PaletteNet: Image Recolorization with Given Color Palette

Junho Cho, Sangdoo Yun, Kyoungmu Lee, Jin Young Choi
ASRI, Dept. of Electrical and Computer Eng., Seoul National University
{junhocho, yunsd101, kyoungmu, jychoi}@snu.ac.kr

Abstract

Image recolorization enhances the visual perception of an image for design and artistic purposes. In this work, we present a deep neural network, referred to as PaletteNet, which recolors an image according to a given target color palette that is useful to express the color concept of an image. PaletteNet takes two inputs: a source image to be recolored and a target palette. PaletteNet is then designed to change the color concept of a source image so that the palette of the output image is close to the target palette. To train PaletteNet, the proposed multi-task loss is composed of Euclidean loss and adversarial loss. The experimental results show that the proposed method outperforms the existing recolorization methods. Human experts with a commercial software take on average 18 minutes to recolor an image, while PaletteNet automatically recolors plausible results in less than a second.

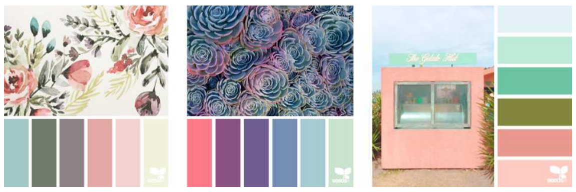


Figure 1. **The Images and the Corresponding Palettes.** The palettes express color concept of the images. Collected from *Designseeds.com* [1]

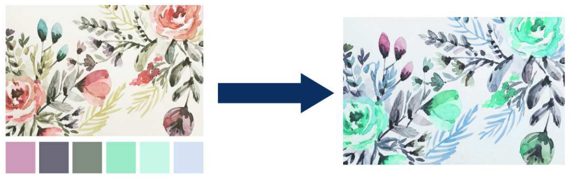


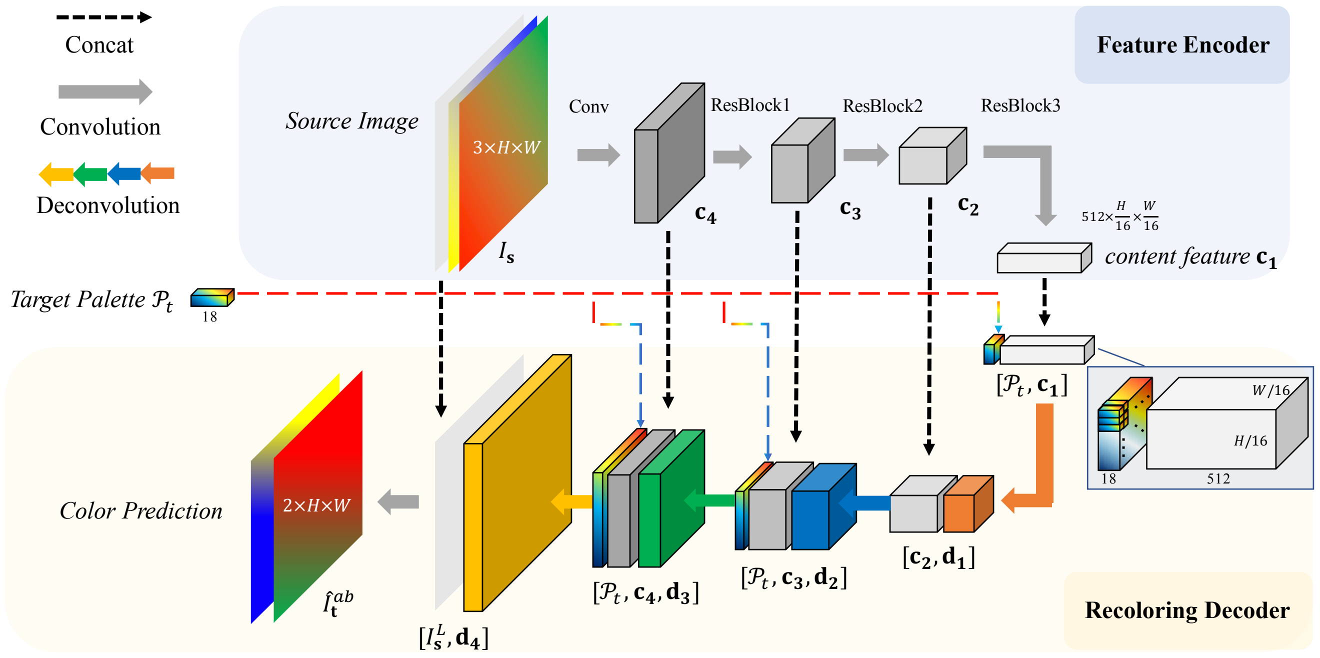
Figure 2. **Our Conceptual Recoloring Model.** From a pair of a source image and a target palette, the resulted image is recolored according to the color concept of the target palette.

1. Introduction

Color is an essential element of humans' visual perceptions of their daily lives. Beautiful color harmony in artworks or movies fulfills our desires for color. Thus, designers and artists must put effort into building basic color concepts into their works. A sophisticated selection of color gives a sense of stability, unity, and identity to works. In general, designers express a color concept through a color palette. The color palette of an image represents the color concept of an image with six colors ordered as shown in Figure 1. The corresponding color palette that contains distinctive color concept is subjective, and the number of palettes is uncountable. Typical designers would carefully select a color concept by the palette prior to the work. Furthermore, recoloring an image with a target color palette is preferred for images to maintain uniformity and identity among artworks. Thus, the recolorization problem occupies a critical position in enhancing the visual understanding of viewers.

Researchers have been tackling the recolorization problem with various approaches and purposes. Kuhn *et al.* [9] proposed a practical way to enhance visibility for the colorblind (dichromat) by exaggerating color contrast. However, it ignored the color concept and lacked aesthetics. Casaca *et al.* [2] proposed a colorization algorithm that requires segmentation masks and user's hints for the colors of some pixels. Even though the colorization based on the color hints was considered the desired color for each pixel, the algorithms were far from automatic colorization.

