Artistic Style Transfer with Color Preservation - A reimplementation

Yesid Cano Castro, Niklas Heidemann, John Jairo Madrid Carvajal

March 2021

1 Introduction

The quest to understand how the human mind creates and perceives the world still remains open. The skills and the time painters and artists put into their creations is undeniable and astonishing, so are the master pieces they come up with. Nowadays, with the aid of artificial intelligence and certain algorithms we can have some hints on how it is possible to artificially produce visually appealing images by separating the content and style representations of two images. Gatys' *et al.* [4] algorithm for neural style transfer (NST) is one such pioneers. Due to its success, NST has lead to an ever increasing popularity and has extended to even reach other fields beyond the arts such as sports and entertainment and most of the current efforts concentrate on minimizing style and content losses as well as time performance of the network [11]. Quite unexpected and rather controversial is the fact that artificially generated art has become trend and is even worth thousands of dollars [2] [8].

Artistic Style Transfer (AST) is the challenge to draw the *content* of an image in the *style* of another image. To succeed in this task the artificial system must be able to separate the content and style representations from both: the content (C) and the style image (S). The newly created artistic image consists of a synthesis of both representations. Exemplary use cases range from transforming the picture of a house front into an impressionist painting to turning a selfie into a portrait, tasks that would take a human artist a long time to achieve. Most recent AST approaches use pre-trained image recognition models, more specifically Deep Convolutional Neural Networks, to extract the feature representations which define content and style of images as used in the losses during optimization. Instead of training a model to learn features, they train the image itself that is to be generated, by using the pre-trained model to determine whether content and style of the respective source images are well represented in the generated image. In this project we replicated one of the major studies that first did AST and combined it with newer approaches [4]. Additionally, we implemented two different techniques to address the problem of undesired color output [7][3], both allows us to maintain the original colors of the content image

- which is desired in some cases. Finally, we explored using different training parameters in order to assess the behaviours and implications in the network's output image.

2 Background and Related Work

It was not only until 2015 that NST came and outperformed any other approach to combine style and content of two images. Previous work from the 1980's known as 'image stylization' had achieved stylized images for cartoon-like effects on images through low level image processing effects [14]. By 2001, Hertzmann *et al.* [6] had developed a texture transfer algorithm called 'image analogies' which allowed them to create a new stylized image (B') given both a photograph (A) and a hand-drawn illustration of that photograph (A'). The image analogies algorithm worked by completing the analogy A:A'::B:B'.
Today's main approaches on AST started with a paper by Gatys *et al.* [4] whose findings we attempt to replicate in this project. Based on that work from 2015 other works went further. One of these additions, also by Gatys *et al.* [3], trying to achieve color preservation, is also implemented in this project. Other approaches focused on transferring the style of whole videos[9][10], or on faster image generation by using an off-line learning strategy instead of the on-line strategy used by Gatys *et al.* [13]. Gatys *et al.* [4] key findings involved the possibility of "decoupling" the content and the style representations of an image in a CNN making use of an already pre-trained model. This, as mentioned above, opened up whole new possibilities in other fields and set the stage for more modern and exciting takes on 'image stalyzation'. Ever since, we can only expect improvement after improvement, it was not ever this easy to create a master piece of art by simply taking two images and feeding them into a neural algorithm. Now this algorithms have been extended to work with images, videos, text, audio and so on and at ever increasingly faster times.

2.1 VGG19

Similar to Gatys *et al.* approach on AST, for our reimplementation we also turned to the pre-trained and publicly accessible VGG19-Network [12], a deep convolutional neural network with 19 weighted layers (plus several pooling layers). The last three layers are dense fully connected layers only relevant for classification tasks. Briefly summarized, VGG19 has 144 million parameters, trained with 1.3 million images of 1000 different categories, which guarantees that the model was exposed to and having learned the most important features of a wide variety of images. As the training data was augmented, the model should have learned to tolerate image variation and learned instead only core features, in the best case semantically relevant ones.

