Published in Image Processing On Line on 2022–09–05.
Submitted on 2022–04–29, accepted on 2022–08–09.
ISSN 2105–1232 © 2022 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2022.403

DeOldify: A Review and Implementation of an Automatic Colorization Method

Antoine Salmona[1], Lucía Bouza[1], Julie Delon[1,2]

[1] Université Paris Cité, CNRS, MAP5, France
[2] Institut Universitaire de France (IUF)
(antoine.salmona@u-paris.fr, lucia.bouza-heguerte@u-paris.fr, julie.delon@u-paris.fr)

Communicated by Gregory Randall *Demo edited by* Lucía Bouza

Abstract

DeOldify is a recent automatic colorization method based on Convolutional Neural Networks which yields impressive results. The method was initially created by Jason Antic with the support of the Californian start-up Fast.ai and thus does not come from the academic research world. The goal of this paper is twofold. First we propose a rigorous mathematical presentation of the method along with a critical analysis of its different steps. Second, we provide an open-source implementation of a simplified but effective version of the approach, based on Pytorch and without dependence on the Fast.ai framework.

Source Code

The reviewed source code and documentation for this implementation are available from the web page of this article[1]. Compilation and usage instruction are included in the README.md file of the archive.

Supplementary Material

A video colorized by the network is provided in the web page of the article.

Keywords: colorization; transfer learning; neural networks; CNN

[1]https://doi.org/10.5201/ipol.2022.403

1 Introduction

1.1 The Problem of Automatic Colorization

For a given image of N pixels, automatic colorization can be naively seen as a regression problem from $\mathbb{R}^N$ to $\mathbb{R}^{N\times 3}$. Indeed, it consists of predicting the three components Red, Green, Blue using only one single component of luminance. Yet, the problem can be easily simplified to a regression from $\mathbb{R}^N$ to $\mathbb{R}^{N\times 2}$ by working in a luminance-chrominance color space: such spaces are designed to extract separately, on the one hand, the luminance, that is, all the information contained in the gray-scale image, and on the other hand the chrominance, i.e. all the information associated to the color in itself. The luminance is encoded with the first component of the space and the chrominance with the other two components.Thus, working in such space allows to directly use the gray-scale image as the first component of the resulting color image.

There exist several different luminance-chrominance color spaces. Among them, the CIELAB color space has been recently one of the most commonly used for image colorization [5, 25]. Indeed, this space has the interesting property of modeling well the human perception of distance between colors. Thus, uniform changes of components in this space correspond approximately to uniform changes in perceived color. Yet, this space suffers from a major drawback, the mapping between the RGB space and the CIELAB space is highly non-linear. This implies that the RGB gamut in the CIELAB space, i.e. the mapping between the RGB to the CIELAB space is hard to compute. Besides, due to the inherent design of the space, the gamut strongly depends on the luminance in a non-linear fashion. As an outcome, one can easily predict points outside of the gamut, which do not correspond to any existing color. For this reason, it is preferable to work in another luminance-chrominance color space, like the YUV color space. Indeed, the mapping between the RGB color space and the YUV color space is linear and so the gamut can be easily derived.

The colorization problem can be described as follows: let u be a gray-scale image of size N. We can directly derive the first component in the YUV space, $u_Y = u$. We aim to learn a transformation $\mathcal{F} : \mathbb{R}^N \longrightarrow \mathbb{R}^{2\times N}$ such as

$$(\hat{u_U}, \hat{u_V}) = \mathcal{F}(u_Y)\ , \tag{1}$$

where $\hat{u_U}$ and $\hat{u_V}$ are the estimated second and third components in the YUV space. We can see that the under-determined structure of this problem leads to an infinity of solutions, which models well the ambiguous feature of the automatic colorization problem. Our goal is to pick a realistic solution among all possible ones.

1.2 Related Work

The first automatic colorization methods date from the early 2000s. At the time, two types of approaches were distinguished: the first kind, initially proposed by Levin et al. [14], consists in propagating colored user-specific scribbles to the whole image using geometrical information (colors are propagated in smooth regions but not across edges). The second type, as in Welsh et al. [29], consists in seeing the colorization problem as a particular case of the more general color transfer problem [18]. Thus, gray-scale images are colorized using reference color images by cutting both target and reference in small patches and by computing statistics to compare and match similar patches.

With the outburst of deep learning in computer vision, several methods using end-to-end Convolutional Neural Networks (CNN) have been proposed. The idea is to train a network to understand the semantic of a given scene and then to deduce the right colors associated to those semantic priors. Of course, the semantic information is not always sufficient to recover the exact ground truth color: for instance, a car in a gray-scale image could actually be gray as well as it could be red or even almost any possible color. Therefore, certain methods, like [3, 4] are returning probability distributions over the set of the possible colors in order to model this ambiguous feature of colorization using semantic information.

Colorization using semantic priors is often linked with other typical problems of computer vision. In particular, Iizuka et al. [8] propose a network trained not only to achieve colorization but also for performing rudimentary classification. Yet, the colorization task is done by simple ℓ_2 regression on a luminance-chrominance space, which does not model at all the ambiguous feature of the problem, and the classification task is kept elementary for constraints in computing capacity. More recently, Vitoria et al. [27] propose an adversarial framework where a generative network is trained jointly to achieve colorization and classification.

On the other hand, Zhang et al. [31] and Larsson et al. [13] propose an approach of colorization related to segmentation, classification at the scale of the pixels. The classes embody the different possible colors and then a mapping from the class to its corresponding color is proposed. Such an approach provides a better modeling of the ambiguous characteristic of the problem than a ℓ_2 regression. In order to go one step further, Su et al. [25] propose an instance-aware image colorization, directly built upon an object detector. The authors use different networks to extract features from the cropped objects and from the whole image. These features are used by a fusion module to predict the final colors. In [12], Kimar et al. propose the Colorization Transformer, an approach for diverse image colorization based on self-attention. In [21], Saharia et al. achieve colorization (as well as inpainting, uncropping, and JPEG decompression) by image-to-image translation using conditional diffusion models.

DeOldify [1][2] builds up on the idea of linking colorization with classification but in a context of transfer learning. The method uses a pretrained ResNet [7] as a backbone for the architecture of its network and the loss function used for optimization during training involves the intermediary features maps of a pretrained VGG network [23]. This loss function is inspired by the feature reconstruction loss used in style transfer and defined in Johnson et al. [10]. The whole network is a U-Net [19], which is a classical architecture for segmentation problems. The method achieves strikingly better results than [31]. More surprisingly, the method yields also very good results on videos in terms of temporal consistency while simply proceeding frame by frame without adding any temporal stabilization process.

2 DeOldify: An Overview of the Original Method

DeOldify [1] is a recent open-source fully automatic colorization method created by Jason Antic with the support of the Californian start-up Fast.ai. It is an end-to-end CNN based method and yields impressive results on images and videos, despite the fact that frames are processed independently as simple images with no additional temporal stabilization process. Since this method does not come from the academic research world, it is not presented in any paper, and only code and a blog with brief explanations are available[3]. In this section, we propose a rigorous presentation of the original method and its different steps. Then, in the next section, we provide a critical analysis of the different steps that highlights some defects in the original method and we give details about our implementation of the accordingly modified method.

Versions. The original method is presented in three different versions:

- “artistic”: focused on colorful colorizations rather than spatial stability

[2]More information about the original project is available here: https://github.com/jantic/DeOldify.

[3]the blog is available here: https://www.fast.ai/2019/05/03/decrappify/

- "video": focused on spatial and temporal stability
- "stable": focused only on spatial stability and yields a bit more colorful colorizations than the "video" version but less than the "artistic" version.

Those versions correspond to slight differences in architecture and in training procedures. In this work, unless otherwise specified, we focus on the "video" version.

2.1 Network Architecture

The colorizer network is a U-Net [19], illustrated in Figure 1, constructed from a pretrained ResNet [7]: the latter corresponds to the encoding part and a decoding part is added in order to reconstruct the color image from the abstract representation derived by the ResNet. Moreover, the outputs of the intermediate layers of the ResNet are directly used and interpolated with the outputs of the intermediate layers of the decoder (skip connections). The major interest of the U-Net architecture is its property of mixing features from different scales, both during coding and decoding.

The network architecture is exactly the same in the "video" and the "stable" version, and the version of ResNet used is ResNet101. The "artistic" version uses ResNet34 and the depths of the layers in the dilating part are doubled. The reason for using ResNet34 over ResNet101 is simply related to computational capacity limitations.

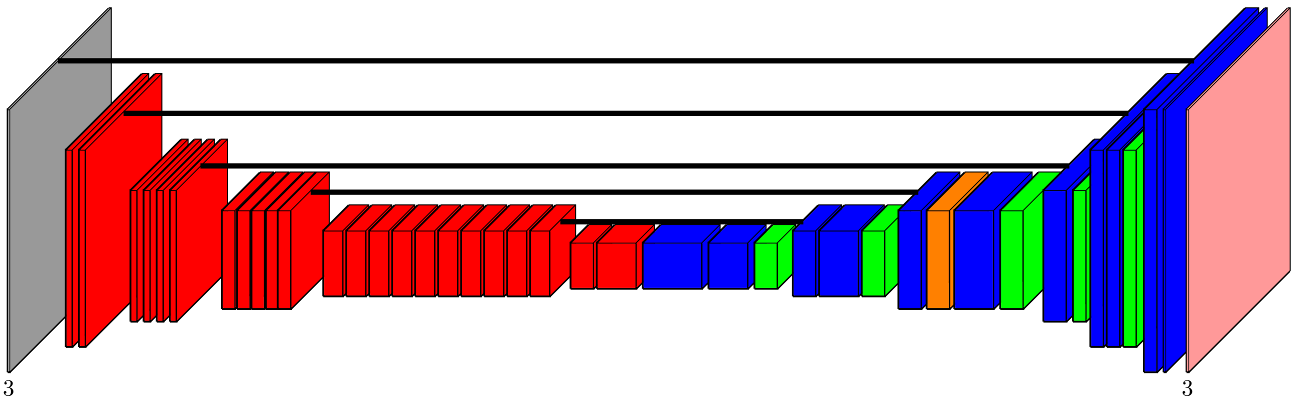


Figure 1: Architecture of DeOldify. In red: pretrained ResNet, in blue: convolutional blocks, in green: upsample layers, in orange: self-attention layer, and in rose: sigmoid layer. The black lines stand for the skip connections.