To reflect the intended color concept, the color palette-based methods [5, 3] have been proposed. Greenfield *et al.* [5] proposed a color association method using palettes. It extracted the color palettes of the source and target images and recolored the source image by associating the palettes in the color space. Chang *et al.* [3] proposed a color transferring algorithm using the relationship between the palettes of the source and target images. This approach helped users to have elaborate control over the intended color concept. However, it is questionable how well the color transform function [5, 3] in the palette space could be utilized for the content-aware recolorization. For example, flowers look



Concat
Convolution
Deconvolution
Feature Encoder
Source Image
3×H×W
I_s
Conv
ResBlock1
ResBlock2
ResBlock3
$\mathbf{c_4}$
$\mathbf{c_3}$
$\mathbf{c_2}$
$512\times\frac{H}{16}\times\frac{W}{16}$
content feature $\mathbf{c_1}$
Target Palette $\mathcal{P}_t$
18
$[\mathcal{P}_t, \mathbf{c_1}]$
W/16
H/16
18
512
Color Prediction
2×H×W
$\hat{I}_t^{ab}$
$[I_s^L, \mathbf{d_4}]$
$[\mathcal{P}_t, \mathbf{c_4}, \mathbf{d_3}]$
$[\mathcal{P}_t, \mathbf{c_3}, \mathbf{d_2}]$
$[\mathbf{c_2}, \mathbf{d_1}]$
Recoloring Decoder

Figure 3. **The Proposed Framework.** PaletteNet has two subnets: Feature encoder network which extracts the content feature from the source image and recoloring decoder network which decodes the content feature and the target palette into the recolored output.

more complicated than the sky. Accordingly, the recoloring of flowers necessitates more effort than the recoloring of the sky. Each of the objects has different color characteristics, and the simple palette matching recolorization neglects them. Moreover, performing color transformation globally on images might not be appropriate. For example, we might want the red tulip and the red bird in an image to be recolored separately to a yellow tulip and a green bird. Thus, it is natural to deploy a deep neural network that has strength in understanding the contents (tulips, bird, etc.) of the source image.

In this paper, we propose a deep learning architecture for the content-aware image recoloring based on the given target palette. The proposed deep architecture requires two inputs, which are a source image and a target palette. As described in Figure 2, the output image is a recolored version of the source image with respect to the target palette. In our paper, the color palette contains six of the most representative colors in an artwork. Six is minimal and still representative enough to express analogous, monochromatic, triad, complementary, or compound combinations of colors. Although the spatial dimension of the palette is small, we assume the amount of information in the palette is abundant to express a specific color concept. To obtain a realistic recolorized image with the given palette, we propose an encoder-decoder network and multi-task loss function composed of Euclidean loss and adversarial loss. To gather image and palette pairs to train the proposed network, we scraped the *Design-seeds* website [1] and created a dataset. Since the different color versions of an image do not usually exist, we propose the color augmentation method to expand the dataset for training the deep neural network. The proposed network is trained in an end-to-end and data-driven way. In the experiments, we show our model outperforms the existing recolorization model and produces plausible results in a second, while a human expert takes 18 minutes on average.

2. Structure of PaletteNet

Figure 3 depicts the overall structure of the proposed PaletteNet. PaletteNet includes two subnets: feature encoder (FE) and recoloring decoder (RD). The inputs of PaletteNet are $I_\mathbf{s}$, the source ($\mathbf{s}$) image in LAB and $\mathcal{P}_\mathbf{t}$, the target ($\mathbf{t}$) palette. Target palette $\mathcal{P}_\mathbf{t}$ of PaletteNet is the LAB color value of 18-dimensional vector, defined by the six representative colors. The output of PaletteNet is $\hat{I}_\mathbf{t}^{ab}$, the ab channel image, whose ab (color) is altered from source. Final output $\hat{I}_\mathbf{t}$ is formed by concatenating output of network $\hat{I}_\mathbf{t}^{ab}$ and source image luminance I_s^L. Thus, $I_\mathbf{t}$ has identical spatial size of the source image. In short, PaletteNet changes color channel conditioned to fixed luminance value.

FE in PaletteNet, which is fully convolutional neural network, is responsible to recognize contextual information of I_s and encode objects, texture, color as a content feature $\mathbf{c}$. FE reduces the spatial size of each feature map in half by residual blocks [6]. It also outputs each intermediate hierarchical feature map $\mathbf{c}_i$ as the content feature. With a simple notation,

$$\mathbf{FE}(I_\mathbf{s}) = \mathbf{c} = \{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}\}. \tag{1}$$

In RD of PaletteNet, the target palette $\mathcal{P}_t$ is combined with the content feature $\mathbf{c}$ to perform recolorization. At first, RD takes $\mathbf{c_1}$ and $\mathcal{P}_t$ as an initial input. After repeating $\mathcal{P}_t$ spatially on every pixel of $\mathbf{c_1}$ to match the dimensions, the repeated $\mathcal{P}_t$ and $\mathbf{c_1}$ are concatenated in depth, which denoted as as $[\mathcal{P}_t, \mathbf{c_1}]$. Then deconvolution (Deconv) layer upsamples $[\mathcal{P}_t, \mathbf{c_1}]$ into $\mathbf{d_1}$. The Deconv operations are depicted with colored arrows in Figure 3 and the output of the operation with same color. The following Deconv layers upsample $[\mathbf{c_2}, \mathbf{d_1}]$ into $\mathbf{d_2}$, $[\mathcal{P}_t, \mathbf{c_3}, \mathbf{d_2}]$ into $\mathbf{d_3}$, $[\mathcal{P}_t, \mathbf{c_4}, \mathbf{d_3}]$ into $\mathbf{d_4}$ in the same mechanism. Finally, a convolution layer transforms $[I_\mathbf{s}^L, \mathbf{d_4}]$ into ab color prediction, $\hat{I}_\mathbf{t}^{ab}$. The architecture with skip-connections from FE to RD is similar to U-net [11] which is powerful at segmentation tasks. Because recolorization depends massively on image content, RD uses the hierarchical content feature from FE, which encodes the spatial information of image. Since all the operations are differentiable, FE and RD can be trained jointly for encoding the contents and recoloring with the target palette. For fast convergence and stable learning, the non-linear function of $tanh$ follows after the final convolution layer. Our PaletteNet G can be denoted simply as:

