图像处理与建模课程作业

《Fast and Effective L_0 Gradient Minimization by Region Fusion》

[ICCV2015]　Rang，　Michael

论文赏析与实现

宋 杰 11521045

1 引言

L0 梯度最小化算法可以应用在信号处理方面，以控制信号中非零梯度的数量。这在平滑信号，去除信号中的噪音，同时又保持信号中的重要信息有着非常关键的作用。L0 梯度最小化在图像去燥，3D mesh 去燥,图像增强方面都有着广泛的应用。然而，因为此问题的非凸性，最小化 L_0 范数是一个 NP hard 的问题。因此当前方法都是用估计的策略来解决此问题。在这篇论文中，作者提出了一个新的能够快速有效的解决 L0 梯度最小化问题。该方法是用了一种基于区域融合的目标函数下降算法，该算法收敛速度比其他算法更快并且能够更好的估计最佳的 L0 范数。
大量实验证明了该算法的有效性。

2 L0 梯度最小化

令 I 代表源信号，S 代表滤波过的信号。S 的梯度为▽S ，L0 梯度最小化的目标函数为：

$$F = \min_S ||S - I||^2 + \gamma||\nabla S||_0 \qquad (1)$$

其中γ控制两项的权重，该值越大，S 中梯度越小，结果越粗糙。

公式（1）可以改写成如下形式：

$$F = \min_S \sum_i^M \left[||S_i - I_i||^2 + \frac{\gamma}{2}\sum_{j\epsilon N_i} ||S_i - S_j||_0\right] \qquad (2)$$

其中 M 代表信号长度，N_i代表第 i 个元素的相邻元素集合。这里，γ被除以 2，因为 i 和 j 的相邻关系在这种情况下被计算了 2 次。相邻集合对于一维信号，二维信号，三维信号定义如下：

$$N_i = \begin{cases} \{i-1, i+1\} & 1D \\ \{4-\text{connected pixels}\} & 2D \\ \{\text{all neighbor faces of the } i^{th} \text{ face}\} & 3D \end{cases}$$

(3)

3 优化方法

由于目标函数的非凸性，优化目标函数是 NP 难的问题。作者提出一种基于区域融合的目标估计算法。在每一次迭代优化的过程中，仅仅考虑一对相邻的区域集，而不是整个信号。对于两个相邻元素，他们对目标函数的贡献表达如下：

$$f = \min_{S_i,S_j} ||S_i - I_i||^2 + ||S_j - I_j||^2 + \gamma ||S_i - S_j||_0$$

(4)

我们的目的是寻找最小的 S_i 和 S_j 来最小化子函数 f。把问题分解成 2 种情况：

- Case $S_i \neq S_j$：公式（4）变为：

$$f = \min_{S_i,S_j} ||S_i - I_i||^2 + ||S_j - I_j||^2 \quad (5)$$

  在这种情况下，我们有平凡解：

$$\begin{cases} S_i = I_i, \quad S_j = \\ f = \gamma \end{cases} \quad (6)$$

- Case $S_i = S_j$：公式（4）变为

$$f = \min_{S_i,S_j} ||S_i - I_i||^2 + ||S_j - I_j||^2 \quad （7）$$

  通过求解导数，该情况下的解为：

$$\begin{cases} S_i = S_j = (I_i + I_j) \\ f = (I_i + I_j)^2 / \end{cases} \quad (8)$$

  综合两种情况，公式（4）的近似解为：

$$\{S_i, S_j\} = \begin{cases} \{A, A\} & \text{if } \frac{\|I_i - I_j\|^2}{2} : \\ \{(I_i, I_j\} & \text{other} \end{cases} \tag{9}$$

其中，A = （$I_i + I_j$）/2.

