

# Neural Neighbor Style Transfer

Nick Kolkin<sup>1,2</sup> Michal Kučera<sup>3</sup> Sylvain Paris<sup>2</sup>  
 Daniel Sýkora<sup>3</sup> Eli Shechtman<sup>2</sup> Greg Shakhnarovich<sup>1</sup>

Toyota Technological Institute at Chicago<sup>1</sup> Adobe Research<sup>2</sup>  
 Czech Technical University in Prague<sup>3</sup>

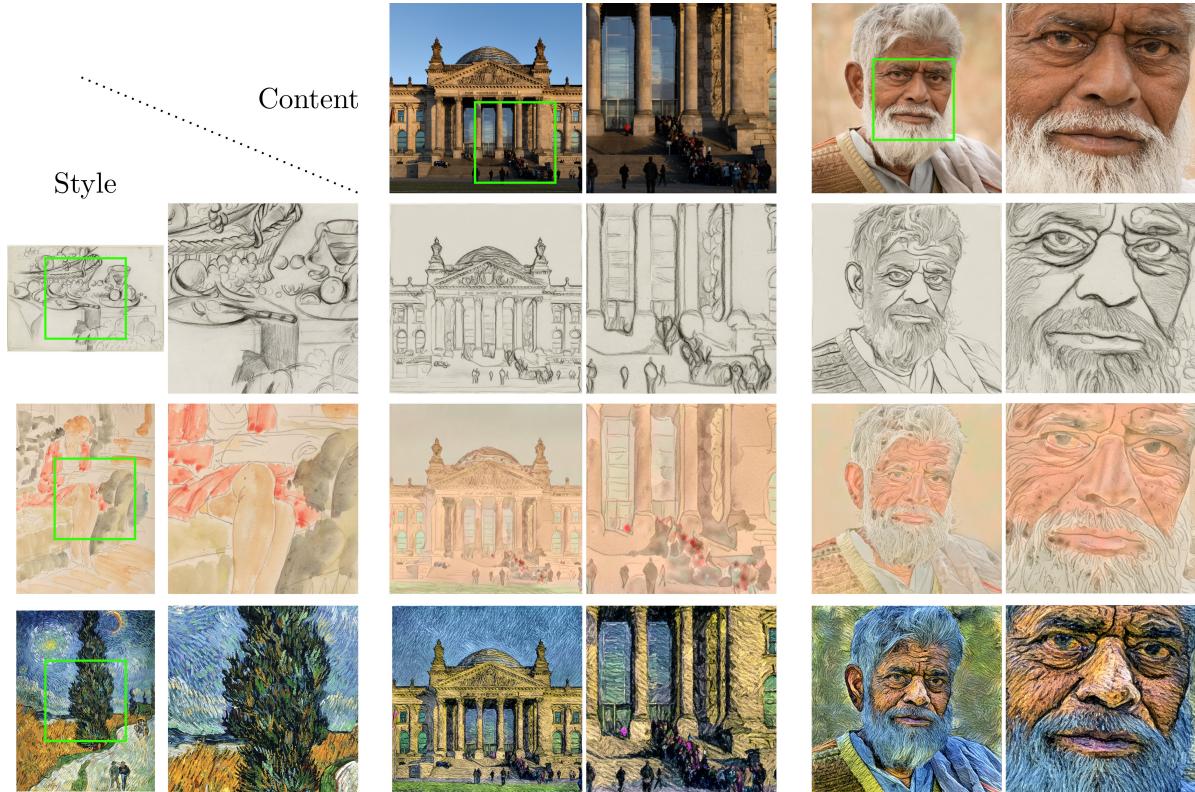


Figure 1. Examples at 1k resolution of our proposed method, Neural Neighbor Style transfer. Our method synthesizes a stylized output based on rearranging features extracted from the target style using a pre-trained CNN. Synthesis can be implemented either as direct optimization of output pixels (pictured above), or as inference of pixels from features by a learned decoder. Code is available [here](#)

## Abstract

We propose Neural Neighbor Style Transfer (NNST), a pipeline that offers state-of-the-art quality, generalization, and competitive efficiency for artistic style transfer. Our approach is based on explicitly replacing neural features extracted from the content input (to be stylized) with those from a style exemplar, then synthesizing the final output based on these rearranged features. While the spirit of our approach is similar to prior work, we show that our design decisions dramatically improve the final visual quality.

There are two variants of our method. NNST-D uses a CNN to directly decode the stylized output from the rearranged style features; it offers similar or better quality than much slower state-of-the-art optimization based methods, and outperforms prior fast feed-forward methods. This version takes a few seconds to stylize a  $512 \times 512$  pixel output, fast enough for many applications. NNST-Opt, our optimization-based variant offers higher quality, albeit at lower speed, taking over thirty seconds on the same input size. We compare the stylization quality of both NNST variants with prior work qualitatively and via a large user study

with 400 participants which confirms superior performance of our approach. We also demonstrate that NNST can be used for video stylization or extended to support additional guidance and higher output resolution.

## 1. Introduction

Thanks to recent advances in neural style transfer pioneered by Gatys et al. [11] input photos can be turned into a stylized artwork that has a similar content while mimicking distinctive visual features of the style exemplar. The great advantage of neural methods is that they usually do not require additional information to perform the stylization, only the content and the style image need to be specified by the user. In contrast, traditional patch-based approaches [9, 16] require additional guidance prepared manually or automatically generated to achieve convincing results. Due to this requirement, patch-based methods usually expect a specific target domain (e.g., example-based stylization of 3D models [9] or faces [10]) where automatic computation of guidance channels is feasible. Or in a more generic scenarios such as video stylization, they need style exemplars to be precisely aligned with the target content to establish a one-to-one mapping between the content and style [19]. However, a key benefit of patch-based techniques is their ability to accurately preserve visual aspects of the given artistic media, including the notion of individual brush strokes or other details vital to matching the style of the original artwork. This is possible due to the non-parametric nature of patch-based methods, i.e., their ability to generate stylized images by stitching a set of smaller bitmap chunks copied directly from the style exemplar. In contrast, state-of-the-art neural techniques such as [24] use parametric synthesis where the output image is generated by directly optimizing the values of the output pixels. In this scenario it is difficult to compete with the ability to reuse exact patches from the style exemplar. Historically this has resulted in a visual gap between the style preservation quality of neural and patch-based techniques. On the other hand, besides patch-based method’s need for guidance, relying on patches also limits the output’s degrees of freedom since such a method can only copy and shift larger chunks of the original style exemplar. To alleviate such drawback, recent style transfer methods propose to combine aspects of both directions [13, 26, 30, 47], i.e., use either non-parametric approach in the space of neural features and then perform parametric synthesis to obtain the final image or vice versa. For instance, Deep Image Analogy of Liao et al. [30] delivers results approaching quality of patch-based synthesis [10] without the need to prepare extra guidance, however, it still requires the style image to have a similar content as the target image.

In this work our aim is to further elevate the visual qual-

ity of neural style transfer without requiring extra guidance or domain specific style exemplars. We adopt the combined approach to style transfer, i.e., we construct target neural features by rearranging features of the style image and then we find an image which produces these features either by optimization or a learned decoder network. Our main contribution is hidden in the feature construction phase. Here we demonstrate how to find a good balance between the overall visual diversity, stylistic fidelity, and content preservation by combining three improvements: (1) zero-centering the content and style features, (2) using the cosine distance for measuring feature similarity, (3) performing feature splitting during the nearest-neighbour matching phase which is computed in every iteration of the target feature construction, and most importantly (4) matching individual neural feature vectors rather than patches as proposed in [4, 26]. In our evaluations and ablation study we demonstrate how those four components enable significant improvement over current state-of-the-art. Qualitatively our results more accurately capture the texture of the target media than prior work, particularly when seen at high resolution.

## 2. Related Work

Example-based style transfer, non-photorealistic rendering guided by a single piece of artwork, is a widely studied image synthesis task. Approaches like our own, in which image synthesis is guided by explicit matches between spatially localized content and style features, can be traced to [16] and related work on texture synthesis [7, 8, 49]. Recent patch-based style transfer algorithms have focused on producing high quality outputs when additional guidance is available, either in the form of boundary annotations [32, 33], or a set of custom tailored guiding channels [2, 9, 10, 19]. These methods, however, are applicable only in scenarios when such guidance can be provided either manually or computed automatically. Such requirements are usually violated in our arbitrary style transfer setting where there are no assumption imposed on the input style image.

