410415069 電機三 陳毅軒 HW2

Super Resolution by Lanczos resampling

目錄

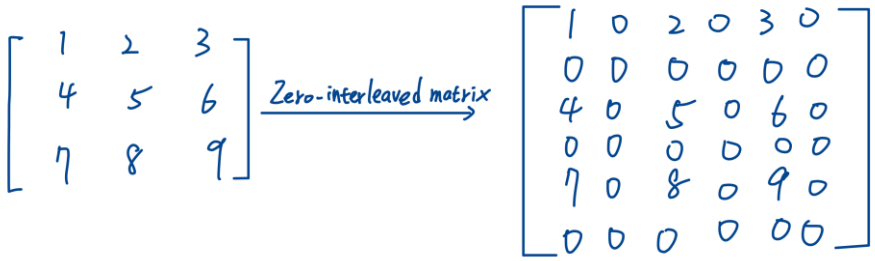
SR=Super Resolution

A. 為什麼要做 Super Resolution

當影像放大時，不論是將單一圖片放大 n 倍，或是局部放大圖片，都會導致影像解析度降低，所以使用 SR 來重建圖片，提高圖片的分辨率。而 SR 也分成 SISR 和 MFSR，SISR 通過處理單個圖像來提高解析度，而 MFSR 利用多個圖像序列來合成高解析度圖像，這次作業將以 SISR 為主，並在 Spatial domain，使用 spatial filter 做影像超解析。

B. 在 Spatial domain 做 SR 的基本原理與步驟

a、 Zero-interleaved matrix

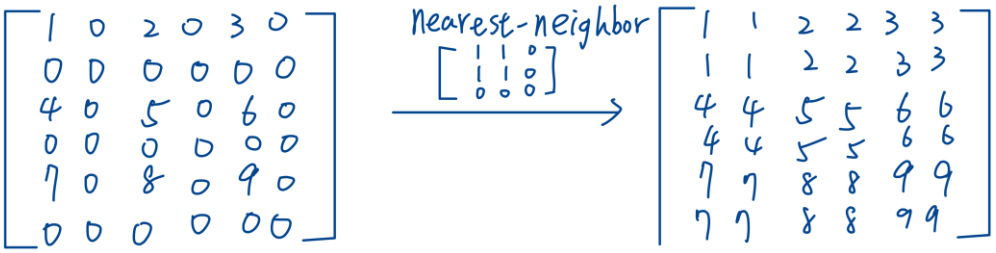


圖一、Zero-interleaved matrix

將矩陣放大兩倍，並在數值跟數值之間填上 0。

b、 Enlargement by spatial filtering

i. Nearest-neighbor=$\begin{bmatrix}1 & 1 & 0\\ 1 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}$



圖二、convolute Nearest-neighbor matrix

使用 imfilter(Zero-interleaved matrix, Nearest-neighbor)，對 matrix 做 convolution，得到放大後的矩陣。

ii. Bilinear=$\begin{bmatrix}\frac{1}{4} & \frac{1}{2} & \frac{1}{2}\\ \frac{1}{2} & 1 & \frac{1}{2}\\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4}\end{bmatrix}$

使用與 Nearest-neighbor matrix 同樣的方法，將 bilinear matrix 與 Zero-

interleaved matrix 做 convolution。

iii. $\text{Bicubic}=\frac{1}{64}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$