$$G(I_\mathbf{s}, \mathcal{P}_\mathbf{t}) = \mathbf{RD}(\mathbf{FE}(I_\mathbf{s}), \mathcal{P}_\mathbf{t})) = \mathbf{RD}(\mathbf{c}, \mathcal{P}_\mathbf{t}) = \hat{I}_\mathbf{t}^{ab}. \tag{2}$$

Instance Normalization [13] layer follows all the convolution and deconvolution layers in FE and RD. LeakyReLU activation function is applied after the normalization layers.

3. Training of PaletteNet

For training PaletteNet, there must be pairs of an image-palette $(I_j, \mathcal{P}_j)$. We define N pairs of dataset as $\mathbf{D_{orig}} = \{(I_j, \mathcal{P}_j) | j = 1, ..., N\}$. We need a source and a target image-palette pairs which has different color concept of $(I_j, \mathcal{P}_j)$ in order to learn recoloring. Of course, different color version of an image usually does not exist. Therefore we generate more image-palette pairs from $(I_j, \mathcal{P}_j)$ through the proposed color augmentation method. The detailed color augmentation step is explained in Section 4.1. We generate training data tuple $(I_\mathbf{s}, \mathcal{P}_\mathbf{t}, I_\mathbf{t})$ from $(I_j, \mathcal{P}_j)$ through color augmentation. PaletteNet accepts $I_\mathbf{s}$ and $\mathcal{P}_\mathbf{t}$ as inputs, and learns to recolor output $\hat{I}_\mathbf{t}$ with chromaticity of $\mathcal{P}_\mathbf{t}$.

PaletteNet is trained by optimizing two loss functions: Euclidean loss (E-loss, $\mathcal{L}_E$) and Adversarial loss (Adv-loss, $\mathcal{L}_{Adv}$). Training has two phases and is depicted in Figure 4. The first phase is pretraining FE and RD with E-loss. In this process, FE learns how to extract the content feature of

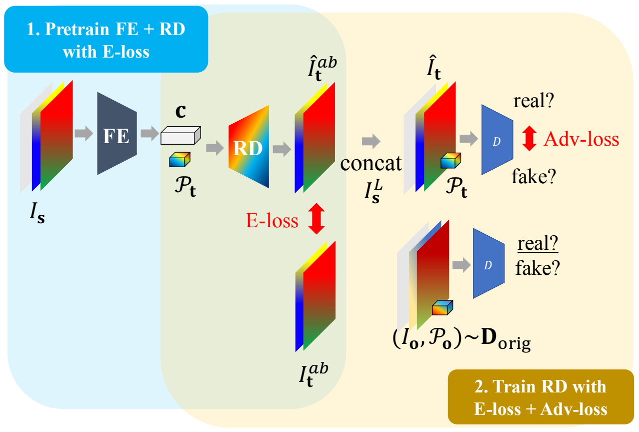


Figure 4. The training PaletteNet involves two phases: 1. Pretrain FE and RD with E-loss (Section 3.1), 2. Freeze the parameters of FE and train RD with additional Adv-loss (Section 3.2). This split training stables learning recolorization process with Adv-loss. See how to compose the training data tuple in Section 3.3.

the image and RD learns how to recolorize with the content feature and the given target palette. E-loss trains G by minimizing pixel-wise distance between $\hat{I}_\mathbf{t}$ and $I_\mathbf{t}$.

However with E-loss, G only learns the color augmented relation between $I_\mathbf{s}$ and $I_\mathbf{t}$. Color augmentation is an essential means of generating different color versions of an image, but not the ultimate function to learn. Therefore in second phase, we introduce additional loss term, Adv-loss to train G to generate more realistic images like $I_j \in \mathbf{D_{orig}}$, instead of learning the color augmented relations. Adv-loss is first proposed from GAN [4], which is a promising framework for generating realistic images. GAN adopts two neural networks, the discriminator network and the generator network. The discriminator network is trained to distinguish natural images and generated image by the generator network. On the other hand, the generator network is trained to produce images that are indistinguishable from natural images by the discriminator network. This competitive training against each other trains the generator network to output realistic images. But if either one of the discriminator network and generator network becomes too powerful, the competitive learning breaks and the other one fails to learn from the powerful opponent. Since our PaletteNet G has lots of parameters, applying GAN framework from the beginning of train happens to be a trouble. Thus as depicted in Figure 4, we pretrain FE and RD enough with E-loss at first phase and adopt Adv-loss at second phase to train RD and the discriminator network D.

3.1. Pretraining of FE and RD with E-loss

With E-loss, update the parameters of G (FE and RD) so that the Euclidean Norm between the output of PaletteNet $G(I_s, \mathcal{P}_\mathbf{t}) = \hat{I}_\mathbf{t}^{ab}$ and desired ab image $I_\mathbf{t}^{ab}$ minimizes. E-loss will be as followed:

$$\mathcal{L}_E = \sum^H \sum^W (\hat{I}_{\mathbf{t}}^{ab} - I_{\mathbf{t}}^{ab})^2, \tag{3}$$

where H, W are height and width of an image and pixel (x, y) is abbreviated. We overlay the source image luminance $I_{\mathbf{s}}^L$ on $\hat{I}_{\mathbf{t}}^{ab}$ and denote the final output LAB image as $\hat{I}_{\mathbf{t}}$.

