Multimedia HW1

1. DCT

在這題中，我讀進來的圖檔都先將其 pixel 值換成 double，再除以 255，得到的 RGB 區間為 0~1，這樣做的目的是因為轉換成 YIQ 需要使用這樣的區間去表示 RGB。

a.

這一題要實作 DCT，將原來的圖片從 spatial domain 的表示方式轉換到 frequency domain 的表示方式。接著再使用 inverse DCT 與 2*2、4*4、8*8 的 DCT coefficient 轉換回去。並看各種情況時的 psnr 為多少。

要算 DCT coeffiecient，首先要來算 1D 的 DCT basis。利用 C(u)和公式，即可得出。用以下程式碼實作。（i 的值為講義公式中的 u+1）

```
for i = 1:8
    for r = 0:7
        if i==1
            C = 2^0.5/2;
        else
            C = 1;
        end
        u(r+1,1,i) = (2^0.5)*C/(8^0.5)*cos((2*r+1)*(i-1)*pi/16);
    end
end
```

其中當 u = i-1 = 0 的時候，變數 C 代表 C(u)，值為 2^0.5/2；其餘 C = 1。

而此處用一個三維的 matrix 來存 DCT basis，以利於稍後的計算。

算完 1D 的 DCT basis 後，要來算 2D 的。也就是讓其中一組 basis，和另外一組 basis 的轉至矩陣相乘。用以下程式碼實作。

```
for i = 1:8
    for j = 1:8
        a = u(:,1,i);
        b = u(:,1,j);
        U(:,:,i,j) = a*b';
    end
end
```

其中 U 是一個四維的矩陣，前兩個維度來記錄 2D DCT basis 的值，後兩個維度的 index 表示是哪一個 2D DCT basis。

算出全部的 2D DCT basis 之後，開始針對每 8*8 的圖片單位進行轉換，並在轉換好後，選擇欲使用 2*2 或 4*4 或 8*8 的 DCT coefficient，進行 inverse DCT。

```
for rgb = 1:3
    for row = 0:(height/8-1)
        for column = 0:(width/8-1)
            data_t(:,:) = imdata(row*8+1:row*8+8,column*8+1:column*8+8,rgb);
            for i = 1:8
                for j = 1:8
                    U_t(:,:) = U(1:8,1:8,i,j);
                    t = dot(U_t,data_t,2);
                    val = sum(t);
                    T(i,j) = val(1,1);
                end
            end

            data_t(:,:) = zeros(8,8);
            for i = 1:2
                for j = 1:2
                    data_t(:,:) = data_t(:,:) + T(i,j)*U(1:8,1:8,i,j);
                    %disp(data_t);
                end
            end

            imdata(row*8+1:row*8+8,column*8+1:column*8+8,rgb) = data_t(:,:);

        end
    end
end
```

得出來的 imdata 即為所求。

原始圖檔：

2*2：

4*4：

8*8

Psnr：在本次作業中，我的 psnr 值是算 R、G、B 三個向度，用原始圖檔的每一個 pixel 值減掉經過 DCT 處理（轉換+inverse）的圖檔的每一個 pixel 值。算出來的差平方相加，除以圖片的長*寬，再除以 3。算出來的均方誤差再透過公式 PSNR = 10 * log10(255^2/MSE)，得出來的值。程式碼如下：

```
SE = 0;
for rgb = 1:3
    for i = 1:height
        for j = 1:width
            SE = SE + (org(i,j,rgb)*255 - imdata(i,j,rgb)*255)^2;
        end
    end
end
MSE = SE/(height*width*3);
PSNR = 10 * log10(255^2/MSE);
disp("psnr:");
disp(PSNR);
```

經過這個算法，以下為各種DCT coefficient size求出來的PSNR值。

2*2：27.2584

4*4：35.6371

8*8：306.9350

透過PSNR與生成圖片的觀察，得出一致的結論：DCT coeffiecient的使用size越小，生成的圖片與原始圖檔差距越多，在這裡使用size越小的，生成圖檔越模糊，反之，則和原始圖檔差距越小。此處PSNR的意義除了表示生成圖檔和原始圖檔的差距，透過肉眼的觀察，亦可以表示圖檔的清晰程度。此外可以觀察到在這裡，使用8*8的DCT coefficient算出來的PSNR理論上應該為無限大，但這裡我推測是因為在做DCT運算的時候，因為有些小數點後的位數無法精確地表示，所以處理後的圖檔和原始圖檔還是有一點差距，故算出來的PSNR並非是無限大。

b.

