计算机视觉实验报告



作业一

**学 院 智能与计算学部**

**专    业 软件工程**

**姓 名 白文杰**

**学 号 3018216031**

**年 级 2018**

**班    级 1**

# Part 1 OpenCV内部函数实现

## 使用OpenCV中对应的API即可，使用到的几个API如下：

* GaussianBlur —— 高斯滤波
* medianBlur —— 中值滤波
* bilateralFilter —— 双边滤波
* filter2D —— 卷积运算

前3个函数没有什么可以述说的，而最后一个函数filter2D我用来提升图像的对比度的。

虽然模糊后有了美颜的模糊效果，但是图像并不清晰，虽然我并不清楚美颜的具体细节还有哪些，但是我想让我的美颜图片变的清晰一点。刚好老师提到了锐化的滤波器，即有负值的滤波器。于是我便想使用锐化滤波器使我的图像对比度增强一些。

我定义的滤波器如下：

|  |  |  |
| --- | --- | --- |
| 0 | -1 | 0 |
| -1 | 5 | 0 |
| 0 | -1 | 0 |

## 效果图：

1. 原图：  
   注：原图来自于AI生成人脸网站：<https://thispersondoesnotexist.com/>  
   
2. 高斯滤波  
   
3. 高斯滤波锐化  
   
4. 中值滤波  
   
5. 中值滤波锐化  
   
6. 双边滤波  
   
7. 双边滤波锐化  
   

## 结果分析：

可以看出双边滤波后再进行锐化的效果最好。

可能的原因：

1. 双边滤波是非线性滤波
2. 保留了边缘信息

# Part 2 高斯滤波和中值滤波手动实现

高斯滤波整体实现方法是一样的，区别就在于高斯滤波的卷积核使用的函数是高斯函数，而中值滤波使用的是所有数字取中值。

### 效果图：

1. 原图：



1. 高斯滤波：



1. 高斯滤波锐化：



1. 中值滤波：



1. 中值滤波锐化：



## 结果分析：

在实验中，我发现卷积核不能太大，太大的效果并不理想，会将斑点放大。如下图：



# Part 3 双边滤波的手动实现

在手动实现双边扫描的时候，我对空间和相似度求了权值，然后对他们进行了双边扫描计算。

### 效果图：

1. 原图：



1. 双边滤波：



1. 双边滤波锐化：



## 结果分析：

相比与自己手动实现的高斯滤波和中值滤波，双边滤波的效果（不锐化）还是很不错的，有了美颜的效果。

整体来说，手动实现的效果并不如OpenCV的内置函数效果好。我猜想是因为内置的函数还有其他的一些处理或者条件，但是还未深入研究。而手动实现的过程中，也让我对几种滤波的过程有了更多的了解。

# 代码

1. 使用OpenCV内置函数（UseOpenCVMethods.cpp）

|  |
| --- |
| #include<opencv2/opencv.hpp>  //#include<iostream>  **using** **namespace** cv**;**  int main**(**int argc**,** char**\*\*** argv**)** **{**  Mat image **=** imread**(**"images/image.jpg"**);**  **if** **(**image**.**empty**())** **{**  printf**(**"could not load image...\n"**);**  **return** **-**1**;**  **}**  //namedWindow("test\_opencv\_setup", 0);  //imshow("test\_opencv\_srtup", image);  Mat dst**;**  Mat dstf**;**  Mat kernel **=** **(**Mat\_**<**char**>(**3**,** 3**)** **<<** 0**,** **-**1**,** 0**,** **-**1**,** 5**,** **-**1**,** 0**,** **-**1**,** 0**);**  bilateralFilter**(**image**,** dst**,** 10**,** 100**,** 5**);**  //imshow("dst", dst);  filter2D**(**dst**,** dstf**,** dst**.**depth**(),** kernel**);**  imwrite**(**"images/1/image\_bilateralFilter.jpg"**,** dst**);**  imwrite**(**"images/1/image\_bilateralFilter\_filter.jpg"**,** dstf**);**  medianBlur**(**image**,** dst**,** 9**);**  //imshow("dst", dst);  filter2D**(**dst**,** dstf**,** dst**.**depth**(),** kernel**);**  imwrite**(**"images/1/image\_medianBlur.jpg"**,** dst**);**  imwrite**(**"images/1/image\_medianBlur\_filter.jpg"**,** dstf**);**  GaussianBlur**(**image**,** dst**,** Size**(**11**,** 11**),** 3**,** 3**);**  filter2D**(**dst**,** dstf**,** dst**.**depth**(),** kernel**);**  imwrite**(**"images/1/image\_GaussianBlur.jpg"**,** dst**);**  imwrite**(**"images/1/image\_GaussianBlur\_filter.jpg"**,** dstf**);**    //waitKey(0);  **return** 0**;**  **}** |