**Optimization-based Neural Style Transfer:** Recent years have marked a significant departure from this line of work, building off techniques pioneered by Gatys et al. [11] in their ‘A Neural Algorithm of Artistic Style’. This work contributed in two key ideas. Firstly, it shows how to leverage features extracted by a convolutional neural network pre-trained for image classification tasks (typically VGG [44]) as a way how to measure high-level compatibility between two images. The second idea was direct optimization of the output’s pixels using gradient descent to simultaneously minimize a ‘style loss’ (matching statistics derived from the features of the VGG artwork), and a ‘content loss’ (minimizing deviation from the content image’s

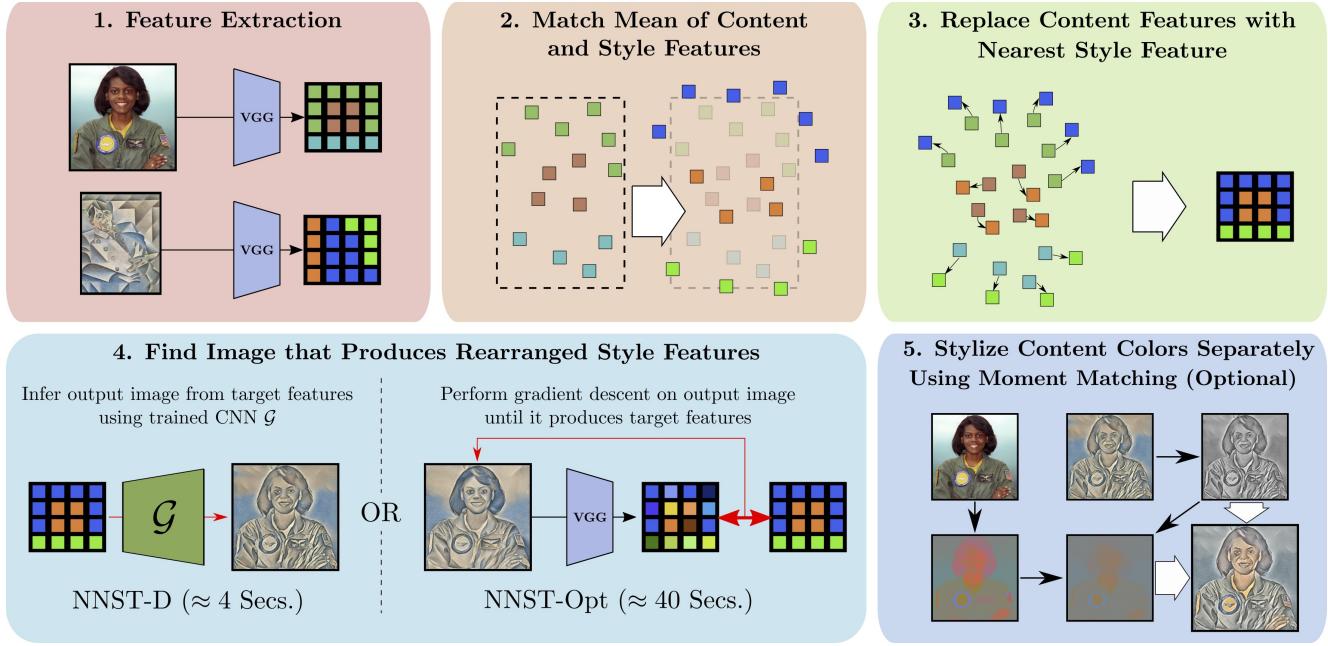


Figure 2. Overview of our method. The fast and slow variants of our method, NNST-D and NNST-Opt, only differ in step 4; mapping from the target features to image pixels. This simplified diagram omits several details for clarity, namely: we apply steps 1-4 at multiple scales, coarse to fine; we repeat steps 1-4 several times at the finest scale; and we only apply step 5 once (optionally) at the very end.

features). Many follow-up works [3, 11, 13, 24, 26, 35, 41] proposed alternative style losses or introduced content loss invariant to translations and rotoreflections in the feature space [24].

In the arbitrary style transfer setting, these approaches perform better than patch-based techniques, nevertheless, methods that use a content and style loss, both based on VGG features, face a fundamental challenge. In general it is impossible to match the statistics of the style features (satisfying the style loss) while keeping the content features unchanged (satisfying the content loss). Even if the content loss is invariant to rotoreflections [24], matching the second order statistics captured by the simplest original style loss [11, 27] generally requires an affine transformation (of which rotoreflections are a subset, and therefore do not provide sufficient invariance). In practice we observe that content losses based on deep VGG layers tend to cause photographic high frequencies of the original content to bleed into the final output.

To avoid such a drawback, our proposed algorithm borrows the idea of guided patch-based synthesis to explicitly match content and style features, however, instead of working directly on image patches it uses features extracted from the responses of pre-trained VGG network.

Similar approach was used previously in CNNMRF [26] that replaces style loss of [11] with minimizing each patch of content feature's distance from its nearest neighbor patch of style features under the cosine distance. However, it

also regularizes its outputs using the content loss proposed in [11], leading to the fundamental tension outlined above. Other approaches tried to explicitly construct a set of target features without considering content loss. In [4, 26] overlapping feature patches from the style image are averaged. This approach preserves content well, however, at the cost of fidelity to the target style. Gu et al. [13] add a soft constraint on the matches found by nearest neighbors that each style vector may only be used at most  $k$  times. Nevertheless, this constraint can be overly restrictive and lead to content distortion artifacts. Liao et al. [30] proposed a coarse-to-fine strategy for finding feature correspondences between the content and style image. Their approach produces excellent results on style-content pairs with matching semantics and similar poses (and based on our user study, ink based styles using sparse lines), but in general does not work for more disparate style-content pairs (see Figure 4). Finally, Texler et al. [46] propose a method based on explicit neural feature matching to guide patch-based synthesis. However, since the final output can only be a mosaic of small bitmap chunks taken from the original style image the flexibility to express different target content can be fairly limited.

**Feed-Forward Neural Style Transfer:** A shortcoming of optimization-based style transfer methods is their computational overhead. To address this a large body research has focused on training feed-forward networks which can perform style transfer quickly [1, 5, 6, 17, 29, 38, 43, 50, 51]. Many of these techniques use closed-form modifications of

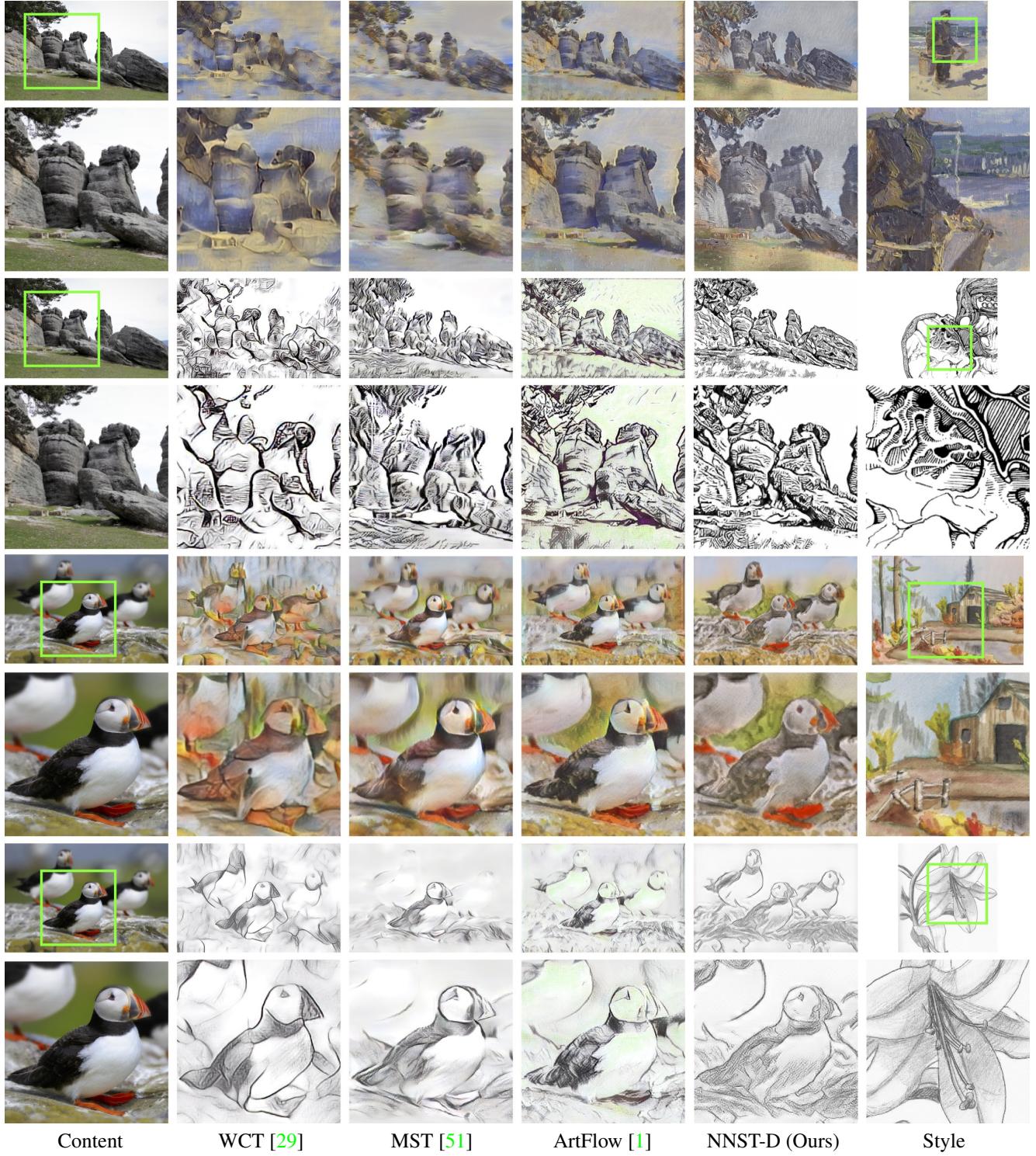


Figure 3. Qualitative comparison between NNST-D and the top three feed-forward methods from our user study, using oil painting, ink, watercolor, and pencil styles. Below each input and result is a zoomed-in portion of the image. While all neural methods to date fail to entirely capture many styles' long range correlation of textural features and high frequency details, our results are dramatically closer than prior work.



