Day-Night Scene Transfer for Architectural Renderings Based on Color Transfer Approaches

HE Wanyu[i], NIE GuangYang[ii], WANG Chuyu[iii], JACKIE YONG Leong Shong[iv]

Abstract. This paper presents a refined algorithm basing on main color transfer approaches to turn architectural renderings from a daytime scene to a night scene. The aim is to achieve a more accurate day-night scene transfer between architectural images which share some semantically-related content but may vary dramatically in appearance or structure, such as two images of two different buildings in different scenes. For that two key improvements: Feature Normalization and Adding Extra Features techniques are made to the main approaches for color transfer, and the new algorithm implementation scheme is presented to effectively perform day-night transfer for architectural renderings. This method addresses time and cost consuming problems of manual scene transfer for architectural renderings. By testing with various architectural renderings, the proposed method shows good performance in achieving the desired goal.

Key words: Scene Transfer, Color Transfer, Semantic Matching, Deep Learning

1. Introduction

Architects sometimes face tasks to turn a fully rendered daytime scene into a night scene. Normally, setting up night scenes and lighting can be done in some external render engine like Kerkythea or V-Ray, or by resorting to Photoshop. Those ways require a series of manual operations such as removing sharp shadows, darkening the image, lighting etc. which is time consuming and inefficient. In the past one or two years, transfer the color style of a reference image onto a source image has gained tremendous progress due to deep network-based methods, such that automating the process of turning renderings from a daytime scene to a night scene becomes feasible and effective. With the advances of these techniques, computers can help architects altering the color style without changing the original rendering content in order to emulate different illumination, weather conditions, scene materials, or even artistic color effects.

[i] Xkool Tech. Co. LDT. (✉)
B210, Vanke Design Commune, Liu Guang Rd., Nanshan, Shenzhen, China
e-mail: in@xkool.org
[ii] Xkool Tech. Co. LDT.
B210, Vanke Design Commune, Liu Guang Rd., Nanshan, Shenzhen, China
e-mail: nieguangyang@foxmail.com
[iii] Xkool Tech. Co. LDT.
B210, Vanke Design Commune, Liu Guang Rd., Nanshan, Shenzhen, China
e-mail: wcy@xkool.xyz
[iv] Xkool Tech. Co. LDT.
B210, Vanke Design Commune, Liu Guang Rd., Nanshan, Shenzhen, China
e-mail: jackie@falab.org

In fact, color transfer has been a long-standing research direction in the field of machine learning or computer vision, but the transfer results are not quite effective or precise for years due to the key challenging to establish the semantically dense correspondences between two images, which is necessary to achieve accurate color transfer. Traditionally, matching methods depending on hand-crafted features or additional segmentation[1, 2], or user specifications[3] are not quite effective or precise enough. Since deep neural networks was introduced to solve the dense correspondence problem[4, 5], transfer precision was largely improved, but their color transfer results still suffered from artifacts such as ghosting or halo. To refine correspondences and reduce interferential artifacts, He et al. proposed a progressive framework to estimate correspondences in the deep feature domain (extracting features from a pre-trained VGG19[6]) and do color transfer in the image domain, which achieve natural and consistent color transfer effects[7].

Inspired by the work of He et al. we refined the algorithm to perform day-night scene transfer for architectural renderings. Given an image S (source) and another image R (reference), whose main objects are both buildings but the buildings may vary dramatically in appearance or structure, our algorithm builds semantic correspondences between them and then applies color transfer on image S, thereby generating a new image T (target) which preserves the structure and content of S while emulating the color style of R, as shown in Fig. 1.1.

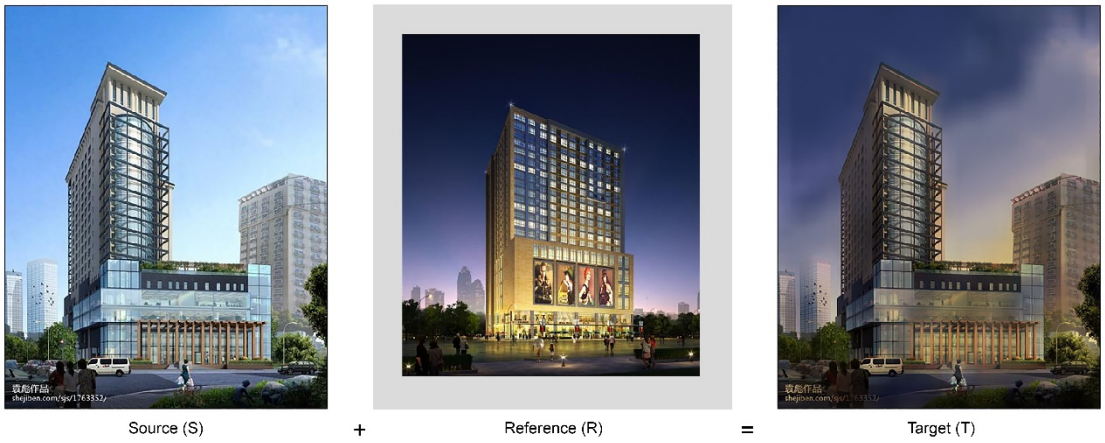

Fig. 1.1 Color transfer example of architectural renderings

In our method, two major adjustments listed below are made:

1) Feature normalization is applied to balance the weight of each channel when using the well-trained VGG19 to extract image features (Section 3.2).
2) Clustering and semantic segmentation are applied to add extra features in order to better build mapping between image S and image R, which improves the accuracy of color transfer (Section 3.3).