將 bicubic matrix 與 Zero-interleaved matrix 做 convolution。

上述三種方法，bicubic 的效果最好，Nearest-neighbor 會使圖像周圍有鋸齒狀的不連續狀況。

C. SR basic solution 遇到的問題

a、 失真和偽影：插值法在圖像中明顯變化的區域引起失真或偽影。會導致插值後的圖像與原始圖像不完全匹配。

b、 邊緣效應： 插值法對邊緣的處理不夠精確，導致圖像邊緣產生不理想的效果。

c、 計算複雜度： 因為是在 Spatial domain 做處理，使用卷積計算，使得計算量較大。

d、 插值算法選擇： 基本的有 nearest-neighbor、bilinear、bicubic，使用的場合不一樣，買次都需要都嘗試才能找到合適的方法。

e、 採樣率不匹配：插值的目標分辨率與原始圖像的採樣率不匹配，導致插值效果不佳。

f、 處理噪聲： 插值方法對圖像中的噪聲可能過於敏感，導致插值後的圖像中出現更多的噪聲。

D. ChatGPT 所提供的解決方法

a、 失真和偽影：選擇更高級的插值方法，如雙三次插值或 Lanczos 插值，以減小失真的影響。

b、 邊緣效應：使用邊緣保持插值算法，以改善在邊緣處理上的表現。或是在插值前對圖像進行邊緣增強或邊緣檢測，以提高邊緣信息的準確性。

c、 計算複雜度：優化現有算法的實現，以降低計算複雜度，或是轉到 frequency domain 做計算。

d、 插值算法選擇：根據狀況選擇最適合的插值算法，在不同狀況使用不同的插值方法以獲得最佳效果。

e、 採樣率不匹配：使用自適應插值方法，根據目標分辨率和原始圖像的採樣率動態調整插值方法和參數。

f、 處理噪聲：在插值前對圖像進行降噪處理，以減少插值過程中對噪聲的敏感性。

SR=Super Resolution

g、 使用 SRCNN 做影像處理。

E. 使用 Lanczos resampling

解決方法中提到使用更高級的內插法，所以將使用 Lanczos resampling，做插值應用。

I. 介紹

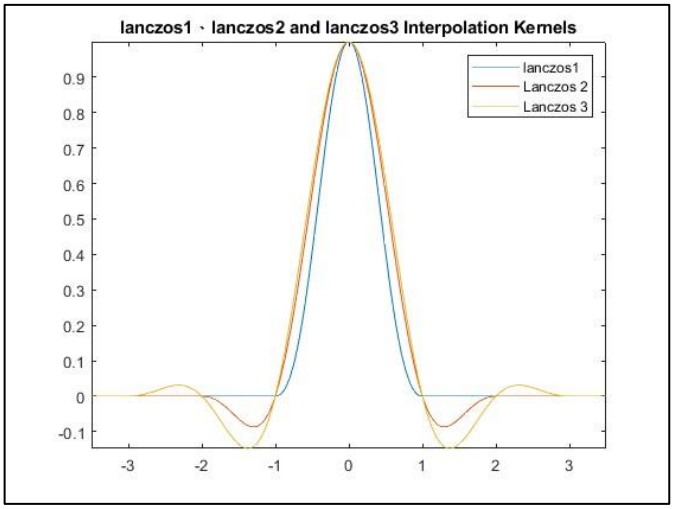
使用 Lanczos resampling 進行濾波。目標是在進行圖像縮小或放大時盡可能地保留圖像的細節和邊緣信息。進行插值計算時對像素進行加權，對一定範圍內的點作加權運算，因使用卷積計算，且每次都要做一定數量的乘法，所以計算速度不快。

- 一維插值

$$L(x) = \begin{cases} 1, & x = 0 \\ \dfrac{\text{asin}(\pi x)\ \sin\left(\frac{\pi x}{a}\right)}{\pi^2 x^2}, & -a \le x < a\ and\ x \neq 0 \\ 0, & otherwise \end{cases} \text{(式一)}$$

$$f(x) = \sum_{i=\lfloor x \rfloor - a+1}^{\lfloor x \rfloor + a} s_i \times L(x-i) \text{(式二)}$$

(式一)是計算權重的公式，si 是已知的數值，在(式一)(式二)中 a 是 Lanczos resampling 的影響半徑，數值越大，會有更多數值進入加權以及更高的比重。



圖三、lanczos1、lanczos2 and lanczos3 Interpolation Kernels

(圖三)顯示不同 a 值所產生的 filter，可以發現當 a=3 時，越接近中心權重越高，但是離越遠甚至有負的權重，可以對邊緣做更有效的處理。

實際計算:

假設$f(0)=2,\ f(1)=0, f(2)=0.5, f(3)=1$, 且$a=2$, 求$f(1.5)$

```
S=[2 0 1.5 1];
lanczos(1.5,S,2)

function L = lanczosResampling(x,a)
    L = a*sin(pi*x) .* sin(pi*x/a) ./ ...
        (pi^2 * x.^2);
    L(abs(x) > a) = 0;
    L(x == 0) = 1;
end

function f = lanczos(x,S,a)
    f = 0.0;
    for i = (floor(x)-a+1):(floor(x)+a)
        f=f+S(clamp(i,4))*lanczosResampling(x-i,a);
    end
end

function c = clamp(x,b)
    if x<=0
        c=1;
    elseif x>b
        c=b;
    else
        c=x;
    end
end
```

圖四、計算程式碼

$$clamp(x,b)=\begin{cases}1, & x\le 0\\ b, & x>b\\ x, & otherwise\end{cases} \quad (式三)$$

lanczosResampling 函式為(式一)；lanczos 函式為(式二)，因為要做累加，使用 for 迴圈來做迭代，因為現在只有一維且數值較少，計算速度快；clamp 函式為限制取值得範圍不超過原本矩陣的範圍避免溢位。所以整個計算過程為

$$f(1.5)=\sum_{i=\lfloor 1.5\rfloor-2+1}^{\lfloor 1.5\rfloor+2} s_i\times L(1.5-i)=0.9234 \text{(式四)}$$

