班級：資工三
學號：110590034
姓名：楊榮鈞

```
'''
Modules import
'''
import numpy as np
import cv2

def get_n_kernel(image, row, col, kernel_size):
    '''
    Get n kernel
    '''
    height = image.shape[0]
    width = image.shape[1]
    kernel = np.zeros((kernel_size, kernel_size)).astype(np.float64)
    for i in range(-kernel_size//2, kernel_size//2+1):
        for j in range(-kernel_size//2, kernel_size//2+1):
            if 0 <= row+i < height and 0 <= col+j < width:
                kernel[i+kernel_size//2][j+kernel_size//2] = image[row+i][col+j][0]

    return kernel
```

Function get_n_kernel:
先用 image.shape 找出 height 和 width，再根據 row 和 col(當前 pixel 的位置)抓取 kernel_size x kernel_size 大小的 kernel。
如作業附圖的藍色的部分。

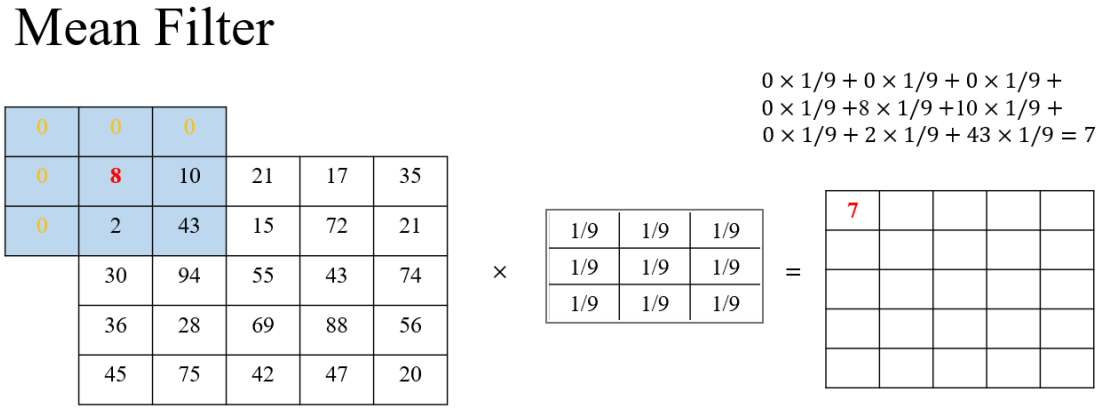
0	0	0			
0	**8**	10	21	17	35
0	2	43	15	72	21
	30	94	55	43	74
	36	28	69	88	56
	45	75	42	47	20

```python
def mean_filter(image, kernel_size):
    '''
    Mean filter
    '''
    height = image.shape[0]
    width = image.shape[1]
    image = image.astype(np.float64)
    mean_filtered_image = np.zeros((height, width)).astype(np.float64)
    # for row in range(height):
    #     for col in range(width):
    #         if image[row][col][0] != image[row][col][1] or \
    #            image[row][col][2] != image[row][col][1] or \
    #            image[row][col][2] != image[row][col][0]:
    #             print('diff')

    # print(get_n_kernel(image, 0, 0, kernel_size))

    for row in range(height):
        for col in range(width):
            kernel = get_n_kernel(image, row, col, kernel_size)
            mean_filtered_image[row][col] = np.sum(kernel) / kernel_size**2
    return mean_filtered_image
```

Function mean_filter:
利用 shape 找出 image 的 height 和 width，並利用 astype 設定 image 的 type 為 np.float64，然後利用 np.zeros 創建 mean_filtered_image 的 array。

