# Coherent rendering of virtual smile previews with fast neural style transfer

Valentin Vasiliu*
Kapanu AG and EPFL
Switzerland

Gábor Sörös†
Kapanu AG and Nokia Bell Labs
Switzerland and Hungary

Figure 1: Our method improves the look of the rendered virtual teeth. Left: original frame. Middle: AR preview with default rendering settings. Right: AR preview after our post-processing.

**ABSTRACT**

Coherent rendering in augmented reality deals with synthesizing virtual content that seamlessly blends in with the real content. Unfortunately, capturing or modeling every real aspect in the virtual rendering process is often unfeasible or too expensive. We present a post-processing method that improves the look of rendered overlays in a dental virtual try-on application. We combine the original frame and the default rendered frame in an autoencoder neural network in order to obtain a more natural output, inspired by artistic style transfer research. Specifically, we apply the original frame as *style* on the rendered frame as *content*, repeating the process with each new pair of frames. Our method requires only a single forward pass, our shallow architecture ensures fast execution, and our internal feedback loop inherently enforces temporal consistency.

**Index Terms:** Computing methodologies—Image manipulation Computing methodologies—Mixed / augmented reality

## 1 INTRODUCTION

### 1.1 Motivation

Augmented reality (AR) technology blends in digital information with the real world. The computer-generated digital content is matched with the reality in space and time in an interactive way. AR has found uses in applications such as navigation, medicine, maintenance guides, virtual try-on, and others.

One example application of AR allows creating live visualizations for dental restorations (see Figure 2). The patient can simply smile into the camera of a tablet and immediately see his or her teeth replaced by a digital 3D model on the displayed image. For patients who need or wish a dental restoration, a photorealistic virtual preview of the expected outcome is highly desired. In this example, AR technology provides a solution for patients who wish to know how the restoration will look like and also for dentists who wish to simplify the communication of treatment options with their patients.

To achieve great user experience, it is vital that the rendered content matches the real content in every aspect. While the overlaid virtual content in Figure 2 is geometrically correct and looks highly realistic, it does not perfectly blend in with the real photograph. The reason for this is the virtual content being created with an idealized model that does not take into account various deficiencies of the real camera such as blur, image noise, tone, and others. These effects cannot be modeled or easily identified by the rendering method, which is a general limitation for most augmented reality applications where a photorealistic result is expected, and solutions to this problem are termed coherent rendering in the literature.

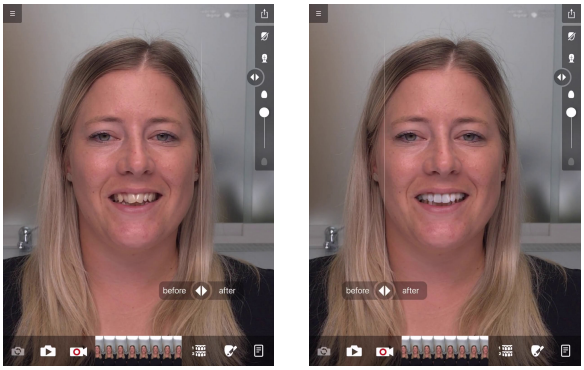



Figure 2: An existing virtual preview app for dentistry. Left: original camera image. Right: virtual teeth overlaid. The rendered denture appears realistic but its style does not perfectly match the background. Our goal is to enhance the rendering.

### 1.2 Contributions

This paper deals with improving the look of rendered virtual dentures in a dental try-on application, but can be further generalized to other AR scenarios. We apply ideas from recent artistic style transfer methods and adapt them to the coherent rendering problem. Using traditional style transfer terminology, we take the *style* of a real image and apply it to the corresponding rendered image, the *content*. The main novelty for AR is that instead of simply overlaying virtual content on the background, our method fully re-generates the output image in an autoencoder respecting both the real style and the real plus virtual content.

We propose and compare various convolutional neural network architectures and evaluate the quality of the generated images. We experiment with and expand upon existing work for fast style transfer and temporally consistent video style transfer. We also verify in a small user study that the solutions presented here allow more realistic and seamless blending of real and virtual content.

*e-mail: valentin@kapanu.com—valentin.vasiliu@epfl.ch
†e-mail: gabor@kapanu.com—gabor.soros@nokia-bell-labs.com

## 2 Related work

A detailed survey on coherent rendering by Kronander et al. [16] and a survey on real-time perception-driven rendering by Weier et al. [32] have been published recently. We only highlight the most important related works here with a focus on AR applications.

Besides geometric registration, perhaps the most important aspect in AR is to match the color tone of real and virtual. Earlier color transfer methods adapt the colors of a destination image to match those of a source image based on pixel statistics [24], color histogram matching, or linear color transformations [23].

More complex effects of real cameras (motion blur, defocus blur, Bayer mosaicing, chromatic aberrations, lens distortion, etc.) are addressed by Klein et al. [15] for realistic AR compositing. Park et al. [22] extend the ESM tracking algorithm for handling severe motion blur and also a very fast method for rendering realistic motion blur as image warping. Oskam et al. [21] propose a fast color balancing method for matching the tone of images, given that the user manually selects multiple reference points in both the source and destination images. The works of Meilland et al. [20], Rohmer et al. [26] [27], and Rhee et al. [25] deal with the realistic reproduction of lighting on virtual objects. Our goal is rather to match the low-level properties of real and virtual (tone, noise, blur, etc.) and we aim for a fast and robust method without need for user intervention.

An interesting and novel application of deep learning, "A neural algorithm of artistic style" was introduced by Gatys et al. [7] in 2015. This work formulated a method to separate the "style" and "content" of an image and transfer arbitrary styles from one image to another by matching feature activations produced by a convolutional neural network pre-trained on an image classification task. While this method is very slow, it showcased the versatility of deep learning models to process and alter higher level concepts in an image such as style or content and it inspired a great line of works on style transfer.

To address the runtime issue, Johnson et al. [12] train a specific architecture for an individual style so that at inference time only one pass is needed to produce the stylized result. Concurrently, Ulyanov et al. [31] replace the batch normalization layers with instance normalization layers, which intuitively helps in removing the contrast information in each instance in the content image and simplifies the style transfer process. The drawback of these methods is that they are limited to specific styles, i.e., one has to retrain the architectures for each new style image as opposed to our desired method which would work independently from the style image used.