- 二維插值

$$L(x,y)=L(x)\times L(y) \text{(式五)}$$

$$f(x,y)=\sum_{i=\lfloor x\rfloor-a+1}^{\lfloor x\rfloor+a}\ \sum_{j=\lfloor y\rfloor-a+1}^{\lfloor y\rfloor+a} s_{ij}\times L(x-i)\,L(y-j) \text{(式五)}$$

二維計算更為複雜，要迭代兩層，所以需要更多次的計算。

```
function L = lanczosResampling(x,a)
    L = a*sin(pi*x) .* sin(pi*x/a) ./ ...
        (pi^2 * x.^2);
    L(abs(x) > a) = 0;
    L(x == 0) = 1;
end

function f = lanczos(x,y,S,a,high,width)
    f = 0.0;
    for k = (floor(x)-a+1):(floor(x)+a)
        for w = (floor(y)-a+1):(floor(y)+a)
            f=f+S(clamp(k,high),clamp(w,width))*lanczosResampling(x-k,a)*lanczosResampling(y-w,a);
        end
    end
end

function c = clamp(x,b)
    if x<=0
        c=1;
    elseif x>b
        c=b;
    else
        c=x;
    end
end
```

圖五、二維插值實作程式碼

lanczosResampling 和 clamp 與一維插值一樣，lanczos 需要做兩次的迭代，且取值範圍也不能超過所傳入的矩陣大小。

II. 實際步驟

因為 matlab 裡面 imresize 函數有提供 lanczos 的插值法，所以我用我自己寫的插值法與 matlab 函式做比較。

```
big=4;
I=double(imread("HW2-dataset\LR\LR1.jpg"));
ref=imread("HW2-dataset\SR\SR1.jpg");
rs=imresize(I,big,"lanczos3");
[high,width,~]=size(I);
z=zeros(high*big,width*big,3);
for cc = 1:3
    Ic=I(:,:,cc);
    for i = 1:high*big
        for j = 1:width*big
            z(i,j,cc)=lanczos(i/big,j/big,Ic,3,high,width);
        end
    end
end
psnr(uint8(rs),ref)
figure,imshowpair(uint8(rs),ref,"montage")
psnr(uint8(z),ref)
figure,imshowpair(uint8(z),ref,"montage")
```

圖六、對影像作 resize 處理

i. big 設定放大的倍率
ii. 讀入檔案轉換成 double
iii. 使用 imresize(I,big,"lanczos3")，” lanczos3”意思是使用 lanczos 插值法且 a=3

