Supplemental Material for WISE: Whitebox Image Stylization by Example-based Learning

Winfried Lötzsch[1], Max Reimann[1], Martin Büssemeyer[1], Amir Semmo[2], Jürgen Döllner[1], and Matthias Trapp[1]

[1] Hasso Plattner Institute, University of Potsdam, Germany
[2] Digital Masterpieces GmbH, Germany

Due to space restrictions, some details had to be omitted from the main paper; we present those details here. In Sec. 1 we describe filter modifications to obtain gradients using auto-grad enabled frameworks on the example of the seperated orientation-aligned bilateral filter and color quantization, and elaborate on filter pipelines concerning their structure, learnability and memory consumption. In Sec. 2, we elaborate on training global Parameter Prediction Networks (PPNs) by presenting three PPN architectures and evaluating their performance in an ablation study. In Sec. 3, we provide additional details and results for the architecture ablation study on image-to-image translation tasks using the APDrawing dataset[22] as well as editability of such effects. In Sec. 4 we give visual examples for the style transfer capability of different effects. Finally, in Sec. 5, additional results for parametric style transfer, image-to-image translation, and effect variants are shown. Further, our supplemental video[3] demonstrates parametric style transfer optimization and interactive editing for the images shown in the main paper.

1 Differentiable Filters

In the main paper we note that while most filters are straight-forward to implement in auto-grad enabled frameworks, for some several filters re-formulations or approximations are needed. Specifically, for structure-adaptive neighborhood operations, a re-formulation to retain sub-gradients in adaptive kernel neighbourhoods must be employed, and for non-differentiable operations a numeric gradient approximation is needed. In the following, we show an example for each.

1.1 Bilateral Filter

Commonly used techniques in image filtering pipelines are bilateral filters. The bilateral filter [19] using Gaussians G of size σ_d (distance kernel) and σ_r (range kernel) for an image $I \in \mathbb{R}^{C \times W \times H}$ is defined for a pixel coordinate x by

$$\hat{I}(x) = \frac{1}{W} \sum_{y \in \Omega(x)} I(y) G_{\sigma_d}\left(\|y - x\|\right) G_{\sigma_r}\left(\|I(y) - I(x)\|\right) \tag{1}$$

[3] https://youtu.be/wIndN7cr0PE

where $\Omega(x)$ denotes the window centered in x and $y \in \Omega$ represent pixel coordinates of the kernel neighbourhood. W represents the weight normalization term, which we omit in the following for the sake of brevity. Computing the bilateral filter for large images and at large kernel neighbourhoods is, however, computationally expensive. Several approaches have been proposed to approximate the filter by separation into multiple passes for improved efficiency.

For image abstraction and stylization, the *orientation-aligned separated bilateral filter*[15,11] has shown a good trade-off between quality and performance. The filter is guided in two passes along the gradient and tangent direction of an edge tangent field. The tangent field of the input image is obtained by an eigenanalysis of the smoothed structure tensor [2]; for more details the reader is kindly referred to the work of Kyprianidis and Döllner [15]. The tangent field computation consists of point-based and fixed-neighbourhood kernels only, and thus is straightforward to implement using common auto-differentiable functions. However, the following separated bilateral filter is iterative and structure adaptive (i.e., the size of the kernel neighbourhood depends on the content), and thus cannot be ported to the fixed-neighbourhood functions (e.g., convolutions) of auto-grad enabled frameworks in a straightforward way.

Specifically, in the work of Kyprianidis and Döllner [15], the filter response at a sampling point x of a pass of the separated orientation-aligned bilateral filter in direction t (gradient or tangent direction) is defined as:

$$\hat{I}(x) = I(x) + \sum_{y \in \Omega_t(x)} I(y) G_{\sigma_d}\left(\|y - x\|\right) G_{\sigma_r}\left(\|I(y) - I(x)\|\right) \tag{2}$$

where $\Omega_t(x) = \{x + kt, \mid k \in [-N, N] \wedge k \in \mathbb{Z}\}$ represents sampling positions along the direction t defined in x, and N denotes the cut-off kernel radius. N is typically based on σ_d, e.g., set to $\lfloor 2\sigma_d / \|t\| \rfloor$, and the kernel size locally depends on the magnitude of the direction vector t.

It would be possible to implement a custom kernel with associated backward pass for the above to manually compute (sub-) gradients for both input and parameters, and similarly repeat the procedure for every other structure-adaptive filter in Tab. 1 of the main paper. However, as our goal is to minimize the effort of porting existing (shader-based) filters to our framework while maximizing reusability and portability, we propose to implement the filters using common auto-differentiable components. To this end, we reformulate Eq. (2) into a grid sampling-based operation. Specifically, we use spatial coordinate grids $C \in \mathbb{R}^{2 \times W \times H}$ and $T \in \mathbb{R}^{W \times H \times 2D}$ that represent the mapping of output pixel locations to input pixels for grid sampling. Grid C is the identity mapping, i.e., contains coordinates $C_{wh} = (w, h)$. Grid T contains the sampling offsets in dimension D: $T_{wh} = \{kt_{wh}, \mid k \in [-D, D] \wedge k \in \mathbb{Z}\}$. Then the neighbourhood sampling values $V \in \mathbb{R}^{C \times W \times H \times 2D}$ for the entire image I are computed as:

$$V = \mathcal{I}(C + T) \delta \left[G_{\sigma_d}\left(\|T\|\right)\right] G_{\sigma_r}\left(\|\mathcal{I}(C + T) - I\|\right) \tag{3}$$

where in slight abuse of notation, $\mathcal{I}(C + T)$ denotes bilinear grid sampling [9] of I over the coordinate grid of unfolded kernels. It is thus equivalent to retrieving

$I(\Omega_t(x))$ for every pixel x with fixed number of samples. To clip values outside of the adaptive kernel neighborhood, δ is computed as:

$$\delta_{whd}(v) = \begin{cases} v & \text{if } \|T_{whd}\| \leq N_{wh} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Note that multiplications between tensors in Eq. (3) are performed elementwise. Filter responses in dimension D are then folded (summed) to obtain the response over the full kernel neighborhood (i.e., the filtered image $\hat{I}$):

$$\hat{I} = I + \sum_{d=1}^{D} V_d \tag{5}$$

The size of dimension D can be varied for performance-quality trade-offs (as shown in Fig.4 in the main paper).

