cv::Mat 物件的設計巧思

姓名：吳宇昕
學號：B10831020

前言

- cv::Mat 使用設計很不直觀
  - 無法直接用index operator[]索引像素值
  - 三通道平面影像不以三維陣列(height, width, channel)形式儲存，而是以 長度(height* width * channel)的一維陣列儲存

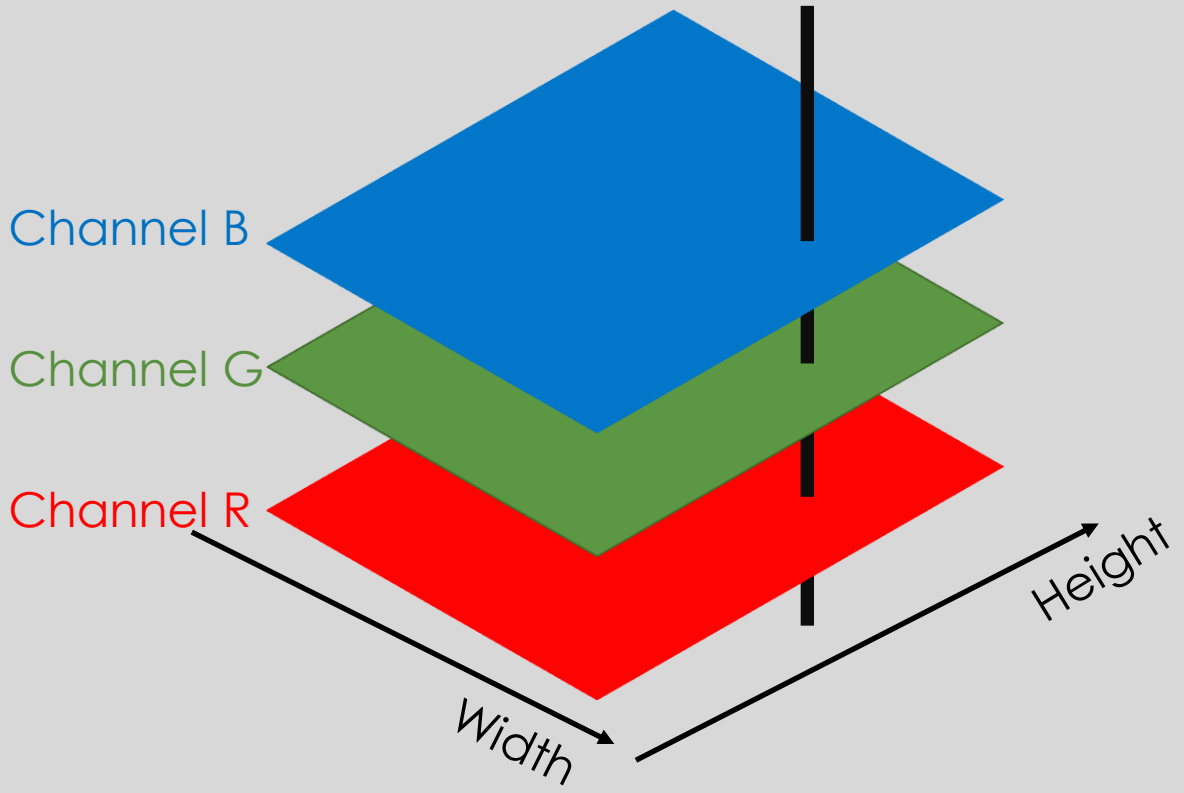
- 嘗試寫出功能類似的C++物件，模擬cv::Mat的功能，比較其效能並了解其設計思維
- 自己寫的Mat原始碼: GuaiGuaiOnMyComputer/ColorImageProcessingPraticeHomework (github.com)

儲存像素陣列的形狀

Pixel BGR value at (x, y) = (x0, y0)

