Coursework Assignment C -2017

CS4125 Seminar Research Methodology for Data Science

Group 12: Lu Liu (4621832), Yadong Li(4608283), Ziqi Wang(4590058)

April 4, 2017

Task 1: Gradient-based Image Sharpening

In this method, the input is a image of $h \times w$ size, parameter c_s and $c_{\bar{U}}$, and the output is a sharped image. To simplify the calculation with values of pixels, we reshape all the pixels into a vector $\bar{U}$ of $l = h \times w$ length by ordering them column after column. To implement this method, firstly gradient matrix should be constructed. The function *gradient* in *gradient.m* deals with this problem.

The number of elements in its gradient vector is $g_{size} = (h-1) \times w + (w-1) \times h$. Since $g = G\bar{U}$, G is a $g_{size} \times l$ matrix. In this function we construct G as a sparse matrix with $2 \times g_{size}$ elements with values 1 or -1, while the others are 0. Start with the vertical gradient, for every pixel i except the pixels on the bottom line in the image, its gradient $g_i = \bar{U}_i - \bar{U}_{i+1}$. Thus in G, the $G_{ai} = 1$ and $G_{ai+1} = -1$, where a is the index of the corresponding gradient. For horizontal gradient, for every pixel i except the pixels on the right edge of the image, its gradient $g_i = \bar{U}_{i+h} - \bar{U}_i$. Thus in G, the $G_{ai} = -1$ and $G_{ai+h} = 1$. Now we have gradient matrix G, gradient vector can be obtained by $\bar{g} = G\bar{U}$. To get the sharped pixel values U, we need to solve

$$\min_U(||GU - c_s\bar{g})||^2 + c_{\bar{U}}||U - \bar{U}||^2$$

. The minimum is the solution of the linear system

$$(G^TG + c_{\bar{U}}I)U = c_sG^T\bar{g} + c_{\bar{U}}\bar{U}$$

where I is a $l \times l$ sparse identity matrix. After we obtain U, U is reshaped back to a new $h \times w$ image. In Figure 6, (b) is the sharped image obtained.

(a) Inital image

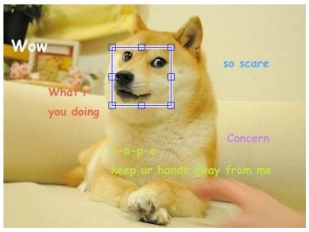
(b) Sharped image with $c_s = 3$, $c_{\bar{U}} = 0.5$

Figure 1: Result of Gradient-based Image Sharpening

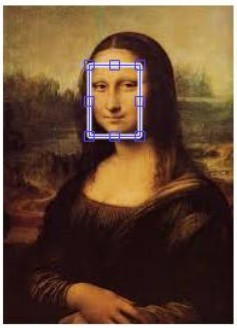
Task 2: Gradient-based Image Blending

Single Source

In this task we designed an application with which the user can select a rectangular region with random size and random location in the source image as shown in figure 2(a). Then the user can also adjust the size and location after select it. Afterwards, user is allowed to select the region of interest in the destination image(as shown in figure 2) since sometimes the size of selected region in the source image is not desirable. So. with this step, the source region can be adapted to the same size of the region selected in the destination image as shown in figure 3(a). After blending, the final result is shown in figure 3.

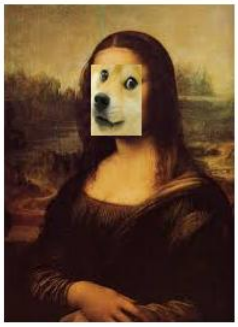


(a) Select region in source image



(b) Select region in destination image

Figure 2: Selection of region of interest in both source image and destination image



(a) Direct clone

(b) Blending

Figure 3: Result of direct cloning and blending

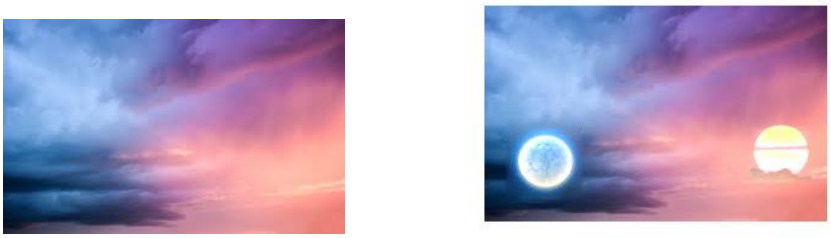
Multiple sources

Our user interface also allow the user to select different objects from different source images randomly and put them into the same destination image as shown in figure 5.

