Image Style Transfer using Convolutional Neural Networks

Simon Hessner - Supervised by Saquib Sarfraz

Computer Vision for Human-Computer Interaction Lab
Karlsruhe Institute of Technology

Abstract

Recently there has been lots of progress in the field of image style transfer, a process which aims at redrawing an image in the style of another image. Gatys et al. proposed the first approach using Convolutional Neural Networks, but their iterative algorithm is not efficient. Many others followed and improved their approach in terms of speed. This report gives an overview over various approaches for image style transfer and its extension to videos. Furthermore, their advantages and limitations will be discussed.

1. Introduction

Image style transfer is the process of redrawing an image in the style of another image while preserving the semantic content of the original image. For example, an user could automatically create a portrait of himself in the style of Vincent van Gogh. The first work that used Convolutional Neural Networks (CNNs) for image style transfer was published by Gatys et al. [1] in 2016. Their *Algorithm of Neural Style* uses an iterative approach that can transfer arbitrary styles to arbitrary content images. The iterative approach is quite slow and therefore not usable in real-time applications. Johnson et al. [2] propose a feed-forward network that solves the performance issue at the cost of losing the flexibility to use arbitrary styles with one model.

There seems to be a three-way tradeoff between speed, quality and flexibility [3, 4] in image style transfer. This report presents approaches by Huang and Belongie [3] and Li et al. [4] that aim at being both fast and flexible while still producing good results compared to the previously mentioned algorithms.

Video style transfer is an extension of image style transfer to videos where every frame is transferred to the same style. The easiest approach is to apply an image style transfer algorithm to every frame, but this produces flicker artifacts. Ruder et al. [5] present an approach that eliminates these artifacts by incorporating optical flow. However, their procedure is very slow. This work presents a paper by Huang et al. [6] that proposes a stable video style transfer algorithm.

Outline. Most approaches presented in this report share some foundations which are described in section 2. Related work on the field of image style transfer is discussed in section 3. A selection of state-of-the art approaches for both image and video style transfer is presented in section 4. Finally, section 5 compares these approaches and section 6 points out some issues that need to be further researched.

2. Foundations

In order to transfer the style of an image to another image, the style and content of these images and the output image must be represented in a suitable way. Gatys et al. [1] introduce representations for both style and content that can be extracted using CNNs trained on the classification task. Most papers in the field of image style transfer use these or similar representations.

