

插值法的程式實作與影像進階應用

組員：蔡承翰、王家衛、蔣承燁、吳宗桓

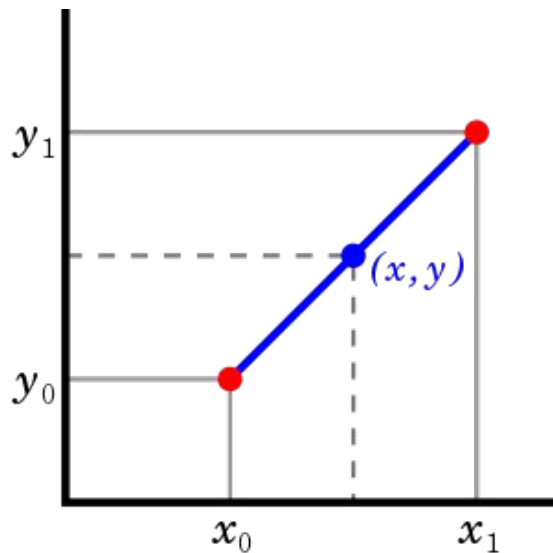
目錄

- 研究背景與動機
- 研究方法的比較
- 數值實驗
- 討論與結論
- 參考資料
- 組員貢獻

研究背景與動機

研究背景與動機

1. **插值法**: 透過已知的數據來預測未知的數據。
2. **圖␣放␣**: 透過原有圖␣的像素點, 來找出新插␣像素點的顏␣。
3. 可利␣插值法來放␣圖␣。



文獻回顧

Prof. PankajS. Parsania¹, Dr. Paresh V.Virparia²,

“A Review: Image Interpolation Techniques for Image Scaling”



ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

**International Journal of Innovative Research in Computer
and Communication Engineering**

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 12, December 2014

A Review: Image Interpolation Techniques for Image Scaling

Prof. PankajS. Parsania¹, Dr. Paresh V.Virparia²

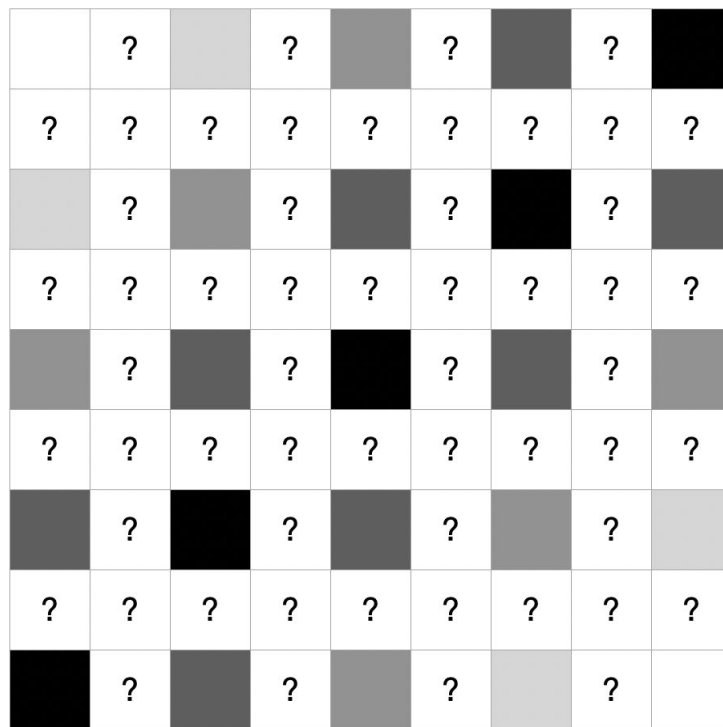
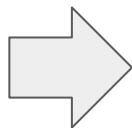
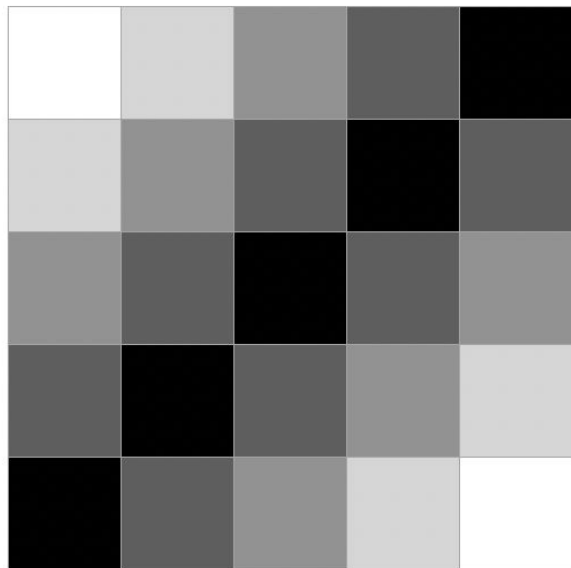
Assistant Professor, College of FPTBE, Anand Agricultural University, Anand, Gujarat, India¹

Director, Department of Computer Science, Sardar Patel University, VallabhVidyanagar, Gujarat, India²

文獻回顧

1. 圖片放大的插值方式分為 Non-Adaptive 與 Adaptive 兩種。
 - a. Non-Adaptive: **直接針對像素點做操作**, 不考慮圖中的內容或是任何的特徵。
例如: 最近鄰插法、雙線性插值法、雙三次插值法。
 - b. Adaptive: **考慮到強度、邊緣資訊、字等額外的特徵**。
例如: Context aware image resizing、Segmentation-Based Approach、Seam Carving。
2. Non-Adaptive 計算量少, Adaptive 效果佳。

