Computer Vision - Homework 9

開發環境

- OS: Windows 10 Pro
- Program Language: C# (with .Net Core 3.1)
- IDE: Visual Studio 2019
- Project: Console Application

程式說明

程式碼主要寫在 Program.cs，各題目程式皆已實作個別方法，由 Main entry 進行呼叫，答案結果儲存於 answers 資料夾。

各題目相關演算法說明如下：

(A). Robert's Operator:

- Threshold: 12
- 遞迴取得所有 x, y 值, 根據每個 x, y 進行處理
- 取得 (x0, y0), (x1, y1) 及 (x1, y0), (x0, y1)
- 根據公式計算 r1, r2
- 藉由公式取得 magnitude 後與 threshold 比較給定新的 pixel 值

```
// 取得 (x0, y0), (x1, y1) 及 (x1, y0), (x0, y1)
var x0 = x;
var y0 = y;
var x1 = x + 1 < srcImg.Width - 1 ? x + 1 : srcImg.Width - 1;
var y1 = y + 1 < srcImg.Height - 1 ? y + 1 : srcImg.Height - 1;

// 計算 r1 及 r2
var r1 = -srcImg.GetPixel(x0, y0).R + srcImg.GetPixel(x1, y1).R;
var r2 = -srcImg.GetPixel(x1, y0).R + srcImg.GetPixel(x0, y1).R;

// 藉由公式取得 gradient magnitude，並藉由 magnitude 與 threshold 比較 pixel 的結果
var magnitude = Convert.ToInt32(Math.Sqrt(Math.Pow(r1, 2) + Math.Pow(r2, 2)));
result.SetPixel(x, y, magnitude >= threshold ? Color.Black : Color.White);
```

(B). Prewitt's Edge Detector

- Threshold: 24
- 遞迴取得所有 x, y 值, 根據每個 x, y 進行處理
- 取得橫向 (x0, y2), (x1, y2), (x2, y2) 及 (x0, y0), (x1, y0), (x2, y0) 及縱向 (x2, y0), (x2, y1), (x2, y2) 及 (x0, y0), (x0, y1), (x0, y2) 的值
- 根據公式計算 p1, p2
- 藉由公式取得 magnitude 後與 threshold 比較給定新的 pixel 值

```
// 取得橫向 (x0, y2), (x1, y2), (x2, y2) 及 (x0, y0), (x1, y0), (x2, y0)
// 及縱向 (x2, y0), (x2, y1), (x2, y2) 及 (x0, y0), (x0, y1), (x0, y2)
var x0 = Math.Max(x - 1, 0);
var y0 = Math.Max(y - 1, 0);
var x1 = x;
var y1 = y;
var x2 = Math.Min(x + 1, srcImg.Width - 1);
var y2 = Math.Min(y + 1, srcImg.Height - 1);

// 計算 p1, p2
var p1 = (srcImg.GetPixel(x0, y2).R + srcImg.GetPixel(x1, y2).R +
srcImg.GetPixel(x2, y2).R) -
         (srcImg.GetPixel(x0, y0).R + srcImg.GetPixel(x1, y0).R +
srcImg.GetPixel(x2, y0).R);
var p2 = (srcImg.GetPixel(x2, y0).R + srcImg.GetPixel(x2, y1).R +
srcImg.GetPixel(x2, y2).R) -
         (srcImg.GetPixel(x0, y0).R + srcImg.GetPixel(x0, y1).R +
srcImg.GetPixel(x0, y2).R);

// 藉由公式取得 gradient magnitude，並藉由 magnitude 與 threshold 比較 pixel 的結果
var magnitude = Convert.ToInt32(Math.Sqrt(Math.Pow(p1, 2) + Math.Pow(p2, 2)));
result.SetPixel(x, y, magnitude >= threshold ? Color.Black : Color.White);
```