註解的部分是測試 image 的三個通道的 value 是否皆相等，由於之前測試的結果皆為相等，因此有先註解使程式跑快一點。

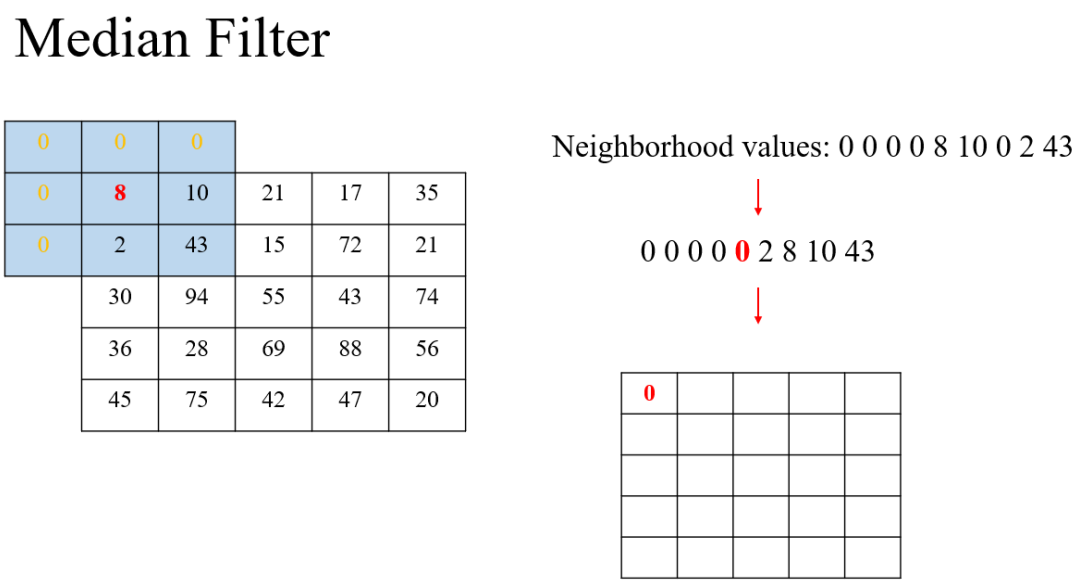
接著利用 for 迴圈將計算每個 pixel 的 value，其中會先用 get_n_kernel 抓取當前 pixel 的 kernel，並利用 np.sum(kernel) / kernel_size ** 2 計算 mean filter，如同作業的 PPT 提到的算法，我將 1/9 的部分(1/ kernel_size ** 2)提出來在最後相乘。



```python
def median_filter(image, kernel_size):
    '''
    Mean filter
    '''
    height = image.shape[0]
    width = image.shape[1]
    image = image.astype(np.float64)
    median_filtered_image = np.zeros((height, width)).astype(np.float64)

    # print(get_n_kernel(image, 0, 0, kernel_size))
    # print(np.median(get_n_kernel(image, 0, 0, kernel_size)))

    for row in range(height):
        for col in range(width):
            kernel = get_n_kernel(image, row, col, kernel_size)
            median_filtered_image[row][col] = np.median(kernel)
    return median_filtered_image
```

Function median_filter:
利用 shape 找出 image 的 height 和 width，並利用 astype 設定 image 的 type 為 np.float64，然後利用 np.zeros 創建 median_filtered_image 的 array。

接著利用 for 迴圈將計算每個 pixel 的 value，其中會先用 get_n_kernel 抓取當前 pixel 的 kernel，並利用 np.median(kernel)找出 kernel 中的中值(median value)。




```python
def gaussian_kernel(kernel_size, sigma):
    '''
    Gaussian kernel
    '''
    kernel = np.zeros((kernel_size, kernel_size)).astype(np.float64)
    for x in range(-kernel_size//2, kernel_size//2+1):
        for y in range(-kernel_size//2, kernel_size//2+1):
            kernel[x+kernel_size//2][y+kernel_size//2] = 1 / (2 * np.pi * sigma**2) * np.exp(-(x**2 + y**2) / (2 * sigma**2))
    return kernel
```

Function gaussian_kernel:
計算 gaussian 的 value，以 Gaussian 2D filter 的公式進行計算，如作業簡報的公式算法 G(x, y) = (1 / 2 * pi * sigma ** 2) * exp ** (-(x ** 2 + y ** 2) / (2 * sigma ** 2))。
其中會以 kernel_size 的大小去做整個 kernel 的處理如圖所示(圖片是 3x3 的 kernel 大小)。

Gaussian 2D Filter

1. $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

If $\sigma = 1$,

(-1,-1)	(0,-1)	(1,-1)
(-1,0)	(0,0)	(1,0)
(-1,1)	(0,1)	(1,1)

⟶

0.0585	0.0965	0.0585
0.0965	0.1591	0.0965
0.0585	0.0965	0.0585

```python
def gaussian_filter(image, kernel_size, sigma):
    '''
    Gaussian filter
    '''
    height = image.shape[0]
    width = image.shape[1]
    image = image.astype(np.float64)
    gaussian_filtered_image = np.zeros((height, width)).astype(np.float64)
    gaussian = gaussian_kernel(kernel_size, sigma)
    gaussian = gaussian / np.sum(gaussian)
    for row in range(height):
        for col in range(width):
            kernel = get_n_kernel(image, row, col, kernel_size)
            gaussian_filtered_image[row][col] = np.sum(kernel * gaussian)
    return gaussian_filtered_image
```

