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

Lab 3

第一部分：高斯/Prewitt/Sobel滤波

高斯滤波

In [ ]:
```
#导入卓别林的照片并查看
img = cv2.imread('lab3_img/chaplin.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img,cmap=plt.cm.gray)
```

In [ ]:
```
# 手动设置一个简单的高斯滤波器
kernel_gaussian = np.array([[1, 2, 1],
                    [2, 4, 2],
                    [1, 2, 1]])/16
# 用可视化查看
plt.imshow(kernel_gaussian,cmap=plt.cm.gray)
```

In [ ]:
```
# 用滤波器处理图像并预览
filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_gaussian)
plt.imshow(filtered,cmap=plt.cm.gray)
```

In [ ]:
```
# 手动设置一个更大的高斯滤波
kernel_gaussian = np.array([[1, 4, 7, 4, 1],
                            [4,16,26,16, 4],
                            [7,26,41,26, 7],
                            [4,16,26,16, 4],
                            [1, 4, 7, 4, 1]])/273

# 用可视化查看
plt.imshow(kernel_gaussian,cmap=plt.cm.gray)
```

In [ ]:
```
# 用滤波器处理图像并预览
filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_gaussian)
plt.imshow(filtered,cmap=plt.cm.gray)
```

任务1

编写一个生成高斯滤波器的函数。该函数输入两个参数：高斯滤波器的边长、标准差sigma。返回高斯滤波器数组。

In [ ]:
```
# 替换下面的pass，完成gaussian_filter()函数
def gaussian_filter(kernel_size=3, sigma=1):
    pass
```

In [ ]:
```
# 用上面的gaussian_filter()函数创建一个自定义的高斯核，让其边长为30，标准差为5
kernel_size=30 # 边长
sigma = 5 # 标准差

# 创建高斯核并可视化查看
custom_gaussian = gaussian_filter(kernel_size, sigma)
plt.imshow(custom_gaussian,cmap=plt.cm.gray)
```

In [ ]:
```
# 用自定义的高斯滤波器处理图像并预览
filtered = cv2.filter2D(src=img, ddepth=-1, kernel=custom_gaussian)
plt.imshow(filtered,cmap=plt.cm.gray)
```

In [ ]:
```
# 用上面的gaussian_filter()函数创建一个自定义的高斯核，让其边长为30，标准差为10
kernel_size=30 # 边长
sigma = 10 # 标准差

# 创建高斯核并可视化plt.imshow(custom_gaussian)查看
custom_gaussian = gaussian_filter(kernel_size, sigma)
plt.imshow(custom_gaussian,cmap=plt.cm.gray)
```

In [ ]:
```
# 用自定义的高斯滤波器处理图像并预览
filtered = cv2.filter2D(src=img, ddepth=-1, kernel=custom_gaussian)
plt.imshow(filtered,cmap=plt.cm.gray)
```

Prewitt 滤波器

In [ ]:
```
# 手动设置一个Prewitt水平梯度滤波器
kernel_prewitt = np.array([[-1, 0, 1],
                           [-1, 0, 1],
                           [-1, 0, 1]])
# 用可视化查看
plt.imshow(kernel_prewitt,cmap=plt.cm.gray)
```

In [ ]:
```
# 对卓别林的照片进行边缘检测并输出结果
# Prewitt水平梯度滤波器检测的是竖直方向的边缘

filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_prewitt)
plt.imshow(filtered,cmap=plt.cm.gray)
```

In [ ]:
```
# 改变Prewitt的方向，设置垂直梯度的滤波器
kernel_prewitt = np.array([[-1, -1, -1],
                           [0,  0,  0 ],
                           [1,  1,  1 ]])
# 用可视化查看
plt.imshow(kernel_prewitt,cmap=plt.cm.gray)
```

In [ ]:
```
# 对图像进行边缘检测并输出结果
# Prewitt垂直梯度滤波器检测的是水平方向的边缘

filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_prewitt)
plt.imshow(filtered,cmap=plt.cm.gray)
```

In [ ]:
```
# 尺度更大的水平梯度滤波器
kernel_prewitt = np.array([[-2, -1, 0, 1, 2],
                           [-2, -1, 0, 1, 2],
                           [-2, -1, 0, 1, 2],
                           [-2, -1, 0, 1, 2],
                           [-2, -1, 0, 1, 2]])
# 用可视化查看
plt.imshow(kernel_prewitt,cmap=plt.cm.gray)
```

In [ ]:
```
# 对图像进行边缘检测并输出结果

filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_prewitt)
plt.imshow(filtered,cmap=plt.cm.gray)
```

In [ ]:
```
# 尺度更大的垂直梯度滤波器
kernel_prewitt = np.array([[-2, -2, -2, -2, -2],
                           [-1, -1, -1, -1, -1],
                           [0,   0,  0,  0,  0],
                           [1,   1,  1,  1,  1],
                           [2,   2,  2,  2,  2]])
# 用可视化查看
plt.imshow(kernel_prewitt,cmap=plt.cm.gray)
```