(C). Sobel's Edge Detector

- Threshold: 38
- 遞迴取得所有 x, y 值, 根據每個 x, y 進行處理
- 取得橫向 (x0, y2), (x1, y2), (x2, y2) 及 (x0, y0), (x1, y0), (x2, y0) 及縱向 (x2, y0), (x2, y1), (x2, y2) 及 (x0, y0), (x0, y1), (x0, y2) 的值
- 根據公式計算 s1, s2
- 藉由公式取得 magnitude 後與 threshold 比較給定新的 pixel 值

```
// 取得橫向 (x0, y2), (x1, y2), (x2, y2) 及 (x0, y0), (x1, y0), (x2, y0)
// 及縱向 (x2, y0), (x2, y1), (x2, y2) 及 (x0, y0), (x0, y1), (x0, y2)
var x0 = Math.Max(x - 1, 0);
var y0 = Math.Max(y - 1, 0);
var x1 = x;
var y1 = y;
var x2 = Math.Min(x + 1, srcImg.Width - 1);
var y2 = Math.Min(y + 1, srcImg.Height - 1);

// 計算 s1, s2
var s1 = (srcImg.GetPixel(x0, y2).R + 2 * srcImg.GetPixel(x1, y2).R +
srcImg.GetPixel(x2, y2).R) -
         (srcImg.GetPixel(x0, y0).R + 2 * srcImg.GetPixel(x1, y0).R +
srcImg.GetPixel(x2, y0).R);
var s2 = (srcImg.GetPixel(x2, y0).R + 2 * srcImg.GetPixel(x2, y1).R +
srcImg.GetPixel(x2, y2).R) -
         (srcImg.GetPixel(x0, y0).R + 2 * srcImg.GetPixel(x0, y1).R +
srcImg.GetPixel(x0, y2).R);

// 藉由公式取得 gradient magnitude，並藉由 magnitude 與 threshold 比較 pixel 的結果
var magnitude = Convert.ToInt32(Math.Sqrt(Math.Pow(s1, 2) + Math.Pow(s2, 2)));
result.SetPixel(x, y, magnitude >= threshold ? Color.Black : Color.White);
```

(D). Frei and Chen's Gradient Operator

- Threshold: 30
- 遞迴取得所有 x, y 值, 根據每個 x, y 進行處理
- 取得橫向 (x0, y2), (x1, y2), (x2, y2) 及 (x0, y0), (x1, y0), (x2, y0) 及縱向 (x2, y0), (x2, y1), (x2, y2) 及 (x0, y0), (x0, y1), (x0, y2) 的值
- 根據公式計算 s1, s2

- 藉由公式取得 magnitude 後與 threshold 比較給定新的 pixel 值

```
// 取得橫向 (x0, y2), (x1, y2), (x2, y2) 及 (x0, y0), (x1, y0), (x2, y0)
// 及縱向 (x2, y0), (x2, y1), (x2, y2) 及 (x0, y0), (x0, y1), (x0, y2)
var x0 = Math.Max(x - 1, 0);
var y0 = Math.Max(y - 1, 0);
var x1 = x;
var y1 = y;
var x2 = Math.Min(x + 1, srcImg.Width - 1);
var y2 = Math.Min(y + 1, srcImg.Height - 1);

// 計算 f1, f2
var f1 = (srcImg.GetPixel(x0, y2).R + Math.Sqrt(2) * srcImg.GetPixel(x1, y2).R +
          srcImg.GetPixel(x2, y2).R) -
         (srcImg.GetPixel(x0, y0).R + Math.Sqrt(2) * srcImg.GetPixel(x1, y0).R +
          srcImg.GetPixel(x2, y0).R);
var f2 = (srcImg.GetPixel(x2, y0).R + Math.Sqrt(2) * srcImg.GetPixel(x2, y1).R +
          srcImg.GetPixel(x2, y2).R) -
         (srcImg.GetPixel(x0, y0).R + Math.Sqrt(2) * srcImg.GetPixel(x0, y1).R +
          srcImg.GetPixel(x0, y2).R);

// 藉由公式取得 gradient magnitude，並藉由 magnitude 與 threshold 比較 pixel 的結果
var magnitude = Convert.ToInt32(Math.Sqrt(Math.Pow(f1, 2) + Math.Pow(f2, 2)));
result.SetPixel(x, y, magnitude >= threshold ? Color.Black : Color.White);
```

