Digital Image Processing (2017)

Homework 1 Report

{Image input/output + Resolution + Scaling}

1. Image input/output [1,2,3]

BMP 格式 : (皆以 Little-Endian 的方式儲存)
File Header + Info Header + (optional palette) + Raw Data

- File Header (14 bytes)

```
struct FILEHEADER {
    unsigned char id[2];
    unsigned fsize; //bmp_file size
    unsigned short reserve_1;
    unsigned short reserve_2;
    unsigned offset; //header bias
};
```

Identifier (id[2]) : 2 bytes, 一般都是 ‘ B ’ (0x42), ‘ M ’ (0x4D)
fsize : 4 bytes, BMP 檔案大小 (單位 : byte)
reserved : 4 bytes, 保留欄位
offset : 點陣圖資料開始之前的偏移量 (單位 : byte)

- Info Header (40 bytes)

```
struct INFOHEADER {
    unsigned info_size; //info header size
    int width; //image width
    int height; //image height
    unsigned short planes;
    unsigned short bits; //bit per pixel
    unsigned compress;
    unsigned im_size;
    int x_reso;
    int y_reso;
    unsigned color;
    unsigned imp_color;
};
```

(這邊只提及 Info Header 中較為重要的資料，而 width, height, bits，用來估算讀寫資料所需要的記憶體容量)

info_size	: 4 bytes, Info Header 的總長度 (包含 palette)
width	: 4 bytes, 圖片寬度
height	: 4 bytes, 圖片高度
bits	: 2 bytes, 每個 pixel 的顏色深度 (單位 : bit)
im_size	: 4 bytes, 點陣圖的資料大小 (Raw Data)

- Raw Data (包含 padding (null char) 的資料)
  正常的點陣圖掃描列是由底向上儲存的 (height 為正的情況下)，若顏色深度為 24 bits，即 3 bytes，資料會以 B(藍)G(綠)R(紅)的形式存取。若顏色深度為 24 bits，即 3 bytes，資料會以 BGRA(透明)的形式存取。
  然而因為 Dword-alignment[4] 的問題，每一掃取決於圖片的寬度(width)及顏色深度(bytes per pixel=bits/8)，若圖片的寬度乘上顏色深度不是 4 的倍數，擇要使用 zero padding 的方式，將資料變為 4 的倍數。
  因此，在讀取點陣圖資料時，不需要 malloc im_size 大小的記憶體空間，只需要 image_size – pad * height。而在寫入時，記得考慮 padding，將其寫入即可。(下面以 read 作為例子)

```
int pad = (4 - (IH.width * IH.bits / 8) % 4);
if ((IH.width * IH.bits / 8) % 4 != 0)
    * raw_data = new unsigned char[image_size - pad * IH.height];
else
    *raw_data = new unsigned char[image_size];
```

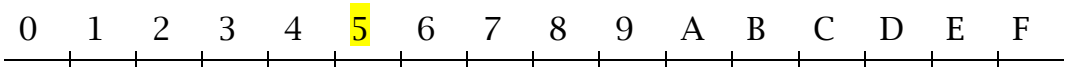
- Alignment[4,5]
  電腦中記憶體的資料放置在某個(2^N)倍數的記憶體位址，稱為對齊(alignment)。記憶體通常會設計成以 word-sized 進行存取會最有效率，即上述所提到的 Dword-alignment。若是不做 alignment，則可能會 access 更多次記憶體。
  在建構 File header 和 Info header 時，也會遇到類似的情形。例如 File header 的 Identifier (id[2])，若不做特別的處理，則會因為 alignment 的關係，去對齊 structure 中最大的 size(2→4)，將 structure 變為 16bytes。
  我們可以使用 pragma pack 來解決此問題。[6]

2. Resolution

通常 RGB 都分別用 1 bytes 做存取，即 8 bits，因此有 256 種可能。為了方便解釋，下面使用 4 bits quantize 到 2 bits 為例子。

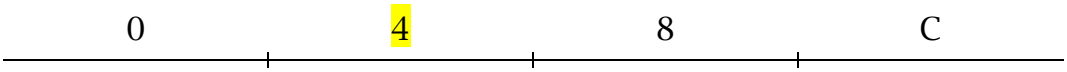
Quantization example:

