Using Neural Networks to Create Art

Algot Sandahl[1], David Robín Karlsson[1], Arvid Magnusson[1]

Abstract
This paper present an implementation of using deep learning to transfer a style from an image and apply it to another one. A pretrained convolutional neural network (VGG19) is used where data from certain layers are extracted to calculate and minimize a combined loss function. The loss function consists of content loss, style loss and total variation loss. Using Tensorflow to implement the algorithm, a large amount of images with an arbitrary style applied to them were produced.

Keywords
Convolutional neural networks, Neural style transfer

[1] *Media Technology student, Linköping University, Norrköping*

Contents

1. Introduction

Convolutional neural networks and deep learning have become prominent tools in solving modern problems. However, the networks are more flexible than one could think and could not only be used to solve problems but also as a tool for artistic expression. This paper details the theory and implementation of an artificial intelligence capable of learning the style of an image and applying it to another, arbitrary image.

2. Theory

Neural style transfer is an optimization technique used to take three images, a content image, a style image and a generated image that can be initialized with noise or the content image. The generated image is then morphed so that it looks like the content image but "painted" like the style image. This is achieved using a convolutional neural network (CNN). The method was originally presented by Gatys, Ecker, and Bethge in 2015 [1].

A CNN can be used for many tasks, one of them being to analyze visual imagery. A pre-trained network called VGG19 was used for this project. It was developed for image recognition and localization tasks. The network consists of 19 layers and has been trained on the ImageNet dataset which contains millions of images. For neural style transfer, the output of the network is of no interest, but rather the intermediate layers.

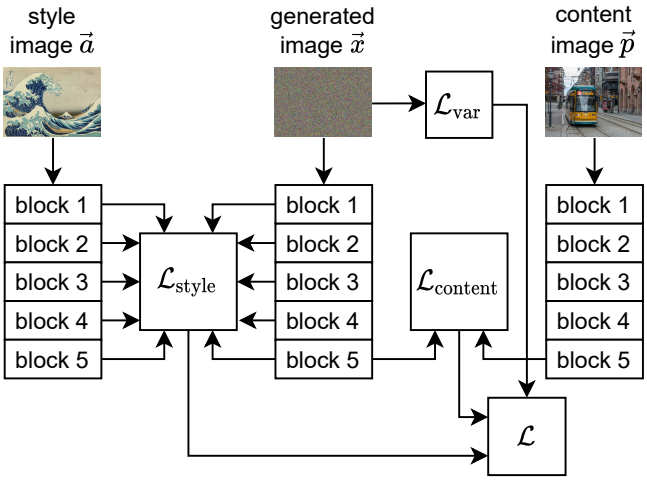

Figure 1. An overview of how the loss function $\mathscr{L}(\vec{p},\vec{a},\vec{x})$ is calculated.

In order for the network to classify an image, it must understand it. It does this by taking the image, and through the layers of the model it builds an understanding of the features present within the image. So, somewhere in between where the raw image is fed in and the classification label is output, the network is able to describe the content and the style of the image.

The neural style transfer process involves minimizing a loss function consisting of three parts: style loss, content loss and total variation loss. An overview of the calculation of the loss function is shown in figure 1.

Content loss measures the distance between the feature representations of the content image and the generated image. Usually only one layer is used for the content loss. The choice of layer usually falls on one of the latter layers of the network,

since these represent higher level features as opposed to the first layers which have a more direct connection to the pixels.

Style loss is more complex but quite similar to content loss. The distance is computed between two Gram matrices which acts as style representations of the generated image and the style image. The Gram matrices themselves represent the correlation between the different feature maps extracted from the neural network. This loss is calculated for multiple layers to capture style features for multiple abstraction levels.

Total variation loss is calculated from the difference between neighboring pixels in the generated image and is used to reduce noise by encouraging the generated image to exhibit piece-wise coherent areas. Our implementation is based on [2], extended with an internal exponential weight to force the optimizer to heavily iron out noise.

The loss is used by the optimizer to iteratively update the genereated image such that it matches the content of the content image, and the style of the style image. Note that the weights of the network remain static. Two optimizers based on *stochastic gradient descent* were used in this project [3].

