

# Homework #1

劉士弘-103060004

## ❖ 1

- A.** 一開始我們要使用DCT去轉換讀進來的圖片，在function中，首先我們先把讀進來的圖片轉成double，之後把圖片的RGB個別取出來，再來因為8個pixel要變成一個block，所以我們用兩層for迴圈去計算講義上的公式，會得到一個矩陣，用於接下來圖片DCT的轉換。

```
for i = 1 : 8
    for j = 1 : 8
        if i == 1 || j == 1
            f(i,j) = 1 / sqrt(8) ;
        else
            f(i,j) = cos(pi * (2*(j-1) +1)*(i-1)/16) * sqrt(2/8) ;
        end
    end
end
```

第二步，我們要使用第一步得到的矩陣，去求得DCT完後的圖片，因為是8個pixel為一個block，所以我們必須以一個block為一個單位去實作，而這次是實作2D-DCT，所以我們要把第一步得到的矩陣乘上input再乘上轉置矩陣，之後再用兩層for迴圈把自訂每一個row的n到8個pixel歸零即可得到答案。

```
for i = 1 : 8 : h
    for j = 1 : 8 : w
        R = in_r(i : i+7 , j : j+7 ) ;
        G = in_g(i : i+7 , j : j+7 ) ;
        B = in_b(i : i+7 , j : j+7 ) ;
        R1 = DCT_2(R , f , n) ;
        G1 = DCT_2(G , f , n) ;
        B1 = DCT_2(B , f , n) ;
        result(i : i+7 , j : j+7 , 1) = R1 ;
        result(i : i+7 , j : j+7 , 2) = G1 ;
        result(i : i+7 , j : j+7 , 3) = B1 ;

        end
    end
function [ output ] = DCT_2( input , mat , n )
%UNTITLED Summary of this function goe here
% Detailed explanation goes here
output = zeros(8,8) ;
output = mat*input*(mat') ;

for i = n+1 : 8
    for j = n+1 : 8
        output(i,j) = 0 ;
    end
end
```

---

第三步則是把DCT完後的圖片，使用IDCT把圖片轉回來，而實作方法也跟DCT類似，只需要把DCT使用到的矩陣去inverse即可。

```
for i = 1 : 8 : h
    for j = 1 : 8 : w
        R = result(i : i+7 , j : j+7 , 1) ;
        G = result(i : i+7 , j : j+7 , 2) ;
        B = result(i : i+7 , j : j+7 , 3) ;
        R1 = IDCT_2(R , f ) ;
        G1 = IDCT_2(G , f ) ;
        B1 = IDCT_2(B , f ) ;
        result2(i : i+7 , j : j+7 , 1) = R1 ;
        result2(i : i+7 , j : j+7 , 2) = G1 ;
        result2(i : i+7 , j : j+7 , 3) = B1 ;
    end
end
```

```
function [ output ] = IDCT_2( input , mat )
%UNTITLED Summary of this function goe here
% Detailed explanation goes here
output = zeros(8,8) ;
output = inv(mat)*input*inv(mat') ;
```

而最後我們需要去計算PSNR，因為是彩色圖片，所以需要分別去計算RGB的MSE，之後再把計算出來的MSE去做平均，之後再套公式即可算答案。

```
input1 = im2double(input1) ;
input2 = im2double(input2) ;

MSER = sum(sum((input1(:,:,1) - input2(:,:,1)).^2)) / h / w;
MSEG = sum(sum((input1(:,:,2) - input2(:,:,2)).^2)) / h / w;
MSEB = sum(sum((input1(:,:,3) - input2(:,:,3)).^2)) / h / w;

MSE = (MSER+MSEG+MSEB)/3 ;

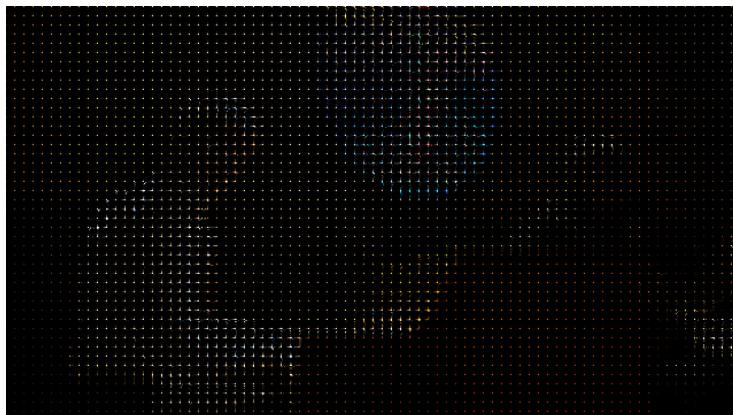
%disp(MSE) ;

psnr = 10 * log10(1 / MSE);

%disp(psnr) ;
```

