Computer Vision Homework 10

R08922079 資工所一 洪浩翔

Part 0

```python
def readImg(filename='lena.bmp'):
    #read img
    image = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    return image
```

```python
def lapMask(img, kernel, paddingSize, threshold, name):
    w,h = img.shape
    image = np.pad(img, ((paddingSize,paddingSize),(paddingSize,paddingSize)), 'constant', constant_values=0)
    print("check shape:", image.shape, "with paddingSize:", paddingSize)
    tmp = np.zeros(img.shape)
    for i in range(paddingSize, w+paddingSize):
        for j in range(paddingSize, h+paddingSize):
            value = np.sum(image[i-paddingSize:i+paddingSize+1, j-paddingSize:j+paddingSize+1]*kernel)
            if value >= threshold:
                tmp[i-paddingSize,j-paddingSize] = 1
            elif value <= -threshold:
                tmp[i-paddingSize,j-paddingSize] = -1
            else:
                tmp[i-paddingSize,j-paddingSize] = 0
    tmp = np.pad(tmp, ((paddingSize,paddingSize),(paddingSize,paddingSize)), 'constant', constant_values=0)
    output = img.copy()
    for i in range(paddingSize, w+paddingSize):
        for j in range(paddingSize, h+paddingSize):
            output[i-paddingSize,j-paddingSize] = 255
            if tmp[i,j] == 1:
                if -1 in tmp[i-1:i+2, j-1:j+2]:
                    output[i-paddingSize,j-paddingSize] = 0
    cv2.imwrite(name+".jpg", output)
    return output
```

```python
def maskFactory(name):
    if name == "LM1":
        return np.array([[0,1,0],[1,-4,1],[0,1,0]])
    elif name == "LM2":
        return np.array([[1,1,1],[1,-8,1],[1,1,1]])/3.0
    elif name == "MVL":
        return np.array([[2,-1,2],[-1,-4,-1],[2,-1,2]])/3.0
    elif name == "LOG":
        return np.array([[0 , 0 , 0 , -1 , -1 , -2 , -1 , -1 , 0 , 0 , 0] ,
           [0 , 0 , -2 , -4 , -8 , -9 , -8 , -4 , -2 , 0 , 0] ,
           [0 , -2 , -7 , -15 , -22 , -23 , -22 , -15 , -7 , -2 , 0] ,
           [-1 , -4 , -15 , -24 , -14 , -1 , -14 , -24 , -15 , -4 , -1] ,
           [-1 , -8 , -22 , -14 , 52 , 103 , 52 , -14 , -22 , -8 , -1] ,
           [-2 , -9 , -23 , -1 , 103 , 178 , 103 , -1 , -23 , -9 , -2] ,
           [-1 , -8 , -22 , -14 , 52 , 103 , 52 , -14 , -22 , -8 , -1] ,
           [-1 , -4 , -15 , -24 , -14 , -1 , -14 , -24 , -15 , -4 , -1] ,
           [0 , -2 , -7 , -15 , -22 , -23 , -22 , -15 , -7 , -2 , 0] ,
           [0 , 0 , -2 , -4 , -8 , -9 , -8 , -4 , -2 , 0 , 0] ,
           [0 , 0 , 0 , -1 , -1 , -2 , -1 , -1 , 0 , 0 , 0]])
    elif name == "DOG":
        return np.array([[-1 , -3 , -4 , -6 , -7 , -8 , -7 , -6 , -4 , -3 , -1] ,
           [-3 , -5 , -8 , -11 , -13 , -13 , -13 , -11 , -8 , -5 , -3] ,
           [-4 , -8 , -12 , -16 , -17 , -17 , -17 , -16 , -12 , -8 , -4] ,
           [-6 , -11 , -16 , -16 , 0 , 15 , 0 , -16 , -16 , -11 , -6] ,
           [-7 , -13 , -17 , 0 , 85 , 160 , 85 , 0 , -17 , -13 , -7] ,
           [-8 , -13 , -17 , 15 , 160 , 283 , 160 , 15 , -17 , -13 , -8] ,
           [-7 , -13 , -17 , 0 , 85 , 160 , 85 , 0 , -17 , -13 , -7] ,
           [-6 , -11 , -16 , -16 , 0 , 15 , 0 , -16 , -16 , -11 , -6] ,
           [-4 , -8 , -12 , -16 , -17 , -17 , -17 , -16 , -12 , -8 , -4] ,
           [-3 , -5 , -8 , -11 , -13 , -13 , -13 , -11 , -8 , -5 , -3] ,
           [-1 , -3 , -4 , -6 , -7 , -8 , -7 , -6 , -4 , -3 , -1]])
    else:
        print('error name')
        return []
```

Part 1

Laplacian Mask 1

Kernel: `([[0,1,0],[1,-4,1],[0,1,0]])`

Threshold: 15

Code:

```
lm1 = maskFactory("LM1")
lapMask(image.copy(), lm1, int(lm1.shape[0]/2), 15, "lm1")
```

