Fast Patch-based Style Transfer of Arbitrary Style

Tian Qi Chen
Department of Computer Science
University of British Columbia
tqichen@cs.ubc.ca

Mark Schmidt
Department of Computer Science
University of British Columbia
schmidtm@cs.ubc.ca

Abstract

Artistic style transfer is an image synthesis problem where the content of an image is reproduced with the style of another. Recent works show that a visually appealing style transfer can be achieved by using the hidden activations of a pretrained convolutional neural network. However, existing methods either apply (i) an optimization procedure that works for any style image but is very expensive, or (ii) an efficient feedforward network that only allows a limited number of trained styles. In this work we propose a simpler optimization objective based on local matching that combines the content structure and style textures in a single layer of the pretrained network. We show that our objective has desirable properties such as a simpler optimization landscape, intuitive parameter tuning, and consistent frame-by-frame performance on video. Furthermore, we use 80,000 natural images and 80,000 paintings to train an inverse network that approximates the result of the optimization. This results in a procedure for artistic style transfer that is efficient but also allows arbitrary content and style images.

Content	Style

Our Style Transfer	Our Feed-Forward Approx.

Figure 1: An example of our artistic style transfer method and its feedforward approximation. The approximation network has never seen the content or style image during training.

1. Introduction

Famous artists are typically renowned for a particular artistic style, which takes years to develop. Even once perfected, a single piece of art can take days or even months to create. This motivates us to explore efficient computational strategies for creating artistic images. While there is a large classical literature on texture synthesis methods that create artwork from a blank canvas [8, 20, 22, 35], several recent approaches study the problem of transferring the desired *style* from one image onto the structural *content* of another image. This approach is known as artistic style transfer.

The vague notion of artistic style is difficult to quantitatively capture. Early works define style using similarity measures or local statistics based on the pixel values [7, 8, 13, 14, 20, 24]. However, recent methods that achieve impressive visual quality come from the use of convolutional neural networks (CNN) for feature extraction [9, 10, 11, 21]. The success of these methods has even created a market for mobile applications that can stylize user-provided images on demand [15, 29, 32].

Despite this renewed interest, the actual process of style transfer is based on solving a complex optimization procedure, which can take minutes on today's hardware. A typical speedup solution is to train another neural network that approximates the optimum of the optimization in a single feed-forward pass [6, 17, 33, 34]. While much faster, existing works that use this approach sacrifice the versatility of being able to perform style transfer with any given style image, as the feed-forward network cannot generalize beyond its trained set of images. Due to this limitation, existing applications are either time-consuming or limited in the number of provided styles, depending on the method of style transfer.

In this work we propose a method that addresses these

1

limitations: a new method for artistic style transfer that is efficient but is not limited to a finite set of styles. To accomplish this, we define a new optimization objective for style transfer that notably only depends on one layer of the CNN (as opposed to existing methods that use multiple layers). The new objective leads to visually-appealing results while this simple restriction allows us to use an "inverse network" to deterministically invert the activations from the stylized layer to yield the stylized image.

Section 2 reviews related work while Sections 3-4 describe our new optimization objective that combines style and content statistics in a single activation layer. We then propose a method of training a neural network that can invert the activations to yield an image. Section 5 presents experiments with the new method, showing it has desirable properties not found in existing formulations.

2. Related Work

Style Transfer as Optimization. Gatys *et al*. [11] formulates style transfer as an optimization problem that combines texture synthesis with content reconstruction. Their formulation involves additive loss functions placed on multiple layers of a pretrained CNN, with some loss functions synthesizing the textures of the style image and some loss functions reconstructing the content image. Gradients are computed by backpropagation and gradient-based optimization is carried out to solve for the stylized image. An alternative approach uses patch-based similarity matching between the content and style images [9, 10, 21]. In particular, Li and Wand [21] constructs patch-based loss functions, where each synthetic patch has a nearest neighbour target patch that it must match. This type of patch-matching loss function is then combined additively with Gatys *et al*.'s formulation. While these approaches allow arbitrary style images, the optimization frameworks used by these methods makes it slow to generate the stylized image. This is particularly relevant for the case of video where we want to style a huge number of frames.

Feed-forward Style Networks As mentioned previously, it is possible to train a neural network that approximates the optimum of Gatys *et al*.'s loss function for one or more fixed styles [6, 17, 33, 34]. This yields a much faster method, but these methods need to be re-trained for each new style.

Style Transfer for Video. Ruder *et al*. [30] introduces a temporal loss function that, when used additively with Gatys *et al*.'s loss functions, can perform style transfer for videos with temporal consistency. Their loss function relies on optical flow algorithms for gluing the style in place across nearby frames. This results in an order of maginitude slowdown compared to frame-by-frame application of style transfer.

