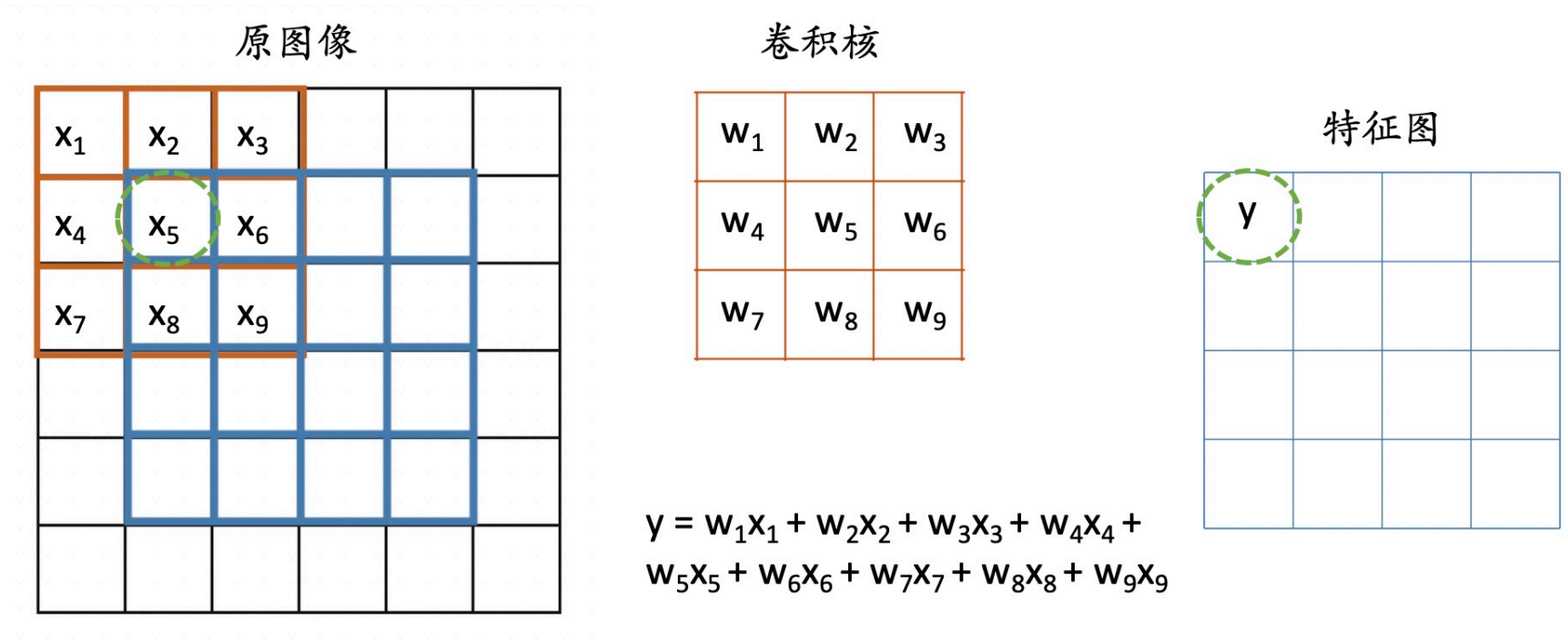
In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import statistics as stats
```

Lab 2

第一部分：卷积和滤波

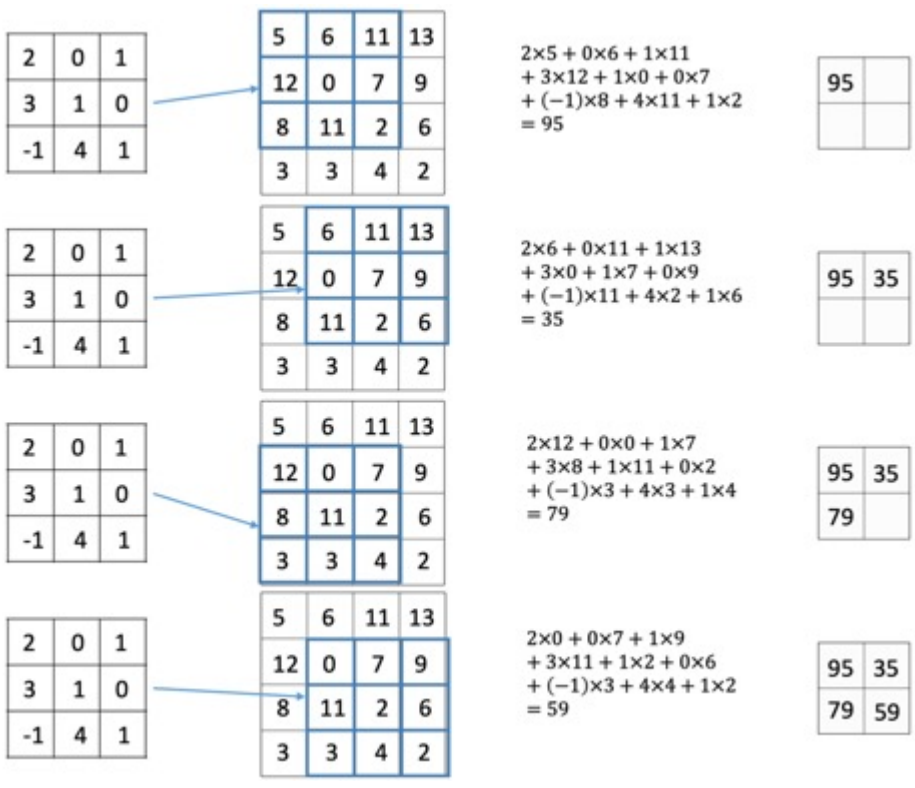
本次Lab的第一部分为课堂内容的补充，旨在加深对卷积操作、均值滤波、中值滤波的理解。

图像的卷积操作：使用卷积核（convolution kernel，也称为模板、算子或掩膜）对原图像中的每个像素进行重新计算，形成特征图（feature map）。具体来说，卷积核是一个矩阵，通常是一个较小的四方形网格结构，边长通常为基数个像素（如3x3的矩阵）。在进行卷积计算时，卷积核的中心会对准原图像的某个像素点。然后，**卷积核中的每个元素**与图像上**对应位置的像素值**一一**相乘**。这些乘积**求和**的结果，就是特征图中该位置的新像素值。



$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7 + w_8x_8 + w_9x_9$$

举例：



编写卷积操作

第一步：定义卷积核

我们先定义一个 3 * 3 的简单卷积核

In [ ]:
```
# 定义一个3*3的卷积核
kernel = [[2,1,0],
          [1,0,1],
          [0,1,2]]
# 可视化查看卷积核
plt.imshow(kernel,cmap=plt.cm.gray)
```

第二步，准备图片

为了便于理解，我们不用真实图片，而是使用一张宽和高都是4个像素的矩阵作为“伪图片”，且忽略三个色彩通道（视为灰度图）。

In [ ]:
```
# 定义一张4*4的”图片“
img = [[150,140, 20, 50],
       [100,125, 10, 30],
       [ 10, 50,200,130],
       [ 20, 45,150,220]]
# 可视化查看图像
plt.imshow(img,cmap=plt.cm.gray)
```

