ADIP 2022 專題報告

黃俊傑、吳翊睿、吳佩霖
學號：111318093　實驗室：Lab414　指導教授：張陽郎
學號：111C52005　實驗室：Lab414　指導教授：張陽郎
學號：111C52016　實驗室：Lab414　指導教授：張陽郎

摘要 — 本 ADIP2022 期末專題題目為「Video sorting」，使用 C++內之 OpenCV 相關功能實現本期末專題之目標。本期末專題共分三大階段，第一階段先將影像進行前處理相對應之選擇，將影像的輪廓強化以利於後續排序，第二階段將前面所得到的結果進行雜湊值計算，並個別兩兩進行比較，進而得出所有圖片互相之間的關聯性，再以關聯性為基準進行排序，第三階段將排序出來的結果編排成影片，並計算所有過程花費時間與評估指標，得出所需數據與最終成果；程式都以平行處理進行，以提升執行效率，更有效的利用電腦硬體。

簡介

本專題共分三階段，第一階段影像前處理，對所有影像進行一套預設的前處理方，若有需求也可以針對不同影像資料，以手動的方式調整處理方案，擁有的前處理選項個別為：blur 初步處理雜訊問題、sharp 將圖片進行銳化、增加對比度、值方圖均衡化…等；完成影像前處理調整後即可進入下一步驟。

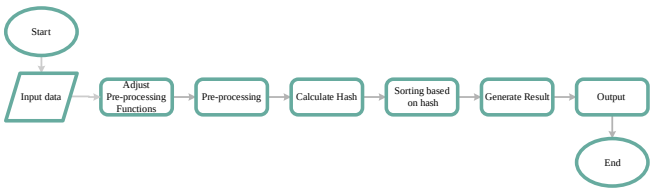
第二階段雜湊函式計算，將影像進行雜湊值計算，我們先將所有影像進行灰階與自適應函數二值化處理，以利於後續的計算，再以 Marr-Hildreth Hash 進行雜湊值計算，Marr-Hildreth Hash 為 OpenCV 中延伸模組，而 Marr-Hildreth Hash 是以影像的輪廓為主的雜湊函式，能有效地排除不需要的資訊讓排序可以更加不受到雜訊干擾。

第三階段輸出成果，把最終排序出來的順序，使用經過前處理後完成的影像輸出成影片，並拿最終結果計算出兩種評估指標，分別為 SRCC 和 MSE，得出所有結果。

為了優化執行時長，本專題對上述所有步驟最後進行了平行處理，將要處理的任務都先提交到 Thread Pool 中，待 Thread Pool 完成所有任務後再執行剩下的程式，即可省下大量時間。

提出方法

本專題研究流程如圖一所示，共分成三個階段，第一階段影像前處理，第二階段雜湊值計算，第三階段輸出成果。



圖一：專案流程圖

1. 第一階段 – 圖片前處理

由於所提供的測試資料有部分為含有雜訊或其他通道錯置等問題，因此需要對影像先進行所需的處理，在此處我們對於所有影像先預設一個通用的處理方式，若調整的需求可以在使用者介面進行手動調整。

i. 預設前處理

中間值濾波器的原理是將各像素的數值替換成鄰近像素的中間值，使用中間值濾波器可以以有效的移除胡椒鹽雜訊，但如同圖所示，使用中間值濾波器雖然移除了雜訊，但也造成了圖片銳度下降，如果將中間值濾波器修改成只在極值處套用的話可有效的降低雜訊，相較於整張圖套用中間值濾波器，只在極值處套用濾波的影像銳度較高。