研究議題說明



我們將針對插值法做研究，並比較各插值法的差異。

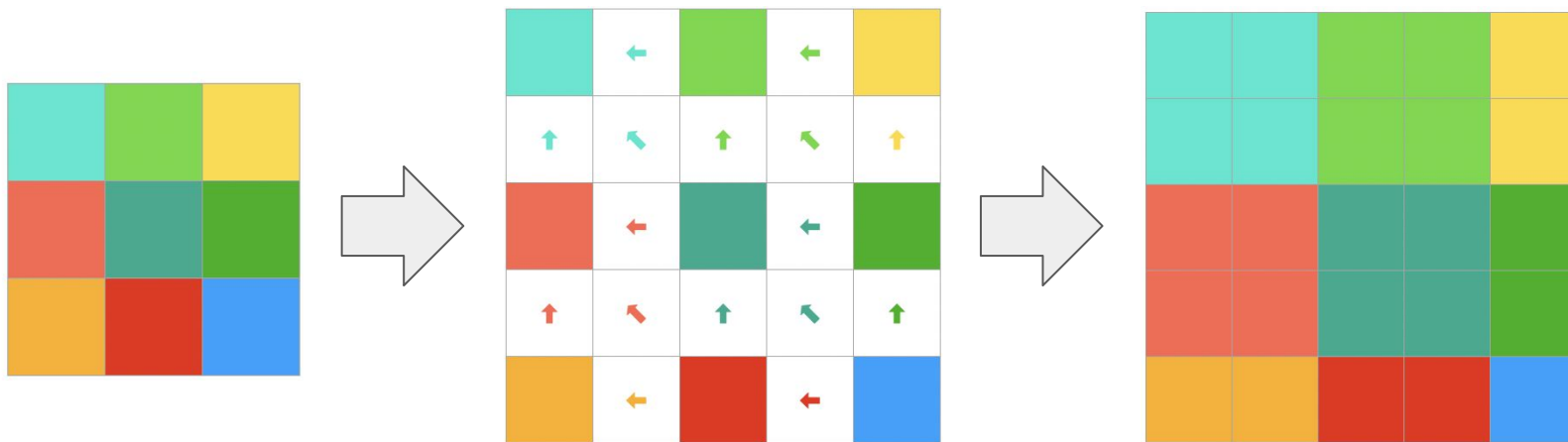
研究方法的比較

問題結論猜測

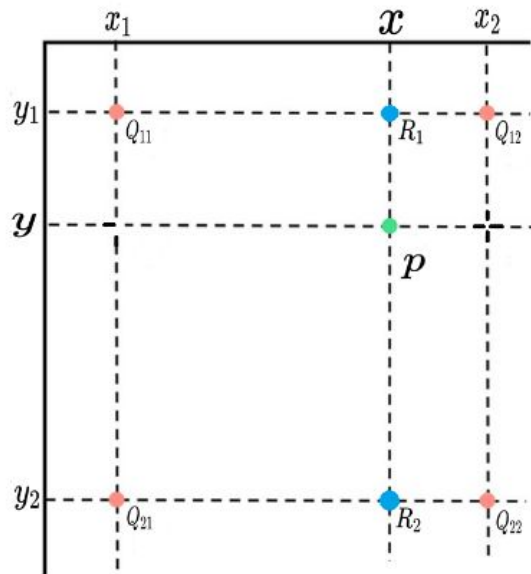
1. **龍格現象**:需要挑選用來做插值的像素點。
2. 使用**缺失點附近**的像素點做插值。
3. **同時考慮兩軸**上的像素點來進行插值。

龍格現象:高次多項式插值不一定總能提高插值的準確度。

最近鄰插法 (Nearest-Neighbor Interpolation)



雙線性插值法 (Bilinear Interpolation)



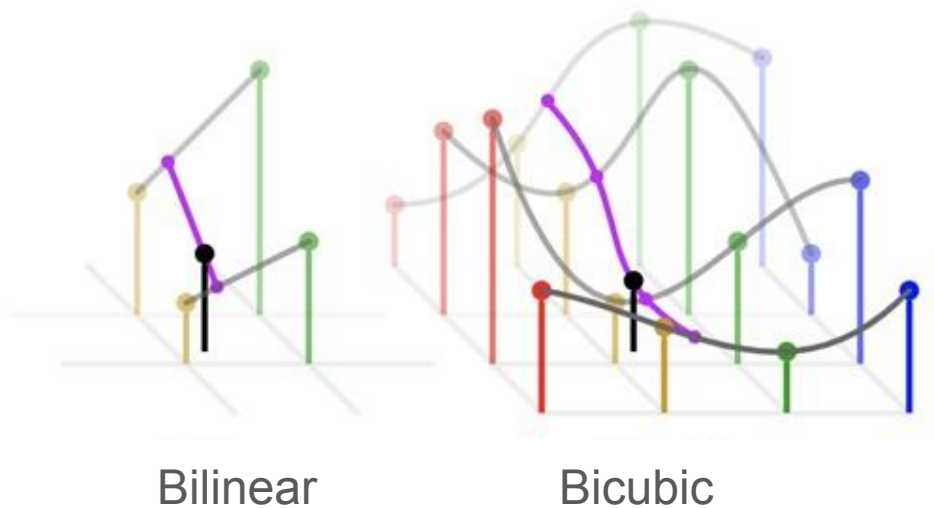
$$R_1 : f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{12})$$

$$R_2 : f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{21}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$P : f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

$$P : f(x, y) \approx \frac{1}{(x_2 - x_1)(y_2 - y_1)} (f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{12})(x - x_1)(y_2 - y) + f(Q_{21})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1))$$