现在，我们有了一个 3 * 3 的卷积核以及一张 4 * 4 的灰度图。

第三步，进行卷积操作。

不考虑0填充的情况下，用3 * 3 的卷积核给 4 * 4 的图像做卷积操作，最终将得到一个 2 * 2 的特征图。

接下来，我们定义两个函数。

第一个函数Filter()负责处理卷积核的中心点对准图像中**某一个**像素点时，需要进行的“先乘后加”操作。它传入4个参数，分别是**卷积核**、**图像**，以及卷积核中心位置当前对准的像素点在图像中的**行列数**。 它返回一个值，即当前位置的卷积操作完成后，“先乘后加”的结果。

In [ ]:
```
# 定义当前位置的卷积（“相乘后相加”）的函数，其中kernel为卷积核，img为图像
# row和col为当前位置卷积核中点所对应的像素点，在图像中的行列数
def Filter(kernel,img,row,col):
    result=0
    # 卷积核的边长
    dim=np.array(kernel).shape[0]
    for i in range(dim):
        for c in range(dim):
            result=result+kernel[i][c]*img[i+row][col+c]
    return result
```

第二个函数Conv()主要负责把前面的Filter()函数作用于所有需要操作的点位上。

该函数需要传入4个参数，分别是**卷积核**、**图像**以及**特征图（卷积操作的结果）的维度（行列数）**。

In [ ]:
```
# 定义卷积操作的完成函数，其中kernel为卷积核，img为图像
# res_row,res_col为卷积结果（特征图）的行列数
def Conv(kernel,img,res_row,res_col):
    Result = np.zeros((res_row,res_col))
    # 通过调用前面的Filter函数来对特征图每一个位置进行卷积操作
    for i in range(res_row):
        for c in range(res_col):
            Result[i][c]=Filter(kernel,img,i,c)