The above two techniques strengthen the extraction and matching of similar semantics such that the refined algorithm better fits scene transfer requirements in the architectural field. At the end, by testing with various architectural renderings we show how our techniques can be effectively applied to a variety of real day-night scenes transfer scenarios.

2. Related Work

Color transfer is a rather broad research topic that includes transferring colors to grayscale images or color source images. Early researches focus on colorize grayscale images depending on the user sketching similar regions for computers to put different colors[8]. In recent years, learning-based algorithms appeared to automize grayscale images colorization by learning image statistics (pixel distributions) from large extensive datasets[9, 10], which further inspired approaches for transferring chrominance from one image to another containing similar image statistics[11]. However, apart from chrominance, luminance also has much to do with how an image look. Therefore, He et al. [2018] further proposed a feasible method to transfer both luminance and chrominance between a color image pair. Our focus is to refine and adjust this method to deal with day-night scene transfer for architectural renderings.

2.1 Image Feature Extraction

Feature extraction is a well-studied problem in machine learning. It greatly influences accuracy of object recognition in image processing and better object recognition leads to better semantic correspondence between the processed image pair. If the content of the source image S is well corresponded to the reference image R, for instance, buildings in S corresponding to buildings in R, sky in S corresponding t to sky in R, road in S corresponding to road in R etc., transferring chrominance and luminance from R to S is more accurate and natural.

Many feature extraction techniques have been accumulated for years, such as independent component analysis, kernel PCA, autoencoder etc. Each of them can be a large research topic, thus instead of going deep into this issue, we can apply existing excellent accomplishments. For instance, He et al. tried to apply a pre-trained deep learning model – VGG19 to address the feature extraction issue in order to improve efficiency and accuracy of building semantic matching between S and R.

VGG-19 is a Convolutional Neural Network (CNN) that is trained on more than a million images from the ImageNet database[v]. The network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. A typical VGG-19 structure is shown as Fig. 2.1. As a result, the network has learned rich feature representations for a wide range of images[12]. Since pre-trained models like VGG-19 have learned features of many categories of objects, the weights and biases of these network can be transferred to different data. For instance, as VGG-19 has been trained with a large amount of building images, it contains learned features like edges, lines etc. of the building category. By

[v] The ImageNet project is a large visual database designed for use in visual object recognition software research. It contains more than 20,000 categories with a typical category, such as “balloon” or “strawberry” consisting of hundreds of images.

feeding image pairs to a pre-trained model for object recognition, we can save time and compute resources to achieve effective feature extraction in our particular scenarios.

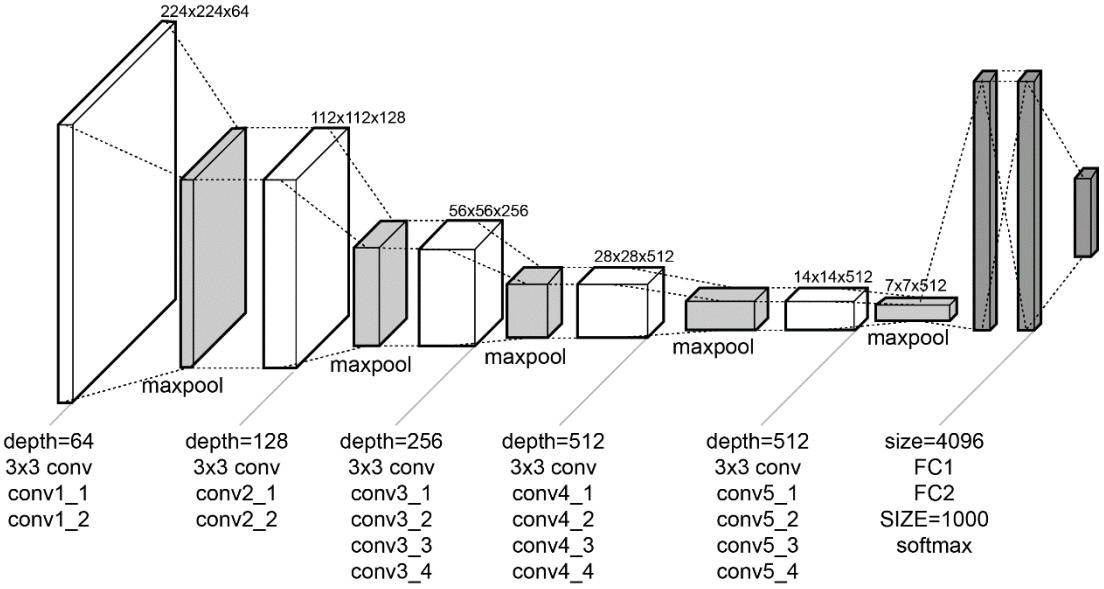

Fig. 2.1 Illustration of the network architecture of VGG-19 model[13]

Applying VGG-19 to the source image S and reference image R, feature maps of them can be extracted from different layers. At a layer L, the feature map of S and R can be defined as F_S^L and F_R^L respectively. As feature maps contains abstract features of identifying certain objects, things of the same category would have similar feature maps though they have largely different appearance. This begs a question: how to measure similarity of feature maps? Computing nearest-neighbor fields (NNF) introduced in the next section has been proven a useful technique to address this issue.

