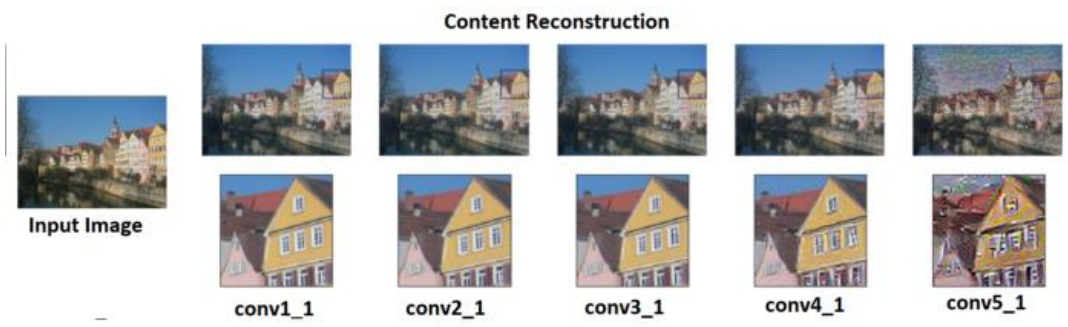
Deep Learning: It is a subset of Machine Learning based on Artificial Neural Networks. The main idea behind Deep Learning is to mimic the working of a human brain. Some of the use cases in Deep Learning involves **Face Recognition**, **Machine Translation**, **Speech Recognition**, etc. Learning can be supervised, semi-supervised, or unsupervised.

3.4 Overall Description

3.4.1 Content and Style Representation

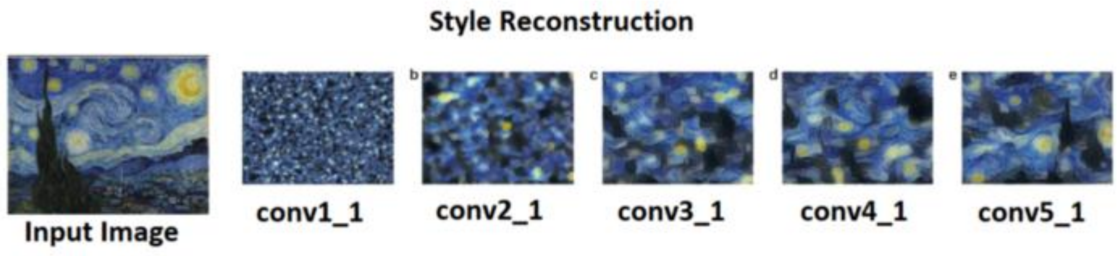
Content Representation

The convolution 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.



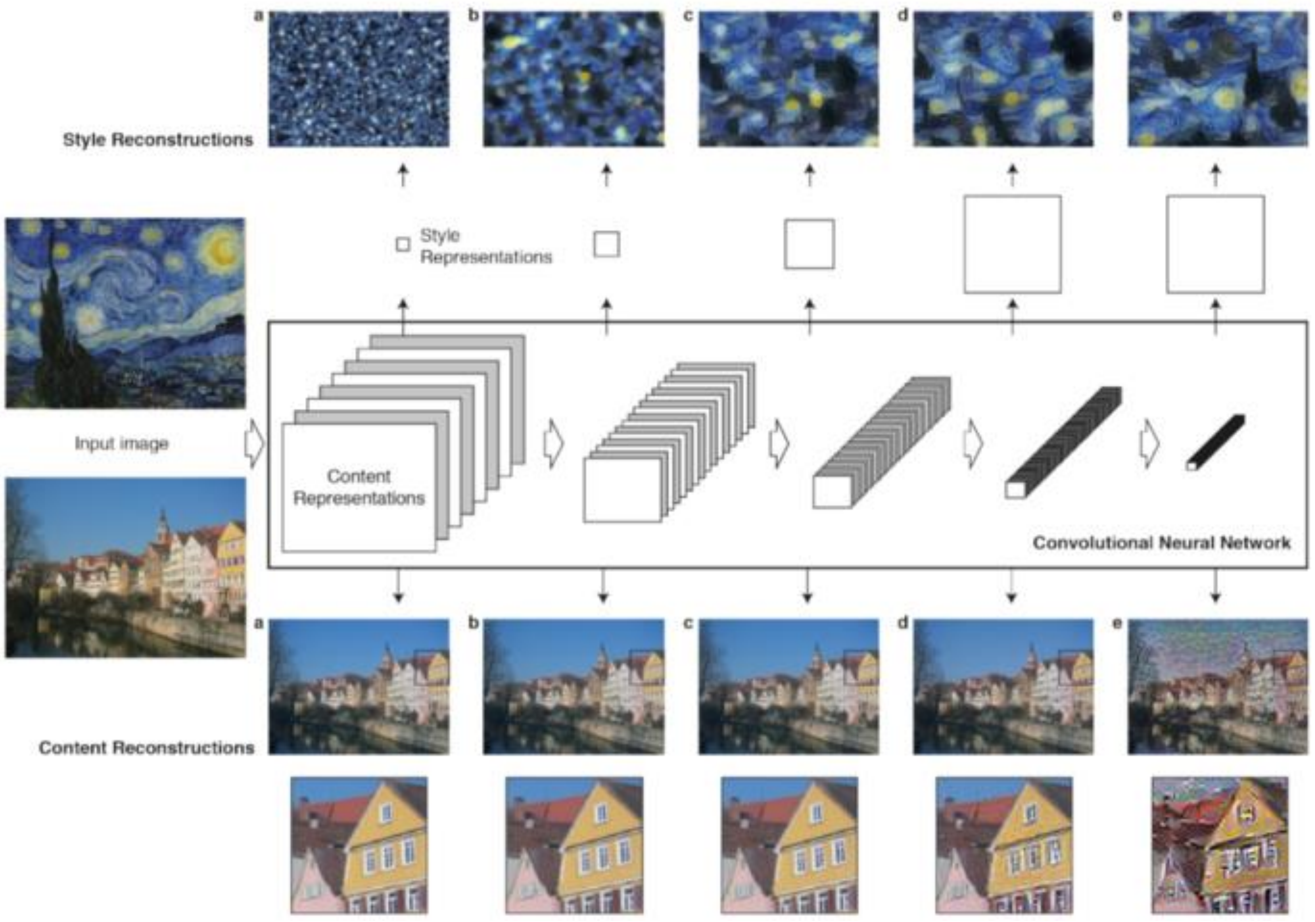
Style Representation

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.