Inverting Deep Representations. Several works have trained inverse networks of pretrained convolutional neural networks [3, 26] for the goal of visualization. Other works have trained inverse networks as part of an autoencoder [19, 27, 36]. To the best of our knowledge, all existing inverse networks are trained with a dataset of images and a loss function placed in RGB space.

In comparison to existing style transfer approaches, we propose a method for directly constructing the target activations for a single layer in a pretrained CNN. Like Li and Wand [21], we make use of a criteria for finding best matching patches in the activation space. However, our method is able to directly construct the entire activation target. This deterministic procedure allows us to easily adapt to video, without the need to rely on optical flow for fixing consistency issues. In addition to optimization, we propose a feed-forward style transfer procedure by inverting the pretrained CNN. Unlike existing feed-forward style transfer approaches, our method is not limited to specifically trained styles and can easily adapt to arbitary content and style images. Unlike existing CNN inversion methods, our method of training does not use a pixel-level loss, but instead uses a loss on the activations. By using a particular training setup (see Section 4.1), this inverse network is even able to invert activations that are outside the usual range of the CNN activiations.

3. A New Objective for Style Transfer

The main component of our style transfer method is a patch-based operation for constructing the target activations in a single layer, given the style and content images. We refer to this procedure as "swapping the style" of an image, as the content image is replaced patch-by-patch by the style image. We first present this operation at a high level, followed by more details on our implementation.

3.1. Style Swap

Let C and S denote the RGB representations of the content and style images (respectively), and let $\Phi(\cdot)$ be the function represented by a fully convolutional part of a pretrained CNN that maps an image from RGB to some intermediate activation space. After computing the activations, $\Phi(C)$ and $\Phi(S)$, the *style swap* procedure is as follows:

1. Extract a set of patches for both content and style activations, denoted by $\{\phi_i(C)\}_{i \in n_c}$ and $\{\phi_j(S)\}_{j \in n_s}$, where n_c and n_s are the number of extracted patches. The extracted patches should have sufficient overlap, and contain all channels of the activations.

2. For each content activation patch, determine a closest-matching style patch based on the normalized cross-

Content Activations — **Channel-wise Argmax** — **Target Activations**

2D Convolution With Normalized Style Patches as Filters → **2D Transposed Convolution With Style Patches as Filters**
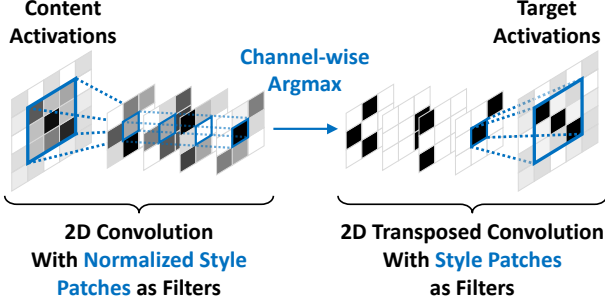
Figure 2: Illustration of a style swap operation. The 2D convolution extracts patches of size 3×3 and stride 1, and computes the normalized cross-correlations. There are $n_c = 9$ spatial locations and $n_s = 4$ feature channels immediately before and after the channel-wise argmax operation. The 2D transposed convolution reconstructs the complete activations by placing each best matching style patch at the corresponding spatial location.

correlation measure,

$$\phi_i^{ss}(C,S) := \underset{\phi_j(S),\, j=1,\ldots,n_s}{\operatorname{argmax}} \frac{\langle \phi_i(C), \phi_j(S) \rangle}{||\phi_i(C)|| \cdot ||\phi_j(S)||}. \tag{1}$$

3. Swap each content activation patch $\phi_i(C)$ with its closest-matching style patch $\phi_i^{ss}(C,S)$.

4. Reconstruct the complete content activations, which we denote by $\Phi^{ss}(C,S)$, by averaging overlapping areas that may have different values due to step 3.

This operation results in hidden activations corresponding to a single image with the structure of the content image, but with textures taken from the style image.

3.2. Parallelizable Implementation

To give an efficient implementation, we show that the entire style swap operation can be implemented as a network with three operations: (i) a 2D convolutional layer, (ii) a channel-wise argmax, and (iii) a 2D transposed convolutional layer. Implementation of style swap is then as simple as using existing efficient implementations of 2D convolutions and transposed convolutions[1].

To concisely describe the implementation, we re-index the content activation patches to explicitly denote spatial structure. In particular, we'll let d be the number of feature channels of $\Phi(C)$, and let $\phi_{a,b}(C)$ denote the patch $\Phi(C)_{a:a+s,\, b:b+s,\, 1:d}$ where s is the patch size.

---

[1]The transposed convolution is also often referred to as a "fractionally-strided" convolution, a "backward" convolution, an "upconvolution", or a "deconvolution".