2.2 Nearest-Neighbor Field Search

In the image domain, matching patches between two images is: given two images S and R, find for every patch in S a similar patch in B, which is called PatchMatch method. The milestone PatchMatch algorithm[14] is fast for computing approximate nearest-neighbor correspondence between two patches of two image regions and has been used in various computer vision problems, including object removing from images, reshuffling or moving contents of images, inpainting/retargeting images, texture synthesis etc. Though patch matching is usually carried out in the image domain, it can also be performed in the feature domain, since CNN keeps the spatial relationship of input images, i.e. correspondence between feature maps represents correspondence between images to a great extent.

The goal of the PatchMatch algorithm is to find the patch correspondence by defining a nearest-neighbor field (NNF) as a mapping function. In this paper, for each patch around image coordinate p in the source feature map F_S^L, its nearest-neighbor patch around $\mathrm{q} = \emptyset_{S\to R}^L(p)$ in the reference feature map F_R^L needs to be found. If we search for every point in F_R^L (brute search), the computation cost is too high. Inspired by natural coherence in the imagery, Barnes et al. suggested that if we can find some good patch matches via random sampling, other good matches could be quickly found in the surrounding areas, which avoids global searching[14]. Three key phases of Barnes' randomized nearest neighbor algorithm are shown in Fig. 2.2.

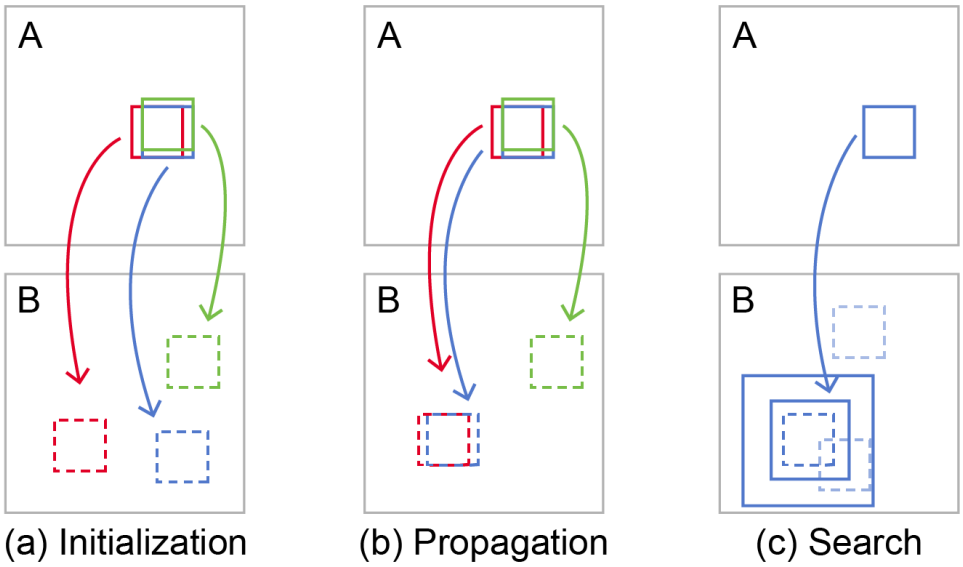

Fig. 2.2 (a) Patches are initially assigned with random values; (b) The blue patch of A checks above/green and left/red neighbors to see if they will improve the blue mapping (denoted as blue arrow). In this example, the red patch in B has higher similarity with blue patch in A, then assign the offset of left/red neighbor patch to the blue patch; (c) The patch searches randomly for improvements in concentric neighborhoods

The complete algorithm starts from phase (a) in Figure 3, and then perform iterative phase (b) and phase (c) to improve NNF. It has been proved that a few iterations (4-5) the NNF can converge to its limit, which is very efficient.

With this technique, the mapping denoted as $\emptyset^L_{S\to R}$ from F^L_S to F^L_R as well as reverse mapping denoted $\emptyset^L_{R\to S}$ from F^L_R to F^L_S can be built efficiently. The bidirectional correspondences allow us to use Bidirectional Similarity (BDS) voting[15] to respectively reconstruct the guidance image G^L and the corresponding feature map F^L_G. G^L contains S's content and R's color style, therefore it can serve as the guidance for color transfer in the next step – local color transfer.

