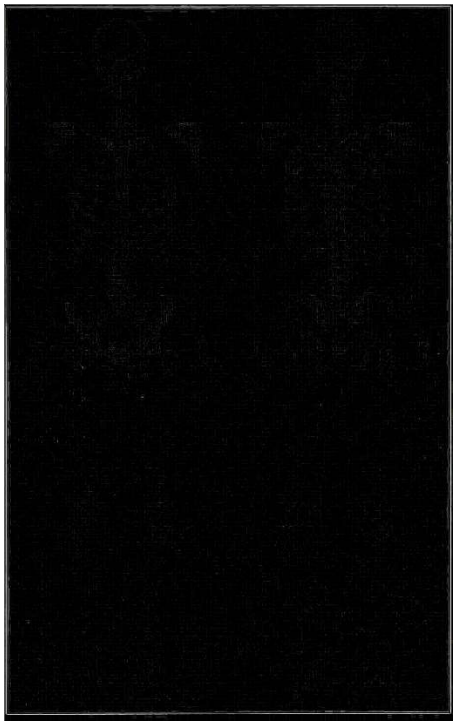
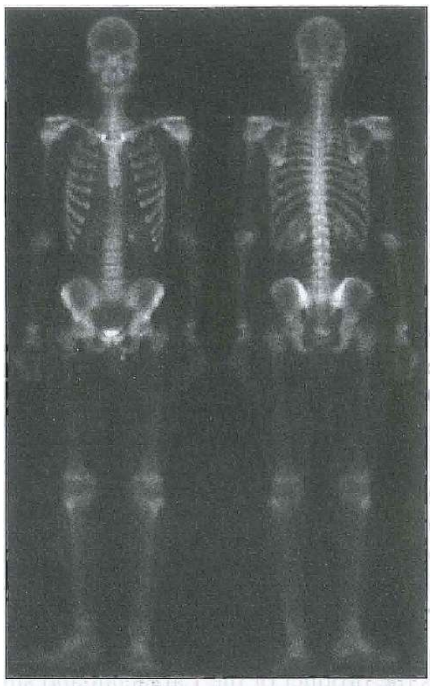
電信碩 309513061 簡詠倫 HW3_4

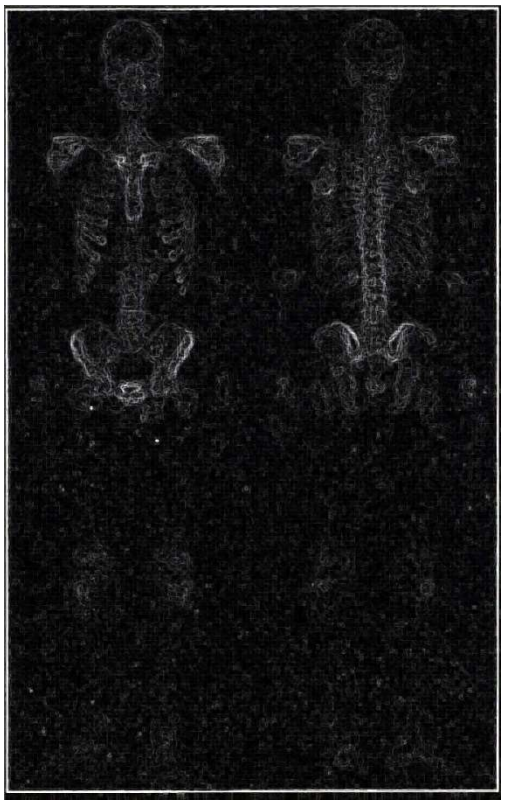
1. 按照課本 p.183~195 的內容來實作。首先是 Laplacian kernel 的部分，我使用參數為 kernel size=3*3，中間值為 8 其餘為-1 的核。經過此運算子的結果如下，雖然不明顯，但仍有偵測到邊緣。