(E). Kirsch's Compass Operator

- Threshold: 135
- 遞迴取得所有 x, y 值, 根據每個 x, y 進行處理
- 取得公式矩陣的 pixel value
- 計算 k0 ~ k7, 取最大值 kn 作為 magnitude
- 比較 magnitude 與 threshold, 更新 pixel 值

```
// 取得相對應的 x, y 值
var x0 = Math.Max(x - 1, 0);
var y0 = Math.Max(y - 1, 0);
var x1 = x;
var y1 = y;
var x2 = Math.Min(x + 1, srcImg.Width - 1);
var y2 = Math.Min(y + 1, srcImg.Height - 1);

var kList = new List<int>();

// 計算 k0 ~ k7 的值
var k0 = -3 * srcImg.GetPixel(x0, y0).R - 3 * srcImg.GetPixel(x1, y0).R +
         5 * srcImg.GetPixel(x2, y0).R
         - 3 * srcImg.GetPixel(x0, y1).R + 5 * srcImg.GetPixel(x2, y1).R
         - 3 * srcImg.GetPixel(x0, y2).R - 3 * srcImg.GetPixel(x1, y2).R +
         5 * srcImg.GetPixel(x2, y2).R;
var k1 = -3 * srcImg.GetPixel(x0, y0).R + 5 * srcImg.GetPixel(x1, y0).R +
         5 * srcImg.GetPixel(x2, y0).R
         - 3 * srcImg.GetPixel(x0, y1).R + 5 * srcImg.GetPixel(x2, y1).R
         - 3 * srcImg.GetPixel(x0, y2).R - 3 * srcImg.GetPixel(x1, y2).R -
         3 * srcImg.GetPixel(x2, y2).R;
var k2 = 5 * srcImg.GetPixel(x0, y0).R + 5 * srcImg.GetPixel(x1, y0).R +
         5 * srcImg.GetPixel(x2, y0).R
```

```
          - 3 * srcImg.GetPixel(x0, y1).R - 3 * srcImg.GetPixel(x2, y1).R
          - 3 * srcImg.GetPixel(x0, y2).R - 3 * srcImg.GetPixel(x1, y2).R -
          3 * srcImg.GetPixel(x2, y2).R;
var k3 = 5 * srcImg.GetPixel(x0, y0).R + 5 * srcImg.GetPixel(x1, y0).R -
    3 * srcImg.GetPixel(x2, y0).R
    + 5 * srcImg.GetPixel(x0, y1).R - 3 * srcImg.GetPixel(x2, y1).R
                                    - 3 * srcImg.GetPixel(x0, y2).R -
                                    3 * srcImg.GetPixel(x1, y2).R -
                                    3 * srcImg.GetPixel(x2, y2).R;
var k4 = 5 * srcImg.GetPixel(x0, y0).R - 3 * srcImg.GetPixel(x1, y0).R -
         3 * srcImg.GetPixel(x2, y0).R
         + 5 * srcImg.GetPixel(x0, y1).R - 3 * srcImg.GetPixel(x2, y1).R
         + 5 * srcImg.GetPixel(x0, y2).R - 3 * srcImg.GetPixel(x1, y2).R -
         3 * srcImg.GetPixel(x2, y2).R;
var k5 = -3 * srcImg.GetPixel(x0, y0).R - 3 * srcImg.GetPixel(x1, y0).R -
         3 * srcImg.GetPixel(x2, y0).R
         + 5 * srcImg.GetPixel(x0, y1).R - 3 * srcImg.GetPixel(x2, y1).R
         + 5 * srcImg.GetPixel(x0, y2).R + 5 * srcImg.GetPixel(x1, y2).R -
         3 * srcImg.GetPixel(x2, y2).R;
var k6 = -3 * srcImg.GetPixel(x0, y0).R - 3 * srcImg.GetPixel(x1, y0).R -
         3 * srcImg.GetPixel(x2, y0).R
         - 3 * srcImg.GetPixel(x0, y1).R - 3 * srcImg.GetPixel(x2, y1).R
         + 5 * srcImg.GetPixel(x0, y2).R + 5 * srcImg.GetPixel(x1, y2).R +
         5 * srcImg.GetPixel(x2, y2).R;
var k7 = -3 * srcImg.GetPixel(x0, y0).R - 3 * srcImg.GetPixel(x1, y0).R -
         3 * srcImg.GetPixel(x2, y0).R
         - 3 * srcImg.GetPixel(x0, y1).R + 5 * srcImg.GetPixel(x2, y1).R
         - 3 * srcImg.GetPixel(x0, y2).R + 5 * srcImg.GetPixel(x1, y2).R +
         5 * srcImg.GetPixel(x2, y2).R;

kList.Add(k0);
kList.Add(k1);
kList.Add(k2);
kList.Add(k3);
kList.Add(k4);
kList.Add(k5);
kList.Add(k6);
kList.Add(k7);

// 取得 k0 ~ k7 的最大值
var magnitude = kList.Max();
result.SetPixel(x, y, magnitude >= threshold ? Color.Black : Color.White);
```

