

# 《计算机视觉（1）》实验报告

## 实验八 我的拍照/扫描全能王

实验小组成员 (学号+班级+姓名)	分工及主要完成任务	成绩
201800820081 18 数据科学 都一凡	算法处理 实验报告 GUI 界面	
201800820109 18 数据科学 刘润迪	算法处理 实验报告 微信小程序界面	
201800820045 18 数据科学 许函嘉	算法处理 实验报告 微信小程序界面	

(贡献相同，排序不区分先后)

山东大学

2021 年 3 月

# 实验七 扫描全能王

## 1 实验目的

完成综合实验，实现对相机拍摄或者扫描仪扫描的文档照片的处理，目的在于去除图像中的阴影区域或光照明区，使得处理后的图像打印清晰，视觉显示效果大幅提升。算法利用学过的知识可自行设计，以处理效果优秀，速度快的方法为胜。具体实现方案可依托计算机实现、微信小程序或手机 APP 上实现（三选二）。

## 2 实验原理

### 2.1 图像矫正

图像矫正（反卷绕）的步骤是：给定  $p$  和  $p'$ ，求解单应矩阵  $H$ 。即求解方程： $w * p' = H * p$ ，其中未知数为  $w$  和系数  $H$ 。 $H$  被定义为具有任意比例因子。具体可参考计算机视觉 II 的图像矫正部分。

### 2.2 去噪

#### 2.2.1 均值滤波

平滑线性空间滤波器的输出是包含在滤波器模板邻域内的像素的简单平均值。这种滤波器称为均值滤波器，是低通滤波器的一种。

该滤波器可以通过产生模板下方的标准像素平均值或计算模板邻域内像素的加权平均值，从而进行滤波。一幅  $M \times N$  的图像  $f(x, y)$  经过一个大小为  $m \times n$ （奇数）的加权均值滤波器  $w(s, t)$  滤波的过程如式1：

$$g(x, y) = \frac{\sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x + s, y + t)}{\sum_{s=-a}^a \sum_{t=-b}^b w(s, t)} \quad (1)$$

其中， $m = 2a + 1$ ， $n = 2b + 1$ 。

#### 2.2.2 双边滤波

双边滤波是一种非线性的滤波方法，是结合图像的空间邻近度和像素值相似度的一种折中处理，同时考虑空域信息和灰度相似性，达到保边去噪的目的，具有简单、非迭代、局部的特点。

双边滤波的改进就在于在采样时不仅考虑像素在空间距离上的关系，同时加入了像素间的相似程度考虑，是基于空间分布的高斯滤波函数。在边缘附近，离的较远的像素不会太多影响到边缘上的像素值，这样就保证了边缘附近像素值的保存。

在双边滤波器中，输出像素的值依赖于邻域像素值的加权值组合（如式2）：

$$g(i, j) = \frac{\sum_{k,l} f(k, l) w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)} \quad (2)$$

加权系数  $w(i, j, k, l)$  取决于定义域核和值域核的乘积，定义域核表示为：

$$d(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right) \quad (3)$$

值域核表示为：

$$r(i, j, k, l) = \exp\left(-\frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right) \quad (4)$$

两者相乘后产生依赖于数据的双边滤波权重函数：

$$w(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right) \quad (5)$$

## 2.3 锐化

**非锐化掩蔽（高提升滤波）：**本实验中采用非锐化掩蔽进行处理。通过从原图像减去一幅非锐化（平滑过的）图像实现图像的锐化处理。包括：

1. 模糊原图像；
2. 从原图像中减去模糊图像，产生差值图像（即模板）。令  $\bar{f}(x, y)$  表示模糊图像，非锐化掩蔽模板为：

$$g_{mask}(x, y) = f(x, y) - \bar{f}(x, y) \quad (6)$$

3. 将模板加到原图像中

$$g(x, y) = f(x, y) + k * g_{mask}(x, y) \quad (7)$$

其中，权重系数  $k \geq 0$ 。当  $k = 1$  时得到非锐化掩蔽；当  $k > 1$  时为高提升滤波。

## 2.4 阈值处理

### 2.4.1 自适应阈值处理

在不均匀照明或者灰度值分布不均匀的情况下，如果使用全局阈值分割，通常得到的结果都很不理想，光亮的地方会变成白色，而阴影部分则会全部变成黑色。因此我们考虑针对每一个位置的灰度值设置一个对应的阈值，而这个位置阈值的设置也和其邻域有必然的关系。计算局部阈值的步骤如下：

