Executive Report on the Implementation of a Real-Time Style Transfer Model

Robert Weeke
BSc IT-Systems Engineering
Hasso Plattner Institut
Potsdam, 14482
robert.weeke@student.hpi.de

Nils Schmitt
BSc IT-Systems Engineering
Hasso Plattner Institut
Stahnsdorf, 14532
nils.schmitt@student.hpi.de

Abstract

We were to implement our own real-time style transfer convolutional neural network to practice the use of neural networks for real-life applications. Therefore we trained a convolutional neural network using a *perceptual loss function*, similar to the approach proposed by Johnson et al. [1]. Our model is fast enough to transform a webcam input stream into an output stream with the desired effect.
In order to optimise our results we then used techniques such as upsampling and instance normalization and compared our results.

Keywords: Style transfer, deep learning, convolutional neural network

1 Introduction

In times of social media and permanent video streaming it can be important to set oneself apart from others by preferably strong visual effects. One of such effects is seeing a video or picture in a style you could not create yourself.
In order to implement this, we first implemented the methodology of perceptual loss described by Gatys et al. [2]. However, this directly optimises the image on which the style is to be applied. Therefore, it is not suitable for real-time style transfer because it requires several iterations of backpropagating the loss on this image, which takes too much time for our purposes. So that we could actually run our application in real-time, we used the procedure presented by Johnson et al. [1]. For this, we trained a convolutional neural network with the previously mentioned *perceptual loss*, consisting of a style loss, a content loss and a total variation regulariser. The inference of this convolutional neural network is several magnitudes faster than the previously mentioned method. Thus this network can then be applied to each individual frame of a video, so that, for example, a video stream from the webcam can be adjusted with the previously trained style.

2 Dataset

2.1 COCO Data set

As explained before, we have a selected image that contains the style to be transferred. During training we then try to apply this style to any other images. For this we used the COCO Image Data set [3] as suggested. We decided to train on the 2017 Train image data set, which contains 118 thousand images. As we needed to fit the images into our CNN we resized them to have a width or height (the smallest of their boundaries) of 256 and then performed a center crop. We did not perform any split of the data set, because we were not aware of any metrics that allow the comparison of two successfully trained models other then the visual results and for testing we used selected images. Due

Preprint. Under review.

to the size of the data set we decided to just train for a single epoch which is also why we did not apply any measures against overfitting.

3 Architecture and Training

3.1 Perceptual Loss Function:

First of all we needed an objective function in order to know the performance of our model and especially to optimise its performance by using maximum likelihood. Therefore we used the so called perceptual loss suggested by Gatys et al. [2]:

$$\lambda_{\text{style}} * \ell_{\text{style}}(y_{\text{style}}, \hat{y}) + \lambda_{\text{content}} * \ell_{\text{content}}(y_{\text{content}}, \hat{y}) + \lambda_{\text{TV}} * \ell_{\text{TV}}(y)$$

Here $\hat{y}$ is a generated image whose loss should be calculated, y_{style} is the style image that was selected and y_{content} is the content image the generated image is supposed to resemble.

For the content- and style-loss functions we used a CNN that was trained to perform image classification, as proposed by Gatys et al. [1]. The general idea behind this approach is that the CNN already captured an idea of how the input image features look like in order to classify the image.

Content Loss This is why the content loss $\ell_{content}(y_{\text{content}}, \hat{y})$ takes the activations of a certain layer in the Loss-CNN and compares this activation when using $y_{content}$ and $\hat{y}$ as input by taking the 2nd norm of their difference:

$$\ell_{content}(y_{\text{content}}, \hat{y}) = \frac{1}{2 * c * h * w} ||\phi(y_{\text{content}})_i - \phi(\hat{y})_i||_2^2$$

where ϕ is the Loss-CNN and i is layer where the activations are taken from and c, h, w are the channel, height and width size of the activation.