圖二：預設前處理　(a) 原圖；（b）直接套用中值濾波效果圖；（c）僅對極值進行處理效果圖

ii. 平均濾波

對於雜訊問題進行第一步的處理我們選擇了平均濾波對雜訊進行模糊化，使雜訊不那麼的明顯。

它只是簡單地計算 kernel 裡所有 pixel 的平均值，並將該平均值取代 kernel 中心元素，一樣可以在前端進行手動調整。

iii. 雙邊濾波器

也是針對雜訊進行模糊化處理的方式之一，濾波器的作用是將信號中的特定頻率或頻段濾除或加強。雙邊濾波器的名字源於它能夠同時對信號的高頻和低頻進行濾波。比起上一個平均濾波器，更能保留影像的線條。有兩個參數可以調整，一是以距離決定權重的高斯濾波器的式子，然後加上後半部以像素色差決定權重的式子。

iv. 銳化

設計一 kernel 為：

0	-1	0
-1	5	-1
0	-1	0

圖三：銳化 kernel

對目標圖片進行銳化處理，適用在圖片因為模糊失去過多細節時。

v. 對比度

可調整 alpha 值進行運算，當 alpha 值越高時對比度越高，適用在當圖片對比過低，導致特徵不易被萃取出的狀況。

vi. 限制對比度自適應直方圖均衡化

限制對比度自適應直方圖均衡化是指在進行自適應直方圖均衡化時，通過設置一個對比度限制來防止圖像對比度過高的情況。

這種方法通常用於處理圖像對比度過低的情況，以便在保留圖像的細節的同時，提高圖像的對比度。限制對比度自適應直方圖均衡化的方法是，在進行自適應直方圖均衡化之前，首先計算圖像的最大對比度。然後，計算圖像的自適應直方圖均衡化之後的對比度。如果自適應直方圖均衡化之後的對比度大於最大對比度，則限制自適應直方圖均衡化之後的對比度為最大對比度。

通過限制對比度自適應直方圖均衡化，可以避免圖像對比度過高，使圖像看起來更自然。同時，這也可以有效地提高圖像的對比度，使圖像看起來更清晰。

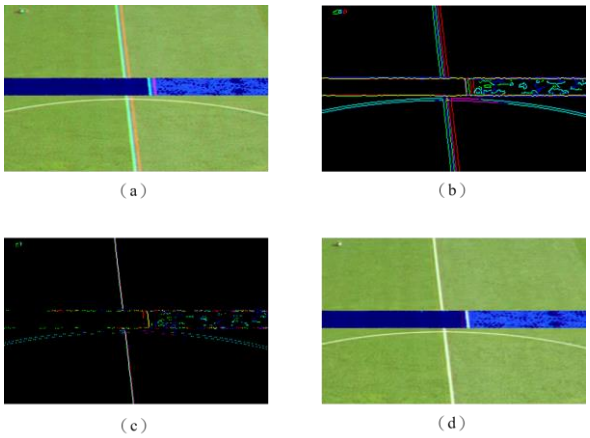
vii. 高斯模糊

使用高斯函數來平滑圖像並去除雜訊。通過將每個像素的灰度值與周圍像素的灰度值進行加權平均來實現。加權平均意味著較靠近中心像素的像素具有更高的權重，而較遠的像素具有較低的權重。

viii. Glitch

針對於有色板未對齊以及有橫向錯位的圖片，如 soccer_style 此作為範例，可以發現於本張圖片中干擾由三種元素所構成:隨機白色方塊、隨機橫條錯位、未對其色板所組成，由於沒有找出較佳的切割白色方塊的方法，所以在此針對隨機橫條錯位、未對其色板做處理。

針對色板未對齊的部分，主要的處理方式是於各色板尋找邊緣，並嘗試將其對齊，執行方法是先將各色板進行高斯模糊後使用 canny 邊緣偵測，再將紅、藍兩色色板嘗試與綠色色板對齊，對其的方法是紅、藍兩色色板進行左右平移，並將兩色色板單獨對綠色色板進行 element-wise AND，並將結果進行加總，加總最大值為其對齊的位置。