Figure 4. Qualitative comparison between NNST-Opt and the top three optimization based methods from our user study, using oil painting, ink, watercolor, and pencil styles. Below each input and result is a zoomed-in portion of the image. While all neural methods to date fail to entirely capture many styles' long range correlation of textural features and high frequency details, our results are dramatically closer than prior work.

features extracted from the content image based on the first and second order statistics of the style features [5, 17, 29]. Sheng et al. [43] propose a ‘style decorator’ to hallucinate alternate content features more amenable to stylization. Chiu et al. [6] replace single stage moment matching with an efficient method for iterative stylizing the content features via analytical gradient descent. Yao et al. [50] uses self-attention to improve content preservation in perceptually important regions and Park et al. [38] use self-attention to distribute localized style features according to the content’s self-similarity. Zhang et al. [51] improve stylization quality by relaxing the assumption that the feature distributions are uni-modal, and matching clusters of features using a graph cut. Recently, An et al. [1] uses normalizing flows to stabilize style transfer, improving stylization quality and preventing results from changing after multiple rounds of stylization. However, because these methods either learn to extract information from the style image, or only have access to limited feature statistics, they often struggle to reproduce distinctive elements of the target style. This gives them a disadvantage relative to optimization-based methods, which ‘learn from scratch’ for each new input image pair, and consequently the visual quality of the feed-forward results is lower. By taking as input a large tensor of rearranged style features extracted by pre-trained VGG, NNST-D has direct access to much richer information about the target style.

**Targeted Feed-Forward Neural Style Transfer:** Another common scenario, which we do not address in this work, is when the styles of interest are known beforehand, and a neural network can be pre-trained to produce stylizations of the predetermined type(s) [20, 21, 25, 42, 45, 52]. These methods are very fast, and can produce high quality results. However, they require enough training data for a given style, and must be retrained for new styles.

### 3. Neural Neighbor Style Transfer

We implement our method using the PyTorch framework [40]. The feed-forward variant of our method, NNST-D takes 4.5 seconds to process a pair of 512x512 content/style images. Our optimization based variant, NNST-Opt, takes 38 seconds to process the same input. Timing results are based on an NVIDIA 2080-TI GPU.

#### 3.1. Feature Extraction

Our pipeline relies on a pre-trained feature extractor  $\Phi(x)$ , where  $x$  is an RGB image.  $\Phi(x)$  extracts the hypercolumns [14, 36] formed from the activations produced for convolutional layers in the first four blocks of pre-trained VGG16 [44] when  $x$  is passed in. We use bilinear interpolation on activations from all layers to give them spatial resolution equal to one quarter of the original image. For an image with height  $H$ , and width  $W$ , this yields an image rep-

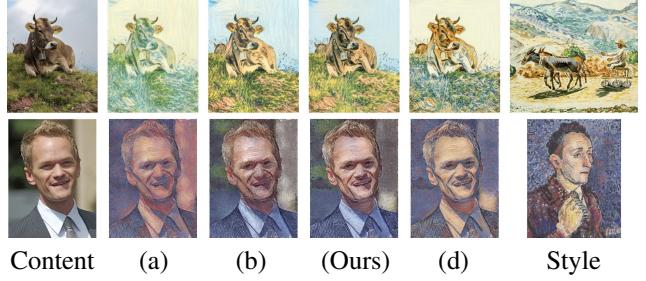


Figure 5. Demonstration of the affect of zero-centering features before nearest-neighbor matching. In (a) there is no zero-centering and no color processing, resulting in lower quality feature pairings that lead to more homogeneous colors and worse content preservation. (b) largely fixes (a) by adding color processing, although the features of the face are poorly defined. (Ours) is the default setting of NNST which uses zero-centering and color-correction, producing good results in both cases. (d) is the same as (Ours) but with no color processing, this setting is less reliable and more prone to introducing visual errors than (Ours), but when colors are mapped correctly the results can be stunning and zero-centering makes this more frequent. As zero-centering is efficient and we never observe it to hurt results, we always include it.

resentation  $\Phi(x) \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times 2688}$ . Generally we consider style to be rotation invariant, and to reflect this we extract features from the style image rotated at  $0^\circ, 90^\circ, 180^\circ$  and  $270^\circ$  in all experiments.

#### 3.2. Feature Matching

The core steps of our pipeline are outlined in Figure 2. We extract features from the style image and content image (1) zero-center the content features and style features (2). Then use nearest-neighbors matching under cosine distance (3) to replace each content feature (hypercolumn) with the closest style feature. If the content image  $C$  is of size  $H_c \times W_c$ , and style image  $S$  is of size  $H_s \times W_s$ , this yields a new target representation for our stylized output  $T \in \mathbb{R}^{\frac{H_c}{4} \times \frac{W_c}{4} \times 2688}$  where the feature vector  $T_i \in \mathbb{R}^{2688}$  at each spatial location is derived from the original style image, or a rotated copy. For simplicity let  $\Phi'(x)$  be the function extracting features from  $x$  and its rotations, where an individual feature vector (from any spatial location in any rotation) can be indexed as  $\Phi'(x)_j$ . Formally:

$$T_i = \underset{\Phi'(S)_j}{\operatorname{argmin}} D(\Phi(C)_i - \mu_C, \Phi'(S)_j - \mu'_S) \quad (1)$$

Where  $D$  is the cosine distance,  $\mu_C$  is the average feature extracted from the content image, and  $\mu'_S$  is the average feature extracted from the style image and its rotated copies. While mean subtraction does not have a huge impact when using our color post-processing, it extremely important without it, enabling some stunning results in cases

where the content and style are well matched (Figure 5).

In sections 3.3 and 3.4 we describe our feed-forward and optimization based methods for recovering image pixels from  $T$  (choosing between these differentiates between NNST-D and NNST-Opt). In the main loop of our pipeline we produce stylizations at each scale, coarse to fine, and each result is used to initialize the next scale. Throughout this process we match hypercolumns wholesale, and keep the  $T$  unchanged throughout the synthesis process at a particular scale. While this is efficient (since  $T$  need only be computed once per scale, and computing a single large distance matrix is well suited to GPU parallelism), and the result roughly captures many aspects of the target style, stopping at this point leads to images that are fail to capture the high-frequencies of the target style (Figure 6).

We believe that this effect is due to incompatible hypercolumns, which are not adjacent in the original style, being placed next to each other in  $T$ . Because these features have overlapping receptive fields, the output is optimized to produce the average of several features (each taken from a different region of the style) at a single output location. This manifests visually as a ‘washed out’ quality, an issue noted by prior work in style transfer [13], and other patch-based synthesis work [9, 18, 22].

We find that these issues can be largely resolved by a final phase where the the feature matching process is less constrained, a similar solution to one used by Luan et al. in the image compositing [31]. In this final phase, which we call ‘Feature Splitting’, matches are computed for each layer separately, resulting in  $T$  consisting of novel hypercolumns where features at different layers are mixed and matched from different locations/rotations of the style image. Unlike [31] we do not compute matches only once, we recompute them after every update to the output image. In this phase features are matched relative to the current output, rather than the initial content. When using our learned decoder  $\mathcal{G}$  to synthesize, this amounts to feeding the output back into the same network as ‘content’ five times (recomputing  $T$  each time). When directly optimizing the output image, this amounts to recomputing  $T$  using the current output as the ‘content’ after each Adam update.

### 3.3. Neural Network Decoder (NNST-D)

#### 3.3.1 Architecture

The decoder  $\mathcal{G}$  takes as input the target representation  $T \in \mathbb{R}^{\frac{H_c}{4} \times \frac{W_c}{4} \times 2688}$ .  $T$  is then fed into 4 independent branches, each responsible for producing one level of a 4-level laplacian pyramid parameterizing the output image. Each branch has virtually the same architecture (but separate parameters), consisting of five 3x3 convolutional layers with leaky relu [34] activations (except the last layer, which is linear), and a linear residual 3x3 convolution [15] directly from  $T$

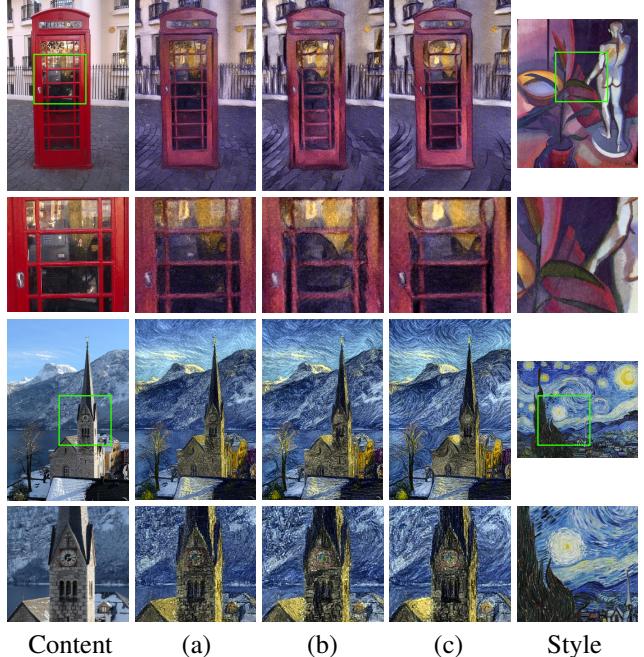


Figure 6. Demonstration of the effect of our final feature splitting phase (c). (a) is our result without any final phase, content is well preserved, but too many photographic details are preserved and brushstrokes in the 2nd row are poorly defined. (b) mimics our feature splitting phase, recomputing feature matches many times, but uses hyper-columns instead of computing matches separately for each layer. This leads to unnecessary loss of content details and muddier high frequencies relative to (c).

to the branch’s output. All intermediate hidden states have 256 channels. The four branches differ only in number of output channels, having  $4^2 \times 3$ ,  $2^2 \times 3$ , 3, and 3 output channels respectively. Transposed convolutions are applied to the first two branches to trade off channel depth for resolution (resulting in one  $H_c \times W_c \times 3$  output and one  $\frac{H_c}{2} \times \frac{W_c}{2} \times 3$  output). The third branch is not altered (resulting in a  $\frac{H_c}{4} \times \frac{W_c}{4} \times 3$  output), and the output of the fourth branch is bilinearly downsampled by a factor of two (resulting in a  $\frac{H_c}{8} \times \frac{W_c}{8} \times 3$  output). The final output image is synthesized by treating the output of the four branches as levels of a laplacian pyramid and combining them appropriately.

