Natural Image Colorization

Abstract

In this paper, we present an interactive system for users to easily colorize the natural images of complex scenes. In our system, colorization procedure is explicitly separated into two stages: Color labeling and Color mapping. Pixels that should roughly share similar colors are grouped into coherent regions in the color labeling stage, and the color mapping stage is then introduced to further fine-tune the colors in each coherent region. To handle textures commonly seen in natural images, we propose a new color labeling scheme that groups not only neighboring pixels with similar intensity but also remote pixels with similar texture. Motivated by the insight into the complementary nature possessed by the highly contrastive locations and the smooth locations, we employ a smoothness map to guide the incorporation of intensity-continuity and texture-similarity constraints in the design of our labeling algorithm. Within each coherent region obtained from the color labeling stage, the color mapping is applied to generate vivid colorization effect by assigning colors to a few pixels in the region. A set of intuitive interface tools is designed for labeling, coloring and modifying the result. We demonstrate compelling results of colorizing natural images using our system, with only a modest amount of user input.

1. Introduction

Colorization is the process of coloring monochrome images. It has been widely used in photo processing and scientific illustration. Traditionally, colorization process is tedious, time consuming and requires artistic skills to precisely add the appropriate colors to the image. In recent years, several interactive colorization techniques [WAM02, LLW04, ICOL05,HTC_05,HK05,YS06,QWH06,KV06] have been proposed to effectively colorize images with significantly reduced amount of user efforts.

These techniques colorize the image based on the user's input examples, which can be given in form of a similar colored image [WAM02, TJT05, ICOL05] or manually marked strokes in the input image [LLW04,HTC_05,YS06, QWH06,KV06]. The task in image colorization is in specifying which parts of the image should be colorized by what colors. The method by Levin et al. [LLW04], for instance, uses strokes to indicate colors of certain pixels and colorizes the image using an optimization based on intensity continuity constraints where adjacent pixels with similar intensity have similar colors. However, it may still require a very large number of strokes to achieve high quality colorization of images with complex textures, as shown in Figure 1(b). Moreover, selecting colors for more than a hundred strokes is no simple task.

The colorization problem is closely related to image segmentation since it aims to propagate user-specified colors (e.g., with strokes or example segments) to nearby image regions. Colorization techniques assume, explicitly or implicitly, that image segments can be well defined as coherent segments in intensity space [LLW04,HTC_05,HK05,YS06, KV06], or in texture space [ICOL05,QWH06]. The interactive Manga colorization technique [QWH06], for example, groups uniform pattern regions into a small number of distinctive clusters before colorization. While such patterncontinuity constraints work well for Manga cartoons, they are not effective for natural images with rich and inhomogeneous texture distributions.

In this paper, we present an interactive colorization system that requires modest amounts of user interactions for natural image colorization. Colorization is explicitly divided into two steps in our system, Color Labeling and ColorMapping. Instead of assgining colors directly to the image using strokes, users first scribble to group regions that would roughly share similar colors in the color labeling step, without worrying about the specific color for each local region. Then the color mapping step is applied in each labeled region to create vivid colorization effect.

In the color labeling step, we designed a new labeling scheme to handle texture regions commonly seen in the natural images, in which not only nearby pixels with similar intensities but also remote pixels with similar texture features should share similar colors. This new framework makes it possible to segment natural images into coherent regions with a small number of strokes specified by the user.

In the color mapping step, colorization with rich color variation can be obtained using only a few color pixels assigned in the labeled region. We provide the user with realtime feedback so that he can simply select appropriate colors or create colorization of a variety of different styles.

Overview of our system An example of our system is shown in Figure 2.With only several strokes in (a), we obtain a color labeling in (b) where coherent regions (sky, house, grasses, and flowers) are segmented. By specifying colors for a couple of points in each coherent region, we get the colorized image in (c).

The user starts by scribbling distinctive color labels on the regions of interest. For example, when scribbling a couple of strokes on two flowers, the user says something like “I’d like to colorize the flower field similar to these two flowers". The yellow color labels associated with these two strokes are iteratively propagated across the whole image. Our energy optimization propagates color labels to intensity-continuous and texture-similar regions that may be far apart and disconnected. This labeling scheme reduces a large amount of interaction in scribbling the strokes.

Once the regions are labeled with colors, we use a simple color mapping scheme for rich colorization. For each coherent region in the image, we choose a few pixels that have significantly different luminance values and assign colors to them. Their luminance (Y) values define a piecewise linear mapping that can be applied to corresponding chroma channels (U,V) to effectively colorize rich textured regions. This process of specifying a few color pixels for those with high and low luminance values is intuitive for the user. This becomes a simpler task for the user to think of colors for only a few pixels in a coherent region (e.g., a flower field), without having to worry about the whole image.

The colorized image now is ready for the user to inspect. The user may be unsatisfied with some parts of the colorized image, possibly due to errors that occurred while we labeling complex images. Whenever necessary, the process of color labeling can be refined by adding a few more strokes, and by constraining the optimization in a much smaller search range. We have designed two intuitive UI tools to help with this refinement task.

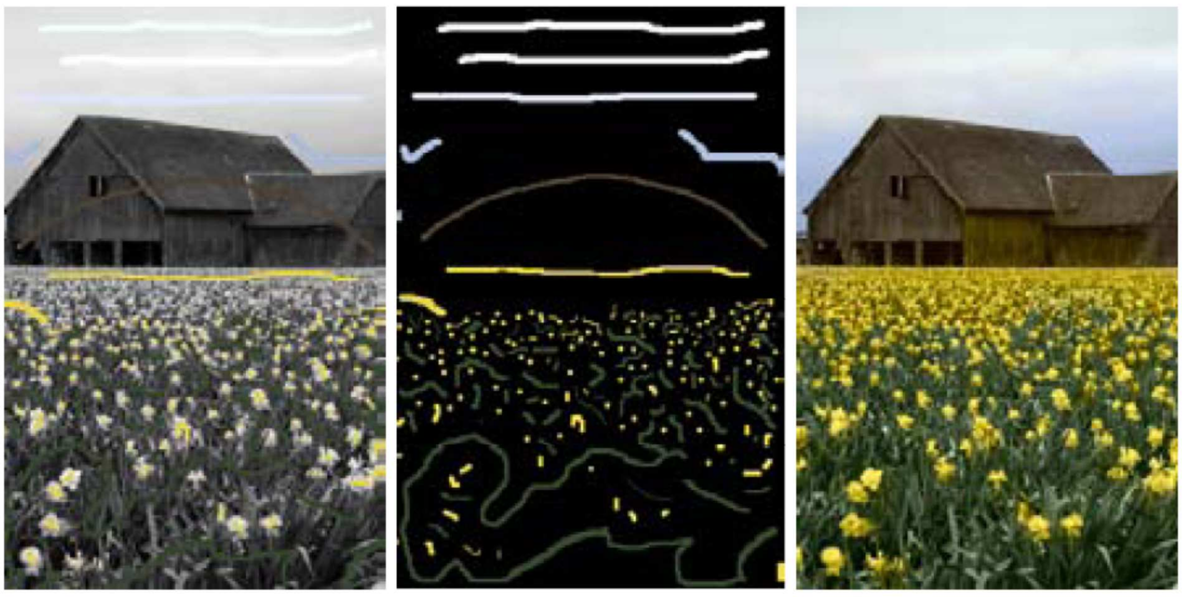
(a) (b) (c)

Figure 1: Levin's method. (a) Strokes on the image. (b) Strokes alone. (c) Results using [Levin et al.2004]. Hundreds of strokes are needed for colorizing a natural image with texture regions like this.

