**高等影像處理**

**作業#2**

姓名： 蘇柏凱

學號： 111c71007

指導老師： 蔣欣翰、李曉祺

|  |
| --- |
| **1.1** |
| Figure |
| Lena256\_512\_out.raw    Result |
| Disgussion |
| MSE為48.1183,PSNR為31.3077，MSE與PSNR的計算僅用同一對應位置的像素點，而缺乏考量了空間關係，因此在部分時候會忽略掉一些視覺上的特徵，有可能在我們看來十分相似的圖片卻因為shift而導致MSE、PSNR表現很差，而一般來說PSNR在30~40表示圖片質量還是可以接受。 |

|  |
| --- |
| **1.2** |
| Figure |
| Lena512\_128\_blur\_out.raw    Lena512\_128\_out.raw |
| Disgussion |
| 可以在實作中發現有經過模糊處理後再下採樣的圖片整體較為自然，而沒有做過模糊處理直接下採樣的圖片在一些邊角的地方顯得較為銳利。 |

|  |
| --- |
| **1.3** |
| Figure |
| |  |  | | --- | --- | | Name | lena128\_256\_a\_out.raw | | Image  Neighbor  (256\*256) |  | | Name | lena128\_256\_b\_out.raw | | Image  Bilinear  (256\*256) |  | | Name | lena128\_256\_c\_out.raw | | Image  Bicubic  (256\*256) |  | | Name | lena128\_512\_a\_out.raw | | Image  Neighbor  (512\*512) |  | | Name | lena128\_256\_c\_out.raw | | Image  Bilinear  (512\*512) |  | | Name | lena128\_256\_c\_out.raw | | Image  Bicubic  (512\*512) |  | |
| Disgussion |
| 利用Neighbor 128x128->256x256的圖片MSE為101.649,PSNR為28.0598,運算時間為0.011秒 利用Neighbor 128x128->512x512的圖片MSE為172.788,PSNR為25.7557,運算時間為0.046秒  利用Bilinear 128x128->256x256的圖片MSE為87.1461,PSNR為28.7283,運算時間為0.019秒 利用Bilinear 128x128->512x512的圖片MSE為123.265,PSNR為27.2224,運算時間為0.068秒  利用Bicubic 128x128->256x256的圖片MSE為72.8098,PSNR為29.5089,運算時間為0.059秒 利用Bicubic 128x128->512x512的圖片MSE為113.568,PSNR為27.783,運算時間為0.227秒 可以發現bicubic的效果>bilinear的效果>neighbor的效果，不論是在MSE或是在PSNR上的指標都可以看出這個特性，但是也同時可以發現在肉眼看起來bilinear會有格子點的暈邊現象，就肉眼上看起來的效果並不如neighbor，這些演算法都有自己的侷限性，當圖片被放大到一定程度後三個演算法的差異會被縮小，而同時也可以發現bicubic視效果最好、計算量最大的演算法，在計算時間上bicubic>bilinear>neighbor的計算時間。 |

