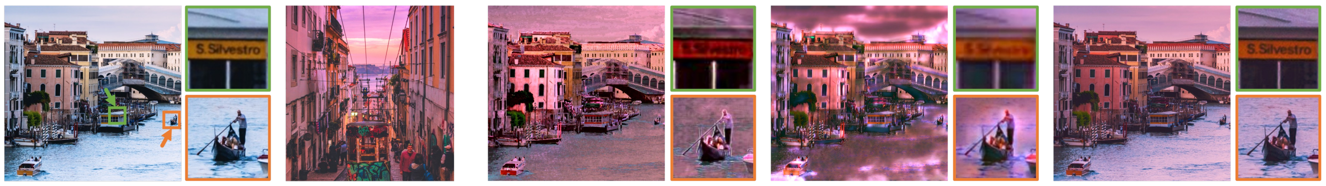
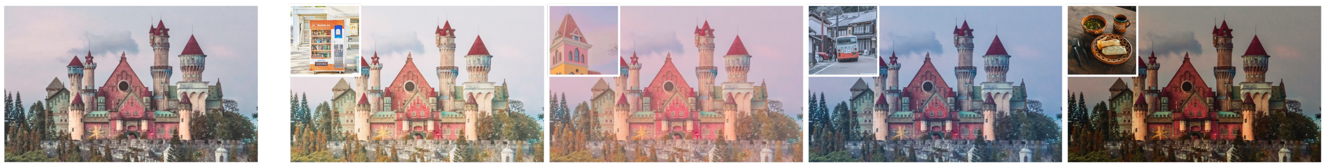
Neural Preset for Color Style Transfer

Zhanghan Ke[1] Yuhao Liu[1] Lei Zhu[1] Nanxuan Zhao[2] Rynson W.H. Lau[1†]
[1]City University of Hong Kong [2]Adobe Research



(a) Comparison with State-of-the-art Color Style Transfer Methods on 2K Images



(b) Diverse Color Style Transfer Results Produced by Our Method on 8K Images (The Top-Left Corner Shows the Style Image)

Figure 1. **Our Color Style Transfer Results.** (a) State-of-the-art methods PhotoNAS [1] and PhotoWCT^2 [6] produce distort textures (*e.g.*, text in green box) and dissonant colors (*e.g.*, content in orange box). Besides, they have long inference time even on the latest Nvidia RTX3090 GPU (red numbers in brackets). In contrast, our method avoids artifacts and is $\sim$28$\times$ faster. (b) Our method can produce faithful results on 8K images, but both PhotoNAS and PhotoWCT^2 run into the out-of-memory problem. Zoom in for better visualization.

Abstract

In this paper, we present a Neural Preset technique to address the limitations of existing color style transfer methods, including visual artifacts, vast memory requirement, and slow style switching speed. Our method is based on two core designs. First, we propose Deterministic Neural Color Mapping (DNCM) to consistently operate on each pixel via an image-adaptive color mapping matrix, avoiding artifacts and supporting high-resolution inputs with a small memory footprint. Second, we develop a two-stage pipeline by dividing the task into color normalization and stylization, which allows efficient style switching by extracting color styles as presets and reusing them on normalized input images. Due to the unavailability of pairwise datasets, we describe how to train Neural Preset via a self-supervised strategy. Various advantages of Neural Preset over existing methods are demonstrated through comprehensive evaluations. Besides, we show that our trained model can naturally support multiple applications without fine-tuning, including low-light image enhancement, underwater image correction, image dehazing, and image harmonization. The project page is: https://ZHKKKe.github.io/NeuralPreset.

1. Introduction

With the popularity of social media (*e.g.*, Instagram and Facebook), people are increasingly willing to share photos in public. Before sharing, color retouching becomes an indispensable operation to help express the story captured in images more vividly and leave a good first impression. Photo editing tools usually provide color style presets, such as image filters or Look-Up Tables (LUTs), to help users explore efficiently. However, these filters/LUTs are handcrafted with pre-defined parameters, and are not able to generate consistent color styles for images with diverse appearances. Therefore, careful adjustments by the users is still necessary. To address this problem, color style transfer techniques have been introduced to automatically map the color style from a well-retouched image (*i.e.*, the style image) to another (*i.e.*, the input image).

Earlier color style transfer methods [41–43, 49] focus on retouching the input image according to low-level feature statistics of the style image. They disregard high-level information, resulting in unexpected changes in image in-

[†] Corresponding author. This project is in part supported by a General Research Fund from RGC of Hong Kong (RGC Ref.: 11205620).

herent colors. Although recent deep learning based models [1,6,19,34,36,54] give promising results, they typically suffer from three obvious limitations in practice (Fig. 1 (a)). First, they produce unrealistic artifacts (*e.g.*, distorted textures or inharmonious colors) in the stylized image since they perform color mapping based on convolutional models, which operate on image patches and may have inconsistent outputs for pixels with the same value. Although some auxiliary constraints [36] or post-processing strategies [34] have been proposed, they still fail to prevent artifacts robustly. Second, they cannot handle high-resolution (*e.g.*, 8K) images due to their huge runtime memory footprint. Even using a GPU with 24GB of memory, most recent models suffer from the out-of-memory problem when processing 4K images. Third, they are inefficient in switching styles because they carry out color style transfer as a single-stage process, requiring to run the whole model every time.

In this work, we present a Neural Preset technique with two core designs to overcome the above limitations:

(1) Neural Preset leverages Deterministic Neural Color Mapping (DNCM) as an alternative to the color mapping process based on convolutional models. By multiplying an image-adaptive color mapping matrix, DNCM converts pixels of the same color to a specific color, avoiding unrealistic artifacts effectively. Besides, DNCM operates on each pixel independently with a small memory footprint, supporting very high-resolution inputs. Unlike adaptive 3D LUTs [7, 55] that need to regress tens of thousands of parameters or automatic filters [23, 27] that perform particular color mappings, DNCM can model arbitrary color mappings with only a few hundred learnable parameters.

(2) Neural Preset carries out color style transfer in two stages to enable fast style switching. Specifically, the first stage builds a *nDNCM* from the input image for color normalization, which maps the input image to a normalized color style space representing the "image content"; the second stage builds a *sDNCM* from the style image for color stylization, which transfers the normalized image to the target color style. Such a design has two advantages in terms of efficiency: the parameters of *sDNCM* can be stored as color style presets and reused by different input images, while the input image can be stylized by diverse color style presets after normalized once with *nDNCM*.