To allow arbitrary style transfer, Huang and Belongie [11] introduce a novel adaptive instance normalization (AdaIN) layer which aligns the mean and variance of a set of features to another set of features. This operation transfers style by transferring first level statistics between the feature activations of any pair of content and style images which rivals the speed of simple feed-forward methods. While AdaIN concerns itself with transferring first order statistics, the whitening-coloring transform (WCT) by Li et al. [17] transfers second order statistics. Both transformations can be seen as steering the distribution of features of one image to become closer to the distribution of features of another image, and both are reasonably fast even on an average CPU.

Similar to our goal are the painterly harmonization method of Luan et al. [19] which allows to seamlessly paste objects into an image, and the spatial fusion GAN by Zhan et al. [34] which combines artistic images with additional objects in a coherent manner. Generative adversarial networks (GANs) have shown remarkable quality in generating images [13], and even though a very recent method by Zakharov et al. [33] is able to generate single face images in real time on a very powerful GPU, we consider these methods too complex for interactive applications.

Style transfer applications typically serve artistic purposes, for example transferring the style from a painting to a normal photograph while preserving the content of the photograph. Recently, there have been attempts to extend this idea towards video style transfer and photorealistic style transfer [4, 6, 10, 28]. We adapt and apply these works and their variations for a different purpose, specifically to make the virtual content in AR applications appear more realistic. We are not aware of any other work applying style transfer for AR, which is quite different from traditional offline or online image or video stylization. In coherent rendering, we have the main goals of both spatial- and temporal consistency (realism as opposed to abstraction) and speed, and we show how we can adapt neural style transfer methods to this rather different task, even if only in a quite specific application area.

## 3 Coherent rendering with fast style transfer

### 3.1 Overview

Instead of directly improving the AR renderer by modeling all subtle effects of reality, we propose a post-processing step that can be applied on the virtual content created by any AR renderer. The method is inspired by the recent success of artistic style transfer research. Features inside a convolutional neural network (CNN) represent more and more abstract versions of an image when going deeper in the network. Style transfer creates a new image by mixing two inputs, taking higher level features from the *content* but lower level features from the *style*, which is intuitively similar to what we aim for in our coherent rendering problem.

The core idea of our method is to apply the *style* of the original unaltered image to the rendered virtual *content* image such that the two seamlessly blend together. Our hypothesis was that the geometry defined by the rendering is preserved while different image attributes such as noise, color tone, blur etc. are transferred from the original image. Our initial tests with the method of `deepart` [1] (presumably based on [7]) showed great potential for transferring the original image characteristics such as noise, color tone, etc. onto the rendered image, while also preserving the geometry of the latter. However, we cannot directly apply the method to AR applications for two reasons.

First, the computational requirements are too high. In our experiments with the method [7], it takes more than 10 minutes to process a pair of content and style images on an nvidia GTX 1080 GPU. Similarly, most other methods that promise "real-time" performance imply using a state-of-the-art powerful desktop GPU for inference, which prohibits the use of such methods on most average laptops let alone on mobile devices. In contrast, our goal is a fast post-processing step in a single feed-forward pass which can be potentially implemented even on a mobile device in the future.

Second, only a few style transfer methods deal with temporal consistency in videos, and those either rely on the computation of optical flow which is slow, or have to train a different network for each style which does not provide enough flexibility. We enforce local temporal consistency within our architecture with only little computational overhead.

### 3.2 General structure

Our first step is to construct autoencoders from various popular architectures but truncated at different depths (the exact configurations are explained in Section 3.4). An autoencoder model involves an encoder network which takes an image as input and produces a set of feature maps. The encoded features are further passed to a decoder network which tries to reconstruct an output image that resembles the original input image. Either the whole network is trained from scratch or the encoder can be pretrained on some other task in advance, in which case only the decoder needs to be trained.

The training loss for our autoencoders is a combination of the standard pixel loss (so that the output image is as similar as possible to the input image) and of the perceptual loss proposed in [12] (so that the corresponding feature maps of the two inputs inside a so called loss network are as similar as possible), in other words

comparing images in the CNN feature space rather than in pixel space. For example, if we decrease the intensity of every pixel by 1, the perceptual loss is minimal but the pixel loss is large. The loss network extracts high-level representations of the images and compares them at higher abstraction levels. Formally,

$$\mathcal{L} = \alpha\mathcal{L}_{pixel} + \beta\mathcal{L}_{perceptual} = \alpha||O-I||_2 + \beta||F[O]-F[I]||_2 \quad (1)$$

where $I$ is the input image, $O$ is the output of the autoencoder, and $\alpha$, $\beta$ are weighting parameters. Similar to [12], for loss network we chose the normalized VGG19 architecture pre-trained on ImageNet [5] for the image classification task, and $F[\cdot]$ represents the feature map encoding at its `relu4_1` layer. The VGG19 is capable of producing good feature representations and has been used successfully in many other tasks, but some other pre-trained architecture could also be used as a loss network. For training all our autoencoders, we used the updated COCO 2017 dataset [18] with a batch size of 8 and the ADAM optimizer [14] with a starting learning rate $10^{-4}$. Note that our autoencoders are not specific to faces or mouth images but are trained to reconstruct any kinds of images. There is no need to do any adaptations to a specific person either. Training on randomly cropped images from COCO helps with the generalization of the network to obtain robust features. All training sessions were performed on an nvidia GTX 1080 GPU. We implemented all methods in PyTorch [2].

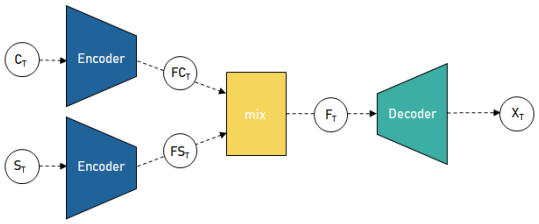



Figure 3: General architecture for style transfer: autoencoder with a feature mixing block at the bottleneck. We apply style transfer between the original video frame at time $t$ (style) and the corresponding rendered AR frame at time $t$ (content). We experiment with various encoder-decoder pairs and also compare AdaIN and WCT as mixing blocks.