1. 手动实现（UseMyMethods.cpp）

|  |
| --- |
| #include <opencv2/opencv.hpp>  #include <algorithm>  #include <vector>  **using** **namespace** cv**;**  **using** **namespace** std**;**  // 获取中值  double sort\_color**(**vector**<**double**>** list**);**  /\*  \* 中值滤波  \* src是输入图像  \* dst是输出图像  \* size是滤波器的大小，只能够为奇数  \*/  void myMedianBlur**(**const Mat src**,** Mat**&** dst**,** int size**);**  double**\*\*** getGaussionArray**(**int arr\_size**,** double sigma**);**  /\*  \* 高斯滤波  \* src是输入图像  \* dst是输出图像  \* size表示滤波器的大小  \* sigma表示高斯函数方差  \*/  void myGaussianBlur**(**Mat src**,** Mat**&** dst**,** int size**,** double sigma**);**  /\* 计算空间权值 \*/  double**\*\*** get\_space\_Array**(**int \_size**,** int channels**,** double sigmas**);**  /\* 计算相似度权值 \*/  double**\*** get\_color\_Array**(**int \_size**,** int channels**,** double sigmar**);**  /\* 双边 扫描计算 \*/  void doBialteral**(**cv**::**Mat**\*** \_src**,** int N**,** double**\*** \_colorArray**,** double**\*\*** \_spaceArray**);**  /\*  \* 双边滤波  \*  \*/  void myBialteralFilter**(**cv**::**Mat**\*** src**,** cv**::**Mat**\*** dst**,** int N**,** double sigmas**,** double sigmar**);**  int main**()** **{**  Mat image **=** imread**(**"images/image.jpg"**);**  **if** **(**image**.**empty**())**  **{**  printf**(**"could not load image\n"**);**  **return** **-**1**;**  **}**  Mat dst**;**  Mat dstf**;**  Mat kernel **=** **(**Mat\_**<**char**>(**3**,** 3**)** **<<** 0**,** **-**1**,** 0**,** **-**1**,** 5**,** **-**1**,** 0**,** **-**1**,** 0**);**  cout **<<** "Program is running, please wait for a moment." **<<** endl**;**  //myGaussianBlur(image, dst, 3, 20);  //imwrite("images/2/image\_GaussianBlur.jpg", dst);  //filter2D(dst, dstf, dst.depth(), kernel);  //imwrite("images/2/image\_GaussianBlur\_filter.jpg", dstf);  myMedianBlur**(**image**,** dst**,** 3**);**  imwrite**(**"images/2/image\_MedianBlur.jpg"**,** dst**);**  filter2D**(**dst**,** dstf**,** dst**.**depth**(),** kernel**);**  imwrite**(**"images/2/image\_MedianBlur\_filter.jpg"**,** dstf**);**  //myBialteralFilter(&image, &dst, 25, 12.5, 50);  //imwrite("images/2/image\_BialteralFilter.jpg", dst);  //filter2D(dst, dstf, dst.depth(), kernel);  //imwrite("images/2/image\_BialteralFilter\_filter.jpg", dstf);  waitKey**(**0**);**  **return** 0**;**  **}**  double**\*\*** getGaussionArray**(**int arr\_size**,** double sigma**)**  **{**  int i**,** j**;**  double**\*\*** array **=** **new** double**\*** **[**arr\_size**];**  **for** **(**i **=** 0**;** i **<** arr\_size**;** i**++)** **{**  array**[**i**]** **=** **new** double**[**arr\_size**];**  **}**  int center\_i**,** center\_j**;**  center\_i **=** center\_j **=** arr\_size **/** 2**;**  double pi **=** 3.141592653589793**;**  double sum **=** 0.0f**;**  **for** **(**i **=** 0**;** i **<** arr\_size**;** i**++)** **{**  **for** **(**j **=** 0**;** j **<** arr\_size**;** j**++)** **{**  array**[**i**][**j**]** **=**  exp**(-(**1.0f**)** **\*** **(((**i **-** center\_i**)** **\*** **(**i **-** center\_i**)** **+** **(**j **-** center\_j**)** **\*** **(**j **-** center\_j**))** **/**  **(**2.0f **\*** sigma **\*** sigma**)));**  sum **+=** array**[**i**][**j**];**  **}**  **}**  **for** **(**i **=** 0**;** i **<** arr\_size**;** i**++)** **{**  **for** **(**j **=** 0**;** j **<** arr\_size**;** j**++)** **{**  array**[**i**][**j**]** **/=** sum**;**  **}**  **}**  **return** array**;**  **}**  void myGaussianBlur**(**Mat src**,** Mat**&** dst**,** int size**,** double sigma**)** **{**  **if** **(!**src**.**data**)** **return;**  double**\*\*** arr**;**  Mat tmp**(**src**.**size**(),** src**.**type**());**  **for** **(**int i **=** 0**;** i **<** src**.**rows**;** **++**i**)**  **for** **(**int j **=** 0**;** j **<** src**.**cols**;** **++**j**)** **{**  **if** **((**i **-** 1**)** **>** 0 **&&** **(**i **+** 1**)** **<** src**.**rows **&&** **(**j **-** 1**)** **>** 0 **&&** **(**j **+** 1**)** **<** src**.**cols**)** **{**  arr **=** getGaussionArray**(**size**,** sigma**);**  tmp**.**at**<**Vec3b**>(**i**,** j**)[**0**]** **=** 0**;**  tmp**.**at**<**Vec3b**>(**i**,** j**)[**1**]** **=** 0**;**  tmp**.**at**<**Vec3b**>(**i**,** j**)[**2**]** **=** 0**;**  **for** **(**int x **=** 0**;** x **<** 3**;** **++**x**)** **{**  **for** **(**int y **=** 0**;** y **<** 3**;** **++**y**)** **{**  tmp**.**at**<**Vec3b**>(**i**,** j**)[**0**]** **+=** arr**[**x**][**y**]** **\*** src**.**at**<**Vec3b**>(**i **+** 1 **-** x**,** j **+** 1 **-** y**)[**0**];**  tmp**.**at**<**Vec3b**>(**i**,** j**)[**1**]** **+=** arr**[**x**][**y**]** **\*** src**.**at**<**Vec3b**>(**i **+** 1 **-** x**,** j **+** 1 **-** y**)[**1**];**  tmp**.**at**<**Vec3b**>(**i**,** j**)[**2**]** **+=** arr**[**x**][**y**]** **\*** src**.**at**<**Vec3b**>(**i **+** 1 **-** x**,** j **+** 1 **-** y**)[**2**];**  **}**  **}**  **}**  **}**  tmp**.**copyTo**(**dst**);**  **}**  double sort\_color**(**vector**<**double**>** list**)** **{**  nth\_element**(**list**.**begin**(),** list**.**begin**()** **+** **(**list**.**size**()** **/** 2**)** **+** 1**,** list**.**end**());**  **return** list**[(**list**.**size**()** **/** 2**)];**  **}**  void myMedianBlur**(**const Mat src**,** Mat**&** dst**,** int size**)** **{**  **if** **((**size **%** 2**)** **==** 0**)** **{**  **return;**  **}**  **if** **(!**src**.**data**)** **return;**  Mat \_dst**(**src**.**size**(),** src**.**type**());**  int bide **=** size **/** 2**;**  int kernel\_size **=** size **\*** size**;**  **for** **(**int i **=** 0**;** i **<** src**.**rows**;** i**++)** **{**  **for** **(**int j **=** 0**;** j **<** src**.**cols**;** j**++)** **{**  **if** **((**i **-** bide**)** **>=** 0 **&&** **(**j **-** bide**)** **>=** 0 **&&** **(**i **+** bide**)** **<** src**.**rows **&&** **(**j **+** bide**)** **<** src**.**cols**)** **{**  vector**<**double**>**kernel\_value\_0**(**kernel\_size**);**  vector**<**double**>**kernel\_value\_1**(**kernel\_size**);**  vector**<**double**>**kernel\_value\_2**(**kernel\_size**);**  **for** **(**int kernel\_row **=** i **-** bide**;** kernel\_row **<=** i **+** bide**;** kernel\_row**++)** **{**  **for** **(**int kernel\_col **=** j **-** bide**;** kernel\_col **<=** j **+** bide**;** kernel\_col**++)** **{**  kernel\_value\_0**.