1.2 Color Quantization

Color quantization is often applied to achieve a flat, cartoon-like impression (Fig 3., main paper). It is usually defined using the floor function: $y = \lfloor xb \rfloor + 0.5/b$ [21], where x denotes the color value before and y the color value after quantization, and b (number of quantization bins) is the parameter that should receive gradients. Due to the non-differentiable nature of the floor function, we introduce a differentiable approximation. Common differentiable proxies like the straight-through estimator[1] work well for approximating the gradient of quantization functions with fixed numbers of quantization bins. However, for color quantization we want to differentiate with respect to the strength of the quantization or the number of bins, which is not possible using common differentiable proxies. As reasoning about the number of quantization bins requires information about the complete image, a global optimization is formulated instead:

$$f(r) = \sum_i \operatorname{sign}\left(\boldsymbol{y}_i - \frac{\lfloor r\boldsymbol{x}_i \rfloor + 0.5}{r}\right) \operatorname{sign}(\boldsymbol{g}_i) \tag{6}$$

where $\boldsymbol{x}$ denotes a vector containing the input pixel values, $\boldsymbol{y}$ the outputs of $y(\boldsymbol{x})$, and $\boldsymbol{g}$ the gradient vector for $\boldsymbol{y}$.

Intuitively, f sums values for each pixel, which are positive if the pixel's gradient points in the direction of r and negative otherwise. The maximum number of per-pixel gradients should point in the direction of r.

We choose $\hat{b} = \arg\max_r f(r)$ to derive the optimal value for the parameter b, which minimizes the learning target. This value is obtained in a single optimization step. To obtain a continuous gradient, the difference $b - \hat{b}$ is scaled by the gradient magnitude $|g_i|$ of the loss function computed with respect to all pixels $y_i \in \boldsymbol{y}$, such that gradients for b decrease, once the optimal value is approached:

$$\frac{\partial y}{\partial b} \coloneqq \sum_i |g_i|(b - \hat{b}) \tag{7}$$

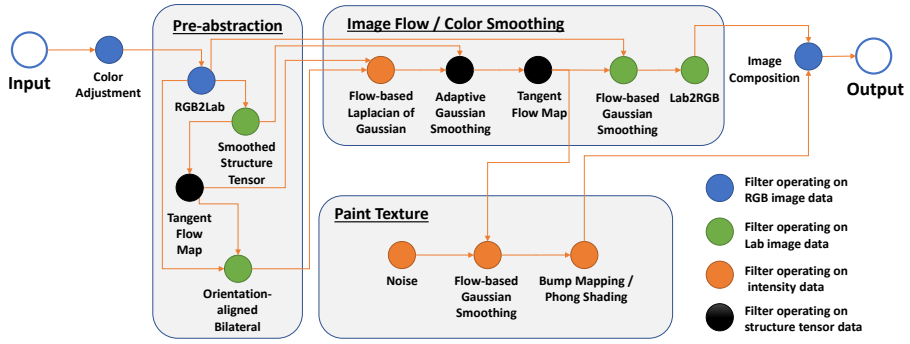


Fig. 1: Oilpaint filter pipeline adapted from Semmo *et al.* [18].

Table 1: Parameter optimization to compare learned and target stylizations. Scores are computed on Non-photorealistic Rendering (NPR) benchmark [17] for both high-quality (level I) and low-quality (level III) portraits.

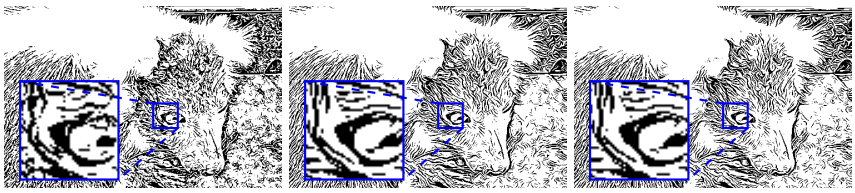
	NPR level I		NPR level III	
Loss	SSIM	PSNR	SSIM	PSNR
Cartoon ℓ_2	0.937	28.144	0.958	34.124
Cartoon ℓ_1	0.931	26.939	0.947	30.429
Watercolor ℓ_2	0.922	30.634	0.897	32.342
Watercolor ℓ_1	0.883	31.032	0.895	29.048

1.3 Differentiable filter pipeline - Oilpaint

As noted in the main paper, we implement differentiable filter pipelines analogous to their originally published versions and show an example for the toon pipeline in Fig. 3 of the main paper. In Fig. 1 we further show the filter pipeline of the oilpaint effect [18]. In contrast to Semmo *et al.* we do not use color palette extraction and color quantization and instead use a color adjustment layer.

1.4 Functional Benchmark

To verify that all parameters of an effect pipeline can be learned, we set up a functional benchmark to evaluate how well effects can be adapted to an example stylization in the same domain. We developed an OpenGL-based "ground truth" implementation to generate reference stylizations, where parameter values are randomized during the benchmark. Using an image-based loss, gradients are then backpropagated to optimize the parameters of the differentiable effect using gradient descent. We measure the Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR). Tab. 1 shows that both the cartoon



(a) 0.98 GiB, 0.12 s (b) 1.51 GiB, 0.17 s (c) OpenGL reference

Fig. 2: GPU VRAM and run-time in seconds for varying kernel sizes $D = 1$ a and $D = 5$ b for XDoG. (Image resized to 1MP from [3]).

and watercolor effect achieve very high similarity to their reference using both ℓ_1 and ℓ_2 losses. The photographic quality level of the input images does not seem to play a role in the style-matching capability. As the eXtended difference-of-Gaussians (XDoG) filter is contained in both cartoon and watercolor pipelines, it is not evaluated separately.

1.5 Implementation Aspects - Memory

During training, limiting the memory consumption of individual filters is important, as they operate in the full input resolution. Generally, filters that accumulate sampling of multiple different locations in a single tensor have the highest memory usage. For our filters, wet-in-wet stylization, Kuwahara, and bilateral filtering are the most expensive with 3 GB to 5 GB peak memory usage for 1 MP input images. The trade-off between the quality of the generated results and computing resources can be controlled by the kernel size parameter D as shown in Fig. 2. To further reduce memory consumption during training, we use gradient checkpointing to recompute intermediate gradients on the fly. Thereby, activations are stored only at the end of each filter, which is usually a single RGB image. Intermediate activations are re-computed on the fly during back-propagation. The peak memory usage thus then depends on the single most memory-intensive filter. For example, the non-checkpointed XDoG uses 4GB of GPU memory at peak for 1MP resolution input, while the checkpointed version uses 2GB. The loss in speed is small and can be mitigated through the larger batch size that can be processed with the available memory.

