Neural style transfer

Problem

Neural style transfer (NST) refers to a class of software algorithms that change digital images or videos in order to adopt the appearance or a visual style of another image. These image transformations are done by deep neural networks which are characteristic for the NST problem. NST can be used for various tasks, such as creating artificial artwork from photographs, a wide range of applications in experimental studies, data augmentation, etc. The innovation of NST is the use of deep learning to separate the representation of the content (structure) of an image, from the appearance (style) in which it is depicted. The original paper that did this first used a convolutional neural network (CNN) VGG-19 architecture that has been pre-trained to perform object recognition using the ImageNet dataset.

Solution

Deep neural networks have become a standard tool for solving computer vision problems, more specifically convolutional neural networks (CNN). CNNs are made up of layers which all individually process visual information using small computational units hierarchically in a feed-forward manner. Each layer extracts and represents a feature (or activation) map.
Along the processing hierarchy of the network, the input image is converted into representations that focus more and more on the *content* of the image compared to the pixel values. Layers in the higher level take into account high-level content (objects and their arrangement), while the layers in the lower level reproduce the exact pixel values of the original image.
This is very useful for style transfer, as we are able to extract the information from the lower layers of an image and use that to get a hold of the image *style*. This information explains the correlation between the different filter responses over the space of the activation maps. We can extract the multi-scale representation of the input image capturing its texture information (not the global arrangement of objects) by using this correlation from multiple layers. By increasing the receptive field sizes and feature complexity, the complexity and size of the image structures increase along the hierarchy. We refer to this multi-scale representation as style representation.
The spatial arrangement of the original objects in the image remains the same, but the colors and textures of the artwork image are used on the original. This renders the original image in the style of the artwork, such that the appearance of the synthesised image resembles the work of art, even though it shows the same original content.
In order to create a smooth visual experience, the style representations are matched to the higher layer representations of the content of the image, since those layers contain the local structures on a very large scale.
When synthesising an image that combines the content of one image with the style of another, there usually does not exist an image that perfectly matches both constraints at the same time. However, the loss function we minimise during image synthesis contains two terms for content and style respectively, which are separated. In this way we can regulate if we want the output

image to be more based on the artwork, or more focused on the content. The neural representations for the content and style of an image are derived from the feature responses of Deep Neural Networks trained on object recognition, specifically VGG19.

Network Design

Convolution Neural Network(CNN)

In order for us to understand the **VGG19** Model, we first have to grasp the basic understanding of CNN.
Convolutional Neural Network (CNN) is a neural network which extracts or identifies a feature in a particular image. This forms one of the most fundamental operations in Machine Learning and is widely used as a base model in the majority of Neural Networks like GoogleNet, VGG19 and others for various tasks such as Object Detection, Image Classification and others.

CNN has the following five basic components:

Convolution : to detect features in an image
ReLU : to make the image smooth and make boundaries distinct
Pooling : to help fix distorted images
Flattening : to turn the image into a suitable representation
Full connection : to process the data in a neural network