2.3 Local Color Transfer

However, G^L and S have different resolutions which leads to great difficulty in building in-place correspondence between them, therefore a well-designed color transfer algorithm is needed to change the colors of the source image S to better match those of G^L.

Given the guidance image G^L at layer L, we could downscale S to S^L to match the resolution of G^L. To perform local color matching, He et al. constructed the color transfer function as a linear function for every pixel p in S^L. Thus, the transferred result T^L_p can be obtained by Equation (1).

$$T^L_p\left(S^L(p)\right) = a^L(p)S^L(p) + b^L(p) \qquad (1)$$

In Equation (1), a^L and b^L is the linear coefficients and the aim becomes to estimate $a^L(p)$ and $b^L(p)$ for each pixel p, such that the target result $T^L_p\left(S^L(p)\right)$ is similar to $G^L(p)$ as much as possible. To estimate proper coefficients for linear functions given a certain data distribution is a common problem which can be solved by constructing an objective function then minimizing or maximizing it according to specific requirements.

As He et al. considered, to construct such an object function, three aspects listed below should be involved:

1) a term E_D to makes the color transfer result similar to the guidance G^L;
2) a term E_L to encourage locally adjacent pixels to have similar linear transforms while preserving edges in the source S^L;
3) a term E_{NL} to enforces the non-local constraint to penalize global inconsistency, which is sourced from the assumption that pixels with identical colors in the source should get similar transferred colors in the result.

Therefore, it leads to constructed the overall object function as Equation (2).

$$E_{total} = E_D + \lambda_L * E_L + \lambda_{NL} * E_{NL} \quad (2)$$

In Equation (2), λ_L and λ_{NL} are weights of E_L and E_{NL} and $\lambda_L = 0.125, \lambda_{NL} = 2.0$ by default.

By minimizing the objective function, the coefficients a^L and b^L of the color transfer function can be estimated from higher layer to low layer (the higher layer, the lower resolution), thereby obtaining the corresponding color transfer result.

2.4 Fast Guided Filter

Since in the above section the parameters $a^L(p)$ and $b^L(p)$ are estimated from low resolution to high resolution layers, therefore in order to get the full-resolution result in the finest layer (L = 1) we need to upsample $a^L(p)$ and $b^L(p)$. One of the tricky issues in this process is edge-preserving smoothing[vi].

The guided filter[16, 17] is one of the popular algorithms for edge-preserving smoothing. It depends on an important assumption that the guided image has linear relation with the filtered output image, involving coefficients $a_\uparrow^L(p)$ and $b_\uparrow^L(p)$. Because linear relation can preserve both regional information and edge information, for instance, if region A and region B are both smooth color areas and region C is an edge, the gradient between A and B would not be too large while the gradient between A and C or B and C would be relatively huge. Therefore, a linear transformation between the guide image G^L and the filtered output image $\tilde{S}^L$ can amplify the original gradients by the same proportion, such that the sharp edges in the source image will not be smoothened in the de-noising process.

The aforesaid properties make the guided filter algorithm very useful in image edge-preserving, image matting and image haze removal etc. In 2015, K. He et al. continued proposing the Fast Guided Filter[18] by speeding up the guided filter method to further popularize this method. The difference is made by first reduce numbers of pixels by subsampling (corresponding to $a^L(p)$ and $b^L(p)$) and then recover to the full resolution by upsampling (corresponding to $a_\uparrow^L(p)$ and $b_\uparrow^L(p)$) [vii], which is consistent with the layered color transfer process described in Section 2.3.

By least square estimation, the upscaled $a_\uparrow^L$ and $b_\uparrow^L$ can be achieved and the desired output image with high resolution can be obtained from Equation (3):

$$\tilde{S}^L(p) = a_\uparrow^L(p) S^L(p) + b_\uparrow^L(p), \forall p \in \tilde{S}^L \quad (3)$$

[vi] Edge-preserving smoothing is an image processing technique that smooths away noise or textures while retaining sharp edges. Refer to https://en.wikipedia.org/wiki/Edge-preserving_smoothing.

[vii] Subsampling refers to making an image smaller and upsampling refers to making an image larger.

3. Method

The goal of this paper is to tune He et al.'s color transfer approach from the engineering point of view such that the approach can fit day-night Scene transfer requirements for architectural renderings. Our scenario is to perform day-night scene transfer given two input architectural renderings sharing semantically-related contents whose appearance or structure can vary dramatically. The key challenge is to build semantic corresponding between these two input images, and we also resort to a CNN VGG19 to encode input images from low-level features to high-level semantics. However, as a generalized model, purely relying on VGG19 cannot not achieve satisfying results in recognizing objects of architectural renderings. Observing this, we optimize the color transfer algorithm by normalizing features (Section 3.1) and adding extra features (Section 3.3) to fit our application scenario.

