Reproducibility challenge: 'AdaAttN: Revisit Attention Mechanism in Arbitrary Neural Style Transfer'

Bastien Aymon, Arina Lozhkina, Ricardo Ferreira Ribeiro
EPFL
Lausanne, Switzerland
bastien.aymon@epfl.ch arina.lozhkina@epfl.ch ricardo.ferreiraribeiro@epfl.ch

Reproducibility Summary

Scope of Reproducibility

The authors claim their model allows a better trade-off between style pattern transferring and content structure-preserving than previous style transfer models. Notably, they claim their model can take into account both shallow and deep features, which is not the case for existing models.

Methodology

We start by reproducing the author's code [2], based on the instructions given in the paper only. We report implementation issues and details not mentioned by the authors in the paper. We proceed to describe the multiple networks used and their interconnection thoroughly, comparing our code, the author's code, and its description in the body of the paper. We assess the model's performance by using it on a new set of style and content images and comment on the qualitative results. We propose robust metrics to assess the model's performance quantitatively.

Results

We describe the model's implementation in great detail and highlight discrepancies between the author's description of the model and what it actually is. The outputs we compute are reasonably close to the ones presented in the original paper. We also show that AdaAttN indeed has a never-seen-before ability to transfer both deep and shallow features.

What was easy

Through an existing GitHub implementation [4], generating new style transfer outputs based on the author's already trained model was very convenient. An existing online implementation [1] also made individual outputs generation very convenient, even though it could not be used for image batches.

What was difficult

The paper lacked information about the code implementation, notably about the delicate feature concatenation step, which is one of the main novelties introduced by the paper. The fact that linear algebra operations are applied on large-sized feature vectors and matrices required an extensive amount of RAM and GPU memory, which made it difficult to run and test our code. Also, because of model complexity, training took a lot of time, even with a powerful NVIDIA Tesla P4 GPU. Even for images, to get the first results, it was necessary to wait for hours. Experiments on style transfer on videos would have taken even more time, so we could not verify them. Also, we could not verify the survey that authors use as proof they obtained better results than in previous papers.

Communication with original authors

The authors sadly never answered our solicitations via email.

[1] https://replicate.com/huage001/adaattn

1 Introduction

Style transfer algorithms have received increasing interest in the last few years, and current models are getting closer to outputting proper works of art. The model presented by Liu *et al.* [6] is one of the latter, and claims to achieve never-seen-before performance and an ability to encapsulate both deep and shallow features. This is especially interesting since existing models are usually either very good at transferring the actual *style*, *e.g.*, the color palette, or at keeping the original image's contents intact, that is, keeping it *recognizable*.

For future developments to emerge, it is capital to first assess the model's true performance, and then make new implementations based on that model as straightforward as possible. This is especially important since as the authors mention, this model has the potential of improving other manipulation or translation tasks.

Figure 1: General working principle of the style transfer model. Given a content and style input, the model outputs the content image *in the style of* the style image. This stylized image was generated using the author's pre-trained model.

We will thus begin by extracting the authors' main claims, before confronting them. This shall be done using our own experience of rewriting the algorithm *from scratch*, and both new and state-of-the-art style transfer quality assessment techniques.

2 Scope of reproducibility

First of all, we will verify an implicit, yet probably the most important claim of the paper, that is, that **the model is built - and buildable - as described in - and from - the paper**. Said differently, we want to verify that the model *is what the authors say it is*. Then, we will verify that **the model behaves as presented in the paper**, that is, that it outputs the same stylized pictures when fed the same content and style images. Finally, the authors' most explicit claim, that **their model can encapsulate both deep and shallow features**, unlike existing models, will be examined.

3 Methodology

3.1 Basic reproducibility check

We shall re-train the author's model following their exact instructions. We shall then verify that the results presented in the paper can be reproduced using a freshly-trained model. We will also verify that inferencing the model and using the authors' weights gives the same outputs as in the paper.