3.4.2 The Model Architecture

Here, we use a pre-trained VGG19 network's convolution neural network and perform the content and style reconstructions. By entangling the structural information from the content representation and the texture/style information from the style representation, we generate the artistic image. We can emphasize either reconstructing the style or the content. A strong emphasis on style will result in images that match the artwork's appearance, effectively giving a textured 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. We perform the gradient descent on the generated image to find another image that matches the original image's feature responses.



CHAPTER - 4

WORKING/IMPLEMENTATION OF PROJECT

4.1 WORKING

Here's what the program can do:

1. Load the content image
2. Load the style image
3. Randomly initialize the image to be generated
4. Load the VGG19 model
5. Compute the content cost
6. Compute the style cost
7. Compute the total cost
8. Define the optimizer and learning rate

4.2 IMPLEMENTATION

4.2.1 Front End (HTML, CSS, JAVASCRIPT, FLASK)

Page for Loading Content Image:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Style</title>
</head>

<body style = "background-color: antiquewhite;">

    <form style="text-align: center; font-family:Verdana, Geneva, Tahoma, sans-serif ;"
id="upload-form"
        action="{{url_for('upload')}}" method="POST" enctype="multipart/form-data">
        <h1>Upload your image(s) and select a style</h1>
        <input type="file" name="file" accept="image/*" multiple>

        <br><br>

      <div>
            <label for="style">Choose an image style:</label>
            <select id="style" name="style">
                <option value="Fire_Style_22000_Iterations.t7">Fire Style</option>
                <option value="gold_black_2700.t7">Gold Black Style</option>
                <option value="pink_style_1800.t7">Pink Style</option>
                <option value="triangle_style_1000.t7">Triangle Style</option>
                <option value="flame_style_4500.t7">Flame Style</option>
                <option value="landscape_style_11600.t7">Landscape Style</option>
                <option value="rain_style_iter_4000.t7">Rain Style</option>
                <option value="starry_night_2500.t7">Starry Night Style</option>
                <option value="mosaic.t7">Mosaic Style</option>
                <option value="the_scream.t7">The Scream Style</option>
                <option value="the_wave.t7">The Wave Style</option>
            </select><br>

            <input type="submit" value="Upload">
        </div><br><br>

        <h2>How it works -</h2>
        <div>

            <img src="static\works.webp" width = 400>

        </div><br><br>


      <h2>You can choose from following style image's</h2>
```

```
        <div>

            <img src="static\fire.jpg" width=300>
            <img src="static\Rain.jpg" width=300>
            <img src="static\starrynight.jpg" width=300>
            <img src="static\the-scream.jpg" width=300>
            <img src="static\Tsunami_by_hokusai_19th_century.jpg" width=300>

        </div>

</body>

</html>
```

Final Art Rendering Page:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Style image</title>
</head>
<body style = "background-color: antiquewhite;">
    Upload complete!

    {% for image_name in image_names %}
        <div>
            <p>
            <label for = "original image" >Original image</label><br>
            <img  id = "original image" src="{{url_for('send_original_image',
filename=image_name)}}", width = 600px>
            <label for = "Changed image" >Changed image</label>
            <img id = "changed image" src="{{url_for('send_processed_image',
filename=image_name, selected_style=selected_style)}}">
        </div>
    {% endfor %}

</body>
</body>
</html>
```

Flask Controller:

```
# Import required packages:
from flask import Flask, request, render_template, send_from_directory, redirect,
send_file
import os
import test
import neuralStyleProcess
import cv2

app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath(__file__))

@app.route("/")
def index():
    return render_template("upload.html")

@app.route("/upload", methods=['POST'])
def upload():
    target = os.path.join(APP_ROOT, 'images/')
    print("TARGET", target)

    if not os.path.isdir(target):
        os.mkdir(target)
    else:
        print("Couldn't create upload directory: {}".format(target))

    data = request.form.get("style")
    print(data)

    myFiles = []

    for file in request.files.getlist("file"):
        print("file", file)
        filename = file.filename
        print("filename", filename)
        destination = "".join([target, filename])
        print("destination", destination)
        file.save(destination)
        myFiles.append(filename)
    print(myFiles)

    return render_template("complete.html", image_names=myFiles, selected_style=data)

# in this function send_image will HAVE to take in the parameter name <filename>
@app.route('/upload/<filename>')
def send_original_image(filename):
    return send_from_directory("images", filename)

# this app route cant be the same as above
@app.route('/complete/<filename>/<selected_style>')
def send_processed_image(filename, selected_style):
    directoryName = os.path.join(APP_ROOT, 'images/')
    newImg = neuralStyleProcess.neuralStyleTransfer(directoryName, filename,
selected_style)

    return send_from_directory("images", newImg)

if __name__ == "__main__":
    app.run(debug=True)
```

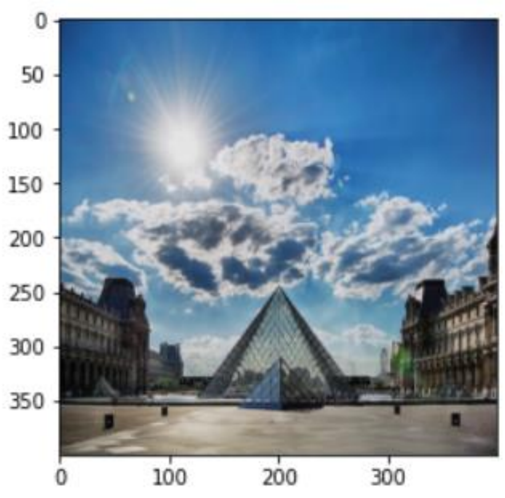
Neural Style Process:

```
import imutils
import cv2
import os
import numpy as np

APP_ROOT = os.path.dirname(os.path.abspath(__file__))

def neuralStyleTransfer(directoryName, filename, selected_style):

    # Neural style transfer codes adapted from pyimageSearch

    # load the neural style transfer model from disk
    target = os.path.join(APP_ROOT, 'static/models/')
    net = cv2.dnn.readNetFromTorch(target + selected_style)

    # load the input image, resize it to have a width of 600 pixels, and
    # then grab the image dimensions
    image = cv2.imread(directoryName+filename)
    image = imutils.resize(image, width=600)
    (h, w) = image.shape[:2]

    # construct a blob from the image, set the input, and then perform a
    # forward pass of the network
    blob = cv2.dnn.blobFromImage(image, 1.0, (w, h),
        (103.939, 116.779, 123.680), swapRB=False, crop=False)
    net.setInput(blob)

    output = net.forward()

    # reshape the output tensor, add back in the mean subtraction, and
    # then swap the channel ordering
    output = output.reshape((3, output.shape[2], output.shape[3]))
    output[0] += 103.939
    output[1] += 116.779
    output[2] += 123.680
    #uncomment this for imshow etc to display (if running as a script)
    #output /= 255.0
    output = output.transpose(1, 2, 0)

    #cv2.imshow('image',output)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()
    #output = (output * 255).astype(np.uint8)

    filename, file_extension = os.path.splitext(filename)
    print(filename)
    newFileName = 'processedImg'+ '_' + filename + file_extension
    cv2.imwrite(directoryName + newFileName, output)
    print(newFileName)
    print(directoryName)

    return newFileName

if __name__=='__main__':
    #swapT with output/=255 doesnt work
    #swapT without output/=255 doesnt look nice
    #swapF with output/=255 doesnt work
    #swapF without output/=255 looks best
    neuralStyleTransfer('/Users/fangran/Documents/pyFlaskCV/images/', 'bbq.jpg',
'gold_black_2700.t7')
```

