NEURAL STYLE TRANSFER

Neural Style Transfer (NST) is a technique in the field of computer vision that involves merging the content of one image with the style of another using convolutional neural networks (CNNs). The concept was introduced by Gatys et al. in 2015, revolutionizing the way artistic effects can be applied to digital images. After the publication of this paper a lot of further work has been done in the domain of neural style transfer.

This report delves into the fascinating world of Neural Style Transfer (NST).

I have started with the overview of what NST can do with the content image and style image and then further delved in explaining the role of Convolutional Neural Networks (CNNs) in the field of image processing.

After that I have described the VGG19 architecture and further explained the architecture ant the role of VGG19 in transforming the content image by applying the different artistic style.

We then explore the process of generating a new image that incorporates the desired content and style through an optimization process. The report further discusses various NST algorithms, highlighting their strengths and limitations.

Finally, the report ends with explaining the results and limitations of my implementation.

VGG19 Architecture: VGG-19 is a deep convolutional neural network with 19 weight layers, comprising 16 convolutional layers and 3 fully connected layers. The architecture follows a straightforward and repetitive pattern, making it easier to understand and implement.

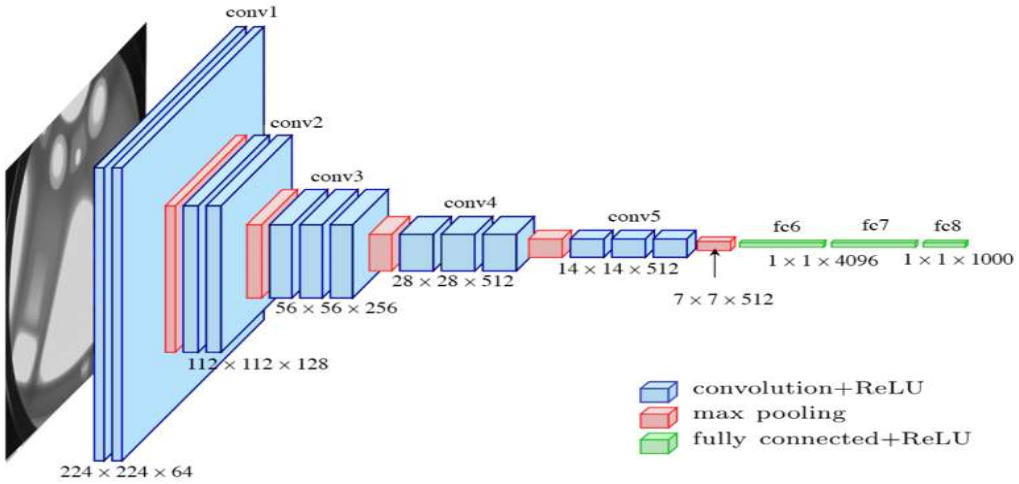
The key components of the VGG-19 architecture are:

1. **Convolutional Layers**: 3×3 filters with a stride of 1 and padding of 1 to preserve spatial resolution.
2. **Activation Function**: ReLU (Rectified Linear Unit) applied after each convolutional layer to introduce non-linearity.
3. **Pooling Layers**: Max pooling with a 2×2 filter and a stride of 2 to reduce the spatial dimensions.

4. **Fully Connected Layers**: Three fully connected layers at the end of the network for classification.
5. **Softmax Layer**: Final layer for outputting class probabilities.

Detailed Layer-by-Layer Architecture of VGG-Net 19

The VGG-19 model consists of five blocks of convolutional layers, followed by three fully connected layers. Here is a detailed breakdown of



