Art style transfer using tensorflow Core v2.4.1

Qirui Zhu and Frederik Wollatz

14.03.2021

Keywords: Art style transfer,style transfer, Deep Learning, tensorflow2;

1 Introduction

The following report describes our re-implementation of the work by Gatys et al. (2015a), showing that, by using a pretrained object detection Convolutional Neural Network (CNN), one can transfer the (art) style of one image onto the content of another image. Thus giving us insight on how CNNs can learn to distinguish between the style and the content of an image.

2 Task Description

Our task is to re-implement the work of Gatys et al. (2015a) demonstrating that we can reconstruct an image based on the style of one image and the content of another image.

The **content image** depicts the objects, persons, situations and the arrangement of these. The **style image** is used as a reference for the style of an artist: the thickness of the brush, the amount of different colours used and the texture of the canvas. By generating a third image based on these two images, we get the **reconstructed image**. The reconstructed image is a representation of the content of the content image and the art style of the style image.

3 Summary of related approaches

The idea of transferring the style of one picture to the content of another one has been described and implemented in Pytorch by Gatys et al. (2015a). Since then it has been

implemented multiple times, for example by Joan and Olson (2021), TensorFlow (2021) and Yuan (2018). Whilst most of these implementation stick close to the original, there are some differences that should be acknowledged. Yuan (2018), Athalye (2015) and Engstrom (2016) do not use the L-BFGS optimizer, as recommended in the original paper, but Adam. Nonetheless their results are still appealing. Additionally, Yuan (2018) and Engstrom (2016) do not generate their reconstructed image from a randomly initialized white noise image, as proposed in the paper, but from the content image itself. Another difference lies in the usage of the VGG19 network with average- instead of max-pooling. The paper by Gatys et al. (2015a) proposed that average-pooling grants visually more appealing results.

The usage of style transfer-algorithms is widespread and reaches from photo-editors like the AI-Painter (Instapainting, 2021) to changing the style of videos (Ruder et al., 2016) and even the style of video games (Google Inc., 2019).

4 Theory

One of the biggest contributions by Gatys et al. (2015a) was the separability of content and style. How content and style representation are defined, can be seen in 4.1 and 4.2.

4.1 Content

Convolutional Neural Networks (CNNs) are widely used in diverse image-processing tasks, e.g. object recognition. CNNs are made up from multiple (intermediate) layers, which the data is fed through one layer after another. All these layers filter out one feature of the image. These filtered images are called feature maps. Whilst low-level feature maps (from earlier layers) represent the content more on a pixel basis, the higher-level feature maps represent content in a more generalized way (e.g. shapes). By extracting feature maps from intermediate layers from an object recognition model like VGG19 (Simonyan and Zisserman, 2015), we can get a representation of the content. To get an estimate of what these feature maps look like, the reconstruction image of an image, that are based solely on one single layer can be used. An example for this is shown in figure 1.