一般想像儲存三通道2D影像，應該是以三維陣列形式

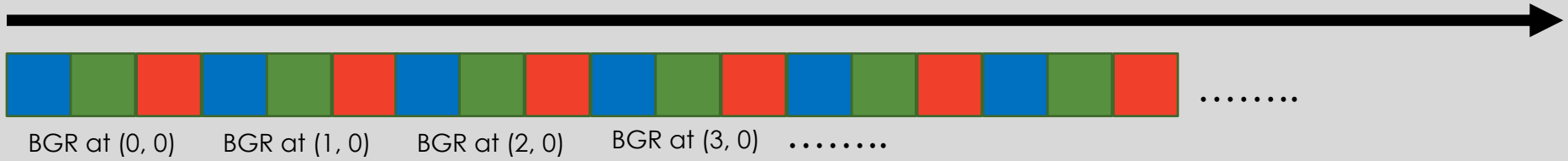
- 把通道放在三維陣列的最後一個維度
- 前兩個維度分別是長與寬
- 索引位於特定位置(x, y)的像素顏色值，就給三個index得到一個uint8



cv::Mat儲存影像的陣列形狀

- 但是Opencv::Mat並不是這樣儲存影像的像素數值
- Opencv::Mat把影像拉平，一列一列頭尾相接，把三通道影像儲存成一維陣列
- 每三個元素，代表一個像素的BGR值

flatten image and store it as a 1D array of length (3 * width * height)



索引像素值

假設想要把某個cv::Mat位於(x, y) = (100, 100)的像素藍色值改成23：

```
 5    int main()
 6    {
 7        cv::Mat myMat = cv::Mat::zeros(1440, 1920, CV_8UC3);
 8        myMat[100][100][2] = 23;
 9        return 0;
10    }
```

用index operator []索引cv::Mat物件行不通

Opencv::Mat索引像素值

```
19    int main()
20    {
21        cv::Mat myMat = cv::Mat::zeros(1440, 1920, CV_8UC3);
22        myMat.ptr(100)[100 + 2] = 23;
23        return 0;
24    }
```

必須使用.ptr函式，得到指向第100列起始位置的pointer，用index operator偏移100個uchar的距離找到該像素的起始位置，再偏移2個uchar的距離找到該像素的藍色通道值

自己土炮做出來的Mat物件

- 試著自己寫出一個功能類似cv::Mat的物件，名為my::Mat
- 自定義index operator，使土炮版的Mat物件可以用index operator索引資料
- 測試兩種不同版本Mat效能，比較兩者執行以下任務所需時間
  - 創建Mat
  - 將兩個Mat相加
  - 計算與另一個陣列的捲積，以5x5 Laplacian filter為例

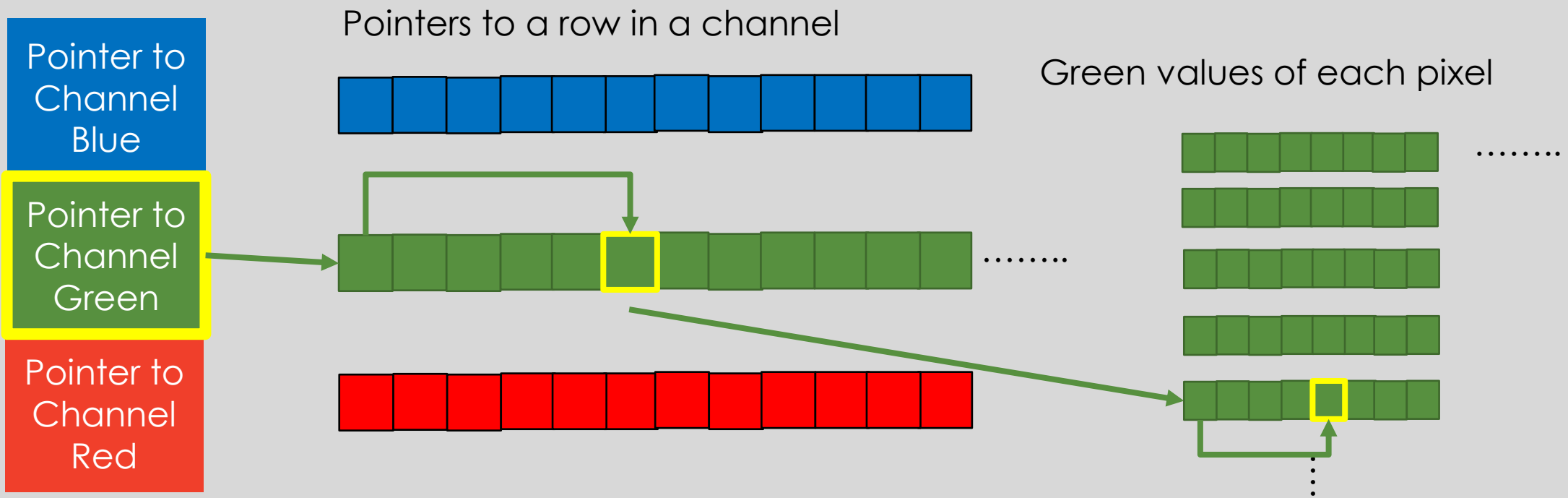
```cpp
#include "../slowmat.hpp"

int main()
{
    // create an instance of my::SlowMat having size 1440x1920
    // and 3 channels
    my::SlowMat myMat = my::SlowMat(1440, 1920, 3);
    myMat[2][1400][1900] = 23; // index its data using custom [] operators
}
```

```cpp
    // custom indexing operator
    uint8_t** operator[](int channel)
    {
        return m_data[channel];
    }
```