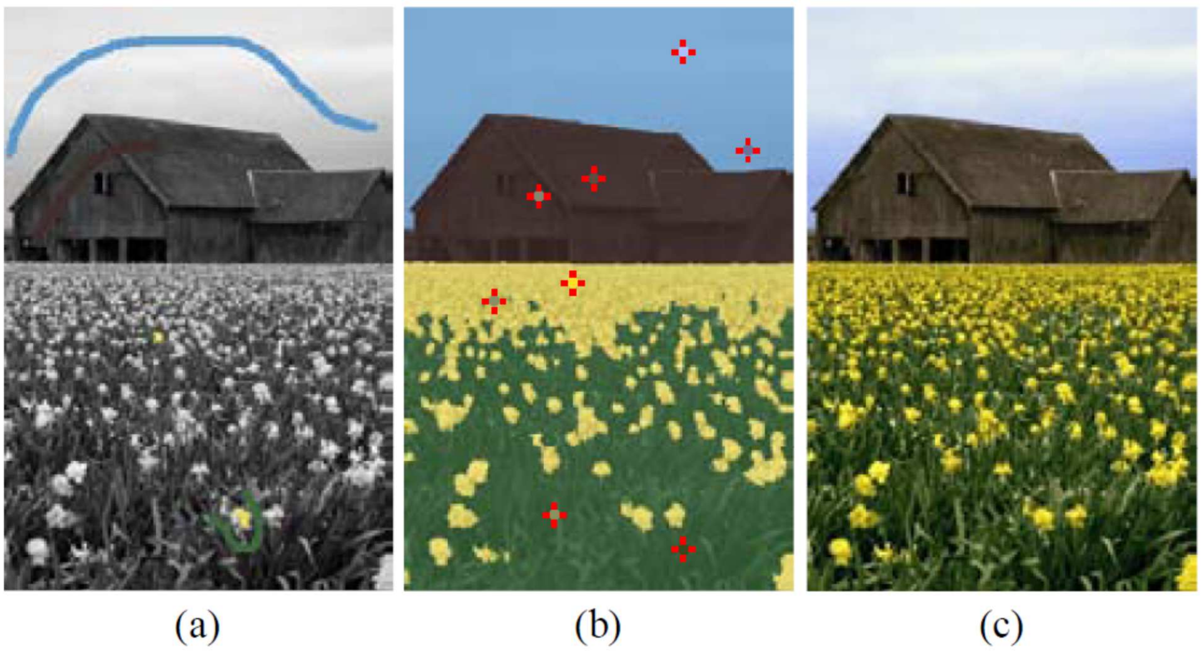

Figure 2: Our colorization procedure: First, several strokes with pseudo colors are drawn to group the regions that roughly share similar colors, as shown in (a). Only six strokes are drawn to label the image. Then, in each label region, we fine-tune the color by specifying colors for several pixels, as shown in (b). Only two pixels in each region are chosen to assign colors in this example. (c) Our final colorization effect.

Previous work To facilitate the colorization of images, a number of techniques have been recently proposed [WAM02, LLW04, ICOL05, TJT05, HTC_05, HK05, YS06, QWH06, KV06]. One approach for colorization is based on learning techniques [WAM02]. The relation between the grey level image and its colored version is learned from examples [RAGS01,TJT05]. Irony [ICOL05] use a supervised learning technique to better classify feature-space and a voting technique to increase the spatial consistency of the colorized image. This technique assumes that a similar example image is presegmented. Otherwise, the task of segmenting the example image itself can be almost as hard as colorizing the input image.

A new class of colorization techniques is user-guided [LLW04, HTC_05, YS06, QWH06, KV06]. The user draws color strokes over the image, and the colors diffuse from the strokes outward across the image. Levin et al. [LLW04] propagate the colors from the strokes to the entire image by solving a simple optimization problem, based on the premise that neighboring pixels in image space that have similar (monochromatic) intensity should also have similar colors. Yatziv et al. [YS06] color pixels with a weighted average of stroke colors, where the weights are proportional to the geodesic distance between the pixel and corresponding stroke. These techniques assume that the intensities are locally smooth, an assumption that does not hold for textured images.

Qu et al. [QWH06] introduce a method for colorizing Manga images. They analyze the texture space, and define an affinity to measure distances between a pixel and a stroke in feature space. Then for a given stroke they evolve a level set around it to associate a spatially-coherent region with that scribble. The level set is defined by considering the affinity and the smoothness of the image. This technique is designed to deal with Manga illustrations which can be successfully segmented into regions of homogeneous textures. In a natural image the variety of textures is large, and the texture space cannot be clearly clustered, thus there is no hope to successfully define an effective affinity and hence define a good criterion to control the level set. The technique we introduce in this paper avoids defining a global affinity or a metric, but rather defines a color distribution function by taking local decisions only. This allows propagating through smoothly varying textures. Figure 15 shows how our technique can successfully deal with the Kimono example from [QWH06].

Our work is also very much related to texture clustering and segmentation. Texture clustering often uses features such as filter banks (e.g., [VZ03]), random fields, and wavelets. Texture clusters have been used for image segmentation in computer vision, and texture patches have also been successfully used for efficient texture synthesis [EL99] in computer graphics. Similar to [QWH06], we make use of texture clusters for colorization.

2. Color Labeling

In our interface, a user scribbles a few strokes on the image indicating which regions or objects should be colored by what specific label colors (as shown in Figure 2(a)). This high level user interaction does not need precise input since our method is not sensitive to where we draw strokes(shown in the video). The objective of color labeling is to assign a color label to every pixel in the image, given the pseudo colors associated with user-drawn strokes.

2.1. Energy Optimization Framework

Labeling of the natural image with texture regions is a challenging task. Existing interactive image segmentation methods like [LSTS04] would fail in handling texture regions since only intensity distribution is considered (as shown in Figure 3(b)). To address this problem, the textures should be explicitly taken into the consideration. One straightforward method is using the texture feature as the likelihood term in graphcut. However, erroneous labels may be brought in, as shown in Figure 3(c). These errors essentially originate from the confusion among texture features in different regions of the image, which incurs a high risk of misclassification.

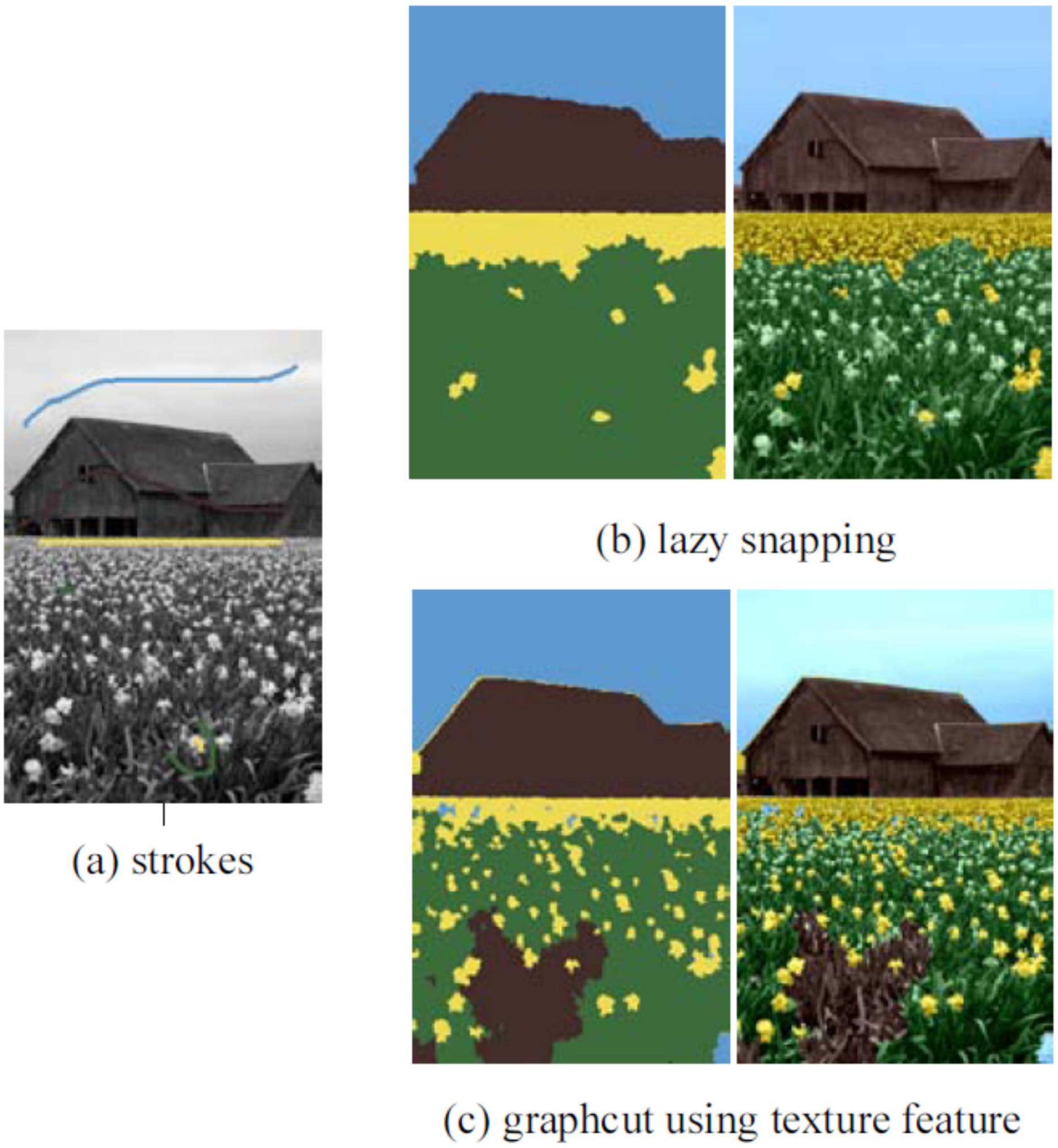

Figure 3: Comparison with other segmentation methods. (a) The strokes (we tried several sets of strokes, and these were the ones with the best segmentation results) (b) The lazy snapping result [LSTS04](we segment two regions at a time). Notice the repeated texture is not well handled. (c) The result using graphcut with texture feature as likelihood. The spatial smooth term in graphcut cannot handle severe misclassication.