2.2 AST by Gatys *et al.*

Gatys *et al.* [4] took advantage of the capacity of the VGG19-Network to rival human level performance on object recognition and successfully developed and implemented an algorithm capable of AST by separating both content and style representations of a given set of images. To achieve this, they defined their respective target image as the image with the same content representation as the content source image and the same style representation as the style source image, where the content representation is defined as the activation of the ninth convolutional layer of VGG when doing a forward pass on the respective image. As style representation they used a gram matrix based on the activation of five different layers of VGG. While $L_{content}$ is simply defined as the squared error loss between the representations, L_{style} is the weighted mean-squared error of the respective gram matrices.

$$L_{style}(\vec{a}, \vec{x}) = \sum_{l=1}^{L} (w_l * \frac{1}{4N_l^2 M_l^2} \sum (G_{ij}^l - {A_{ij}^l}^2) \tag{1}$$

where w_l is the weighting of the respective layer, $\vec{a}$ and $\vec{x}$ the input images and G^l and A^j the inner products of the feature maps of the respective layers for the input images (i.e. the gram matrices).
Both losses are then combined to:

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha * L_{content}(\vec{p}, \vec{x}) + \beta * L_{style}(\vec{a}, \vec{x}) \tag{2}$$

The authors determined $1e-3$ to be a good ratio for $\frac{\alpha}{\beta}$, but pointed out that it varies for different image pairs.
The training procedure then consisted of computing the style and content representations for the respective input/source images and then to perform Gradient Descent on a white-noise image by computing in each epoch the above loss with the help of a forward-pass of VGG19.
Although there is a trade-off between content and style matching directly implied by the formula of the final loss, Gatys *et al.* achieved good results for transferring styles of art works. They did not preserve photorealism though, when using styles of such photos.

3 Model and Training

In general this work sticks close to the original papers. However we made slight deviations at certain places (mentioned below), if we noticed that they produce better results. Our architecture supports the simultaneous execution (it is also parallelisable, though we did not implement that due to time constraints) of multiple trainings with different parameters, easing the selection of the best image by regular plotting.

Model and training procedure are implemented with Tensorflow. We optimized our hyperparameters and found out that Adam produced the best results,

though we changed it's parameters. We did not implement additional regularization techniques, as we did not train a model and in the special case of training an image, there can be no overfitting, as there is no need for generalization.

Similarly we were restricted in preprocessing our images, as they had to be in the correct shape to be an input for the VGG-model and were also the output for the algorithm, prohibiting any non-reversible measures that would reduce the image quality. The implementation of color preservation methods however can be seen as image preprocessing/enhancement, as it happened outside of the training routine to achieve better results.

3.1 Model

At first we imported the pre-trained VGG19 model without its last three layers, which are only relevant for classification tasks. Based on it we created our own model, which wraps the VGG19 model by returning the feature maps corresponding to the following layers: 'block1 conv1','block2 conv1', 'block3 conv1', 'block4 conv1', 'block5 conv1'. The content representation was extracted from 'block4 conv1'. To obtain the style representations we utilized all of the layers above mentioned.

While the content loss can be easily computed by MSE, for the style loss we had to use the function 'tf.linalg.einsum' to first construct the gram matrices (five for each image). Also we made it possible to weight the different layers, but settled with the same default as the paper, 0.2 for all five. We did not replicated the paper in this exactly but modified the computation of the matrices slightly by normalizing in regard to the shape of the matrix, getting an improved performance with our choice of hyperparameters.

3.2 Training

As mentioned, training for AST does not mean to train a neural network but instead training the result image itself. Aside from this difference though, the training procedures are very similar. In epoch zero we started with the content image, as this produced faster better results than starting with random noise and a little better results than starting from the style image. In this we deviate from the original paper which gets similar results regardless of starting image, but refuses the bias and loss of variation that comes with not starting from noise. However our implementation also support the other two image initialisation methods.

After the initialisation, a forward pass through the VGG19 model is performed and the layers of interest are extracted to compute content and style representation of the image. Subsequently, the losses were computed by comparing with the content and style representations of the respective source images as described by the paper and Gradient Descent performed on the image. For

(a) Style image

(b) Content image

Figure 1: (a) *Starry night* by Vincent van Gogh, the style source image we used for hyperparameter organization, and (b) the content source image [1].

this we used Adam as optimizer and finetuned the learning rate to 0.01, beta_1 to 0.95 and epsilon to 1e-1. The paper did not specify these parameters, so our choice is another possible deviation, helping to explain the differences in produced images. Other optimizers like RMSprop also achieved good results, while Adagrad and default SGD produced worse, at least with the tested configurations.

Note that all hyperparameters, including the optimizer and its learning rate, the initial image and the style-content-weight-ratio, were tested, but not in an extensive search of good combinations. Instead there were usually optimized isolated, due to time and computing power constraints. For the same reason they were only optimized on a single source-image pair (see Figure 1), so to be able to put confidence in our parameters it would be necessary to do a more extensive search with more images.