3.2 Code reimplementation and analysis

In the spirit of reproducibility, it is capital that one can reproduce the model's implementation based on the indications given in the paper by the authors. That is why we shall start by reimplementing the whole model from scratch. In light of this, we shall provide specific indications that should make future implementations and redesigning of the model very clear. We shall also highlight any discrepancies between the author's description of the model and its actual architecture to clear off any confusion they may cause in the future.

3.3 Model performance assessment

Lastly, we will try to assess the model's performance *quantitatively*, notably, its ability to transfer both deep and shallow features. The authors do it through a survey, which is both impossible to put in place for us, and not a rigorous tool. For all we know, the authors might have been presenting the surveyed people with a certain type of *very good* outputs only. Using state-of-the-art style transfer loss functions and other existing metrics we adapted, we will *quantify* the model's performance, also comparing it with existing style transfer algorithms. Since no universally accepted metric exists, we will have to choose exactly what aspect of the model we want to test. Given the author's assumptions, it is reasonable to focus on **shape and color transfer performance**, which shall evaluate the model's ability to represent both shallow and deep features, respectively.

A model's ability to transfer *shallow* features means that it has to preserve the original content to some extent. That is, the original picture's shape, what it *is*, must remain recognizable even after the style transfer. The following can typically be evaluated through so-called *local self-similarity descriptors*. Kolkin [5] mentions that the latter works the same way pareidolia, humans ability to recognize faces in everyday objects. In particular, we shall adopt Chatfield *et al.*'s reimplementation [3] of Irani's 2007 work [7] on matching local self-similarities as a robust estimator of the model's *content preservation* ability.

The definition for *deep* features is less clear, but they generally correspond to more *vague* concepts such as the textures, the way the colors are arranged across the picture, etc. Naturally, this is a much more difficult aspect to evaluate quantitatively, since it has to do with the *artistic* quality of the image. We shall then limit ourselves to evaluate how well the style image's **color palette** is transferred. This will be done by adapting and using the loss function of a very recent model [1] specially designed to transfer color palettes, CAMS (Color-Aware Multi-Style Transfer). We will also implement our histogram-based, chi-square distance measurement.

Through those two quantitative experiments, we will be able to verify if the author's model indeed performs better than AdaIn and SANET, which are the two state-of-the-art style transfer models the authors mention in their paper.

4 Model descriptions

Based on the author's model description, and more importantly our experience of rewriting it, our first contribution is to present two full, detailed schematics of the model proposed by the authors that should allow for easier reproduction of the model's structure in the future on Figs. 2 and 3.

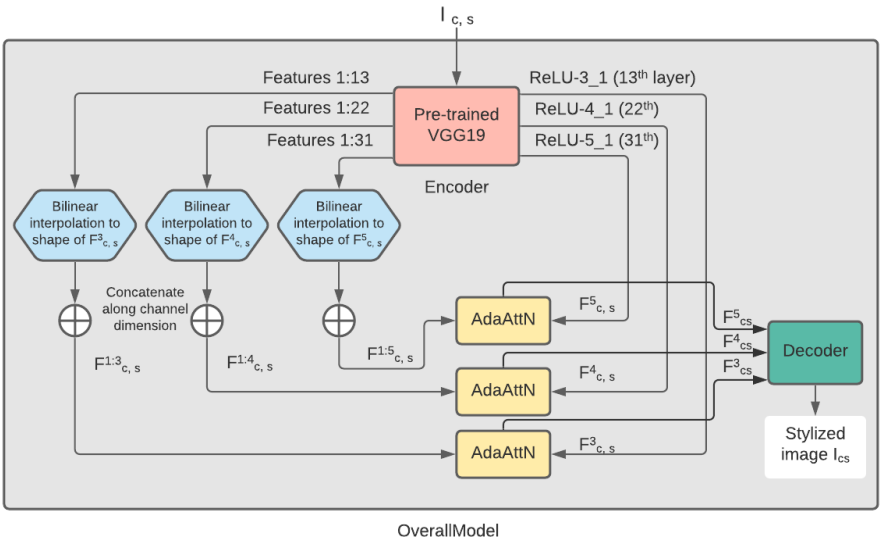


Figure 2: Detailed schematic of the full model.