SR=Super Resolution

接著是使用我自己所撰寫的 lanczos 插值法做計算

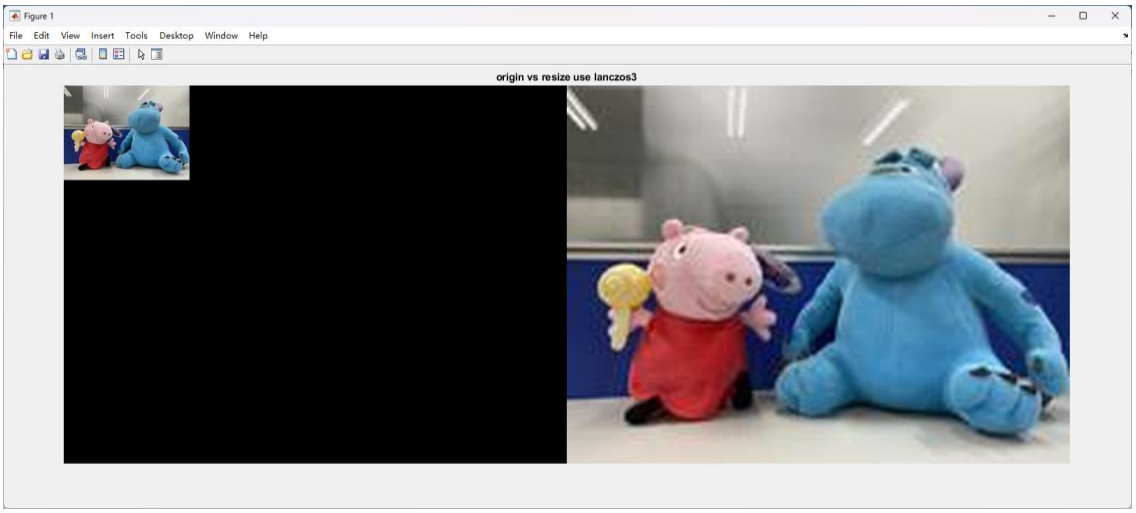
iv. 先把圖像 RGB 分成三次計算

v. 要計算每個點，且每個點都是一次卷積，所以計算非常久

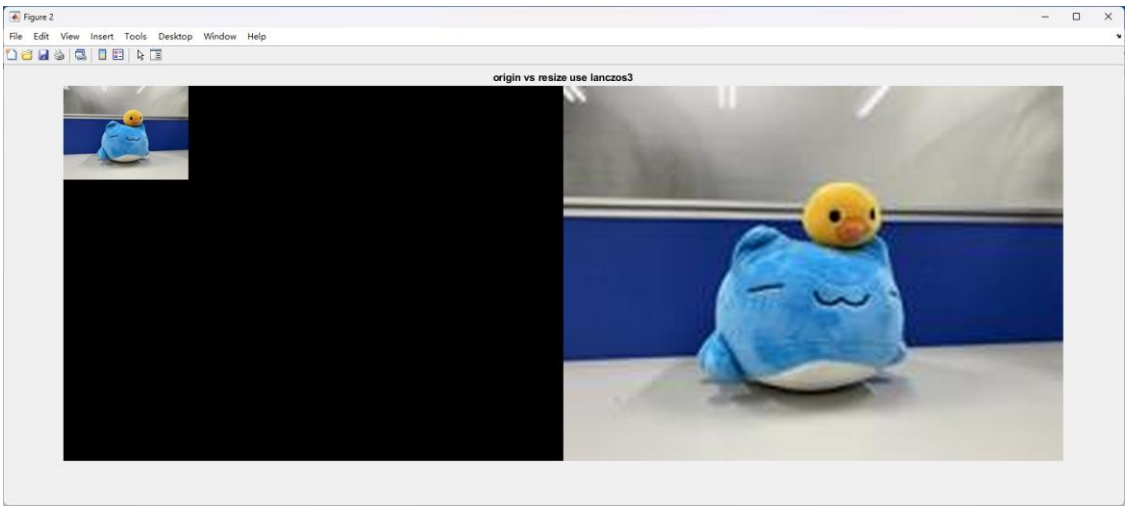
vi. 傳入的 x,y 設定為 i/big 和 j/big 是為了符合原始矩陣的大小，如果沒有除以 big，在 clamp(式三)會導致值永遠是矩陣的最後一個數字。