### 3.3 Stylization component

A new idea in our approach is to use every single frame as a new style, because the style has to continuously adapt to the input video or camera stream. There are multiple ways for transferring statistics from the style feature maps to the content feature maps, but this adaptation requires style mixing blocks that work with arbitrary styles without retraining anything.

We use the whitening-coloring transform (WCT) proposed in [17]. Statistical whitening means transforming the data such that it has an identity covariance matrix, then statistical coloring is applied to convert the data to have a different set of target statistics. Alternatively, we could use AdaIN proposed in [11] which is simpler and faster, but requires the training process to be modified with the mixing block enabled at the bottleneck. In contrast, the WCT operation can be inserted into any previously trained autoencoder without extra training effort. As the authors of [17] point out, AdaIN which transfers first-order statistics across feature maps can also be considered as a special case of the more general WCT operation which transfers second-order statistics. WCT generally delivers higher quality results at the expense of more computations. When using deeper architectures, the WCT operation has to be repeated at multiple depths [17]. In practice, this is done by chaining multiple autoencoders with different depths, each with a WCT block in the middle. The chain starts with the deepest and ends with the shallowest autoencoder, the whole chain producing significantly higher quality results then each of its parts. We provide a derivation of the WCT formulas and an illustration of chaining in the supplement. In our method, we omit the chaining of multiple autoencoders as we use relatively shallow archtitectures because (i) we aim to transfer only low-level features and (ii) we aim for very fast inference.

### 3.4 Architectural variations

We created and compared various autoencoder architectures. The encoders are truncated versions of the original MobileNet [9], MobileNetV2 [29], and VGG19 [30] architectures while the decoders are mirrored versions of them. The preference for smaller (truncated) networks is not only motivated by the computation time and memory consumption, but also by the fact that we do not require the highest level features at the deepest parts of the networks as the image characteristics we are transferring are mostly local features.

More specifically, we experimented with the following architectures: MobileNetV1 (see [9]) truncated at `ReLU6_3` ($9^{th}$ layer) or `ReLU6_7` ($21^{st}$ layer) or `ReLU6_11` ($33^{rd}$ layer), MobileNetV2 (see [29]) truncated at `BatchNorm_3` ($8^{th}$ layer) or `BatchNorm_9` ($24^{th}$ layer) or `BatchNorm_18` ($48^{th}$ layer), VGG19 (see [30]) truncated at `ReLU_1` ($3^{rd}$ layer) or `ReLU_3` ($8^{th}$ layer) or `ReLU_8` ($22^{nd}$ layer). For all these configurations, we tested the strategy of training the whole autoencoder from scratch, and also tested the strategy of using pre-trained weights for the encoder and training only the decoder.

MobileNets perform very well on image classification and are significantly faster than other networks. Unfortunately, our architectures based on MobileNets proved to be insufficient in terms of style transfer quality, even though they were capable to reproduce the original images when used as simple autoencoders. Nevertheless, we also report our negative results to share knowledge with interested readers. In particular, we found little variation in the stylized outputs when different style images were used with the same content image, for example only the color palette changed but the detailed patterns were not transferred.

A general issue we experienced with deeper versions of the autoencoder models is that the outputs lacked finer details and often the geometrical structures became significantly deformed. Instead of using the autoencoder training approach, we also tried the AdaIN training approach proposed in [11], i.e., having the mixing block enabled during training. However, our results were rather underwhelming and uncanny (see Appendix). Next, we turned our attention to the normalized VGG19 architecture, specifically truncated versions of it. While slower than MobileNet, the VGG model and its extensions have been used with success in other style transfer works.

In our final version, we use the architectures detailed in tables 1, 2 and 3, which consist of encoder-decoder networks where the encoder is a subset of the full VGG19 architecture and the decoder is a mirrored version of the encoder. The encoder weights are fixed by values of the VGG19 model pre-trained on the ImageNet dataset for the image classification task, and only the decoders are trained as described in Section 3.2. Once the encoders and decoders are trained in an autoencoder fashion, we can use them for style transfer by applying the WCT operation between the feature maps produced by the encoder and passing the transformed features through the decoder to reconstruct a stylized output image.

### 3.5 Temporal consistency

Another key aspect to consider for AR is the temporal consistency when applying style transfer on a sequence of images (frame by frame in a video or in a live camera stream). Because the style transfer methods reconstruct each frame individually, it is natural to expect flickering artifacts when using them for videos, which

| Type | Filter | Input Size |
|---|---|---|
| **Encoder** | | |
| Conv2D | $3\times3\times3\times3$ | $224\times224\times3$ |
| Conv2D | $3\times3\times3\times64$ | $224\times224\times3$ |
| **Decoder** | | |
| Conv2D | $3\times3\times64\times3$ | $224\times224\times64$ |

Table 1: Reduced VGG19 architecture type 1, referred from now on as VGG19.1

| Type | Filter | Input Size |
|---|---|---|
| **Encoder** | | |
| Conv2D | $3\times3\times3\times3$ | $224\times224\times3$ |
| Conv2D | $3\times3\times3\times64$ | $224\times224\times3$ |
| Conv2D | $3\times3\times64\times64$ | $224\times224\times64$ |
| Maxpool $2\times2$ | | |
| Conv2D | $3\times3\times64\times128$ | $112\times112\times64$ |
| **Decoder** | | |
| Conv2D | $3\times3\times128\times64$ | $112\times112\times128$ |
| Upsampling NN | | $112\times112\times64$ |
| Conv2D | $3\times3\times64\times64$ | $224\times224\times64$ |
| Conv2D | $3\times3\times64\times3$ | $224\times224\times64$ |