In addition, since there are no pairwise datasets available, we propose a new self-supervised strategy for Neural Preset to be trainable. Our comprehensive evaluations demonstrate that Neural Preset outperforms state-of-the-art methods significantly in various aspects. Notably, Neural Preset can produce faithful results for 8K images (Fig. 1 (b)) and can provide consistent color style transfer results across video frames without post-processing. Compared to recent deep learning based models, Neural Preset achieves $\sim 28\times$ speedup on a Nvidia RTX3090 GPU, supporting real-time performances at 4K resolution. Finally, we show that our trained model can be applied to other color mapping tasks without fine-tuning, including low-light image enhancement [30], underwater image correction [52], image dehazing [16], and image harmonization [37].

2. Related Works

Color Style Transfer. Unlike artistic style transfer [2, 5, 9–11, 20, 21, 24, 26, 31, 32] that alters both textures and colors of images, color style transfer (*aka* photorealistic style transfer) aims to shift only the colors from one image to another. Traditional methods [41–43, 49] mostly match the statistics of low-level features, such as the mean and variance of images [43] or the histograms of filter responses [41]. However, these methods often transfer unwanted colors if the style and input images have large appearance differences. Recently, many methods exploiting convolutional neural networks (CNNs) [1, 6, 19, 34, 36, 54] are proposed for color style transfer. For example, Yoo *et al.* [54] introduce a model with wavelet pooling/unpooling to reduce distortions. An *et al.* [1] use network architecture search to explore a more effective asymmetric model. Chiu *et al.* [6] propose to obtain a more compact model by block-wise training with a coarse-to-fine transformation. To address the limitations (*i.e.*, visual artifacts, huge memory consumption, and inefficient in switching styles) of the aforementioned methods as stated in Sec. 1, we present Neural Preset that supports artifact-free color style transfer with only a small memory footprint, via DNCM, and enables fast style switching, via a two-stage pipeline.

Deterministic Color Mapping with CNNs. Filters and LUTs avoid artifacts as they perform *deterministic color mapping* to produce consistent outputs for the same input pixel values. Some recent image enhancement and harmonization methods [7, 23, 27, 55] have attempted to implement color mapping using filters/LUTs with image-adaptive parameters predicted by CNNs. However, combining filters/LUTs with CNNs for color mapping has clear drawbacks. Filter-based methods [23, 27] integrate a finite number of image filters, and can only handle basic color adjustments, *e.g.*, brightness and contrast. LUT-based methods [7, 55] need to predict coefficients to linearly merge several template LUTs, because LUTs have a large number of learnable parameters that are difficult to optimize via CNNs. Although the affine bilateral grid [12] may model complex color mapping with fewer learnable parameters, it cannot provide deterministic color mapping. Applying it to color style transfer [50] may lead to inharmonious colors in different regions of the image. Instead of adopting the aforementioned schemes, we propose DNCM that has only a few hundred learnable parameters but can model arbitrary deterministic color mappings.

Self-Supervised Learning (SSL). SSL has been widely explored for pre-training [3, 4, 13, 15, 17, 18, 38–40]. Some works also solve specific vision tasks via SSL [22, 25, 29]. Since it is expensive to annotate ground truths for color style transfer, most methods either minimize perceptual losses [2, 11, 24, 36] or match the statistics of image features [6, 33, 34, 54]. However, such weak constraints usually result in severe visual artifacts. Yim *et al.* [53] and Ho *et al.* [19] suggest imposing stronger constraints by reconstructing perturbed images, but their trained models can only transfer color styles to images with natural appearances, *e.g.*, images taken by a camera without post-processing. To this end, we present a new SSL strategy for Neural Preset, which not only learns from reconstructing perturbed images but also enables the trained models to transfer color styles between arbitrary images.

3. Method

Our Neural Preset performs color style transfer through a two-stage pipeline, where both stages employ DNCM for color mapping. In this section, we first introduce DNCM in details (Sec. 3.1). We then present the two-stage DNCM-based color style transfer pipeline (Sec. 3.2). Finally, we describe our self-supervised learning strategy for training Neural Preset (Sec. 3.3).

3.1. Deterministic Neural Color Mapping (DNCM)

A straightforward idea to model deterministic color mapping that can adapt to different images is to combine filters/LUTs with the image-adaptive parameters predicted by CNNs. However, each image filter can only provide a single color mapping. Integrating a finite number of filters can only cover a limited range of color mappings. Besides, as a common 32-level 3D LUT has ∼10K parameters, it is infeasible to regress image-specific 3D LUTs. The approach based on predicting coefficients to merge template LUTs still needs to optimize tens of thousands of parameters to build template LUTs.

Here, we propose DNCM to model arbitrary deterministic color mapping with much fewer learnable parameters. As shown in Fig. 2, given an input image $\mathbf{I}$ of size $(h, w, 3)$, we downsample it to obtain a thumbnail $\tilde{\mathbf{I}}$, which provides image-adaptive color mapping matrix $\mathbf{T}$ for DNCM. Specifically, we feed $\tilde{\mathbf{I}}$ into an encoder E to predict $\mathbf{T}$ of size $(k \times k)$, and then reshape it to a size of (k, k), as:

$$\mathbf{T}^{(k\times k)} = E(\tilde{\mathbf{I}}), \quad \mathbf{T}^{(k\times k)} \rightarrow \mathbf{T}^{(k,k)}, \tag{1}$$

where $\rightarrow$ denotes the reshape operation, and k is empirically set to a small value (*e.g.*, 16). With matrix $\mathbf{T}^{(k,k)}$, we form DNCM to alter the colors of $\mathbf{I}$. In DNCM, we first unfold $\mathbf{I}$ as a 2D matrix of size $(h \times w, 3)$. We then embed each pixel in $\mathbf{I}$ into a k-dimensional vector by a projection

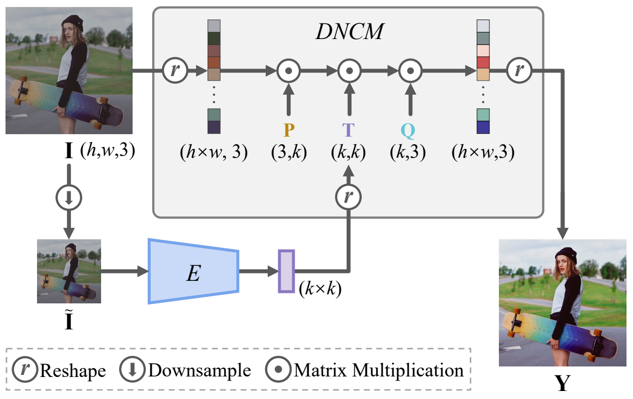


Figure 2. **Illustration of DNCM.** The DNCM parameters consist of two color projection matrices ($\mathbf{P}$, $\mathbf{Q}$) and image-adaptive parameters $\mathbf{T}$ predicted by an encoder E. Here, we obtain $\tilde{\mathbf{I}}$ by downsampling $\mathbf{I}$. DNCM maps the input image $\mathbf{I}$ to output image $\mathbf{Y}$ by multiplying $\mathbf{I}$ with $\mathbf{P}$, $\mathbf{T}$, and $\mathbf{Q}$ sequentially.