Investigation of the problem leads us to the insight into the complementary nature inherently possessed by the highly contrastive locations and the smooth locations.

1. The texture features at the highly contrastive locations tend to be well clustered in the feature space.

2. The smooth regions are characterized by the coherence between neighboring colors, thus a pixel therein can be reliably colorized based on the intensity continuity.

Motivated by the observation above, we propose a new formulation that integrates the intensity continuity and the texture-similarity. A smoothness value identifying different natures of locations is used to guide the incorporation of the two constraints. We trust more in the texture similarity term for highly contrastive locations, while the intensity continuity is considered as more reliable for smooth locations.

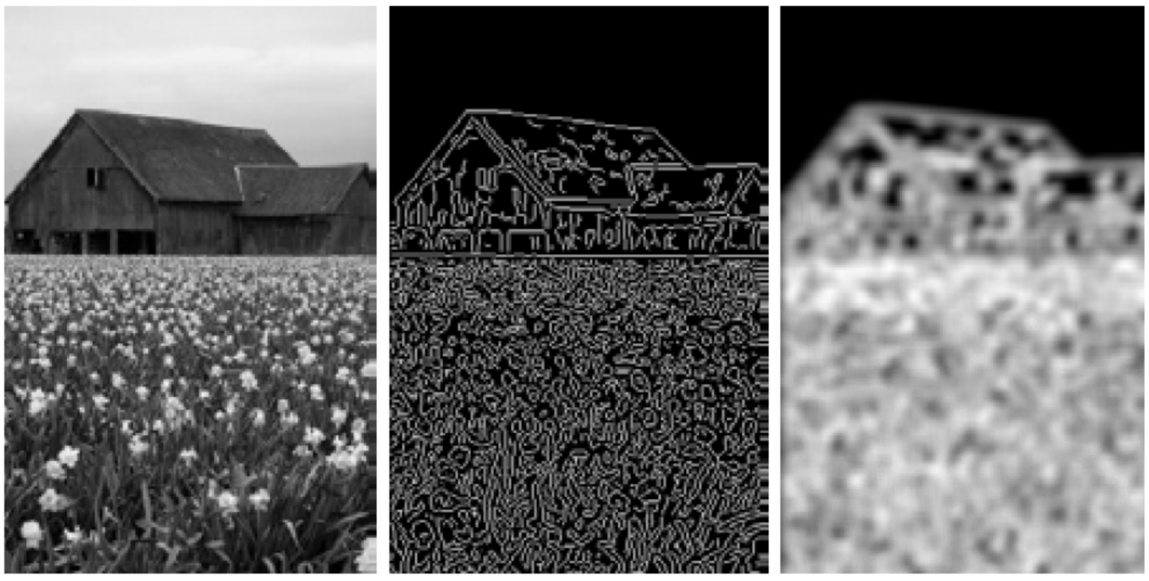
(a) (b) (c)

Figure 4: The smoothness map. (a) A gray image. (b) Edge image for (a) generated by canny operator. (c) The smoothness map obtained by applying Gaussian filter with kernel N(0,3^2) to the edge map shown in (b).

Specifically, we obtain the likelihood of every pixel to be colored by each label color. This color label likelihood function is denoted as L(**C**;p), where **C**=$[C_1,C_2,...,C_N]^T$ represents all the label colors.

We introduce an energy optimization framework that incorporates both intensity continuity and texture similarity constraints for all the pixels p in the image:

$$E = \sum_{p \in img} (\lambda(p)E_1 + (1-\lambda(p))E_2), \qquad (1)$$

where the textural term E_1 and the spatial term E_2 are

$$E_1 = \sum_{q \in \Omega_t(p)} w_{pq}^t \|L(\mathbf{C};p) - L(\mathbf{C};q)\| \qquad (2)$$

$$E_2 = \sum_{q \in \Omega_s(p)} w_{pq}^s \|L(\mathbf{C};p) - L(\mathbf{C};q)\| \qquad (3)$$

under hard constrain: L(**C**;p)=$[0,...,1_k,...0]^T$, p∈$stroke_k$, where $stroke_k$ means strokes with the color C_k.

To solve the above optimization, we need to define the weight map λ(p), textural neighbors Ω_t (p) and spatial neighbors Ω_s(p).

2.2. Smoothness Map: λ(p).

There are many ways to estimate the smoothness around a specific location. In our method, we use the filtered edge map. At each pixel p, the value of λ(p) is related to the distance between p to its nearest edge. Specifically, we use the Canny operator to extract the edges of the image. We then apply a Gaussian filter with kernel $N(0,\sigma^2)$ on the edge image to obtain a smoothed edge map. If p is near edges, λ(p) is large, otherwise λ(p) is small. In our implementation, the σ is set as a quarter of the patch size that we used for texture space analysis, as shown in Figure 4.

2.3. Textural neighbors Ω_t(p) and w_{pq}^{t}.

We regard the texture neighbors of a pixel p as those pixels that are similar to p in feature space. The set of the texture neighbors is denoted as Ω_t(p). Before the user interaction, a pre-processing of texture analysis is done to cluster the patches collected in the image. The calculations of the Ω_t(p) and w_{pq}^{t} are based on these pre-computed clusters.

Texture Analysis: Since the value of λ(p) vanishes for pixels which are away from any edge, the textural term E_1 can be ignored at these pixels to speed up the texture analysis process. Therefore, we collect only the texture patches whose centers are on the edges. These patches are then clustered according to their appearance. Sometimes patches with similar appearance belong to regions that are spatially apart. Inspired by the texture analysis method in [MBLS01], we further cluster those texture patches with similar appearance according to their spatial relationship. Thus, our texture clustering consists of two levels: appearance clustering and spatial sub-clustering.

We first apply k-means clustering on all collected patches, using Sum of Squared Distance (SSD) as the distance metric. More sophisticated clustering techniques like expectation maximization (EM) have not shown significant improvement in our experiments. In general, the cluster number is set much larger than the number of color labels to guarantee that the patches in one cluster are very similar in appearance.

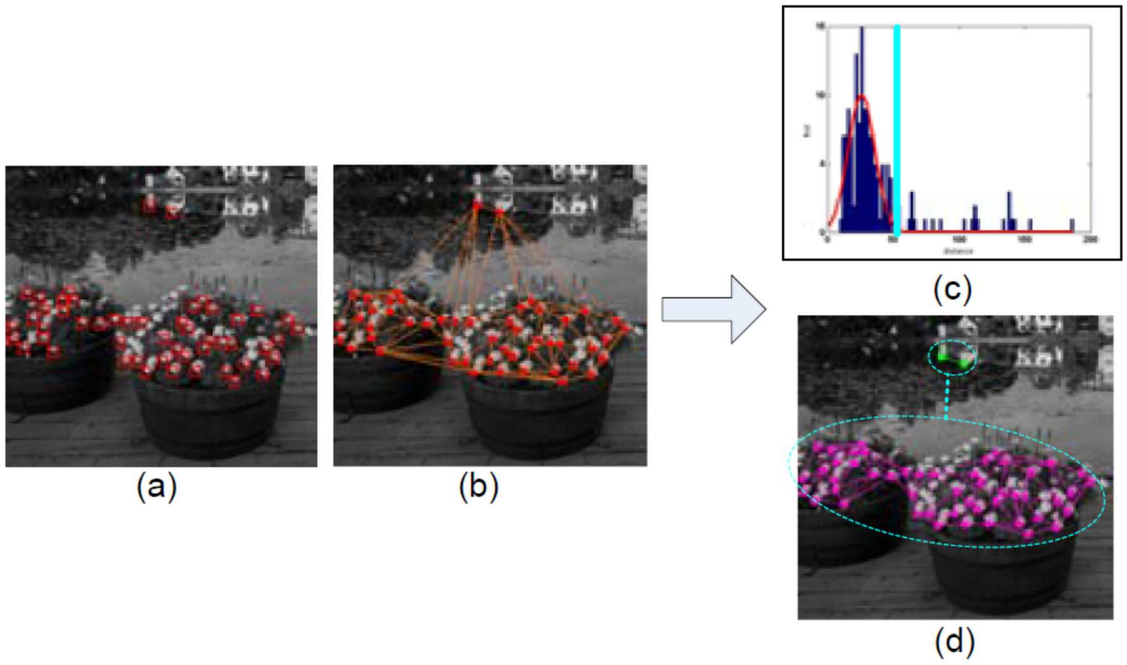

Figure 5: Illustration of spatial sub-clustering process. (a) red squares show all the texture patches in one patch cluster. (b) centers of the patches are connected to build a graph (Delaunay Triangulation). (c) Distribution for the length of the graph edges is shown by dark blue bar. Red line represents the fitted Gaussian distribution. The position of light blue line is the Expected Distance threshold. (d) Two sub-clusters obtained by spatial clustering are shown. Purple points represent patches of one sub-cluster while green points for the other.