第二題的在 DCT 的作法上和上一題的做法一樣。但要丟進去 DCT 的是圖檔要先從 RGB 轉成 YIQ，做完 DCT 之後，再把處理完的 YIQ 圖檔轉換成 RGB。

```
RGB = zeros(3,1);
YIQ = zeros(3,1);
YIQ_transform = [0.299 0.587 0.114 ; 0.596 -0.275 -0.321 ; 0.212 -0.523 0.311];
RGB_transform = [1 0.956 0.619 ; 1 -0.272 -0.647; 1 -1.106 1.703];
```

首先宣告兩組 3*1matrix，這是為了稍後轉換運算要去存每一個 pixel 上的值。再宣告 YIQ_transform 和 RGB_transform，這兩個是為了進行轉換的轉換

matrix。

```
for row = 1:height
    for column = 1:width
        RGB(1,1) = imdata(row,column,1);
        RGB(2,1) = imdata(row,column,2);
        RGB(3,1) = imdata(row,column,3);
        YIQ = YIQ_transform * RGB;
        imdata(row,column,1) = YIQ(1,1);
        imdata(row,column,2) = YIQ(2,1);
        imdata(row,column,3) = YIQ(3,1);
    end
end
```

之後針對每一個 pixel 進行 YIQ 轉換。
接著去做 DCT 轉換與 DCT inverse。做完了之後再進行從 YIQ 轉至 RGB 的轉換。

```
for row = 1:height
    for column = 1:width
        YIQ(1,1) = imdata(row,column,1);
        YIQ(2,1) = imdata(row,column,2);
        YIQ(3,1) = imdata(row,column,3);
        RGB = RGB_transform * YIQ;
        imdata(row,column,1) = RGB(1,1);
        imdata(row,column,2) = RGB(2,1);
        imdata(row,column,3) = RGB(3,1);
    end
end
```

經過這些步驟，得出來的圖片即為所求。

YIQ_2*2：

YIQ_4*4：

YIQ_8*8：

PSNR：

2*2：27.2584

4*4：35.6369

8*8：78.4672

透過 PSNR 與生成圖片的觀察，得出一致的結論：DCT coeffiecient 的使用 size 越小，生成的圖片與原始圖檔差距越多，在這裡使用 size 越小的，生成圖檔越模糊，反之，則和原始圖檔差距越小。此處 PSNR 的意義除了表示生成圖檔和原始圖檔的差距，透過肉眼的觀察，亦可以表示圖檔的清晰程度。

c.

比較 a 和 b 做出來的圖片結果，用肉眼觀察其實看不出甚麼差異。但如果從資料運算上可以看出兩者其實還是有些差異。單舉 PSNR 的例子，在 DCT

coefficient 2*2 的情況下 a 和 b 的 PSNR 值皆為 27.2584。在 DCT coefficient 4*4 的情況下 a 的 PSNR 值為 35.6371，b 的值為 35.6369。在 DCT coefficient 8*8 的情況下 a 的 PSNR 值為 306.9350，b 的值為 78.4672。顯示出如果先將原始圖檔轉換成 YIQ 的形式，接著進行 DCT 的運算，最後再將生成的圖檔轉換回 RGB 的形式，則這個過程將會導致生成圖檔與原始圖檔之間的差距變大。

2.Dithering

a.

使用兩層迴圈對於輸入的灰階圖檔做 pixel 上的處理。先在對每一個 pixel 作處理前，生成一個 0~255 的亂數，接著開始處理 pixel，若 pixel 的值大於亂數，則使其為 255；反之則使其為 0。

b.

先設一個變數 average，使其為所有原始圖檔 pixel 值的總和。然後再將 average 除以原始圖檔的長度和寬度，也就是算出一個圖檔所有 pixel 值的平均值。
然後再對於每一個 pixel 的數值與 average 這個變數做比較，若比其大，pixel 值為 255，比其小，pixel 值為 0。

c.

為了確保圖檔的邊界也能夠處理到，我設了一個變數 expand_imdata 去存原始圖檔的 pixel 值，但其長度和寬度皆比原始圖檔多兩個 pixel，多出來的部分值為 0。(這是為了確保原始圖檔邊界的 pixel 也可以被處理到)。

接著先跑 Floyd-Steinberg algorithm，算出每一個 pixel 的值，演算法的程式如下。

```
for row = 2:height+1
    for column = 2:width+1
        p = expand_imdata(row,column);
        if p < 128
            e = p;
        else
            e = p - 255;
        end
        expand_imdata(row,column+1) = expand_imdata(row,column+1) + (7/16)*e;
        expand_imdata(row+1,column-1) = expand_imdata(row+1,column-1) + (3/16)*e;
        expand_imdata(row+1,column) = expand_imdata(row+1,column) + (5/16)*e;
        expand_imdata(row+1,column+1) = expand_imdata(row+1,column+1) + (1/16)*e;
    end
end
```

接著再對於跑完演算法的 expand_imdata，先將其比原始圖檔多出來的部分切掉，並把值賦予給 imdata。最後對於每個 pixel，拿其值和 128 這個數值做比較，如果比 128 小，則使該 pixel 值為 0，反之為 255。

```
imdata = expand_imdata(2:height+1,2:width+1);
for row = 1:height
    for column = 1:width
        if imdata(row,column) < 128
            imdata(row,column) = 0;
        else
            imdata(row,column) = 255;
        end
    end
end
```

d.