matrix $\mathbf{P}^{(3,k)}$. After that, we multiply the embedded vectors by $\mathbf{T}^{(k,k)}$. Finally, we apply another projection matrix $\mathbf{Q}^{(k,3)}$ to convert the embedded vectors back to the RGB color space and reshape pixels to output $\mathbf{Y}$ with new colors. Note that both $\mathbf{P}$ and $\mathbf{Q}$ are learnable matrices shared by all images. Formally, DNCM can be defined as:

$$\mathbf{Y} = \textit{DNCM}(\mathbf{I},\, \mathbf{T}) = \mathbf{I}^{(h\times w,3)} \cdot \mathbf{P}^{(3,k)} \cdot \mathbf{T}^{(k,k)} \cdot \mathbf{Q}^{(k,3)}, \tag{2}$$

where $\cdot$ denotes matrix multiplication. We omit the reshape operations in Eq. 2 for simplicity. Besides, for DNCM to be a non-linear transformation, we add a *tanh* function after each multiplication operation, except the last one.

Implementing color mapping as Eq. 2 brings three main benefits. First, it effectively avoids visual artifacts as pixels of the same color in $\mathbf{I}$ will still have the same color after being mapped to $\mathbf{Y}$. Second, it requires only a small memory footprint since each pixel is processed independently with efficient matrix multiplications. Third, it makes $E()$ easy to optimize as only $k \times k$ image-adaptive parameters (*i.e.*, $\mathbf{T}$) should be regressed.

3.2. Two-Stage Color Style Transfer Pipeline

Recent methods implicitly embed the color style transfer process into CNNs to form single-stage pipelines. Therefore, they must run the whole pipeline every time to compute the color mapping between two images, which is inefficient when applying diverse color styles to an image or transferring a color style to multiple images.

In contrast, we design an explicit two-stage pipeline based on DNCM. The key insight behind our pipeline is that if we can separate the color style of an image from its "image content", we can effectively transfer different color styles to the "image content". However, to achieve this, we need to answer two questions. First, how to remove or add

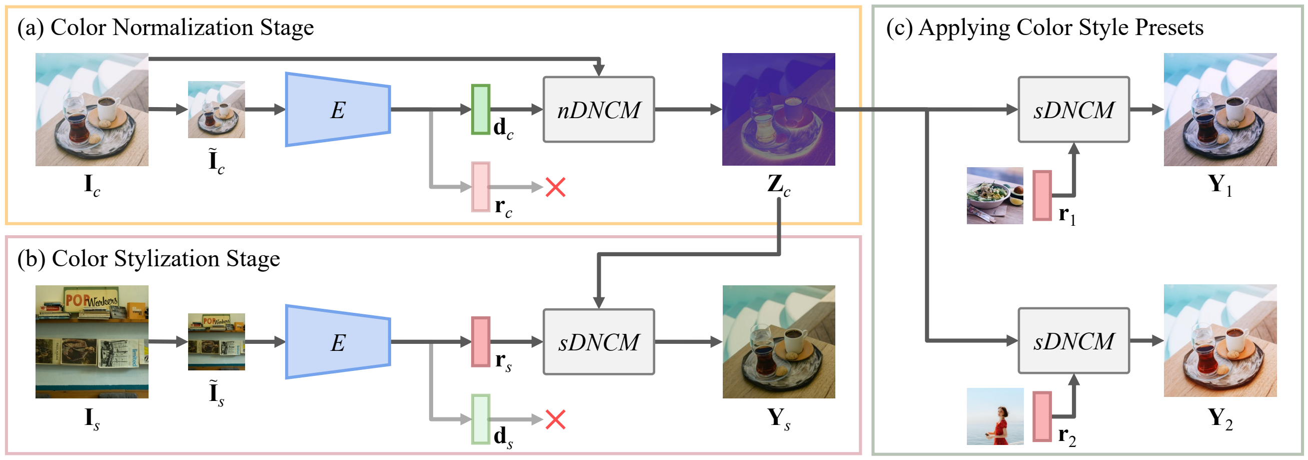


Figure 3. **Overview of Our Pipeline.** Our pipeline consists of two stages: (a) in the first stage, the input image $\mathbf{I}_c$ is converted to an image $\mathbf{Z}_c$ in the normalized color style space via *nDNCM* with parameters $\mathbf{d}_c$; (b) in the second stage, the color style parameters $\mathbf{r}_s$ are extracted from the style image $\mathbf{I}_s$ for *sDNCM* to map $\mathbf{Z}_c$ to $\mathbf{Y}_s$, which will then have the same color style as $\mathbf{I}_s$. Besides, the design of our pipeline supports fast style switching: in (c), the preset color style parameters $\mathbf{r}_1$/$\mathbf{r}_2$ can be reused by *sDNCM* to stylize $\mathbf{Z}_c$ to obtain $\mathbf{Y}_1$/$\mathbf{Y}_2$.

color styles? Since we need to alter the image color style but preserve the "image content", we propose to utilize a pair of *nDNCM* and *sDNCM*. While *nDNCM* converts the input image to a space that contains only the "image content", *sDNCM* transfers the "image content" to the target color style, using parameters extracted from the style image. Second, how to represent "image content"? Since this concept is difficult to define through hand-crafted features, we propose to learn a normalized color style space representing the "image content" by back-propagation. In such a normalized color style space, images of the same content but with different color styles should have a consistent appearance, *i.e.*, the same normalized color style.