Function gaussian_filter:
利用 shape 找出 image 的 height 和 width，並利用 astype 設定 image 的 type 為 np.float64，然後利用 np.zeros 創建 gaussian_filtered_image 的 array。

利用 gaussian_kernel 找出 kernel 的 gaussian value，然後將 gaussian 除 gaussian 的總和並存在 gaussian 中。
利用 for 迴圈計算每個 pixel 的 value，其中會先用 get_n_kernel 抓取當前 pixel 的 kernel，並利用 np.sum(kernel * gaussian)得出 kernel * gaussian 的總和，再存入當前 pixel 的 value 中。

Gaussian 2D Filter

1. $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

   If $\sigma = 1$,

(-1,-1)	(0,-1)	(1,-1)
(-1,0)	(0,0)	(1,0)
(-1,1)	(0,1)	(1,1)

⟶

0.0585	0.0965	0.0585
0.0965	0.1591	0.0965
0.0585	0.0965	0.0585

2. Normalization
3. Convolution ：I*G

```python
def custom_filter(image, kernel_size, sigma, filter_type):
    '''
    Custom filter
    1. mean filter + median filter
    2. mean filter + gaussian filter
    3. median filter + gaussian filter
    4. mean filter + median filter + gaussian filter
    '''
    height = image.shape[0]
    width = image.shape[1]
    image = image.astype(np.float64)
    custom_filtered_image = np.zeros((height, width)).astype(np.float64)
    gaussian = gaussian_kernel(kernel_size, sigma)
    gaussian = gaussian / np.sum(gaussian)
    for row in range(height):
        for col in range(width):
            kernel = get_n_kernel(image, row, col, kernel_size)
            if filter_type == 1:
                custom_filtered_image[row][col] = (np.sum(kernel) / kernel_size**2 + np.median(kernel)) / 2
            elif filter_type == 2:
                custom_filtered_image[row][col] = (np.sum(kernel) / kernel_size**2 + np.sum(kernel * gaussian)) / 2
            elif filter_type == 3:
                custom_filtered_image[row][col] = (np.median(kernel) + np.sum(kernel * gaussian)) / 2
            elif filter_type == 4:
                custom_filtered_image[row][col] = (np.sum(kernel) / kernel_size**2 + np.median(kernel) + np.sum(kernel * gaussian)) / 3

    return custom_filtered_image
```

Function custom_filter:
結合前面的所提到的 filter，組合成新的 filter，其中有：(mean_filter + median_filter) / 2、(mean_filter + gaussian_filter) / 2、(median_filter + gaussian_filter) / 2 和(mean_filter + median_filter + gaussian_filter) / 3。

利用 shape 找出 image 的 height 和 width，並利用 astype 設定 image 的 type 為 np.float64，然後利用 np.zeros 創建 custom_filtered_image 的 array。

利用 gaussian_kernel 找出 kernel 的 gaussian value，然後將 gaussian / gaussian 的總和並存在 gaussian 中，利用 for 迴圈計算每個 pixel 的 value，其中會先用 get_n_kernel 抓取當前 pixel 的 kernel，並根據 filter_type 決定接下來的步驟。

當 filter_type == 1 時，會計算(mean_filter + median_filter) / 2，也就是 (np.sum(kernel) / kernel_size**2 + np.median(kernel)) / 2，並把結果的 value 存入 custom_filtered_image 當前的 pixel 的 value 中。

當 filter_type == 2 時，會計算(mean_filter + gaussian_filter) / 2，也就是 (np.sum(kernel) / kernel_size**2 + np.sum(kernel * gaussian)) / 2，並把結果的 value 存入 custom_filtered_image 當前的 pixel 的 value 中。

當 filter_type == 3 時，會計算(median_filter + gaussian_filter) / 2，也就是 (np.median(kernel) + np.sum(kernel * gaussian)) / 2，並把結果的 value 存入 custom_filtered_image 當前的 pixel 的 value 中。