Table 2: Reduced VGG19 architecture type 2, referred from now on as VGG19.2

| Type | Filter | Input Size |
|---|---|---|
| **Encoder** | | |
| Conv2D | $3\times3\times3\times3$ | $224\times224\times3$ |
| Conv2D | $3\times3\times3\times64$ | $224\times224\times3$ |
| Conv2D | $3\times3\times64\times64$ | $224\times224\times64$ |
| Maxpool $2\times2$ | | |
| Conv2D | $3\times3\times64\times128$ | $112\times112\times64$ |
| Conv2D | $3\times3\times128\times128$ | $112\times112\times128$ |
| Maxpool $2\times2$ | | |
| Conv2D | $3\times3\times128\times256$ | $56\times56\times128$ |
| Conv2D | $3\times3\times256\times256$ | $56\times56\times256$ |
| Conv2D | $3\times3\times256\times256$ | $56\times56\times256$ |
| Conv2D | $3\times3\times256\times256$ | $56\times56\times256$ |
| Maxpool $2\times2$ | | |
| Conv2D | $3\times3\times256\times512$ | $28\times28\times256$ |
| **Decoder** | | |
| Conv2D | $3\times3\times512\times256$ | $28\times28\times512$ |
| Upsampling NN | | $28\times28\times256$ |
| Conv2D | $3\times3\times256\times256$ | $56\times56\times256$ |
| Conv2D | $3\times3\times256\times256$ | $56\times56\times256$ |
| Conv2D | $3\times3\times256\times256$ | $56\times56\times256$ |
| Conv2D | $3\times3\times256\times128$ | $56\times56\times256$ |
| Upsampling NN | | $56\times56\times128$ |
| Conv2D | $3\times3\times128\times128$ | $112\times112\times128$ |
| Conv2D | $3\times3\times128\times64$ | $112\times112\times128$ |
| Upsampling NN | | $112\times112\times64$ |
| Conv2D | $3\times3\times64\times64$ | $224\times224\times64$ |
| Conv2D | $3\times3\times64\times3$ | $224\times224\times64$ |

Table 3: Reduced VGG19 architecture type 4, referred from now on as VGG19.4

is indeed confirmed in our experiments. Among the works we reviewed for video style transfer, only the one proposed by Ruder et al. [28] can be applied for arbitrary pairs of content and style images, however, it is based on the slow optimization approach by Gatys et al. [7] and the expensive procedure of computing the optical flow, making the whole process impractical for our purposes.

Our goal is to stylize $C_{t-1}$ (content) with $S_{t-1}$ (style) and $C_t$ with $S_t$ while ensuring that $X_{t-1}$ (output) and $X_t$ are also very similar. In the classical video style transfer applications $S_{t-1} = S_t$, and in our AR use case they are not equal but very similar. We propose two extensions to the basic architecture that add minimal extra computational cost compared to single-frame stylization.

The first extension we propose is shown in Figure 4 denoted by $\alpha_F$. In this network architecture, we introduce a parameter $\alpha_F$ to weigh the current (stylized) features $F_t$ with the previous (stylized) features $F_{t-1}$, formally $F_t^* = (1-\alpha_F)F_t + \alpha_F F_{t-1}$. Afterwards, the feature maps $F_t^*$ are passed through the decoder to obtain the output frame $X_t$. This additional operation acts like a temporal filter, it smooths out variations in the output in nearby frames, reducing artifacts such as flickering and structural inconsistencies. We could also feed back multiple previous frames with for example exponentially decaying weights over time, but then in case of fast motion in the video, the results may exhibit trailing artifacts as each feature map is essentially blended with several previous feature maps. As our results with more complex feedback were not significantly better, we feed back only a single history frame.

The second extension we propose is feeding back the output style, denoted by $\alpha_S$ in Figure 4. In this setup, we combine the style features $FS_t$ with the stylized output of the previous frame ($X_{t-1}$), formally $FS_t^* = (1-\alpha_S)FS_t + \alpha_S FX_{t-1}$. The motivation behind this step is the following: assuming we have the content image $C_t$ and we wish to stylize it in a specific way $S_t$, the ideal style input image would be an image in content similar to $C_t$ and already stylized by the respective style. In fact, the previous output $X_{t-1}$ is very similar to $C_t$ and it is already stylized.

Note that in both extensions, we could feed back directly $F_{t-1}$, but instead we feed back $X_{t-1}$ and pass it through an encoder again to get $FX_{t-1}$. This intermediate step is supposed to overcome any particular effects that the decoder may introduce to the image because those should also be consistent across the outputs $X_{t-t}$ and $X_t$. The cost of this feedback is the additional execution of the encoder. Also note that our two simple temporal extensions do not require retraining the networks.

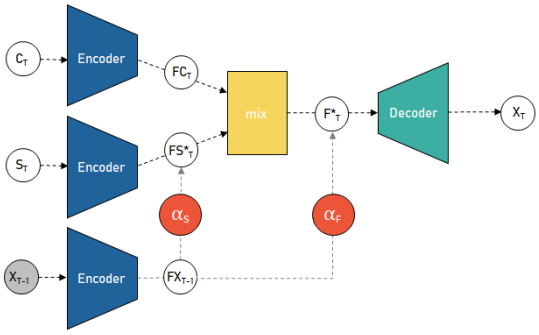



Figure 4: Our proposed architecture. Temporal consistency is enforced by feeding back the previous output to the style ($\alpha_S$) and directly to the features ($\alpha_F$).

### 3.6 Discussion

We noticed a quality decrease in the decoder's output when going deeper with the autoencoder architectures. Specifically, while the reconstructed background and reconstructed overlay seamlessly blend together as intended, the general quality of the whole output image is often blurrier than the original image (cf. the $1^{st}$, $3^{rd}$, and $4^{th}$ columns in Figure 5). This is because the decoders may not be able to perfectly reconstruct the input from the more abstract features. By going deeper in the networks we may loose fine details, especially with networks trained for image classification, because for classification only the whole image content matters. For our case, loosing local details is a problem with deeper architectures and results in the blurry outputs mentioned above. The problem may be addressed by further training the decoders, but we can also ignore it when using shallow architectures as we are primarily interested in transferring local properties.

There are multiple ways of dealing with the region of interest (ROI). In our experiments, the input images and videos were cropped around the mouth area. A potential extension would be to apply style transfer only on a specific selection like the inner mouth area by constructing a binary segmentation mask around the rendered content. Ideally, one would extract a binary mask from the renderer, and process and recombine the two images only inside the mouth region. Restricting a style transfer network to a specific area covered by segmentation masks defined in the image space has been shown also by Gatys et al. [8], which is in general the way to go.