The optimizer *SGD* is based on using momentum to successfully navigate troublesome curves commonly found near local optima, such as steep climbs in one dimension.

The optimizer SGD was used initially but was switched to another optimizer, *Adam*, after trials displayed better loss-minimizing performance. Adam is based on adaptive moment estimation, an extended version of SGD's momentum approach. Adam adapts the learning rates for each parameter and also incorporates a decaying average of past gradients to find a better gradient, thus minimizing the loss better.

3. Method

The implementation was done using Tensorflow and Keras. As stated in the theory chapter, the task is to define three loss functions and then iteratively transform the input image in order to minimize those functions as described in [1].

First the images must be preprocessed according to how the VGG network was trained. The network expects an image with each channel normalized by the mean values $[103.939, 116.779, 123.68]$ and the channels being *BGR*. In order to view the correct output after the optimization, the inverse of this step should be applied before viewing the image.

The content loss for layer l is defined as

$$\mathscr{L}_{\text{content}}(\vec{p},\vec{x},l) = \sum_{i,j}\left(F^l_{ij} - P^l_{ij}\right)^2 \tag{1}$$

where P^l and F^l are the feature representation at layer l of the content image $\vec{p}$, and the generated image $\vec{x}$ respectively. The index i denotes the filter, and j denotes the position in the filter. Layer two of the fifth block of the network was primarily used in this project.

The Gram matrix for the generated image is defined by

$$G^l_{ij} = \sum_k F^l_{ik}F^l_{jk} \tag{2}$$

where F^l is the feature representation at layer l. The distance between the two gram matrices is defined as

$$E_l = \frac{1}{4N_l^2M_l^2}\sum_{i,j}\left(G^l_{ij} - A^l_{ij}\right)^2 \tag{3}$$

where A^l_{ij} is the Gram matrix of the style image, N_l is the number of feature maps, and M_l is the size of the feature maps. The style loss function is then defined as

$$\mathscr{L}_{\text{style}}(\vec{a},\vec{x}) = \sum_l w_l E_l \tag{4}$$

where w_l is a weight which determines each layer's contribution to the loss. For this project w_l was defined as one divided by the number of style layers used.

The total variation loss is defined as

$$\mathscr{L}_{\text{var}}(\vec{x}) = \sum_{i,j}\left((\vec{x}_{i+1,j} - \vec{x}_{i,j})^2 + (\vec{x}_{i,j+1} - \vec{x}_{i,j})^2\right)^{1.25} \tag{5}$$

where $\vec{x}_{i,j}$ is a single pixel in the generated image and the exponent 1.25 is the exponential weight used to amplify loss in a noisy generated image.

Finally, the total loss function is defined as a weighted sum of the individual loss functions:

$$\begin{aligned}\mathscr{L}(\vec{p},\vec{a},\vec{x}) =& \alpha\mathscr{L}_{\text{content}}(\vec{p},\vec{x}) \\ &+\beta\mathscr{L}_{\text{style}}(\vec{a},\vec{x}) \\ &+\gamma\mathscr{L}_{\text{var}}(\vec{x})\end{aligned} \tag{6}$$

where α, β and γ are weights for each separate loss function.

4. Result

An example of a generated image together with the content and style images can be seen in figure 2. The role different layers play in style loss is demonstrated in figure 3. The effect of the total variation weight γ is exemplified in figure 4.

5. Discussion

The method of optimizing the neural network for each individual content and style image on every render is a computationally intensive task. While a high-end GPU can finish the task in around ten minutes, this approach is not viable for consumer electronics such as smartphones or laptops. There is a solution to this which requires a fundamental redesign of the current workflow. A network can be trained to learn a particular style which would allow near or true real-time application of the style to an image. This could be used to style video sequences, live streams and real time camera filters.