1. A data with 4 bits :



2. Divided by 4 :

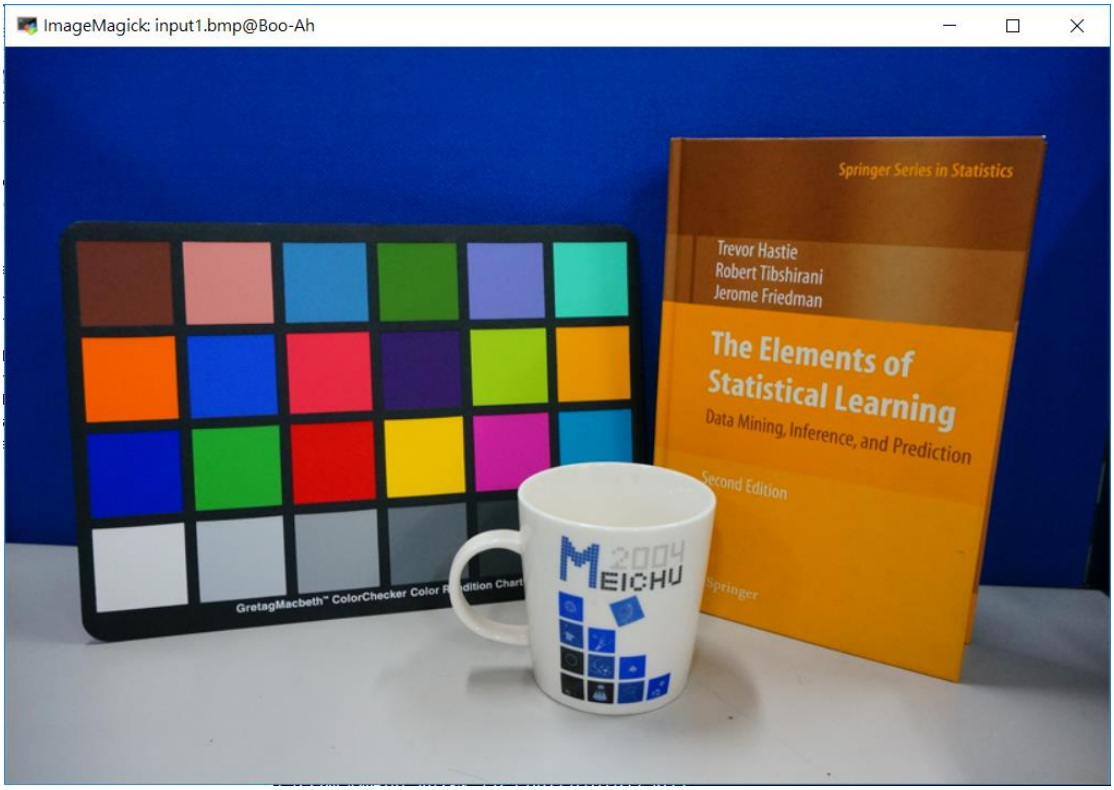
3. The data quantized to 2 bits (multiply by 4) :



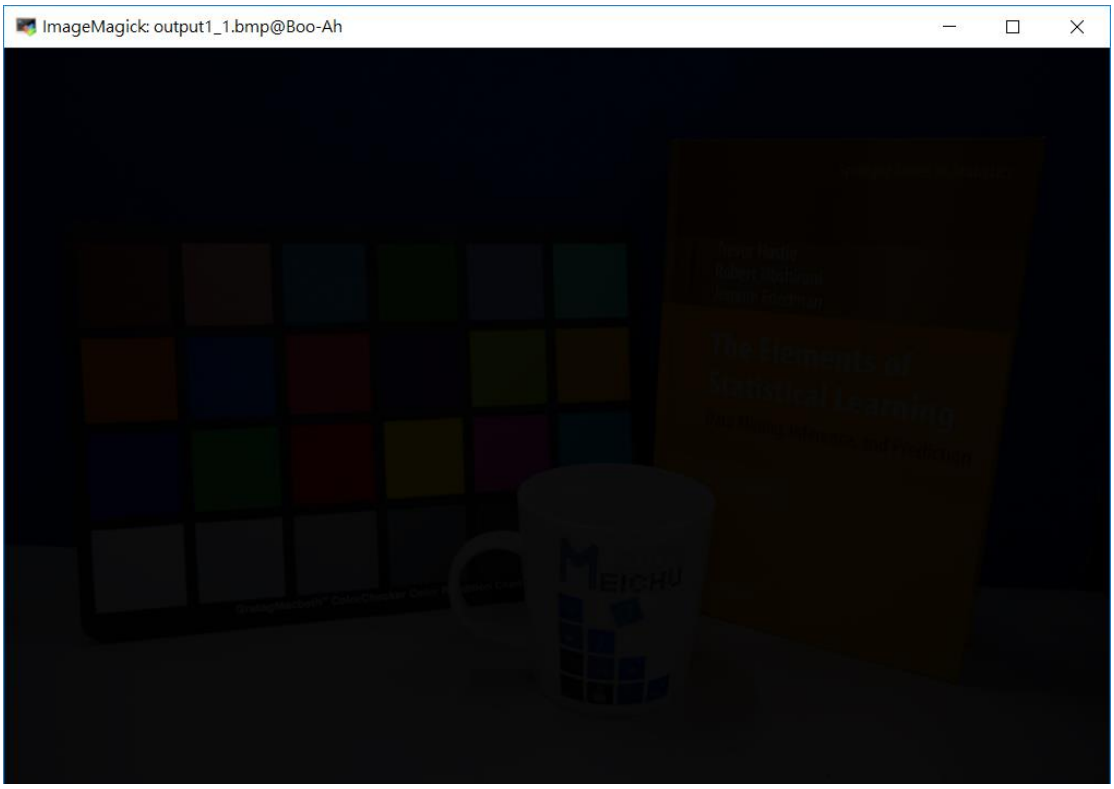
假設某個 4 bits 的資料的 R 值為 5 (highlight 黃色)，我們想要將其 quantize 為 4 bits 的資料。首先，我們需要將 16 等分的資料變為 4 等分，因此將其除於 4。然而，因為原先規定的顏色還是被劃分為 0~16 的 scale，需要將資料拉回原先的 scale，乘上 4 之後即完成 quantization 的動作。