3.1 Overview

Given a source image S and a reference image R, our algorithm can be described as below.
For a given layer L (5 layers in total by default):

- Extract feature maps F_S^L and F_R^L of S and R by VGG19 respectively.
- Perform normalization in each channel of F_S^L and F_R^L (Section 3.2).
- Adding more features by clustering or semantic segmentation techniques to help building mappings (Section 3.3).
- Use PatchMatch method to build bidirectional mappings by use of VGG19 features and new features: $\widetilde{\emptyset}_{S\to R}^L$, $\widetilde{\emptyset}_{R\to S}^L$.
- Reconstruct the guide image G^L from R^L and $\widetilde{\emptyset}_{S\to R}^L$, $\widetilde{\emptyset}_{R\to S}^L$.
- Build linear relation between S^L and G^L, and estimate the linear coefficients a^L and b^L.
- Use Fast Guided Filter to amplify a^L and b^L to $a_\uparrow^L$ and $b_\uparrow^L$ thereby upsampling S^L.
- Construct the result image with full resolution by $\tilde{S}^L = a_\uparrow^L S^L + b_\uparrow^L$.

According to previous research, the focus is how to accurately match regions with similar semantic between input images. We mainly consider two aspects in our scenario:
1) how to deal with features extracted by VGG19 such that important semantic information will not be lost (feature normalization);
2) how to add extra features to help building more accurate mappings (clustering and semantic segmentation).

3.2 Feature Normalization

When using VGG19 to extract image features, feature maps in different layers have different size. For instance, as Fig. 2.1 shown, the corresponding feature map F_S^2 is of size (112, 112, 128), which means the feature map contains both 112 pixels in the width and height, and the number of channels is 128. The feature map in each channel represents a feature of the input image, therefore weights of features should be balanced in case missing some important features. By observation, we noticed that value of pixels can vary dramatically in each channel. In some channels, the pixel value range can be $[-500, +100]$ while in some channels the pixel value range is $[-1, +1]$, which leads to a situation that some important

semantic information has been extracted in a channel but has little influence in the PatchMatch process due to narrow pixel value range and further impacts the accuracy or even effectiveness of the correspondence between S and R.

Feature normalization techniques are useful to balance contributions of all features. There are many ways for feature normalization, among which Standardization and Min-Max Scaling are the most common used two methods. In this paper, we apply Standardization method to normalize the data. The general process shown as Equation (4) is to carry out the calculation below for each feature map in each channel:

$$z = (x - \mu)/\sigma \tag{4}$$

In Equation (4), x stands for a pixel value, and μ and σ are the mean pixel value and standard deviation of a channel respectively.

After normalization, pixel value ranges of all channels are in an equal order of magnitude, thereby avoiding emphasized contributions of high value data and impaired influence of low value data.

3.3 Adding Extra Features

As aforesaid, VGG19 is a generalize pre-trained model that may not perform well enough in some specific application scenarios. Therefore, apart from applying feature normalization to original image features that are extracted by VGG19, adding extra features to join the original features can be considered an effective way to increase recognition accuracy and build better correspondence between S and R. Clustering and semantic segmentation techniques are often used to meet this goal.

3.3.1 Clustering

Clustering is a useful technique for statistical analysis and data mining. Usually, a cluster involves some patterns, and based on similarity there are more similarities between patterns in a cluster than patterns in different clusters. Considering similarities of some features, we use clustering techniques to produce feature clusters as extra features to increasing accuracy of mappings between S and R.

In our scenario, feature maps F_S^L and F_R^L extracted by VGG19 are of size $(224, 224, 64)$ at layer 1 (L=1). After massive experiments, we found that classifying all the 64 feature maps at this layer into 3 feature clusters (representing sky, building and ground) can achieve best performance. The specific approach is described as following.

Step 1: Concatenate F_S^L and F_R^L in the horizontal direction to get the new feature map $F_{(S,R)}^L$ whose size is $(224, 448, 64)$.

Step 2: Use K-Means clustering algorithm to cluster all pixels of the new feature map $F_{(S,R)}^L$ to obtain a one-hot format clustering result image $C_{(S,R)}^L$ with size $(224, 448, 3)$.

Step 3: Divide $C_{(S,R)}^L$ horizontally into two cluster images C_S^L, C_R^L both of size $(224, 224, 3)$

Step 4: Join the cluster images C_S^L, C_R^L and the original feature maps F_S^L, F_R^L in the previous channel and produce new feature maps FC_S^L and FC_R^L, both of size $(224, 224, 63+3)$

Step 5: Replace F_S^L and F_R^L with FC_S^L and FC_R^L to build mappings between S and R.

As new extra features are added to help building semantic correspondences, the precision of scene transfer is increased, which will be demonstrated in Section 4.

3.3.2 Semantic Segmentation

Semantic segmentation is a very important field in computer vision. It refers to the pixel-level recognition of images which is marking out the object category of each pixel in the image. In simple terms, the goal of segmentation is to output a segmentation image where each pixel contains the label of its category taking a grayscale or RGB image as input. For instance, given a GRB image only contains three categories of things: person, bicycle and background denoted as label 1, 2, 3 respectively, the output segmentation image replaces each pixel value with its corresponding label value. Because of this property, semantic segmentation has similar effects as clustering that can classify pixels into a certain number of groups.