    return Result
```

第四步，运行卷积操作

In [ ]:
```
# 调用Conv函数，获得特征图（因为没有padding，特征图的维度是2*2）
featuremap = Conv(kernel, img, 2, 2)
# 打印特征图的数值
print(featuremap)
# 可视化查看特征图
plt.imshow(featuremap,cmap=plt.cm.gray)
```

Padding操作

现在，我们引入零填充。

In [ ]:
```
# 对原图进行0填充（zero-padding），这一步可以调用numpy直接完成
img_pad = np.pad(img, 1, mode='constant')
# 可视化查看padding后的原图
plt.imshow(img_pad,cmap=plt.cm.gray)
```

重复前面的操作，再次运行卷积。这一次，卷积输出的特征图维度和原图一样，为 4 * 4。

In [ ]:
```
# 调用Conv函数，获得特征图
featuremap_pad = Conv(kernel, img_pad, 4, 4)
# 打印特征图的数值
print(featuremap_pad)
# 可视化查看特征图
plt.imshow(featuremap_pad,cmap=plt.cm.gray)
```

使用真实图片

现在，我们设计一个 5 * 5 的卷积核，对lab2_img中的真实图片man.jpg做卷积操作。

In [ ]:
```
# 定义一个5*5的卷积核
kernel = [[0, 1, 2, 1, 0],
          [1, 0, 1, 0, 1],
          [0, -1, -2, -1, 0],
          [1, 0, 1, 0, 1],
          [2, 1, 0, 1, 2]]
# 可视化查看卷积核
plt.imshow(kernel,cmap=plt.cm.gray)
```

In [ ]:
```