4 算法流程

Our algorithm loops through all groups of a current filtered signal. For each group G_i, we consider its neighbors G_j. Like prior methods, we use an auxiliary parameter β ($0 \leq \beta \leq \lambda$) that increases for each iteration. Details to this parameter are provided in Section 3.2. Factoring in the auxiliary parameter, Equation 4 becomes as follows:

$$\min_{S_i, S_j} w_i \|S_i - Y_i\|^2 + w_j \|S_j - Y_j\|^2 + \beta c_{i,j} \|S_i - S_j\|_0. \tag{11}$$

Recall that Y_i and Y_j represent the mean signal values for the groups G_i and G_j containing w_i and w_j elements respectively. The above equation can be solved in the exact same manner as described for Equation 4 as follows:

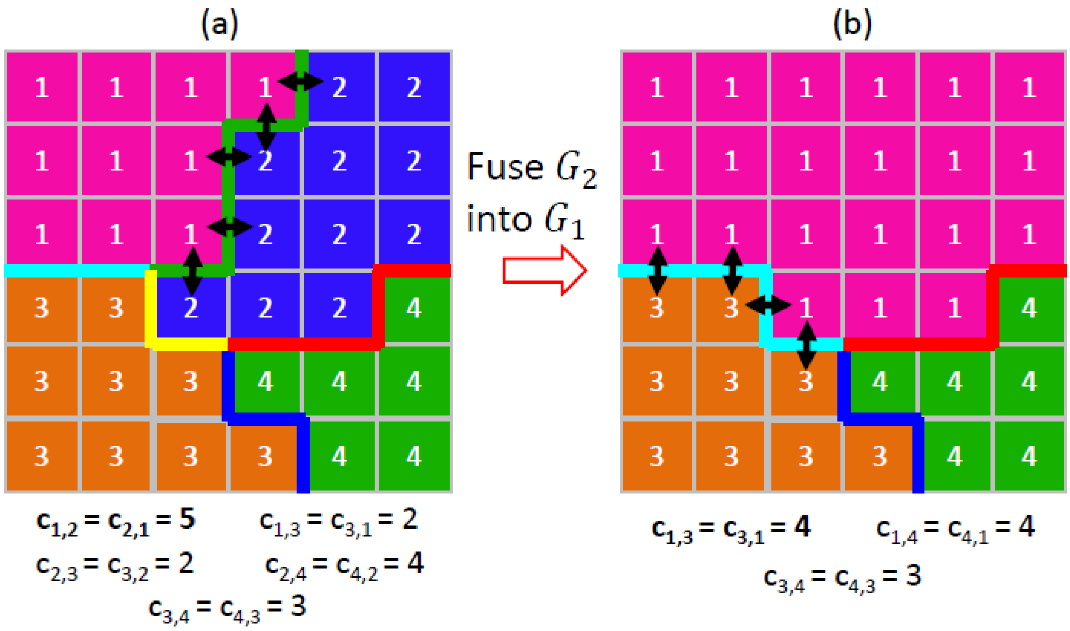


Figure 2. This figure shows an example of the connection numbers for a 2D image. (a) shows the initial configuration with four groups of pixels, while (b) shows the configuration after fusing group G_2 into G_1. The numbers below each image show the corresponding connection numbers for each pair of neighboring groups.

$$\{S_i, S_j\} = \begin{cases} \{B, B\} & \text{if } w_i w_j ||Y_i - Y_j||^2 \leq \beta c_{i,j}(w_i + w_j) \\ \{Y_i, Y_j\} & \text{otherwise} \end{cases} \tag{12}$$

where $B = (w_i Y_i + w_j Y_j)/(w_i + w_j)$ is the weighted average of the two groups G_i and G_j.

