**高等影像處理**

**作業#(2)**

姓名：梁仔弘

|  |  |
| --- | --- |
| 1.1 | |
| Figures | |
| *lena\_512.raw* (比較用原圖) | Resized from *lena\_256.raw* by row-column replication |
| Discussion | |
| 由於row-column replication是一個簡單暴力的演算法，單純就是將row及column放大兩倍(例如第二個row的值其實就是第一個row)，所以可以看的出來所產生的圖片有某些地方失真(模糊)。比如說頭髮以及帽子邊緣，可以明顯看出一格一格的鋸齒，而在左圖原圖上，則是非常平滑的。 | |

|  |  |
| --- | --- |
| 1.2 | |
| **Figures** | |
| **Nearest Neighbors Interpolation**  *1\_2\_NN.png* | **Bilinear Interpolation**  *1\_2\_BL.png* |
| **Bicubic Interpolation**  *1\_2\_BC.png* | |
| Discussion | |
| **品質比較**：  這三種演算法放大後的圖片或多或少都會有些鋸齒現象，鋸齒嚴重程度大概是**最近鄰>雙線性>雙三次插值**。   1. 最近鄰插值只是單純取最近的元素填上，所以大量的鋸齒失真是可預期的，而這個演算法導致**無論是細節還是主體輪廓都沒辦法良好地保存下來**。 2. 雙線性插值則是根據插入點與周圍取樣點的距離比值來填上。直覺上這是一個非常好的演算法，但根據維基百科，這個演算法具有**低通濾波器**的性質，因此會造成高頻部分受損嚴重，而**高頻部分通常是圖片的細節**。以此圖為例，頭髮部分受損極為嚴重，明顯可以看到一格一格的鋸齒。 3. 雙三次插值，取三次函數，並取樣十六個點來使像素插值能夠更加的平滑，但是也因此計算複雜程度更高。不過換來的是高品質的圖片，可以看到幾乎是完美的表現。   **邊界處理**：  其實並不難，若取樣點會超出邊界，那就**直接將其設為邊界上的點**，如此一來便不會造成邊緣問題。 | |

|  |  |
| --- | --- |
| 1.3 | |
| **Figures** | |
| *lena\_256.raw*(比較用原圖) | |
| Resized from *lena\_512.raw* by row-column deletion  (Without Blurring) | Resized from *lena\_512.raw* by row-column deletion  (With Blurring) |
| Discussion | |
| Row-column deletion和row-column replication一樣，是一個簡單直覺又暴力的縮小圖片方式，單純就是將某些行列刪除來構成更小的圖片。比較左右兩圖，左邊為原圖，右邊為縮小後的圖片，可以看得出來右邊圖片較左邊更為**銳利**，這是因為新圖片**跳過了某些資訊**(被刪掉)，造成銜接處出現尖銳的感覺。而先模糊過後再做row-column deletion的成果，我認為他比沒有先模糊更加平滑，更接近原圖的感覺，尖銳感較小。我的推測是，一開始的模糊會讓圖片像素間更加平滑，所以當Row-column deletion被應用上去時，所造成的銳化感就變小了。 | |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.4 | | | |
| **Figures** | | | |
|  | **↑3.25↓2.5** | **↓2.5↑3.25** | **↑1.3** |
| **Nearest Neighbor** | *1\_4\_NNa.png* | *1\_4\_NNb.png* | *1\_4\_NNc.png* |
| **Bilinear** | *1\_4\_BLa.png* | *1\_4\_BLb.png* | *1\_4\_BLc.png* |
| **Bicubic** | *1\_4\_BCa.png* | *1\_4\_BCb.png* | *1\_4\_BCc.png* |
| Discussion | | | |
| 這一題用了三種演算法來作三種不同的縮放，但他們最終都會變成原圖的1.3倍。  **縮放品質**：   1. 第一種縮放方式(**↑**3.25↓2.5)bilinear跟bicubic看起來差不多，但nearest neighbor的結果明顯較差，我想這是因為他本身放大的效果就已經不好了，因此造成降採樣的時候又出現更糟糕的結果。 2. 第二種縮放(**↓**2.5↑3.25)，三種演算法的表現都蠻糟糕的，這是因為down sampling的時候造成了大量的**資訊丟失**，所以當我們要升採樣回來的時候，就會非常難以還原出原本的圖片。這個縮放方式可以**模擬傳送圖片時所出現的問題**，因為傳送通常沒辦法傳送完整的原圖，而是會先降採樣後嘗試將其還原。 3. 第三種縮放(**↑**1.3)是直接放大到1.3倍，這是最佳的放大方式。   **執行時間**：  我沒有特別去計算執行時間，因為都是按下去的瞬間結果就已經出現了，但我會認為Bicubic的執行時間所需最長。因為他必須要先採樣16個點，才能計算出插值，而bilinear只需一次取樣4個點。而nearest neighbor只需直接採樣最近的點，所以會最快。  **結論**：  無論是以實用面（降採樣後還原）或單純放大的性能表現來講，**Bicubic都是絕對的贏家**，即使他所需要的計算時間可能較其他演算法來的更長，但我認為這是可以接受的trade-off。 | | | |

