Image Style Transfer

Sarah Berenji, Jonathan Guichard, Dennis Malmgren

Kungliga Tekniska Högskolan

Abstract. The fine arts, especially painting, are still largely considered to be unique to mankind, to be a skill out of the reach of machines and algorithms which are simply incapable of grasping the complexity of such masterpieces. In this work, we explore and implement a neural algorithm capable of rendering images with different styles, as if machines had finally learnt how to paint. We compare several similar techniques in an effort of showing what parameters affect the final results and how, and what the limitations of those techniques are. We find that we are able to create very convincing and eye-pleasing results with a fairly simple approach. Moreover, and according to our experiments, this approach appears to be very general as it can be implemented with a variety of architectures.

Keywords: Deep Learning, Convolutional Networks, Style Transfer

1 Introduction

The goal of style transfer is to create a new image with the content of one picture but with the style of another one. For example, style transfer can be used to transform a picture of our choice into an imitation of a master's work, as shown by figure 1.

This style transfer problem can elegantly be formulated as an optimization problem for which we want to find an image $\boldsymbol{x}$ such that :

$$\boldsymbol{x}* = \underset{\boldsymbol{x}}{\operatorname{argmin}}(\mathcal{L}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{c})) \tag{1}$$

$$\mathcal{L}_m = \sum O^T * M_l * O \tag{2}$$

With $\boldsymbol{s}$ representing the style image and $\boldsymbol{c}$ the content image. The loss function $\mathcal{L}$ represents the unfaithfulness of the generated image $\boldsymbol{x}$ to the style of $\boldsymbol{s}$ and to the content of $\boldsymbol{c}$.
The exact definition of the loss function is approach-dependent, as is the definition of content and style.

In this paper we try to faithfully reproduce current solutions to this problem and explore hyperparameter tuning in a comparative study.

Fig. 1. Example of style transfer. Stockholm as (possibly) seen by Picasso. The top left image is called *content image*, the bottom left *style image*. The result of the style transfer is shown on the right and was generated with our implementation.

2 Background

Some of the first approaches proposed to tackle this problem rely on non-parametric, non machine learning algorithms [1] [2]. Those techniques were initially conceived to expand images outwards based on a seed image, which seems to suggest that they are able to capture some notions of content and style in images.
When applied to style transfer [1], these algorithms are capable of producing interesting and convincing results, but are however limited in the fact that they can only pick up low-level style features, i.e. textures. It is for example possible to transfer the texture of a lemon onto a banana, however performing style transfer as shown in figure 1, *i.e.* being able to completely "repaint" an image, is not.

An other non-machine learning technique proposed in 2005 [3] has been able to successfully solve a restricted version of this problem, namely color style transfer. The goal of color style transfer is to only transfer the color atmosphere of a picture onto a second one, leaving out any features such as recurring geometrical forms or painting techniques. Problems that need to be tackled include defining what a style is, as well as how to implement a successful blend of the style and content.

The first work to solve this with deep learning in a way that garnered wide attention was proposed by Gatys et al. [4] in 2015, and relies on Convolutional Neural Networks.
In [4], the loss function $\mathcal{L}$ is defined as a linear combination of a style loss function and a content loss function, representing respectively the unfaithfulness of the

result image to the style of the style image, and the unfaithfulness of the result image to the content of the content image. Those style and content functions are based on the feature representation produced by the network when processing (as if we wanted to classify) the content, style and result images. The result image, which can initially be some noise, is then updated by using gradient descent. This new result image is then reprocessed by the network, in order to compute the new gradient update step, and so on and so forth.
This method is able to achieve very eye-pleasing results, as showcased in the paper.

As presented above, Gatys' algorithm [4] yields very interesting and eye-pleasing results when performing style transfer with paintings. This is however not the case if we use photographs as both style and content images, since the result will exhibit some painting-like distortions that shouldn't be present in a photograph. This problem, known as *photorealistic style transfer*, has only very recently been tackled by Luan et al. [5] using Deep Learning techniques. Since we only want to perform style transfer mainly in the color space, this problem can be seen as an instance of the color style transfer problem presented previously.
In order to achieve this, the authors introduce a photorealism regularization term in the total loss function. An optional guidance to the style transfer process is also introduced to improve photorealism. This guidance is based on image segmentation, the goal being that the transfer should respect the semantics of the images (*e.g.* buildings should only be styled based on the appearance of buildings in the style image).