Without such a mask, we still have the following fallback options: (i) We can take the whole input image as style and regenerate the whole output image at once, but then also the background may influence the style of the teeth, i.e., in our example application the results may depend on the face and the background. This works well as long as mostly mouth areas are in the video. Alternatively, (ii) we can take a rectangular region in the style image and apply its style to the whole output at once, similar to traditional style transfer. However, in our case when we may improve the virtual content, we may deteriorate the background, but this is not apparent with the proposed variants. If we consider more important for the output real and virtual to be indistinguishable from each other compared to having the output indistinguishable from the input (in a virtual mirror the background might be also beautified), then this can be an acceptable trade-off. Finally, (iii) processing only a rectangular ROI of the output no matter whether the style comes from a small part or from the whole input image is not recommended, because a post-processed rectangular ROI could stand out from an unprocessed background. Blending may help to smooth the borders, but having a mask from the renderer should be preferred.

In the default case, the content and style images are aligned, as

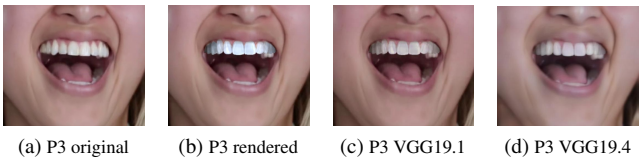
the content image is nothing else but the style image with a rendered denture model on the mouth area. Since this setup reduces the chance of introducing structural artifacts, we also tested traditional color transfer methods but the CNN models proved to be more successful in also matching other low-level features like image noise.

Our solution for temporal consistency requires that subsequent incoming frames are similar. This is true for camera or uncut video input, but it is obviously violated at video cuts.

Finally, as our architectures and our training procedure are not restricted to faces or teeth, our method can be easily transferred to other application domains. However, it is advantageous if the virtual overlay is *replacing* a real counterpart in the image because then otherwise it is difficult to decide which part of the original image should be taken as style. This restriction is less important if segmentation masks are available and we perform style transfer between segmented counterparts.

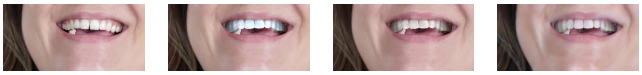
## 4 Evaluation

In this section, we present our results combining the proposed methods and applying them on our dental AR improvement task. Section 4.1 showcases several examples of our post-processing method applied on the rendering outputs, Section 4.2 describes the effect of the temporal extensions, Section 4.4 presents a preliminary user study to measure the subjective quality of the results, and Section 4.5 lists statistics regarding the computation time of various architectures.

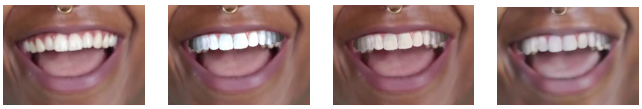


(a) P3 original (b) P3 rendered (c) P3 VGG19.1 (d) P3 VGG19.4

(e) P16 original (f) P16 rendered (g) P16 VGG19.1 (h) P16 VGG19.4



(i) P18 original (j) P18 rendered (k) P18 VGG19.1 (l) P18 VGG19.4



(m) P15 original (n) P15 rendered (o) P15 VGG19.1 (p) P15 VGG19.4

Figure 5: Example results using shallow VGG19.1 and deep VGG19.4, further examples and results with other architectures can be found in the supplement. All test frames are from [3].

### 4.1 Quality

We present and compare how our proposed style transfer methods behave on a set of rendered frames. As a short reminder, the VGG19.1, the VGG19.2 and the VGG19.4 are encoder-decoder architectures with the encoders being truncated versions of the popular VGG19 architecture and the decoders are the mirrored versions of the encoders. With the encoders being fixed, the architectures are trained in an autoencoder fashion with the decoder weights being the learned parameters. For style transfer, we use the WCT operation between the feature maps of the content and style images obtained at the bottleneck of the autoencoders. After this step, the transformed features are passed through the decoder and the final result is produced.

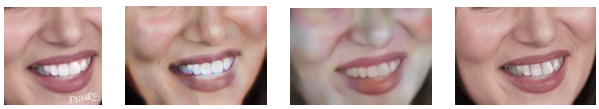
Example results are illustrated in Figure 5, further results can be found in the appendix. From the samples shown, we can identify certain behaviours depending on the style method used. The shallower networks, for example VGG19.1 and VGG19.2 primarily deal with low-level features and manage to capture features such as colors and noise patterns, our primary targets. In contrast, the deeper VGG19.4 produces outputs that in some cases can exhibit high smoothing. Interestingly, at the same time it is able to capture the level of blur in images, i.e., when the scene goes in and out of focus, a particularly difficult case for coherent rendering (also see the video supplement).

For comparison, we show in Figure 6 the outputs of some classical tone mapping approaches like histogram matching and linear color transfer. RGB histogram matching may introduce false colors, therefore color transfer in the Lab color space is preferred. Linear color transfer with PCA and with MK [23] work reasonably well except in a few bright overflow regions. Our method (cf. Figure 5(g)) with style transfer generates comparable or higher quality output, and it can not only transfer color distributions, but also other low-level properties. More images can be found in the supplement. We also experimented with the work of Reinhard et al. [24], however, this method cannot capture the subtle differences when the input images are very similar.

(a) P16 HM-RGB (b) P16 HM-LAB (c) P16 LCT-PCA (d) P16 LCT-MK

Figure 6: Qualitative comparison with traditional color transfer methods. HM-RGB and HM-LAB stand for histogram matching in RGB and Lab color space, respectively. LCT-PCA and LCT-MK stand for linear color transfer with principal component analysis and with Monge-Kantorovitch distance [23].

We also show a qualitative comparison with other style transfer methods in Figure 7. The result of `deepart.io` is of highest quality, but the processing takes minutes. The original AdaIN [11] is fast but tends to produce wavy artefacts and flickering in the frame stream. The original WCT [17] is too deep and may distort the geometry too much. Our method (VGG19.2) is a good trade-off between quality and speed. More examples can be found in the supplement.



(a) deepart (b) AdaIN (c) WCT (d) **ours**

Figure 7: Qualitative comparison with other style transfer methods [1, 11, 17]. Input content P16 in Fig. 5(f) and style in Fig. 5(e).