Notice that the normalization term for content activation patches $\phi_i(C)$ is constant with respect to the argmax operation, so (1) can be rewriten as

$$K_{a,b,j} = \left\langle \phi_{a,b}(C), \frac{\phi_j(S)}{||\phi_j(S)||} \right\rangle$$
$$\phi_{a,b}^{ss}(C,S) = \underset{\phi_j(S),\, j \in N_s}{\operatorname{argmax}} \{K_{a,b,j}\} \tag{2}$$

The lack of a normalization for the content activation patches simplifies computation and allows our use of 2D convolutional layers. The following three steps describe our implementation and are illustrated in Figure 2:

- The tensor K can be computed by a single 2D convolution by using the normalized style activations patches $\{\phi_j(S)/||\phi_j(S)||\}$ as convolution filters and $\Phi(C)$ as input. The computed K has n_c spatial locations and n_s feature channels. At each spatial location, $K_{a,b}$ is a vector of cross-correlations between a content activation patch and all style activation patches.

- To prepare for the 2D transposed convolution, we replace each vector $K_{a,b}$ by a one-hot vector corresponding to the best matching style activation patch.

$$\overline{K}_{a,b,j} = \begin{cases} 1 & \text{if } j = \operatorname{argmax}_{j'}\{K_{a,b,j'}\} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

- The last operation for constructing $\Phi^{ss}(C,S)$ is a 2D transposed convolution with $\overline{K}$ as input and unnormalized style activation patches $\{\phi_j(S)\}$ as filters. At each spatial location, only the best matching style activation patch is in the output, as the other patches are multiplied by zero.

Note that a transposed convolution will sum up the values from overlapping patches. In order to average these values, we perform an element-wise division on each spatial location of the output by the number of overlapping patches. Consequently, we do not need to impose that the argmax in (3) has a unique solution, as multiple argmax solutions can simply be interpreted as adding more overlapping patches.

3.3. Optimization Formulation

The pixel representation of the stylized image can be computed by placing a loss function on the activation space with target activations $\Phi^{ss}(C,S)$. Similar to prior works on style transfer [11, 21], we use the squared-error loss and define our optimization objective as

$$I_{stylized}(C,S) = \underset{I \in \mathbb{R}^{h \times w \times d}}{\operatorname{argmin}} ||\Phi(I) - \Phi^{ss}(C,S)||_F^2 \tag{4}$$
$$+ \lambda \ell_{TV}(I)$$

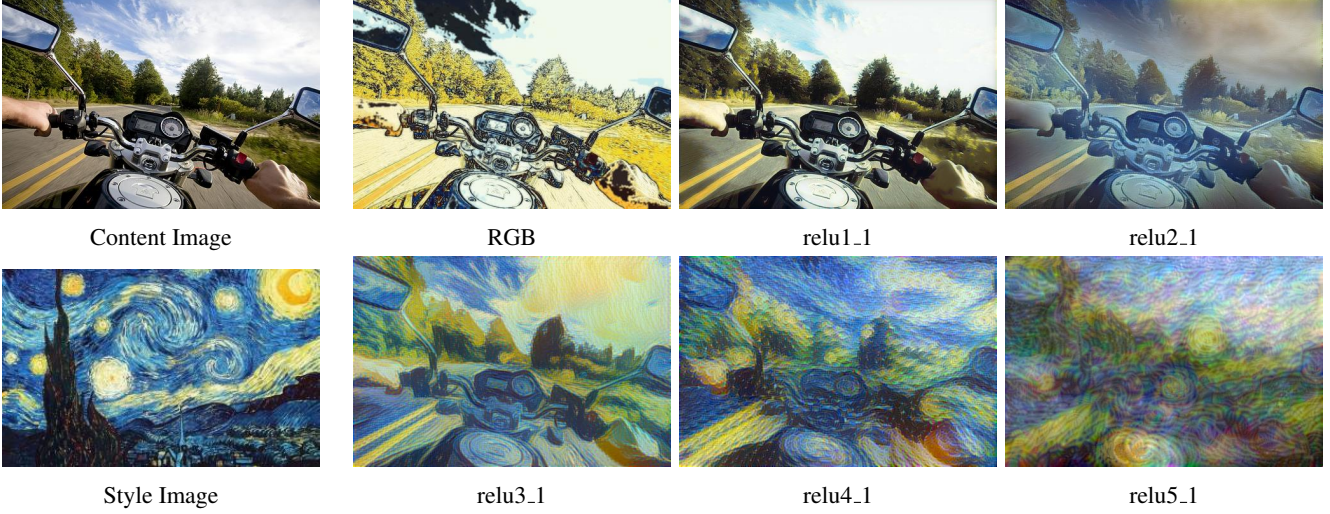| Content Image | RGB | relu1_1 | relu2_1 |

| Style Image | relu3_1 | relu4_1 | relu5_1 |

Figure 3: The effect of style swapping in different layers of VGG-19 [31], and also in RGB space. Due to the naming convention of VGG-19, "reluX_1" refers to the first ReLU layer after the $(X - 1)$-th maxpooling layer. The style swap operation uses patches of size 3×3 and stride 1, and then the RGB image is constructed using optimization.