# 导入图片、将其灰度化、调整图幅大小为180*230，并做padding
img = cv2.imread('lab2_img/man.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img = cv2.resize(img, (180, 230),
                 interpolation = cv2.INTER_LINEAR)
# 我们将使用5*5的卷积核，因此需要做2层的padding
img_pad = np.pad(img, 2, mode='constant')
# 可视化查看padding后的原图
plt.imshow(img_pad,cmap=plt.cm.gray)
```

In [ ]:
```
# 调用Conv函数，获得特征图（注意，我们做了padding，因此输出特征图的维度和输入图像相同）
featuremap_pad = Conv(kernel, img_pad, 230, 180)
# 可视化查看特征图
plt.imshow(featuremap_pad,cmap=plt.cm.gray)
```

任务1

在下面的两个Cell里，请自定义一个 9 * 9 的卷积核（具体的设计随意），对man.jpg进行卷积操作。

In [ ]:
```
# 请在这里定义9*9的卷积核，并可视化呈现
kernel = []
# 可视化查看卷积核
plt.imshow(kernel,cmap=plt.cm.gray)
```

In [ ]:
```
# 请在这里完成任务1的剩余操作，可视化呈现特征图
```

均值滤波和中值滤波

卷积操作在深度学习中多用于提取特征。

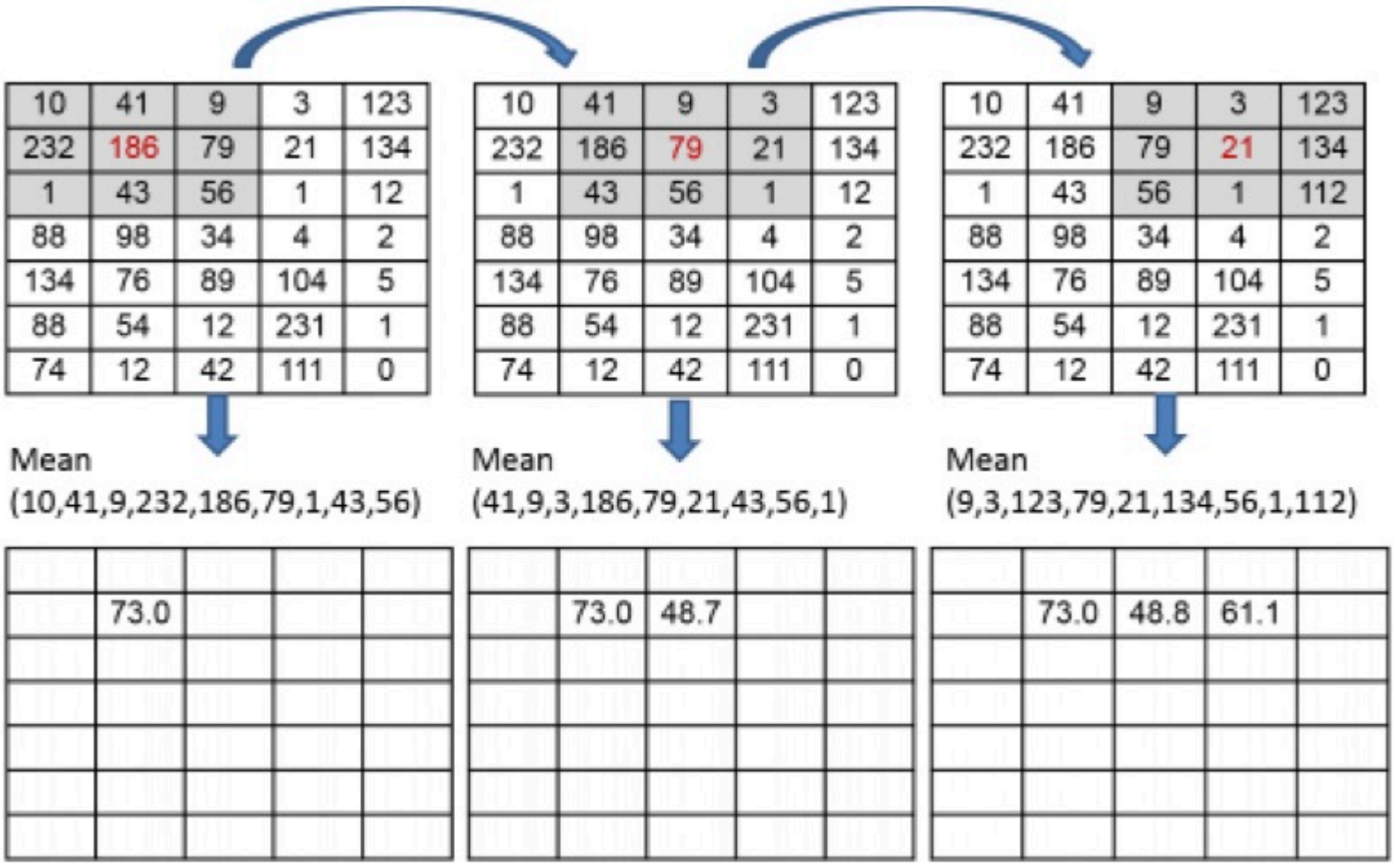
除此之外，我们也可以用**类似于**卷积操作的手段，在预处理阶段对图像做**降噪**处理。

常见的滤波方法分均值滤波和中值滤波。

均值滤波是典型的线性滤波算法，它是指在图像上对目标像素给一个模板（滤波器），该模板包括了目标像素及其周围的临近像素，滤波后，模板覆盖的全体像素的平均值，将被用来代替目标像素的原像素值。这种滤波方法可以有效地抑制图像中的加性噪声，但容易引起图像模糊，可以对其进行改进，主要避开对景物边缘的平滑处理。