#### 3.3.2 Training

We train our model using MS-COCO as a source of content images, and Wikiart as a source of style images, matching the training regime of [1, 38, 51]. Content/Style training pairs are randomly sampled independently from each dataset. For each input pair, two outputs are generated during training, a reconstruction of the style image and a style

transfer:

$$\hat{S} = \mathcal{G}(\Phi(S)) \quad (2)$$

$$\hat{C}_S = \mathcal{G}(T) \quad (3)$$

Recall that an intermediate output of  $\mathcal{G}$  is a laplacian pyramid that is collapsed to form the final output image. Let the levels of this pyramid for the style reconstruction be  $\hat{S}_{1..4}$ , let a 4-level laplacian pyramid constructed directly  $S$  be  $S_{1..4}$ , let  $P_i$  be the number of pixels at level  $i$ . These are used to compute the reconstruction loss:

$$\mathcal{L}_r = \sum_{i=1}^4 \frac{\|S_i - \hat{S}_i\|_1}{P_i} \quad (4)$$

We do not know what the pixels of the style transferred result should be, so we instead optimize this output using a cycle loss on  $T$ :

$$\mathcal{L}_{cycle} = \frac{16}{H_C W_C} \sum_{i=0}^{16} D(T_i, \Phi(\hat{C}_S)_i) \quad (5)$$

Where  $D$  computes the cosine distance, and  $i$  indexes over the spatial indexes of  $T$  and  $\Phi(\hat{C}_S)$  (which are both a quarter of the original resolution of  $C$ ). To further improve the 'realism' of our results and encourage better capturing the target style we also employ the adversarial patch co-occurrence loss proposed by Park et al. [39]:

$$\mathcal{L}_{adv} = -\log \mathcal{D}(\Theta^{(4)}(S), \Theta^{(1)}(\hat{C}_S)) \quad (6)$$

Where  $\Theta^{(k)}(x)$  is a function that extracts  $k$  random patches of size  $\frac{\max(H,W)}{8}$  from  $x$ , and  $H, W$  are the height and width of  $x$  respectively.  $\mathcal{D}$  is a discriminator that evaluates a single patch, conditioned on 4 patches extracted from the style image. We use the same discriminator architecture and discriminator training described in [39], where further details can be found. We fit the parameters of our model,  $\theta_G$  to minimize the full objective:

$$\min_{\theta_G} \mathbb{E}_{C \sim \mathbb{P}_C, S \sim \mathbb{P}_S} [\mathcal{L}_r + \mathcal{L}_{cycle} + \mathcal{L}_{adv}] \quad (7)$$

Where  $\mathbb{P}_C, \mathbb{P}_S$  are the distributions of content and style training images respectively. A separate decoder is trained for each output scale (64, 128, 256, 512 pixels on the long side). Training converges fairly quickly, and we use models trained for a three epochs on MS-COCO (Wikiart images are sampled independently with replacement for each MS-COCO example). We train using a batch size of 4, and the Adam optimizer [23] with parameters  $\eta = 2e^{-3}, \beta_1 = 0.0, \beta_2 = 0.99$ . The same set of decoder models (four total, one for each scale) are used in all experiments.

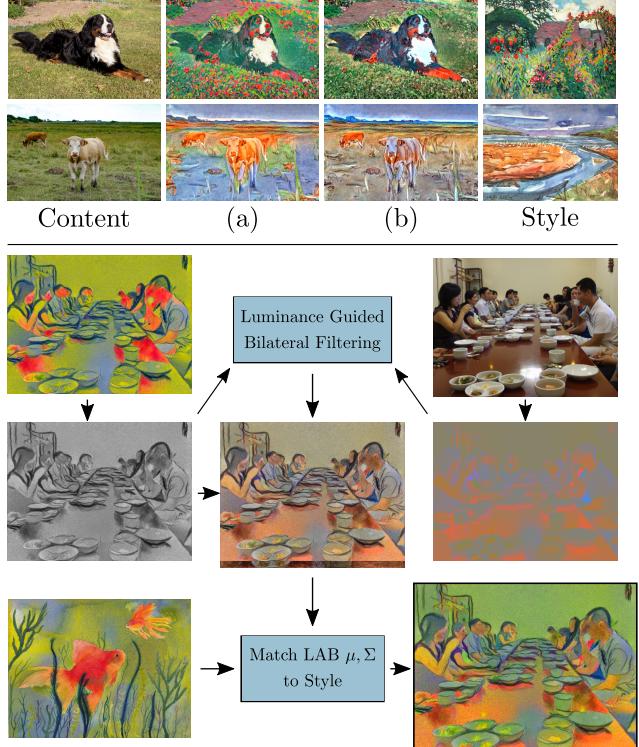


Figure 7. In the first two rows we give examples of NNST-Opt with color processing (b) and without (a). Color processing helps fix common content preservation errors due to features with inconsistent colors being matched to the same object or region. Below these examples we outline our the color processing procedure. First the stylized luminance produced by optimization or the decoder is extracted. Then it is used as a guide for bilateral filtering on the original content’s AB channels, this aligns the boundaries of colored regions to the stylized L channel. After combining this with the stylized L channel with the filtered AB channels we use simple moment matching to align the output’s color distribution with the style.

### 3.4. Image Optimization (NNST-Opt)

While performing style transfer using  $\mathcal{G}$  is fast, there are many cases where optimizing the output image directly produces sharper results with fewer artifacts. Given our target features  $T$  and feature extractor  $\Phi$ , we find output image  $x$  by minimizing following objective:

$$\min_x -\frac{1}{P} \sum_{i=0}^{P-1} \cos(\Phi(x)_i, T_i) \quad (8)$$

where  $P = W_c H_c / 16$ , the number spatial locations in  $\Phi(x)$  and  $T$ .

Equation 8 is minimized via 200 updates of  $x$  using Adam [23] with parameters  $\eta = 2e^{-3}, \beta_1 = 0.9$ , and  $\beta_2 = 0.999$ . To allow the average color of large regions to quickly change within 200 updates, we parameterize  $x$  as

a laplacian pyramid with 8 levels.

### 3.5. Color Post-Processing

We find that a common source of perceptual errors in the outputs produced by NNST-D, NNST-Opt, and other methods, is when a region with a single color in the original content is mapped to multiple colors in the output (see the first two rows of Figure 7). However, after converting our outputs to Lab colorspace, the luminance channel generally matches the target style well, and is free of artifacts. This motivates our post-processing step, which is to take the luminance generated by NNST-D or NNST-Opt, but discard the AB channels and replace them with the AB channels of the original content. In order to match the content’s original AB channels to the generated L channel we perform bilateral filtering [37, 48] on the AB channels guided by the L channel. Then we match the mean and covariance of color distribution formed by the output’s L channel and filtered AB channels to the style’s color distribution. (see the bottom section of Figure 7). This is similar to the color control proposed in [12], but allows stylizations which more closely match the palette of the target style.

In the vast majority of cases this post-processing step improves results, however we have observed a few scenarios where it does not. First, for apparently monochrome styles (e.g., pencil or pen drawings), visually imperceptible color variations in the style can distort the second-order statistics used in moment matching, leading to results where desaturated colors are visible. Fortunately in these cases the unprocessed results of NNST are typically monochrome, and we detect this situation by examining the maximum between the variance of the A and B channels, then not applying the color processing if the value is below a threshold (we find 4e-5 to work well).

Second, this post-processing step can prevent the output from matching distinctive color features of the style, such as local color variations as in the dots of pointillism, the abrupt color shifts within objects of cubism, or the limited multi-modal palettes used in some artwork (see Figures 5 and 8). Third, colorful styles with a large white background can lead to moment matching causing over-saturation. In these cases it can be better to not use this post-processing step.

Fortunately, our post-processing step is simple and computationally efficient, taking less than a millisecond in our PyTorch implementation. In a practical setting it is essentially free for users to generate results both with and without the post-processing, and choose the one which best suits their needs.

### 3.6. Control of stylization degree

Including or omitting our color processing is an important mechanism for trading off between content preserva-

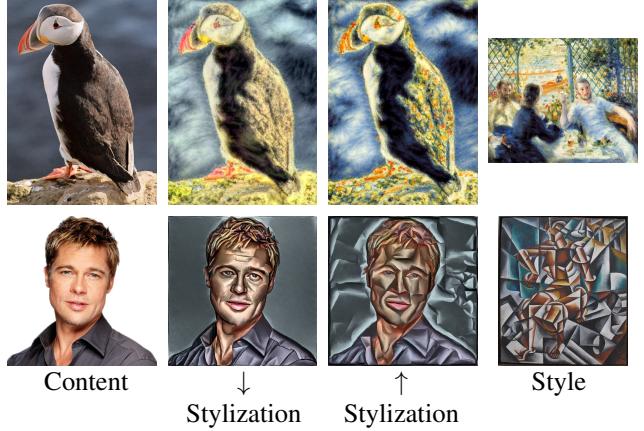


Figure 8. Our method has two mechanisms for controlling the output’s level of stylization. In the first row we demonstrate the effect of increasing stylization by omitting our color post-processing, which increases stylization by allowing greater variation from the hue and chroma of the original content image. In the second row we demonstrate the effect of varying  $\alpha$ , the parameter controlling the weight of the stylization at the previous scale in the initialization of the next scale. We show results with  $\alpha = 0.0$  (minimum stylization) and  $\alpha = 1.0$  (maximum stylization).

tion and stylization quality (row 1 of Figure 8). However, we can also take advantage of our multi-scale procedure to control the stylization level of our final output (row 2 of Figure 8).

