R09922190 王祥任

CV Homework 9

Abstract

本次作業中，我們將實作7種不同的edge detection 方式: (1) Robert's Operator, (2) Prewitt's Edge Detector (3) Sobel's Edge Detector, (4) Frei and Chen's Gradient Operator (5) Kirsch's Compass Operator (6) Robinson's Compass Operator (7) Nevatia-Babu 5x5 Operator。以上操作除輸入、輸出以外，不得直接套用現成套件。

Implementation

- Programming Language: Python3
- Python Package: opencv-python, matplotlib, numpy, copy
- Execution: python3 hw9-main.py

在執行時，請務必將 lena.bmp 放置在和 hw9-main.py 相同的檔案目錄下，並確認檔名相同。opencv-python用於讀取和寫出圖片，matplotlib僅用於runtime時即時顯示和儲存圖片，須配合jupyter使用，在此不贅述，而最後的copy和numpy則用於純計算上。

Robert, Prewitt, Sobel and Frei and Chen

```python
def Roberts(lena, thres = 12):
    h, w = lena.shape
    pad_lena = padding(lena, 1)
    after = np.zeros((h, w))
    for r in range(1, h + 1):
        for c in range(1, w + 1):
            r1 = -pad_lena[r][c] + pad_lena[r+1][c+1]
            r2 = pad_lena[r+1][c] - pad_lena[r][c+1]
            if (np.sqrt(r1**2 + r2**2) >= thres):
                after[r - 1][c - 1] = 0
            else:
                after[r - 1][c - 1] = 255
    return after
```

```python
def Prewitt(lena, thres = 24):
    h, w = lena.shape
    pad_lena = padding(lena, 1)
    after = np.zeros((h, w))
    for r in range(1, h + 1):
        for c in range(1, w + 1):
            p1 = -(pad_lena[r-1][c-1] + pad_lena[r-1][c  ] + pad_lena[r-1][c+1]) + \
                  (pad_lena[r+1][c-1] + pad_lena[r+1][c  ] + pad_lena[r+1][c+1])
            p2 = -(pad_lena[r-1][c-1] + pad_lena[r  ][c-1] + pad_lena[r+1][c-1]) + \
                  (pad_lena[r-1][c+1] + pad_lena[r  ][c+1] + pad_lena[r+1][c+1])
            if (np.sqrt(p1**2 + p2**2) >= thres):
                after[r - 1][c - 1] = 0
            else:
                after[r - 1][c - 1] = 255
    return after
```

```python
def Sobel(lena, thres = 38):
    h, w = lena.shape
    pad_lena = padding(lena, 1)
    after = np.zeros((h, w))
    for r in range(1, h + 1):
        for c in range(1, w + 1):
            p1 = -(pad_lena[r-1][c-1] + 2*pad_lena[r-1][c  ] + pad_lena[r-1][c+1]) + \
                  (pad_lena[r+1][c-1] + 2*pad_lena[r+1][c  ] + pad_lena[r+1][c+1])
            p2 = -(pad_lena[r-1][c-1] + 2*pad_lena[r  ][c-1] + pad_lena[r+1][c-1]) + \
                  (pad_lena[r-1][c+1] + 2*pad_lena[r  ][c+1] + pad_lena[r+1][c+1])
            if (np.sqrt(p1**2 + p2**2) >= thres):
                after[r - 1][c - 1] = 0
            else:
                after[r - 1][c - 1] = 255
    return after
```

```python
def FreiAndChen(lena, thres = 30):
    h, w = lena.shape
    pad_lena = padding(lena, 1)
    after = np.zeros((h, w))
    for r in range(1, h + 1):
        for c in range(1, w + 1):
            cof = np.sqrt(2)
            p1 = -(pad_lena[r-1][c-1] + cof*pad_lena[r-1][c  ] + pad_lena[r-1][c+1]) + \
                  (pad_lena[r+1][c-1] + cof*pad_lena[r+1][c  ] + pad_lena[r+1][c+1])
            p2 = -(pad_lena[r-1][c-1] + cof*pad_lena[r  ][c-1] + pad_lena[r+1][c-1]) + \
                  (pad_lena[r-1][c+1] + cof*pad_lena[r  ][c+1] + pad_lena[r+1][c+1])
            if (np.sqrt(p1**2 + p2**2) >= thres):
                after[r - 1][c - 1] = 0
            else:
                after[r - 1][c - 1] = 255
    return after
```

以上四種作法基本上基於一個相同的邏輯。首先將圖片Reflect padding 1 pixel (我實作padding的方式是先在中間填充完原本的圖片後，再依序在邊角填充補足的像素)。而這四種做法的差別在於，它們套用的是不同的mask組合。故我們在遍歷所有像素的時候，根據想要實作的方式更改mask(即係數組合)即可。再根據計算出來的數值和threshold做判定，若不小於threshold，則該像素為邊界像素(黑色)，否則為背景(白色)。成果貼於最後的圖組中。