Style Loss The style loss is now constructed in a similar fashion:

$$\ell_{\text{style}}(y_{\text{style}}, \hat{y}) = \frac{1}{(2 * c * h * w)^2} ||G(y_{\text{style}})_i - G(\hat{y})_i||_2^2$$

G is the so called Gram-matrix which is calculated by reshaping $\phi_i \in \mathbb{R}^{c \times h \times w}$ into $\psi_i \in \mathbb{R}^{c \times h \cdot w}$ and then $G_i = \psi_i \cdot \psi_i^T$, so $G_i \in \mathbb{R}^{c \times c}$. The intuition behind the use of this Gram-Matrix for the style-loss is that the Gram-matrix is approximately proportional to the Covariance-matrix if the vectors (in our case the given feature maps) are centered random variables [10]. This means that the Gram-matrix expresses the relation between the different channels or features.

Regularisation The so called total variation regularisation term is used in order to control the smoothness of the image. Therefore:

$$\ell_{\text{TV}}(\hat{y}) = \sum_c \sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j}|^2 + |y_{i,j+1} - y_{i,j}|^2}$$

It is important to note that due to use of this Loss-CNN y_{content}, y_{style}, $\hat{y}$ need to have the same fixed dimensions. In our case this was $(3, 256, 256)$. We decided to use VGG16 as a loss network, as suggested by Johnson et al., because this already provided good results and it would therefore not have made sense to switch to a deeper network or even to train our own. Additionally we used $\lambda_{\text{content}}, \lambda_{\text{style}}, \lambda_{\text{TV}}$ to weight the different terms accordingly. The layers we used for calculating the style and content loss were the same as the ones proposed by Johnson et. al [1], ReLU3_3 for the content, and ReLU1_2, ReLU2_2, ReLU3_3, ReLU4_3 for the style-loss.

3.2 Image Optimisation Approach

Our first approach was to simply initialize an image to the content image or gaussian noise and then use as proposed by Gatys et al. the LBFGS optimiser to minimize the complete loss term on this image by optimising this image with backpropagation. This approach achieved visually pleasing results but due to the optimisation process that needs to be applied to every single content image, the method was too slow to be used for real-time style transfer, and therefore was not suitable for our project.

Table 1: Architecture Overview

Layer	Input shape	Parameters #
Convolution:		
9x9 Convolution, stride 1	3, 256, 256	7808
3x3 Convolution, stride 2	32, 256, 256	18496
3x3 Convolution, stride 2	64, 128, 128	73856
Residual Block (5x):		
3x3 Convolution, stride 1	128, 64, 64	147584
3x3 Convolution, stride 1	128, 64, 64	147584
Deconvolution:		
3x3 Convolution, stride 1	64, 128, 128	73792
3x3 Convolution, stride 1	32, 256, 256	18464
9x9 Convolution, stride 1	3, 256, 256	7779

3.3 Style-Transfer Network:

Our second approach was slightly different. Here we trained a CNN to apply a specific stylisation on its input image. The architecture we decided on is depicted in Table 1. The overall structure of the model is very similar to the model presented by Johnson et al. [1] but here a convolution layer denotes a convolution with following instance normalisation and ReLU activation afterwards.
The network consists of five residual blocks which themselves consist of two convolution layers (with instance normalisation and ReLU) and a residual connection, the addition of the input and output of the block. In the Deconvolution block we replaced the original transposed convolution layers by upsampling followed by convolution layers (again with instance normalisation and ReLU activation). Specifically, we used padding in each convolution layer, which kept the input shapes consistent along with the output shapes and stride. We used reflective padding with the idea that this would preserve the original shape of the image. In addition we used affine instance normalisation. This affine instance normalisation is very similar to batch normalisation except that it is applied instance-wise. Hence it is calculated as follows:

$$y = \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}} * \gamma + \beta$$

Here x is the input, a single channel of a single instance and not a mini-batch, y is the output, μ_x is the input mean, σ_x^2 is the input variance, γ and β are affine parameters and ϵ is a minimal constant term to ensure that the denominator does not become to small.