Any given cluster can then be further clustered according to their spatial relationships. In Figure 5, we show an example describing the details of sub-clustering. For all the patches in one cluster (Figure 5(a)), we compute a Delaunay triangulation of the patch centers, shown in Figure 5(b). Next we analyze the distances of the nearby nodes in the graph. The rationale is that texture patches in the same cluster tend to also form clusters in image space, and the outlier patches tend to be remote. The distribution of distances of nearby nodes is shown in Figure 5(c). We model this distribution as a Gaussian N(gm,gv) and set the Expected Distance as gm+2gv. All edges that are longer than the Expected Distance in the graph are disconnected to form several subgraphs, which are now represented as sub-clusters (see Figure 5(d)).

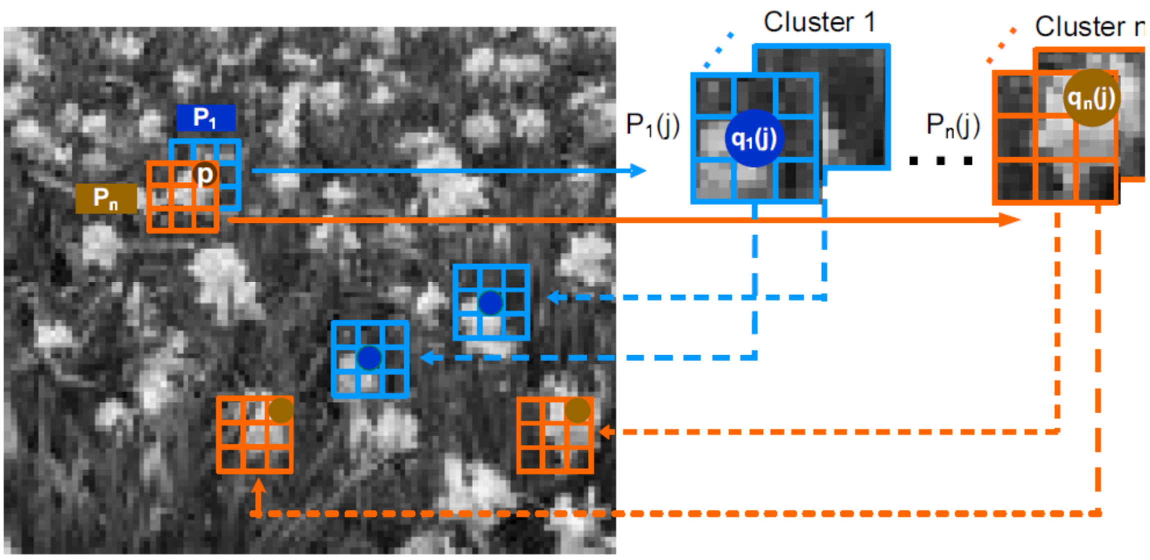

Figure 6: Multi-patch decision. The color of the pixel p is decided by all the patches that contains the pixel, such as the blue patch P1 and the yellow patch P_2. All such patches belong to the patch clusters, shown as Cluster 1 and Cluster n. The color likelyhood of p is accordingly decided by all the patches in these clusters, shown as $P_1(j)$ and $P_n(j)$. The set of texture neighbors of p is formed by all the pixels in the patches with the same relative spatial location as p (such as $q_1(j)$ and $q_n(j)$).

Multi-patch decision for $\Omega_t(p)$: Now we need to find all the texture neighbors of pixel p and form $\Omega_t(p)$. Note that we consider all the image patches that contain the p, not only the one which is centered on it. Each patch that covers the pixel p belongs to a patch cluster. We get all the patches that belong such clusters, and take the pixels on the same relative spatial positions of p to form the $\Omega_t(p)$. As shown in Figure 6. Let P_k,k=1,...,n be the patches that contain the pixel p, and $\{P_k(j), j=1...m\}$ be the set of patches that share the same appearance cluster with P_k, m is the number of patches in this cluster. Let $q_k(j)$ be the pixels in patches $P_k(j)$ that have the same relative spatial position as p in patch P_k. All these $q_k(j)$ are included to compose the set $\Omega_t(p,k)$ and $\Omega_t(p)=U_k\Omega_t(p,k)$

Decide w_{pq}^t. using sub-clusters: When we compute w_{pq}^t, texture neighbors from the different sub-clusters get much smaller weights than those from the same sub-cluster. Specifically, for every texture neighbor $q \in \Omega_t(p)$, if the patch that contains p and the patch that contains q belong to the same sub-cluster, w_{pq}^t=0.9/Z; while if they belong to different sub-clusters, w_{pq}^t =0.1/Z. The Z is a normalization constant to guarantee $\sum_{q \in \Omega t(p)} w_{pq}^t = 1$.

Since we take patches from different sub-clusters as well to define Ω_t, we can colorize the same textures that are spatially separated by drawing strokes in only one region.

2.4. Spatial neighbors $\Omega_s(p)$ and w_{pq}^s.

We follow [LLW04] to define Ω_s and w_{pq}^s is defined as 8-neighbors of a pixel and $w_{pq}^s = \exp\left(\frac{-(I(p)-I(q))^2}{2\delta^2}\right) / (\sum_q w_{pq}^s)$. In our implementation d is set to 10.

2.5. Iterative Energy Optimization

The energy function in Equation 1 is solved by iteratively propagating the L with both textural and spatial neighbors:

$$L^0(\mathbf{C};p) = \begin{cases} (0,\ldots,1_k,\ldots,0)^T & p \in stroke_k \\ (\frac{1}{N},\ldots,\frac{1}{N},\ldots,\frac{1}{N})^T & otherwise \end{cases} \qquad (4)$$

$$\begin{aligned} L^{n+1}(\mathbf{C};p) &= \lambda(p) \sum_{q\in\Omega_t(p)} w^t_{pq} L^n(\mathbf{C};q) \\ &\quad + (1-\lambda(p)) \sum_{q\in\Omega_s(p)} w^s_{pq} L^n(\mathbf{C};q) \end{aligned} \qquad (5)$$

Since the Ω_s(p) is defined as the 8-neighbors of the pixel p, the color label propagation along the spatial neighbors is rather slow. To speed up, we over-segment the image to super-pixels using the Mean Shift algorithm [CM02]. Instead of nearby pixels, we use the nearby super-pixels to propagate the color labeling information to the current pixel. Weights are decided by the intensities of adjacent pixels in the nearby super-pixels. Figure 7(b)(c)(d) show the iterations in the color labeling. For this image, L stops changing after 3 iterations to the result in Figure 7(d), using the iterative algorithm with super-pixels. It would require more than 70 iterations without using super-pixels. Note that the super pixel only extends the spatial neighbors of current pixel, the texture neighbors of current pixel remain the same.

When we get the label map after several iterations, it is straightforward to assign each pixel the color label with maximum likelihood, as shown in the Figure 7(d). However, some color noises and color leaks near the boundary may exist, for example, at the left edge of the roof. To refine the color label, we use the a-expansion multi-label graph-cut algorithm [BVZ01]. The labeling noise is suppressed and the boundaries of color regions are enforced along the strong roof edge in the gray-scale image, as shown in Figure 7(e).

2.6. Discussion

More experiments are done to validate the use of the smoothness map and the sub-cluster in our labeling algorithm.

Smoothness map plays an important role in integrating the texture-similarity constraint and the intensity-continuity constraint. We tried using a constant λ instead of the smoothed edge map as the weighting in our energy function. Figure 8(a) shows the result using λ=0.5.We can see errors in both the grass region and house region, which is mainly caused by misclassifications of the texture in a smoothed region. In our method, we use the smoothed edge map to set λ(p) for the textural term. By putting more weights on the spatial term in the smooth region, the labeling information from spatial neighbors helps to correct some mistakes from textural space. Thus we can get a better labeling result as shown in Figure 8(c).

