機器視覺 HW1

資工三 109590041 范遠皓

環境

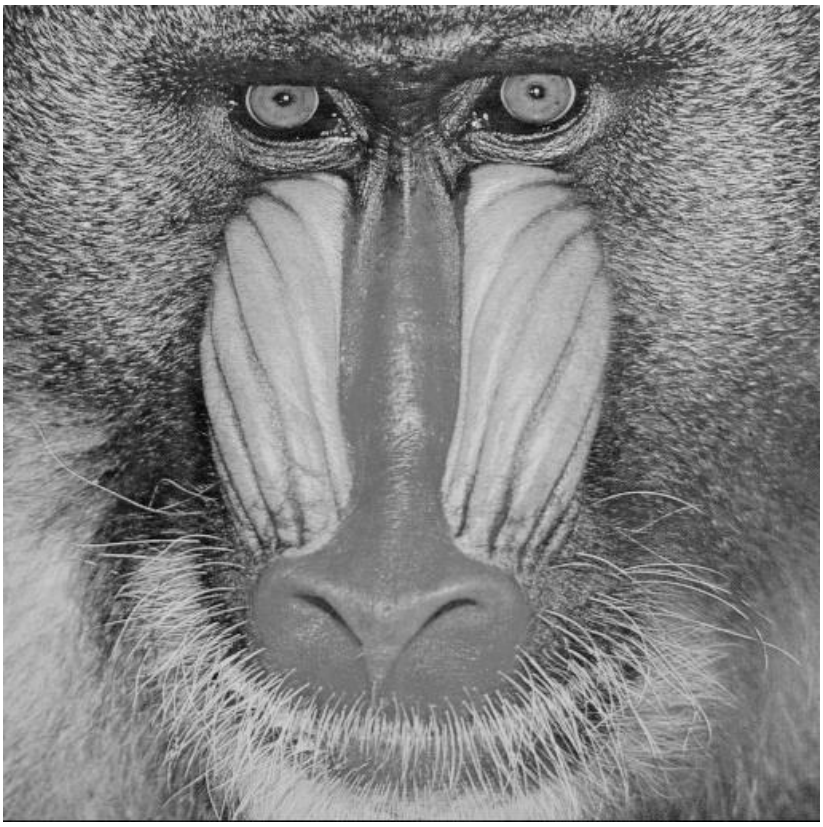
Visual Studio 2019 、 C++ 、 openCV2.4.13.6