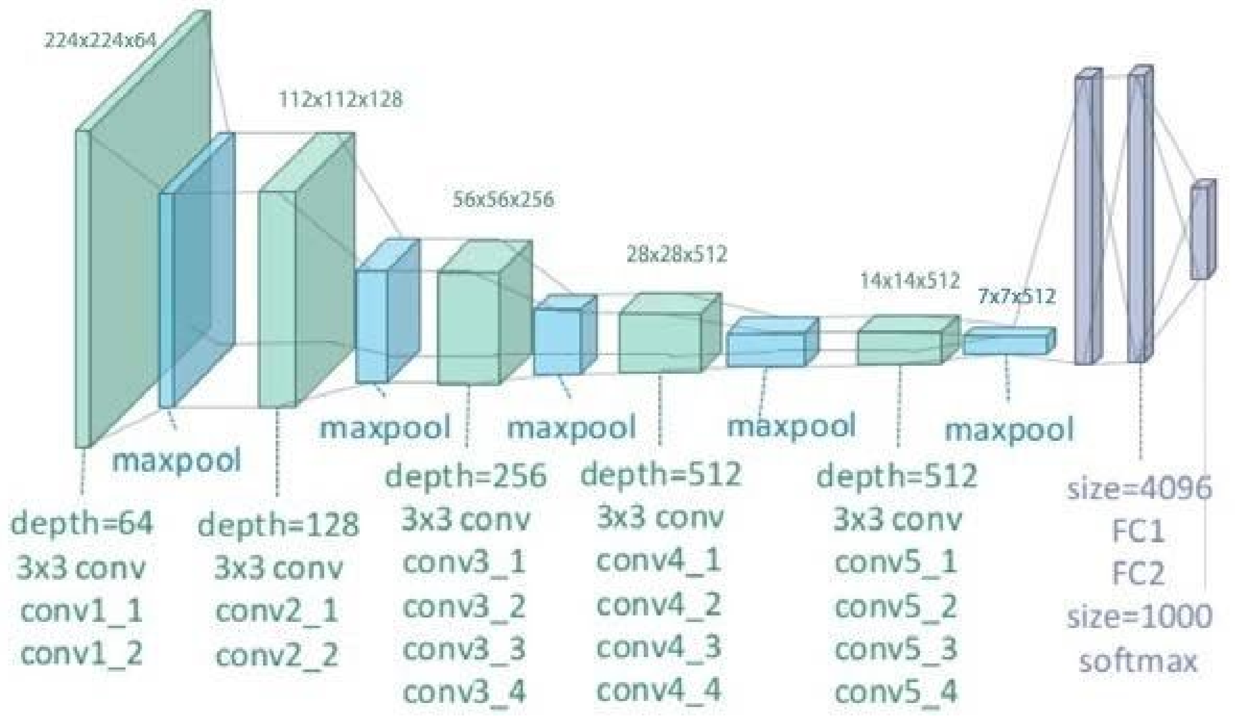
VGG19 - Convolutional Network for Classification and Detection

VGG-19 is a trained Convolutional Neural Network, from **V**isual **G**eometry **G**roup, Department of Engineering Science, University of Oxford. The number 19 stands for the number of layers with trainable weights. 16 Convolutional layers and 3 Fully Connected layers.

Architecture

- A fixed size of (224 * 224) RGB image was given as input to this network which means that the matrix was of shape (224,224,3).
- The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
- Used kernels of (3 * 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
- Spatial padding was used to preserve the spatial resolution of the image.
- Max pooling was performed over a 2 * 2 pixel windows with stride 2.
- This was followed by Rectified linear unit(ReLu) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions that proved much better than those.

- Implemented three fully connected layers from which first two were of size 4096 and after that a layer with 1000 channels for 1000-way ILSVRC classification and the final layer is a softmax function.



Content and style extraction from an image

Higher layers in the network capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction. In contrast, reconstructions from the lower layers simply reproduce the exact pixel values of the original image. We therefore refer to the feature responses in higher layers of the network as the content representation.
In higher layers of the network, detailed pixel information is lost while the high-level content of the image is preserved.

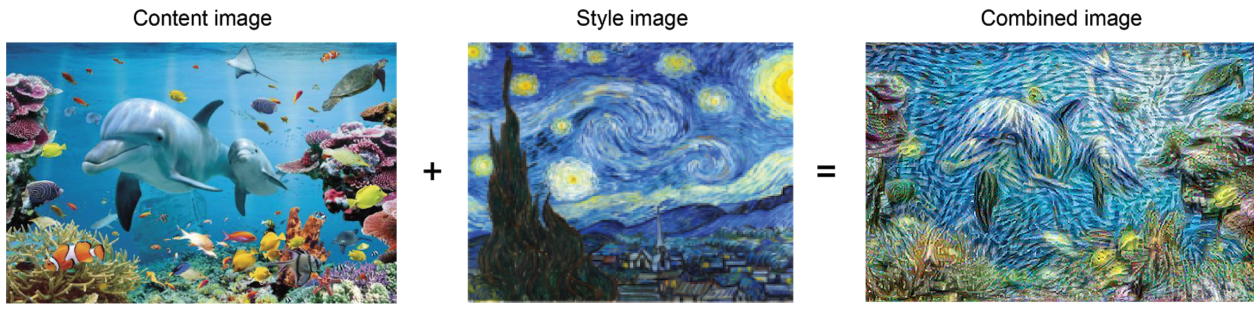
Content Reconstructions

We can visualise the information at different processing stages in the CNN by reconstructing the input image from only knowing the network's responses in a particular layer. We reconstruct the input image from layers 'conv1 1', 'conv2 1', 'conv3 1', 'conv4 1' and 'conv5 1' of the original

VGG-Network. We find that reconstruction from lower layers is almost perfect. In higher layers of the network, detailed pixel information is lost while the high-level content of the image is preserved.
The 4_2 Convolution layer of the pre-trained VGG-19 network is used as a content-extractor.

