

Image Style Transfer Using Convolutional Neural Networks

MVS Charan (IMT2015023), P Sai Rakshith (IMT2015032),
Venkata K C Ritvik (IMT2015047)

December 2018

1 Problem Statement

Neural Style Transfer - Use deep learning to compose images in the style of another image. We'll be using this to convert normal images into artistic images.

2 Dataset Description

The dataset is a collection of images that are used as content images, style images and masks. Further details about how they will be used will be mentioned in the later parts of the report.

3 Introduction

Neural style transfer is an optimization technique used to take two images, a content image, a style reference image, and blend them together such that the input image is transformed to look like the content image, but “painted” in the style of the reference image. Previously used techniques involved different target image representation to preserve the semantic content of the target image. This paper that we implemented uses state-of-the-art convolutional neural networks. A fundamental prerequisite is to find image representations that independently model variations in the semantic image content and the style in which it is presented. In summary, we'll take the base input image, a content image that we want to match, and the style image that we want to match. We'll transform the base input image by minimizing the content and style distances (losses) with backpropagation, creating an image that matches the content of the content image and the style of the style image. (As shown in fig 1 and 2)

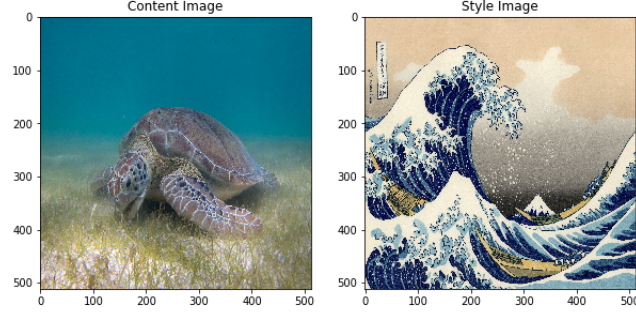


Figure 1: Content Image and Style Image

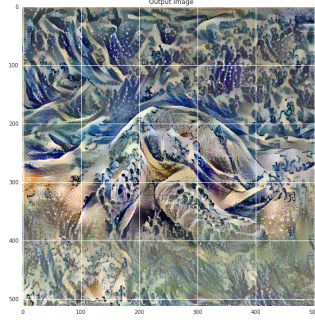


Figure 2: Result Image

4 Algorithm Details

4.1 Content Loss Representation

For content loss representation, each layer in the network defines a non-linear filter bank whose complexity increases with the position of the layer in the network. So, for a given input image x is encoded in each layer of the CNN by the filter responses to that image. A layer with N_l distinct filters has N_l feature maps each of size M_l , where $M_l = (\text{height of the feature map}) * (\text{width of the feature map})$. So the responses in a layer l can be stored in a matrix $F^l \in \mathbb{R}^{N_l M_l}$ where F_{ij}^l is the activation of the i^{th} filter at position j in layer l .

Let p and x be the original image and the image that is generated, and P_l and F_l their respective feature representation in layer l . We then define the squared-error loss between the two feature representations.

The derivative of this loss with respect to the activations in layer l as shown in Fig 4, from which the gradient with respect to the image x can be computed

$$\mathcal{L}_{\text{context}}(\vec{F}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - p_{ij}^l)^2$$

Figure 3: Squared error loss function

using standard error back-propagation.

$$\frac{\partial \mathcal{L}_{\text{context}}}{\partial F_{ij}^l} = \begin{cases} (F^l - p^l)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}$$

Figure 4: Derivative of L

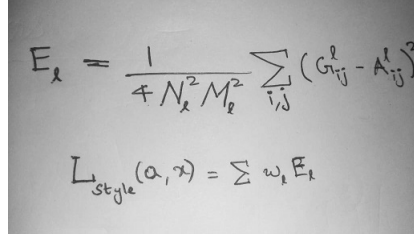
4.2 Style Loss Representation

For style loss representation, 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_{ij}^l is the inner product between the vectorized feature map i and j in layer l . G_{ij}^l generated over the feature map for a given image represents the correlation between feature maps i and j .