1. 对图像进行平滑处理，结果记作  $f_{smooth}(I)$ ，其中平滑的手段可以使用均值平滑、高斯平滑、中值平滑；
2. 令自适应阈值矩阵为  $A_{thresh} = (1 - ratio) * f_{smooth}(I)$ ，一般按照经验， $ratio = 0.15$ ；
3. 按照局部阈值分割的规则（如式）进行分割。

$$O(r, c) = \begin{cases} 0, & I(r, c) < \text{Thresh}(r, c) \\ 255, & I(r, c) \geq \text{Thresh}(r, c) \end{cases} \quad (8)$$

### 2.4.2 移动平均法

实际上，移动平均是上述局部阈值处理方法的一种特殊情形，它以一幅图像的扫描行计算移动平均为基础。为了减少光照偏差，扫描是以 Z 字形逐行执行的。令  $z_{k-1}$  表示步骤  $k+1$  中扫描序列中遇到的点的灰度。这个新点处的移动平均（平均灰度）由下式给出：

$$m(k+1) = \frac{1}{n} \sum_{i=k+2-n}^{k+1} z_i = m(k) + \frac{1}{n} (z_{k+1} - z_{k-n}) \quad (9)$$

根据这个值来设定上述步骤二中的自适应阈值矩阵  $A_{threshold}$ 。

### 2.4.3 Sauvola 算法

Sauvola 算法以当前的像素点为中心，根据当前像素点邻域内的灰度均值与标准差来动态计算该像素点的阈值。假定当前像素点的坐标为  $(x, y)$ ，以该点为中心的领域为  $r * r$ ， $g(x, y)$  表示  $(x, y)$  处的灰度值，Sauvola 算法的步骤如下：

1. 计算  $r * r$  邻域内的灰度均值  $m(x, y)$  与标准差  $s(x, y)$ ；
2. 根据灰度均值和标准差，计算像素点  $(x, y)$  的阈值  $T(x, y)$

$$T(x, y) = m(x, y) \cdot \left[ 1 + k \cdot \left( \frac{s(x, y)}{R} - 1 \right) \right] \quad (10)$$

其中  $R$  是标准差的动态范围，若当前输入图像为 8 位灰度图像，则  $R = 128$ ， $k$  是我们自定义的一个修正参数， $k$  的取值对算法的结果影响不显著，一般取  $0 < k < 1$ 。

## 2.5 对比度增强

### 限制对比度的自适应直方图均衡 (CLAHE) :

CLAHE 同普通的自适应直方图均衡不同的地方主要是其对比度限幅，用来克服自适应直方图均衡化的过度放大噪音的问题。主要通过对于每个小区域使用对比度限幅来实现。在指定的像素值周边的对比度放大主要是由变换函数的斜度决定的。这个斜度和领域的累积直方图的斜度成比例。

直方图被裁剪的值（裁剪限幅）取决于直方图的分布，因此也取决于领域大小的取值。通常情况下，直接忽略掉那些超出直方图裁剪限幅的部分是不好的，而是将这些裁剪掉的部分均匀的分布到直方图的其他部分（如图1所示）。

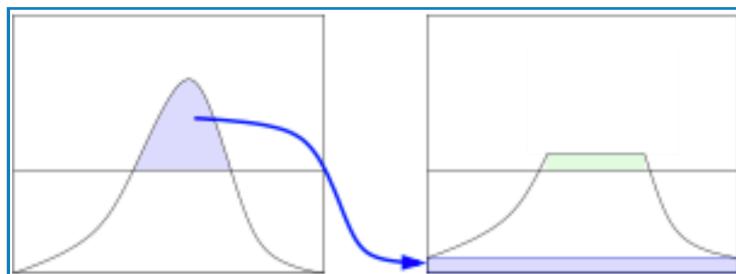


图 1: 限制对比度的自适应直方图均衡 (CLAHE)