There are many semantic segmentation architectures available, like FCN, SegNet, U-Net, PSPNet (Pyramid Scene Parsing Network) and Mask-RCNN etc. In our scenario, PSPNet[viii] trained with the ADE20K dataset of MIT is adopted to segment the input images. The specific approach is described as following steps:

Step 1: Input S and R to PSPNet to obtain output images M_S and M_R which are of size $(473, 473, 150)$.

Step 2: Reduce size of M_S and M_R to $(224, 224, 150)$ to fit VGG19. At each layer of VGG19, the reduced images are denoted as M_S^L and M_R^L.

Step 3: Concatenate M_S^L, M_R^L and feature maps F_S^L, F_R^L extracted by VGG19 at each channel to produce new feature maps FM_S^L and FM_R^L, both of size $(224, 224, 64+150)$.

Step 4: Replace F_S^L and F_R^L with FM_S^L and FM_R^L to build mappings between S and R.

It is worth noting that when adopting pre-trained semantic segmentation model to deal with images not all channel results will fit our application scenario since some different channels may represent same contents. Therefore, manually adjusting (combine or abandon some channels) outputs of semantic segmentation model are recommended to achieve better performance.

Apart from using pre-trained semantic segmentation models, users can also utilize self-trained semantic segmentation models that meet specified application needs to provide directive guidance for the process of color transfer.

3.4 Algorithm Implementation Scheme

The complete algorithm is described in Section 3.1, which is basing on He et al.'s progressive color transfer algorithm and the proposed feature reinforcing techniques (Section 3.2 & Section 3.3). The pseudo code of our implementation is listed in Algorithm 1 shown in Fig 3.1.

[viii] https://github.com/Vladkryvoruchko/PSPNet-Keras-tensorflow

ALGORITHM 1: Day-Night Scene Transfer for Architectural Renderings Algorithm

Input: Source image S, reference image R.

Initialization: set L = a random number between 1 and 5

Do

NNF search (Section 2.2):

$F_S^L,\ F_R^L \leftarrow$ feed S, R to VGG19 and get features.

Feature Normalization (Section 3.2):

$F_S^L,\ F_R^L \leftarrow$ normalize pixel values of $F_S^L,\ F_R^L$ by Equation (4).

Adding Extra Features (Section 3.3):

$FC_S^L,\ FC_R^L$ or $FM_S^L,\ FM_R^L \leftarrow$ apply clustering or semantic segmentation.

$\widetilde{\emptyset}_{S\to R}^L \leftarrow$ map FC_S^L to FC_R^L or map FM_S^L to FM_R^L.

$\widetilde{\emptyset}_{R\to S}^L \leftarrow$ map FC_R^L to FC_S^L or map FM_R^L to FM_S^L.

$G^L \leftarrow$ reconstruct S^L with R^L and bidirectional mappings.

Local color transfer (Section 2.3):

$a^L,\ b^L \leftarrow$ optimize linear transform from S^L to G^L by minimizing Equation (2).

$a_\uparrow^L,\ b_\uparrow^L \leftarrow$ upscale $a^L,\ b^L$ with Fast Guided Filter.

$\tilde{S}^L \leftarrow$ transfer the color of S by Equation (3).

end

Output: day-night scene transfer result $\tilde{S}^1$.

Fig. 3.1 Algorithm pseudo code

In our refined algorithm, we cancel the loop from layer 5 to layer 1 but only perform NNF search and local color transfer at one layer in order to increase the transfer speed. Therefore, we initialize the layer value to a specific number between 1 and 5 and our test results showed that layer 1 outputs the best performance since at this layer the image resolution is highest. Besides, to add extra features to help building more accurate correspondence between S and R, one method of the two: clustering and semantic segmentation can be chosen. We will evaluation the effects of these two methods respectively in the next section.

In the algorithm implementation[19, 20], C language is applied to optimize heavy computation modules like NNF search and local color transfer, thereby reducing the whole runtime compared with Python implementation.

4. Evaluation and Results

4.1 Performance

Our core algorithm is developed in the following environment, shows as Table 1.1.

Table 1.1 Algorithm development environment

Experiment Environment	
Machine Type	X64 compatible personal computer

Operating System	Windows 10, X64
CPU	Intel Core i7-7700 @ 3.60GHz
Mainboard	ASUS PRIME B250M-K (200 Series, Family - A2C8)
RAM	16 GB (GLOWAY DDR4 2133MHz)
Primary Hard Drive	Samsung SSD 850 EVO 1TB
GPU	Nvidia GeForce GTX 1050 Ti (4 GB)
Deep Learning Platform	Keras + TensorFlow

When using clustering to add extra features to optimize pixel corresponding building, the runtime is about 65 seconds for single-reference day-night scene transfer with an average resolution of 800 X 600. The time cost mainly occurs in three processes: 1) Setting up the architecture, including initiating TensorFlow and importing VGG19 etc. (approximately 10-20 seconds); 2) Building mappings between S and R by PatchMatch method (~40 seconds), which involves extensive computation to measure similarities on hundreds of channels. 3) The optimization of color transfer by feature normalization and adding extra features also take a relatively small portion of time cost (~10 seconds).

If PSPNet is used as semantic segmentation network to enhance feature extraction, the overall runtime is about 30 seconds longer than using clustering. The extra time mainly occurs in the process of setting up PSPNet and building mappings between S and R.

4.2 Evaluation

Main stream color transfer algorithms have been verified by many previous works. Our algorithm mainly refers to He et al.'s work, which has been verified with series images from artistic images to scenery images. In order to adjust the color transfer algorithms to fit day-night scene transfer for architectural renderings, we used feature normalization and adding extra features as feature enhancement techniques. To evaluate these techniques, we conducted two sets of experiments: feature normalization plus clustering and feature normalization plus semantic segmentation. In our tests, day architectural renderings are used as source images while night renderings are used as reference images. By comparing the results of applying these two ways to the same S and R, we found that both ways performed well enough under fine tuning.

Feature normalization plus clustering. In this set of tests, feature normalization and clustering were used to optimize the process of building mappings between S and R. We found that with these two feature enhancement techniques, good enough results (penultimate column in Fig. 4.1) can be achieved for most of our test images, which proved that our algorithm showed good adaptability in dealing with architectural renderings.

Feature normalization plus semantic segmentation. As aforesaid semantic segmentation requires manually tuning the outputs of segmentation networks, which depends on experience or even intuition. Therefore, using this way to enhance feature extraction leads to relatively unstable results in the application. With carefully tuning, we also obtained satisfied transfer results (last column in Fig. 4.1) in this set of tests and the results have little difference with applying feature normalization plus clustering method.

Fig. 4.1 Comparisons between clustering and semantic segmentation applied to architectural renderings

4.3 Day-Night Scene Transfer

To validate our approach on day-night scene transfer for architectural renderings, we first explore the main approaches of color transfer (Section 2) and propose our improvements (Section 3). In this section, parts of our experiment results are shown in Fig. 4.2.

These results proved that by making proper adjustment to well-developed color transfer technologies based on deep learning, image processing tasks like turning a fully rendered daytime scene into a night scene, indoor scene transfer, weather transfer etc. can be automatically finished by computers which saves time and human resource cost. Though experiments in this paper focus on day-night scene transfer, the adopted approach can be easily extended to more application scenarios by little adjustment, for instance, based on proper luminance and chrominance transfer, adding extra light source to the transferred images to create new scene graph. In the next section, we will demonstrate results of our attempts to created mid-autumn full-moon night renderings.

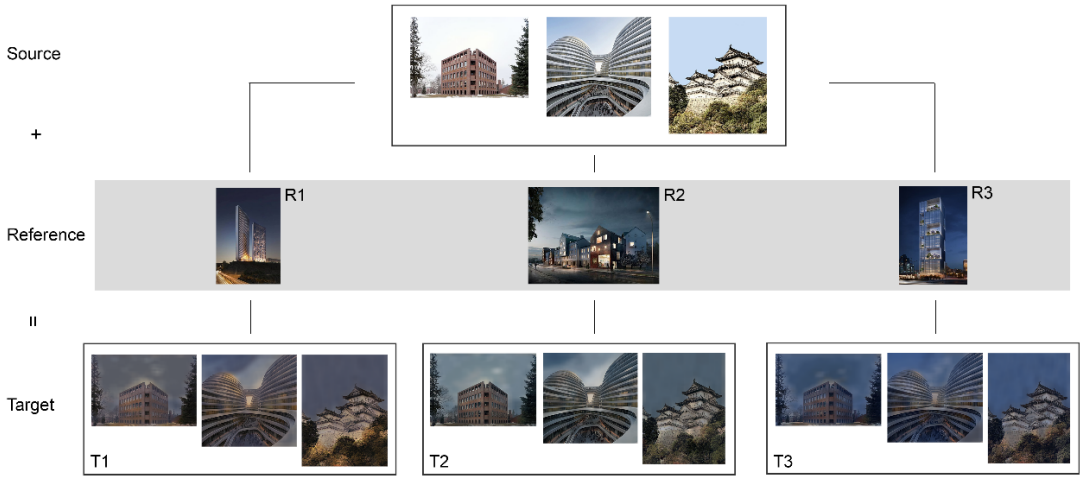

Fig. 4.2 Partial results of day-night scene transfer using our approach

4.4 Creating Full-moon Night Renderings

Since our approach transfers both luminance and chrominance, it makes scene transfer more effective and natural. Inspired by this property, we tried to create mid-autumn full-moon night renderings from daytime architectural images. This is a novel creation and the outputs are stirring (Fig. 4.3).

Fig. 4.3 Partial results of rreating full-moon night renderings

5. Conclusion

In this paper, we demonstrate a refined algorithm basing on main color transfer methods for day-night scene transfer between semantically-related architectural renderings. It handles both color and luminance transfer, which improves the naturality of scene transfer. And feature normalization and adding exta features techniques strengthen the consistency and accuracy. The evaluation results have shown that this approach is applicable and effective to day-night scene transfer, and it can also be adapted to more scenes transfer such as different weather conditions transfer and indoor scenes transfer.

