CPSC 863 Course Project

Image Colorization Using Similar Images

Stephen Huang and Dachao Sun

School of Computing, Clemson University

April 21, 2015

Abstract

This project ...
(left to write finally.)

I. Introduction

This project aims at implementing a colorization algorithm on grayscale images, with the aid of reference color information — corresponding color images that are semantically similar to the grayscale ones. Scholars Raj Kumar Gupta et. al. proposed an example-based method in the paper "*Image Colorization Using Similar Images*" [1], whose pipeline is the original "building block" of this project. The highlights of this paper are:

- Fast superpixel representation (segmantation) for speed-up;
- Simple but effective (descriptive) feature extraction;
- Fast cascade feature matching scheme, which prunes the search space significantly;
- Space voting as a post-refinement step.

, which match the four phases of the pipeline respectively. An overview of colorization is to come first, before we go to details about the project.

A. Motivation: What Is Image Colorization?

Colorization (or manual colorization), as a self-explanatory term, is an intuitive action/process to add color on black-and-white photographs, pencil sketches or other grayscale images. In drawing, for instance, we may first come up with a pencil sketch and then to make it more real by colorizing it.

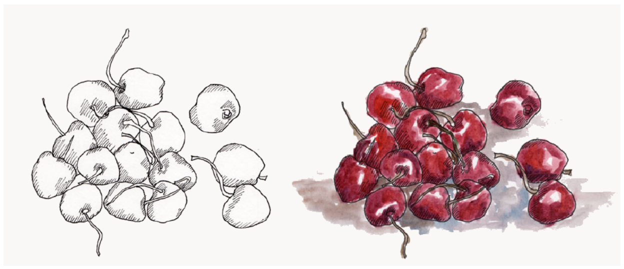


Fig. 1. Add color to a pencil sketch picture [2].

To further generalize the definition to the field of digital image processing and/or digital art producing, it can be summarized as a process to **convert a 1-channel, intensity-only image to a 3-channel color image**. Technically speaking, the goal/benchmark here is not merely engineering-like but to make the result "perceptually meaningful" and "visually appealing" [1]. The challenge, as a consequence, somewhat depends upon whether or not appropriate reference images are selected; meanwhile, in normal cases, the task has no "ground-truth" answer since it is judged subjectively by the viewers.

B. *Interactive vs. Automatic*

Colorization methods can be divided into two categories:

- **Interactive**: to insert color into the garyscale image manually by users, as a drawing process. After marking color scribbles for different regions/segments on the "target image", the grayscale one, each scribble will propagate under some certain scheme so as to colorize the whole region/segment.
- **Automatic**: to take color from a reference image and then transfer it to the target image. It usually includes a proper segmentation step to break the whole target and reference images into segments before colorizing them one after another. It is much more convenient and efficient, while usually calls for adjustments of parameters.

C. *This Project: A Four-phases Automatic Approach*

The colorization pipeline proposed in [1] is an automatic approach, which includes four phases:

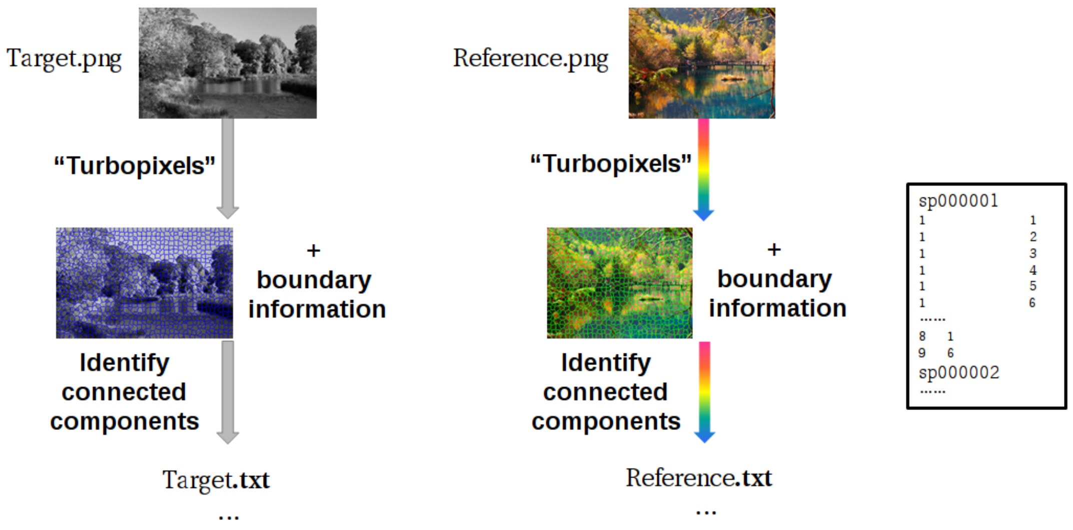
1) Do segmentation on both the reference and target images, using a method called "Turbopixels".
2) For each image segment, compute a 172-D feature that concludes: intensity (2), standard deviation (2), Gabor (40) and SURF (128).
3) Colorize each superpixel (segment) of the target image, by finding the "most similar" superpixel in the reference image.
4) Refinement: reassign the colors using a voting scheme.

We are first going to discuss how we implement (and modify) the four phases of the approach in chapter II, with some Matlab code details included. We will then demonstrate our experiment on this colorization pipeline, illustrate the results and discuss what might have caused the problem in Chapter III. Challenges encountered and future work will be discussed before coming to the conclusion.

II. Approach

A. *Phase 1: Identify Superpixels*

As a preparation step, it splits both the target and the reference images into many segments, which will be treated and colorized respectively later on. Fig. 4 gives a demonstration of Phase 1.



```
sp000001
1              1
1              2
1              3
1              4
1              5
1              6
......
8    1
9    6
sp000002
......
```

Fig. 2. Phase 1 demonstration.

The segmentation method used here is “turbopixels” [3], a fast segmentation algorithm propsed in 2009 by Alex Levinshtein, a PhD student at University of Toronto. It computes a dense over-segmentation of an image, and resulting segments are referred to as “superpixels”. We seek for the reference code shared by the author at [4], where the key function is “`superpixel`” and its usage is as:

```
[phi,boundary,disp_img] = superpixels(img,
                                      (num_pixels/40)/1.5,
                                      0,
                                      [0,0,1]);
```

, in which we specify the input image, expected size of each superpixel, a ‘zero’ flag and the color of the boundary. Fig. 3 shows that the boundries and contours are well detected.

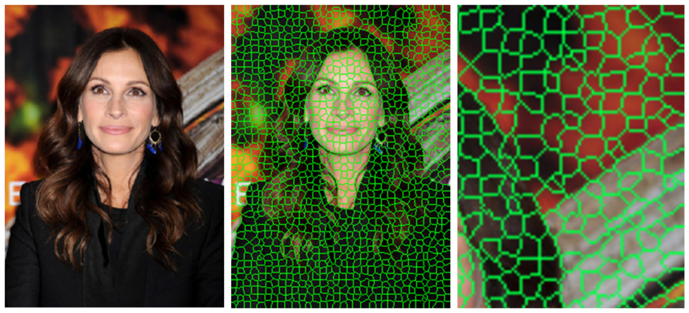


Fig. 3. Result of “turbopixel”: around 40 pixels per superpixel.

B. Phase 2: Feature Extraction

This phase is to generate features at the superpixel level, in preparation of the colorization phase (phase 3) using cascade feature matching.