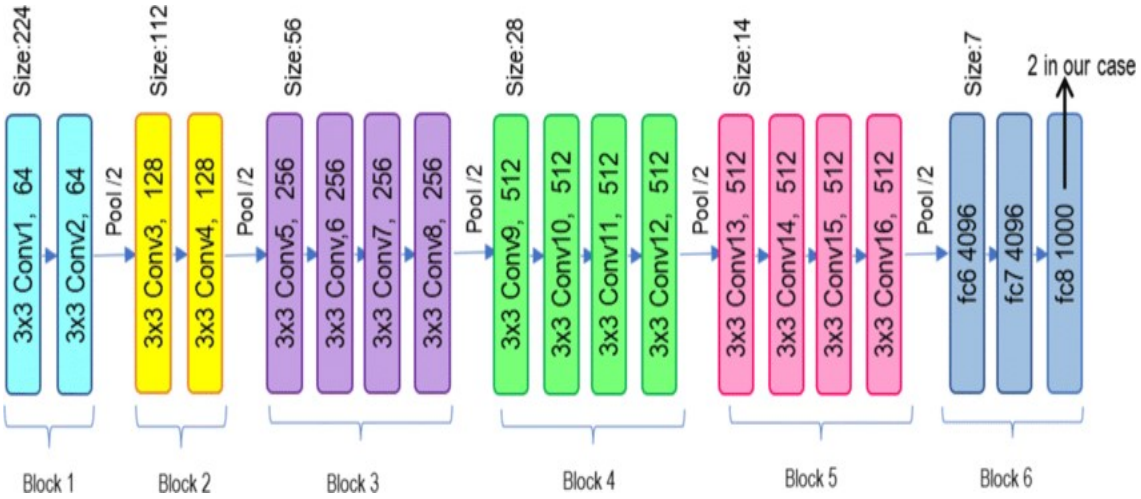


Image Source: Google Images

Use in Transfer Learning: VGG-19 has been extensively used in transfer learning due to its robust feature extraction capabilities. Pre-

trained VGG-19 models on large datasets like ImageNet are often fine-tuned for various computer vision tasks, including object detection, image segmentation, and style transfer.

Use of VGG19 in NST model:

I have used the pre-trained VGG19 model, which has already been trained large scale image classification tasks.
Additionally, from the 19 layers of VGG19 I have truncated off the fully connected layers. I have used the feature space provided by the first 16 convolutional and 5 pooling layers for this task.

Content Representation: Lower layers of the VGG network capture the content of an image, focusing on object shapes, edges, and spatial relationships. To extract the content representation, we utilize the activations of a specific layer within these lower levels.

Style Representation: Higher layers in the VGG network capture the style of an image, encompassing artistic elements like brushstrokes, colour patterns, and textures. Style representation is obtained by calculating the Gram matrix, which measures the correlations between feature maps activated in a particular layer.

