20210510攝影測量報告

組員:

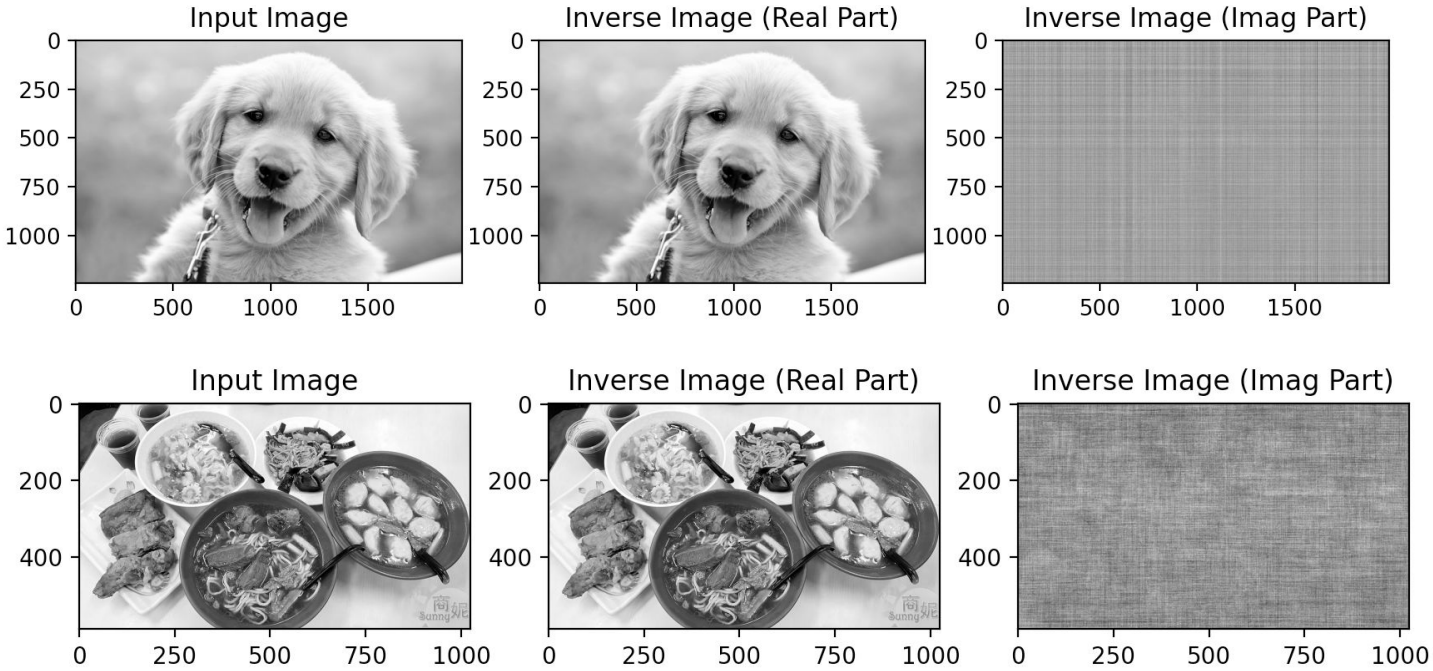
B06501015 土木四 周遠同

B06501103 土木四 何文杰

B06501075 土木四 潘佳旻

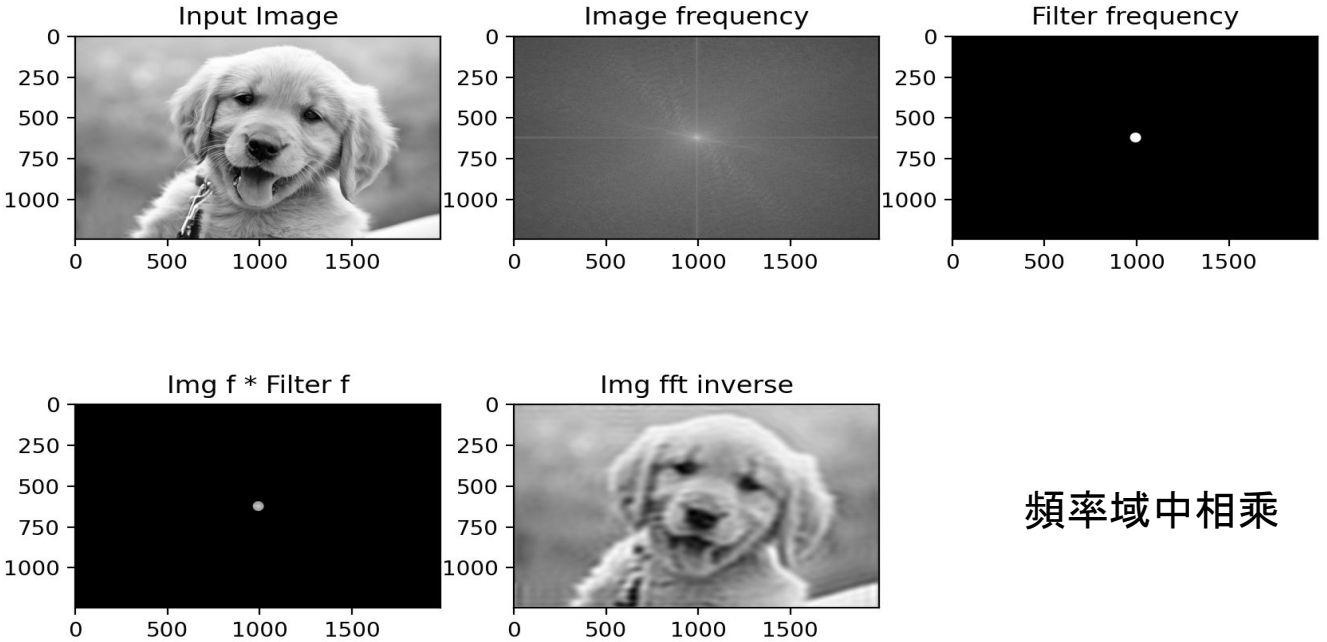
一、ifft2後，實部與虛部的比較

- 在頻率域相乘後，轉回空間域需要用ifft2，會產生實數、虛數