土炮版Mat物件像素儲存方式

索引位於第6列，第5行的像素綠色通道值
myMat[1, 5, 4]



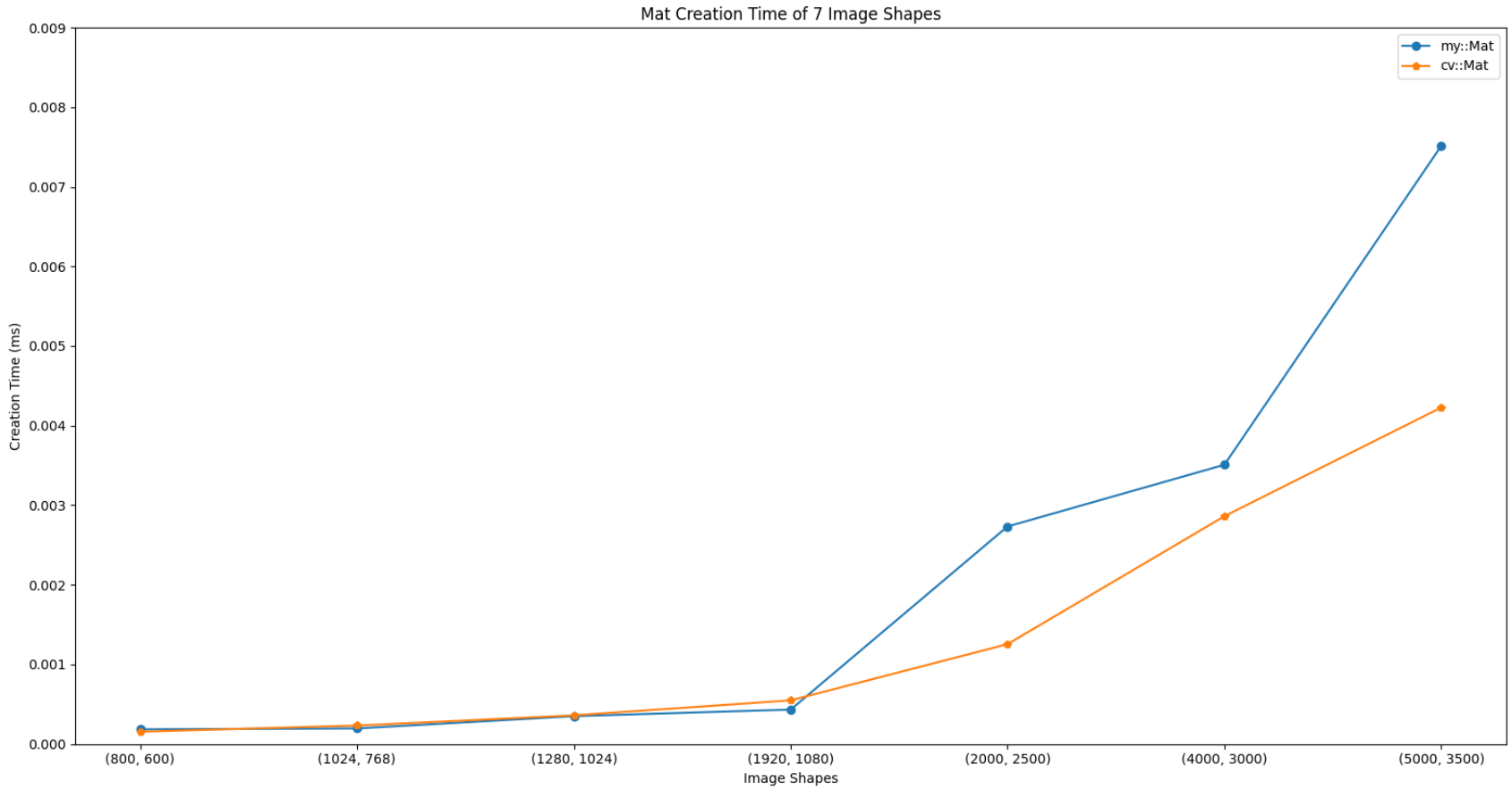
my::Mat Constructor

My::Mat以三維陣列形式儲存影像像素值，以三通道影像為例：

1. 產生一個含三個uint8_t**的一維陣列，代表影像通道
2. 產生三個含_rows個uint8_t*的一維陣列，代表影像各個通道的一列
3. 產生_rows個含_cols個uint8_t的一維陣列，代表各個通道各個列的元素

```
50      // regular constructor, heap-allocates memory to store pixel values
51      // use of new operator can easily lead to memory leak
52      // heap allocation is bad for performance, calling new operator for many times is discouraged
53      explicit SlowMat(size_t _rows, size_t _cols, int _channels) : rows{_rows}, cols{_cols}, channels{_channels}
54      {
55          m_data = new uint8_t**[_channels]; // allocate an array of uint8_t** channel pointers for an image
56          for(size_t channel = 0; channel < _channels; channel ++){
57              m_data[channel] = new uint8_t*[_rows]; // allocate an array of uint8_t* row pointers for each channel
58              for(size_t row = 0; row < _rows; row ++){
59                  m_data[channel][row] = new uint8_t[_cols]; // allocate an array of uint8_t for each row
60              }
61          }
62      }
```

