Neural Style transfer (NST)

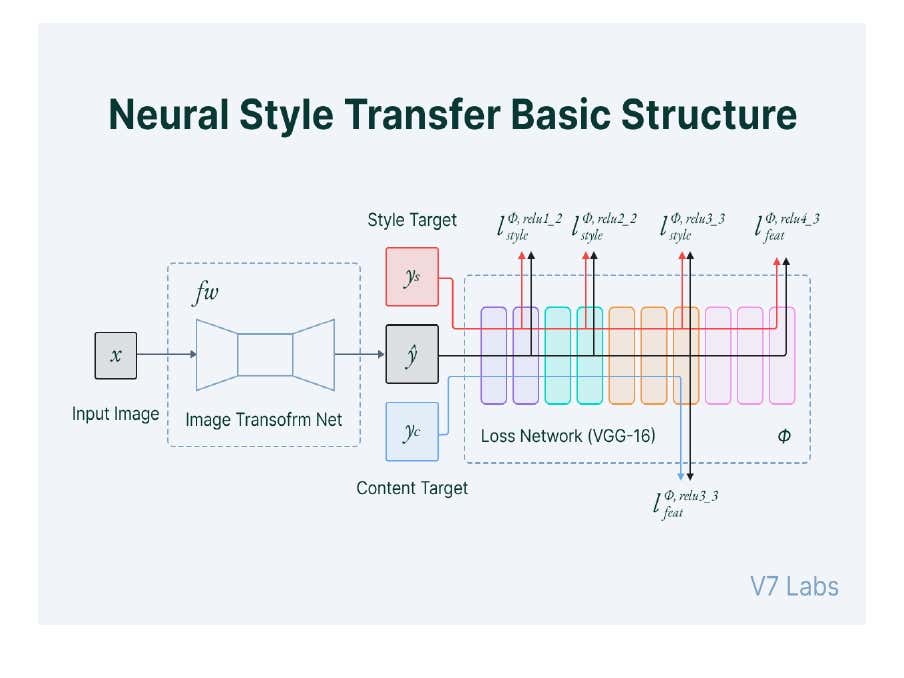
-Vedanti Kale

INTRODUCTION

NST is the technique of blending style from one image into another image keeping its content intact. The only change is the style configurations of the image to give an artistic touch to your image. The content image describes the layout or the sketch and the Style is the painting or the colors. It is an application related to image processing techniques and Deep Convolutional Neural Networks. The Keras library contains a pretrained VGG19 model that can be imported. We define two Keras variables to hold the content image and style image and a placeholder that will

contain the generated combined image. The input tensor to the VGG19 model

is a concatenation of the three images.



DATA PREPROCESSING AND DEPROCESSING

 The preprocess function standardizes each of the three RGB channels of the input image and transforms the results into the CNN input format

def preprocess\_image(image\_path):

    # Util function to open, resize and format pictures into appropriate tensors

    img = keras.utils.load\_img(image\_path, target\_size=(img\_nrows, img\_ncols))

    img = keras.utils.img\_to\_array(img)

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

    img = vgg19.preprocess\_input(img)

    return tf.convert\_to\_tensor(img)

The deprocess function restores the pixel values in the output image to their original values before standardization. Since the image printing function requires that each pixel has a floating point value from 0 to 2.

def deprocess\_image(x):

    # Util function to convert a tensor into a valid image

    x = x.reshape((img\_nrows, img\_ncols, 3))

    # Remove zero-center by mean pixel

    x[:, :, 0] += 103.939

    x[:, :, 1] += 116.779

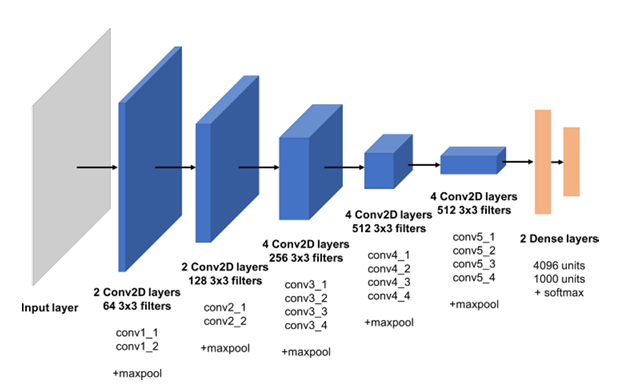
    x[:, :, 2] += 123.68

    # 'BGR'->'RGB'

    x = x[:, :, ::-1]

    x = np.clip(x, 0, 255).astype("uint8")

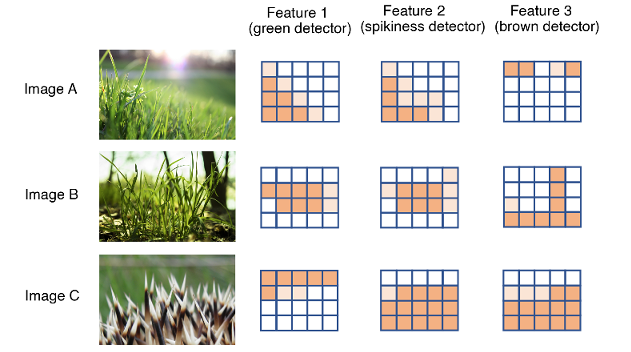
    return x



STYLE LOSS

The style loss function, which keeps the generated image close to the local textures of the style reference image.