As shown in Fig. 3 (a)(b), we modify the encoder E to output $\mathbf{d}$ and $\mathbf{r}$, which are applied as the parameters of *nDNCM* and *sDNCM*, respectively. Suppose that we want to transfer the color style of a style image $\mathbf{I}_s$ to an input image $\mathbf{I}_c$. In the first stage, we convert $\mathbf{I}_c$ to $\mathbf{Z}_c$ in the normalized color space via *nDNCM* with $\mathbf{d}_c$ predicted from $\tilde{\mathbf{I}}_c$, as:

$$\mathbf{Z}_c = nDNCM(\mathbf{I}_c,\, \mathbf{d}_c), \quad \text{where } \{\mathbf{d}_c, \mathbf{r}_c\} = E(\tilde{\mathbf{I}}_c). \quad (3)$$

In the second stage, we extract $\mathbf{r}_s$, *i.e.*, the parameters containing the color style of $\mathbf{I}_s$, to transfer $\mathbf{Z}_c$ to the stylized image $\mathbf{Y}_s$ via *sDNCM*, as:

$$\mathbf{Y}_s = sDNCM(\mathbf{Z}_c,\, \mathbf{r}_s), \quad \text{where } \{\mathbf{d}_s, \mathbf{r}_s\} = E(\tilde{\mathbf{I}}_s). \quad (4)$$

$E()$ used in Eq. 3 and 4 have shared weights. *nDNCM* and *sDNCM* have different projection matrices $\mathbf{P}$ and $\mathbf{Q}$.

By storing color style parameters (*e.g.*, $\mathbf{r}_s$) as presets and reusing them to construct *sDNCM*, our pipeline can support fast style switching using color style presets – we only need to normalize the color style of the input image once, and can then quickly retouch it to diverse color styles by *sDNCM* with the stored presets (*e.g.*, $\mathbf{r}_s$). For example, in Fig. 3 (c), we apply presets $\mathbf{r}_1$ and $\mathbf{r}_2$ on $\mathbf{Z}_c$ to obtain $\mathbf{Y}_1$ and $\mathbf{Y}_2$.

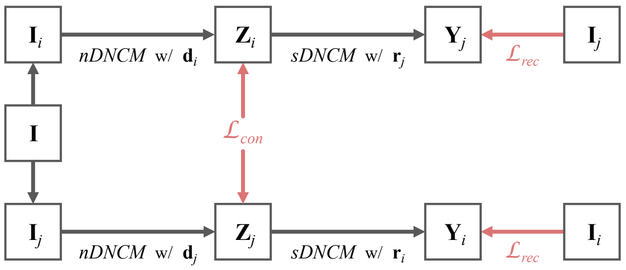


Figure 4. **Our Self-Supervised Training Strategy.** Both $\mathbf{I}_i$/$\mathbf{I}_j$ are generated from $\mathbf{I}$ via random color perturbations. We constrain $\mathbf{Z}_i$/$\mathbf{Z}_j$ to be the same via a consistency loss $\mathcal{L}_{con}$ and learn style transfer results $\mathbf{Y}_i$/$\mathbf{Y}_j$ via a reconstruction loss $\mathcal{L}_{rec}$.

3.3. Self-Supervised Training Strategy

We develop a self-supervised strategy to train Neural Preset, as shown in Fig. 4. Since no ground truth stylized images are available, we create pseudo stylized images from the input image $\mathbf{I}$. Specifically, we add perturbations on $\mathbf{I}$ to obtain two augmented samples with different color styles, which are denoted as $\mathbf{I}_i$ and $\mathbf{I}_j$. The perturbations we use involve operations that only change image colors, *e.g.*, random image filters or LUTs.

The first stage of our pipeline aims to normalize the color style of input images, which means that input images with the same content but different color styles should be consistent in the normalized color style space. Hence, we apply a L2 consistency loss between the outputs of this stage. Formally, we predict the *nDNCM* parameters $\mathbf{d}_i$/$\mathbf{d}_j$ to transfer

$\mathbf{I}_i/\mathbf{I}_j$ to $\mathbf{Z}_i/\mathbf{Z}_j$, and we constrain $\mathbf{Z}_i/\mathbf{Z}_j$ by:

$$\begin{aligned}\mathcal{L}_{con} &= ||\mathbf{Z}_i - \mathbf{Z}_j||_2 \\ &= ||nDNCM(\mathbf{I}_i,\, \mathbf{d}_i) - nDNCM(\mathbf{I}_j,\, \mathbf{d}_j)||_2.\end{aligned} \tag{5}$$

The second stage of our pipeline aims to stylize the normalized images. To convert the input images to new styles, we swap the predicted *sDNCM* parameters $\mathbf{r}$ to stylize the two samples, *i.e.*, $\mathbf{Z}_i$ will be stylized by $\mathbf{r}_j$ while $\mathbf{Z}_j$ by $\mathbf{r}_i$, as:

$$\mathbf{Y}_i = sDNCM(\mathbf{Z}_j,\, \mathbf{r}_i), \quad \mathbf{Y}_j = sDNCM(\mathbf{Z}_i,\, \mathbf{r}_j). \tag{6}$$

We apply a L1 reconstruction loss between $\mathbf{I}$ and $\mathbf{Y}$ to learn color style transfer, as:

$$\mathcal{L}_{rec} = ||\mathbf{Y}_i - \mathbf{I}_i||_1 + ||\mathbf{Y}_j - \mathbf{I}_j||_1. \tag{7}$$

The final loss is a combination of Eq. 5 and 7, as:

$$\mathcal{L} = \mathcal{L}_{rec} + \lambda\, \mathcal{L}_{con}, \tag{8}$$

where λ is a controllable weight. Refer to the analysis in Supplemental A on how to derive our training constraints from the fundamental color style transfer objective.

4. Experiments

In this section, we first introduce our experimental setting, including datasets, implementation, and quantitative metrics. We then extensively compare Neural Preset with existing color style transfer methods (Sec. 4.1). We further analyze the components and hyper-parameters of Neural Preset through ablation experiments (Sec. 4.2).

Datasets. Following recent color style transfer methods [1, 34, 54], we train our model on the images from the MS COCO [35] dataset. We use about 5,000 LUT files, along with the random image filter adjustment strategy [27], as input color perturbations during training. We collect 50 images with diverse color styles and pair each two of them to build a validation set consisting of 2,500 samples.

Implementation. We adopt EfficientNet-B0 [47] as the encoder E in Neural Preset. We fix the input size of E to 256×256. We set the parameter dimension k in DNCM to 16, so the number of parameters predicted by E is only 256 for each image. We train Neural Preset by the Adam [28] optimizer for 32 epochs. With a batch size of 24, the initial learning rate is $3e^{-4}$ and is multiplied by 0.1 after 24 epochs. We set the loss weight λ to 10.