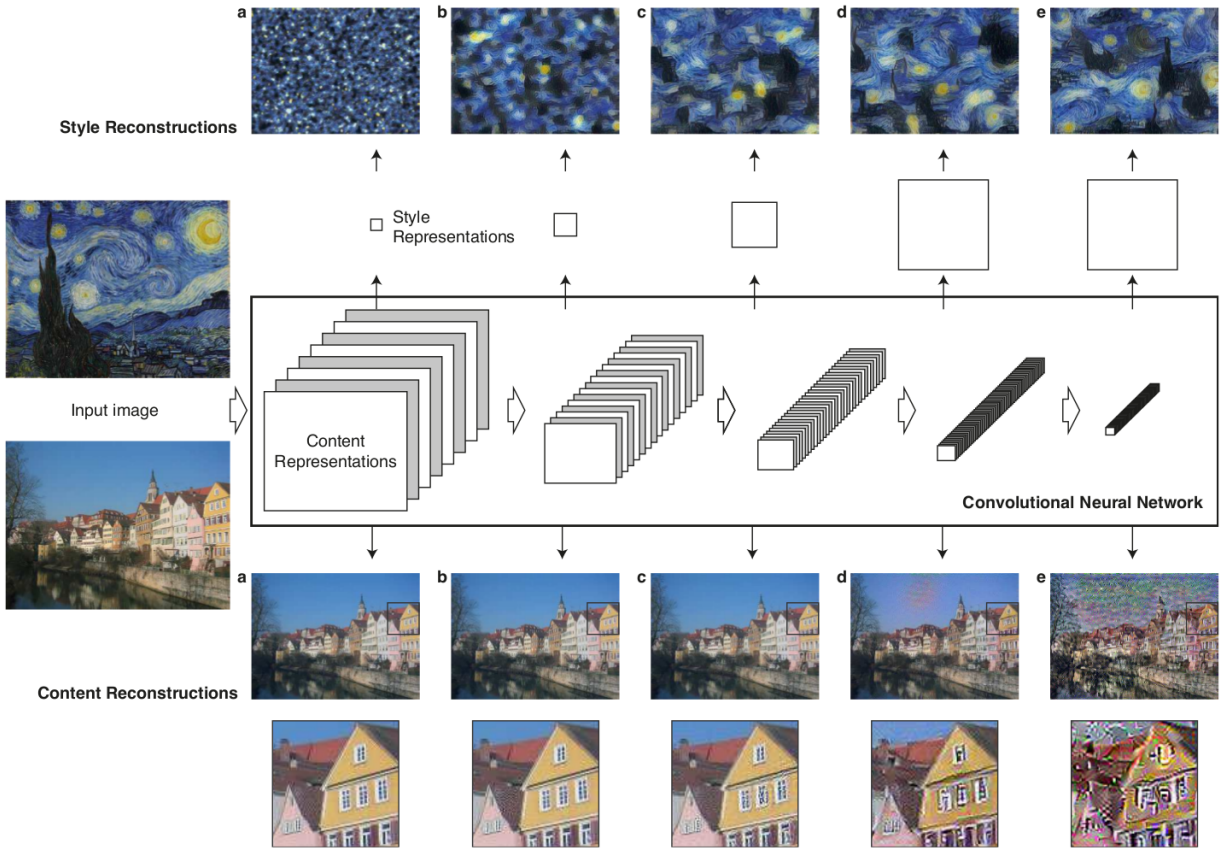


Figure 1: Content and style reconstructions from feature maps of different CNN layers. As the receptive field grows, the details in the content representation get lost while the semantic content is kept (bottom images). The style can be reconstructed using the correlations of feature maps (top images). Picture a shows the reconstruction from the first feature map, b from the first two feature maps, and so on. *(Image from [1])*

2.1. CNN as feature extractor

CNNs are neural networks that can be used for image classification. They are trained on large labeled datasets and learn both feature extraction and classification in an end-to-end manner. During training they implicitly learn how to represent the content of an image internally. This representation is generated by convolving the output of the previous layer with filters (which are learned during training). The filter responses are stored in feature maps. Due to the pooling operations that follow the convolutions, the receptive field of the neurons in the feature maps increases in every layer. While lower layers filter local structures like edges, lines, curves, colors and textures, higher layers have learned filters that capture more complex structures like parts of persons, cats, wheels or houses.

The feature maps of a layer l containing N_l different features and $M_l = w_l * h_l$ activation values with w_l being the width and h_l the height of the feature map can be stored in a matrix $F^l \in \mathbb{R}^{N_l * M_l}$. These feature maps can be used to represent content and style of an image [1] as described below.

2.2. Content representation

The activation values of feature maps in higher layers can be used to roughly reconstruct the content of an image without exactly reconstructing local structures [1]. Figure 2.1 (bottom row (a)-(e)) shows the reconstruction of the content input image using different CNN layers. Image (a) is reconstructed from features of the first feature map and therefore most details are retained. However, image (e) is reconstructed from a high layer and because of that has lost many details of the original image (like exact colors or local structures) but the semantic content (river, sky, houses) is still well recognizable. Hence, feature maps from a high layer represent the image content.

Reconstruction from different feature maps is achieved by optimizing a white-noise image using a loss function that is minimized with respect to the pixel values of the image. This is the same procedure as training a neural network using backpropagation, but instead of deriving the loss function by the network's weights, it is derived by the pixels. The weights of the network are not changed when reconstructing an image.

The content loss in layer l is defined as

$$L_{content}(c, x, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - C_{ij}^l)^2 \tag{1}$$

where c is the original content image (with feature map F^l), x the transferred style image (with feature map C^l) and l the layer where the loss should be calculated.

2.3. Style representation

Correlations of features in a feature map can be used to describe the style of an image. Imagine features for different kinds of brush strokes and a painter who uses a certain combination of them extensively. In this case, the spatial correlation between these brush stroke features would be high. To describe correlations between different features in a feature map F_l, Gram matrices are used:

$$G_{ij}^l = \sum_{k=0}^{M_l} F_{ik}^l * F_{jk}^l \tag{2}$$

The Gram matrix G^l has a high value at position (i, j) if the features i and j are strongly correlated across spatial dimensions and a low value if they are weakly correlated. Hence, the whole matrix G^l describes the degree of correlation of all features in a layer l and can be used to define a style loss in one layer:

$$E_l(s, x) = \frac{1}{4 * N_l^2 * M_l^2} \sum_{i,j} (G_{ij}^l - S_{ij}^l)^2 \tag{3}$$

where G is the Gram matrix of the stylized image x and S is the Gram matrix of the style image s. Since the style can be better captured by using multiple layers, the total style loss is the weighted sum of the style losses in different layers L:

$$L_{style}(s, x) = \sum_{l \in L} w_l * E_l(s, x) \tag{4}$$

where the weights w_l and the set of layers L are hyperparameters.

Figure 2.1 (top row) shows style reconstructions using different layers of the CNN. Note that (b)-(e) are reconstructions from the lowest layer up to a certain high layer while (a) is only reconstructed from the lowest layer. Apparently local texture patterns from the style image are reproduced when only low layers are used for the reconstruction. When adding more high layers, more and more global structures are replicated.

3. Related Work

There is plenty of related work in the field of image and video style transfer. This section covers the most important papers that had great influence on state-of-the art solutions.

