

Homework #3

第一點

104062241 莫雅雯

1. full search

(1)做法(截圖只是 code 的一部分)

讀進 reference image 和 target image 後，設定 N(macroblock size)和 p(search range)，

傳入 full_search.m 進行 full_search，最後回傳 predated image。

full_search.m 檔

<pre>if (rem(i,N)==1 && rem(j,N)==1) si=i-p; sj=j-p; ei=i+p; ej=j+p; if(si<1) si=1; end if(sj<1) sj=1; end if(ei>s(1)) ei=s(1); end if(ej>s(2)) ej=s(2); end</pre>	<p>根據設定的 N(macroblock size)，從 reference image 找出一塊與 target image 切好後的一塊最相似的。</p> <p>(1)先找好範圍： p 為 search range target image 的一塊的左上的座標(+)(-)p 得到一個搜尋範圍 (要注意範圍不能超過原本 image 的範圍)</p>
<pre>for kl=si:ei for k2=sj:ej if(kl+N-1<s(1)+1 && k2+N-1<s(2)+1) SAD_value_r= SAD(Target(i:i+N-1,j:j+N-1,1),Reference(kl:kl+N-1,k2:k2+N-1,1)); SAD_value_g= SAD(Target(i:i+N-1,j:j+N-1,2),Reference(kl:kl+N-1,k2:k2+N-1,2)); SAD_value_b= SAD(Target(i:i+N-1,j:j+N-1,3),Reference(kl:kl+N-1,k2:k2+N-1,3)); SAD_value=SAD_value_r+SAD_value_g+SAD_value_b;</pre>	<p>(1)從搜尋範圍找一塊跟 target image 切好後的一塊最相似的。</p> <p>(2)用 for loop 掃過整個搜尋範圍 算出 SAD(利用寫好的 SAD.m)。</p> <p>(3) if(kl+N-1<s(1)+1 && k2+N-1<s(2)+1) 這行是為了確保不超出原 image 範圍。</p>
<pre>if(SAD_value<fsad) fsad=SAD_value; findi=kl; findj=k2;</pre>	<p>(1)fsad 用來記錄目前最小的 SAD 值，</p> <p>(2)findi、findj 用來記錄目前最小 SAD 值所在的區塊的左上角做標。</p> <p>(3)for loop 搜尋過程中如果算出來的 SAD 比目前最小的(fsad)還要小，就把 fsad、findi、findj 代換掉。</p>
<pre>MV(floor(i/N)+1,floor(j/N)+1,1)=findi; MV(floor(i/N)+1,floor(j/N)+1,2)=findj; image(i:i+N-1,j:j+N-1,1)=Reference(findi:findi+N-1,findj:findj+N-1,1); image(i:i+N-1,j:j+N-1,2)=Reference(findi:findi+N-1,findj:findj+N-1,2); image(i:i+N-1,j:j+N-1,3)=Reference(findi:findi+N-1,findj:findj+N-1,3);</pre>	<p>搜尋完後找到 findi、findj 存入 MV 裡面(為了之後印出 vector) 把 image(即 predated image)設成從 reference image 找到的最相似的區塊。</p>
重複上面步驟直到整個 predated image 建好。	

(2)計算 SAD

SAD.m 檔

<pre>function SAD_value= SAD(Target,Reference) SAD_value = sum(abs(Target(:) - Reference(:))); end</pre>	<p>傳入兩個矩陣，代入講義 43 頁 SAD 的公式，回傳算好的值。</p>
--	---

(3)計算 SAD 並 plot 出來

HW3_main.m 檔

<pre> %full f_8_8= SAD(target,predicted_image_8_8); f_16_8= SAD(target,predicted_image_16_8); f_8_16= SAD(target,predicted_image_8_16); f_16_16= SAD(target,predicted_image_16_16); %2D d_8_8= SAD(target,two_d_8_8); d_16_8= SAD(target,two_d_16_8); d_8_16= SAD(target,two_d_8_16); d_16_16= SAD(target,two_d_16_16); SAD_full = [f_8_8,f_16_8,f_8_16,f_16_16]; SAD_2D = [d_8_8,d_16_8,d_8_16,d_16_16]; x = [1 2 3 4]; plot(x, SAD_full, x, SAD_2D); x = [1 2 3 4]; legend('full', '2D'); xticks(x); xticks(x); xticklabels({'8, 8x8', '8, 16x16', '16, 8x8', '16, 16x16'}); title('SAD'); </pre>	<p>將 target image 跟 predicted image 傳入 SAD.m 算出 SAD 值，最後用助教給的範例方法印出來。</p>
---	---