where we'll say that the synthesized image is of dimension h by w by d, $|| \cdot ||_F$ is the Frobenius norm, and $\ell_{TV}(\cdot)$ is the total variation regularization term widely used in image generation methods [1, 17, 26]. Because $\Phi(\cdot)$ contains multiple maxpooling operations that downsample the image, we use this regularization as a natural image prior, obtaining spatially smoother results for the re-upsampled image. The total variation regularization is as follows:

$$
\ell_{TV}(I) = \sum_{i=1}^{h-1} \sum_{j=1}^{w} \sum_{k=1}^{d} (I_{i+1,j,k} - I_{i,j,k})^2 \\
+ \sum_{i=1}^{h} \sum_{j=1}^{w-1} \sum_{k=1}^{d} (I_{i,j+1,k} - I_{i,j,k})^2
\tag{5}
$$

Since the function $\Phi(\cdot)$ is part of a pretrained CNN and is at least once subdifferentiable, (4) can be computed using standard subgradient-based optimization methods.

4. Inverse Network

Unfortunately, the cost of solving the optimization problem to compute the stylized image might be too high in applications such as video stylization. We can improve optimization speed by approximating the optimum using another neural network. Once trained, this network can then be used to produce stylized images much faster, and we will in particular train this network to have the versatility of being able to use new content and new style images.

The main purpose of our inverse network is to approximate an optimum of the loss function in (4) for any target activations. We therefore define the optimal inverse func-

tion as:

$$
\underset{f}{\mathrm{arginf}} \; \mathbb{E}_H \left[||\Phi(f(H)) - H||_F^2 + \lambda \ell_{TV}\left(f(H)\right) \right]
\tag{6}
$$

where f represents a deterministic function and H is a random variable representing target activations. The total variation regularization term is added as a natural image prior similar to (4).

4.1. Training the Inverse Network

A couple problems arise due to the properties of the pretrained convolutional neural network.

Non-injective. The CNN defining $\Phi(\cdot)$ contains convolutional, maxpooling, and ReLU layers. These functions are many-to-one, and thus do not have well-defined inverse functions. Akin to existing works that use inverse networks [4, 25, 36], we instead train an approximation to the inverse relation by a parametric neural network.

$$
\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} ||\Phi(f(H_i; \theta)) - H_i||_F^2 + \lambda \ell_{TV}\left(f(H_i; \theta)\right)
\tag{7}
$$

where θ denotes the parameters of the neural network f and H_i are activation features from a dataset of size n. This objective function leads to unsupervised training of the neural network as the optimum of (4) does not need to be known. We place the description of our inverse network architecture in the appendix.

Non-surjective. The style swap operation produces target activations that may be outside the range of $\Phi(\cdot)$ due to
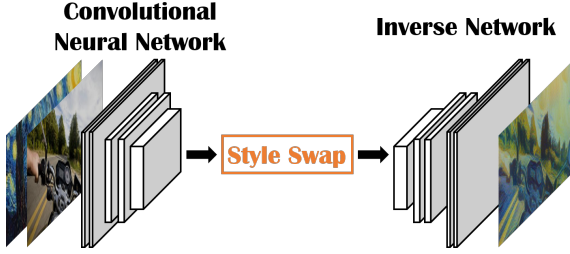
4

Figure 4: We propose the first feedforward method for style transfer that can be used for arbitrary style images. We formulate style transfer using a constructive procedure (Style Swap) and train an inverse network to generate the image.

the interpolation. This would mean that if the inverse network is only trained with real images then the inverse network may only be able to invert activations in the range of $\Phi(\cdot)$. Since we would like the inverse network to invert style swapped activations, we augment the training set to include these activations. More precisely, given a set of training images (and their corresponding activations), we augment this training set with style-swapped activations based on pairs of images.

4.2. Feedforward Style Transfer Procedure

Once trained, the inverse network can be used to replace the optimization procedure. Thus our proposed feedforward procedure consists of the following steps:

1. Compute $\Phi(C)$ and $\Phi(S)$.
2. Obtain $\Phi^{ss}(C, S)$ by style swapping.
3. Feed $\Phi^{ss}(C, S)$ into a trained inverse network.

This procedure is illustrated in Figure 4. As described in Section 3.2, style swapping can be implemented as a (non-differentiable) convolutional neural network. As such, the entire feedforward procedure can be seen as a neural net with individually trained parts. Compared to existing feedforward approaches [6, 17, 33, 34], the biggest advantage of our feedforward procedure is the ability to use new style images with only a single trained inverse network.

5. Experiments

In this section, we analyze properties of the proposed style transfer and inversion methods. We use the Torch7 framework [2] to implement our method[2], and use existing open source implementations of prior works [16, 21, 30] for comparison.

5.1. Style Swap Results

Target Layer. The effects of style swapping in different layers of the VGG-19 network are shown in Figure 3.