BMP display in each step :

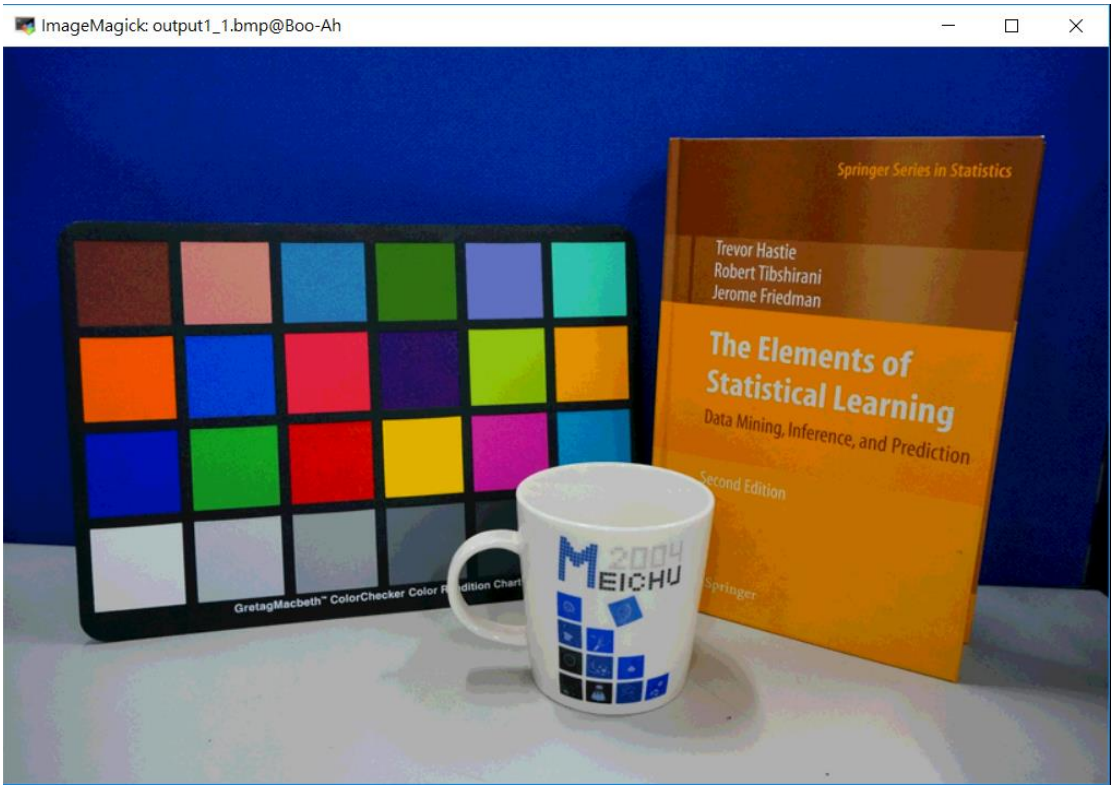
1. A data with 256 bits :