Style Reconstructions

On top of the original CNN representations we built a new feature space that captures the style of an input image. The style representation computes correlations between the different features in different layers of the CNN. We reconstruct the style of the input image from style representations built on different subsets of CNN layers (‘conv1 1’ (a), ‘conv1 1’ and ‘conv2 1’ (b), ‘conv1 1’, ‘conv2 1’ and ‘conv3 1’ (c), ‘conv1 1’, ‘conv2 1’, ‘conv3 1’ and ‘conv4 1’ (d), ‘conv1 1’, ‘conv2 1’, ‘conv3 1’, ‘conv4 1’ and ‘conv5 1’ (e)). This creates images that match the style of a given image on an increasing scale while discarding information of the global arrangement of the scene.



Loss functions (content and style distances)

Content Loss

Our content loss definition is actually quite simple. We’ll pass the network both the desired content image and our base input image. This will return the intermediate layer outputs (from the layers defined above) from our model. Then we simply take the euclidean distance between the two intermediate representations of those images.

More formally, content loss is a function that describes the distance of content from our input image x and our content image, p . Let C□□ be a pre-trained deep convolutional neural network. Again, in this case we use VGG19. Let X be any image, then C□□(x) is the network fed by X. Let F^{l}_{i}□(x)∈ C□□(x)and P^{l}_{i}□(x) ∈ C□□(x) describe the respective intermediate feature representation of the network with inputs x and p at layer l . Then we describe the content distance (loss) formally as:

$$L^{l}_{content}(p, x) = \sum_{i,j} (F^{l}_{ij}(x) - P^{l}_{ij}(p))^2$$

We perform backpropagation in the usual way such that we minimize this content loss. We thus change the initial image until it generates a similar response in a certain layer (defined in content_layer) as the original content image.
This can be implemented quite simply. Again it will take as input the feature maps at a layer L in a network fed by x, our input image, and p, our content image, and return the content distance.

Style Loss

Computing style loss is a bit more involved, but follows the same principle, this time feeding our network the base input image and the style image. However, instead of comparing the raw intermediate outputs of the base input image and the style image, we instead compare the Gram matrices of the two outputs.

Mathematically, we describe the style loss of the base input image, x, and the style image, a, as the distance between the style representation (the gram matrices) of these images. We describe the style representation of an image as the correlation between different filter responses given by the Gram matrix G^{l}, where G^{l}_{i}□ is the inner product between the vectorized feature map i and j in layer l. We can see that G^{l}_{i}□ generated over the feature map for a given image represents the correlation between feature maps i and j.

To generate a style for our base input image, we perform gradient descent from the content image to transform it into an image that matches the style representation of the original image. We do so by minimizing the mean squared distance between the feature correlation map of the style image and the input image. The contribution of each layer to the total style loss is described by

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^{l}_{ij} - A^{l}_{ij})^2$$

where G^{l}_{i}□ and A^{l}_{i}□ are the respective style representation in layer l of input image x and style image a. Nl describes the number of feature maps, each of size Ml=height∗width. Thus, the total style loss across each layer is

$$L_{style}(a, x) = \sum_{l \in L} w_l E_l$$

where we weigh the contribution of each layer's loss by some factor wl. In our case, we weight each layer equally:

$$(w_l = \frac{1}{||L||})$$

Total Loss

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha\mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta\mathcal{L}_{style}(\vec{a}, \vec{x})$$

where alpha and beta are weights for content and style, respectively. They can be tweaked to alter our final result.

So our total loss function basically represents our problem- we need the content of the final image to be similar to the content of the content image and the style of the final image should be similar to the style of the style image.

Now all we have to do is to minimize this loss. We minimize this loss by changing the input to the network itself. We basically start with a blank grey canvas and start altering the pixel values so as to minimize the loss. Any optimizer can be used to minimize this loss. Here, L-BFGS is used and obtained quite good results on this task.