(F). Robinson's Compass Operator

- Threshold: 43
- 遞迴取得所有 x, y 值, 根據每個 x, y 進行處理
- 取得公式矩陣的 pixel value
- 計算 r0 ~ r7, 取最大值 rn 作為 magnitude
- 比較 magnitude 與 threshold, 更新 pixel 值

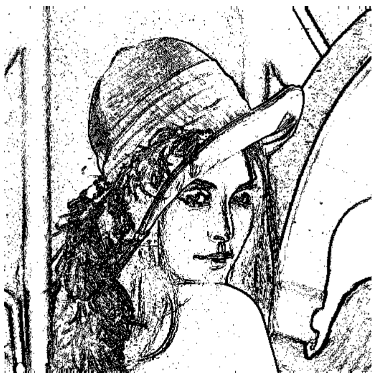
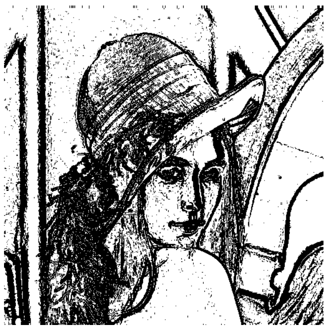
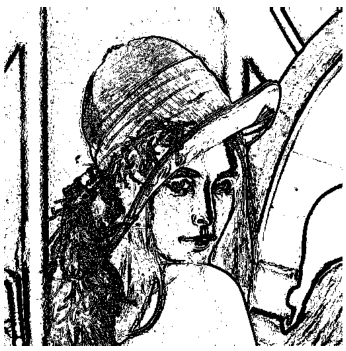
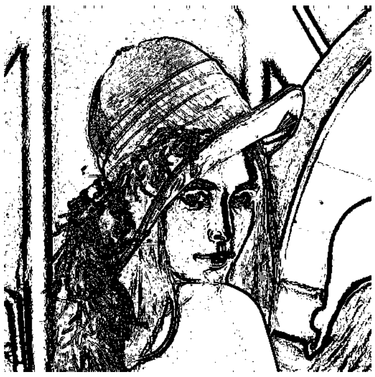
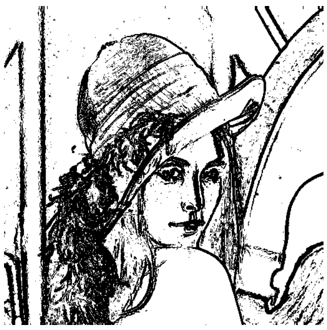
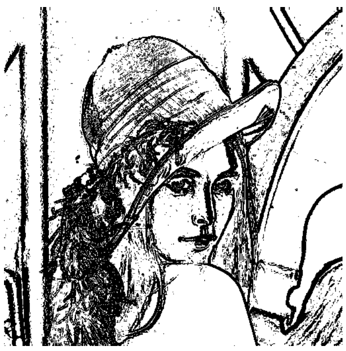
```
// 取得相對應的 x, y 值
var x0 = Math.Max(x - 1, 0);
var y0 = Math.Max(y - 1, 0);
var x1 = x;
var y1 = y;
var x2 = Math.Min(x + 1, srcImg.Width - 1);
var y2 = Math.Min(y + 1, srcImg.Height - 1);

var rList = new List<int>();

var r0 = -1 * srcImg.GetPixel(x0, y0).R - 2 * srcImg.GetPixel(x0, y1).R -
         1 * srcImg.GetPixel(x0, y2).R
         + 1 * srcImg.GetPixel(x2, y0).R + 2 * srcImg.GetPixel(x2, y1).R +
         1 * srcImg.GetPixel(x2, y2).R;
var r1 = -1 * srcImg.GetPixel(x0, y1).R - 2 * srcImg.GetPixel(x0, y2).R -
         1 * srcImg.GetPixel(x1, y2).R
         + 1 * srcImg.GetPixel(x1, y0).R + 2 * srcImg.GetPixel(x2, y0).R +
         1 * srcImg.GetPixel(x2, y1).R;
var r2 = -1 * srcImg.GetPixel(x0, y2).R - 2 * srcImg.GetPixel(x1, y2).R -
         1 * srcImg.GetPixel(x2, y2).R
         + 1 * srcImg.GetPixel(x0, y0).R + 2 * srcImg.GetPixel(x1, y0).R +
         1 * srcImg.GetPixel(x2, y0).R;
