山东大学 计算机科学与技术 学院

计算机视觉 课程实验报告

|  |  |  |
| --- | --- | --- |
| 学号：201600301304 | 姓名：贾乘兴 | 班级：人工智能16 |
| 实验题目：图像滤波 | | |
| 实验内容：   1. 高斯滤波 2. 高斯滤波是一种线性平滑滤波，适用于消除高斯噪声，广泛应用于图像处理的减噪过程。通俗的讲，高斯滤波就是对图像进行平均加权的过程。 3. 高斯滤波通过特定的滤波器对图像像素进行扫描，得到的新图像的每一个像素点由原位置像素点及其周围某一特定的邻域内的像素点数值得到。   二维图像的高斯滤波器公式及其分布图如下：  image  image  其中sigma为超参数，同时，指数前面的系数在滤波器中可进行计算，最后对滤波器各个权重求和，进行归一化。  以下为5\*5滤波器的归一化：  image   1. 实现高斯滤波函数，将sigma作为超参数输入以控制平滑程度，函数形式为：   void Gaussian(const Mat &input, Mat &output, double sigma);  同时，设置滤波器窗口的大小为[6\*sigma-1]，[.]为向下取整  在边界上，考虑边界在本问题并不重要，使用0对边界padding  设计的高斯滤波器如下：  **// 普通运算下的滤波器  for(int y=0;y<input.rows;y++){  for(int x=0;x<input.cols;x++){  for(int c=0;c<3;c++){**  **int e=0;  output.at<Vec3b>(y,x)[c]=0;  for(int i=0;i<m;i++){  for(int j=0;j<m;j++){  if(y+i-k>=0&&x+j-k>=0&&x+i-k<input.cols&&y+j-k<input.rows){  e+=input.at<Vec3b>(y+i-k,x+j-k)[c]\*ro[i]\*co[j];  }  }  }  output.at<Vec3b>(y,x)[c]=e/sum;  }  }  }** }  这样可实现最简单的高斯滤波  我们发现高斯滤波器函数具有以下性质：    该性质为行列可分离性，如果直接进行计算，我们需要进行M\*M次乘法，其中M为滤波器长度，但分离后我们可以仅进行2\*M次运算，分离后设计的滤波函数如下：  *// 行列可分离性加速* **for**(**int** y=0;y<input.rows;y++){  **for**(**int** x=0;x<input.cols;x++){  **for**(**int** c=0;c<3;c++){  *//cout<<(int)input.at<Vec3b>(y,x)[c]<<endl;* **int** p[m];  **int** e=0;  **for**(**int** i=0;i<m;i++){  p[i]=0;  **for**(**int** j=0;j<m;j++){  **if**(y+i-k>=0&&x+j-k>=0&&x+i-k<input.cols&&y+j-k<input.rows){  p[i]+=input.at<Vec3b>(y+i-k,x+j-k)[c]\*co[j];  }  }  }  **for**(**int** i=0;i<m;i++){  **if**(y+i-k>=0&&y+i-k<input.rows){  e+=p[i]\*ro[i];  }  }  *//cout<<(int)output.at<Vec3b>(y,x)[c]<<endl;* output.at<Vec3b>(y,x)[c]=e/sum;  *//cout<<(int)output.at<Vec3b>(y,x)[c]<<endl;* }  } }  利用行列可分离性进行加速可以减小计算代价  最终设计的高斯滤波函数代码如下：  **void** Gaussian(**const** Mat &input,Mat &output,**double** sigma){  **int** m=(**int**)(6\*sigma-1);  **if**(m%2==0){  m++;  }  **int** k=m/2;  *//cout<<k<<m<<endl;* **int** sum=0;  **int** ro[m];  **int** co[m];  **double** b=exp(-pow(k,2)/(2\*pow(sigma,2)));  **for**(**int** i=0;i<m;i++){  ro[i]=(**int**)(exp(-pow(i-k,2)/(2\*pow(sigma,2)))/b);  co[i]=(**int**)(exp(-pow(i-k,2)/(2\*pow(sigma,2)))/b);  }  **for**(**int** i=0;i<m;i++){  **for**(**int** j=0;j<m;j++){  sum+=ro[i]\*co[j];  }  }  *// 行列可分离性加速* **for**(**int** y=0;y<input.rows;y++){  **for**(**int** x=0;x<input.cols;x++){  **for**(**int** c=0;c<3;c++){  *//cout<<(int)input.at<Vec3b>(y,x)[c]<<endl;* **int** p[m];  **int** e=0;  **for**(**int** i=0;i<m;i++){  p[i]=0;  **for**(**int** j=0;j<m;j++){  **if**(y+i-k>=0&&x+j-k>=0&&x+i-k<input.cols&&y+j-k<input.rows){  p[i]+=input.at<Vec3b>(y+i-k,x+j-k)[c]\*co[j];  }  }  }  **for**(**int** i=0;i<m;i++){  **if**(y+i-k>=0&&y+i-k<input.rows){  e+=p[i]\*ro[i];  }  }  *//cout<<(int)output.at<Vec3b>(y,x)[c]<<endl;* output.at<Vec3b>(y,x)[c]=e/sum;  *//cout<<(int)output.at<Vec3b>(y,x)[c]<<endl;* }  }  }}  **实现了该方法后，设置sigma为3，传入相同大小的input和output的image调用高斯滤波函数代码如下：**  **int** main(){  Mat input=imread(**"/Users/apple/Desktop/cq.jpg"**);  Mat output=Mat::zeros(input.size(),input.type());  Gaussian(input,output,3);  imshow(**"out"**,output);  imwrite(**"/Users/apple/Desktop/gq.jpg"**,output);  waitKey(0); }  原图像与最终生成图像对比如下：  ../../b.jpeg ../../h1.jpeg  ../../a.png ../../h.png  选取了带有高斯噪声的图像和纯高斯噪声图像，在去除高斯噪声上，对比如下：  ../../cq.jpg ../../gq.jpg  ../../c.jpg ../../g.jpg  可见高斯滤波在处理高斯噪声上有较好的效果，但容易造成图像模糊，主要较为适合处理高斯噪声，在其它噪声上效果不够好   1. 