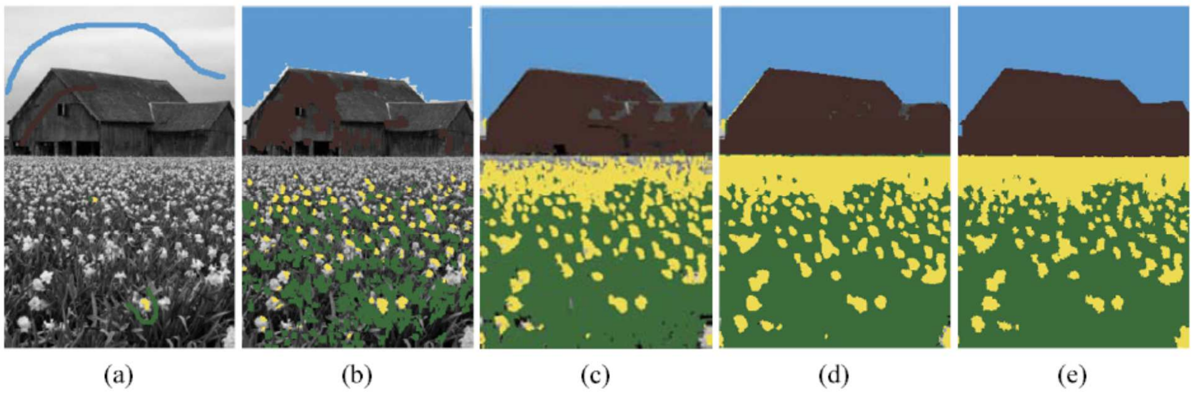

Figure 7: The process of color labeling. (a) The user input strokes. (b) (c) (d) The color labels after the 1st, 2nd and 3rd iterations. We can see that, with small number of input strokes, color labels are propagated to neighbor pixels with similar intensity and remote pixels with similar texture pattern. (e) Color labeling after applying multi-label graph-cut.

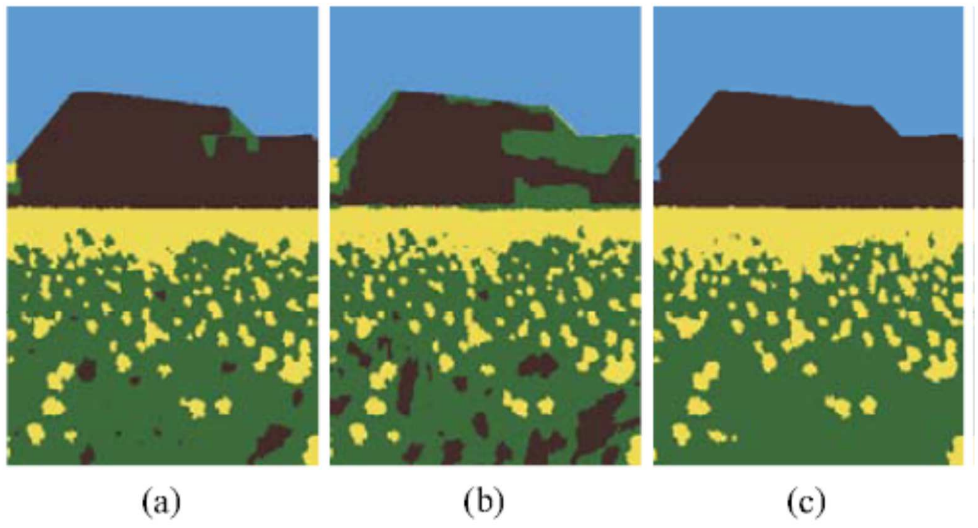

Figure 8: (a) The color label using constant λ, labeling errors can be seen in the house and grass regions, which are mainly due to the misclassification in texture features in smooth locations. (b) The color label without sub-cluster, large misclassified regions can be seen in the house and grass region, since regions of house and grass are mixed in feature spaces. (c) Our algorithm proves to be effective in labeling the natural images.

Using sub-clustering is another novel idea in our algorithm. Figure 8(b) shows the result without using spatial sub-clustering. We can see many misclassified regions in the house and grass parts. Using sub-clustering, the house region and the grass region can be well divided into two different sub-clusters. By weighting less the information from different sub-clusters, we can get a better labeling result as shown in Figure 8(c).

3. Color Mapping

In this section, we show an simple yet effective step that enables users to generate rich colorization with vivid colors. We call this step Color Mapping . Figure 9 shows the color mapping process.

Once a region is selected, the user chooses a few pixels as shown in the top image of Figure 9(b). These pixels represent a significant luminance variation in the region. Each pixel is then given a corresponding color by the user, as shown in Figure 9(c). The chroma (UV) values for any other pixels are then interpolated by piece-wise linear mapping in luminance (Y) space to get a color palette, as shown in Figure 9(d). More sophisticated non-linear functions have been tried but have not shown much improvement in our experiments. The color mapping result of the selected region is shown in the bottom image of Figure 9(b).

Note that in our system, the final colorization result is not a hard composite of each colorized region. After colorizing the regions, we do a soft blending around the region boundary to make the color transition natural, as shown in Figure 9(f). Over a band along the boundary, we run the intensity continuity term in Equation 1 to get a blending weight for each label. The final color in this boundary region is the weighted average of the colors for each label. This enables our system to colorize the fine structures in the image.

We have designed an easy-to-use UI for colorizing. When a user clicks a pixel in the image, the corresponding labeled region is selected. Then the user can choose an appropriate color for this specified pixel. This action is shown in the attached video. Our UI allows the user to go back and forth easily to see the original image, color labels, blended image, and partially finished colorized image.

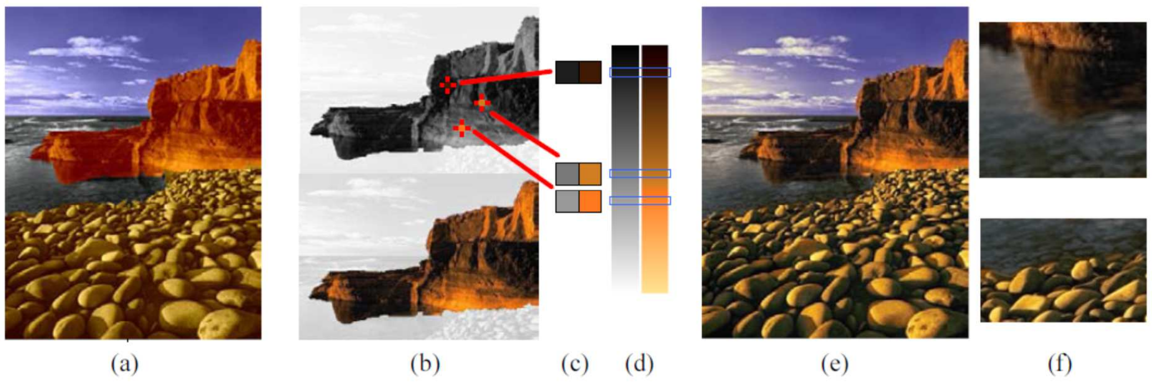

Figure 9: Illustration of color mapping process. (a) Color label is directly blended with the gray image. (b) Top: the region of mountain is selected for color mapping. The user chooses three pixels to assign colors for this region. Bottom: the mountain after color mapping. (c) Three pixels with their intensity values from the image. Their corresponding colors specified by the user are shown on the right. (d) From the three known pairs, colors for all the possible intensities are interpolated, shown with the two bars here. (e) The final image after color mapping all the regions. (f) Zoom-in views of regions where soft blending between nearby regions is applied.

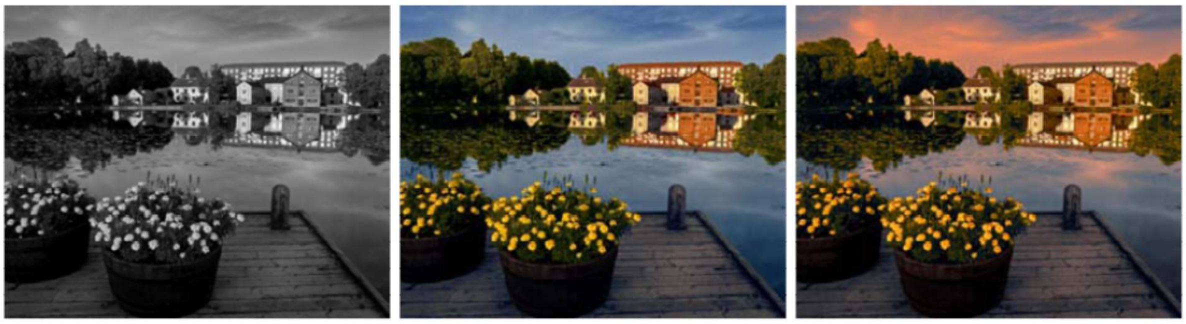

Figure 10: We can obtain various colorization effects from only one color label using our color mapping tool. Here we show an example. Left image shows a natural image with highly textured regions. Middle and right images show different colorizations using our system.

Compared with other stroke-based colorization methods, our color mapping is effective in editing the final effect. In most cases, users do not know exactly what color to use in each location before they preview the effect. By separating the colorizing process from the labeling process, users can edit in realtime the color in each region. Users can also obtain different colorizing effects of an image once its color labels are obtained. Figure 10 shows the impressive results we get for a single natural image. Moreover, our color mapping step can overcome small amount of inaccuracy in color labeling, we do not assume perfect color labeling for images with complex textures.