Thought it is a successful application attempt in the field of architecture, there are still limitations of this approach. The first one is the precision of color transfer which also troubles scholars of computer

vision. To address this issue, generally we need to figure out better ways to build pixel correspondence between image pairs rather that fully relying on pre-trained models like VGG19. Because those models are trained on very generic dataset, thought they save us a large amount of time to do semantic matching across images they sacrifice accuracy or adaptation to some extent. Particularly in our application scenario, more targeted semantic segmentation models should be developed to improve object (building, sky, tree, road, etc.) matching precision thereby achieving more accurate color transfer. Secondly, computation cost should be well considered in the future. Currently, to deal with normal size renderings in personal computers time cost is about 60 seconds and it gets more for high resolution images. The time cost can be reduced by accelerating the process of building mappings between image pairs or resorting to parallel computing framework based on GPU.

For future work, we would explore tailored semantic segmentation models for the field of architecture and further refined color transfer algorithm. We could foresee that color transfer technologies will gain greater progress in day-night scene transfer and will be extend to various scenes transfer scenarios.

Reference:

1. Kevin Dale, Micah K Johnson, Kalyan Sunkavalli, Wojciech Matusik, and Hanspeter Pfister (2009). Image restoration using online photo collections. In 2009 IEEE International Conference on Computer Vision (ICCV). IEEE, 2217–2224.
2 . Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. (2017). Deep Photo Style Transfer. arXiv:1703.07511 [cs.CV]
3. Xiaobo An and Fabio Pellacini (2008). AppProp: all-pairs appearance-space edit propagation. In ACM Transactions on Graphics (TOG), Vol. 27. ACM, 40.
4. Dongdong Chen, Jing Liao, Lu Yuan, Nenghai Yu, and Gang Hua.(2017). Coherent online video style transfer. In Proc. Intl. Conf. Computer Vision (ICCV).
5. Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua (2017). Stylebank: An explicit representation for neural image style transfer. In Proc. CVPR, Vol. 1. 4.
6. Karen Simonyan and Andrew Zisserman (2014). Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556.
7. Mingming He, Jing Liao, Dongdong Chen, Lu Yuan, Pedro V. Sander (2018). Progressive Color Transfer with Dense Semantic Correspondences. arXiv:1710.00756 [cs.CV]
8. Anat Levin, Dani Lischinski, and Yair Weiss (2004). Colorization using optimization. In ACM transactions on graphics (TOG), Vol. 23. ACM, 689–694
9. Yuki Endo, Satoshi Iizuka, Yoshihiro Kanamori, and Jun Mitani (2016). DeepProp: extracting deep features from a single image for edit propagation. In Computer Graphics Forum, Vol. 35. Wiley Online Library, 189–201
10. Zhicheng Yan, Hao Zhang, Baoyuan Wang, Sylvain Paris, and Yizhou Yu (2016). Automatic photo adjustment using deep neural networks. ACM Transactions on Graphics (TOG) 35, 2 (2016), 11
11. Benoit Arbelot, Romain Vergne, Thomas Hurtut, and Joëlle Thollot (2017). Local texturebased. color transfer and colorization. Computers & Graphics 62, 15–27
12. Internet contributors. Pretrained VGG-19 convolutional neural network [Internet]. MathWorks [cited 6 Sep, 2019]. Available from: https://www.mathworks.com/help/deeplearning/ref/vgg19.html
13. Yufeng Zheng, Clifford Yang, and Alex Merkulov (2018). Breast cancer screening using convolutional neural network and follow-up digital mammography", In Proc. SPIE 10669, Computational Imaging III, 1066905. https://doi.org/10.1117/12.2304564

14. C. Barnes, E. Shechtman, A. Finkelstein and D. B. Goldman (2009). Patchmatch: a randomized correspondence algorithm for structural image editing. In SIGGRAPH

15. Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani (2008). Summarizing visual data using bidirectional similarity. In Computer Vision and Pattern Recognition (CVPR), 2008 IEEE Conference on. IEEE, 1–8

16. Kaiming He, Jian Sun, and Xiaoou Tang (2010). Guided image filtering. In ECCV, pages 1014

17. Kaiming He, Jian Sun, and Xiaoou Tang (2013). Guided image filtering. TPAMI, 35(6):1397–1409

18. Kaiming He, Jian Sun (2015). Fast Guided Filter. arXiv:1505.00996 [cs.CV]

19. GitHub contributors. Implementation of Neural Color Transfer between Images by PyTorch [Internet]. GitHub 2018 [cited 5 Sep, 2019]. Available from: https://github.com/rassilon712/Neural_Color_Transfer [accessed]

20. GitHub contributors. Fast End-to-End Trainable Guided Filter [Internet]. GitHub 2018 [cited 5 Sep, 2019]. Available from: https://github.com/wuhuikai/DeepGuidedFilter/tree/master/GuidedFilteringLayer/GuidedFilter_TF