The input of the general model receives 2 images, one for content and one for style, both of size `(batch size, channels, H, W)`, where H and W are the height and width of the image, respectively. It outputs a stylized image I_{cs} of the same size as the input.

In more detail, the content and style images are fed as input of a pre-trained VGG19 neural network that acts as an encoder. Features are then extracted from both images, from layer 1 to layers 13, 22, and 31 respectively (after applying the ReLU activation function). They are then bilinearly interpolated and concatenated according to the dimensions of the image channels (we shall call these *merged* features $F_{c,s}^{1:x}$, where x is the index of ReLU features (3, 4, or 5)). This step is crucial since it is what allows the model to encapsulate *both* shallow and deep features, that is, features from the first and last layers of the VGG19 network *simultaneously*. Together with the individual features $F_{c,s}^{x}$ of layers 13, 22 and 31, each feature set is processed by the AdaAttN module, which is the central part of this paper.

The AdaAttN module is presented in Fig.3. It takes as input the previously described features and merged features of the style and content image. The first stage is the attention map generation. For each layer, the features of the style image are processed by the convolutional layer h. The rest first go through the mean-variance normalization Norm and the merged features also go through the convolutional layers (f and g). The features for the content and style images will be denoted by F_c, F_s, respectively, and the merged features. We also define $Q = f(Norm(F_c^{1:x}))$, $K = g(Norm(F_s^{1:x}))$ and $V = h(F_s)$. Furthermore, each feature set is *flattened*, and the dimensions are converted from `(batch size, channels, H, W)` to `(batch size, channels, H * W)`. Based on these new quantities, an attention map $A = softmax(Q^T * K)$ is calculated. Importantly, this is now a *matrix* quantity.

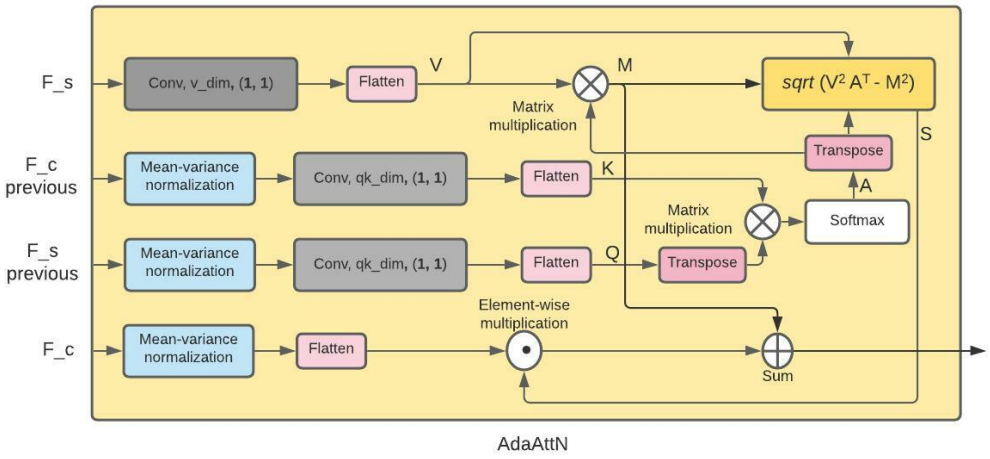


Figure 3: Detailed schematic of the AdaAttN module.

The next step is to derive the attention-weighted mean and standard variance map. We compute the mean as $\mathbf{M} = \mathbf{V}\mathbf{A}^T$, and the standard variance is $\mathbf{S} = \sqrt{\mathbf{V}^2\mathbf{A}^T - \mathbf{M}^2}$. Importantly, if the radical's expression is less than 0, then it should be replaced with zero instead. This detail is not mentioned by the authors but implemented in their actual code.

Finally, the last step is the *adaptive normalization*. AdaAttN's output is then $\mathbf{SV} + \mathbf{M}$, where $\mathbf{S}$ is called the *scale* and $\mathbf{M}$ the *shift*. All feature sets are then sent to the input of the decoder which will output the stylized image. The decoder structure is represented in Appendix B.