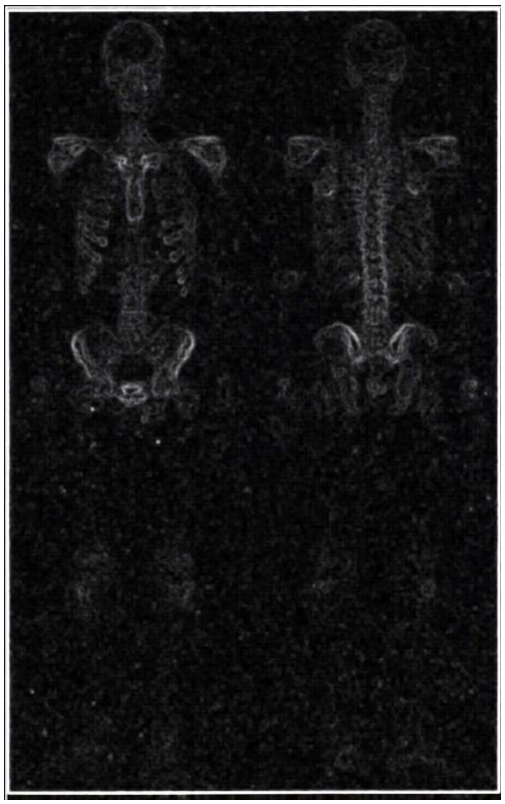
再將此結果加上原圖，進行影像銳化，結果如下。



接下來利用 Sobel 核運算子作邊緣偵測，運算子的數值參照課本，實現方式是 x 方向結果的絕對值加上 y 方向結果的絕對值，結果如下圖。

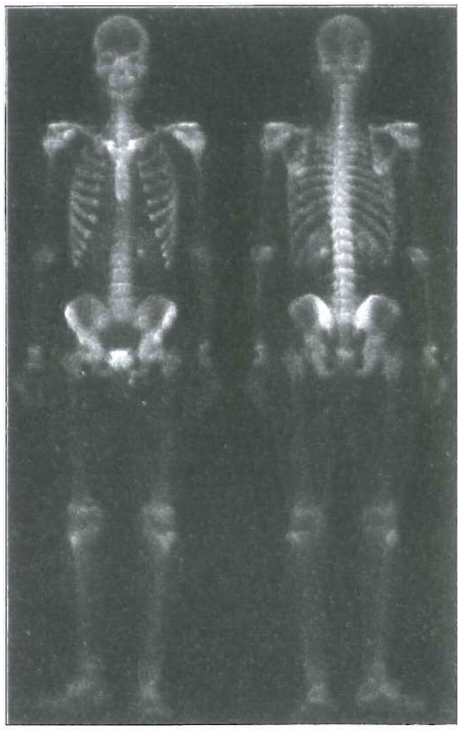


可看出偵測的結果不差。接著將其經過大小為 5*5 的平均濾波器模糊化，結果如下圖。

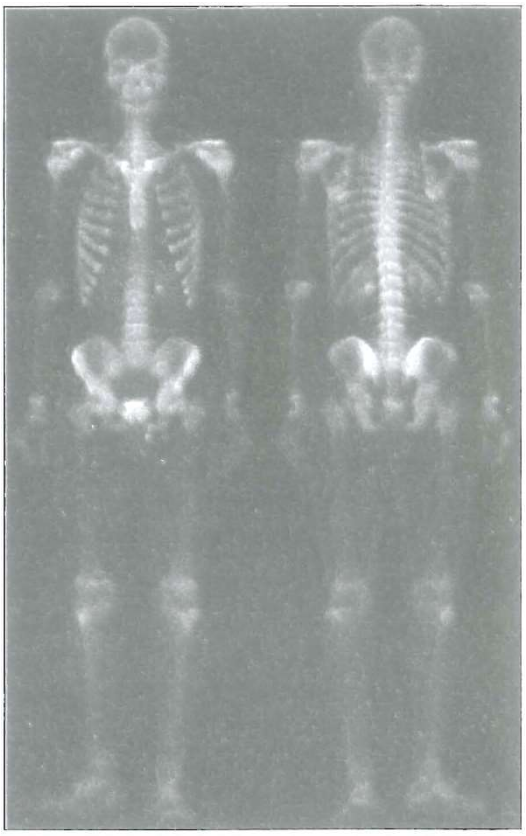


在細節部分已被模糊處理，接著將此結果與經 Laplacian 運算銳化過的圖片相乘，結果如下。

最後再跟原始圖片相加，即完成影像增強，結果如下。



最後再照課本所說，再做 power-law transformation，結果如下。



結果沒有課本上好看，但再做 power-law traansformation 前的增強結果還不錯。逐步的程式碼如下。

```
clc; clear;
fig_original = double(imread('Bodybone.bmp'))/255 ;
sz = size(fig_original);

figure;
subplot(2, 4, 1);
imshow(fig_original);
imwrite(fig_original, 'origin.png');
laplace_result = zeros(sz);
for i=1:3
    laplace_result(:,:,i) = imgfilter2([-1 -1 -1; -1 8 -1; -1 -1 -1], fig_original(:,:,i),1);

end

subplot(2, 4, 2);
imshow(laplace_result);
imwrite(laplace_result, 'laplace_result.png');
%%
sharpened_laplace_result = fig_original + laplace_result;
subplot(2, 4, 3);
imshow(sharpened_laplace_result);
imwrite(sharpened_laplace_result, 'sharpened_laplace_result.png');
%%
sobel_grad = zeros(sz);
for i=1:3
    sobel_grad(:,:,i) = abs(imgfilter2([-1 -2 -1; 0 0 0; 1 2 1], fig_original(:,:,i),1)) + abs(imgfilter2([-1 0 1; -2 0 2; -1 0 1], fig_orig
end
subplot(2, 4, 4);
imshow(sobel_grad);