3.4 Training

For the training of our style transfer network, we used Adam as the optimisation method and decided on a learning rate of 1e-3, as a lower learning rate did not seem to produce any further performance increases. The batch size for our training was 8 and we trained on the entire dataset, but only for one epoch, because due to the size of the dataset the loss already converged during this epoch.

3.5 Gradient Clipping

During the training of our model, we noticed that after long training phases, when the loss was already very low, there were sudden jumps in the loss. To solve this problem, we introduced gradient clipping to a normalised value of 1.0 (meaning that for the gradient of every single parameter g_x $g_x = \min(1, \frac{1}{||g_x||_2}) * g_x$). This has not only led to no more problems with exploding gradients, but has also greatly improved the general training performance. Thus, it can be observed in Figure 1 that the variant with gradient clipping converges significantly 'smoother' and faster than the variant without.

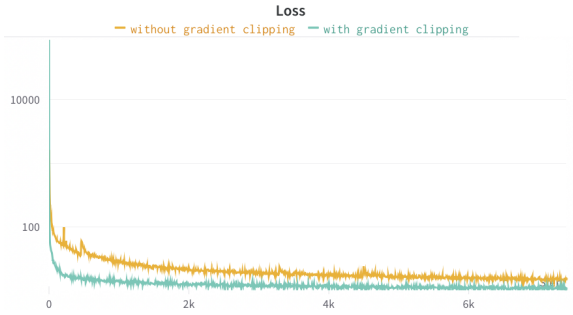

Figure 1: In yellow the training process without and in blue the one with gradient clipping

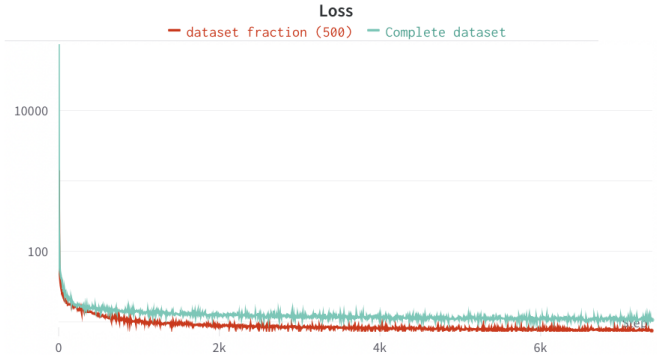

Figure 2: Comparison of the two optimisations, once with reduced data set size and once with original size

3.6 Smaller Data set

We wanted to see how good our model captures and transfers the style and the features of the original images. Therefore we decided to compare it to the same model that was trained on a smaller fraction (500 samples) of the data set in order to see how much better the model could do if it could capture more details of the image. We assume that by reducing the size of the data set, our model sees a less representative part of our whole data set, as our data set is very heterogeneous and contains many different shapes and forms, which is why it is easier for the model to deal with this less representative portion. This effect can be observed in Figure 2. This means if there is a discrepancy between the performance of the differently trained models our model does not have the complexity to generalise as good on the whole data set as on the small data set what means that we could probably increase the performance of our model by increasing its complexity.

4 Evaluation

4.1 Why common evaluation techniques are not applicable

Unfortunately, it is not possible to use conventional evaluation techniques. This is due to the nature of the problem. A style is not a fixed quantity that can be used for calculation. The same applies to the appearance of the images. Only two things can be evaluated, firstly how much the model has succeeded in minimising the loss and secondly whether the output images are visually desirable.