4.2.2 **Solving the Optimization Problem**

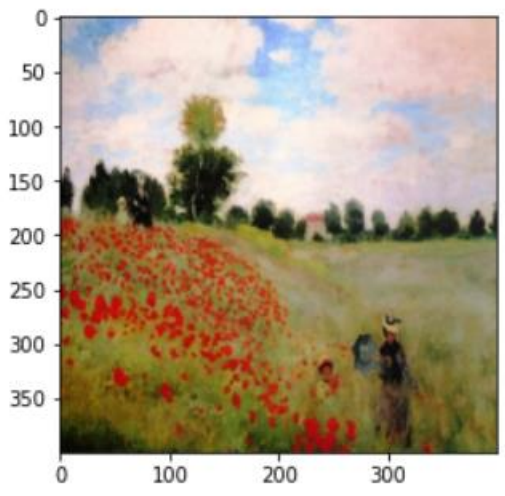
1. Load the Content Image:

```
content_image = np.array(Image.open("images/louvre_small.jpg").resize((img_size,
img_size)))
content_image = tf.constant(np.reshape(content_image, ((1,) + content_image.shape)))

print(content_image.shape)
imshow(content_image[0])
plt.show()
```



2. Load the Style Image:

```
style_image =  np.array(Image.open("images/monet.jpg").resize((img_size, img_size)))
style_image = tf.constant(np.reshape(style_image, ((1,) + style_image.shape)))

print(style_image.shape)
imshow(style_image[0])
plt.show()
```

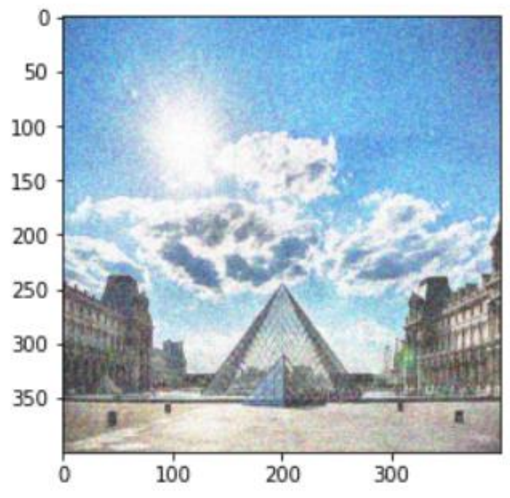


3. Randomly Initialize the Image to be Generated:

Now we will initialize the "generated" image as a noisy image created from the content_image.

- The generated image is slightly correlated with the content image.
- By initializing the pixels of the generated image to be mostly noise but slightly correlated with the content image, this will help the content of the "generated" image more rapidly match the content of the "content" image.

```
generated_image = tf.Variable(tf.image.convert_image_dtype(content_image, tf.float32))
noise = tf.random.uniform(tf.shape(generated_image), 0, 0.5)
generated_image = tf.add(generated_image, noise)
generated_image = tf.clip_by_value(generated_image, clip_value_min=0.0,
clip_value_max=1.0)

print(generated_image.shape)
imshow(generated_image.numpy()[0])
plt.show()
```



4. Load Pre-trained VGG19 Model:

defining a function which loads the VGG19 model and returns a list of the outputs for the middle layers.

```
def get_layer_outputs(vgg, layer_names):
    """ Creates a vgg model that returns a list of intermediate output values."""
    outputs = [vgg.get_layer(layer[0]).output for layer in layer_names]

    model = tf.keras.Model([vgg.input], outputs)
    return model
```

Now, we will define the content layer and build the model.

```
content_layer = [('block5_conv4', 1)]

vgg_model_outputs = get_layer_outputs(vgg, STYLE_LAYERS + content_layer)
```

Saving the outputs for the content and style layers in separate variables.

```
content_target = vgg_model_outputs(content_image)  # Content encoder
style_targets = vgg_model_outputs(style_image)     # Style enconder
```

5. Compute Total Cost:

5.1 Compute Content Cost:

We've built the model, and now to compute the content cost, we will now assign `a_C` and `a_G` to be the appropriate hidden layer activations. You will use layer `block5_conv4` to compute the content cost. The code below does the following:

1. Set a_C to be the tensor giving the hidden layer activation for layer "block5_conv4".
2. Set a_G to be the tensor giving the hidden layer activation for the same layer.
3. Compute the content cost using a_C and a_G.

```
# Assign the content image to be the input of the VGG model.
# Set a_C to be the hidden layer activation from the layer we have selected
preprocessed_content =  tf.Variable(tf.image.convert_image_dtype(content_image,
tf.float32))
a_C = vgg_model_outputs(preprocessed_content)

# Set a_G to be the hidden layer activation from same layer. Here, a_G references
model['conv4_2']
# and isn't evaluated yet. Later in the code, we'll assign the image G as the model
input.
a_G = vgg_model_outputs(generated_image)

# Compute the content cost
J_content = compute_content_cost(a_C, a_G)

print(J_content)
```

5.2 Compute Style Cost:

The code below sets a_S to be the tensor giving the hidden layer activation for `STYLE_LAYERS`:

```
# Assign the input of the model to be the "style" image
preprocessed_style =  tf.Variable(tf.image.convert_image_dtype(style_image,
tf.float32))
a_S = vgg_model_outputs(preprocessed_style)

# Compute the style cost
J_style = compute_style_cost(a_S, a_G)
print(J_style)
```

