# 國　立

# 台　北　科　技　大　學

NATIONAL TAIPEI UNIVERSITY OF TECHNOLOGY

# 高等數位影像處理 HW2


指導老師: 郭天穎

班級: 電機碩一

姓名: 蘇冠宇

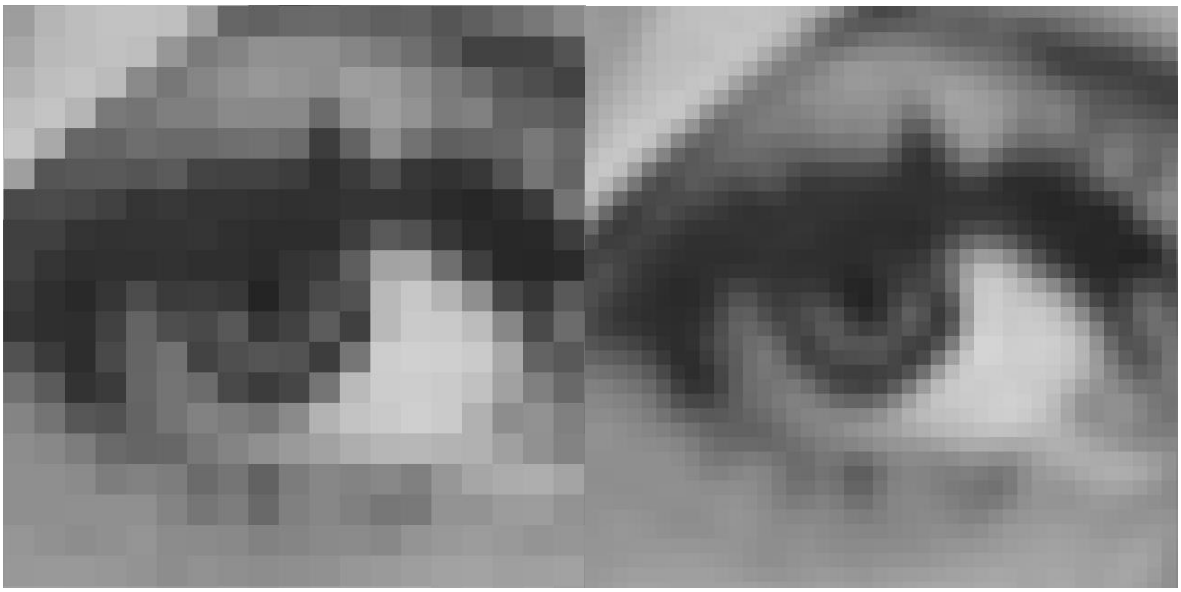
學號: 108318047

實驗室: DIVL 212

# 1.Zooming and Shrinking

## (a)

左圖為輸出 zoomX2 圖，右圖為比較原圖。在大小正常下比較兩者發現其實差不多，但是在放大就會發現其實原圖的色階較平滑，因為我使用的方法是複製 peixl 所以會看起來沒有那麼多漸層。



程式碼:

```
for (int i = 0; i < height; i++)
{
    for (int a = 0; a < 2; a++)
    {
        vector<BYTE> v1;
        for (int j = 0; j < width; j++)
        {
            for (int b = 0; b < 2; b++)
            {
                v1.push_back(in[i][j]);
            }
        }
        pz.push_back(v1);
    }
}
```

先創立一個兩倍空間的矩陣，原座標的 pixel 放到右邊、右下、下面的座標，來達成兩倍效果。

## (b)

- Nearest neighbor

上圖為↑3.5↓2

上圖為↓3.5↑2

上圖為↑1.75

程式碼:

```
float m = 1.0 / aa;
vector<vector<BYTE>> out;
int px, py;
for (int i = 0; i < in.size()*aa; i++)
{
    vector<BYTE>    v1;
    for (int j = 0; j < in[0].size() * aa; j++)
    {
        px = int(i * m);
        py = int(j * m);

        v1.push_back(in[px][py]);
    }
    out.push_back(v1);
}
return out;
```

Nearest nrighbor 是將近鄰考慮進去，以最近的鄰作為放大縮小的點。

- Bilinear

上圖為↑3.5↓2

上圖為↓3.5↑2

上圖為↑1.75

程式碼:

```
vector<vector<BYTE>> out;
float   m = (float)(in.size() * aa) / (float)(in.size() - 1);
float   a, b;
int x, y,pi;

for (int i = 0; i < in.size() * aa; i++)
{
    vector<BYTE> v1;
    for (int j = 0; j < in[0].size() * aa; j++)
    {
        y   =   i / m;
        x   =   j / m;
        a   =   ((float)i - (float)y * m) / m;
        b   = ((float)j - (float)x * m) / m;
        pi = (1 - a) * (1 - b) * in[y][x] + a * (1 - b) * in[y][x + 1] + b * (1 - a) * in[y + 1][x] + a * b * in[y + 1][x + 1];
        v1.push_back(pi);
    }
    out.push_back(v1);
}
return out;
```