---

[2]Code available at https://github.com/rtqichen/style-swap

Figure 5: Our method achieves consistent results compared to existing optimization formulations. We see that Gatys *et al.*'s formulation [11] has multiple local optima while we are able to consistently achieve the same style transfer effect with random initializations. Figure 6 shows this quantitatively.
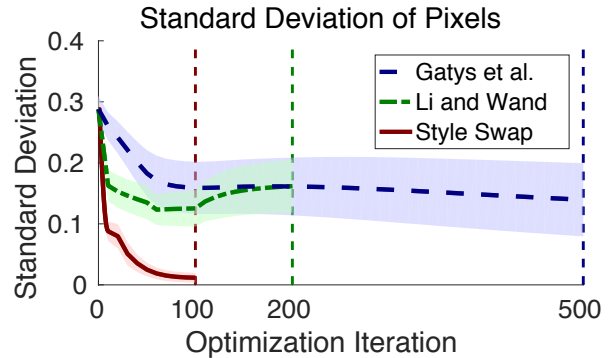

Figure 6: Standard deviation of the RGB pixels over the course of optimization is shown for 40 random initializations. The lines show the mean value and the shaded regions are within one standard deviation of the mean. The vertical dashed lines indicate the end of optimization. Figure 5 shows examples of optimization results.

In this figure the RGB images are computed by optimization as described in Section 3. We see that while we can style swap directly in the RGB space, the result is nothing more than a recolor. As we choose a target layer that is deeper in the network, textures of the style image are more pronounced. We find that style swapping on the "relu3_1" layer provides the most visually pleasing results, while staying structurally consistent with the content. We restrict our method to the "relu3_1" layer in the following experiments and in the inverse network training. Qualitative results are shown in Figure 10.

Consistency. Our style swapping approach concatenates the content and style information into a single target feature vector, resulting in an easier optimization formulation com-

patch size 3×3 patch size 7×7 patch size 12×12
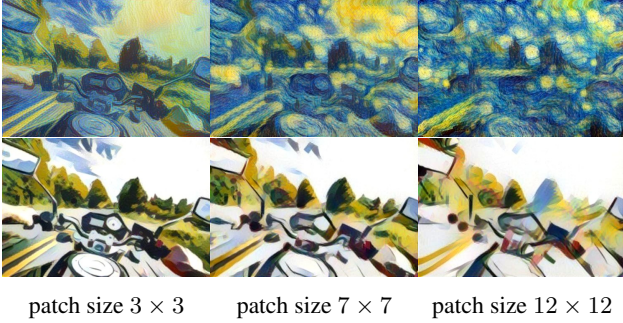
Figure 7: We can tradeoff between content structure and style texture by tuning the patch size. The style images, *Starry Night* (top) and *Small Worlds I* (bottom), are shown in Figure 10.
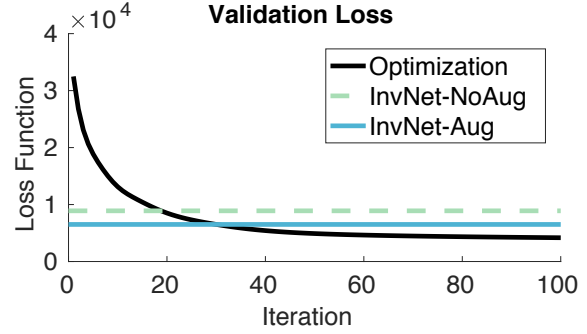

Figure 8: We compare the average loss (7) achieved by optimization and our inverse networks on 2000 variable-sized validation images and 6 variable-sized style images, using patch sizes of 3×3. Style images that appear in the paintings dataset were removed during training.

pared to other approaches. As a result, we find that the optimization algorithm is able to reach the optimum of our formulation in less iterations than existing formulations while consistently reaching the same optimum. Figures 5 and 6 show the difference in optimization between our formulation and existing works under random initializations. Here we see that random initializations have almost no effect on the stylized result, indicating that we have far fewer local optima than other style transfer objectives.

Straightforward Adaptation to Video. This consistency property is advantageous when stylizing videos frame by frame. Frames that are the same will result in the same stylized result, while consecutive frames will be stylized in similar ways. As a result, our method is able to adapt to video without any explicit gluing procedure like optical flow [30]. We place stylized videos in the code repository.

Simple Intuitive Tuning. A natural way to tune the degree of stylization (compared to preserving the content) in the proposed approach is to modify the patch size. Figure 7 qualitatively shows the relationship between patch size and the style-swapped result. As the patch size increases, more of the structure of the content image is lost and replaced by textures in the style image.

5.2. CNN Inversion

Here we describe our training of an inverse network that computes an approximate inverse function of the pre-trained VGG-19 network [31]. More specifically, we invert the truncated network from the input layer up to layer "relu3_1". The network architecture is placed in the appendix.