Other approaches have not mainly improved on the *quality* of the final results but on the computational performance of the algorithm [6] [7] [8], improving it by orders of magnitude. The primary approach is to train a feed-forward convolutional neural network on producing the target style as output and combining this with any content image, in essence reducing the problem to one of an image transformation task. This implies however that a new network has to be trained for every style we wish to imitate, displacing the burden to the learning step. The end results are comparable but not entirely of the same quality. As presented in [7], this approach seems however to give better results when the style image is a texture.

At a higher abstraction level, blending invariant extracted *features* from one training set onto the output of another algorithm can be applied to other problem domains. One recent paper explores this in the context of chess [9], where a network is trained to play *like* a given grandmaster, using the style transfer algorithm proposed in [4] and GANs.

3 Approach

3.1 Theory

We decided to first follow the method detailed in Gatys et al. [4] for the implementation of our project, as it is a block on which many other papers build. As outlined above, this approach is based on convolutional neural networks, and more precisely on a VGG19 network already trained on ImageNet.

In the Gatys paper [4], the total loss function $\mathcal{L}$ we want to minimize is defined as:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{c}) = \alpha \mathcal{L}_{style}(\boldsymbol{x}, \boldsymbol{s}) + \beta \mathcal{L}_{content}(\boldsymbol{x}, \boldsymbol{c}) \tag{3}$$

With α and β being scalars allowing us to adjust the importance of the style/content when generating the final result.
The content loss function is then defined as below, with F^l and P^l being the feature representation of x and c respectively, in a layer l of our choice.

$$\mathcal{L}_{content}(\boldsymbol{x}, \boldsymbol{c}) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 \tag{4}$$

The style loss is calculated as follows, with G^l and A^l being the style representation of x and s respectively, in layer l. The style representation of an image in layer l is defined to be the gram matrix G^l, which is the inner product of the vectorised feature representation in the layer, i.e. $G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$. N is the number of channels of the input image, and M is the product of the height and the width of the image. Finally, w_l is a scalar of our choice, acting as a weight for the layers.

$$\mathcal{L}_{style}(\boldsymbol{x}, \boldsymbol{s}) = \sum_{l=0}^{L} w_l E_l \tag{5}$$

$$E_l = \frac{1}{4N^2M^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \tag{6}$$

Johnson et al. [8] provide a method to significantly speed up the initial style transfer algorithm. This speedup is however out of the scope of our project, but we can nonetheless make use of some other improvements introduced in the paper.
One of those improvements is the addition of a total variation component to the total loss function $\mathcal{L}_{total}$ in order to encourage spatial smoothness. The loss function used in our implementation now becomes:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{c}) = \alpha \mathcal{L}_{style}(\boldsymbol{x}, \boldsymbol{s}) + \beta \mathcal{L}_{content}(\boldsymbol{x}, \boldsymbol{c}) + \gamma \mathcal{L}_{variation}(\boldsymbol{x}) \tag{7}$$

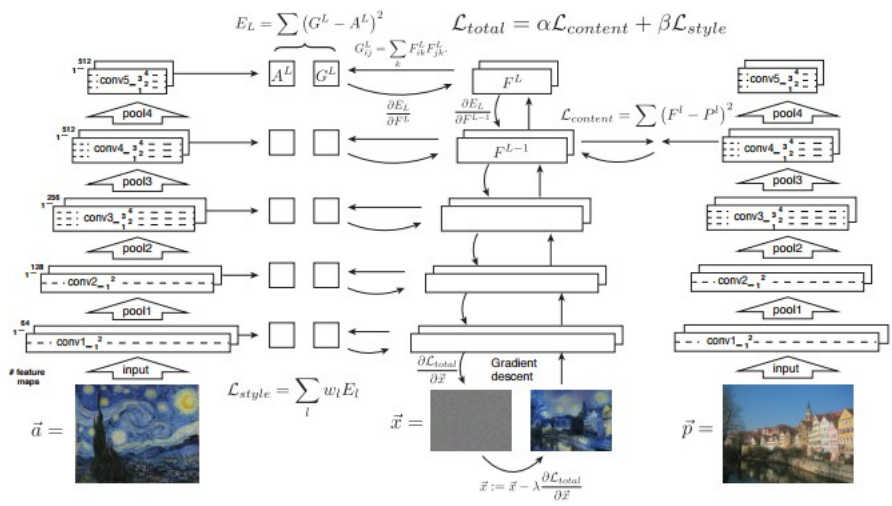

Fig. 2. The style transfer algorithm visualized. The style, content and result images are forwarded into the CNN. The loss $\mathcal{L}_{total}$, which is computed based on the feature representation of those images, is minimized by using gradient descent on the result image. Image from [4].

