Computer Vision HW9 Report

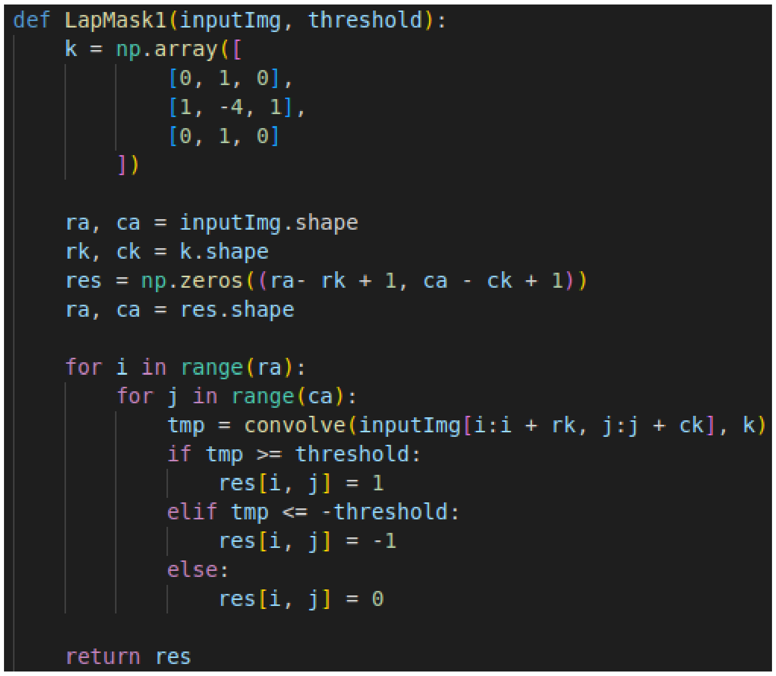
蕭恩慈 / B07902095

The task is to implement the following:

a) 2 Laplace Mask (threshold = 15)
b) Minimum Variance Laplacian (threshold = 30)
c) Laplacian of Gaussian (threshold = 3000)
d) Difference of Gaussian (DoG: inhibitory sigma = 3, excitatory sigma = 1, and kernel size 11 x 11)

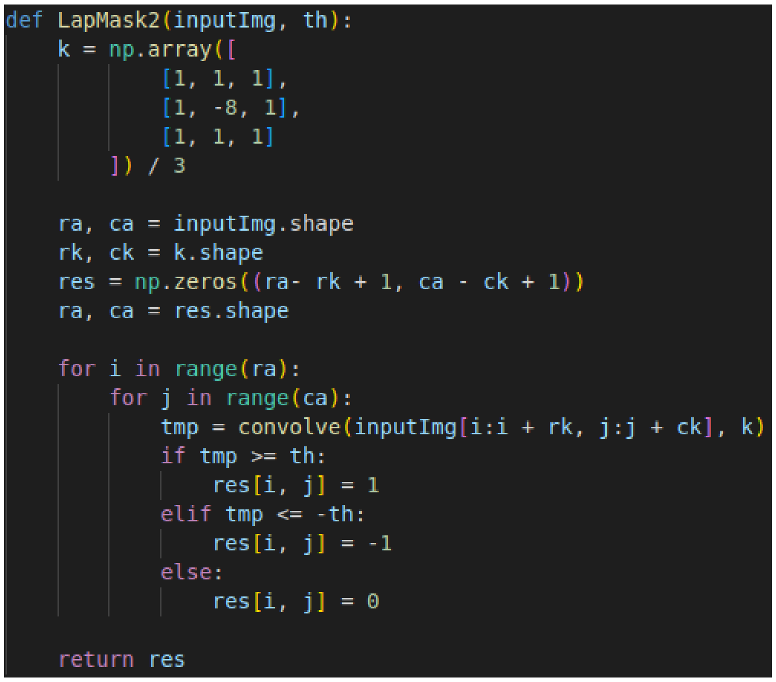
To finish the task, the language that is used to make the program is Python, with cv2 and numpy libraries. The image input that is going to be implemented is the original image lena.bmp. In order to run the code, the original image has been included in the directory to create the image results. First, write each implementation based on the guide from the class slides. First do the convolve function to be used in the functions, then continue to write each function of the task according to the matrices on the slides for each function's kernel. Lastly, do the function for zero-crossing for the output image.