2. Divided by 16 :



3. The data quantized to 16 bits (multiply by 16):

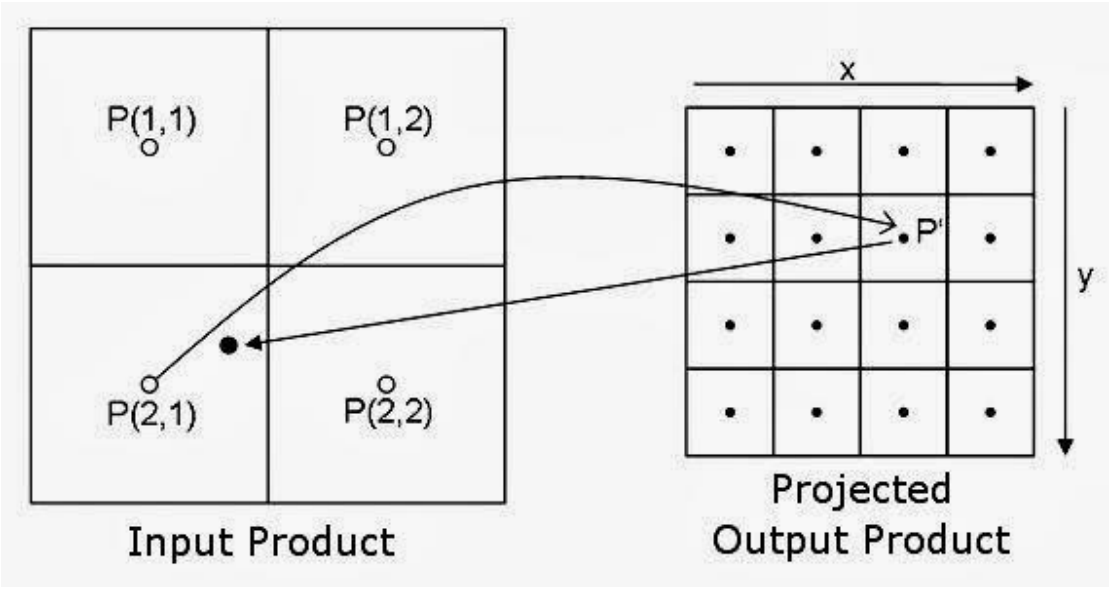


3. Scaling

影像縮放有下列 3 種方法，Nearest neighbor interpolation、Bilinear interpolation、Bicubic interpolation。[10,11,12]

1. Nearest neighbor interpolation :

此方法是利用最近的相鄰點的值來填入，演算法最為簡單，然而 scaling 效果較差。如下圖，新(右)圖的 P'點因為最相鄰原(左)圖的 P 點，所以就將 P 點的值填入 P'。



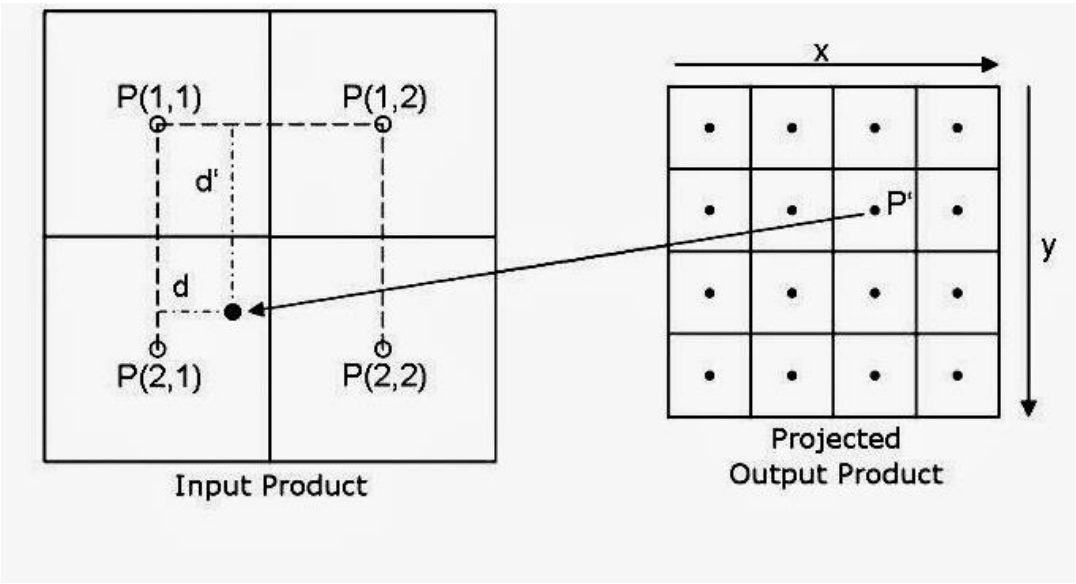
2. Bilinear Interpolation : [8,9]