(4)計算 PSNR 並 plot 出來

<pre> target=im2uint8(target); predicted_image_8_8=im2uint8(predicted_image_8_8); mseR=(double(target(:,:,1))-double(predicted_image_8_8(:,:,1))).^2; mseG=(double(target(:,:,2))-double(predicted_image_8_8(:,:,2))).^2; mseB=(double(target(:,:,3))-double(predicted_image_8_8(:,:,3))).^2; mR=sum(sum(mseR))/(s(1)*s(2)); mG=sum(sum(mseG))/(s(1)*s(2)); mB=sum(sum(mseB))/(s(1)*s(2)); mse=(mR+mG+mB)/3; PSNR_full_8_8=10*log10(255^2/mse); P_full_8_8=psnr(target,predicted_image_8_8); </pre>	<p>(1) 用 uint8 的圖去直接用 double()轉成 double， (2) 各自算三個 channel 的 MSE 再取平均。 (3)用 psnr 公式只是確認我的運算式有沒有一樣，印出的還是自己運算的 PSNR。</p>
---	--

2. 2D logarithmic search method

(1)做法(截圖只是 code 的一部分)

讀進 reference image 和 target image 後，設定 N(macroblock size)和 p(search range)，傳入 twoD_logarithmic_search.m 進行 2D logarithmic search，最後回傳 predicted image。

twoD_logarithmic_search.m 檔

<pre> findi=i; findj=j; n_temp=floor(log2(p)); n=max(2,2^(n_temp-1)); </pre>	<p>(1)將 findi、findj 預設成 i、j(簡稱中間點) findi、findj 是目前找到 SAD 最小的區塊的左上角座標。 (2)找出 n(即中間點要+)(-n 得到的點)</p>
<pre> if (rem(i,N)==1 && rem(j,N)==1) si=i-n; sj=j-n; ei=i+n; ej=j+n; ci=findi; cj=findj; if(si<1) si=1; end if(sj<1) sj=1; end if(ei>s(1)) ei=s(1); end if(ej>s(2)) ej=s(2); end </pre>	<p>中點+)(-n 後，得到的 4 個點跟中點要找出分別區塊 SAD 最小的。 (這裡稱的點都代表區塊的左上角的那一點) (要注意範圍不能超過原本 image 的範圍)</p>