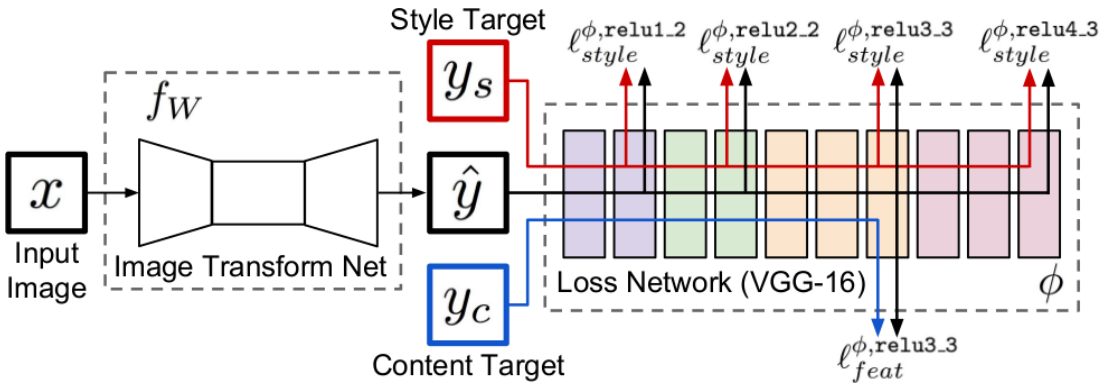


Figure 2: System overview: The image transform net is trained to produce stylized images using a loss network. The system architecture can be used for both image style transfer and image super-resolution. In case of image style transfer, they set $y_c = x$. Input x, y_s and output $\hat{y}$ are RGB images. *(Image from [2])*

3.1. Optimization-based approach

As mentioned in section 1, Gatys et al. [1] use an iterative approach. Their algorithm starts with a random image x and tries to optimize this image iteratively in order to combine the content c with the style s. Their optimization objective is to minimize an error function that consists of a content loss (Equation 1) and a style loss (Equation 4). Both losses are computed based on CNN features (Section 2.1). The total loss is

$$L_{total}(c, s, x) = \alpha L_{content}(c, x) + \beta L_{style}(s, x) \tag{5}$$

where α and β are hyperparameters that can be used to define a trade-off between style and content.

Combining these two loss functions makes image style transfer possible. In case only the content loss was used, the original content image would be reconstructed to a certain degree. If just the style loss was used, the texture of the style image would be replicated (compare Figure 2.1).

They use the pre-trained VGG-19 network [7] to extract the features of the images, but other CNNs trained on the classification task could be used as well.

Their algorithm requires up to 500 iterations to produce visually appealing images where every iteration consists of a forward and a backward pass. This process can take up to 16 seconds for 256x256px images and 215 seconds for 1024x1024px images on a modern GPU, which is too slow for many applications.

3.2. Feed-forward approach

The follow-up work by Johnson et al. [2] addresses the efficiency problem by training a feed-forward network that approximates the solution produced by the optimization-based approach by Gatys et al. [1]. In other words, they try to move the computational effort from test time (applying style transfer) to train time. Thus, once a model for the feed-forward network is trained, image style transfer can be done much faster compared to optimizing an image iteratively. They refer to their feed-forward network as Image Transform Net.

Their system is shown in Figure 2. The image transform net f_W is a deep convolutional residual network that takes an image x as input and returns an output image $\hat{y}$. As the network is fully convolutional, it can be applied to images of arbitrary sizes. They train the network using loss functions that are themselves computed using a CNN (Section 2). These loss functions extract features from the VGG-16 network and compute the style and content loss similar to the optimization-based approach (Section 3.1). Since they use the VGG-16 network instead of the VGG-19 network used by the optimization-based approach [1], they extract the content and style features from different layers than Gatys et al. [1]. Besides the content and style loss, their total loss function also contains a total variation regularization loss that reduces noise in the output image [8]. Their feed-forward network

f_W is trained on one specific given style image y_s and 80,000 images from MS-COCO [9] used as content images $y_c = x$.

While the feed-forward approach is up to three orders of magnitudes faster than the optimization-based approach, it introduces new problems. In contrast to the approach by Gatys et al. [1] it can only transfer pictures into one style, the style it was trained for. Thus, a new model has to be trained for every style which is not achievable for applications where a user wants to get fast results for arbitrary styles. Section 4 covers new algorithms that solve both the speed problem and can be used for any styles without the need for training a per-style model.

3.3. Video Style Transfer

Image style transfer can be extended to videos by applying an image style transfer algorithm to each single frame. However, many image style transfer methods only find a local minimum because of the non-convex loss function. This causes flickering in areas with motion between two frames.

Ruder et al. [5] present an optimization-based approach for videos which introduces temporal consistency to remove these flicker artifacts. Temporal consistency is achieved by computing the optical flow on the content video and penalizing differently stylized pixels in two consecutive frames that correspond to each other. The optical flow is computed on-the-fly using a neural network.

Their content and style loss functions are the same as in the optimization-based image style transfer approach by Gatys et al. [1]. These two losses are combined with the temporal loss to get the total loss which is used to generate the stylized video frame by frame. In their paper they state that multiple passes over a video are required in order to get pleasant results on image borders.

Since they rely on multiple passes and on-the-fly optical flow computation, their approach is extremely slow. According to [6] it can take up to three minutes to process a single frame even when the optical flow is pre-computed. Although they achieve good results with highly reduced flicker artifacts, the time needed to produce a stylized video is too high for real-world applications. A faster method is presented in section 4.3.