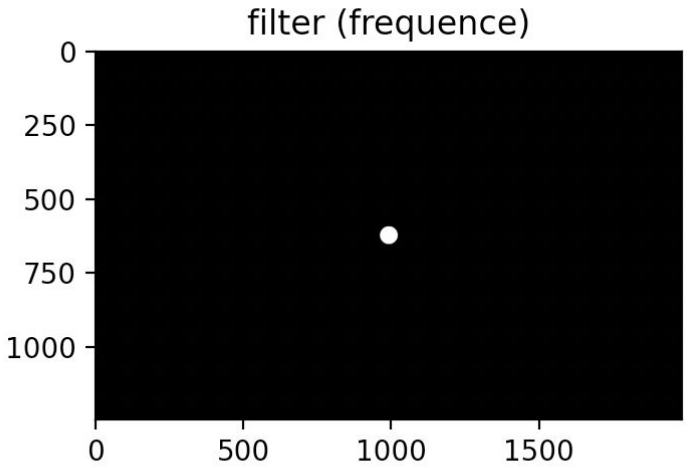
二、濾波函數是在空間域中或頻率域中設計？

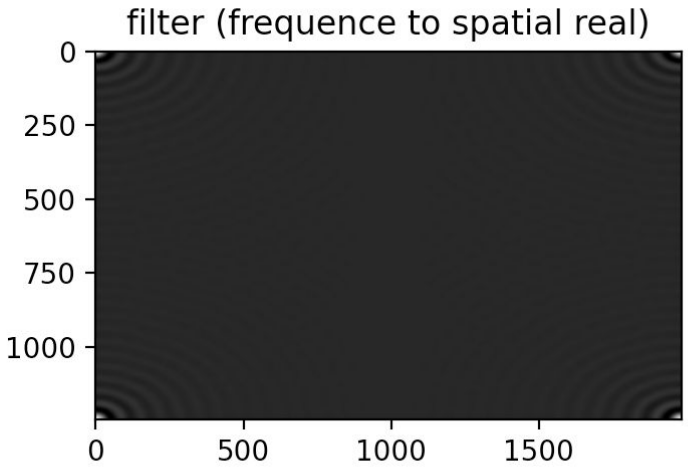
- (1) 在頻率域中設計：



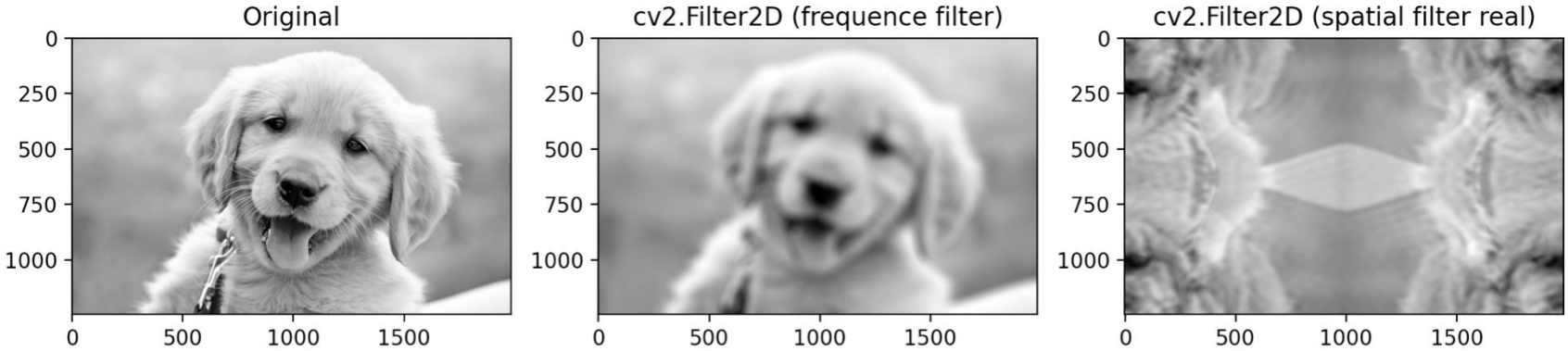
頻率域中相乘