Binear 是會考慮附近的點的權重，依照比例較靠近的點取比較多。

- Bicubit

上圖為↑3.5↓2

上圖為↓3.5↑2

上圖為↑1.75

程式碼:

```
void getW_x(float w_x[4], float x) {
    int X = (int)x;
    float a = -0.5;
    float stemp_x[4];

    stemp_x[0] = 1 + (x - X);
    stemp_x[1] = x - X;
    stemp_x[2] = 1 - (x - X);
    stemp_x[3] = 2 - (x - X);

    w_x[0] = a * abs(stemp_x[0] * stemp_x[0] * stemp_x[0]) - 5 * a * stemp_x[0] * stemp_x[0] + 8 * a * abs(stemp_x[0]) - 4 * a;
    w_x[1] = (a + 2) * abs(stemp_x[1] * stemp_x[1] * stemp_x[1]) - (a + 3) * stemp_x[1] * stemp_x[1] + 1;
    w_x[2] = (a + 2) * abs(stemp_x[2] * stemp_x[2] * stemp_x[2]) - (a + 3) * stemp_x[2] * stemp_x[2] + 1;
    w_x[3] = a * abs(stemp_x[3] * stemp_x[3] * stemp_x[3]) - 5 * a * stemp_x[3] * stemp_x[3] + 8 * a * abs(stemp_x[3]) - 4 * a;
}
```

計算權重的公式

```
vector<vector<BYTE>> out;
float Row_out = in.size() * a;
float Col_out = in[0].size() * a;
for (int i = 0; i < Row_out; i++)
{
    vector<BYTE> v1;
    for (int j = 0; j < Col_out; j++)
    {
        float x = i * (in.size() / Row_out);
        float y = j * (in[0].size() / Col_out);
        float w_x[4], w_y[4];
        getW_x(w_x, x);
        getW_y(w_y, y);
        int temp = 0;
        for ( int s = 0 ; s <= 3; s++) {
            for ( int t = 0 ; t <= 3; t++) {
                int u = int(x) + s - 1;
                int v = int(y) + t - 1;
                if (u < 0) { u = 0; };
                if (v < 0) { v = 0; };
                if (u >= in.size()) { u = in.size() - 1; }
                if (v >= in[0].size()) { v = in[0].size() - 1; }
                temp = temp + in[u][v]* w_x[s] * w_y[t];
            }
        }
        v1.push_back(temp);
```

Bicubit 也是取權重，但是她考慮的權重算式更加複雜。

- 討論差異

三種演算法都有一個共同的問題，就是先縮小再放大都會造成某些 pixel 資料丟失，因為縮小會造成某些資料被壓縮，而在放大這些資料原本擺放的地方就會跟原本的質有所差異。