快速均值滤波   1.将滤波器设计为常数项则为均值滤波，均值滤波可视为一种特殊的高斯滤波    所以与高斯函数在平滑图像上有着相似的作用  2.均值滤波也可以具有行列的可分离性，但均值滤波也具有特殊的加速方法，即积分图加速方法：  积分图  图像I的积分图S是与其大小相同的图像，S的每一像素S(u,v)存贮的是  I(u,v)左上角所有像素的颜色值之和。    积分图可增量计算，只需对原图进行一遍扫描：    3.基于积分图的快速滤波  4.设滤波窗口大小为2w+1，滤波结果为图像O，则：          基于积分图实现的快速运算如下  **void** MeanFilter(**const** Mat &input,Mat &output,**int** z){  **if**(z%2==0){  z++;  }  **int** w=z/2;  **int** sum=z\*z;**int** s[input.rows][input.cols][3];s[0][0][0]=(**int**)(input.at<Vec3b>(0,0)[0]);  s[0][0][1]=(**int**)(input.at<Vec3b>(0,0)[1]);  s[0][0][2]=(**int**)(input.at<Vec3b>(0,0)[2]);**for**(**int** i=1;i<input.cols;i++){  **for**(**int** c=0;c<3;c++){  s[0][i][c]=0;  s[0][i][c]=(**int**)(input.at<Vec3b>(0,i)[c])+s[0][i-1][c];}  }  **for**(**int** i=1;i<input.rows;i++){  **for**(**int** c=0;c<3;c++){  s[i][0][c]=0;  s[i][0][c]=(**int**)(input.at<Vec3b>(i,0)[c])+s[i-1][0][c];}  }  **for**(**int** y=1;y<input.rows;y++){  **for**(**int** x=1;x<input.cols;x++){  **for**(**int** c=0;c<3;c++){  s[y][x][c]=0;  s[y][x][c]=(**int**)(input.at<Vec3b>(y,x)[c])+s[y-1][x][c]+s[y][x-1][c]-s[y-1][x-1][c];}  }  } *// 普通滤波器* **for**(**int** y=0;y<input.rows;y++){  **for**(**int** x=0;x<input.cols;x++){  **for**(**int** c=0;c<3;c++){**int** k=0;  **if**(y-w-1<0&&x-w-1<0&&y+w<input.rows&&x+w<input.cols){  k=s[y+w][x+w][c];  }**else if**(y-w-1<0&&x-w-1>=0&&y+w<input.rows&&x+w>=input.cols){  k=s[y+w][input.cols-1][c]-s[y+w][x-w-1][c];  }**else if**(y-w-1>=0&&x-w-1<0&&y+w>=input.rows&&x+w<input.cols){  k=s[input.rows-1][x+w][c]-s[y-w-1][x+w][c];  }**else if**(y-w-1>=0&&x-w-1>=0&&y+w>=input.rows&&x+w>=input.cols){  k=s[input.rows-1][input.cols-1][c]+s[y-w-1][x-w-1][c]-s[input.rows-1][x-w-1][c]-s[y-w-1][input.cols-1][c];  }**else if**(y-w-1>=0&&x-w-1<0&&y+w<input.rows&&x+w<input.cols){  k=s[y+w][x+w][c]-s[y-w-1][x+w][c];  }**else if**(y-w-1>=0&&x-w-1>=0&&y+w<input.rows&&x+w>=input.cols){  k=s[y+w][input.cols-1][c]+s[y-w-1][x-w-1][c]-s[y+w][x-w-1][c]-s[y-w-1][input.cols-1][c];  }**else if**(y-w-1<0&&x-w-1>=0&&y+w<input.rows&&x+w<input.cols){  k=s[y+w][x+w][c]-s[y+w][x-w-1][c];  }**else if**(y-w-1>=0&&x-w-1>=0&&y+w>=input.rows&&x+w<input.cols){  k=s[input.rows-1][x+w][c]+s[y-w-1][x-w-1][c]-s[input.rows-1][x-w-1][c]-s[y-w-1][x+w][c];  }**else if**(y-w-1>=0&&x-w-1>=0&&y+w<input.rows&&x+w<input.cols){  k=s[y+w][x+w][c]+s[y-w-1][x-w-1][c]-s[y+w][x-w-1][c]-s[y-w-1][x+w][c];}output.at<Vec3b>(y,x)[c]=k/sum;  }  }  }}  在代码中判定了8类边界条件，计算中对边界使用padding=0的方式  最终实现的效果对比如下  ../../a.png ../../h2.png | | |
| 实验过程中遇到和解决的问题：  （记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）   1. 本次实验遇到的问题较多，解决方法虽然不难但需要思考，一开始生成的图像都为黑色图像，像素输出后发现都是0，然后将像素放大，发现图像像素数值分布较为混乱，最终发现opencv的mat格式超过255后会发生溢出，在处理过程中多次发生超出边界的问题，所以采用int类型变量存储，最后赋值给输出图像 2. 多种边界条件判定，需要集中精力进行判断 3. 滤波器归一化时，转整数注意变成0的问题，最终会导致，先进行放大 | | |
| 结论分析与体会：熟悉了滤波算法的实现，同时还在不断的调试代码中掌握了除算法之外一些技巧和经验 | | |