圖四：色板對齊 (a) 原圖；（b）各色板邊緣影像；（c）各色板對齊結果；（d）色板對齊結果圖

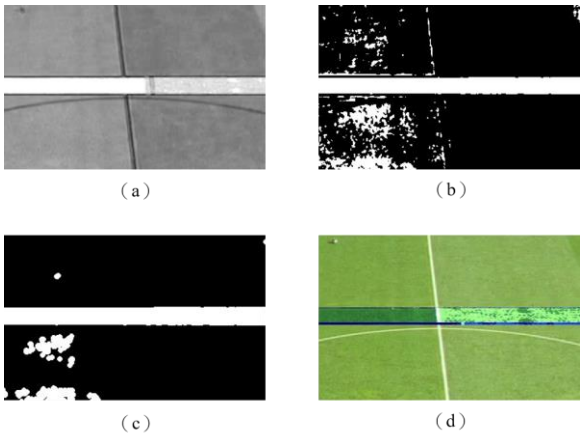
橫向錯位的處理方式為先將圖片轉換成 HSV 色域，由於發現橫條紋對比度特別高，所以針對 S 通道做二質化，具體作法是保留 S 通道最亮的 15%為白色，再將結果經過結構元素為 kernel 2 的侵蝕以及結構元素為 kernel 1 的膨脹後即可得到相對完整的橫條紋擴，最後再將橫向白色占比>90%的區域視為橫向錯位區域，再找出區域後就可以如同色板一樣對齊，於對其之後再調整其色相、飽和度、亮度使其與周圍接近。

0	0	1	1	1	0	0
0	1	1	1	1	1	0
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
0	1	1	1	1	1	0
0	0	1	1	1	0	0

圖五： kernel 1

0	1	1	1	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	1	1	1	0

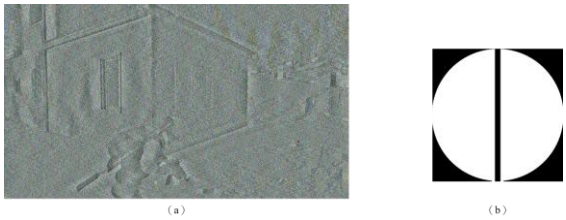
圖六： kernel 2



圖七：橫向錯位 (a) s channel；（b）二值化後圖像；（c）形態學處理後；（d）橫向錯位處理結果圖

ix. Lowpass Filter

此方法適合用在當高頻區域有雜訊時，先將圖片先拆成三通道，分別拿三通道做傅立葉轉換，把傅立葉轉換後的頻譜圖對 filter 做 bit wise and，然後反傅立葉轉換轉回原圖，把三通道合併；filter 是以 radius 作為半徑對中心點畫圓，在半徑內數值為 1，其他為 0，並把距離中心+-10 的 x 設為 0。最後計算與原圖差異的 size，周圍用 zero padding。



圖八：Lowpass 效果示意
(a)處理後影像；（b）filter 樣式

圖九：Bandpass 效果示意
(a)處理後影像；（b）filter 樣式

x. Bandpass Filter

用在當中頻域有雜訊時，可以保留高頻和低頻區域的特徵。先將圖片先拆成三通道，分別拿三通道做傅立葉轉換，把轉換後的頻譜圖對 filter 做 bit wise and，然後使用反傅立葉轉回原圖後把三通道合併 filter 是以 radius 作為半徑對中心點畫圓，在距離圓周 band_width 的半徑範圍內設為 0，其他為 1，並把中心的 x 設為 0。

2. 第二階段 – 雜湊計算

本專案接續第一階段產生之前處理後的結果進行下一步處理，在這邊使用了 Marr-Hildreth Hash 對每一圖像進行雜湊值計算與比較，將所有比較結果計算得出所有的影像之間的關係，之後再進一步進行排序得出最終影像順序列表。

i. Marr-Hildreth Hash

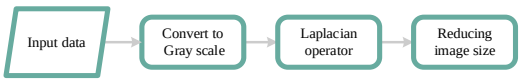
Marr-Hildreth 算法的基本思想是使用高斯平滑濾波器將圖像進行平滑，然後使用拉普拉斯算子來計算圖像的二階導數。根據二階導數的大小和方向，來計算出圖像中的邊緣。

