**Aim:** Write a program to perform the Neural Style Transfer algorithm

**Objective:**

The objective of this project is to implement a deep learning model that applies the style of one image (e.g., a painting) to the content of another image (e.g., a photograph), producing a visually artistic output by combining both images using convolutional neural networks (CNNs), specifically leveraging the VGG19 architecture.

**Steps Overview:**

 **Import Necessary Libraries**: We'll need TensorFlow or PyTorch, along with specific image processing libraries to work with neural networks.

 **Load and Preprocess Images**: Load the content and style images. Resize them to match the size of the content image, normalize them, and convert them to the format required for the neural network.

 **Load Pre-trained Model**: Use a pre-trained Convolutional Neural Network (CNN) like VGG19, which is often used for NST. VGG19 is a popular model because it has been trained on large datasets and can extract high-level features for content and style representation.

 **Define Loss Functions**:

* **Content Loss**: Measures how much the content of the generated image differs from the content image.
* **Style Loss**: Measures how much the style of the generated image differs from the style image.
* **Total Variation Loss**: Optional, used to reduce high-frequency noise in the final image.

 **Optimization**: Use an optimizer (like Adam) to minimize the combined loss (content loss + style loss) and update the generated image in an iterative manner.

 **Display the Final Image**: After several iterations, the generated image will combine both the style and content.

**Implementation:**

*Code Implementation in Python (Using TensorFlow/Keras)*

*First, ensure that you have TensorFlow installed:* ***pip install tensorflow***

*import tensorflow as tf*

*import numpy as np*

*import matplotlib.pyplot as plt*

*from tensorflow.keras.preprocessing import image*

*from tensorflow.keras.applications.vgg19 import VGG19, preprocess\_input*

*from tensorflow.keras.optimizers import Adam*

*# Load the content and style images*

*from google.colab import files*

*uploaded = files.upload()*

*# Paths to the uploaded images*

*content\_image\_path = 'content\_image (2).jpg'*

*style\_image\_path = 'style\_image (1).jpg'*

*# Function to load and preprocess the image*

*def load\_and\_preprocess\_image(image\_path):*

*img = image.load\_img(image\_path, target\_size=(400, 400))*

*img = image.img\_to\_array(img)*

*img = np.expand\_dims(img, axis=0)*

*img = preprocess\_input(img) # Preprocessing for VGG19*

*return img*

*# Load the images*

*content\_image = load\_and\_preprocess\_image(content\_image\_path)*

*style\_image = load\_and\_preprocess\_image(style\_image\_path)*

*# Show the images*

*plt.figure(figsize=(10, 5))*

*plt.subplot(1, 2, 1)*

*plt.imshow(content\_image[0])*

*plt.title("Content Image")*

*plt.subplot(1, 2, 2)*

*plt.imshow(style\_image[0])*

*plt.title("Style Image")*

*plt.show()*

*# Load the VGG19 model without the top layer*

*vgg = VGG19(include\_top=False, weights='imagenet')*

*# Define the layers to extract features from*

*layer\_names = ['block5\_conv2', 'block4\_conv2', 'block3\_conv3', 'block2\_conv2', 'block1\_conv2']*

*layers = [vgg.get\_layer(name).output for name in layer\_names]*

*# Define the model for feature extraction*

*model = tf.keras.Model(inputs=vgg.input, outputs=layers)*

*# Function to get features from the model*

*def get\_features(model, content\_image, style\_image):*

*content\_features = model(content\_image)*

*style\_features = model(style\_image) # Directly use style\_image here*

*return content\_features, style\_features*

*# Function to compute gram matrix for style features*

*def gram\_matrix(tensor):*

*channels = int(tensor.shape[-1])*

*tensor = tf.reshape(tensor, (-1, channels))*

*gram = tf.matmul(tensor, tensor, transpose\_a=True)*

*return gram*

*# Compute content and style features*

*content\_features, style\_features = get\_features(model, content\_image, style\_image)*

*# Compute gram matrices for style features*

*style\_grams = [gram\_matrix(style\_feature) for style\_feature in style\_features]*