雙三次插值法 (Bicubic Interpolation)



$$P(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j$$

透過以下方程求解 16 個未知係數：

$$\begin{cases} f(\hat{x}_0, \hat{y}_0) = P(\hat{x}_0, \hat{y}_0) \\ f(\hat{x}_0, \hat{y}_1) = P(\hat{x}_0, \hat{y}_1) \\ f(\hat{x}_0, \hat{y}_2) = P(\hat{x}_0, \hat{y}_2) \\ f(\hat{x}_0, \hat{y}_3) = P(\hat{x}_0, \hat{y}_3) \\ \dots \\ f(\hat{x}_3, \hat{y}_1) = P(\hat{x}_3, \hat{y}_1) \end{cases}$$

雙三次插值法 (Bicubic Interpolation)

在本次報告中,

我們使用 Cubic Convolution Algorithm 來產生 Bicubic Interpolation function。

參考文獻:

ROBERT G. KEYS, “Cubic Convolution Interpolation for Digital Image Processing”

雙三次插值法 (Bicubic Interpolation)

Cubic Convolution Algorithm:

一維函數的插值: $f(x) \approx g(x) = \sum_k c_k u\left(\frac{x - x_k}{h}\right)$

c_k parameters depends on sample data

u interpolation kernel

x_k : interpolation nodes

h sampling increment

雙三次插值法 (Bicubic Interpolation)

$$f(x) \approx g(x) = \sum_k c_k u\left(\frac{x - x_k}{h}\right) \quad \text{其中,} \quad u(s) = \begin{cases} A_1|s|^3 + B_1|s|^2 + C_1|s| + D_1, & \text{if } 0 < |s| < 1 \\ A_2|s|^3 + B_2|s|^2 + C_2|s| + D_2, & \text{if } 1 < |s| < 2 \\ 0, & \text{if } 2 < |s| \end{cases}$$

為了方便計算, 假設 $u(0) = 1, u(n) = 0, \forall n \in \mathbb{Z} - \{0\}$

根據插值法的 conditions: $g(x_j) = f(x_j) \implies c_j = f(x_j)$

雙三次插值法 (Bicubic Interpolation)

$$f(x) \approx g(x) = \sum_k c_k u\left(\frac{x - x_k}{h}\right) \quad \text{其中,} \quad u(s) = \begin{cases} A_1|s|^3 + B_1|s|^2 + C_1|s| + D_1, & \text{if } 0 < |s| < 1 \\ A_2|s|^3 + B_2|s|^2 + C_2|s| + D_2, & \text{if } 1 < |s| < 2 \\ 0, & \text{if } 2 < |s| \end{cases}$$

共 8 個未知數, 透過以下線性方程組求解:

$$\begin{cases} u(0) = 1 \\ u(1^-) = 0 \\ u(1^+) = 0 \\ u(2^-) = 0 \\ u'(0^-) = u'(0^+) \\ u'(1^-) = u'(1^+) \\ u'(2^-) = u'(2^+) \end{cases} \quad + \quad A_2 = a \quad = \quad u(s) = \begin{cases} (a+2)|s|^3 - (a+3)|s|^2 + 1, & \text{if } 0 < |s| < 1 \\ a|s|^3 - 5a|s|^2 + 8a|s| - 4a, & \text{if } 1 < |s| < 2 \\ 0, & \text{if } 2 < |s| \end{cases}$$

constraint

根據文獻, $a = -0.5$ 的誤差最小

雙三次插值法 (Bicubic Interpolation)

$$f(x) \approx g(x) = \sum_k c_k u\left(\frac{x - x_k}{h}\right) \quad \text{其中,} \quad u(s) = \begin{cases} A_1|s|^3 + B_1|s|^2 + C_1|s| + D_1, & \text{if } 0 < |s| < 1 \\ A_2|s|^3 + B_2|s|^2 + C_2|s| + D_2, & \text{if } 1 < |s| < 2 \\ 0, & \text{if } 2 < |s| \end{cases}$$

兩插值點 x_j, x_k 之間的距離為 $(j - k)h$ 則

$$x \in (x_j, x_{j+1}) \implies s = \frac{x - x_j}{h} \implies \frac{x - x_k}{h} = \frac{x - x_j + x_j + x_k}{h} = s + j - k$$

雙三次插值法 (Bicubic Interpolation)

$$\begin{aligned}f(x) &\approx g(x) = \sum_k c_k u(s + j - k) \\&= c_{j-1}u(s+1) + c_j u(s) + c_{j+1}u(s-1) + c_{j+2}u(s-2) \\&= u(s+1)f(x_{j-1}) + u(s)f(x_j) + u(s-1)f(x_{j+1}) + u(s-2)f(x_{j+2}) \\&= \begin{bmatrix} u(s+1) & u(s) & u(s-1) & u(s-2) \end{bmatrix} \begin{bmatrix} f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \\ f(x_{j+2}) \end{bmatrix}\end{aligned}$$

雙三次插值法 (Bicubic Interpolation)