Network input. Although the method considers the luminance component as input, the network input is a gray-scale image encoded in RGB, i.e. a three channel tensor with the same values for each component. This comes from the fact that the pretrained ResNet needs images encoded in RGB as inputs. Then, the input images are normalized according to the training data statistics: for each channel R, G, and B, the values of the mean and the standard deviation are computed over all the training images, and the input images are then transformed accordingly so that each of their channels follows a reduced and centered distribution. Mathematically speaking, it corresponds to apply to the data the following linear transformation

$$f(x) = A(x-b) \quad \text{with} \quad A = \operatorname{diag}(\sigma_R, \sigma_G, \sigma_B)^{-1} \quad \text{and} \quad b = (\mu_R, \mu_G, \mu_B)^T \ . \tag{2}$$

Since the ResNet was trained with the Imagenet dataset [20], the values of A and b come from empirical statistics on Imagenet. When the network input is a gray-scale image encoded in RGB, since we apply different corrections (mean and standard deviation) to each channel, the image is not gray-scale anymore after normalization.

This pre-processing step is quite common for machine learning methods and allows to have training data with similar statistical properties. Furthermore, the exact same normalization process has been already applied when training the ResNet network so it is important to normalize data in the same way in order to get the most of the pretrained ResNet network. Following the same logic, the training images are reshaped to be square in order to be similar to the images used during the training of the ResNet.

2.2 Network Layers

ResNet. The ResNet architecture is kept unchanged and the weights are frozen. The fully connected layers are not kept, we retrieve directly the output of the last convolutional block.

Convolutional blocks. All the convolutions have a kernel of size 3×3 and are almost always followed by batch normalization [9] and ReLU activation. In accordance with [9], convolutions are encoded without biases, as the use of batch normalization cancels them out. When enabled, spectral normalization [16] is applied to the parameters of all convolutions in the network, but the convolution of the last Upsample layer. The spectral normalization was presented in [16] in order to stabilize the training of discriminators in Generative Adversarial Networks (GANs), but the benefits of the technique are not limited to this particular problem as shown in [6]. The stabilization aims to achieve Lipschitz continuity of the network with the smallest possible value of the Lipschitz constant. One way to achieve this is to ensure that each layer of the network is Lipschitz continuous with the smallest possible Lipschitz constant in each case. As shown in [16], $\text{Lip}(\ell) = \sigma(W^\ell)$, where $\text{Lip}(\ell)$ is the Lipschitz constant of layer ℓ and $\sigma(W^\ell)$ is the spectral norm of the weights W of the layer ℓ. By rescaling the layer weights, we get that $\text{Lip}(\ell) = \sigma(W^\ell) = 1$.

Upsample layers. For the up-sampling we used the sub-pixel convolution defined in [22]. This operation consists in increasing the size of the image by a factor r using a convolution with a non-integer stride of $\frac{1}{r}$. A naive implementation of such an operation consists in padding the low resolution image by adding zero-valued pixels between each pixel of the original image and then realizing a convolution with stride 1 (Figure 2).

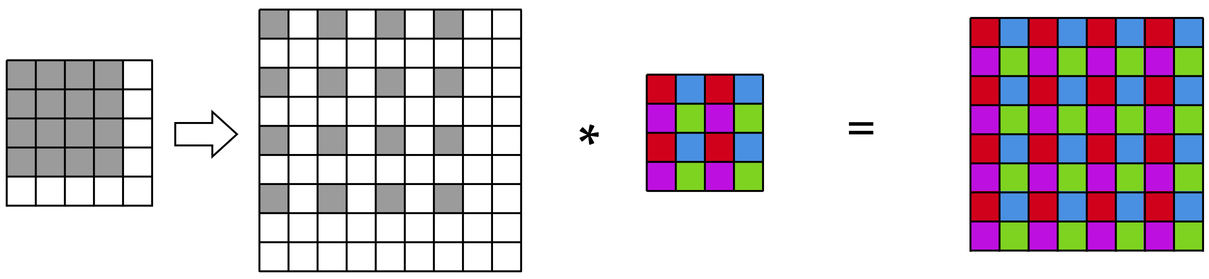


Figure 2: The naive implementation of the sub-pixel convolution: in grey the original pixels and in white the zero-padded pixels. The color matrix represents the convolution kernel.

Yet this implementation is inefficient in terms of computational cost because the convolution is realized in the sub-pixel space, i.e. on the padded image. To overcome this problem, [22] proposes to realize directly the convolution in the low resolution space with r^2 channels and then to retrieve the target image by using a periodic shuffling operator that rearranges the elements of a $H \times W \times r^2C$ tensor to a tensor of shape $rH \times rW \times C$ (Figure 3).

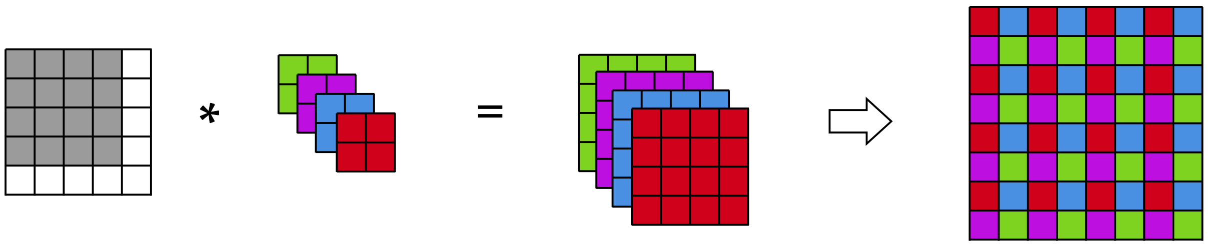


Figure 3: The efficient implementation of the sub-pixel convolution.

In addition, this operation can introduce checkerboard artifacts. To face this problem, [26] proposes to introduce a Gaussian blur with a kernel of size $r \times r$ after the periodic shuffling operator. The experiments conducted in [26] show that if this blur is introduced during training, the network will learn to reconstruct the high resolution target image without checkerboard artifacts.

Self-Attention. This layer has been introduced in [30] using a model defined in [28]. The idea is to introduce a new generic non-local operation in deep neural networks inspired by the Non-local Means algorithm defined in [2]. Indeed, the new operation can be written

$$o_i = \frac{1}{\mathcal{C}(x)} \sum_{j}^{N} \phi(x_i, x_j) h(x_j) \ , \tag{3}$$

where $o_i \in \mathbb{R}^C$ is the output position to be computed ($o \in \mathbb{R}^{C\times N}$), $x \in \mathbb{R}^{C\times N}$ is the input, $\mathcal{C}$ is a normalization factor, N is the number of feature locations in x and C is the number of channels of x. The integer $i \in [0, N-1]$ is the index of the output position whose response is to be computed.

The pairwise function ϕ computes a scalar, representing a relationship (such as an affinity) between x_i and x_j. The unary function h computes a representation of the input signal at position j. The Non-local Means algorithm is a particular case of the previous formulation, where the image is cut in small patches and the operation becomes

$$o_i = \frac{1}{\mathcal{C}(x)} \sum_{j} e^{-\frac{\|P(x_i)-P(x_j)\|^2}{\sigma^2}} x_j \ , \tag{4}$$

where $P(x_i)$ and $P(x_j)$ are respectively the patches centered at i and j and σ is a filter parameter. In this formulation, ϕ acts as a discriminator of all possible positions x_j, giving weights exponentially smaller to positions associated with patches that are far (in terms of similarity) from the patch associated to the position x_i.

In [28], the authors propose to make the network learn the functions ϕ and h and suggest several structures for ϕ: $\phi(x_i, x_j) = e^{x_i^T x_j}$ (Gaussian), $\phi(x_i, x_j) = e^{f(x_i)^T g(x_j)}$ (embedded Gaussian), $\phi(x_i, x_j) = f(x_i)^T g(x_j)$ (dot product), and $\phi(x_i, x_j) = \mathrm{ReLU}(w_f^T[f(x_i), g(x_j)])$ (concatenation), where f and g are applications from $\mathbb{R}^{C\times N}$ to $\mathbb{R}^{\bar{C}\times N}$ with C the number of input channels, N the number of feature locations from the input, and $\bar{C}$ the number of channels of the output. The experiments they conducted show that the choice among those structures has a minor impact on performance. For implementation convenience, the embedded Gaussian structure is chosen and for a matter of computational cost, f, g and h are restricted to linear applications. The number of output channels is chosen as $\bar{C} = C/8$.

The self-attention layer can be easily implemented with classical tools of deep learning, writing $f(x) = W_f x$, $g(x) = W_g x$ and $h(x) = W_h x$ where $W_f \in \mathbf{R}^{\bar{C}\times C}$, $W_g \in \mathbf{R}^{\bar{C}\times C}$ and $W_h \in \mathbf{R}^{\bar{C}\times C}$ are learned weight matrices. In practice they are implemented as 2D convolutions for the sake of simplicity.

One can observe that with the embedded Gaussian structures, $\frac{1}{C(x)}\phi(x_i, y_j)$ becomes the *Softmax* computation along the dimension j, and so (3) becomes

$$o = \text{Softmax}((W_f x)^T W_g x) h(x), \tag{5}$$

where $\text{Softmax}(z)_i := \frac{e^{z_i}}{\sum e^{z_j}}$. The computational graph for the self-attention layer is schematized in Figure 4.

