**武 汉 工 程 大 学**

**计算机科学与工程学院**

**《图像处理与机器视觉》实验报告[2]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业班级 | | 智能科学与技术1班 | 实验时间 | | **2022年11月2日** | | |
| 学生学号 | | 2005120203 | 实验地点 | | 机房 | | |
| 学生姓名 | | 方成杰 | 指导教师 | | 闵锋 | | |
|  | | | | | | | |
| 实验项目 | | 图像的平滑与锐化 | | | | | |
| 实验类别 | | 基础性 | 实验学时 | | **4** | | |
| 实验目的及要求 | | 1. 掌握空域滤波的基本原理及方法；  2. 实现图像均值和中值滤波;  3. .实现图像的平滑与锐化。 | | | | | |
| 成 绩 评 定 表 | | | | | | | |
| 类 别 | | 评 分 标 准 | | 分值 | | 得分 | 合 计 |
| 上机表现 | | 按时出勤、遵守纪律  认真完成各项实验内容 | | **30**分 | |  |  |
| 报告质量 | 程序代码规范、功能正确  填写内容完整、体现收获 | | | **70**分 | |  |
| **说明：**  **评阅教师：**  **日 期： 年月日** | | | | | | | |
| 实 验 内 容 | | | | | | | |
| 1. 实现图像的均值滤波，分别使用3x3和7x7的模板，将滤波前和滤波后的图像保存下来，与核心代码一起粘贴到此:   def middle\_filter(img, kernel\_size):  H, W, C = img.shape  kernel = np.ones([kernel\_size, kernel\_size])  margin = int(kernel\_size / 2)  result = img.copy()  img = cv2.copyMakeBorder(img, margin, margin, margin, margin, borderType=cv2.BORDER\_REPLICATE)   for i in range(margin, H + margin):  for j in range(margin, W + margin):  for c in range(C):  result[i - margin, j - margin, c] = sum(  sum(kernel \* img[i - margin: i + margin + 1, j - margin: j + margin + 1, c])) // (kernel\_size \*\* 2)  # 为什么不能直接？sum(sum(img[i - margin: i + margin + 1, j - margin: j + margin + 1, c]))  # 已解决，因为INT8会溢出，导致图片又紫又暗  return result   1. 实现图像的中值滤波，分别使用3x3和7x7的模板，将滤波前和滤波后的图像保存下来，与核心代码一起粘贴到此:   def median\_filter(img, kernel\_size):  H, W, C = img.shape  margin = int(kernel\_size / 2)  result = img.copy()  img = cv2.copyMakeBorder(img, margin, margin, margin, margin, borderType=cv2.BORDER\_REPLICATE)  for i in range(margin, H + margin):  for j in range(margin, W + margin):  for c in range(C):  result[i - margin, j - margin, c] = np.median(  img[i - margin: i + margin + 1, j - margin: j + margin + 1, c])  return result   1. 实现图像的最大值和最小值滤波，分别使用3x3和7x7的模板，将滤波前和滤波后的图像保存下来，与核心代码一起粘贴到此:   def max\_min\_filter(img, kernel\_size):  H, W, C = img.shape  margin = int(kernel\_size / 2)  max\_result = img.copy()  min\_result = img.copy()  img = cv2.copyMakeBorder(img, margin, margin, margin, margin, borderType=cv2.BORDER\_REPLICATE)  for i in range(margin, H + margin):  for j in range(margin, W + margin):  for c in range(C):  max\_result[i - margin, j - margin, c] = np.max(  img[i - margin: i + margin + 1, j - margin: j + margin + 1, c])  min\_result[i - margin, j - margin, c] = np.min(  img[i - margin: i + margin + 1, j - margin: j + margin + 1, c])  return [max\_result, min\_result]   1. 实现Sobel算子，分别使用水平和垂直方向算子，将结果图像与核心代码一起粘贴到此:   def sobel(img, threshold=0):  H, W = img.shape  x\_kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  y\_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  kernel\_size = len(x\_kernel)  margin = int(kernel\_size / 2)  x\_result = np.zeros(img.shape)  y\_result = np.zeros(img.shape)  img = cv2.copyMakeBorder(img, margin, margin, margin, margin, borderType=cv2.BORDER\_REPLICATE)  for i in range(margin, H + margin):  for j in range(margin, W + margin):  x\_pix = sum(  sum(x\_kernel \* img[i - margin: i + margin + 1, j - margin: j + margin + 1])) // (kernel\_size \*\* 2)  y\_pix = sum(  sum(y\_kernel \* img[i - margin: i + margin + 1, j - margin: j + margin + 1])) // (  kernel\_size \*\* 2)  if x\_pix > threshold:  x\_result[i - margin, j - margin] = x\_pix  if y\_pix > threshold:  y\_result[i - margin, j - margin] = y\_pix  return [img, x\_result, y\_result, x\_result + y\_result]   1. 实现Laplace算子，将结果图像与核心代码一起粘贴到此:   def Laplace(img):  H, W = img.shape  kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])  kernel\_size = len(kernel)  margin = int(kernel\_size / 2)  result = np.zeros(img.shape)  img = cv2.copyMakeBorder(img, margin, margin, margin, margin, borderType=cv2.BORDER\_REPLICATE)  for i in range(margin, H + margin):  for j in range(margin, W + margin):  result[i - margin, j - margin] = sum(  sum(kernel \* img[i - margin: i + margin + 1, j - margin: j + margin + 1])) // (kernel\_size \*\* 2)  return [img, result]   1. 实现Laplace图像锐化，将结果图像与核心代码一起粘贴到此:   kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])  kernel\_1 = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])  def Laplace(img, kernel):  temp = np.array(img)  H, W = img.shape  kernel\_size = len(kernel)  margin = int(kernel\_size / 2)  result = np.zeros(img.shape)  img = cv2.copyMakeBorder(img, margin, margin, margin, margin, borderType=cv2.BORDER\_REPLICATE)  img = np.array(img)  for i in range(margin, H + margin):  for j in range(margin, W + margin):  pix = np.sum(kernel \* img[i - margin: i + margin + 1, j - margin: j + margin + 1])  result[i - margin, j - margin] = pix   result = temp - result  result[result < 0] = 0  result[result > 255] = 255  return result | | | | | | | |