中值滤波是一种非线性滤波方法，经常用于消除图像或者其它信号中的噪声。这种滤波方法对于斑点噪声（speckle noise）和椒盐噪声（salt-and-pepper noise）尤其有效。中值滤波的基本原理是，将图像中某一点的像素值，用该点及其邻域中各点值的中位数代替，从而消除孤立的噪声点。这种滤波方法对于滤除图像的椒盐噪声非常有效，可以保持图像的边缘特性，不会让图像产生显著的模糊，适合消除通过扫描得到的图像中的颗粒噪声。

均值滤波



编写均值滤波的函数

参考前面的卷积操作，编写两个新的函数，进行均值滤波。

首先，定义Mean_Filter函数，该函数处理图像上**某个特定位置**的均值滤波操作。

均值滤波在这里不需要定义具体的滤波器（或可理解为滤波器里面所有的值都是1/n，其中n为滤波器的网格数量，没有加权操作），只需要定义滤波器的边长。

它传入4个参数，包括**图像**、**滤波器的边长**、滤波器中心点当前对应图像像素的**行列数**。

In [ ]:
```
# 定义Mean_Filter函数（参考上面的Filter），参数img为图像
# row和col为当前位置卷积核中点所对应的像素点，在图像中的行列数
# 此次不需要传入具体的卷积核，只需要传入卷积核的边长dim
def Mean_Filter(dim,img,row,col):
    result=[]
    # 卷积核的边长
    for i in range(dim):
        for c in range(dim):
            result.append(img[i+row][col+c])
    # 返回卷积核覆盖区域的像素点的平均值
    return int(np.mean(result))
```

然后定义Mean_Conv函数来让滤波器遍历图像。它传入2个参数，包括滤波器边长和图像本身。

这一次我们需要让程序根据传入的滤波器边长，来自动进行padding操作。

默认滤波器边长为奇数，则padding的层数为（滤波器边长-1）/2

In [ ]:
```
# 定义整体的卷积函数，其中dim为卷积核的边长，img为图像
# 这一次我们要求图像必须做zero padding，并让用户传入padding的层数
# padding之后，不用再定义输出图像的尺度（和原图像相同）

