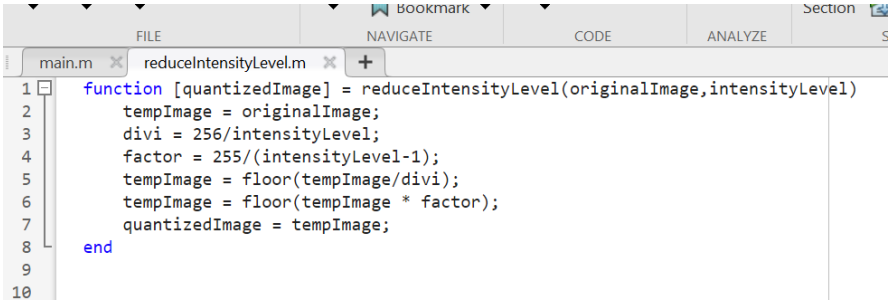
影像處理 Lab1 Report 109062110 祝語辰

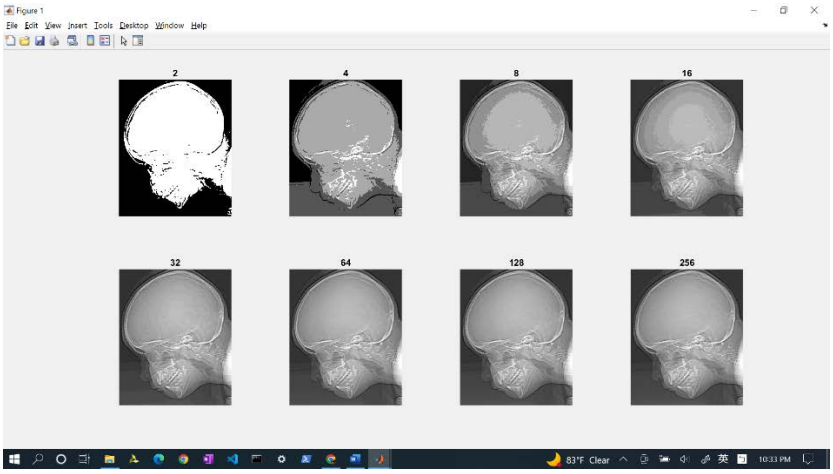
一、 Proj02-02: Reducing the Number of Intensity Levels in an Image

1. 實作方式：

將影像讀取之後，因為他是一個 Matrix，先將其轉為 double，以確保接下來對影像的處理，如對每一個 pixel (element)做除法時，除出來的是小數。在 function 中存入 tempImage(下圖 code line 2)。接下來，因為是要將原本的 intensity 重新對比，所以先將 256 除以 intesity level，求出來的值，暫且稱之為 divi(下圖 code line 3)。利用此值去除以每個 pixel 的 intensity，就可以將原先的 256 個 level(0~255)轉換想要的 intensity level，各個 pixel 在處理後，其內的值會呈現 a.b 形式。再對整個 tempImage 取 floor，也就是各個對各個 pixel 取 floor，就會得到各個 pixel 對應的新 level(下圖 code line 5)。最後要求這些 level 對應到 0~225 下，他們的 intensity。假設現在是 n level，則因為在 0 時對應的是零，所以最後每個 level 的差是 255/n-1，將這個差乘到每個 pixel 中，然後取 floor(因為每個 level 的差，不一定是整數) (下圖 code line 6)。最後在輸出 image 前，要注意，剛開始為了處理影像，我們將 image 轉成 double，這裡樣將他轉回 uint8。