4. Approaches

This section covers two state-of-the-art solutions for image style transfer and one solution for video style transfer.

4.1. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

Huang and Belongie [3] present an approach that can transfer style in real-time (in contrast to Gatys et al. [1]) without needing to train a per-style model (in contrast to Johnson et al. [2]). Their general approach is to use an encoder-decoder network that takes the content and style images as input and returns the transferred style image in one feed-forward pass. Figure 3 shows their architecture.

Adaptive Instance Normalization. They insert a new layer called Adaptive Instance Normalization (AdaIN) between the encoder and the decoder. Its task is to perform style transfer on the feature maps of a specific layer. AdaIN aligns the features (received from the encoder) of the content image c with those of the style image s. The output of the AdaIN layer are the content features with the mean and variance of the style features.

$$AdaIN(c, s) = \sigma(s) \left(\frac{c - \mu(c)}{\sigma(c)} \right) + \mu(s) \tag{6}$$

$\sigma(\cdot)$ and $\mu(\cdot)$ are computed channel-wise across spatial dimensions: $\mu(x)$ is the channel-wise mean of x and $\sigma(x)$ is the channel-wise variance of x.
In contrast to batch normalization [10] and instance normalization [11] there are no affine parameters γ (scale), β (shift) that have to be learned from the training data. Instead, $\gamma = \sigma(s)$ and $\beta = \mu(s)$ can be calculated during test time. This is why it is called *Adaptive* Instance Normalization. Instance

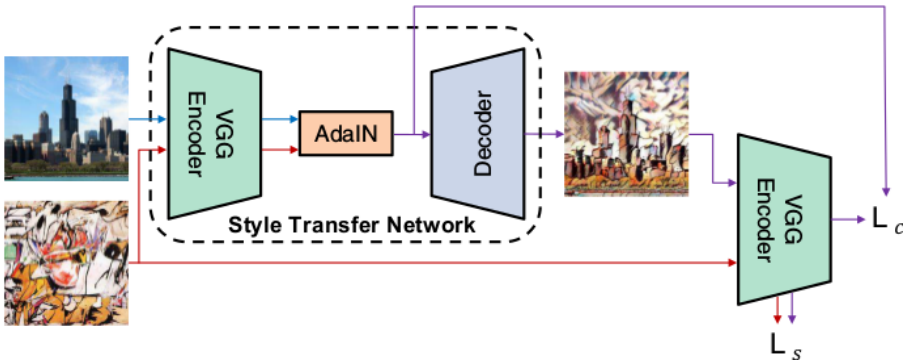


Figure 3: The encoder consists of the VGG-19 layers up to *relu4_1*. The newly introduced AdaIN layer transforms the encoded features of the content images to match the mean and variance of the style image. The decoder transforms the AdaIN result back to RGB space. *(Image from [3])*

normalization can be interpreted as style normalization because the affine parameters of the normalization can completely change the style of the output image [3]. The AdaIN result is passed to the decoder in order to produce the final stylized image.

Encoder and decoder. The encoder is the pre-trained VGG-19 network [7] up to layer *relu4_1*. It simply extracts the features from the input image.

The decoder transforms the features (in this case obtained from AdaIN) back to RGB space. It is the only part of their architecture that needs to be trained. The decoder is trained on 80,000 style images from WikiArt [12] and 80,000 content images from MS-COCO [9]. They train the decoder to minimize the loss function

$$L = L_c + \lambda L_s \tag{7}$$

where L_c is the content loss and L_s is the style loss. In difference to section 2, the content loss is defined as

$$L_c = \|f(g(t)) - t\|_2 \tag{8}$$

with $f(\cdot)$ being the encoder, $g(\cdot)$ the decoder and $t = AdaIN(f(c), f(s))$ the AdaIN output. Equation 8 is minimal if $f(\cdot)$ is the inverse of $g(\cdot)$.

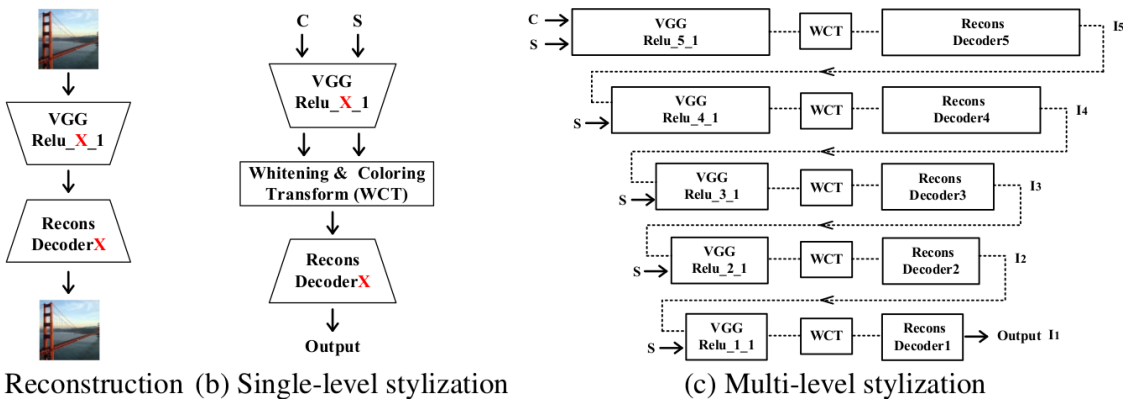
In contrast to Gatys et al. [1] and Johnson et al. [2] this paper does not use the Gram matrix to define the style loss. Instead, they define the style loss using the differences of channel-wise mean and variance of the encoded style and output images. It was shown that various different statistics on the style image can be used to capture its style [13]. Among these statistics are Gram matrices and mean/variance. The style loss is defined as