imwrite(sobel_grad, 'sobel_grad.png');
%%
smoothed_sobel_grad = zeros(sz);
for i=1:3
smoothed_sobel_grad(:,:,i) = imgfilter2(ones(5)/25, sobel_grad(:,:,i),2);
end
subplot(2, 4, 5);
imshow(smoothed_sobel_grad);
```

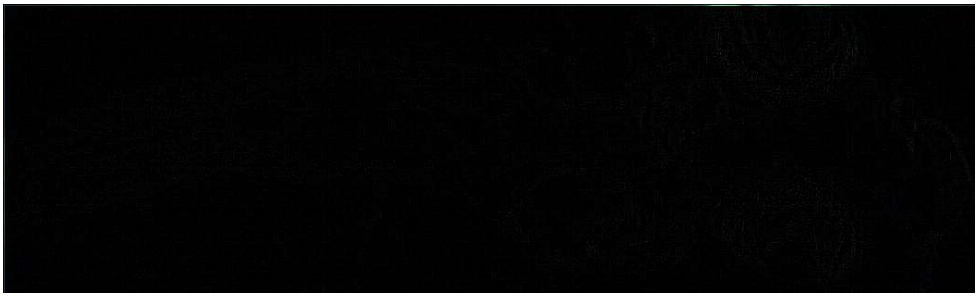
```
%%
product_laplace_sobel  = zeros(sz);
for i=1:3
product_laplace_sobel(:,:,i) = sharpened_laplace_result(:,:,i) .* smoothed_sobel_grad(:,:,i);
end
subplot(2, 4, 6);
imshow(product_laplace_sobel);
imwrite(product_laplace_sobel, 'product_laplace_sobel.png');
```

```
%%
sharpened_image  = zeros(sz);
for j=1:3
sharpened_image(:,:,j) = max(fig_original(:,:,j) + product_laplace_sobel(:,:,j), 0);
end
subplot(2, 4, 7);
imshow(sharpened_image);
imwrite(sharpened_image, 'sharpened_image.png');
```

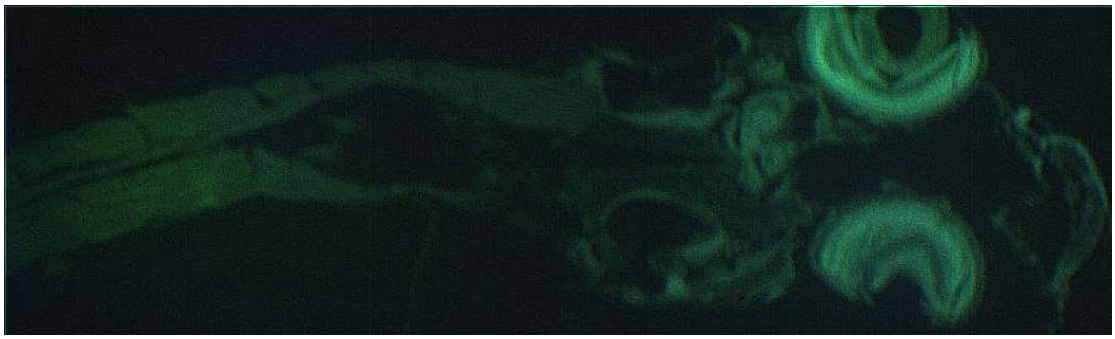
```
%%
final_result  = zeros(sz);
for k=1:3
final_result(:,:,k) = power(sharpened_image(:,:,k), 0.5);
end
subplot(2, 4, 8);
imshow(final_result);
imwrite(final_result, 'final_result.png');
```

```
function [ g ] = imgfilter2( w, x ,p)
    [h_x, w_x] = size(x);
    g = zeros(h_x, w_x);
    D = padarray(x,[p p],0,'both');
    for i = p+1 : h_x
        for j = p+1 : w_x
            g(i,j) = sum(w.*D(i-p:i+p,j-p:j+p),'All');
        end
    end
end
```