Code snippet of first Laplacian Mask:



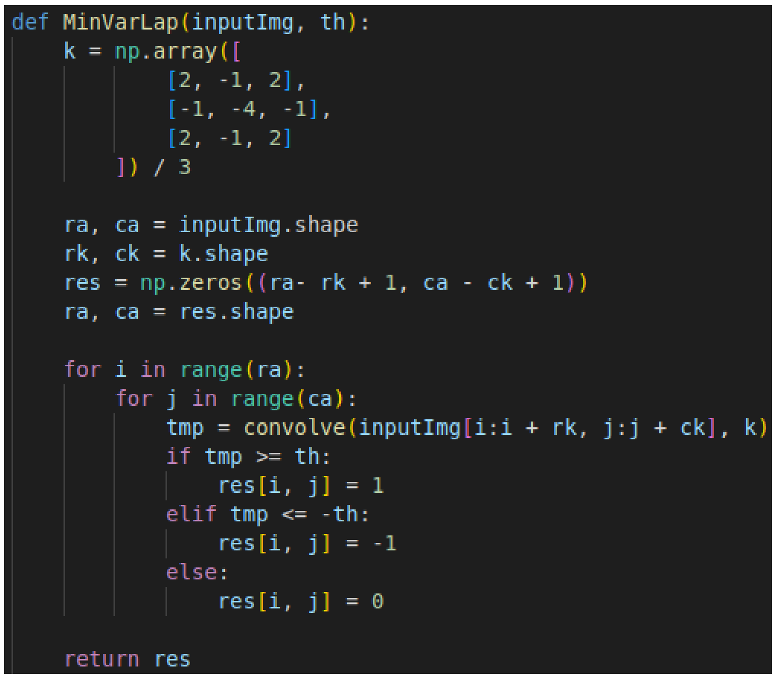
```python
def LapMask1(inputImg, threshold):
    k = np.array([
            [0, 1, 0],
            [1, -4, 1],
            [0, 1, 0]
        ])

    ra, ca = inputImg.shape
    rk, ck = k.shape
    res = np.zeros((ra- rk + 1, ca - ck + 1))
    ra, ca = res.shape

    for i in range(ra):
        for j in range(ca):
            tmp = convolve(inputImg[i:i + rk, j:j + ck], k)
            if tmp >= threshold:
                res[i, j] = 1
            elif tmp <= -threshold:
                res[i, j] = -1
            else:
                res[i, j] = 0

    return res
```

Code snippet of second Laplacian Mask:



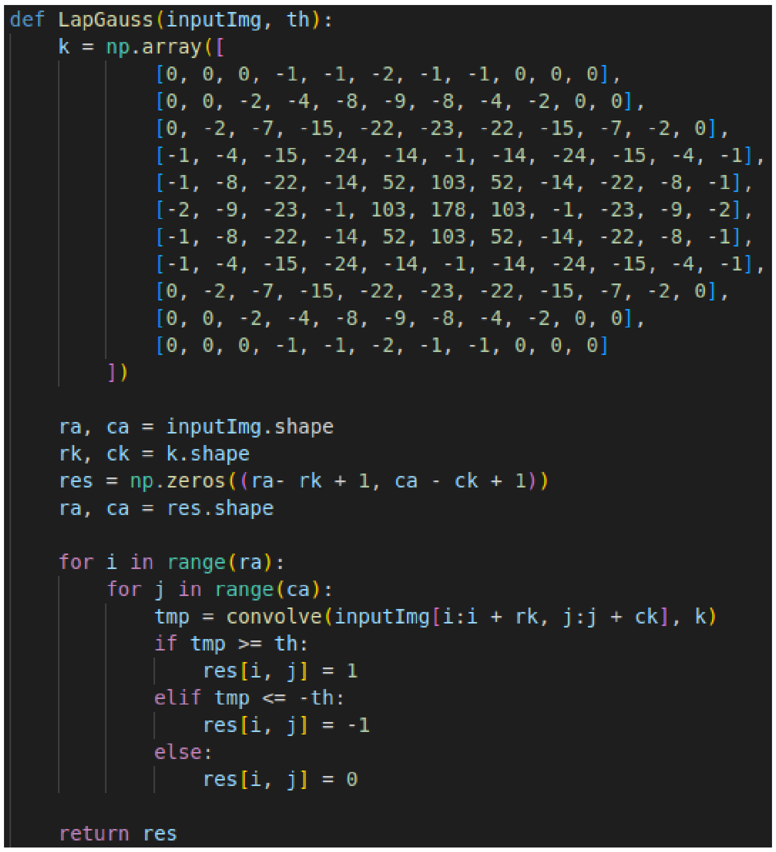
```python
def LapMask2(inputImg, th):
    k = np.array([
            [1, 1, 1],
            [1, -8, 1],
            [1, 1, 1]
        ]) / 3

    ra, ca = inputImg.shape
    rk, ck = k.shape
    res = np.zeros((ra- rk + 1, ca - ck + 1))
    ra, ca = res.shape

    for i in range(ra):
        for j in range(ca):
            tmp = convolve(inputImg[i:i + rk, j:j + ck], k)
            if tmp >= th:
                res[i, j] = 1
            elif tmp <= -th:
                res[i, j] = -1
            else:
                res[i, j] = 0

    return res
```