Below are the utils that will display the images generated by the style transfer model:

```
def clip_0_1(image):
    """
    Truncate all the pixels in the tensor to be between 0 and 1

    Arguments:
    image -- Tensor
    J_style -- style cost coded above

    Returns:
    Tensor
    """
    return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)

def tensor_to_image(tensor):
    """
    Converts the given tensor into a PIL image

    Arguments:
    tensor -- Tensor

    Returns:
    Image: A PIL image
    """
    tensor = tensor * 255
    tensor = np.array(tensor, dtype=np.uint8)
    if np.ndim(tensor) > 3:
        assert tensor.shape[0] == 1
        tensor = tensor[0]
    return Image.fromarray(tensor)
```

6. Calculating the Loss

The net loss for style transfer is defined as :

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

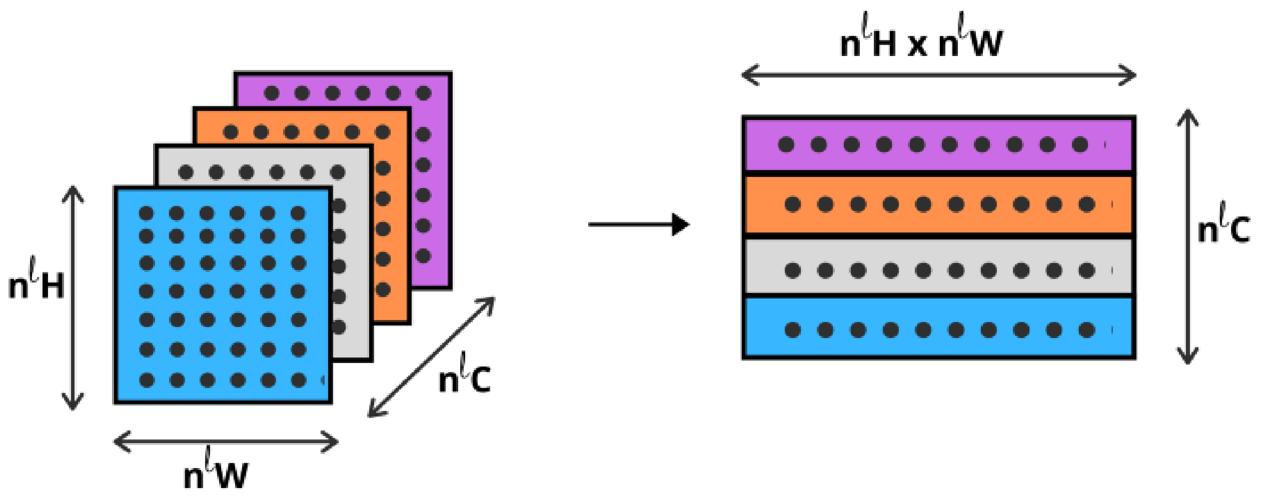
In the above equation, L_{total} is the total loss, L c_{ontent} is the content loss of all the intermediate layers and L_{style} is the style loss of all the intermediate layers. Here, α and β are the weighting coefficients of the content and the style loss, respectively. p, a and x are the content 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.

6.1 Content Loss

The content image and the input base image are passed to our model and the intermediate layers' outputs (listed above as ‘conv1_1’, ‘conv2_1’, ‘conv3_1’, ‘conv4_1’ and ‘conv5_1’) are extracted using the above-defined class. Then we calculate the Euclidean distance between the intermediate representation of the content image and the input base image. Hence the content loss for a layer l is defined by:

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

In the above equation, we calculate the squared error between the feature representation of the content image (p) and the input base image (x) of layer (l). Here, n^l_h, $n^l w$, $n^l c$ are the height, width and channels of layer l. To calculate the content loss, the intermediate representation of dimensions n^l_h x $n^l w$ x $n^l c$ is unrolled into vectors of dimensions $n^l c$ x $n^l_h * n^l w$. Unrolling the feature representation is not a compulsory step, but it's a good practice. The following diagram will help us to visualize this transformation.



F^l_{ij} and P^l_{ij} are the $n^l c$ x $n^l_h * n^l w$ dimension vectors representing the intermediate representation of the input base image and the content image.

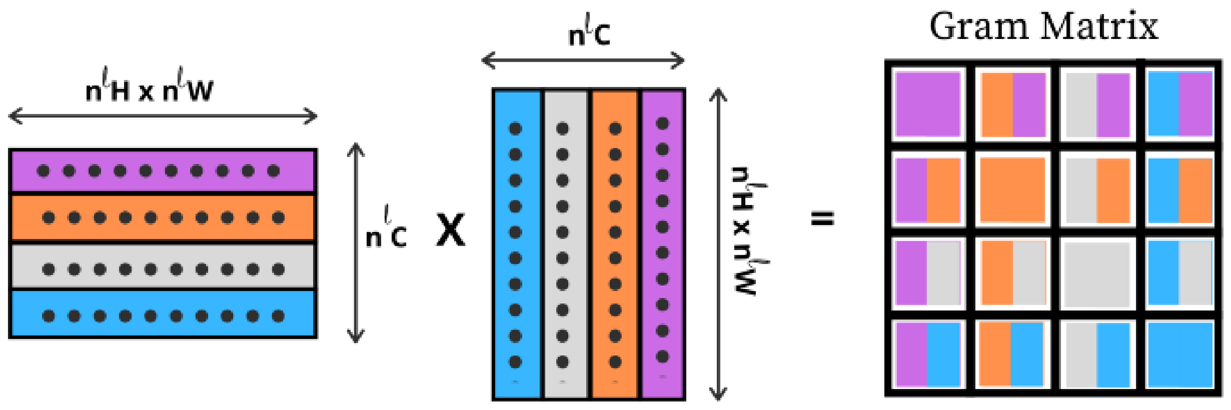
6.2 Style Loss

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 above 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.



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

The style loss of layer l is the squared error between the gram matrices of the intermediate representation of the style image and the input base image.

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

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.