In comparison to default DL-tasks, AST-training needs less time per epoch but more epochs, our images usually looked good after two to four thousand epochs. The time needed for training an image largely depends on the resolution, as the number of trainable parameters is the same as the number of pixels, which obviously is quadratic in terms of width or height. A 256x256 image needs around five seconds per hundred epochs while a 512x512 image needs fifteen seconds.

3.3 Color preservation

A crucial aspect in the process of implementing AST is color. Visually appealing images are as much the result of a delicate balancing between content and style as they are the outcome of an adequate use of color. Add excessive focus on the content and the output image gets bland, overemphasize the style and the features of the content image lose strength and the output image looks too abstract (see section 4). Yet let the image have the wrong colors and the out-

put image will be rather unpleasant (see top right and left corner images from Figure 5).

In order to face the problem of undesired color output and have some sort of control flow over it, we decided to investigate and implement existing techniques for preserving image color. Broadly speaking, color preservation is a pre-processing technique done to both content (C) and style (S) images and aims to produce a new style image (S'). The desired outcome (S'), once (C) and (S) have been fed to the model, is a synthesized image that preserves both the colors and the content (objects) of the content image while rendering them in the style of a chosen artistic work. According to Gatys *et al.* [3], this technique is meant to fix a shortcoming and disadvantage in his first version of the AST model: the colors of the style image are transferred onto the the content image if not color preservation is applied to them.

3.3.1 Color preservation using YUV from cv2

When it comes to style transfer, the colors that make up the style image (S) are, to a certain extent, irrelevant [1]. For example, all we might want from S are its 'brushstrokes' or geometric shapes, that is, its style.
To avoid this image-style- color transferring, first, we converted (C) and (S) from BGR to the YUV color space [2]. Second, We extracted the UV channels from (C) and the Y channel from (S) [3]. Finally we merged these extracted channels and converted back to BGR in order to create the new style image. Thus, the newly created style image (S') contains the the colors of (C). This approach takes advantage of the fact that the YUV color space separates the white and black information from the color information [7]. Once we carried out this image pre-processing step we fed both (C) and (S') to the model [4].

3.3.2 Linear Histogram Matching

Gatys *et.al.* (2016) [3] layed out two different approaches to accomplishing color preservation: Histogram matching and Luminance- only transfer. For this project we implemented the former one. In a nutshell, linear histogram matching consists of manipulating *the pixel values in an image so that they will have a similar histogram to another image* [5]. More precisely, if S is the style image and (C) the content image, the key to this approach is to produce a style image S' that contains the colors of (C). Subsequently, (S') and (C) are fed to the

[1] This is mostly a matter of taste, one might rather have the content image (C) keep the colors of the artwork besides the style itself.

[2] As shown by the implementation, cv2.cvtColor() converts images from one color space to another.

[3] From (C) we took the U (blue projection) and V (red projection) channels since it is the colors in (C) that we want to preserve. On the other hand, from S we only took Y (brightness) because it is its colors we want to avoid transferring.

[4] Refer to the Github repository 'Images – color_preservation' folder in order to have a feeling of how this image S' looks like.

model. To achieve color preservation we implemented a linear transformation of the source pixel x_i. That is, we want to find a linear transformation of each RGB pixel of the style image x_i to a new space:

$$x_{s'} \leftarrow Ax_s + b \tag{3}$$

To achieve this transformation we proceeded as follows: 1. First we computed the mean colors of the content and style images μ_s and μ_c respectively:

$$\mu = \sum_i \frac{x_i}{N} \tag{4}$$

2. We calculated the content and style pixel covariance σ_c and Σ_s respectively:

$$\Sigma = \sum_i \frac{(x_i - \mu)(x_i - \mu)^T}{N} \tag{5}$$

3.The transformation is chosen such that $\mu_{s'} = \mu_c$ and $\Sigma_{s'} = \Sigma_c$. Besides, these constrains must be satisfied:

$$b = \mu_c - A\mu_s \tag{6}$$

$$A\Sigma_s A^T = \Sigma_c \tag{7}$$

4. To find a solution to A that satisfies the constrains; the authors introduced two variants: Cholesky decomposition and 3D color matching. We implemented the latter one in this project since, according to the paper [3], this approach rendered better results.