Marr-Hildreth 算法的優點在於它能夠檢測出圖像中的細微邊緣，並且在處理噪聲較多的圖像時仍能有較好的表現。

算法的工作流程如下：

1. 將圖像轉換為灰度圖像。
2. 對圖像應用 Laplacian 運算子以增強邊緣和角點。
3. 將結果圖像縮小到較小的大小。

雜湊值是基於縮小後圖像中像素的強度值計算出來的。



圖十： 雜湊計算前處理

Laplacian 算子是一種圖像處理算子，用於增強圖像中的邊緣和角點。它通過計算每個像素與周圍像素的差值來實現這一目的。使用 Laplacian 運算子後，圖像中的邊緣和角點會變得更加明顯，其他區域會變得模糊。

縮小圖像後，Marr-Hildreth 雜湊函式會計算雜湊值。這通常是通過比較縮小後圖像中像素的強度值來實現的。例如，可以將圖像中的每個像素的強度值視為二進位制，並將這些二進位制拼接在一起形成雜湊值。這樣生成的雜湊值可以用來比較圖像的視覺內容。

然而，Marr-Hildreth 算法也有一些缺點。首先，它對圖像的尺寸敏感，因此在處理不同尺寸的圖像時可能會出現問題，但因為在本專案同資料集影像大小尺寸都相同，在這邊並不受到這點影響。

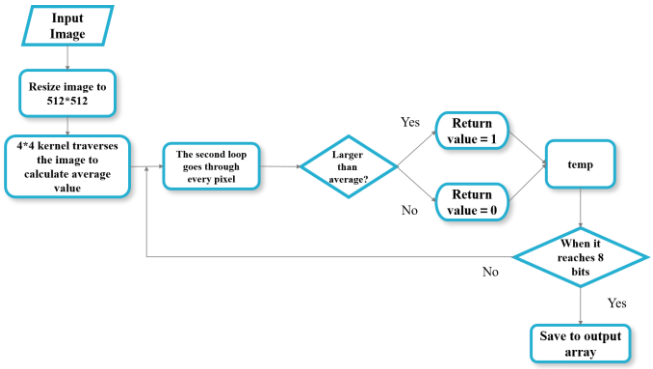
此外，因為 Marr-Hildreth 算法的計算複雜度較高，因此在處理大型圖像時可能會出現效能問題。

儘管如此，Marr-Hildreth 算法仍然是一種非常有用的圖像處理工具，並且在許多應用中得到了廣泛使用。

而雜湊函式的計算方式，輸出影像的大小是建立一個大小為 1x72 的單通道 8 位元無號整數型影像 (CV_8U)，並將其賦值給最後輸出的變數。

輸出影像的大小之所以會改變，是因為雜湊值的長度是固定的，並且是以 8 位元為一組來進行儲存。計算的流程首先會將輸入圖片縮放至 512*512，之後以一 4*4 的 kernel 進行平均值運算，再遍歷每一個像素質判斷該像素是否大於平均，若是是則回傳 1，否則回傳 0，待運算至 8 位元時，回傳至輸出陣列並將暫存清空，之後持續做到結束為止。在這段程式中，雜湊值的長度是 72 個字元，因此輸出影像的大小就會設定為 1x72。

雜湊值的長度也可能會因為其他因素而有所變化，例如雜湊演算法的不同，或者是輸入資料的大小和結構的不同。但因為有進行尺寸縮放的緣故，在我們所使用的方法中，雜湊值的長度是固定的，並且輸出影像的大小也是固定的。



圖十一： 雜湊值計算流程

圖十二：Hash 計算結果
(a) 原圖（1280*720）；（b）雜湊計算後（72*1）

ii. Hamming Distance

在圖像處理中，可以使用漢明距離來度量兩張圖像的差異程度，從而比較兩張圖像的相似度。

漢明距離計算方法如下：

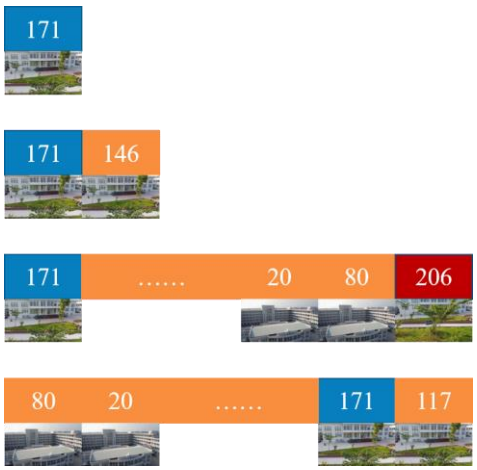
1. 取得兩張圖像的二進制串像素值。
2. 將兩個二進制串按位進行比較，計算不同的位數。
3. 將不同的位數相加，即得到漢明距離。

