A Flexible Convolutional Solver for Fast Style Transfers

Gilles Puy
Technicolor
975 Avenue des Champs Blancs
F-35576 Cesson-Sévigné
gilles.puy@technicolor.com

Patrick Pérez
Valeo.ai
15 Rue de la Baume
F-75008 Paris
patrick.perez@valeo.com

Abstract

We propose a new flexible deep convolutional neural network (convnet) to perform fast neural style transfers. Our network is trained to solve approximately, but rapidly, the artistic style transfer problem of [15] for arbritary styles. While solutions already exist, our network is uniquely flexible by design: it can be manipulated at runtime to enforce new constraints on the final output. As examples, we show that it can be modified to perform tasks such as fast photo style transfer, or fast video style transfer with short term consistency, with no retraining. *This flexibility stems from the proposed architecture which is obtained by unrolling the gradient descent algorithm used in [15]. Regularisations added to [15] to solve a new task can be reported on-the-fly in our network, even after training.*

1. Introduction

Style transfer is a longstanding problem [12, 23, 38, 13] for which impressive results have recently been obtained with deep convnets. In [15], a stylised image is obtained by minimisation of a loss built from features provided by a pre-trained convnet. The loss involves two terms: the first preserves the content of one image; the second transfers the style of another image. The drawback of this method is its speed as the stylised image is the result of a long optimisation process. Nevertheless, this method is highly flexible as the original style transfer loss can be manipulated to guide the solution towards a desired result. This flexibility allows one to control perceptual factors during style transfer [16]. It permits one to augment the original style loss with regularisers favouring temporal consistency to perform video style transfer [40, 41]. It also allows one to achieve photo style transfer with the use of a regulariser encouraging the transformation to be locally affine [33]. This flexibility is crucial to construct a tool giving maximum freedom to an artist. A methodological challenge is thus to make the method of [15] faster while keeping most of its original flexibility, hence avoiding retraining for any new style transfer feature one wishes to add.

All the methods solving [15] rapidly rely on the same principle: a deep network taking as input a natural image is trained to estimate a minimiser of the style transfer loss, hence obtaining a stylised version of the input. The first fast methods [26, 48] crucially lacked flexibility as one network needed to be trained for each style. This shortcoming was partly addressed by subsequent works showing that the style can be encoded by a small subset of the network parameters [6, 11]. Recent works show that it is possible to train a network that takes as inputs any pair of painting and content images and produces a stylised image, even for paintings not viewed at training time [17, 25, 29]. Deep networks trained for fast *artistic* style transfer have thus gained more and more flexibility. Yet, these fast style transfer methods are not as flexible as the method of [15] as one cannot plug easily a new feature in them. For example, fast video style transfer is achieved after adaptation of the network architecture and specific retraining [5, 20, 24, 41]. Similarly, the authors of [30] had to adapt the network architecture of [29] and retrain it to get it to work for fast photo style transfer.

Contributions. Specialising a network for each style transfer task is a cumbersome work. Our main contribution is to propose instead a new flexible network that is (a) trained to solve rapidly the artistic style transfer problem of [15] for arbitrary styles via unsupervised learning and (b) which can be modified at test time to take into account important modifications of the artistic style transfer loss. These modifications do not need to be known at training time. *No retraining* is required when adding them in the network. This high level of flexibility stems from the proposed network architecture that has the structure of the gradient descent algorithm used in [15]: modifications brought to the initial style transfer loss can be reported directly in our trained network. Like existing methods, our network yields fast controllable artistic style transfer results but, unlike them, it can be used to perform several other tasks without retraining. This flexibility permits us to make this net-

work perform photo style transfer faster than the state-of-the-art methods, which constitutes our second contribution. Our third contribution is to show that we can introduce, at testing time, a regulariser to stylise videos without suffering from flickering artefacts. In comparison, existing state-of-the-art methods which, like ours, are not specifically designed for this task suffers from flickering even when using the same regulariser. As a last illustration of the network's flexibility, we show that it can be used for user-guided[1] texture super-resolution, again without retraining. Finally, let us highlight that, unlike several works, we train our network without using any painting as style but show that it generalises to paintings.

2. Other related works

Related works about visual style transfer are discussed in the introduction. We discuss here related works that inspired the design of our network architecture.

Algorithms unrolling. A wide range of classic iterative solvers amount to repeated applications of linear and component-wise non-linear maps. Unfolding such an algorithm over a fixed number of iterations allows one to see it as several layers of a neural network with shared, predefined weights. This can be taken as the starting point for an actual, trainable neural net. This idea was introduced by [19] in the context of sparse coding: the iterative shrinkage thresholding algorithm (ISTA) [2] gives rise of a learnable network (LISTA) for fast approximate sparse coding. Each layer has independent trainable weights and the whole network is trained under supervision. In the related context of linear inverse problems, including compressed sensing [3, 36, 37, 34, 52] and image restoration [39, 7, 32, 35, 49, 51], several works have followed this unrolling approach. In some cases, all the weights (tied or untied) are trained for the specific task to solve, *e.g.*, [49]. In some other cases, only the weights defining the image prior are trained, the other weights being defined by the data fidelity term of the problem to solve, *e.g.*, [7]. Note that, by essence of such inverse problems, full supervision is possible: input-output samples of the transform to be inverted are indeed readily obtained, *e.g.*, by sensing or artificially degrading natural images. By contrast, our approach is fully unsupervised, as other fast style transfer alternatives.

Unsupervised neural solvers. Training without supervision a (fast) feed-forward network to solve approximately a complex optimization problem is a challenging problem, with important applications. Using the cost function of interest as training loss seems straightforward, but the amount of actual guidance provided by the loss must be sufficient for such an unsupervised training to succeed. Besides works in fast style transfer, other recent works have demonstrated successful instances of this paradigm. In [46], efficient fluid simulation is addressed. Within each time step of a classic solver of the incompressible Euler equation, the costly solving of the Poisson equation on the pressure field is replaced by a trained 3D convnet. In this case the loss amounts to the divergence square norm. In a different context, [45] learn a deep neural solver for a complex inverse rendering problem: estimating the shape, expression and reflectance of a face in a single image, by minimization of the pixel-wise discrepancy between rendered and real faces. In both works, trained solvers provide good approximation of the solutions of the initial minimisation problem, but with massive acceleration compared to iterative solvers. In the present paper, we also propose to learn an unsupervised neural solver for a complex optimization problem, but following the unrolling principle explained above. As a consequence, the acceleration, already obtained by others, is not our main contribution. More importantly, the unrolling approach gives the possibility to adapt at runtime our network in order to accommodate important changes to the original cost function.