此演算法會利用最近相鄰的四個點依相鄰的(比例)程度來填入。公式如下，

$$P'(x,y) =$$
$$(1-d)(1-d')P(1,1) + d(1-d')P(1,2) + (1-d)d'P(2,1) + d*d'*P(2,2)$$

如下圖，新(右)圖的 P'點因為最相鄰原(左)圖的 P(1,1),P(1,2),P(2,1),P(2,2) 四點，所以就將此四點的值依比例填入 P'。

若是距離某個點較近，則權重較大，反之較小。例如 P'因為距離 P(2,1)較近，乘上 1-(P'到 P(2,1)) X 方向的距離及 1-(P'到 P(2,1)) Y 方向的距離。



3. Bicubic interpolation :

Bicubic interpolation 和 Bilinear 的想法極為類似，Bicubic 考慮周圍 16 的點作估計。因為考慮的點比較多，因此縮放效果通常會比 Bilinear 還要好，但因為參數較多，計算上也較為複雜。

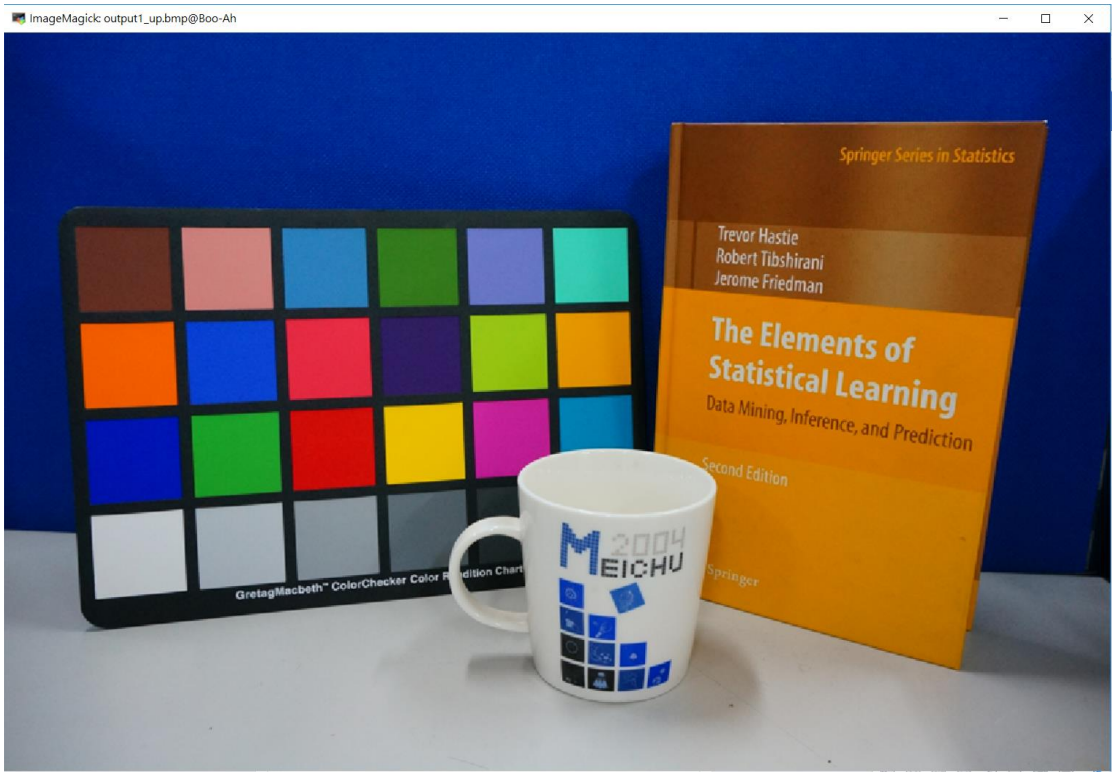
Boundary option : [7]

在實作影像縮放功能時，通常到了邊界(最後一個 row 或最後一個 column)會發生超出範圍的問題。這時候有幾種解決方式，symmetric、replicate、circular。