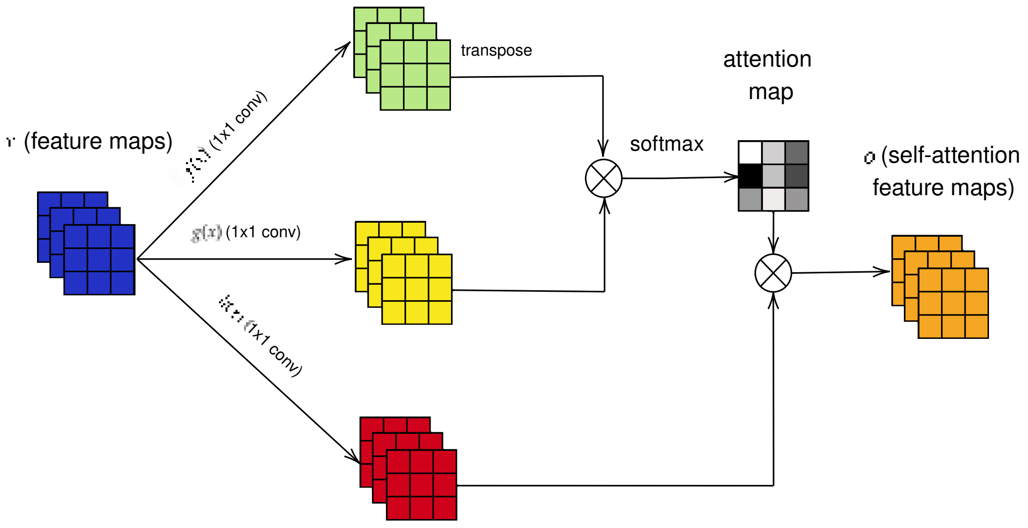


Figure 4: The computational graph for the self-attention layer. $\bigotimes$ stands for matrix product. The *Softmax* is applied on each row.

Furthermore, the layer is residual: if x is the input and o the output, the input of the next layer will be $y = \gamma o + x$ (where γ is a scalar to be learned). This choice is mainly due to practical reasons: it allows to introduce self-attention layers in an already trained network without the necessity of retraining the whole network.

Lastly, the uniqueness of the self-attention layer and its position in the network architecture is related to a matter of computational capacity. Yet, the experiments conducted in [28] have shown that the use of more than one layer of self-attention increases the global performances.

Network output. The network outputs the three components $\overline{\text{R}}$, $\overline{\text{G}}$ and $\overline{\text{B}}$ of the normalized color image. Thus the output values must be in the normalized RGB space, i.e.

$$\overline{\text{RGB}} = \left[-\frac{\mu_R}{\sigma_R}, \frac{1-\mu_R}{\sigma_R}\right] \times \left[-\frac{\mu_G}{\sigma_G}, \frac{1-\mu_G}{\sigma_G}\right] \times \left[-\frac{\mu_B}{\sigma_B}, \frac{1-\mu_B}{\sigma_B}\right]. \tag{6}$$

To do so, the sigmoid operator is used to rescale the values to the target intervals. In practice, the intervals are chosen larger than their theoretical range to avoid gradient vanishing during training. The range used in the implementation is $[-3, 3]$.

Post-processing. A first post-processing step during inference is to apply the inverse transformation of (2). Then, the predicted color image is converted to YUV and only the chrominance components U and V are kept. Then those components are concatenated with the luminance component Y derived from the original gray-scale image and the obtained image is converted back to RGB. This step allows to have colorized images with exactly the same luminance component as the input images.

2.3 Training Details

The network is trained on Imagenet[4] in a supervised fashion. The training dataset is composed of pairs of input-output images $\left\{\left(U_j^{gs}, U_j^{rgb}\right)\right\}_{j=0}^{m}$ where $\left\{U_j^{rgb}\right\}_{j=0}^{m}$ are the ground truth images in RGB and $\left\{U_j^{gs}\right\}_{j=0}^{m}$ are the corresponding gray-scale images converted back to RGB (with the three components having the exact same values). In the original implementation, the training is done in two distinct phases. First, the network is trained in a "conventional" fashion, in the sense that the training involves the optimization of a loss function designed specifically for the problem. Then, a binary classifier is trained to recognize whether a given image has been colorized by the previous network or is a ground truth image and then a GAN-type training is involved using the binary classifier as Discriminator and the Colorizer Network as Generator. Our implementation drops the second phase because we observed this latter could be replaced by a much simpler post-processing process achieving the same quality of result. Still, we describe in the following the steps of the original code.

2.3.1 Conventional Training

Loss function. The loss function used for training is inspired by the Feature Reconstruction Loss defined in Johnson et al. [10] which uses a VGG network. Let ϕ be a pretrained VGG network. We note $\phi_j(x)$ the output feature maps of the layer j when x is the input image. Let note J the set of indices corresponding to the layers preceding the max-pooling layers. As it is shown in Figure 5, this set is non empty and contains exactly 5 elements. The loss function is then written

$$\ell(x, y) = \lambda_0 \|x - y\|_1 + \sum_{j \in J} \lambda_j \|\phi_j(x) - \phi_j(y)\|_1 \;, \tag{7}$$

where the λ_j are weights chosen empirically. As said before, this type of loss function has been introduced in [10] in the context of style transfer. For a given image y , the image $\hat{y}$ that minimizes $\|\phi_j(x) - \phi_j(y)\|$ for a given j does not necessarily has to be the exact image y itself: it can be an image with the same overall structure and content, but where the color, texture and the exact shape are not preserved. The more j increases, the more $\hat{y}$ can be perceptually different from y. In practice, for the two first layers, $\hat{y}$ must still be perceptually very close to y so those layers are not kept during the actual implementation of (7) so as not to add useless computational costs during training: indeed the minimizers of $\|\phi_1(x) - \phi_1(y)\|$ and $\|\phi_2(x) - \phi_2(y)\|$ will be very close to the minimizer of $\|x - y\|$ so those terms can be simply omitted and replaced equivalently with a bigger λ_0 during the minimization of (7). It may seem confusing to introduce terms in the loss function which do not preserve color when the goal itself is to colorize the image. Yet, it should be kept in mind that the colorization task is done using semantic priors. Thus, although these terms do not contribute directly to the introduction of color itself, they can help to better identify those semantic priors. The choice of the ℓ_1 norm over the ℓ_2 norm as base distance is motivated by the fact that

[4]The dataset is available here: `http://www.image-net.org/`

the ℓ_1 norm is less sensitive to outliers, which is an appealing property when doing colorization: for instance, we do not want an image taken during sunset and showing a red sky to have too much impact in the final result when colorizing images with skies.

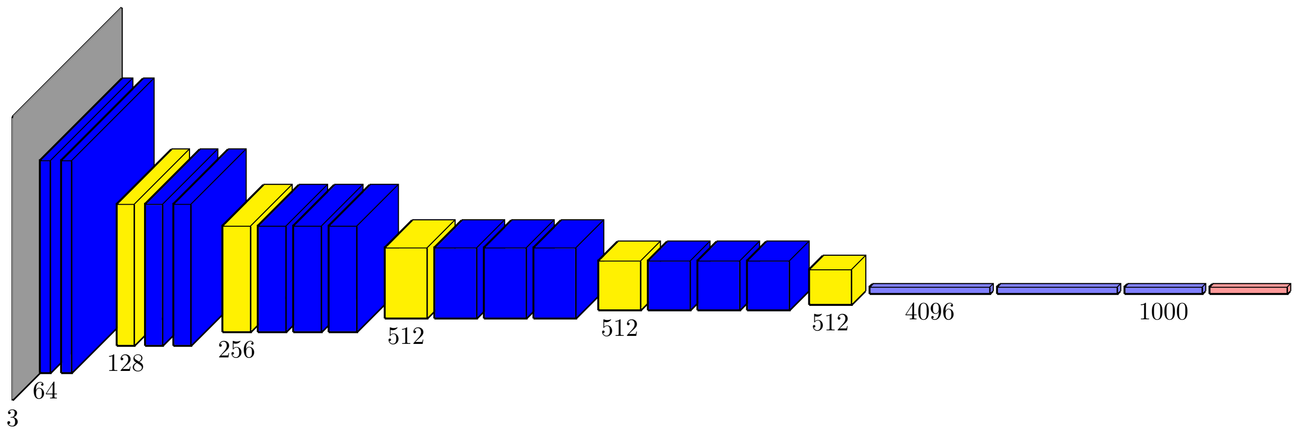


Figure 5: Architecture of the VGG-16 network. In blue: convolution + ReLU, in yellow: max-pooling layer, in purple: fully connected + ReLU, and in rose: softmax layer. The other VGG networks (VGG-11,VGG-13,VGG-19) keep the same global architecture, only the number of convolutions changes.

Progressive training. The training procedure is loosely inspired by [11]: during training, the image size is progressively increased. Indeed, the network is trained with four epochs of Imagenet. During the two first epochs, the training images are down-sampled to 64×64 pixels. During the third epoch, they are down-sampled only to 128×128 pixels, and 192×192 pixels for the last epoch. The underlying idea is that training first on small images will make the network learn large-scale image structures and as the resolution of the training images is increased, the network will learn smaller and smaller structures. This follows the fact that since the objective function is not convex, the final result depends heavily on the initialization. Thus, each epoch aims to give a good initialization to the following one. That way, the network is a bit less likely to converge to an unwanted local minimum. The choice of 192×192 as the larger image size used during training is simply related to computational capacity.

Training scheme. Training is done with the Adamw optimizer [15], a "corrected" version of the Adam optimizer for weight decay. To accelerate the training, the 1cycle learning rate policy and cyclical momentum is applied as presented in [24]. On this scheme, each epoch is divided into two phases. In the first phase, the learning rate increases linearly from the smallest to the largest defined value while the momentum decreases linearly over the defined range. In the second phase, the learning rate follows a cosine annealing from the largest defined value to zero, as the momentum increases over the defined range with the same annealing.

Temporal consistency. An additional training phase is involved in the "video" version where the network is trained again on 25% of Imagenet chosen randomly, but this time with noisy images. This part is directly inspired by [17] and aims to improve the temporal consistency.