而效果最好的是 bicubit 因為她考慮的四周圍的權重，但是也是跑最久的部分，然後在 Bilinear 會造成類似方格效應。

## (c)

左邊是沒有經過模糊處理的照片，看起來相對於右邊經過模糊處理的照片較粗糙，而右邊因經過模糊處裡所以較平滑。

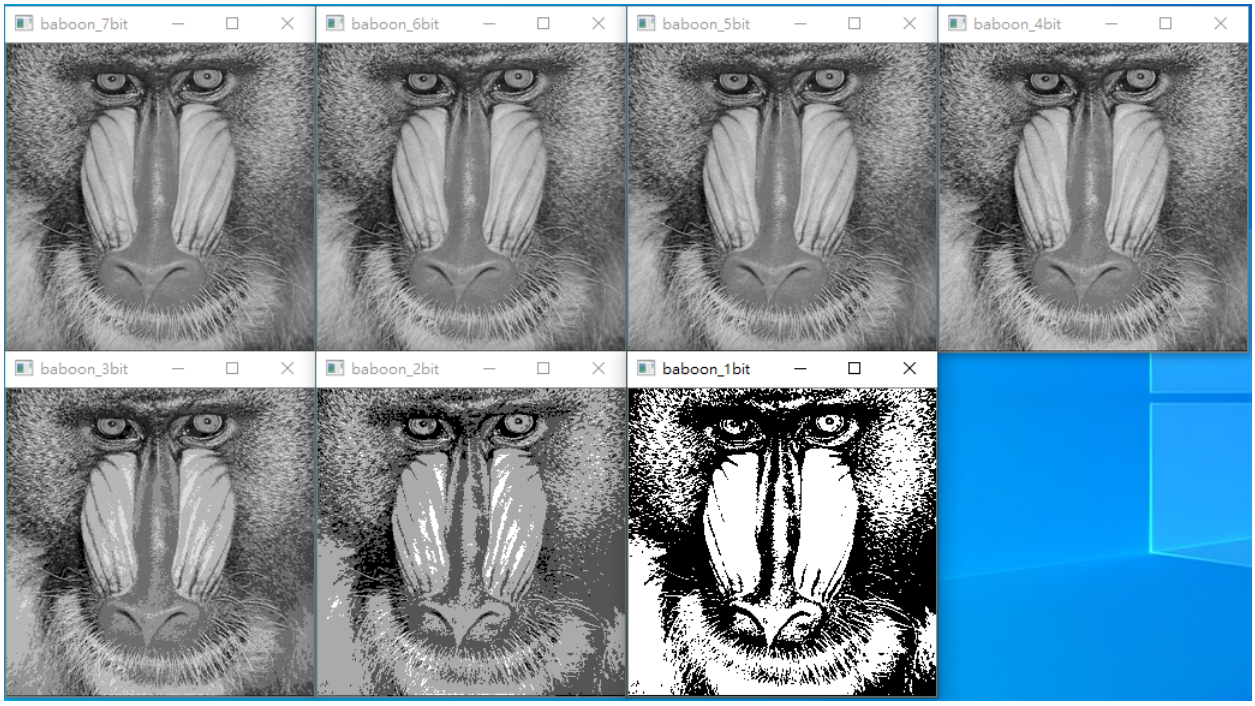
程式碼:

```
vector<vector<BYTE>> ps;
for (int i = 0; i < height1; i=i+2)
{
    vector<BYTE> v1;
    for (int j = 0; j < width1; j=j+2)
    {
        v1.push_back(in[i][j]);
    }
    ps.push_back(v1);
}
```

先創立大小為 1/2 之矩陣，然後每兩個座標取一個 pixel。

# 2.8Bit->1Bit


```
1.      mse: 0.50       psnr: 51.13
2.      mse: 2.00       psnr: 45.13
3.      mse: 7.65       psnr: 39.30
4.      mse: 33.74      psnr: 32.85
5.      mse: 149.52     psnr: 26.38
6.      mse: 1044.09    psnr: 17.94
7.      mse: 8522.23    psnr: 8.83
```




```
1.      mse: 0.50       psnr: 51.11
2.      mse: 1.79       psnr: 45.59
3.      mse: 6.96       psnr: 39.71
4.      mse: 29.47      psnr: 33.44
5.      mse: 139.23     psnr: 26.69
6.      mse: 759.58     psnr: 19.33
7.      mse: 9678.37    psnr: 8.27
```

Lena 在大概 4bit 的時候開始會失真，baboon 大概在 3bit 才失真。原因是因為 lena 的圖片平滑的部分較多所以少一個 bit 會造成圖片看起來較為失真，而 baboon 的圖片因為圖片的細節較多(高頻成分)，所以減少 bit 數也較不會看出差異。

而 MSE 的部分是計算輸出與原圖預測值和實際觀測值間差的平方的均值，數字越大差異越多。

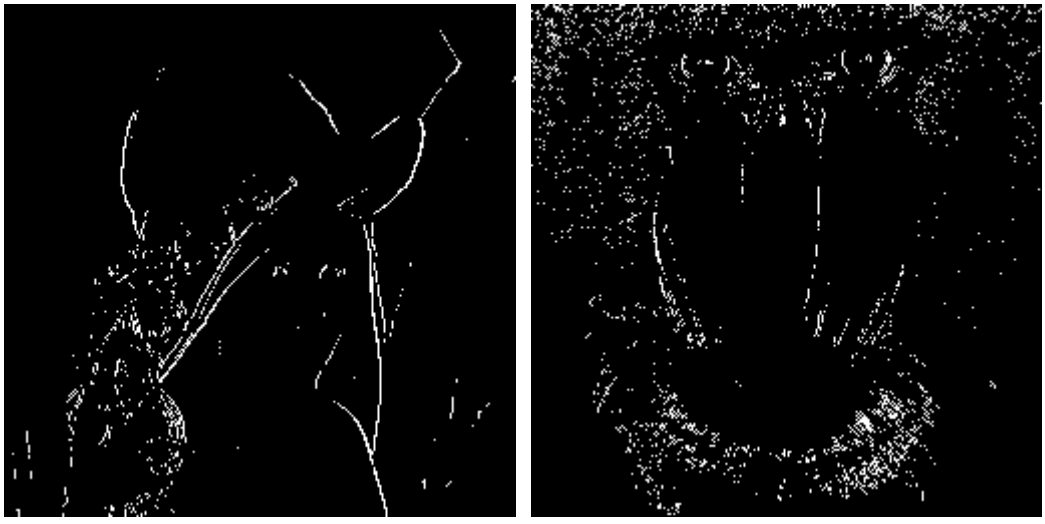
PSNR 原圖像與被處理圖像之間的均方誤差相對於(2^n-1)^2 的對數值(信號最大值的平方，n 是每個採樣值的比特數)，它的單位是 dB。PSNR 越大越好。