Now we perform gradient descent from the content image to transform it into an image that matches the style representation of the original image. This is obtained by minimizing the mean squared distance between the feature coorelation map f the style image and the input image. Contribution of each layer to the total style loss is described as shown in the first equation in Fig 5. Here G_{ij}^l and A_{ij}^l are the respective style representation in layer l of input image x and style image a . N_l is the number of feature maps. M_l is the size of each feature map. Therefore the total style loss across each layer is as shown in the second equation in Fig 5.

4.3 Style Transfer

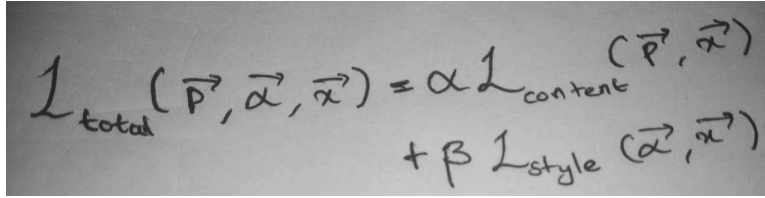
To transfer the style, we make a new image that matches the content representation of the photograph and the style representation of the artwork. The loss function we minimize is as shown in fig 6. Here, α and β are the weighing factors for content and style reconstruction respectively. We always re-size the style image to the same size as the content image before computing its feature representations.



$$E_k = \frac{1}{4 N_k^2 M_k^2} \sum_{i,j} (G_{ij}^k - A_{ij}^k)^2$$

$$L_{style}(\alpha, x) = \sum w_k E_k$$

Figure 5: Total style loss contributed by each layer and Total style loss across each layer



$$L_{total}(\vec{p}, \vec{\alpha}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{\alpha}) + \beta L_{style}(\vec{\alpha}, \vec{x})$$

Figure 6: Total loss function

5 Implementation

We load VGG19 and feed in our input tensor to the model. This will allow us to extract the feature maps of the content, style, and generated images. To access the intermediate layers corresponding to our style and content feature maps, we get the corresponding outputs by using the Keras Functional API to define our model with the desired output activations. Now, we define our style and content loss functions as shown in fig 7.

We used the Adam optimizer in order to minimize our loss function. We iteratively updated our output image such that it minimizes our loss, but we don't update the weights associated with the network. After computing the loss and gradients, they go as inputs for the style transfer process.

We are implementing style transfer in three different ways.

- 1) Simple Style Transfer - Normal style transfer with one content image and one style image.
- 2) Image Segmentation Style transfer with an additional mask - Each image has a semantic segmentation mask which will divide the original image into a small set of meaningful segments with considerable size.
- 3) Multiple Style transfer - Style transfer with one content image and multiple style images with weights associated with each of them. So, the input takes multiple style images and input float number ($0 < x < 1.0$) for each of them as weight.

```

def content_layer_loss_func(p, x):
    _, h, w, d = p.get_shape()
    m = h.value * w.value
    n = d.value
    k = 1. / (2. * n**0.5 * m**0.5)
    loss = k * tf.reduce_sum(tf.pow((x - p), 2))
    return loss

def style_layer_loss_func(a, x):
    _, h, w, d = a.get_shape()
    m = h.value * w.value
    n = d.value
    A = gram_matrix(a, m, n)
    G = gram_matrix(x, m, n)
    loss = (1./(4 * n**2 * m**2)) * tf.reduce_sum(tf.pow((G - A), 2))
    return loss

```

Figure 7: Loss functions

```

def stylize(content_img, style_imgs, init_img, frame=None):
    with tf.device(args.device), tf.Session() as sess:
        net = build_model(content_img, args.model_weights, args.pooling_type)
        if args.style_mask:
            L_style = sum_masked_style_losses(sess, net, style_imgs)
        else:
            L_style = sum_style_losses(sess, net, style_imgs)
        L_content = sum_content_losses(sess, net, content_img)
        L_tv = tf.image.total_variation(net['input'])
        alpha = args.content_weight
        beta = args.style_weight
        theta = args.tv_weight
        L_total = alpha * L_content
        L_total += beta * L_style
        L_total += theta * L_tv
        optimizer = get_optimizer(L_total)
        if args.optimizer == 'adam':
            minimize_with_adam(sess, net, optimizer, init_img, L_total)
        elif args.optimizer == 'lbfgs':
            minimize_with_lbfgs(sess, net, optimizer, init_img)
        output_img = sess.run(net['input'])
        write_image_output(output_img, content_img, style_imgs, init_img)