<pre> while(1) if(si+N-1<s(1)+1 && cj+N-1<s(2)+1) SAD_value_l_r= SAD(Target(i:i+N-1,j:j+N-1,1),Reference(si:si+N-1,cj:cj+N-1,1)); SAD_value_l_g= SAD(Target(i:i+N-1,j:j+N-1,2),Reference(si:si+N-1,cj:cj+N-1,2)); SAD_value_l_b= SAD(Target(i:i+N-1,j:j+N-1,3),Reference(si:si+N-1,cj:cj+N-1,3)); SAD_value_l = SAD_value_l_r+SAD_value_l_g+SAD_value_l_b; if(fsad==0) fsad=SAD_value_l; findi=si; findj=cj; else if(SAD_value_l<=fsad) fsad=SAD_value_l; findi=si; findj=cj; end end end% end of if(si+N-1<s(1)+1 && cj+N-1<s(2)+1) </pre>	<p>(1)大概概念:用 while 迴圈持續去找點 →比較分別區塊 SAD →如果新找到點的區塊的 SAD 比目前最小的,就代換掉,而且若 5 個點代表的區塊當中 SAD 最小的仍是中間點,就要把 n 除以 2 →直到 n=1 時,最後只要從中間點和其周圍的 8 個點中找到最小 SAD 的那塊就可以跳出 while (2)左邊那段 code 只列出中間點左邊部分(即中間點 i-n)計算其 SAD(利用寫好的 SAD.m)並跟目前最小的 SAD(這裡用 fsad 代表)比,若 <=fsad,就把 fsad、findi、findj 代換掉。 (3) if(si+N-1<s(1)+1 && cj+N-1<s(2)+1) 這行是為了確保不超出原 image 範圍。 (4)還有中間點右邊、上、下的部分,跟左邊那塊差不多,就不截圖站版面了。</p>
<pre> if(SAD_value<=fsad) fsad=SAD_value; findi=ci; findj=cj; n=floor(n/2); if(n==1) [fsad,findi,findj]=endFind(Target,Reference,N,i,j,ci,cj,n); break; end si=findi-n; sj=findj-n; ei=findi+n; ej=findj+n; ci=findi; cj=findj; if(si<1) si=1; end if(sj<1) sj=1; end if(ei>s(1)) ei=s(1); end if(ej>s(2)) ej=s(2); end </pre>	<p>(1)左邊那段 code 列出中間點部分,計算其 SAD(利用寫好的 SAD.m)並跟目前最小的 SAD(這裡用 fsad 代表)比,若 <=fsad,就把 fsad、findi、findj 代換掉。 且 n 要除以 2,並且 n=1 時,最後只要從中間點和其周圍的 8 個點中找到最小 SAD 的那塊就可以用 break 跳出 while。 (最後比較部分寫再 endFind.m 直接呼叫即可) (2)若仍要繼續比則要更新 si、sj、ei、ej、ci、cj 值。 (要注意範圍不能超過原本 image 的範圍) (3)如果中間點 SAD 大於目前最小 SAD 值,仍須更新 si、sj、ei、ej、ci、cj,只是 n 不變。(見 code else 之後的部分)</p>

<pre> else si=findi-n; sj=findj-n; ei=findi+n; ej=findj+n; ci=findi; cj=findj; if(si<1) si=1; end if(sj<1) sj=1; end if(ei>s(1)) ei=s(1); end if(ej>s(2)) ej=s(2); end end %end of else </pre>	
<pre> MV(floor(i/N)+1,floor(j/N)+1,1)=findi; MV(floor(i/N)+1,floor(j/N)+1,2)=findj; image(i:i+N-1,j:j+N-1,1)=Reference(findi:findi+N-1,findj:findj+N-1,1); image(i:i+N-1,j:j+N-1,2)=Reference(findi:findi+N-1,findj:findj+N-1,2); image(i:i+N-1,j:j+N-1,3)=Reference(findi:findi+N-1,findj:findj+N-1,3); </pre>	<p>搜尋完後找到 findi、findj 存入 MV 裡面(為了之後印出 vector) 把 image(即 predicted image)設成從 reference image 找到的最相似的區塊。</p>



endFind.m 檔







<pre> function [fsad,fi,fj]= endFind(Target,Reference,N,i,j,findi,findj,n) fsad=0; s=size(Target); fi=findi; fj=findj; %左上 si=findi-n; sj=findj-n; ei=findi+n; ej=findj+n; if(si<1) si=1; end if(sj<1) sj=1; end if(ei>s(1)) ei=s(1); end if(ej>s(2)) ej=s(2); </pre>	<p>(1)傳入 Target image、Reference image、 N(macroblock size)、中間點(findi、findj)，以及 n (2)對中間點以及其周圍的 8 個點圍左上角的區塊 比較 SAD 值，找出 SAD 最小的區塊回傳其值及左 上角做標回去。 (2)做法跟前面的差不多，也是用 fsad、fi、fj 去記 錄目前 SAD 最小區塊的資訊，這裡只列一部分。</p>
---	---