考慮二維函數： $f(x, y) \approx g(x, y) = \sum_i \sum_j c_{ij} u\left(\frac{x - x_i}{h}\right) u\left(\frac{y - y_j}{h}\right)$

$$f(i + s, j + t) \approx \sum_{x=-1}^2 \sum_{y=-1}^2 f(i + x, j + y) u(s - x) u(t - y)$$

$$= \begin{bmatrix} u(s+1) & u(s) & u(s-1) & u(s-2) \end{bmatrix} \begin{bmatrix} f(i-1, j-1) & f(i-1, j) & f(i-1, j+1) & f(i-1, j+2) \\ f(i, j-1) & f(i, j) & f(i, j+1) & f(i, j+2) \\ f(i+1, j-1) & f(i+1, j) & f(i+1, j+1) & f(i+1, j+2) \\ f(i+2, j-1) & f(i+2, j) & f(i+2, j+1) & f(i+2, j+2) \end{bmatrix} \begin{bmatrix} u(t+1) \\ u(t) \\ u(t-1) \\ u(t-2) \end{bmatrix}$$

拉格朗日插值法 (Lagrange Interpolation)

對某個多項式函數，已知有 $k+1$ 個取值點 $(x_0, y_0), \dots, (x_k, y_k)$

其中 x_j 對映著自變數的位置，而 y_j 對映著函數在這個位置的取值。

假設任意兩個不同的 x_j 都互不相同，那麼拉格朗日差值公式所得到的多項式為：

$$P_n(x_i) = \sum_{k=0}^n y_k \varphi_k(x_i) = y_i, i \neq 0$$

其中每個 $\varphi_k(x)$ 為拉格朗日基本多項式，其表達式為：

$$\varphi_k(x) = \prod_{j=0, j \neq k}^n \frac{x - x_j}{x_k - x_j}, k = 0, \dots, n$$

數值實驗

程式語言與套件挑選

我們將使用 **Python** 來進行數值實驗，使用到的套件如下：

- **numpy**: 圖陣列的實作。
- **opencv**: 圖檔檔案 I/O 操作。
- **matplotlib**: 將資料視覺化庫，用於建立高品質的靜態、動態和互動式圖形。

數值實驗：事前準備

我們先建立以下共識：

1. 將大小為 $m * n$ 的圖片視為一個 $m * n$ 的矩陣。
2. 以左上角為座標 $(0, 0)$ 。
3. X 軸向右遞增、Y 軸向下遞增。

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)
(0, 4)	(1, 4)	(2, 4)	(3, 4)	(4, 4)

最近鄰插法 (Nearest-Neighbor Interpolation)

Step 1. 取得原圖片長寬與放大後圖片長寬。

Step 2. 計算放大後圖片上像素點對應到原圖片的位置 P。

Step 3. 找出離 P 最靠近的像素點，並將其值賦予給 P。

```
# Interpolation
for dst_x in range(dst_height):
    for dst_y in range(dst_width):
        x, y = dst_x / magni, dst_y / magni
        src_x, src_y = int(x), int(y)

        if x - src_x >= 0.5:
            src_x += 1
        if y - src_y >= 0.5:
            src_y += 1

        dst[dst_x, dst_y] = src[src_x, src_y]
```


雙線性插值法 (Bilinear Interpolation)

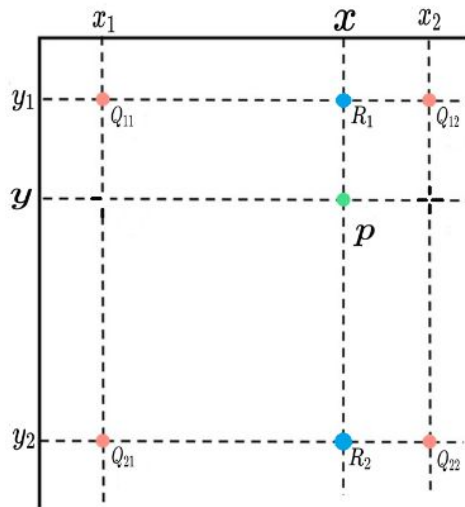
Step 1. 取得原圖片長寬與放大後圖片長寬。

Step 2. 計算放大後圖片上像素點對應到原圖片的位置 P 。

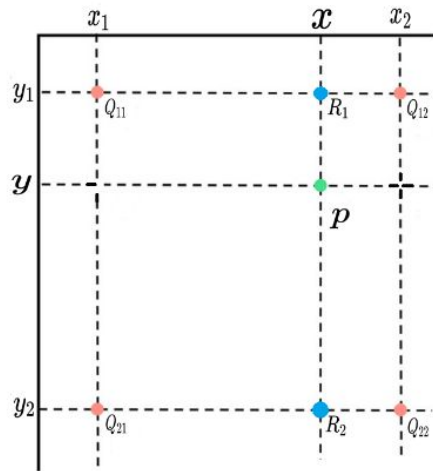
Step 3. 找出 Q_{11} , Q_{12} , Q_{21} , Q_{22} 四個點。

Step 4. 使用以下公式對 P 進行雙線性插值。

$$P : f(x, y) \approx \frac{1}{(x_2 - x_1)(y_2 - y_1)} (f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{12})(x - x_1)(y_2 - y) + f(Q_{21})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1))$$



雙線性插值法 (Bilinear Interpolation)



為簡化計算, 設定:

Q11 為 (0, 0)、Q12 為 (0, 1)、Q21 為 (1, 0)、Q22 為 (1, 1)。

$$\begin{aligned} P : f(x, y) &\approx \frac{1}{(x_2 - x_1)(y_2 - y_1)} (f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{12})(x - x_1)(y_2 - y) \\ &\quad + f(Q_{21})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1)) \\ &= (1 - x)(1 - y)f(Q_{11}) + x(1 - y)f(Q_{12}) + (1 - x)yf(Q_{21}) + xyf(Q_{22}) \end{aligned}$$

雙線性插值法 (Bilinear Interpolation)

$$P : f(x, y) = (1 - x)(1 - y)f(Q_{11}) + x(1 - y)f(Q_{12}) + (1 - x)yf(Q_{21}) + xyf(Q_{22})$$

```
# Interpolation
for dst_x in range(dst_height):
    for dst_y in range(dst_width):
        x, y = dst_x / magni, dst_y / magni
        src_x, src_y = int(x), int(y)

        dst[dst_x, dst_y] = (src_x + 1 - x) * (src_y + 1 - y) * src[src_x, src_y] + \
            (x - src_x) * (src_y + 1 - y) * src[src_x + 1, src_y] + \
            (src_x + 1 - x) * (y - src_y) * src[src_x + 1, src_y] + \
            (x - src_x) * (y - src_y) * src[src_x + 1, src_y + 1]
```

雙三次插值法 (Bicubic Interpolation)

Step 1. 取得原圖片長寬與放大後圖片長寬。

Step 2. 計算放大後圖片上像素點對應到原圖片的位置 P 。

Step 3. 計算 interpolation kernel 和矩陣乘法。

雙三次插值法 (Bicubic Interpolation)

Interpolation kernel:

$$u(s) = \begin{cases} (a+2)|s|^3 - (a+3)|s|^2 + 1, & \text{if } 0 < |s| < 1 \\ a|s|^3 - 5a|s|^2 + 8a|s| - 4a, & \text{if } 1 < |s| < 2 \\ 0, & \text{if } 2 < |s| \end{cases}$$

```
def W(_x):  
    a = -0.5  
    x = abs(_x)  
    if x <= 1:  
        return (a + 2) * (x ** 3) - (a + 3) * (x ** 2) + 1  
    elif x < 2:  
        return a * (x ** 3) - 5 * a * (x ** 2) + 8 * a * x - 4 * a  
    else:  
        return 0
```

雙三次插值法 (Bicubic Interpolation)

矩陣乘法:

$$f(i+s, j+t) \approx \sum_{x=-1}^2 \sum_{y=-1}^2 f(i+x, j+y) u(s-x) u(t-y)$$

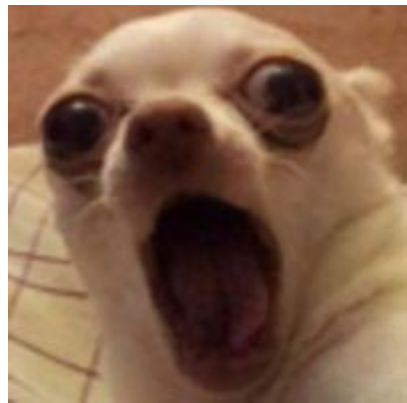
$$= \begin{bmatrix} u(s+1) & u(s) & u(s-1) & u(s-2) \end{bmatrix} \begin{bmatrix} f(i-1, j-1) & f(i-1, j) & f(i-1, j+1) & f(i-1, j+2) \\ f(i, j-1) & f(i, j) & f(i, j+1) & f(i, j+2) \\ f(i+1, j-1) & f(i+1, j) & f(i+1, j+1) & f(i+1, j+2) \\ f(i+2, j-1) & f(i+2, j) & f(i+2, j+1) & f(i+2, j+2) \end{bmatrix} \begin{bmatrix} u(t+1) \\ u(t) \\ u(t-1) \\ u(t-2) \end{bmatrix}$$

```
def bicubic_matmul(src, src_x, src_y, u, v):  
    A = np.array([W(u + 1), W(u), W(u - 1), W(u - 2)])  
    A = A[np.newaxis, :]  
  
    B = src[src_x: src_x + 4, src_y: src_y + 4]  
  
    C = np.array([W(v + 1), W(v), W(v - 1), W(v - 2)])  
    C = C[:, np.newaxis]  
  
    dst = np.zeros(1, dtype=np.float32)  
  
    tmp = np.matmul(np.matmul(A, B[:, :]), C)  
    if tmp > 255:  
        dst = 255  
    elif tmp < 0:  
        dst = 0  
    else:  
        dst = tmp  
  
    return dst
```

雙三次插值法 (Bicubic Interpolation)

```
# Interpolation
for dst_x in range(dst_height):
    for dst_y in range(dst_width):
        x, y = dst_x / magni, dst_y / magni
        src_x, src_y = int(x), int(y)
        u, v = x - src_x, y - src_y
        dst[dst_x, dst_y] = bicubic_matmul(src, src_x, src_y, u, v)
```