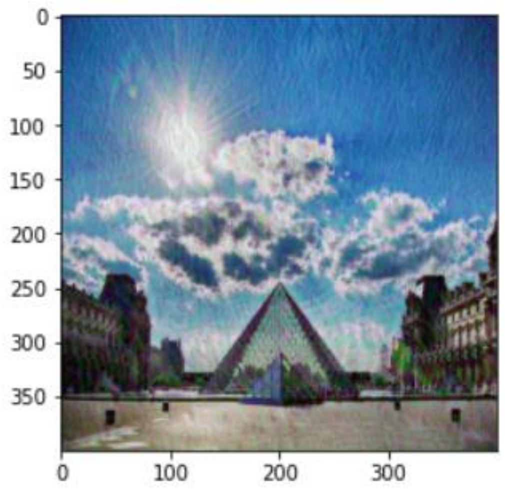
7. Train the Model:

We will run the following cell to generate an artistic image. It should take about 3min on a GPU for 2500 iterations. Neural Style Transfer is generally trained using GPUs.

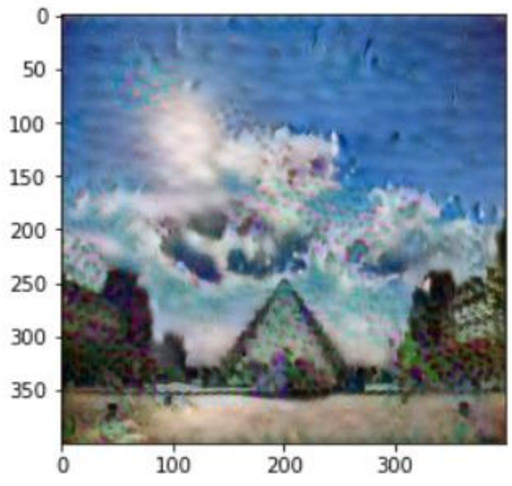
If we increase the learning rate we can speed up the style transfer, but often at the cost of quality.

```
# Uncoment to reset the style transfer process. You will need to compile the train_step
function again
epochs = 2501
for i in range(epochs):
    train_step(generated_image)
    if i % 250 == 0:
        print(f"Epoch {i} ")
    if i % 250 == 0:
        image = tensor_to_image(generated_image)
        imshow(image)
        image.save(f"output/image_{i}.jpg")
        plt.show()
```