2. 問題與討論：



以上為我的 program 產出的結果。

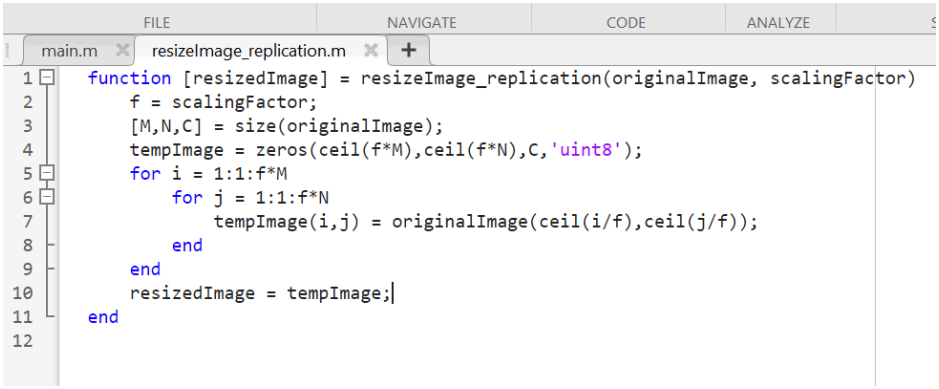
這裡的作法就是將原圖中的 pixel intensity 重新作對應。

二、Proj02-03: Zooming and Shrinking Images by Pixel Replication

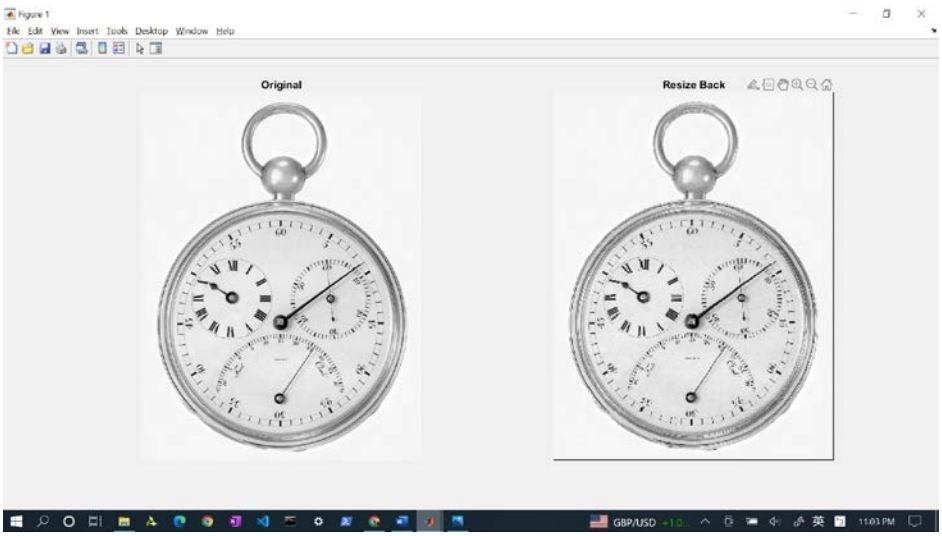
1. 實作方式：

我在這裡的實作方式是先根據縮放的比例，建立一個空的 Matrix。雖然說題目有假設縮放的比例為整數，但考慮到縮小的情形，所以要處理不是整數的情形。在這裡我是使用 ceil，如此以來，假如計算後，影像的長寬不為整數的問題就可以被解決。

接下來的想法就是將這個空的 Matrix 的每個 element 填滿，也就是使用兩個 for 去跑。而不論是放大或縮小，都會有些點沒有對應到原先的 pixel。這個時候我使用的方式是將改 pixel 除以縮放比例，在取 ceil，就可以得到向對應在原圖鄰近的相鄰點，將該點的 intensity 讀進此 pixel。最後再傳出。



2. 問題與討論：



以上為我的 program 產出的結果。

從圖中可以看到雖然兩者的大小相同，且具有相同數量的 pixel，但因為在縮小的時候，有丟失掉一些資訊，再次放大的時候，就只能用 nearest neighbor 的 intensity 來補，雖然圖像看起來大體接近，但可以注意到在非純色的區域，以此圖來說就是這個懷錶的刻度以指針處會有明顯的不連續。由此可見，此一方式比較適合在整個影像顏色較為接近，沒有明顯對比以及細節的情形。