As E-loss forces G to learn the color augmented relation between $I_{\mathbf{s}}$ and $I_{\mathbf{t}}$, we use it only for pretraining FE and RD. We pretrain G until value of $\mathcal{L}_E$ converges on training set.

3.2. Training of RD with Adv-loss

Our proposed Discriminator Network D accepts an image I and a palette $\mathcal{P}$ and classifies if the pair $(I, \mathcal{P})$ is related. Therefore, the purpose of G is to generate the output $\hat{I}_{\mathbf{t}}$ to have the color concept of $\mathcal{P}_{\mathbf{t}}$. D accepts the pair $(I, \mathcal{P})$ by replicating $\mathcal{P}$ spatially and concatenating in depth to I, which is identical operation $[I, \mathcal{P}]$ explained in RD architecture. D of original GAN [4] views an output of G as *fake*, a sample from target data as *real*. Our D performs binary classification on a pair $(I, \mathcal{P})$ so that $D_{fake}(I, \mathcal{P})$ is the probability of the pair classified as *fake* (unrelated), and $D_{real}(I, \mathcal{P})$ is the probability of the pair classified as *real* (related). The summation of the two probabilities is equal to 1.

In our adversarial network architecture, G and D are optimized to solve the following min-max problem:

$$\begin{aligned}\min_G \max_D \; & \mathbb{E}_{(I_1, \mathcal{P}_1) \sim P_{real}}[\log D_{real}(I_1, \mathcal{P}_1)] \\ & + \mathbb{E}_{(I_2, \mathcal{P}_2) \sim P_{fake}}[\log D_{fake}(I_2, \mathcal{P}_2)],\end{aligned} \tag{4}$$

where $(I_1, \mathcal{P}_1)$ is a *fake* pair and $(I_2, \mathcal{P}_2)$ is a *real* pair. To be more specific, our D views a pair of network generated image and target palette $(\hat{I}_{\mathbf{t}}, \mathcal{P}_{\mathbf{t}})$ as *fake* and a randomly sampled pair $(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}}) \in \mathbf{D_{orig}}$, which are genuine dataset and not color augmented, as *real*:

$$D_{real}(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}}) = 1, \qquad D_{fake}(\hat{I}_{\mathbf{t}}, \mathcal{P}_{\mathbf{t}}) = 1. \tag{5}$$

But practically, the size of $\mathbf{D_{orig}}$ is too small and causes D to cheat by memorizing all the pairs $(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}}) \in \mathbf{D_{orig}}$. As a matter of fact, when G generates recolored $\hat{I}_{\mathbf{t}}$ comparatively well with the color concept of $\mathcal{P}_{\mathbf{t}}$, D barely observes the pair $(I, \mathcal{P})$ having entirely different color concept. Therefore, D finds it very hard to discriminate between $(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}})$ and $(\hat{I}_{\mathbf{t}}, \mathcal{P}_{\mathbf{t}})$, eventually tries to cheat by memorizing $(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}})$. We experimentally observed D performing strikingly well and not easily fooled ever after 1 epoch training D on $\mathbf{D_{orig}}$. Therefore following classification term is added to prevent D from cheating:

$$D_{fake}(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{t}}) = 1, \qquad D_{fake}(\hat{I}_{\mathbf{t}}, \mathcal{P}_{\mathbf{o}}) = 1. \tag{6}$$

The two terms prevent cheating of D by classifying the unrelated pairs as *fake*. They are crucial to induce well balanced training of G and D, no longer causing too powerful D. The classification loss of D is calculated as:

$$\begin{aligned}\mathcal{L}_D = & -\mathbb{E}[\log D_{fake}(\hat{I}_{\mathbf{t}}, \mathcal{P}_{\mathbf{t}})] - \mathbb{E}[\log D_{fake}(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{t}})] \\ & - \mathbb{E}[\log D_{fake}(\hat{I}_{\mathbf{t}}, \mathcal{P}_{\mathbf{o}})] - \mathbb{E}[\log D_{real}(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}})].\end{aligned} \tag{7}$$

And the Adv-loss to train G (specifically RD) is calculated as:

$$\mathcal{L}_{Adv} = -\mathbb{E}[\log D_{real}(\hat{I}_{\mathbf{t}}, \mathcal{P}_{\mathbf{t}})]. \tag{8}$$

Finally, the total loss function of G is the weighted sum of the E-loss and the Adv-loss:

$$\mathcal{L}_G = \lambda \mathcal{L}_E + \mathcal{L}_{Adv}, \tag{9}$$

where λ is a weighting parameter between the two losses which has been set to 10 in our work. We optimize $\mathcal{L}_D$ and $\mathcal{L}_G$ together at each iteration and stop training via validation.

3.3. Training Data Composition

Here, we explain how we prepare the training data tuple $(I_{\mathbf{s}}, \mathcal{P}_{\mathbf{t}}, I_{\mathbf{t}})$ while training FE and RD with E-loss. Initially, we have the original image-palette dataset $\mathbf{D_{orig}} = \{(I_j, \mathcal{P}_j) | j = 1, ..., N\}$. We perform color augmentation on each jth image-palette $(I_j, \mathcal{P}_j)$ pair into N_a number of different image-palette pairs. Then, we denote the augmented image set as $\mathbb{I}_j = \{I_{(j,n)} | n = 1, ..., N_a\}$ and the corresponding augmented palette set as $\mathbb{P}_j = \{\mathcal{P}_{(j,n)} | n = 1, ..., N_a\}$. Within $\mathbb{I}_j, \mathbb{P}_j$, we randomly sample two pairs, the source pair $(I_{\mathbf{s}}, \mathcal{P}_{\mathbf{s}})$ and the target pair $(I_{\mathbf{t}}, \mathcal{P}_{\mathbf{t}})$. A training data tuple is a source image $I_{\mathbf{s}}$, a target palette $\mathcal{P}_{\mathbf{t}}$, and a target image $I_{\mathbf{t}}$. We do not use the source palette $\mathcal{P}_{\mathbf{s}}$ during training unlike the previous palette matching methods [3, 5]. The total number of possible training data tuples $(I_{\mathbf{s}}, \mathcal{P}_{\mathbf{t}}, I_{\mathbf{t}})$ is $N_a \times N_a \times N$. In addition, the source pair and the target pair can be identical. In this case, PaletteNet reconstructs the input image with its palette like Auto-encoder model.

When it comes to training with Adv-loss, we additionally sample $(I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}})$ from $\mathbf{D_{orig}}$. Thus, the training data tuple to train G and D together is $(I_{\mathbf{s}}, \mathcal{P}_{\mathbf{t}}, I_{\mathbf{t}}, I_{\mathbf{o}}, \mathcal{P}_{\mathbf{o}})$. Training also includes random horizontal flip of the images in the probability of 0.5.

4. Experiments

4.1. Data Preparation and Color Augmentation

We generate the dataset using 1,611 image-palette pairs scrapped from the *Design-seeds.com* [1] website. Since we train PaletteNet to change a source image into a target image, we need a target ground truth image, which is

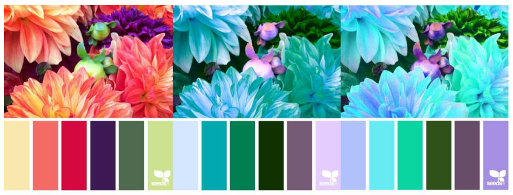


Figure 5. **The Proposed Color Augmentation and Naive Hue-shift method** (a) the original image (b) the result of the proposed color augmentation (c) the result of the naive hue-shift by +180. Compared to (c), (b) alters the color concept only, retaining the luminance. (c) distorts the luminance from the original image.

the different-colored version of the source image. However, a different-colored version of a specific image generally does not exist. Therefore, color augmentation is an essential step to define the input and output of our network. Color augmentation means altering channel-wise pixel values of an image in a certain color space, like HSV, RGB, and LAB. We mainly use hue-shift in the HSV color space. The naive color augmentation shifts the hue value of an image between 1 and 360 in HSV. The problem is that hue-shift causes a luminance distortion to the image. Since HSV does not separate luminance as the characteristics of color, the naive way causes the luminance distortion. Figure 5 (c) clearly shows that the naive hue-shift distorts luminance from original image (a). Thus, we reinforce the naive hue-shift algorithm with the LAB color space, which is known to best express an image's luminance:

$$\begin{aligned}&RGB \rightarrow LAB \text{ and cache } L\\ &RGB \rightarrow HSV \xrightarrow{hue-shift} H^*SV \rightarrow L^*A^*B^* \qquad (10)\\ &\text{Final hue-shifted image: } LA^*B^*.\end{aligned}$$

The above procedure describes the proposed hue-shift algorithm. The main idea is fixing the luminance of an original image during color augmentation. As shown in Figure 5 (b), it successfully alters color concept only while the less luminance distortion occurs than the naive hue-shift algorithm. Fixing luminance is important because we aim only to change the color concept. We assume that the corresponding palette of the hue-shifted image is also hue-shifted by the same amount from the palette of the original image. We augmented each image-palette pair $(I_j, \mathcal{P}_j)$ 18 times (step of 20 in 360) with the proposed color augmentation method. We split 1,611 image-palette pairs into 1,561 as the training set and 50 as the validation set, resulting in 28,098 training pairs and 900 validation pairs. Finally, we resized the images into 288 $\times$ 432 to keep a constant input size for the neural network.

4.2. Training and Architecture Details

We trained networks using NVIDIA GTX TitanX and GTX 1080 GPUs. Because of the image resolution 288 $\times$ 432 is relatively large compared to general image recognition models, we used a small mini-batch size of 12 at GTX TitanX and 8 at GTX 1080 not to exceed GPU memory. We used the Adam optimizer [8] while training G and D. Most of the hyper-parameters are from DCGAN [10]. The learning rate was set to 0.0002 and β_1 as 0.5.

The values of LAB images range: L in [0, 100], a in [-86.185, 98,254], and b in [-107.863, 94.482]. For better input format, we normalized each channel to be the range in [-1, 1] by linear transforms. We used a palettes in LAB and normalized in the same way as above.

The most famous normalization is Batch Normalization [7]. Applying Batch Normalization has been seemed mandatory in recent deep neural network architectures. It helps training the model faster by normalizing a whole mini-batch and acting like a regularizer. However, some of image generation tasks show that alternative normalization, Instance Normalization (also called Contrast Normalization) [13], enhances generated images. It was first proposed in TextureNet [12] and reported enhanced stylization performance, even with a desaturated input images. Instance Normalization performs normalization at each instance of a mini-batch rather than throughout the mini-batch as Batch Normalization. In our recolorization task, we want each instance of mini-batch not interfered by different saturations of the others. We used Instance Normalization as it enhanced our recolorization significantly.

Empirically, the last layer as the convolution was better at generalization on the validation set compared to the deconvolution. Moreover, initializing FE without bias and RD with bias were the best choice through the validation.

Because we aim to recoloring artwork, we set our input size to $H \times W$ of PaletteNet very large as 432×288. We have tested various architectures of D for stable learning. Our final architecture of D is 2-strided 4×4 fully convolutional network of 4 layers. Thus, the output of D is binary heat-map with a spatial size of $H/16 \times W/16$. Instance Normalization and LeakyReLU follow each convolution layer of D.

4.3. Palette Generalization

To evaluate the generalization performance of the proposed method, we tested on the validation images set with the randomly sampled target palettes. If the model is well generalized, the output images are recolored according to the color concept of the any arbitrary target palettes. Figure 6 shows the results of the generalization experiment. Sometimes, the source image is monochromatic, while the target palette is complementary as the first row in Figure 6. Alternatively, the source image is variegated, and the target

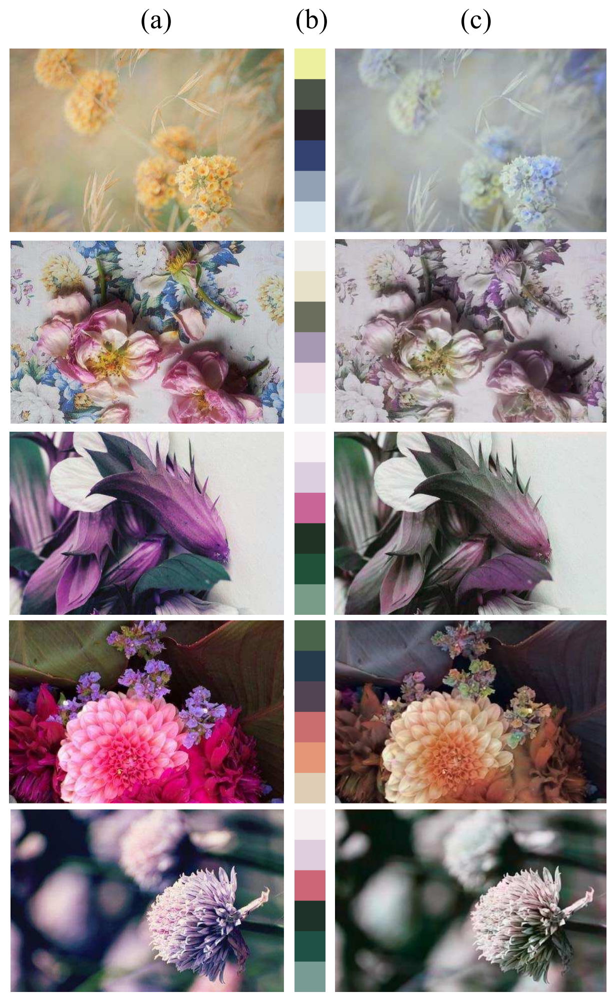


Figure 6. The results of PaletteNet generalization on the randomly sampled image-palette pair. (a) source image (c) target palette (b) resulted output

palette is desaturated as the second row in Figure 6. Different characteristics between a source image and a target palette do not happen while training, because source and target image-palette pairs are color augmented and share similar characteristics of the color distribution. Accordingly, for the given $I_{\mathbf{s}}$, there are only N_a training data tuples $(I_{\mathbf{s}}, \mathcal{P}_{\mathbf{t}}, I_{\mathbf{t}})$. However, our model can generalize even in those random cases as shown in Figure 6 (c). In the first row of the figure, PaletteNet recolors flowers with yellow and blue, while the source image flowers were monochromatic orange. In the second row of the figure, PaletteNet accepts a desaturated palette and recolors the variegated source image in a desaturated fashion. These new generated outputs (c) in Figure 6 cannot be recolored by color augmentation, which proves our proposed model learns the generalized recolorization process with a given palette.

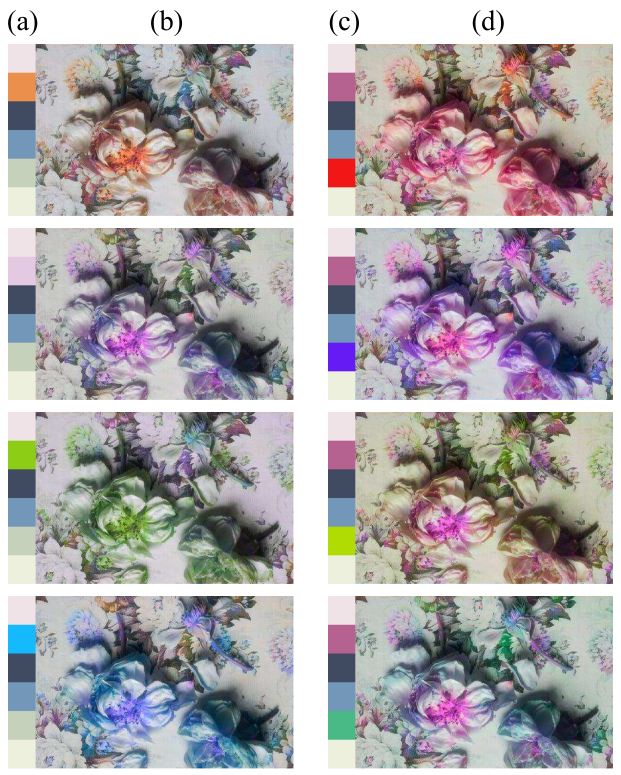


Figure 7. (a) Control on the 2nd color of the target palette and (b) resulted output. (c) Control on the 5th color of the target palette and (d) resulted output.

4.4. Palette Control

PaletteNet does not accept color hints for specific pixel locations like [2]. The model must deduce where to recolor with which color in the target palette. To discover how the colors of palette affect the recoloring process, we conducted an experiment which we control one color while other colors in the palette are fixed on a given image. Figure 7 shows that the second color of the palette presumably colors the overall tone of the flowers, while the fifth color of the palette colors background leaves, while fixing the flower bud in pink. These results also prove that PaletteNet is not learning color augmentation relations but interprets the target palette and reflects on the recolored output.

4.5. Comparisons

In this section, we evaluated PaletteNet by comparing with the previous color transfer function method [3] and the human expert. As shown in Figure 8, (e) by PaletteNet shows more realistic results compared to (d) by [3]. There are several differences between [3] and PaletteNet. First, [3] takes the source palette $\mathcal{P}_{\mathbf{s}}$, which ours does not need. Second, [3] cannot accept a user-favored target palette explicitly, but the user must adjust the source palette interac-

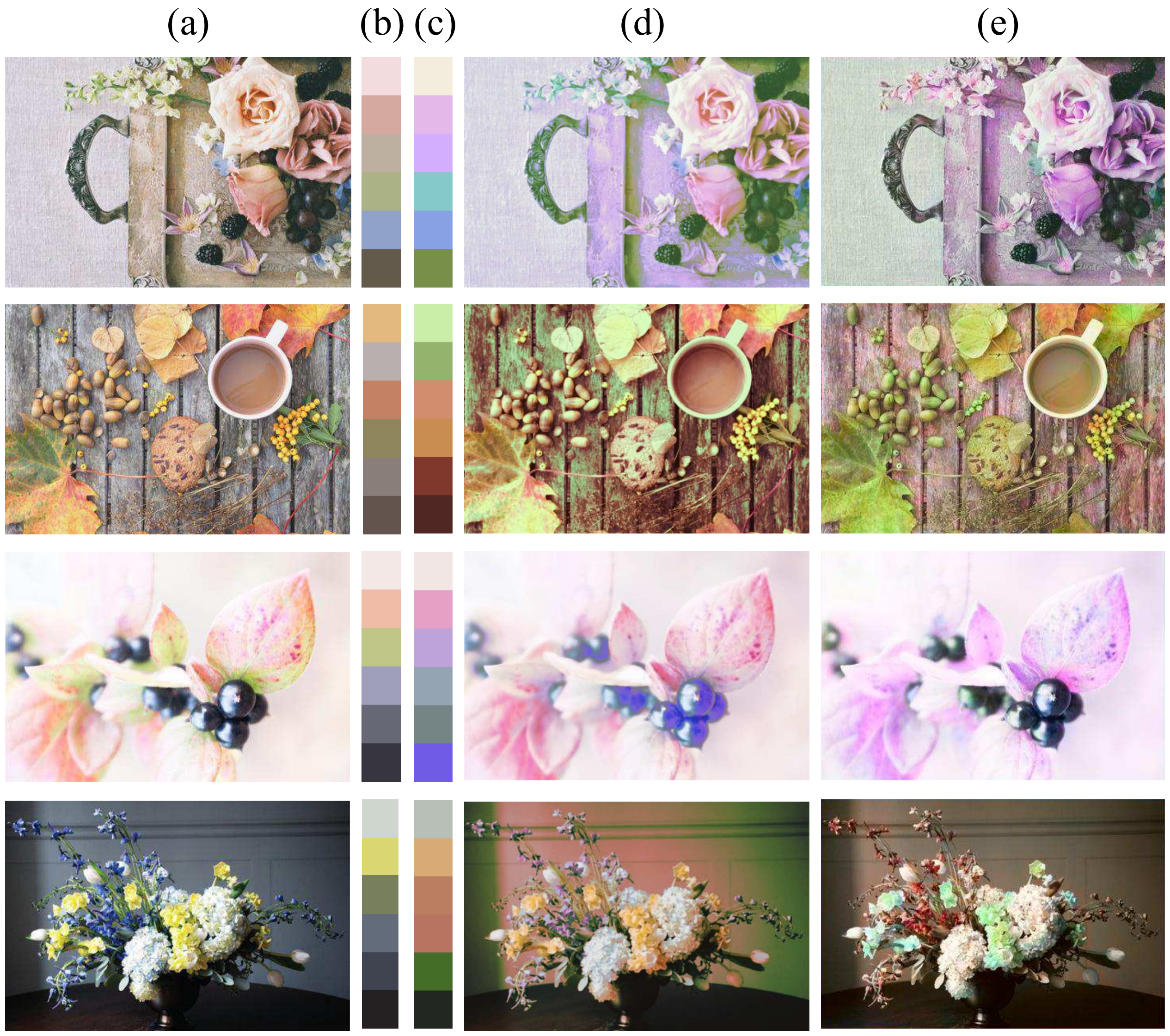


Figure 8. Compare with the existing recolorization model. (a) source image (b) source palette (c) target palette (d) Chang *et al*. [3] (e) Proposed method. Our method do not use the source palette (b). (b) is only used in method (d).

tively through the GUI. Each time an user adjust a color of the palette, the other colors in the palette are altered in response to calculate suitable color transform function. PaletteNet directly accepts any arbitrary target palette. Third, [3] modifies the source image by a transfer function defined by the association of the source palette and the altered palette. Thus, the transfer function acts independently of pixel locations and image context but recolors the source image globally. On the other hand, PaletteNet associates the target palette and the image content feature and infers which pixel to recolor with which color in a data-driven manner.

Because the recolorization setting between [3] and ours differs, in Figure 8, we first applied [3] on the source image (a) by adjusting the source palette (b) to the palette (c), and the corresponding recolored outputs by color the color transfer function are (d). In comparison with our method, we used the pairs of the source image (a) and the palette (c) as the target palette, and the corresponding recolored outputs are (e). Our results (e) look more realistic compared to the results of [3] (d). The color transfer function method is vulnerable to a large-scaled adjustment of the source palette. In addiction, defining the source palette and adjusting it to the target palette require much user efforts and affect recolorization performance. Our method minimizes user effort by explicitly designating the target palette and gains realistic results.

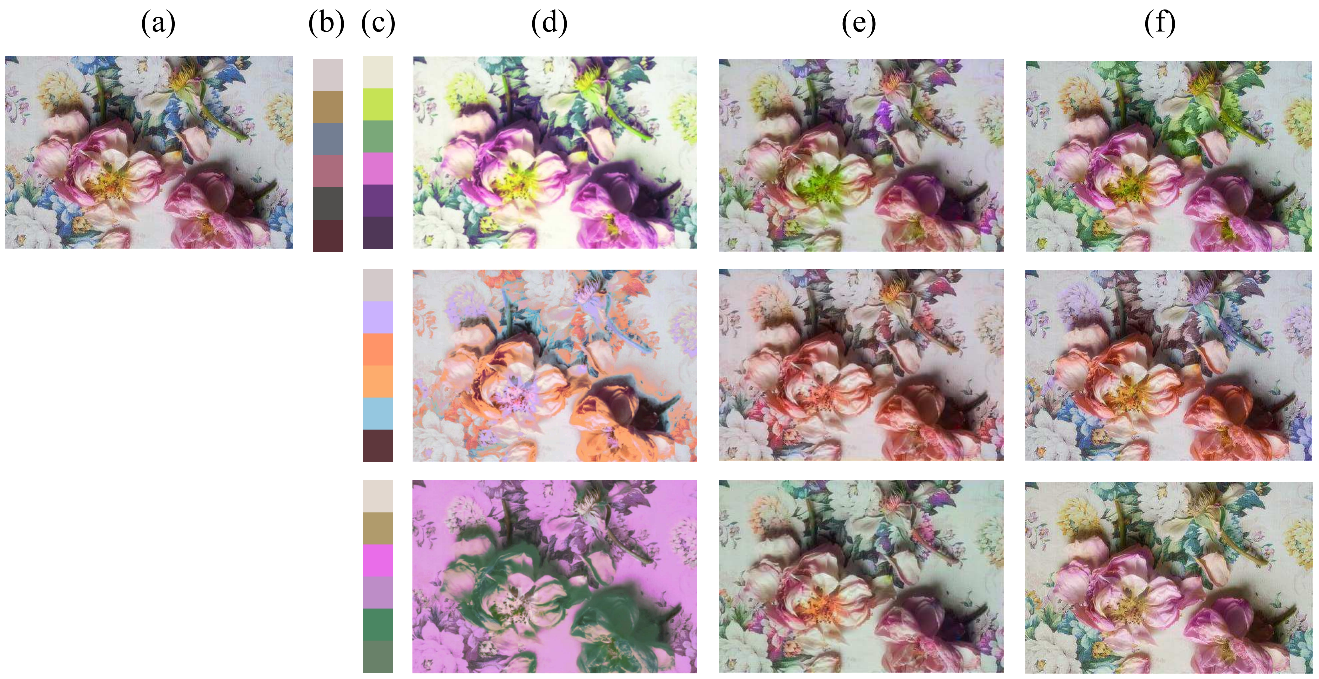


Figure 9. Compare on the extreme target palette on a single image. (a) source image (b) source palette (c) target palette (d) Chang *et al*. [3] (e) Proposed method (f) Human expert with Adobe Photoshop. (b) is only used in (d).

We tested more extreme target palettes on a single image. In this case, professional designers also struggle to adjust an image for an extreme target palette. Recolorization is a common task for designers to post-process color tone of their work. To compare our model against a designer in terms of time and quality, we asked a designer, who is an expert with the commercial software Adobe Photoshop to recolor a source image with a given target palette. We included the Adobe expert's results and compared them with [3] and PaletteNet in Figure 9.

The easiest way to recolor an image using Photoshop is to tweak the RGB color curve until one is satisfied with the color tone of the image. However, tweaking the RGB curve recolors an image globally like color transfer function without considering the image's content. Thus, the human expert additionally segmented objects like flowers, leaves, and background as paths, creating layers for each object path and recoloring them with a brush tool rather than tweaking the color curve. This local content-aware recoloring procedure of the human expert costs lots of time but presents the best recoloring results. PaletteNet performs similar procedure of human expert: extracting the content feature and recoloring with target palette conditioned to the content feature. The human expert's results are in Figure 9 (f). The human expert took each 16, 17, and 20 minutes on each result of 3 rows. Our proposed method in (e) again outperforms color transformation function method (d). PaletteNet produces the plausible results compared to those of human experts, and each takes less than a second.

5. Conclusion

We have proposed PaletteNet that automatically recolors an image with a given target color palette. Contrary to recolorization by the existing method using a color transfer function, PaletteNet extracts the content features and combines them with the target palette to perform content-aware recolorization in a data-driven way. As shown in the experiments, it is practically meaningful that PaletteNet outperforms the existing recolorization method and has an excellent ability comparable to human experts in generating recolored images. Furthermore, PaletteNet could make a realistic and plausible image in less than a second, while a human expert using Adobe Photoshop takes 18 minutes on average for the corresponding recoloring work.

Acknowledgement

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (2014-0-00059, Development of Predictive Visual Intelligence Technology) and the Brain Korea 21 Plus Project. We also appreciate Hannah Park for the expert recolorization works with Adobe Photoshop.

References

[1] Design seeds - for all who love color. `http://design-seeds.com/`. Accessed: 2017-04-24.

[2] W. Casaca, M. Colnago, and L. G. Nonato. Interactive image colorization using laplacian coordinates. In *International Conference on Computer Analysis of Images and Patterns*, pages 675–686. Springer, 2015.

[3] H. Chang, O. Fried, Y. Liu, S. DiVerdi, and A. Finkelstein. Palette-based photo recoloring. *ACM Transactions on Graphics*, 34(4):139:1–139:11, 2015.

[4] I. Goodfellow, J. Pouget-Abadie, and M. Mirza. Generative Adversarial Networks. *arXiv preprint arXiv: ...*, pages 1–9, 2014.

[5] G. R. Greenfield and D. H. House. Image recoloring induced by palette color associations. *Journal of WSCG*, 11(1):189—-196, 2003.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

[9] G. R. Kuhn, M. M. Oliveira, and L. A. Fernandes. An efficient naturalness-preserving image-recoloring method for dichromats. *IEEE transactions on visualization and computer graphics*, 14(6):1747–1754, 2008.

[10] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[11] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[12] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *arXiv preprint arXiv:1603.03417*, 2016.

[13] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.