- 在空間域做convolution時，不確定要將filter轉換到空間域還是維持在頻率域 → 都試試看



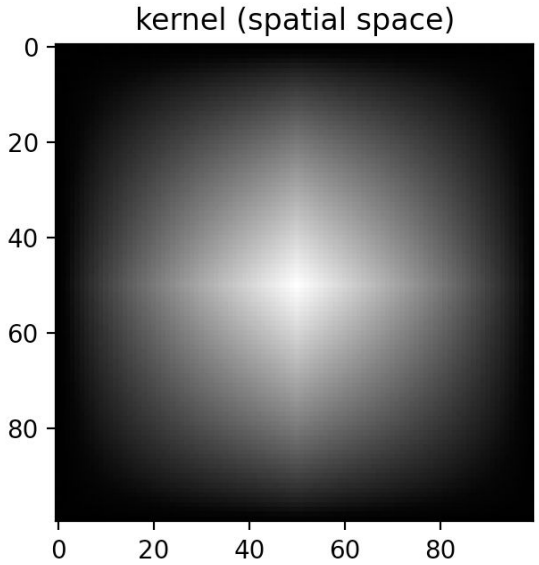


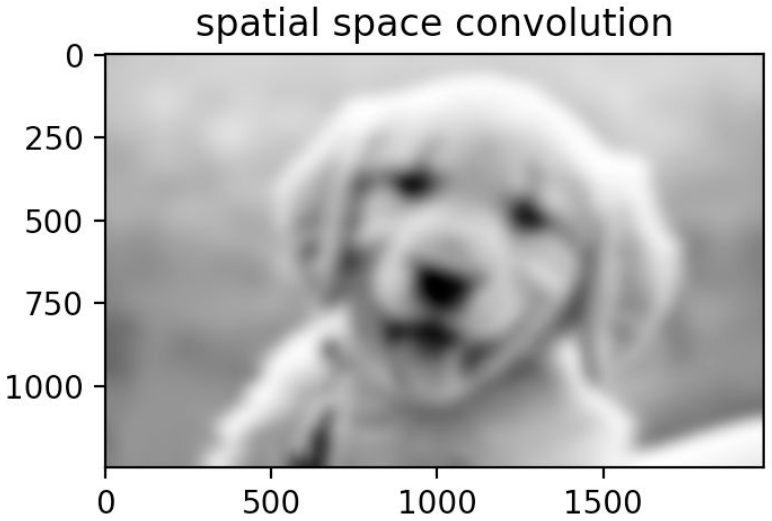
- 頻率域、空間域的filter都convolution試試看



- 頻率域中的filter，convolution後還算正常，但與頻率域中相乘的仍有差別