var r3 = -1 * srcImg.GetPixel(x1, y2).R - 2 * srcImg.GetPixel(x2, y2).R -
         1 * srcImg.GetPixel(x2, y1).R
         + 1 * srcImg.GetPixel(x0, y1).R + 2 * srcImg.GetPixel(x0, y0).R +
         1 * srcImg.GetPixel(x1, y0).R;
var r4 = -1 * srcImg.GetPixel(x2, y0).R - 2 * srcImg.GetPixel(x2, y1).R -
         1 * srcImg.GetPixel(x2, y2).R
         + 1 * srcImg.GetPixel(x0, y0).R + 2 * srcImg.GetPixel(x0, y1).R +
         1 * srcImg.GetPixel(x0, y2).R;
var r5 = -1 * srcImg.GetPixel(x1, y0).R - 2 * srcImg.GetPixel(x2, y0).R -
         1 * srcImg.GetPixel(x2, y1).R
         + 1 * srcImg.GetPixel(x0, y1).R + 2 * srcImg.GetPixel(x0, y2).R +
         1 * srcImg.GetPixel(x1, y2).R;
var r6 = -1 * srcImg.GetPixel(x0, y0).R - 2 * srcImg.GetPixel(x1, y0).R -
         1 * srcImg.GetPixel(x2, y0).R
         + 1 * srcImg.GetPixel(x0, y2).R + 2 * srcImg.GetPixel(x1, y2).R +
         1 * srcImg.GetPixel(x2, y2).R;
var r7 = -1 * srcImg.GetPixel(x0, y1).R - 2 * srcImg.GetPixel(x0, y0).R -
         1 * srcImg.GetPixel(x1, y0).R
         + 1 * srcImg.GetPixel(x1, y2).R + 2 * srcImg.GetPixel(x2, y2).R +
         1 * srcImg.GetPixel(x2, y1).R;

rList.Add(r0);
rList.Add(r1);
rList.Add(r2);
rList.Add(r3);
rList.Add(r4);
rList.Add(r5);
rList.Add(r6);
rList.Add(r7);

// 取得 r0 ~ r7 的最大值
var magnitude = rList.Max();
result.SetPixel(x, y, magnitude >= threshold ? Color.Black : Color.White);
```