In summary, our proposed methods greatly improve the photo-realistic look of the virtual dentures. The VGG19.1 and VGG19.2 methods focus more on the fine details but fail to correctly capture the defocus blur of the camera. The VGG19.4 method can deal well with the camera blur, however, at the price of deteriorating the overall quality of the original background – the decoder is not capable to perfectly reconstruct the original video frame.

### 4.2 Temporal consistency

In Section 3.5, we proposed two extensions to improve the temporal stability of a style transfer network on videos without significantly increasing the overall computation time.

Here, we analyze the temporal consistency of the results when applied on the test videos. Because of subtle differences, the results can be better seen in video format, ideally compared side by side. We recommend the reader to refer to the video supplement[1] for a closer inspection of example videos that show the influence of the two extensions with various weights. Note that we cannot directly compare our method with video style transfer methods like [6, 10, 28] because they (1) require re-training for each style, and our 'style' changes at every frame and (2) rely on dense optical flow and optimization which is costly. While our simple feedback can be expected to have lower quality, it is very fast and appears sufficient for the presented application.

The experiments validate our initial assumption that for shallower architectures (VGG19.1 and VGG19.2), the temporal artifacts are hardly noticeable and in these cases the addition of our temporal feedback brings little contribution. However, as the architectures grow deeper (as is the case of VGG19.4), the temporal artifacts become more pronounced consisting of unwanted flickering and distortions in the final output. In these cases, our methods smooth out such effects to a considerable degree, improving the overall quality of the stylized video. While most results presented here were generated with $\alpha_S = 0.2$ and $\alpha_F = 0.1$, we leave these parameters manually adjustable, for example if there is a lot of motion and quick movements in the video, a trailing effect may appear if the weight of feedback is too high.

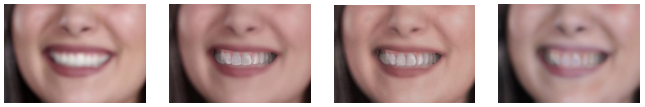
We use only 1 frame history in smoothing, but it has longer effect due to the feedback loop. One could also limit the extent a single frame can influence in the stream by storing and recombining the features in a buffer instead of always feeding back the generated frames. We also experimented with directly combining multiple previous features with exponentially decaying weight but this introduced trailing effects, as can be expected.

In conclusion, our architectural extensions produce better results compared to performing style transfer frame-by-frame independently, while doing so at no significant additional cost. We also acknowledge that general video style transfer methods using optical flow provide better results both short term and long term, however, at significant computational costs. In our approach, we are smoothing the features which is even faster than smoothing the output and gives similar results. Due to the nature of our AR task, we use fairly shallow architectures which, in our experiments, are not as prone to temporal artifacts as deeper architectures. In fact, if we use the VGG19.2 model, which was preferred in the user study, temporal smoothing may not even be necessary. Our conclusion is that the methods described in this paper for improving the temporal consistency are sufficient for handling the subtle artifacts between consecutive frames and during short temporal intervals.

### 4.3 Defocus blur and masking

In this example, we would like to highlight a particularly difficult case for coherent rendering, where the amount of defocus blur is continuously changing in the original video. This is very difficult to measure and reproduce in real time. However, it is well captured and transferred by the deep VGG19.4 network, as we can see in the video of Person16 in the supplement[2].

A second interesting observation we can make here is that towards the end of the videos of Person16, the teeth become too dark. This is because the video is not cropped to only around the mouth region, and when the dark background becomes visible, the dark color also gets transferred and applied on the whole image. We could overcome this by cropping closer to the mouth or applying a masks for processing, as also mentioned in the discussion section of the paper. We kept this video as is for illustration purposes.



(a) original (b) WCT1, $\alpha_S = 0.2$, $\alpha_F = 0.1$ (c) WCT2, $\alpha_S = 0.2$, $\alpha_F = 0.1$ (d) WCT4, $\alpha_S = 0.2$, $\alpha_F = 0.1$

### 4.4 Qualitative user study

We conducted a small user study with 8 participants in order to assess the quality of our results when compared to the default results produced by the rendering method. The study consists of a short questionnaire in which users have to answer several questions about the quality of the outputs according to their own perceptual standards.

For the study, we randomly extracted individual frames from 8 reference videos [3] as original frames, and we generated the corresponding output variations we wish to compare, specifically the output produced by the default rendering method, and the outputs of VGG19.1, VGG19.2, VGG19.4 produced by the respective style transfer methods (in total 128 images per person). Given a set of images, the participants had to answer two questions: *Q1: How well does the smile area fit the rest of the image?* and *Q2: How well does the new smile match the style of the original image?* As Q2 is concerned only with the mouth area for each frame, we replace the region outside of the mouth with the same region from the original frame in order for the users to focus on the region of interest.

We requested the users to rank 6 frames (1 default rendered and 5 post-processed with 5 different autoencoders) such that a frame ranked 1 answers the question better than all the other frames and a rank of 6 suggests that the respective frame is the least preferred option. The motivation behind adding a ranking system is that humans are better at perceiving relative differences rather than absolute values, therefore only having labels would merely provide a rough estimate of the quality but without knowing which option is preferred. The results of the survey are summarized in Table 4. The ranking verifies our hypothesis that all VGG19.x methods improved over the original rendering, and the outputs of VGG19.2 and VGG19.1 were ranked best and without significant difference.

| Question | Type | Rank average | Rank standard deviation |
|---|---|---|---|
| Q1 | Rendered | 5.39 | 0.63 |
| | VGG19.1 | 3.00 | 1.27 |
| | VGG19.2 | **2.98** | 1.22 |
| | VGG19.4 | 3.04 | 1.56 |
| Q2 | Rendered | 5.66 | 0.60 |
| | VGG19.1 | 2.63 | 1.34 |
| | VGG19.2 | **2.52** | 1.25 |
| | VGG19.4 | 3.45 | 1.39 |

Table 4: Survey results: average rank given by the users and the standard deviation of the rank for output images generated by the respective methods.