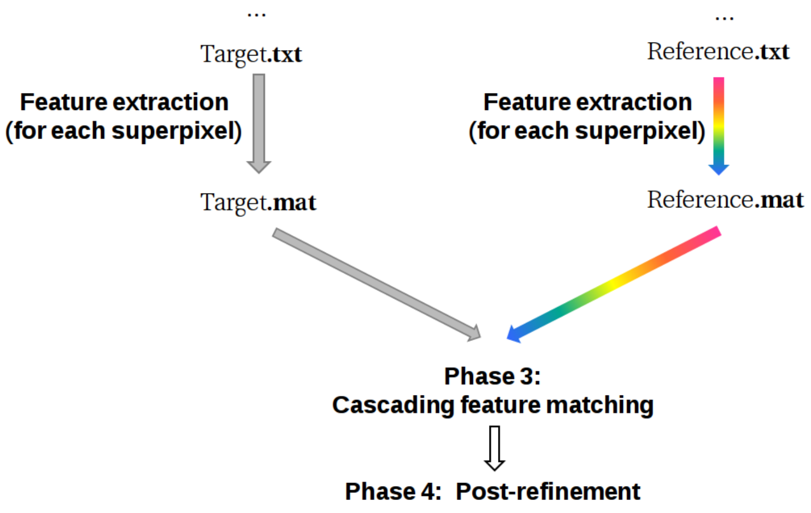


Fig. 4. Phase 2 demonstration.

We extract a 172-dimensional feature for each superpixel (both target and reference images), as is illustrated in Fig. 5.

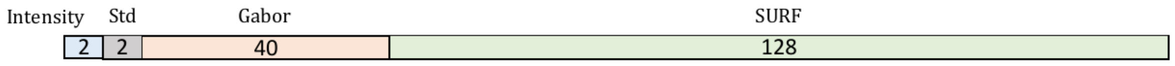


Fig. 5. 172-dimensional feature for each superpixel.

- **Intensity (2):** The first dimension is the average intensity of all pixels inside the superpixel S_i (n refers to the number of pixels within the superpixel):

$$f_1(i) = \frac{1}{n} \sum_{(x,y)\in S_i} I(x,y)$$

The second is the average f_1 of neighboring superpixels of S_i (N refers to how many neighboring superpixels that S_i has):

$$f_2(i) = \frac{1}{N} \sum_{\substack{j\in Neighboring \\ Superpixelsof S_i}} f_1(j)$$

A challenge here is how to determine a neighboring superpixel of S_i. Here we use an approximation by checking if two superpixels have "overlapped" part in row/col dimension — if so, the two are considered neighboring to each other.

- **Standard deviation (2):** Similar to two dimensions in "intensity", we are to calculate the first dimension of "standard deviation" feature, for a particular S_i, by taking the average of **every standard deviations for pixels inside** S_i:

$$g_1(i) = \frac{1}{n} \sum_{(x,y)\in S_i} Std\ (x,y)$$

And the second dimension:

$$g_2(i) = \frac{1}{N} \sum_{\substack{j\in Neighboring \\ Superpixelsof S_i}} g_1(j)$$

To compute $Std(x,y)$, the "standard deviation value (of intensity)" of each image pixel, we are to use a 5×5 square window centered at the corresponding pixel, which means:

$$Std(x,y) = \sqrt{\frac{1}{5\times 5} \sum_{(i,j)\in 5\times 5 neighbor} [MeanIntensity - I(i,j)]^2}$$

- **Gabor (5×8=40):** According to the paper, there should be 8 available "orientations" $\theta = \frac{n\pi}{8}$ (where $n = 0, \cdots, 7$) and 5 available exponential scales $e^{i\pi}$ (where $i = 0, \cdots, 4$), thus 40 possible combinations. We first compute the Gabor feature for each single **image pixel** and then to get the Gabor feature for each **superpixel** by taking the average of Gabor features of all pixels within the superpixel.