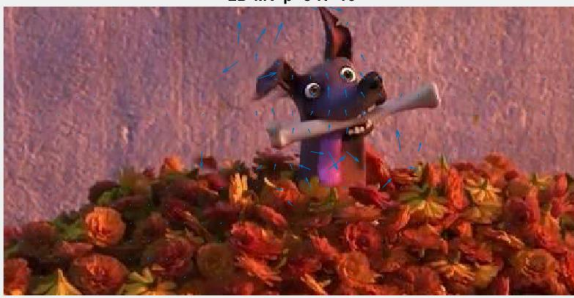




(2)計算 SAD 跟 PSNR 部分跟 1.一樣


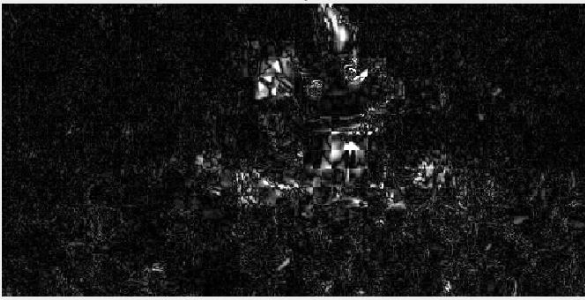
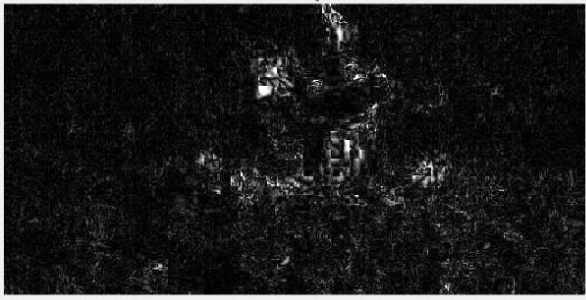



3.結果

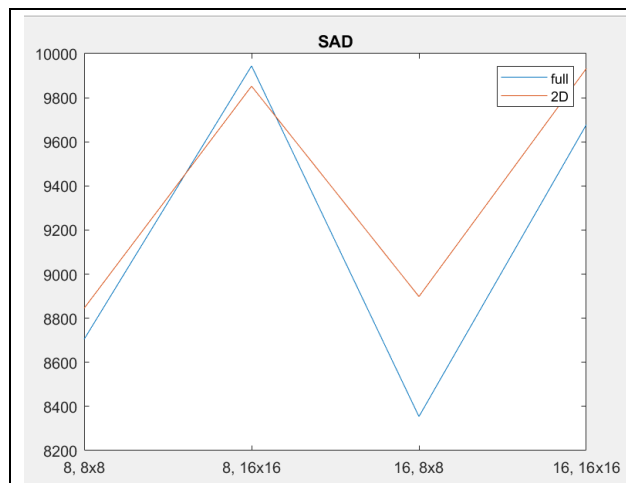
圖片

	N=8，p=8	N=16，p=8
<p>full search predicted images</p>	<p>full-predicted-p=8-N=8</p> 	<p>full-predicted-p=8-N=16</p> 

	N=8 , p=16	N=16 , p=16
	full-predicted-p=16-N=8 	full-predicted-p=16-N=16 
	PSNR (N=8 , p=8→28.7054) (N=16 , p=8→27.1772) (N=8 , p=16→29.5829) (N=16 , p=16→27.6819) PSNR 值越大代表失真越少。理論上 N=8 , p=16 那組應是最大的，因為他搜尋範圍大且 block 又切得比較小。相對地，N=16 , p=8 應是最小的。而實際上也是 N=8 , p=16 那組 PSNR 最大，N=16 , p=8 那組最小。	
2D_logarithmic_search predicted images	N=8 , p=8	N=16 , p=8
	2D-predicted-p=8-N=8 	2D-predicted-p=8-N=16 
	N=8 , p=16	N=16 , p=16
	2D-predicted-p=16-N=8 	2D-predicted-p=16-N=16 
(1)PSNR (N=8 , p=8→28.5912) (N=16 , p=8→27.3856) (N=8 , p=16→28.3626) (N=16 , p=16→27.2275) PSNR 值越大代表失真越少。理論上 N=8 , p=16 那組應是最大的，因為他搜尋範圍大且 block 又切得比較小。相對地，N=16 , p=8 應是最小的。但實際上是 N=8 , p=8 那組最大，N=16 , p=16 那組最小，可能是因為使用 2D_logarithmic_search 的話有一些誤差。 (2)2D_logarithmic_search 的 4 張圖跟 full search 得比較明顯比較有些地方比較模糊，full search 的做法能得到比較接近 target 的圖。		
full	N=8 , p=8	N=16 , p=8