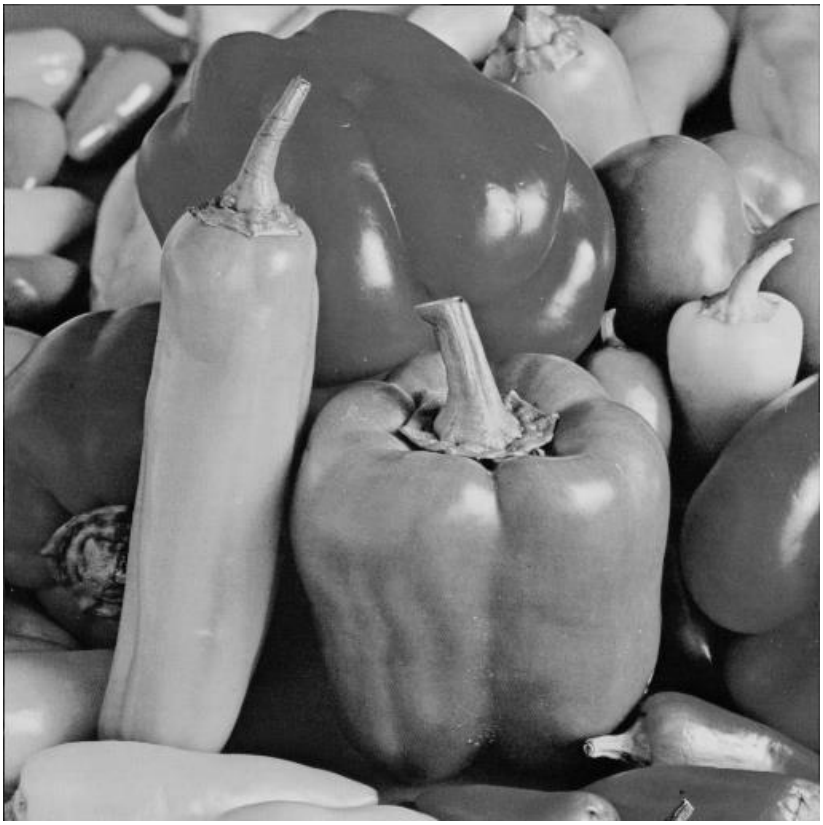
Q1-1

根據題目使用 Formula: (0.3 * R) + (0.59 * G) + (0.11 * B) 將像素乘上一定比例並相加，轉換成灰階值。

```
// 轉灰階
Mat ConvertToGray(Mat colorImage) {
    Mat grayImage(colorImage.size(), CV_8UC1);

    for (int i = 0; i < colorImage.rows; i++) {
        for (int j = 0; j < colorImage.cols; j++) {
            Vec3b pixel = colorImage.at<Vec3b>(i, j);
            int grayValue = 0.3 * pixel[2] + 0.59 * pixel[1] + 0.11 * pixel[0];
            grayImage.at<uchar>(i, j) = grayValue;
        }
    }
    return grayImage;
}
```

Q1-1 成果





Q1-2

將灰階影像二值化 Threshold 設定為 128 ， 像素值大於

Threshold 值則變為 255 小於則為 0。

```
// 灰階二值化
Mat ConvertToBinary(Mat colorImage, uchar threshold = 128) {
    Mat grayImage = this->ConvertToGray(colorImage);
    Mat binaryImage(grayImage.size(), CV_8UC1);

    for (int i = 0; i < grayImage.rows; i++)
        for (int j = 0; j < grayImage.cols; j++)
            binaryImage.at<uchar>(i, j) = grayImage.at<uchar>(i, j) > threshold ? 255 : 0;
    return binaryImage;
}
```

Q1-2 成果

OBJ OBJ

Q1-3

建立 color map

B 4 個值(0, 64, 128, 192)

G 8 個值(0, 32, 64, 96, 128, 160, 192, 224)

R 8 個值(0, 32, 64, 96, 128, 160, 192, 224)

4 * 8 * 8 = 總共 256 種顏色

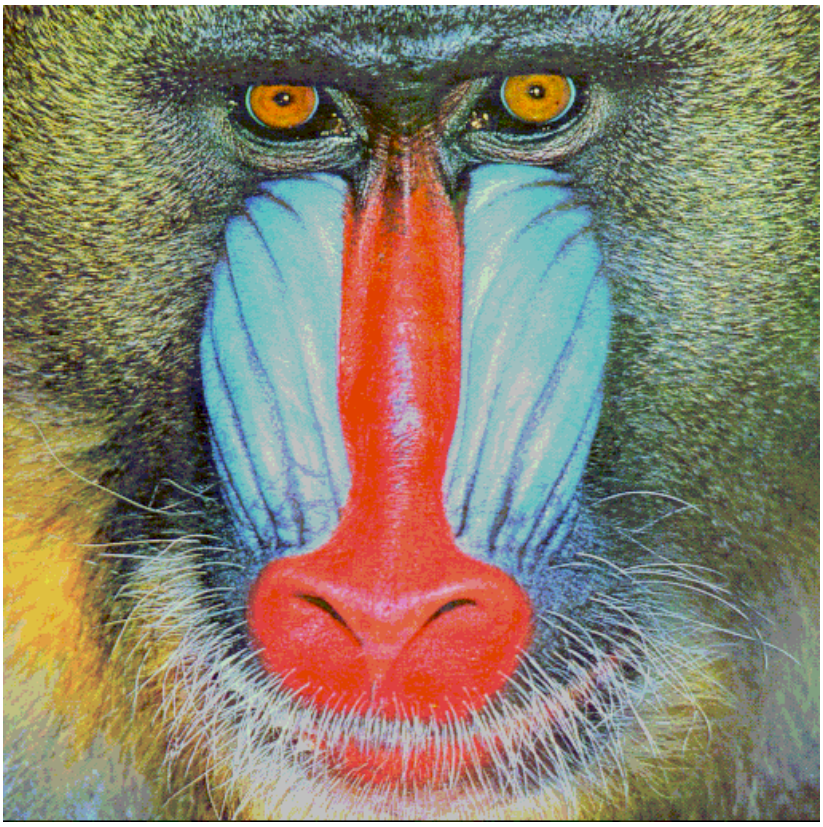
```
class ImageLibrary
{
private:
    Mat _colorMap;

    // 建立 indexed image 的 color map
    void CreateColorMap() {
        _colorMap = Mat(1, 256, CV_8UC3);
        // B 4種, G 8種, R 8種 = 4 * 8 * 8 = 256
        int colorDiv[3] = { 4, 8, 8 };
        for (int i = 0; i < colorDiv[0]; i++)
            for (int j = 0; j < colorDiv[1]; j++)
                for (int k = 0; k < colorDiv[2]; k++) {
                    int index = i * colorDiv[1] * colorDiv[2] + j * colorDiv[2] + k;
                    _colorMap.at<Vec3b>(0, index) = Vec3b(i * (256 / colorDiv[0]), j * (256 / colorDiv[1]), k * (256 / colorDiv[2]));
                }
    }
```

