B06902042 劉愷為

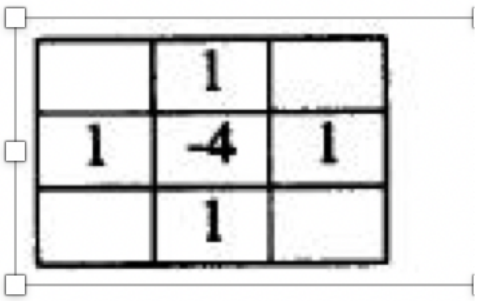
HW10

以下部分都會使用到extend，這是我用來擴增矩陣的函數

```
def extend(img):
    img_ext = np.zeros((img.shape[0] + 2, img.shape[1] + 2))
    img_ext[0, 1:-1] = img[0, :]
    img_ext[-1, 1:-1] = img[-1, :]
    img_ext[1:-1, 0] = img[:, 0]
    img_ext[1:-1, -1] = img[:, -1]
    img_ext[0, 0] = img[0, 0]
    img_ext[0, -1] = img[0, -1]
    img_ext[-1, 0] = img[-1, 0]
    img_ext[-1, -1] = img[-1, -1]
    img_ext[1:-1, 1:-1] = img[:, :]
    return img_ext
```

Part1: Laplace type1(threshold = 15)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果gradient大於threshold且周遭有任一gradient小於-threshold，把pixel值設定成0。g即為乘完矩陣的值。



```
def laplace1(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    tmp1 = np.zeros(img.shape, np.int)

    for x in range (1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            tmp1[x - 1, y - 1] = (tmp[x + 1, y] + tmp[x, y - 1] + tmp[x - 1, y] + tmp[x, y + 1] - tmp[x, y] * 4)
    tmp2 = extend(tmp1)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            if(tmp2[x, y] >= threshold):
                for p in range(-1, 2):
                    for q in range(-1, 2):
                        if(tmp2[x + p, y + q] <= -threshold):
                            ans[x - 1, y - 1] = 0
    return ans
```

Part2: Laplace type2(threshold = 15)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果gradient大於threshold且周遭有任一gradient小於-threshold，把pixel值設定成0。g即為乘完矩陣的值。

$$\frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
def laplace2( img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    tmp1 = np.zeros(img.shape, np.int)

    for x in range (1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            tmp1[x - 1, y - 1] = (tmp[x, y + 1] + tmp[x - 1, y + 1] + tmp[x - 1, y] + tmp[x - 1, y - 1] + tmp[x, y - 1
] + tmp[x + 1, y - 1] + tmp[x + 1, y] + tmp[x + 1, y + 1] - tmp[x, y] * 8)

    tmp2 = extend(tmp1)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            if(tmp2[x, y] >= 3 * threshold):
                for p in range(-1, 2):
                    for q in range(-1, 2):
                        if(tmp2[x + p, y + q] <= -3 * threshold):
                            ans[x - 1, y - 1] = 0
    return ans
```

Part3: minimum-variance Laplacian(threshold = 20)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果gradient大於threshold且周遭有任一gradient小於-threshold，把pixel值設定成0。g即為乘完矩陣的值。

$$\frac{1}{3}\begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix}$$

```
def min_var_Laplacian( img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    tmp1 = np.zeros(img.shape, np.int)

    for x in range (1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            tmp1[x - 1, y - 1] = (-tmp[x, y + 1] + 2 * tmp[x - 1, y + 1] - tmp[x - 1, y] + 2 * tmp[x - 1, y - 1] - tmp
[x, y - 1] + 2 * tmp[x + 1, y - 1] - tmp[x + 1, y] + 2 * tmp[x + 1, y + 1] - tmp[x, y] * 4)

    tmp2 = extend(tmp1)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            if(tmp2[x, y] >= 3 * threshold):
                for p in range(-1, 2):
                    for q in range(-1, 2):
                        if(tmp2[x + p, y + q] <= -3 * threshold):
                            ans[x - 1, y - 1] = 0
    return ans
```

Part4: Laplacian of Gaussian(threshold = 3000)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果

gradient大於threshold且周遭有任一gradient小於-threshold，把pixel值設定成0。g即為乘完矩陣的值。

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

```
def log(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(extend(extend(extend(extend(img)))))
    tmp1 = np.zeros(img.shape, np.int)

    k =  ([ [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]])

    for x in range(5, img.shape[0] + 5):
        for y in range(5, img.shape[1] + 5):
            g = 0
            for p in range(-5, 6):
                for q in range(-5, 6):
                    g += (tmp[x + p, y + q] * k[p + 5][q + 5])
            tmp1[x - 5, y - 5] = g

    tmp1 = extend(tmp1)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            if(tmp1[x, y] >= threshold):
                for p in range(-1, 2):
                    for q in range(-1, 2):
                        if(tmp1[x + p, y + q] <= -threshold):
                            ans[x - 1, y - 1] = 0
    return ans
```

Part5: Difference of Gaussian(threshold =)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果gradient大於threshold且周遭有任一gradient小於-threshold，把pixel值設定成0。g即為乘完矩陣的值。

```
-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
```

```python
def gol(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(extend(extend(extend(extend(img)))))
    tmp1 = np.zeros(img.shape, np.int)

    k = ([[-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]])

    for x in range(5, img.shape[0] + 5):
        for y in range(5, img.shape[1] + 5):
            g = 0
            for p in range(-5, 6):
                for q in range(-5, 6):
                    g += (tmp[x + p, y + q] * k[p + 5][q + 5])
            tmp1[x - 5, y - 5] = g

    tmp1 = extend(tmp1)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            if(tmp1[x, y] >= threshold):
                for p in range(-1, 2):
                    for q in range(-1, 2):
                        if(tmp1[x + p, y + q] <= -threshold):
                            ans[x - 1, y - 1] = 0
    return ans
```