F. 實作結果比較

I. SR vs imresize use lanczos3

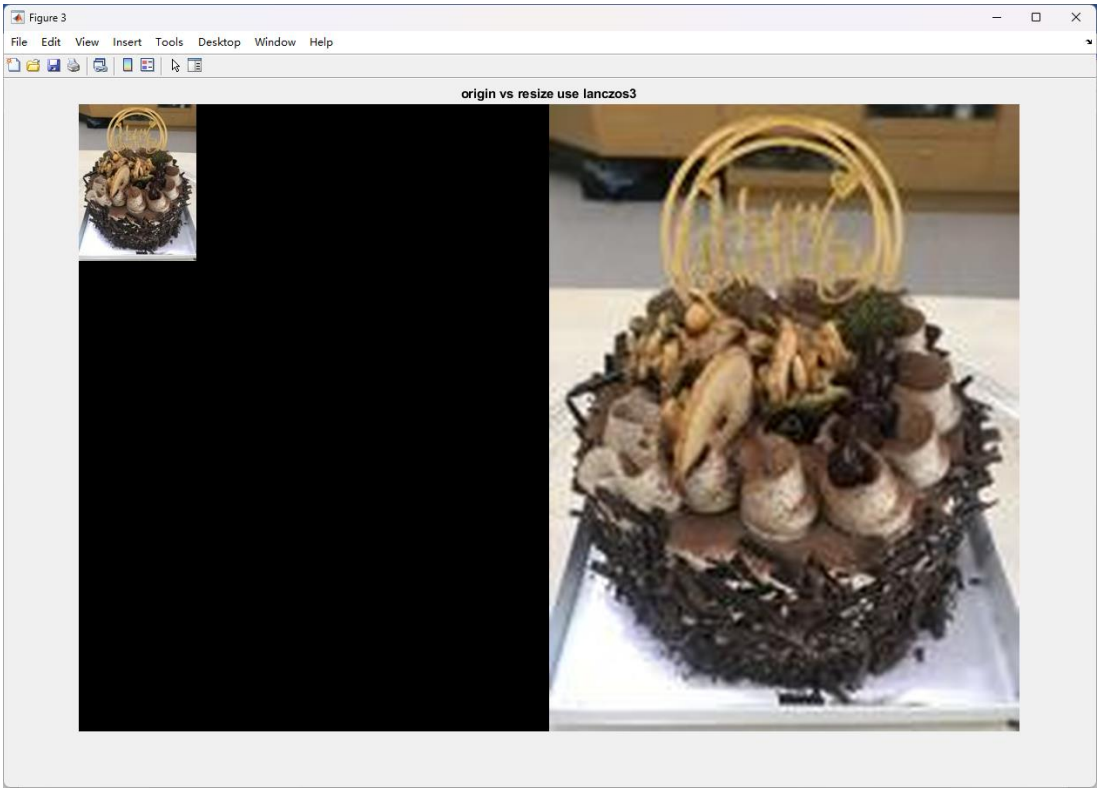


圖七、LR1.jpg 與使用 lanczos3 放大四倍圖片比較

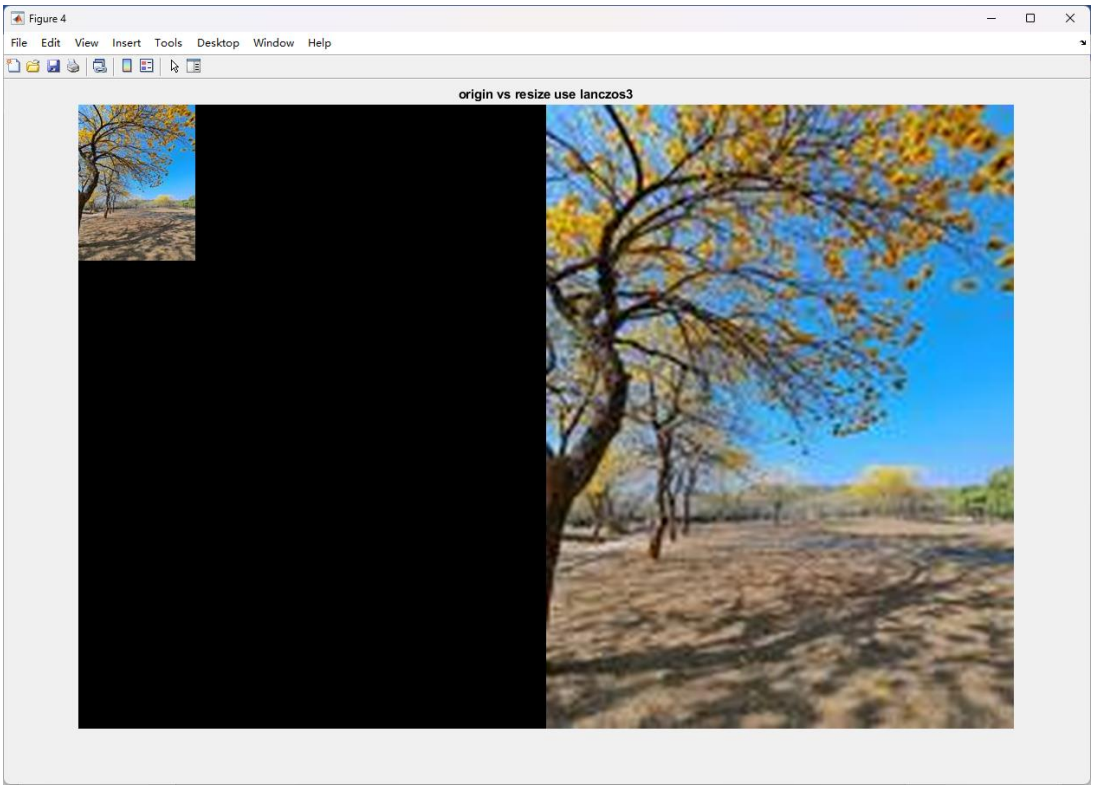


圖八、LR2.jpg 與使用 lanczos3 放大四倍圖片比較

SR=Super Resolution

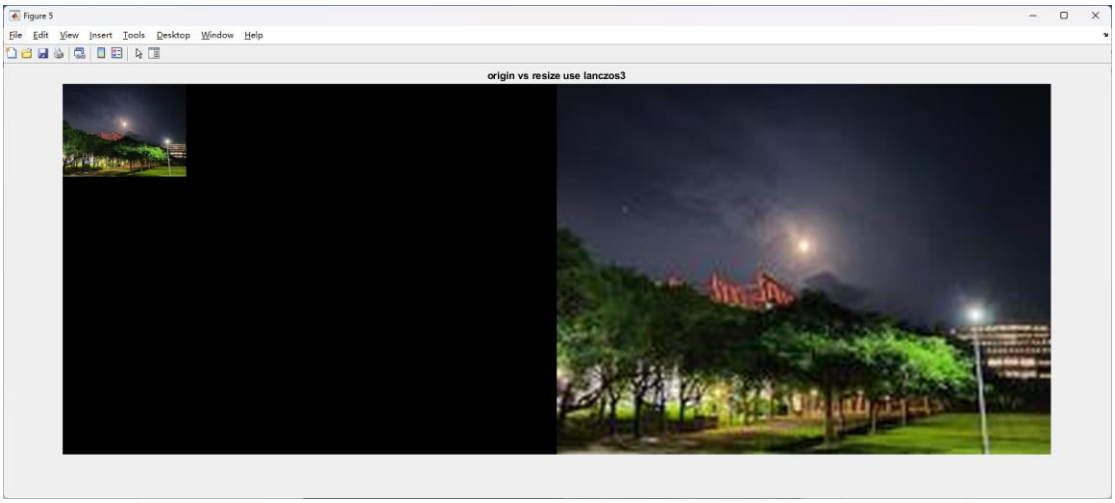


圖九、LR3.jpg 與使用 lanczos3 放大四倍圖片比較



圖十、LR4.jpg 與使用 lanczos3 放大四倍圖片比較

SR=Super Resolution



圖十一、LR5.jpg 與使用 lanczos3 放大四倍圖片比較

Use lanczos3	PSNR	MS-SSIM
LR1	28.318640	0.915253
LR2	30.051820	0.946544
LR3	24.415641	0.797273
LR4	18.736952	0.630278
LR5	22.120137	0.798286