效能比較-創建Mat

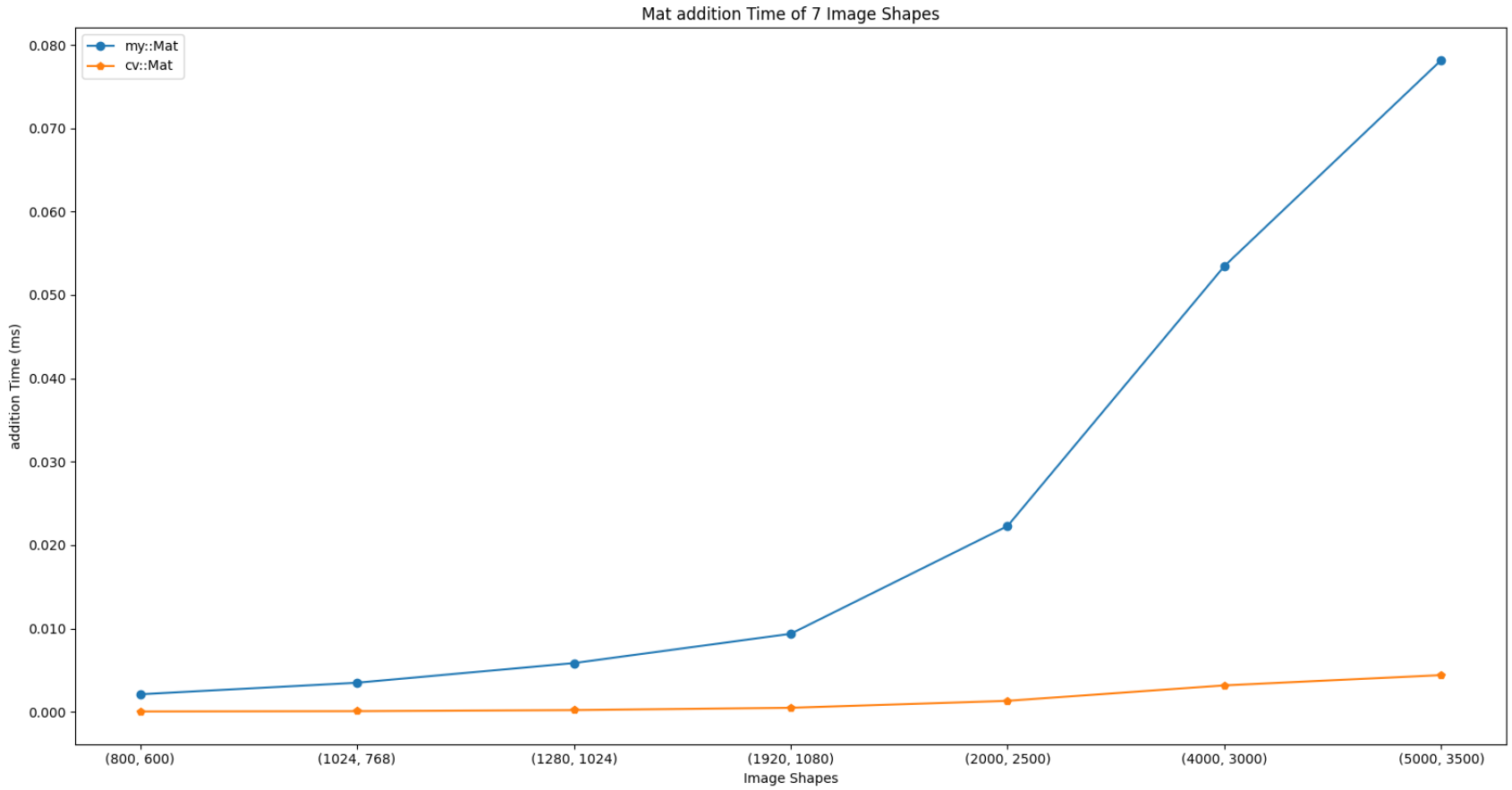

Mat Creation Time of 7 Image Shapes
my::Mat
cv::Mat
Creation Time (ms)
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
(800, 600)
(1024, 768)
(1280, 1024)
(1920, 1080)
(2000, 2500)
(4000, 3000)
(5000, 3500)
Image Shapes

my::Mat::Add

用三層for迴圈走訪三維陣列
取兩個my::SlowMat物件，分別是*this與other，點對點相加
量測索引三維陣列耗費的時間

```
116         void Add(const SlowMat& other)
117         {
118             for (int channel = 0; channel < this->channels; channel++){
119                 for (int row = 0; row < this->rows; row ++){
120                     for (int col = 0; col < this->cols; col++){
121                         this->m_data[channel][row][col] += other.m_data[channel][row][col];
122                     }
123                 }
124             }
125         }
```

效能比較-Mat相加


Mat addition Time of 7 Image Shapes
my::Mat
cv::Mat
addition Time (ms)
0.080
0.070
0.060
0.050
0.040
0.030
0.020
0.010
0.000
(800, 600)
(1024, 768)
(1280, 1024)
(1920, 1080)
(2000, 2500)
(4000, 3000)
(5000, 3500)
Image Shapes

my::Mat::Convolution

用五層for迴圈走訪三維陣列，以及二維_kernal，計算my::SlowMat與5x5 Laplacian filter捲積
暫時不考慮filter蓋到影像邊界外，只考慮filter完全蓋在影像內的情況

```
94          void Convolution(const int32_t _kernal[])
95          {
96              constexpr int KER_SIZE = 5;
97              SlowMat original = SlowMat(*this);
98              const int halfKerLen = (KER_SIZE - 1) / 2;
99              const int convHozStart = (KER_SIZE - 1) / 2;
100             const int convVrtStart = (KER_SIZE - 1) / 2;
101             const int convHozEnd = cols - (KER_SIZE - 1) / 2 - 1;
102             const int convVrtEnd = rows - (KER_SIZE - 1) / 2 - 1;
103             for (int channel = 0; channel < channels; channel ++){
104                 for (int row = convVrtStart; row < convVrtEnd; row ++){
105                     for (int col = convHozStart; col < convHozEnd; col ++){
106                         for (int i = 0; i < KER_SIZE; i ++){
107                             for (int j = 0; j < KER_SIZE; j++){
108                                 m_data[channel][row][col] += original.m_data[channel][row - halfKerLen + i][col - halfKerLen + j] * _kernal[j + KER_SIZE * i];
109                             }
110                         }
111                     }
112                 }
113             }
114         }
```

效能比較-與Laplacian Filter捲積