```

Figure 8: Outline of our code

6 Results

Here are the results for the different types of style transfer implemented (In Fig 9, 10, 11). We can notice how the style is transferred from the style images to content images. In fig 9, we can see the simple style transfer with a single style reference image. In fig 10, a mask has been used on the penguin so that we can redefine the image as smaller segments of reasonable size and style transfer is done to those parts. It is somewhat similar to texture mapping where we try

to map the coordinates from one image to another shape and give colour, but here we are transferring the style in these regions. In fig 11, we can notice that the result image has style transferred from two different style reference images with equal weight values for each of them.

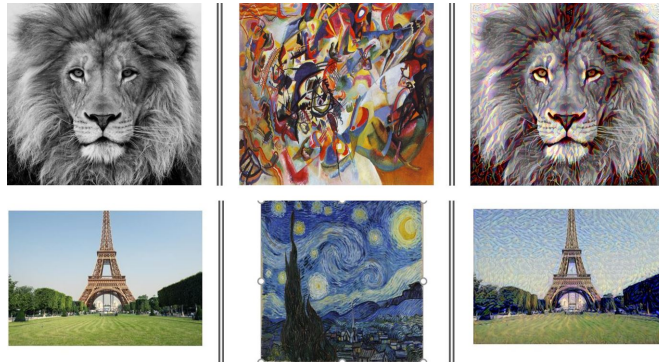


Figure 9: Simple style transfer

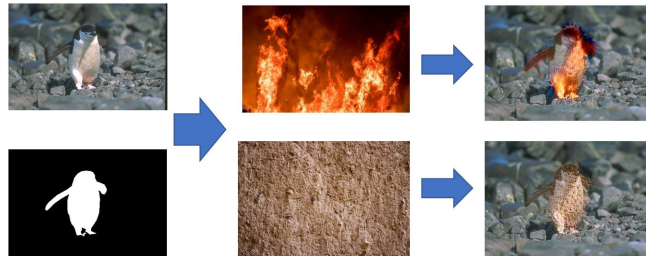


Figure 10: Segmentation style transfer



Figure 11: Multiple Style transfer

7 Project and Dataset submission

Here is the github link for the project code and dataset. The dataset can be found in the "image_input" directory.

<https://github.com/Charan000/NN-Project>

8 Code Execution

OpenCV and tensorflow are the required libraries to run the project. Open a terminal, navigate to the project directory and run the following command lines in the terminal.

Here is an example of how to run the code.

For Simple Style Transfer -

```
python neural_style1.py -content_img pengu.jpg -style_imgs rok.jpg -max_size 1000 -max_iterations 50 -device /gpu:0
```

For Multiple Style Transfer -

```
python neural_style1.py -content_img pengu.jpg -style_imgs rok.jpg starry-night.jpg -style_imgs_weights 0.5 0.5 -max_size 1000 -max_iterations 50 -device /gpu:0
```

For Segmentation Style Transfer -

```
python neural_style1.py -content_img pengu.jpg -style_imgs rok.jpg -style_mask 1 -style_mask_imgs face_mask.png -max_size 1000 -max_iterations 50 -device /gpu:0
```

9 Future Work

There are some technical limitations to the algorithm. Probably the most limiting factor is the resolution of the synthesised images. Something that can be improved over this is that the segmentation can be done in such a way that there is one-one relation between specific parts of the images. For example, we can make the style transfer better by matching the eyes, nose or some other features of the images.

References

Image Style Transfer Using Convolutional Neural Networks

https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf

<http://cs231n.github.io/convolutional-networks/>