當 filter_type == 4 時，會計算(mean_filter + median_filter + gaussian_filter) / 3，也就是(np.sum(kernel) / kernel_size**2 + np.median(kernel) + np.sum(kernel * gaussian)) / 3，並把結果的 value 存入 custom_filtered_image 當前的 pixel 的 value 中。

```python
def image(number, sigma):
    '''
    For img{number}.png
    '''
    origin_image = cv2.imread(f'images/img{number}.jpg')

    mean_filtered_image_3 = mean_filter(origin_image, 3)
    cv2.imwrite(f'results/img{number}_q1_3.jpg', mean_filtered_image_3)
    mean_3 = cv2.imread(f'results/img{number}_q1_3.jpg')

    mean_filtered_image_7 = mean_filter(origin_image, 7)
    cv2.imwrite(f'results/img{number}_q1_7.jpg', mean_filtered_image_7)
    mean_7 = cv2.imread(f'results/img{number}_q1_7.jpg')

    median_filtered_image_3 = median_filter(origin_image, 3)
    cv2.imwrite(f'results/img{number}_q2_3.jpg', median_filtered_image_3)
    median_3 = cv2.imread(f'results/img{number}_q2_3.jpg')

    median_filtered_image_7 = median_filter(origin_image, 7)
    cv2.imwrite(f'results/img{number}_q2_7.jpg', median_filtered_image_7)
    median_7 = cv2.imread(f'results/img{number}_q2_7.jpg')

    gaussian_filtered_image = gaussian_filter(origin_image, 5, sigma)
    cv2.imwrite(f'results/img{number}_q3.jpg', gaussian_filtered_image)
    gaussian = cv2.imread(f'results/img{number}_q3.jpg')

    for i in range(1, 5):
        custom_filtered_image = custom_filter(origin_image, 5, sigma, i)
        cv2.imwrite(f'results/img{number}_q4_{i}.jpg', custom_filtered_image)

    cv2.imshow('Origin Image', origin_image)
    cv2.imshow('Mean Filtered Image (3x3)', mean_3)
    cv2.imshow('Mean Filtered Image (7x7)', mean_7)
    cv2.imshow('Median Filtered Image (3x3)', median_3)
    cv2.imshow('Median Filtered Image (7x7)', median_7)
    cv2.imshow('Gaussian Filtered Image', gaussian)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Function image:
先用 cv2 的 imread 讀取圖片到 origin_image。

接著使用 mean_filter(origin_image, 3)得出 mean_filter 且 kernel 為 3x3 大小的圖片並存入 mean_filtered_image_3 中，再來使用 cv2.imwrite 存檔，最後再使用 cv2.imread 讀取圖片到 mean_3。

接著使用 mean_filter(origin_image, 7)得出 mean_filter 且 kernel 為 7x7 大小的圖片並存入 mean_filtered_image_7 中，再來使用 cv2.imwrite 存檔，最後再使用 cv2.imread 讀取圖片到 mean_7。

接著使用 median_filter(origin_image, 3)得出 median_filter 且 kernel 為 3x3 大小的圖片並存入 median_filtered_image_3 中，再來使用 cv2.imwrite 存檔，最後再使用 cv2.imread 讀取圖片到 median_3。

接著使用 median_filter(origin_image, 7)得出 median_filter 且 kernel 為 7x7 大小的圖片並存入 median_filtered_image_7 中，再來使用 cv2.imwrite 存檔，最後再使用 cv2.imread 讀取圖片到 median_7。

接著使用 gaussian_filter(origin_image, 5, sigma)得出 gaussian_filter 且 kernel 為 5x5 大小和傳入的 sigma 做運算後的圖片，並存入 gaussian_filtered_image 中，再來使用 cv2.imwrite 存檔，最後再使用 cv2.imread 讀取圖片到 gaussian。

然後利用 for 迴圈得出 custom_filter 的 4 種 filter type 的圖片，並使用 cv2.imwrite 存檔。

利用 cv2 的 imshow 查看 original image、mean_3(Mean Filtered Image (3x3))、mean_7(Mean Filtered Image (7x7))、median_3(Median Filtered Image (3x3))、median_7(Median Filtered Image (7x7))和 gaussian(Gaussian Filtered Image)。

利用 cv2 的 waitKey(0)和 destroyAllWindows()將 imshow 顯示出來的圖片關閉。