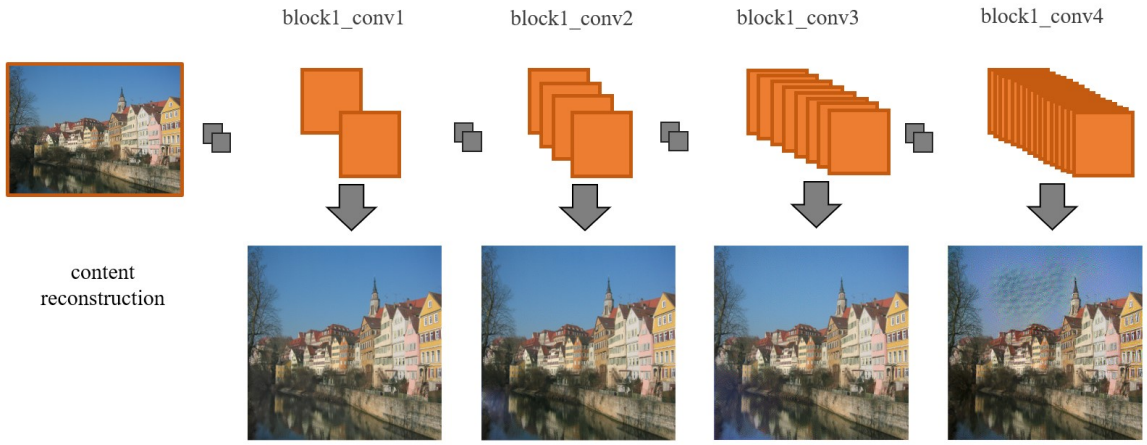

Figure 1: Reconstructions of the image from different content layer depths, without any style interference.

4.2 Style

The style can be seen as the artifacts created by the methods that have been used during the creation of the original style image and therefore as the texture of the image. Therefore the so called texture of the style image is used. Texture is an structuring element and is usually steady over a region/object. To describe the textures of an image a correlation between the different feature maps of one layer of the network can be used (Gatys et al., 2015b). The correlation for multiply layers is computed to retrieve a multi-scale representation of this texture. Again we can reconstruct an image using only these style representations and therefore displaying the style that image has (see Figure 2).

4.3 Training the Image

The Image starts out as either a random, white-noise image or the original content image. The image is trained to minimize the difference between the style and the content based on the two style/content images. To do so a pretrained CNN model (VGG19) with its weights fixed is used.

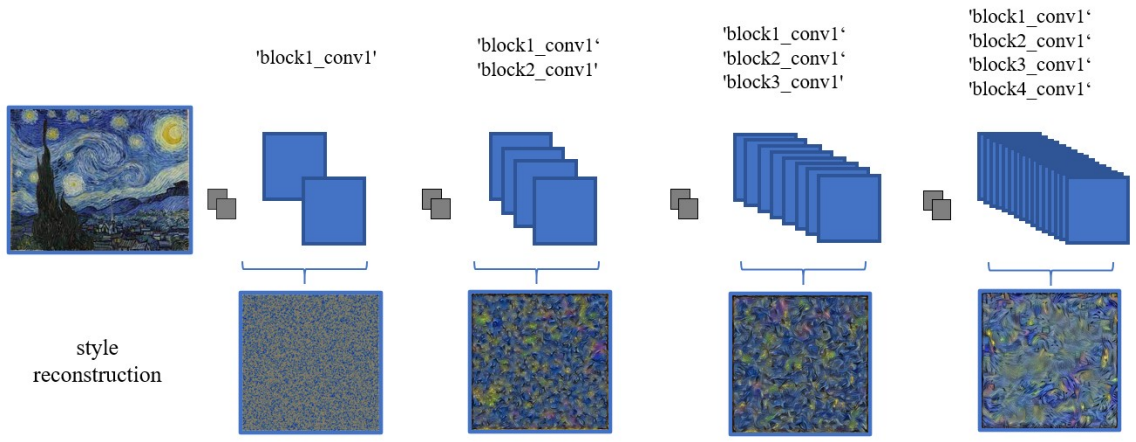

Figure 2: Reconstructions of the image from different amounts of style layers, without any content.

5 Training Environment

5.1 Preprocessing

Since the VGG19 Model has been trained with switched color channels, the color channels of the image are switched from RGB to BGR as the first preprocessing step. During the second preprocessing step the images are normalized. Because VGG19 is trained on the imagenet database, we applied the same VGG19 preprocessing steps and substract the imagenet means. At last the images are being resized to a quadratic shape, to allow for better processing. To display the images in a normal color-scheme, a reversing function "reverse_preprocess" is implemented. This function reverses most of the preprocessing steps (excluding resizing). The preprocessing and reverse processing can be seen in Figure 3.