3. Network architecture

3.1. Style transfer by gradient descent

Style transfer consists in transforming an image $\mathsf{X}_c \in \mathbb{R}^{n\times 3}$ – of n pixels and 3 color channels[2] – to give it the "style" of another image $\mathsf{X}_s \in \mathbb{R}^{n'\times 3}$ while preserving the "content" in X_c. In [15], the content and style of an image are defined using deep features obtained from VGG-19 [44]. The style transfer method then consists in solving a minimisation problem involving these deep features. The minimisation is done over an image $\mathsf{X} \in \mathbb{R}^{n\times 3}$. We denote by $\mathsf{F}_\ell, \mathsf{C}_\ell \in \mathbb{R}^{n_\ell \times c_\ell}$ and $\mathsf{S}_\ell \in \mathbb{R}^{n'_\ell \times c_\ell}$ the features at the ℓ^{th} layer of VGG-19 for the images X, X_c and X_s, respectively. In [15], the features C_ℓ encode the content of X_c while the Gram matrices $\mathsf{S}_\ell^\intercal \mathsf{S}_\ell$ encode the style of X_s. The loss to minimise for style transfer thus takes the form

$$\mathcal{L}(\mathsf{X}) = \mathcal{L}_c(\mathsf{X}, \mathsf{X}_c) + \mathcal{L}_s(\mathsf{X}, \mathsf{X}_s), \tag{1}$$

where the content loss $\mathcal{L}_c(\mathsf{X}, \mathsf{X}_c) = \sum_{\ell\in\mathcal{I}_c} \lambda_c^\ell \left\|\mathsf{F}_\ell - \mathsf{C}_\ell\right\|_{\mathrm{F}}^2 / (n_\ell\, c_\ell)$, with $\lambda_c^\ell \geqslant 0$, ensures transfer of the content of X_c to the final image ($\|\cdot\|_{\mathrm{F}}$ stands for matrix Frobenius norm), while the style loss $\mathcal{L}_s(\mathsf{X}, \mathsf{X}_s) = \sum_{\ell\in\mathcal{I}_s} \lambda_s^\ell\, \mathcal{L}_s^\ell(\mathsf{X}, \mathsf{X}_s)$ with

$$\mathcal{L}_s^\ell(\mathsf{X}, \mathsf{X}_s) = \frac{1}{c_\ell^2} \left\| \frac{1}{n_\ell}\, \mathsf{F}_\ell^\intercal \mathsf{F}_\ell - \frac{1}{n'_\ell}\, \mathsf{S}_\ell^\intercal \mathsf{S}_\ell \right\|_{\mathrm{F}}^2 \tag{2}$$

and $\lambda_s^\ell \geqslant 0$, ensures transfer of the style of X_s to the final image. We use $\mathcal{I}_s = \{\text{conv}_{1\_1}, \text{conv}_{2\_1}, \text{conv}_{3\_1}, \text{conv}_{4\_1},$

[1] We consider a setting where the user provides a texture example to guide the result of the super-resolution algorithm.

[2] Throughout, the two spatial dimensions of images and features maps are flattened into a single vector dimension for notational convenience.

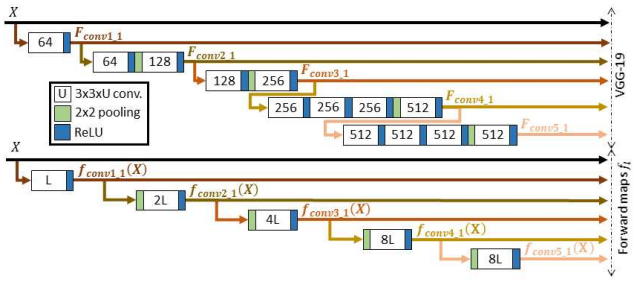

Figure 1. Comparison of the architecture of VGG-19 and our forward maps f_ℓ.

$\text{conv}_{5\_1}\}$ for the style loss and $\mathcal{I}_c = \{\text{conv}_{4\_2}\}$ for the content loss, as, *e.g.*, in [15].

A stylised image X^* can be obtained by starting from an image $\mathsf{X}^{(0)}$ and by progressively updating it to minimise the loss $\mathcal{L}$ (1) by gradient descent:

$$\mathsf{X}^{(t+1)} = \mathsf{X}^{(t)} - \mu \, \nabla \mathcal{L}(\mathsf{X}^{(t)}), \tag{3}$$

where $\mu > 0$ is the stepsize. While achieving impressive results, this method is nevertheless slow but convnets have been designed to minimise (1) approximately at a much lower computational cost [17, 26, 48].

3.2. Learned gradient descent for style transfer

3.2.1 Global architecture

We propose a new convnet to achieve fast style transfer. Its architecture follows the update rule of the gradient descent algorithm (3) where the actual gradient is replaced by a learned update g_t:

$$\mathsf{X}^{(t+1)} = \mathsf{X}^{(t)} - g_t\left(\mathsf{X}^{(t)}, \mathsf{X}_s\right), \quad t = 0, \ldots, N-1, \tag{4}$$

where N is the number of unrolled iterations. Note that the proposed network can be viewed as a residual network [22].

If one follows exactly the idea of unrolling [19], the computational architecture of g_t should be identical to the architecture of $\nabla\mathcal{L}$. To reduce the computational costs and number of parameters to train, we allow ourselves to make few simplifications. While the gradient $\nabla\mathcal{L}$ is made of a content term and a style term, we design g_t by mimicking the computational architecture of $\nabla\mathcal{L}_s$ only; hence, the sole dependence of g_t on X_s in (4). However, as in [26], we initialise $\mathsf{X}^{(0)}$ with the RGB image X_c, instead of the classic random initialization of the gradient descent, and we use the complete cost $\mathcal{L}$ in the training loss. This allows us to keep the content of X_c in the final image $\mathsf{X}^{(N)}$.

3.2.2 Architecture of g_t.

The gradient $\nabla\mathcal{L}_s$, which g_t should replace, is obtained in 4 steps:

(i) computing each feature F_ℓ via a forward-pass in VGG-19;

(ii) computing the partial derivative:

$$\frac{\partial \mathcal{L}_s^\ell}{\partial \mathsf{F}_\ell} \propto \mathsf{F}_\ell \cdot \left[\frac{1}{n_\ell}\,\mathsf{F}_\ell^\intercal \mathsf{F}_\ell - \frac{1}{n'_\ell}\,\mathsf{S}_\ell^\intercal \mathsf{S}_\ell\right]; \tag{5}$$

(iii) back-propagating this partial derivative to the input to obtain $\nabla\mathcal{L}_s^\ell$;

(iv) compute the weighted sum $\nabla\mathcal{L}_s = \sum_{\ell\in\mathcal{I}_s} \lambda_s^\ell \nabla\mathcal{L}_s^\ell$.

We construct g_t by mimicking these 4 steps, but we replace the original VGG-19 by a new convnet whose filters are trained to stylize any natural image in N learned updates (4). To simplify notation, we drop the iteration index t below. However, we highlight that all the filters are different at each iteration t (untied weights) in our implementation.