Run Gradient Descent

We iteratively update our output image such that it minimizes our loss: we don't update the weights associated with our network, but instead we train our input image to minimize loss. In order to do this, we must know how we calculate our loss and gradients. Note that the L-BFGS optimizer, which is used in this project, is much more efficient than Adam optimizer and is more preferred.

Gram Matrix

The correlation between feature maps is called the gram matrix.
Gram matrix is simply the matrix of the inner product of each vector and its corresponding vectors in the same. It found use in the current machine learning is due to deep learning loss where while style transferring the loss function is computed using the gram matrix.

$$G_{ij} = \langle v_i , v_j \rangle$$

Gram matrix is computed over the flatten feature vector and says that if two features are very close then its value is large so its more loss that is two features are almost the same. So, do not transfer the features that are close (more similar) i.e. do not transfer the content of the image.

Image = content + style

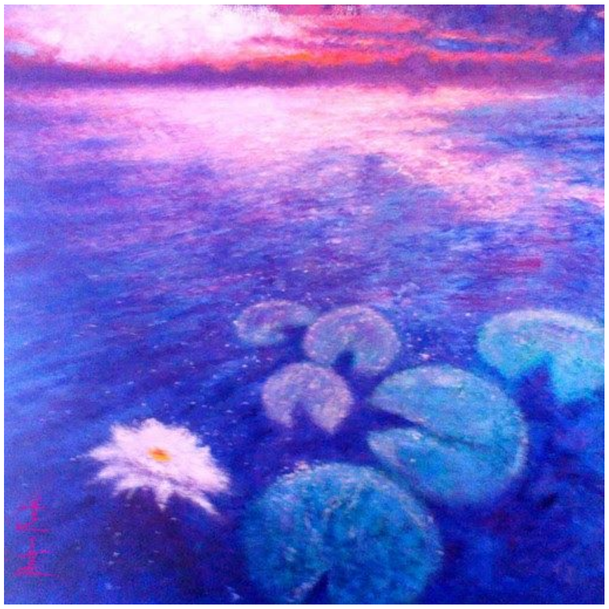
Otherwise, the earlier gram matrix used in Mercer's theorem was to prove the existence of a kernel in the feature space that is the gram matrix must be positive definite to exist which implies that space exists.

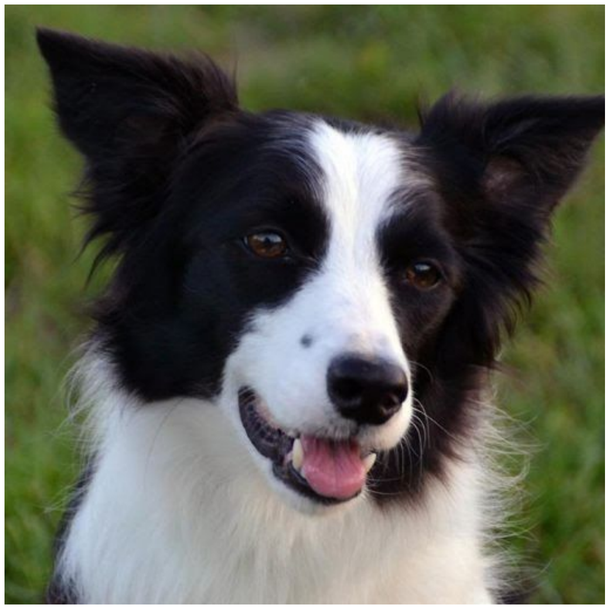
$$G(x_1, \ldots, x_n) = \begin{vmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \ldots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & \ldots & \langle x_2, x_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_n, x_1 \rangle & \langle x_n, x_2 \rangle & \ldots & \langle x_n, x_n \rangle \end{vmatrix}.$$

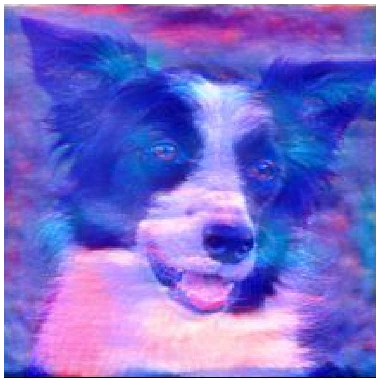
Other applications include the calculation of the correlation coefficient, Also the positivity (determinant not equal to zero) implies that the K-dimensional (for K x K matrix) space is linearly independent.

