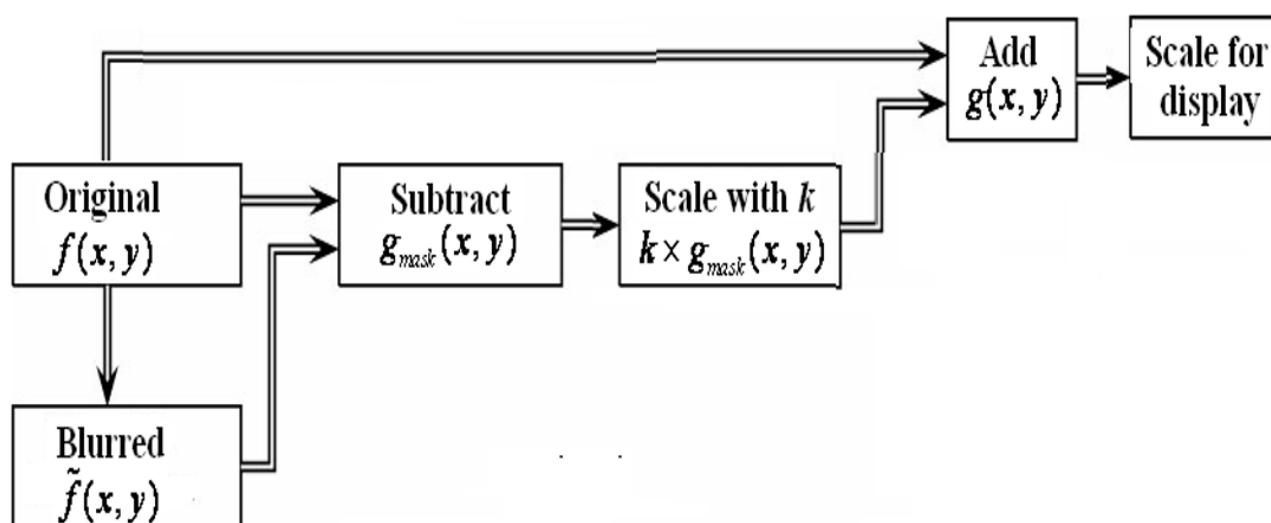


# Homework No.4

Student ID: 41047902S Name: 鄭淮薰

## Problem Statement

1. Select an experimental image
2. Apply a  $n$  by  $n$  (a) average filter and (b) median filter to the image
3. Unsharp masking



## Input / Output

Using python to run this program, and here are some parameters user can use to specify the input and output

```
usage: hw4.py [-h] [-i IMAGE] [-o OUTPUT] [-t TYPE] [-n N] [-k SCALE]
```





options:

-h,	--help	show this <b>help</b> message and <b>exit</b>
-i IMAGE,	--image IMAGE	path to the image
-o OUTPUT,	--output OUTPUT	name of the output image
-t TYPE,	--type TYPE	<b>type</b> of the output image
-n N,	--n N	size of the mask
-k SCALE,	--scale SCALE	scale of the image

## Test Results

Test case 1 將 Fig1.1 作為 experimental image 輸入，Fig1.2 為程式讀入圖片後的初始圖片，與 Fig1.1 相同。Fig1.3 以及 Fig1.5 是將 Fig1.2 分別套上 average filter 與 median filter 後的結果，從這兩張圖可觀察到雖然兩者皆能消除 noises，達到平滑的效果，但使用 median filter 較能保留圖片中的邊緣細節。而最後 Fig1.4 與 Fig1.6 則是分別使用 average filter 與 median filter 作為 low pass filter 來實作 unsharp masking，將兩張圖片與原圖比較，可明顯觀察到圖片中蘋果的邊緣更銳利，圖片的邊緣與細節也更突出。