[Transforming Step (i)] We replace the VGG-19 features F_ℓ by new features $f_\ell(\mathsf{X})$, $\ell = \text{conv}_{1\_1}, \ldots, \text{conv}_{5\_1}$ (see Fig. 1). We call each f_ℓ a forward map as it replaces the forward pass in VGG-19. For each VGG-19 layer that is involved in the style loss, we have a corresponding layer. In both architectures, the spatial dimensions are successively halved and the number of channels doubled (except at the last layer). VGG-19 has at least an extra convolutional layer before any pooling. The initial number of channels is 64 in VGG-19 and L in our network.

[Transforming Step (ii)] The computation of this partial derivative can be transformed by substituting $f_\ell(\mathsf{X})$ for F_ℓ in (5):

$$f_\ell(\mathsf{X}) \cdot \left(\frac{1}{n_\ell}\, f_\ell(\mathsf{X})^\intercal\, f_\ell(\mathsf{X}) \;-\; \frac{1}{n'_\ell}\,\mathsf{G}_\ell^\intercal\,\mathsf{S}_\ell^\intercal \mathsf{S}_\ell\,\mathsf{H}_\ell\right). \tag{6}$$

However, we faced sudden explosions of the loss during training when using (6). We suspect that this effect is due to the matrix product $f_\ell(\mathsf{X}) f_\ell(\mathsf{X})^\intercal f_\ell(\mathsf{X})$ as it can amplify large values during inference and backpropagation. Therefore, we substituted

$$f_\ell(\mathsf{X}) \cdot \left(\frac{1}{n'_\ell}\,\mathsf{G}_\ell^\intercal\,\mathsf{S}_\ell^\intercal \mathsf{S}_\ell\,\mathsf{H}_\ell\right) \tag{7}$$

for (6). The learned matrices $\mathsf{G}_\ell, \mathsf{H}_\ell$ reduce the dimension of the VGG-19 style matrix $\mathsf{S}_\ell^\intercal \mathsf{S}_\ell$ to make it compatible with the channel dimension of our network. The matrix $\mathsf{G}_\ell^\intercal\,\mathsf{S}_\ell^\intercal \mathsf{S}_\ell\,\mathsf{H}_\ell$, called style filter in Fig. 2, governs the style that is applied to the input image. This filter corrects $f_\ell(\mathsf{X})$ so that the style of X gets closer to the target style after (mimicked) backpropagation. Any style can be applied at runtime by computing the VGG-19 features S_ℓ of the chosen style image. Let us clarify that we use the original pretrained VGG-19 to compute the features S_ℓ.

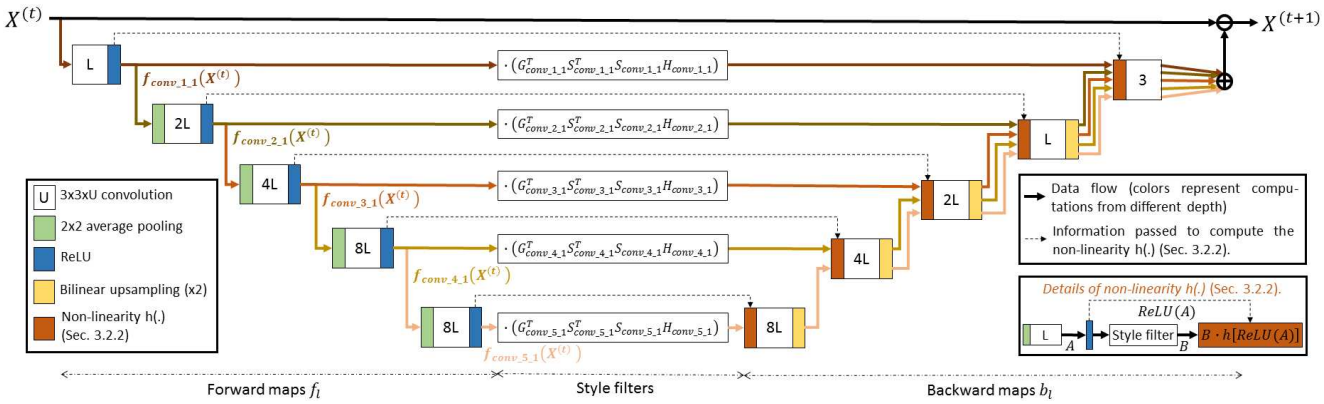

Figure 2. Structure of one learned iteration g_t for fast style transfer. The features S_ℓ are the VGG-19 features of the style image, allowing application of arbitrary styles at runtime. The matrices $\mathsf{G}_\ell, \mathsf{H}_\ell$, and all convolutional filters are learned. The parameter L governs the number of channels at each layer and the number of filters in one learned iteration.

[**Transforming (iii)**] We mimic backpropagation using backwards maps b_ℓ. The structure of each b_ℓ is symmetric to the structure of the corresponding map f_ℓ. Each b_ℓ takes as input the partial derivative (7) at layer ℓ and transforms it to a learned update for this scale:

$$b_\ell \left(\frac{1}{n'_\ell} f_\ell(\mathsf{X}) \cdot \mathsf{G}_\ell^\intercal \, \mathsf{S}_\ell^\intercal \mathsf{S}_\ell \, \mathsf{H}_\ell \right). \tag{8}$$

Note that during exact backpropagation to compute $\nabla \mathcal{L}_s^\ell$, the non-linearities applied are not ReLUs but subgradients of ReLU. Similarly, the non-linearities we use in b_ℓ are also subgradients of ReLU. These non-linearities have the form $\mathsf{B} \cdot h(\mathrm{ReLU}(\mathsf{A}))$ (see Fig. 2), where " $\cdot$ " denotes the pointwise multiplication and $h(\cdot)$ is the heaviside step function:

$$h\left(\mathrm{ReLU}(\mathsf{A})\right)_{jc} = h\left(\mathsf{A}\right)_{jc} = \begin{cases} 0, & \text{if } \mathsf{A}_{jc} \leqslant 0, \\ 1, & \text{if } \mathsf{A}_{jc} > 0, \end{cases} \tag{9}$$

where j, c index the spatial position and the feature channel, respectively. This choice of non-linearities allowed us to obtain better results than when using ReLU in both f_ℓ and b_ℓ, as usually done in encoder-decoder architectures (see Sec. 5). During backpropagation, the gradient satisfies $\partial \mathsf{B} \cdot h(\mathsf{A}) + \mathsf{B} \cdot \partial h(\mathsf{A}) = \partial \mathsf{B} \cdot h(\mathsf{A})$, as $\partial h(\mathsf{A}) = 0$ (with a discontinuity at 0 treated below). Hence the gradient continues to flow wherever $\mathsf{A} > 0$. Concerning the discontinuity at 0, one should note that A is obtained after convolution and *before* ReLU. There is low probability that any value of A is exactly 0. If this event occurs, we can set $\partial h(0) = 0$.