4. Local Refinement

Our system works well on many images such as those on Figures 2 and 9. With the casually specified strokes, our global optimization labeling algorithm can find a sufficiently good label map for colorization. For images with more complex textures, however, texture misclassification may still cause problems, as shown in Figure 11. Recognizing that there are inevitable mistakes in texture clustering, we have designed two simple UI's to help the user to interactively correct the color labeling.

For instance, some yellow flowers were missing at the bottom of image in Figure 11. The user may not be satisfied with the incorrectly labeled regioin. He can draw a short stroke with the correct color, then a local optimization of energy in Equation 1 using the surrounding areas of the strokes would be run to get the desired color labels, as shown in Figure 11. Pixels on the stroke drawn by the user are used as hard constraints (with yellow labels) in the optimization.

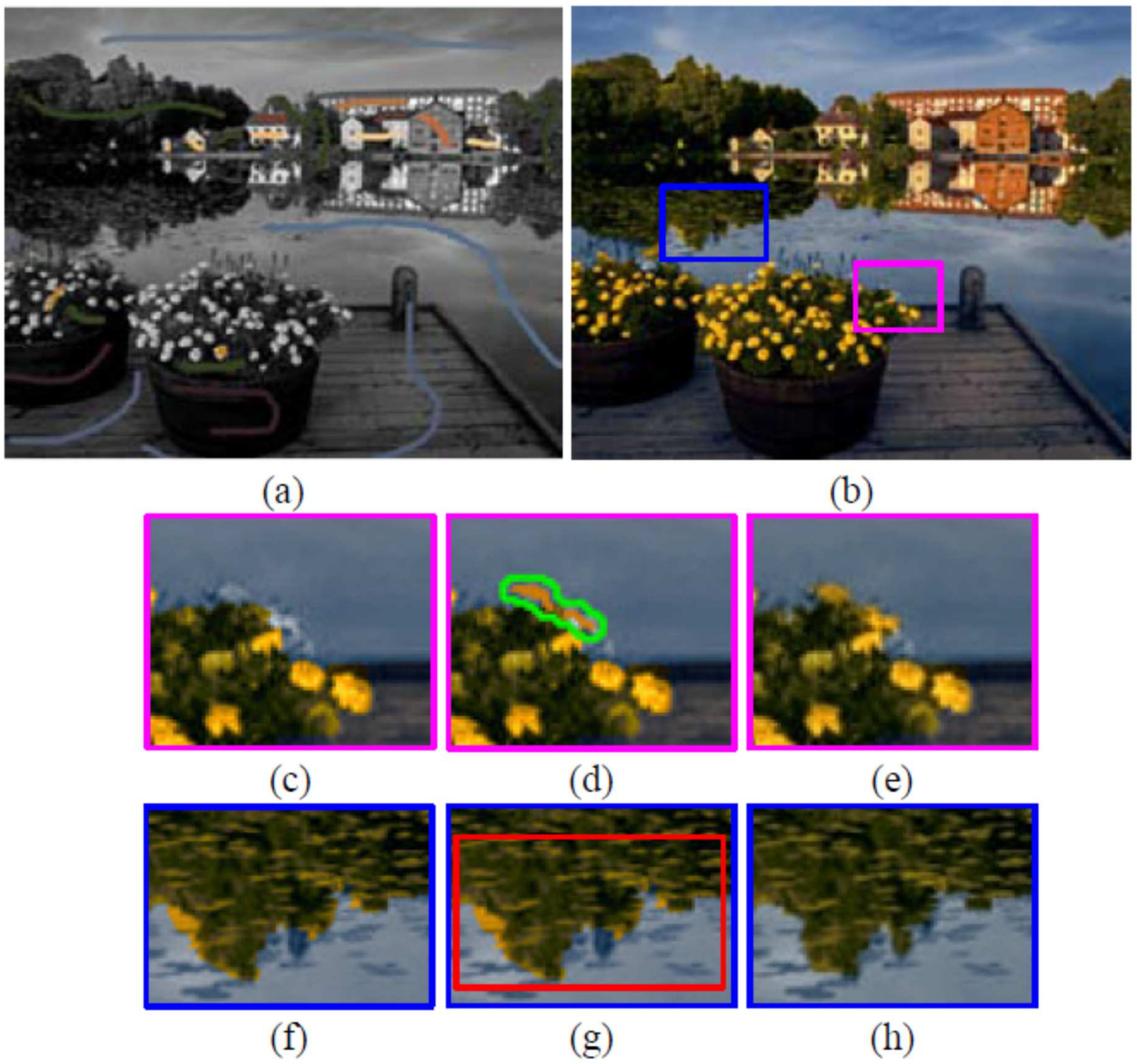

Figure 11: Two UI's for local refinement. (a) An image with user specified strokes. (b) Colorization result before local refinement. (c)-(e) show an example of using short stroke tool for refinement. A yellow stroke is drawn inside the green region (d), here we use a green line to highlight the small yellow stroke on the flower. Local optimization is applied to propagate the yellow color label and obtain the desired result (e). (f)-(h) show an example of using the rectangle tool for refinement, in (g) a rectangle is drawn indicating the error in this specific region. Local optimization runs inside the region to get a better colorization (h).

As shown in the Figure 11(f), some patches near the inverted reflection of the tree cannot be distinguished with patches in the flower region, which results in some errors in the color labeling. the user can also draw strokes to correct the label, but this time the label cannot be easily corrected with one or two of strokes. Instead, we designed another simple UI for the user to draw a rectangle (or “grab" like in [RKB04]) which includes the erroneous pixels and correctly labeled pixels with the desired color. Equation 1 is optimized only for the indicated region to refine the colorization. By confining the optimization in a local region, we prevent the propagation of error information from the outside of the region. Thus a better label map can be obtained. The result is shown in Figure 11(h). Sophisticated selection tools other than a rectangle can also be used.

Note that after the color mapping step, most of the labeling problems become unnoticable in the colorization. We do not need a perfect segmentation to get a nice colorization result. The user can usually get satisfactory results with a rather small number of refinement.

5. Experiments

The clustering (pre-processing) time depends on the patch size, the number of edge pixels and the cluster number. For all the images in our experiment we use cluster number 500, which shows to be sufficient for images with different level of complexity. Patch size is manually select which roughly equals the largest size of texture element in the image. The width of the square patch used in the examples are: 13 (Figure 2), 17 (Figure 10), 13 (Figure 12), 17 (Figure 13), 19 (Figure 14), 9 (Figure 15), 11 (Figure 16(a)), 15 (Figure 16(c)), 19 (Figure 16(e)), respectively. For the image(size of 262×392) in Figure 2 with the edge map shown in Figure 4 and the patch size 13×13, the time for clustering is about 2 minutes on a Pentium 4 3GHz PC. Each iteration in optimization requires 0.7 second. It took 3 iterations (less than 3 seconds) to get a satisfactory color label for this example. All the examples in this paper require no more than 10 iterations to be stable. Color mapping for all the examples is done in real-time. Local refinement has also been done interactively as shown in the video.

We show a number of examples to demonstrate the effectiveness of our approach.

In Figure 12 we compare our method with some previous methods.We show that with 10 interactions (3 strokes(in(a)) + 1 local refinement + 6 specified color pixels), we obtain a compelling colorization in (b). For a fair comparison, we also use 10 interactions for other methods. Using 10 strokes as shown in (c), (We tried many sets of strokes, this is the one with the best colorization) we show the colorization results of Welsh et al. in (d) [WAM02], Levin et al. in (e) [LLW04], and Yatziv et al. in (f) [YS06]. we can see that the strokebased method like Levin's method and Yatziv's method cannot colorize the texture regions of the image properly within 10 interactions, resulting in many visible mis-colored regions particularly for trees and rocks. In the experiment of Welsh's method, we use the region under the user stroke as the swatch described in [WAM02]. We also use patches with size of 13×13, as in our k-means clustering. In Welsh's method, the patch distance is not sufficient to distinguish the different regions, especially the sky region and the grass region as shown in Figure 12(d).

Labeling methods based solely on intensity would fail in identifying the texture boundary. We can see in Figure 13(b) that Lazy Snapping [LSTS04] cannot catch the boundary around the wall, while our method perfectly segment this region (Figure 13(c)). We show the final colorization result of our method in Figure 13(d).

As discussed previously, using existing methods to obtain results with vivid color is not a simple task. We demostrate this in Figure 14. Even with several carefully chosen strokes that well labeled the image, the colorization still looks flat and unnatural in Figure 14(a). Of course, one can add more tiny strokes with different colors to create color variation in each region (Figure 14(b)). However, it is very hard to specify the location and the color for hundreds of strokes, especially without the aid of an interactive preview. The example in Figure 1(b) shows one attempt to get the vivid effect, and the result is still unsatisfactory. We show our result in Figure 14(c), with only 17 interactions (7 strokes + 10 specified color pixels), we can get a compelling colorization effect.