4.2 Influence of different layers for style and content loss

As already described, the style loss is composed of the activations of different layers of the loss CNN. Roughly speaking, the deeper or later the layer is in the network, the greater the style properties held

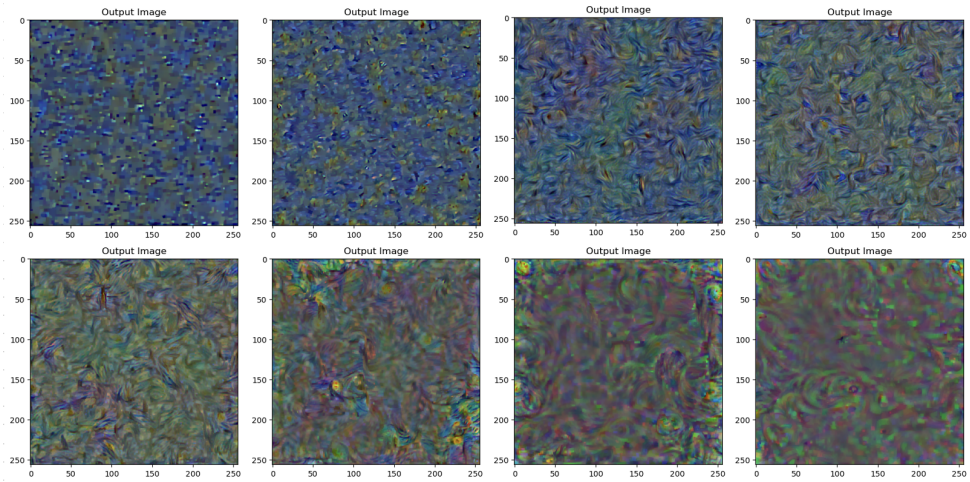

Figure 3: Images minimize the loss for the activations of style image Starry Night by van Gogh (from Left to Right and Top to Bottom in ReLu1-1, ReLu1-2, ReLu2-1, ReLu2-2, ReLu3-1, ReLu3-3, ReLu4-3, ReLu5-3)

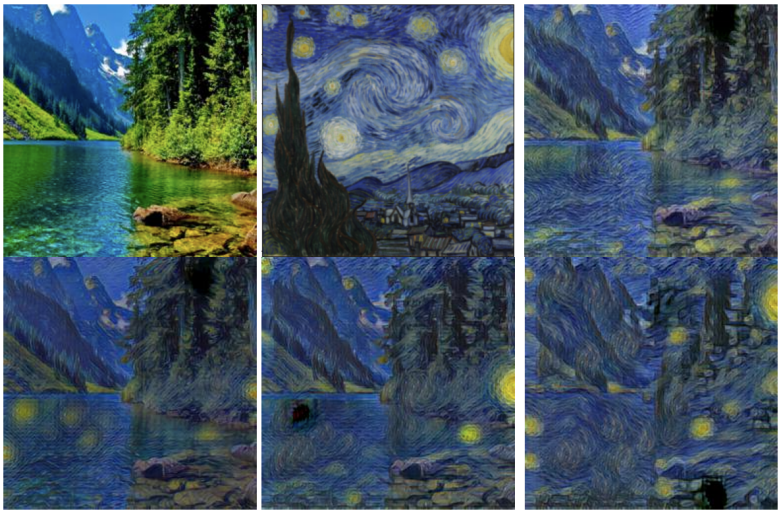

Figure 4: From top to bottom and reft to right: Content Image, Style Image, our result with a $w_{\text{style activations}} = (0.7, 0.3, 0.03, 0.03, 0.03)$, results of the method from Johnson et al. with $\lambda_{\text{style}} = 10$, $\lambda_{\text{style}} = 25$, $\lambda_{\text{style}} = 50$

there. This trend, that deeper layers capture bigger and more complex style elements, can be clearly seen in Figure 3. Roughly speaking, the activations in the first layers focus on colour and small shapes in the image, while the activations in the later layers seem to neglect colour and focus more on larger patterns. To illustrate this the graphs were created with the method presented by Gatys et al. [1]. This method starts at a certain image and optimises it according to the loss function. Therefore our initial image was Gaussian noise, which is why in the later layers, which depend less on the individual values, i.e. colours, clearly show more colourful pattern properties. This observation allowed us to select certain layers we want to include in our loss (note that for all other experiments in this report we used the layers used by Johnson et al. [1]).