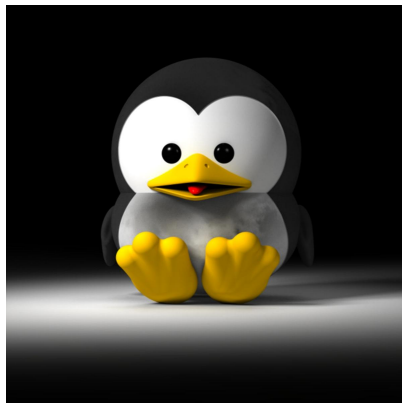
實驗結果



100 * 100



316 * 316

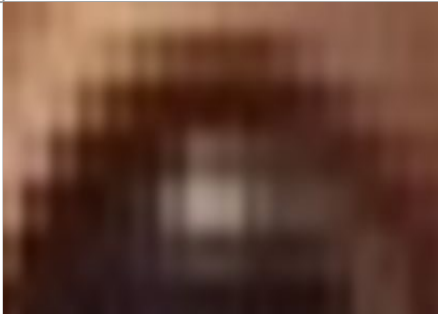







600 * 600



1600 * 1200

實驗結果：放大 2 倍效果比較

	Nearest	Bilinear	Bicubic
100 * 100			
316 * 316			

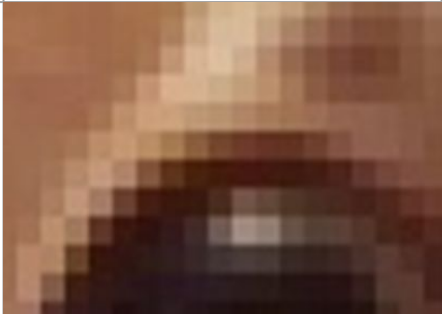
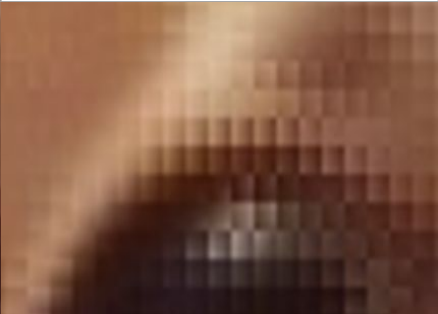
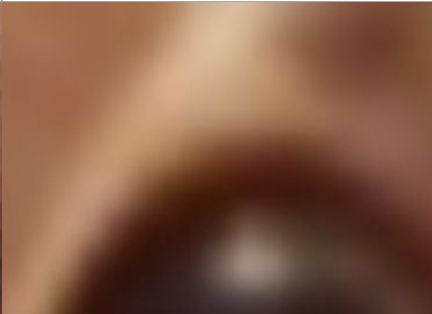

實驗結果：放大 2 倍效果比較

	Nearest	Bilinear	Bicubic
600 * 600			
1600 * 1200			







實驗結果：放大 2 倍花費時間比較 (單位：秒)

	Nearest	Bilinear	Bicubic
100 * 100	0.021	0.212	0.677
316 * 316	0.262	1.591	6.100
600 * 600	0.775	5.836	22.059
1600 * 1200	4.012	30.285	117.490

實驗結果：放大 4 倍效果比較

	Nearest	Bilinear	Bicubic
100 * 100			
316 * 316			

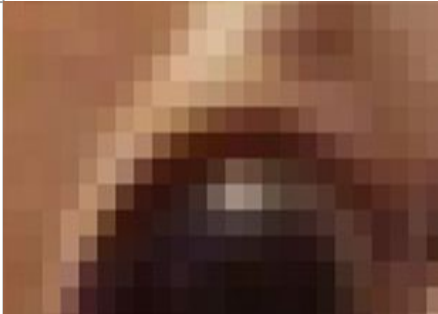
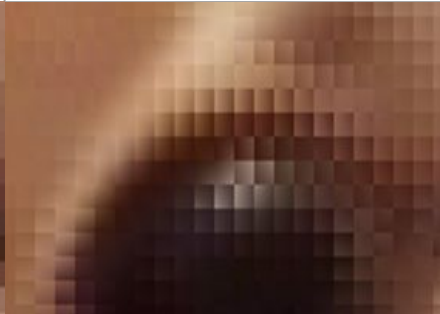




實驗結果：放大 4 倍效果比較

	Nearest	Bilinear	Bicubic
600 * 600			
1600 * 1200			







實驗結果：放大 4 倍花費時間比較 (單位：秒)

	Nearest	Bilinear	Bicubic
100 * 100	0.123	0.664	2.547
316 * 316	0.840	6.206	25.407
600 * 600	2.917	22.437	89.001
1600 * 1200	15.456	121.878	479.874

實驗結果：放大 8 倍效果比較

	Nearest	Bilinear	Bicubic
100 * 100			
316 * 316			

實驗結果：放大 8 倍效果比較

	Nearest	Bilinear	Bicubic
600 * 600			
1600 * 1200			

實驗結果：放大 8 倍花費時間比較 (單位：秒)

	Nearest	Bilinear	Bicubic
100 * 100	0.381	2.504	9.962
316 * 316	3.249	24.914	99.087
600 * 600	11.383	89.756	359.214
1600 * 1200	61.681	485.967	1923.200

拉格朗日插值法 (Lagrange Interpolation)

```
5 def lagrange_interpolation(x, y, target_x):
6     result = 0
7     for i in range(len(y)):
8         term = y[i]
9         for j in range(len(x)):
10             if j != i:
11                 term = term * (target_x - x[j]) / (x[i] - x[j])
12         result += term
13     return result
```