5.2 The Model

The model is based on the VGG19 model that is implemented in Keras (TensorFlow Documentation, 2021). As suggested by (Gatys et al., 2015a) the max-pooling layers of this model has been replaced with average-pooling. A further examination of the differences can be seen in 6.2. To access the output of the intermediate layers an output dictionary has been implemented where the outputs of these layers are stored to be retrieved on demand.

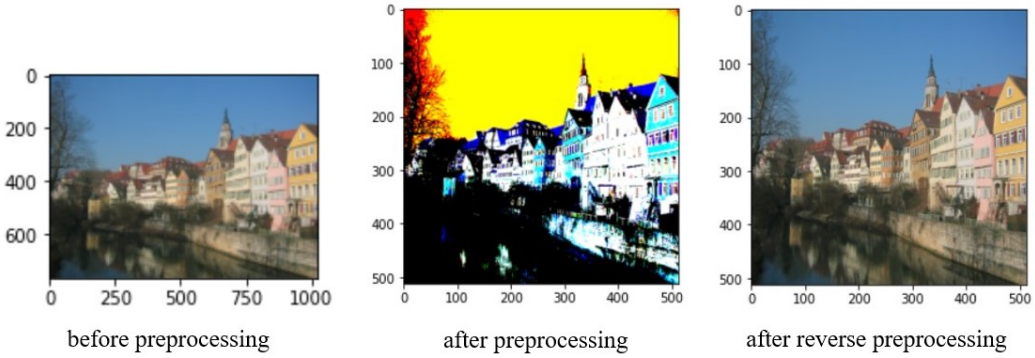

Figure 3: The pictures at different stages of the preprocessing.

5.3 Loss

5.3.1 Content Loss

As mentioned in 4.1, the content of an image can be defined by the feature maps from the intermediate layers of an image recognition model. The model that we used was VGG19 (Simonyan and Zisserman, 2015).

The content loss describes the difference between two pictures contents. Therefore, the activation (feature maps) of l different layers of the VGG19-model is used to compare the content of two images $\vec{p}$ (content image) and $\vec{x}$ (reconstructed image). By computing the Mean Squared Error between these feature maps, we can identify how different the pictures are content wise, following this formula by Yuan (2018).

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{i * j} \sum_{ij} (F_{ij}^l - P_{ij}^l)^2$$

with $F^l, P^l \in \mathbb{R}^{N_l \times M_l}$ (N_l is the filter count and M_l the product of height and width) and where F_{ij}^l is the the activation of the i-th filter at position j in layer l of the image $\vec{x}$. P_{ij}^l is the the activation of the i-th filter at position j in layer l of the image $\vec{p}$. We found that taking the mean over the MSE instead produces more appealing results.

5.3.2 Style Loss

In 4.2 the idea has been elaborated, that the correlations between feature maps from the same intermediate layer can be used to identify the style. In order to realize the style representation correlation matrices, also called Gram-Matrices $G^l \in \mathbb{R}^{N_l \times N_l}$, are used. A Gram-Matrix G^l of layer l is defined as:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}, \ for\ all\ filters\ i\ and\ j$$

where G^l_{ij} is the entry of the matrix, describing the correlation between the feature maps of filter i and j in layer l. G^l_{ij} is obtained by summing over all inner products between two feature maps F^l_{ik} and F^l_{jk} over all pixels k of the feature maps. Therefore, the style difference between the image $\vec{a}$ and the image $\vec{x}$ can be identified by comparing the Gram-Matrices G^l_{ij}, based on $\vec{x}$, and A^l_{ij}, based on $\vec{a}$, over all predefined style layers l. The contribution of each layer to the total style-loss can therefore be expressed as the following (Yuan, 2018).

$$E_l = \frac{1}{i * j} \sum_{ij} (G^l_{ij} - A^l_{ij})^2$$

Again here we found that taking the mean instead of plain MSE and removing the scaling term $\frac{1}{4N_l^2 M_l^2}$ (as per the original papaer) produces more appealing results. The total style-loss between a style image $\vec{a}$ and a reconstructed image $\vec{x}$ is the weighted sum of E_l over all style layers l, weighted by the corresponding w_l:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l,$$

5.3.3 Total Loss

The total loss L describes the difference between the reconstructed image $\vec{x}$ and the content-image $\vec{p}$ on a content level, as well as the difference between $\vec{x}$ and the style-image $\vec{a}$ on a style level. Since $\mathcal{L}_{content}$ and $\mathcal{L}_{style}$ should not always be weighted the same, they are weighted by α and β.