5. To implement 3D color matching, first, we used eigen decomposition to factorize the covariance matrices (from step 2) into $U\Lambda U^T$ (that is to say, $\Sigma = U\Lambda U^T$). Using the eigenvalues and vectors from above, we computed the the square root $\Sigma^{1/2} = U\Lambda^{1/2}U^T$. This gave us all the parameters needed to obtain A and thus perform the transformation.

$$A_{IA} = \Sigma_c^{1/2}\Sigma_s^{-1/2} \tag{8}$$

4 Ablation studies

4.1 Content layer modifications

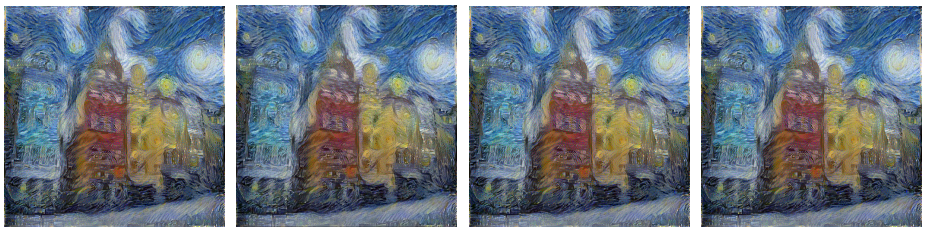
In this section we study the effects of switching the content layer block4conv2.[5].

The results shown in the charts [6] below hint at the fact that changing the content layer does not actually make much of a difference in terms of performance[7]. The total losses do not seem to be extremely sensitive to layer alterations. On the other hand, the aesthetic of the resulting images does not change

[5] As mentioned above, this is the content layer utilized in the original implementation

[6] The column 'Layer' indicates the content layer used. 'Loss it.' displays the the loss obtained at iteration 500, 2000 and 4900

[7] To test the effects of these modifications, we ran the model for 5000 epochs.



(a) Fig. A (b) Fig. B (c) Fig. C (d) Fig. D

Figure 2: Visualization of the content layers modifications. (a), (b), (c) and (d) are the output images for the respective content layer modifications as mentioned above.

much either, as one can tell from the images below. In all cases, after a 100 iterations it was already possible to see the main features of the style image being depicted onto the content image.

Layer	Loss it. 500	Loss it. 2000	Loss it. 4900	Result
block3 conv1	7.92e+06	9.26e+05	2.76e+05	Fig. A
block4 conv2	8.03e+06	1.05e+06	3.95e+05	Fig. B
block5 conv3	7.90e+06	9.07e+05	2.57e+05	Fig. C
block5 conv4	7.90e+06	9.10e+05	2.58e+05	Fig. D

4.2 Style layer modifications

As mentioned above, the paper's implementation[4] uses five layers as the raw material for the style extraction. Thus, in this section we assess the importance the amount of layers plays in style transferring.
Initially, we decided to add two more layers (block5 conv2 ,block5 conv3, see first row of the table below) to study how an increasement in the number of these, the layers, impacts the quality of the generated images [8]. After running the model for 5000 epochs we found that adding more style layers for the extraction of texture may lead to an unwanted predominance of the style in the final image. This may end up causing a partial loss of the objects from the content image. This seems to be confirmed after having added three more layers to the initial five from the original implementation (block5 conv2, block5 conv, block5 conv, second row of the table below). A careful observer might notice that in Figure F part of the houses from the content image get lost in Van Gogh's brush strokes. It is important to notice that the training took longer as more layers were added. This was, nonetheless, an expected side effect since the sizes of the

[8]the first and second row of the table below show the results obtained after adding two and three layers respectively. The third and fourth row display the results after subtracting two layers. Loss it. contains the total loss after 500, 2000 and 4900 iterations respectively.

(a) Fig. E (b) Fig. F (c) Fig. G (d) Fig. H

Figure 3: Visualization of the style layer modifications. (a), (b), (c) and (d) are the output images for the respective style layer modifications as discussed below.

involved matrices were larger.
We also investigated the effects that might follow up after decreasing the amount of layers used for the style extraction. First we subtracted from the original implementation block4 conv1 and block5conv1 (which are among the last layers that make up VGG19). This change had an immediate impact on the resulting image (Figure G) as it looked as though what had been transferred was more the colors of the style image rather than its style. One can barely see the characteristic brush strokes of Van Goth.
Lastly, we removed the first two layers: block1 conv1,block2 conv1. As one can tell from Figure H, this change did not affect much the aesthetics of the final image. According to these results, it looks like the later layers of the VGG19 are more suited for texture extraction than, for example, the early ones. Turning our analysis to the total costs, we can see that more style layers can be translated into higher losses, whereas fewer layers (row 3 and 4 of the table below) yielded smaller ones. A small loss is, almost always, the desired outcome, however, this does not guarantee a nice-looking synthesized image (for example Figure C).