算法伪代码

Algorithm 1 Region Fusion Minimization for L_0

Input: signal I of length M, the level of spareness λ

$G_i \leftarrow \{i\}, Y_i \leftarrow I_i, w_i \leftarrow 1$
Initialize N_i as Equation 3
Initialize $c_{i,j}$ as Equation 10
$\beta \leftarrow 0,\ \ iter \leftarrow 0,\ \ P \leftarrow M$
repeat
  $i \leftarrow 1$
  while $i \leq P$ **do**
    for all $j \in N_i$ **do**
      if $w_i w_j ||Y_i - Y_j||^2 \leq \beta c_{i,j}(w_i + w_j)$ **then**
        $G_i \leftarrow G_i \cup G_j$
        $Y_i \leftarrow (w_i Y_i + w_j Y_j)/(w_i + w_j)$
        $w_i \leftarrow w_i + w_j$
        Remove j in N_i and delete $c_{i,j}$
        for all $k \in N_j \setminus \{i\}$ **do**
          if $k \in N_i$ **then**
            $c_{i,k} \leftarrow c_{i,k} + c_{j,k}$
            $c_{k,i} \leftarrow c_{i,k} + c_{j,k}$
          else
            $N_i \leftarrow N_i \cup \{k\}$
            $N_k \leftarrow N_k \cup \{i\}$
            $c_{i,k} \leftarrow c_{j,k}$
            $c_{k,i} \leftarrow c_{j,k}$
          end if
          Remove j in N_k and delete $c_{k,j}$
        end for
        Delete G_j, N_j, w_j
        $P \leftarrow P - 1,\ \ i \leftarrow i + 1$
      end if
    end for
  end while
  $iter \leftarrow iter + 1$
  $\beta \leftarrow g(iter, K, \lambda)$ ▷ Defined in Equation 13
until $\beta > \lambda$

for $i = 1 \rightarrow P$ **do** ▷ Reconstruct the output signal
  for all $j \in G_i$ **do**
    $S_j \leftarrow Y_i$
  end for
end for

Output: filtered signal S of length M

5 算法实现

为了重现论文方法，本人采用 python 语言。**注意，虽然在作者个人主页上可以找到该论文的 C++版本实现，但是本人从未下载成功。为了不引起抄袭的嫌疑，本人采用 python 语言实现，并且没有参考作者源代码（因为压根就没下载下来）。**代码总长度 100 来行，由于 python 语言的特性以及本人编程在某些代码优化上做的不够好，速度上并没有达到论文中讲的那样能够一秒钟处理一张 600*400 的图片那么快。本人实现版本速度上大约慢作者 3 倍。处理效果上几乎同作者论文中的描述。