An issue that surfaced during the development of the algorithm is the tuning of different parameters. Although a lot of parameters, such as learning- or decay rate for the optimizer, can be somewhat intuitively tuned with good knowledge of the theory behind it, some parameters require a trial and error approach. The loss weights α, β and γ were tuned using this

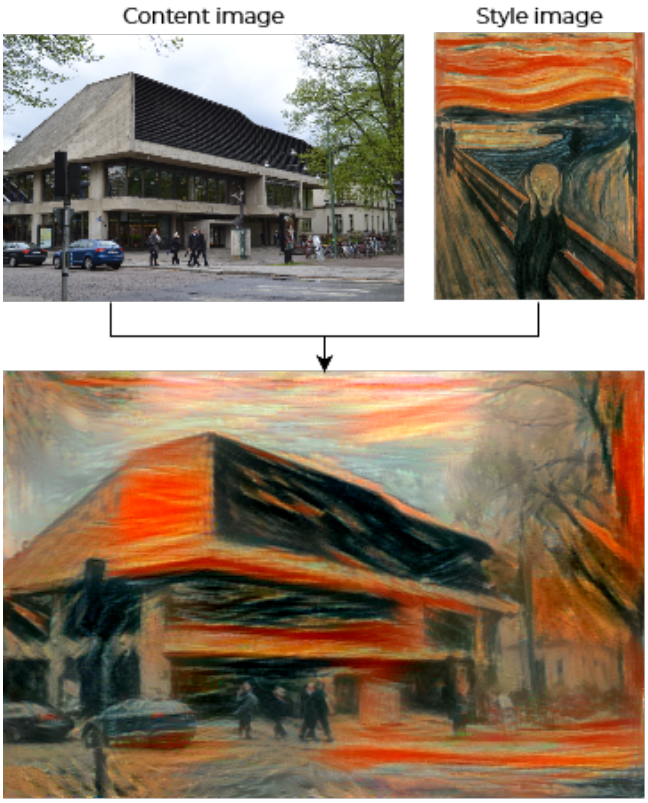

Figure 2. Norrköping city library in the style of The Scream.

approach which prolonged development time as a trial of new parameter settings required at least ten minutes of render time to evaluate. The results themselves are not always straightforward to interpret either since it might be difficult to discern which loss weight contributed to what in the final image. This is a classic issue in neural networks since knowing what a network actually does in the hidden layers is nearly impossible, therefore it's hard to debug certain parts of algorithms using neural networks.

In addition to the weights α, β, and γ, the layers used in the calculations also affect the result. As seen in figure 3, the lower layers ensure that the color and texture of the style image is matched, while the higher layers match higher level style features, in this example the waves.

6. Conclusion

While being a computationally laborious task, style transfer using deep learning in neural networks can be used to generate great results. The technology has potential for commonplace usage in consumer applications.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015.

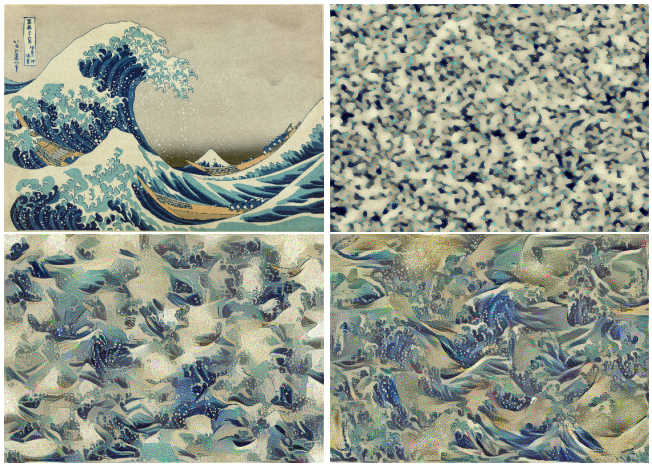

Figure 3. The effect of layer choice on style reconstruction. The first image is the style image and the following images are style reconstructions ($\alpha = \gamma = 0$) using the first layer of blocks [1], [1, 2, 3], and [1, 2, 3, 4, 5].

Figure 4. A demonstration of different amounts of total variation loss used. The first image uses no total variation loss ($\gamma = 0$) and the rest of the images use increasing amounts.

[2] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1, pp. 259–268, 1992.

[3] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.