The Gabor kernel here is defined in [5] as:

$$g_{mn}(x,y) = a^{-m} \cdot G(\hat{x},\hat{y}), \quad a > 1$$

$$\begin{cases} \hat{x} & = a^{-m}(x\cos\theta + y\sin\theta) \\ \hat{y} & = a^{-m}(-x\sin\theta + y\cos\theta) \end{cases}$$

$m = 0, 1, \cdots$ and $\theta = \frac{n\pi}{K}$ with $n = 0, 1, \cdots$, K is the total number of orientations.

$G(u,v) = exp\left\{-\frac{1}{2}\,[\frac{(u-1/T)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}]\right\}$ is the Fourier transform of the original Gabor function $g(x,y)$.

1) Create a Gabor filter kernel using each of the 40 combinations of parameters;

2) Apply the Gabor filter through all pixels in a grayscale image (matrix) so get a new matrix (let it denoted as $Gb(x, y)$) with each element representing the "Gabor feature" of that pixel;
3) For each superpixel S_i, take the average of all $Gb(x, y)$ values within it to get S_i's Gabor feature G_i.
4) Switch to the next combination of parameters and get the next one of the 40-dimensional Gabor feature.

- **Extended SURF (128): S**peeded **U**p **R**obust **F**eatures is a robust local feature detector, first presented by Herbert Bay et al. [6] in May 2006. It is widely used in image registration, camera calibration, object recognition etc., to detect feature points and their descriptors.

1) For each superpixel, detect feature point(s) with corresponding 128-dimentional decriptor(s);
2) Take the average of those descriptor(s) to get the SURF feature of the superpixel.

We use the "OpenSurf" [7], the Matlab version of SURF implementation, to detect interest points and get their extended 128-dimensional descriptors. To generate more interest points in the whole image (so as to minimize the number of superpixels that have no interest point located in it), we change the "threshold" option from 2×10^{-4} to 1×10^{-6}.

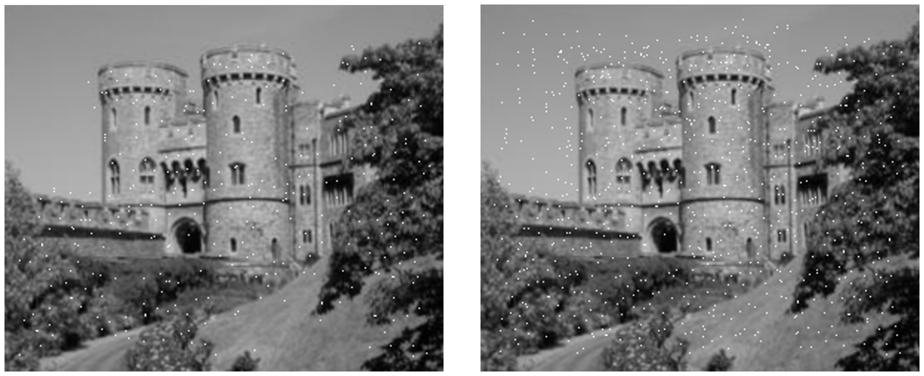
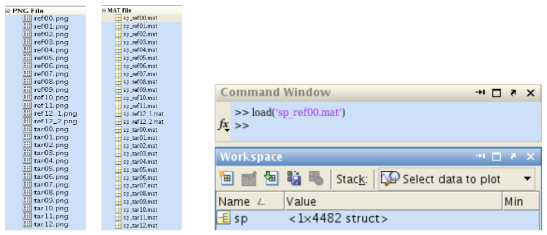


Fig. 6. More SURF interest points with a lower thredshold (312 to 753).