In [ ]:
```
# 对图像进行边缘检测并输出结果

filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_prewitt)
plt.imshow(filtered,cmap=plt.cm.gray)
```

任务2

编写一个函数，用于自定义Prewitt滤波器。它传入2个参数：

- 第1个参数为滤波器的边长。
- 第2参数为布尔值，当它为True时，输出水平梯度滤波器，当它为False时，输出垂直梯度滤波器。

In [ ]:
```
# 替换下面的pass，完成函数
def prewitt(kernel_size = 3, is_horizontal = True):
    pass
```

In [ ]:
```
# 完成prewitt()后，运行下面的代码
# 利用prewitt函数，自定义一个边长为11的水平梯度滤波器，以及一个边长为11的垂直梯度滤波器
# 并对以下图像进行边缘检测
img_building = cv2.imread('lab3_img/building.jpg')
img_building = cv2.cvtColor(img_building, cv2.COLOR_BGR2GRAY)

prewitt_h = np.array(prewitt(11, True))
filtered_h = cv2.filter2D(src=img_building, ddepth=-1, kernel=prewitt_h)

prewitt_v = np.array(prewitt(11, False))
filtered_v = cv2.filter2D(src=img_building, ddepth=-1, kernel=prewitt_v)

fig, axes = plt.subplots(3, 1, figsize = (14,14))
plt.gray()
axes[0].imshow(img_building)
axes[1].imshow(filtered_h)
axes[2].imshow(filtered_v)

titles = ["original", "horizontal gradient (vertical edge)", "vertical gradient (horizontal edge)"]
for i, ax in enumerate(axes.ravel()):
    ax.set_title(titles[i])
    ax.axis("off")
plt.show()
```

Sobel滤波器

In [ ]:
```
#导入人造卫星的图片并查看
img = cv2.imread('lab3_img/satellite.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img)
```

In [ ]:
```
# 手动设置一个Sobel水平梯度滤波器
kernel_sobel = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])
# 用可视化查看结果
plt.imshow(kernel_sobel,cmap=plt.cm.gray)
```

In [ ]:
```
# 对卓别林的照片进行边缘检测并输出结果
# Sobel水平梯度滤波器检测的是竖直方向的边缘

filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_sobel)
plt.imshow(filtered)
```

In [ ]:
```
# 手动设置一个Sobel垂直梯度滤波器
kernel_sobel = np.array([[ 1,  2,  1],
                         [ 0,  0,  0],
                         [-1, -2, -1]])
# 用可视化查看结果
plt.imshow(kernel_sobel,cmap=plt.cm.gray)
```

In [ ]:
```
# 对卓别林的照片进行边缘检测并输出结果
# Sobel垂直梯度滤波器检测的是水平方向的边缘

filtered = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_sobel)
plt.imshow(filtered)
```

In [ ]:
```
# 我们可以将水平和垂直两个方向的Sobel滤波器联合使用，提取所有的边缘
# 重新定义两个Sobel滤波器，一个为水平梯度，一个为垂直梯度
kernel_sobel_h = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]])
kernel_sobel_v = np.array([[ 1,  2,  1],
                           [ 0,  0,  0],
                           [-1, -2, -1]])

# 分别滤波，提取相应方向的边缘
filtered_h = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_sobel_v) #注意，水平方向的边缘由垂直梯度的滤波器提取
filtered_v = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_sobel_h) #垂直方向的边缘由水平梯度的滤波器提取

# 把两个方向的滤波结果结合起来
filtered_combined = np.sqrt(np.square(filtered_h) + np.square(filtered_v))
filtered_combined *= 255.0 / filtered_combined.max() # 把结果重新投射到0-255区间
plt.imshow(filtered_combined)
```

拆分2D卷积核为两个1D卷积核

In [ ]:
```
# 定义第一组1D卷积核
sobel1_v = np.array([1,2,1])[:,None] #列
sobel1_h = np.array([-1,0,1])[None,:] #行

print(sobel1_v)
print(sobel1_h)
```

In [ ]:
```
# 查看这两个1D卷积核的Kronecker积是否为Sobel滤波器
print(np.kron(sobel1_v, sobel1_h))
```

In [ ]:
```
# 用这两个1D滤波器完成滤波
filtered1 = cv2.sepFilter2D(img, -1, sobel1_v, sobel1_h)
plt.imshow(filtered1)
```

In [ ]:
```
# 定义第二组1D卷积核
sobel2_v = np.array([-1,0,1])[:,None] #列
sobel2_h = np.array([1,2,1])[None,:] #行

# 用这两个1D滤波器完成滤波
filtered2 = cv2.sepFilter2D(img, -1, sobel2_v, sobel2_h)
plt.imshow(filtered2)
```