- 轉到空間域中的filter，convolution後變得很奇怪

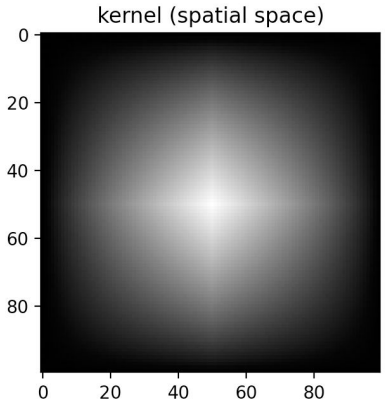
- (2) 在空間域中設計

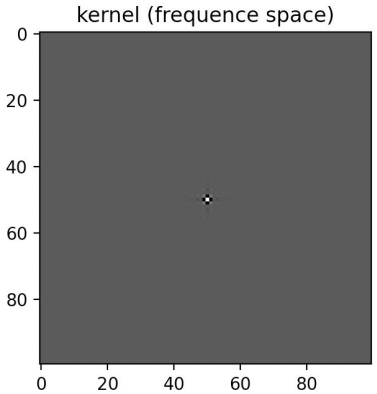


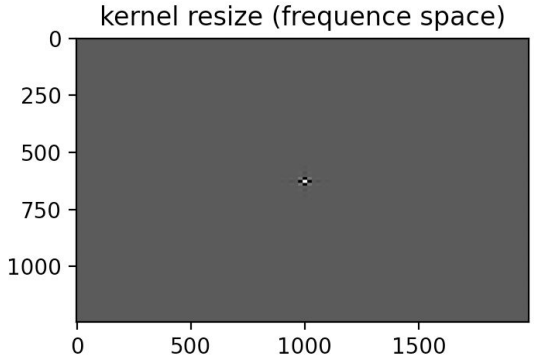


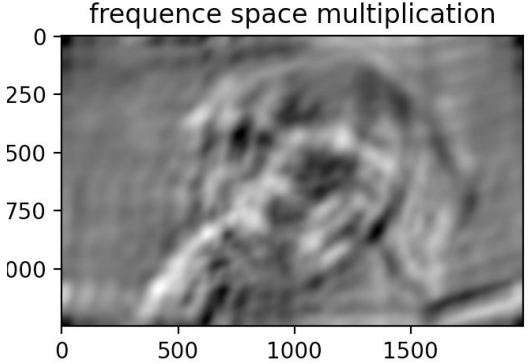