Finally, by the end of phase 2, we get a `.mat` file for each target/reference image, storing the information of its superpixels along with the features (as illustrated in Fig. 7).



```
ref00.png      sp_ref00.mat
ref01.png      sp_ref01.mat
ref02.png      sp_ref02.mat
ref03.png      sp_ref03.mat
ref04.png      sp_ref04.mat
ref05.png      sp_ref05.mat
ref06.png      sp_ref06.mat
ref07.png      sp_ref07.mat
ref08.png      sp_ref08.mat
ref09.png      sp_ref09.mat
ref10.png      sp_ref10.mat
ref11.png      sp_ref11.mat
ref12_1.png    sp_ref12_1.mat
ref12_2.png    sp_ref12_2.mat
tar00.png      sp_tar00.mat
tar01.png      sp_tar01.mat
tar02.png      sp_tar02.mat
tar03.png      sp_tar03.mat
tar04.png      sp_tar04.mat
tar05.png      sp_tar05.mat
tar06.png      sp_tar06.mat
tar07.png      sp_tar07.mat
tar08.png      sp_tar08.mat
tar09.png      sp_tar09.mat
tar10.png      sp_tar10.mat
tar11.png      sp_tar11.mat
tar12.png      sp_tar12.mat
```

```
>> load('sp_ref00.mat')
>>
```

Name	Value	Min
sp	<1x4482 struct>	

```
>> sp(1000)
ans =
   coordinates: [1x71 struct] % with fields 'row' and 'col'
   row_min: 96
   row_max: 108
   col_min: 435
   col_max: 445
   neighbors: [863 878 901 964 982 999 1059 1067 1109 1132 1133]
   feature: [1x172 double]
```

Fig. 7. Illustration of `.mat` file.

C. Phase 3: Cascade Feature Matching

There were two goals of this phase, the first of which was **to associate each target superpixel with a reference superpixel that is most similar to it**. The second goal was **to then take the colors of the reference pixels and colorize the target image using those colors**. For the most part, we kept the authors implementation of this phase and made some modifications to a specific step in the matching scheme.

The cascade matching scheme made use of the features that we extracted from the previous step. Essentially, we compared the feature vectors of the target and reference superpixels using the Euclidean distance measure formula below:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

where p and q were the nth-dimensional feature vectors for the target and reference superpixels, respectively. A short distance indicated similarity between the vectors.

However, if we were to go through each target superpixel and compare its entire feature vector to every superpixel in the reference image, we would be waiting hours for our matching scheme to complete. Instead, we took a cascading approach proposed by the authors, where we only compared a part of the feature vectors first, discarded reference superpixels with low similarity, and then compared a different part. Each part of the vector is a set of features for a specific feature type. Recall in the previous phase that we computed features for intensity, standard deviation, etc. and then consolidated all of that information into a 172-dimensional vector. Here, we broke that vector up again into individual feature type vectors.

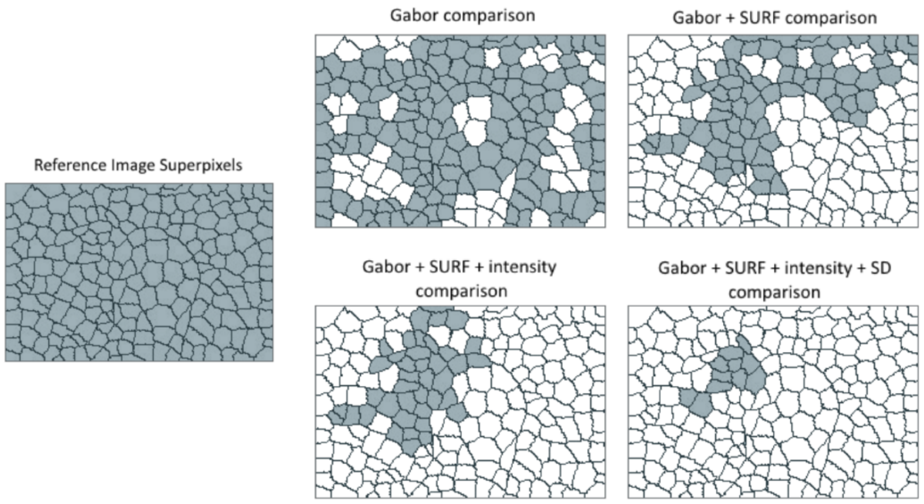

Reference Image Superpixels
Gabor comparison
Gabor + SURF comparison
Gabor + SURF + intensity comparison
Gabor + SURF + intensity + SD comparison

Fig. 8. Illustration of the cascading scheme.