2 Global PPN

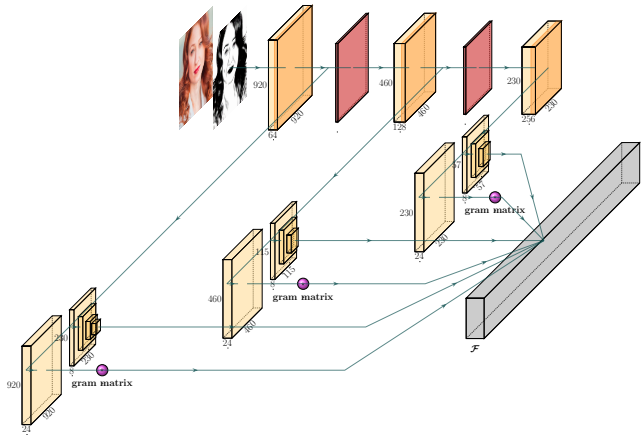
This section provides details on the global PPN, as well as an ablation study for architecture variants and losses.

2.1 Architecture

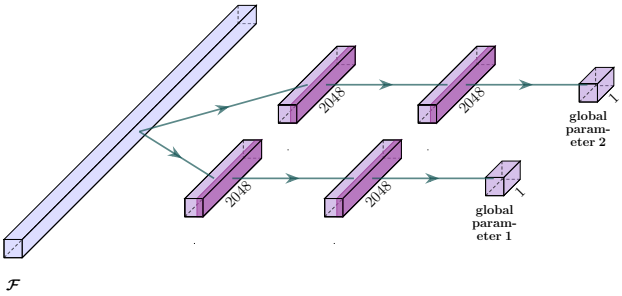
The network architecture receives two images: an input image and a stylized image, which has been generated with unknown parameters. We develop and benchmark three network architectures for parameter prediction, as follows:

1. **SimpleNet**: A simple convolution network with three Conv-ReLU layers (channel count: 16,32,64) followed by MaxPooling layers. All features are concatenated at the end and transformed with an additional linear layer to yield the global parameters. Input and stylized image are concatenated along the channel dimension and subsequently passed through the network.
2. **ResNet + Multi-head**: A ResNet50 architecture [6] without BatchNormalization layers is used to extract features. We initialize the ResNet with pre-trained weights on the ImageNet dataset [4]. Again, input and stylized image are concatenated along the channel dimension before passing them through the network. Features $\mathcal{F}$ are extracted after the last convolution layer and passed to a custom multi-head module depicted in Fig. 3b. The streams of linear layers process the features for each global parameter independently.
3. **Multi-feature + Multi-head**: A VGG backbone without BatchNormalization and pre-trained weights on ImageNet is used. Input and stylized image are passed separately through the feature extractor in two passes and all compact features are accumulated afterward. We extract features and compute Gram matrices for every convolution step as depicted in Fig. 3a. As the Gram matrices contain essential information about style, we hypothesize that these features help understand the input stylizations. The accumulated features $\mathcal{F}$ from both images are passed to the multi-head module of Fig. 3b to predict the final global parameters.

The design of the PPN architectures is motivated by two assumptions: (1) using a multi-head module for independent processing of each parameter should improve the accuracy as parameters for algorithmic effects model independent aspects of the stylization process and thus benefit from learning parameter-specific representations; (2) extraction of features at multiple scales (multi-feature) should be beneficial for the PPN's performance, as parameter changes can affect larger parts of the image as well as small details. As Huang *et al.* [7] find that normalization layers also perform normalization of style, we suspect that leaving out BatchNormalization in all architectures would yield better results



(a) Multi-feature extractor



(b) Multi-head module

Fig. 3: **Multi-feature extractor a:** Features are extracted from all convolution stages of a pre-trained VGG-11 network. We only show the first 3 convolution stages of the VGG-11 here for brevity. The features are passed through small stacks of strided convolutions with stride 4 and kernel size 4×4 pixels to generate compact representations. In parallel, the same features are passed through a single 1×1 pixel convolution and a subsequent operation to calculate the Gram matrices. All compact features $\mathcal{F}$ are concatenated. Convolution operations are visualized in yellow, ReLU layers in orange, and pooling layers in red. **Multi-head module b:** After the extraction of all features, we continue with multiple streams of linear layers. A separate stream of 3 linear layers is created for each global parameter: The first two layers process the features and have ReLU activations. The last layer outputs the final prediction as a single number. Fully connected layers are visualized in light violet and ReLU layers in dark violet.

2.2 Ablation Study

We conduct an ablation study for the previously discussed PPN architectures and loss functions. We use the XDoG filter effect [20] and compute the SSIM and PSNR metric in image space between the predicted and reference stylizations. We also indicate the mean absolute difference (ℓ_1) between the predicted and ground truth global parameters.

Loss Functions. For our ablation study, we compare parameter space- and image space losses. Parameter space losses are computed from the output of the PPN directly (i.e., in parameter space), thus the differentiable effect is not used during training. For image space losses, the predicted parameters are used to parameterize the differentiable effect, its output is then compared to the target output image using a pixel-wise similarity metric and gradients are back-propagated through the effect back to the PPN.

Network Training. For training, we use a random extract of 7000 images from the FFHQ-dataset of portrait images [12] at a resolution of 1024×1024 pixels. For data augmentation, images are randomly cropped to resolution 920×920 pixels. We use Adam [13] with a learning rate of 10^{-5} and train for 25 epochs, using a virtual batch size of 64. For testing, we use all 60 portrait images of the NPR benchmark portrait [17] and randomly generated global parameters. The final parameter predictions are passed through a tanh activation function for all networks and multiplied by 0.5 to yield parameters in the range $[-0.5, 0.5]$.

Results. Results are summarized in Tab. 2. Interestingly, the simple network architecture performs better than or similar to the ResNet in most experiments. We argue that for the ResNet, too much spatial information gets lost, as features are derived only after all downsampling and convolution operations. The multi-feature architecture effectively recovers the ability to derive representative features at all scales, as shown by Zhang *et al.* [24], and outperforms the other two options. We observe that computing the loss in image space instead of parameter space generally yields better results in terms of SSIM and PSNR, even though the ground-truth parameters are not directly available to the network in this case. As expected, computing the loss in parameter space still leads to a closer approximation of the ground truth parameters as measured by the ℓ_1 distance between ground truth and predicted parameters. If gradients are used as a learning signal, the model learns to use parameters more effectively to achieve better results, sometimes deviating more from the ground truth parameters. Some parameter changes within the XDoG effect (e.g., changing either contour or blackness) can similarly impact the final outcome. We reason that the gradients, which are derived from our differentiable effects, enable the model to build a better understanding of how the various parameters influence the final outcome.