$$L_s = \sum_{i=1}^{L} \|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 + \sum_{i=1}^{L} \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2 \tag{9}$$

Similar to Gatys et al. [1] they use one layer for the content loss L_c and multiple layers L for the style loss L_s. $\phi_i(x)$ are the encoded features of image x.

4.2. Universal Style Transfer via Feature Transforms

Li et al. [4] present a feed-forward approach using so-called whitening and coloring transforms. These transformations aim at matching feature covariances similar to the Gram matrix optimization of Gatys et al. [1]. However, they do not need to iterate but can transform the features in one forward pass. The basic idea is similar to the AdaIN approach (section 4.1), however they don't require training on any style images and they use multiple layers to transfer the style.



(a) Reconstruction (b) Single-level stylization (c) Multi-level stylization

Figure 4: (a) A decoder is trained for Relu_X_1 features of the VGG-19 network. (b) A Whitening & Coloring Transform (WCT) is applied to the encoded features before decoding them. The WCT transfers style S to the content image C in layer Relu_X_1. (c) WCT is applied to multiple layers (Relu_5_1 to Relu_1_1) successively. *(Image from [4])*

Figure 4 (a) shows an encoder that applies the VGG-19 network up to a given layer Relu_X_1 to the image. They train a decoder for features of different layers. The decoder should reconstruct the original image from the encoded features as good as possible (compare Figure 2.1). Figure 4 (b) shows their basic approach to transfer style. First, both content image C and style image S are encoded with a specific encoder. The *Whitening & Coloring Transform* (WCT) then transforms the content features to match the covariance matrix and mean of the style features. Figure 4 (c) shows the architecture that applies the WCT to multiple layers. The output of the decoder is used as content input for the next encoder. This process transfers the style layer by layer, coarse-to-fine as the WCT is first applied on a high layer and then to lower layers (compare Figure 2.1).

Whitening transform. The purpose of the whitening transform is to remove style information from the content features. It transforms a vector that is drawn from a distribution with known covariance matrix and mean to a centered vector that has the identity matrix as covariance matrix.

The input of the whitening transform are the feature maps $f_c = C^l$ of the content image in layer l. First, the channel-wise mean m_c is subtracted from C^l in order to center it. The next step is to decorrelate and norm the dimensions of the centered vector.

$$\hat{f}_c = E_c D_c^{-\frac{1}{2}} E_c^T f_c \tag{10}$$

where D_c is a diagonal matrix with the eigenvalues of the covariance matrix $f_c f_c^T$ and E_c is the matrix of eigenvectors corresponding to the eigenvalues. In other words, $E_c D_c E_c^T$ is the eigendecomposition of the covariance matrix.

Equation 10 first decorrelates the dimensions of f_c and then normalizes them in order to have unit variance. Since the whitening transform removes style information from the content features, it can be interpreted as extracting the content information of a feature map. The whitened content features are then fed into the coloring transform in order to add the style of the style image to them.

Coloring transform. The input to the coloring transform are the whitened content features $\hat{f}_c$ and the style features $f_s = S^l$ in layer l. First, the channel-wise mean m_s is subtracted from f_s. The result is then used to calculate the eigenvalues of the covariance matrix $f_s f_s^T$ which are stored in a diagonal matrix D_s. Furthermore, the corresponding eigenvectors are stored in the orthogonal matrix E_s. The coloring transform is defined as:

$$\hat{f}_{cs} = E_s D_s^{\frac{1}{2}} E_s^T \hat{f}_c + m_s \tag{11}$$

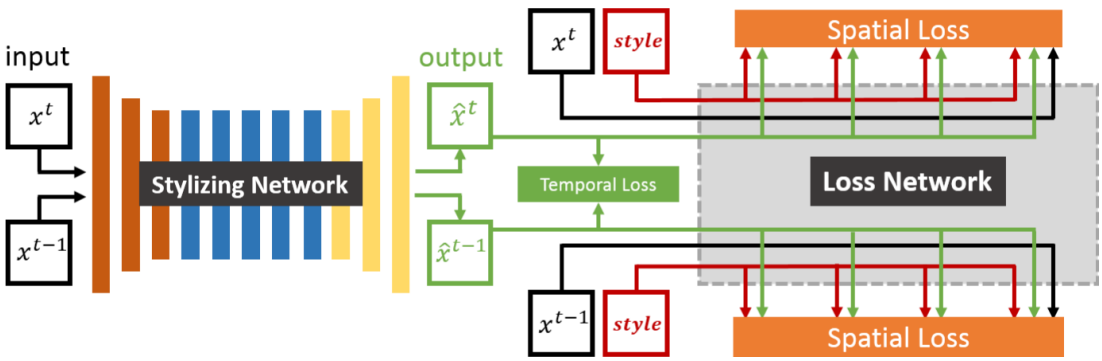


Figure 5: System overview. A stylizing network is trained using a hybrid loss consisting of a temporal loss and a spatial loss. *(Image from [6])*