Moreover, Johnson et al. use the VGG16 network instead of VGG19. Even if their approach differs significantly from the one used in [4], we think that using a shallower network is still likely to provide an improvement performance-wise, for hopefully the same qualitative results.

One (optional) adjustment implemented in [4] is to capture the output of each filter across the validation set of ILSVRC2012 for every position and every image and normalize the mean output per filter to 1 through scaling the weights. That is, for each filter in each layer $F_{l,i}$ we produce a scaling factor $k_{l,i}$ such that

$$\frac{1}{N} \sum_p max(0, k_{l,i}(P_p * F_{l,i})) = 1 \tag{8}$$

where P is the pixel window we convolve with, p is taken to run across every position in every image, l is the layer and i is the index for the filter in that particular layer. The total number of pixels is N. We reproduced this normalization and compared our resulting weights with the publically available weights in [4]. We also evaluated the effect of using these normalized weights on the style transfer function.

3.2 Implementation details

In the following, we refer to layers in the VGG16 and VGG19 networks with the notation `convX_Y`, with X referring to a block[1] number, and Y referring to a convolutional layer within that block, *after* applying the ReLu function[2]. `X` and `Y` start at 1 and increase the higher the position in the layer. See figure 2 for more details.

Our implementation is based on Python, Tensorflow and Keras. The algorithm used for optimizing the loss function is L-BFGS-B, which is supposed to have good performances for gradient descent on images. The result image can be initialized with either white noise, the content or the style image.

Based on [4] and [8], we have identified three different standard configurations, detailed below, that we used in our implementation. The VGG16 and VGG19 networks we use have been pretrained on the ImageNet dataset.

GatysMax For this configuration, based on [4], we use the VGG19 network with max pooling. The layer used for the content loss function is `conv4_2`; The layers used for the style loss function are `conv1_1`, `conv2_1`, `conv3_1`, `conv4_1` and `conv5_1`, with $w_l = 1/5$ for all those layers.
We use this configuration as reference for our experiments, being the quickest to implement while remaining close to the original paper.

GatysAvg Same configuration as *GatysMax*, but we use the VGG19 network with average pooling.
This configuration requires us to manually define the VGG19 model as opposed to loading it from the Keras standard library. This approach is closer to the original paper.

Johnson This configuration is based on [8], and uses the VGG16 network with max pooling. The layer used for the content loss function is `conv2_2`; The layers used for the style representation are `conv1_2`, `conv2_2`, `conv3_3`, and `conv4_3`, with $w_l = 1/4$ for those layers.

[1] A block is a set of successive convolutional and ReLu operations delimited by a pooling operation.

[2] Gatys et al. [4] are unclear whether they read the feature representations of a given layer before or after applying the ReLu function on that layer. We therefore follow Johnson et al. [8] for our implementation.

4 Results and Conclusion

In the following experiments, and unless otherwise indicated, all the generated images are of size 512x512 pixels. The style and content images are resized to the size of the result picture before being processed by the network. The weights and weight ratios specified in the following are for images of size 512x512 pixels. The result images where initialized with white noise unless otherwise mentioned. The style and content images used for our experiments are shown in figures 3 and 4.

Fig. 3. The two content images we used for our experiments, Stockholm (left) and Tübingen (right).

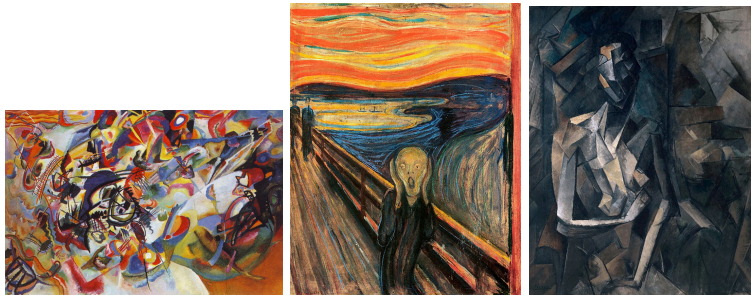

Fig. 4. The three style images we used for our experiments. *Composition VII* by Kandinksy (left), *Der Schrei* by Munk (center), *Femme Nue Assise* by Picasso (right).