### 3 实验软件

Python、OpenCV、PyQt5、微信开发者工具

### 4 实验步骤及实验结果

任意拍摄一幅具有阴影区域的图（如图4(a)），为了使图片具有更好的处理效果，首先对图片进行矫正（如图4(b)），然后采用不同的方法对图像进行处理。

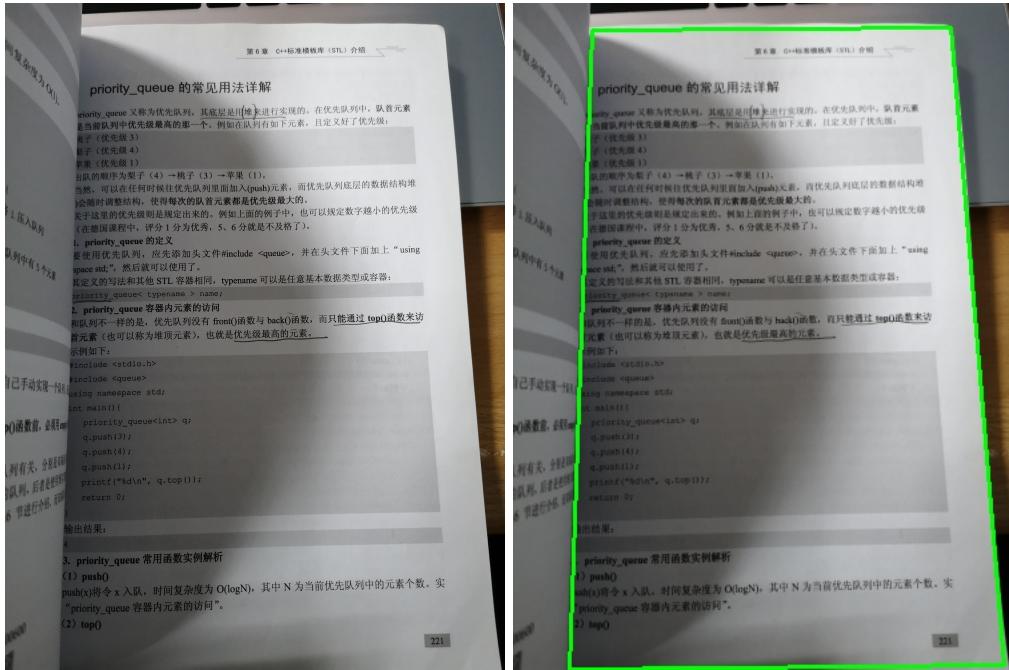


图 2: 有阴影区域影响的图和矫正过程

1. 二值化处理。对图像进行二值化处理观察效果（如图3(a))。
2. 均值滤波处理。使用滤波器模板确定的邻域内像素的平均灰度值代替图像中每个像素的值，从而去除图像中的不相关细节，降低图像灰度的“尖锐”变化。这里使用的是 OpenCV 中的函数 `cv2.blur(src, ksize[, dst[, anchor[, borderType]]])`。经过模糊后的图像如图3(b) 所示。
3. 双边滤波处理。为了解决高斯模糊处理中没有考虑像素值之间的相似程度的问题，采用双边滤波器做边缘保存。这里使用的是 OpenCV 中的函数 `cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]])`。处理后的图像如图4(a) 所示：
4. 锐化处理。主要目的是突出灰度的过渡部分，从而使目标图像变得清晰。这里使用的是 OpenCV 中的函数 `cv2.GaussianBlur(src, ksize, sigmaX [, dst [, sigmaY [, borderType]]])`。处理后的图像如图4(b) 所示：

### priority\_queue 的常见用法详解

priority\_queue 又称为优先队列，其底层是用堆来实现的。在优先队列中，队首元素是当前队列中优先级最高的一个，例如在队列有如下元素，且定义好了优先级：