Code snippet of minimum-variance Laplacian:



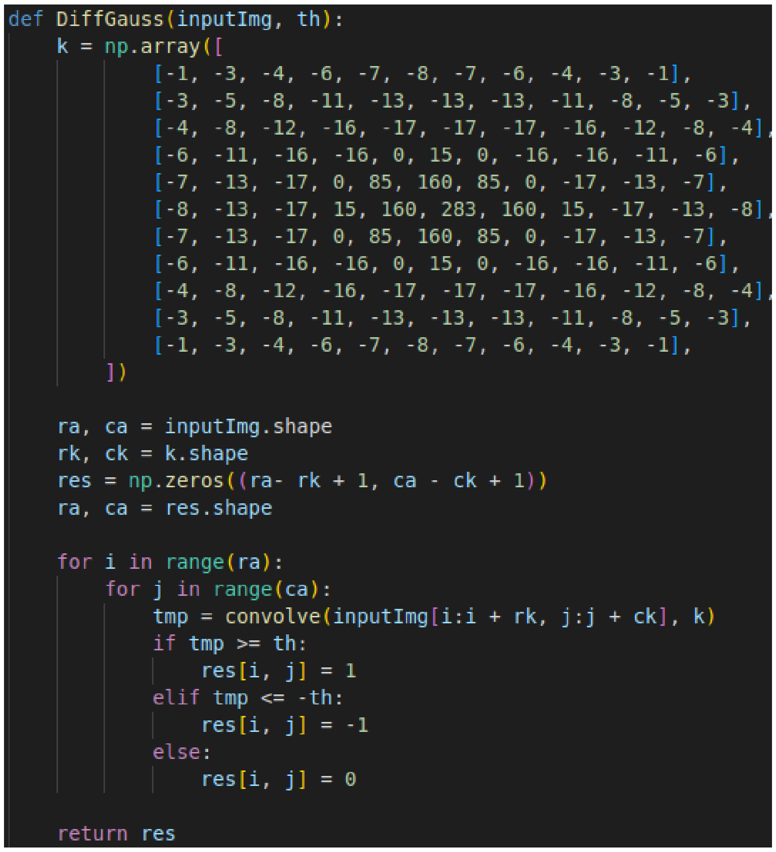
```python
def MinVarLap(inputImg, th):
    k = np.array([
            [2, -1, 2],
            [-1, -4, -1],
            [2, -1, 2]
        ]) / 3

    ra, ca = inputImg.shape
    rk, ck = k.shape
    res = np.zeros((ra- rk + 1, ca - ck + 1))
    ra, ca = res.shape

    for i in range(ra):
        for j in range(ca):
            tmp = convolve(inputImg[i:i + rk, j:j + ck], k)
            if tmp >= th:
                res[i, j] = 1
            elif tmp <= -th:
                res[i, j] = -1
            else:
                res[i, j] = 0

    return res
```

Code snippet of Laplacian of Gaussian:



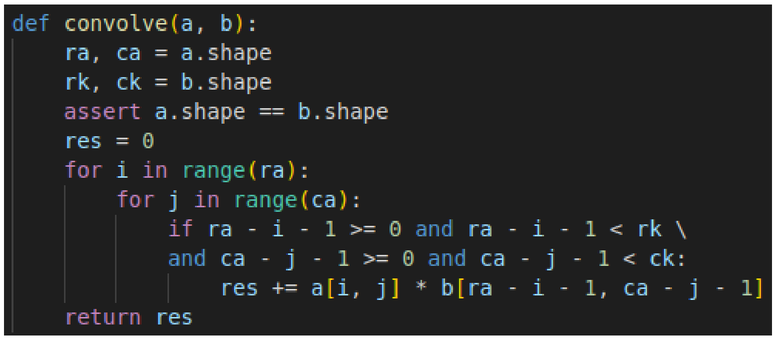
```python
def LapGauss(inputImg, th):
    k = np.array([
            [0, 0, 0, -1, -1, -2, -1, -1, 0, 0, 0],
            [0, 0, -2, -4, -8, -9, -8, -4, -2, 0, 0],
            [0, -2, -7, -15, -22, -23, -22, -15, -7, -2, 0],
            [-1, -4, -15, -24, -14, -1, -14, -24, -15, -4, -1],
            [-1, -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]
        ])

    ra, ca = inputImg.shape
    rk, ck = k.shape
    res = np.zeros((ra- rk + 1, ca - ck + 1))
    ra, ca = res.shape

    for i in range(ra):
        for j in range(ca):
            tmp = convolve(inputImg[i:i + rk, j:j + ck], k)
            if tmp >= th:
                res[i, j] = 1
            elif tmp <= -th:
                res[i, j] = -1
            else:
                res[i, j] = 0

    return res
```

Code snippet of Difference of Gaussian:

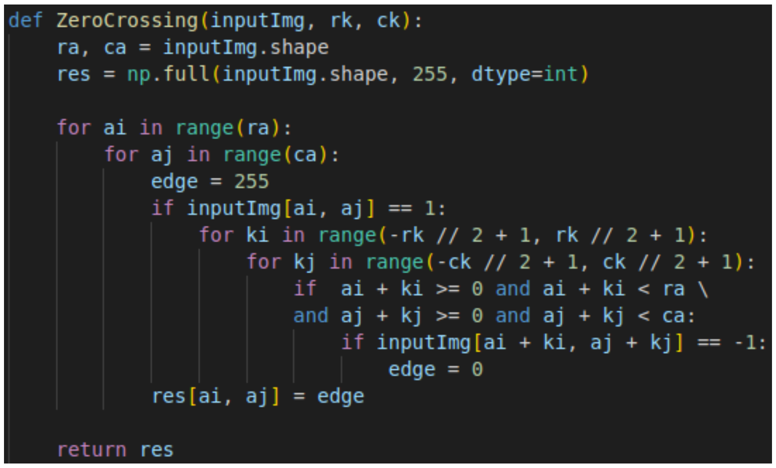


```
def DiffGauss(inputImg, th):
    k = np.array([
            [-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1],
            [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
            [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
            [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
            [-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],
        ])

    ra, ca = inputImg.shape
    rk, ck = k.shape
    res = np.zeros((ra- rk + 1, ca - ck + 1))
    ra, ca = res.shape

    for i in range(ra):
        for j in range(ca):
            tmp = convolve(inputImg[i:i + rk, j:j + ck], k)
            if tmp >= th:
                res[i, j] = 1
            elif tmp <= -th:
                res[i, j] = -1
            else:
                res[i, j] = 0

    return res
```

Code snippet for convolving and zero-crossing:



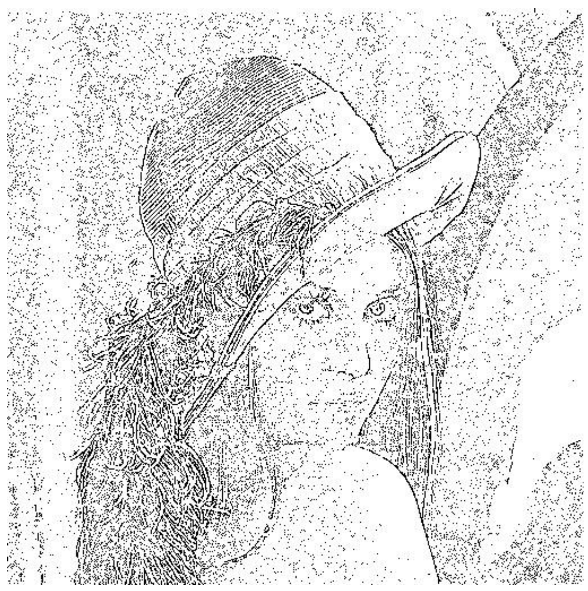
```
def convolve(a, b):
    ra, ca = a.shape
    rk, ck = b.shape
    assert a.shape == b.shape
    res = 0
    for i in range(ra):
        for j in range(ca):
            if ra - i - 1 >= 0 and ra - i - 1 < rk \
            and ca - j - 1 >= 0 and ca - j - 1 < ck:
                res += a[i, j] * b[ra - i - 1, ca - j - 1]
    return res
```



