Neural Style Transfer (NST) is a method in deep learning where the details of one picture are combined with the artistic style of another to create a new image. This process allows you to generate an image that has the subject of the first picture but the art style, colors and textures of the second. In this guide we will explain how NST works and how to apply it using TensorFlow.

**Understanding Neural Style Transfer**

Neural Style Transfer or NST involves three different images:

Content Image: This is the picture with the main subject or scene you want to change. It holds the basic structure and layout you want to keep

Style Image: the picture with the artistic feel, like colors, textures and patterns. You want to use this style to change how your content image looks.

Generated Image: The output image that combines the content of the first image with the style of the second.

An example of style transfer A is a content image, B is output with style image in the bottom left corner

The process of NST uses a type of computer model called a [convolutional neural network (CNN).](https://www.geeksforgeeks.org/introduction-convolution-neural-network/) A well-known CNN often used in NST is called [VGG19.](https://www.geeksforgeeks.org/vgg-net-architecture-explained/) This computer model analyzes details and features in both the content and style images. To make the new image the computer model works with a formula known as a loss function. This formula keeps the computer to find the best mix of subject from the content image and the artistic elements from the style image.

*To understand the Neural Style Transfer more refer to:* [*Overview of Style Transfer*](https://www.geeksforgeeks.org/overview-of-style-transfer-deep-harmonization/)

## Implementing Neural Style Transfer with TensorFlow

In implementation of NST we'll use the VGG19 model, pre-trained on the ImageNet dataset to extract features from our images.

### Step 1: Import Necessary Libraries

First we import the necessary module like [TensorFlow v2](https://www.geeksforgeeks.org/tensorflow-2-0/) with [Keras](https://www.geeksforgeeks.org/python-tensorflow-tf-keras-layers-conv2d-function/), [NumPy](https://www.geeksforgeeks.org/python-numpy/) for numerical operations, [Matplotlib](https://www.geeksforgeeks.org/python-introduction-matplotlib/) for data visualisation and Keras-specific components for working with pre-trained models and image processing.

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

import math

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

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

from tensorflow.keras.models import Model

Step 2: Image processing

Now, we load and process the image using Keras preprocess input in VGG 19. The expand\_dims function adds a dimension to represent a number of images in the input. This preprocess\_input function used in VGG 19 converts the input RGB to BGR images and centre these values around 0 according to ImageNet data.

1 Def load\_and\_process\_image(image\_path):

2 Img = load\_img(image\_path)

3

4 Img = img\_to\_array(img)

5 Img = preprocess\_input(img)

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

7 Return img

Now, we define the deprocess function that takes the input image and perform the inverse of preprocess\_input function that we imported above. To display the unprocessed image we also define a display function.

Def deprocess(img):

Img[:, :, 0] += 103.939

Img[:, :, 1] += 116.779

Img[:, :, 2] += 123.6

Img = img[:, :, ::-1]

Img = np.clip(img, 0, 255).astype(‘uint8’)

Return img

Def display\_image(image):

If len(image.shape) == 4:

Img = np.squeeze(image, axis=0)

Img = deprocess(img)

Plt.grid(False)

Plt.xticks([])

Plt.yticks([])

Plt.imshow(img)

Return

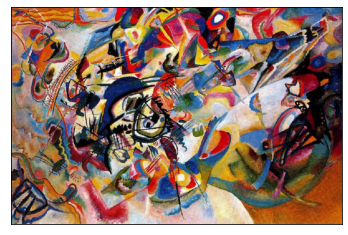
Now we use the above function to display the style and content images

content\_img = load\_and\_process\_image(content\_path)

display\_image(content\_img)

style\_img = load\_and\_process\_image(style\_path)

display\_image(style\_img)



Step 3: Model Initialization

Now we initialize the VGG model with ImageNet weights we will also remove the top layers and make it non-trainable. We use VGG19 to extract image features.

Include\_top=False removes the classification head.

We freeze the model to use it as a fixed feature extractor.