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

5.4 Training Loop

To generate an image $\vec{x}$ with the style of image $\vec{a}$ and the content of image $\vec{p}$, we need to minimize the total loss $\mathcal{L}_{total}$. Therefore, the reconstructed image $\vec{x}$, which is either initialized as the content image ($\vec{p}$) or as a random white-noise image, is trained until the loss is sufficiently minimized to see appealing results.

As for optimizers, Gatys et al. (2015a) used the L-BFGS-Optimizer. Some other implementations like Yuan (2018), Athalye (2015) and Engstrom (2016) used the Adam optimizer. We decided to use the Adam optimizer.

6 Training Choices

During the research and implementation we found, that many other implementations slightly differ from the original paper. To identify differences that led to more appealing results we compared some variations in the reconstructed image initialisation (6.1), the pooling mechanism (6.2) and different amount of style layers (6.3). Additionally, we compared the results with variation in the typical hyper-parameters like the number of training iterations (6.4). It is worth mentioning, that tuning these parameters will have a major impact on the reconstructed image, leading to more or less appealing results. Thus, finding a good set of hyper-parameters turned out to be one of the biggest challenges. These methods were tested on "Neckarfront in Tübingen" (Praefcke, 2003) as content image and (A)"Starry Night" (van Gogh, 1889), (B)"Composition VIII" (Kandinsky, 1923) and (C)"Two Men Contemplating the Moon" (Friedrich, 1820) as style images (See Figure 4) .

6.1 Initialisation of the Reconstructed Image

As mentioned earlier, different initialisation methods for the reconstructed image have been applied. Most common is the usage of the content image as the starting point. Even the original code (Gatys, 2021) provided by the paper´s main author (Gatys et al., 2015a) is using this method, even though it is stated differently in the paper. The other method is the usage of a random white-noise image. This is barely used, but most of the implementations still have the option to switch to this method. We found significantly better reconstructions, when using the content image as initialization instead of a white-noise image. Since we could

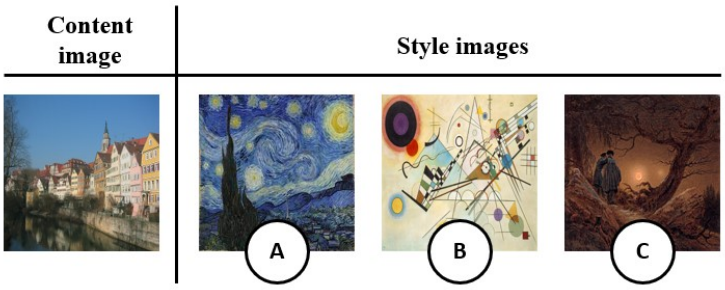

Figure 4: The Images, that were used to test the various implementations on, were "Neckarfront in Tübingen" (Praefcke, 2003) as content image and (A)"Starry Night" (van Gogh, 1889), (B)"Composition VIII" (Kandinsky, 1923) and (C)"Two Men Contemplating the Moon" (Friedrich, 1820) as style images.

not find a suitable learning rate which led to good results for both methods, we decided to follow Yuan (2018) and chose the content image initialisation approach.

6.2 Average-Pooling instead of Max-Pooling

Gatys et al. (2015a) suggest using average-pooling instead of max-pooling, to achieve more appealing results. The result of the two methods are shown in Figure 5. Since edges always are the sudden change of values in an image, the use of average-pooling instead of max-pooling leads to fewer edges. The remaining ones also end up being smoother. This can be seen especially well in the results of (B).