Table 2: Global parameter prediction: ablation study for XDoG on FFHQ[12]. For network variants SimpleNet (SN), ResNet + Multi-head (R+M), Multi-feature + Multi-head (M+M), using image space-based losses $I[\ell_{1/2}]$ and parameter space-based losses $P[\ell_{1/2}]$.

Network	Loss	SSIM	PSNR	$P[\ell_1]$
		– across all NPR levels –		
M+M	$I[\ell_2]$	0.764	13.286	0.190
SN	$I[\ell_2]$	0.733	12.523	0.218
R+M	$I[\ell_2]$	0.726	12.234	0.235
M+M	$P[\ell_2]$	0.738	12.530	**0.158**
SN	$P[\ell_2]$	0.686	11.277	0.197
R+M	$P[\ell_2]$	0.677	10.874	0.205
M+M	$I[\ell_1]$	**0.780**	**13.875**	0.183
SN	$I[\ell_1]$	0.728	12.407	0.227
R+M	$I[\ell_1]$	0.721	12.038	0.236
M+M	$P[\ell_1]$	0.737	12.927	0.162
SN	$P[\ell_1]$	0.697	11.805	0.200
R+M	$P[\ell_1]$	0.692	11.408	0.200

3 Ablation study for APDrawing Stylization

Fig. 4: **Learned task seperation**. Results on APDrawing[22] after XDoG (top) and then after convolution (bottom) are shown.

Here, we provide additional details to the ablation study in the section on combining algorithmic effects and Convolutional Neural Networks (CNNs) in the main paper. The results are summarized in Tab. 3. For the ResNet-50 backbone of the PPN, we compare initialization with random weights and weights pre-trained on ImageNet. For the convolutional postprocessing network in our generator, we compare the ResNet-based network architecture of Johnson *et al.*

Table 3: Ablation study for our model on the APDrawing [22] dataset. The Fréchet Inception Distance (FID) score between the train and the test set can be used as a baseline for all results.

XDoG	PPN weights	Postrocessing Architecture	Dropout	FID score	Key in Fig. 12
✗	-	U-Net	✓	75.27	-
✗	-	U-Net	✗	71.26	-
✗	-	ResNet	✓	70.00	-
✗	-	ResNet	✗	62.44	(a)
✓	Random	U-Net	✓	86.73	-
✓	Random	U-Net	✗	89.93	-
✓	Random	ResNet	✓	71.92	-
✓	Random	ResNet	✗	**60.55**	(c)
✓	ImageNet	U-Net	✓	100.41	-
✓	ImageNet	U-Net	✗	79.56	-
✓	ImageNet	ResNet	✓	76.25	-
✓	ImageNet	ResNet	✗	64.81	-
✓	-	U-Net	✓	71.64	-
✓	-	U-Net	✗	75.40	-
✓	-	ResNet	✓	71.56	-
✓	-	ResNet	✗	73.77	-
APDrawing GAN [22]				62.14	(b)
Train vs. Test				49.72	-

[10] and a classical U-Net[16] without residual blocks following Isola *et al.* [8][4]. We also investigate the effect of dropout. As it decreases the FID compared to architectures trained without dropout, the model clearly benefits from having the full learning capacity at its disposal during training. Also, pre-trained weights on ImageNet do not increase the ability of the model to adapt to data successfully.

We also test whether parameter prediction and thus differentiable effects are necessary by comparing our method to the same approach using a fixed parameter preset, which is optimized for portrait data specifically. Our PPN network can dynamically adapt parameters locally to input images. As the results using our PPN and those with a fixed preset differ by a large margin, we conclude that parameter prediction and thus differentiable effects play an important role in the success of our method. Fig. 4 visualizes the results after the XDoG stage and after post-processing using a CNN; the learned separation of edge detection and abstraction is apparent.

We do not include results using only the differentiable XDoG effect in conjunction with a PPN. All experiments without a separate convolution network failed with mode collapse of the generator, i.e., the PPN started to predict the same parameters regardless of the input. We argue that this behavior is related to the missing ability of the XDoG effect to model the artist's style accurately.

[4] We adapt the implementation of https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