4.1 Tuning the loss function

Figure 5 showcases the results yielded for different content/style ratios. As expected, a higher ratio will create images that tend to clearly preserve the structure of the content image, but at the cost of a less noticeable style. Lower ratios will however yield opposit results, with result images tending to poorly

Fig. 5. The impact of the content/style ratio on the final result. Images generated without any regularization, using the GatysMax configuration, with *Der Schrei* as style and *Tübingen* as content. The β/α ratios are 5×10^{-2} (left), 1×10^{-2}, 5×10^{-3} and 1×10^{-3} (right).

preserve the structure of the content image, but with a style closer to the reference.
Interestingly, we can see that the color palette is very similar for all those images, suggesting that colors can be transferred with little impact on the semantic faithfulness of the result images. This therefore seems to justify the design of the content and style loss functions, since they seem able to efficiently pick up and isolate those two concepts.

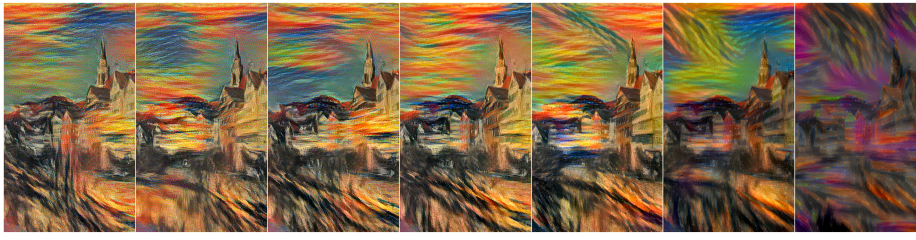

Fig. 6. The impact of regularization on the final result. Images generated with $\alpha = 1000$, $\beta = 5$, using the GatysMax configuration, with *Der Schrei* as style and *Tübingen* as content. The regularization weight varies from 0 (left) to 10000 (right), with intermediate values 0.1, 1, 10, 100 and 1000.

Figure 6 compares the impact of regularization on the generated images. As expected, a higher regularization weight gives us images with significantly less noise, with some very smooth and painting-like results for the higher weights. This regularization term seems to have some sort of impact on the final result, as some features will vary in size and position, but it doesn't seem to have any destructive effect (*e.g.* blur or loss of details) but for very high regularization weights (starting when the style and regularization weights are equal). It thus allows us to generate low noise images for the same amount of iterations, at no

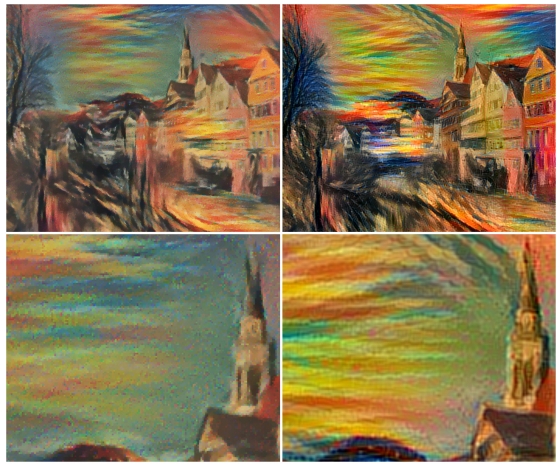

Fig. 7. Postprocessing with total variation denoising (left) vs. total variation term in the total loss function (right). The denoising was performed using the Rudin-Osher-Fatemi algorithm with $\lambda = 29$. The images used are those of figure 6, with a regularization weight of 0 for the left image and 100 for the right one.

visual quality cost.

However the results with over regularization don't look reasonable anymore. The very right photo in figure 6 shows this effect where the regularisation weight is 100000. As it is shown in the pictures, with the over regularisation, both content image and style images colors start to slowly disappear and blend into some sort of "average color".

Finally, and to conclude our discussions regarding denoising, we compare denoising in postprocessing to denoising when performing the style transfer, as suggested in [8] (figure 7). The parameter λ (for the Rudin-Osher-Fatemi algorithm) was chosen visually, so that it minimizes the noise without blurring the image too much. As we can see, the denoising seems to be less effective and blurs the image when applying it in postprocessing, compared to a much crisper result and painting-like artifacts for the righthand image. Overall, we think that denoising in postprocessing yields results of lower quality, which comforts us in our design choice.