```
if __name__ == '__main__':
    image(number=1, sigma=1)
    image(number=2, sigma=1.4)
    image(number=3, sigma=1.2)
```

執行 image，產生所有助教給的圖片的 Mean Filtered Image (3x3 kernel size)、Mean Filtered Image (7x7 kernel size)、Median Filtered Image (3x3 kernel size)、Median Filtered Image (7x7 kernel size)、Gaussian Filtered Image 以及自己製作的 filter image。

Result images

img1_q1_3.jpg

我預期的結果與輸出大致相同，與原圖相比模糊程度輕微，細節略有損失。

img1_q1_7.jpg

我預期的結果與輸出大致相同，與原圖相比模糊程度輕微，細節略有損失。

使用 mean_filter 的結果看起來是比較像是模糊化影像，其中 7x7 大小的 kernel 模糊的程度比 3x3 高。

img1_q2_3.jpg

我預期的結果與輸出大致相同，與原圖相比去除了部分 noise value，圖像相對清晰。

img1_q2_7.jpg

我預期的結果與輸出大致相同，與原圖相比去除了更多 noise value，但同時細節也有所減少，整體效果比 3x3 kernel 更為平滑。

使用 median_filter 的結果看起來會 filter 掉一些 noise value，像是樹枝上的斑

點，從 7x7 大小的 kernel 看，會發現幾乎消失掉，其中 7x7 大小的 kernel 的 filter 程度比 3x3 高。

img1_q3.jpg

我預期的結果與輸出大致相同，與原圖相比 gaussian_filter 的效果介於 mean_filter 和 median_filter 之間，模糊效果比較均勻，細節保持相對較好。

img2_q1_3.jpg

我預期的結果與輸出大致相同，與原圖相比模糊程度輕微，細節略有損失。

img2_q1_7.jpg

我預期的結果與輸出大致相同，與原圖相比模糊程度增加，細節損失更多，圖像變得更加平滑。