|  |
| --- |
| **1.4** |
| Figure |
| |  |  | | --- | --- | | Name | Lena512\_384\_a\_out.raw | | Image  Neighbor |  | | Name | Lena512\_384\_b\_out.raw | | Image  Bilinear |  | | Name | Lena512\_384\_c\_out.raw | | Image  Bicubic |  | |  | | |
| Disgussion |
| 1.4 Neighbor運算所花費的時間：0.023 S  1.4 Bilinear運算所花費的時間：0.04 S  1.4 Bicubic運算所花費的時間：0.137 S  從肉眼看圖片的品質可以發現bicubic的效果與bilinear的效果相近都優於neighbor的效果，但是也同時可以發現這些演算法都有自己的侷限性，以bilinear跟bicubic為例可以發現在邊緣的處理上採用不同方式會有不同效果，如果是採用補0的話圖片邊緣會顯得較為暗淡，而我採用的是依據離邊緣最近的block計算，可以發現在邊緣會有一些輕微鋸齒狀，而在計算的時間也可以發現bicubic>bilinear>neighbor的計算時間。。 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **1.5** | | Figure | | |  |  | | --- | --- | |  | Neighbor | | Name | lena256\_576\_384\_a\_out | | Image  (↑2.25↓1.5)  Neighbor |  | | Name | lena256\_128\_384\_a\_out | | Image  (↓1.5↑2.25)  Neighbor |  | | Name | lena256\_384\_a\_out | | Image  (↑1.5)  Neighbor |  | |  | Bilinear | | Name | lena256\_576\_384\_b\_out | | Image  (↑2.25↓1.5)  Bilinear |  | | Name | lena256\_128\_384\_b\_out | | Image  (↓1.5↑2.25)  Bilinear |  | | Name | lena256\_384\_b\_out | | Image  (↑1.5)  Bilinear |  | |  | Bicubic | | Name | lena256\_576\_384\_c\_out | | Image  (↑2.25↓1.5)  Bicubic |  | | Name | lena256\_128\_384\_c\_out | | Image  (↓1.5↑2.25)  Bicubic |  | | Name | lena256\_384\_c\_out | | Image  (↑1.5)  Bicubic |  | |  | | | | Disgussion | | 1.5 ↑2.25↓1.5 Neighbor運算所花費的時間：0.082 S  1.5 ↑2.25↓1.5 Bilinear運算所花費的時間：0.12 S  1.5 ↑2.25↓1.5 Bicubic運算所花費的時間：0.415 S  1.5 ↓1.5↑2.25 Neighbor運算所花費的時間：0.028 S  1.5 ↓1.5↑2.25 Bilinear運算所花費的時間：0.047 S  1.5 ↓1.5↑2.25 Bicubic運算所花費的時間：0.153 S  1.5 ↑1.5 Neighbor運算所花費的時間：0.023 S  1.5 ↑1.5 Bilinear運算所花費的時間：0.037 S  1.5 ↑1.5 Bicubic運算所花費的時間：0.128 S  可以發現先縮小再放大的圖片在細節上損失最多，畫面整體也較模糊，可能是因為資料像下降維的壓縮破壞掉一部分資訊，而再放大時這些受到破壞的資訊就會被凸顯出來，而先放大再縮小的圖片在細節上較為銳利，畫面整體比起直接放大的模糊一些，可能是因為在縮小時並沒有先作模糊處理，而細部在放大時被凸顯導致，而在計算的時間也可以發現bicubic>bilinear>neighbor的計算時間。 | |

|  |
| --- |
| **2.1** |
| Figure |
| |  |  |  |  | | --- | --- | --- | --- | |  | D4 | D8 | Dm | | 72 |  |  |  | | 總步數 | 39 | 30 | 39 | |
| Disgussion |
| D4 Traveled path:  (0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (0, 4) -> (0, 5) -> (0, 6) -> (0, 7) -> (0, 8) -> (1, 8) -> (2, 8) -> (3, 8) -> (4, 8) -> (4, 7) -> (4, 6) -> (3, 6) -> (2, 6) -> (2, 5) -> (2, 4) -> (3, 4) -> (4, 4) -> (4, 3) -> (5, 3) -> (6, 3) -> (6, 2) -> (6, 1) -> (6, 0) -> (7, 0) -> (8, 0) -> (9, 0) -> (9, 1) -> (9, 2) -> (9, 3) -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)  總步數 :39  D8 Traveled path:  (0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (0, 4) -> (0, 5) -> (0, 6) -> (0, 7) -> (1, 8) -> (2, 8) -> (3, 8) -> (4, 7) -> (3, 6) -> (2, 5) -> (3, 4) -> (4, 3) -> (5, 3) -> (6, 2) -> (6, 1) -> (7, 0) -> (8, 0) -> (9, 1) -> (9, 2) -> (9, 3) -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)  總步數 :30  Dm Traveled path:  (0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (0, 4) -> (0, 5) -> (0, 6) -> (0, 7) -> (0, 8) -> (1, 8) -> (2, 8) -> (3, 8) -> (4, 8) -> (4, 7) -> (4, 6) -> (3, 6) -> (2, 6) -> (2, 5) -> (2, 4) -> (3, 4) -> (4, 4) -> (4, 3) -> (5, 3) -> (6, 3) -> (6, 2) -> (6, 1) -> (6, 0) -> (7, 0) -> (8, 0) -> (9, 0) -> (9, 1) -> (9, 2) -> (9, 3) -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)  總步數 :39  這是一個標準的路徑規劃題目，起初採用Dynamic Programming的方式，採用Dynamic Programming的方式可以很輕鬆地找到最短路徑長度，但是卻較難還原出行走路徑，於是改成採用BFS(寬度優先演算法)把每一個路徑push進vector中並同時記錄已經走過的路徑直到走到終點為止，路徑則依據走過的路徑步數進行回推，找到步數-1且符合D4/D8/Dm規則的路徑，並進行記錄，圖片部分用xnview的NN放大為100\*100 並儲存。 |

