原圖：

640 * 640

1280 * 960

4032 * 3024

主函式：

```python
if __name__ == '__main__':
    image = cv2.imread('img/Block.jpeg')  # 讀取要處理的檔案
    sign = cv2.imread('img/sign.jpeg')  # 讀取簽名的檔案
    print("1:resize_NN_interpolation\n" +
          "2:resize_Bilinear_interpolation\n" +
          "3:filter_Average_blur\n" +
          "4:filter_Median_blur\n" +
          "5:filter_Gaussian_blur\n")
    key_controls = {"1": NN_interpolation,   # 建立函式索引的dic
                    "2": Bilinear_interpolation,
                    "3": Average_blur,
                    "4": Median_blur,
                    "5": Gaussian_blur}
    a = input("choose:")
    # b = input("put sign(y/n):")
    key_controls[a](image, sign)  # 呼叫所選的函式
```

可選擇要什麼功能，選擇後可選擇要變成的尺寸大小，或者是 filter 的大小。

放大縮小圖片：

- 最近鄰：(700 * 1000)

```python
def NN_interpolation(image, sign):  # 最近鄰內差 (縮小放大圖片)
    h = int(input("h(高度):"))
    w = int(input("w(寬度):"))
    rows, cols, _c = image.shape  # (640,640,3)
    create = np.zeros((h, w, 3), np.uint8)  # 建立一個所選範圍的影像矩陣
    for i in range(h):
        for j in range(w):
            a = round(j*(cols/w))  # 求出目標矩陣對應於原本矩陣的座標 (a,b) = (x,y)
            b = round(i*(rows/h))
            create[i, j] = image[b, a]  # 把原本矩陣的像素值給目標矩陣
    # 加上簽名
    rows, cols, _c = create.shape  # (640,640,3)
    resize_sign = cv2.resize(sign, dsize=(100, 50))
    create[(rows-50):(rows), (cols-100):(cols)] = resize_sign

    h = str(h)
    w = str(w)
    create = create.astype(np.uint8)  # 把目標矩陣轉成特定的dtype，不然有機會跑錯
    cv2.imshow("NN_interpolation_range:"+h+"*"+w, create)
    # cv2.imwrite("img/NN_interpolation_"+h+"*"+w+".jpeg", create)
    cv2.waitKey(0)
```

- 線性：(700 * 1000)

```python
def Bilinear_interpolation(image, sign):  # 雙線性（縮小放大圖片）
    h = int(input("h(高度):"))
    w = int(input("w(寬度):"))
    rows, cols, _c = image.shape
    create = np.zeros((h, w, 3), np.uint8)  # 建立一個所選範圍的影像矩陣
    for y in range(h):
        for x in range(w):
            # 求出目標矩陣對應於原本矩陣的座標範圍 (float_x = x + u,float_y = y + v)
            # 原中心對齊 (0,5像素的部分)
            float_x = round(x*(cols/w)+0.5*(cols/w-1), 1)
            float_y = round(y*(rows/h)+0.5*(rows/h-1), 1)
            # 分別取出整數跟小數
            u, i = math.modf(float_x)
            v, j = math.modf(float_y)
            i = int(i)
            j = int(j)
            if (i >= (cols-1) or j >= (rows-1)):  # 當所選座標在邊緣時，讓它就等於原本矩陣的邊緣的像素就好
                create[y, x] = image[j, i]
            else:                                 # a1,a2,a3,a4 分別為對應座標周圍鄰近的四個像素
                a1 = image[j, i]
                a2 = image[j, i+1]
                a3 = image[j+1, i]
                a4 = image[j+1, i+1]

                # vals = (1-u)*((1-v)*a1+v*a2) + u*((1-v)*a3+v*a4) # 做三次內差公式（先垂直再水平），求出預計像素值
                vals = (1-v)*((1-u)*a1+u*a2) + v * \
                    ((1-u)*a3+u*a4)  # 做三次內差公式（先水平再垂直），求出預計像素值
                create[y, x] = vals
    # 加上圖名
    rows, cols, _c = create.shape  # (640,640,3)
    resize_sign = cv2.resize(sign, dsize=(100, 50))
    create[(rows-50):(rows), (cols-100):(cols)] = resize_sign

    h = str(h)
    w = str(w)
    create = create.astype(np.uint8)  # 把目標矩陣轉成特定的dtype，不然有機會跑錯
    cv2.imshow("Bilinear_interpolation_range:"+h+"*"+w, create)
    #cv2.imwrite("img/Bilinear_interpolation_"+h+"*"+w+".jpeg", create)
    cv2.waitKey(0)
```

- 比較不同方向的線性：

先垂直再水平

先水平再垂直

我感覺不出差異。

模糊圖片 ：(filter: (5 * 5))