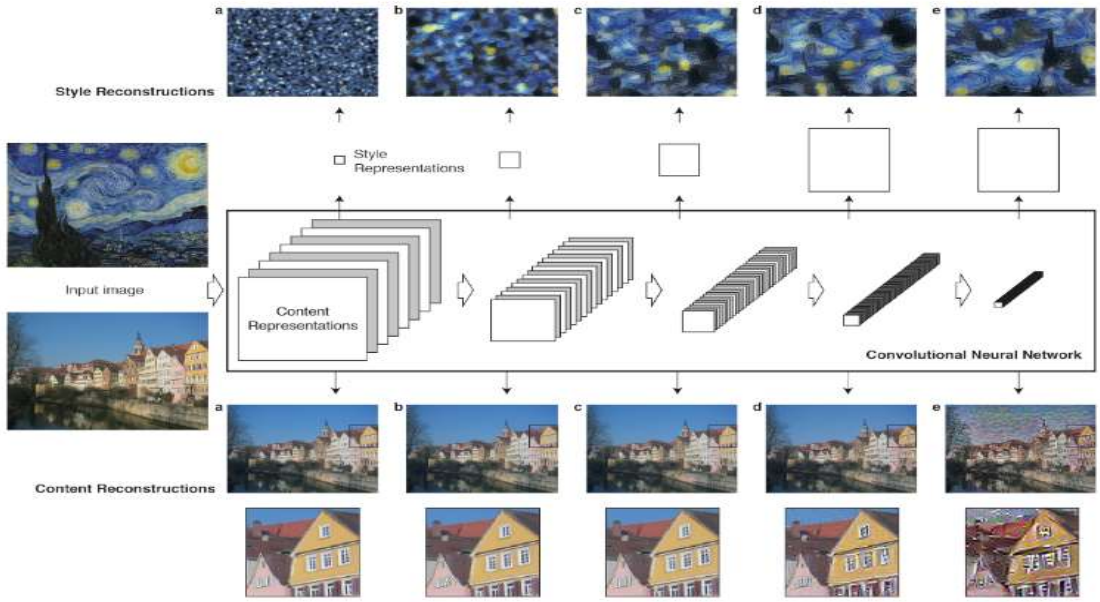


Image Source: Google Images

Architecture of NST Model:

The style transfer model takes a content image and an arbitrary style image transforms them and convert them in such a way that they meet the requirements of pre-trained VGG19 Neural Net input.
Fully connected layers (FCLs) and softmax layer of VGG19 is removed.
Then, we initialize a target image which is a copy of the content image (or a random noise generated image can also be used).
We define a loss function which is combination of the content loss (how far the content image is to the initialized target image) and style loss (how far the style image is to the target image). Then the model uses the gradient descent (an optimization algorithm) and backpropagation learning algorithm to minimize the loss.

Requirements to run the model:
Python 3.10.11
torch==2.3.1
torchvision==0.18.1
matplotlib==3.9.0
numpy==1.26.4
pillow==10.3.0

Loss function:

The loss function which we minimize using the optimization algorithms and backpropagation learning is-

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

where **p** be the content image, **a** be the style image, **x** be the initialized target image, α and β are the weighting factors for the content and style reconstruction respectively.

Content Loss:
It is the Mean Squared Error Loss between the feature representation of the generated image and the original image at different layers.

```
#1. Defining the content loss
def get_content_loss(target_image, content_image):
  return torch.mean((target_image-content_image)**2)
```

Style Loss:

```
# Defining the gram_matrix
def get_gram_matrix(tensor):
    c, h, w = tensor.size()
    tensor = tensor.view(c, h * w)
    gram_matrix = torch.mm(tensor, tensor.t())
    return gram_matrix
#2. Defining the style loss
def get_style_loss(target_image,style_image):
  c,h,w = target_image.size()
  gram_target = get_gram_matrix(target_image)
  gram_style = get_gram_matrix(style_image)
  return torch.mean((gram_target - gram_style) ** 2)/(c*h*w)
```

To generate a texture that matches the style of a given image we use gradient descent from a white noise image to find another image that matches the style representation of the original image. This is done by minimising the mean-squared distance between the entries of the Gram matrix from the original image and the Gram matrix of the image to be generated.

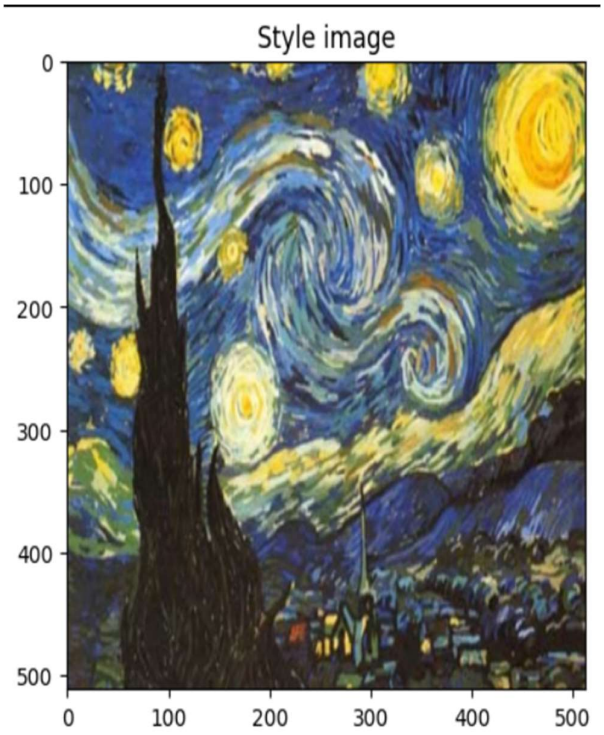
Training:

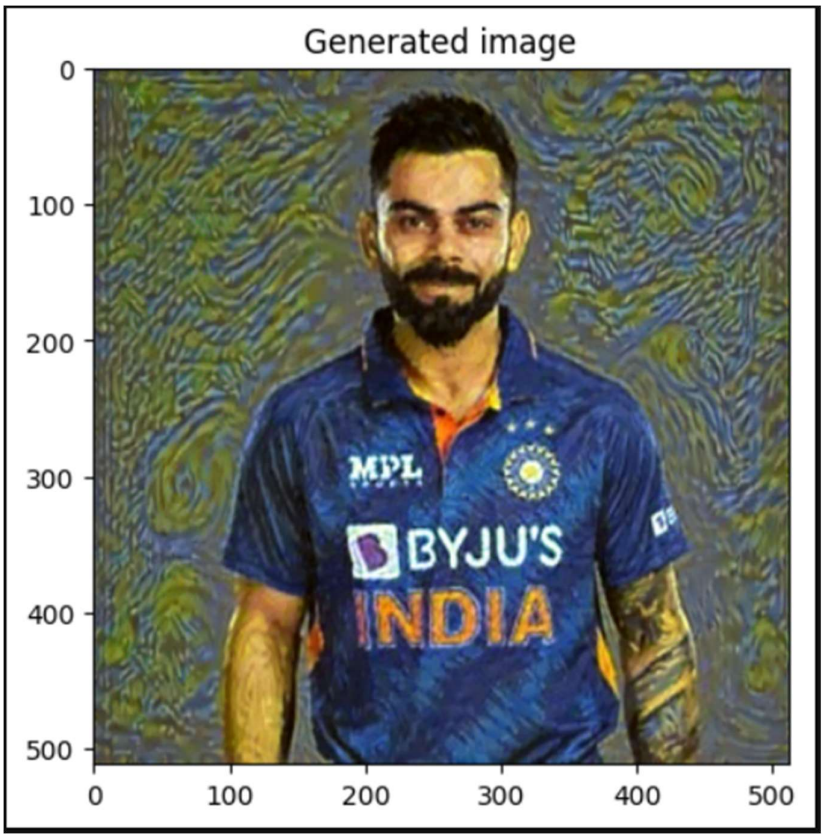
I have trained the model using the target image (which is a copy of content image) and content image, and style image and target image. I have used the Adam optimizer and learning rate = 0.001 and $\alpha = 5$ and $\beta = 20$ as hyperparameters. After running through 5000 epochs on the "CUDA" I got an error of less than 60 (though quite high). It took me around 13-14 minutes.

```
epochs = 5000
display_every = 50
for e in range(epochs):
    target_features = model(target_image)
    content_features = model(content)
    style_features = model(style)

    content_loss = get_content_loss(target_features[1], content_features[1])

    style_loss = 0
    for target_feat, style_feat in zip(target_features, style_features):
        style_loss += get_style_loss(target_feat, style_feat)

    total_loss = alpha * content_loss + beta * style_loss

    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()
```

Example run: content image and style image are taken from google images


Content image
0
100
200
300
400
500
MPL
BYJU'S
INDIA


Style image
0
100
200
300
400
500


Generated image
0
100
200
300
400
500
MPL
BYJU'S
INDIA

Conclusion: My NST model is giving only the basic recombination image. It is giving different error with different set of images. However, the best (total loss) which I could achieve is around 54.
I will further try to bring it down to less than 2 by applying other improvements which were published after 2015. I also tried to build a web interface where users can upload content and style images from device and get the generated image as output, but I could only build the fronted and was not able integrate my model with the frontend because of time limitations of the deadline. I try to complete the remaining portion of web interface in future.

References:

1. "A Neural Algorithm of Artistic Style Leon" by
   Leon A. Gatys, Alexander S. Ecker, Matthias Bethge
2. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.
3. PyTorch documentation: https://pytorch.org/docs/stable/index.html
4. Google Images.