表一、使用 imresize lanczos3 的圖片與 SR 圖片比較的 PSNR 和 MS-SSIM

從(表一)比較 PSNR 的狀況，只有(圖八)LR2 的 PSNR 較好，(圖十)LR4 的 PSNR 還低於 20，從圖片中就可以很明顯的感受到放大後圖片解析明顯變差。比較 MS-SSIM 也是與 PSNR 同樣的狀況，好壞程度差距還是很大。

推測 LR2 的放大狀況較好可能是因為圖片的內容較不複雜，且光影比較均勻，使得放大後，並沒有丟失太多資訊。相較(圖九)~(圖十)LR3~LR5，圖片內明暗明顯，且有更多資訊，所以在使用 lanczos3 放大時，因為需要用周圍的數值做插值，且邊緣需要更多資訊，所以 PSNR 和 MS-SSIM 表現不佳。

II. lanczos3 by myself vs imresize use lanczos3

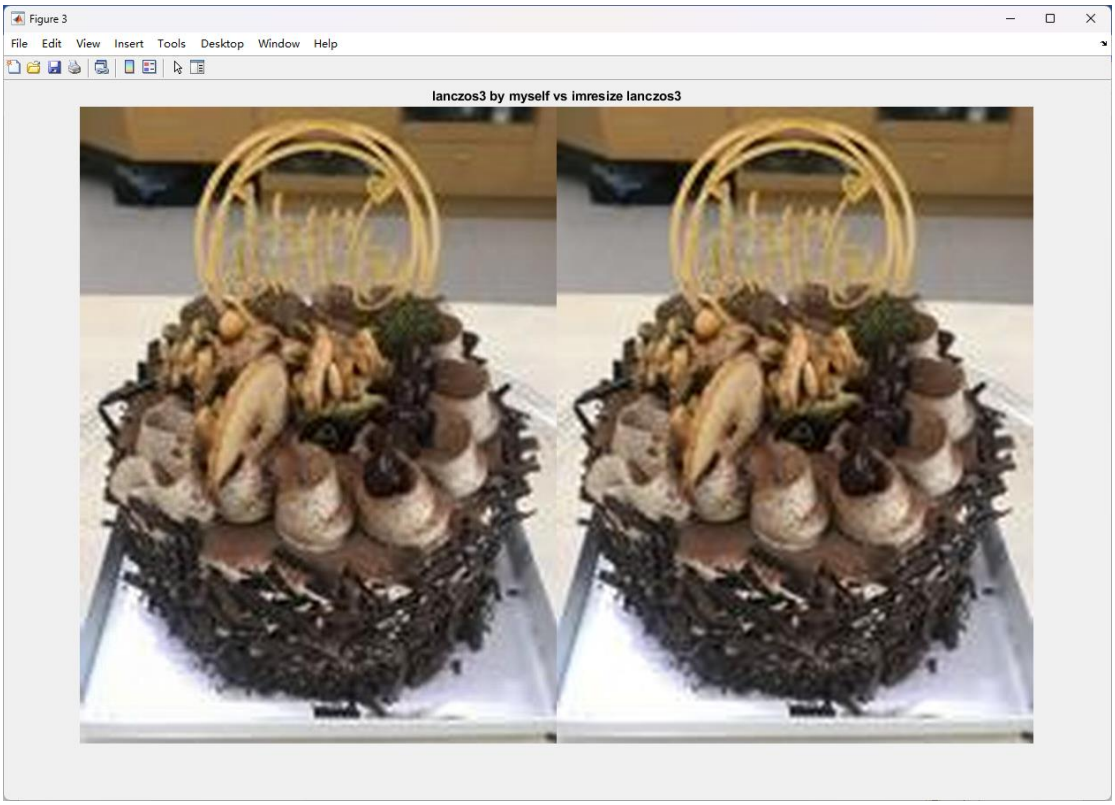
展示 lanczos3 by myself vs imresize use lanczos3 的差異，並且比較 lanczos3 by myself 和 SR 照片的 PSNR 和 MS-SSIM。