*# Initialize the generated image (copy of the content image)*

*generated\_image = tf.Variable(content\_image)*

*# Define the loss weights*

*content\_weight = 1e3*

*style\_weight = 1e-2*

*# Function to compute total loss*

*def compute\_loss(content\_features, style\_features, generated\_image):*

*generated\_features = model(generated\_image)*

*content\_loss = content\_weight \* tf.reduce\_mean((generated\_features[0] - content\_features[0]) \*\* 2)*

*style\_loss = 0*

*for gen, style in zip(generated\_features, style\_grams):*

*style\_loss += style\_weight \* tf.reduce\_mean((gram\_matrix(gen) - style) \*\* 2)*

*total\_loss = content\_loss + style\_loss*

*return total\_loss*

*# Optimizer*

*optimizer = Adam(learning\_rate=0.02)*

*# Function to compute gradients*

*def compute\_grads(model, content\_image, style\_image, generated\_image):*

*with tf.GradientTape() as tape:*

*total\_loss = compute\_loss(content\_features, style\_features, generated\_image)*

*grads = tape.gradient(total\_loss, generated\_image)*

*return grads*

*# Training loop*

*epochs = 10*

*for epoch in range(epochs):*

*grads = compute\_grads(model, content\_image, style\_image, generated\_image)*

*optimizer.apply\_gradients([(grads, generated\_image)])*

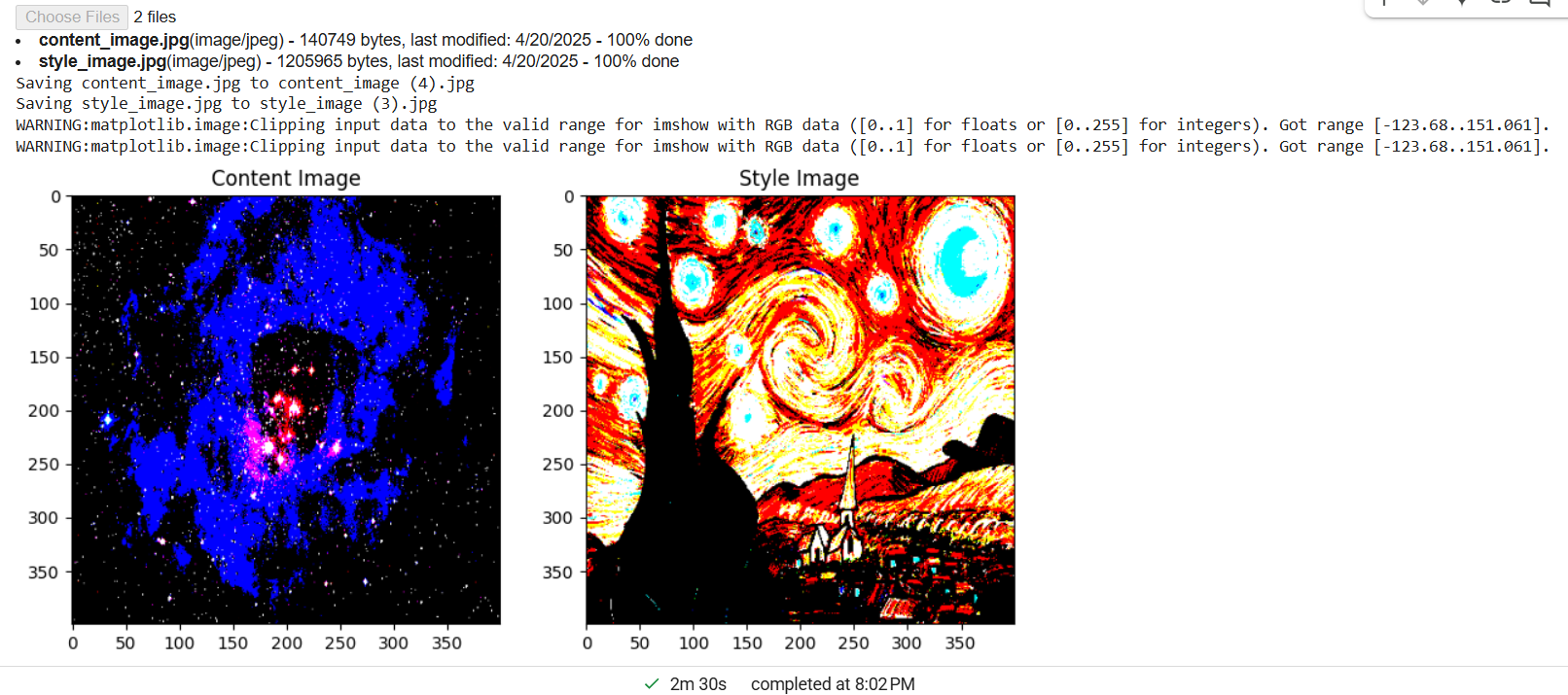
*if (epoch + 1) % 1 == 0:*

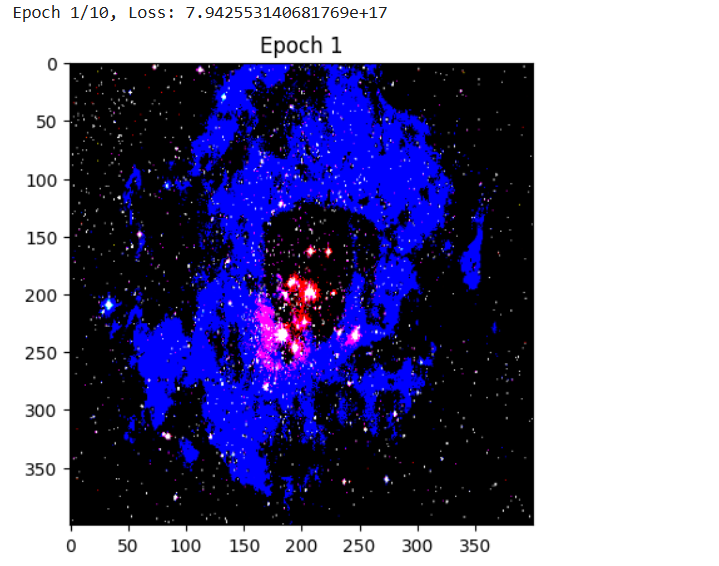
*print(f"Epoch {epoch+1}/{epochs}, Loss: {compute\_loss(content\_features, style\_features, generated\_image)}")*

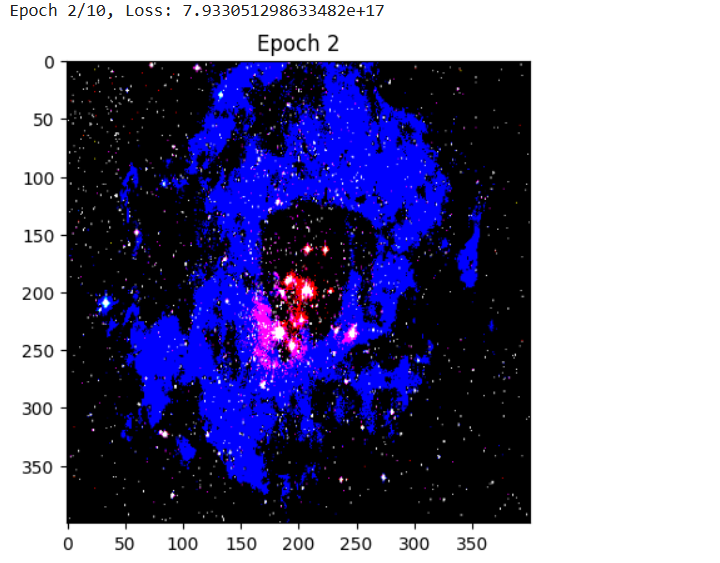
*plt.imshow(generated\_image.numpy()[0])*