6.3 Amount of Style Layers

While feature maps of earlier layers describe features that are closer to the pixel basis, like color, the features are getting more complex in deeper layers. The reconstruction using only the first layer ("block1_conv1") is basically a color-shift. Using the two style layers ("block1_conv1","block2_conv1") influences the shape of the edges. Using up to three layers result in a loss of detailed information, that can not be depicted with this art style. This reduction of details is increased by every layer that is added. Adding the fifth layer does not make a big difference, which is why we would suggest leaving it out. The result of the different methods are shown in figure 6.

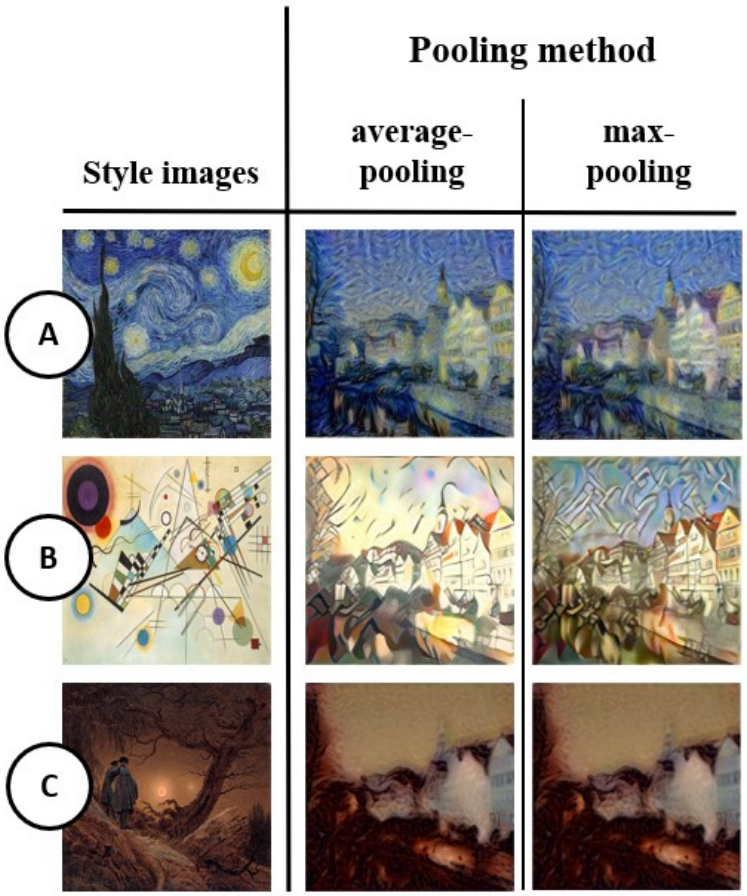

Figure 5: Different methods of pooling. On the left column the three used style images are displayed. In the middle column the results of applying average pooling is shown, while the last column presents the results when using max-pooling. The content image was Praefcke (2003).

6.4 Number of Iterations

During the training process every iteration helps to generate better results. While the changes between two reconstructions in early iterations are bigger than in later iterations, they still contribute to the transferred style. To identify which number of iterations is optimal, images from the iterations 50, 100, 250, 500 and 1000 have been generated, while keeping the other parameters (learning rate, amount of style layers, loss functions, initialisation method) fixed (see Fig.7). We suggest no more than 500 Iterations, since changes made after that are only minor ones.