SR=Super Resolution

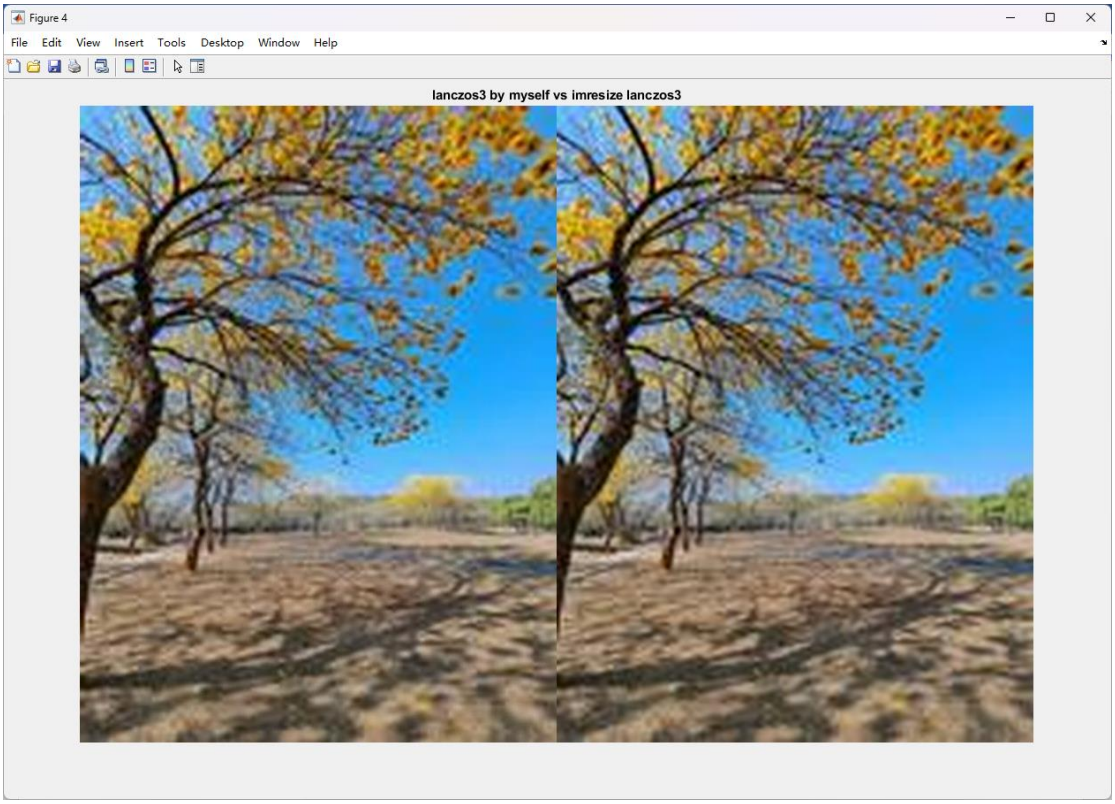
圖十二、LR1 使用 lanczos3 by myself 和 imresize use lanczos3 放大四倍圖片比較