6 实验结果

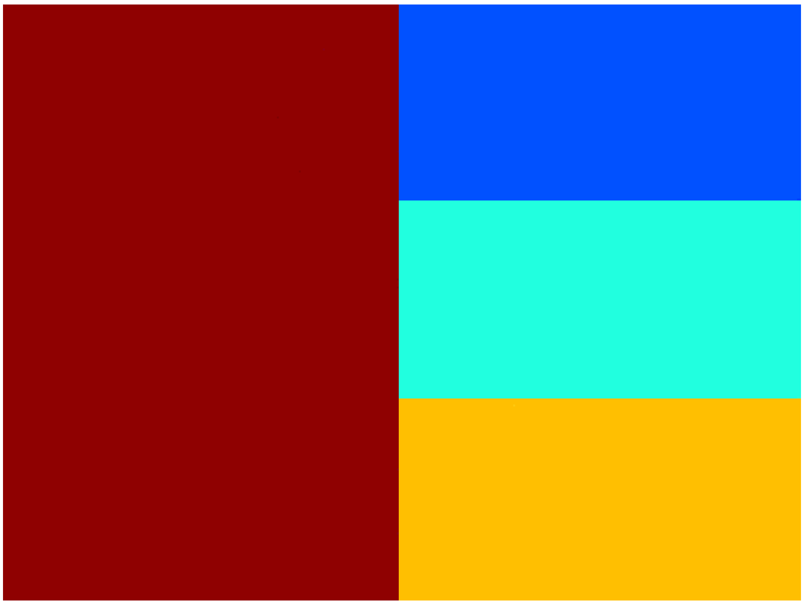
color quantization （该图片是本人随机选取的图片）

原图 $\gamma=100$

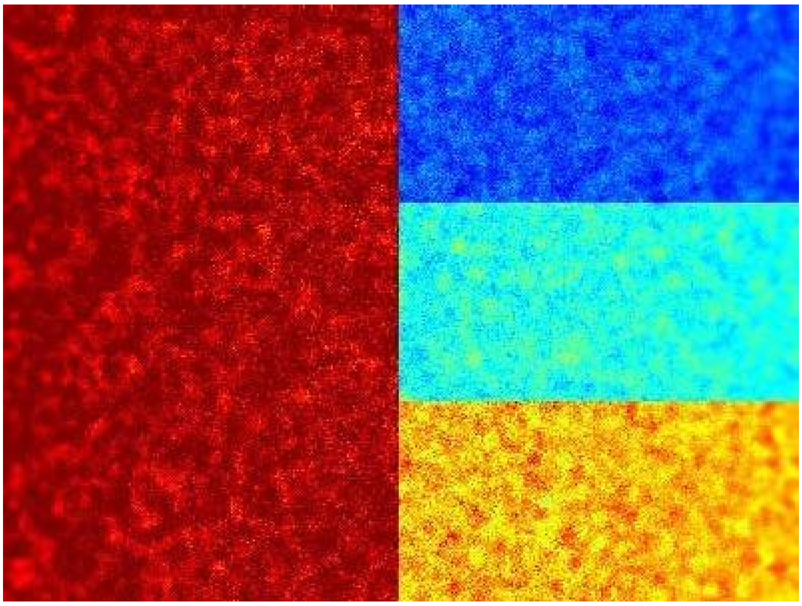
$\gamma=1000$ $\gamma=5000$

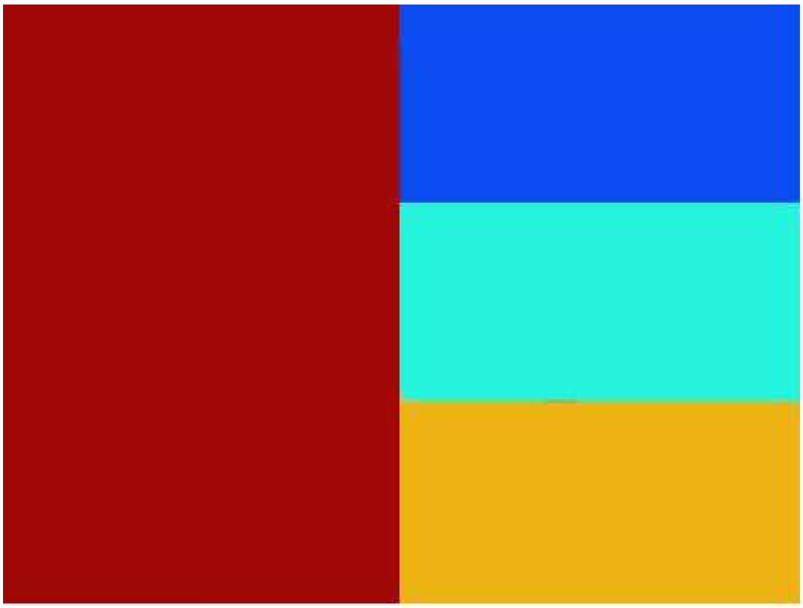
γ =10000 γ =20000

denoise(该图片截取论文中图片，然后在自己代码上跑出的结果)