Finally, we would like to point out that the weights for each component of the loss function need to be manually adjusted depending on the style and content images used, in order to get the most visually appealing results. Those weights also depend on the size of the generated image, which makes this already slow finetuning process even more tedious. Moreover, there isn't to our knowledge

any simple metric that would be able to rate the visual quality of the generated images, forcing us to rely on visual inspection and subjective appreciations.

4.2 Picking layers

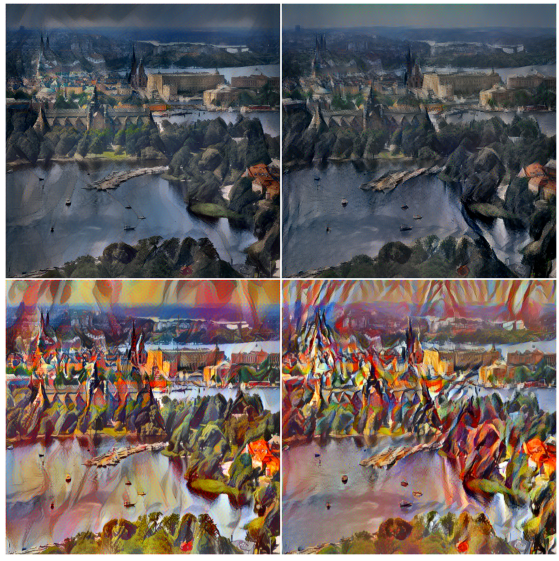

Fig. 8. The impact of the content layer on the final result. Images generated with $\alpha = 1000$, $\beta = 5$ and $\gamma = 10$, using the GatysMax configuration, with *Femme Nue Assise* and *Composition VII* as style, and *Stockholm* as content. The content layers are `conv2_2` (left) and `conv4_2` (right).

Figure 8 compares the effect of using different content layers. As we can see, the images generated with a lower layer tend to better preserve the structure of the content image, with fewer distortions and of lesser magnitude.
This can be explained by the fact that lower layers of a convolutional neural network tend to learn and memorize fine grain features (*e.g.* a straight line), while higher layers will tend to learn and memorize more abstract representations (*e.g.* a door). Therefore, using lower layers for content will tend to limit geometrical distortions in the style transfer process. Indeed, geometrical forms might not be "recognized" anymore by those lower layers if the distortions are too important, even if the abstract meaning of those forms remain the same. In other words, and using our previous example, lower layers will try to preserve straight lines, while for higher layers straight lines will be (to some extent) irrelevant as long as we can still identify something that resembles a door.

Fig. 9. Comparison of different layer weights for the style loss. Images generated with the same parameters as in figure 6 and no regularization, with *GatysAvg*. Style mainly from the lower layers (weights 16/8/4/2/1) on the left, uniform weights in the center, weights mainly from the higher layers (weights 1/2/4/8/16) on the right.

Finally, we tried changing the weights in the style function, in order to either mainly read the style from lower or higher layers. As figure 9 shows, there is no major visual differences regarding style.
The "style perception" of a convolutional network therefore seems to be the same at all layers, which we could interpret as a positive sign regarding the design of the style loss function as style should be perceived similarly whether being at a low "pixel level" or at a higher abstraction one.

4.3 Layer pooling

In [4] the author indicates that the obtained results were more pleasing when replacing the max pool operation in the original VGG19 with average pooling operations. We implemented a VGG19 model with these adjustments and compared the results.

Fig. 10. Comparison of the results obtained when using GatysAvg (left), GatysMax (right) with same parameters as in figure 6 and no regularization, with *Der Schrei* as style, and *Tübingen* as content.

In this individual sample the structure of the content image is better preserved while still transfering style adequately, and we agree with the notion that the average pooling resulted in a slightly more pleasing image.

4.4 Weights normalization

We implemented two versions of networks with normalized weights. One based on the weights from [4] and one where we extracted the activation values ourselves from the 50k images in ILSVRC2012 validation set and normalized the mean activation values based on that. We then ran samples with these two normalized weight sets and compared them to the default VGG19 weights trained on imagenet.

Fig. 11. Same parameters as in figure 6 and no regularization (left), our normalized weights (center), gatys normalized weights (right), with *Der Schrei* as style, and *Tübingen* as content.