|  |
| --- |
| 2.1 |
| **Answer** |
| The MSE score = 48.12  The PSNR = 31.31dB |
| **Discussion** |
| 由於單看MSE其實並不能判斷優劣，頂多能知道**越接近0越好**，但**我們不知道48是多好，我們不能武斷地說這是好的MSE分數**。因此我們可以看看PSNR，根據維基百科，**PSNR越大越好**，並且*「PSNR大於 30dB ，人眼很難察覺壓縮後和原始影像的差異。」*因此，我們可以知道，我們的PSNR=31dB代表其實1(1)放大的圖片失真的程度是還可以接受的範圍。 |

|  |
| --- |
| 2.2 |
| **Answer** |
| ↓2↑2: MSE = 203.59, PSNR = 25.04dB  ↓4↑4: MSE = 653.992, PSNR = 19.98dB |
| **Discussion** |
| 由於將圖片縮小會造成資訊丟失，所以我們可以預期下降倍數較大的圖片，還原後的MSE、PSNR表現會較糟糕。而我們透過這題也可以清楚看到，縮小四倍較縮小兩倍的MSE高出了三倍，PSNR也降到了20以下。根據維基百科，PSNR<20dB，雖然人眼可以看得出是原圖的結構，但卻已經是正常典型值(30dB~50dB)以下。 |

|  |
| --- |
| 3.1 |
| **Answer** |
| ↓2↑2: MSE = 37.951, PSNR = 32.34dB  Received Image *3\_1.png* |
| **Discussion** |
| 這一題我是將圖片大小縮小兩倍後進行傳輸，縮小兩倍後的大小為：  還原後的效果還可以接受，並沒有太大的失真，PSNR分數也有超過30。 |

|  |
| --- |
| 3.2 |
| **Answer** |
| ↓2↑2: MSE = 32.9291, PSNR = 32.8251dB  Received Image *3\_2.png* |
| **Discussion** |
| 這一題我是將圖片大小的bit數下調至4 bits，大小剛好等於傳輸限制： |

|  |  |
| --- | --- |
| 3.3 | |
| **Figures** | |
| **圖一、**  都降到65,536 bytes的情況下，兩種方式的PSNR和MSE | |
| *3\_3.png* | **圖二、**  左圖之MSE及PSNR |
| **Discussion** | |
| **推論**：  有兩種方式可以降低圖片的檔案大小，1. 降低圖片長寬2. 降低圖片bit數。而為了混合這兩種方式來壓縮圖片，所以必定要決定以哪一個方法為重。為了公平起見，我將分別用這兩種方式來將圖片降到65,536 bytes，並比較他們之間的PSNR來決定哪一個方式能夠對圖片的受損程度為最低。根據圖一得到的結果，可以知道降低bit數造成的損失較大，因此降低長寬會是主要的選擇。  **實作**：  再來，我決定將圖片長寬調至原本的0.75倍，並且將bit數下降至7 bits，於是圖片大小為  最後，根據圖二，可以看出PSNR表現並不如3(1)和3(2)所做之結果，3(1)甚至檔案大小為此作法之0.5倍。因此我的結論是，只選擇其中一種做即可，不需要兩種方式混合，而其中**最佳的方式就是將長寬縮小再嘗試將其還原**。 | |

|  |
| --- |
| 4 |
| **Discussion** |
| 沒做QQ |