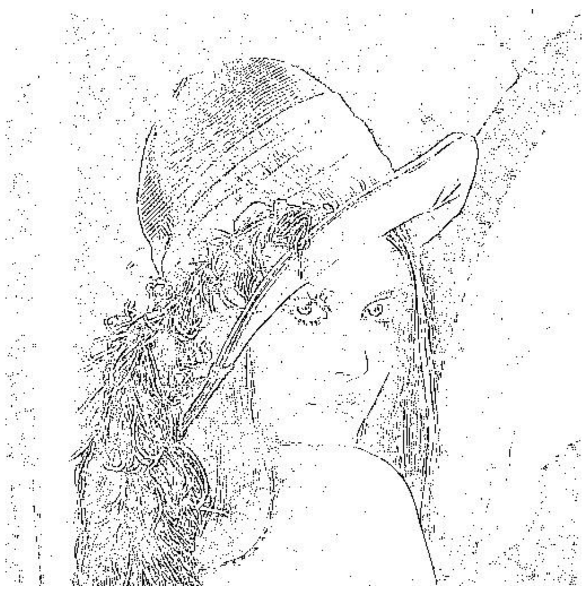
```
def ZeroCrossing(inputImg, rk, ck):
    ra, ca = inputImg.shape
    res = np.full(inputImg.shape, 255, dtype=int)

    for ai in range(ra):
        for aj in range(ca):
            edge = 255
            if inputImg[ai, aj] == 1:
                for ki in range(-rk // 2 + 1, rk // 2 + 1):
                    for kj in range(-ck // 2 + 1, ck // 2 + 1):
                        if  ai + ki >= 0 and ai + ki < ra \
                        and aj + kj >= 0 and aj + kj < ca:
                            if inputImg[ai + ki, aj + kj] == -1:
                                edge = 0
            res[ai, aj] = edge

    return res
```

Results:

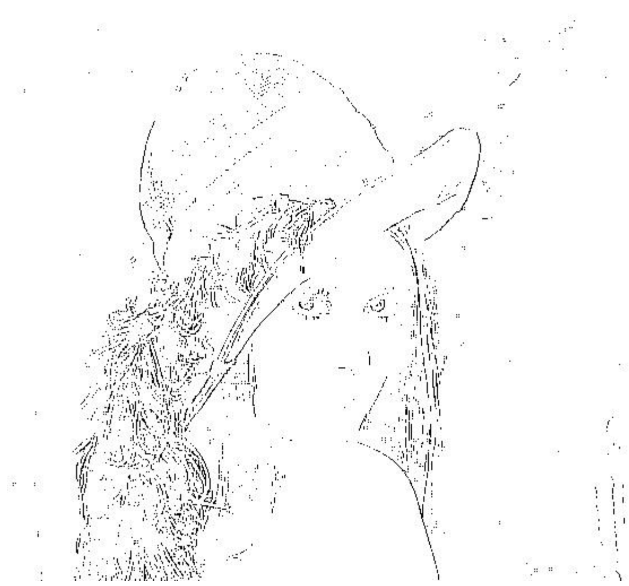
First Laplacian Mask



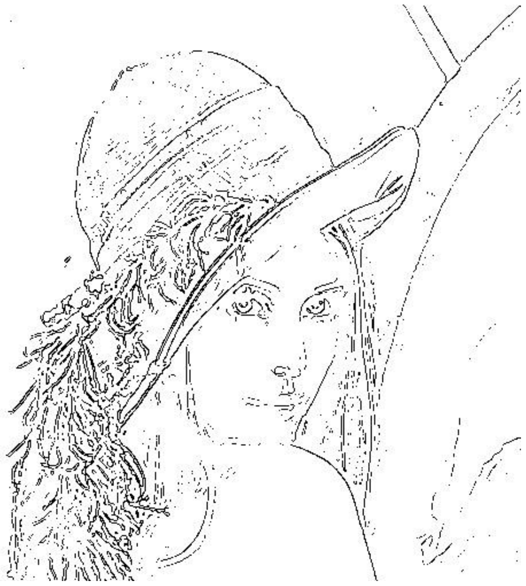
Second Laplacian Mask



Minimum-variance Laplacian



Laplacian of Gaussian



Difference of Gaussian: