Neural Style Transfer

Ketan Sethi

August 20, 2019

1 Introduction

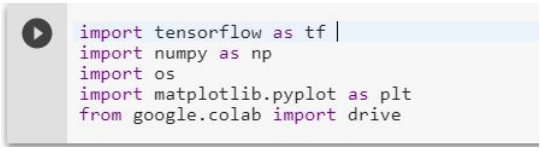
Neural Style Transfer refers to a class of software algorithms that manipulate digital images, or videos, to adopt the appearance or visual style of another image.NST algorithms are characterized by their use of deep neural networks in order to perform the image transformation

1.1 Algorithm

The algorithm includes feature extraction from VGG16 architecture as well Gram matrix creation from features extracted , while the detailed explanation will be provided with code blocks

2 Algorithm Explanation

2.1 Libraries



```
import tensorflow as tf
import numpy as np
import os
import matplotlib.pyplot as plt
from google.colab import drive
```

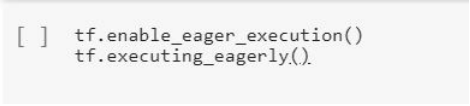
TensorFlow:It is an open source library for numerical computation and large scale machine learning.It help create multidimensional matrices which are similar to numpy but tensorflow matrices which are called tensors can be processed using GPU. Hence Tensorflow is more frequently used in deep learning applications

Numpy : It's the most frequently used library to create arrays with zero data or random data and then utilized to input data in the tensors

matplotlib.pyplot as plt : we use this library to generate output of the images

rest of the libraries are used to change directory and mount google drive folder for colab!!

2.2 Eager Execution



```
[ ]  tf.enable_eager_execution()
     tf.executing_eagerly()
```

Actual Tensor-flow process: creation of graph and then using a session to instantiate the graph
Using Eager Execution: both the above mentioned steps happen simultaneously

2.3 Mounting Drive

```
drive.mount('/content/drive')
```

```
[ ]  os.chdir('/content/drive/My Drive/Artistic style transfer')
     img_list=os.listdir('/content/drive/My Drive/Artistic style transfer')
```

The above commands are used to update the directory and extract the image paths from a folder in drive.

2.4 Loading Image

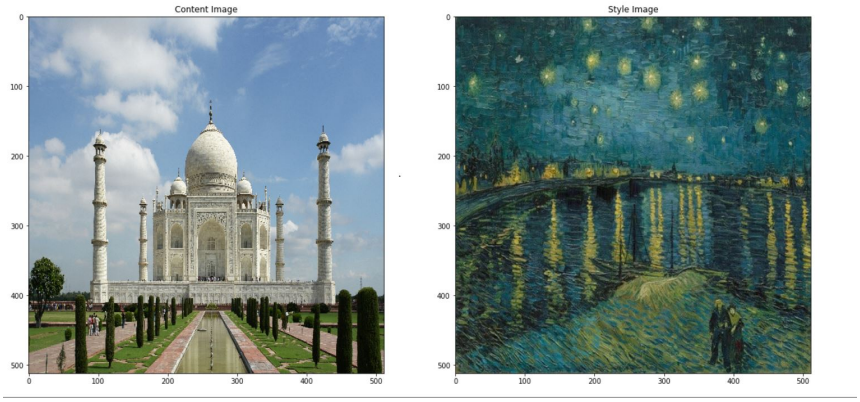
```
[ ]  def load_img(path):
       img=tf.io.read_file(path)
       img=tf.image.decode_image(img)

       img=tf.image.convert_image_dtype(img,dtype=tf.float32)

       img=tf.image.resize(img,(512,512))
       print(img.shape)

       img=img[tf.newaxis,:]
       return img
```

The custom function loadimg is used to :

- Load image into a variable
- Convert the image into array of digits
- Convert image into datatypes of float32
- Resize the image resolution to 512 ,512,3
- Add a new axis to indicate batch
- Return the variable