使用 mean_filter 的結果看起來是比較像是模糊化影像(7x7 kernel 有種 144p 的感受)，其中 7x7 大小的 kernel 模糊的程度比 3x3 高。

img2_q2_3.jpg

我預期的結果與輸出大致相同，與原圖相比減少了圖像中的 noise value，保持了較多的細節。

img2_q2_7.jpg

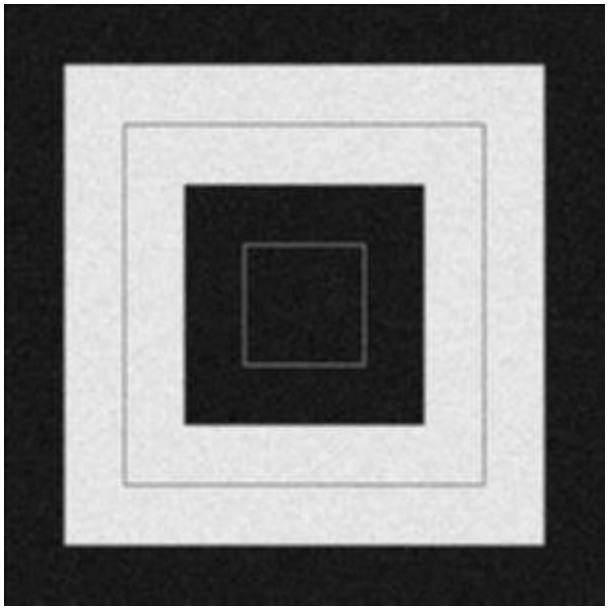
我預期的結果與輸出大致相同，與原圖相比更多 noise value 被去除，但圖像細節也有所損失，整體效果比 3x3 kernel 更為平滑。

使用 median_filter 的結果會 filter 掉一些 noise value，像是圖片上的一堆 noise value，其中 7x7 大小的 kernel 的 filter 程度比 3x3 高，但是 7x7 的 kernel 也會讓老虎的細節損失更多。

img2_q3.jpg

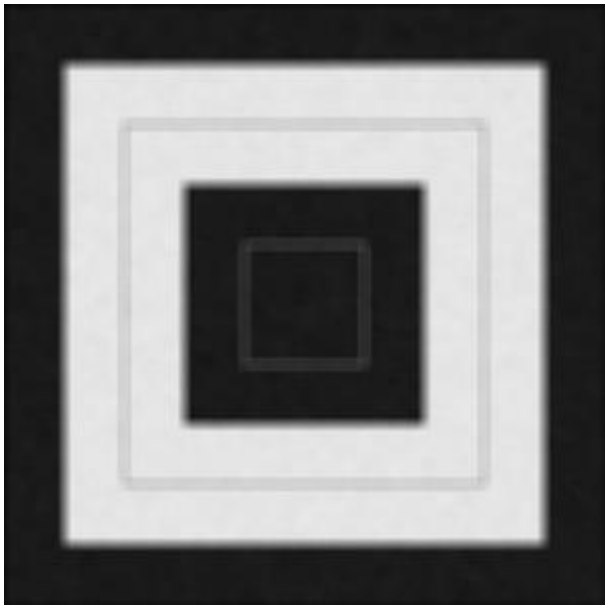
我預期的結果與輸出大致相同，與原圖相比 gaussian_filter 對圖像進行了均勻的模糊處理，細節保持較好，圖像變得更柔和。

img3_q1_3.jpg



我預期的結果與輸出大致相同，與原圖相比邊緣變得稍微模糊，圖像整體效果略有改變。

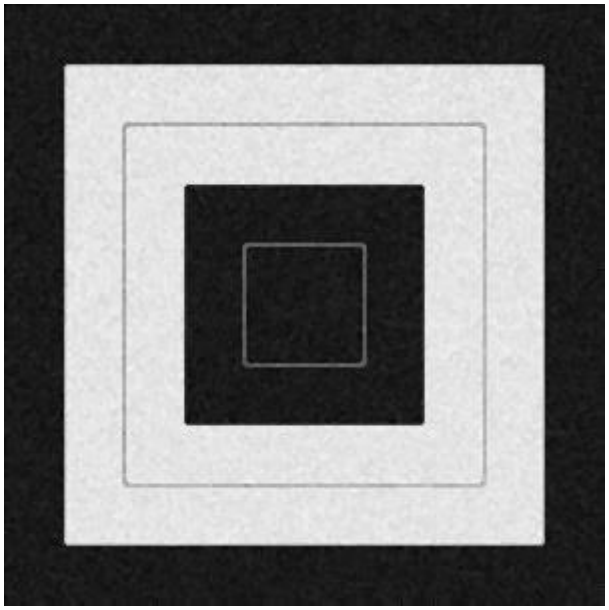
img3_q1_7.jpg



我預期的結果與輸出大致相同，與原圖相比邊緣模糊更加明顯，圖像變得更平滑。

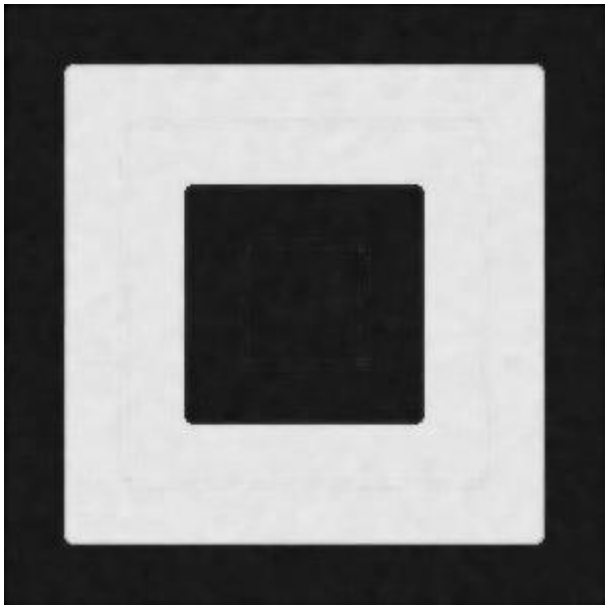
使用 mean_filter 的結果看起來是模糊化影像，其中 7x7 大小的 kernel 模糊的程度比 3x3 高，而 7x7 大小的 kernel 的 filter 結果讓眼睛相對於 3x3 大小的 kernel 舒服許多。

img3_q2_3.jpg



我預期的結果與輸出大致相同，與原圖相比去除了部分 noise value，邊緣保持相對清晰。

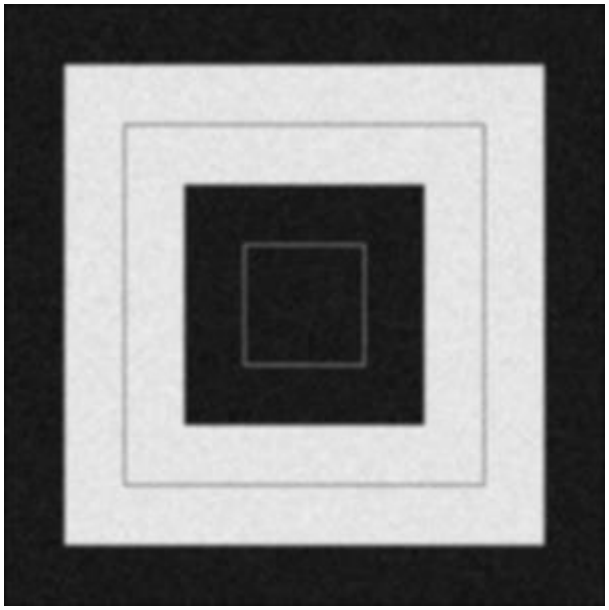
img3_q2_7.jpg