We repeated the survey with 6 users on a different set of images (in total $7 \times 8$ images per person) in which we asked the users to compare outputs of our method with traditional color transfer methods, answering the same questions Q1 and Q2 as before. The results of this survey are summarized in Table 5. We learned that users preferred every post-processed result over the default rendered

[1] See the conference proceedings and the authors' website

[2] person16/video_fps-15_arch-wct_mode-4_alphaS-08_alphaF-09.mov

images, and among the traditional methods the simple histogram matching in the Lab color space was preferred in most of the cases. Still, the images processed with our methods ranked highest in the answers of both questions, which verifies our assumption that more low-level properties than just color need to be transferred in order to get seamless and coherent virtual smile previews.

| Question | Type | Rank average | Rank standard deviation |
|---|---|---|---|
| Q1 | Rendered | 6.02 | 0.91 |
| | HM-RGB | 4.29 | 1.88 |
| | HM-LAB | 3.31 | 1.34 |
| | LCT-PCA | 3.33 | 1.33 |
| | LCT-MK | 5.02 | 1.26 |
| | VGG19.1 | **3.02** | 1.45 |
| | VGG19.2 | **3.00** | 1.50 |
| Q2 | Rendered | 6.23 | 0.63 |
| | HM-RGB | 4.35 | 1.93 |
| | HM-LAB | 3.67 | 1.22 |
| | LCT-PCA | 3.58 | 1.38 |
| | LCT-MK | 5.13 | 1.16 |
| | VGG19.1 | **2.06** | 0.97 |
| | VGG19.2 | **2.98** | 1.51 |

Table 5: Survey results: average rank given by the users and the standard deviation of the rank for output images generated by the respective methods. The abbreviations are defined in Section 4.1.

### 4.5 Speed

In this section, we compare our methods VGG19.1, VGG19.2, VGG19.4 with respect to computation time, and also include the AdaIN method proposed by [11]. We selected AdaIN as it is the fastest method for arbitrary style transfer that we tested, and also because it is among the fastest style transfer methods in general. The results of our tests concerning the inference speed of the selected models are presented in table 6.

We obtained these measurements by repeating the inference step 51 times for each of the respective models using a content image of size $512 \times 512$ and a style image of size $512 \times 512$. Our unoptimized PyTorch code ran on a 2017 MacBook Pro with Intel Core i7 3.1 GHz CPU (without GPU), and we also compared the runtime on a PC with an nvidia 1080 GTX GPU.

| Model | M i7 | A i7 | S i7 | M GPU | A GPU | S GPU |
|---|---|---|---|---|---|---|
| VGG19.1 | 1.237 | 1.237 | 0.020 | 0.031 | 0.032 | 0.002 |
| VGG19.2 | 3.455 | **3.460** | 0.030 | 0.044 | **0.044** | 0.001 |
| VGG19.4 | 9.590 | 9.614 | 0.131 | 0.147 | 0.149 | 0.003 |
| AdaIN | 8.204 | 8.231 | 0.118 | 0.147 | 0.147 | 0.007 |

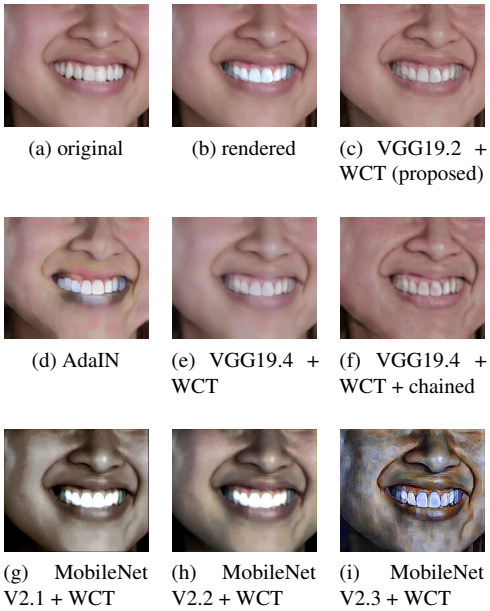
Table 6: Inference time in seconds (without feedback for temporal consistency, PyTorch, CPU only and GPU, $512 \times 512$ resolution). M: median, A: average, S: standard deviation.

Note that VGG19.2, which was preferred by the users, takes $3.56\,s$ on the MacBook, and only $44\,ms$ on the GPU. The VGG19.1 and VGG19.2 are considerably faster than the AdaIN model as expected. Interestingly, VGG19.4 seems to be slower when compared to AdaIN even though the corresponding encoder and decoder of VGG19.4 are shallower than the ones used by the latter. The higher computation time is caused by the WCT operation which is more computationally intensive than the AdaIN transfer operation which only normalizes the feature maps based on mean and variance. It is worth noting that the computational cost of the WCT (please refer to the Appendix for the complete derivation of the WCT formulas) will not increase with the image size as the covariance matrix only depends on the number of channels.

In the example dental AR application, the mouth area would be typically smaller by at least a factor of 2 in both width and height compared to the $512 \times 512$ image sizes we used, hence the computation time would be reduced by a factor of 4. Still, the current speed allows only video processing but no interactive AR on the CPU. The next step in the future would be to test these models on a mobile device, leveraging popular deep learning libraries and hardware acceleration provided by embedded GPUs and neural network chips.

### 4.6 Unsuccesful architectures

For completeness, we also showcase a few unsuccessful architectures in Figure 8 for comparison. MobileNets performed well in our experiments when used as autoencoders to re-generate images, however, when we tried to transfer MobileNet feature statistics with WCT, the results were underwhelming. AdaIN fails to transfer the style properly and produces wavy artifacts. Even worse, the artifacts are different at every frame and lead to severe flickering in the videos. Chaining several WCT blocks as suggested in [17] also led to unwanted artefacts. The reason why the features constructed by one network work better than the other remains an open research question.



(a) original (b) rendered (c) VGG19.2 + WCT (proposed)

(d) AdaIN (e) VGG19.4 + WCT (f) VGG19.4 + WCT + chained

(g) MobileNet V2.1 + WCT (h) MobileNet V2.2 + WCT (i) MobileNet V2.3 + WCT

Figure 8: Outputs of various architectures – Person3. The first row a)-c) shows the original image, the rendered output, and our post-processed output. The second and third rows d)-i) show negative examples.