**push\_back**(**src**.**at**<**Vec3b**>(**kernel\_row**,** kernel\_col**)[**0**]);**  kernel\_value\_1**.**push\_back**(**src**.**at**<**Vec3b**>(**kernel\_row**,** kernel\_col**)[**1**]);**  kernel\_value\_2**.**push\_back**(**src**.**at**<**Vec3b**>(**kernel\_row**,** kernel\_col**)[**2**]);**  **}**  **}**  \_dst**.**at**<**Vec3b**>(**i**,** j**)[**0**]** **=** sort\_color**(**kernel\_value\_0**);**  \_dst**.**at**<**Vec3b**>(**i**,** j**)[**1**]** **=** sort\_color**(**kernel\_value\_1**);**  \_dst**.**at**<**Vec3b**>(**i**,** j**)[**2**]** **=** sort\_color**(**kernel\_value\_2**);**  **}**  **}**  **}**  \_dst**.**copyTo**(**dst**);**  **}**  /\* 计算空间权值 \*/  double**\*\*** get\_space\_Array**(**int \_size**,** int channels**,** double sigmas**)**  **{**  int i**,** j**;**  double**\*\*** \_spaceArray **=** **new** double**\*** **[**\_size **+** 1**];** //多一行，最后一行的第一个数据放总值  **for** **(**i **=** 0**;** i **<** \_size **+** 1**;** i**++)** **{**  \_spaceArray**[**i**]** **=** **new** double**[**\_size **+** 1**];**  **}**  int center\_i**,** center\_j**;**  center\_i **=** center\_j **=** \_size **/** 2**;**  \_spaceArray**[**\_size**][**0**]** **=** 0.0f**;**  **for** **(**i **=** 0**;** i **<** \_size**;** i**++)** **{**  **for** **(**j **=** 0**;** j **<** \_size**;** j**++)** **{**  \_spaceArray**[**i**][**j**]** **=**  exp**(-(**1.0f**)** **\*** **(((**i **-** center\_i**)** **\*** **(**i **-** center\_i**)** **+** **(**j **-** center\_j**)** **\*** **(**j **-** center\_j**))** **/**  **(**2.0f **\*** sigmas **\*** sigmas**)));**  \_spaceArray**[**\_size**][**0**]** **+=** \_spaceArray**[**i**][**j**];**  **}**  **}**  **return** \_spaceArray**;**  **}**  /\* 计算相似度权值 \*/  double**\*** get\_color\_Array**(**int \_size**,** int channels**,** double sigmar**)**  **{**  int n**;**  double**\*** \_colorArray **=** **new** double**[**255 **\*** channels **+** 2**];**  double wr **=** 0.0f**;**  \_colorArray**[**255 **\*** channels **+** 1**]** **=** 0.0f**;**  **for** **(**n **=** 0**;** n **<** 255 **\*** channels **+** 1**;** n**++)** **{**  \_colorArray**[**n**]** **=** exp**((-**1.0f **\*** **(**n **\*** n**))** **/** **(**2.0f **\*** sigmar **\*** sigmar**));**  \_colorArray**[**255 **\*** channels **+** 1**]** **+=** \_colorArray**[**n**];**  **}**  **return** \_colorArray**;**  **}**  /\* 双边 扫描计算 \*/  void doBialteral**(**cv**::**Mat**\*** \_src**,** int N**,** double**\*** \_colorArray**,** double**\*\*** \_spaceArray**)**  **{**  int \_size **=** **(**2 **\*** N **+** 1**);**  cv**::**Mat temp **=** **(\***\_src**).**clone**();**  **for** **(**int i **=** 0**;** i **<** **(\***\_src**).**rows**;** i**++)** **{**  **for** **(**int j **=** 0**;** j **<** **(\***\_src**).**cols**;** j**++)** **{**  **if** **(**i **>** **(**\_size **/** 2**)** **-** 1 **&&** j **>** **(**\_size **/** 2**)** **-** 1 **&&**  i **<** **(\***\_src**).