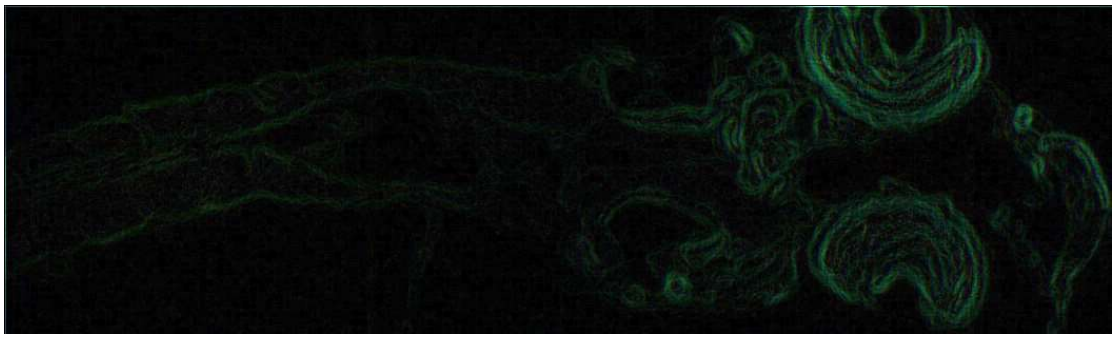
2. 步驟大致與第一題相同，先做 Laplacian 運算，結果如下。



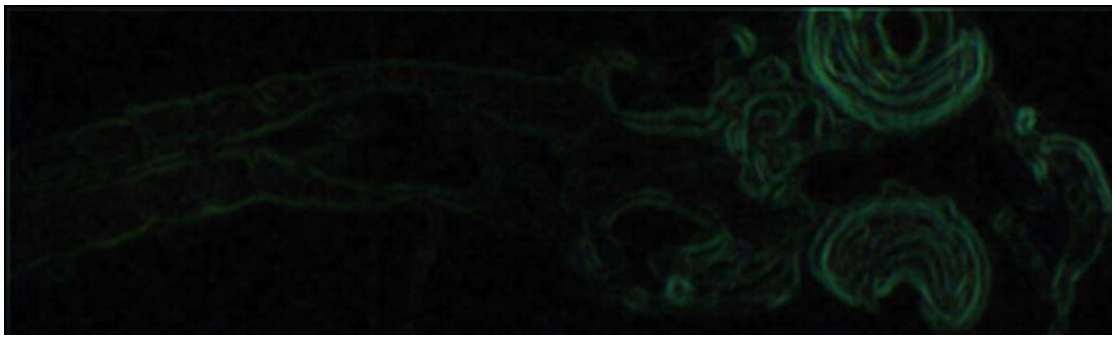
加上原圖進行銳化，結果如下。



接著用 Sobel 運算子進行邊緣偵測，結果如下。

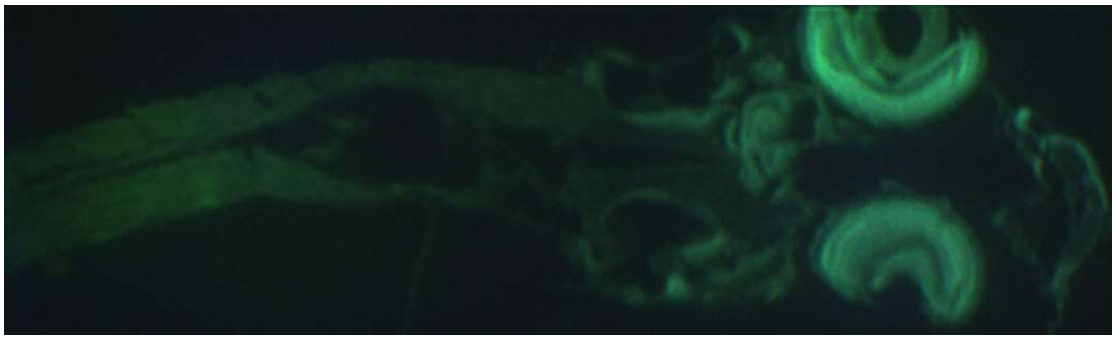


再將結果做模糊化，結果如下。

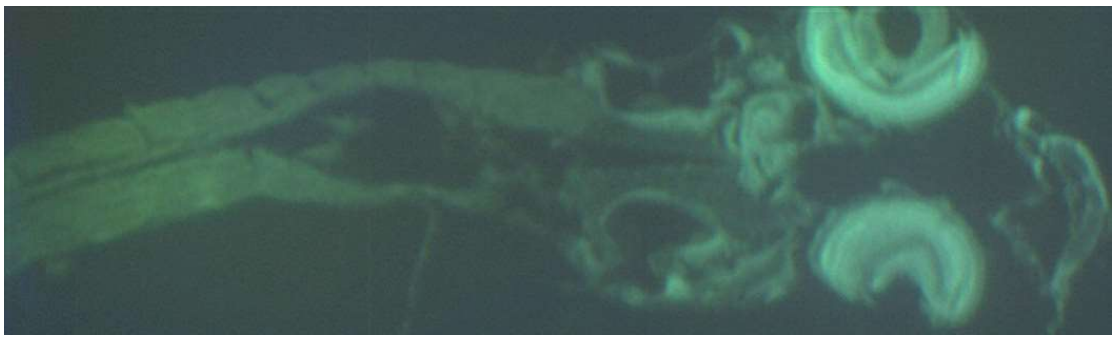


再與銳化過的圖片相乘作 mask，結果如下。

最後再與原圖相加，結果如下。



最後作 power-law transformation，結果如下。



逐步程式碼如下。

```
clc; clear;
fig_original = double(imread('fish.jpg'))/255;
sz = size(fig_original);
figure;
subplot(2, 4, 1);
imshow(fig_original);
imwrite(fig_original, 'fish_origin.png');
laplace_result = zeros(sz);
for i=1:3
    laplace_result(:,:,i) = imgfilter2([-1 -1 -1; -1 8 -1; -1 -1 -1], fig_original(:,:,i),1);
end

subplot(2, 4, 2);
imshow(laplace_result);
imwrite(laplace_result, 'fish_laplace_result.png');

sharpened_laplace_result = fig_original + laplace_result;
subplot(2, 4, 3);
imshow(sharpened_laplace_result);
imwrite(sharpened_laplace_result, 'fish_sharpened_laplace_result.png');

sobel_grad = zeros(sz);
for i=1:3
    sobel_grad(:,:,i) = abs(imgfilter2([-1 -2 -1; 0 0 0; 1 2 1], fig_original(:,:,i),1)) + abs(imgfilter2([-1 0 1; -2 0 2; -1 0 1], fig_orig
end
subplot(2, 4, 4);
imshow(sobel_grad);