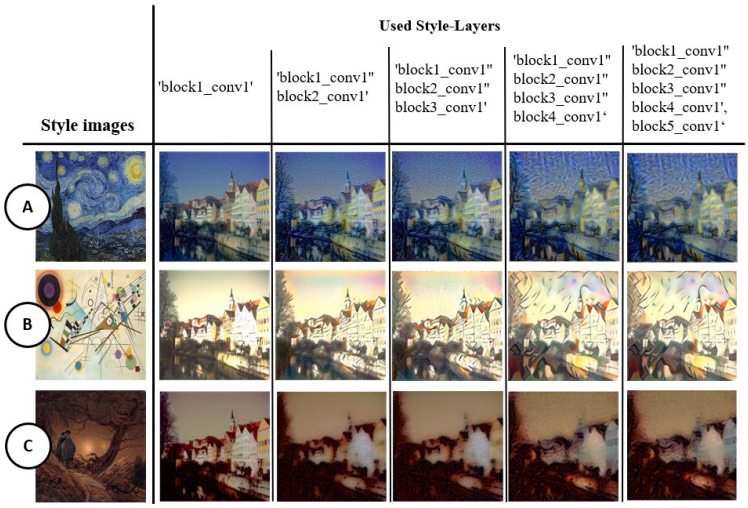

Figure 6: Different amounts of style layers.

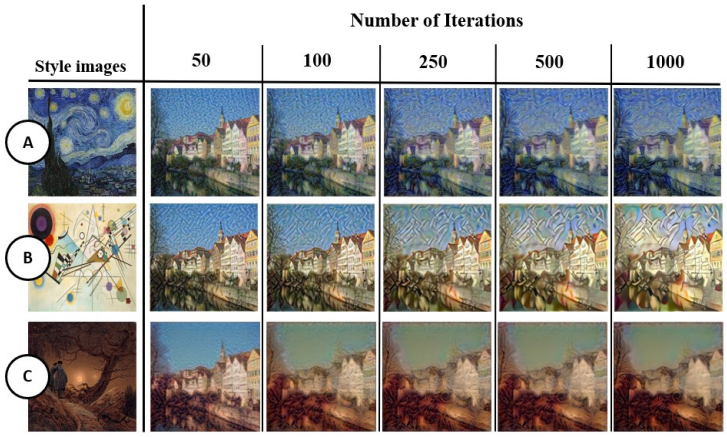

Figure 7: Different numbers of iterations

7 Conclusion

To conclude: the re-implementation worked as planned. The model differentiates between style and content and applies a style of an image to another image. We identified differences in the results of various training-parameters. We have also shown, that the results are highly dependent from these training-parameters, which themselves are interdependent from another and therefore might differ from implementation to implementation. Further refinements could be done by fine-tuning the parameter-combinations, as well as an evaluation study conducted with human raters.

Acknowledgements

We want to thank Leon Gatys for his answers to some of our questions and the reference link to the original code project.

References

Athalye, A. (2015). Neural style.

Engstrom, L. (2016). Fast style transfer.

Friedrich, C. D. (1819-1820). Two men contemplating the moon.

Gatys, L. A. (13.03.2021). leongatys/pytorchneuralstyletransfer.

Gatys, L. A., A. S. Ecker, and M. Bethge (2015a). A neural algorithm of artistic style.

Gatys, L. A., A. S. Ecker, and M. Bethge (2015b). Texture synthesis using convolutional neural networks.

Google Inc. (2019). Stadia gdc 2019 gaming announcement.

Instapainting (07.03.2021). Ai-painter by instapainting.

Joan and E. Olson (07.03.2021). Lasagne/recipes.

Kandinsky, W. (1923). Composition viii.

Praefcke, A. (February 2003). Tuebingen neckarfront.

Ruder, M., A. Dosovitskiy, and T. Brox (2016). Artistic style transfer for videos. *German Conference on Pattern Recognition (GCPR), LNCS 9796*(3), 26–36.

Simonyan, K. and A. Zisserman (2015). Very deep convolutional networks for large-scale image recognition.

TensorFlow (26.02.2021). Künstlerische stilübertragung mit tensorflow lite.

TensorFlow Documentation (22.01.2021). tf.keras.applications.vgg19 — tensorflow core v2.4.1.

van Gogh, V. (1889). Starry night.

Yuan, R. (3.8.2018). Neural style transfer: Creating art with deep learning using tf.keras and eager execution. *Medium.com*.