4.1 Datasets

The authors train their model on the COCO and Wikiart datasets, for the content and style images respectively. In particular, the Wikiart dataset regroups more than 50 000 paintings from 195 artists, covering a wide range of styles, which allows for robust style transfer for any style image. We shall use the same datasets for the training of our reimplementation. Given our limited computational power, the datasets were reduced to only 10000 pictures. In each case, 80% of the dataset is dedicated to training, and the remaining images are kept for validation. Also, while the authors randomly crop the images to 256x256 subsets, we had to reduce the latter to 128x128 squares for RAM and GPU memory reasons.

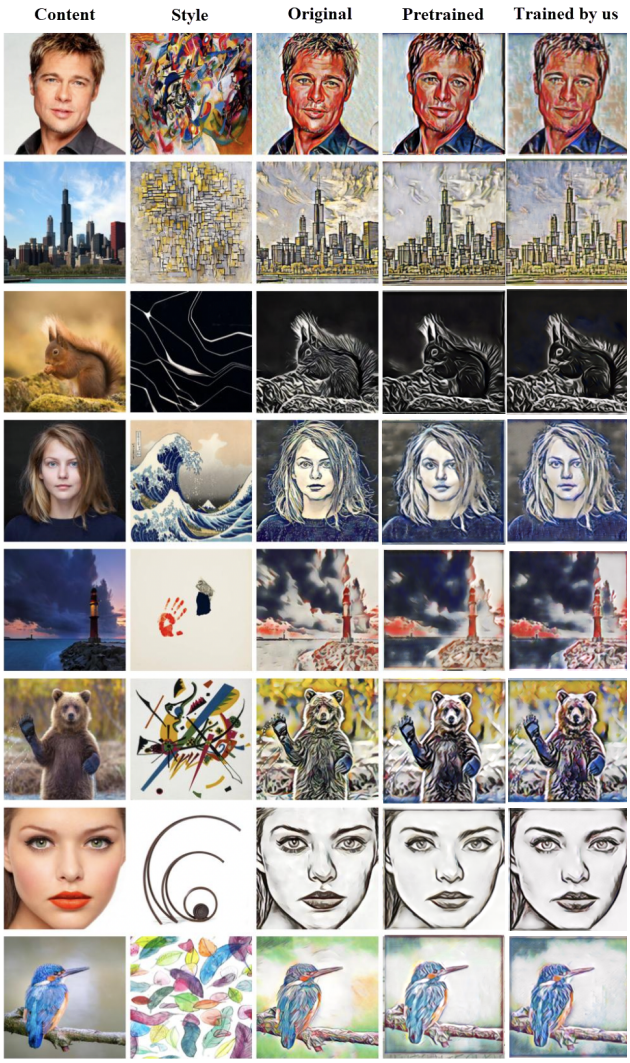


Figure 4: Stylized images: authors' and ours

4.2 Hyperparameters

Although our computational resources were not sufficient to reproduce precisely the original training process, we aimed to take the same parameters as the authors. These are the parameters of the Adam optimizer ($\alpha = 1e-4$, $\beta = (0.9, 0.999)$), batch size = 8 and losses parameters λ_g and λ_l equal to 10 and 3 respectively.

4.3 Experimental setup and code

We ran our implementation on a Google Cloud virtual machine because of the high RAM consumption. Unfortunately, we did not have access to a GPU with sufficient memory allocation, so we were forced to train using a CPU only. Our code repository and instructions on how to run the model training can be found on GitHub[2]. Our preliminary results are presented in AppendixA.

4.4 Computational requirements

Experiments were conducted on an Ubuntu workstation equipped with an 8 cores Intel Haswell processor with 30Gb of RAM and and NVIDIA Tesla P4 GPU. These resources were obtained through the Google Cloud VM services.

5 Results

5.1 Result 1: Basic reproducibility check

We use the pre-trained model of the *officially unofficial* PyTorch implementation of the paper to reproduce the outputs based on a pre-trained model. We also train the author's model ourselves, setting it up as described in the paper (this took 3 GPU days). We can see in Fig. 4 that our results are almost the same as the authors' ones. The difference is probably due to two reasons: the images we used as input had a worse quality than the authors' ones and the model the authors used for the paper is not the same as the one on GitHub.