It basically transforms $\hat{f}_c$ to have the mean and correlations of f_s. In other words, after applying both the whitening and coloring transforms, the following relationship is true: $\hat{f}_{cs}\hat{f}_{cs}^T = f_s f_s^T$. Thus, $\hat{f}_{cs}$ contains the content features with the transferred style in layer l.

The output of the WCT is then fed into the decoder for that layer in order to transform the features back to RGB space. To produce good results they apply the WCT not only to one layer but to multiple layers. The decoded WCT result is the content input for the next layer.

Decoder. Their decoders are basically symmetric to the VGG network up to the layer that should be decoded. They are trained on the MS-COCO dataset [9]. The loss function used to train a decoder is defined as:

$$L = \|I_o - I_i\|_2^2 + \lambda \|\phi(I_o) - \phi(I_i)\|_2^2 \tag{12}$$

where λ serves as a parameter that controls the influence of the two parts of the loss function and I_i, I_o are the input image and reconstructed image. $\phi(\cdot)$ is the encoder that extracts the features in the specific layer. So the loss combines both the per-pixel loss between the original image and its reconstruction and the feature loss. They train a decoder for every layer from Relu_1_1 to Relu_5_1.

4.3. Real-Time Neural Style Transfer for Videos

The video style transfer approach by Ruder et al. [5] gives visually pleasant results but is too slow for real-time applications. In order to solve the speed problem, Huang et al. [6] extend the idea of Johnson et al. [2] to videos. They train a feed-forward network that can produce stylized videos in real-time. The system architecture is shown in Figure 5.
Their stylizing network is a deep residual convolutional network, similar to [2], but uses less channels and therefore is even faster. It is trained with a hybrid loss that consists of a spatial loss and a temporal loss:

$$L_{hybrid} = \lambda L_{temporal}(\hat{x}^t, \hat{x}^{t-1}) + \sum_{i=t-1}^{t} L_{spatial}(x^i, \hat{x}^i, s) \tag{13}$$

The spatial loss comprises the style and content losses that [1, 2, 5] use (see Section 2 for the definitions). It is obtained using the VGG-19 network [7] as loss network and calculated for two frames x^{t-1}, x^t independently.

The temporal loss is guided by pre-computed optical flows and ensures that moving objects look the same in consecutive frames. The optical flow is computed on the content frame sequence x. During training, two frames x^{t-1} and x^t are fed into the network, producing the stylized frames $\hat{x}^{t-1}$ and $\hat{x}^t$. These output images are compared pixel-wise, respecting the optical flow correspondences of

the content video. In other words, a pixel at position (x, y) in frame $t-1$ is compared to the pixel $f(x, y)$ in frame t with $f(x, y)$ describing the new position of (x, y) according to the optical flow:

$$L_{temporal}(\hat{x}^t, \hat{x}^{t-1}) = \frac{1}{D} \sum_{k=1}^{D} c_k (\hat{x}_k^t - f(\hat{k}^{t-1}))^2 \tag{14}$$

$D = H * W * C$ is the number of channel-wise pixels to compare (H = height, W = width, $C = 3$ (R, G, B)). c_k is 0 in occluded regions and at motion boundaries and 1 otherwise. This ensures that the loss is only calculated for regions that are visible in both frames.

They train their stylizing network on a given style image and about 40.000 content frames of 91 videos from Videvo [14].

During test time, the stylizing network only has to be applied to one frame at a time. Moreover, the optical flow does not have to be computed during test time as the network learns to encode temporal consistency during training.

5. Results & Discussion

The approaches presented in section 3 and 4 are discussed here.

5.1. Image Style Transfer

Table 5.1 gives an overview about the image style transfer approaches described in this work. The ratings in the table are discussed below.

Approach	Training	Speed	Flexibility	Quality
Iterative optimization Gatys et al. [1]	No training required	- -	++	++
Feed-forward network Johnson et al. [2]	Per-style model	++	- -	+
Adaptive Instance Normalization (AdaIN) Huang and Belongie [3]	Decoder (Style and content)	+	++	+
Whitening & Coloring Transform (WCT) Li et al. [4]	Decoder (Content only)	0	++	++

Table 1: Comparison of various approaches. Speed: How fast can style be transferred? Flexibility: How many styles can be used with one model? Quality: Subjective visual quality of the result images based on a user study [4] and statements in the other papers. Rating: ++ = best, + = good, 0 = neutral, - = slightly bad, - - = worst

Training. Gatys et al. [1] don't need to train any models as they use a pre-trained VGG network to evaluate their loss function and optimize the result image. Johnson et al. [2] train a feed-forward network for every single style image. Huang and Belongie [3] train their decoder on a large set of style and content images and therefore generalize well. Li et al. [4] train their decoder only on content images, thus their decoder does not need to know anything about styles and their approach is generalizing even better.