## 5 Conclusions

With this work, we improved the natural aspect of rendered AR overlays in a virtual try-on application. We also provided extensive analysis and experiments with different architectures. The core idea of using autoencoders for seamlessly blending real and virtual content is not limited to a dental scenario and could be generalized to other applications as well. However, an important limitation of the presented technique is that it only works when a photorealistic virtual content is *replacing* real objects in the scene. Obviously, because the style is taken from the original image, the real object needs to be present, which limits the application domain. While the method finds good use in virtual previews of teeth (as in our

example), it cannot be applied in just any AR use case. Also, to achieve high frame rates, the post-processing should be limited to a smaller area of the image.

While our method is able to match the tone, noise, and to some extent the defocus blur between images, it does not take into account occlusions and shadows. It is an interesting open question whether and how a CNN method could be extended to also handle severe lens distortions as a component of style. Generally, a promising and interesting direction to explore is AR rendering in such neural architectures. In particular, several attempts have been shown recently with generative adversarial networks that can add, remove, and even edit certain objects in the scene. The obvious advantage is that when a completely new image is generated, the real background and virtual overlays match well in style. However, determining the correct position and orientation of a virtual object and representing this information inside a deep network remain open questions.

## ACKNOWLEDGMENTS

We thank all participants of our study for their time and effort. We thank Roland Mörzinger and Sabine Süsstrunk for valuable feedback on our ideas.

## REFERENCES

[1] DeepArt.io. `http://www.deepart.io`. Accessed: 2019-06-05.

[2] PyTorch. `http://www.pytorch.org`. Accessed: 2019-06-05.

[3] YouTube people smiling (first hit on Google query "youtube people smiling"). `http://www.youtube.com/watch?v=f8OmSWxF6h8`. Accessed: 2019-06-05.

[4] D. Chen, J. Liao, L. Yuan, N. Yu, and G. Hua. Coherent online video style transfer. In *International Conference on Computer Vision*, ICCV'17, 2017.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'09, 2009.

[6] C. Gao, D. Gu, F. Zhang, and Y. Yu. Reconet: Real-time coherent video style transfer network. In *Asian Conference on Computer Vision*, ACCV'18. Springer, 2018.

[7] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'16, 2016.

[8] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman. Controlling perceptual factors in neural style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'17, 2017.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] H. Huang, H. Wang, W. Luo, L. Ma, W. Jiang, X. Zhu, Z. Li, and W. Liu. Real-time neural style transfer for videos. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'17, pp. 7044–7052. IEEE, 2017.

[11] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *International Conference on Computer Vision*, ICCV'17, 2017.

[12] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, ECCV'16, pp. 694–711. Springer, 2016.

[13] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018.

[14] D. P. Kingma and J. Ba. ADAM: A method for stochastic optimization. In 3$^{rd}$ *International Conference on Learning Representations*, ICLR'14, 2014.

[15] G. Klein and D. W. Murray. Simulating low-cost cameras for augmented reality compositing. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):369–380, May 2010.

[16] J. Kronander, F. Banterle, A. Gardner, E. Miandji, and J. Unger. Photorealistic rendering of mixed reality scenes. *Computer Graphics Forum*, 34(2):643–665, May 2015.

[17] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems*, NIPS'17, pp. 386–396, 2017.

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, ECCV'14, pp. 740–755, 2014.

[19] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep painterly harmonization. In *Computer Graphics Forum*, vol. 37, pp. 95–106. Wiley Online Library, 2018.

[20] M. Meilland, C. Barat, and A. Comport. 3D high dynamic range dense visual SLAM and its application to real-time object re-lighting. In *2013 IEEE International Symposium on Mixed and Augmented Reality*, ISMAR'13, pp. 143–152, Oct 2013.

[21] T. Oskam, A. Hornung, R. W. Sumner, and M. Gross. Fast and stable color balancing for images and augmented reality. In 2$^{nd}$ *International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, 3DIMPVT'12, pp. 49–56, Oct 2012.

[22] Y. Park, V. Lepetit, and W. Woo. ESM-Blur: Handling & rendering blur in 3D tracking and augmentation. In *Proceedings of the 8$^{th}$ IEEE International Symposium on Mixed and Augmented Reality*, ISMAR'09, pp. 163–166, 2009.

[23] F. Pitié and A. Kokaram. The linear Monge-Kantorovitch linear colour mapping for example-based colour transfer. In 4$^{th}$ *European Conference on Visual Media Production*, pp. 1–9, Nov 2007.

[24] E. Reinhard, M. Adhikhmin, B. Gooch, and P. Shirley. Color transfer between images. *IEEE Computer Graphics and Applications*, 21(5):34–41, 2001.

[25] T. Rhee, L. Petikam, B. Allen, and A. Chalmers. MR360: Mixed reality rendering for 360 panoramic videos. *IEEE Transactions on Visualization and Computer Graphics*, 23(4):1379–1388, April 2017.

[26] K. Rohmer, W. Buschel, R. Dachselt, and T. Grosch. Interactive near-field illumination for photorealistic augmented reality with varying materials on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(12):1349–1362, Dec. 2015.

[27] K. Rohmer, J. Jendersie, and T. Grosch. Natural environment illumination: Coherent interactive augmented reality for mobile and non-mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 23(11):2474–2484, Nov 2017.

[28] M. Ruder, A. Dosovitskiy, and T. Brox. Artistic style transfer for videos. In *German Conference on Pattern Recognition*, GCPR'16, pp. 26–36. Springer, 2016.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'18, pp. 4510–4520, 2018.

[30] K. Simonyan and A. Zisserman. ery deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, ICLR'15, 2015.

[31] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

[32] M. Weier, M. Stengel, T. Roth, P. Didyk, E. Eisemann, M. Eisemann, S. Grogorick, A. Hinkenjann, E. Kruijff, M. Magnor, K. Myszkowski, and P. Slusallek. Perception-driven accelerated rendering. *Computer Graphics Forum*, 36(2):611–643, May 2017.

[33] E. Zakharov, A. Shysheya, E. Burkov, and V. S. Lempitsky. Few-shot adversarial learning of realistic neural talking head models. *arXiv preprint arXiv:1905.08233*, 2019.

[34] F. Zhan, H. Zhu, and S. Lu. Spatial fusion GAN for image synthesis. In *International Conference on Computer Vision and Pattern Recognition*, CVPR'19, 2019.