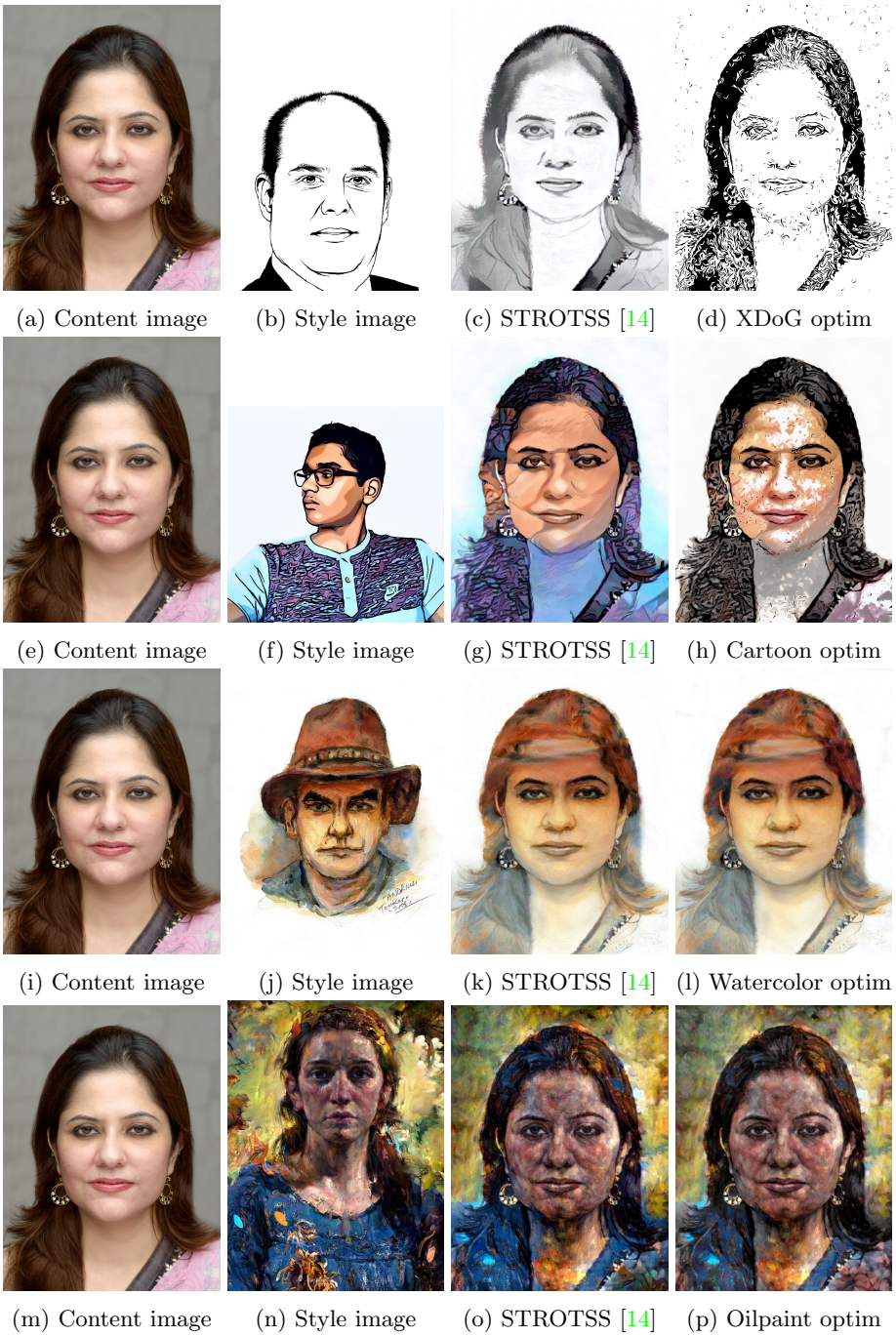
4 Style Transfer Capability

(a) Watercolor ℓ_1 optim (b) Oilpaint $\mathcal{L}_C + \mathcal{L}_S$ (c) Watercolor $\mathcal{L}_C + \mathcal{L}_S$

Fig. 5: Optimization of parametric style transfer with different losses. In a), the effect is optimized by matching the result of STROTSS style transfer [14] using an ℓ_1 loss, as described in the main paper. In b) and c), the effect is directly optimized using neural style transfer [5] losses $\mathcal{L}_C + \mathcal{L}_S$. The content/style image are the same as in the teaser figure.

In our parametric style transfer, we use Style Transfer by Relaxed Optimal Transport and Self-Similarity (STROTSS) [14] to create references which are then matched during optimization using ℓ_1 losses in image space. We also investigate directly optimizing effect parameters using the neural style transfer losses introduced by Gatys *et al.* [5]. The results are shown in Fig. 5. The optimization does not fail, however, the resulting style elements are less pronounced and more artifacts are generated. As optimizing effect parameters to affect the output is harder than direct pixel optimization (as done in NST), stricter supervision (i.e., pixel-wise losses) is necessary to achieve similar outputs. Directly using $\mathcal{L}_S + \mathcal{L}_C$ could however be appropriate for photographic style transfer.

In Tab. 2 of the main paper we compared the potential of our differentiable effects to be optimized in a (general) style transfer framework. In Fig. 6 we show example images from the in-domain optimization for the implemented effects. For this, we used content images from the NPR benchmark dataset[17] (e.g., Fig. 6a) and several style images from the same artistic domain for each effect (one example per effect is shown Fig. 6 b,f,j,n). The reference for optimization is created by a neural style transfer [14]. Note that it does not necessarily produce references that would be considered strictly in-domain with the style image (e.g., cartoon or line drawing domains) as the style transfer method has no concept of the actual drawing techniques used. The created reference (e.g., Fig. 6k) is then matched as closely as possible during local parameter optimization. As is visible, only the Watercolor and Oilpaint effects are able to produce closely matching results Fig. 6l. The Cartoon and XDoG effects can, on the other hand, only be optimized with references that are closer to their range of achievable effects and are not suited for the general style transfer case. In Fig. 7 and Fig. 8 we further show the general style transfer capability of oilpaint and watercolor on the set of common NST styles measured in Tab. 2. It is visible that oilpaint achieves a decent matching on most styles, but fails to create structure in some regions, while watercolor achieve an almost perfect matching of all references.



(a) Content image (b) Style image (c) STROTSS [14] (d) XDoG optim

(e) Content image (f) Style image (g) STROTSS [14] (h) Cartoon optim

(i) Content image (j) Style image (k) STROTSS [14] (l) Watercolor optim

(m) Content image (n) Style image (o) STROTSS [14] (p) Oilpaint optim

Fig. 6: Local parameter optimization using different algorithmic effects

Fig. 7: Style transfer capability on common NST styles.

Fig. 8: Style transfer capability on common NST styles.

5 Additional Results

This section provides additional results of our method, i.e., content-adaptive effects Sec. 5.1, and parametric style transfer Sec. 5.2. For a demonstration of interactive style transfer adjustment, please watch the provided supplemental video. Sec. 5.3 shows additional results for our model trained on APDrawing [22]. Comparisons to the state-of-the-art are shown in Fig. 12. Further, results depicted in Fig. 15 demostrates that our model generalizes to portraits from other datasets, such as the NPR benchmark[17]. Finally, we show baseline results of our implemented algorithmic differentiable effects with different parameter variants in Sec. 5.4.

5.1 Content-adaptive Effects

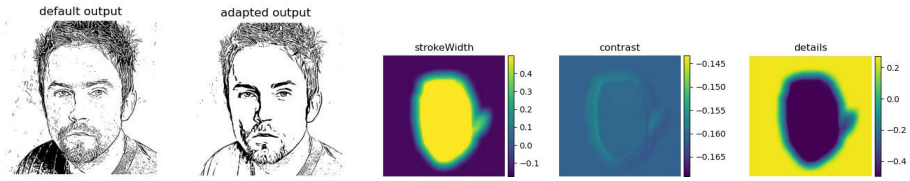


Fig. 9: Facial feature enhancement and predicted local parameter maps.

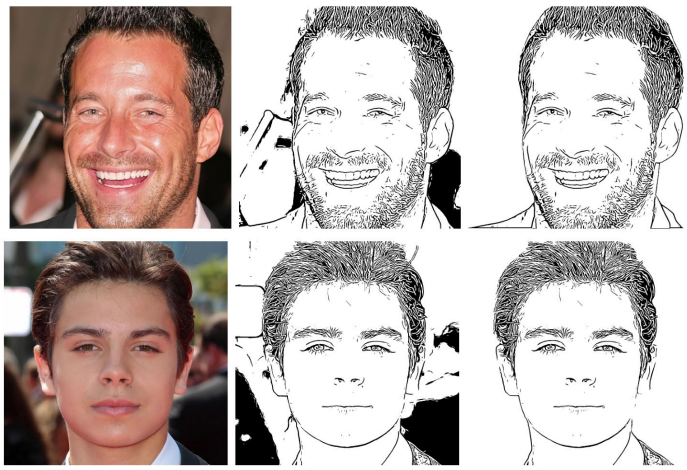


Fig. 10: Background removal. The PPN learns to reduce details and stroke-width for the XDoG filter in such a way that background details seen in the default output (middle) are removed the output with locally adapted parameters (right).