最終得到的結果 0 代表為相同圖片，數字越大代表兩者越不相似。

iii. Sorting

排序的部分是先將各圖片計算雜湊後先比較出相差最大的圖片，並將其中一張作為排序的起始位置，會使用這種手法是因為我們發現雖然相差最大不一定會是影片的頭尾，但可以確定的是如果是使用此手法開始排序的話，可以先連續排序較多張照片。

在遇到雜湊值變化較大時(當下雜湊的差值)會將整個影片序列反向後繼續排列，由此手法可以環境有大範圍不重複移動的影片中排序出相對完整的序列。



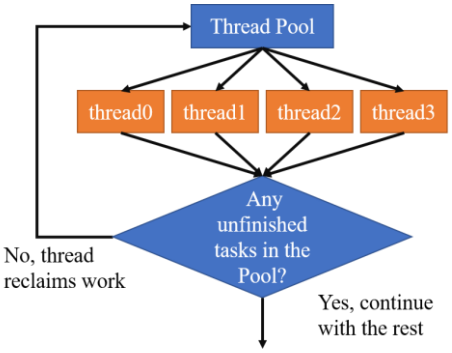
圖十三： sorting 方式示意圖

3. 第三階段 – 輸出成果

本專案接續階段二得到了亂序影像的最終排序結果，將運算出的結果順序輸出成影片檔案，並計算出評估指標 SRCC 和 MSE 與所有執行過程時長顯示出。

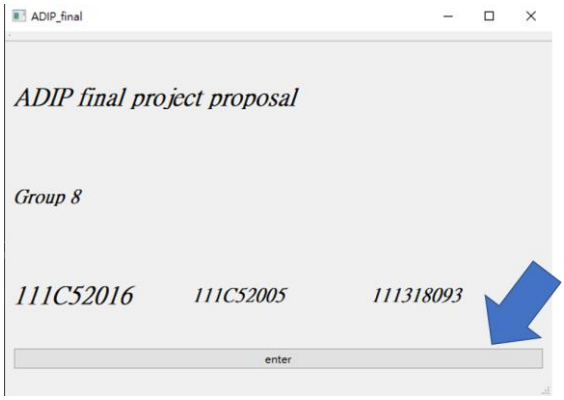
4. 執行時間優化 – 平行處理

由於影像前處理以及計算雜湊值需要花許多的時間，且由於 C 語言的特性，在預設的情況下都只會使用 1 個 thread，在需要對多張影像前處理以及計算雜湊的情況下等待時間就會被延長，以及不能有效的利用電腦硬體，但如果可以先將要處理的任務都先提交到 Thread Pool 中，在等 Thread Pool 完成所有任務後再執行剩下的程式即可省下大量時間。