In [17], the temporal consistency of a style transfer method is improved by training the network in a way that an image and its noisy version would give very close results. To do so, during training, given an input image x of size N, a noisy image $\hat{x}$ is constructed in the following way

$$\forall i \leq N, \; \hat{x}_i = x_i + \epsilon_i u_i, \tag{8}$$

where the $(\epsilon_i)_{i\leq N}$ are i.i.d and follow a Bernouilli distribution of parameter p and the $(u_i)_{i\leq N}$ are i.i.d and follow a Uniform distribution $\mathcal{U}([-a,a])$, and p and a are two hyperparameters to be determined empirically. Then, we note y the output of the network when x is the input and $\hat{y}$ the output when $\hat{x}$ is the input. The network is trained using the following loss:

$$\ell_{\text{noise}} = l(x,y) + \lambda_{\text{noise}}\|y-\hat{y}\|_2^2, \tag{9}$$

where $l(x,y)$ is the loss defined by (7) and λ_{noise} is an hyperparameter to be chosen empirically. The experiments of [17] show that the previous method is efficient to reduce the "popping" effect, which can be considered as the analog of the flickering effect in style transfer.

2.3.2 GAN Training: "NoGAN"

We do not discuss in detail this last phase since we observed it could be replaced by a much simpler post-processing step leading to the same quality of results (see next section). The training of type GAN is done in the following way: firstly, a binary classifier is trained in a supervised fashion to recognize whether an image has been colorized (by the network trained with the loss described above) or is natural. For that part, the loss function used is simply the binary cross entropy. Then, this binary classifier is used as discriminator for the GAN training.

Loss function. The classic GAN loss function is used for training

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}[\log(D(x))] + \mathbb{E}_{z\sim p_z}[\log(1-D(G(z))], \tag{10}$$

where G is the generator (here the colorizer network), D the discriminator (here the binary classifier previously trained). p_{data} is the true distribution of the data that we want to generate elements from and p_z is a Gaussian distribution (white noise). As the training goes, the discriminator becomes better at recognizing if a given image is real or has been colorized by the generator and the generator becomes better to lure the discriminator.

NoGAN. This GAN training is done in a quite peculiar way that the author of DeOldify called "NoGAN": during training, intermediary models are stored periodically. Then, each model is tested by colorizing a few images. The author of DeOldify observed that the model giving the most realistic results was found after training on only a very small portion of the whole dataset chosen randomly (between 2% and 5%). Afterwards, results oscillate between results equivalent to those obtained with the best model and bad over-saturated results (with typically orange skin), however there is no evident indication of such optimal spot when looking the evolution of the loss.

Training cycles. In the "artistic" and "stable" versions, the colorizer network is trained in cycle, alternating the two training phases described above. There are 5 cycles for the "artistic" version and 3 for the "stable" version. For the "video" version, the network is trained with only one occurrence of each phase.

3 Critical Analysis and Details of our Implementation

In this section, we provide a critical analysis of the DeOldify colorization method and we describe the details of our implementation, which is slightly different from the original one.

3.1 Architecture Review

Network output. A minor problem with the architecture of the original method is that it predicts the three components R, G and B, although the method uses only the components U and V in practice. In terms of computational cost and for a matter of concision, it is better to make the network learn to predict directly the components U and V rather than to predict the three RGB components and then convert the image to YUV and replace the predicted Y component with the original one. Since the transformation between the RGB space and the YUV space is linear, it is sufficient to only modify the number of output channels of the last convolutional layer. In practice, the network predicts the normalized components $\overline{\text{U}}$ and $\overline{\text{V}}$ from $\overline{\text{R}}$, $\overline{\text{G}}$, and $\overline{\text{B}}$, and the unnormalization process is applied after converting back the image to RGB. This choice can be explained by the fact that a pretrained ResNet and a pretrained VGG are used for training, and both need to have their input images in $\overline{\text{RGB}}$. Lastly, one should note that Equation (6) should be modified accordingly: if we denote by M the linear transformation between the RGB space and the YUV space, we should retrieve the last two components of $M(\overline{\text{RGB}})$, where $\overline{\text{RGB}}$ is the set defined in Equation (6).

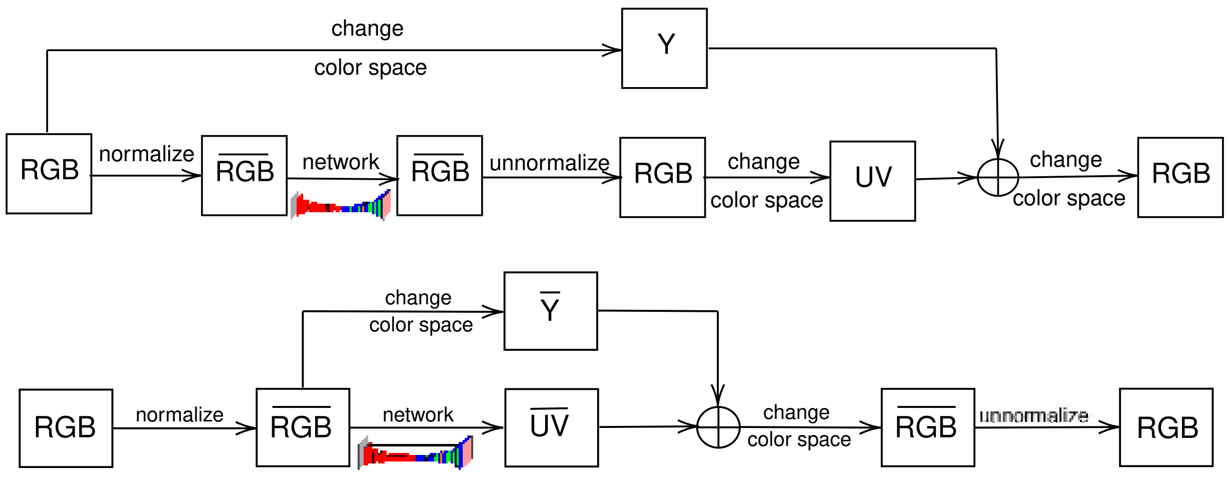


Figure 6: Top: the original method architecture. Bottom: our modified architecture. The symbol $\bigoplus$ stands for concatenation.

Self-Attention layer. The Self-Attention layer described above is residual. Thus, it can be added or removed without having to re-train the whole network. In order to measure the implication of this layer in the result, one can compare the results obtained with or without this layer, as illustrated by Figure 7. Intuitively, one would expect the results with the Self-Attention layer to be more spatially consistent at large scale than the results without. Indeed, as the Self-Attention layer is non-local, a pixel at any position can have an impact on the result of a pixel at any other position. We observe in practice that the Self-Attention separates better the different image zones in terms of color (for instance sky and earth, or faces and background), yielding more realistic results.

3.2 Training Review

Loss function. In order to better understand the role of the terms in Equation (7), we trained the network using only the ℓ_1 distance as loss function. The images colorized by the resulting network seem to contain globally the same colors as the images colorized with the network trained using (7), and are more colorful, but are strikingly less satisfying in terms of spatial consistency and contain more artifacts and uncolorized regions. Thus, during optimization of (7), it seems that the color

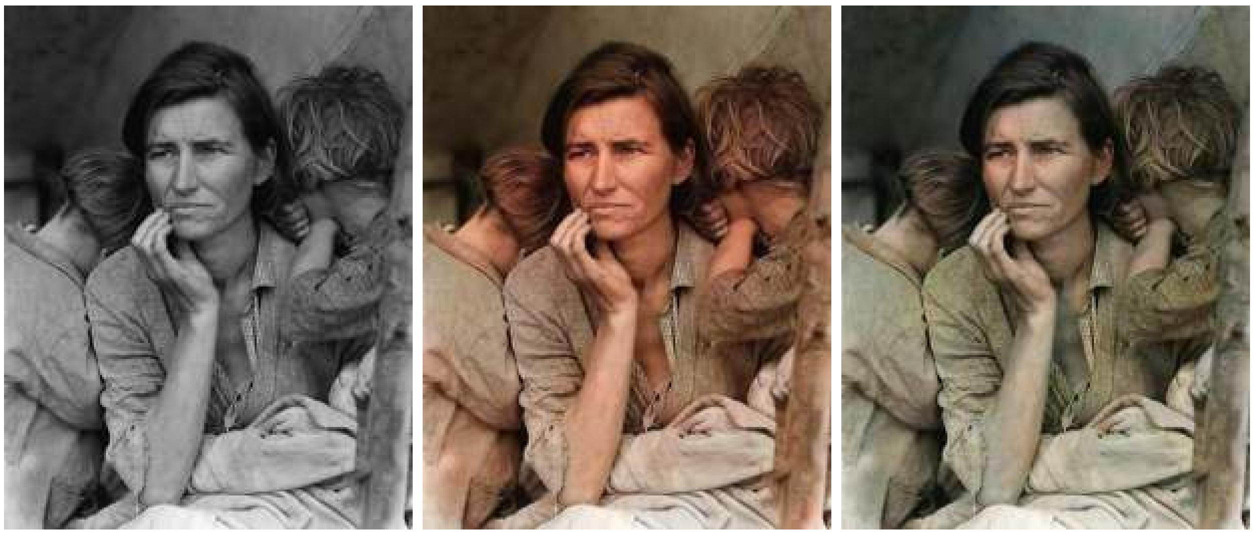


Figure 7: From left to right: gray-scale image, image colorized with the network using Self-Attention, image colorized without the Self-Attention layer. In this last image, some regions (arm, chest, forehead) are not colored correctly.

introduced mainly comes from the term $\|x - y\|_1$ but the other terms help the network to better classify the different elements of the images and so indirectly to achieve more realistic colorizations.

An important difference between (7) and the loss introduced in [10] is the use of more than one term of type $\|\phi_j(x) - \phi_j(y)\|$. Since it is difficult to anticipate what will be the consequences of summing over several intermediary feature maps of VGG, we retrained the network adding to the ℓ_1 loss only the term associated to the greatest λ_j. We observed that the results are most of the time perceptually very close to the results obtained by colorizing with the network trained with (7) but the model is a bit more prone to take risks. As an outcome, the results can seem better in terms of colorfulness, but are nevertheless a bit less stable.