In [ ]:
```
# 把两个方向的滤波结果结合起来
filtered_combined = np.sqrt(np.square(filtered1) + np.square(filtered2))
filtered_combined *= 255.0 / filtered_combined.max() # 把结果重新投射到0-255区间
plt.imshow(filtered_combined)
```

对比运行速度

In [ ]:
```
import time

sobel1_v = np.array([1,2,1])[:,None]
sobel1_h = np.array([-1,0,1])[None,:]
sobel2_v = np.array([-1,0,1])[:,None]
sobel2_h = np.array([1,2,1])[None,:]

kernel_sobel_h = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]])
kernel_sobel_v = np.array([[ 1,  2,  1],
                           [ 0,  0,  0],
                           [-1, -2, -1]])
# 因为每次运行都有误差，因此做1000次实验查看结果
res_1d = []
res_2d = []
# 分解为1D滤波器所用时间:
for i in range(1,1001):
    timestart = time.time()
    filtered1 = cv2.sepFilter2D(img, -1, sobel1_v, sobel1_h)
    filtered2 = cv2.sepFilter2D(img, -1, sobel2_v, sobel2_h)
    time_used = time.time() - timestart
    res_1d.append(time_used)
# 直接用2D滤波器所用时间:
for i in range(1,1001):
    timestart = time.time()
    filtered_h = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_sobel_v)
    filtered_v = cv2.filter2D(src=img, ddepth=-1, kernel=kernel_sobel_h)
    time_used = time.time() - timestart
    res_2d.append(time_used)
```

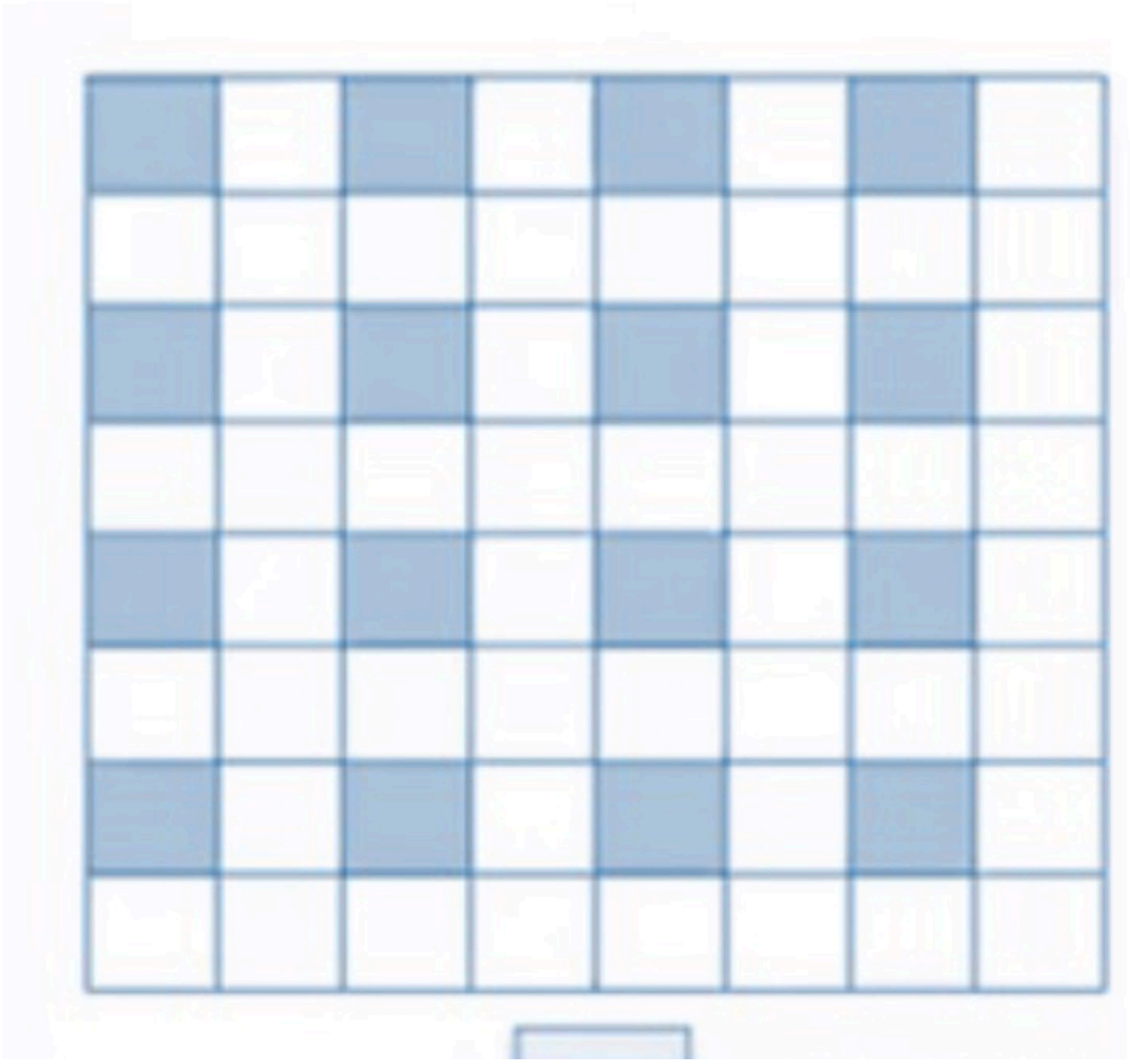
In [ ]:
```
print("直接用2D滤波器所用的平均时间：", np.mean(res_2d),"秒")
print("分解为1D滤波器所用平均时间：", np.mean(res_1d),"秒")
```