空間域做convolution

- 將kernal 轉換到頻率域
- 在頻率域中相乘，再轉回空間域

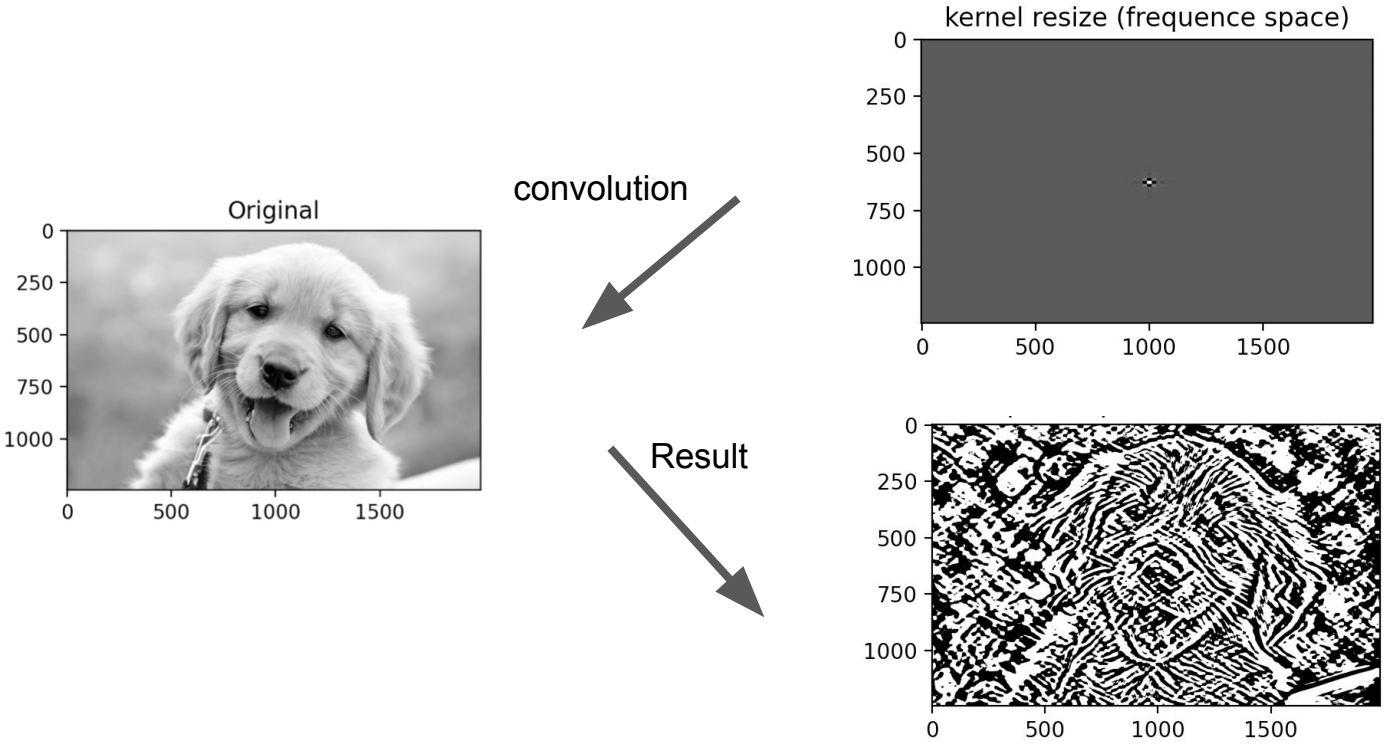








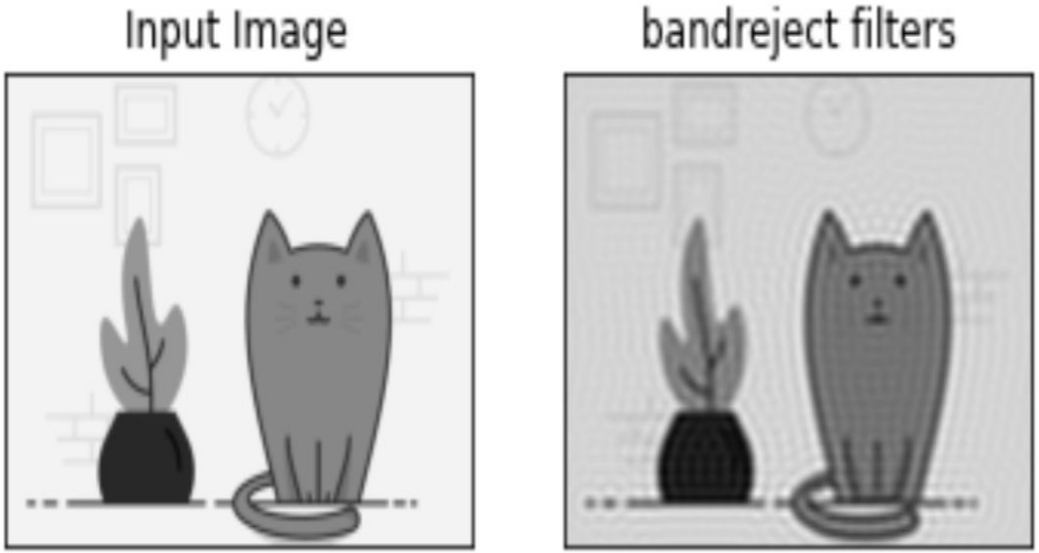
- 亂試試看:
- 將轉到頻率域的kernel，與image做convolution



三、比較不同濾波函數的過濾效果與差異

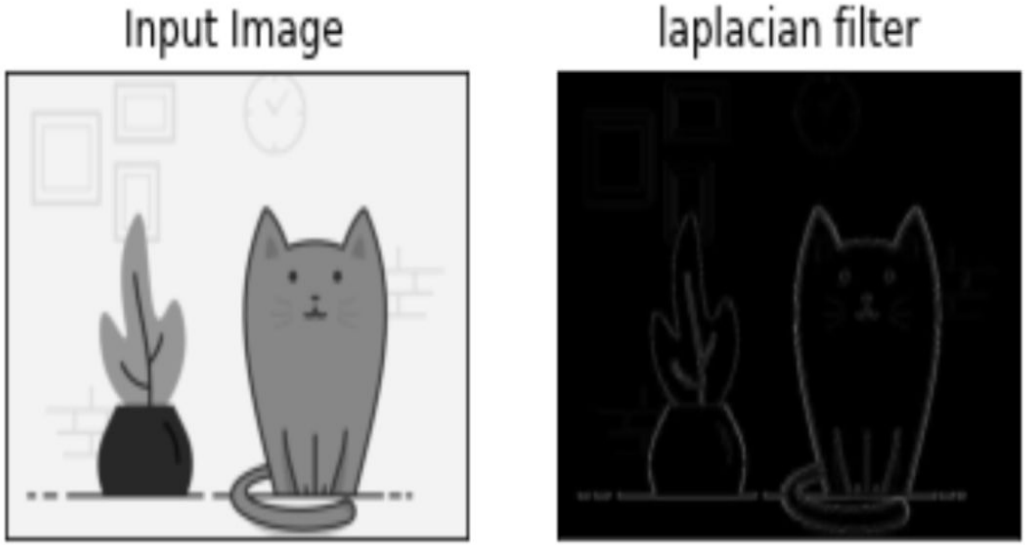
- 濾波器

濾波器



濾波器

濾波器



濾波器混合

LF+GHPF

BF+GHPF

混合濾波器

LF+GLPF

BF+GLPF

混合濾波器

LF+BF

LF+BF+GLPF

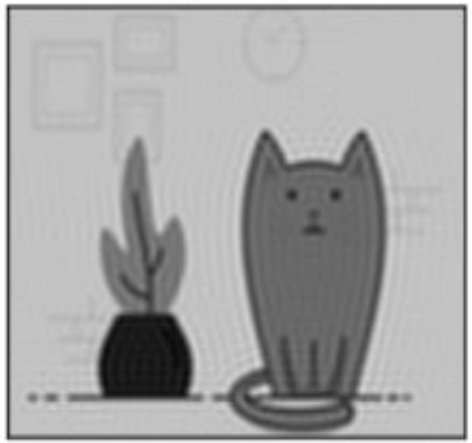
HPF 濾波半徑不同

HPF r=40

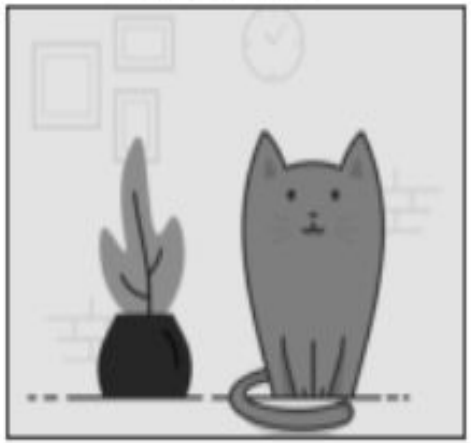
HPF r=150

LPF 濾波半徑不同

LPF r=40



LPF r=150



四、比較numpy與OpenCV的計算、效能差異

1. Developer: Travis Oliphant
2. Functional: A large collection of high-level mathematical functions to operate large and multi-dimensional arrays.
3. Program: Python