Quantitative Metrics. We follow prior works [1, 50, 54] to quantitatively evaluate color style transfer quality in terms of style similarity (between the output and the reference style image) and content similarity (between the output and the input image). However, we observe from our experiments that the metrics used by prior works cannot reflect color style transfer quality precisely. Therefore, we propose improved style/content similarity measures as our quantitative metrics. For style similarity measure, prior works use the VGG [45] features learned from ImageNet [8] to compute the Gram metric. Since the VGG features are not designed for comparing color styles and contain semantic information, the style similarity that they produce may not be accurate due to semantic bias. Instead, we train a discriminator [14] model on an annotated dataset (containing 700+ color style categories, each with 6-10 images of the same color style retouched by human experts) to accurately predict the color style similarity score (in $[0, 1]$) of two images. For content similarity measure, prior works compute the SSIM metric based on image edges extracted by an edge detection model HED [51]. However, HED only predicts rough edges and is often incorrect. Hence, we replace HED with a recent model LDC [46], which can output fine and correct edges for SSIM calculation, providing a reliable content similarity score (in $[0, 1]$). Refer to Supplemental B for more details on our improved metrics.

4.1. Comparisons

We compare Neural Preset with deep learning based methods (PhotoWCT [34], WCT^2 [54], PhotoNAS [1], PhotoWCT^2 [6], and Deep Preset [19]) as well as a traditional method (CT [43]). We evaluate the pre-trained models released by their authors. We do not compare with methods whose codes, models, and demos are all unavailable.

Qualitative Results. Fig. 5 shows the superiority of our qualitative results. First, Neural Preset produces more natural stylized images (*e.g.*, the colors of the car and wall in Fig. 5 (a)). Second, Neural Preset can preserve fine textures with target color styles (*e.g.*, the enlarged text in Fig. 5 (b)). Third, Neural Preset is better at maintaining the inherent object colors (*e.g.*, the human hair and clothing regions in Fig. 5 (c)). Fourth, the outputs of Neural Preset have more consistent color properties with the style images (*e.g.*, the brightness and contrast in Fig. 5 (d)). The results of PhotoWCT are omitted in Fig. 5 as its visual results are similar to PhotoWCT^2. Refer to Supplemental C.1 for more visual results of Neural Preset. Refer to Supplemental C.2 for video stylization results of Neural Preset.

Quantitative Results. Fig. 6 shows that our Neural Preset comes closest to the "Ideal" stylization quality. Although PhotoWCT^2 has high style similarity scores in Fig. 6, we can observe from Fig. 5 that it tends to overfit the color styles of the reference images, which however leads to worse visual results. Deep Preset has high content similarity scores in Fig. 6 since it often fails to alter the color style of input images, as shown in Fig. 5. Besides, both scores for CT are low as it sometimes produces very erratic results.

User Study. We further conduct a user study to evaluate

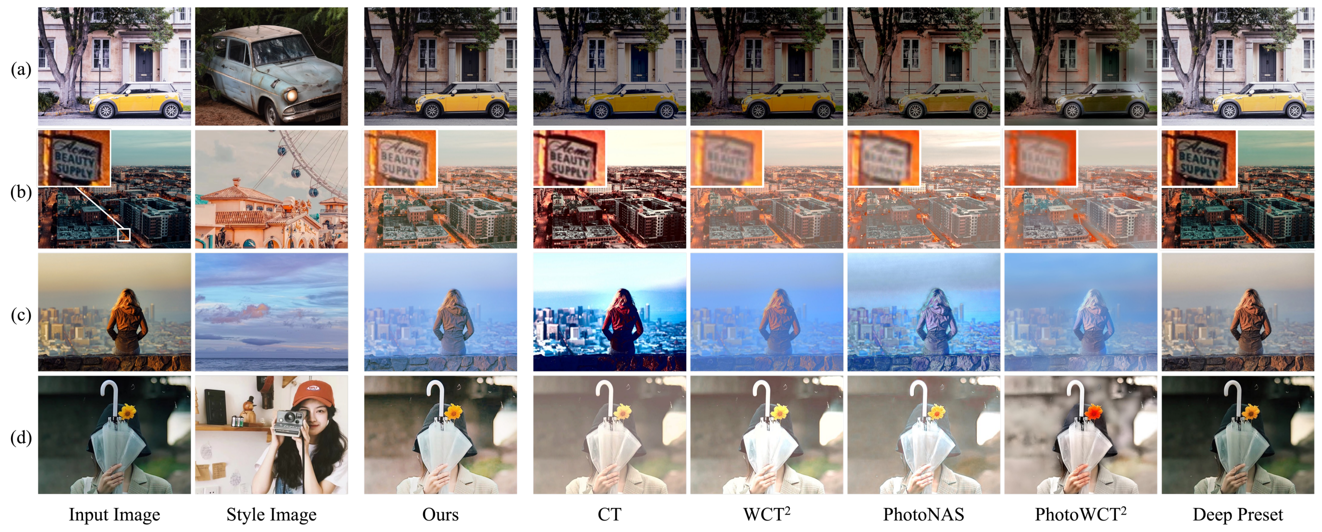


Figure 5. **Qualitative Comparison.** Our method has advantages in (a) producing natural stylized images, (b) preserving image textures, (c) maintaining object inherent colors, and (d) providing color properties more consistent with the style images.

Method	CT [43]	PhotoWCT [34]	WCT^2 [54]	PhotoNAS [1]	PhotoWCT^2 [6]	Deep Preset [19]	Ours
Average Ranking ↓	4.97	5.75	2.67	3.39	4.11	5.30	**1.81**

Table 1. **Average Ranking of Different Methods in the User Study.** The lower the number, the better the human subjective evaluation.

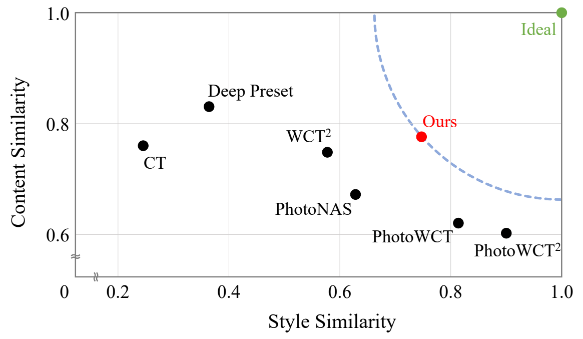


Figure 6. **Quantitative Comparison.** The higher the content similarity and the style similarity, the better the stylization quality. "Ideal" refers to the best possible quality. The points on the blue dashed curve have equal distance from "Ideal" as our method, *i.e.*, have equivalent color style transfer quality as our method.

the subjective quality of different methods. We invite 58 users and show them 20 image sets randomly selected from our validation set, with each image set consisting of an input image, a reference style image, and 7 randomly shuffled color style transfer results. For each image set, the users are required to rank the overall stylization quality of the 7 results by considering the style/content similarity and the photorealism of the results as well as whether the color style of the results is visually pleasing. After collected 1,160 (58×20) results, we compute the average ranking of each method. Table 1 shows that results from our method are largely preferred by the users. Note that the second-ranked WCT^2 can only handle images of FHD resolution (see Table 2). We provide Top1-Top3 ratios in Supplemental C.4.