Kirsch, Robinson, Nevatia-Babu 5x5

```python
def Kirsch(lena, thres = 135):
    h, w = lena.shape
    pad_lena = padding(lena, 1)
    after = np.zeros((h, w))
    ofst = [
        (-1, -1), (-1, 0), (-1, 1),
        ( 0, -1),          ( 0, 1),
        (+1, -1), (+1, 0), (+1, 1),
    ]
    cofs = [
        [-3, -3,  5, -3,  5, -3, -3,  5],
        [-3,  5,  5, -3,  5, -3, -3, -3],
        [ 5,  5,  5, -3, -3, -3, -3, -3],
        [ 5,  5, -3,  5, -3, -3, -3, -3],
        [ 5, -3, -3,  5, -3,  5, -3, -3],
        [-3, -3, -3,  5, -3,  5,  5, -3],
        [-3, -3, -3, -3, -3,  5,  5,  5],
        [-3, -3, -3, -3,  5, -3,  5,  5],
    ]
    for r in range(1, h + 1):
        for c in range(1, w + 1):
            k = []

            for cof in cofs:
                tmp = 0
                for i in range(8):
                    ro, co = ofst[i]
                    tmp += cof[i] * pad_lena[r+ro][c+co]
                k.append(tmp)

            if (np.max(k) >= thres):
                after[r - 1][c - 1] = 0
            else:
                after[r - 1][c - 1] = 255
    return after
```

```python
def Robinson(lena, thres = 43):
    h, w = lena.shape
    pad_lena = padding(lena, 1)
    after = np.zeros((h, w))
    ofst = [
        (-1, -1), (-1, 0), (-1, 1),
        ( 0, -1),          ( 0, 1),
        (+1, -1), (+1, 0), (+1, 1),
    ]
    cofs = [
        [-1,  0,  1, -2,  2, -1,  0,  1],
        [ 0,  1,  2, -1,  1, -2, -1,  0],
        [ 1,  2,  1,  0,  0, -1, -2, -1],
        [ 2,  1,  0,  1, -1,  0, -1, -2],
        [ 1,  0, -1,  2, -2,  1,  0, -1],
        [ 0, -1, -2,  1, -1,  2,  1,  0],
        [-1, -2, -1,  0,  0,  1,  2,  1],
        [-2, -1,  0, -1,  1,  0,  1,  2],
    ]
    for r in range(1, h + 1):
        for c in range(1, w + 1):
            k = []

            for cof in cofs:
                tmp = 0
                for i in range(8):
                    ro, co = ofst[i]
                    tmp += cof[i] * pad_lena[r+ro][c+co]
                k.append(tmp)

            if (np.max(k) >= thres):
                after[r - 1][c - 1] = 0
            else:
                after[r - 1][c - 1] = 255
    return after
```

```python
def NevatiaBabu(lena, thres = 12500):
    h, w = lena.shape
    pad_lena = padding(lena, 2)
    after = np.zeros((h, w))
    ofst = [
        (-2, -2), (-2, -1), (-1, 0), (-2, 1), (-2, 2),
        (-1, -2), (-1, -1), (-1, 0), (-1, 1), (-1, 2),
        ( 0, -2), ( 0, -1), ( 0, 0), ( 0, 1), ( 0, 2),
        (+1, -2), (+1, -1), (+1, 0), (+1, 1), (+1, 2),
        (+2, -2), (+2, -1), (+1, 0), (+2, 1), (+2, 2)
    ]
    cofs = [
        [ 100,  100, 100, 100,  100,   100,  100, 100, 100,  100,     0,    0, 0,    0,    0, -100, -100, -100, -100, -100,  -100, -100, -100, -100, -100],
        [ 100,  100, 100, 100,  100,   100,  100, 100,  78,  -32,   100,   92, 0,  -92, -100,   32,  -78, -100, -100, -100,  -100, -100, -100, -100, -100],
        [ 100,  100, 100,  32, -100,   100,  100,  92, -78, -100,   100,  100, 0, -100, -100,  100,   78,  -92, -100, -100,   100,  -32, -100, -100, -100],
        [-100, -100,   0, 100,  100,  -100, -100,   0, 100,  100,  -100, -100, 0,  100,  100, -100, -100,    0,  100,  100,  -100, -100,    0,  100,  100],
        [-100,   32, 100, 100,  100,  -100,  -78,  92, 100,  100,  -100, -100, 0,  100,  100, -100, -100,  -92,   78,  100,  -100, -100, -100,  -32, -100],
        [ 100,  100, 100, 100,  100,   -32,   78, 100, 100,  100,  -100,  -92, 0,   92,  100, -100, -100, -100,  -78,   32,  -100, -100, -100, -100, -100],
    ]
    for r in range(2, h + 2):
        for c in range(2, w + 2):
            k = []

            for cof in cofs:
                tmp = 0
                for i in range(25):
                    ro, co = ofst[i]
                    tmp += cof[i] * pad_lena[r+ro][c+co]
                k.append(tmp)

            if (np.max(k) >= thres):
                after[r - 2][c - 2] = 0
            else:
                after[r - 2][c - 2] = 255
    return after
```

以上三種方法也是基於同一種類型的做法。Kirsch和Robinson必須將圖片先reflect padding 1 pixel，而Nevatia-Babu 5x5則必須將圖片reflect padding 2 pixels因為mask是5x5。我們首先將要遍歷的鄰近點的偏移量全部放入一個固定的list方便遍歷，然後同樣將所有的mask即對應的係數表寫為一組二維list，同樣為方便遍歷。然後，遍歷所有的像素，並套用不同的mask對所有鄰近點相乘加總，最後取套用mask算出之最大值和threshold做比較，同樣若不小於threshold，則該像素為邊界像素(黑色)，否則為背景(白色)。成果貼於最後的圖組中。

Results

圖標最後的數字為該方法所對應的threshold。因排版因素，圖片尺寸略有縮放。

Robert's Operator: 12

Prewitt's Edge Detector: 24

Sobel's Edge Detector: 38

Frei and Chen's Gradient Operator: 30

Kirsch's Compass Operator: 135

Robinson's Compass Operator: 43

Nevatia-Babu 5x5 Operator: 12500