Computer Vision HW1 Report

作者：105062509 / 羅右鈞

註：此 report 原先在 dropbox paper 上編輯，直接點此觀看就能看到更美的排版。

檔案架構

- program 資料夾包含各個 part 的程式碼以及執行結果。
- 每個 part 的主程式以 “part_<第幾大題第幾小題>.m” 來命名，例如，part 1-A 的主程式的檔案則命名為 “part_1a.m”。
- 每個 part 的主程式皆附其執行結果的暫存檔案 (.mat 檔案)，命名方式跟主程式相同 (只是後面多了 “_result”)，以供主程式做快速 demo 為用 (每個 part 的主程式內有 “QUICK_DEMO” 控制參數，其作用在下面會補充)。
- part 1 以及 part 2 的程式執行結果在 part1_demo, part2_demo 內。
- program 資料夾內的 “CornerDetection.m” 以及 “LBP.m” 兩個 class 檔分別包含了 part1, part2 中常用的函式 (下面會再補充這兩個檔案的內容)。
- 每個檔案皆有詳細的註解。

如何觀看程式執行結果

- 例如想看 part 1-A 的程式執行結果，打開 “part_1a.m” 直接跑就行了，其他 part 也是一樣。
- 每個 part 的程式中有 “QUICK_DEMO” 控制參數，若設為 1 (default)，則會從 .mat 檔讀取之前就計算好的執行結果並顯示出來（為了在 demo 的時候省時）；若設為 0，則會完整跑完計算過程，需要稍微等候才會顯示執行結果。

常用函式說明

此部分解釋 “CornerDetection.m” 以及 “LBP.m” 兩個 class 中的常用函式，也就是分別在 part1, part2 中常用到的核心函式：

CornerDetection (part 1 所用到的常用函式)

---

function result = guassianFilter(img, kernel_size, sigma)

```
% Apply guassian smoothing to image
function result = gaussianFilter(img, kernel_size, sigma)
    gaussian = fspecial('gaussian', kernel_size, sigma);
    result = imfilter(img, gaussian);
end
```

功能概述：對一張 image 作 gaussian smoothing

參數說明：

- img: 圖片 (全彩或灰階)
- kernel_size: gaussian kernel 的 size
- sigma: 決定 gaussian distribution 的 standard deviation

回傳變數：

- result: gaussian smoothed image

做法：

- 用 fspecial 產生 gaussian filter
- 使用 imfilter with gaussian filter 對 image 作 filtering 達到 gaussian smoothing 的效果

---

function [grad_x, grad_y] = imgradientxy(img)

```
% Compute gradient of image
function [grad_x, grad_y] = imgradientxy(img)
    sy = fspecial('sobel');
    sx = sy';
    grad_x = conv2(img, sx, 'same');
    grad_y = conv2(img, sy, 'same');
end
```

功能概述：算出一張 image 的 gradient

參數說明：

- img: 圖片 (灰階)

回傳變數：

- grad_x: 圖片對 x 的偏微分，尺寸等同原圖
- grad_y: 圖片對 y 的偏微分，尺寸等同原圖

做法：

- 用 fspecial 產生 sobel y operator，將之轉置可得到 sobel x operator
- 用 conv2 分別對圖片作 sobel x, sobel y 的 convolution，可得到 grad_x, grad_y

function [grad_mag, grad_dir] = imgradient(img)

```
% Compute magnitude and direction of gradient of image
% @param img: must be converted to grayscale and gaussian smoothed and normalized to [0-1]
function [grad_mag, grad_dir] = imgradient(img)
    [grad_x, grad_y] = CornerDetection.imgradientxy(img);
    grad_mag = sqrt((grad_x.^2) + (grad_y.^2));
    grad_dir = ((atan2(grad_y,grad_x) * (180/pi)) + 180) / 360;
end
```

功能概述：算出一張 image 的 gradient magnitude 以及 gradient direction

參數說明：

- img: 圖片 (normalized 的灰階圖)

回傳變數：

- grad_mag: image 的 gradient magnitude
- grad_dir: image 的 gradient direction，值域 [0, 1]

做法：

- 用此 class 的 imgradientxy 先算出圖片對 x 的偏微分 (grad_x) 以及對 y 的偏微分 (grad_y)
- grad_mag 的算法參照下列公式：

$$g(x,y) \cong (\Delta x^2 + \Delta y^2)^{\frac{1}{2}}$$

- grad_dir 的算法參照下列公式：

$$\theta(x,y) \cong atan\left(\frac{\Delta y}{\Delta x}\right)$$

用 atan 算出來後因為要用 colormap 視覺化的關係，將徑度量乘上轉 180/pi 轉成度度量，又因為值域在 [-180, 180]，因此再加上 180 使值域轉成 [0, 360]，再除上 360 ，使值域轉成 [0, 1]。

---

function Ieig = detectCorner(img, window_size)

```
% corner detection
% @param img: must be converted to grayscale and gaussian smoothed and normalized to [0-1]
% @return Ieig: smallest eigenvalues of every pixel's tensor structure H
function Ieig = detectCorner(img, window_size)
    % apply sobel on magitude to compute Ix, Iy
    [Ix, Iy] = CornerDetection.imgradientxy(img);
    % compute Ix^2, Ix*Iy, Iy^2
    Ixx = Ix.^2;
    Ixy = Ix.*Iy;
    Iyy = Iy.^2;
    % compute components of structure tensor H = [H11, H12; H21, H22]
    window = ones(window_size);
    H11 = conv2(Ixx, window, 'same');
    H12 = conv2(Ixy, window, 'same');
    H22 = conv2(Iyy, window, 'same');
    % for every pixel, compute pixel's eigenvalue of H
    [num_row, num_col] = size(img);
    Ieig = zeros(num_row, num_col);
    for x=1:num_row
        for y=1:num_col
            % form pixel's tensor structure H by its components
            H = [H11(x,y), H12(x,y); H12(x,y), H22(x,y)];
            % compute smallest eignvalue of H and save to Ieig(x,y)
            Ieig(x,y) = min(eig(H));
        end
    end
end
```

功能概述：偵測一張 image 的 corner

參數說明：

- img: 圖片 (normalized 的灰階圖)
- window_size: structure tensor window size

回傳變數：

- Ieig: 與原圖同 size，對應原圖每個 pixel 的 smallest eigenvalue，大於某個 threshold 就是 corner

原理解釋：

計算原圖每個 pixel 所對應的 eigenvalue 參照下列公式 (Reference)：

$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

將 window sum 移到 [Ix^2, IxIy ; IyIx, Iy^2] 內，並將之看成 M (程式碼中為 H)，並寫成下式：

$$E(u,v) \cong [u,v] \quad M \quad \begin{bmatrix} u \\ v \end{bmatrix}$$

M (程式碼中為 H) 即為：

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- w(x, y) 是 windowing function，用來計算 weighted sum (在程式中 window 內皆為 1)
- [Ix^2, IxIy ; IyIx, Iy^2] 此矩陣的每個 component 只是 products of Ix, Iy

E(u, v) 表示 “change of intensity of the shift [u, v]” (上式為經過推導後)，而如果我們要找出 corner 的所在，就需要找出 E(u, v) 大的地方，也就是可以計算 M (程式碼中為 H) 的 eigenvalues (λ1, λ2)：

- 若 λ1 >> λ2 或 λ2 >> λ1 則判定為 edge
- 若 λ1 與 λ2 都很大，則判定為 corner，（程式中會選一個最小的 eigenvalue，若大於某 threshold 值則判定為 corner）
- 若 λ1 與 λ2 都很小，則為 flat region，也就是 E(u,v) 趨近為 0 或等於 0

做法：

- 用此 class 的 imgradientxy 先算出圖片對 x 的偏微分 (Ix) 以及對 y 的偏微分 (Iy)
- 再用 Ix, Iy 算出 [Ixx, Ixy ; Ixy, Iyy]
- 用 conv2 分別對 Ixx, Ixy, Iyy 作 windowing sum，分別算出 H 的 components, H11, H12, H22，也就是上面投影片中的 M structure tensor
- 跑 for loop 去算原圖每個 pixel 所對應的 H 的 eigenvalues，再用 min 取 smallest eigenvalue
- 算完每個 pixel 所對應的 smallest eigenvalue，存到與原圖同 size 的另一個矩陣，也就是回傳值 Ieig

function Ieig_filtered = nonMaximumSuppression(Ieig, threshold)

```
% perform non-maximum suppression for results from detectCorner()
function Ieig_filtered = nonMaximumSuppression(Ieig, threshold)
    [num_row, num_col] = size(Ieig);
    Ieig_filtered = zeros(num_row, num_col);
    % preform non-maximum suppresion
    for x = 2:num_row-1
        for y = 2:num_col-1
            % label current position as corner
            % if eignvalue is larger then threshold and neighbors'eigenvals
            if Ieig(x,y) > threshold...
                && Ieig(x,y) > Ieig(x-1,y-1)...
                && Ieig(x,y) > Ieig(x-1,y)...
                && Ieig(x,y) > Ieig(x-1,y+1)...
                && Ieig(x,y) > Ieig(x,y-1)...
                && Ieig(x,y) > Ieig(x,y+1)...
                && Ieig(x,y) > Ieig(x+1,y-1)...
                && Ieig(x,y) > Ieig(x+1,y)...
                && Ieig(x,y) > Ieig(x+1,y+1)
                Ieig_filtered(x,y) = 1;
            end
        end
    end
end
```

功能概述：經過 detectCorner 算出與原圖每個 pixel 對應的 smallest eigenvalue 矩陣後，用 non-maximum suppression 保留最突出的 corners。

參數說明：

- Ieig: detectCorner 所算出來的與原圖每個 pixel 對應的 smallest eigenvalue 矩陣
- threshold: 判斷某 pixel 是否為 corner 的 threshold 值

回傳變數：

- Ieig_filtered: 將 Ieig apply non-maximum suppression 的結果，與原圖同 size，值只有 0 跟 1，被標註成 1 的即為 corner。

做法：

- 用 for loop 對 smallest eigenvalue 矩陣中每個 eigenvalue，去檢查它周邊八個鄰居，若它比周邊八個鄰居的 eigenvalue 都還要大，而且也大過 threshold 值，則標示為 corner (標在對應的矩陣 Ieig_filtered 中)。

LBP (part 2 所用到的常用函式)

---

function lbp = computeLBP(image)

```
% compute LBP(dim=256) for image
function lbp = computeLBP(image)
    [num_row, num_col] = size(image);
    % pad image with -Inf
    image_padded = padarray(image, [1,1], -inf);
    % compute lbp decimal values and save it to 2-D array (dec_val)
    lbp = zeros(num_row, num_col);
    for x = 2:size(image_padded, 1)-1
        for y = 2:size(image_padded, 2)-1
            bin = [0, 0, 0, 0, 0, 0, 0, 0];
            center = image_padded(x,y);
            % perform lbp binary comparison, if neighbor pixel >= center pixel,
            % then neighbor set to 1, else 0;
            % start from top-left and perform in clockwise
            if image_padded(x-1,y-1) >= center
                bin(1) = 1;
            end
            % top-top
            if image_padded(x-1,y) >= center
                bin(2) = 1;
            end
            % top-right
            if image_padded(x-1,y+1) >= center
                bin(3) = 1;
            end
            % right-right
            if image_padded(x,y+1) >= center
                bin(4) = 1;
            end
            % bottom-right
            if image_padded(x+1,y+1) >= center
                bin(5) = 1;
            end
```

```
            % bottom-bottom
            if image_padded(x+1,y) >= center
                bin(6) = 1;
            end
            % bottom-left
            if image_padded(x+1,y-1) >= center
                bin(7) = 1;
            end
            % left-left
            if image_padded(x,y-1) >= center
                bin(8) = 1;
            end
            % convert binary to decimal number (LBP value)
            power_of_2 = [128, 64, 32, 16, 8, 4, 2, 1];
            lbp(x-1,y-1) = dot(bin, power_of_2);
        end
    end
end
```

(第二張圖接續第一張圖的迴圈裡)

功能概述：計算原圖的 LBP

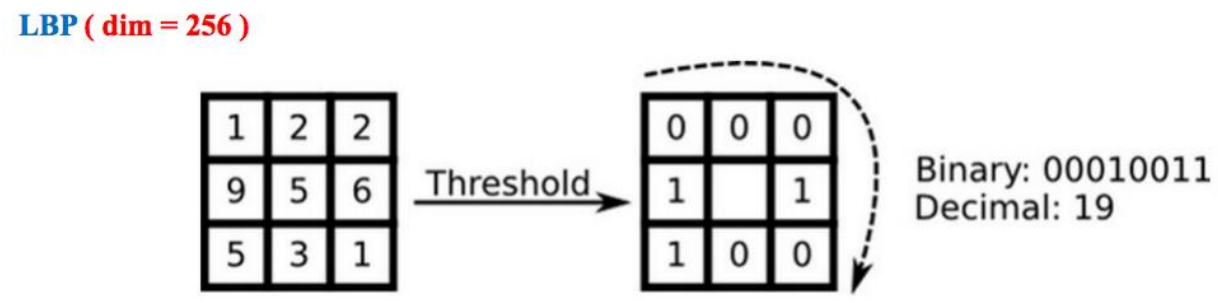
參數說明：

- image: 圖片 (灰階圖)

回傳變數：

- lbp: 與原圖同 size 的 LBP image

原理：



左邊為原圖中某個 3x3 的 window 裡，目標要算 window 裡中間 pixel 所對應的 LBP 值，算法則是跟周邊八個鄰居 pixel 值比 (程式中遵照上圖的順序，從左上 => 上上 => 右上，順時針的方式比對)，若鄰居 pixel 比中間 pixel 還小，則 binary bit 為 0，反之為 1，這樣比對下來最後會得到一串 binary code，然後再算成 decimal 就是該 window 裡中間 pixel 所對應的 LBP 值。

做法：

- 先對原圖作一層 padding (值為 -inf，值不設為 0 是因為要避免在算 LBP 時跟 center pixel 為 0 的時候，把鄰居的 binary 時把 bit 設為 1）
- 用 for loop 對原圖的每個 pixel 算其 LBP 值，算法在上面的原理有解釋，以及程式碼中也有註解。

---

function uniform_lbp = LBP2UniformLBP(lbp)

```
% convert LBP(dim=256) to uniform LBP(dim=59)
function uniform_lbp = LBP2UniformLBP(lbp)
    % 58 different kind of uniform binary to decimal cases.
    uniform_lookup_table = [0,1,2,3,4,6,7,8,12,14,15,16,...
        24,28,30,31,32,48,56,60,62,63,64,96,...
        112,120,124,126,127,128,129,131,135,...
        143,159,191,192,193,195,199,207,223,...
        224,225,227,231,239,240,241,243,247,...
        248,249,251,253,253,254,255];
    % map every value in lbp matrix to uniform lookup table index
    [num_row, num_col] = size(lbp);
    uniform_lbp = zeros(num_row, num_col);
    for x=1:num_row
        for y=1:num_col
            [is_exist, index] = ismember(lbp(x,y), uniform_lookup_table);
            if is_exist
                uniform_lbp(x,y) = index;
            else
                uniform_lbp(x,y) = 59;
            end
        end
    end
end
```

功能概述：將原圖計算好的 LBP (dim=256) 轉成 uniform LBP (dim=59)

參數說明：

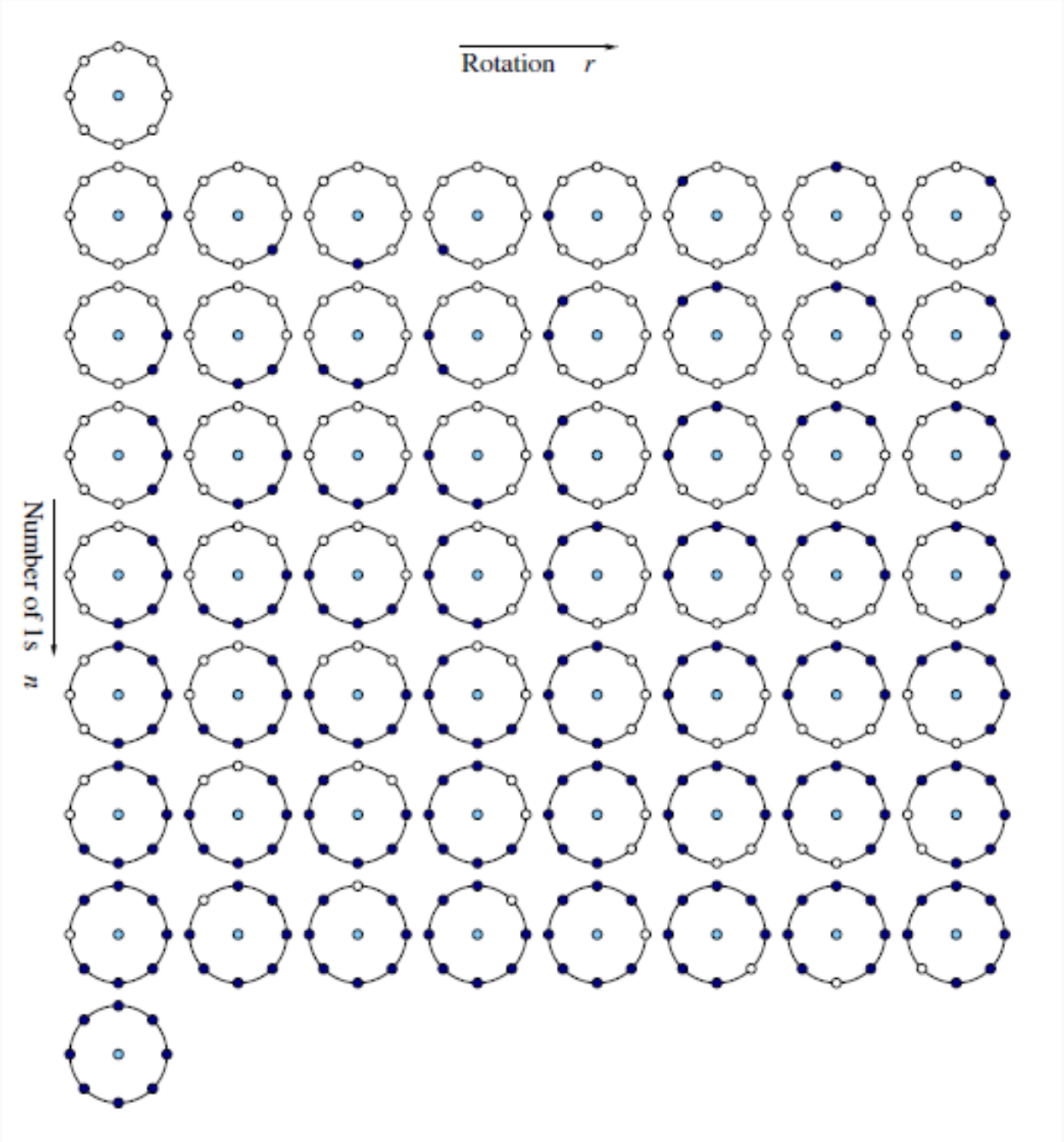
- lbp: 圖片對應的 LBP，也就是 computeLBP 所算出來的結果

回傳變數：

- uniform_lbp: 與原圖同 size 的 uniform LBP image

原理：

在 computeLBP 所介紹的原理中的 binary code，我們定義只要 binary code 中只要有 0 變成 1 或 1 變成 0 的時候，就算一個 transition，而這個 binary code 只要有超過 2 次以上的 transitions，就不算是 uniform LBP 的 binary code。



上圖為所有符合 uniform LBP binary code 的所有組合，總共 58 種，在程式中我們預先算好這 58 種 binary code 的 decimal 值，再由小到大排序過，建立出 uniform LBP

lookup table。假設原圖某個 pixel 的 LBP 值是 4，因為 4 符合 uniform LBP，因此 map 到 lookup table 中 4 這個值的 index，也就是 5，則 5 就是這個 pixel 的 uniform LBP 值；假設原圖某個 pixel 的 LBP 值是 13，因為 13 不符合 uniform LBP，因此就直接 map 到 uniform LBP 值 59 (只要是不符合 uniform LBP 規定的 LBP 值就變成 59)。

做法：

- 建立 uniform LBP lookup table，從小到大排序過
- 用 for loop 將原圖每個 pixel 所對應的 LBP 值 map 到 uniform LBP lookup table 的 index，若它的 LBP 值不在 lookup table 裡，則直接設為 59。
- 把 mapping index 的結果都存到另一個同等原圖 size 的矩陣 uniform_lbp。

---

function [histogram_vector, normalized_histogram_vector] = image2NormalizedHistogramVector(image, dim)

```
% convert image to normalized histogram vector
function [histogram_vector, normalized_histogram_vector] = image2NormalizedHistogramVector(image, dim)
    % compute every frequency of intensity (0 ~ dim-1) of image
    histogram_vector = zeros(1, dim);
    for intensity=0:dim-1
        histogram_vector(intensity+1) = sum(image(:) == intensity);
    end
    normalized_histogram_vector = histogram_vector / norm(histogram_vector);
end
```

功能概述：計算 image 的 intensity [0, dimension-1] frequency，跟在算 histogram 是一樣的事情

參數說明：

- image: 圖片 (灰階)
- dim: 圖片 intensity 的 dimension，像 LBP 就為 256，uniform LBP 則為 59。

回傳變數：

- histogram_vector: 1-D intensity frequency vector
- normalized_histogram_vector: normalized 1-D intensity frequency vector (長度為一)

做法：

- for intensity = 0 ~ dimesion-1，統計每個 intensity 在 image 中出現的次數，結果存到 histogram_vector
- normalized_histogram_vector 就是把 histogram_vector 與 norm(histogram_vector) 相除即可得到

---

function [histogram_vector, normalized_histogram_vector] = image2ConcatedNormalizedHistogramVector(image, split_num, isUniform)

```
% split image to (split_num x split_num), then compute LBP
% histogram vector for every splitted image,
% then concatenate all LBP histogram vectors to one
% normalized histogram vector
function [histogram_vector, normalized_histogram_vector] = image2ConcatedNormalizedHistogramVector(image, split_num, isUniform)
    [num_row, num_col] = size(image);
    split_row = num_row / split_num;
    split_col = num_col / split_num;
    % if we want to split 360x360 image to 2x2 image
    % split_row = 360/2 = 180 = split_col
    % num_row_splits = [1, 1] * 180 = [180, 180] = num_col_splits
    num_row_splits = ones(1, split_num) * (split_row);
    num_col_splits = ones(1, split_num) * (split_col);
    % split image to (split_num x split_num) cell
    cell = mat2cell(image, num_row_splits, num_col_splits);
    [num_row, num_col] = size(cell);
    histogram_vector = [];
    % for every image in cell
    for x=1:num_row
        for y=1:num_col
            image = cell{x,y};
            % compute LBP for every image
            imageLBP = LBP.computeLBP(image);
            % compute LBP or uniform LBP histogram vector
            if isUniform == 0
                [hv, nhv] = LBP.image2NormalizedHistogramVector(imageLBP, 256);
            else
                imageUniformLBP = LBP.LBP2UniformLBP(imageLBP);
                [hv, nhv] = LBP.image2NormalizedHistogramVector(imageUniformLBP, 59);
            end
            % append current image histogram vector to result histogram vector
            histogram_vector = [histogram_vector, hv];
        end
    end
    % normalize result histogram vector
    normalized_histogram_vector = histogram_vector / norm(histogram_vector);
end
```

功能概述：把原圖 (灰階) 先切割成 split_num x split_num 等分，然後個別算它們的 LBP 或 uniform LBP，然後在計算他們的 histogram vector，串在一起後，再算整個的 normalized histogram vector

參數說明：

- image: 圖片 (灰階)
- split_num: 原圖要切成幾等分，若 2x2，則 split_num 為 2
- isUniform: 是否是計算 uniform LBP

回傳變數：

- histogram_vector: 串起來的 high dimensional vector，假設將原圖切成 4 等分，分別算它們的 LBP histogram vectors，再串起來後得到 dimension 為 1 x (4 x 256) 的 histogram vector；若是算 uniform LBP，則串起來後得到的 dimension 為 1 x (4 x 59)
- normalized_histogram_vector: normalize 上述串起來的 histogram vector

做法：

- 用 mat2cell 將原圖進行切割
- 重複利用此 class 的 computeLBP, LBP2UniformLBP, image2NormalizedHistogramVector 來算

各題做法解釋

各題會用到上述常用函式，做法都在上面解釋過了，這部分在解釋的時候只會說明步驟跟用到哪些常用韓式，並不會再重新解釋常用函式的部分。

Part 1-A

做法：

- 讀取圖片 “J4Poro.png”。
- 分別設置 gaussian filter 的參數。
  - kernel size = 10, sigma = 5
  - kernel size = 20, sigma = 5
- 對圖片套用上述不同的兩組參數的 CornerDetection.gaussianFilter，得到兩個 gaussian smoothed 的結果，並用 imshow 顯示。

結果 (可放大來看)：

- 左：原圖，沒有做 guassian smoothing
- 中：kernel size = 10, sigma = 5 的 gaussian smoothing 後的結果
- 右：kernel size = 20, sigma = 5 的 gaussian smoothing 後的結果

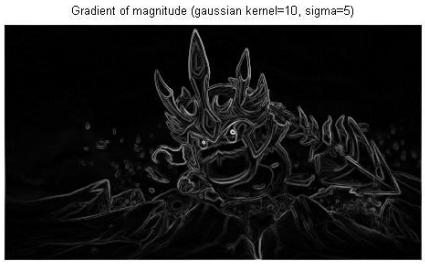
從結果中可看出 kernel size 越大，也就代表 convolution mask 越大，因此受到 gaussian filter 的程度也就越大，所以 kernel size 越大，圖片也越模糊。

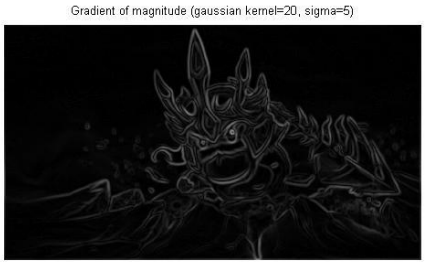
Part 1-B

做法：

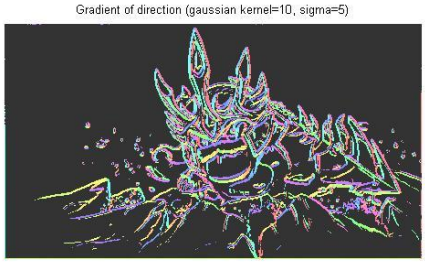
- 先建立好視覺化 gradient direction 的 colormap
- 承 part 1-A guassian smoothed 後的兩張圖片，對他們 apply im2double 得到 normalized 後的圖片，值域 [0, 1]
- 用 CornerDetection.imgradient 算出這兩張圖片的 gradient
- 把 weak gradient 給 filter 掉，若 gradient magnitude 小於某個 threshold (程式設為 0.1)，就把對應的 gradient direction 設為 0
- 用 imshow 顯示結果

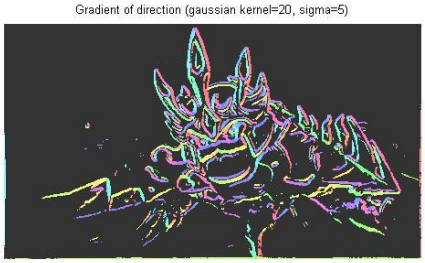
結果 (可放大來看)：





- 左：對 kernel size = 10, sigma = 5 的 gaussian smoothing 後的圖算 gradient magnitude
- 右：對 kernel size = 20, sigma = 5 的 gaussian smoothing 後的圖算 gradient magnitude





- 左：對 kernel size = 10, sigma = 5 的 gaussian smoothing 後的圖算 gradient direction
- 右：對 kernel size = 20, sigma = 5 的 gaussian smoothing 後的圖算 gradient direction

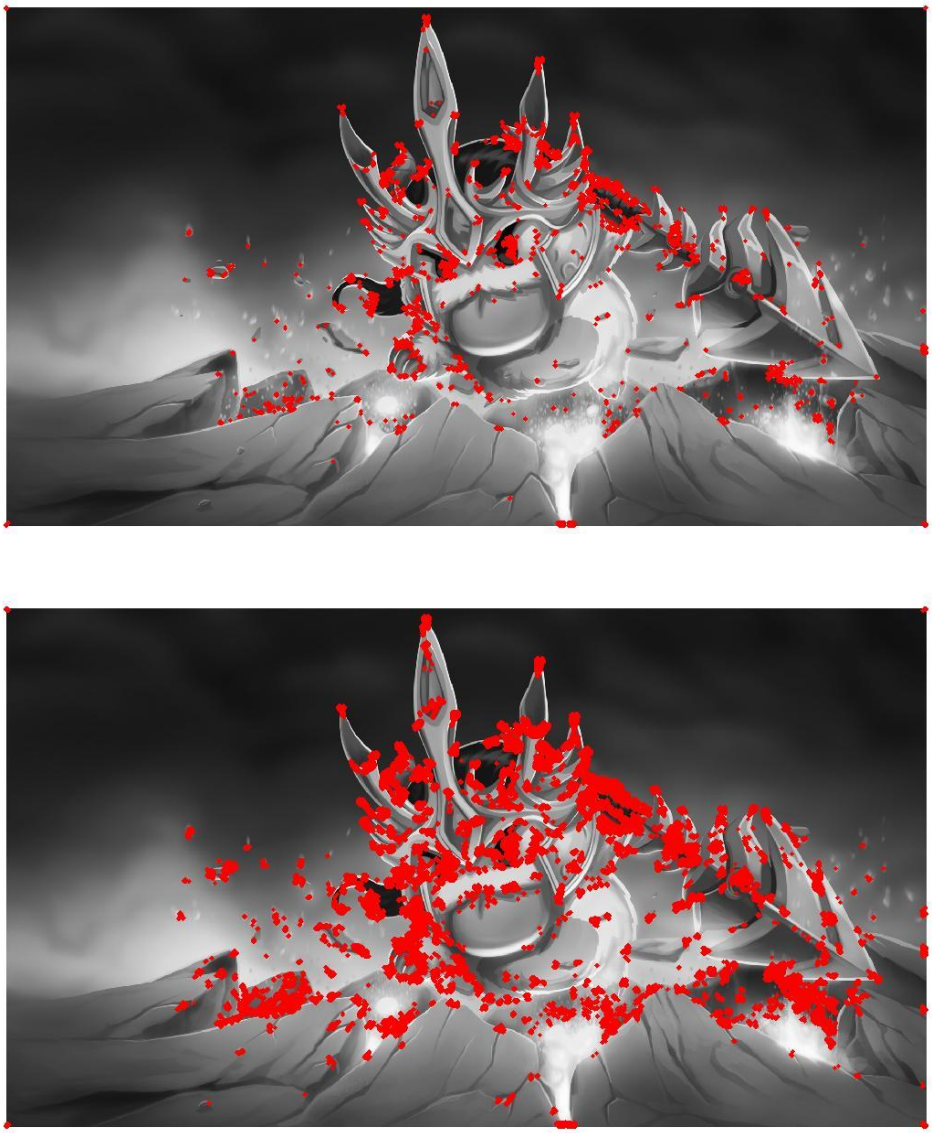
---

Part 1-C

做法：

- 承 part 1-B 算出來的兩張 gradient 圖，對他們分別用 3x3, 5x5 window 計算 corner (用 CornerDetection.detectCorner)
- 算出來後得到了四張圖個別對應的 smallest eigenvalue matrix (CornerDetection.detectCorner 的回傳值 Ieig)，然後在分別算它們各自的 threshold，程式中的算法是 0.05 * max(max(Ieig))，作為它們各自的 threshold (part 1-D 會用到)
- 這題因為要比較四張圖片的結果，因此就取同一個 threshold = min(他們四個各自的 thresholds)
- 用 find(Ieig > threshold) 的方式找出 corner 的 positions，再用 imshow + plot 把他們畫在原圖上

結果 (可放大來看)：



- 左：kernel size = 10, sigma = 5, window size = 3x3
- 右：kernel size = 10, sigma = 5, window size = 5x5

- 左：kernel size = 20, sigma = 5, window size = 3x3
- 右：kernel size = 20, sigma = 5, window size = 5x5

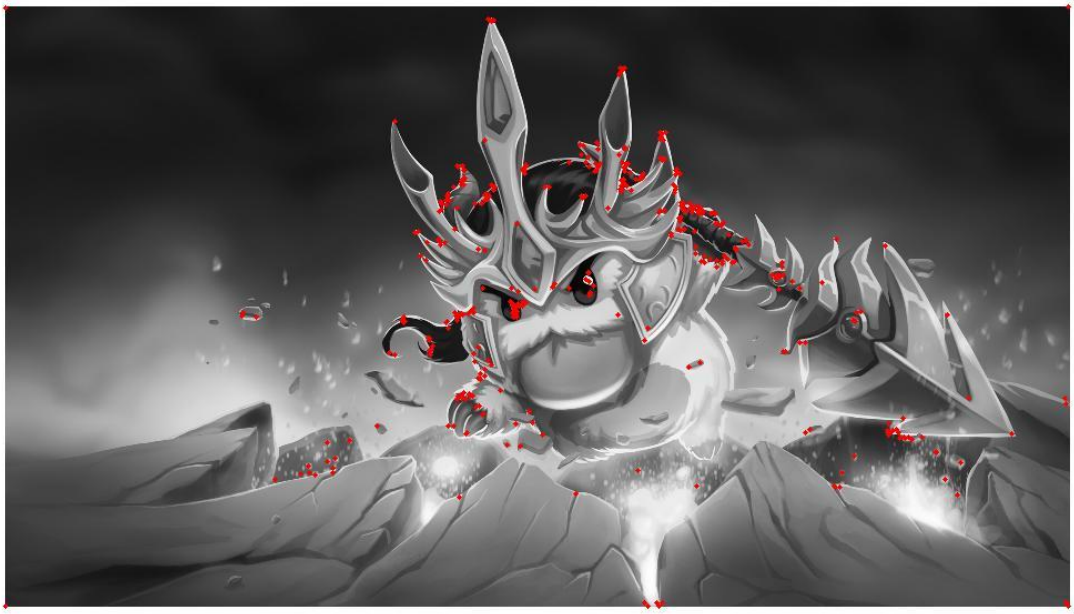
從結果可看出越模糊的照片 (guassian kernel size 越大)，所偵測到的 corners 越少，而 structure tensor window 越大，則越容易判定為 corner。

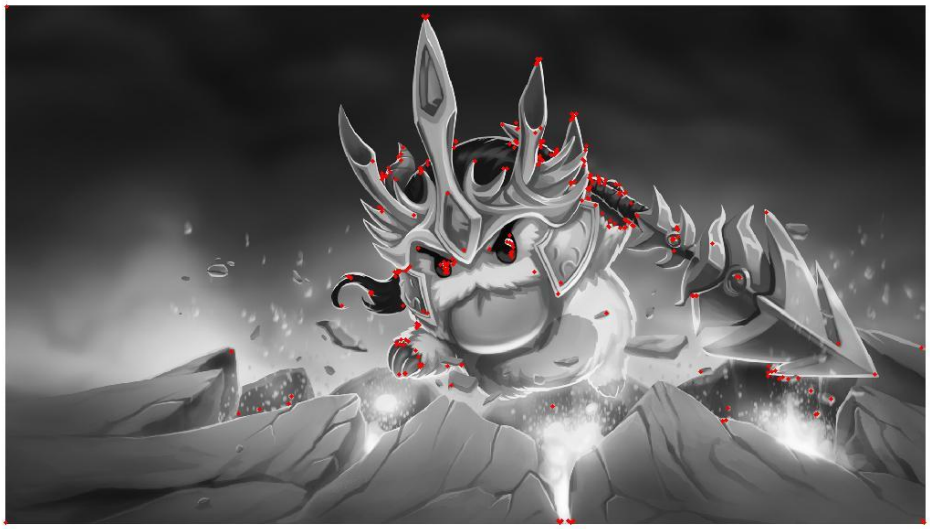
Part 1-D

做法：

- 承 part 1-C 算出來的四張 corner 圖，使用 CornerDetection.nonMaximumSuppression 對他們作 non-maximum suppression，各自的 threshold 也承自 part 1-C 的結果
- 把 CornerDetection.nonMaximumSuppression 的計算結果用 find(Ieig_filtered == 1) 的方式找出 filtered 過後的 corner 位置 (這時候每 3x3 window 內只會有一個 corner)，再用 imshow + plot 把他們畫在原圖上

結果 (可放大來看)：





- 左：kernel size = 10, sigma = 5, window size = 3x3
- 右：kernel size = 10, sigma = 5, window size = 5x5

- 左：kernel size = 20, sigma = 5, window size = 3x3
- 右：kernel size = 20, sigma = 5, window size = 5x5

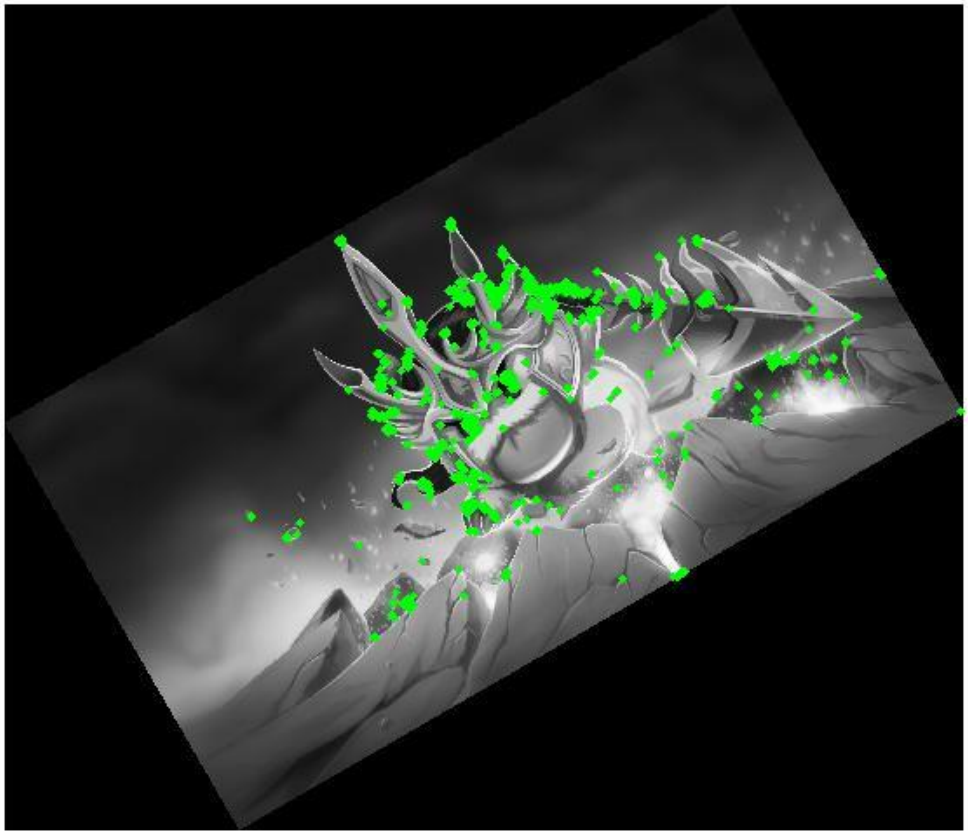
從結果可看出因為作了 non maximum suppression，因此每 3x3 window 內只會有一個 corner，結果的 corner 會變得更清楚。

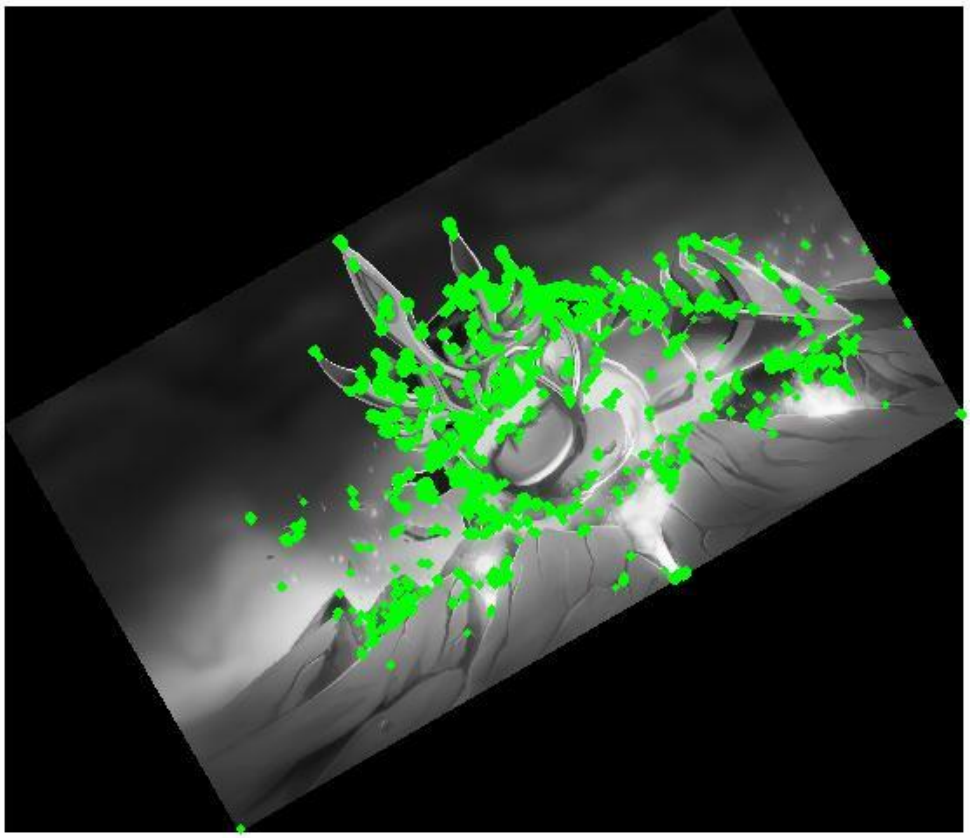
Part 1-E

做法：

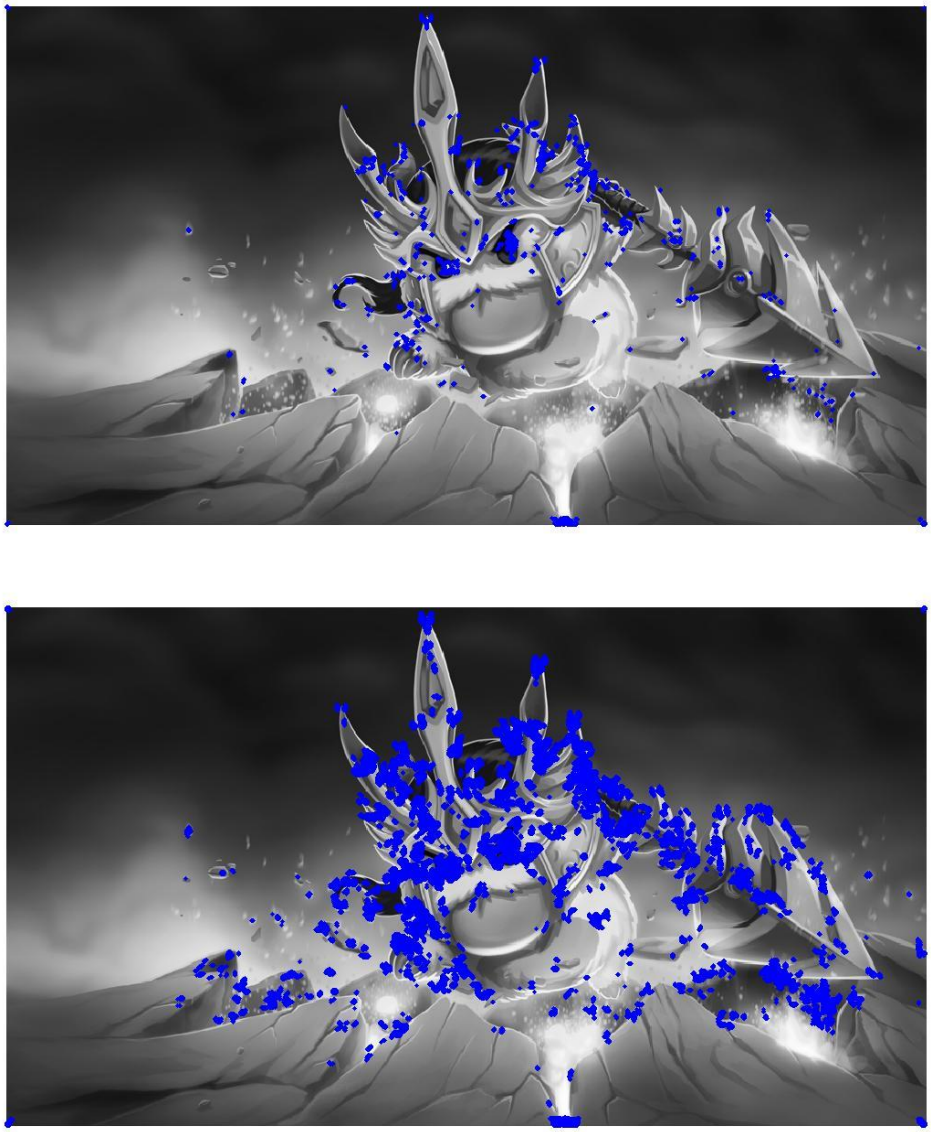
- 用 imread 讀進圖片後，轉成 grayscale，再用 im2double normalize 圖片使值域在 [0, 1]
- 分別對原圖用 imrotate(I, 30) 轉 30 度，以及用 imresize(I, 0.5) 縮小 0.5 倍
- 設定 guassian filter 的參數：
  - kernel size = 10
  - sigma = 5
- 對旋轉以及縮放後的兩張圖用 CornerDetection.gaussianFilter 作 gaussian smoothing
- 分別對兩張 guassian filtered 的圖片，用 CornerDetection.imgradient 算出他們的 gradient magnitude
- 對兩張 gradient magnitude 用不同的 window size (3x3, 5x5) 套用到 CornerDetection.detectCorner 算出各自的 corners
- 用 part 1-C 一樣的方式，算它們四張圖片各自的 threshold
- 這題因為要比較四張圖片的結果，因此就取同一個 threshold = min(他們四個各自的 thresholds)
- 用 find(Ieig > threshold) 的方式找出 corner 的 positions，再用 imshow + plot 把他們畫在原圖上

結果（threshold 沒有定很大，題目也沒有要求作 non-maximum suppression，所以會有很多的 pixel 被判定成 corner）：





- 左：kernel size = 10, sigma = 5, rotate = 30, window size = 3x3
- 右：kernel size = 10, sigma = 5, rotate = 30, window size = 5x5



- 左：kernel size = 10, sigma = 5, scale = 0.5, window size = 3x3
- 右：kernel size = 10, sigma = 5, scale = 0.5, window size = 5x5

Part 1-F

做法：

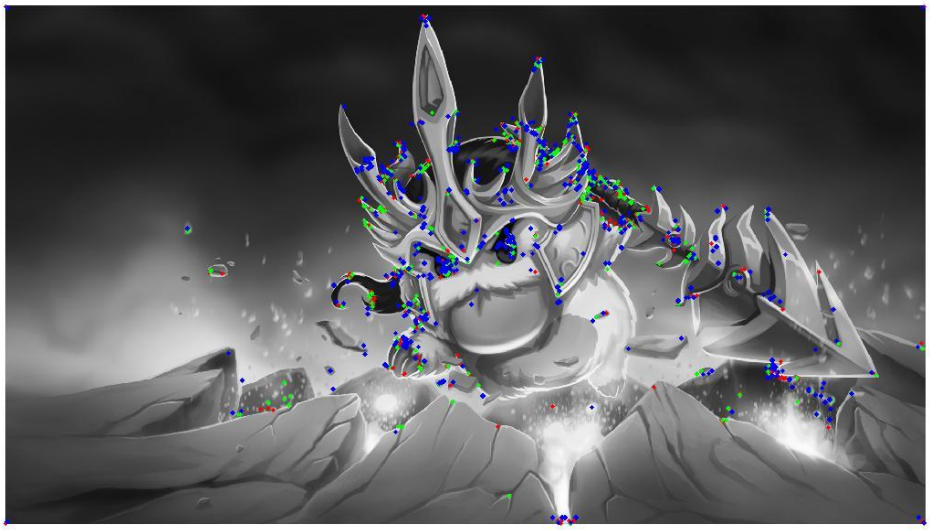
- 用 imread 讀進圖片後，轉成 grayscale，再用 im2double normalize 圖片使值域在 [0, 1]
- 分別對原圖用 imrotate(I, 30) 轉 30 度，以及用 imresize(I, 0.5) 縮小 0.5 倍
- 設定 guassian filter 的參數：

    - kernel size = 10
    - sigma = 5
- 對原圖、旋轉後以及縮放後的三張圖用 CornerDetection.gaussianFilter 作 gaussian smoothing
- 分別對三張 guassian filtered 的圖片，用 CornerDetection.imgradient 算出他們的 gradient magnitude
- 對三張 gradient magnitude 用 window size = 3 套用到 CornerDetection.detectCorner 算出各自的 corners
- 雖然題目沒要求，但為了減少 corner 數量，我們再對這三張 corners 用 CornerDetection.nonMaximumSuppression 作 non maximum suppression

1 由於旋轉過後的圖片會變大，我們先對原圖與對應原圖對應的 smallest eigenvalue matrix (Ieig) 轉 -30 度回來，再用 find 找出原圖的位置，再用 imcrop 把 Ieig 給 crop 出來；而 scaled image 對應的 smallest eigenvalue matrix 把它再 scale 回來，也就是放大 2 倍即可

- 用 part 1-C 一樣的方式，算它們三張圖片各自的 threshold 後，用 find(Ieig > threshold) 的方式找出 原圖、旋轉後以及縮放後的 corner 的位置，再將他們 plot 到原圖上

結果：



    - 紅色：原圖的 corner
    - 綠色：rotate 過後的圖的 corner
    - 藍色：scale 過後的圖的 corner

由於用了不同的 threshold 去偵測他們各自的 corners，因此但從結果中可以觀察出原圖跟旋轉後的 corner 大致上會在同個地方，因為我們用的方法是 rotate-invariant，即

rotate 過後的圖片的 corner 還是不會變，但是就不保證在原圖會判斷成 corner 的地方，scale 之後仍然會判斷成 corner，因為不是 scale-invariant。

Part 2-A

做法：

- 讀取 kobe, gasol 兩張圖並轉成 grayscale
- 用 LBP.computeLBP 算各自的 LBP，並用 imshow 把結果顯示出來。(注意要將 LBP image 轉成 uint8 才能正確用 imshow 顯示)

結果：

Part 2-B

做法：

- 承自 part 2-A 的結果，用 LBP.image2NormalizedHistogramVector 算 kobe, gasol 的 normalized histogram vector，再用 dot product 即可得到 similarity，值域[0, 1] (自己跟自己的 dot product 即為 1)

結果：Similarity:0.9912 (print 在 console 中)

Part 2-C

做法：

- 讀取 kobe, gasol 兩張圖並轉成 grayscale
- 用 LBP.image2ConcatedNormalizedHistogramVector 來算 kobe 跟 gasol 的 2x2, 3x3, 4x4, 9x9, 20x20 的 normalized histogram vectors，一樣用 dot product 即可得到他們不同切割 case 之下的 similarity

結果：(print 在 console 中)

- Similarity (2x2 case):0.97837
- Similarity (3x3 case):0.97422
- Similarity (4x4 case):0.95852
- Similarity (9x9 case):0.87311
- Similarity (20x20 case):0.70176

從結果中可觀察到 split 越多的則 similarity 越低，因為失去了全局的考量，比對的是非常局部的 LBP histogram vector，因此也就導致這樣的結果。

Part 2-D

做法：

- 承自 part 2-A 的結果得到 kobe, gasol 各自的 LBP
- 用 LBP.LBP2UniformLBP 將他們的 LBP 轉成 uniform LBP，並用 imshow 把結果顯示出來。(注意要將 LBP image 點除(“./”) 59 才能正確用 imshow 顯示)

結果：

Part 2-E

做法：

- 承自 part 2-D 的結果得到 kobel, gasol 的 uniform LBP image，再用 LBP.image2NormalizedHistogramVector 算 kobe, gasol 的 normalized histogram vector，再用 dot product 即可得到 similarity，值域[0, 1] (自己跟自己的 dot product 即為 1)

結果：Similarity:0.99118 (print 在 console 中)

Part 2-F

做法：

- 讀取 kobe, gasol 兩張圖並轉成 grayscale
- 用 LBP.image2ConcatedNormalizedHistogramVector (isUniform) 來算 kobe 跟 gasol 的 2x2, 3x3, 4x4, 9x9, 20x20 的 normalized histogram vectors，一樣用 dot product 即可得到他們不同切割 case 之下的 similarity

結果：(print 在 console 中)

- Similarity (2x2 case):0.97843
- Similarity (3x3 case):0.97442
- Similarity (4x4 case):0.95886
- Similarity (9x9 case):0.87494
- Similarity (20x20 case):0.70835

從結果中可觀察 LBP 與 uniform LBP 的結果並不會差太多，但也是 split 越多的則 similarity 越低，因為失去了全局的考量，比對的是非常局部的 LBP histogram vector，因此也就導致這樣的結果。