圖十三、LR2 使用 lanczos3 by myself 和 imresize use lanczos3 放大四倍圖片比較

SR=Super Resolution

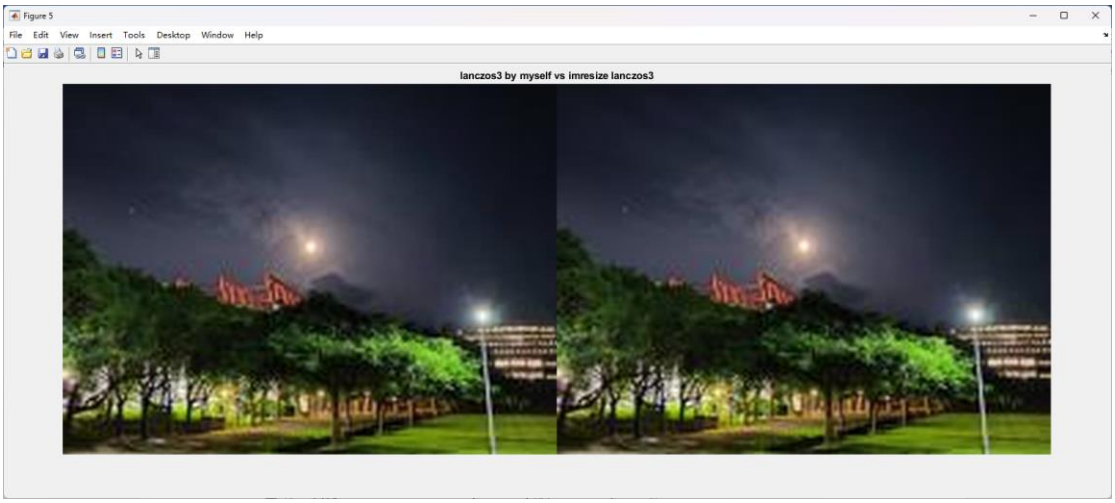


圖十四、LR3 使用 lanczos3 by myself 和 imresize use lanczos3 放大四倍圖片比較



圖十五、LR4 使用 lanczos3 by myself 和 imresize use lanczos3 放大四倍圖片比較

SR=Super Resolution



圖十六、LR5 使用 lanczos3 by myself 和 imresize use lanczos3 放大四倍圖片比較

Origin image	method	PSNR	MS-SSIM
LR1	imresize use lanczos3	28.318640	0.915253
	lanczos3 by myself	25.887340	0.898875
LR2	imresize use lanczos3	30.051820	0.946544
	lanczos3 by myself	26.791545	0.929359
LR3	imresize use lanczos3	24.415641	0.797273
	lanczos3 by myself	22.056984	0.751160
LR4	imresize use lanczos3	18.736952	0.630278
	lanczos3 by myself	17.589202	0.576575
LR5	imresize use lanczos3	22.120137	0.798286
	lanczos3 by myself	20.985052	0.777188

表二、imresize lanczos3 和 lanczos3 by myself 與 SR 圖片比較的 PSNR 和 MS-SSIM

自己所撰寫的 Lanczos resampling 比 matlab 所撰寫的 imresize lanczos3 還要差，在 PSNR 和 MS-SSIM 都更低，不過在撰寫程式的過程實際了解到對於整個影像如何做插值，和如何做卷積運算，有更深入的了解，當自己所撰寫的程式碼，輸出圖片時，雖然結果不如預期，但是在過程中學到很多。

G. Summarize

這次作業實作 SR 是一項挑戰，從公布作業到撰寫報告，中間不斷嘗試不同方法，都不盡人意，找到使用 Lanczos resampling 後開始實作，第一關就遇到如何了解函式內部所做的運算，其實只要用 matlab 所提供的函式就可以放大圖片了，但是為了真正學習到東西，自己實作了計算過程，雖然結果並不好，但在撰寫和閱讀許多資料後，獲得更多，希望這些知識和學習熱情可以繼續下去。

SR=Super Resolution

H. Experience

Challenge

這次作業較為困難，因為在搜尋時，大部分的資訊都是 SRCNN 的內容，但是這次作業沒有可以訓練的圖片檔案，所以使用 SRCNN 會有障礙難以突破，最後選用 Lanczos resampling 作為 SR 的 improve solution。另一個困難點是我嘗試自己寫 Lanczos interpolation 但是效果卻不如預期，改了很久還是不知道問題在哪裡。

Cooperation

這次作業的相關資料真的不是很好找，和同學反覆嘗試許多可能的方法，都以失敗告終，也一起討論了很多種自創方法，但都難以實現 SR，最後查找到 Lanczos resampling 作為作業的實作方向。

I. Feedback of using AI guidance

使用 ChatGPT，詢問 SR 的方法，大部分一值回答 SRCNN，再繼續追問下去，也只提供了 basic 的插值法，所以這次題目是從 matlab 的網站查找到的。雖然在找資料的地方 AI 並沒有幫上忙，但詢問缺點和改善方法，提供給我很多種狀況的處理方法，也詳細介紹了 Lanczos resampling 的作法。
在程式碼方面，當遇到不會撰寫的部分，可以詢問 AI，它會提供簡單的範例與解釋，讓我在自學上更方便，並且能夠糾正程式碼裡面的問題，或是在撰寫時，提供建議的寫法。
未來在使用 AI 方面，除了 chatgpt，github copilot 也是一個很強大的撰寫程式碼的機器人，他可以在 IDE 裡面給予很大的幫助，除了可以更快撰寫外，當我們突然當機時，也能及時提供想法，幫助撰寫程式上更順利。

J. Reference

https://ww2.mathworks.cn/help/matlab/ref/imresize.html?searchHighlight=imresize&s_tid=srchtitle_support_results_1_imresize
https://ww2.mathworks.cn/help/matlab/creating_plots/create-and-compare-resizing-interpolation-kernels.html
https://en.wikipedia.org/wiki/Lanczos_resampling
https://www.twblogs.net/a/5c9f5416bd9eee73ef4b8563

SR=Super Resolution

https://blog.csdn.net/wgx571859177/article/details/78963267
https://www.youtube.com/watch?v=ijmd6XyG2HA&t=198s&ab_channel=EdgarProgrammator
chatgpt

SR=Super Resolution