5.2 Style Transfer

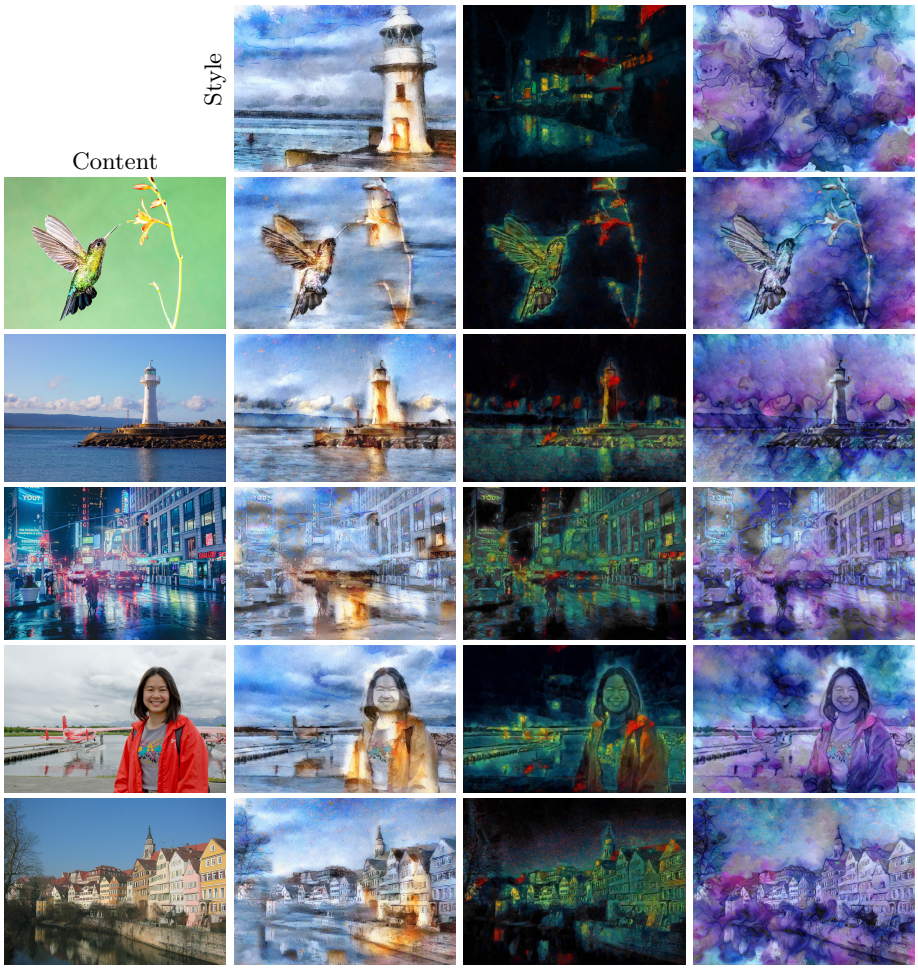


Fig. 11: Additional results for style transfer optimization. Our parametric style transfer can match a wide variety of styles and remains editable after optimization (not performed here). While the presented parameter smoothing schedule removes most artefacts stemming from parameters falling into local optima, some can remain in the final image (best seen zoomed in). These can be easily removed in a final, manual parameter editing step, as shown in the supplemental video.

5.3 Image Translation on APDrawing

(a) APDrawing GAN[22] (b) Our method (c) CNN-only (d) Ground truth

Fig. 12: Results on the APDrawing [22] dataset, our method uses XDoG and a postprocessing CNN. Our method, in contrast to APDrawing Generative Adversarial Networks (GANs)[22,23], often produces more consistent lines e.g., for the eyes, and does not need any known facial landmarks at both training and inference time.

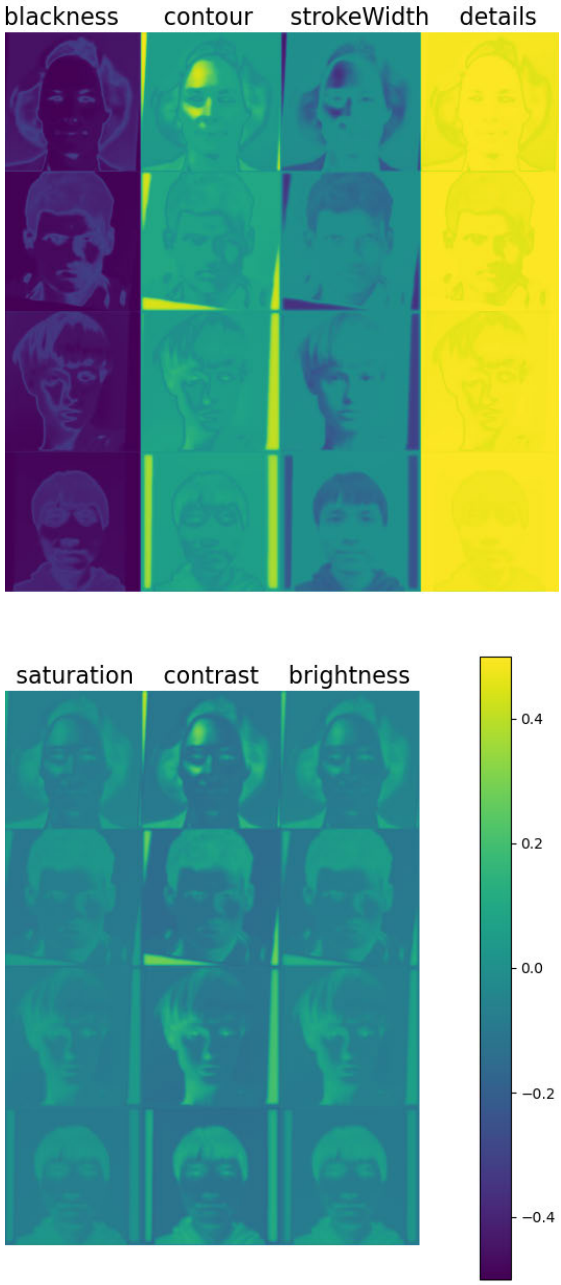


Fig. 13: In addition to the generated outputs in figure 12, we show the parameter masks that have been predicted for the first stage of our method (XDoG). See figure 4 for a plot of the intermediate results after processing with XDoG.