Speed. Table 5.1 shows quantitative results in terms of speed. Since Gatys et al. [1] do not provide speed measurements in their paper, the table lists the time given by the other papers. They use the optimization-based approach as a baseline and calculate the speed-up based on this value. All authors use Titan X GPUs and 256x256px images. The different timing results could stem from different hyperparameters and different re-implementations of the baseline algorithm.

Approach	Time/image	Img/sec	Speed-up	Rating
Gatys et al. [1] (iterative)	14.19 [3] 15.86 [2] 21.2 [4]	0.07	1	- -
Li et al. [4] (WCT)	0.83	1.2	25	0
Huang and Belongie [3] (AdaIN)	0.027	37	525	+
Johnson et al. [2] (feed-forward)	0.015	66	1057	++

Table 2: Time (seconds) required to produce a 256x256px stylized image using different approaches. The values are taken from the respective papers. Since every paper states different values for the baseline [1], the real speed-up can slightly differ. Therefore the speed-up values should be understood as orders of magnitude rather than exact values. The rating corresponds to the speed column in Table 5.1

Gatys et al. [1] is the slowest approach as it requires many iterations to produce good results. All other approaches [2, 4, 3] require only one forward pass to produce the stylized image. Li et al. [4] (1.2 images per second) is 25 times faster than the baseline but might still be too slow for real-time applications. Johnson et al. [2] (66 images per second) and Huang and Belongie [3] (37 images per second) are fast enough for real-time applications.

Flexibility. The approaches of [1, 3, 4] are all very flexible in terms of how many styles can be processed with one model. They either do not require any training at all [1] or only have to be trained once and can generalize on arbitrary styles [3, 4]. In contrast to these solutions, the feed-forward network of Johnson et al. [2] is very inflexible as it is bound to a specific style image and has to be re-trained for every new style. It furthermore has to be re-trained if the hyperparameters for the content-style tradeoff should be modified for the same style. Since training is time-consuming and the optimal hyperparameters differ between different styles, this is a big disadvantage of this approach.

Quality. As there is no ground truth for the task of image style transfer and style is a very subjective concept, it is not possible to automatically compare and rate these approaches in terms of quality. Huang and Belongie [3] claim that their results are comparable to Gatys et al. [1] in most cases but sightly worse in some other cases. Li et al. [4] state that the AdaIN layer of Huang and Belongie [3] is a sub-optimal approximation of their WCT and therefore produces less appealing results. Since the AdaIN operation is only executed on features of one specific layer while the WCT is applied to a sequence of layers, the results of Li et al. [4] are somewhat more appealing than those of Huang and Belongie [3].

The WCT paper [4] compares the covariance matrices of different approaches ([1, 3, 4]) by calculating the difference between the covariance of the style image and the result image in different layers. They find that their result has the lowest difference and thus best matches the desired style. They also conduct a user study where they ask 80 persons to rate the results from the previously mentioned approaches. The user study reveals that their results are rated as the best in 30% of the cases while the second-placed (AdaIN [3]) is only rated as the best in 24% of the cases. The results of Gatys et al. [1] were rated best in 16.4% of the cases. Thus, most users prefer the results of Li et al. [4] for the chosen examples which comprise the combinations of 5 content and 30 style images. Unfortunately, they did not include the feed-forward approach [2] in their study, so its quality rating is based on subjective ratings of the paper authors. Furthermore, although the results of Gatys et al. [1] are only ranked on place 3 in the user study of Li et al. [4], other papers [3] state that they are slightly better. To conclude with, rating the quality of image style transfer results is not objectively possible, so these ratings should be treated with care.

Conclusion. The results shown in table 5.1 suggest that there is a three-way tradeoff between speed, flexibility and quality. None of the discussed approaches is both fast, usable with arbitrary styles and producing optimal perceived quality. Yet it's still impressive how good the results in this young area of research are. More research will have to be done in order to find out if the methods can be improved to resolve the tradeoff.

5.2. Video Style Transfer

Optimization-based approach. The first algorithm for video style transfer using CNNs was presented by Ruder et al. [5]. It produces visually appealing results without flicker artifacts and is very flexible. It does not need to be trained since it uses pre-trained networks to calculate content, style and temporal loss. The downside of this approach is the vast amount of time required to produce a single stylized frame. In short, it is very flexible, produces good results but is very slow.

Feed-forward network. Huang et al. [6] came up with a solution to the performance issue. Their approach follows the same idea as Johnson et al. [2] who speed up the optimization-based image style transfer algorithm [1] by training a feed-forward network. Their network needs 0.041 seconds for 1024x436px per frame, which is equivalent to 24 frames per second. This is over 4,000 times faster than the optimization-based video style transfer method.