|  |
| --- |
| **2.2** |
| Figure |
| |  |  |  |  | | --- | --- | --- | --- | |  | D4 | D8 | Dm | | {72,145} |  |  |  | | 總步數 | 21 | 12 | 18 | |
| Disgussion |
| D4 Traveled path:  (0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (0, 4) -> (1, 4) -> (2, 4) -> (3, 4) -> (4, 4) -> (4, 3) -> (5, 3) -> (6, 3) -> (6, 4) -> (7, 4) -> (7, 5) -> (8, 5) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)  總步數 :21  D8 Traveled path:  (0, 0) -> (1, 0) -> (2, 1) -> (3, 2) -> (4, 3) -> (5, 3) -> (6, 4) -> (7, 5) -> (8, 6) -> (9, 7) -> (9, 8) -> (9, 9)  總步數 :12  Dm Traveled path:  (0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (3, 1) -> (3, 2) -> (3, 3) -> (4, 3) -> (5, 3) -> (6, 3) -> (6, 4) -> (7, 4) -> (7, 5) -> (8, 5) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)  總步數 :18  這是一個標準的路徑規劃題目，改成採用BFS(寬度優先演算法)把每一個路徑push進vector中並同時記錄已經走過的路徑直到走到終點為止，路徑則依據走過的路徑步數進行回推，找到步數-1且符合D4/D8/Dm規則的路徑，並進行記錄，圖片部分用xnview的NN放大為100\*100 並儲存。 |

|  |
| --- |
| **2.3** |
| Figure |
| |  |  |  |  | | --- | --- | --- | --- | |  | D4 | D8 | Dm | | {72,145,218} |  |  |  | | 總步數 | 19 | 10 | 16 | |
| Disgussion |
| D4 Traveled path:  (0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (3, 1) -> (3, 2) -> (3, 3) -> (4, 3) -> (5, 3) -> (6, 3) -> (6, 4) -> (7, 4) -> (7, 5) -> (8, 5) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8) -> (9, 9)  總步數 :19  D8 Traveled path:  (0, 0) -> (1, 1) -> (2, 2) -> (3, 3) -> (4, 4) -> (5, 5) -> (6, 6) -> (7, 7) -> (8, 8) -> (9, 9)  總步數 :10  DmTraveled path:  (0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (3, 1) -> (3, 2) -> (3, 3) -> (4, 3) -> (5, 3) -> (5, 4) -> (5, 5) -> (6, 6) -> (7, 7) -> (8, 8) -> (9, 8) -> (9, 9)  總步數 :16  這是一個標準的路徑規劃題目，是採用BFS(寬度優先演算法)把每一個路徑push進vector中並同時記錄已經走過的路徑直到走到終點為止，路徑則依據走過的路徑步數進行回推，找到步數-1且符合D4/D8/Dm規則的路徑，並進行記錄，圖片部分用xnview的NN放大為100\*100 並儲存。 |