5.2 Result 2.1: Code reimplementation and analysis

We have implemented a solution based on the PyTorch framework. For this, a class was created for iterating the dataset (`simple_dataset.py`), configuring the model (`models.py`), implementing the error function (`loss.py`), and generally running training (`train.py`). Dataset implementation class `ContentStyleDataset` is based on the built-in PyTorch class and includes methods `__len__` and `__getitem__`. The training implementation class `TrainModel` includes initialization and methods `load_data` to fill `DataLoaders`, `train_epoch`, and `validation_epoch`, which correspond to train and validation iterations respectively. `plot_result_images` is used to plot intermediate results and `train_full` to run the full loop of training the model.

The model implementation consists of classes `AdaAttN`, `Decoder`, `EncoderModel`, and `OverallModel`. The encoder gets features from ReLU 3, 4, and 5 and all layers before them to concatenate them. Its implementation contains methods `bilinear_interpolation`, which down-samples the input feature to the shape of features from VGG19, `get_previous` to concatenate features of current layer with down-sampled features of its previous layers by channel dimension and `forward` which runs all features preprocessing to return them in the correct shapes. The AdaAttN class

is based on the built-in PyTorch neural network module and includes methods `init`, `norm` which makes channel-wise mean-variance normalization and `forward`, including attention map generation, attention-weighted mean, and standard variance map and adaptive normalization. The decoder transforms vectors to images of the initial shape, stylized by the style image. Its structure is described in Appendix B.

5.3 Result 2.2: Implementation problems

The model includes an encoder based on the ReLU 3, 4, and 5 layers of the VGG19 pre-trained neural network, three AdaAttN layers and a decoder. When implementing the model, difficulties arose with the decoder, since with a kernel size = (3, 3) padding is necessary to maintain the required dimensions. This was not included in the description of the model structure. Also, the mathematical description of the AdaAttN class given in the main body of the article deferred from the code implementation for some linear algebra operations. Notably, Eq.(6) in the original paper states that $\mathbf{M} = \mathbf{V}\mathbf{A}^T$, while the Appendix' implementation writes $\mathbf{M} = \mathbf{V}^T\mathbf{A}$. As a result of checking the outputs in both cases, no significant differences were found.

One of the main problems that we encountered is the computing power required to train the algorithm. This is a consequence of the complexity of the model, notably its usage of multiple tensors, which are *very* memory demanding. Moreover, there were problems with the square root present in the AdaAttN implementation, as well as in the loss function. When a negative radicand appears, we have to replace it with zero. To avoid errors during backpropagation, we also had to add a small additive value 1e-8 to it. That is detected with `torch.autograd.set_detect_{}anomaly(True)`. Otherwise, we would get NaN values in training. This important detail was eluded by the authors.

During training we did not obtain adequate results because of the combination between a softmax function and specific loss, so we had to replace them with a sigmoid function. Also, the authors' code is based on a pix2pix and CycleGAN implementation, which leads to numerous functional parameters which are not described in the paper (like the scheduler, different types of datasets, network initialization method, etc.). That made the original code analysis cumbersome.

The code also includes details that are not in the paper, like the presence of convolutional layers in the transformer model after applying AdaAttN. The code also uses a content loss which is never described in the paper. The implementation includes different encoder features which are not discussed. In general, the original code corresponds to the paper description, but more details would have been needed for proper implementation. Also, by comparing the model's structure in the paper and the code, we can suppose that difference between the original implementation and paper description could change results.