Lastly, we trained the network using the ℓ_2 distance instead of ℓ_1 as base loss in all the terms of (7) in order to see the effect of the ℓ_1 loss rather than the standard mean square error. The results seems surprisingly as good as the ones obtained with the model trained with the ℓ_1 loss. It seems that using the ℓ_1 distance rather than the ℓ_2 distance does not specially increase the overall quality of the results.

Spectral normalization. Spectral normalization was added to all convolution layers except the convolution of the last upsample layer to stabilize network training. To better understand the effects of such normalization, we trained the network with and without spectral normalization. Spectral normalization provides more stable results. When we train without any normalization of weights on convolutions, false colors may appear on some images, even if the results are usually close to those obtained with normalization. This is illustrated by Figures 8 and 9.

Temporal consistency. We observed that there was a flaw in the original implementation of the training phase added for increasing temporal consistency in the "video" version. Indeed, the additional term in Equation (9) had been omitted. As an outcome, there was no reason that this training phase (as implemented in the original method) would increase the temporal consistency: the network is not trained anymore to colorize similarly a given image and its noisy version but was trained to perform denoising while colorizing. We observed that adding this phase was only leading to less colorful colorizations, but did not increase the temporal consistency. For this reason, we decided simply to withdraw this phase from our implementation of the method.

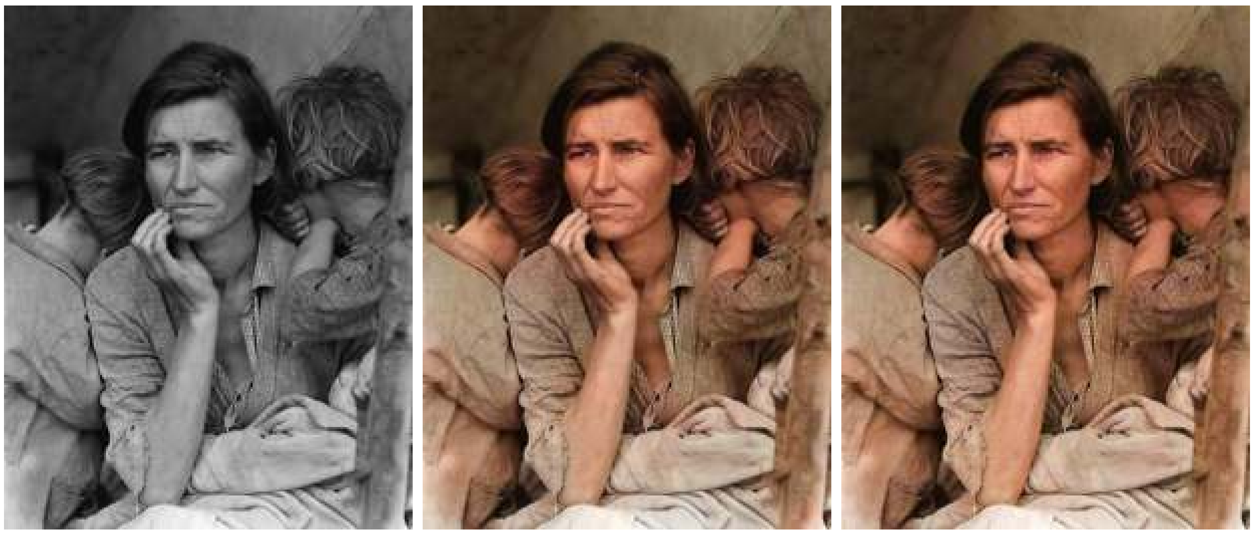


Figure 8: From left to right: gray-scale image, image colorized with the network trained with spectral normalization, image colorized without normalization. The colorization obtained by the network trained without spectral normalization does not differ greatly from the one obtained with the network trained with spectral normalization.

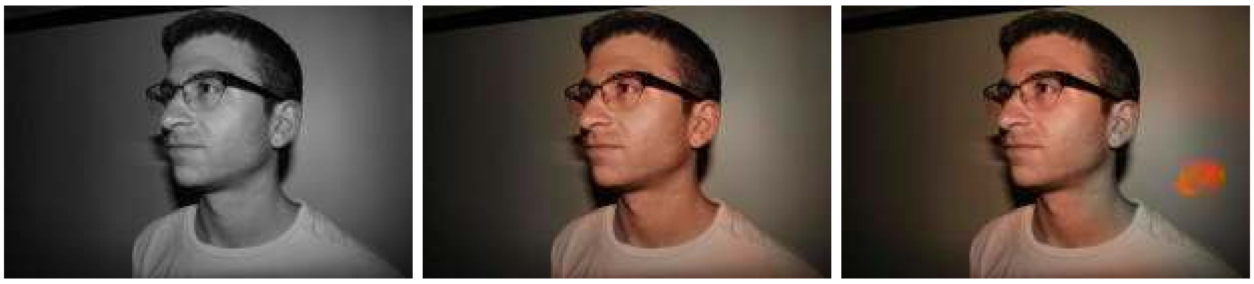


Figure 9: From left to right: original image, image colorized with the network trained with spectral normalization, image colorized without normalization. The colorization obtained by the trained network without spectral normalization produces false colors and leaves sections of the image uncolored.

About NoGAN. Focusing on the "video" version, we observed that the so-called NoGAN training phase could be replaced by a much simpler post-processing step at inference time leading to comparable results. It is enough to simply saturate colors in the output images colorized by the network trained with only (7). This color saturation is done in the following way

$$u^{rgb}_{satu} = (1-\alpha)u^{rgb}_{gs} + \alpha u^{rgb}_{color}, \tag{11}$$

where u^{rgb}_{color} is the color image in RGB, u^{rgb}_{gs} is the gray-scale image converted back to RGB (all channels contain the same values), u^{rgb}_{satu} is the target image with saturated colors, and α is a saturation parameter. The value of α has to be determined empirically, we found that $\alpha = 2.0$ offers realistic results in most cases. If necessary, the result is clipped to fit into the allowed output range. The GAN training is not completely equivalent to a simple color saturation: it seems to yield a bit more colorful colorizations, but those are less stable and contain a lot more artifacts. For this reason and also for a matter of efficiency, we decided to withdraw that second training phase and replace it by this simple color saturation process.

4 Results

In this section we provide results of the implemented colorization method, showing the impact of the inference parameters, as well as images where the method does not work satisfactorily.

4.1 Parameters Review

At inference time, two parameters are important.The render factor determines the resolution at which the image is rendered for inference. When set at a low value, the process is faster and the colors tend to be more vibrant, but the results are less stable. For higher resolution photos or portraits, larger render factor values are recommended. Values between 12 and 16 usually work relatively well for most images. The saturation parameter can be used to tone down unwanted colors or transform the colorized image to more vibrant colors. Values between 1.5 and 2 are suggested.

We present in Figure 10 the colorization of a high resolution image. The original color image is converted to gray-scale by the script, and then the colorization process is applied to the gray-scale image.

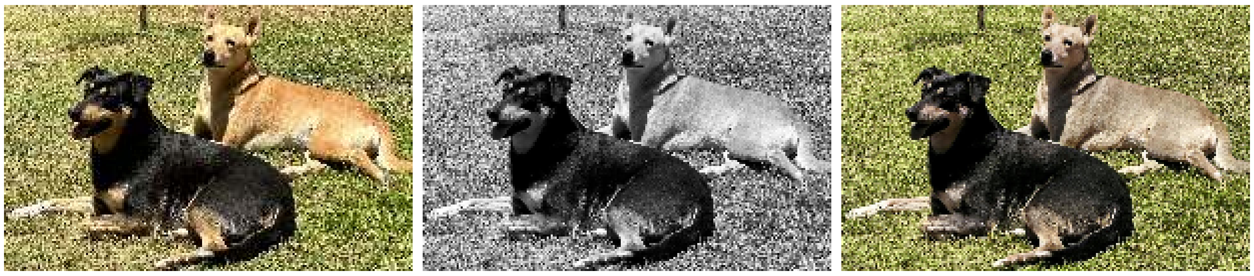


Figure 10: From left to right: original image, gray-scale image, colorized image with render factor equal to 24.

In Figure 11 we show zoomed versions of the images in Figure 10, to illustrate the effect of the render factor and saturation parameters and to understand how the network works on edges. All the images presented in Figure 11 were generated with a saturation value of 2, except the last one, which was generated with a value of 1.5.

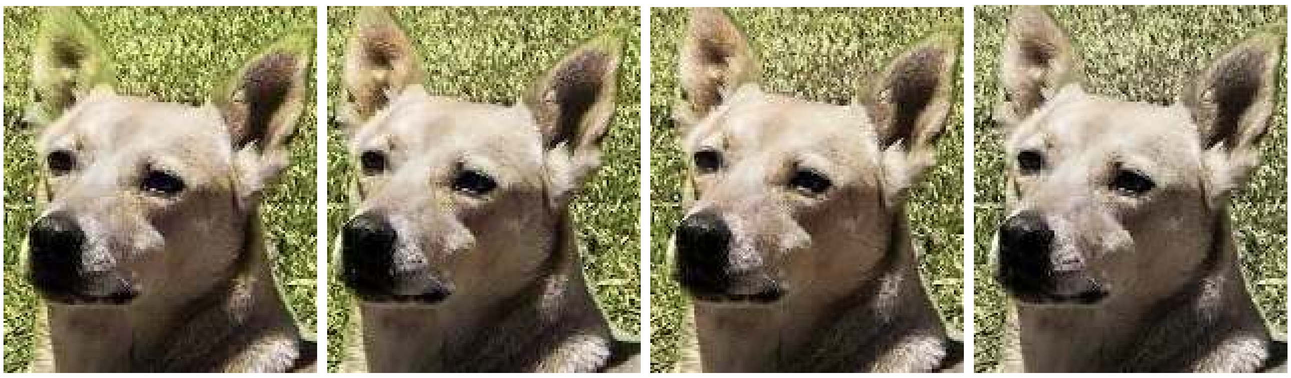


Figure 11: From left to right: colorized image with render factor of 16, 20, and 24, respectively, with a saturation value of 2. The last image was colorized with render factor of 24 and saturation of 1.5.