The cascading approach significantly sped up the matching process because the set of reference superpixels it compares against the target superpixel in question would continuously be shrinking. We proved this to be true by implementing both the cascade approach and the "brute force" method described earlier. In running the code on a pair of images, we found that finding a match for one target superpixel would take approximately two seconds on average, whereas in our cascading approach finding a single match took a tiny fraction of a second. Both implementations are available in separate scripts for viewing.

In paper [1], the authors decided to compare feature types in the following order:

1) Gabor
2) SURF
3) Intensity
4) Standard deviation

The Gabor and SURF features were thought to have the highest potentially-discriminatory information out of the four types. **However, because our implementation of the SURF feature extractor generates a limited number of SURF "interest**

point"s, we are modifying this matching order to the following:

1) Standard deviation;
2) Intensity;
3) SURF;
4) Gabor.

(Add reason for this change here.)

In the final step of the cascade matching scheme, we would have a small set of reference superpixels that were most similar to the target superpixel in question. From this set, we would then choose a final reference suerpixel to be associated with the target superpixel. For each reference superpixel in the set, we computed the following formula:

$$a = \arg_b \min F(r_b, t_i), \quad r_b \in \Upsilon_i$$

$$F(r_b, t_i) = w_1 C_1(r_b, t_i) + w_2 C_2(r_b, t_i) + w_3 C_3(r_b, t_i) + w_4 C_4(r_b, t_i)$$

, where C_1, C_2, C_3, and C_4 are the Euclidean distances for the Gabor, SURF, intensity, and std. deviation features, respectively, and the corresponding w values are their weights. The authors set these weights to .2, .5, .2. and .1 respectively. Again, because of our implementation of the SURF feature extractor, we decided to cut some weight from the SURF features and distribute it amongst the other three feature types, with standard deviation receiving a majority of the distribution. The final weights that we used were **.1**, **.1**, **.3** and **.5**.

Once we computed the F values for every reference superpixel, we then ultimately chose the reference superpixel with the lowest F value, i.e. the shortest overall distance from the target superpixel. That reference superpixel is then associated with the target superpixel. We keep track of each association in an association array, which uses key-value pairs. The key would be the numeric identifier of the target superpixel, and the value would be the numeric identifier of the reference superpixel. This data structure helped make the colorization process simpler to implement.

After each and every target superpixel had been associated with a reference superpixel, we were ready to move on to actually colorizing the target image. For their experiment, the authors used a color interpolation algorithm created by another group of researches in a different study. The algorithm takes in the target image (kept in grayscale) and a variant of that image, which contains colored scribbles throughout it. The algorithm essentially expands those scribbles so that they eventually fill the entire image with color. We decided to keep this colorization approach because of its effectiveness and efficiency. We took the source code written for the algorithm and modified it slightly in order to more easily incorporate it into our project. The code can be found in the subdirectory labeled 'colorization'.

In order to create the variant of the target image that contains the colored scribbles, we first converted the target and reference images from RGB to the CIELab color space, just like what the authors did. The conversion code we used for this was taken from a Berkeley-hosted website, where it was uploaded for free use. The code can be found in the files 'RGB2Lab.m' and 'Lab2RGB.m'.

After converting to the CIELab color space, we then went through each target superpixel, retrieved its associated reference superpixel, and found the mean chroma values a and b of all of the pixels within the reference superpixel. We then took the middle pixel in the corresponding target superpixel and assign the mean chroma values to its chroma values. In other words, we were coloring only the middle pixel of each target superpixel and made those color dots our colored scribbles. After each target superpixel is given a colored dot, we then wrote the modified target image to a new file, which would be loaded into the color interpolation algorithm for final colorization.