轉換成 indexed-image

尋訪每個像素點，算出像素點與 color map 中像素之差，平方後再加起來取最小值，即為 color map 中最接近的配色。

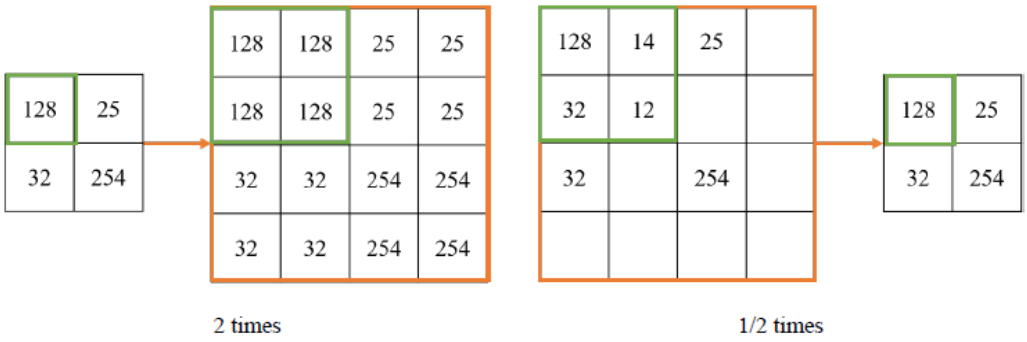
```cpp
// 使用對應表轉成 indexed image
Mat ConvertToIndexedColor(Mat colorImage, Mat* colorMap = nullptr) {
    if (colorMap == nullptr)
        colorMap =  &(this->_colorMap);

    Mat indexImage(colorImage.size(), CV_8UC1);
    Mat mappingImage(colorImage.size(), CV_8UC3);
    const int MAX_DIST = 255.0 * 255.0 * 3.0;
    for (int i = 0; i < colorImage.rows; i++) {
        for (int j = 0; j < colorImage.cols; j++) {
            Vec3b color = colorImage.at<Vec3b>(i, j);
            uchar index = 0;
            int minDist = MAX_DIST;
            for (int k = 0; k < 256; k++) {
                Vec3b mapColor = colorMap->at<Vec3b>(0, k);
                int dist =
                    (color[0] - mapColor[0]) * (color[0] - mapColor[0]) +
                    (color[1] - mapColor[1]) * (color[1] - mapColor[1]) +
                    (color[2] - mapColor[2]) * (color[2] - mapColor[2]);
                if (dist < minDist) {
                    minDist = dist;
                    index = k;
                }
            }
            indexImage.at<uchar>(i, j) = index;
            mappingImage.at<Vec3b>(i, j) = colorMap->at<Vec3b>(0, index);
        }
    }
    return mappingImage;
}
```