Layer	Loss it. 500	Loss it. 2000	Loss it. 4900	Result
block5 conv2, block5 conv3	8.04e+06	1.04e+06	3.93e+05	Fig. E
block5 conv2, block5 conv3, block5 conv4	8.05e+06	1.05e+06	3.95e+05	Fig. F
block4 conv1', block5conv1	1.12e+05	6.06e+04	4.60e+04	Fig. G
block1 conv1,block2 conv1	6.68e+06	8.36e+05	3.40e+05	Fig. H

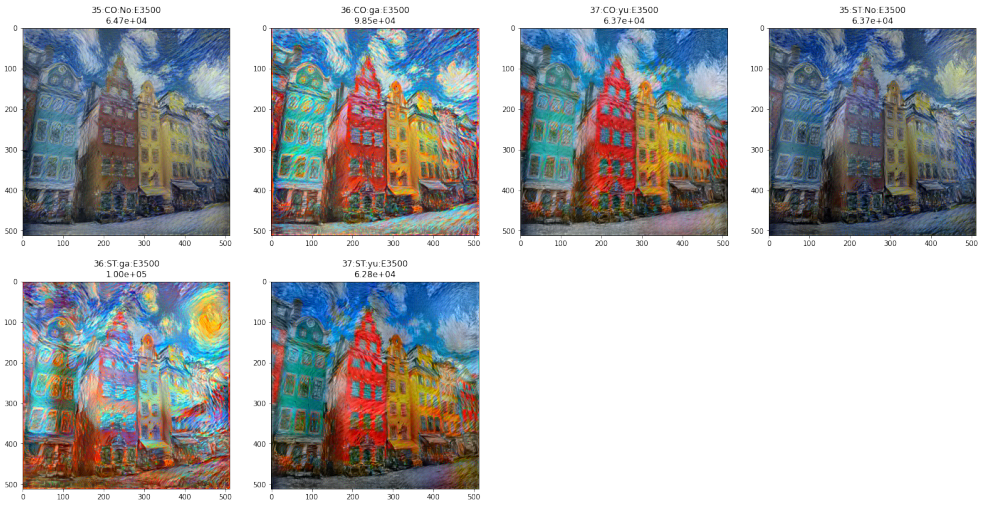


Figure 4: Different results for different parameters after 3500 epochs. The first three started initally with the content image, the other with the style source image. Number 1 and 4 used no color preservation, 2 and 5 the method from Gatys *et al.*, 3 and 6 the 'YUM-method'

5 Results and Discussion

Our implementation achieved similar results as the original papers, as one can see in the various figures in this report for different parameter settings (fig 4, fig 5). While they are probably not good enough to compete with commercially used algorithms, our algorithm could be used as a baseline for more modern approaches. Also our produced images show that it is hard to decide on an objectively best picture or parameter setting, as the optimums of latter depend on the source image combination. It is not even clear that color preservation improves the result, as in the case of the city image one might find the first image from the vanilla algorithm best. In other cases however, as with the lion, color preservation seems to clearly improve the image quality, though it again depends on subjective aesthetics which method is better in this case.

The main disadvantages of the used method for AST are that photorealism is not preserved and that as the training happens on-line, it is not feasible to produce images in real-time.

Another major problem of training images for such a task, is that the used loss can only be seen as a heuristic for generating good images. For one, it depends on the assumption that the VGG-model layers actually capture the relevant content and style. Furthermore even then and if an image has a low loss this does not necessarily mean that the image actually *looks* good. For many source image pairs there might not even be any good combination of style and content. This means, that it is simply not sufficient to just choose the final