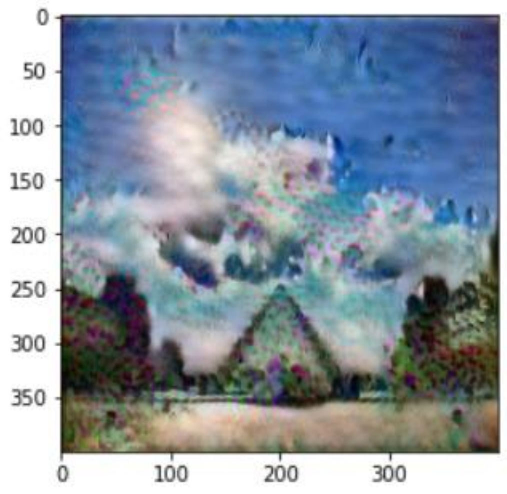
```
Epoch 0
```



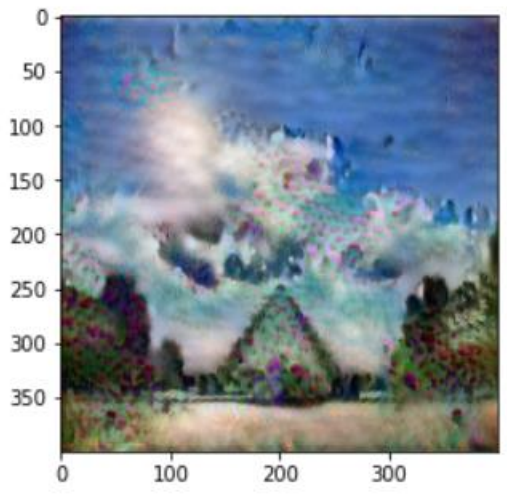
```
Epoch 250
```



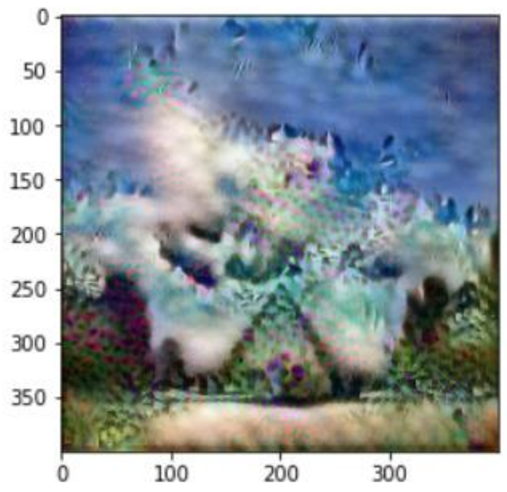
```
Epoch 500
```



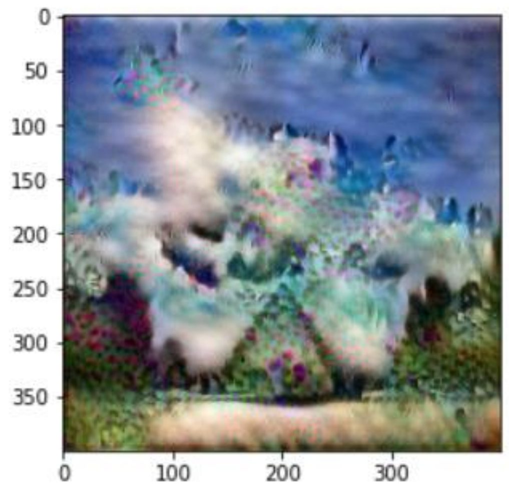
```
Epoch 750
```



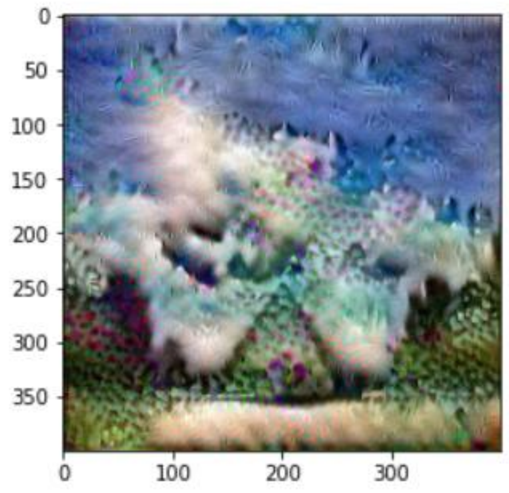
```
Epoch 1000
```



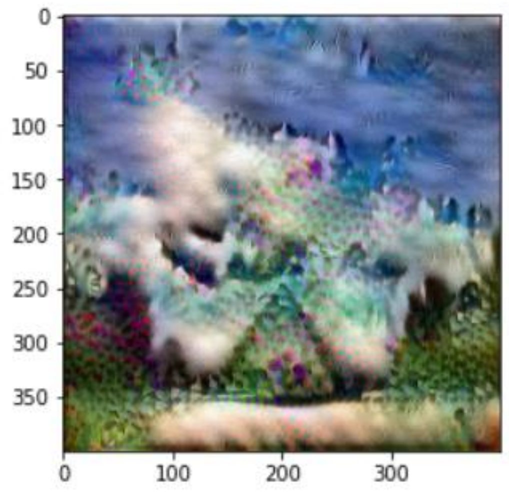
```
Epoch 1250
```



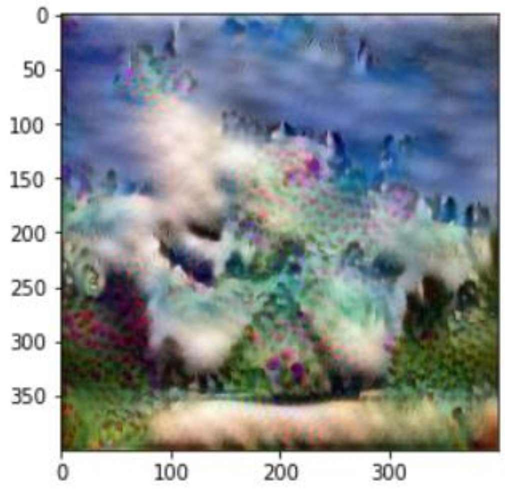
```
Epoch 1500
```



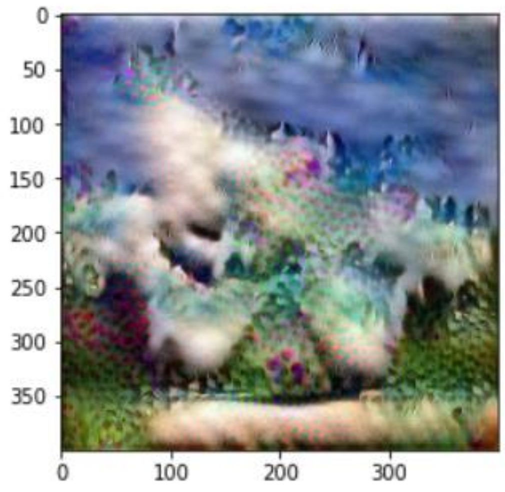
```
Epoch 1750
```



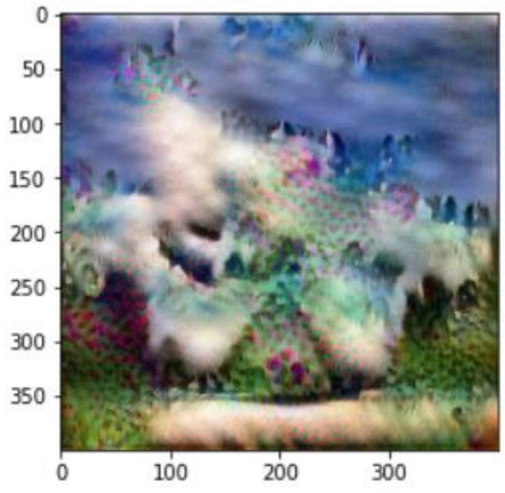
```
Epoch 2000
```



```
Epoch 2250
```

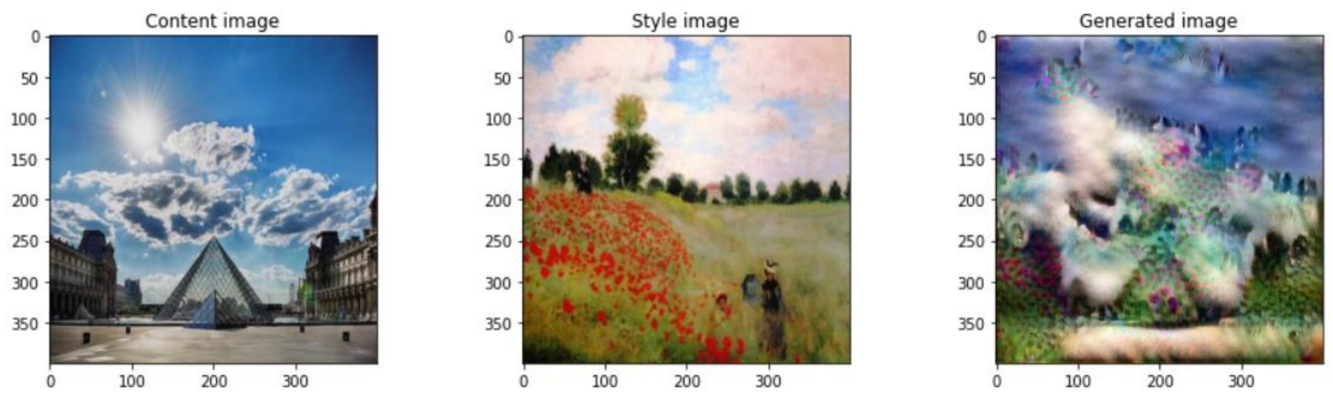


```
Epoch 2500
```



Now, we will run the following code cell to see the results!