1. Developer: Intel
2. Functional: real-time ray tracing and 3D displaying walls.
3. Program: C/C++

OpenCV2 optimization

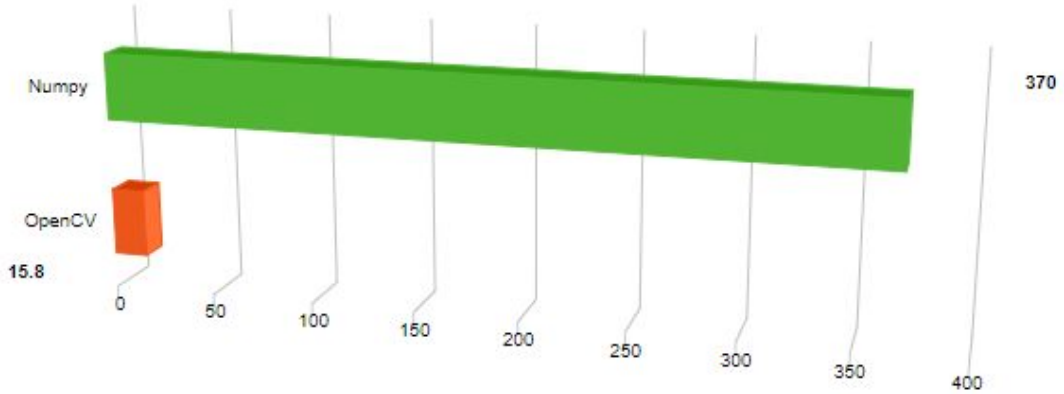
- **cv2.useoptimized()**
  - **Recalled this function to activate optimized version of CV2.**
- **cv2.cv2.setUseOptimized(False)**
  - **Recalled this function to de-activate optimized version of CV2.**
- **Optimized version able to shorten the duration by half.**

```
In [30]: runfile('D:/Civil Engineering/Senior/
Photogrammetry/cv2 with optimization performance.py'
wdir='D:/Civil Engineering/Senior/Photogrammetry')
0.0195031

In [31]: runfile('D:/Civil Engineering/Senior/
Photogrammetry/cv2 without optimization (median).py'
wdir='D:/Civil Engineering/Senior/Photogrammetry')
0.0389865
```

Time Performance (Numpy vs OpenCV)

- **Same condition: images, task, best of three results.**
- **cv2.countNonZero()**
  - **Recalled this to calculate non-zero values by openCV.**
- **np.count_nonzero()**
  - **Recalled this to calculate non-zero values by numpy.**
- **OpenCV 23.42x faster than Numpy.**

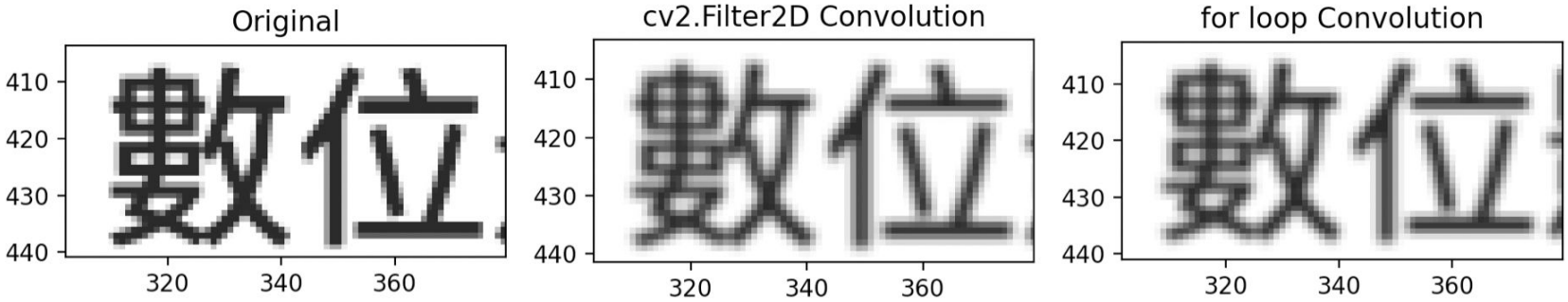


Conclusion & Limitations

- **OpenCV functions over Numpy for same calculations. However, Numpy are faster on views instead of copies.**
- **Both functions are designed for vector optimization, thus try to vectorize the code to the maximum extent.**
- **Copying an array significantly increase the time of calculations. Therefore, try to use view over copy.**
- **OpenCV has faster rate of computation by 20x faster.**