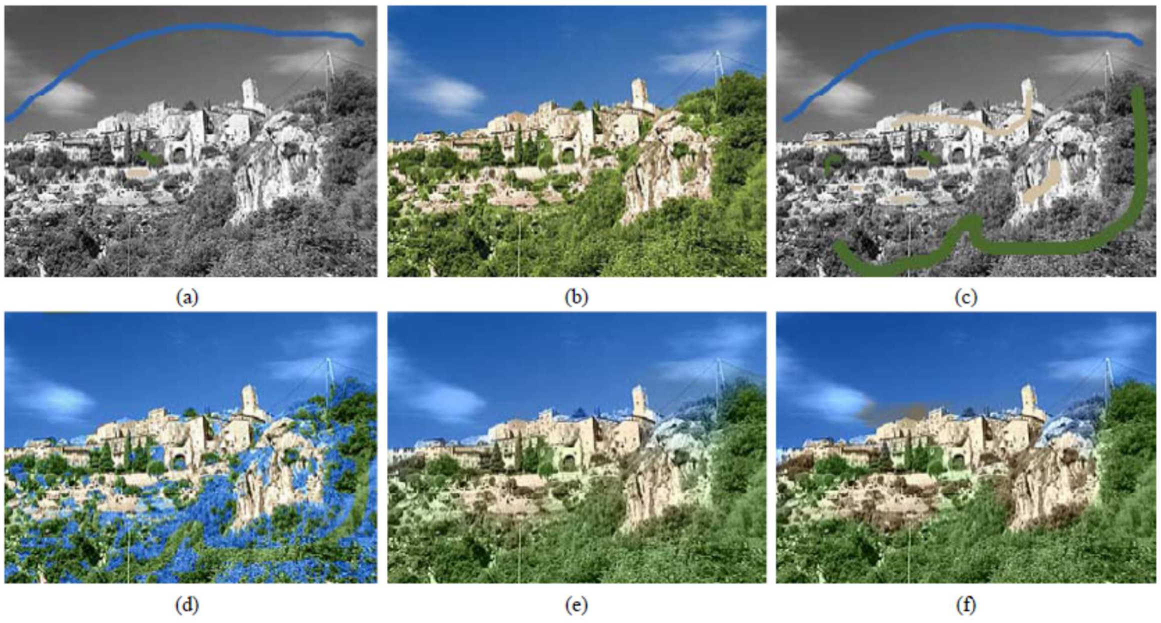

Figure 12: Comparison of different colorization methods. For comparison proposes, we use a total 10 of actions in all the methods. Using ten operations (3 strokes(in(a)) + 1 rectangle refine + 6 color specified color pixels), we get a nice colorization result shown in (b). With the ten strokes in (c), we tried Welsh's [WAM02], Levin's [LLW04] and Yatziv's [YS06] methods, results are shown in (d), (e), and (f).

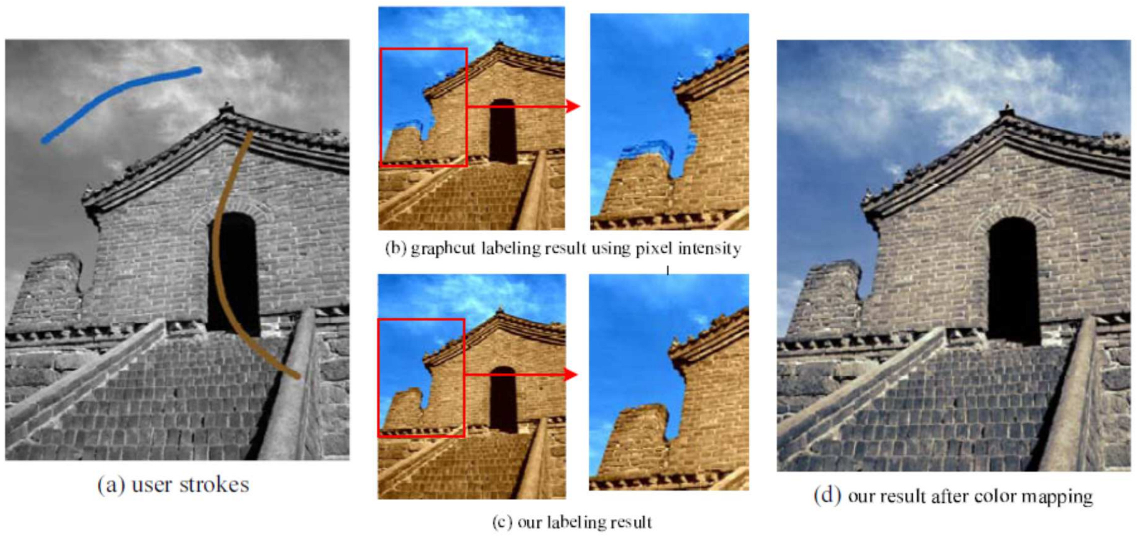
(a) user strokes
(b) graphcut labeling result using pixel intensity
(c) our labeling result
(d) our result after color mapping

Figure 13: Comparison with lazy snapping. Here we show a relative simple foreground segmentation case that lazy snapping would fail due to the lack of texture feature. (a) shows the user interaction, (b) is the result by lazy snapping, we can see that lazy snapping fails on the texture boundary around the wall, since only intensity distribution is considered in their likelihood for graphcut. Our method can well handle the texture boundary as shown in (c). We show our final result after color mapping in (d)

In Figure 15 we compare our algorithm with the Manga colorization method [QWH06]. The manga method failed in the inhomogeneous texture region, as shown in Figure 15(b). Since we only use the local decision of neighbors in our labeling scheme, we can handle the inhomogeneous texture in images, and the result is shown in Figure 15(c) Moreover, we use the patch appearance distance instead of statistical features for texture analysis, which enables us to colorize the texture region with multiple colors. We can see that using strokes shown in Figure 15(d), we can get the result as in Figure 15(e). This is the unique property of our method. (Note that in this example, we add three white strokes for the flat white regions in order to label the entire image).

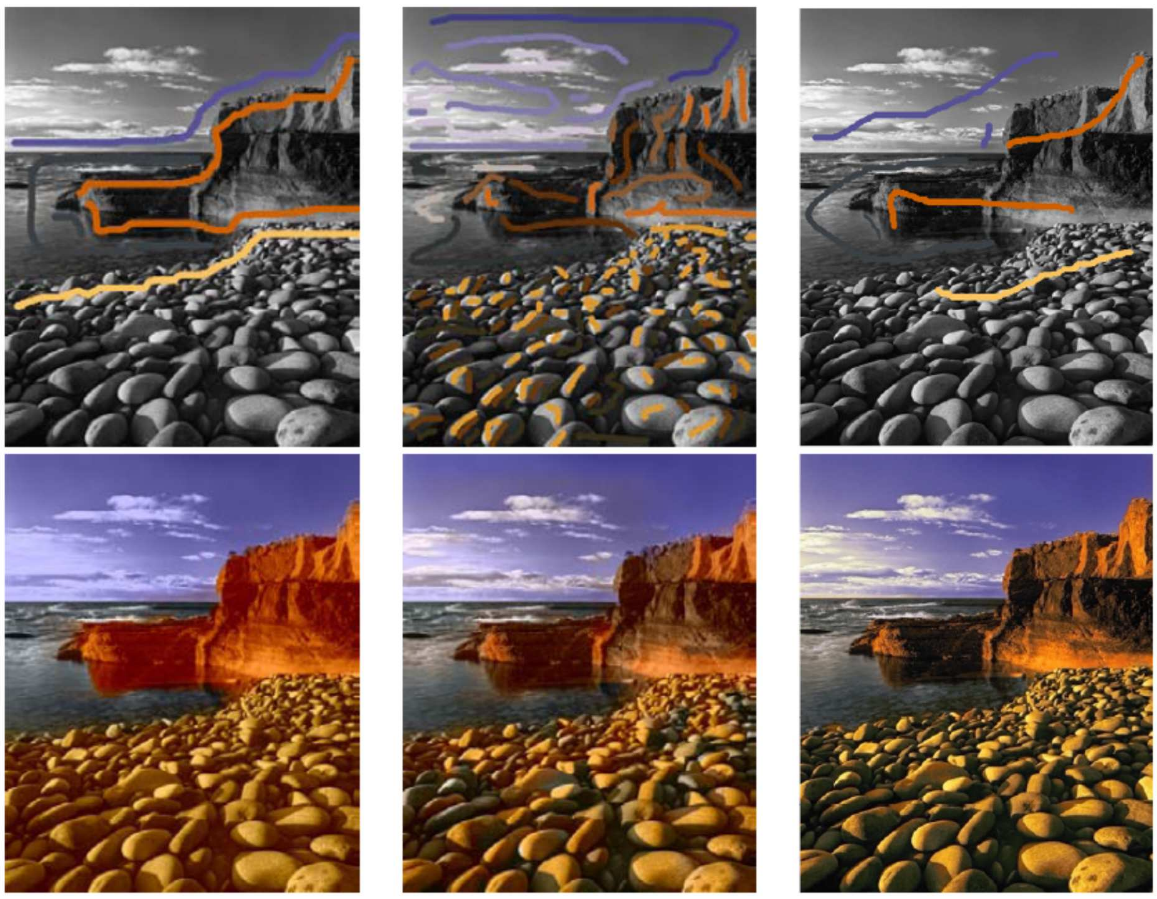
(a) Levin's result 1 (b) Levin's result 2 (c) Our result

Figure 14: Comparison with Levin's method [LLW04]. The first row shows the input strokes, the second row shows the colorization results. Using Levin's method, it is tedious to get a vivid colorization as in the genuine natural scene. We can see in (a) that strokes well segment at the image is not sufficient for a vivid colorization. An attempt is made to specify variant colors in different locations in the image. It is still hard to get a satisfactory colorization(as shown in (b)). We show our colorization result in (c). Compelling effects can be easily achieved using small number of interactions.