1 Model = VGG19(

2 Include\_top=False,

3 Weights=’imagenet’

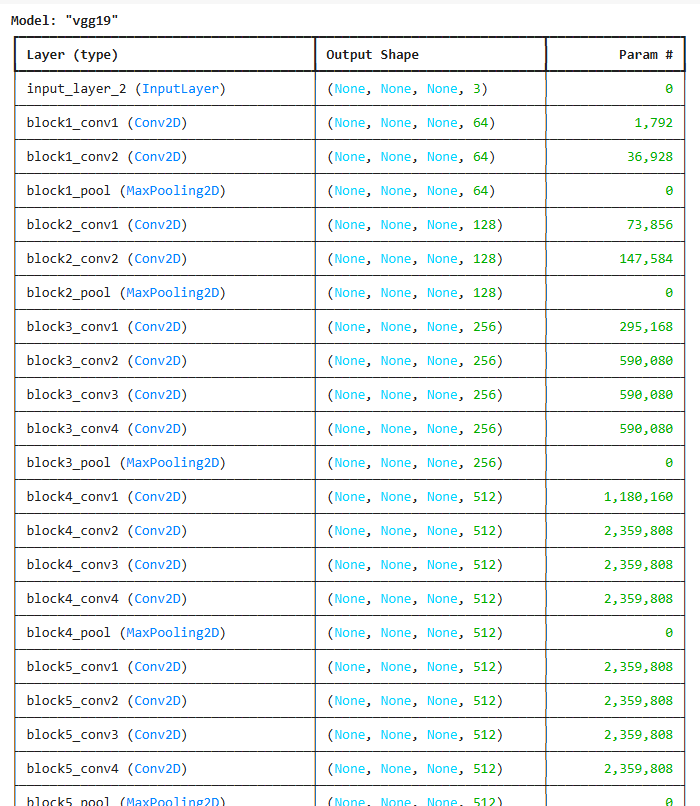
4 )

5

6 Model.trainable = False

7

8 Model.summary()

**Output:**

Step 4: Content Model defining

Now we define the content and style model using Keras.Model API. The content model takes the image as input and output the feature map from “block5\_conv1” from the above VGG model.

1 Content\_layer = ‘block5\_conv2’

2 Content\_model = Model(

3 Inputs=model.input,

4 Outputs=model.get\_layer(content\_layer).output

5 )

6 Content\_model.summary()

Output:

Step 5: Style Model defining:

Now, we define the content and style model using Keras.Model API. The style model takes an image as input and output the feature map from “block1\_conv1, block3\_conv1 and block5\_conv2” from the above VGG model. We use multiple layers for style to capture texture patterns at different scales. Each layer captures style at different levels of abstraction like edges, patterns and textures.

1 Style\_layers = [

2 ‘block1\_conv1’,

3 ‘block3\_conv1’,

4 ‘block5\_conv1’

5. ]

6 Style\_models = [Model(inputs=model.input,

7 Outputs=model.get\_layer(layer).output) for layer in style\_layers]

Content Loss: Now we define the content loss function it will take the feature map of generated and real images and calculate the mean square difference between them.

1 Def content\_loss(content, generated):

2 A\_C = content\_model(content)

3 A\_G = content\_model(generated) # Add this line to compute a\_G

4 Loss = tf.reduce\_mean(tf.square(a\_C – a\_G))

5 Return loss

Gram Matrix: Now we define the gram matrix function. This function also takes the real and generated images as the input of the model and calculates gram matrices of them before calculate the style loss weighted to different layers.

1 Def gram\_matrix(A):

2 Channels = int(A.shape[-1])

3 A = tf.reshape(A, [-1, channels])

4 N = tf.shape(a)[0]

5 Gram = tf.matmul(a, a, transpose\_a=True)

6 Return gram / tf.cast(n, tf.float32)

7

8

9 Weight\_of\_layer = 1. / len(style\_models)

Style Loss: The function style\_cost defined by this code determines the style loss between a generated image and a style image that is supplied. In neural style transfer algorithms style loss is frequently employed to create an image that blends the content of two different images with their styles.

1 Def style\_cost(style, generated):

2 J\_style = 0

3

4 For style\_model in style\_models:

5 A\_S = style\_model(style)

6 A\_G = style\_model(generated)

7 GS = gram\_matrix(a\_S)

8 GG = gram\_matrix(a\_G)

9 Content\_cost = tf.reduce\_mean(tf.square(GS – GG))

10 J\_style += content\_cost \* weight\_of\_layer

11

12 Return J\_style

Content Loss: The content loss between a style image and a generated image is determined by the function content\_cost, which is defined in this code. To make sure that the generated image preserves the original image’s content, neural style transfer algorithms frequently employ content loss.