For both NNST-D and NNST-Opt we produce stylizations at eighth, quarter, half, and full resolution. The upsampled output of the previous scale serves as initialization for the next. We initialize the coarsest scale with a downsampled version of the content image. Let  $O_s$  be the output of our algorithm at scale  $s$ . Let  $C_{s+1}, S_{s+1}$  be the content and style images at finer scale  $s + 1$ . Let  $O_s^\uparrow$  be  $O_s$  upsampled to be the same resolution as  $C_{s+1}$ . Instead of constructing  $T$  by finding matches between  $\Phi(C_{s+1})$  and  $\Phi(S_{s+1})$ , we instead find matches between  $\Phi(\alpha O_s^\uparrow + (1 - \alpha)C_{s+1})$  and  $\Phi(S_{s+1})$ . The parameter  $\alpha$  controls stylization level, with  $\alpha = 0$  corresponding to the lowest stylization level, and  $\alpha = 1$  the highest. By default, we set  $\alpha = 0.25$ , as this generally produces a visually pleasing balance between stylization and content preservation.

## 4. Evaluation

### 4.1. Traditional Media Evaluation Set

In order to benchmark the performance of our method and prior work we gather a dataset of 30 high-resolution content photographs from Flickr, chosen for their diversity and under the constraint that they be available under a creative commons license allowing modification and redistribution. We followed the same procedure (also using Flickr)

	Feed-Forward Baselines			
	WCT	Avatar	MST	ArtFlow
NNST-D	71% (1.0e-10)	74% (1.0e-13)	70% (2.5e-10)	60% (1.6e-3)
NNST-Opt	83% (< 1e-15)	85% (< 1e-15)	72% (5.9e-12)	69% (1.4e-9)
Optimization-Based Baselines				
	DIA	CNNMRF	Gatys	STROTSS
NNST-D	61% (4.1e-4)	64% (1.6e-7)	60% (1.0e-3)	49% (0.66)
NNST-Opt	61% (2.5e-4)	82% (< 1e-15)	65% (1.3e-6)	55% (1.2e-2)

Table 1. The percentage of votes received by NNST in our forced choice user study when benchmarked against prior work. In parentheses is the p-value of rejecting the null hypothesis that the preference rate for NNST is less than 50%. Our feed forward variant NNST-D is preferred over all baselines except STROTSS [24], a much slower optimization based method. Our own optimization based variant, NNST-Opt, is preferred over all baselines

to gather ten ink drawings and ten watercolor paintings. From the Riksmuseum’s open-source collection we take ten impressionist oil paintings created between 1800-1900. We supplement these with ten pencil drawings taken from the dataset used in Im2Pencil [28]. In total this gives us 40 high-resolution style images. We use this dataset in the following user study, and will make it available to download.

#### 4.2. User Study

In order to assess the stylization quality of NNST-D and NNST-Opt relative prior work we generate stylizations for all pairwise content/style combinations in the traditional media evaluation set described above (A total of 1200 outputs per method). We conduct a user study using Prolific (<https://www.prolific.co/>) where users are shown the output of two algorithms (randomly ordered) for the same content/style pair (randomly selected from the 1200 possible combinations), along with the target style, and asked ‘Does “Image A” or “Image B” better match the “Target Style”?’. Users are asked to judge 9 such triplets in sequence, among which is mixed one attention verification question (selecting the image that shows a cartoon whale in a randomly ordered triplet). In total we collected 225 votes per method pair, from a total of 400 unique participants. We include 40 comparisons from the user study, along with the content/style inputs, and examples of the study interface in the supplement.

Optimization-based methods are in general considered of higher quality than fast feed-forward ones, therefore each family of techniques are generally compared separately. However, while we group these methods in Table 1, we compare both variants of our method to both families of technique. For optimization-based methods we benchmark against Gatys [11], CNNMRF [26], Deep Image Analogies (DIA) [30], and STROTSS [24]. We were unable to run the official code for CNNMRF and Deep Image Analogies, and re-implemented their methods. For fast feed-forward methods we benchmark against WCT [29], AvatarNet [43], MST [51], and ArtFlow [1].

The results of our study are summarized in Table 1, along with the p-values of rejecting the null hypothesis that the preference rate for NNST-D/Opt is less than 50%. We calculate these p-values under the assumption that the votes are independent and the sum of votes received by a method is distributed as a binomial. In summary there is a statistically significant preference for our fast variant NNST-D over all benchmarked methods (fast and optimization-based) except STROTSS, a state-of-the-art optimization-based method, for which NNST-D is on par with (STROTSS is preferred but not by a statistically significant margin). There is a statistically significant preference for NNST-Opt over *all* benchmarked methods.

#### 4.3. Discussion

Our approach performs well in general but there is nonetheless areas where there remains room for improvement. For instance, physical phenomena like the drips of paint on the bear in Figure 11 are not reproduced. Also lines and hatching patterns like in Figure 3 (second and last rows) can be distorted. These cases are challenging for all methods, and while our approach often better reproduce these phenomena and patterns than previous work, we believe that further exploring this direction would be a worthwhile effort in the future.

We also observe that even though the decoder variant produces good results, they are not as good as the optimization-based reconstructions. Reducing that gap will be key to creating a high-quality practical stylization algorithm.

### 5. Extensions and Applications

Although the primary goal of our approach is to perform a generic style transfer without requiring additional knowledge about the style and content image, in the case when such information is available our technique can be easily extended to incorporate it. In this scenario we follow the concept of Image Analogies [16] and extend our objective (1)

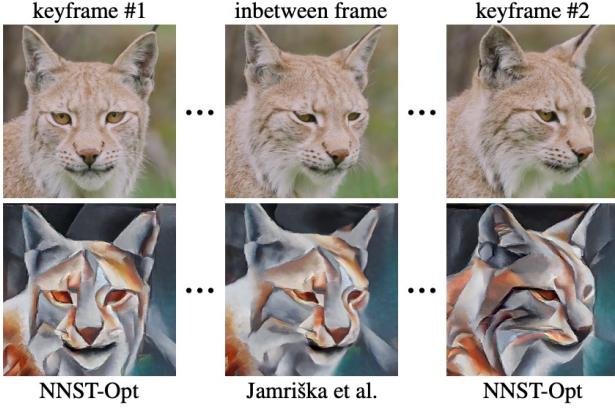


Figure 9. Our approach used as a generator of stylized frames for example-based video stylization—a few selected keyframes are stylized using NNST-Opt and the rest of the sequence is stylized using the method of Jamriška et al. [19]. See video [here](#).

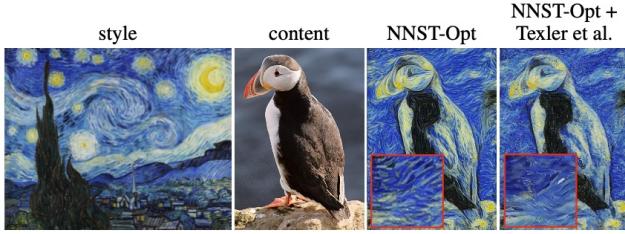


Figure 10. Our approach combined with the method of Texler et al. [46]—the output of NNST-Opt is used as a guide for patch-based synthesis algorithm that can operate at notably higher resolution, in this case a 4K resolution output, and thus faithfully preserve high-frequency details (see the zoom-in insets in red squares). At the mid-scale level, however, our technique still outperforms patch-based synthesis as it can convey style features better to delineate salient structures in the content image.

by adding a term that incorporates further guidance on top of the cosine distance:

$$T_i = \arg \min_{\Phi'(S)_j} w_{\cos} D(\Phi(C)_i - \mu_C, \Phi'(S)_j - \mu'_S) + w_{guide} D^g(C_i^g, S_j^g)$$

Here  $S_i^g$  and  $C_i^g$  are downsampled versions of style and content guiding channels (e.g., segmentation masks, see Figure 11),  $D^g$  is a metric which evaluates guide similarity at pixels  $i$  and  $j$  (in our experiments we use sum of squared differences), and  $w_{\cos}$  and  $w_{guide}$  are weights that balance the influence of the cosine and guiding term (in our experiments we set  $w_{\cos} = 0.5$  and  $w_{guide} = 0.5$ ). In Figure 11 we demonstrate the effect of incorporating additional segmentation masks as a guiding channels. It is visible that with guidance the content from the style is transferred in a more semantically meaningful way. In contrast to previous neural approaches that also support guidance [12, 24]