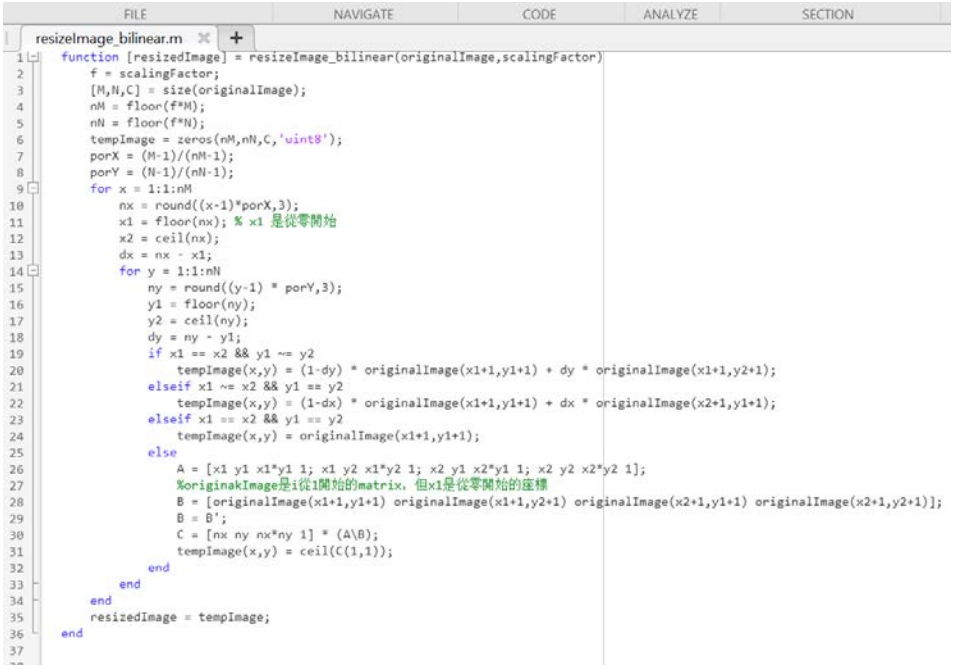
三、Proj02-04: Zooming and Shrinking Images by Bilinear Interpolation

1. 實作方式：

在這裡我的作法主架構與 proj02-03 接近，一樣是先根據縮放比例(利用 desire dpi 除以 original dpi。當影像是 jpg 或 tiff 時 original dpi 可以利用 imfinfo 取得。)來建立一個空的 matrix，並且同樣為了避免 matrixt 長寬非整數，在計算時新建 matrix 的 size 時，我使用 floor 來處理。接下來，計算新 matrix 中每個 pixel 對應到原先 image 時，最近的是哪四個點，這所選的鄰近點，使採用類似 Diagonal neighbors 的方式。並且因為因為計算方便，我設想 Image 是從 0 開始編號的，但因為 matlab 中，for 是要從正整數開始，所以在處理時，要注意轉換。