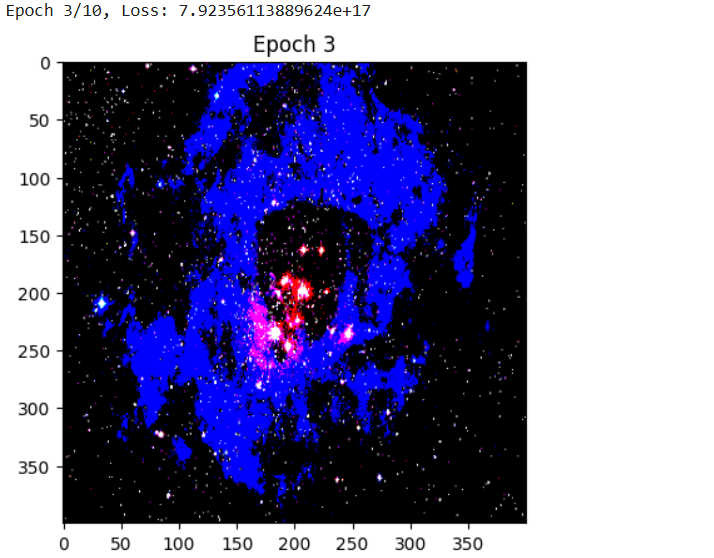
*plt.title(f"Epoch {epoch+1}")*

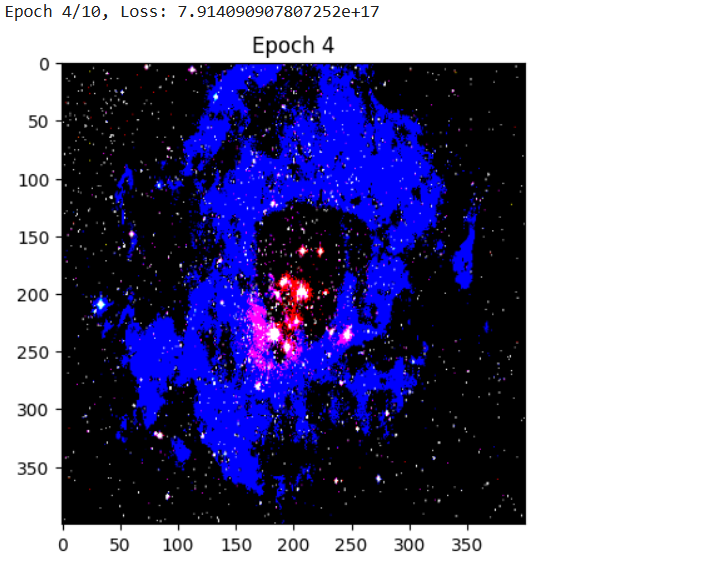
*plt.show()*

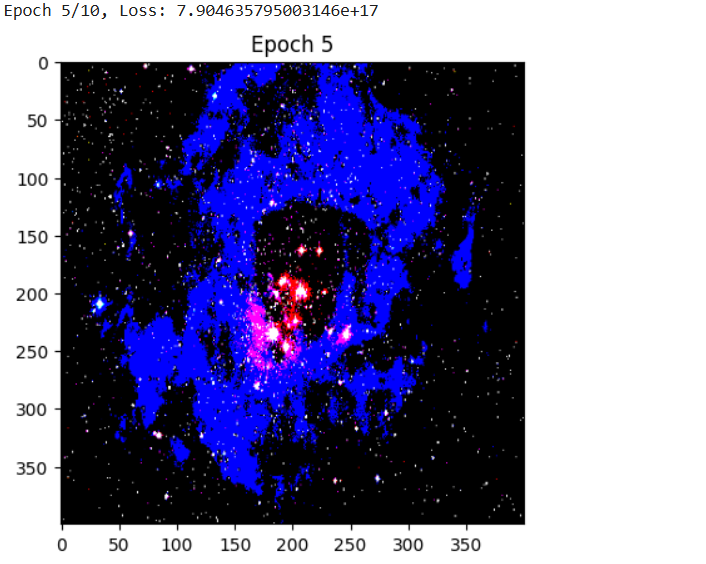
**Output: **

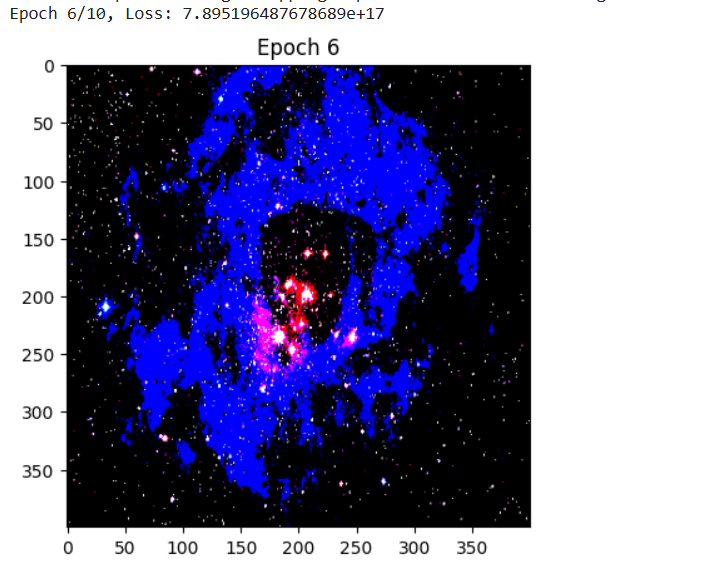
****

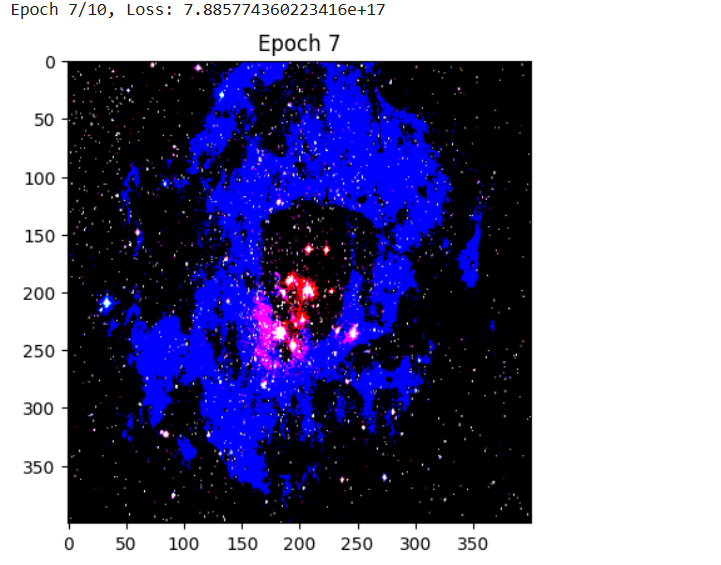
****

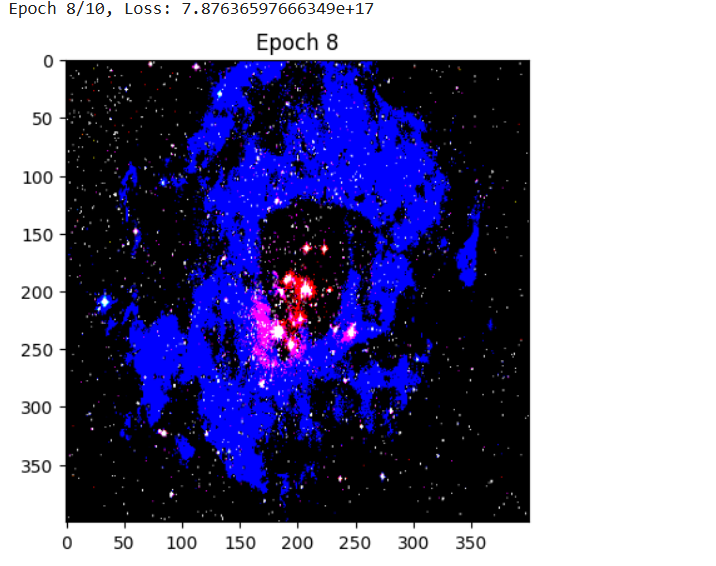
****

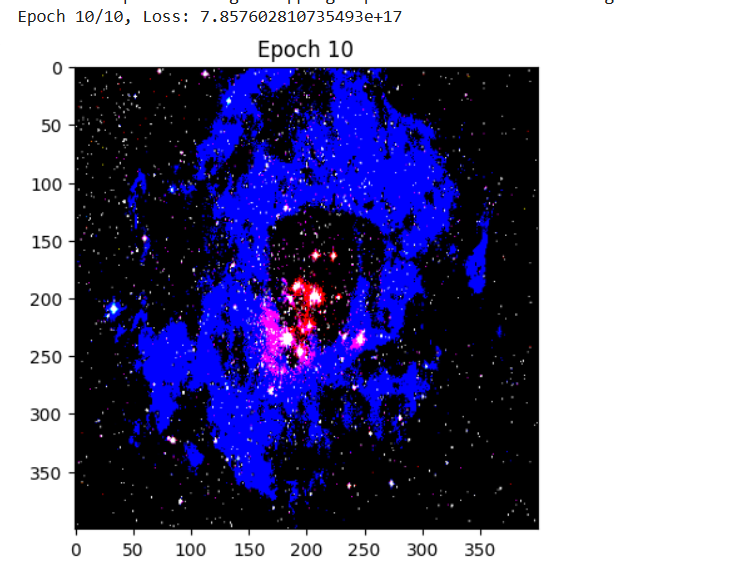
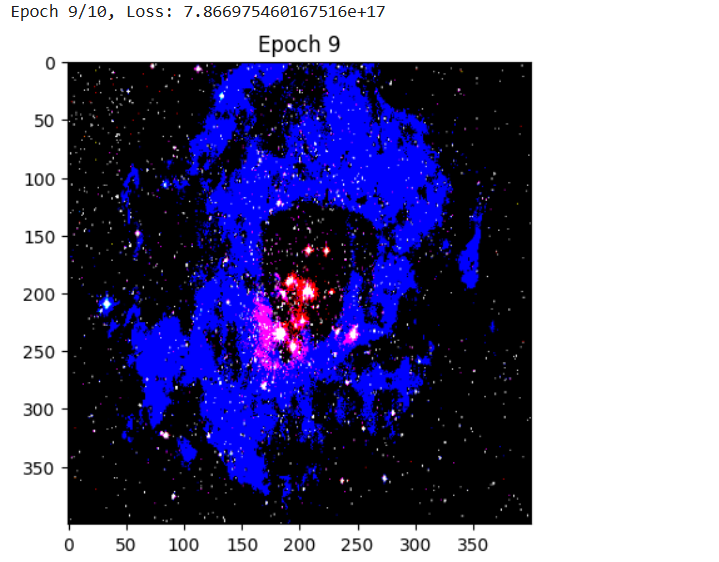
****

****

****

****

****

****

 Preprocessing and Loading Images:

* The load\_and\_preprocess\_image function loads and preprocesses images to be compatible with the VGG19 model.
* The deprocess\_image function converts the processed image back to its original form for visualization.

 VGG19 Model:

* We use VGG19, a deep CNN trained on ImageNet, to extract features. We use layers that are good for content (e.g., block5\_conv2) and style (e.g., block1\_conv1, block2\_conv1).

 Loss Calculation:

* Content Loss: Measures the difference between the content of the generated image and the original content image.
* Style Loss: Uses Gram matrices to capture the style (texture patterns) of the style image and compares it with the generated image.

 Optimization:

* We use the Adam optimizer to update the generated image based on the computed gradients.

 Display:

* The generated image is displayed at regular intervals and after all epochs to visualize the progress.

**Conclusion:**

In this experiment, we successfully implemented **Neural Style Transfer (NST)** using a pre-trained **VGG19** model in TensorFlow. The aim was to generate a new image (generated\_image) that preserves the **content** of one image while adopting the **style** of another.