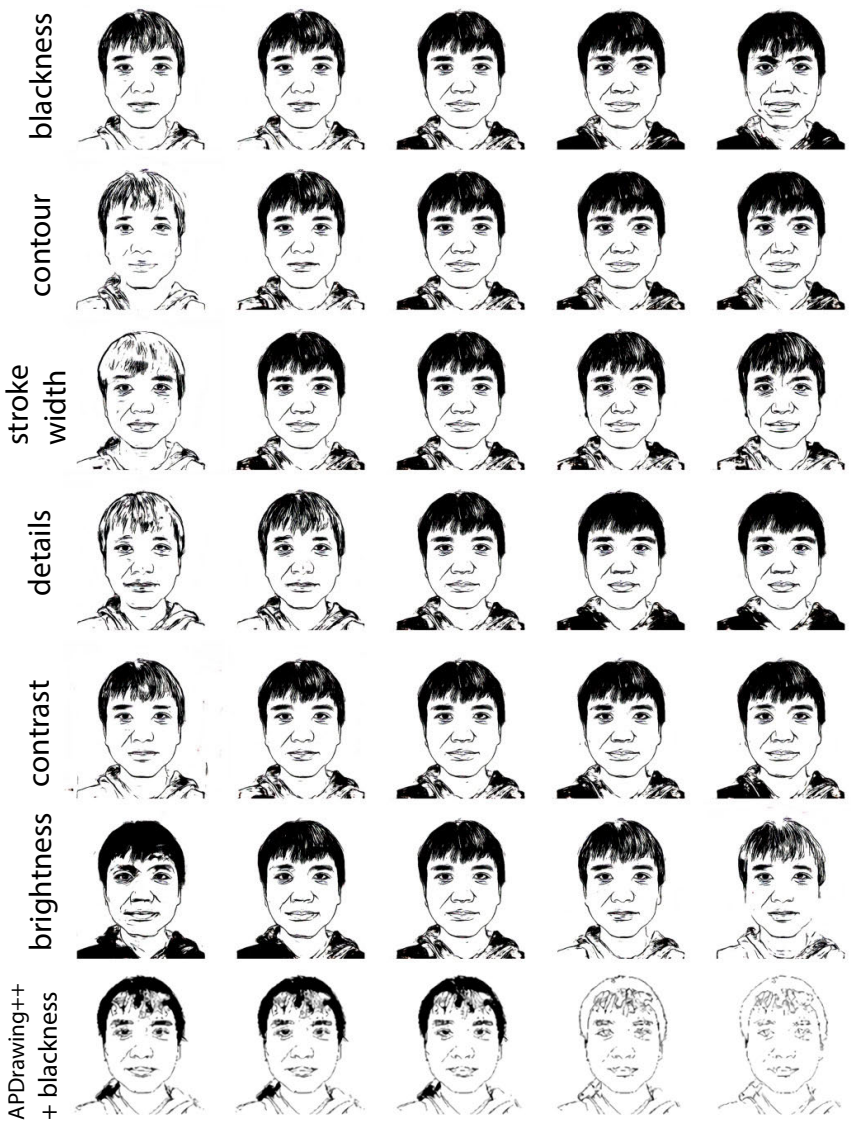


Fig. 14: Adjusting parameters of the XDoG effect after prediction using our XDoG+CNN method trained on APDrawing [22]. Each row corresponds to global changes of one parameter from lowest to highest value in the visual range, where unedited results (i.e., the predicted parameters) are shown in the middle column. While the postprocessing CNN reduces the stylistic variance that can be achieved with XDoG, results main editable vs. results of APDrawingGAN [22] and also remain stylistically close to the reference. Combining pretrained GANs such as APDrawingGAN++ [23] with a XDoG effect for postprocessing, on the other hand, does not yield visually appealing results and results deviate from the style of the reference, as shown in the last row, thus validating our integrated training approach.