[**Transforming (iv)**] The global learned update is obtained by summing up the learned updates (8) for the 5 scales $\mathcal{I}_s$ involved in the style loss:

$$g\left(\mathsf{X}, \mathsf{X}_s\right) = \sum_{\ell \in \mathcal{I}_s} \lambda_s^\ell \, b_\ell \left(\frac{1}{n'_\ell} f_\ell(\mathsf{X}) \times \mathsf{G}_\ell^\intercal \, \mathsf{S}_\ell^\intercal \mathsf{S}_\ell \, \mathsf{H}_\ell \right). \tag{10}$$

The complete structure of g_t is presented in Fig. 2. All convolutions are computed using reflection padding. Note that the matrix multiplication denoted by " $\times$ " in (7) can be implemented as a convolution with a filter of spatial size 1×1, so that the whole network is convolutional.

3.2.3 Connections with existing methods

Our convnet architecture is fundamentally different from the one proposed by [26] and by follow-up works such as [11, 17]. Beyond the residual architecture, the main difference resides in the control of the style: we control it by filtering the features f_ℓ independently at each scale, while the style is controlled by instance normalisation parameters in [11, 17]. Our convnet architecture shares more similarities with [6, 29, 30]. While these architectures do not mimick the iterative procedure of the gradient descent algorithm, the style is also controlled by filtering deep features. We note however that [29, 30] require matrix inversions for style transfer, whereas ours does not.

4. Runtime restructuring

In this section, we explain how our network can be restructured at runtime to enforce new properties on the stylised image. Results obtained via various such restructurings are given in Section 5.

4.1. Runtime modification of the style loss $\mathcal{L}_s(\mathsf{X}, \mathsf{X}_s)$

In [16], the authors leverage the flexibility of the method of [15] to control perceptual factors during style transfer. Such controls are done via modifications of the original style loss $\mathcal{L}_s$. As our network has the same structure as $\nabla \mathcal{L}_s$, these modifications brought to $\mathcal{L}_s$ can be transferred in our network, at runtime, in a systematic way. It indeed

suffices to study the consequences of this change in $\nabla\mathcal{L}_s$ and report them in our network. We give 3 examples below.

In the method of [15], one can control the effect of each style scale during stylisation by adapting the weights λ_s^ℓ in (1). Any change of these weights directly impacts the weighted sum in Step (iv) of the computation of $\nabla\mathcal{L}_s$. We can thus control the effect of each style scale in our network by using the new value of λ_s^ℓ in (10).

One can also wish to mix different style images X_s^i for stylisation. Such a mixing is possible in [15, 16] by using $\mathcal{L}_s\left(\mathsf{X}, \{\mathsf{X}_s^i\}\right) = \sum_i \alpha_i\, \mathcal{L}_s(\mathsf{X}, \mathsf{X}_s^i)$, with $\sum_i \alpha_i = 1$, as new style loss. The partial derivatives (ii) for this new loss simplify to

$$\mathsf{F}_\ell \cdot \left(\frac{1}{n_\ell}\, \mathsf{F}_\ell^\intercal \mathsf{F}_\ell - \frac{1}{n'_\ell} \sum_i \alpha_i\, \mathsf{S}_\ell^\intercal[\mathsf{X}_s^i] \mathsf{S}_\ell[\mathsf{X}_s^i] \right), \tag{11}$$

where $\mathsf{S}_\ell[\mathsf{X}_s^i]$ denotes the features of the image X_s^i at the ℓ^{th} layer of VGG-19. By propagating this modification, we can mix different styles in our network by changing (7) to

$$\frac{1}{n'_\ell}\, f_\ell(\mathsf{X}) \cdot \left[\mathsf{G}_\ell^\intercal \left(\sum_i \alpha_i\, \mathsf{S}_\ell^\intercal[\mathsf{X}_s^i] \mathsf{S}_\ell[\mathsf{X}_s^i] \right) \mathsf{H}_\ell \right]. \tag{12}$$

Finally, one can refine the above mixture of styles with spatial control, *e.g.*, by associating each style X_s^i to a different region in the content image. This is done by introducing masks in the style loss [16]. Let M_ℓ^i denote the mask at layer ℓ for the i^{th} style and i^{th} region. The masked style loss satisfies

$$\sum_{\ell\in\mathcal{I}_s} \frac{\lambda_s^\ell}{c_\ell^2} \sum_i \left\| \frac{1}{n_\ell} (\mathsf{M}_\ell^i \mathsf{F}_\ell)^\intercal (\mathsf{M}_\ell^i \mathsf{F}_\ell) - \frac{1}{n'_\ell}\, \mathsf{S}_\ell^\intercal[\mathsf{X}_s^i] \mathsf{S}_\ell[\mathsf{X}_s^i] \right\|_{\mathrm{F}}^2.$$

After propagation of these changes in our network, spatial control of the style is obtained by changing (7) to $\sum_i {n'_\ell}^{-1}\, \mathsf{M}_\ell^i f_\ell(\mathsf{X}) \cdot \left(\mathsf{G}_\ell^\intercal\, \mathsf{S}_\ell^\intercal[\mathsf{X}_s^i] \mathsf{S}_\ell[\mathsf{X}_s^i]\, \mathsf{H}_\ell\right)$.

4.2. Adding new regularisers

Beyond the control of the style loss, one can easily impose additional constraints on the solution. Indeed, let $\mathcal{R}(\mathsf{X})$ denote a regulariser and augment (1) with it:

$$\min_{\mathsf{X}} \mathcal{L}(\mathsf{X}) + \mathcal{R}(\mathsf{X}). \tag{13}$$

A strategy to solve this problem is to use the proximal gradient descent algorithm whose iterations satisfy

$$\mathsf{X}^{(t+1)} = \mathrm{P}_{\mu\mathcal{R}} \left[\mathsf{X}^{(t)} - \mu\, \nabla\mathcal{L}\left(\mathsf{X}^{(t)}\right) \right], \tag{14}$$

where $\mathrm{P}_{\mu\mathcal{R}}$ is the proximal operator associated to $\mathcal{R}$:

$$\mathrm{P}_{\mu\mathcal{R}}\left(\mathsf{Y}\right) \in \arg\min_{\mathsf{X}} \frac{1}{2} \left\|\mathsf{X} - \mathsf{Y}\right\|_{\mathrm{F}}^2 + \mu\mathcal{R}(\mathsf{X}). \tag{15}$$

Proximal operators, which generalise projection operators, are widely used in optimisation [8]. Note that if $\mathcal{L}$ and $\mathcal{R}$ were convex, then the above algorithm would converge to a solution of (13) [8]. In the non-convex case, the above algorithm converges to a saddle point of (13) upon some conditions on $\mathcal{L}$ and $\mathcal{R}$ [1].

We remark that the only difference between (3) and (14) is the computation of $\mathrm{P}_{\mu\mathcal{R}}$ after each gradient update. By copying this modification in our network, the updates (4) become

$$\mathsf{X}^{(t+1)} = \mathrm{P}_{\mathcal{R}} \left[\mathsf{X}^{(t)} - g_t \left(\mathsf{X}^{(t)}, \mathsf{X}_s \right) \right]. \tag{16}$$

Hence, we are able to add the effect of any regulariser in our network at runtime – without any retraining. Our network inherits the flexibility of optimisation algorithms.