# 3.Edge

## (a) Var=25，Right

在閥值=25 的時候，因為是跟右邊的 pixel 做比較，所以會映照出較多垂直方向的 edge 線條。

## (b) Var=50，Right



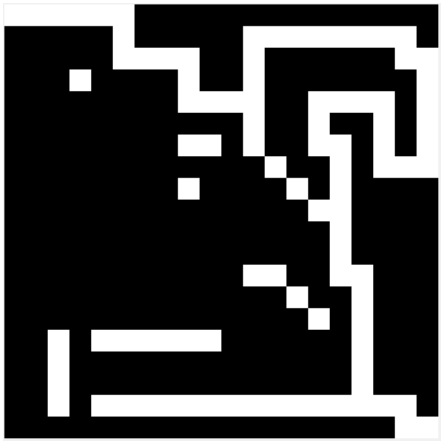
在閥值=50 的時候，因為所需的閥值提高所以差異較大者才會被顯示出來，一樣是跟右邊比較所以會顯示出較為垂直的線。

## (c) Var=25，lower

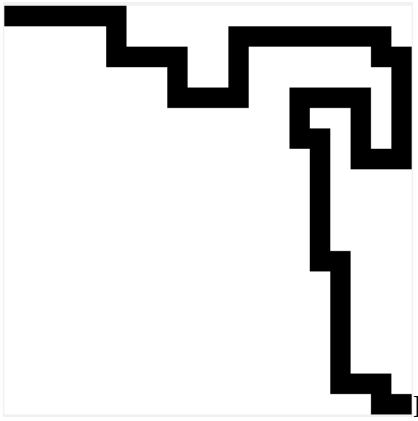

因為跟下面的像素比較，所以會顯示出較為水平的線段，像是鼻子不分的線段就消失了。

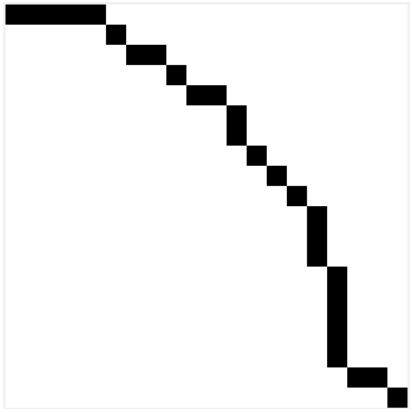
# 4. Distance and Path

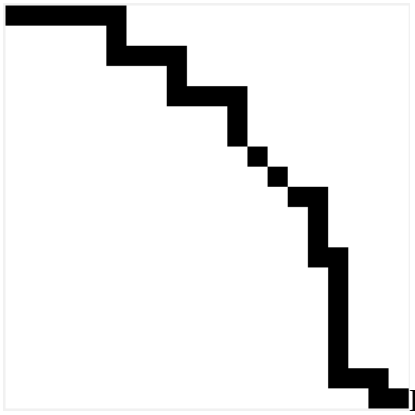
## (a) V set = {85}



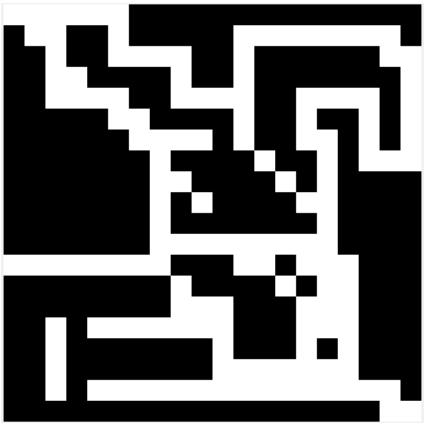
閥值為 85 可以走的路線

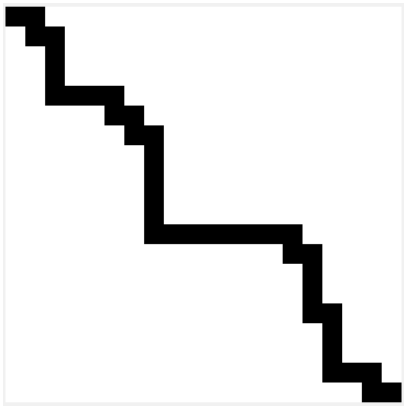

D4 輸出圖，步數:61。

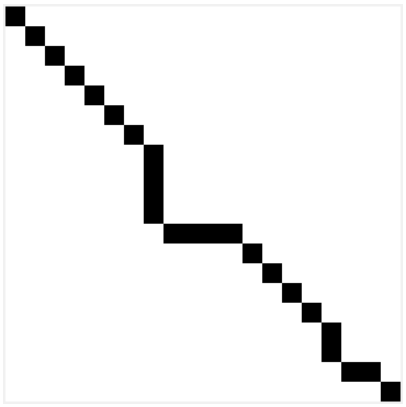