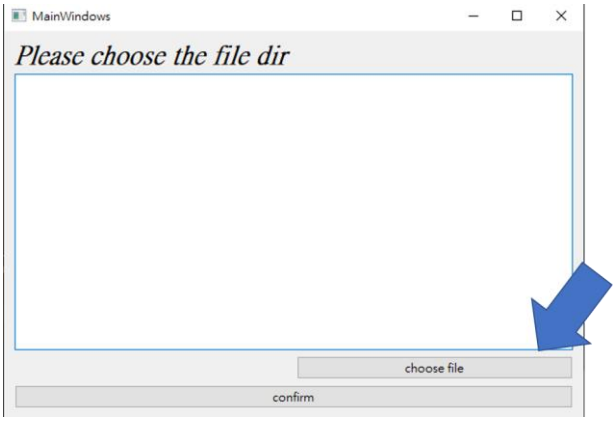


圖十四：平行處理流程圖

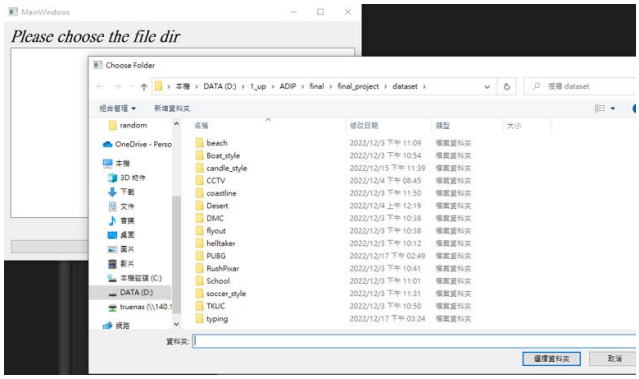
介面操作流程



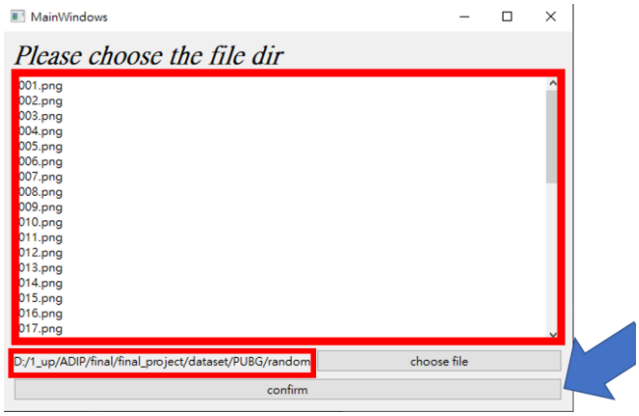
圖十五：初始畫面



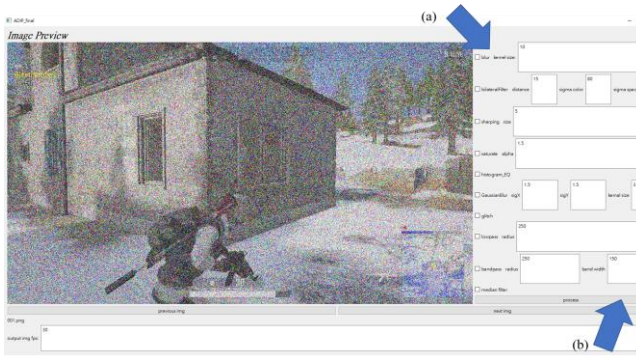
圖十六：選取讀取檔案按鈕



圖十七：選取資料夾

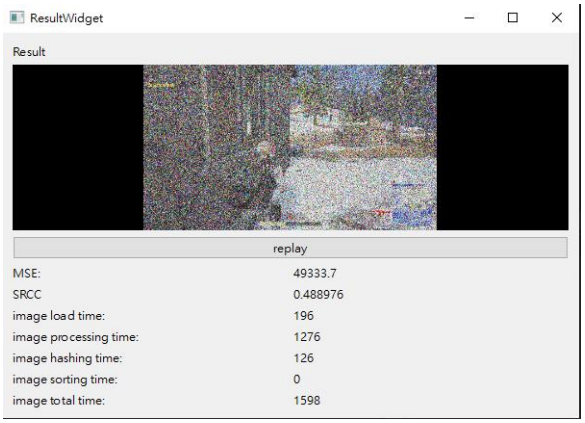


圖十八：讀取完成後按下確認



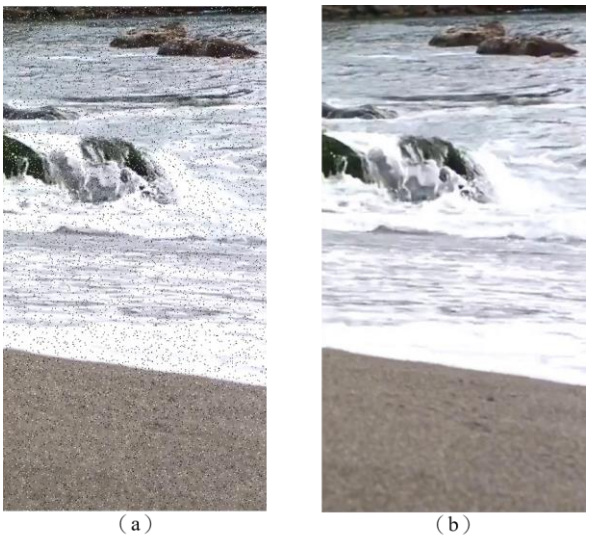
圖十九：調整前處理流程
(a)各個方案與數值調整；（b）執行前處理

圖二十：輸出影片
(a) 確認處理後的圖片；（b）輸入輸出影片的 fps；（c）點擊輸出



圖二十一：最終結果介面

實驗結果與比較



圖二十二：Case 1 結果圖
(a)進行預設前處理前；（b）進行預設前處理後

由圖上結果明顯可看出椒鹽雜訊獲得大幅度的改善，且圖像邊緣獲得較好的保留，Glitch 則無對此種圖像造成影響。

圖二十三：Case 2 結果圖

(a)進行預設前處理前；（b）進行預設前處理後

而在完全無雜訊的影像上，前處理並不會造成任何丟失特徵的影響，依舊保留原本的影像品質。

在 SRCC 中，beach、flyout、School 都取得了不錯的結果，以我們所觀察到的，影像若是線條不複雜、構圖大、張數充足，都能獲得較好的成績。而在計算時長因為使用了雜湊函數進行運算比較，加上以平行運算運行程式，因此都獲得了相當不錯的成果。

表 I： 評估結果

資料集	圖片張數	前處理時間 (sec)	SRCC	MSE	排序計算時間 (sec)	處理時長總計 (sec)
beach	270	6.718	0.9995	1477.7	0.271	7.522
Boat_style	338	8.063	0.6948	2827.6	0.357	8.980
candle_style	340	11.31	0.0242	7211.3	0.365	12.26