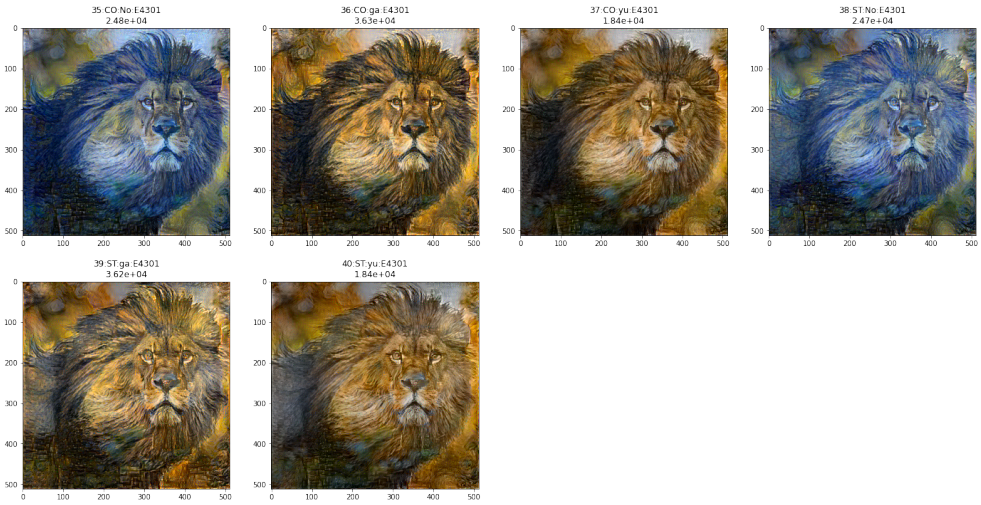


Figure 5: Different results for different parameters after 3500. As content source image a photo of a lion was used, for the style the starry night painting from above. The first three started initally with the content image, the other with the style source image. Number 1 and 4 used no color preservation, 2 and 5 the method from Gatys *et al.*, 3 and 6 the 'YUV-method'

output of the training process but instead at some point a human *must* look at the images and either decide whether the result is acceptable or even choose the best result from a variety of generated images. To ease this step we modified our implementation in such a way that several (differently parameterized) images can be trained at the same time. Regular printing allows for choosing the best intermediary image and early stopping if a source image combination seems not to produce any adequate images. This method of subjectively judging different output images works well when having only to consider few image combinations and was also used during hyperparameter optimization.

6 Conclusion

Neural style transfer can lead to impressive and highly appealing artistic images that rather than being the product of the human mind and creativity alone, are the result of an attempt to understand how the human mind creates. In this work we successfully reimplemented an approach for AST and we complemented it with two methods for color preservation. We were able to achieve similar results which sometimes resemble and even outperform those on the original paper, the differences can be explained by minor differences in choice of parameters and algorithm. We also found out that neither of the two color preservation techniques is clearly superior, and it seems wise to always try out AST with both as well as without any. The reason for this is that although

color preservation is in many cases desired—like when we want to control the colors the generated image will have—, it can also lead to rather simple and less impressive results.
Our work also shares the main concerns of Gatys' *et al.* [4] AST approach: the lack of generalisation to ensure the output image to be the successful balance between content and style. More specifically, every image has to be trained separately and many parameters have to be tried out in order to find out that one good image. Thus, given the scope of our project, stipulating a criteria for automatic selection of good output images was out of reach and would constitute something to be further investigated.

References

[1] Ashley. 10 Reasons to Visit Stockholm, 2018. https://www.edreams.com/blog/reasons-visit-stockholm/ (accessed: 25.02.2021).

[2] Gabe Cohn. AI Art at Christie's Sells for $432,500, 2018. https://www.nytimes.com/2018/10/25/arts/design/ai-art-sold-christies.html (accessed: 01.02.2021).

[3] Leon A. Gatys, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Preserving color in neural artistic style transfer, 2016.

[4] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[5] Aaron Hertzmann. *Algorithms for Rendering in Artistic Styles*. PhD thesis. New York University, 2001.

[6] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340, 2001.

[7] Michal Podpora, Grzegorz Pawel Korbas, and Aleksandra Kawala-Janik. YUV vs RGB –choosing a color space for human-machine interaction. In *FedCSIS (Position Papers)*, pages 29–34, 2014.

[8] Scott Reyburn. JPG File Sells fro $69 Million, as 'NFT Mania' Gathers Pace, 2021. https://www.nytimes.com/2021/03/11/arts/design/nft-auction-christies-beeple.html (accessed: 20.02.2021).

[9] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In *German conference on pattern recognition*, pages 26–36. Springer, 2016.

[10] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos and spherical images. *International Journal of Computer Vision*, 126(11):1199–1219, 2018.

[11] Kabid Hassan Shibly, Sazia Rahman, Samrat Kumar Dey, and Shahadat Hossain Shamim. Advanced artistic style transfer using deep neural network. In *International Conference on Cyber Security and Computer Science*, pages 619–628. Springer, 2020.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] Xin Wang, Geoffrey Oxholm, Da Zhang, and Yuan-Fang Wang. Multimodal transfer: A hierarchical deep convolutional neural network for fast artistic style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5239–5247, 2017.

[14] Holger Winnemöller, Sven C Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Transactions On Graphics (TOG)*, 25(3):1221–1226, 2006.