imwrite(sobel_grad, 'fish_sobel_grad.png');

smoothed_sobel_grad = zeros(sz);
for i=1:3
smoothed_sobel_grad(:,:,i) = imgfilter2(ones(5)/25, sobel_grad(:,:,i),2);
end
subplot(2, 4, 5);
imshow(smoothed_sobel_grad);
imwrite(smoothed_sobel_grad, 'fish_smoothed_sobel_grad.png');
```

```
product_laplace_sobel  = zeros(sz);
for i=1:3
product_laplace_sobel(:,:,i) = sharpened_laplace_result(:,:,i) .* smoothed_sobel_grad(:,:,i);
end
subplot(2, 4, 6);
imshow(product_laplace_sobel);
imwrite(product_laplace_sobel, 'fish_product_laplace_sobel.png');

sharpened_image  = zeros(sz);
for j=1:3
sharpened_image(:,:,j) = max(fig_original(:,:,j) + product_laplace_sobel(:,:,j), 0);
end
subplot(2, 4, 7);
imshow(sharpened_image);
imwrite(sharpened_image, 'fish_sharpened_image.png');

final_result  = zeros(sz);
for k=1:3
final_result(:,:,k) = power(sharpened_image(:,:,k), 0.5);
end
subplot(2, 4, 8);
imshow(final_result);
imwrite(final_result, 'fish_final_result.png');
```

3.

Lapacian 運算子是一種二階微分算子，對於階躍狀邊緣，二階導數在邊緣點有 zero-crossing，邊緣點兩旁的像素二階導數異號。若只考慮邊緣點的位置而不考慮周圍的灰度差時可以用該運算子檢測邊緣，但對雜訊也較敏感。Sobel 運算子是用離散型的差分算子，用來運算圖象亮度函數的梯度的近似值。由我自己的實作過程發現用 Laplacian filter 邊緣偵測的效果較差，若程式沒有寫錯，則我想可能是原本圖片中含有雜訊而導致此結果。