Unfortunately we are unable to reproduce the claims in [4] that using normalized weights leads to the same end result. It is our conjecture that normalized weights are intended to have the same effect as mentioned in [10] and speed up convergence. If we compare the scale factors obtained through our measurements with those in the weights obtained in [4], we find the following, sampling from filters in the first layer, a mid layer and final layer.

Dataset	Ratio Conv1 1	Ratio Conv3 4	Ratio Conv5 4
VGG19	1	1	1
Gatys	0.0501809	0.00099329	0.90801466
Ours	0.05650292	0.33963752	3.52301192

We conclude that the activations calculated by Gatys are implemented in another way for subsequent layers, but the initial layer is likely implemented similarly, with variations ascribable to data handling or manipulation factors.

4.5 Comparing approaches

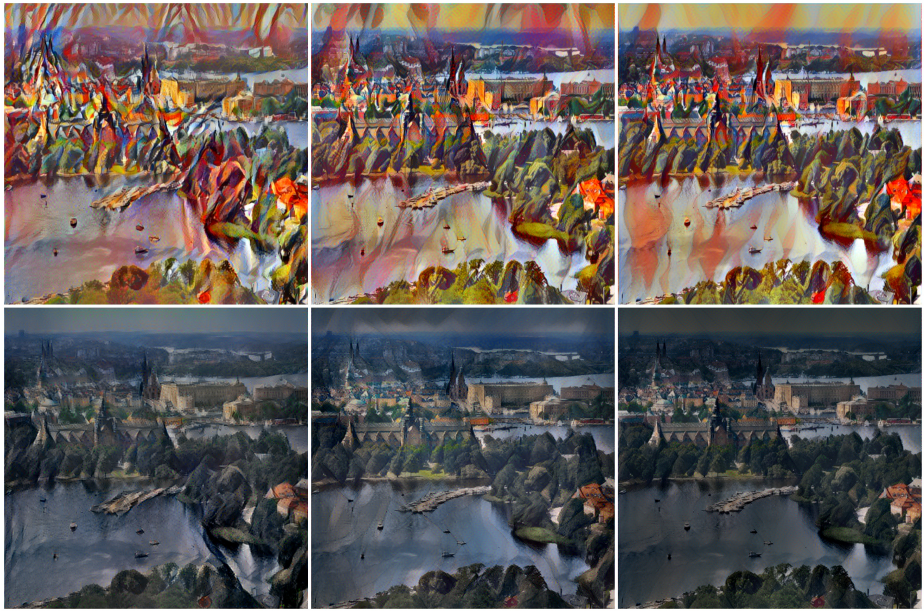

Fig. 12. Comparison of the results obtained when using GatysMax (left), GatysMax with `conv2_2` as content layer (center), and Johnson (right), with *Femme Nue Assise* and *Composition VII* as style, and *Stockholm* as content.

Figure 12 showcases the results yielded by three different configurations of the style transfer algorithm. As we can see, the results are very similar to each other, and of the same visual quality. The images on the right and center of this figure were generated with a VGG19 network and took approximately 2h25 each, while the images on the right were obtained with VGG16 and took approximately 1h35m each.
As we can see, using VGG16 seems to provide a significant speedup over VGG19, for results of the same visual quality. This experiment therefore seems to justify the choice of using a shallower network as introduced in [8], as we attribute this difference in performances to the depth of the network.

4.6 Photo style transfer

As mentioned before, Luan et al. [5] introduced an approach to have a photorealistic style transfer in which they try to suppress distortions in order to generate a photorealistic result.
As one of our experiments, we tried to compare the results of *GatysMax* with

this photorealistic style transfer. Figure 13 shows this comparison, based on two images found in [5]. As expected, using Gatys results in an image that has distortions and looks more like a painting rather than a real photo, since we do not force the result image to retain original shapes and geometrical features.

Fig. 13. Comparison of the result obtained by Luan et al. in [5] (center left) with our implementation, based on GatysMax (center right). The style photo used is top left, content bottom left.

Our implementation of the deep photo regularization needs additional work to find the correct weight scaling. The algorithm itself suffered from numerical instability and losses could end up negative if photo regularization weight was pushed too high.

Fig. 14. Our deep photo regularization implementation, based on a regularization weight of 10000, initialized with noise image.

4.7 Final Conclusions

As we could see with our experiments, the style transfer algorithm originally proposed by Gatys et al. works for a variety of configurations. We can indeed use different convolutional networks, different pooling functions, build the style and loss functions from different layers, add some new terms to the loss function, and still be able to generate very convincing and eye pleasing results. The

visual quality of the images produced comes however at the expense of computational complexity, as it usually takes hours to generate medium sized images on powerful machines, preventing any interactive use.