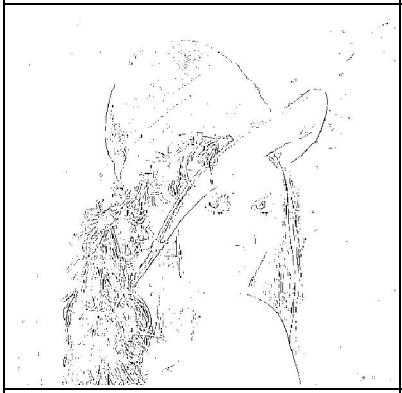
Laplacian Mask 2

Kernel: `([[1,1,1],[1,-8,1],[1,1,1]])/3.0`

Threshold: 15

Code:

```
lm2 = maskFactory("LM2")
lapMask(image.copy(), lm2, int(lm2.shape[0]/2), 15, "lm2")
```



Minimum Variance Laplacian

Kernel: `([[2,-1,2],[-1,-4,-1],[2,-1,2]])/3.0`

Threshold: 20

Code:

```
mvl = maskFactory("MVL")
lapMask(image.copy(), mvl, int(mvl.shape[0]/2), 20, "mvl")
```

Laplacian of Gaussian

Kernel:

```
return np.array([[0 , 0 , 0 , -1 , -1 , -2 , -1 , -1 , 0 , 0 , 0] ,
   [0 , 0 , -2 , -4 , -8 , -9 , -8 , -4 , -2 , 0 , 0] ,
   [0 , -2 , -7 , -15 , -22 , -23 , -22 , -15 , -7 , -2 , 0] ,
   [-1 , -4 , -15 , -24 , -14 , -1 , -14 , -24 , -15 , -4 , -1] ,
   [-1 , -8 , -22 , -14 , 52 , 103 , 52 , -14 , -22 , -8 , -1] ,
   [-2 , -9 , -23 , -1 , 103 , 178 , 103 , -1 , -23 , -9 , -2] ,
   [-1 , -8 , -22 , -14 , 52 , 103 , 52 , -14 , -22 , -8 , -1] ,
   [-1 , -4 , -15 , -24 , -14 , -1 , -14 , -24 , -15 , -4 , -1] ,
   [0 , -2 , -7 , -15 , -22 , -23 , -22 , -15 , -7 , -2 , 0] ,
   [0 , 0 , -2 , -4 , -8 , -9 , -8 , -4 , -2 , 0 , 0] ,
   [0 , 0 , 0 , -1 , -1 , -2 , -1 , -1 , 0 , 0 , 0]])
```

Threshold: 3000

Code:

```
log = maskFactory("LOG")
lapMask(image.copy(), log, int(log.shape[0]/2), 3000, "log")
```

Difference of Gaussian

Kernel:

```
return np.array([[-1 , -3 , -4 , -6 , -7 , -8 , -7 , -6 , -4 , -3 , -1] ,
   [-3 , -5 , -8 , -11 , -13 , -13 , -13 , -11 , -8 , -5 , -3] ,
   [-4 , -8 , -12 , -16 , -17 , -17 , -17 , -16 , -12 , -8 , -4] ,
   [-6 , -11 , -16 , -16 , 0 , 15 , 0 , -16 , -16 , -11 , -6] ,
   [-7 , -13 , -17 , 0 , 85 , 160 , 85 , 0 , -17 , -13 , -7] ,
   [-8 , -13 , -17 , 15 , 160 , 283 , 160 , 15 , -17 , -13 , -8] ,
   [-7 , -13 , -17 , 0 , 85 , 160 , 85 , 0 , -17 , -13 , -7] ,
   [-6 , -11 , -16 , -16 , 0 , 15 , 0 , -16 , -16 , -11 , -6] ,
   [-4 , -8 , -12 , -16 , -17 , -17 , -17 , -16 , -12 , -8 , -4] ,
   [-3 , -5 , -8 , -11 , -13 , -13 , -13 , -11 , -8 , -5 , -3] ,
   [-1 , -3 , -4 , -6 , -7 , -8 , -7 , -6 , -4 , -3 , -1]])
```

Threshold: 1

Code:

```
dog = maskFactory("DOG")
lapMask(image.copy(), dog, int(dog.shape[0]/2), 1, "dog")
```

```
def main():
    image = readImg()

    lm1 = maskFactory("LM1")
    lapMask(image.copy(), lm1, int(lm1.shape[0]/2), 15, "lm1")
    lm2 = maskFactory("LM2")
    lapMask(image.copy(), lm2, int(lm2.shape[0]/2), 15, "lm2")

    mvl = maskFactory("MVL")
    lapMask(image.copy(), mvl, int(mvl.shape[0]/2), 20, "mvl")

    log = maskFactory("LOG")
    lapMask(image.copy(), log, int(log.shape[0]/2), 3000, "log")
    dog = maskFactory("DOG")
    lapMask(image.copy(), dog, int(dog.shape[0]/2), 1, "dog")

if __name__ == "__main__":
    main()
```

Main code fragment