Q1-3 成果



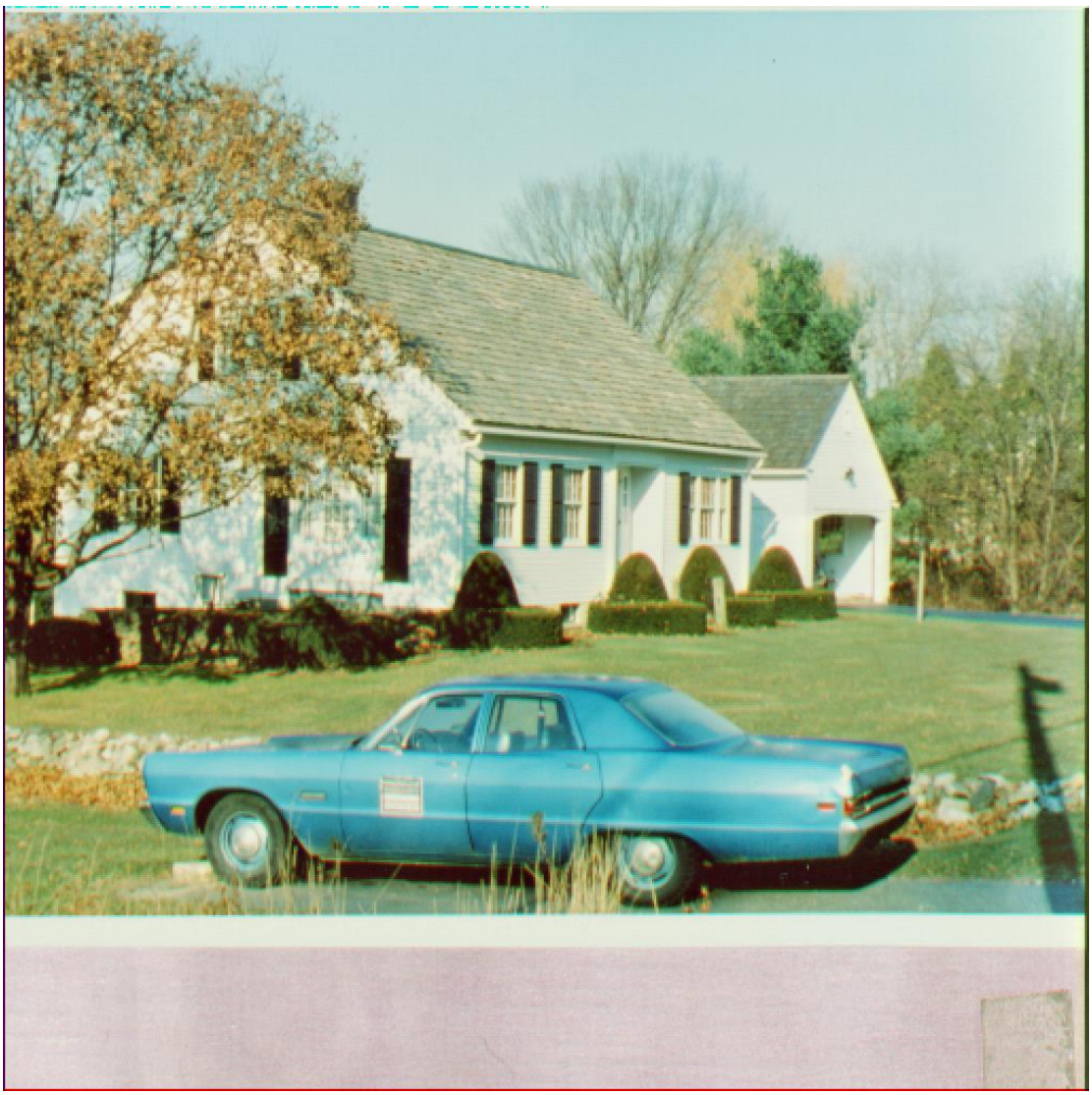
Q2-1

根據題目，放大直接使用左上角的像素填滿，縮小只保留左上角的像素。




```
Mat ResizeWithoutInterpolation(Mat colorImage, int scale, bool zoomIn) {
    Mat resizeImage;
    if (zoomIn) {
        resizeImage = Mat(colorImage.rows * scale, colorImage.cols * scale, CV_8UC3);
        for (int i = 0; i < resizeImage.rows; i++)
            for (int j = 0; j < resizeImage.cols; j++)
                resizeImage.at<Vec3b>(i, j) = colorImage.at<Vec3b>(i / scale, j / scale);
    }
    else {
        resizeImage = Mat(colorImage.rows / scale, colorImage.cols / scale, CV_8UC3);
        for (int i = 0; i < resizeImage.rows; i++)
            for (int j = 0; j < resizeImage.cols; j++)
                resizeImage.at<Vec3b>(i, j) = colorImage.at<Vec3b>(scale * i, scale * j);
    }
    return resizeImage;
}
```