search motion vectors images	full-MV-p=8-N=8	
		
	N=8 , p=16	N=16 , p=16
	full-MV-p=16-N=8	
		
	motion vectors images 顯示出找到的點跟 reference image 的點相量形成的箭頭。	
2D_logarithmic_search motion vectors images	N=8 , p=8	
		
	N=8 , p=16	N=16 , p=16
	2D-MV-p=16-N=8	
		
	motion vectors images 顯示出找到的點跟 reference image 的點相量形成的箭頭。	
full search residual images	N=8 , p=8	
		

	N=8 , p=16	N=16 , p=16
	full-residual-p=16-N=8 	full-residual-p=16-N=16 
	<p>residual images 如果白色部分越多代表誤差越大。雖然 4 張圖看起來白色部分都差不多，但 N=8 , p=16 那組白色部分明顯較其他少，且他的 PSNR 值也最大，所以他誤差應是最小。</p> <p>PSNR (PSNR 值越大代表失真越少。)</p> <p>(N=8 , p=8→28.7054) (N=16 , p=8→27.1772) (N=8 , p=16→29.5829) (N=16 , p=16→27.6819)</p>	
2D_logarithmic_search residual images	N=8 , p=8	N=16 , p=8
	2D-residual-p=8-N=8 	2D-residual-p=8-N=16 
	N=8 , p=16	N=16 , p=16
	2D-residual-p=16-N=8 	2D-residual-p=16-N=16 
	<p>residual images 如果白色部分越多代表誤差越大。雖然 4 張圖看起來白色部分都差不多，但 N=8 , p=8 那組白色部分明顯較其他少，且他的 PSNR 值也最大，所以他誤差應是最小。</p> <p>PSNR (PSNR 值越大代表失真越少。)</p> <p>(N=8 , p=8→28.5912) (N=16 , p=8→27.3856) (N=8 , p=16→28.3626) (N=16 , p=16→27.2275)</p>	



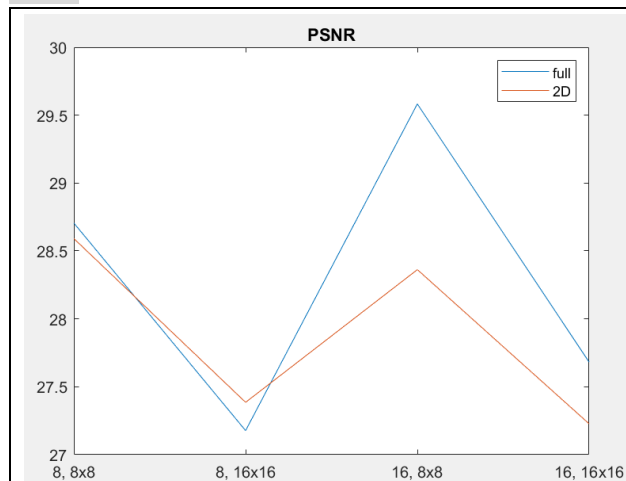
SAD 越小代表失真越少。

(1)full search 部分，(p=16，8*8)那組最小，(p=8，16*16)那組最大。

(2)2D_logarithmic_search 部分，(p=8，8*8)和 (p=16，8*8)差不多但比(p=8，16*16)和(p=16，16*16)小。

(3)整體看來除了(p=8，16*16)外，full search 的 SAD 都比 2D_logarithmic_search 小，因為 full search 得到的圖片誤差較小。