- Average blur:（模糊最明顯，而且雜訊能被消除的應該不多）

```python
def Average_blur(image, sign):  # 均值濾波器
    r = int(input("filter range:"))
    rows, cols, _c = image.shape  # (640,640,3)
    create = np.zeros((rows, cols, 3), np.uint8)  # 建立一個與要模糊的圖像同等大小的影像矩陣

    for y in range(rows):
        for x in range(cols):
            vals = average_mask(r, image, x, y)  # 得到通過平均濾波器(average_mask)的值
            # print(vals)
            create[y, x] = vals
    # 加上簽名
    rows, cols, _c = create.shape  # (640,640,3)
    resize_sign = cv2.resize(sign, dsize=(100, 50))
    create[(rows-50):(rows), (cols-100):(cols)] = resize_sign

    r = str(r)
    create = create.astype(np.uint8)  # 把目標矩陣轉成特定的dtype，不然有機會跑錯
    cv2.imshow("Average_blur_"+"filter:"+r, create)
    #cv2.imwrite("img/Average_blur_"+"filter:"+r+".jpeg", create)
    cv2.waitKey(0)
```

```python
def average_mask(r, image, x, y):  # 平均濾波器遮罩
    average = 0
    rows, cols, _c = image.shape  # (640,640,3)
    u = int(r/2)   # 所選遮罩的大小對應的座標範圍 (x+s,y+t) (-u <= (s,t) <= u)
    for i in range(-u, u+1):
        for j in range(-u, u+1):
            if (x+j) > 0 and (x+j) < cols:  # 當遮罩座標沒超過原本矩陣的範圍時
                if (y+i) > 0 and (y+i) < rows:
                    # 計算每一點的平均像素值並累加
                    average = average + image[(y+i), (x+j)]*(1/(r*r))
    return average
```



- Median blur:（雖然看得出模糊，但沒有 Average blur 明顯）

```python
def Median_blur(image, sign):  # 中值濾波器
    r = int(input("filter range:"))
    rows, cols, _c = image.shape  # (640,640,3)

    create = np.zeros((rows, cols, 3), np.uint8)  # 建立一個與要模糊的圖像同等大小的影像矩陣

    for y in range(rows):
        for x in range(cols):
            vals = median_mask(r, image, x, y)  # 得到通過中值濾波器(median_mask)的值
            # print(vals)
            create[y, x] = vals
    # 加上簽名
    rows, cols, _c = create.shape  # (640,640,3)
    resize_sign = cv2.resize(sign, dsize=(100, 50))
    create[(rows-50):(rows), (cols-100):(cols)] = resize_sign

    r = str(r)
    create = create.astype(np.uint8)  # 把目標矩陣轉成特定的dtype，不然有機會跑錯
    cv2.imshow("Median_blur_"+"filter:"+r, create)
    #cv2.imwrite("img/Median_blur_"+"filter:"+r+".jpeg", create)
    cv2.waitKey(0)
```

```python
def median_mask(r, image, x, y):  # 中值濾波器遮罩
    # 存放三個通道的像素值
    median_b = []
    median_g = []
    median_r = []

    rows, cols, _c = image.shape  # (640,640,3)
    u = int(r/2)    # 所選遮罩的大小對應的座標範圍 (x+s,y+t) (-u <= (s,t) <= u)
    for i in range(-u, u+1):
        for j in range(-u, u+1):
            if (x+j) > 0 and (x+j) < cols:  # 當遮罩座標沒超過原本矩陣的範圍時
                if (y+i) > 0 and (y+i) < rows:
                    # 存放所選座標遮罩範圍內三個通道的像素值
                    median_b.append(image[(y+i), (x+j)].tolist()[0])
                    median_g.append(image[(y+i), (x+j)].tolist()[1])
                    median_r.append(image[(y+i), (x+j)].tolist()[2])
    # 返回三個通道的中位數像素值
    vals = [np.median(median_b), np.median(median_g), np.median(median_r)]
    # print(type(np.median(median_b)))
    return vals
```


- Gaussian blur:（對照片的模糊影響較小）

```python
def Gaussian_blur(image, sign):  # 高斯濾波器
    r = int(input("filter range:"))
    # create = np.zeros((rows, cols, 3), np.uint8)

    create = cv2.GaussianBlur(image, (r, r), 0)  # opencv 的高斯函式 (0是偏差值)

    # 加上簽名
    rows, cols, _c = create.shape  # (640,640,3)
    resize_sign = cv2.resize(sign, dsize=(100, 50))
    create[(rows-50):(rows), (cols-100):(cols)] = resize_sign

    r = str(r)
    cv2.imshow("Gaussian_blur_"+"filter:"+r, create)
    #cv2.imwrite("img/Gaussian_blur_"+"filter:"+r+".jpeg", create)
    cv2.waitKey(0)
```

- 三種方式在人臉上的差異：

Average Blur

Median Blur

Gaussian Blur

模糊化：高斯 < 中位數 < 平均 ，但不確定雜訊哪個消去比較多。

簽名檔：

```python
    # 加上簽名
    rows, cols, _c = create.shape  # (640,640,3)
    resize_sign = cv2.resize(sign, dsize=(100, 50))
    create[(rows-50):(rows), (cols-100):(cols)] = resize_sign
```

心得討論：

因為要利用程式碼來做出數學的實作，所以對於公式的印象以及了解相對於課堂會更加深刻。而因為本來就會寫點 opencv，所以對於像素的處理部分本身就有一些了解，但透過這次也更加深了如何處理，尤其對於濾波器的概念更加了解，這個對於我的專題應該會提供一些幫助。