The layers used for calculating the gram matrix: 'conv1_1', 'conv2_1' and 'conv3_1' , 'conv4_1, conv5_1 with varied style weight variant constant for each layer.

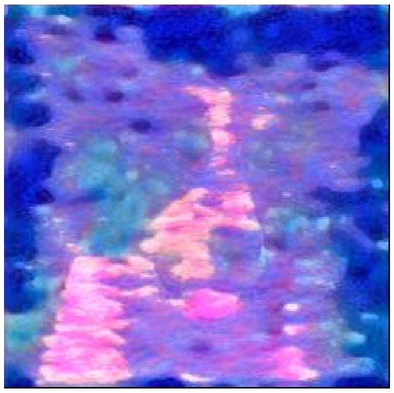
Experiments & Results

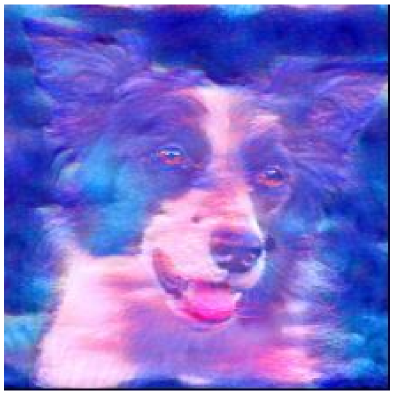

Style image

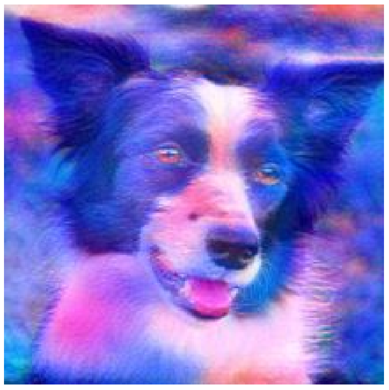

Content image



As stated previously, the layers used for reconstructing the style and content of the image are taken from the VGG19 network.The style makes use of layers 1 to 5, while content information is extracted from layer 4. This however is not strict. Image on the left shows the reconstruction result using the default parameters from the original network.
It is possible to adjust whether the network should focus on the content or the style. We can do this by changing the layer of extraction accordingly.

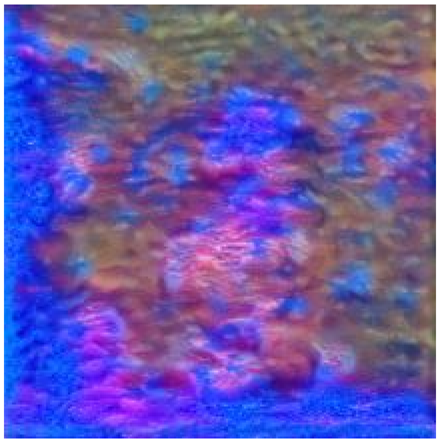

[1]


[2]


[3]

In example [1], content was reconstructed from layer 2, meaning it had taken into account only lower level features in reconstructing the original image. The style layers remain the same (1-5).
In example [2], the content was reconstructed from layer 6, taking into account higher level information of the image along with details from the previous layer.
In example [3], the content was reconstructed from layer 6, while style was taken from layers (1-3). As we can see, the content of the original image is very detailed, while seemingly only the colors from the style image remain.

The VGG19 network [Network Design; Architecture] contains max pool layers, which are generally not used for neural style transfer. In an attempt to try the reconstruction using average pool, which is more preferred for this type of task, we replaced the max pool layers of the network with average pool. However, because the network had been trained and optimized on max pooling, there had been events on some iterations of exploding gradients when applying style transfer on the modified network. The network wasn't retrained to use average pooling although fine tuning it on a smaller dataset would give better results.



We also modified the other hyperparameters, such as initial weights, number of steps, normalization parameters, however the initial parameters show the best results in terms of the style/content tradeoff.

References

- A Neural Algorithm of Artistic Style;
  Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, Paper
- ImageNet Classification with Deep Convolutional Neural Networks;
  Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Paper
- Demystifying Neural Style Transfer;
  Yanghao Li, Naiyan Wang, Jiaying Liu, Xiaodi Hou, Paper
- Neural Style Transfer : Introduction and Implementation
- Neural Style Transfer: Creating Art with Deep Learning using tf.keras and eager execution
- Understanding the VGG19 Architecture
- Tell me again how much Gram is in the matrix?