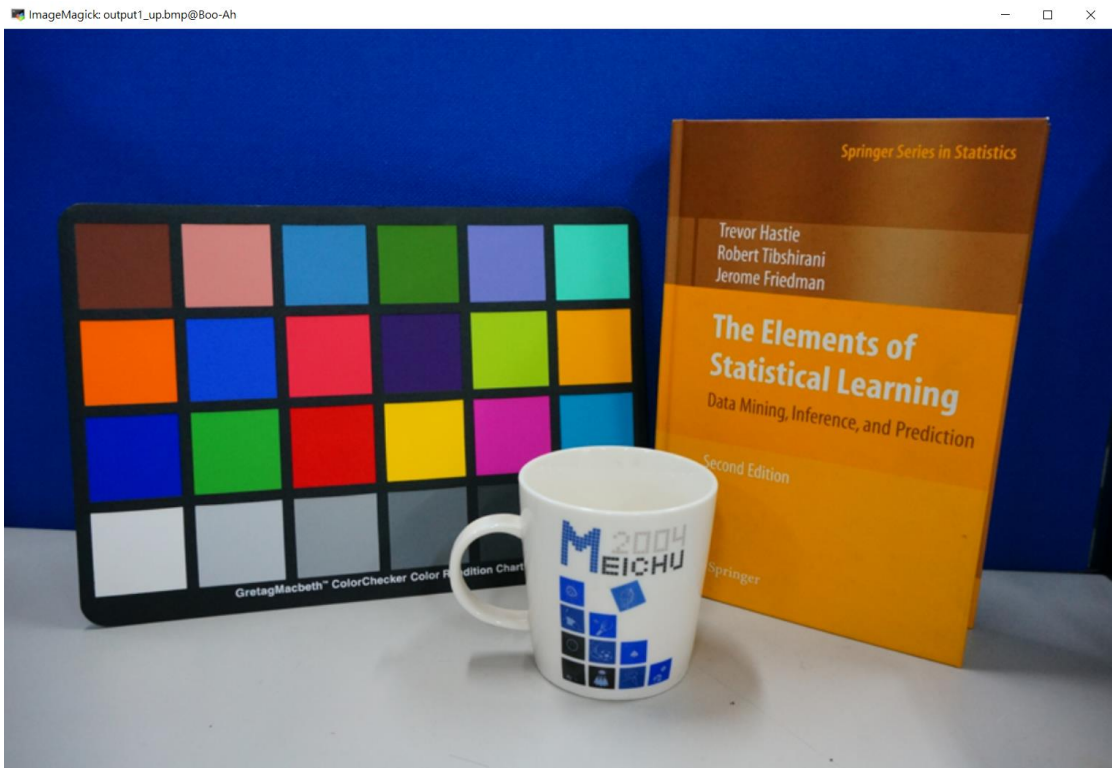
- Symmetric ：超出 boundary 的範圍的 pixel 使用鏡射的方式計算，通常是影像縮放最佳的方法。
- Replicate ：超出 boundary 的範圍的 pixel 假設為距離最近邊的數值，通常是最簡易的方法。
- Circular ：此方法適用於當圖片的數值在任何方向有規律性的重複，然而不適合應用在影像縮放中。

Up-scaling :

Nearest neighbor interpolation :



Bilinear interpolation :



Down-scaling :

Nearest neighbor interpolation :

Bilinear interpolation :

References

[1] https://www.jinnsblog.com/2009/08/bmp-format-graphic-illustration.html
[2] http://crazycat1130.pixnet.net/blog/post/1345538-點陣圖（bitmap）檔案格式
[3] https://zh.wikipedia.org/wiki/BMP
[4] http://opass.logdown.com/posts/743054-about-memory-alignment
[5] http://kezeodsnx.pixnet.net/blog/post/27585076-data-structure 的對齊%28alignment%29
[6] http://white5168.blogspot.tw/2013/04/pragma-pack1.html#.WdyEiCVPlEY
[7] https://stackoverflow.com/questions/16341167/bilinear-interpolation
[8] http://www.deeplearn.me/397.html
[9] https://chu24688.tian.yam.com/posts/44797337
[10] http://yzu1022cs368s991412.blogspot.tw/2014/03/digital-image-processing1-scale-rotate.html
[11] http://yzu1022cs362s1001549.blogspot.tw/2014/03/opencv.html
[12] https://blog.demofox.org/2015/08/15/resizing-images-with-bicubic-interpolation/