Dataset. We train using the Microsoft COCO (MSCOCO) dataset [23] and a dataset of paintings sourced from wikiart.org and hosted by Kaggle [5]. Each dataset has roughly $80,000$ natural images and paintings, respectively. Since typically the content images are natural images and

style images are paintings, we combine the two datasets so that the network can learn to recreate the structure and texture of both categories of images. Additionally, the explicit categorization of natural image and painting gives respective content and style candidates for the augmentation described in Section 4.1.

Training. We resize each image to 256×256 pixels (corresponding to activations of size 64×64) and train for approximately 2 epochs over each dataset. Note that even though we restrict the size of our training images (and corresponding activations), the inverse network is fully convolutional and can be applied to arbitrary-sized activations after training.

We construct each minibatch by taking 2 activation samples from natural images and 2 samples from paintings. We augment the minibatch with 4 style-swapped activations using all pairs of natural images and paintings in the minibatch. We calculate subgradients using backpropagation on (7) with the total variance regularization coefficient $\lambda = 10^{-6}$ (the method is not particularly sensitive to this choice), and we update parameters of the network using the Adam optimizer [18] with a fixed learning rate of 10^{-3}.

Result. Figure 8 shows quantitative approximation results using 2000 full-sized validation images from MSCOCO and 6 full-sized style images. Though only trained on images of size 256×256, we achieve reasonable results for arbitrary full-sized images. We additionally compare against an inverse network that has the same architecture but was not trained with the augmentation. As expected, the network that never sees style-swapped activations during training performs worse than the network with the augmented training set.

Method	N. Iters.	Time/Iter. (s)	Total (s)
Gatys *et al.* [11]	500	0.1004	50.20
Li and Wand [21]	200	0.6293	125.86
Style Swap (Optim)	100	0.0466	4.66
Style Swap (InvNet)	1	1.2483	1.25

Table 1: Mean computation times of style transfer methods that can handle arbitrary style images. Times are taken for images of resolution 300×500 on a GeForce GTX 980 Ti. Note that the number of iterations for optimization-based approaches should only be viewed as a very rough estimate.
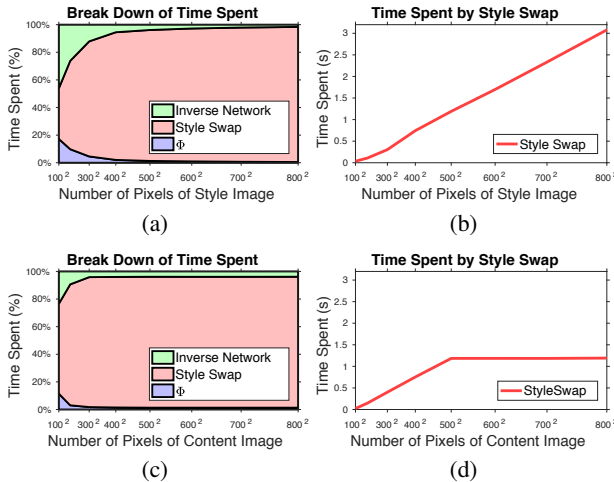

Figure 9: Compute times as (a,b) style image size increases and (c,d) as content image size increases. The non-variable image size is kept at 500×500. As shown in (a,c), most of the computation is spent in the style swap procedure.

5.3. Computation Time

Computation times of existing style transfer methods are listed in Table 1. Compared to optimization-based methods, our optimization formula is easier to solve and requires less time per iteration, likely due to only using one layer of the pretrained VGG-19 network. Other methods use multiple layers and also deeper layers than we do.

We show the percentage of computation time spent by different parts of our feedforward procedure in Figures 9a and 9c. For any nontrivial image sizes, the style swap procedure requires much more time than the other neural networks. This is due to the style swap procedure containing two convolutional layers where the number of filters is the number of style patches. The number of patches increases linearly with the number of pixels of the image, with a constant that depends on the number of pooling lay-

ers and the stride at which the patches are extracted. Therefore, it is no surprise that style image size has the most effect on computation time (as shown in Figures 9a and 9b).

Interestingly, it seems that the computation time stops increasing at some point even when the content image size increases (Figure 9d), likely due to parallelism afforded by the implementation. This suggests that our procedure can handle large image sizes as long as the number of style patches is kept manageable. It may be desirable to perform clustering on the style patches to reduce the number of patches, or use alternative implementations such as fast approximate nearest neighbour search methods [12, 28].

6. Discussion

We present a new CNN-based method of artistic style transfer that aims to be both fast and adaptable to arbitrary styles. Our method concatenates both content and style information into a single layer of the CNN, by swapping the textures of the content image with those of the style image. The simplistic nature of our method allows us to train an inverse network that can not only approximate the result in much less time, but can also generalize beyond its trained set of styles. Despite the one-layer restriction, our method can still produce visually pleasing images. Furthermore, the method has an intuitive tuning parameters (the patch size) and its consistency allows simple frame-by-frame application to videos.