As it can be seen, the algorithm with a higher render factor works better on this image, since it is of high quality. It can be seen that in the sections where the contrast with the background is not excessive, the colorization is not good. For example, the green color of the grass is passed to the ears of the dog, however the same does not happen in the neck or other edges. By increasing the render factor, the colors are less vibrant, but the problem in the sections with less contrast with the background gradually decreases. Using a lower saturation value makes the ears look less green, but the overall image looks less colorful also.

We present in Figure 12 the colorization of an old picture, with less resolution than the previous image. It can be seen that lower render factors lead to better colorization of the landscape.

In Figure 13 we zoom on a part of the images. It can be seen that when the contrast between the objects is important, the colorization works correctly on the edges.

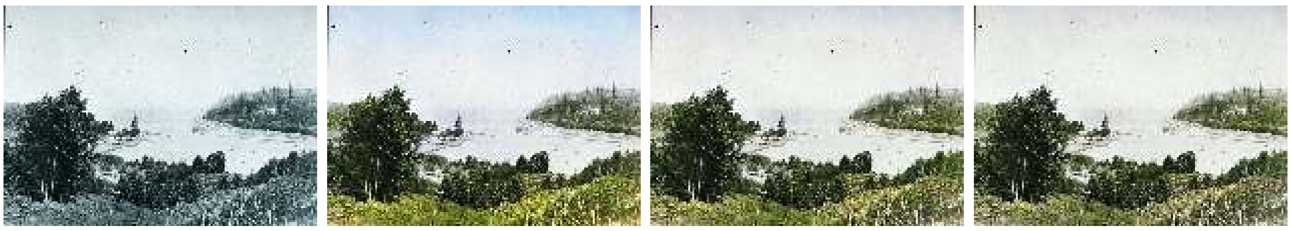


Figure 12: From left to right: gray-scale image, colorized image with render factor of 8, 16 and 24, respectively, with a saturation value of 2.

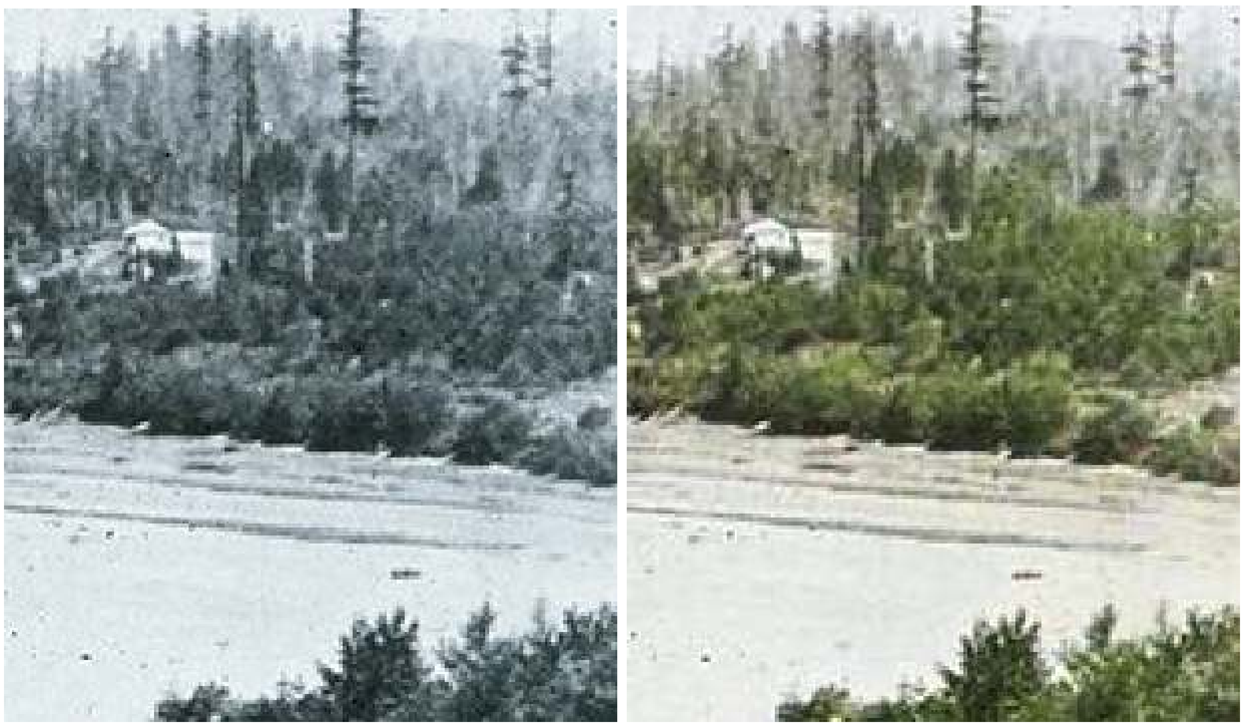


Figure 13: From left to right: zoom in the gray-scale image, zoom in colorized image of Figure 13. The colorization seems to work correctly on the edges.

Time complexity. In inference time, prior to applying the neural network, each channel of the image is resized to $16rf \times 16rf$, where rf stands for the parameter render factor. Let's call $n = 3(16rf)^2$ the total number of values of the network input. Almost all operations involved in the network, such as convolutions, ReLU or Sigmoid activation functions, are linear in the size of the input ($O(n)$). However, we also use a self-attention layer, which has quadratic complexity $O(n^2)$.

4.2 Examples of Satisfactory Colorization

We present below examples of satisfactory colorizations. We present examples of recent and old images, as well as different resolution qualities. The colors chosen by the modified DeOldify network seem surprisingly natural for old and recent images, as shown in Figures 8, 9, 14, 15, 16, 17 and 18. The network results are very stable with respect to image transformations, such as contrast changes.

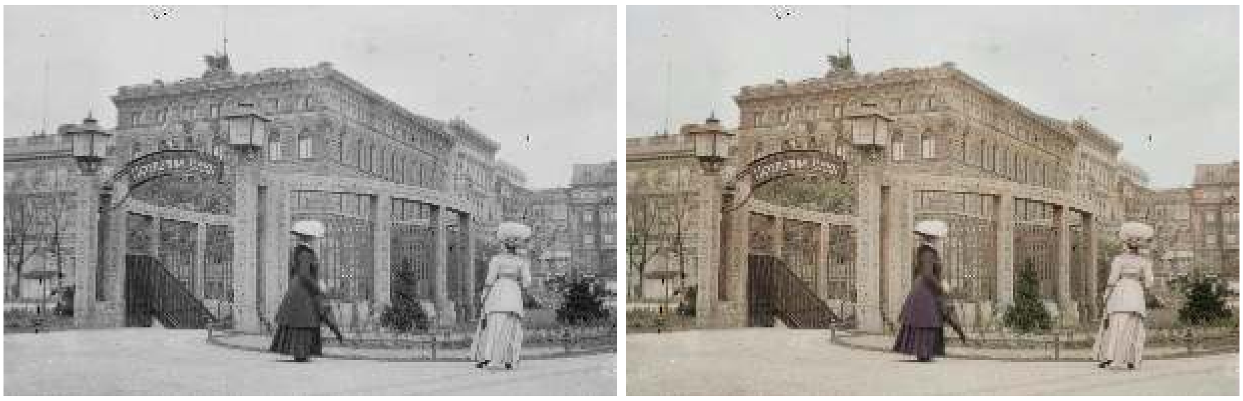


Figure 14: Left: gray-scale image. Right: colorized image.

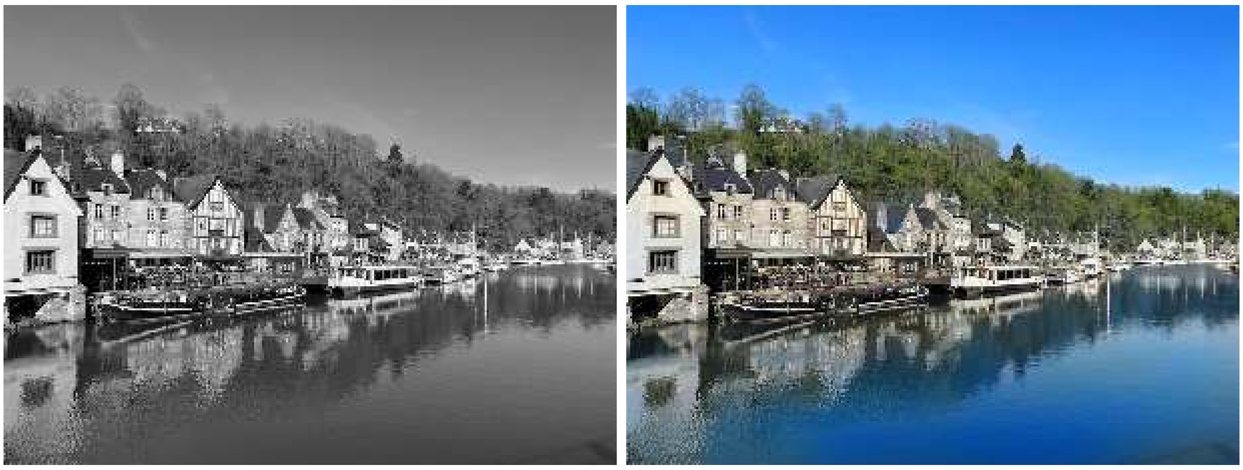


Figure 15: Left: gray-scale image. Right: colorized image.