Note: 為使結果更顯著、易於比較及觀察，這裡使用的 35 x 35 的 filter 來實作，使得 Fig1.3, Fig1.5 變得非常模糊。

Fig1.1 Input Image	Fig1.2 Output Original
	
Fig1.3 Average Filter	Fig1.4 Average Unsharp Masking
	

**Fig1.5 Median Filter**



**Fig1.6 Median Unsharp Masking**



## Comment & Discuss

在這項作業中，我使用了 OpenCV 的 `cv2` 模組來讀取初始圖片，並分別實作了 `average filter` 及 `median filter` 來將圖片模糊化。與上次實作彩色圖像不同的是，為了避免圖片顏色失真，我將 RGB 的圖片轉換到 YUV 色彩空間後，僅針對 Y 通道做降噪操作。這樣做既能更好的保留圖像細節，又能有效的降噪。在第二部分中，我分別使用 `average filter` 與 `median filter` 作為 low pass filter，然後按照題目圖片中的流程做 Unsharp masking。需要注意的是，在將 original image 與 blurred image 相減時，必須將兩圖片的 RGB 值正規化到 0 至 1 的範圍內，完成計算後再轉回 0 至 255 的範圍，以避免 overflow 的問題。此外，我也嘗試使用了 `cv2` 的 `blur` 及 `medianBlur` 方法來對圖片做 `average filtering` 與 `median filtering`。生成的 Unsharp Masking 圖片效果和自己實作的差不多，但效率更高。

## Source Code

```
import numpy as np
import cv2
import argparse

# argument parser
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--image', type=str,
                        default='img/apple.jpeg', help='path to the image')
    parser.add_argument('-o', '--output', type=str,
                        default='output', help='name of the output image')
    parser.add_argument('-t', '--type', type=str, default='',
                        help='type of the output image')
    parser.add_argument('-n', '--n', type=int, default=35,
                        help='size of the mask')
    parser.add_argument('-k', '--scale', type=int, default=0.7,
                        help='scale of the image')
    args = parser.parse_args()
    if(args.output == ''): args.output = args.image.split('/')[1].split('.')[0]
    if(args.type == ''): args.type = args.image.split('.')[1]
    return args

def median_blur(img, n):
    # padding img
    img_pad = np.pad(img, ((n//2, n//2), (n//2, n//2), (0, 0)), 'edge')
    img_yuv = cv2.cvtColor(img_pad, cv2.COLOR_BGR2YUV)

    height, width = img.shape[:2]
    blurred_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

    for i in range(height):
        for j in range(width):
            blurred_yuv[i, j, 0] = np.median(img_yuv[i:i+n, j:j+n, 0])

    blurred = cv2.cvtColor(blurred_yuv.astype(np.uint8), cv2.COLOR_YUV2BGR)

    return blurred
```

```

def average_blur(img, n):
    mask = np.ones((n, n))
    mask = mask / (n*n)
    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    img_yuv[:, :, 0] = cv2.filter2D(img_yuv[:, :, 0], -1, mask)
    blurred = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)

    return blurred

# main
args = parse_args()

img = cv2.imread(args.image, cv2.IMREAD_COLOR)
n = args.n
k = args.scale

# unsharp masking using average filter
# average_burred = cv2.blur(img, (n, n))
average_burred = average_blur(img, n)
high_pass = (img/255 - average_burred/255) * 255
average_sharp = img + k * high_pass

# unsharp masking using median filter
# median_burred = cv2.medianBlur(img, n)
median_burred = median_blur(img, n)
high_pass = (img/255 - median_burred/255) * 255
median_sharp = img + k * high_pass

# save the images
cv2.imwrite(args.output + '_original.' + args.type, img)
cv2.imwrite(args.output + '_average_burred.' + args.type, average_burred)
cv2.imwrite(args.output + '_median_burred.' + args.type, median_burred)
cv2.imwrite(args.output + '_average_sharp.' + args.type, average_sharp)
cv2.imwrite(args.output + '_median_sharp.' + args.type, median_sharp)

```