1 Def content\_cost(style, generated):

2 J\_content = 0

3

4 For style\_model in style\_models:

5 A\_S = style\_model(style)

6 A\_G = style\_model(generated)

7 GS = gram\_matrix(a\_S)

8 GG = gram\_matrix(a\_G)

9 Content\_cost = tf.reduce\_mean(tf.square(GS – GG))

10 J\_content += content\_cost \* weight\_of\_layer

11

12 Return J\_content

Step 6: Training Function

In this step we define the training\_loop function that optimizes the generated image using both content and style losses. It begins by loading and preprocessing the content and style images then initializes the generated image as a trainable variable.

Gradient Tape watches changes in the generated image.

J\_content: difference in content features.

J\_style: difference in style (Gram matrices).

J\_total: weighted combination using a and b.

Generated\_images = []

Def training\_loop(content\_path, style\_path, iterations=50, a=10, b=1000)

Content = load\_and\_process\_image(content\_path)

Style = load\_and\_process\_image(style\_path)

Generated = tf.Variable(content, dtype=tf.float32)

Opt = tf.keras.optimizers.Adam(learning\_rate=0.7)

Best\_cost = math.inf

Best\_image = None

For I in range(iterations):

Start\_time\_cpu = time.process\_time()

Start\_time\_wall = time.time()

With tf.GradientTape() as tape:

J\_content = content\_cost(style, generated)

J\_style = style\_cost(style, generated)

J\_total = a \* J\_content + b \* J\_style

Grads = tape.gradient(J\_total, generated)

Opt.apply\_gradients([(grads, generated)])

End\_time\_cpu = time.process\_time()

End\_time\_wall = time.time()

Cpu\_time = end\_time\_cpu – start\_time\_cpu

Wall\_time = end\_time\_wall – start\_time\_wall

If J\_total < best\_cost:

Best\_cost = J\_total

Best\_image = generated.numpy()

Print(“CPU times: user {} µs, sys: {} ns, total: {} µs”.format(

Int(cpu\_time \* 1e6),

Int(( end\_time\_cpu – start\_time\_cpu) \* 1e9),

Int((end\_time\_cpu – start\_time\_cpu + 1e-6) \* 1e6))

)

Print(“Wall time: {:.2f} µs”.format(wall\_time \* 1e6))

Print(“Iteration :{}”.format(i))

Print(‘Total Loss {:e}.’.format(J\_total))

Generated\_images.append(generated.numpy())

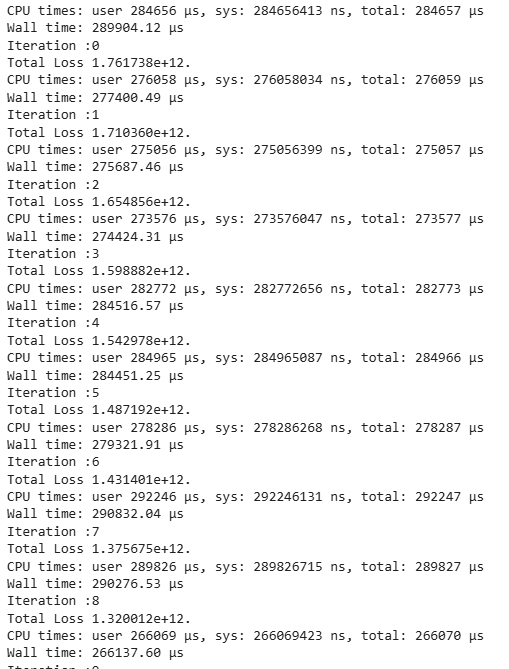
Return best\_image

Step 7: Model Training

Now, we train our model using the training function we defined above.

1 Final\_img = training(content\_path, style\_path)

Output:



Step 8: Model Prediction

In the final step, we plot the final and intermediate results.

1 Plt.figure(figsize=(12, 12))

2

3 For I in range(10):

4 Plt.subplot(4, 3, I + 1)

5 Display\_image(generated\_images[i+39])

6 Plt.show()

7

8 Display\_image(final\_img)

Last 10 generated images



Best generated image

This output shows the last 10 generated images during Neural Style Transfer where the content (dog) is preserved and style patterns are gradually applied. Each image reflects slight improvements in blending style with content across iterations.

*Complete Code :* [*click here*](https://media.geeksforgeeks.org/wp-content/uploads/20250409161336569934/Neural_ipynb.zip)