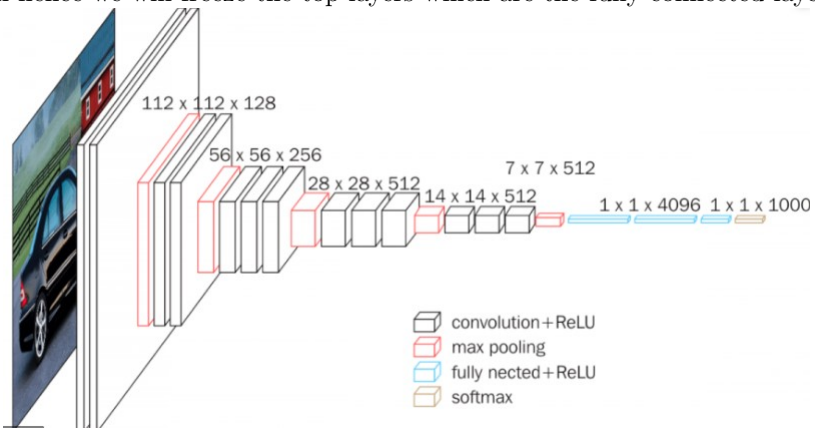
2.5 Plotting Content and Style Image

We will add the texture from the style image to the content image



2.6 Plotting Content and Style Image

VGG16: we will use a pre-trained model for feature extraction and not prediction hence we will freeze the top layers which are the fully connected layers



The input to "cov-1" layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3x3 (which is the smallest size to capture the notion of left/right, up/ 3x3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1x1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3x3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolution layers (not all the conv. layers are followed by

max-pooling). Max-pooling is performed over a 2x2 pixel window, with stride 2. Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

2.7 Layers in VGG16 architecture

```
for i in vgg.layers:
  print (i.name)
```

input_1
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_pool
block4_conv1
block4_conv2
block4_conv3
block4_pool
block5_conv1
block5_conv2
block5_conv3
block5_pool

2.8 Style Layer and Content layer

```
content_layer=['block5_conv2']
style_layer=['block1_conv1',
             'block2_conv1',
             'block3_conv1',
             'block4_conv1',
             'block5_conv1']
num_content_layer=len(content_layer)
num_style_layer=len(style_layer)
```

We use the block 5, which is the last layer for feature extraction for the content correction and we use output from 5 layers for the style(texture correction) of an image

2.9 Process of Feature Extraction

- Instantiate a new model with output as the particular layer output and input as the image input
- This is handled through Keras which is called via tensorflow

```
[ ]  def vgg_layer(layer_names):
       output=[vgg.get_layer(name).output for name in layer_names]
       model=tf.keras.models.Model([vgg.input],output)
       return model
```

2.10 Gram Matrix

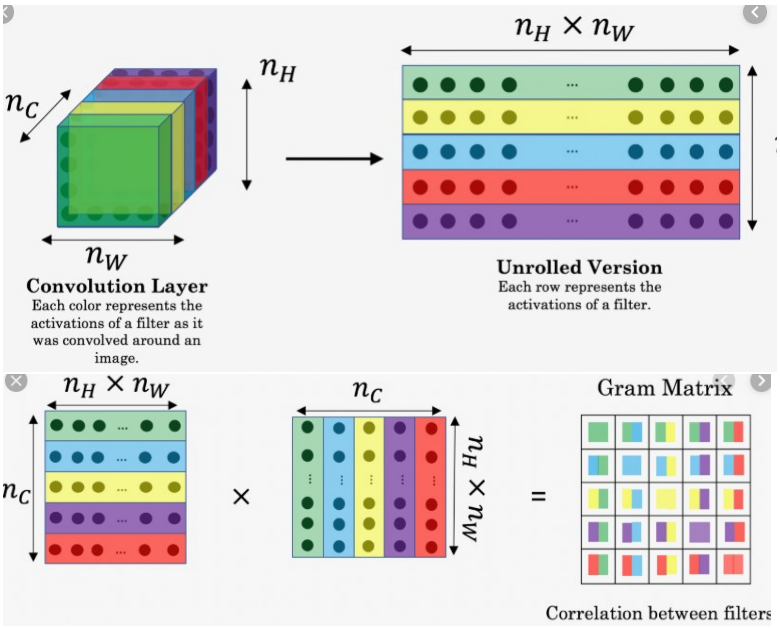
```
[ ]  def gram_matrix(input_tensor):
       result=tf.linalg.einsum('bijc,bijd->bcd',input_tensor, input_tensor)
       input_shape=tf.shape(input_tensor)
       num_loc=tf.cast(input_shape[1]*input_shape[2],tf.float32)
       return result/(num_loc)
```

One of the crucial points in creation of neural style transfer is texture extraction which is done via gram matrix.