Q2-1 成果 zoom in



zoom out

Q2-2

放大使用 Bilinear Interpolation 進行插值。

參考公式:

假如我們想得到未知函數 f 在點 $P = (x, y)$ 的值，假設我們已知函數 f 在 $Q_{11} = (x_1, y_1)$、$Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$ 以及 $Q_{22} = (x_2, y_2)$ 四個點的值。

首先在 x 方向進行線性插值，得到

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad \text{where} \quad R_1 = (x, y_1),$$
$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad \text{where} \quad R_2 = (x, y_2).$$

然後在 y 方向進行線性插值，得到

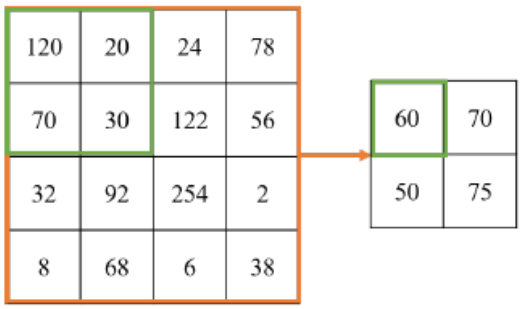
$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

這樣就得到所要的結果 $f(x, y)$，

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

```
Mat ResizeWithInterpolation(Mat colorImage, int scale, bool zoomIn) {
    Mat resizeImage;
    if (zoomIn) {
        // Bilinear Interpolation
        int height = colorImage.rows;
        int width = colorImage.cols;

        int new_height = height * scale;
        int new_width = width * scale;

        resizeImage = Mat(new_height, new_width, CV_8UC3);

        for (int i = 0; i < new_height; i++)
            for (int j = 0; j < new_width; j++)
                for (int k = 0; k < 3; k++) {
                    double x = (double)j / scale;
                    double y = (double)i / scale;
                    int x1 = (int)x;
                    int x2 = min(x1 + 1, width - 1);
                    int y1 = (int)y;
                    int y2 = min(y1 + 1, height - 1);
                    double fx1 = (x2 - x) / ((double)x2 - x1) * colorImage.at<Vec3b>(y1, x1)[k] + (x - x1) / ((double)x2 - x1) * colorImage.at<Vec3b>(y1, x2)[k];
                    double fx2 = (x2 - x) / ((double)x2 - x1) * colorImage.at<Vec3b>(y2, x1)[k] + (x - x1) / ((double)x2 - x1) * colorImage.at<Vec3b>(y2, x2)[k];
                    double fy = (y2 - y) / ((double)y2 - y1) * fx1 + (y - y1) / ((double)y2 - y1) * fx2;
                    resizeImage.at<Vec3b>(i, j)[k] = (uchar)fy;
                }
    }
    else {
```

根據題目，縮小將周圍像素相加後取平均值作為結果。




```
else {
    resizeImage = Mat(colorImage.rows / scale, colorImage.cols / scale, CV_8UC3);
    for (int w = 0; w < resizeImage.rows; w++)
        for (int h = 0; h < resizeImage.cols; h++) {
            uchar color[3] = { 0, 0, 0 };
            for (int k = 0; k < 3; k++) {
                int sum = 0;
                for (int i = 0; i < scale; i++)
                    for (int j = 0; j < scale; j++)
                        sum += colorImage.at<Vec3b>(h * scale + i, w * scale + j)[k];
                color[k] = sum / (scale * scale);
            }
            resizeImage.at<Vec3b>(h, w) = Vec3b(color[0], color[1], color[2]);
        }
}
return resizeImage;
```

Q2-2 成果 zoom in

zoom out