PSNR



PSNR 越大代表失真越少。

(1)full search 部分，(p=8，16*16)那組最小，(p=16，8*8)那組最大。所以(p=8，16*16)較精確。

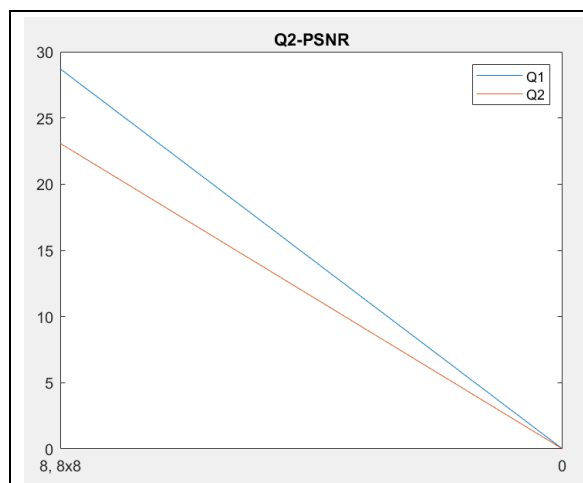
(2)2D_logarithmic_search 部分，(p=16，16*16)最小，(p=8，8*8)最大，可能是因為使用 2D_logarithmic_search 的話有一些誤差。

(3)整體看來除了(p=8，16*16)外，full search 的 SAD 都比 2D_logarithmic_search 大，因為 full search 得到的圖片誤差較小。

第二點

實做部份前面都寫了，這裡只比較 PSNR 的結果

Q1:reference	Q2:reference	target



full search，p=8，8*8

PSNR 越大代表失真越少。

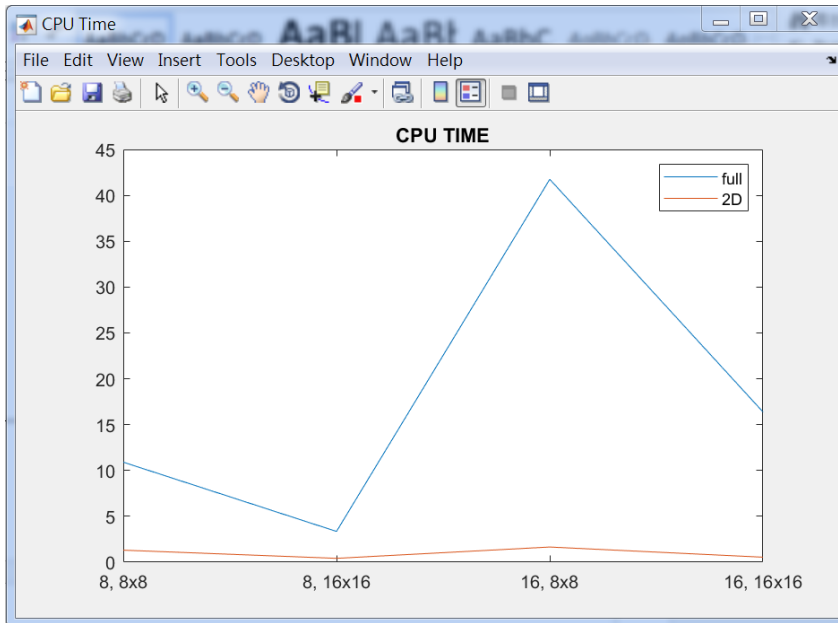
Q1: PSNR=28.7054

Q2:PSNR=23.0774

Q2 的 reference image 跟 target image 比從肉眼看就可以看出差異很大，所以得到的 PSNR 值也會比第一題小。

第三點

a. Measure the execution time required for the two search algorithms with the two different search range sizes ($p=8$ and $p=16$).



b. Compare and discuss the execution time with the theoretical time complexity.

因為理論上 2D_logarithmic_search 是跳著找點算 SAD 所以理論上會比 full search 快。

如上圖，full search 部分：N=8,p=16 搜尋範圍最大所以執行時間最久，N=16,p=8 搜尋範圍最小所以執行時間最快。

整體來看 full search 都比 2D_logarithmic_search 花的時間還要久，所以符合理論上 2D_logarithmic_search 較快。