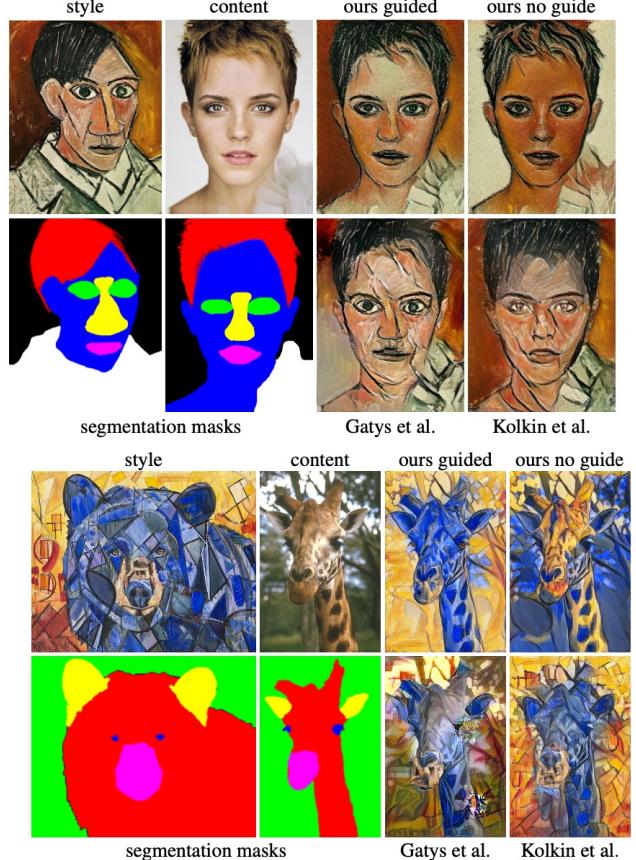


Figure 11. Incorporating additional guidance (segmentation masks) into our technique—in contrast to unguided version the output is semantically meaningful, i.e., background texture in the stylized image corresponds to the background in the style exemplar, etc. When compared to the current state-of-the-art in neural style transfer that also support guidance [12, 24] our approach better preserves style details.

our technique better preserves visual aspects of the original style exemplar.

Besides single image style transfer our approach is practical also in the context of example-based video stylization [19, 47] where the aim is to propagate the style from a sparse set of stylized keyframes to the rest of the video sequence. In the original setting, the stylization of keyframes is tedious as those need to be created by hand to stay perfectly aligned with the content in the video. Using our approach, however, one can stylize the entire sequence fully automatically without the need to preserve alignment. By transferring the style from an arbitrary exemplar image one can stylize a subset of frames and then run an existing keyframe-based video stylization technique of Jamriška et al. [19] or Texler et al. [47] to propagate the style to the rest of the sequence while maintaining temporal coherence (see Figure 9 and our supplementary video).

One of the limiting factors of our technique is that using currently available GPUs it can deliver only outputs in moderate resolutions such as 1k. To obtain higher resolution images our technique can be plugged into the method of Texler et al. [46]. In this approach the result of neural style transfer is used as a guide to drive patch-based synthesis algorithm of Fišer et al. [9] that can produce a high-resolution counterpart of the stylized image generated by the neural method (in our case the generated nearest neighbor field is upsampled to obtain a 4K output). Such a combination can help to ensure a more faithful style transfer thanks to the ability to reproduce high-frequency details of the original style exemplar. However, a compromise here is that when comparing middle scale features our technique can perform better than patch-based synthesis since it can adopt the style features to follow salient structures visible in the content image (c.f. Figure 10).

## 6. Conclusion

We have demonstrated a conceptually simple approach to artistic stylization of images. We explored several key design choices to motivate our algorithm. We showed qualitatively and quantitatively that our approach is flexible enough to support various scenarios and produce high-quality results in all these cases. Put together, we believe that these characteristics make our approach suitable for practical applications and a solid basis for future work.

## References

- [1] An, J., Huang, S., Song, Y., Dou, D., Liu, W., Luo, J.: ArtFlow: Unbiased image style transfer via reversible neural flows. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 862–871 (2021) [3](#), [4](#), [6](#), [7](#), [10](#)
- [2] Bénard, P., Cole, F., Kass, M., Mordatch, I., Hegarty, J., Senn, M.S., Fleischer, K., Pesare, D., Breeden, K.: Stylizing animation by example. ACM Transactions on Graphics **32**(4), 119 (2013) [2](#)
- [3] Berger, G., Memisevic, R.: Incorporating long-range consistency in CNN-based texture generation. In: Proceedings of International Conference on Learning Representations (2017) [3](#)
- [4] Chen, T.Q., Schmidt, M.: Fast patch-based style transfer of arbitrary style. In: Proceedings of Conference on Neural Information Processing Systems (2016) [2](#), [3](#), [15](#), [16](#), [17](#), [18](#), [20](#)
- [5] Chiu, T.Y.: Understanding generalized whitening and coloring transform for universal style transfer. In: Proceedings of IEEE International Conference on Computer Vision. pp. 4452–4460 (2019) [3](#), [6](#)
- [6] Chiu, T.Y., Gurari, D.: Iterative feature transformation for fast and versatile universal style transfer. In: Proceedings of European Conference on Computer Vision. pp. 169–184 (2020) [3](#), [6](#)
- [7] Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: SIGGRAPH Conference Proceedings. pp. 341–346 (2001) [2](#)
- [8] Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. vol. 2, pp. 1033–1038 (1999) [2](#)
- [9] Fišer, J., Jamriška, O., Lukáč, M., Shechtman, E., Asente, P., Lu, J., Sýkora, D.: StyLit: Illumination-guided example-based stylization of 3D renderings. ACM Transactions on Graphics **35**(4), 92 (2016) [2](#), [7](#), [12](#)
- [10] Fišer, J., Jamriška, O., Simons, D., Shechtman, E., Lu, J., Asente, P., Lukáč, M., Sýkora, D.: Example-based synthesis of stylized facial animations. ACM Transactions on Graphics **36**(4), 155 (2017) [2](#)
- [11] Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2414–2423 (2016) [2](#), [3](#), [5](#), [10](#)
- [12] Gatys, L.A., Ecker, A.S., Bethge, M., Hertzmann, A., Shechtman, E.: Controlling perceptual factors in neural style transfer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3730–3738 (2017) [9](#), [11](#)
- [13] Gu, S., Chen, C., Liao, J., Yuan, L.: Arbitrary style transfer with deep feature reshuffle. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8222–8231 (2018) [2](#), [3](#), [7](#)
- [14] Hariharan, B., Arbeláez, P., Girshick, R., Malik, J.: Hypercolumns for object segmentation and fine-grained localization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 447–456 (2015) [6](#)
- [15] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016) [7](#)
- [16] Hertzmann, A., Jacobs, C.E., Oliver, N., Curless, B., Salesin, D.H.: Image analogies. In: SIGGRAPH Conference Proceedings. pp. 327–340 (2001) [2](#), [10](#)
- [17] Huang, X., Belongie, S.J.: Arbitrary style transfer in real-time with adaptive instance normalization. Proceedings of IEEE International Conference on Computer Vision pp. 1510–1519 (2017) [3](#), [6](#)
- [18] Jamriška, O., Fišer, J., Asente, P., Lu, J., Shechtman, E., Sýkora, D.: LazyFluids: Appearance transfer for fluid animations. ACM Transactions on Graphics **34**(4), 92 (2015) [7](#)
- [19] Jamriška, O., Šárka Sochorová, Texler, O., Lukáč, M., Fišer, J., Lu, J., Shechtman, E., Sýkora, D.: Stylizing video by example. ACM Transactions on Graphics **38**(4), 107 (2019) [2](#), [11](#)
- [20] Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: Proceedings of European Conference on Computer Vision. pp. 694–711. Springer (2016) [6](#)
- [21] Junginger, A., Hanselmann, M., Strauss, T., Boblest, S., Buchner, J., Ulmer, H.: Unpaired high-resolution and scalable style transfer using generative adversarial networks. arXiv preprint 1810.05724 (2018) [6](#)
- [22] Kaspar, A., Neubert, B., Lischinski, D., Pauly, M., Kopf, J.: Self tuning texture optimization. Computer Graphics Forum **34**(2), 349–359 (2015) [7](#)
- [23] Kingma, D.P., Ba, J.A.: Adam: A method for stochastic optimization. In: Proceedings of International Conference on Learning Representations (2015) [8](#)