我預期的結果與輸出大致相同，與原圖相比去除了更多 noise value，但邊緣模糊(白色方塊的黑色線條幾乎快看不到，只剩下隱約的線條，黑色方塊的白色線條也是相同結果)，圖像整體變得平滑。

使用 median_filter 的結果會 filter 掉一些 noise value，其中 7x7 大小的 kernel 的 filter 程度比 3x3 高，但是 7x7 的 kernel 讓黑色線條(方形框的線條)和白色線條(方形框的線條)幾乎看不見。

img3_q3.jpg



我預期的結果與輸出大致相同，與原圖相比 gaussian_filter 對圖像進行了均勻的模糊處理，邊緣變得柔和，圖像看起來更平滑且細節保持較好。

Mean filter：適合進行輕度模糊處理，kernel 越大模糊效果越明顯，但會損失更多細節。
Median filter：適合 filter noise value，保持邊緣清晰度，kernel 越大 filter 的 noise value 越多，但也會造成細節損失。
Gaussian filter：模糊處理效果均勻，能夠有效平滑圖像，同時保持較好的細節。

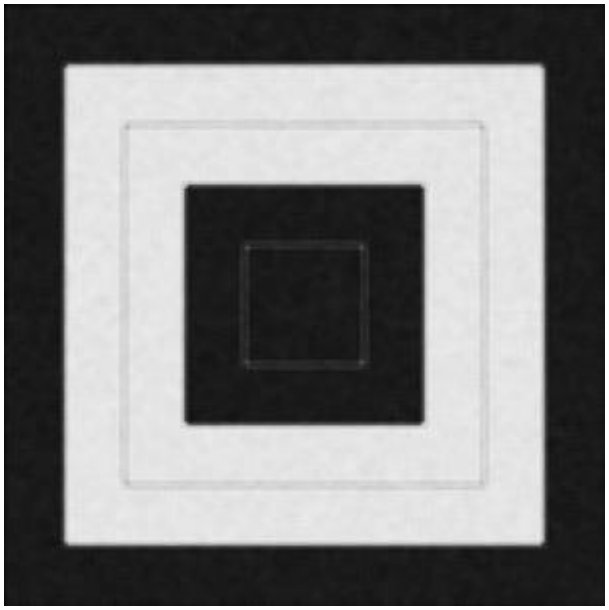
Extra result (custom filter, 5x5 kernel)

(mean_filter + median_filter) / 2

img1_q4_1.jpg

img2_q4_1.jpg

img3_q4_1.jpg



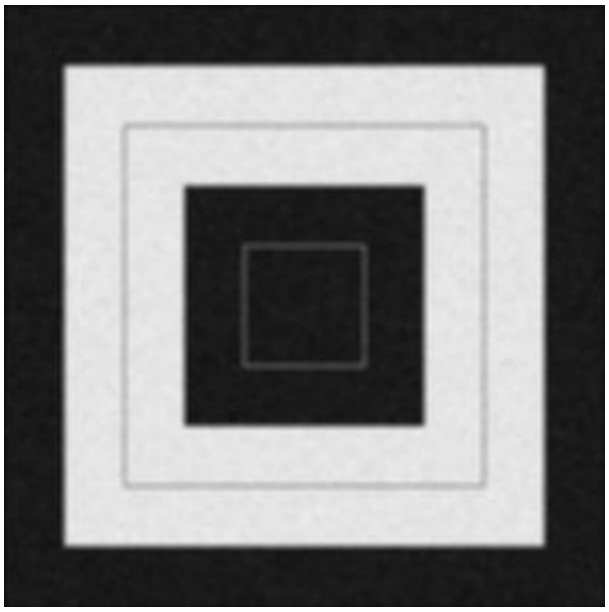
第一種方法我覺得效果不錯，有著 mean_filter 和 median_filter 的優點結合，其中有一些 noise value 有被 filter 掉，且細節沒有損失太多。

(mean_filter + gaussian_filter) / 2

img1_q4_2.jpg

img2_q4_2.jpg

img3_q4_2.jpg



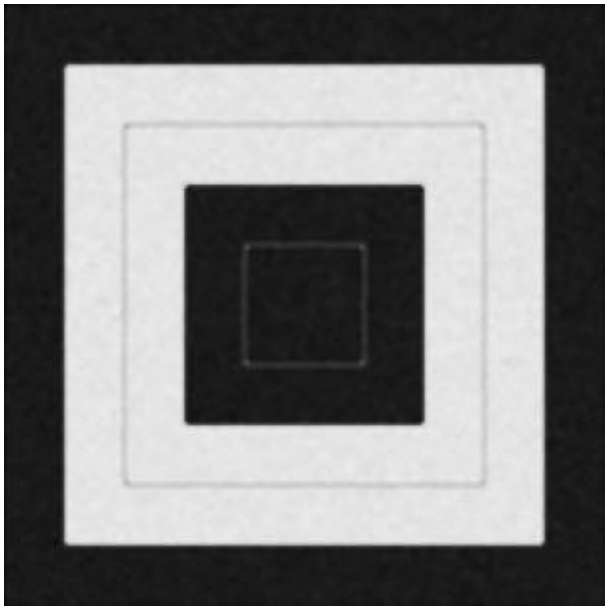
第二種的方法相對於第一種，保留的細節較多，但老虎的圖片中，仍然有許多 noise value。

(median_filter + gaussian_filter) / 2

img1_q4_3.jpg