我在 code 中宣告了兩個變數 porX 和 porY，它們代表的是縮放後的 matrix，其 pixel 與 pixel 之間，對應到原圖的比例。

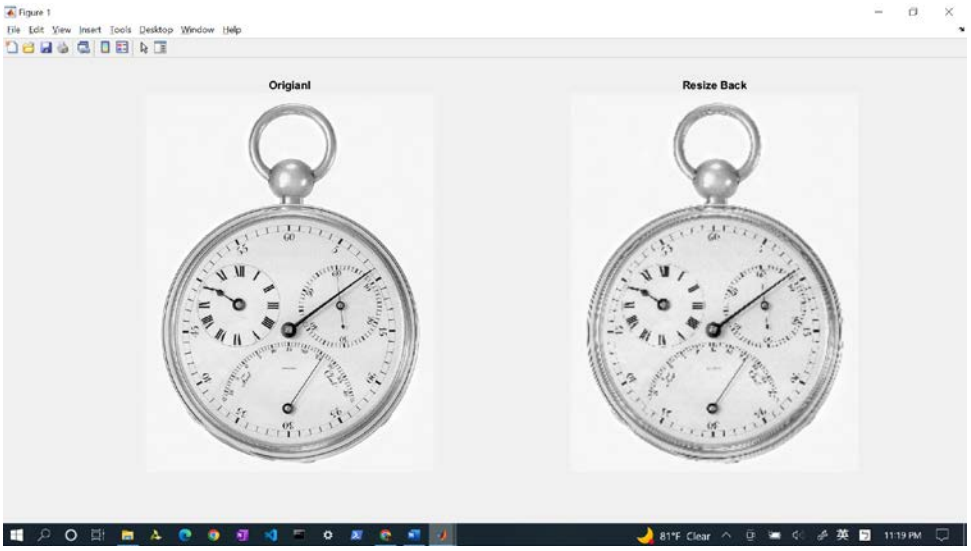
以此來計算出縮放後的 matrix，其 pixel 對應到原圖得相對位置。然後，因為我們在意的是他的鄰近點，所以對他取 floor 和 ceil，這樣就可以知道對應到原圖時鄰近四點的座標。接下來針對邊界條件個別處理，如果剛好對應到原圖中的某 pixel，就直接將其讀進來，以及如果剛好 x 方向或 y 方向無變化，就可以只對其中一個方向處理。另外，因為在極端的情形下，計算出來，對應的位置會極靠近這些邊界條件，如果不特別處理，就會導致在用 ceil 後，求出來的值超出 image 當大小限制，所以在計算對應點時，我使用 round 四捨五入到小數點後三個 digit 來忽略掉極端的情形。最後就是一般情形的處理。參考上課的 slide，利用矩陣運算，透過鄰近四點來求 intensity。



resizeImage_bilinear.m

```
function [resizedImage] = resizeImage_bilinear(originalImage,scalingFactor)
    f = scalingFactor;
    [M,N,C] = size(originalImage);
    nM = floor(f*M);
    nN = floor(f*N);
    tempImage = zeros(nM,nN,C,'uint8');
    porX = (M-1)/(nM-1);
    porY = (N-1)/(nN-1);
    for x = 1:1:nM
        nx = round((x-1)*porX,3);
        x1 = floor(nx); % x1 是從零開始
        x2 = ceil(nx);
        dx = nx - x1;
        for y = 1:1:nN
            ny = round((y-1) * porY,3);
            y1 = floor(ny);
            y2 = ceil(ny);
            dy = ny - y1;
            if x1 == x2 && y1 ~= y2
                tempImage(x,y) = (1-dy) * originalImage(x1+1,y1+1) + dy * originalImage(x1+1,y2+1);
            elseif x1 ~= x2 && y1 == y2
                tempImage(x,y) = (1-dx) * originalImage(x1+1,y1+1) + dx * originalImage(x2+1,y1+1);
            elseif x1 == x2 && y1 == y2
                tempImage(x,y) = originalImage(x1+1,y1+1);
            else
                A = [x1 y1 x1*y1 1; x1 y2 x1*y2 1; x2 y1 x2*y1 1; x2 y2 x2*y2 1];
                %originakImage是i從1開始的matrix, 但x1是從零開始的座標
                B = [originalImage(x1+1,y1+1) originalImage(x1+1,y2+1) originalImage(x2+1,y1+1) originalImage(x2+1,y2+1)];
                B = B';
                C = [nx ny nx*ny 1] * (A\B);
                tempImage(x,y) = ceil(C(1,1));
            end
        end
    end
    resizedImage = tempImage;
end
```

2. 問題與討論：



以上為我的 program 產出的結果。

從圖中可以看到雖然兩者的大小相同，且具有相同數量的 pixel，但和 proj02-03 一樣，因為在縮小的時候，有丟失掉一些資訊，再次放大的時候，就只能用 nearest neighbor 的 intensity 來補。可是相較於 proj02-03，這裡是使用插值法，另用周邊 pixel 的資訊來補。雖然和原圖去比較，在細節處，比方說刻度，會比較模糊，但是和 proj02-03 的結果相比，會感覺比較光滑，連續。