The style loss is calculated over five layers—the first convolutional layer in each of the five blocks of the VGG19 model. Here we extract the style image features and combined image features from the input tensor that has been fed through the VGG19 network. The style loss is scaled by a weighting parameter and the number of layers that it is calculated over.

def style\_loss(style, combination):

    S = gram\_matrix(style)

    C = gram\_matrix(combination)

    channels = 3

    size = img\_nrows \* img\_ncols

    return tf.reduce\_sum(tf.square(S - C)) / (4.0 \* (channels\*\*2) \* (size\*\*2))

style\_layer\_names = [

    "block1\_conv1",

    "block2\_conv1",

    "block3\_conv1",

    "block4\_conv1",

    "block5\_conv1",

]

CONTENT LOSS

The content loss function, which keeps the high-level representation of the generated image close to that of the base image. It is calculated by measuring the difference between the higher-level intermediate-layer feature maps.

def content\_loss(base, combination):

    return tf.reduce\_sum(tf.square(combination - base))

TOTAL VARIANCE LOSS

The loss variation function, a regularization loss which keeps the generated image locally-coherent. The total variance loss is simply a measure of noise in the combined image. To judge how noisy an image is, we can shift it one pixel to the right and calculate the sum of the squared difference between the translated and original images.

def total\_variation\_loss(x):

    a = tf.square(

        x[:, : img\_nrows - 1, : img\_ncols - 1, :] - x[:, 1:, : img\_ncols - 1, :]

    )

    b = tf.square(

        x[:, : img\_nrows - 1, : img\_ncols - 1, :] - x[:, : img\_nrows - 1, 1:, :]

    )

    return tf.reduce\_sum(tf.pow(a + b, 1.25))

The squared difference between the image and the same image shifted one pixel down. The squared difference between the image and the same image shifted one pixel to the right.

loss += total\_variation\_weight \* total\_variation\_loss(combination\_image)

    return loss

The total variance loss is scaled by a weighting parameter. The overall loss is the sum of the content, style, and total variance losses.

NEURAL STYLE TRANSFER

Building a VGG19 model

model = vgg19.VGG19(weights="imagenet", include\_top=False)

# Get the symbolic outputs of each "key" layer (we gave them unique names).

outputs\_dict = dict([(layer.name, layer.output) for layer in model.layers])

feature\_extractor = keras.Model(inputs=model.inputs, outputs=outputs\_dict)

The process is initialized with the base image as the starting combined image. At each iteration we pass the current combined image (flattened) into a optimization function that performs one gradient descent step. Here, evaluator is an object that contains methods that calculate the overall loss, as described previously, and gradients of the loss with respect to the input image.

optimizer = keras.optimizers.SGD(

    keras.optimizers.schedules.ExponentialDecay(

        initial\_learning\_rate=100.0, decay\_steps=100, decay\_rate=0.96

    )

)

base\_image = preprocess\_image(base\_image\_path)

style\_reference\_image = preprocess\_image(style\_reference\_image\_path)

combination\_image = tf.Variable(preprocess\_image(base\_image\_path))

iterations =400

for i in range(1, iterations + 1):

    loss, grads = compute\_loss\_and\_grads(

        combination\_image, base\_image, style\_reference\_image

    )

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

    if i % 100 == 0:

        print("Iteration %d: loss=%.2f" % (i, loss))

        img = deprocess\_image(combination\_image.numpy())

        fname = result\_prefix + "\_at\_iteration\_%d.png" % i

        keras.utils.save\_img(fname, img)

OUTPUT



The neural style transfer technique allows us to transfer the style of a single image onto a base image, using a cleverly chosen loss function that penalizes the model for straying too far from the content of the base image and artistic style of the style image, while retaining a degree of smoothness to the output. This technique has been commercialized by many high-profile apps to blend a user’s photographs with a given set of stylistic paintings.

REFERENCES

1. [2019\_Generative\_Deep\_Learning\_Teaching\_Machines\_to\_Paint,\_Write,\_Compose,\_and\_Play\_by\_David\_Foster[1].pdf](file:///C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\IE\ST81Y5K8\2019_Generative_Deep_Learning_Teaching_Machines_to_Paint,_Write,_Compose,_and_Play_by_David_Foster%5b1%5d.pdf)
2. <https://d2l.ai/chapter_computer-vision/neural-style.html>
3. <https://www.cs.princeton.edu/courses/archive/fall17/cos429/COS429-proj/COS429_neuralstyle_HaochenYuyan.pdf>