img2_q4_3.jpg

img3_q4_3.jpg



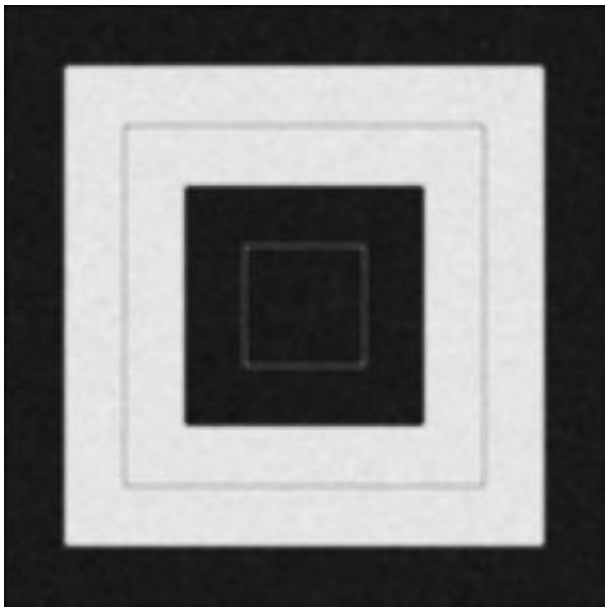
第三種方法相較於第一種，我覺得其保留的細節比較多一點，且 noise value 也有被 filter 一些。

(mean_filter + median_filter + gaussian_filter) / 3

img1_q4_4.jpg

img2_q4_4.jpg

img3_q4_4.jpg



第四種方法，我覺得與第三種的觀感肉眼看起來差不多，只是 noise value 似乎比較多一點。

第一種方法：有 filter 掉 noise value，但細節也有所損失。
第二種方法：保留較多細節，但留的 noise value 較多。
第三種方法：有 filter 掉 noise value，細節相較於第一種方法損失較少。
第四種方法：與第三種方法相似，從老虎的圖片來看，似乎 noise value 被 filter 掉的程度比第三種方法低。

最後的結果總結：
這次 hw5 的作業難度在於理解三種 filter 的概念是什麼，並理解公式是如何運作，我覺得根據講義及作業的簡報內容來看，這次的難度沒有這麼高。其中經過這次作業我了解到：Mean filter 適合輕度模糊處理，kernel 越大模糊效果越明

顯，但會損失更多細節；Median filter 適合 filter noise value，保持邊緣清晰度，kernel 越大 filter 的 noise value 越多，但也會造成細節損失；Gaussian filter 模糊處理效果均勻，能夠有效平滑圖像，同時保持較好的細節。
其中我根據上面三種組合中我發現 Median filter + Gaussian filter 的 filter 效果還不錯，如果希望保存多一點細節的話可以用 Mean filter + Median filter + Gaussian filter，若是希望可以去除較多 noise value，且比較不在意一些細節損失的話，用 Mean filter + Median filter 效果會比較好，只是想要保留更多細節，而不在意 noise value 則是用 Mean filter + Gaussian filter。