在這三種方法中，因為 noise dithering 的比較值是隨機產生的，故會有一種比較黑白點參雜的感覺，圖像比較模糊。Average dithering 是用整張圖的 pixel 值的平均值來當比較值，產生的圖檔也因此而有一大塊黑色與一大塊白色出現的情形，但圖像比 noise dithering 清楚。而 error diffusion dithering 由於透過特定演算法去將誤差擴散出去，故顆粒很細緻，整體圖像的呈現也最清晰。

3. Image Convolution

在這題中，我讀進來的圖檔都先將其 pixel 值換成 double，再除以 255，得到的 RGB 區間為 0~1，這樣做的目的是因為要使 Convolution 的運算較為精準。

a.

我首先先使用 matlab 的 fspecial()這個函式，透過 Guassian function，輸入 sigma 為 1，分別輸入 3*3、5*5、7*7 的大小，生成三種大小的 filter。

然後宣告一個 imdata_expand 的矩陣去儲存 imdata（讀進來的圖片資料），並依照情況擴展 imdata_expand 的長與寬。如遇到 3*3 的 filter，imdata_expand 要向外擴展一個 pixel，也就是長度加二，寬度加二；遇到 5*5 的 filter，imdata_expand 要向外擴展兩個 pixel，也就是長度加四，寬度加四；遇到 7*7 的 filter，imdata_expand 要向外擴展三個 pixel，也就是長度加六，寬度加六。擴展的部分的 rgb 數值都是 0。
這麼做的目的是要讓每一個原始圖片的 pixel 都可以做到 Convolution。

有了要操作的圖片矩陣跟 filter，我接著就用迴圈分別根據 R，G，B 三個向度，一格一格 pixel 的作 Convolution。

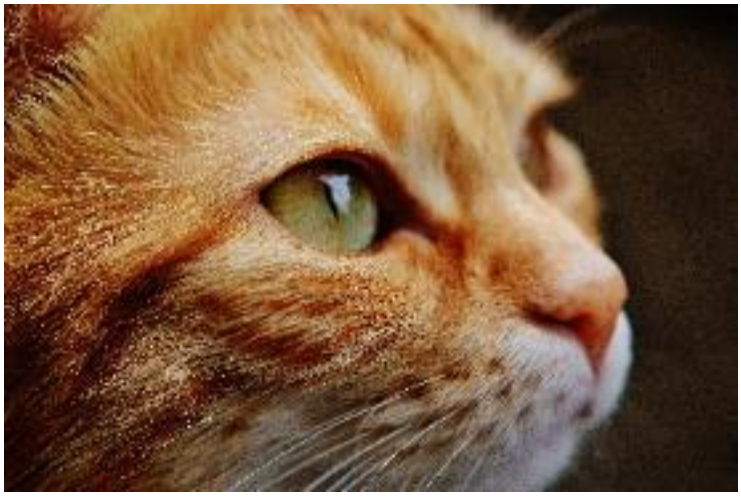
```
for rgb = 1:3
    for i = 2:height+1
        for j = 2:width+1
            g = 0.0;
            for m = 1:3
                for n = 1:3
                    g = g + imdata_expand(i+m-2,j+n-2,rgb)*G(m,n);
                end
            end
            imdata_expand(i,j,rgb) = g;
        end
    end
end
```

其中 m 和 n 的用途是為了要讓以要計算的 pixel 位置為中心，filter 的每一格都與 imdata_expand 中的 pixel 進行運算。
做完之後為了將 imdata_expand 旁邊多出來 pixel 去掉，故使用以下操作讓輸出的檔案大小跟原始檔案一樣。

```
show = imdata_expand(2:height+1,2:width+1,:);
subplot(1,2,1), imshow(show);
subplot(1,2,2), imshow(imdata);

imwrite(show,"cat3_LR_gaussian_sigma1_33.jpg");
```

原圖:



以下為程式跑完輸出的圖檔：

3*3 的 filter:

5*5 的 filter:

7*7 的 filter:

觀察：
用肉眼仔細觀察，會發現圖片的細節隨著 filter 的大小變大，而逐漸模糊。

psnr：
3*3: 23.0372
5*5: 22.4203
7*7: 22.3725

b.
這一題的方法和 a 是一樣的，只不過在生成 filter 的時候，sigma 分別輸入，1，5，10。Filter 的大小為 5*5。
Sigma1:

Sigma5:

Sigma10:

觀察：可以發現隨著 Sigma 越大，圖片細節越模糊。

psnr:
Sigma1: 22.4203
Sigma5: 20.9295
Sigma10: 20.8955

c.
透過以上兩組 Convolution 方式得到的結果，可以推測出當 filter 的 size 越大，或 Gaussian function 的 sigma 值越大，都會造成影像細節越來越模糊。而從 psnr 值也可以看出來，當作出來的圖像細節越模糊，psnr 越低，表示圖像和原始圖檔的差距越大，失真較嚴重。