def Mean_Conv(dim,img):
    # 对图像进行zero padding
    padding = int((dim - 1)/2)
    img = np.pad(img, padding, mode='constant')
    # 这里传入的是padding之后的图像，因此在计算输出图像边长时需要减2*padding
    res_row = np.array(img).shape[0]-2*padding
    res_col = np.array(img).shape[1]-2*padding
    Result = np.zeros((res_row, res_col))
    # 通过调用前面的Mean_Filter函数来对特征图每一个位置进行卷积操作
    for i in range(res_row ):
        for c in range(res_col):
            Result[i][c]=Mean_Filter(dim,img,i,c)
    return Result
```

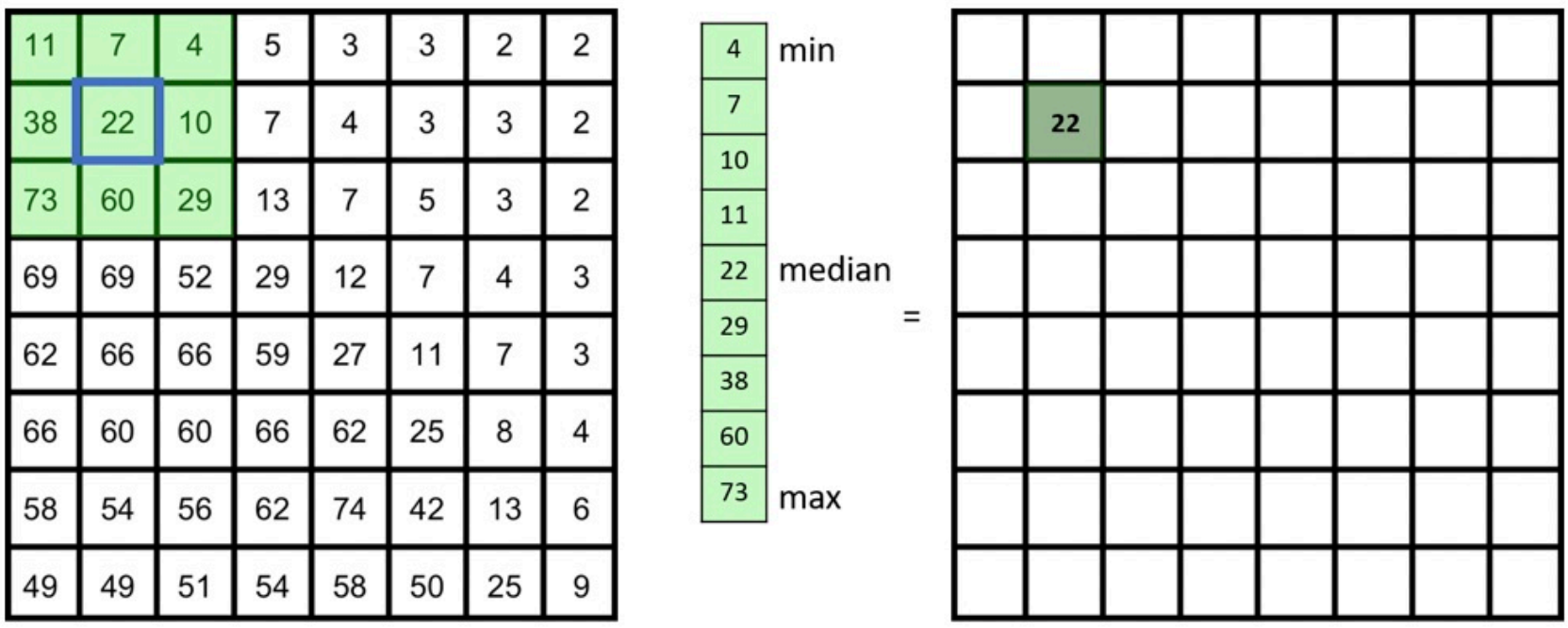
In [ ]:
```
# 定义一张9*9且“噪声”和特征都明显的图像
img = [[ 12, 15, 29,  20, 37, 80, 112, 103, 110],
       [ 10, 20, 33,  70, 59, 90, 100,  88, 70],
       [  8, 17, 52, 225, 61, 49,  93, 100, 90],
       [ 39, 50, 53,  62, 70, 38,  81, 110,100],
       [ 42, 48, 89, 59, 88, 85,  89,  131, 122],
       [ 44, 47, 60,  66, 100, 70,  73,  59, 109],
       [ 38, 50, 50,  70, 121,112, 141, 103,166],
       [ 21, 73, 41,  80, 1,  130, 122, 150,120],
       [ 24, 60, 92,  89, 93, 100, 132, 147,121]]
# 可视化查看原始图像
plt.imshow(img,cmap=plt.cm.gray)
```

In [ ]:
```
# 进行均值滤波
mean_filt = Mean_Conv(3, img)
# 打印特征图的数值
print(mean_filt)
# 可视化查看滤波后的图
plt.imshow(mean_filt,cmap=plt.cm.gray)
```

中值滤波

均值滤波是在滤波器范围内求图像像素点的均值。但是，均值滤波难以完全除掉噪声点，同时可能会破坏图像的细节。

因此，通常我们更偏向使用中值滤波。中值滤波是在滤波器范围内求图像像素点的中位数。



任务2

编写Median_Filter( )及Median_Conv( )，实现中值滤波。

其中，Median_Filter( )传入4个参数，即图像、滤波器边长、滤波器中心当前对准的原图像素点的行列数。

Median_Conv( )传入2个参数，即图像和滤波器边长。该函数需要根据输入的滤波器边长，对图像进行padding操作。

In [ ]:
```
# 请在此编写中值滤波的两个函数（替换掉“Pass”）
# 中位数可由statistics包中的median函数获取，也可以自定义中位数函数。