Inference Efficiency and Model Size. As shown in Table 2, Neural Preset achieves nearly 28× speedup compared to the fastest state-of-the-art method (*i.e.*, PhotoWCT^2 [6]) on 2K images. Neural Preset also enables real-time inference (about 52 fps) at 4K resolution, and can handle 8K resolution images at over 16 fps. Refer to Supplemental C.6 for the inference time on CPU. Table 2 also shows that existing methods require large amounts of memory for inference. Using even a GPU with 24GB memory, many of them still have the out-of-memory problem at 4K resolution, and all of them fail at 8K resolution. In contrast, Neural Preset requires only 1.96GB of memory, irrespective of the image resolution. This is because *nDNCM*/*sDNCM* in Neural Preset operate on each pixel independently, allowing us to save memory via splitting high-resolution images into small patches for processing. Besides, Neural Preset also has the lowest number of parameters.

Comparing Neural Preset with Filters and Luts. After manually retouching an image by a photo editing tool (*e.g.*, Lightroom), we export the editing parameters as a preset in the filters/LUTs format to process a set of images automatically. Meanwhile, we transfer the color style of the re-

Method	GPU Inference Time ↓ / Memory ↓				Model Size ↓
	FHD (1920 × 1080)	2K (2560 × 1440)	4K (3840 × 2160)	8K (7680 × 4320)	Number of Parameters
PhotoWCT [34]	0.599 s / 10.00 GB	1.002 s / 16.41 GB	OOM	OOM	8.35 M
WCT^2 [54]	0.557 s / 18.75 GB	OOM	OOM	OOM	10.12 M
PhotoNAS [1]	0.580 s / 15.60 GB	0.988 s / 23.87 GB	OOM	OOM	40.24 M
Deep Preset [19]	0.344 s / 8.81 GB	0.459 s / 13.21 GB	1.128 s / 22.68 GB	OOM	267.77 M
PhotoWCT^2 [6]	0.291 s / 14.09 GB	0.447 s / 19.75 GB	1.036 s / 23.79 GB	OOM	7.05 M
Ours	**0.013 s / 1.96 GB**	**0.016 s / 1.96 GB**	**0.019 s / 1.96 GB**	**0.061 s / 1.96 GB**	**5.15 M**

Table 2. **Comparison on GPU Inference Time / Memory, and Model Size.** All evaluations are conducted with Float32 model precision on a Nvidia RTX3090 GPU (24GB memory). The values in parentheses under the resolutions are the exact image width and height. The units “s”, “GB”, and “M” mean seconds, gigabytes, and millions, respectively. “OOM” means having the out-of-memory issue.

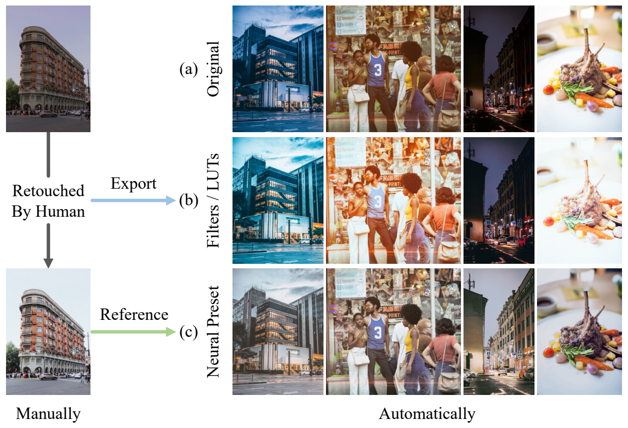


Figure 7. **Comparing Neural Preset with Filters/LUTs.** (c) The results of Neural Preset have more consistent color styles than (b) the results of filters/LUTs.

touched image to other images via Neural Preset. As shown in Fig. 7, filters/LUTs fail to convert images with diverse color styles to a consistent color style (Fig. 7 (b)). For example, after applying filters/LUTs, bright images will be over-exposed while dark images still remain dark. In contrast, with the color style parameters extracted from the retouched image, Neural Preset provides results with more consistent color styles (Fig. 7 (c)). Refer to Supplemental C.3 for more results of applying the same color style to different input images via Neural Preset.

4.2. Ablation Studies

Input Patch Size for DNCM. The input patch size used for DNCM can affect the inference time and memory footprint of Neural Preset. Intuitively, processing a large number of pixels in parallel (*i.e.*, using a larger patch size) will give a lower inference time but at the cost of a higher memory consumption. From Table 3, setting a patch size > 512 brings a limited speed improvement on a Nvidia RTX3090 GPU but a significant increase in memory usage. This is because the 512 patch size already uses up all GPU cores.

At 8K Resolution (7680 × 4320)		
Patch Size	Total Patches	GPU Inference Time ↓ / Memory ↓
256	507	0.1026 s / **1.83 GB**
512	127	0.0613 s / 1.96 GB
1024	32	0.0590 s / 2.14 GB
2048	8	0.0582 s / 2.79 GB
4096	2	0.0575 s / 5.38 GB
8092	1	**0.0569 s** / 8.77 GB

Table 3. **Effects of Different DNCM Input Patch Sizes.** We validate the inference time and memory cost of Neural Preset with different DNCM input patch sizes on 8K images. In Table 2, we use a patch size of 512 (the underlined row).

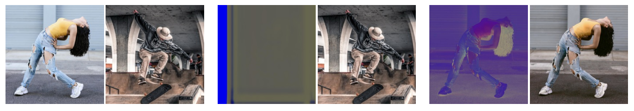


Figure 8. **Ablation of DNCM in Our Pipeline.** Building our two-stage pipeline via two CNNs, *e.g.*, two autoencoders [44], causes our SSL training to converge to a trivial solution, where the first stage can be any function, and the second stage is an identity function *w.r.t.* the style image (see (b)). Applying DNCM can help avoid such a trivial solution (see (c)). Symbols are from Fig. 3.