D. Phase 4: Space Voting

The objective of this phase is to re-assign colors, namely to correct errors from phase 3. It is a post-refinement step that includes the following:

1) A superpixel should have the correct colors if neighboring superpixels of similar features have similar colors.

2) Segment image using mean shift algorithm by Comaniciu et al. [8].
3) Cluster each image segment using k-means clustering.
   - “Densely populated” = strong evidence that corresponding superpixels were correctly colored.
4) Count superpixels in each cluster to determine confidence level.
5) Low-confidence superpixels are reassigned new colors.

In consideration of 1) reducing the computational complexity (execution time) of the whole pipeline, and 2) time limitation scheduled for this project, we decide to skip this phase 4 proposed by the authors of [1], remedying it with more careful color assignment in phase 3.

III. Experiment

We test the colorization method on 13 pairs of images (target and reference) from figures 7, 8 and 10 in [1]. The last target image has two reference images to test the performance of using multiple reference images. The final weights and ordering of cascading are set as:

$$\begin{array}{ccccccc} \text{Standard Deviation} & \longrightarrow & \text{Intensity} & \longrightarrow & \text{SURF} & \longrightarrow & \text{Gabor} \\ 0.5 & & 0.3 & & 0.1 & & 0.1 \end{array}$$

A. *Default and custom settings for cascading*

target	Default settings in [1]

B. Multiple exemplars (reference images)

A direct extension of this colorization method is using multiple exemplars (reference images) for one target image [1]. It can be done by:

- Building the "reference superpixel pool" from the multiple reference images (done in [1]), or
- Simply colorizing the target with each reference respectively, and then to take an average (or a linear combination) of all colorized images. An example of this is tested and illustrated in Fig. 9, where the improvement is achieved after taking the average of the two (separate) results.

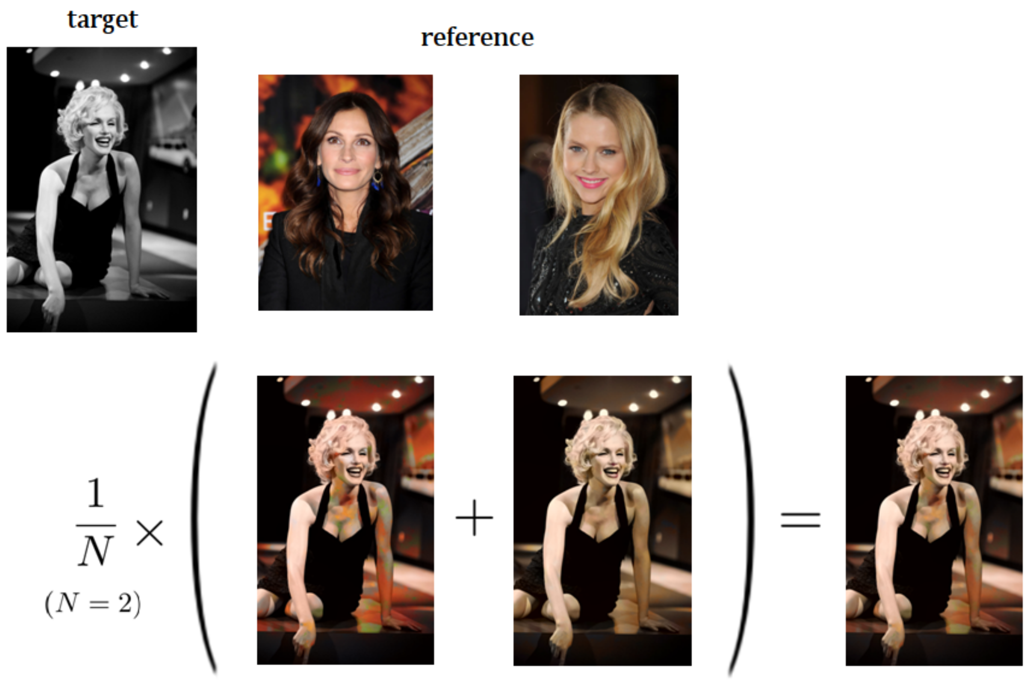


Fig. 9. Example of using multiple exemplars.