(a) moon

(b) sun

Figure 4: Multiple sources



(a) destination image

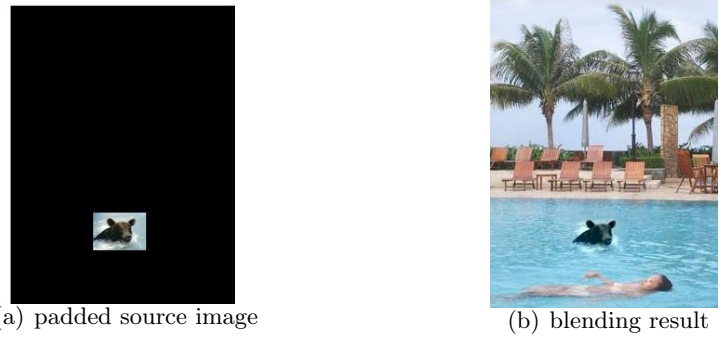
(b) blending

Figure 5: Multiple sources

Difficulties and solution

- We tried to implement some more practical blending option like selecting source object in the source image randomly and selecting the destination location and size randomly. However, at first we have no idea on how to change the size and move the object to the desired location.

  Solution: Use "imresize" to resize the source image and use "padarray" to make an extended source image(like a "mask" but not) which is used as the source image, see figure 7(a).



(a) padded source image

(b) blending result

Figure 6: Result of Gradient-based Image Blending

- Find the select matrix. At first we tried to get the S matrix by multiplying the pixel vector of source image and its transpose matrix, however it did not work.

  Solution: We used the sparse matrix for S which actually works in the same way as the gradient matrix G.

Bonus: Colorization

Implementation of Colorization

In this method, the input is a image of $h \times w$ size whose only minority of pixels have color, while the others are gray. Similar to all the tasks above, first, the image is reshape to a vector $\bar{U}$ of $l = h \times w$ length. Then map the RGB values to YIQ with Y for luminance, $\bar{I}$ and $\bar{Q}$ for chrominance. For the output image, we want the optimized I and Q. To find the color values I_i such that

$$E(I) = 1/2 \sum_{i=1}^{n} (\sum_{j \in N(i)} w_{ij}(I_i - I_j))^2$$

is minimal and $I_i = \bar{I}_i$ for all $i \in C$. Here $N(i)$ is the set of neighbors of pixel i, C is the set of pixels with colors and $\bar{I}_i$ and $\bar{Q}_i$ are the given color values. w_{ij} is the weight for i of its neighbor j. It can be calculated by

- $\tilde{w}_{ij} = e^{-(Y_i - Y_j)^2/\sigma_i}$, where σ_i) is the variance of luminance around i
- $w_{ij} = \tilde{w}_{ij}/(\sum_{j \in N(i)} \tilde{w}_{ij})$

In this implementation, we only consider the soft constrains that constrains are not exactly satisfied $I_i = \bar{I}_i$ for all $i \in C$, but only in least squares sense. Thus we add the term $a \sum_{i \in C} (I_i - \bar{I}_i)^2$ to the objective, where $a > 0$ is a weight. The results can be obtained by solving the linear system

$$(L^T L + aS^T S)I = aS^T \bar{I}$$

Here L is a $n \times n$ matrix, where $n = h \times w$, with

- $L_{ii} = 1$
- $L_{ij} = -w_{ij}$ for $j \in N(i)$
- $L_{ij} = 0$ for other entries

In this case, we construct L as a sparse matrix. Since the pixels inside have 8 neighbors while pixels on the edge and corner have 5 or 3 neighbors, when assigning the values of L, we assign them referring to different types of pixels. To know the type of a certain pixel, we can identity it by its index according to Table 1.

Table 1: Types of pixels

Type	index m	Number of neighbors
Corner	$m == 1, m == h,$ $m == h*(w-1)+1, m == h*w$	3
Edge	$m > 1 \&\& m < h, m > (h*(w-1)+1) \&\& (m < h*w),$ $mod(m,h) == 1, mod(m,h) == 0$	5
Inside	others	8

Then we need to construct the selector matrix S to selects the pixels that are constrained. The size of S is $m \times n$, where m is the number of colored pixels and n is the number of all pixels. We've tried to use both hard constraint and soft constraint to solve the optimization problem. For images with large size, we also tried to solve the linear equation by LU factorization. However, we didn't manage to get a good result. The new image we got didn't change much, as shown in Figure 7.

(a) source image (b) result

Figure 7: Result of colorization

Summary

To summarize, in this assignment, we managed to apply optimization algorithms that we've learned in the lecture to real life cases. We are glad that we have learned and practiced some new concepts and algorithms, such as sparse matrix, optimization concept and so on.

Lu Liu

Gradient based image sharpening and colorization
solve the gradient matrix

Yadong Li

Solving linear Equation
colorization

Ziqi Wang

Gradient based single and multiple images blending.
Derive the select matrix, inner and boundary gradient
User Interface