五、cv2.filter2D vs. 自己寫的convolution

```
kernal = np.array((
        [0.0625, 0.125, 0.0625],
        [0.125, 0.25, 0.125],
        [0.0625, 0.125, 0.0625]))
convolution1 = cv2.filter2D(img,-1,kernal)
convolution2 = convolution_func.my_convolution(img, kernal)
```



- **cv2.filter2D()**
  - Actually it is correlation, not convolution
  - If the kernel is too big (kernal > 11x11), use DTF algorithm

$$\texttt{dst}(x,y) = \sum_{\substack{0 \le x' < \texttt{kernel.cols}, \\ 0 \le y' < \texttt{kernel.rows}}} \texttt{kernel}(x',y') * \texttt{src}(x + x' - \texttt{anchor.x}, y + y' - \texttt{anchor.y})$$

- **my for loop convolution**

```python
def my_convolution(image, kernal):
    m, n = image.shape[0], image.shape[1]
    a, b = kernal.shape[0], kernal.shape[1]
    h, w = m-a+1, n-b+1
    new = np.zeros((h,w))

    # for every point in the conv matrix
    for i in range(h):
        for j in range(w):
            # calculate the aggregated multiplication value in the kernel
            new[i, j] = np.multiply(image[i:i+a, j:j+b], kernal).sum()

    return new
```

- cv2.filter2D, My Convolution效能差異、比較:

```
(base) zhouyuantongdeMacBook-Pro:Lab2 tony$ python3 filter2D_test.py
Total Round: 10
cv2.filter2D time  : 0.000661
my convolution time: 1.867547
-----------------------------------------
(base) zhouyuantongdeMacBook-Pro:Lab2 tony$ python3 filter2D_test.py
Total Round: 100
cv2.filter2D time  : 0.005684
my convolution time: 21.286446
-----------------------------------------
(base) zhouyuantongdeMacBook-Pro:Lab2 tony$ python3 filter2D_test.py
Total Round: 1000
cv2.filter2D time  : 0.050415
my convolution time: 224.596234
-----------------------------------------
```

- **cv2.filter2D, FFT 效能差異、比較:**

```
(base) zhouyuantongdeMacBook-Pro:Lab2 tony$ python3 TimeComparison.py
Total Round: 10
cv2.filter2D time       : 0.017779
Frequence space multiply: 0.033480
-----------------------------------------
(base) zhouyuantongdeMacBook-Pro:Lab2 tony$ python3 TimeComparison.py
Total Round: 100
cv2.filter2D time       : 0.165138
Frequence space multiply: 0.316796
-----------------------------------------
(base) zhouyuantongdeMacBook-Pro:Lab2 tony$ python3 TimeComparison.py
Total Round: 1000
cv2.filter2D time       : 1.648850
Frequence space multiply: 3.681419
-----------------------------------------
```

cv2.filter2D	>	FFT	>	my convolution
4000	>	2000	>	1

- Why OpenCV function can be so fast !?
- The Convolution Theorem, Seperable Convolution

1. The Convolution Theorem
   - 偷偷轉到頻率域做相乘, 再轉回來
2. Seperable Convolution
   - 將kernal矩陣拆成兩個向量相乘
   - img ＊ (v ＊ h) ＝ (img ＊ v) ＊ h
   - 原先:O(M*N*P*Q)
   - 變成:O(M*N*(P+Q))

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{1}{256} \cdot \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

5x5 Separable Gaussian Kernel

參考資料

https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#filter2d

https://stackoverflow.com/questions/31336186/opencv-computational-efficiency-of-filter2d-function

https://makerpro.cc/2019/06/the-convolution-of-opencv/

https://www.cnblogs.com/lfri/p/10599420.html

https://dsp.stackexchange.com/questions/8471/where-does-convolution-fit-in-dft

https://zh.wikipedia.org/wiki/快速傅里叶变换

分工

- 依照上述投影片五個題目來分:
- 一、周遠同
- 二、周遠同
- 三、潘佳旻
- 四、何文杰
- 五、周遠同

Thanks for listening