Being able to use networks that were originally designed for completing a completely different task, namely image classification, is fascinating and gives us a glimpse of the variety of applications deep learning and convolutional networks can be used for.
However, we could argue that image classification and style transfer are perhaps not completely unrelated tasks. Indeed, when performing image classification, the network needs to learn how to identify the same concept in many different images, regardless of variations such as different colors, sizes, and forms. In a sense, we could argue that convolutional networks learn how to separate style from content when training them, and that all we need to do is to extract this information from the network. We believe this to be the main reason behind the success of this technique when compared to the limited results of other non-machine learning approaches.

References

1. Efros, A., Freeman, W.T.: Image Quilting for Texture Synthesis and Transfer. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. (2001) 341–346
2. Efros, A., Leung, T.K.: Texture Synthesis by Nonparametric Sampling. In: The Proceedings of the Seventh IEEE International Conference on Computer Vision. Volume 2. (1999) 1033–1038
3. Neumann, L., Neumann, A.: Color Style Transfer Techniques using Hue, Lightness and Saturation Histogram Matching. In: Computational Aesthetics in Graphics, Visualization and Imaging. (2005)
4. Gatys, L., , Ecker, A., Bethge, M.: Image Style Transfer Using Convolutional Neural Networks (2015)
5. Luan, F., , Paris, S., , Shechtman, E., Bala, K.: Deep Photo Style Transfer. https://arxiv.org/pdf/1703.07511.pdf (March 2017) Accessed: 2017-04-17.
6. Dumoulin, V., Shlens, J., Kudlur, M.: A learned representation for artistic style. CoRR **abs/1610.07629** (2016)
7. Ulyanov, D., Lebedev, V., Vedaldi, A., Lempitsky, V.S.: Texture networks: Feed-forward synthesis of textures and stylized images. CoRR **abs/1603.03417** (2016)
8. Johnson, J., , Alahi, A., Fei-Fei, L.: Perceptual Losses for Real-Time Style Transfer and Super-Resolution (March 2016)
9. Chidambaram, M., Qi, Y.: Style transfer generative adversarial networks: Learning to play chess differently. CoRR **abs/1702.06762** (2017)
10. Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. CoRR **abs/1602.07868** (2016)

A Examples of Style Transfer

The following results were generated using the *Johnson* configuration, whith $\alpha = 5$, $\beta = 0.05$ and regularization is 0.1.