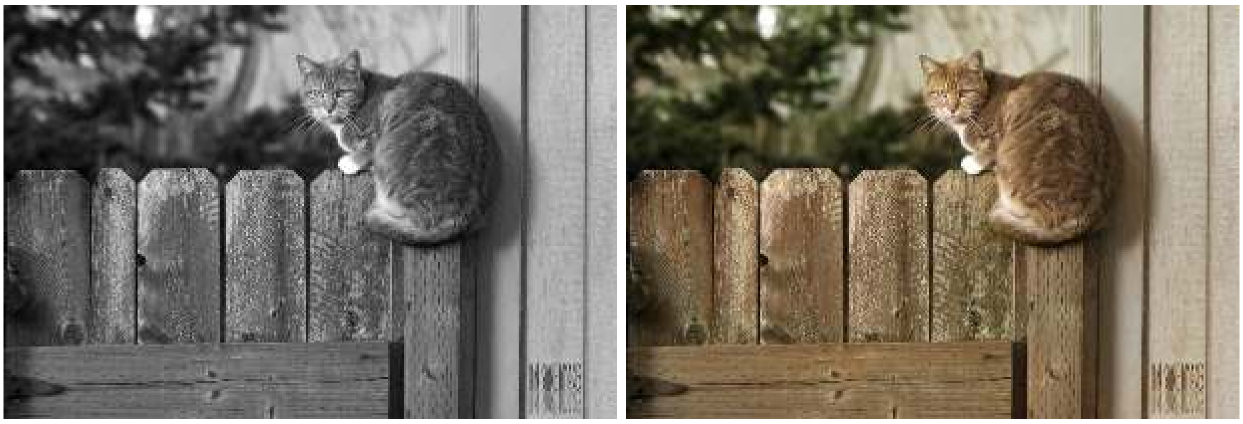


Figure 16: From left to right: gray-scale image, colorized image.

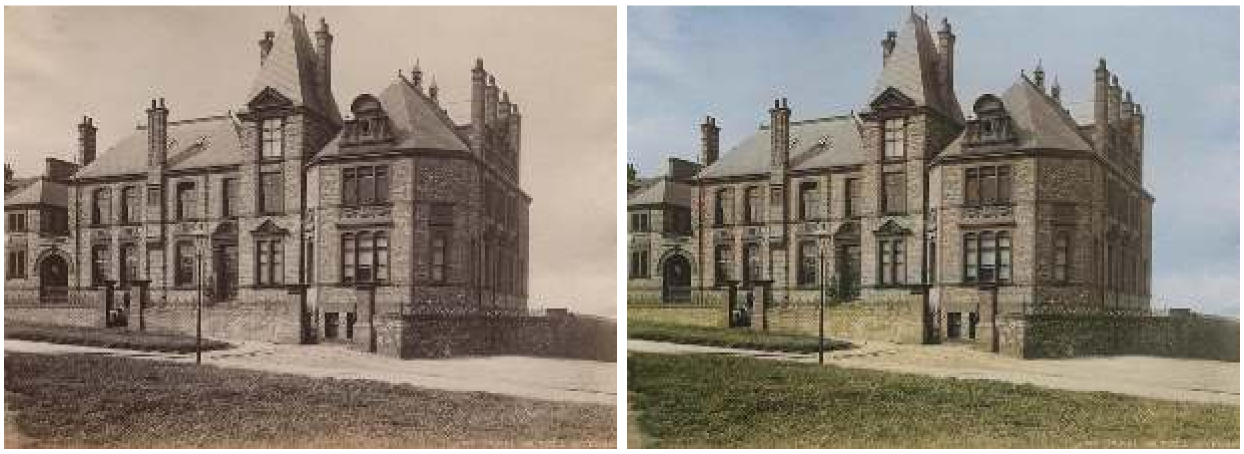


Figure 17: Left: gray-scale image. Right:colorized image.

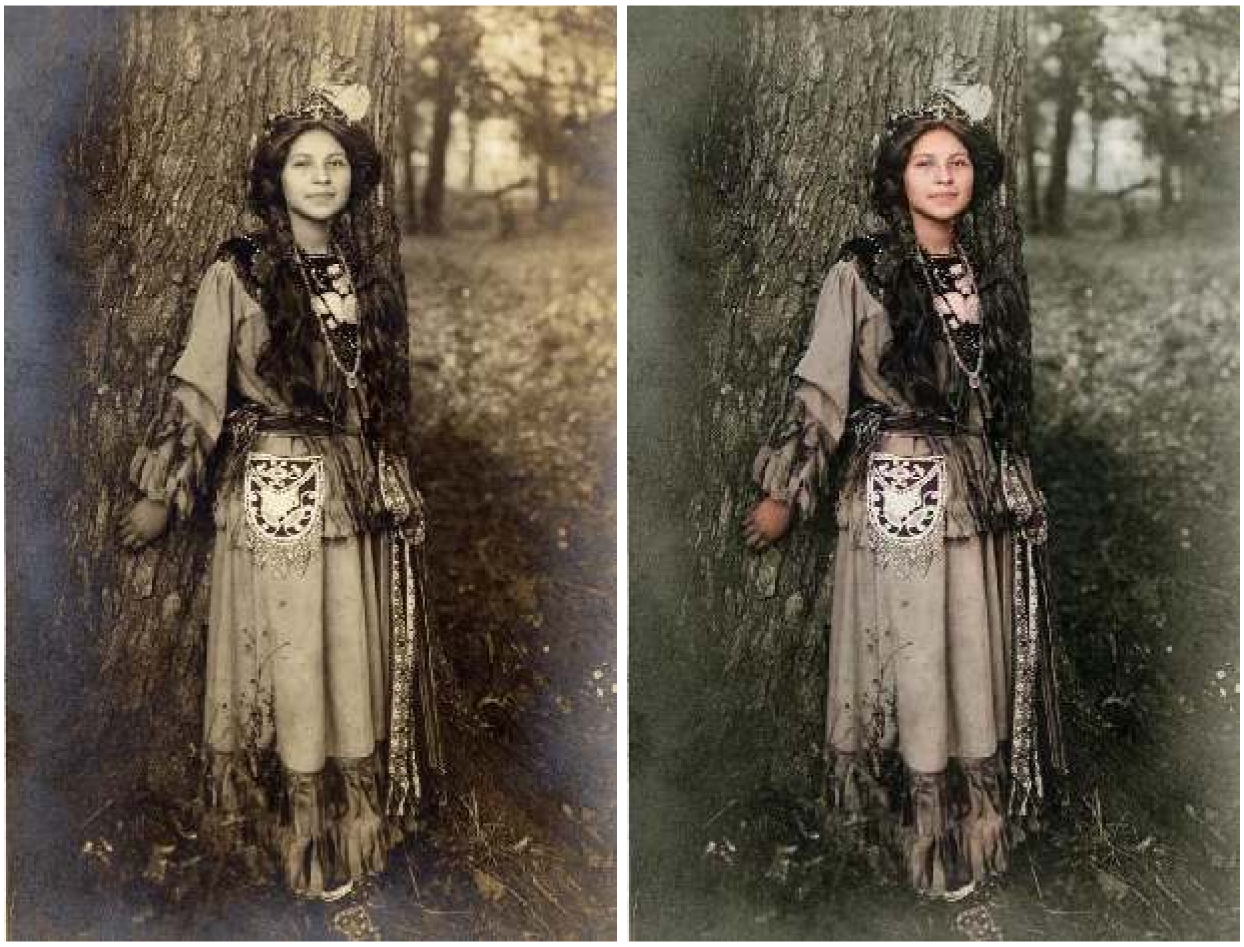


Figure 18: Left: gray-scale image. Right:colorized image.

4.3 Examples of Unsatisfactory Colorization

The network does not always provide satisfactory results, as shown in Figure 19, where the man's hand is completely gray.

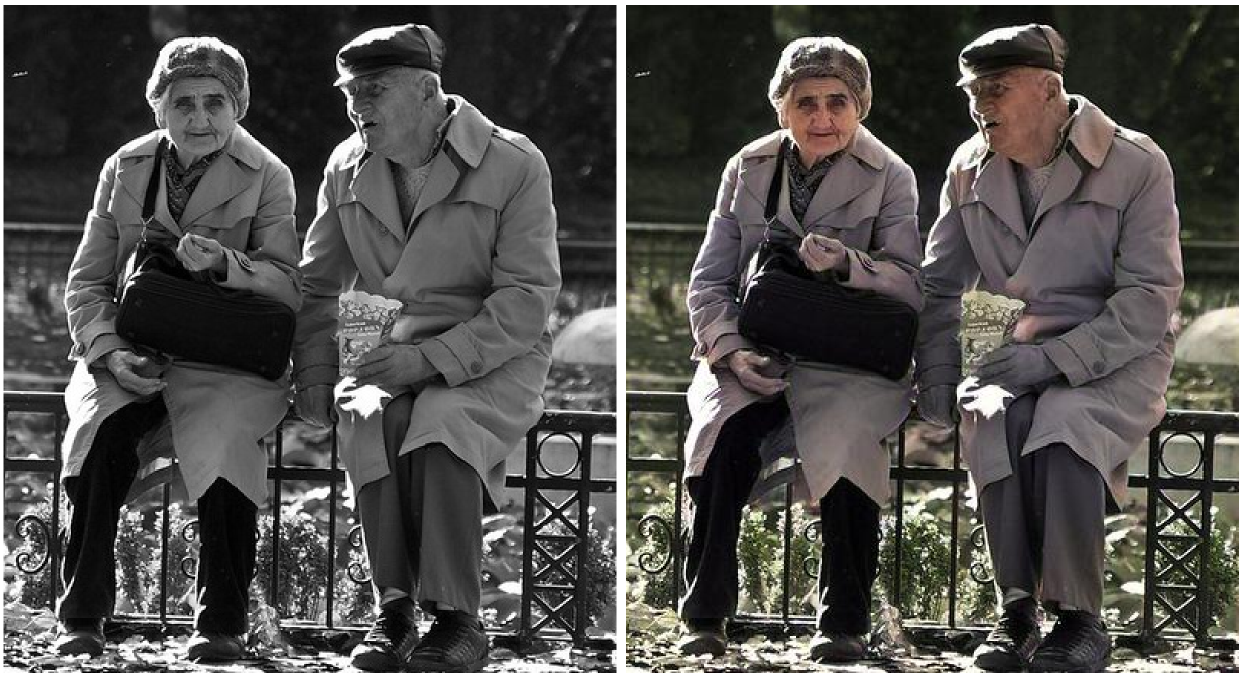


Figure 19: Left: gray-scale image. Right: colorized image with unsatisfactory results (the man's hand is gray).