## Result Image:

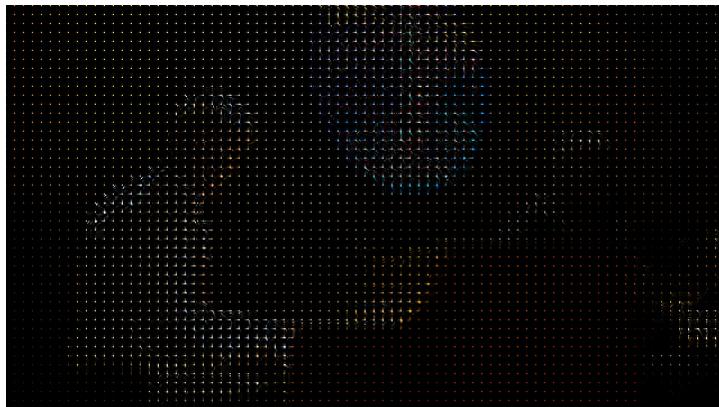
DCT n=2



DCT n=4



DCT n=8



IDCT n=2

PSNR = 34.4167



IDCT n=4

PSNR=43.5330

IDCT n=8

PSNR = 315.7372



## DISCUSSION :

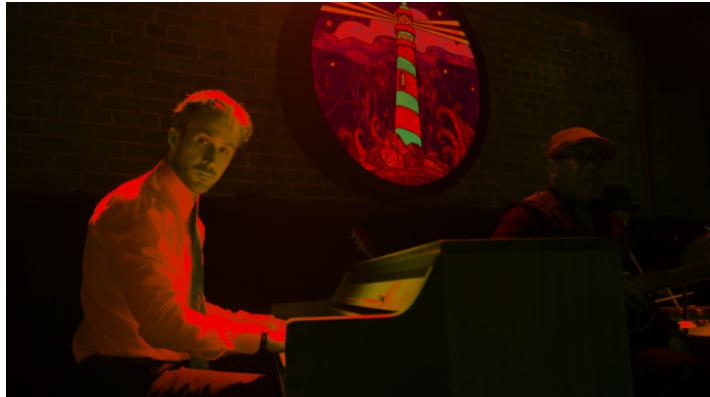
我們可以發現，當n越大時，DCT所轉換出來的影像就會越清楚，因為被遮掉的pixel值越少，而使用IDCT轉換回原本的圖像時，n越大時也會越清楚，從上面的PSNR我們也可以得知，當PSNR越大時，代表與原本圖像的相似度越高，因為MSE越小，代表誤差越小，所以當除上MSE時，PSNR就會越大，所以PSNR在這裡代表的是與原圖的差異，PSNR越大，代表與原圖越相似。

**B.** 第二部分，主要是要把圖片轉到YIQ之後做第一部分的動作再把圖片從YIQ轉回RGB，而一開始，我們需要有一個 $3 \times 3$ 的個陣，目的是為了把圖片從RGB轉到YIQ，有了矩陣後，我們可以利用矩陣乘法，把圖片從RGB轉到YIQ，但因為我們讀進來的input是一個 $3 \times 3$ 的矩陣，而為了要做矩陣乘法，我們必須對input reshape，只要呼叫`reshape(input, 1, 3)`，再做乘法，即可完成。

之後把得到的圖案，重複做第一部分的事情後，即可得到DCT過後的圖案，之後再做IDCT，再來就是把圖片從YIQ轉回RGB，此處也只需要把矩陣取inverse在執行第二部分的code就可以完成了。

### Result Image:

**YIQ space**



**Reconstruct n=2**

**PSNR = 34.4167**



**Reconstruct n=4**

**PSNR = 43.5330**



**Reconstruct n=8**

**PSNR = 315.4850**



---

## **DISCUSSION:**

從上面圖片可以看出來，跟第一小題一樣，當n越小時，影像就會越模糊，PSNR的值也會越高，而隨著n的增加，圖片也會看起來更加的清楚，PSNR也會越高，此處PSNR也代表著兩圖片的相似度，PSNR越高，相似度越高。

C.