Table 5.2 shows the temporal error different methods produce on various videos. The temporal error of a video is defined as the square-root of the sum of temporal errors between two consecutive frames (based on equation 14). It is high if a lot of corresponding pixels look different in consecutive stylized frames.

Approach	Video 1	Video 2	Video 3	Video 4	Video 5	Average	Reduction
Johnson et al. [2]	0.0770	0.0926	0.0517	0.0789	0.0872	0.07748	0%
Ruder et al. [5]	0.0252	0.0512	0.0195	0.0407	0.0361	0.03454	55%
Huang et al. [6]	0.0439	0.0675	0.0304	0.0553	0.0513	0.04980	35%

Table 3: Temporal errors of the approaches on five videos together with their average error. The last column denotes how much the error is reduced compared to applying image style transfer [2] to every frame. (Values from [6])

The feed-forward image style transfer method [2] applied frame-wise serves as baseline. Compared to this, the optimization-based approach reduces the error by 55% and the feed-forward approach for videos reduces the error by 35%.

The qualitative results of this method can compete with those of the optimization-based approach while being much faster. However, this comes at the cost of losing the flexibility to use arbitrary styles. The stylizing network also has to be trained for one specific style which takes about 92 hours with a NVIDIA Tesla K80 GPU.

Conclusion. Although the feed-forward approach is bound to a specific style, it might be worth training a new model for every style depending on the video that should be converted. If the video is several minutes long, the optimization-based approach might take longer than just training a new feed-forward network and then applying it to the whole video.

6. Conclusion

Image Style Transfer. In their seminal work, Gatys et al. [1] proposed the first CNN-based image style transfer algorithm. However, due to its optimization-based nature, it suffered from low efficiency. Therefore, Johnson et al. [2] developed a fast feed-forward approach that is much faster, but lacks in flexibility. This means, in contrast to the optimization-based approach, it cannot transfer arbitrary styles but must be re-trained for every style. Huang and Belongie [3] proposed an approach based on an Adaptive Instance Normalization layer (AdaIN) that is both fast and usable for arbitrary styles. However, their qualitative results are slightly behind those of others. Li et al. [4] proposed another approach that is both fast and flexible in terms of usable styles. They apply a so-called whitening and coloring transform to features of the content image which effectively removes the original style and adds the style from the style image. However, while producing slightly better results than Huang and Belongie [3], it is not as fast as their solution.

Video Style Transfer. Ruder et al. [5] extended the approach by Gatys et al. [1] to videos. Their approach produces high-quality results but is extremely slow. Huang et al. [6] solved the efficiency issue by training a fast feed-forward network, similar to Johnson et al. [2].

Outlook. Presently no approach manages to circumvent the three-way tradeoff between speed, flexibility and quality. Although there are approaches that are very fast and flexible [3] they do not achieve as visually appealing results as the slower approaches [1]. More research will need to be done in order to further optimize the approaches.

Furthermore, currently there is no good method besides large user studies to evaluate style transfer results. Compared to tasks like image segmentation, classification or object detection, there is no single ground truth, as style is perceived individually. Nevertheless, metrics besides difference in correlations or variances need to be developed in order to compare different results. This can be an interesting direction of research in the field of style transfer.

7. References

[1] L. A. Gatys, A. S. Ecker, M. Bethge, Image style transfer using convolutional neural networks, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2414–2423.

[2] J. Johnson, A. Alahi, F. Li, Perceptual losses for real-time style transfer and super-resolution, CoRR abs/1603.08155 (2016).

[3] X. Huang, S. Belongie, Arbitrary style transfer in real-time with adaptive instance normalization, CoRR, abs/1703.06868 (2017).

[4] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, M. Yang, Universal style transfer via feature transforms, CoRR abs/1705.08086 (2017).

[5] M. Ruder, A. Dosovitskiy, T. Brox, Artistic style transfer for videos, in: B. Rosenhahn, B. Andres (Eds.), Pattern Recognition, Springer International Publishing, Cham, 2016, pp. 26–36.

[6] H. Huang, H. Wang, W. Luo, L. Ma, W. Jiang, X. Zhu, Z. Li, W. Liu, Real-time neural style transfer for videos, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[7] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR abs/1409.1556 (2014).

[8] A. Mahendran, A. Vedaldi, Understanding deep image representations by inverting them, CoRR abs/1412.0035 (2014).

[9] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft coco: Common objects in context, in: ECCV 2014, pp. 740–755.

[10] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, CoRR abs/1502.03167 (2015).

[11] D. Ulyanov, A. Vedaldi, V. S. Lempitsky, Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis, CoRR abs/1701.02096 (2017).

[12] K. Nichol, Painter by numbers, WikiArt, `https://www.kaggle.com/c/painter-by-numbers/data`, 2016. [Online; accessed 05-February-2018].

[13] Y. Li, N. Wang, J. Liu, X. Hou, Demystifying neural style transfer, CoRR abs/1701.01036 (2017).

[14] Videvo, Vidveo free footage, `http://www.videvo.net`, 2016.