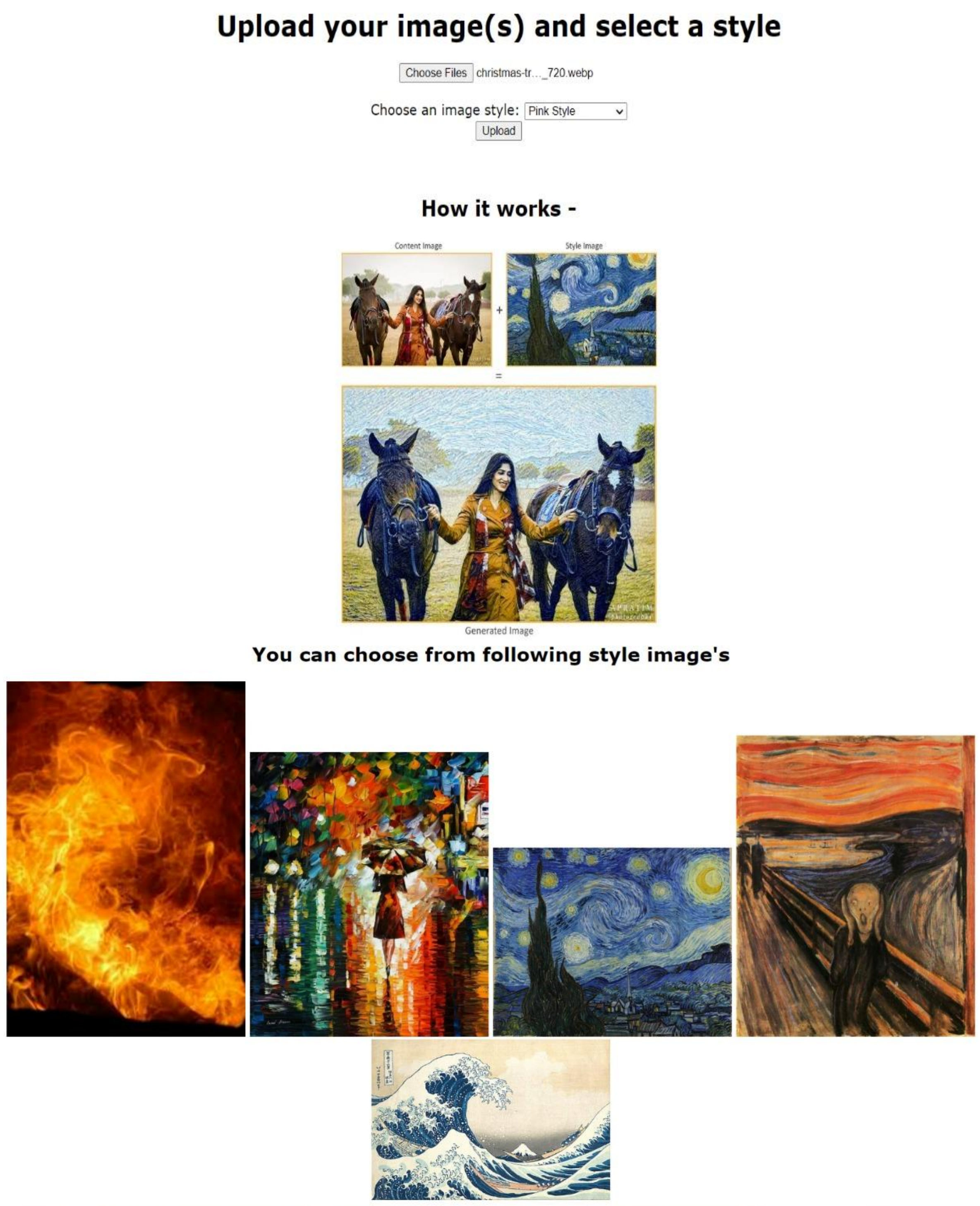
```python
# Show the 3 images in a row
fig = plt.figure(figsize=(16, 4))
ax = fig.add_subplot(1, 3, 1)
imshow(content_image[0])
ax.title.set_text('Content image')
ax = fig.add_subplot(1, 3, 2)
imshow(style_image[0])
ax.title.set_text('Style image')
ax = fig.add_subplot(1, 3, 3)
imshow(generated_image[0])
ax.title.set_text('Generated image')
plt.show()
```