DNCM *vs.* CNN Color Mapping. We experiment with constructing the proposed two-stage color style transfer pipeline using two CNNs, *e.g.*, two autoencoders [44], with all other settings unchanged. The results in Fig. 8 visualize our analysis in Supplemental A: using DNCM instead of CNN for color mapping prevents our self-supervised training strategy from converging to a trivial solution.

Parameter Dimension k in DNCM. Table 4 shows that setting $k < 16$ significantly decreases style similarity but has small effect on content similarity, while setting $k > 16$

k	2	4	8	<u>16</u>	32
Style Similarity ↑	0.128	0.510	0.636	<u>0.746</u>	**0.769**
Content Similarity ↑	0.765	**0.823**	0.781	<u>0.771</u>	0.764

Table 4. **Results of Neural Preset with Different Values of k.** The hyper-parameter k has a huge impact on style similarity. Other figures and tables are based on $k = 16$ (the underlined column).

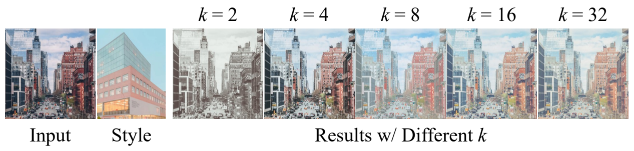


Figure 9. **Results of Neural Preset with Different Values of k.** These visual results are consistent with those in Table 4.

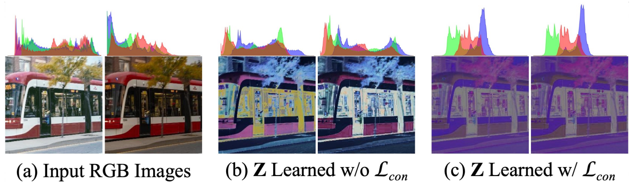

(a) Input RGB Images
(b) **Z** Learned w/o $\mathcal{L}_{con}$
(c) **Z** Learned w/ $\mathcal{L}_{con}$

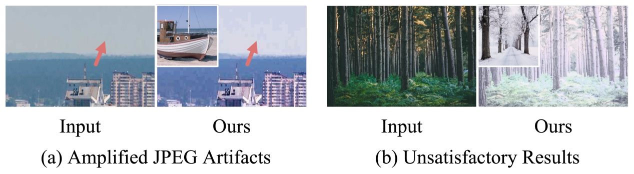
Figure 10. **Impact of $\mathcal{L}_{con}$.** We show the RGB color histogram of each image at the top. Applying $\mathcal{L}_{con}$ can provide a more consistent **Z**, *i.e.*, a better representation of the "image content", for two images with the same content but different color styles.

leads to better performances but also requires longer inference time. Hence, we use $k = 16$, which provides a good balance of speed and performance. Fig. 9 displays the results with different values of k, demonstrating that k also has a huge impact on qualitative results.

Effectiveness of $\mathcal{L}_{con}$. We visualize the "image content" output by the first stage of our pipeline in Fig. 10. Applying $\mathcal{L}_{con}$ provides a more consistent **Z** to represent the "image content". Besides, our experiments also show that learning Neural Preset with $\mathcal{L}_{con}$ can yield better results, *i.e.*, $\mathcal{L}_{con}$ helps the pipeline converge better.

5. Applications

With the proposed self-supervised strategy, Neural Preset is able to learn general knowledge of color style transfer from large-scale data. As a result, given a reference image, our trained model can be naturally applied to other color mapping tasks without fine-tuning. We evaluate four tasks: low-light image enhancement [30], underwater image correction [52], image dehazing [16], and image harmonization [37]. These tasks are challenging for universal color style transfer models as the input images typically come from highly degraded domains. The qualitative results in Supplemental C.5 show that Neural Preset significantly outperforms prior color style transfer methods on all four tasks.



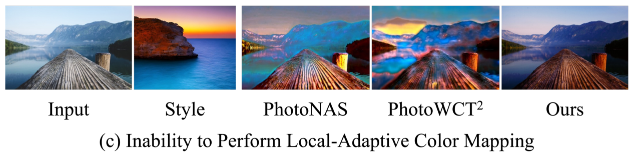


Figure 11. **Limitations of Neural Preset.** For (a) and (b), the top-left corner shows the style image. (a) JPEG artifacts may be amplified (see red arrows) if the input image has a high compression ratio. (b) Results may be unsatisfactory if some colors in the input image (*e.g.*, green color) do not exist in the style image. (c) Our method fails to map blue sky/water in an image to different colors separately. Although prior methods may handle blue sky/water separately, they typically cause heavy artifacts.

We notice that our DNCM effectively avoids heavy visual artifacts produced by other methods, and our color normalization stage successfully decouples color styles from degraded input images.

Besides, training DNCM (Fig. 2) for color mapping tasks using pairwise data is straightforward. We experiment with DNCM on the pairwise datasets of image harmonization [7, 27, 48] and image color enhancement [12, 23, 55]. Without whistles and bells, we get top-level performances on both tasks. Refer to Supplemental D for details and results.

6. Conclusion

In this paper, we have presented a simple but effective Neural Preset technique for color style transfer. Benefited by the proposed DNCM and two-stage pipeline, Neural Preset has shown significant improvements over existing state-of-the-art methods in various aspects. In addition, we have also explored several applications of our method. For example, directly applying Neural Preset to other color mapping tasks without fine-tuning.

Nonetheless, Neural Preset does have limitations. First, if the input is compressed by JPEG with a high compression ratio, the existing JPEG artifacts may be amplified in the output (Fig.11 (a)). Second, it may fail to transfer color styles between images with very different inherent colors (Fig.11 (b)). Third, it cannot perform local-adaptive color mapping to transfer the same color in an image to different colors (Fig.11 (c)). A possible future work is to address these limitations. For example, developing auxiliary regularization to alleviate the effects of JPEG artifacts or incorporating appropriate user interactions for complex cases that involve changing image inherent colors.