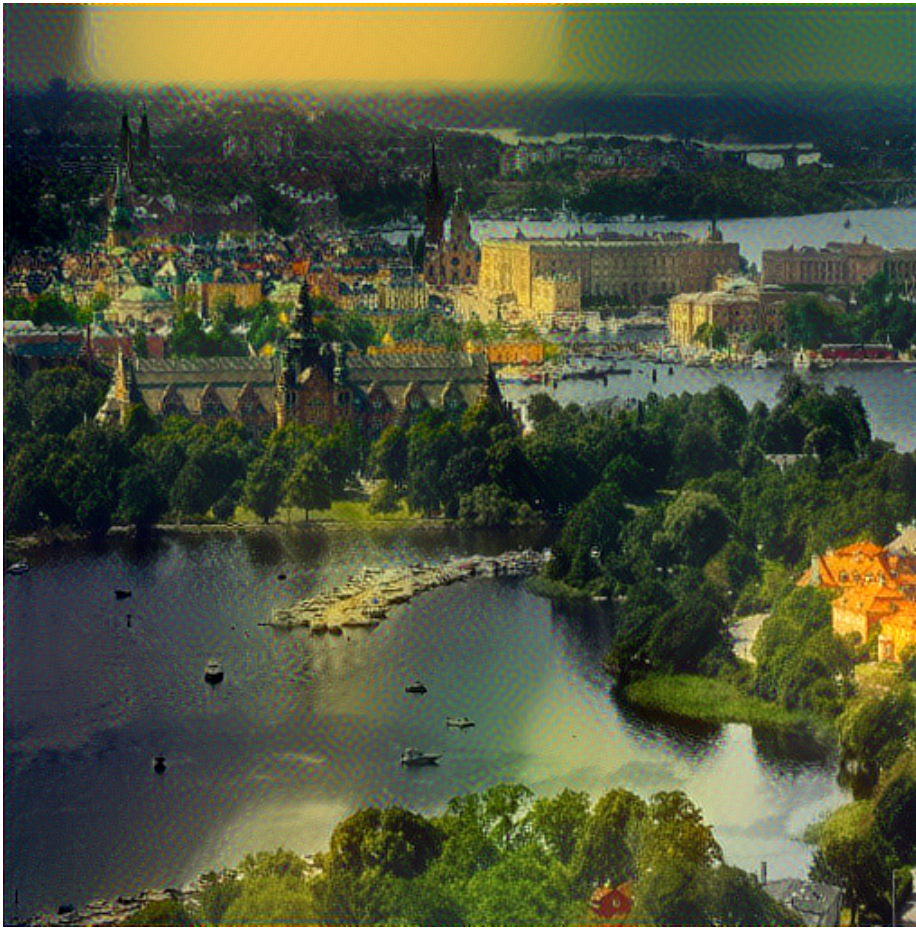

Fig. 15. *Stockholm* with *The shipwreck of the Minotaur* by J. M. W. Turner.

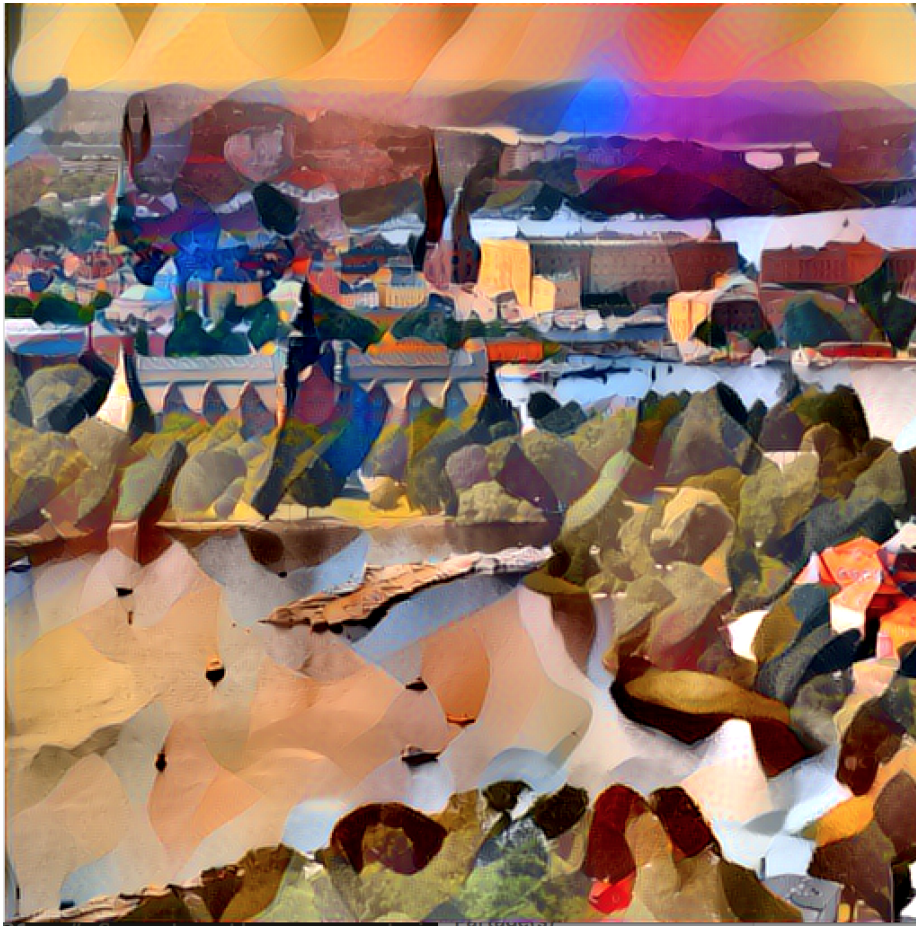

Fig. 16. *Stockholm* with *Edtaonisl (Clergyman)* by Francis Picabia.

Fig. 17. *Stockholm* with *The Starry Night* by Vincent van Gogh.

Fig. 18. *Stockholm* with *Der Schrei* by Edvard Munch.

Fig. 19. *Stockholm* with *Shot Marilyns* by Andy Warhol.

Fig. 20. *Stockholm* with *Orange Smoothness*, by Leonid Afremov.