CHAPTER - 5

OUTPUTS

5.1 User Interface


Upload your image(s) and select a style

Choose Files christmas-tr..._720.webp

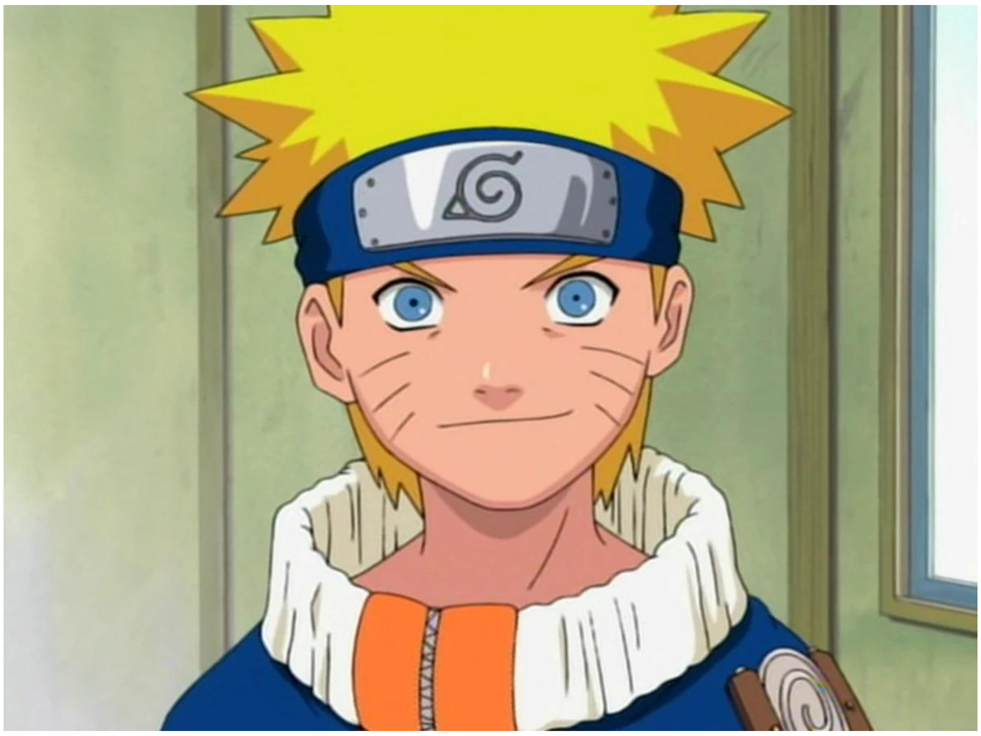
Choose an image style: Pink Style

Upload

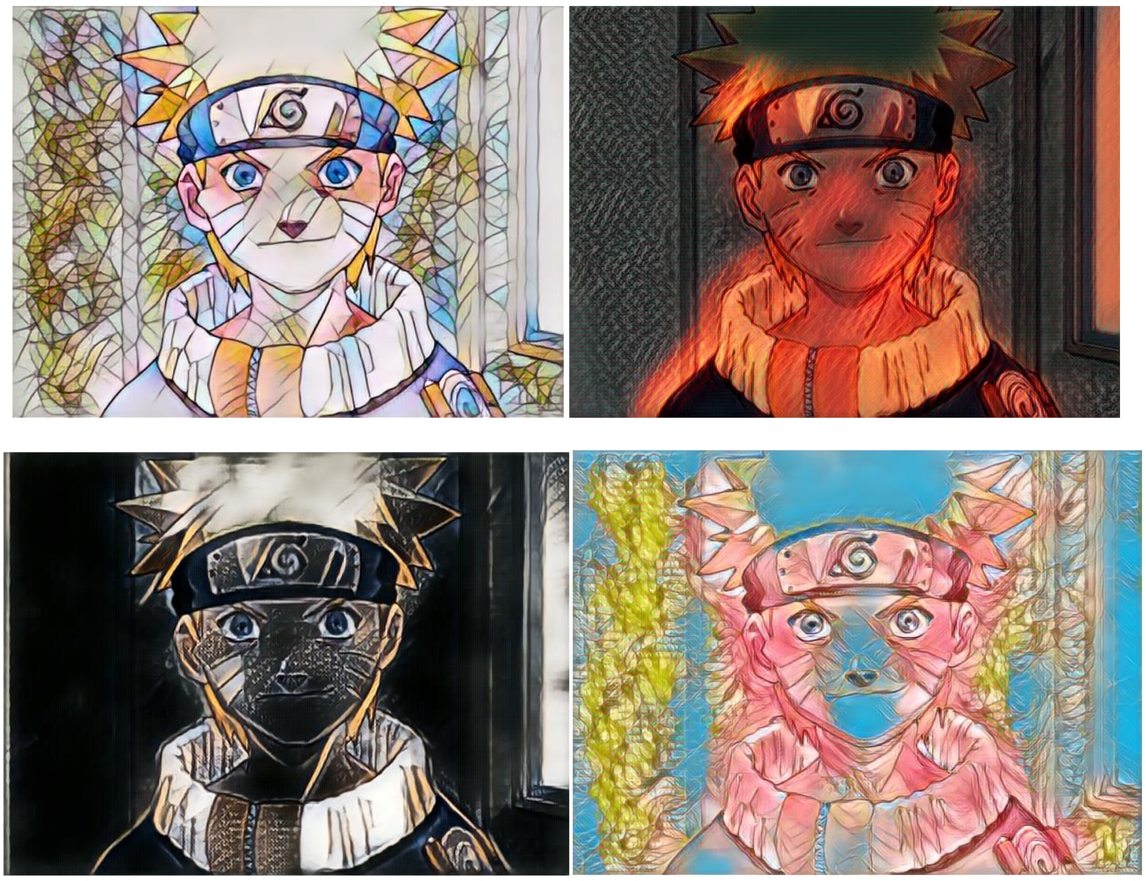
How it works -

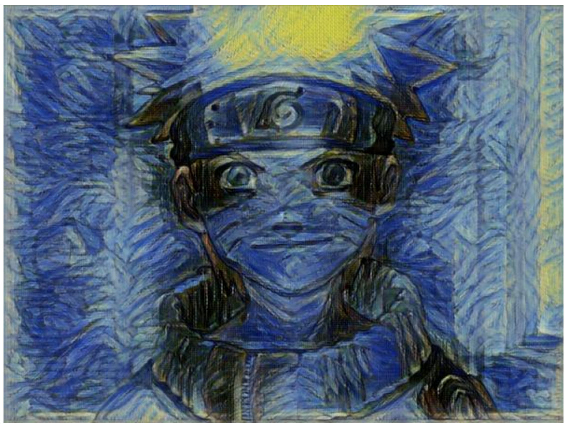
You can choose from following style image's

5.2 Input Image



5.3 Result (After combining with Style Images)





CHAPTER - 6

CONCLUSION

The project successfully implemented a web-based application using Python, Flask, deep learning, and neural style transfer techniques to provide users with a seamless and interactive platform for applying artistic styles to their images. By allowing users to upload their own images and fusing them with a chosen style image, the project effectively demonstrated the power and creative potential of neural style transfer.

The integration of Flask as a web framework facilitated the development of a user-friendly interface, enabling users to easily upload their images and select desired style images for the fusion process. Python, with its extensive libraries and frameworks, provided a robust foundation for implementing the underlying neural style transfer algorithm and handling image processing tasks.

The deep learning approach employed in the project allowed for the extraction and representation of content and style features from the input images. By utilizing pre-trained convolutional neural networks, the model efficiently captured the artistic style of the selected image and transferred it onto the content of the user's uploaded image. This process resulted in visually striking compositions that combined the content of the original image with the desired artistic style.

The project's implementation of neural style transfer showcased its potential applications in the realm of digital art and creative expression. By leveraging deep learning and advanced image processing techniques, users were able to explore different artistic styles and create personalized and stylized artwork. The project also highlighted the fusion of art and technology, underscoring the collaborative and transformative potential of these fields.

Moving forward, the project could be expanded to include additional features and improvements. For instance, incorporating user feedback and preferences could enhance the customization options available to users, allowing them to further fine-tune the style transfer process. Additionally, optimizing the computational efficiency of the neural style transfer algorithm could enhance the application's performance, particularly when handling larger image files or accommodating real-time style transfer.

Overall, the project successfully demonstrated the application of neural style transfer in creating combined art by fusing user-uploaded images with chosen style images. Through the utilization of Python, Flask, deep learning, and neural style transfer techniques, the project showcased the convergence of technology and artistic expression, empowering users to create visually captivating and personalized artwork.

REFERENCES

1. Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576.
2. Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In European conference on computer vision (pp. 694-711). Springer.
3. Li, C., & Wand, M. (2016). Precomputed real-time texture synthesis with Markovian generative adversarial networks. In European conference on computer vision (pp. 702-716). Springer.
4. Dumoulin, V., Shlens, J., & Kudlur, M. (2016). A learned representation for artistic style. arXiv preprint arXiv:1610.07629.
5. Ulyanov, D., Lebedev, V., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022.
6. Luan, F., Paris, S., Shechtman, E., & Bala, K. (2017). Deep photo style transfer. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4990-4998).
7. Li, Y., Liu, M. Y., Li, X., Yang, M. H., & Kautz, J. (2017). A closed-form solution to photorealistic image stylization. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).
8. Zhu, J. Y., Zhang, R., Pathak, D., Darrell, T., Efros, A. A., Wang, O., & Shechtman, E. (2017). Toward multimodal image-to-image translation. In Advances in neural information processing systems (pp. 465-476).