4.2.1 Photo style transfer

Minimizing (1) yields good results when the style is a painting. Unfortunately, the results are less impressive when the style is a photograph as the result is often not photo-realistic. To solve this issue, [33] propose to favour transformations from the content image to the final image that are locally affine. This is done by adding a penalty term to $\mathcal{L}$ in (1) which becomes $\mathcal{L}(\mathsf{X}) + \lambda_{\mathrm{L}}\, \mathrm{Tr}(\mathsf{X}^\intercal \mathsf{L} \mathsf{X})$, where L is the Matting Laplacian [28] of X_c. This regularisation can be integrated in our network easily via (16) with $\mathcal{R}(\mathsf{X}) = \lambda_{\mathrm{L}}\, \mathrm{Tr}(\mathsf{X}^\intercal \mathsf{L} \mathsf{X})$. Note that in this case, $\mathrm{P}_{\mathcal{R}}\left(\mathsf{Y}\right) = \left(\mathsf{I} + 2\lambda_{\mathrm{L}}\mathsf{L}\right)^{-1} \mathsf{Y}$. We remark that [30] propose to post-process the output of their network by multiplying it a matrix similar to $\left(\mathsf{I} + 2\lambda_{\mathrm{L}}\mathsf{L}\right)^{-1}$. In our network, this processing is done after each learned update, as done in the proximal gradient algorithm. Let us highlight another difference with [30]: we never invert the matrix $\left(\mathsf{I} + 2\lambda_{\mathrm{L}}\mathsf{L}\right)$ to compute $\mathrm{P}_{\mathcal{R}}\left(\mathsf{Y}\right)$. Instead, we use a computationally efficient method from graph signal processing [43]. The method is detailed in the supplementary material. In few words, it consists in viewing the estimation of $\mathrm{P}_{\mathcal{R}}\left(\mathsf{Y}\right)$ as filtering Y on a graph with Laplacian L. The corresponding filter can be approximated by a polynomial, which induces a fast graph filtering algorithm. This graph filtering technique is used for, *e.g.*, efficient wavelet decompositions on graphs [21], fast spectral clustering [47], or deep learning on graphs [9].

4.2.2 Video style transfer

One of the challenges in video style transfer is ensuring temporal consistency. The authors of [40] solve this problem by augmenting (1) with regularisers enforcing such a consistency. These regularisers can easily be integrated in our network. For simplicity, we consider the short term consistency regulariser proposed in [40] and limit ourselves to an online scenario: obtaining the $(i+1)^{\text{th}}$ stylised image

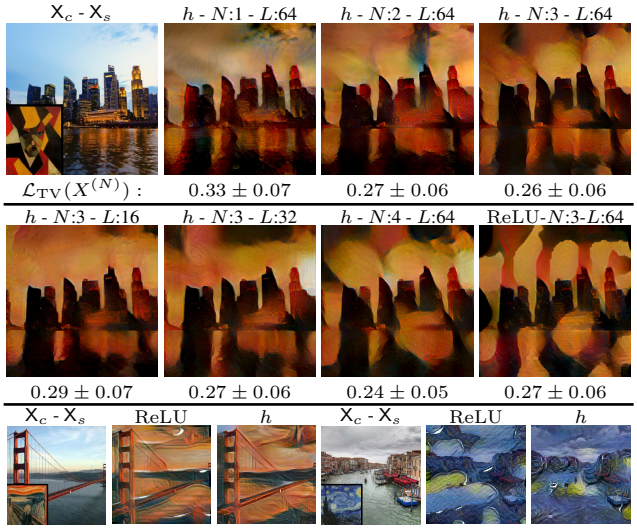

Figure 3. Effect of the choice of N, L, and non-linearity ($h(\cdot)$ or ReLU). The style appears in the bottom left corner of the content image. First and second rows: The mean achieved $\mathcal{L}_{\mathrm{TV}}$ ($\pm$ the standard deviation) on 1000 pairs of validation images appears below each model. Third row: Additional results obtained with $h(\cdot)$ or ReLU ($N = 3$, $L = 64$).

$\mathsf{X}^{(i+1)}$ of a video sequence while $\mathsf{X}^{(i)}$ was computed previously. Let $w_i(\cdot)$ be the function that warp the original image $\mathsf{X}_c^{(i)}$ onto $\mathsf{X}_c^{(i+1)}$, and M_i the binary mask indicating where this warping is valid (removing disocclusions and motion boundaries as in [40]). The $(i+1)^{\text{th}}$ stylised image can be obtained as a solution of (13) using $\mathcal{R}(\mathsf{X}) = \lambda_{\mathcal{R}} \left\| \mathsf{M}_i \odot (w_i(\mathsf{X}^{(i)}) - \mathsf{X}) \right\|_{\mathrm{F}}^2$, where $\odot$ is the pointwise multiplication. A solution of this problem can be estimated rapidly using our trained network via the updates (16). The proximal operator $\mathrm{P}_{\mathcal{R}}$ is fast to compute and satisfies $\mathrm{P}_{\mathcal{R}}(\mathsf{Y}) = \left[\mathsf{Y} + 2\lambda_{\mathcal{R}} \mathsf{M}_i^2 \odot w_i(\mathsf{X}^{(i)})\right] \odot \left[\mathsf{I} + 2\lambda_{\mathcal{R}} \mathsf{M}_i^2\right]^{-1}$, where the inverse is computed elementwise.

4.2.3 User-guided texture super-resolution

We consider the problem of upscaling a *low-resolution* texture $\mathsf{X}_{\mathrm{low}}$ in a user-guided scenario. First, a user provides a *high-resolution* image $\mathsf{X}_{\mathrm{ref}}$ of a texture that he judges similar to $\mathsf{X}_{\mathrm{low}}$. Then, we exploit this information to upscale $\mathsf{X}_{\mathrm{low}}$ by using the fact that two similar textures have similar VGG-19 Gram matrices [14]. This is done by minimising $\mathcal{L}_s(\mathsf{X}, \mathsf{X}_{\mathrm{ref}})$, where X is the image we are optimising on for upscaling $\mathsf{X}_{\mathrm{low}}$. We also seek to minimise $\mathcal{R}(\mathsf{X}) = \lambda_{\mathcal{R}} \left\| \mathsf{X}_{\mathrm{low}} - \mathsf{D}\mathsf{X} \right\|_{\mathrm{F}}^2$, where D models the downsampling operator, to ensure consistency of the solution with $\mathsf{X}_{\mathrm{low}}$. In total, we propose to solve $\min_{\mathsf{X}} \mathcal{L}_c(\mathsf{X}, \mathsf{X}_{\mathrm{bic}}) + \mathcal{L}_s(\mathsf{X}, \mathsf{X}_{\mathrm{ref}}) + \mathcal{R}(\mathsf{X})$. The image $\mathsf{X}_{\mathrm{bic}}$ denotes the bicubic upsampled version of $\mathsf{X}_{\mathrm{low}}$, and the term $\mathcal{L}_c(\mathsf{X}, \mathsf{X}_{\mathrm{bic}})$ acts as a second regulariser ensuring consistency between the observed low-resolution texture and the solution. A solution of this minimisation problem can be estimated rapidly using our trained network via the updates (16), starting from $\mathsf{X}^{(0)} = \mathsf{X}_{\mathrm{bic}}$. Note that the computation of $\mathrm{P}_{\mathcal{R}}$ can be estimated by, *e.g.*, gradient descent.