Colorization failures also often happen on close-up images, as shown in Figure 20. In these images, the flowers in the foreground are colorized with the same color as the background.

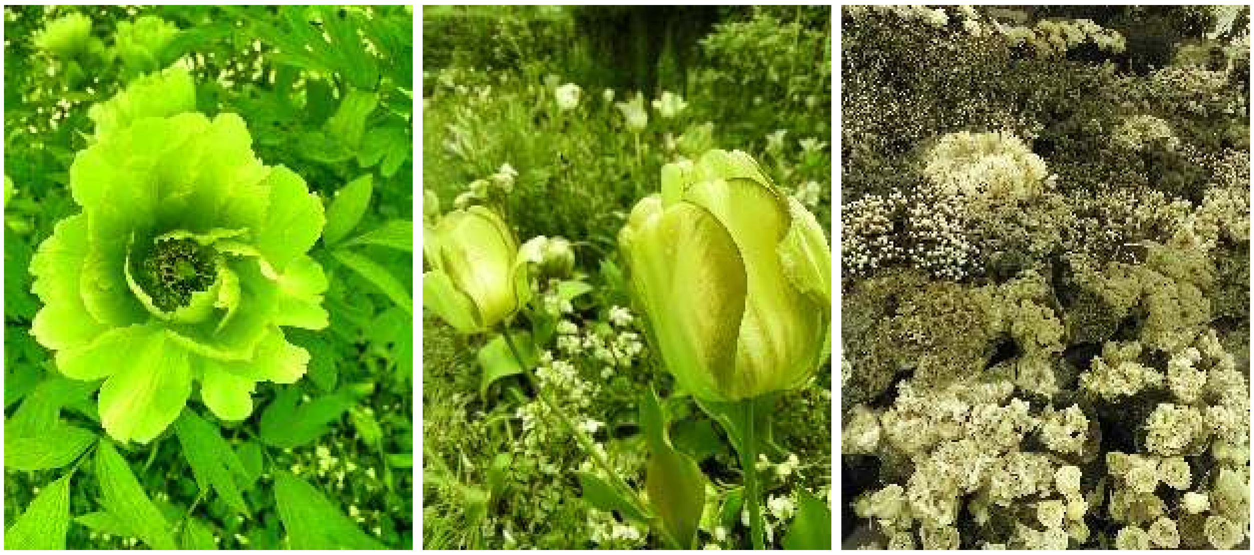


Figure 20: Results of colorization of flowers. It can be seen that the flowers are painted green, even when the background is not, like in the photo on the left.

4.4 Video Colorization

The DeOldify colorization network can be applied separately on each image of a video. Results on archive or degraded videos are not always satisfactory, containing a lot of flickering and colors in sepia tones. Usually, results on higher quality videos are much more satisfying and consistent, even if a slight flicker can still be observed. A video colorized by the network is provided as supplementary material of the article.

5 Demo

In order to use the demo provided on the IPOL website, it is necessary to upload an image or choose one of the available ones (which should not necessarily be in gray-scale). The following parameters can be modified:

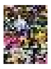
- Render Factor: determines the resolution at which the image is rendered for inference. When set at a low value, the process is faster and the colors tend to be more vibrant, but the results are less stable. For higher resolution photos or portraits, larger render factor values are recommended. Values between 12 and 16 usually work for most images.
- Saturation parameter: a value of 1 does not apply saturation. In practice, we recommend to use values between 1.5 and 2.

6 Conclusions

This paper presents the first complete and detailed description of the DeOldify automatic colorization method, proposed by Jason Antic, along with a modified open-source implementation of the original method. This implementation has no dependency on third-party libraries and is easier to train since the NoGAN training phase is replaced by a simple image saturation step. As shown in the experimental section, colorized images obtained with the new method are often very satisfactory, especially on landscape images and full-length portrait. It remains that image colorization is a very ill-posed problem, and the method fails to work correctly on several images, as shown in the results section. Furthermore, the parameter choice at inference time may lead to different quality of colorizations.

Image Credits

© L.Bouza CC-BY 2.0.


© L.Bouza CC-BY 1.0.

© L.Bouza CC-BY 1.0.

© L.Bouza CC-BY 1.0.

References

[1] J. Antic, J. Howard, and U. Manor, *Decrappification, DeOldification, and Super Resolution*, 2019. https://www.fast.ai/2019/05/03/decrappify.

[2] A. Buades, B. Coll, and J-M. Morel, *A non-local algorithm for image denoising*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005, pp. 60–65. https://doi.org/10.1109/CVPR.2005.38.

[3] A. Deshpande, J. Lu, M-C. Yeh, M.J. Chong, and D. Forsyth, *Learning diverse image colorization*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2877–2885. https://doi.org/10.1109/CVPR.2017.307.

[4] A. Deshpande, J. Rock, and D. Forsyth, *Learning large-scale automatic image colorization*, in IEEE International Conference on Computer Vision (ICCV), 2015, pp. 567–575. https://doi.org/10.1109/ICCV.2015.72.

[5] N. El abbadi and E. Saleem, *Automatic gray images colorization based on lab color space*, Indonesian Journal of Electrical Engineering and Computer Science, 18 (2020), p. 1501. https://doi.org/10.11591/ijeecs.v18.i3.pp1501-1509.

[6] F. Gogianu, T. Berariu, M. Rosca, C. Clopath, L. Busoniu, and R. Pascanu, *Spectral normalisation for deep reinforcement learning: an optimisation perspective*, CoRR, abs/2105.05246 (2021). https://doi.org/10.48550/arXiv.2105.05246.

[7] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. https://doi.org/10.48550/ARXIV.1512.03385.

[8] S. Iizuka, E. Simo-Serra, and H. Ishikawa, *Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification*, ACM Transactions on Graphics, 35 (2016), pp. 1–11. https://doi.org/10.1145/2897824.2925974.

[9] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, pp. 448–456. http://dl.acm.org/citation.cfm?id=3045118.3045167.

[10] J. Johnson, A. Alahi, and L. Fei-Fei, *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*, arXiv e-prints, (2016). https://doi.org/10.48550/arXiv.1603.08155.

[11] T. Karras, T. Aila, S. Laine, and J. Lehtinen, *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, 2017. https://doi.org/10.48550/arXiv.1710.10196.

[12] M. Kumar, D. Weissenborn, and N. Kalchbrenner, *Colorization transformer*, CoRR, abs/2102.04432 (2021). https://arxiv.org/abs/2102.04432.

[13] G. Larsson, M. Maire, and G. Shakhnarovich, *Learning representations for automatic colorization*, 2016. https://doi.org/10.48550/ARXIV.1603.06668.

[14] A. Levin, D. Lischinski, and Y. Weiss, *Colorization using optimization*, ACM Transactions on Graphics, 23 (2004), pp. 689–694. https://doi.org/10.1145/1015706.1015780.

[15] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2017. https://doi.org/10.48550/arXiv.1711.05101.

[16] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, *Spectral normalization for generative adversarial networks*, CoRR, abs/1802.05957 (2018). https://doi.org/10.48550/arXiv.1802.05957.

[17] J. Rainy and A. de Berker, *Stabilizing neural style-transfer for video*, 2018. https://medium.com/element-ai-research-lab/stabilizing-neural-style-transfer-for-video-62675e203e42.

[18] E. Reinhard, M. Adhikhmin, B. Gooch, and P. Shirley, *Color transfer between images*, IEEE Computer Graphics and Applications, 21 (2001), pp. 34–41. https://doi.org/10.1109/38.946629.

[19] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015. https://doi.org/10.48550/ARXIV.1505.04597.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, Mi. Bernstein, A.C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252. https://doi.org/10.1007/s11263-015-0816-y.

[21] C. Saharia, W. Chan, H. Chang, C.A. Lee, J. Ho, T. Salimans, D.J. Fleet, and M. Norouzi, *Palette: Image-to-image diffusion models*, 2021. https://doi.org/10.48550/ARXIV.2111.05826.

[22] W. Shi, J. Caballero, F. Huszr, J. Totz, A.P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, *Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network*, 2016. https://doi.org/10.48550/ARXIV.1609.05158.

[23] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv 1409.1556, (2014). https://doi.org/10.48550/arXiv.1409.1556.

[24] L.N. Smith, *A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay*, CoRR, abs/1803.09820 (2018). https://doi.org/10.48550/arXiv.1803.09820.

[25] J-W. Su, H-K. Chu, and J-B. Huang, *Instance-aware image colorization*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 7965–7974. https://doi.org/10.1109/CVPR42600.2020.00799.

[26] Y. Sugawara, S. Shiota, and H. Kiya, *Super-resolution using convolutional neural networks without any checkerboard artifacts*, 2018. https://doi.org/10.48550/ARXIV.1806.02658.

[27] P. Vitoria, L. Raad, and C. Ballester, *ChromaGAN: Adversarial Picture Colorization with Semantic Class Distribution*, in IEEE Winter Conference on Applications of Computer Vision (WACV), 2020, pp. 2434–2443. https://doi.org/10.1109/WACV45572.2020.9093389.

[28] X. Wang, R. Girshick, A. Gupta, and K. He, *Non-local neural networks*, 2017. https://doi.org/10.48550/ARXIV.1711.07971.

[29] T. Welsh, M. Ashikhmin, and K. Mueller, *Transferring color to greyscale images*, ACM Transactions on Graphics, 21 (2002), pp. 277–280. https://doi.org/10.1145/566570.566576.

[30] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, *Self-attention generative adversarial networks*, 2018. https://doi.org/10.48550/arXiv.1805.08318.

[31] R. Zhang, P. Isola, and A.A. Efros, *Colorful image colorization*, 2016. https://doi.org/10.48550/ARXIV.1603.08511.