To demonstrate the effectiveness of our method in colorizing the texture pattern, we show more examples of colorizing texture using multiple colors in Figure 16. As discussed previously, it would be rather hard for user to colorize these type of textural regions using previous methods [LLW04,QWH06]. In these images, each tiny region inside the texture region requires a new color, which means the user needs to carefully add strokes in all the regions. In our method, the user only needs to colorize an example pattern in the texture region, so that the rest of the similar patterns would be colorized based on the given example. Interactions are largely reduced in our system.

The peacock example in Figure 16 also shows the limitation of our approach: those regions that are not properly labeled is not colorized correctly. Notice that toward the left side of the tail of the peacock, some smaller-scale patterns are not well colorized with current strokes and a small amount of local refinement. Surely the user can make a better colorization if he spends more time on locally refining the result, which is out of the scope of the effectiveness of our labeling algorithm.

6. Summary and Discussion

In this paper, we present an interactive system for colorization of natural images. We show compelling colorization results of natural images with highly textured regions. These natural images are very difficult to colorize by previous colorization methods.

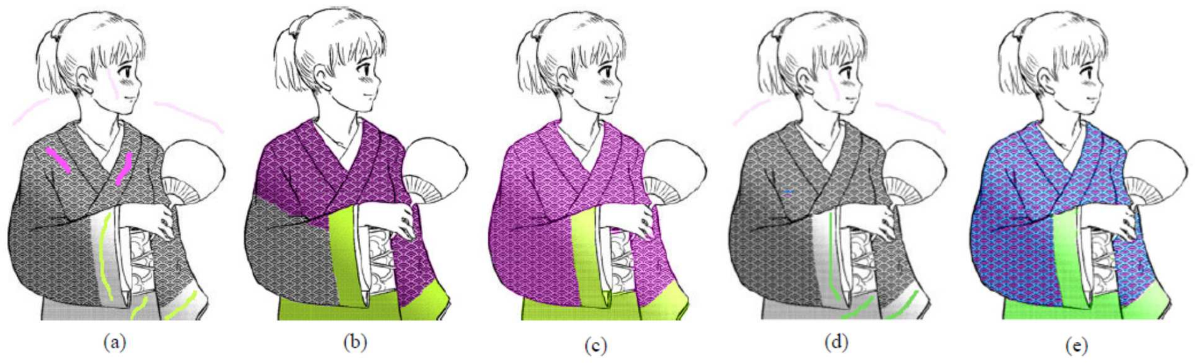

Figure 15: Comparison with Manga Colorization: (a) Strokes used in the paper of Manga Colorization (plus 3 white strokes for the flat white regions in order to get the label of the entire image). (b) Result by Manga Colorization. (c) Our result using strokes in (a). One

key property of our method is colorizing texture with multiple colors, as shown in (d)-(e). ©T.T.Wong/CUHK

Our method labels an image with colors associated with strokes drawn on the image by the user. We introduce a novel energy optimization framework that incorporates both intensity-continuity and texture-similarity constraints to cluster the image into a number of coherent regions, indicated by distinctive colors. Then, for each coherent region, we assign colors to a few chosen pixels, and then color map the rest of pixels in the region.

By separating the color labeling from the color mapping, the interaction of colorization becomes more intuitive for users. What users need to do for color labeling is to specify, with several strokes, some meaningful objects or regions to be labeled, whether it is the sky, or a flower field, or a lake. Specifying colors at a few pixels for color mapping is effective in getting satisfactory final results.

Good colorization results using our system are, first of all, due to the success of color labeling. Although we do not need sub-pixel accuracy boundary segmentation for the purpose of colorization, our novel energy optimization framework makes it possible to label the image into color and texture coherent regions that can be easily colorized. Although texture clustering may never be perfect, small errors resulting from our color labeling step can be easily corrected, by interactively refining incorrect labeled local regions using our easy-to-use UI tools.

In the pre-processing phase, we used texture patches without consideration of scale, orientation or transformation. As shown in the saree example in Figure 16, we had to draw multiple strokes on similar patterns with different orientations and scales to obtain good colorization results. In the future, we plan to study how to incorporate more sophisticated texture clustering techniques such as epitomes [JFK03] to further improve the usability of our system.

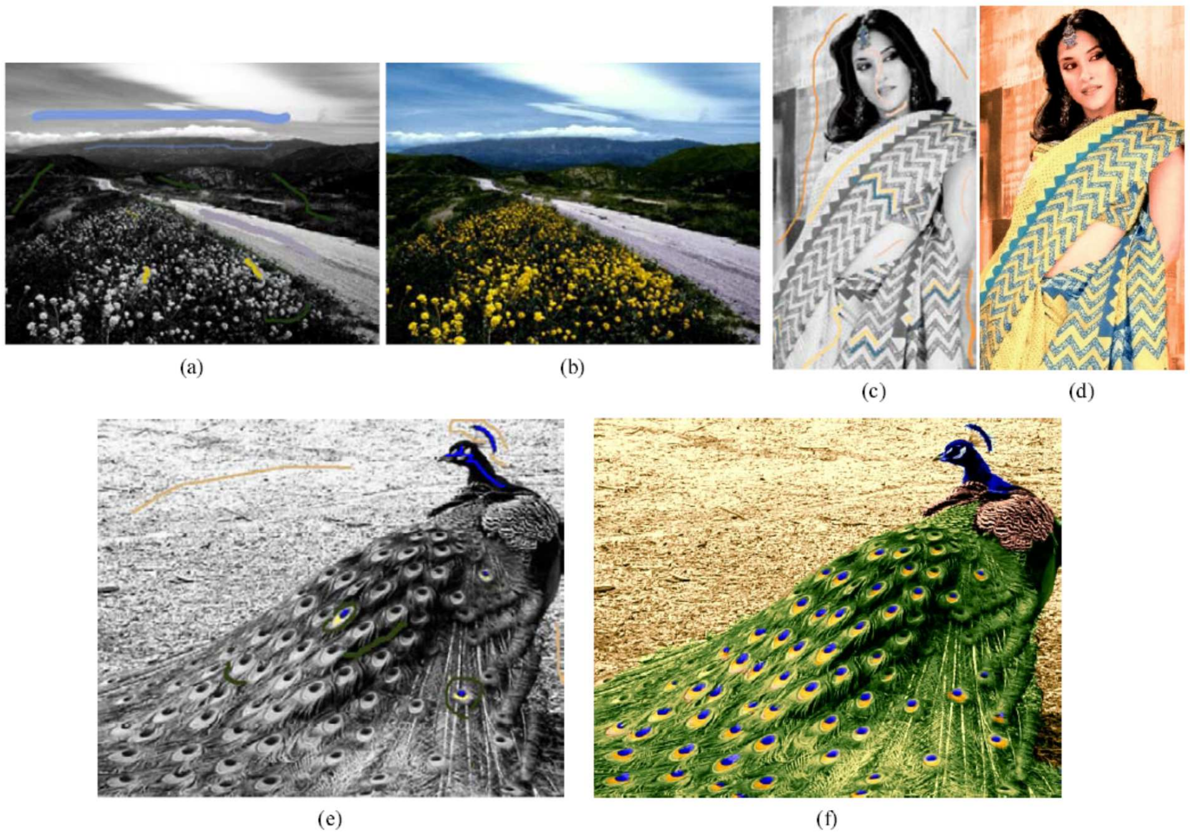

Figure 16: Colorize texture with multiple colors. Strokes are shown in (a) (c) (e), our colorization results are shown in (b) (d) (f). Colorizaion of texture can be quite tedious using the previous methods since each tiny region inside the texture requires a new color. Colorizing texture with multiple color is the unique property of our method.