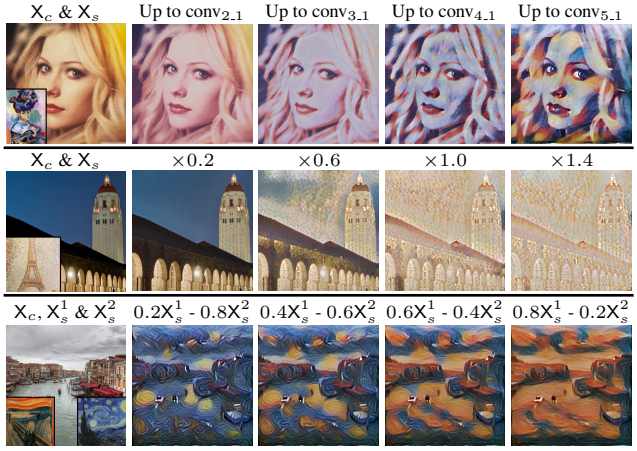

Figure 4. Choice of style scales (top row), global style intensity (middle), and mixing of styles at runtime. The styles used appear in the bottom corners of the content images (first column).

Let us acknowledge that this application was inspired by [42] where the resolution of a skin texture is improved using a database of such high-resolution textures.

5. Style transfer experiments

5.1. Training

As done in [26] for example, we augment $\mathcal{L}$ with the Total Variation (TV) regularisation and use $\sum_{t=1}^{N} \mathcal{L}_{\mathrm{TV}}(\mathsf{X}^{(t)})$, with $\mathcal{L}_{\mathrm{TV}}(\mathsf{X}) = \mathcal{L}(\mathsf{X}) + \lambda_{\mathrm{TV}} \mathrm{TV}(\mathsf{X})$, as training loss. The TV regularisation favours visual "naturalness" of $\mathsf{X}^{(N)}$. The pixel values of $\mathsf{X}^{(t)}$ are clipped between 0 and 1 before evaluation of the loss. All filters are initialised using Xavier initialisation [18], and trained using Adam [27] with a stepsize of $2 \cdot 10^{-5}$ on the 2014 MS-COCO training dataset [31]. The training images are centrally cropped to the largest possible square and resized to 320×320. At each iteration, two images are drawn at random: the first is used as a content image, the second as a style image. Independent draws of zero-mean uniform noise of amplitude randomly chosen in $[0, 0.1)$ is added to both images at each iteration. Note that, unlike in several works on artistic style transfer, we thus do not train our network on any painting. We use $\lambda_c^{\mathrm{conv}_{4\_2}} = 0.02$, $\lambda_{\mathrm{TV}} = 0.3$, and $\lambda_s^{\ell} = \beta^{\ell}/\alpha$ where $\alpha = \sum_{\ell \in \mathcal{I}_s} \left\| \mathsf{S}_{\ell}^{\mathsf{T}} \mathsf{S}_{\ell} \right\|_{\mathrm{F}}^2 / (c_{\ell}^2 \, n'^2_{\ell})$ and β^{ℓ} are independent random variables taking value 0 or 0.9^{-1} with probability 0.9 for the latter. The normalisation α favours visually similar degree of stylization across styles. The variables β^{ℓ} permit us to disentangle, via (10), the effect of each style scale ℓ in

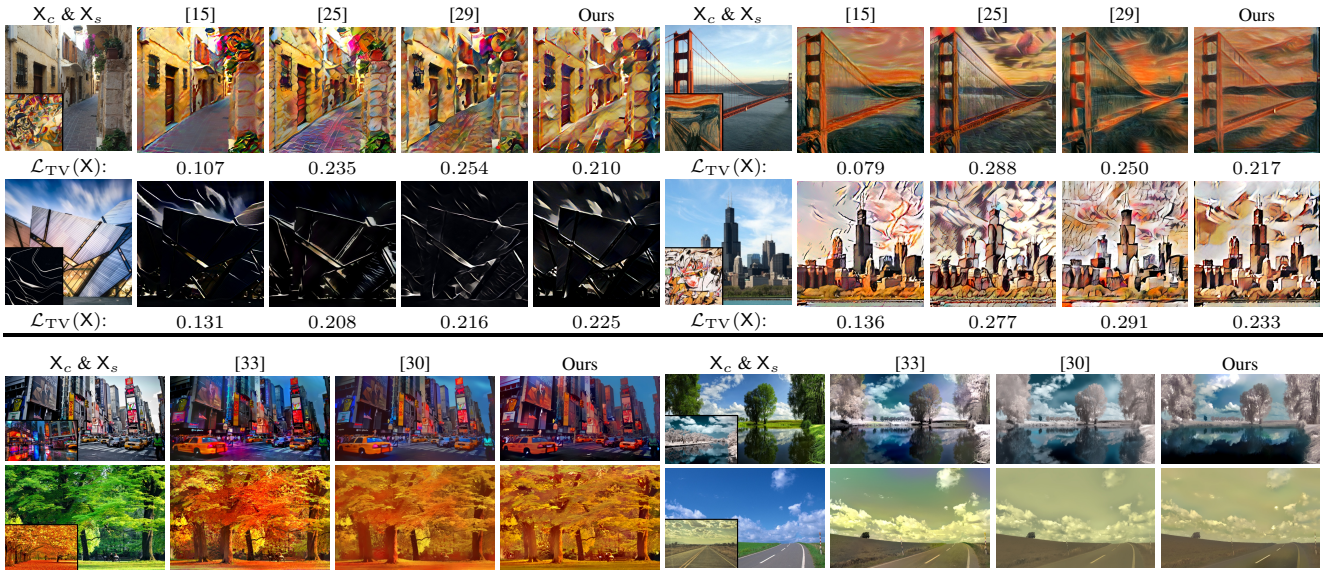

Figure 5. Top: Artistic style transfer results obtained with [15, 25, 29] and our method. The achieved loss $\mathcal{L}_{\mathrm{TV}}(\mathsf{X})$ is reported below each image. Bottom: Photorealistic style transfer results obtained with [33, 30], and our method.

the final result. New β^ℓ are drawn at each iteration.