|  |
| --- |
| **3** |
| Figure |
| |  |  | | --- | --- | | Lena256 Shift 0 bit | lena\_gray\_0bits.raw | | Lena256 Shift 1 bit | lena\_gray\_1bits.raw | | Lena256 Shift 2 bit | lena\_gray\_2bits.raw | | Lena256 Shift 3 bit | lena\_gray\_3bits.raw | | Lena256 Shift 4 bit | lena\_gray\_4bits.raw | | Lena256 Shift 5 bit | lena\_gray\_5bits.raw | | Lena256 Shift 6 bit | lena\_gray\_6bits.raw | | Lena256 Shift 7 bit | lena\_gray\_7bits.raw | | Baboon256 Shift 0 bit | baboon\_gray\_0bits.raw | | Baboon256 Shift 1 bit | baboon\_gray\_1bits.raw | | Baboon256 Shift 2 bits | baboon\_gray\_2bits.raw | | Baboon256 Shift 3 bits | baboon\_gray\_3bits.raw | | Baboon256 Shift 4 bits | baboon\_gray\_4bits.raw | | Baboon256 Shift 5 bits | baboon\_gray\_5bits.raw | | Baboon256 Shift 6 bits | baboon\_gray\_6bits.raw | | Baboon256 Shift 7 bits | baboon\_gray\_7bits.raw | |
| Disgussion |
| img\_lena 保留8Bits MSE :0  img\_lena保留8Bits PSNR :inf  img\_baboon保留8Bits MSE :0.  img\_baboon保留8Bits PSNR :inf  img\_lena 保留7Bits MSE :0.506226  img\_lena保留7Bits PSNR :51.0874  img\_baboon保留7Bits MSE :0.507782  img\_baboon保留7Bits PSNR :51.074  img\_lena保留6Bits MSE :1.66493  img\_lena保留6BitsPSNR :45.9168  img\_baboon\_保留6BitsMSE :1.56209  img\_baboon保留6BitsPSNR :46.1937  img\_lena保留5BitsMSE :7.63368  img\_lena保留5BitsPSNR :39.3035  img\_baboon保留5BitsMSE :6.8869  img\_baboon保留5BitsPSNR :39.7506  img\_lena保留4BitsMSE :33.871  img\_lena保留4BitsPSNR :32.8325  img\_baboon保留4BitsMSE :29.4662  img\_baboon保留4BitsPSNR :33.4376  img\_lena保留3BitsMSE :152.171  img\_lena保留3BitsPSNR :26.3075  img\_baboon保留3BitsMSE :139.663  img\_baboon保留3BitsPSNR :26.68  img\_lena保留2BitsMSE :1038.92  img\_lena保留2BitsPSNR :17.965  img\_baboon保留2BitsMSE :759.58  img\_baboon保留2BitsPSNR :19.3251  img\_lena保留1BitMSE :8493.91  img\_lena保留1BitPSNR :8.83973  img\_baboon保留1BitMSE :9678.37  img\_baboon保留1BitPSNR :8.27278  這一題的儲存上因為常見的儲存形式都是用 8-bits而xnview也只能夠見到8-bits的影像，在影像儲存理論上每少一個bit儲存空間也會少一半，但為了可視化raw檔案在儲存中實際上都還是採用8-bits，而為了可視化，在右移後我再根據數字將其還原在0-255中應該有的數字大小，以右移7bits 為例只剩下0、1，則變成0、255，而shift 0 bit 表示保留8bits，與原圖無異。 |

|  |
| --- |
| **4** |
| Figure    lena1024\_cir\_out.raw |
| Disgussion |
| 這一題把1024\*1024的圖像變成32\*32個block，每一個block中有16\*16個pixels，本題使用open-cv 協助畫圓，並依據每一個block中gray-level的平均分成8種不同半徑，並在最後把結果存回raw檔中，可以發現在把圖像縮小的時候看起來與原圖更為接近，透過不同圓形大小達到視覺上看起來不同的灰階是一個十分有趣的視覺效果。 |