D8 輸出圖，步數:27。

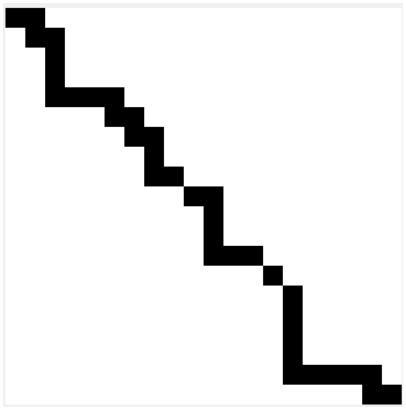

Dm 輸出圖，步數:36。

## (b) V set = {85,170}

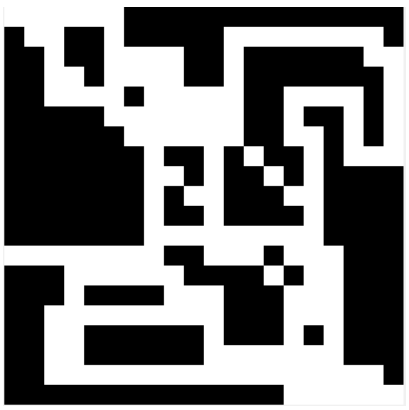

閥值為 85、170 可以走的路線

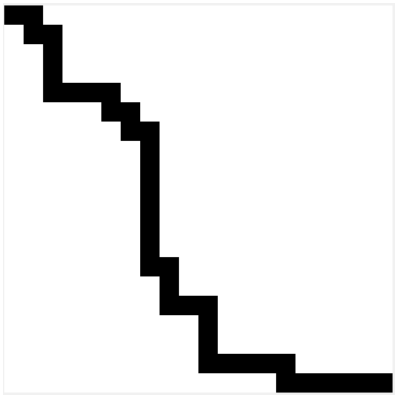

D4 輸出圖，步數:39。

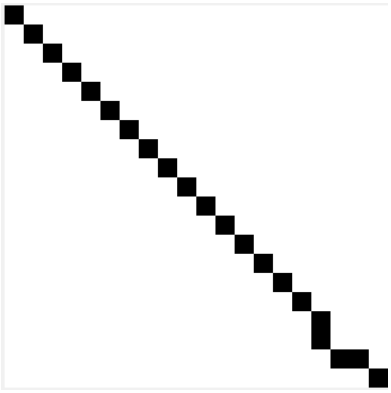

D8 輸出圖，步數:24。

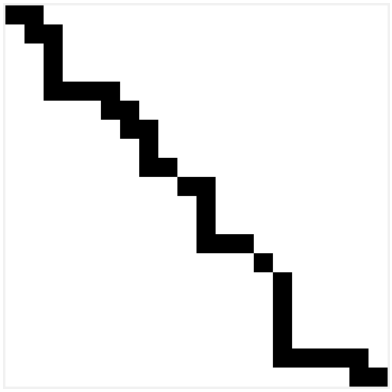

Dm 輸出圖，步數:37。

## (c) V set = {85,170,255}


閥值為 85、170、255 可以走的路線


D4 輸出圖，步數:39。


D8 輸出圖，步數:21。


Dm 輸出圖，步數:36。

- Discussion

  D4 的路線是最長的，D8 的路線是最短的，D4 是只拜訪上下左右四個近鄰的點去找路線，D8 是找周圍 8 個點去找路線，而 Dm 是如果上下左右有就納入路線(不包含已拜訪過的點)如果沒有才考慮左上、左下、右上、右下四個點。