5.4 Result 3: Assessment of the model's performance

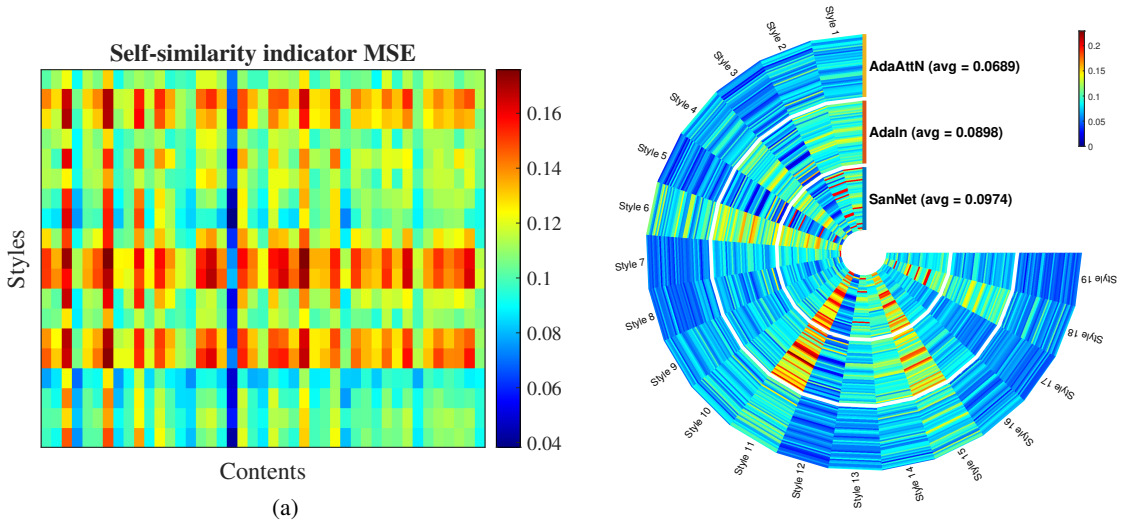
We compute Irani's self-similarity indicator matrices for 817 stylized pictures emanating directly from the author's pre-trained model. The chosen styles were both used by the authors in the paper and for training, or never seen before. The same can be said about the content images, that span from faces and scenery to animals and architecture. We compute the mean square error (MSE) of the indicator matrices (taken component-wise) for each content and output pair. We also compute the mean MSE over the 817 samples. These steps are repeated using SANet and AdaIn, two state-of-the-art style transfer algorithms the authors give as reference. The mean MSE is worse in this case, meaning that based on the metric we chose, state-of-the-art models are *less efficient* at keeping the original content's structure than the author's model. As a proof-of-concept that the indicator works, we present in Fig. 5a a case where the MSE is computed for content image 19 only. In that case, the column corresponding to that image stands out. In Fig. 5b, we present the results for the general case.

To evaluate the model's color palette transfer ability, we proceed to extract the histograms of each pair of style and output images for AdaAttN, AdaIn, and SANET, as shown in Fig. 6a. To evaluate how close the respective distributions are, we then compute the chi-square distance between the histogram pairs for each RGB channel. We obtain an average chi-square distance of 0.2063 for AdaAttN, 0.0174 for AdaIn, and 0.1067 for SANET, meaning that based on that metric only, the author's model performs worse than state-of-the-art models. To further investigate this phenomenon, we also compute the CAMS loss of the same dataset for the same three models, as presented in Fig. 6b.