References

[1] Jie An, Haoyi Xiong, Jun Huan, and Jiebo Luo. Ultrafast photorealistic style transfer via neural architecture search. In *AAAI*, 2020. 1, 2, 5, 6, 7

[2] Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua. Stylebank: An explicit representation for neural image style transfer. In *CVPR*, 2017. 2, 3

[3] Mark Chen, Alec Radford, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *ICML*, 2020. 3

[4] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020. 3

[5] Jiaxin Cheng, Ayush Jaiswal, Yue Wu, Pradeep Natarajan, and Prem Natarajan. Style-aware normalized loss for improving arbitrary style transfer. In *CVPR*, 2021. 2

[6] Tai-Yin Chiu and Danna Gurari. Photowct2: Compact autoencoder for photorealistic style transfer resulting from blockwise training and skip connections of high-frequency residuals. In *WACV*, 2022. 1, 2, 3, 5, 6, 7

[7] Wenyan Cong, Xinhao Tao, Li Niu, Jing Liang, Xuesong Gao, Qihao Sun, and Liqing Zhang. High-resolution image harmonization via collaborative dual transformations. In *CVPR*, 2022. 2, 8

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

[9] Yingying Deng, Fan Tang, Weiming Dong, Chongyang Ma, Xingjia Pan, Lei Wang, and Changsheng Xu. Stytr^2: Image style transfer with transformers. In *CVPR*, 2022. 2

[10] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *ICLR*, 2017. 2

[11] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016. 2, 3

[12] Michaël Gharbi, Jiawen Chen, Jonathan T Barron, Samuel W Hasinoff, and Frédo Durand. Deep bilateral learning for real-time image enhancement. *TOG*, 2017. 2, 8

[13] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *ICLR*, 2018. 3

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014. 5

[15] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent - a new approach to self-supervised learning. In *NeurIPS*, 2020. 3

[16] Jie Gui, Xiaofeng Cong, Yuan Cao, Wenqi Ren, Jun Zhang, Jing Zhang, and Dacheng Tao. A comprehensive survey on image dehazing based on deep learning. In *IJCAI*, 2021. 2, 8

[17] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022. 3

[18] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020. 3

[19] Man M. Ho and Jinjia Zhou. Deep preset: Blending and retouching photos with color style transfer. In *WACV*, 2021. 2, 3, 5, 6, 7

[20] Kibeom Hong, Seogkyu Jeon, Huan Yang, Jianlong Fu, and Hyeran Byun. Domain-aware universal style transfer. In *ICCV*, 2021. 2

[21] Fei Hou, Anqi Pang, Chiyu Wang, and Wencheng Wang. Domain enhanced arbitrary image style transfer via contrastive learning. In *SIGGRAPH*, 2022. 2

[22] Lukas Hoyer, Dengxin Dai, Yuhua Chen, Adrian Köring, Suman Saha, and Luc Van Gool. Three ways to improve semantic segmentation with self-supervised depth estimation. In *CVPR*, 2021. 3

[23] Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. Exposure: A white-box photo post-processing framework. *TOG*, 2018. 2, 8

[24] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017. 2, 3

[25] Yifan Jiang, He Zhang, Jianming Zhang, Yilin Wang, Zhe Lin, Kalyan Sunkavalli, Simon Chen, Sohrab Amirghodsi, Sarah Kong, and Zhangyang Wang. Ssh: A self-supervised framework for image harmonization. In *ICCV*, 2021. 3

[26] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 2

[27] Zhanghan Ke, Chunyi Sun, Lei Zhu, Ke Xu, and Rynson W.H. Lau. Harmonizer: Learning to perform white-box image and video harmonization. In *ECCV*, 2022. 2, 5, 8

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, 2015. 5

[29] Samuli Laine, Tero Karras, Jaakko Lehtinen, and Timo Aila. High-quality self-supervised deep image denoising. In *NeurIPS*, 2019. 3

[30] Chongyi Li, Chunle Guo, Ling-Hao Han, Jun Jiang, Ming-Ming Cheng, Jinwei Gu, and Chen Change Loy. Low-light image and video enhancement using deep learning: A survey. *IEEE TPAMI*, 2021. 2, 8

[31] Chuan Li and Michael Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *CVPR*, 2016. 2

[32] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming Hsuan Yang. Diversified texture synthesis with feed-forward networks. In *CVPR*, 2017. 2

[33] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *NeurIPS*, 2017. 3

[34] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In *ECCV*, 2018. 2, 3, 5, 6, 7

[35] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollaŕ, and C Lawrence Zitnick. Microsoft coco: Common objects in context. in european conference on computer vision. In *ECCV*, 2014. 5

[36] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *CVPR*, 2017. 2, 3

[37] Li Niu, Wenyan Cong, Liu Liu, Yan Hong, Bo Zhang, Jing Liang, and Liqing Zhang. Making images real again: A comprehensive survey on deep image composition. *Preprint*, 2021. 2, 8

[38] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, 2016. 3

[39] Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning features by watching objects move. In *CVPR*, 2017. 3

[40] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016. 3

[41] François Pitié, Anil C. Kokaram, and Rozenn Dahyot. N-dimensional probability density function transfer and its application to color transfer. In *ICCV*, 2005. 1, 2

[42] François Pitié, Anil C. Kokaram, and Rozenn Dahyot. Automated colour grading using colour distribution transfer. In *CVIU*, 2007. 1, 2

[43] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE CGA*, 2001. 1, 2, 5, 6

[44] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 7

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 5

[46] Xavier Soria, Gonzalo Pomboza-Junez, and Angel Domingo Sappa. Ldc: Lightweight dense cnn for edge detection. *IEEE Access*, 2022. 5

[47] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 5

[48] Yi-Hsuan Tsai, Xiaohui Shen, Zhe Lin, Kalyan Sunkavalli, Xin Lu, and Ming-Hsuan Yang. Deep image harmonization. In *CVPR*, 2017. 8

[49] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. *TOG*, 2002. 1, 2

[50] Xide Xia, Meng Zhang, Tianfan Xue, Zheng Sun, Hui Fang, Brian Kulis, and Jiawen Chen. Joint bilateral learning for real-time universal photorealistic style transfer. In *ECCV*, 2020. 2, 5

[51] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015. 5

[52] Miao Yang, Jintong Hu, Chongyi Li, Gustavo Rohde, Yixiang Du, and Ke Hu. An in-depth survey of underwater image enhancement and restoration. *IEEE Access*, 2019. 2, 8

[53] Jonghwa Yim, Jisung Yoo, Won-joon Do, Beomsu Kim, and Jihwan Choe. Filter style transfer between photos. In *ECCV*, 2020. 3

[54] Jaejun Yoo, Youngjung Uh, Sanghyuk Chun, Byeongkyu Kang, and Jung-Woo Ha. Photorealistic style transfer via wavelet transforms. In *ICCV*, 2019. 2, 3, 5, 6, 7

[55] Hui Zeng, Jianrui Cai, Lida Li, Zisheng Cao, and Lei Zhang. Learning image-adaptive 3d lookup tables for high performance photo enhancement in real-time. *IEEE TPAMI*, 2020. 2, 8