5.2. Study of the architecture

We present in Fig. 3 artistic style transfer results for 7 different networks trained during 30 epochs: $N = 1, 2, 3, 4$ with $L = 64$ and $L = 16, 32, 64$ with $N = 3$. We remark that the stylisation is visually improved when both N and L increase. The mean loss achieved over 1000 pairs of content and style images taken from the 2014 MS-COCO validation set also improves when these parameters increase. Finally, when using classical ReLU-decoders as backward maps, we remark in Fig. 3 ellipsoidal structures and white artifacts overlaid on the results. We hypothesise that the pointwise multiplication with $h(\cdot)$ forces the network to better preserve the contours and edges of the content image, avoiding the presence of these parasite structures. These artefacts appeared when training the network for arbitrary styles but were absent when using few fixed styles.

For all remaining experiments, we fix $N = 4$, $L = 64$, and train this network for 50 epochs.

5.3. Fast artistic style transfer

To illustrate that our network is, at least, as flexible as existing state-of-the art solutions, we present in Fig. 4 results where we control, at runtime, the number of style scales, the global style intensity, and where we mix different styles.

We present in Fig. 5 artistic style transfer results obtained with our network and compare them with those of [15, 25, 29]. We use $\beta^\ell = 1$ for all ℓ. For [15], we minimise $\mathcal{L}_{\mathrm{TV}}(\mathsf{X})$ using the L-BFGS algorithm initialised with X_c, as our network is trained to minimise this loss starting from this image. Qualitatively, our results are similar to those of state-of-the-art methods. Quantitatively, the method of [15] reaches the lowest value for $\mathcal{L}_{\mathrm{TV}}(\mathsf{X})$ and all the fast methods obtain similar values. The second best value is, most of the times, obtained by our network but this is expected as it is trained to minimise $\mathcal{L}_{\mathrm{TV}}$ (while the others are not).

5.4. Fast photo style transfer.

Photo style transfer results obtained with [30, 33] and our method are presented in Fig. 5. We use $\beta^\ell = 1$ for all ℓ and $\lambda_{\mathrm{L}} = 50$. We achieve stylisations of similar visual quality as the alternative methods. Nevertheless, no methods are exempt of artifacts. The color in our results are sometimes "flattened", as also noticeable in [30]. This is due to the matting Laplacian filtering: because of it, neighboring regions that are similar in the content image and that get stylized slightly differently, are finally averaged. All methods would benefit from a better filter/regularisation.

The mean computation times measured on an Nvidia Tesla P100 Pascal GPU are reported below. Our method method is nearly twice faster than the one of [30] at all resolutions (columns "With Laplacian"). This advantage is mainly due to the graph filtering technique. After removing any processing involving the matting Laplacian in both methods (hence removing the photorealistic prior), our method is still the fastest except at the highest resolution:

	With Laplacian		No Laplacian	
Resolution	[30]	Ours	[30]	Ours
256×128	0.79	**0.44**	0.20	**0.04**
512×256	3.12	**1.86**	0.23	**0.11**
1024×512	15.23	**8.11**	**0.34**	0.37

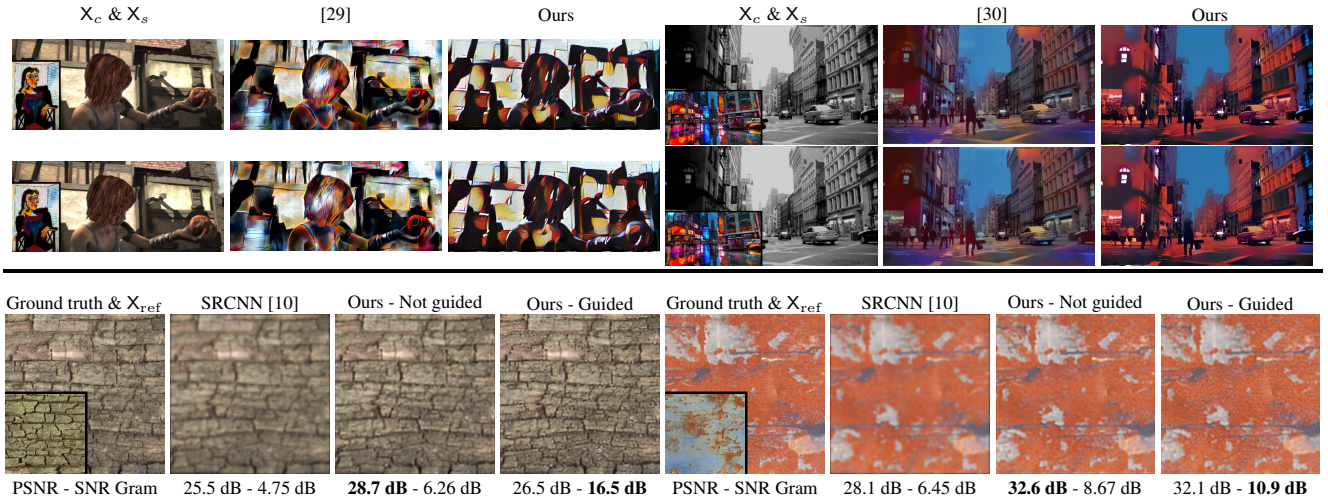

Figure 6. Top: Frames of two video sequences stylised with [29] or [30], and with our method using temporal regularisation. The style appears in the bottom left corner of original frames. Bottom: Informed texture super-resolution ($\times 3$ in both directions) using a user provided texture X_{ref} (bottom left in ground truth image). The PSNR between the reconstructed and ground truth images, and the SNR between the VGG-19 Gram matrices of the same images are provided below each result. Results are better viewed when zooming in.

5.5. Video style transfer.

We present in Fig. 6 video style transfer results. We compare our method, which includes short-term consistency, with those of [29] for artistic style transfer and [30] for photo style transfer. We use a video from the MPI Sintel dataset [4] for artistic style transfer. To enforce both photorealism and short-term consistency, we compose the corresponding proximal operators for simplicity. The optical flows between images are pre-computed using DeepFlow [50] but any fast flow estimation method could be used. We use $\lambda_{\mathcal{R}} = 0.37$. The videos are provided in the supplementary material.

We remark flickering artefacts in the videos obtained with [29, 30]. This flickering is reduced with our method thanks to the short-term consistency regularisation. One can argue that it is also possible to apply the temporal consistency proximal operator on the results of [29, 30]. This indeed attenuates flickering. Nevertheless, the videos provided in the supplementary material show that our results present less flickering than the post-processed results of [29, 30]. In Fig. 6, the lack of temporal consistency is better noticed on the hair of the main character in the video obtained with [29], and in the sky in the video obtained with [30]. Finally, we acknowledge that the stylisation in itself is quite different for the methods of [29, 30] and ours (even without using the temporal consistency proximal operator).