- 梨子（优先级 3）
- 橘子（优先级 4）
- 苹果（优先级 1）

B队的顺序为梨子 (4) → 橘子 (3) → 苹果 (1)。

当然，可以在任何时候往优先队列里面插入(push)元素，而优先队列底层的数据结构堆会自动调整结构，使得每次的队首元素都是优先级最大的。

关于这里的优先级规则是规定出来的，例如上面的例子中，也可以规定数字越小的优先级（在德国课程中，评分 1 分为优秀，5、6 分就是不及格了）。

#### 1. priority\_queue 的定义

要使用优先队列，应先添加头文件#include <queue>，并在头文件下面加上“using space std;”，然后就可以使用了。

其定义的写法和其他 STL 容器相同，typename 可以是任意基本数据类型或容器；  
#include <queue> <typename> name;

#### 2. priority\_queue 容器内元素的访问

和队列不一样的是，优先队列没有 front()函数与 back()函数，而只能通过 top()函数来访取元素（也可以称为堆顶元素），也就是优先级最高的元素。

示例如下：

```
#include <stdio.h>
#include <queue>
using namespace std;
int main()
{
    priority_queue<int> q;
    q.push(3);
    q.push(4);
    q.push(1);
    printf("%d\n", q.top());
    return 0;
}
```

输出结果：

```
4
```

#### 3. priority\_queue 常用函数实例解析

(1) push()  
push(x)将令 x 入队，时间复杂度为 O(logN)，其中 N 为当前优先队列中的元素个数。实“priority\_queue 容器内元素的访问”。

(2) top()

221

### priority\_queue 的常见用法详解

priority\_queue 又称为优先队列，其底层是用堆来实现的。在优先队列中，队首元素是当前队列中优先级最高的一个。例如在队列有如下元素，且定义好了优先级：

- 梨子（优先级 3）
- 橘子（优先级 4）
- 苹果（优先级 1）

B队的顺序为梨子 (4) → 橘子 (3) → 苹果 (1)。

当然，可以在任何时候往优先队列里面插入(push)元素，而优先队列底层的数据结构堆会自动调整结构，使得每次的队首元素都是优先级最大的。

关于这里的优先级规则是规定出来的，例如上面的例子中，也可以规定数字越小的优先级（在德国课程中，评分 1 分为优秀，5、6 分就是不及格了）。

#### priority\_queue 的定义

要使用优先队列，应先添加头文件#include <queue>，并在头文件下面加上“using space std;”，然后就可以使用了。

其定义的写法和其他 STL 容器相同，typename 可以是任意基本数据类型或容器；  
#include <queue> <typename> name;

#### priority\_queue 容器内元素的访问

和队列不一样的是，优先队列没有 front()函数与 back()函数，而只能通过 top()函数来访取元素（也可以称为堆顶元素），也就是优先级最高的元素。

示例如下：

```
#include <stdio.h>
#include <queue>
using namespace std;
int main()
{
    priority_queue<int> q;
    q.push(3);
    q.push(4);
    q.push(1);
    printf("%d\n", q.top());
    return 0;
}
```

输出结果：

#### 4. priority\_queue 常用函数实例解析

(1) push()  
push(x)令令 x 入队，时间复杂度为 O(logN)，其中 N 为当前优先队列中的元素个数。实“priority\_queue 容器内元素的访问”。

(2) top()

221

(a) 二值化后的图像

(b) 模糊后的图像

图 3: 二值化和模糊后的图像

priority\_queue 的常见用法详解

priority\_queue 又称为优先队列，其底层是用堆来实现的。在优先队列中，队首元素是当前队列中优先级最高的一个。例如在队列有如下元素，且定义好了优先级：

- 梨子（优先级 3）
- 橘子（优先级 4）
- 苹果（优先级 1）

B队的顺序为梨子 (4) → 橘子 (3) → 苹果 (1)。

当然，可以在任何时候往优先队列里面插入(push)元素，而优先队列底层的数据结构堆会自动调整结构，使得每次的队首元素都是优先级最大的。

关于这里的优先级规则是规定出来的，例如上面的例子中，也可以规定数字越小的优先级（在德国课程中，评分 1 分为优秀，5、6 分就是不及格了）。

#### 1. priority\_queue 的定义

要使用优先队列，应先添加头文件#include <queue>，并在头文件下面加上“using space std;”，然后就可以使用了。

其定义的写法和其他 STL 容器相同，typename 可以是任意基本数据类型或容器；  
#include <queue> <typename> name;