(G). Nevatia-Babu 5x5 Operator

- Threshold: 12500
- 遞迴取得所有 x, y 值, 根據每個 x, y 進行處理
- 取得公式矩陣的 pixel value
- 計算 n0 ~ n5, 取最大值 nn 作為 magnitude
- 比較 magnitude 與 threshold, 更新 pixel 值

```
// 取得相關的 x, y
var x0 = Math.Max(x - 2, 0);
var y0 = Math.Max(y - 2, 0);
var x1 = Math.Max(x - 1, 0);
var y1 = Math.Max(y - 1, 0);
var x2 = x;
var y2 = y;
var x3 = Math.Min(x + 1, srcImg.Width - 1);
var y3 = Math.Min(y + 1, srcImg.Height - 1);
var x4 = Math.Min(x + 2, srcImg.Width - 1);
var y4 = Math.Min(y + 2, srcImg.Height - 1);

var neighbors = new int[]
                    {
                        srcImg.GetPixel(x0, y0).R,
                        srcImg.GetPixel(x1, y0).R,
                        srcImg.GetPixel(x2, y0).R,
                        srcImg.GetPixel(x3, y0).R,
                        srcImg.GetPixel(x4, y0).R,
                        srcImg.GetPixel(x0, y1).R,
                        srcImg.GetPixel(x1, y1).R,
                        srcImg.GetPixel(x2, y1).R,
                        srcImg.GetPixel(x3, y1).R,
                        srcImg.GetPixel(x4, y1).R,
                        srcImg.GetPixel(x0, y2).R,
                        srcImg.GetPixel(x1, y2).R,
                        srcImg.GetPixel(x2, y2).R,
                        srcImg.GetPixel(x3, y2).R,
                        srcImg.GetPixel(x4, y2).R,
                        srcImg.GetPixel(x0, y3).R,
                        srcImg.GetPixel(x1, y3).R,
                        srcImg.GetPixel(x2, y3).R,
                        srcImg.GetPixel(x3, y3).R,
                        srcImg.GetPixel(x4, y3).R,
                        srcImg.GetPixel(x0, y4).R,
                        srcImg.GetPixel(x1, y4).R,
                        srcImg.GetPixel(x2, y4).R,
                        srcImg.GetPixel(x3, y4).R,
                        srcImg.GetPixel(x4, y4).R
                    };

var nList = new List<int>();

// 根據公式計算 n0 ~ n5
var n0 = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] +
(100) * neighbors[3] +
         (100) * neighbors[4] +
         (100) * neighbors[5] + (100) * neighbors[6] + (100) * neighbors[7] +
(100) * neighbors[8] +
         (100) * neighbors[9] +
         (0) * neighbors[10] + (0) * neighbors[11] + (0) * neighbors[12] + (0) *
neighbors[13] +
         (0) * neighbors[14] +
         (-100) * neighbors[15] + (-100) * neighbors[16] + (-100) *
neighbors[17] +
         (-100) * neighbors[18] + (-100) * neighbors[19] +
         (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) *
neighbors[22] +
         (-100) * neighbors[23] + (-100) * neighbors[24];
var n1 = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] +
(100) * neighbors[3] +
         (100) * neighbors[4] +
         (100) * neighbors[5] + (100) * neighbors[6] + (100) * neighbors[7] +
(78) * neighbors[8] +
         (-32) * neighbors[9] +
         (100) * neighbors[10] + (92) * neighbors[11] + (0) * neighbors[12] +
         (-92) * neighbors[13] + (-100) * neighbors[14] +
         (32) * neighbors[15] + (-78) * neighbors[16] + (-100) * neighbors[17] +
         (-100) * neighbors[18] + (-100) * neighbors[19] +
         (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) *
neighbors[22] +
         (-100) * neighbors[23] + (-100) * neighbors[24];