Groundtruth


噪音图片


在本人代码上跑出的效果图

7. 代码

```
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
from copy import deepcopy


class Group:
    def __init__(self, id_, element, image, im_shape):
        self.__id = id_
```

```
        self.__elements = []
        self.__connections = {}
        self.__elements.append(element)
        self.__mean = np.float64(image[element[0], element[1], ...])
        self.__valid = 1
        if element[0] > 0:
            self.__connections[self.__id - im_shape[1]] = 1
        if element[0] < im_shape[0] - 1:
            self.__connections[self.__id + im_shape[1]] = 1
        if element[1] > 0:
            self.__connections[self.__id - 1] = 1
        if element[1] < im_shape[1] - 1:
            self.__connections[self.__id + 1] = 1

    def union(self, grp):
        for i in grp.get_elements():
            self.__elements.append(i)

    def get_elements(self):
        return self.__elements

    def get_num_elements(self):
        return len(self.__elements)

    def add_connection(self, index, conn):
        if index not in self.__connections.keys():
            self.__connections[index] = conn
        else:
            self.__connections[index] += conn

    def set_connection(self, index, conn):
        self.__connections[index] = conn

    def rm_connection(self, index):
        self.__connections.pop(index)

    def get_connection(self):
        return self.__connections

    def set_mean(self, mean):
        self.__mean = mean

    def get_mean(self):
        return self.__mean
```

```python
    def get_id(self):
        return self.__id

    def invalid(self):
        self.__valid = 0

    def is_valid(self):
        return self.__valid


def region_fusion_minimization(signal, lamda):
    '''
    Region Fusion Minimization algorithm.
    :param signal: Signal I of length M
    :param lamda: Sparseness parameter
    :return: Filtered signal S of length M
    '''

    im_shape = signal.shape
    signal_copy = deepcopy(signal)
    # =========== Initialize ================
    # initialize groups
    groups = []
    for y in range(im_shape[0]):
        for x in range(im_shape[1]):
            # Note: element [y, x]
            grp = Group(x + y*im_shape[1], [y, x], signal_copy, im_shape)
            groups.append(grp)

    # initialize beta, iter
    beta = 0
    iter = 0

    while beta < lamda :
        i = 0
        while i < len(groups):
            if not groups[i].is_valid():
                i += 1
                continue
            for j in list(groups[i].get_connection()):
                if not groups[j].is_valid():
                    continue
                wi = groups[i].get_num_elements()
```

```
                wj = groups[j].get_num_elements()
                yi = groups[i].get_mean()
                yj = groups[j].get_mean()
                cij = groups[i].get_connection()[groups[j].get_id()]
                if wi*wj*np.sum((yi - yj)**2) <= beta*cij*(wi + wj):
                    # Fusion two groups
                    groups[i].union(groups[j])
                    groups[i].set_mean((wi*yi + wj*yj)/(wi + wj))
                    groups[i].rm_connection(j)
                    groups[j].rm_connection(i)
                    for k in groups[j].get_connection().keys():
                        if k in groups[i].get_connection().keys():
                            groups[i].add_connection(k,
groups[j].get_connection()[k])
                            groups[k].add_connection(i,
groups[j].get_connection()[k])
                        else:
                            groups[i].set_connection(k,
groups[j].get_connection()[k])
                            groups[k].set_connection(i,
groups[j].get_connection()[k])

                        groups[k].rm_connection(j)
                    groups[j].invalid()

            i += 1
        iter += 1
        beta = (iter/30)*lamda

        print('beta : {}'.format(beta))

    return reconstruct_output(groups, signal)


def reconstruct_output(groups, signal):
    S = np.zeros(signal.shape)

    print('This is {} pixels in all'.format(signal.size))
    count = 0
    num = 0
    for group in groups:
        if group.is_valid():
            num += 1
            for j in group.get_elements():
```

```
            count += 1
            S[j[0], j[1], ...] = group.get_mean()

    print('Count = {}, Num = {}'.format(count, num))
    return S


img_dir2 = 'C:\\Users\\Dell\\Desktop\\tmp\\noise.png'
img = io.imread(img_dir2)

img2 = np.uint8(region_fusion_minimization(img, 20000))
plt.imshow(img2)
plt.axis('off')
io.imsave('C:\\Users\\Dell\\Desktop\\tmp\\esss.jpg',img2)
plt.show()
```