Consider two vectors(more specifically 2 flattened feature vectors from a convolutional feature map of depth C) representing features of the input space, and their dot product give us the information about the relation between them. The lesser the product the more different the learned features are and greater the product, the more correlated the features are. In other words, the lesser the product, the lesser the two features co-occur and the greater it is, the more they occur together. This in a sense gives information about an image's style(texture) and zero information about its spatial structure, since we already flatten the feature and perform dot product on top of it.

Now take all C feature vectors(flattened) from a convolutional feature map of depth C and compute the dot product with every one of them(including with a feature vector itself). The result is the Gram Matrix(of size CxC)

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$



2.11 Content and Style Feature Extractor Class

```
class style_content_model():
  def __init__(self,style_layer,content_layer):
    self.vgg=vgg_layer(style_layer+content_layer)
    self.style_layer=style_layer
    self.content_layer=content_layer
    self.num_style=len(style_layer)
    self.vgg.trainable=False
  def __call__(self,inputs):
    inputs=inputs*255
    preprocessed_input=tf.keras.applications.vgg16.preprocess_input(inputs)
    outputs=self.vgg(preprocessed_input)
    style_outputs,content_outputs=(outputs[:self.num_style],outputs[self.num_style:])
    style_outputs=[gram_matrix(i) for i in style_outputs]
    content_dict={content_name:value for content_name, value in zip(self.content_layer, content_outputs)}
    style_dict={style_name:value for style_name, value in zip(self.style_layer, style_outputs)}

    return {'content':content_dict,'style':style_dict}
```

The class takes in content and style layer as input and then generates output in a form of dictionary containing two list one for all the content layers in the input and another for the all the style layer in the input.

2.12 Generate Targets for Training

```
extractor=style_content_model(style_layer,content_layer)
style_targets = extractor(style_image)['style']
content_targets = extractor(content_image)['content']
```

The above layer will use the images that were declared earlier to to extract features and gram matrix and use those as targets to train an input image

2.13 Creating a Dual Loss Function

```
[ ]  style_weight=1e-1
     content_weight=1e4
```

```
[ ]  def style_content_loss(outputs):
       style_outputs=outputs['style']
       content_outputs=outputs['content']
       style_loss=tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
                            for name in style_outputs.keys()])
       style_loss *= style_weight / num_style_layer
       content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[name])**2)
                                for name in content_outputs.keys()])
       content_loss *= content_weight / num_content_layer
       loss = style_loss + content_loss
       return loss
```

We will be using L2 loss for both the content as well as the Style of an image and multiplying them with their respective weights which is lower in case of style and higher in case of content

2.14 Creating an custom optimization function

```
[ ]  opt = tf.keras.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
```

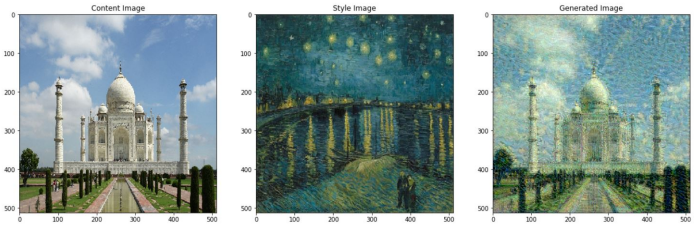
```
[ ]  def clip_0_1(image):
       return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)
```

```
[ ]  def train_step(image):
       with tf.GradientTape() as tape:
         outputs=extractor(image)
         loss=style_content_loss(outputs)
       grad = tape.gradient(loss, image)
       opt.apply_gradients([(grad, image)])
       image.assign(clip_0_1(image))
```

We use the function defined above to generate gradients and optimize an input image with respect to those gradients which are generated using the loss function defined above

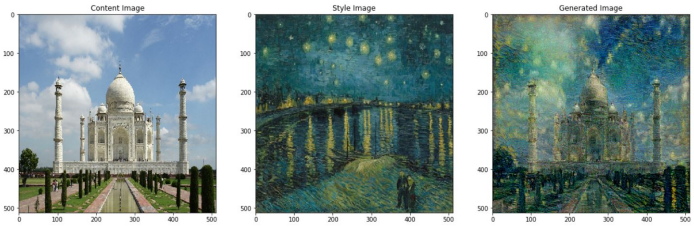
2.15 Results After Training

Epoch : 50



Epoch : 100

Epoch : 200



3 Conclusion

As we can see from the algorithm shown above , transfering texture of an image is merely training a new image to features calculated by the correlation of channels. All the parameters and hyperparameters used have been attained through expriment. The optimizer Adam improves the process by making it run faster than what it was in the research paper also the weights used play a major role and variation in their value may produce different results