第二部分：图像金字塔

In [ ]:
```
# 导入示例图片、灰度化、重设宽和高、查看图片
img = cv2.imread('lab3_img/wukong.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img = cv2.resize(img, (180, 180),
                 interpolation = cv2.INTER_LINEAR)
plt.imshow(img, cmap=plt.cm.gray)
```

任务3

编写降采样函数downsampling()，它传入一张灰度图片，输出1/4尺度的机械降采样结果，采样方式如果下图所示，即每4个像素中，保留左上方的一个像素。



In [ ]:
```
# 替换下面的pass，完成函数
def downsampling(img):
    pass
```

任务4

编写机械降采样图像金字塔的函数raw_pyramid()。

- 输入2个参数：图片、降采样次数（factor）
- 输出一个数组，里面包含原图，以及金字塔每一层的图片
- 报错：如果下采样次数过多，导致输出图像边长<10，报错“下采样次数过多”

In [ ]:
```
# 替换下面的pass，完成函数

def raw_pyramid(img, factor):
    pass
```

任务5

编写组建高斯金字塔及拉普拉斯金字塔的函数gaussian_pyramid()

- 输入参数：图片、降采样次数（factor），高斯核
- 输出一个数组，里面包含两个数组，第一个为高斯金字塔的各层图像（包括原图），第二个为拉普拉斯金字塔的各层图像
- 报错1：如果下采样次数过多，导致输出图像边长<10，报错“下采样次数过多”
- 报错2：如果下采样次数过多，导致输出图像边长<高斯核边长，报错“下采样次数过多”

In [ ]:
```
# 替换下面的pass，完成函数
def gaussian_pyramid(img, factor, kernel):
    pass
```

完成以上三个函数的编写后，运行下面的代码，查看高斯金字塔和机械降采样金字塔的效果对比。

In [ ]:
```
# 用前面的guassian_filter函数定义一个高斯核
kernel = gaussian_filter(9, 1)

# 创建一个高斯金字塔及拉普拉斯金字塔（下采样操作次数为3，共4层）
gaussian_py = gaussian_pyramid(img, 3, kernel)[0]

# 创建一个普通的机械降采样金字塔
normal_py = raw_pyramid(img, 3)

# 计算每一层图像的大小
size = []
for i in gaussian_py:
    size.append(i.shape[0])

fig, axes = plt.subplots(len(gaussian_py), 2, figsize = (18,18), gridspec_kw={'height_ratios': size})
plt.gray()
for i in range(len(gaussian_py)):
    axes[i][0].imshow(gaussian_py[i])
    axes[i][1].imshow(normal_py[i])

plt.show()
```

运行下面的代码，查看高斯金字塔和拉普拉斯金字塔的对比。

In [ ]:
```
# 用前面的guassian_filter函数定义一个高斯核
kernel = gaussian_filter(9, 1)

# 创建一个高斯金字塔及拉普拉斯金字塔
gaussian_py = gaussian_pyramid(img, 3, kernel)[0]
laplacian_py = gaussian_pyramid(img, 3, kernel)[1]

# 计算每一层图像的大小
size = []
for i in gaussian_py:
    size.append(i.shape[0])

fig, axes = plt.subplots(len(gaussian_py), 2, figsize = (18,18), gridspec_kw={'height_ratios': size})
plt.gray()
for i in range(len(laplacian_py)):
    axes[i][0].imshow(gaussian_py[i])
    axes[i][1].imshow(laplacian_py[i])
# 拉普拉斯金字塔少一层，因此高斯金字塔需要再加一层
axes[len(gaussian_py)-1][0].imshow(gaussian_py[len(gaussian_py)-1])

plt.axis('off')
plt.show()
```

提交方式

本次作业有5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“Lab3+姓名+学号.ipynb”，将导出的PDF命名为“Lab3+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。 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21日23:59。超时1天之内将扣除5%的分数，超时1天以上将扣除10%的分数。