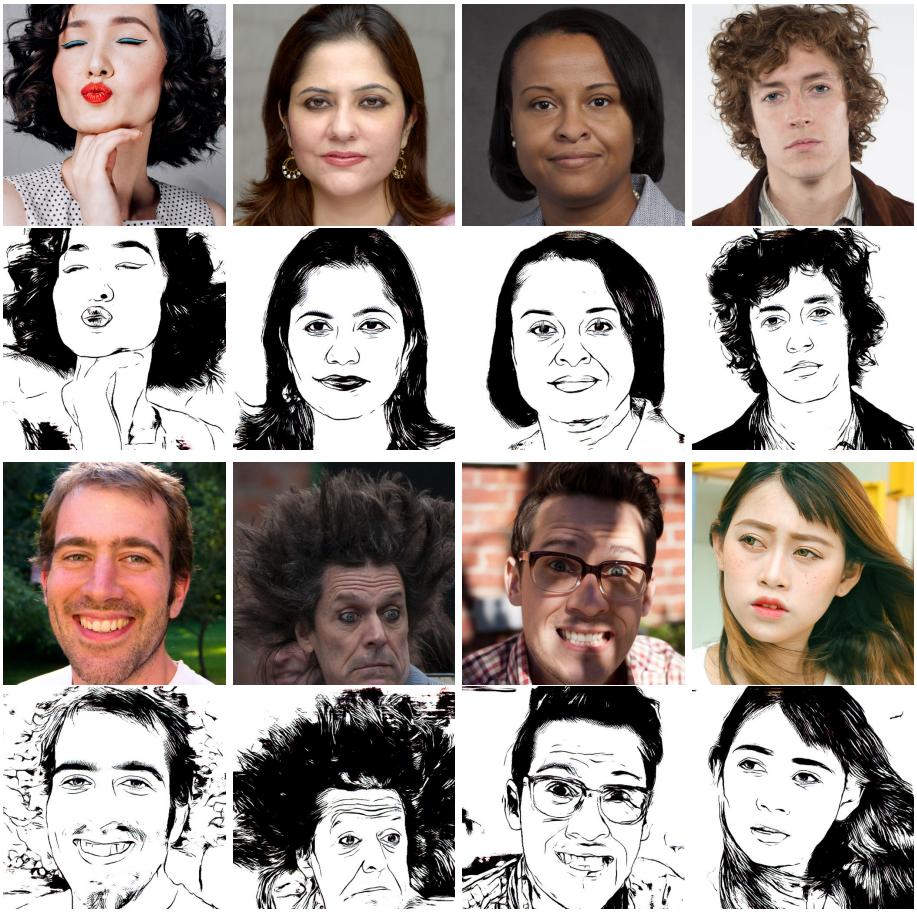
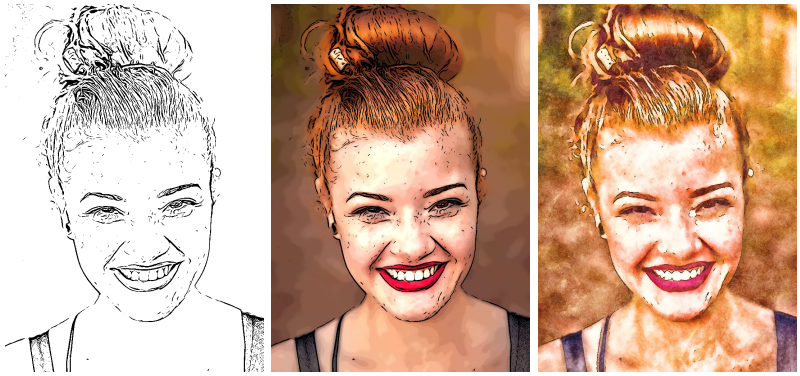
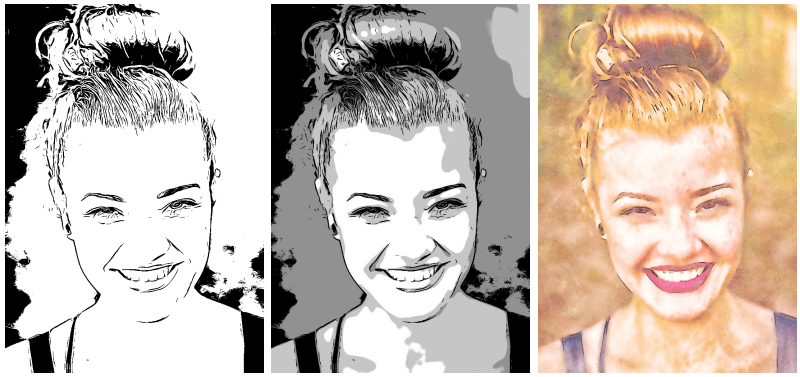


Fig. 15: Our model generalizes to example images drawn from the NPR benchmark [17] (not part of APDrawing trainset). Our method retains only salient lines in the face, abstracting irrelevant details. Furthermore, lines generally flow consistently without unnatural discontinuities. Our model has been trained on portraits with uniform background only, thus prediction on images with non-uniform backgrounds (e.g., images in bottom row) may lead to background artefacts in the stylization.

5.4 Effect Variants



(a) XDoG variant A (b) Cartoon variant A (c) Watercolor variant A



(d) XDoG variant B (e) Cartoon variant B (f) Watercolor variant B

(g) Oilpaint variant A (h) Oilpaint variant B

Fig. 16: A selection of global parametrization variants of our differentiable effects are shown. These represent default states of the effect without any parameter learning applied.

References

1. Yoshua Bengio, Nicholas Leonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3
2. Thomas Brox, Rein Van Den Boomgaard, François Lauze, Joost Van De Weijer, Joachim Weickert, Pavel Mrázek, and Pierre Kornprobst. Adaptive structure tensors and their applications. In *Visualization and Processing of Tensor Fields*, pages 17–47. 2006. 2
3. Benoit Brummer and Christophe De Vleeschouwer. Natural image noise dataset. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1777–1784, 2019. 5
4. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 6
5. Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016. 11
6. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 6
7. Xun Huang and Serge Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, pages 1501–1510, 2017. 6
8. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1125–1134, 2017. 10
9. Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2017–2025, 2015. 2
10. Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *Proc. European Conference on Computer Vision (ECCV)*, pages 694–711, 2016. 10
11. Henry Kang, Seungyong Lee, and Charles K. Chui. Flow-Based Image Abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):62–76, 2009. 2
12. Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4401–4410, 2019. 8, 9
13. Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proc. International Conference on Learning Representations (ICLR)*, 2015. 8
14. Nicholas Kolkin, Jason Salavon, and Gregory Shakhnarovich. Style Transfer by Relaxed Optimal Transport and Self-Similarity. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10051–10060, 2019. 11, 12, 13, 14
15. Jan Eric Kyprianidis and Jürgen Döllner. Image Abstraction by Structure Adaptive Filtering. In *Proc. EG UK Theory and Practice of Computer Graphics (TPCG)*, pages 51–58, 2008. 2
16. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Proc. International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 234–241, 2015. 10

17. Paul L. Rosin, Yu-Kun Lai, David Mould, Ran Yi, Itamar Berger, Lars Doyle, Seungyong Lee, Chuan Li, Yong-Jin Liu, Amir Semmo, Ariel Shamir, Minjung Son, and Holger Winnemöller. NPRportrait 1.0: A three-level benchmark for non-photorealistic rendering of portraits . *Computational Visual Media*, 8(3):445–465, 2022. 4, 8, 11, 15, 20
18. Amir Semmo, Daniel Limberger, Jan Eric Kyprianidis, and Jürgen Döllner. Image Stylization by Interactive Oil Paint Filtering. *Computers & Graphics*, 55:157–171, 2016. 4
19. Carlo Tomasi and Roberto Manduchi. Bilateral Filtering for Gray and Color Images. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, pages 839–846, 1998. 1
20. Holger Winnemöller. XDoG: Advanced Image Stylization with eXtended Difference-of-Gaussians. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 147–156, 2011. 8
21. Holger Winnemöller, Sven C Olsen, and Bruce Gooch. Real-Time Video Abstraction. *ACM Transactions On Graphics (TOG)*, 25(3):1221–1226, 2006. 3
22. Ran Yi, Yong-Jin Liu, Yu-Kun Lai, and Paul L Rosin. APDrawingGAN: Generating Artistic Portrait Drawings from Face Photos with Hierarchical GANs. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10743–10752, 2019. 1, 9, 10, 15, 17, 19
23. Ran Yi, Mengfei Xia, Yong-Jin Liu, Yu-Kun Lai, and Paul L Rosin. Line Drawings for Face Portraits from Photos using Global and Local Structure based GANs. In *Proc. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020. 17, 19
24. Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–595, 2018. 8