def Median_Filter(dim,img,row,col):
    pass

def Median_Conv(dim, img):
    pass
```

Median_Filter( )和Median_Conv( )编写完成后，运行下面的Cell查看效果。

In [ ]:
```
# 进行中值滤波
median_filt = Median_Conv(3, img) # 传入的参数分别是滤波器边长、原图片、padding层数
# 打印特征图的数值
print(median_filt)
# 可视化查看特征图
plt.imshow(median_filt,cmap=plt.cm.gray)
```

把均值滤波和中值滤波用在真实图片上。真实图片请导入破损的老照片lab2_img/girl.jpg

In [ ]:
```
# 导入照片、灰度化、重设宽和高、查看图片
img = cv2.imread('lab2_img/girl.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img = cv2.resize(img, (200, 200),
                 interpolation = cv2.INTER_LINEAR)
plt.imshow(img, cmap=plt.cm.gray)
```

In [ ]:
```
# 进行均值滤波
mean_filt = Mean_Conv(7, img) #如果图像较大，运行时间可能较久
# 可视化查看特征图
plt.imshow(mean_filt, cmap=plt.cm.gray)
```

In [ ]:
```
# 进行中值滤波
median_filt = Median_Conv(7, img) #如果图像较大，运行时间可能较久
# 可视化查看特征图
plt.imshow(median_filt, cmap=plt.cm.gray)
```

第二部分：OpenCV中的物体轮廓识别

本次Lab第二部分主要内容是，利用OpenCV对简单图像中的要素轮廓进行识别。

识别不带边框的白色区域

In [ ]:
```
# 读取图片
img = cv2.imread('lab2_img/shape1.jpg')
# 回顾：OpenCV的颜色通道顺序为BGR，但plt的顺序为RGB，因此需要转换才能看到图像原本的颜色
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

In [ ]:
```
# 把彩色图像转化为灰度图像
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 预览灰度图像
RGB_gray = cv2.cvtColor(gray, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_gray)
```

In [ ]:
```
# 用直方图来辅助判断背景和物体要素之间的阈值
gray_px = np.ravel(gray)
plt.hist(gray_px)
```

In [ ]:
```
# 根据直方图，该图几乎为双色图，白色区域的灰度值大致在220以上，黑色背景的灰度值在130以下。
# 如果要用cv2.threshold二值化操作分离背景和白色区域，则分割的阈值应设在130和240之间
# cv2.threshold传入参数分别为：图片（灰度）、阈值、超过阈值像素的设置值、阈值类型
# 例如，现在让原值超过130的像素都变成255，而原值不足10的像素都变成0，从而分离要素
# 根据上一步的观察，背景区域像素为130，因此阈值要设为大于130的数
ret, thresh = cv2.threshold(gray, 130, 255, cv2.THRESH_BINARY)
```

In [ ]:
```
# 预览二值化阈值操作后的效果
RGB_thresh = cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_thresh)
```

In [ ]:
```
# 给要素物体定位轮廓
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# 在原图上绘制轮廓
cv2.drawContours(img, contours, -1, (0,220,0), 10) # 后三个参数分别是轮廓相对位置，轮廓颜色，轮廓粗细
# 计算要素的像素面积
area = cv2.contourArea(contours[0])
print('像素面积:', area, '像素')
# 这里还可以根据比例尺，将像素面积转为现实面积
# 比如，当比例尺为1像素=0.1厘米时：
scale_factor = 0.1
size = area * scale_factor ** 2
print('现实面积:', size, '平方厘米')
```

In [ ]:
```
# 转换RGB通道后，用plt预览勾勒轮廓后的图像
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

识别不带边框的有色区域

In [ ]:
```
# 录入及预览图片
img = cv2.imread('lab2_img/shape2.jpg')
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

In [ ]:
```
# 转换为灰度图
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 处理灰度图片的矩阵，观察像素值的分布
gray_px = np.ravel(gray)
plt.hist(gray_px)
```

In [ ]:
```
# 此次的背景区域灰度为226
# 因为此次要提取深色部分，因此二值化应用BINARY_INV，阈值的设置应在120和220之间，这里取210
ret, thresh = cv2.threshold(gray, 210, 255, cv2.THRESH_BINARY_INV)
# 预览二值化效果
RGB_thresh = cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_thresh)
```