Furthermore, the activations of the individual layers can be weighted differently as desired. In addition to this weighting of the individual layers, it is also possible to have the different components

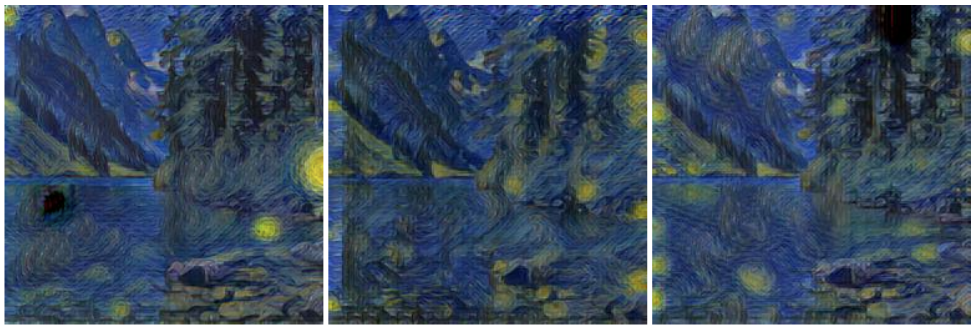

Figure 5: From left to right: The model output when using Instance Normalisation and upsampling, Batch Normalisation and upsampling, Instance Normalisation and Transposed Convolutions

flow into the overall loss to a different extent. An overview of the consequences of the weightings can be found in Figure 4. Here we have used the layers proposed by Johnson et al. [1] for the comparison of the component weightings. To create our own loss, we kept the same layers from Johnson et al. [1] and just decided to compose them differently. See Appendix for further explanations.

4.3 Experiments

In principle, we have kept to the architecture of Johnson et al. [1], but we have nevertheless integrated some optimisations. For example, instead of using the proposed transposed convolutions, we replaced them with upsampling and convolutions. We have also replaced batch normalisation with instance normalisation, because our results 5 show that especially when using natch normalisation the network struggles with the correct transformation at the edges of the image. Our hypothesis is that this is caused by the padding that is needed when applying the Deconvolution part of the network. Learning the edges might be a significantly more difficult task, as there is less information given that can be used to (re-)construct from our hopes were that reflective padding would mitigate this task but seemingly it did not. It has to be noted that the introduction of instance normalisation as described by Ulyanov et al. [7]. seems to ease the task, maybe by offering a more complex function model, that can be learned as the normalisation is always applied batch-independent, per channel of the instance. Ulyanov et al. [7], on the other hand, assumed that the advantage of instance normalisation arises from being 'contrast-agnostic' of the content image, because the normalisation removes this contrast information.

4.4 Inference

Since the aim of our project was to 'Build a model, that takes a style image and mixes it with a content image'. Demonstrate your results by creating a simple application, which takes a webcam frame and stylises it', we build a simple application (which can be found in src/utils.py and also integrated in the demo.ipynb) that takes a webcam video and applies our model to every frame of it by first resizing and then center cropping the frame. This shown video stream 6 reaches a Rate of 15 Frames per Second when executed via the Metal Performance Shader (mps) on a MacBook Pro and a Rate of 5 FPS when executed on the CPU.

4.5 Dataset

Our goal is to end up running the style transfer network on the webcam of a laptop. Since this webcam will take pictures of faces, we decided to train our Style Transfer Network again on a data set containing only faces for comparison. For this we used the CelebA data set [9]. The result can be compared in Figure 7. As it did not seem to provide any visual improvement we decided to stay with our the COCO data set.