IV. Challenges and Future Work

A. Challenges

We encountered some challenges along the way breaking down this colorization method. These challenges/problems are overcome, or simplified so as to move on to the next phase.

- Identifying superpixels:
  1) **Choosing a proper number of average number of pixels per superpixel** — We tested this parameter on several options around 40 and ended up setting it to 30.
  2) **Separating each superpixel (identifying connected components) based on the boundary coordinates** — This was done with a recursive function performed on the whole boundary (logic, 0-1) matrix.
- Feature extraction:
  1) **Identifying the neighboring superpixel of each superpixel** — We achieved this by using an approximation that checks if the rows/columns coordinates of two superpixels overlap.
  2) **Building the Gabor kernel based on the "orientation" and "exponential scale" settings proposed in [1], so as to generate the 40-dimensional Gabor feature** — This definition of spacial Gabor kernel is specified in [5].
  3) **Generating more "interest point"s and their descriptors when applying SURF algorithm to the whole image** — We modified the threshold in SURF calculation to get more interest points.
- Cascading feature matching:

  Implementing the cascading feature matching scheme — This took us a couple of iterations of code to figure out the correct way to do this. The first version did not actually reduce the search space as described by the authors in the paper; instead, it flagged bad-matching reference superpixels so that they would be ignored throughout the remainder of the matching process. We managed to implement the cascade matching scheme with success but spent a significant amount of time thinking up a solution.
- Colorizing each superpixel:

  Coming up with own colorization process – We attempted to write our own algorithm for the colorization process instead of using the proposed algorithm by the authors. However, we found that our attempt did not yield as good of a result. The proposed algorithm ended up producing much more realistic colors than our own algorithm, so we scrapped our code and adopted that algorithm.

B. Future Work

The first addition to this work is to modify the feature extraction part (on each superpixel), namely to extend the 172-dimensional feature with some other discriminative feature(s), such as Histogram of Oriented Gradients (HOG) and Haar-like features.

Another issue to solve is to introduce more interest points in SURF feature extraction, to have fewer superpixel with '0's at the SURF part. However, if we loose the constraint or lower the threshold of the SURF algorithm, it may give out bad descriptors for those interest points because the distinctions among all descriptors are less significant.

Lastly, a proper post-refienment step, such as the space voting approach for color reassignment, will likely fix some problems (e.g. color bleeding) in colorized images.

Conclusion

to be written, probably along with "Abstract"

References

[1] Raj Kumar Gupta1, Alex Yong-Sang Chia et. al., *Image Colorization Using Similar Images*. MM'12 Proceedings of the 20th ACM international conference on Multimedia, pp. 369-378.

[2] Terry Grant, *Adding color to your drawings*. July 7, 2012. http://penpencilpaperdraw.blogspot.com/2012/07/adding-color-to-your-drawings.html

[3] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. *Turbopixels: Fast Superpixels Using Geometric Flows*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(12):22902297, 2009.

[4] Alex Levinshtein's homepage. http://www.cs.toronto.edu/~babalex/research.html

[5] B. S. Manjunath and W. Y. Ma. *Texture Features For Browsing And Retrieval Of Image Data*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(8):837842, 1996.

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, *SURF: Speeded Up Robust Features*. Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346-359, 2008.

[7] Dirk-Jan Kroon. *OpenSURF (including Image Warp)*. http://www.mathworks.com/matlabcentral/fileexchange/28300-opensurf--including-image-warp-July 26, 2010.

[8] D. Comaniciu and P. Meer. *Mean shift: A robust approach toward feature space analysis*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(05):603619, 2002.