6 Unverified claims

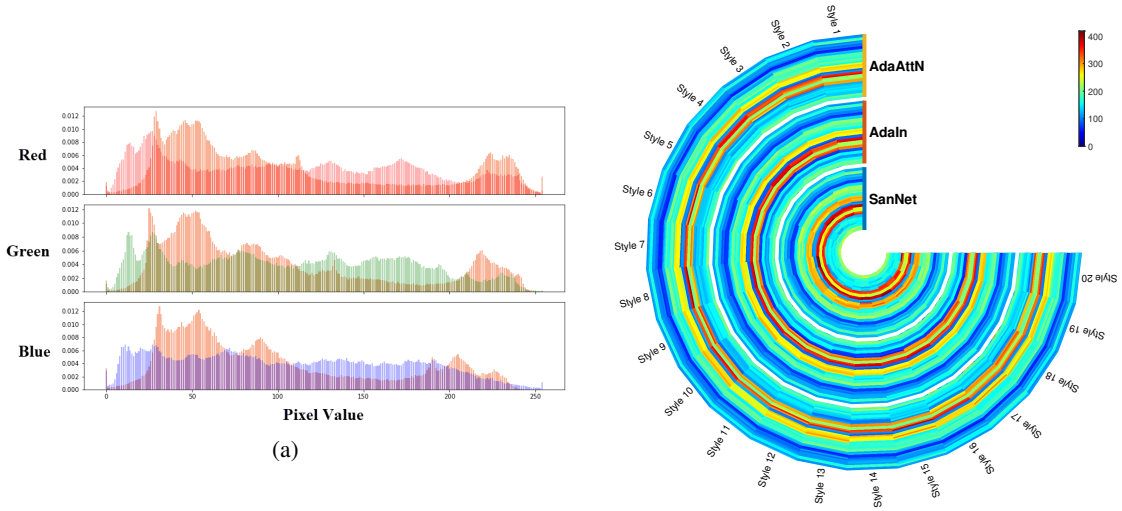
6.1 Video Style Transfer

One major claim of the authors is the development of a novel image-wise similarity loss based on attention mechanism to avoid the flickering artifacts that typically occur when performing style transfer on videos. The latter claim could



(a) (b)

Figure 5: MSE of the self-similarity indicator on 43 content and 19 style images for (a) an example case where all the outputs are compared with content image 19; this image is clearly identified over all styles (b) the general case; content images are spread along the radius while styles change with the polar angle. Some styles systematically result in worse shape transfer performance. Average MSE is 0.0689 for AdaAttN, 0.0898 for AdaIn and 0.0974 for SANET.



(a) (b)

Figure 6: Evaluation of the model's color palette transfer ability. (a) Histogram of the style (foreground) versus the output (background) images using AdaAttN. The histogram's color distribution are far apart in that case (style 7, image 42), as well as in general. (b) CAMS loss function for AdaAttN, AdaIn and SANET. Some contents systematically result in poor color transfer performance. All models tend to perform equally well.

not be verified, however, since on one hand no code was made available by the authors for this specific part of the implementation, and since on the other hand training the model for video style transfer would have required computational resources we do not have access to.

6.2 Multistyle Style Transfer

The author mention that their model can be adapted to perform *multistyle* transfer, that is, transferring multiple styles on the same content image. The results presented in the paper look impressive, but no code about this specific part was published. Also, the paper lacked sufficient information for us to reproduce it: the authors mention a concatenating strategy and a 'mean and standard variance maps averaging', but no more details are given.

7 Discussion

As our computational resources and distributed computations experience are not sufficient to complete our model training, conclusions are mostly based on the analysis of the authors' code training and applying their weights. After checking the authors' code we found out that they made some parameters non-trainable and parallelized encoder features computation, which is out of our reach. In fact, the original code was not exactly the same as it had been described in the paper. However, after the implementation from scratch, we still obtained preliminary results and got a model that tends to decrease loss and leads to adequate stylized images. We expect that more training time will permit us to achieve the same outputs as authors, but we will check it further when we have enough resources.

Retraining the authors' model and inferencing it with the pre-trained weights demonstrated the correctness and complete reproducibility of the results presented in the paper. Since outputs we obtained by retraining the network from scratch also fit the ones from the paper, we can also safely state that the model works as claimed.

Our model quality assessment showed that AdaAttN indeed seems to have an unprecedented ability to conserve the initial content's structure and general appearance. This is qualitatively the case as well, as outputs from AdaIn and SANET typically looks more blurry, or even hardly recognizable in some cases. The color palette transfer ability test was not conclusive, however, as histogram analysis showed existing models better preserve the style image's color distribution. Computing the CAMS loss gave us more insight: unlike structure preservation, color palette transfer is extremely dependent on the original content image. In practice, we notice that content images with large patches of uniform color tend to get overly colorful while stylized. This is especially true with AdaAttN since, in this case, an output blur helps smoothing these coloring *singularities*.

Lastly, we shall remark that outputs from AdaAttN usually simply look *better* than existing models. This bold statement only reflects our personal feeling, but it shall also remind the reader that evaluating art's quality through *metrics* is probably not appropriate, and definitely limited. Still, the tools we presented deemed to be useful when it came to *diagnose* some of the model abilities. From that analysis, we conclude that the author's claims *that we could verify* are justified. We cannot conclude anything about the reproducibility of the claims listed in Sect. 6.