## **Compare:**

我們可以從a和b小題發現，若把影像先轉成YIQ space在運算DCT其實和把原本的影像直接做DCT並沒有太大的差異，由PSNR的值我們可以發現，只有在n=8時，兩者會有些許的差異，也不是完全一模一樣，因為轉到了YIQ space後，有些頻率會不一樣，自然在做DCT後的差別也有所不同。

---

## ❖ 2

- A. 第二題我們要去實作**Gaussian blur filter** 和**median filter**，首先**Gaussian blur filter**，我們可以利用matlab的內建函式**fspecial function**，去創造出我們需要的**mask**，之後利用這個**mask**去做**convolution**，我們利用4層for迴圈去實作，外面兩層主要是控制**output**的index，而裡面兩層是主要計算**convolution**，最後即可得到答案。

```
H = fspecial('gaussian',[3 3] , 0.3);
G = fspecial('gaussian',[9 9] , 1.0);
```

```
[h ,w] = size(input) ;
input = im2double(input);
[h1 , w1] =size(mask) ;
mask = im2double(mask);
mh_trans = int16(h1/2);
mw_trans = int16(w1/2);

output = zeros(h, w );

for i = 1 : h
    for j = 1 : w
        % transform the index
        for k = 1-mh_trans : h1-mh_trans
            for l = 1-mw_trans : w1-mw_trans
                % check overflow
                if(i+k > 0 && j+l > 0 && i+k <= h && j+l <= w)
                    % do the sigma
                    output(i, j) = output(i, j) + input(i+k, j+l) * mask(k+mh_trans, l+mw_trans);

                end
            end
        end
    end
end
```

---

## Result Image:

**mask sizes  $3 \times 3$  with sigma 0.3**



**$9 \times 9$  with sigma 1.0**



## DISCUSSION:

由上面兩張圖可以發現，當mask的size越大時，會讓原本的影像變得更加模糊，而sigma的值也會影響圖片的模糊程度，當sigma的值越大時，他會blur更寬的範圍，所以右邊的圖看起來也更加地模糊。

**B.** 而b小題則是要用median filter讓照片模糊化，作法不同於Gaussian blur filter的地方是它沒有mask，而是利用取中位數的方法來取得output的值，作法則是看filter的大小，去把對應到input的直取中位數即可完成，而取中位數用matlab的內建函式median()即可。

```
for i = 1 : h
    for j = 1 : w
        % transform the index
        index = 1 ;
        for k = 1-mh_trans : sizes-mh_trans
            for l = 1-mw_trans : sizes-mw_trans
                % check overflow
                index = index +1 ;
                if(i+k > 0 && j+l > 0 && i+k <= h && j+l <= w)
                    % do the sigma
                    s(index) = input(i+k, j+l) ;

                end
            end
        end

        output(i,j) =median(s);

    end
end
%display(s) ;
```



## DISCUSSION:

在median filter中，會依照filter的大小去取input的中位數，所以當filter為 $3 \times 3$ 時，是9個pixel取中位數去決定output的值，所以相對於filter為 $9 \times 9$ 時較為清楚。

而median filter 和 Gaussian blur filter的目的都是又把圖片模糊話，但median filter的效果明顯比較強烈，因為它等於是拿其中一個pixel取代了一個filter的值，而Gaussian blur filter則是會去做convolution，所以模糊感比較不會那麼強烈。