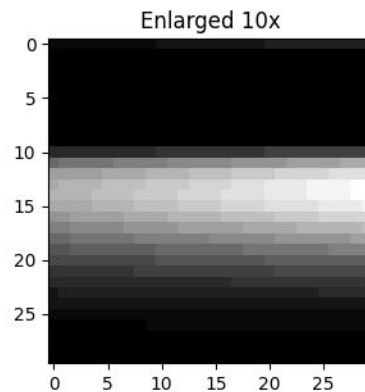
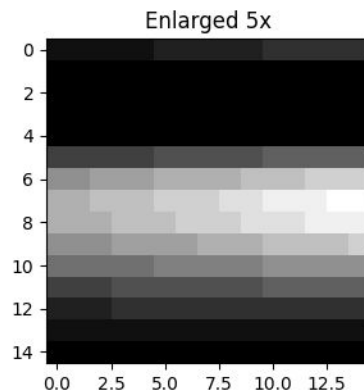
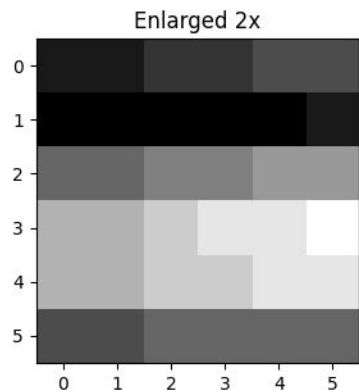
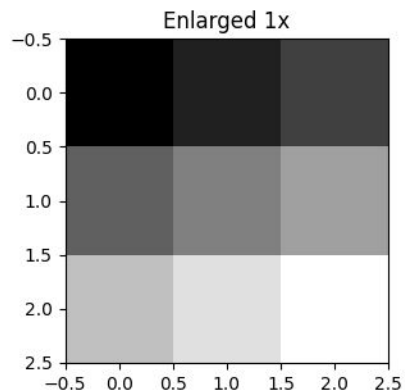
拉格朗日插值法 (Lagrange Interpolation)

```
# 先對 x 軸進行拉格朗日插值
x_enlarged = np.zeros((height, int(width * scale_factor)))
for i in range(height):
    for j in range(int(width * scale_factor)):
        x = j / scale_factor
        x_enlarged[i, j] = lagrange_interpolation(np.arange(width), original_image[i, :], x)

# 再對 y 軸進行拉格朗日插值
y_enlarged = np.zeros((int(height * scale_factor), int(width * scale_factor)))
for i in range(int(height * scale_factor)):
    for j in range(int(width * scale_factor)):
        y = i / scale_factor
        y_enlarged[i, j] = lagrange_interpolation(np.arange(int(width * scale_factor)), x_enlarged[:, j], y)
```

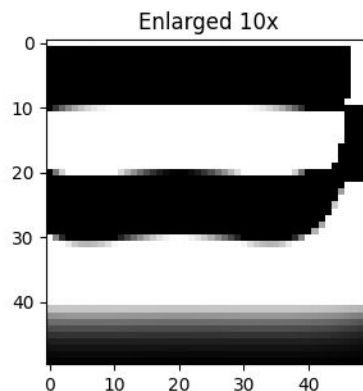
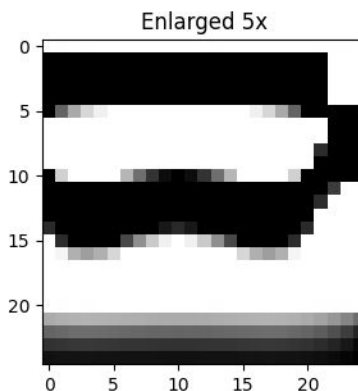
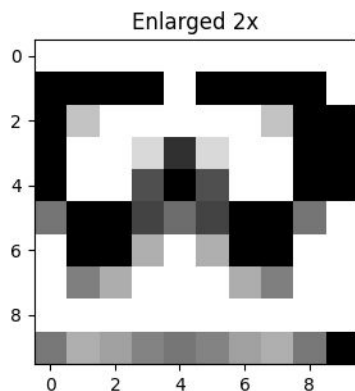
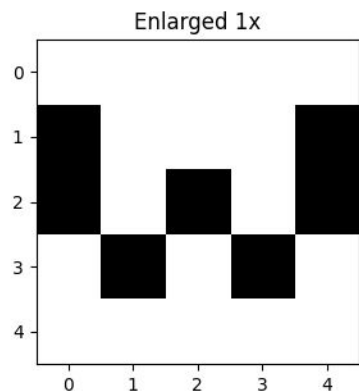
拉格朗日插值法 (Lagrange Interpolation): 漸層圖片