5.6. User-guided texture super-resolution

We present in Fig. 6 user-guided super-resolution results of textures. The downsampling operator D consists of a Gaussian filter of size 17×17 with $\sigma = 3$, followed by a downsampling with stride 3. Note that we only process the luminance. Colors are added back for visualisation only. We present results obtained using SRCNN [10], with our method using $\beta^\ell = 0$ for all ℓ – hence not user-guided – and with $\beta^\ell = 1$ for $\ell = \text{conv}_{1\_1}, \text{conv}_{2\_1}, \text{conv}_{3\_1}, \text{conv}_{4\_1}$, $\beta^{\text{conv}_{5\_1}} = 0$ for the complete proposed method. We set $\lambda_{\mathcal{R}} = 50$. We measure the PSNR between the reconstructed and ground truth images and the SNR between the VGG-19 Gram matrices of the same images (as these matrices are similar for similar textures [14]).

Our method performs better than SRCNN. This is probably explained by the fact that SRCNN is specifically trained for bicubic downsampling, whereas we use a strided Gaussian filtering. Concerning our results, similar PSNRs are reached with and without user inputs but the SNRs between the Gram matrices are improved with the user inputs. We hence obtain a better reconstruction of the overall texture, even though not pixel-accurate. Visually, our method with user guidance permits to enhance high-frequencies.

6. Conclusion

We proposed a new deep, fully convolutional network for fast artistic style transfer which has a key advantage: it can be restructured at runtime to incorporate important modifications of the artistic style transfer loss. Thanks to this property, it is possible to perform photo style transfer faster than state-of-the-art methods, video style transfer without suffering from flickering artefacts, and user-guided super-resolution, all *without* retraining. We provide additional results and discuss some limitations of the technique in the supplementary material.

References

[1] Heddy Attouch, Jérôme Bolte, and Benar F. Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized Gauss–Seidel methods. *Math. Program.*, 137(1-2):91–129, 2013.

[2] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. on Imaging Sci.*, 2(1):183–202, 2009.

[3] Mark Borgerding, Philip Schniter, and Sundeep Rangan. AMP-inspired deep networks for sparse linear inverse problems. *IEEE Trans. on Signal Processing*, 65(16):4293–4308, 2017.

[4] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision (ECCV)*, pages 611–625, 2012.

[5] Dongdong Chen, Jing Liao, Lu Yuan, Nenghai Yu, and Gang Hua. Coherent online video style transfer. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1114–1123, 2017.

[6] Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua. Stylebank: An explicit representation for neural image style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2770–2779, 2017.

[7] Yunjin Chen and Thomas Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1256–1272, 2017.

[8] Patrick L. Combettes and Jean-Christophe Pesquet. *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, chapter Proximal Splitting Methods in Signal Processing, pages 185–212. Springer Optimization and Its Applications. Springer New York, 2011.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[10] Chao Dong, Chen Change Loy, He Kaiming, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision (ECCV)*, pages 184–199, 2014.

[11] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *International Conference on Learning Representations*, 2017.

[12] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 341–346, 2001.

[13] Oriel Frigo, Neus Sabater, Julie Delon, and Pierre Hellier. Split and match: Example-based adaptive patch sampling for unsupervised style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 553–561, 2016.

[14] Leon Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 262–270, 2015.

[15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[16] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3730–3738, 2017.

[17] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. In *British Machine Vision Conference (BMVC)*, 2017.

[18] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256, 2010.

[19] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on International Conference on Machine Learning (ICML)*, pages 399–406, 2010.

[20] Agrim Gupta, Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Characterizing and improving stability in neural style transfer. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4087–4096, 2017.

[21] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Appl. Comput. Harmon. Anal.*, 30(2):129–150, 2011.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[23] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 327–340, 2001.

[24] Haozhi Huang, Hao Wang, Wenhan Luo, Lin Ma, Wenhao Jiang, Xiaolong Zhu, Zhifeng Li, and Wei Liu. Real-time neural style transfer for videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7044–7052, 2017.

[25] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1510–1519, 2017.

[26] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision (ECCV)*, pages 694–711, 2016.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[28] Anat Levin, Dani Lischinski, and Yair Weiss. A closed-form solution to natural image matting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2):228–242, 2008.

[29] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems (NIPS)*, pages 386–396, 2017.

[30] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. *European Conference on Computer Vision (ECCV)*, pages 468–483, 2018.

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.

[32] Risheng Liu, Xin Fan, Shichao Cheng, Xiangyu Wang, and Zhongxuan Luo. Proximal alternating direction network: A globally converged deep unrolling framework. *arXiv:1711.07653*, 2017.

[33] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6997–7005, 2017.

[34] Morteza Mardani, Qingyun Sun, David Donoho, Vardan Papyan, Hatef Monajemi, Shreyas Vasanawala, and John Pauly. Neural proximal gradient descent for compressive imaging. *arXiv:1806.03963*, 2018.

[35] Tim Meinhardt, Michael Moller, Caner Hazirbas, and Daniel Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1799–1808, 2017.

[36] Chris Metzler, Ali Mousavi, and Richard Baraniuk. Learned d-amp: Principled neural network based compressive image recovery. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1772–1783, 2017.

[37] Ali Mousavi and Richard G. Baraniuk. Learning to invert: Signal recovery via deep convolutional networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2272–2276, 2017.

[38] Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer graphics and applications*, 21(5):34–41, 2001.

[39] J. H. Rick Chang, Chun-Liang Li, Barnabas Poczos, B. V. K. Vijaya Kumar, and Aswin C. Sankaranarayanan. One network to solve them all — solving linear inverse problems using deep projection models. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5889–5898, 2017.

[40] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In *German Conference on Pattern Recognition*, pages 26–36, 2016.

[41] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos and spherical images. *International Journal of Computer Vision*, 126(11):1199–1219, 2018.

[42] Shunsuke Saito, Lingyu Wei, Liwen Hu, Koki Nagano, and Hao Li. Photorealistic facial texture inference using deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2326–2335, 2017.

[43] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

[45] Ayush Tewari, Michael Zollhöfer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Pérez, and Christian Theobalt. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. In *IEEE International Conference on Computer Vision (ICCV)*, pages 3735–3744, 2017.

[46] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating Eulerian fluid simulation with convolutional networks. *International Conference on Machine Learning*, 70:3424–3433, 2017.

[47] Nicolas Tremblay, Gilles Puy, Rémi Gribonval, and Pierre Vandergheynst. Compressive spectral clustering. In *International Conference on Machine Learning (ICML)*, volume 48, pages 1002–1011, 2016.

[48] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *International Conference on Machine Learning (ICML)*, volume 48, pages 1349–1357, 2016.

[49] Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Proximal deep structured models. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[50] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1385–1392, 2013.

[51] Dong Yang and Jian Sun. Proximal dehaze-net: A prior learning-based deep network for single image dehazing. In *European Conference on Computer Vision (ECCV)*, pages 702–717, 2018.

[52] Jian Zhang and Bernard Ghanem. ISTA-Net: Interpretable optimization-inspired deep network for image compressive sensing. *IEEE International Conference on Computer Vision (CVPR)*, 2018.