- [24] Kolkin, N., Salavon, J., Shakhnarovich, G.: Style transfer by relaxed optimal transport and self-similarity. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10051–10060 (2019) [2](#), [3](#), [5](#), [10](#), [11](#)
- [25] Kotovenko, D., Sanakoyeu, A., Lang, S., Ommer, B.: Content and style disentanglement for artistic style transfer. In: Proceedings of IEEE International Conference on Computer Vision. pp. 4422–4431 (2019) [6](#)
- [26] Li, C., Wand, M.: Combining markov random fields and convolutional neural networks for image synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2479–2486 (2016) [2](#), [3](#), [10](#)
- [27] Li, Y., Wang, N., Liu, J., Hou, X.: Demystifying neural style transfer. Proceedings of International Joint Conference on Artificial Intelligence pp. 2230–2236 (2017) [3](#)
- [28] Li, Y., Fang, C., Hertzmann, A., Shechtman, E., Yang, M.H.: Im2pencil: Controllable pencil illustration from photographs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1525–1534 (2019) [10](#)
- [29] Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Universal style transfer via feature transforms. In: Proceedings of Conference on Neural Information Processing Systems. pp. 385–395 (2017) [3](#), [4](#), [6](#), [10](#)
- [30] Liao, J., Yao, Y., Yuan, L., Hua, G., Kang, S.B.: Visual attribute transfer through deep image analogy. ACM Transactions on Graphics **36**(4), 120 (2017) [2](#), [3](#), [5](#), [10](#)
- [31] Luan, F., Paris, S., Shechtman, E., Bala, K.: Deep photo style transfer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4990–4998 (2017) [7](#)
- [32] Lukáč, M., Fišer, J., Asente, P., Lu, J., Shechtman, E., Sýkora, D.: Brushables: Example-based edge-aware directional texture painting. Computer Graphics Forum **34**(7), 257–268 (2015) [2](#)
- [33] Lukáč, M., Fišer, J., Bazin, J.C., Jamriška, O., Sorkine-Hornung, A., Sýkora, D.: Painting by feature: Texture boundaries for example-based image creation. ACM Transactions on Graphics **32**(4), 116 (2013) [2](#)
- [34] Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: ICML (2013) [7](#)
- [35] Mechrez, R., Talmi, I., Zelnik-Manor, L.: The contextual loss for image transformation with non-aligned data. In: Proceedings of European Conference on Computer Vision. pp. 768–783 (2018) [3](#)
- [36] Mostajabi, M., Yadollahpour, P., Shakhnarovich, G.: Feedforward semantic segmentation with zoom-out features. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3376–3385 (2015) [6](#)
- [37] Paris, S., Kornprobst, P., Tumblin, J., Durand, F.: Bilateral filtering: Theory and applications. Now Publishers Inc (2009) [9](#)
- [38] Park, D.Y., Lee, K.H.: Arbitrary style transfer with style-attentional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5880–5888 (2019) [3](#), [6](#), [7](#)
- [39] Park, T., Zhu, J.Y., Wang, O., Lu, J., Shechtman, E., Efros, A.A., Zhang, R.: Swapping autoencoder for deep image manipulation. In: Proceedings of Conference on Neural Information Processing Systems (2020) [8](#)
- [40] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Proceedings of Conference on Neural Information Processing Systems, pp. 8024–8035 (2019) [6](#)
- [41] Risser, E., Wilmot, P., Barnes, C.: Stable and controllable neural texture synthesis and style transfer using histogram losses. arXiv preprint 1701.08893 (2017) [3](#)
- [42] Sanakoyeu, A., Kotovenko, D., Lang, S., Ommer, B.: A style-aware content loss for real-time hd style transfer. In: Proceedings of European Conference on Computer Vision. pp. 698–714 (2018) [6](#)
- [43] Sheng, L., Lin, Z., Shao, J., Wang, X.: Avatar-net: Multi-scale zero-shot style transfer by feature decoration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8242–8250 (2018) [3](#), [6](#), [10](#)
- [44] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Proceedings of International Conference on Learning Representations (2014) [2](#), [6](#)

- [45] Svoboda, J., Anoosheh, A., Osendorfer, C., Masci, J.: Two-stage peer-regularized feature recombination for arbitrary image style transfer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 13816–13825 (2020) [6](#)
- [46] Texler, O., Futschik, D., Fišer, J., Lukáč, M., Lu, J., Shechtman, E., Sýkora, D.: Arbitrary style transfer using neurally-guided patch-based synthesis. Computers & Graphics **87**, 62–71 (2020) [3, 11, 12](#)
- [47] Texler, O., Futschik, D., Kučera, M., Jamriška, O., Šárka Sochorová, Chai, M., Tulyakov, S., Sýkora, D.: Interactive video stylization using few-shot patch-based training. ACM Transactions on Graphics **39**(4), 73 (2020) [2, 11](#)
- [48] Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. Proceedings of IEEE International Conference on Computer Vision pp. 839–846 (1998) [9](#)
- [49] Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: SIGGRAPH Conference Proceedings. pp. 479–488 (2000) [2](#)
- [50] Yao, Y., Ren, J., Xie, X., Liu, W., Liu, Y.J., Wang, J.: Attention-aware multi-stroke style transfer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1467–1475 (2019) [3, 6](#)
- [51] Zhang, Y., Fang, C., Wang, Y., Wang, Z., Lin, Z., Fu, Y., Yang, J.: Multimodal style transfer via graph cuts. In: Proceedings of IEEE International Conference on Computer Vision. pp. 5943–5951 (2019) [3, 4, 6, 7, 10](#)
- [52] Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. Proceedings of IEEE International Conference on Computer Vision pp. 2242–2251 (2017) [6](#)

## 7. Supplement - Design Decisions

A pithy description of NNST and the non-parametric neural style transfer algorithm proposed by Chen and Schmidt [\[4\]](#) much earlier in 2016 would reveal little difference between the two. Both methods explicitly construct a tensor of ‘target features’ by replacing vectors of VGG-derived content features with vectors of VGG-derived style features, then optimize the pixels of the output image to produce the ‘target features’ (or use a learned decoder). Yet, there is a dramatic difference between the visual quality of the algorithms’ outputs. As is often the case, the devil is in the details, and this section explores the important design decisions that can boost a style transfer algorithm’s visual quality.

In Figures [12, 13, 14, 15, 16, 17, 18](#), we visually explore the effects of the our design decisions relative to [\[4\]](#), and iteratively modify their method until arriving at NNST. Where appropriate these figures also demonstrate the effect of modifying individual design elements of NNST to match [\[4\]](#).



Figure 12. Visual comparison between (a.) the outputs of Chen and Schmidt [4] and (b.) a simplified variant of NNST which uses the feature splitting regime across all scales and does not employ color correction. While both algorithms share a similar high level framework, they differ in many details, resulting in NNST much better recreating distinctive visual features of the style image.



Figure 13. Visual comparison between matching features separately for each location ( $1 \times 1$  patches) and matching them as  $3 \times 3$  patches: (a.) the outputs of Chen and Schmidt [4] ( $3 \times 3$  feature patches matched and overlaps averaged), (b.) a variant of [4] where feature patches are matched independently for each spatial location ( $1 \times 1$  patches, no averaging), (c.) a variant of NNST where  $3 \times 3$  patches are matched and averaged, and (d.) the simplified NNST variant from Figure 12. Note that matching  $1 \times 1$  rather than  $3 \times 3$  patches (b. and d. relative to a. and c.) allows more high frequency details of the style to appear in the output.



Figure 14. Visual comparison between single-scale and multi-scale stylization: (a.) [4] w/ 1x1 patches (row b of Figure 13), (b.) a. applied coarse-to-fine using the same mechanisms as NNST ( $\alpha = 0.25$ ), (c) simplified NNST at only the finest scale ( $\alpha = 1.0$ ), and (d.) the simplified NNST variant from Figure 12. Stylizing coarse-to-fine increases stylization level and results in visual features of the style with larger spatial extent appearing in the output (and this effect increases with lower  $\alpha$ , see Figure 8). In addition, stylizing coarse-to-fine allows stylistic details to be hallucinated in large flat regions of the content image (compare the floor beneath the monk in c. and d.).



Figure 15. Visual comparison between using zero-centering or not before matching features using the cosine distance: (a.) multi-scale [4] w/ 1x1 patches (row b of Figure 14), (b.) a. using the centered cosine distance for feature matching (instead of the standard cosine distance), (c) d. using the standard cosine distance (instead of the centered cosine distance), and (d.) the simplified NNST variant from Figure 12. Using the centered cosine distance not only results in a more diverse set of style features appearing in the output, the contrast between these features helps preserve the perceived contents of the original input.