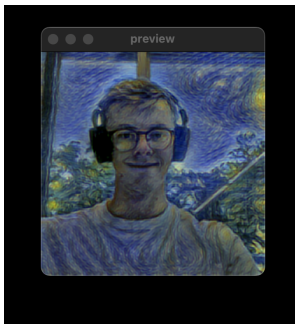

Figure 6: The final application

Figure 7: From left to right: model trained on the CelebA dataset, model trained on COCO and the model trained on COCO with the modified loss function

5 Discussion

5.1 Reflection

We have succeeded in training an appropriate CNN that is able to transfer a style to a content image. However, the network has to be trained for a specific style. Our network (see smaller dataset) also had problems generalising on the large dataset. Here, one could try to create optimisations of the model that have greater power. One approach would be to increase the number of residual blocks, but this would mean that the inference would become more expensive. However, this would not be desirable for us because we do not have the hardware for performing inference on such models efficiently enough (only laptops with webcam).
The arising complexity-performance dilemma probably means that if the same approach is used, completely different model architectures would have to be used.

5.2 Future work

After analysing how we can optimise the known style transfer network and its training, we will now discuss what other ideas could be explored.
Our next step would be to investigate an extension to instance normalisation called adaptive instance normalisation [8] proposed by Karras et al. [8]. This approach assumes that the affine parameters of instance normalisation ensure that a certain style is learned. The extension is that the affine parameters γ and β are simply calculated from the given style image, so that the values that are normalised are adjusted to the same mean and variance as those of the style. This method of Adaptive Instance Normalisation would then allow to use the same decoder for transforming different styles on content images, by simply using the specific style for performing Adaptive Instance Normalisation. This style invariant approach produces impressive results without having any significant performance decreases still allowing a fast inference on less powerful hardware.

Another idea would be to replace Gram-matrix as a form of style loss by a different measure. Therefore one could explore different loss functions, for example the loss functions MRF loss, adversarial loss, histogram loss, CORAL loss, MMD loss and distance between channel-wise mean

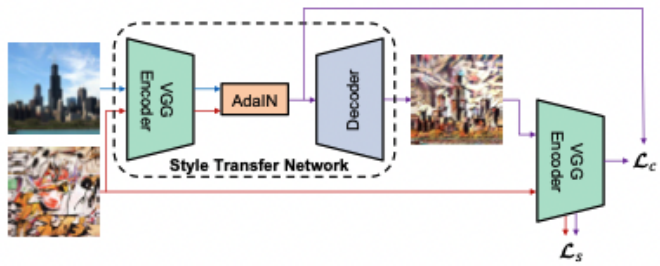

Figure 8: Adaptive Instance Normalisation: The Transformation into a feature space is done with a fixed encoder and then Adaptive Instance Normalization to encode the used style (Figure from [8])

and variance, as mentioned by Karras et al. [8]. This could help to give an interesting insight into what really defines a style 'mathematically'.

5.3 Conclusion

In the end, we can say that we have succeeded in training a model that performs reasonably well and at the same time is efficient enough to be executed on low-power hardware in real-time. We have thus achieved the project goal 'Build a model, that takes a style image and mixes it with a content image'. Nevertheless, the extension with adaptive instance normalisation would be a very interesting next step, which we will probably test in the future.