While our feedforward method does not compete with the feedforward approach of Johnson *et al.* [17] in terms of speed, it should be noted that the biggest advantage of our method is the ability to stylize using new style images, whereas Johnson *et al.*'s method requires training a new neural network for each style image. We've shown that our inverse method can generalize beyond its training set (Figure 8), while another useful application is the ability to change the size of the style image without retraining the inverse network.

Some drawbacks of the proposed style swap procedure include lack of a global style measurement and lack of a similarity measure for neighboring patches, both in the spatial domain and the temporal domain. These simplifications sacrifice quality for efficiency, but can occassionally lead to a local flickering effect when applied to videos frame-by-frame. It may be desirable to look for ways to increase quality while keeping the efficient and versatile nature of our algorithm.

References

[1] H. A. Aly and E. Dubois. Image up-sampling using total-variation regularization with a new observation model. *IEEE Transactions on Image Processing*, 14(10):1647–1659, 2005. 4
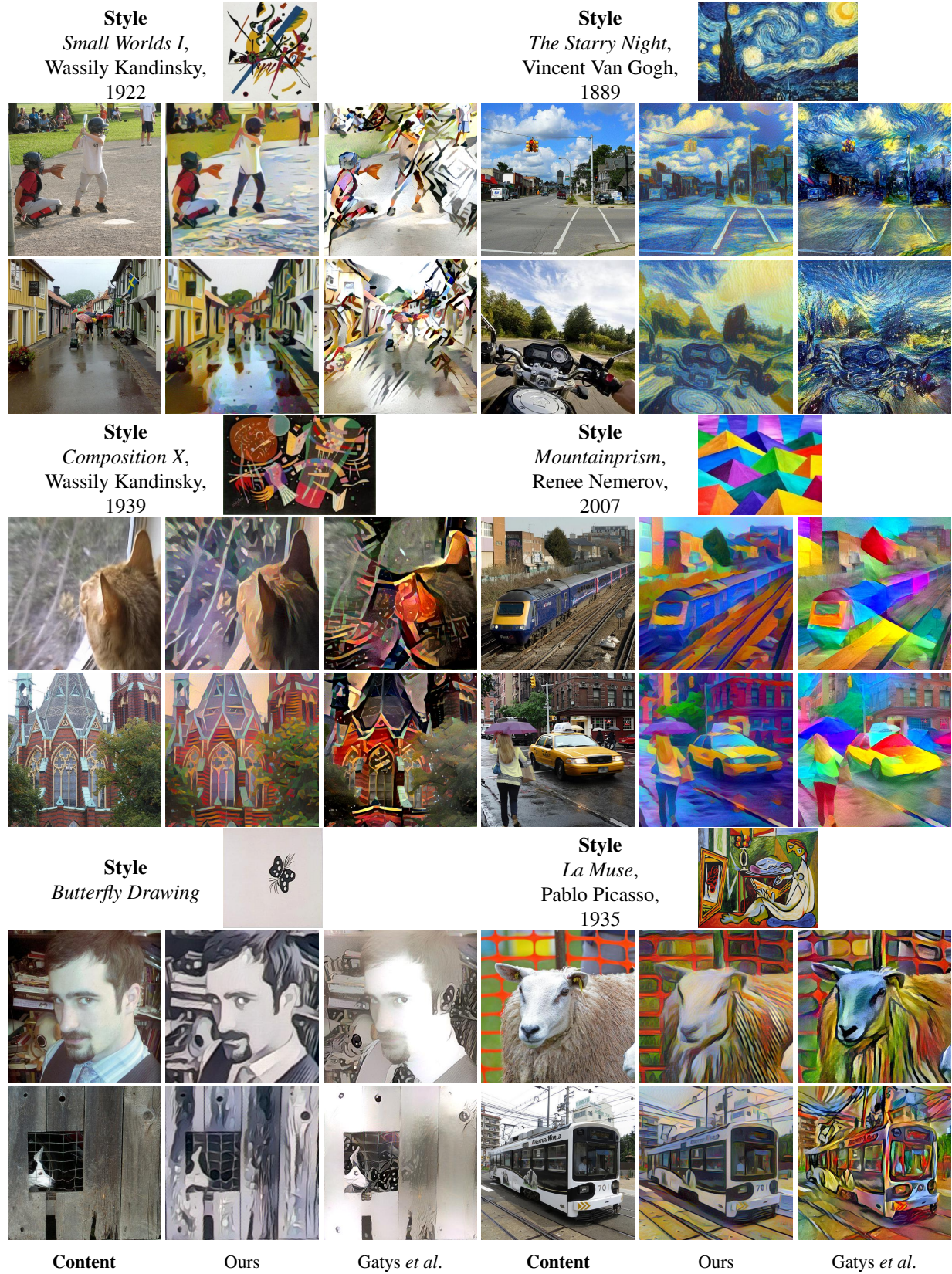
Figure 10: Qualitative examples of our method compared with Gatys *et al.*'s formulation of artistic style transfer. We show results using 3×3 patches (most content-similar) and with the default configuration for Gatys *et al.* [11].