## ❖ 3.

**A.** 此題要我們使用NN去放大一張圖片，而NN則是找離目前pixel最接近的4個整數點，找的方法則是對目前pixel的座標加上一個小數字後取floor()，以及減掉一個小數字後取ceil，排列組合後可以得到4個點，之後去計算距離，找尋距離最短的點，即是答案。

```
for i = 1:H
    for j = 1:W
        a = i/4 ;
        b = j/4 ;
        c = [floor(a+finetune) floor(b+finetune) ; ceil(a-finetune) floor(b+finetune);floor(a+finetune) ceil(b-finetune);ceil(a
d = [a,b] ;
max =999999 ;
index = 1 ;
for k = 1 : 4
    dis = (c(k,1) - d(1)).^2 + (c(k,2) - d(2)).^2;
    if(dis < max)
        index = k ;
        max = dis;
    end
end
m = c(index , 1);
n = c(index , 2) ;
if(m == 0)
    m= 1 ;
end
if(n==0)
    n= 1 ;
end
output(i,j,:)= input(m,n ,:);

end
end
```

PSNR=23.7984



---

## DISCUSSION:

由上面這張圖可以發現，因為是找最鄰近的點去擴大影像，所以會有模糊的情況，以及有些地方會連接的沒有那麼順暢，以及觀察PSNR可以發現，與原圖其實有著不小的差異。

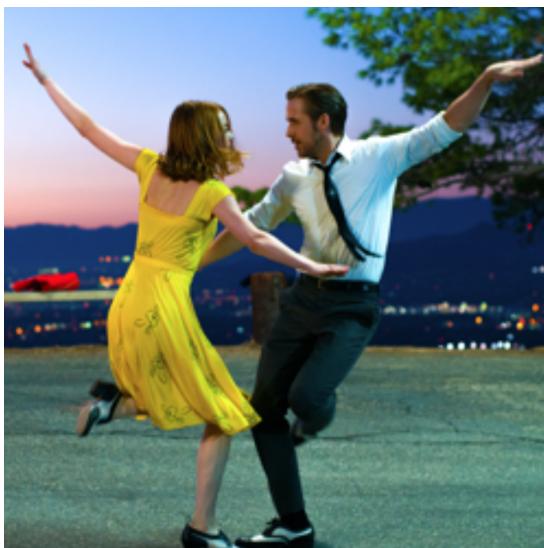
**B.** 而第二個方法則是要使用bilinear去實作內插法，一樣先尋找相鄰的4個點，之後利用這4個點去做線性內差，運用兩層for迴圈，帶入公式後即可得到答案。

```
for i = 1:new_h
    % find the coordinate point in original image
    x = h_factor * (i-1) + 1;
    x1 = floor(x + finetune);
    x2 = ceil(x - finetune);

    for j = 1:new_w
        % find the coordinate point in original image
        y = w_factor * (j-1) + 1;
        y1 = floor(y + finetune);
        y2 = ceil(y - finetune);

        % use the interpolation method
        a = x - x1;
        b = y - y1;
        if(new_d == 1) % grayscale
            x1y1 = input(x1, y1);
            x1y2 = input(x1, y2);
            x2y1 = input(x2, y1);
            x2y2 = input(x2, y2);
            output(i, j) = (1-a) * ((1-b)*x1y1 + b*x1y2) + a * ((1-b)*x2y1 + b*x2y2);
        else % RGB
            x1y1 = input(x1, y1, :);
            x1y2 = input(x1, y2, :);
            x2y1 = input(x2, y1, :);
            x2y2 = input(x2, y2, :);
            output(i, j, :) = (1-a) * ((1-b)*x1y1 + b*x1y2) + a * ((1-b)*x2y1 + b*x2y2);
        end
    end
end
```

PSNR=28.1761



---

## **DISCUSSION:**

由上面的圖可以看出，雖然影像還是有些許不清楚，但相對於NN來說，已經清楚了許多，也可以從PSNR看出，和原圖的相似度也有提高。

C.

### **Compare:**

經過上面兩種方法我們可以發現，因為NN是利用找尋最近的點來當作答案，所以影像的不連貫性以及模糊程度會比較高，而bilinear是利用內差的方法，可以利用鄰近的點去計算出答案，相較於NN精確度更高，也更可以避免不連貫性以及模糊的發生。

而已PSNR來看，後者的PSNR比較高，也意味著，他和放大四倍的原圖比較相近，而前者比較低，則與放大4倍的原圖相差較多。