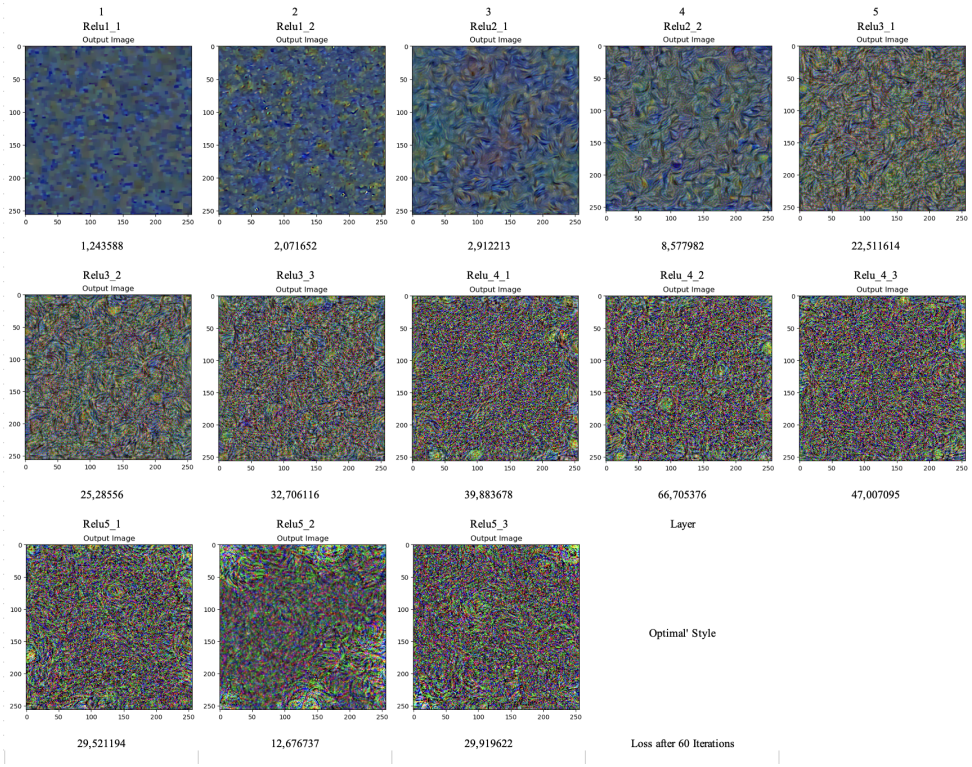

Figure 9: Overview of the influence of different layers for the style loss. As the losses widely vary in the size it stands to reason that a more desirable outcome can be achieved by assigning them different weights

6 Appendix

The weighting of different layers the style loss

As described above, we have once used the weights $(0.7, 0.3, 0.03, 0.03, 0.03)$ to weight the different activations, ReLU1_2, ReLU2_2, ReLU3_3, ReLU4_3, as used for the style loss, against each other. We got this weighting by running the Gatys algorithm on our style image using only a single layer for style loss. Then we compared the loss that was still present after 60 integrations and used this to obtain the individual factors for weighting the layers.

References

[1] Johnson, J. & Alahi, A. & Fei-Fei, L. (2016) Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In https://cs.stanford.edu/people/jcjohns/papers/eccv16/JohnsonECCV16.pdf

[2] Gatys, L. & Ecker, A. & Bethge, M. (2015) A Neural Algorithm of Artistic Style. In https://arxiv.org/pdf/1508.06576.pdf

[3] Microsoft COCO (2014) Common Objects in Context. In https://arxiv.org/abs/1405.0312
Download here: https://cocodataset.org/#download

[4] Rothe, R. & Timofte, R. & Van Gool, L. (2015) IMDB-WIKI – 500k+ face images with age and gender labels. In https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/

[5] Simonyan, K. & Zisserman, A. (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition. In https://arxiv.org/abs/1409.1556

[6] PyTorch Team (2019) PyTorch: An Imperative Style, High-Performance Deep Learning Library. In http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[7] Ulyanov, D. & Vedaldi, A. & Lempitsky, V. (2017) Instance Normalization: The Missing Ingredient for Fast Stylization. In https://arxiv.org/pdf/1607.08022.pdf

[8] Karras, T. & Laine,S. & Aila, T. (2019) A Style-Based Generator Architecture for Generative Adversarial Networks. In https://arxiv.org/pdf/1812.04948.pdf

[9] Liu, Z. & Luo, P. & Wang, X. & and Tang, X. & (2015) Deep Learning Face Attributes in the Wild http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[10] Horn, R. & Johnson, C. (2013). Matrix Analysis (2nd ed.) p.441, Theorem 7.2.10