7.1 What was easy

The author's code was easily available on their GitHub, and both training out output generation was convenient. Also, an existing online implementation of the model allowed for quick, fun, and easy outputs generation.

7.2 What was difficult

The explanations provided by the authors about the model were generally not incorrect but proved to lack details when it comes to actual implementation. In particular, for non-experienced researchers it was problematic to take into account implementation details which could lead to non-adequate results like shape transformations, NaN-values outputs caused by particular layers, dealing with custom losses and memory issues, etc.

We also clearly underestimated the computational power required to train the model. In multiple instances, we had to profoundly change the structure of our model because our current one was computationally too expensive. This required us to carefully choose the batch and subset size, and especially to deal with multiple GPU and RAM issues, which we were not accustomed to. Since the GPU memory we had access to was not sufficient, we were forced to train our implementation on a CPU, which resulted in especially slow training times.

7.3 Communication with original authors

The authors never answered our solicitations via email, either asking direct questions or to arrange a meeting. Many implementation questions thus still remain unanswered.

References

[1] Mahmoud Afifi et al. *CAMS: Color-Aware Multi-Style Transfer*. 2021. arXiv: 2106.13920 [cs.CV].

[2] arinaLozhkina. *RC2021-AdaAttN*. https://github.com/arinaLozhkina/RC2021-AdaAtt. 2021.

[3] Ken Chatfield, James Philbin, and Andrew Zisserman. *Efficient Retrieval of Deformable Shape Classes using Local Self-Similarities*. 2009.

[4] huage001. *AdaAttN*. https://github.com/Huage001/AdaAttN. 2021.

[5] Nicholas Kolkin. *Non-Parametric Neural Style Transfer*. 2021. arXiv: 2108.12847 [cs.CV].

[6] Songhua Liu et al. *AdaAttN: Revisit Attention Mechanism in Arbitrary Neural Style Transfer*. 2021.

[7] Eli Shechtman and Michal Irani. *Matching Local Self-Similarities across Images and Videos*. June 2007. DOI: 10.1109/cvpr.2007.383198. URL: http://dx.doi.org/10.1109/CVPR.2007.383198.

Appendices

A Preliminary results

Fig. 8 presents a preliminary output using our own reimplementation. The network was trained on CPU over 500 pictures for 1 epoch, for a total duration of 14 hours. Since we are operating on a CPU, a lot more training time would be required to obtain outputs close to the authors'.

Figure 7: Preliminary result using our network. From left to right: content, style and output. The output has a gradient-like structure, which is a sign that our model is training properly.

Figure 8: Another preliminary result using our network.

B Decoder structure

1. **Preprocessing** Unflatten(batch, channels, H*W) -> (batch, channels, H, W)
2. **Stage F5 –> shape: (512, H / 8, W / 8)**
   $x = F_{cs}^{5}$
   upsample(scale=2)
   $x + F_{cs}^{4}$
   Conv2d(shape * 2, shape * 2, (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)

3. **Stage F4 –> shape: (256, H / 4, W / 4)**
   Conv2d(shape * 2, shape , (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   upsample(scale=2)

4. **Stage F3 –> shape: (128, H / 2, W / 2)**
   concat([x, F_{cs}^{3}], dim=1)
   Conv2d(shape * 2, shape , (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   Conv2d(shape, shape , (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   Conv2d(shape, shape , (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   Conv2d(shape, shape // 2 , (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   upsample(scale=2)

5. **Stage F2 –> shape: (64, H, W)**
   Conv2d(shape, shape // 2 , (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   Conv2d(shape // 2, shape // 4 , (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   upsample(scale=2)

6. **Stage F1 –> shape: (3, H, W)**
   Conv2d(shape // 4, shape // 4, (3, 3))
   ReLU
   ReflectionPad2d (1, 1, 1, 1)
   Conv2d(shape // 4, 3 , (3, 3))
   ReflectionPad2d (1, 1, 1, 1)