In [ ]:
```
# 剩余的步骤同上
# 给要素物体定位轮廓
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# 在原图上绘制轮廓
cv2.drawContours(img, contours, -1, (0,220,0), 10)
# 预览效果
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

识别带有边框的要素

如果物体带有黑色边缘线，则可以在设置thresh的时候采用OTSU算法，省略手动设置thresh参数的步骤。

关于OTSU算法的细节，可参见：https://blog.csdn.net/X131644/article/details/131100652 (https://blog.csdn.net/X131644/article/details/131100652)

In [ ]:
```
# 录入及预览图片
img = cv2.imread('lab2_img/shape3.jpg')
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

In [ ]:
```
# 转为灰度图
gray = cv2.cvtColor(RGB_img, cv2.COLOR_BGR2GRAY)

# 设置threshold让物体和背景分离，此时采用OTSU法
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

# 绘制轮廓线
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0,220,0), 10)

#计算物体的像素面积
area = cv2.contourArea(contours[0])

print('像素面积:', area, '像素')
```

In [ ]:
```
# 显示轮廓
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

In [ ]:
```
# 在外围绘制范围框 extent box
x, y, w, h = cv2.boundingRect(contours[0])
cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

多个要素物体的识别

识别同一张图像中的多个要素，并标注每个物体的轮廓及大小

In [ ]:
```
# 录入及预览图片
img = cv2.imread('lab2_img/shape4.jpg')
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

In [ ]:
```
# 转为灰度
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 利用Binary_INV及OTSU算法分离要素和背景
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

# 定位轮廓
contours, hierarchy = cv2.findContours( thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 用一个for循环访问每一组轮廓，并计算其围起来的面积
for contour in contours:
    area = cv2.contourArea(contour)

    # 在每个对象物体的外围绘制范围框 extent box
    x, y, w, h = cv2.boundingRect(contour)
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    # 显示像素面积
    cv2.putText(img, str(area), (x, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1)
```

In [ ]:
```
# 转换RGB后用plt显示图像
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
```

任务3

下面的代码已经导入了3张图片，分别命名为fig1、fig2和fig3，已知每张图片都包含不止一个物体要素，且物体要素带边框。

请完成函数“big_obj”的编写（替代原本的pass），该函数的描述如下。

- 功能：在三张输入图像中，找出物体要素**总像素面积**最大的一张图像，并将其用plt绘制出来，且返回这张图像中所有物体要素的**平均像素面积**。
- 绘图要求：包含每个物体要素的范围框（extent box）和像素面积的标注。

In [ ]:
```
# Fig 1
fig1 = cv2.imread('lab2_img/fig1.jpg')
RGB_fig1 = cv2.cvtColor(fig1, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_fig1)
```

In [ ]:
```
# Fig 2
fig2 = cv2.imread('lab2_img/fig2.jpg')
RGB_fig2 = cv2.cvtColor(fig2, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_fig2)
```

In [ ]:
```
# Fig 3
fig3 = cv2.imread('lab2_img/fig3.jpg')
RGB_fig3 = cv2.cvtColor(fig3, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_fig3)
```

In [ ]:
```
# 替换以下的“Pass”，完成任务3

def big_obj(fig1, fig2, fig3):
    pass
```

In [ ]:
```
# 完成big_obj函数的编写后，运行下面的代码
big_obj(fig1, fig2, fig3)
```

任务4

导入toh.jpg（天坛公园的遥感图像），请编写代码，尽可能识别出公园中的人造地貌（包括建筑、道路、广场等）的分布，并勾勒其轮廓。

提示： 在进行阈值操作时，除了THRESH_BINARY和THRESH_BINARY_INV两种二值化操作之外，还有哪些常见的阈值处理方式（参考Lab1的notebook）？它们是否对此类问题更有帮助?

In [ ]:
```
# 在这里完成任务4，可以拆分为多个Cell
```

提交方式

本次作业有4个任务。完成所有cell的运行后，保存为ipynb和PDF格式（保留所有输出）。将导出的ipynb命名为“Lab2+姓名+学号.ipynb”，将导出的PDF命名为“Lab2+姓名+学号.pdf”，并将上述两个文件提交到学习通作业模块的相应位置。 PDF格式文件可以从页面左上角的“File/文件”菜单里选择通过HTML或LaTeX导出，也可以通过打印预览，保存为PDF。 请独立完成练习，参考答案将在截止时间后公布。 截止时间：2024年5月15日23:59