**rows **-** **(**\_size **/** 2**)** **&&** j **<** **(\***\_src**).**cols **-** **(**\_size **/** 2**))** **{**  double sum**[**3**]** **=** **{** 0.0**,**0.0**,**0.0 **};**  int x**,** y**,** values**;**  double space\_color\_sum **=** 0.0f**;**  **for** **(**int k **=** 0**;** k **<** \_size**;** k**++)** **{**  **for** **(**int l **=** 0**;** l **<** \_size**;** l**++)** **{**  x **=** i **-** k **+** **(**\_size **/** 2**);**  y **=** j **-** l **+** **(**\_size **/** 2**);**  values **=** abs**((\***\_src**).**at**<**cv**::**Vec3b**>(**i**,** j**)[**0**]** **+** **(\***\_src**).**at**<**cv**::**Vec3b**>(**i**,** j**)[**1**]** **+** **(\***\_src**).**at**<**cv**::**Vec3b**>(**i**,** j**)[**2**]**  **-** **(\***\_src**).**at**<**cv**::**Vec3b**>(**x**,** y**)[**0**]** **-** **(\***\_src**).**at**<**cv**::**Vec3b**>(**x**,** y**)[**1**]** **-** **(\***\_src**).**at**<**cv**::**Vec3b**>(**x**,** y**)[**2**]);**  space\_color\_sum **+=** **(**\_colorArray**[**values**]** **\*** \_spaceArray**[**k**][**l**]);**  **}**  **}**  **for** **(**int k **=** 0**;** k **<** \_size**;** k**++)** **{**  **for** **(**int l **=** 0**;** l **<** \_size**;** l**++)** **{**  x **=** i **-** k **+** **(**\_size **/** 2**);**  y **=** j **-** l **+** **(**\_size **/** 2**);**  values **=** abs**((\***\_src**).**at**<**cv**::**Vec3b**>(**i**,** j**)[**0**]** **+** **(\***\_src**).**at**<**cv**::**Vec3b**>(**i**,** j**)[**1**]** **+** **(\***\_src**).**at**<**cv**::**Vec3b**>(**i**,** j**)[**2**]**  **-** **(\***\_src**).**at**<**cv**::**Vec3b**>(**x**,** y**)[**0**]** **-** **(\***\_src**).**at**<**cv**::**Vec3b**>(**x**,** y**)[**1**]** **-** **(\***\_src**).**at**<**cv**::**Vec3b**>(**x**,** y**)[**2**]);**  **for** **(**int c **=** 0**;** c **<** 3**;** c**++)** **{**  sum**[**c**]** **+=** **((\***\_src**).**at**<**cv**::**Vec3b**>(**x**,** y**)[**c**]**  **\*** \_colorArray**[**values**]**  **\*** \_spaceArray**[**k**][**l**])**  **/** space\_color\_sum**;**  **}**  **}**  **}**  **for** **(**int c **=** 0**;** c **<** 3**;** c**++)** **{**  temp**.**at**<**cv**::**Vec3b**>(**i**,** j**)[**c**]** **=** sum**[**c**];**  **}**  **if** **((**i **%** 100**)** **==** 0 **&&** j **==** 10**)**  **{**  cout **<<** i **<<** endl**;**  **}**  **if** **(**i **>** 5000000**)**  **{**  **return;**  **}**  **}**  **}**  **}**  **(\***\_src**)** **=** temp**.**clone**();**  **return;**  **}**  void myBialteralFilter**(**cv**::**Mat**\*** src**,** cv**::**Mat**\*** dst**,** int N**,** double sigmas**,** double sigmar**)**  **{**  **\***dst **=** **(\***src**).**clone**();**  int \_size **=** 2 **\*** N **+** 1**;**  int channels **=** **(\***dst**).**channels**();**  double**\*** \_colorArray **=** **NULL;**  double**\*\*** \_spaceArray **=** **NULL;**  \_colorArray **=** get\_color\_Array**(**\_size**,** channels**,** sigmar**);**  \_spaceArray **=** get\_space\_Array**(**\_size**,** channels**,** sigmas**);**  doBialteral**(**dst**,** N**,** \_colorArray**,** \_spaceArray**);**  **return;**  **}** |