[2] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. 5

[3] A. Dosovitskiy and T. Brox. Inverting convolutional networks with convolutional networks. *CoRR*, abs/1506.02753, 2015. 2

[4] A. Dosovitskiy, J. Springenberg, M. Tatarchenko, and T. Brox. Learning to generate chairs, tables and cars with convolutional networks. 4

[5] S. Y. Duck. Painter by numbers, wikiart.org. https://www.kaggle.com/c/painter-by-numbers, 2016. 6

[6] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *CoRR*, abs/1610.07629, 2016. 1, 2, 5

[7] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001. 1

[8] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE, 1999. 1

[9] M. Elad and P. Milanfar. Style-transfer via texture-synthesis. *arXiv preprint arXiv:1609.03057*, 2016. 1, 2

[10] O. Frigo, N. Sabater, J. Delon, and P. Hellier. Split and match: Example-based adaptive patch sampling for unsupervised style transfer. 2016. 1, 2

[11] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. 1, 2, 3, 5, 7, 8

[12] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. 7

[13] A. Hertzmann. Paint By Relaxation. *Proceedings Computer Graphics International (CGI)*, pages 47–54, 2001. 1

[14] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340. ACM, 2001. 1

[15] A. Inc. Artify, 2016. 1

[16] J. Johnson. neural-style. https://github.com/jcjohnson/neural-style, 2015. 5

[17] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *Arxiv*, 2016. 1, 2, 4, 5, 7

[18] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[19] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015. 2

[20] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics (ToG)*, 24(3):795–802, 2005. 1

[21] C. Li and M. Wand. Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis. *Cvpr 2016*, page 9, 2016. 1, 2, 3, 5, 7

[22] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3):127–150, 2001. 1

[23] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 6

[24] P. Litwinowicz. Processing Images and Video for an Impressionist Effect. *Proc. SIGGRAPH*, pages 407–414, 1997. 1

[25] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 4

[26] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *2015 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 5188–5196. IEEE, 2015. 2, 4

[27] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011. 2

[28] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014. 7

[29] I. Prisma Labs. Prisma. http://prisma-ai.com/, 2016. 1

[30] M. Ruder, A. Dosovitskiy, and T. Brox. Artistic style transfer for videos. pages 1–14, 2016. 2, 5, 6

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4, 6

[32] P. P. Studio. Picsart. https://picsart.com/, 2016. 1

[33] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. *CoRR*, 2016. 1, 2, 5

[34] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. 1, 2, 5

[35] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000. 1

[36] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010. 2, 4

Appendix

Inverse Network Architecture

The architecture of the truncated VGG-19 network used in the experiments is shown in Table A1, and the inverse network architecture is shown in Table A2. It is possible that better architectures achieve better results, as we did not try many different types of convolutional neural network architectures.

- Convolutional layers use filter sizes of 3×3, padding of 1, and stride of 1.

- The rectified linear unit (ReLU) layer is an elementwise function $\text{ReLU}(x) = \max\{x, 0\}$.

- The instance norm (InstanceNorm) layer standardizes each feature channel independently to have 0 mean and a standard deviation of 1. This layer has shown impressive performance in image generation networks [34].

- Maxpooling layers downsample by a factor of 2 by using filter sizes of 2×2 and stride of 2.

- Nearest neighbor (NN) upsampling layers upsample by a factor of 2 by using filter sizes of 2×2 and stride of 2.

Layer Type	Activation Dimensions
Input	$H \times W \times 3$
Conv-ReLU	$H \times W \times 64$
Conv-ReLU	$H \times W \times 64$
MaxPooling	$1/2H \times 1/2W \times 64$
Conv-ReLU	$1/2H \times 1/2W \times 128$
Conv-ReLU	$1/2H \times 1/2W \times 128$
MaxPooling	$1/4H \times 1/4W \times 128$
Conv-ReLU	$1/4H \times 1/4W \times 256$

Table A1: Truncated VGG-19 network from the input layer to "relu3_1" (last layer in the table).

Layer Type	Activation Dimensions
Input	$1/4H \times 1/4W \times 256$
Conv-InstanceNorm-ReLU	$1/4H \times 1/4W \times 128$
NN-Upsampling	$1/2H \times 1/2W \times 128$
Conv-InstanceNorm-ReLU	$1/2H \times 1/2W \times 128$
Conv-InstanceNorm-ReLU	$1/2H \times 1/2W \times 64$
NN-Upsampling	$H \times W \times 64$
Conv-InstanceNorm-ReLU	$H \times W \times 64$
Conv	$H \times W \times 3$

Table A2: Inverse network architecture used for inverting activations from the truncated VGG-19 network.