CCTV	162	10.29	0.2582	1553.2	0.303	11.33
coastline	165	4.183	0.4650	15834	0.158	4.681
Desert	142	1.750	0.8145	18181	0.095	1.999
DMC	600	14.38	0.0037	2138.5	0.742	16.36
flyout	450	11.08	0.9860	1102.2	0.493	12.50
helltaker	300	21.46	0.5877	1323.3	0.584	23.16
PUBG	43	1.425	0.0219	47041	0.042	1.573
RushPixar	300	7.498	0.2731	17160	0.287	8.407
School	230	5.942	0.9908	3130.2	0.219	6.637
soccer_style	150	4.537	0.4749	3408.2	0.135	5.000
TKUC	153	3.765	0.5500	15613	0.147	4.220
typing	720	16.78	0.1162	1939.3	0.926	19.070

結論

在本期末專題中，在影像前處理是最為困難的部分，由於後面雜湊函數的設計著重於影像的輪廓與邊緣，因此若有雜訊、需要考量三通道時的狀況，都很容易影響到最後結果的精準度。

但在不同的資料集中會遭遇不同的狀況，很難找出一套通解是可以兼顧所有狀況且運算時間部會過長、影像也不會過度失真，加上部分影像資料在過暗或過亮的狀況時，像素的資訊都已經失去了，無法將其細節還原出來，因此也會造成結果的偏差計算。

使用本專題所提出的方法來處理本次的問題仍然有需多不足並需要改進，期望能加入深度學習系統協助將資料集進行初始分類，並製作符合各個狀況的方法來處理相對應前處理方式，最後在進行排序運算，或許有機會得到較好的結果。

參考資料

[1] https://github.com/opencv/opencv_contrib

[2] https://github.com/vit-vit/CTPL

[3] https://ppfocus.com/0/mic6b8c41.html