#### 2. priority\_queue 容器内元素的访问

和队列不一样的是，优先队列没有 front()函数与 back()函数，而只能通过 top()函数来访取元素（也可以称为堆顶元素），也就是优先级最高的元素。

示例如下：

```
#include <stdio.h>
#include <queue>
using namespace std;
int main()
{
    priority_queue<int> q;
    q.push(3);
    q.push(4);
    q.push(1);
    printf("%d\n", q.top());
    return 0;
}
```

输出结果：

1. priority\_queue 常用函数实例解析

(1) push()  
push(x)将令 x 入队，时间复杂度为 O(logN)，其中 N 为当前优先队列中的元素个数。实“priority\_queue 容器内元素的访问”。

(2) top()

221

### priority\_queue 的常见用法详解

priority\_queue 又称为优先队列，其底层是用堆来实现的。在优先队列中，队首元素是当前队列中优先级最高的一个。例如在队列有如下元素，且定义好了优先级：

- 梨子（优先级 3）
- 橘子（优先级 4）
- 苹果（优先级 1）

B队的顺序为梨子 (4) → 橘子 (3) → 苹果 (1)。

当然，可以在任何时候往优先队列里面插入(push)元素，而优先队列底层的数据结构堆会自动调整结构，使得每次的队首元素都是优先级最大的。

关于这里的优先级规则是规定出来的，例如上面的例子中，也可以规定数字越小的优先级（在德国课程中，评分 1 分为优秀，5、6 分就是不及格了）。

#### priority\_queue 的定义

要使用优先队列，应先添加头文件#include <queue>，并在头文件下面加上“using space std;”，然后就可以使用了。

其定义的写法和其他 STL 容器相同，typename 可以是任意基本数据类型或容器；  
#include <queue> <typename> name;

#### priority\_queue 容器内元素的访问

和队列不一样的是，优先队列没有 front()函数与 back()函数，而只能通过 top()函数来访取元素（也可以称为堆顶元素），也就是优先级最高的元素。

示例如下：

```
#include <stdio.h>
#include <queue>
using namespace std;
int main()
{
    priority_queue<int> q;
    q.push(3);
    q.push(4);
    q.push(1);
    printf("%d\n", q.top());
    return 0;
}
```

输出结果：

1. priority\_queue 常用函数实例解析

(1) push()  
push(x)令令 x 入队，时间复杂度为 O(logN)，其中 N 为当前优先队列中的元素个数。实“priority\_queue 容器内元素的访问”。

(2) top()

221

(a) 双边滤波后的图像

(b) 锐化后的图像

图 4: 双边滤波和锐化后的图像

5. 对比度增强处理。用来提升图像的对比度或局部对比度。这里使用的是 OpenCV 中的函数 cv.createCLAHE([, clipLimit[, tileGridSize]]) 和 clahe.apply() 函数。经过对比度增强后的图片如图6(a) 所示。

6. 移动平均法处理。图6(b) 展示了使用移动平均法对图像的处理结果。

第6章 C++标准模板库(STL)介绍

优先队列

**priority\_queue 的常见用法详解**

priority\_queue 又称为优先队列，其底层是用堆来实现的。在优先队列中，队首元素是当前队列中优先级最高的一个。例如在队列有如下元素，且定义好了优先级：

- 桃子 (优先级 3)
- 梨子 (优先级 4)
- 苹果 (优先级 1)