圖片大小: 3 * 3



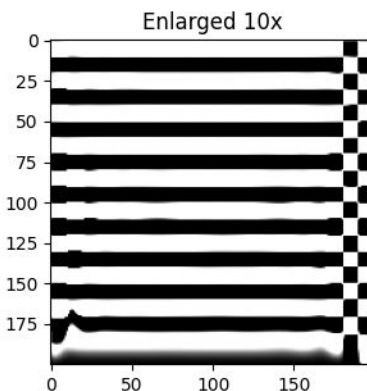
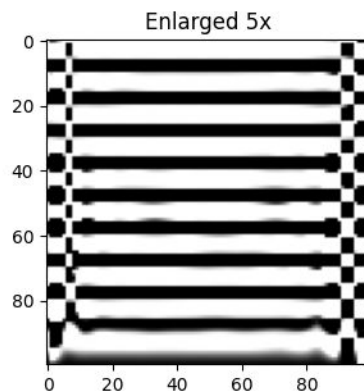
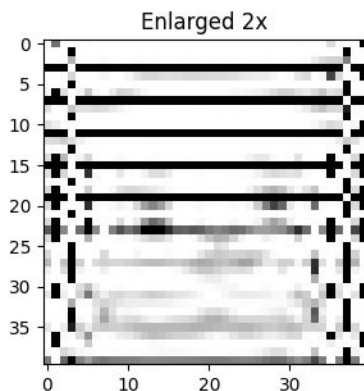
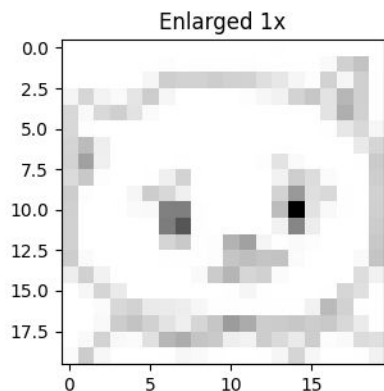
拉格朗日插值法 (Lagrange Interpolation): 高變化圖片

圖片大小: $5 * 5$



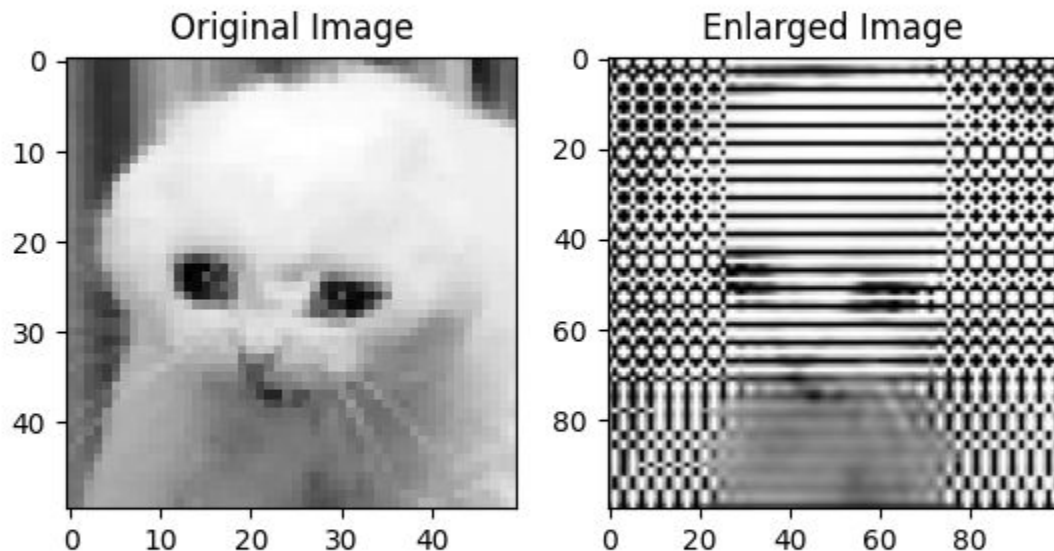
拉格朗日插值法 (Lagrange Interpolation): 一般圖片(1)

圖片大小: 20 * 20



拉格朗日插值法 (Lagrange Interpolation): 一般圖片(2)

圖片大小: 50 * 50 (放大 2 倍)



討論與結論

實驗過程的困難與失敗原因？

困難：

- a. **程式不好實現**: 二維拉格朗日難以用程式呈現出來, 因此改用x 軸 y 軸分別放大
- b. **執行時間過長**: 光是最後一張 $50 * 50$ 像素的圖片放大 2 倍就需要花費大約 2 分鐘, 由於時間太長因此實際應用非常不切實際。

(時間複雜度為 $O(h^2 * w + h * w^2)$ 其中 h 和 w 分別為圖片的高和寬)

實驗過程的困難與失敗原因？

失敗原因：

- a. **計算複雜度高**: 拉格朗日插值的計算複雜度隨著資料點的增加而增加，因為需要計算每個內插點的內插多項式。對於大型影像，這可能導致效能問題。
- b. **過度擬合 (overfitting)**: 內插多項式的高次數可能導致過度擬合，尤其是在有雜訊的影像中。這可能會使得插值結果對原始資料過於敏感。
- c. **振鈴 (Ringing) 效應**: 拉格朗日插值可能會引入振鈴效應，即在影像中出現類似振鈴的現象，導致影像看起來不自然。

結論

- 放大效果：**雙三次插值法 (bicubic)** > 雙線性插值法 > 最近鄰插法。
- 花費時間長度：**雙三次插值法 (bicubic)** > 雙線性插值法 > 最近鄰插法。
- 拉格朗日插值法無論是放大效果和花費時間都比另外三個插值法還要差。

參考資料

- [A Review: Image Interpolation Techniques for Image Scaling](#)
- [Linear Methods for Image Interpolation by Pascal Getreuer](#)
- [Interpolation - 演算法筆記 | 國立臺灣師範大學](#)
- [Lagrange Interpolating Polynomial -- from Wolfram MathWorld](#)
- [Cubic Convolution Interpolation for Digital Image Processing](#)

組員貢獻

	研究理論探討	研究實驗執行	報告撰寫
蔡承翰	25%	35%	25%
王家衛	25%	35%	20%
蔣承燁	25%	20%	25%
吳宗桓	25%	10%	30%