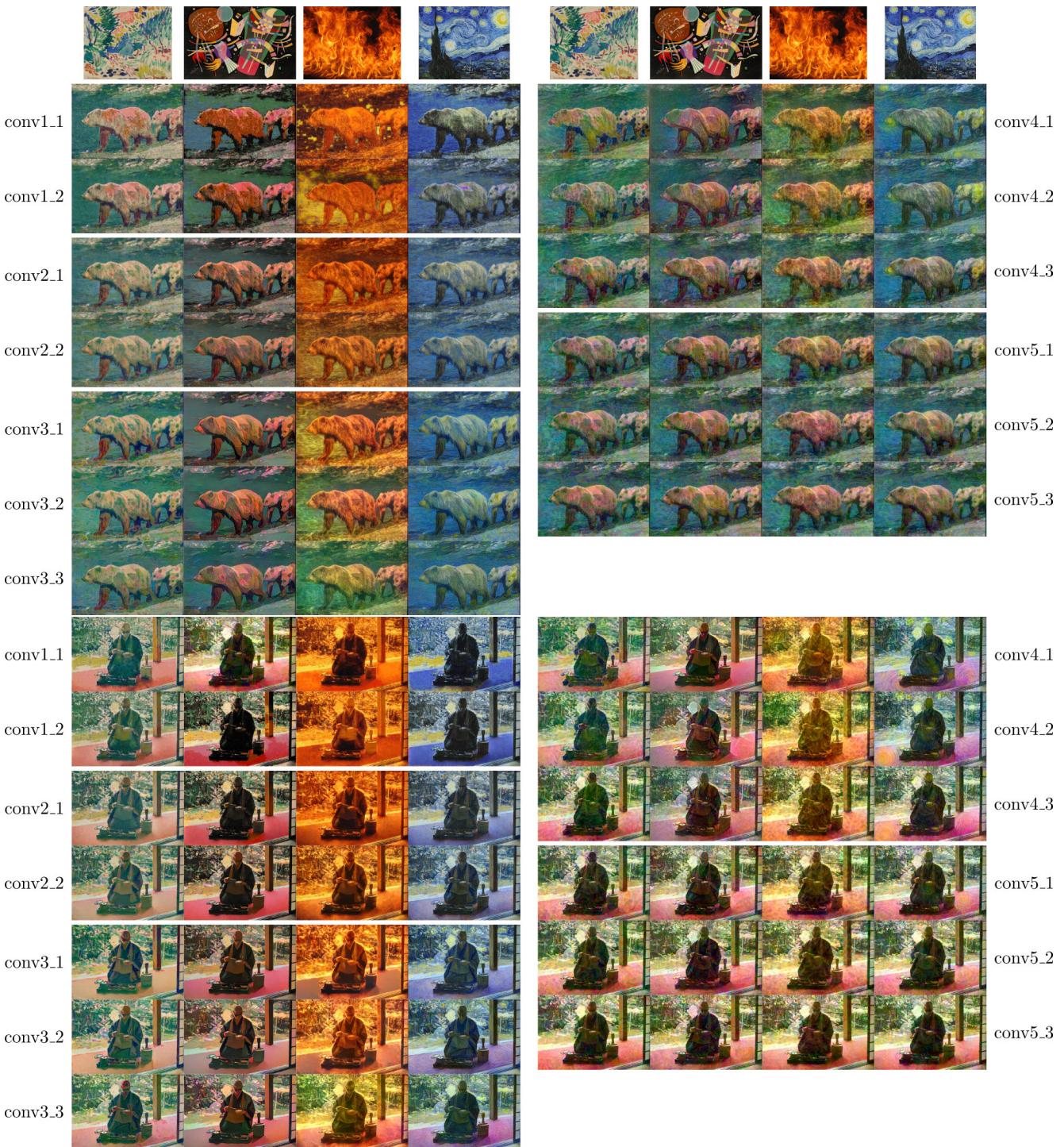


Figure 16. Visual comparison between using different individual convolutional layers of pretrained VGG-16 as a source of features. All images are produced using multi-scale [4] w/  $1 \times 1$  patches matched with the centered cosine distance (row b. of Figure 15 corresponds to row conv3.1 of this figure, the default style features used by [4]). Layers in the first two conv. blocks capture color well, but not more complex visual elements. Layers in the third and fourth conv block capture complex visual elements, but not color. Layers in the fifth block do not seem closely tied to stylistic features. No layer alone is sufficient to capture all desired stylistic features.

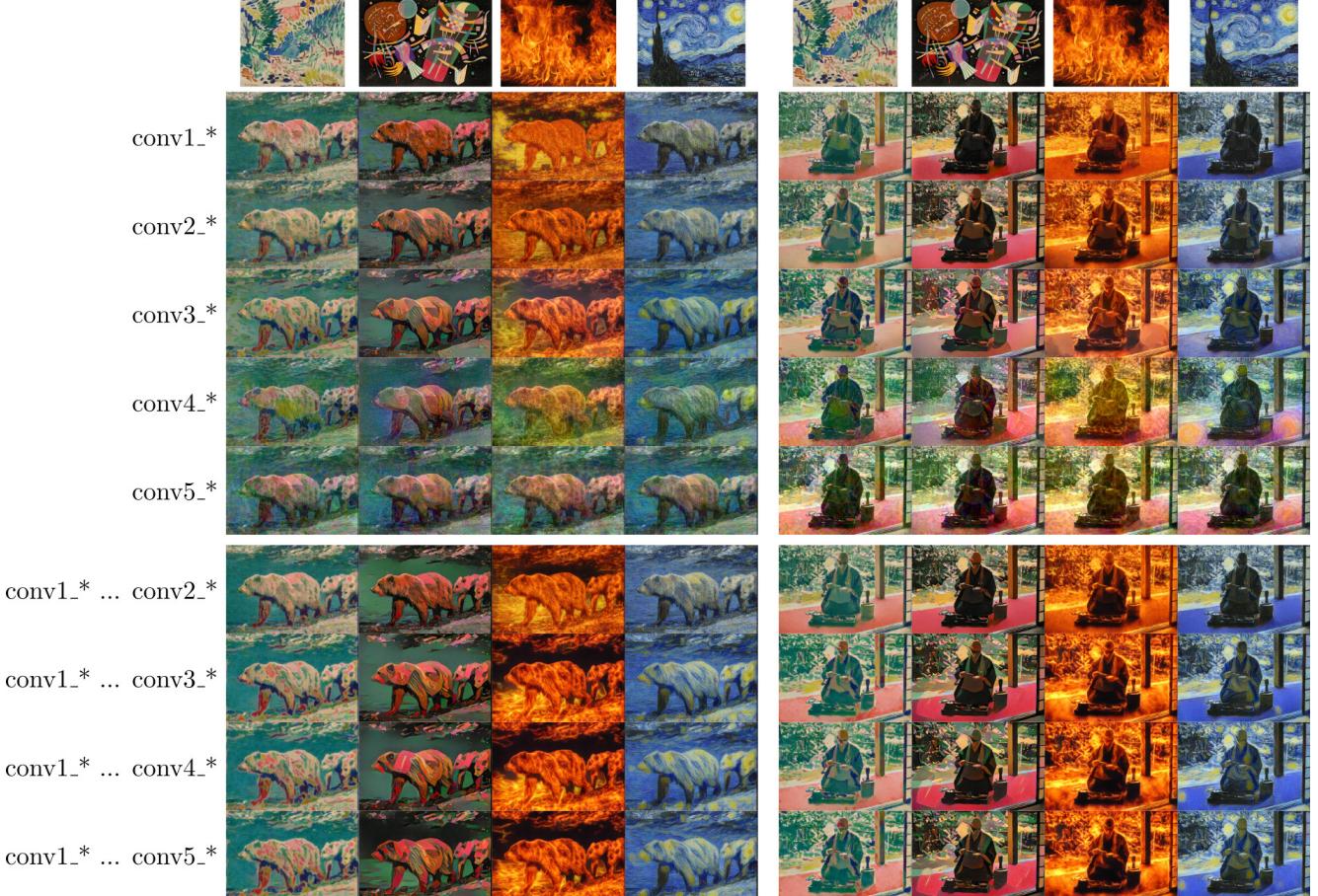


Figure 17. Visual comparison between using features from multiple layers of pretrained VGG-16. The first 5 rows demonstrate the effect of using all the layers from a given conv. block. Just as no single layer is sufficient, no single conv. block contains a rich enough representation of style to produce satisfactory outputs. The 5th-9th rows demonstrate the effect of using all of the features up to a certain depth in the network. Most important stylistic details can be captured using the first three conv. blocks. Small improvements can be made using the 4th and 5th conv. blocks as well, but it is probably not worth the computational cost (the 4th and 5th blocks each contain 36% of the total feature channels). While NNST uses all feature through conv. block 4, only using features through block 3 would be an obvious means to increase efficiency.

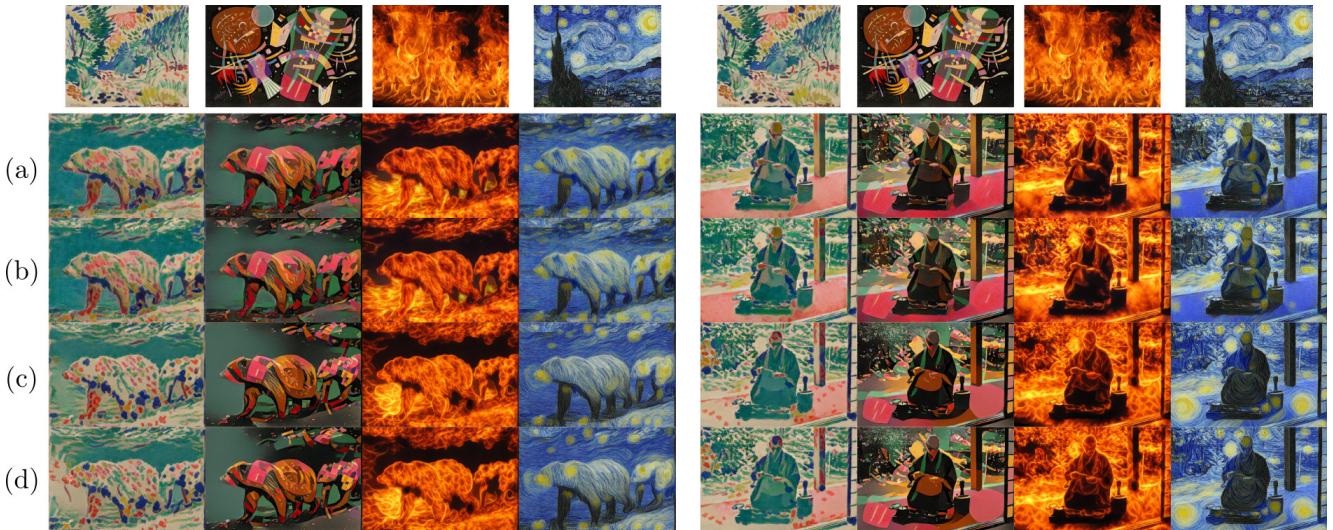


Figure 18. The design decisions so far lead to an algorithm close to NNST, the only difference that remain to be evaluated are the frequency of computing matches and whether or not to compute matches separately for each layer: (a.) multi-scale [4] w/ 1x1 hypercolumns using layers conv1\_1-conv4\_3, matched with the centered cosine distance (row 8 of Figure 17), (b.) a. with nearest neighbors recomputed after each update of the output image, (c.) a. with nearest neighbors computed separately for each layer, and (d.) the simplified NNST variant from Figure 12 (i.e. a. w/ features matched separately for each layer and recomputing matches each update). Nice results can already be achieved without feature splitting or recomputing matches each iteration (a.), however slightly sharper high frequencies can be achieved by recomputing matches (b.), and more diverse stylistic features from each layer are matched separately (c.). When both of these modifications are applied, we essentially arrive at NNST (d.).