var n2 = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] +
(32) * neighbors[3] +
         (-100) * neighbors[4] +
         (100) * neighbors[5] + (100) * neighbors[6] + (92) * neighbors[7] +
(-78) * neighbors[8] +
         (-100) * neighbors[9] +
         (100) * neighbors[10] + (100) * neighbors[11] + (0) * neighbors[12] +
         (-100) * neighbors[13] + (-100) * neighbors[14] +
         (100) * neighbors[15] + (78) * neighbors[16] + (-92) * neighbors[17] +
         (-100) * neighbors[18] + (-100) * neighbors[19] +
         (100) * neighbors[20] + (-32) * neighbors[21] + (-100) * neighbors[22]
+
         (-100) * neighbors[23] + (-100) * neighbors[24];
var n3 = (-100) * neighbors[0] + (-100) * neighbors[1] + (0) * neighbors[2] +
(100) * neighbors[3] +
         (100) * neighbors[4] +
         (-100) * neighbors[5] + (-100) * neighbors[6] + (0) * neighbors[7] +
(100) * neighbors[8] +
         (100) * neighbors[9] +
         (-100) * neighbors[10] + (-100) * neighbors[11] + (0) * neighbors[12] +
         (100) * neighbors[13] + (100) * neighbors[14] +
         (-100) * neighbors[15] + (-100) * neighbors[16] + (0) * neighbors[17] +
         (100) * neighbors[18] + (100) * neighbors[19] +
         (-100) * neighbors[20] + (-100) * neighbors[21] + (0) * neighbors[22] +
         (100) * neighbors[23] + (100) * neighbors[24];
var n4 = (-100) * neighbors[0] + (32) * neighbors[1] + (100) * neighbors[2] +
(100) * neighbors[3] +
         (100) * neighbors[4] +
         (-100) * neighbors[5] + (-78) * neighbors[6] + (92) * neighbors[7] +
(100) * neighbors[8] +
         (100) * neighbors[9] +
         (-100) * neighbors[10] + (-100) * neighbors[11] + (0) * neighbors[12] +
(100) * neighbors[13] +
         (100) * neighbors[14] +
         (-100) * neighbors[15] + (-100) * neighbors[16] + (-92) * neighbors[17]
+ (78) * neighbors[18] +
         (100) * neighbors[19] +
         (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) *
neighbors[22] + (-32) * neighbors[23] +
         (100) * neighbors[24];
var n5 = (100) * neighbors[0] + (100) * neighbors[1] + (100) * neighbors[2] +
(100) * neighbors[3] +
         (100) * neighbors[4] +
         (-32) * neighbors[5] + (78) * neighbors[6] + (100) * neighbors[7] +
(100) * neighbors[8] +
         (100) * neighbors[9] +
         (-100) * neighbors[10] + (-92) * neighbors[11] + (0) * neighbors[12] +
(92) * neighbors[13] +
         (100) * neighbors[14] +
         (-100) * neighbors[15] + (-100) * neighbors[16] + (-100) *
neighbors[17] + (-78) * neighbors[18] +
         (32) * neighbors[19] +
         (-100) * neighbors[20] + (-100) * neighbors[21] + (-100) *
neighbors[22] + (-100) * neighbors[23] +
         (-100) * neighbors[24];

nList.Add(n0);
nList.Add(n1);
nList.Add(n2);
nList.Add(n3);
nList.Add(n4);
nList.Add(n5);

// 取得 n0 ~ n5 的最大值並與 threshold 比較
var magnitude = nList.Max();
result.SetPixel(x, y, magnitude >= threshold ? Color.Black : Color.White);
```

結果圖片

A. Robert's Operator, Threshold: 12	B. Prewitt's Edge Detector, Threshold: 24	C. Sobel's Edge Detector, Threshold: 38
		
D. Frei and Chen's Gradient Operator, Threshold: 30	E. Kirsch's Compass Operator, Threshold: 135	F. Robinson's Compass Operator, Threshold: 43
		
G.. Nevatia-Babu 5x5 Operator, Threshold: 12500