队列的顺序为梨子 (4) → 桃子 (3) → 苹果 (1)。

当然，可以在任何时候往优先队列里面加入(push)元素，而优先队列底层的数据结构会随时调整结构，使得每次的队首元素都是优先级最大的。

关于这里的优先级是根据定出来的，例如上面的例子中，也可以规定数字越小的优先级（在那个程序中，评分 1 分分为优秀，5、6 分就是不及格了）。

**1. priority\_queue 的定义**

要使用优先队列，应先添加头文件#include <queue>，并在头文件下面加上“using space std”，然后就可以使用了。

其定义的写法和其他 STL 容器相同，typename 可以是任意基本数据类型或容器：

```
#include <iostream>
#include <queue>
using namespace std;
```

**2. priority\_queue 容器内元素的访问**

和队列不一样的是，优先队列没有 front()函数与 back()函数，而只能通过 top()函数来访最元素（也可以称为堆顶元素），也就是优先级最高的元素。

示例如下：

```
int main()
{
    priority_queue<int> q;
    q.push(3);
    q.push(4);
    q.push(1);
    printf("%d\n", q.top());
    return 0;
}
```

输出结果：

(1) pushQ  
push(x)将令 x 入队，时间复杂度为 O(logN)，其中 N 为当前优先队列中的元素个数。实“priority queue 容器内元素的访问”。

(2) topQ

221

(a) 对比度增强处理后的图像

(b) 移动平均法处理后的图像

图 5: 对比度增强和移动平均法处理后的图像

## 5 实验结果

使用上面的图片进行处理，选择一种效果比较好的或者综合多种算法进行处理，得到实验结果（如图5）。

第6章 C++标准模板库(STL)介绍

优先队列

**priority\_queue 的常见用法详解**

priority\_queue 又称为优先队列，其底层是用堆来实现的，在优先队列中，队首元素是当前队列中优先级最高的一个。例如在队列有如下元素，且定义好了优先级：

- 桃子 (优先级 3)
- 梨子 (优先级 4)
- 苹果 (优先级 1)

队列的顺序为梨子 (4) → 桃子 (3) → 苹果 (1)。

当然，可以在任何时候往优先队列里面加入(push)元素，而优先队列底层的数据结构会随时调整结构，使得每次的队首元素都是优先级最大的。

关于这里的优先级是根据定出来的，例如上面的例子中，也可以规定数字越小的优先级（在那个程序中，评分 1 分分为优秀，5、6 分就是不及格了）。

**1. priority\_queue 的定义**

要使用优先队列，应先添加头文件#include <queue>，并在头文件下面加上“using space std”，然后就可以使用了。

其定义的写法和其他 STL 容器相同，typename 可以是任意基本数据类型或容器：

```
#include <iostream>
#include <queue>
using namespace std;
```

**2. priority\_queue 容器内元素的访问**

和队列不一样的是，优先队列没有 front()函数与 back()函数，而只能通过 top()函数来访最元素（也可以称为堆顶元素），也就是优先级最高的元素。

示例如下：

```
int main()
{
    priority_queue<int> q;
    q.push(3);
    q.push(4);
    q.push(1);
    printf("%d\n", q.top());
    return 0;
}
```

输出结果：

(1) pushQ  
push(x)将令 x 入队，时间复杂度为 O(logN)，其中 N 为当前优先队列中的元素个数。实“priority queue 容器内元素的访问”。

(2) topQ

221

(a) 带阴影部分的原图

(b) 处理结果

图 6: 实验结果

## 6 GUI 封装

使用 GUI 界面对程序进行封装（如图7）。GUI 界面可以显示图像、进行图片矫正和一键扫描，并进行去噪、锐化、自适应阈值处理、移动平均法、Sauvola 算法和对比度增强的处理以及调整对应的参数。



图 7: PC 端 GUI

## 7 微信小程序

使用微信小程序对程序进行封装（如图8）。微信小程序界面可以显示图像（拍照或上传图片）、进行图片矫正，进行对比度、去噪声、二值化、锐化、自适应阈值和 Sauvola 阈值处理，并通过拖动条调整对应的参数。



图 8: 微信小程序界面

## 8 反思感悟

在本次实验过程中，我们使用了多种算法实现了对图像的矫正和处理，去除图像中的阴影区域或光亮点区，使得处理后的图像更加清晰，视觉显示效果更好。这是一次综合性的尝试，通过我们的共同努力，实现了设计自己的扫描全能王。